NUTRITION ASSISTANT APPLICATION

CHAPTER 1

INTRODUCTION

1.1 DESCRIPTION

This project is a web application that returns nutritional values of the food items that the user has uploaded as an image. This is achieved by using Flask which is a famous python library. I have also used IBM DB to store credentials and APIs that provide me with nutritional information namely Clarifai's AI-Driven Food Detection Model Service to analyze the images and Nutrition API.

1.2 PURPOSE

Nutrition today is a very big concern for all individuals. With the amount of junk food consumption, the world is moving to a very unhealthy place. Nutritionists serve as guides to manage our diet and make sure we do not have an unhealthy lifestyle but not everybody can afford a personal nutritionist. This is where our app comes in, it is an inexpensive way to find out how much calorie you would be consuming and therefore manage it yourself. Not only this teaches discipline but also makes you healthy.

1.3 APPLICATIONS

Applications for a Nutrition Assistant Application are vast. Some of the more common ones are:

- Fitness Tracker/ Calorie Tracker
- Medical Apps for patients with high cholesterol or sugar etc.

1.4 ADVANTAGES & DISADVANTAGES

ADVANTAGES:

1. Provides better information on nutritional values by using visual recognition.
2. Can understand if a item is branded and then provide nutritional values of the specific brand
3. Can distinguish between different amount of servings

DISADVANTAGE:

1. Can misclassify similar looking food
2. Cannot determine specific ingredients from food items

CHAPTER 2

LITERATURE SURVEY

2.1 EXISTING PROBLEM

Similar kinds of applications available on the internet are text based. They return you nutritional values based on a text input such as food input but that does not reflect how much calories your food might have.

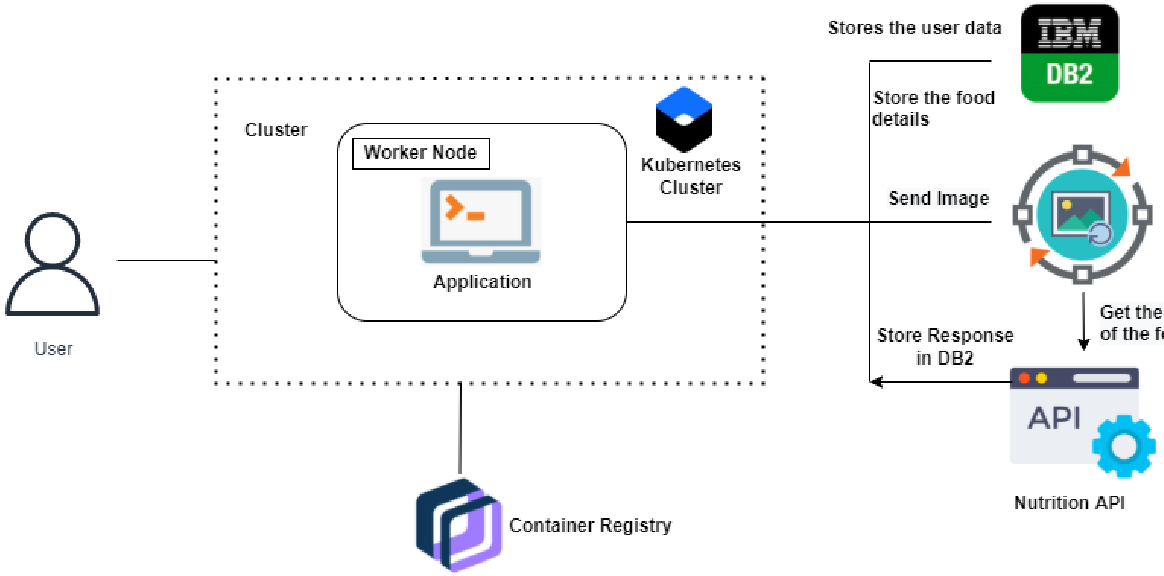
2.2 PROPOSED SOLUTION

Same food might have different calorific values for different ways of cooking. This application is however image based so it can understand more details. If you are not consuming the full dish but only a part of it you can upload the part of food you are consuming and it will return values for only the amount you are consuming. Thus, image-based nutrition management systems perform better.

CHAPTER 3

THEORETICAL ANALYSES

3.1 BLOCK DIAGRAM



3.2 SYSTEM SPECIFICATIONS

HARDWARE REQUIREMENTS

- Windows (minimum 10), Mac OS, Ubuntu
- Ram - 4GB (minimum)
- Hard disk - 256GB (minimum)
- Processor - Intel i3 (minimum), Mac M1

SOFTWARE REQUIREMENTS

- Anaconda Navigator
- VS Code
- Docker
- IBM Cloud

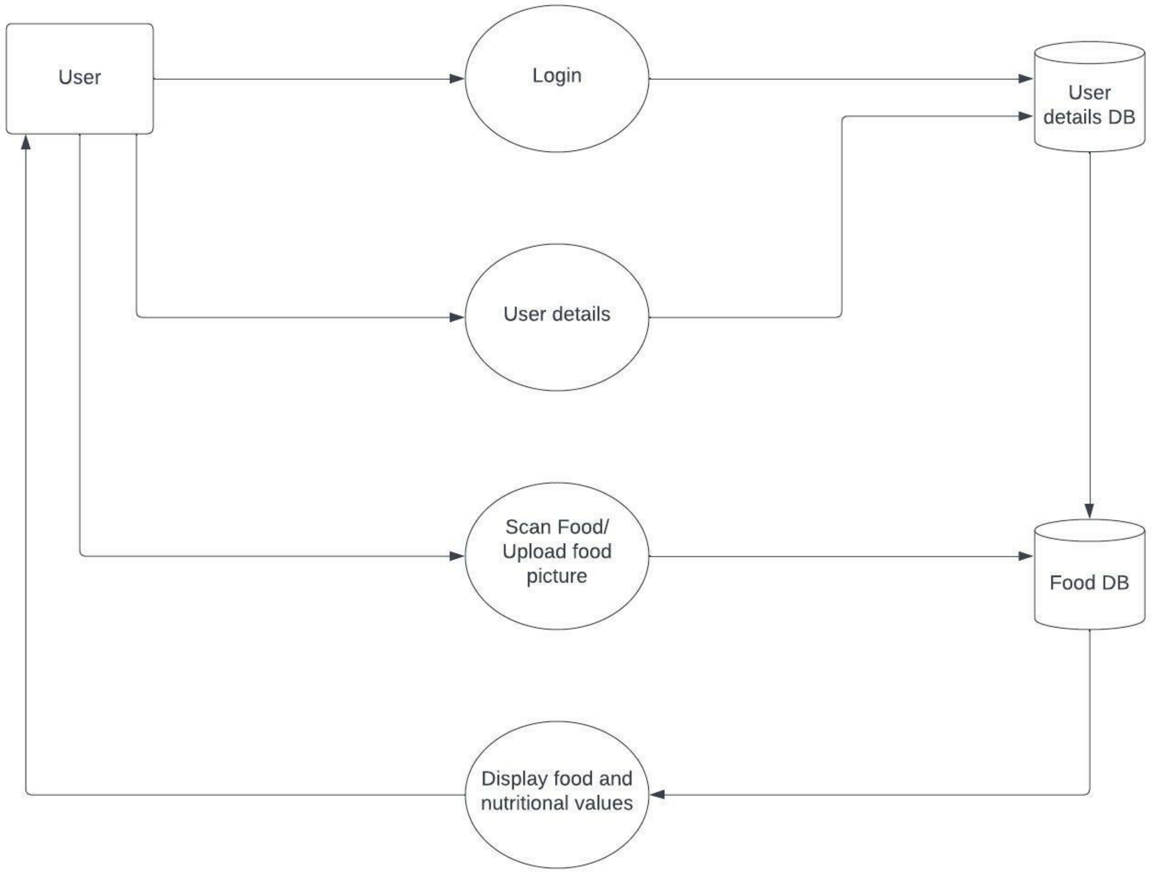
CHAPTER 4

EXPERIMENTAL INVESTIGATIONS

1. Understanding how to insert and fetch data from IBM DB
2. Understanding how to work with APIs - fetch response from various APIs and parse the response they returned (mostly in .json format)
3. Learning how to style using HTML and CSS

CHAPTER 5

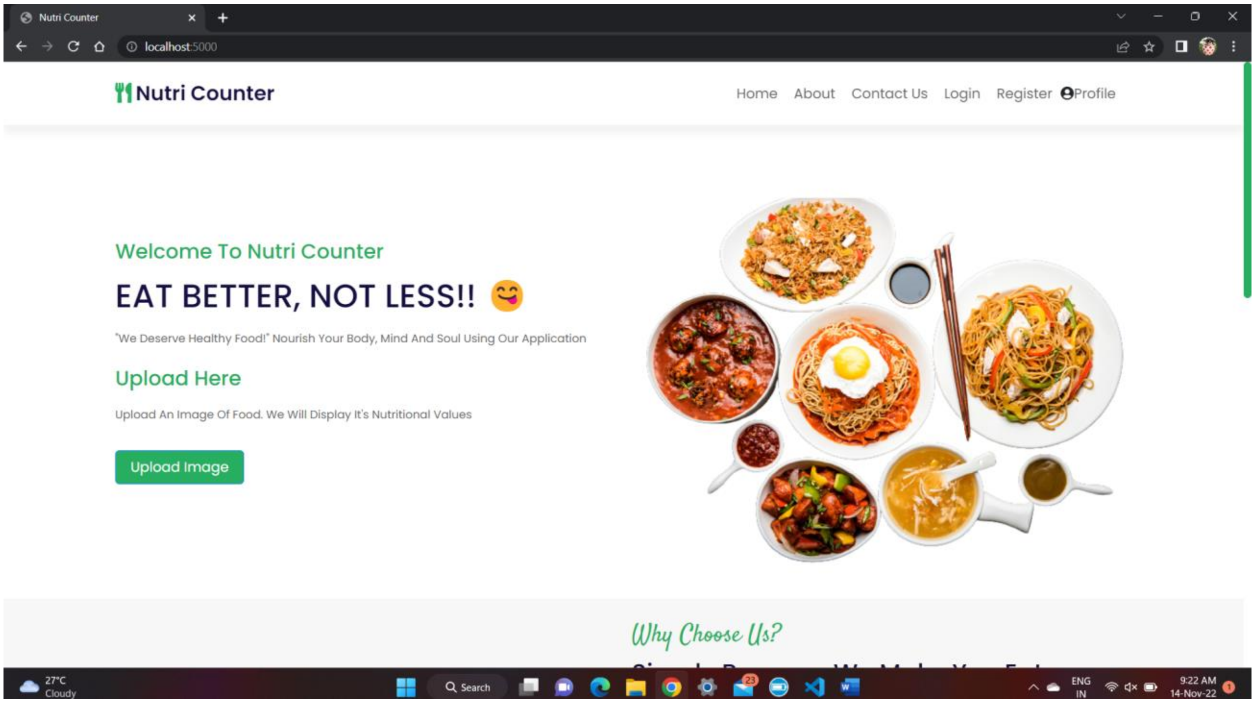
FLOWCHARTS


User
Login
User details DB
User details
Scan Food/ Upload food picture
Food DB
Display food and nutritional values

CHAPTER 6

6.1 RESULTS

HOME PAGE



Welcome To Nutri Counter

EAT BETTER, NOT LESS!! 😋

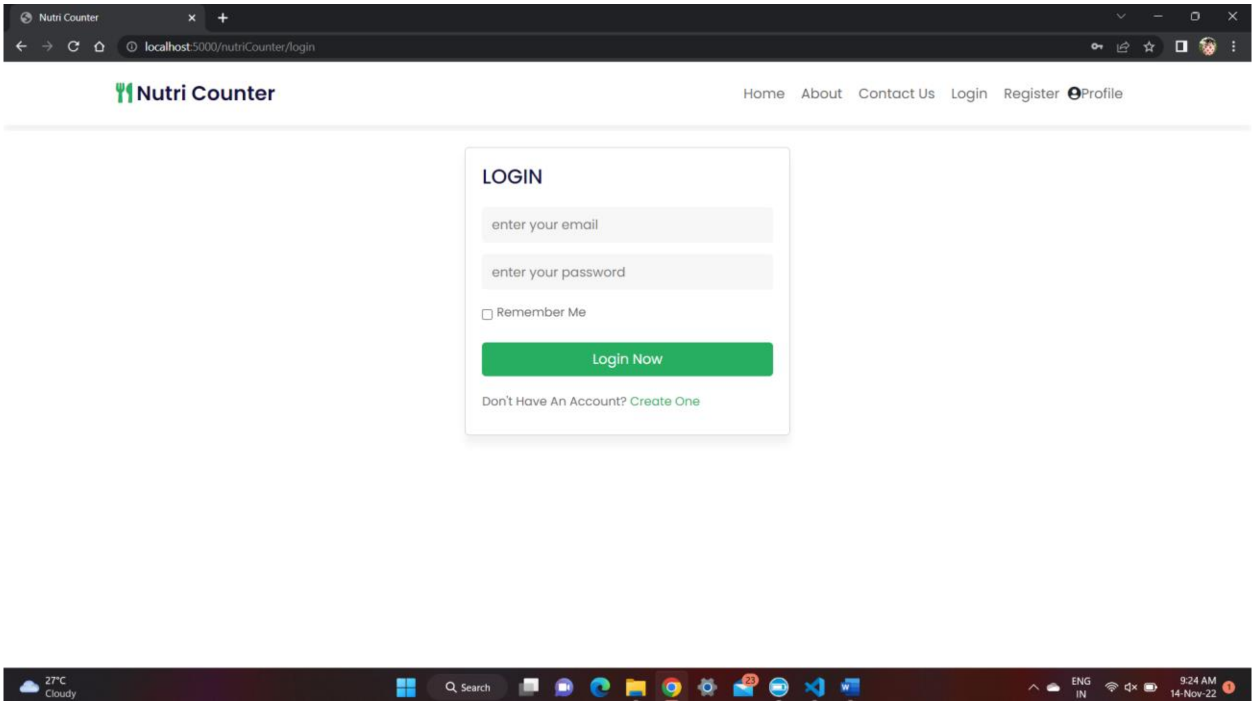
"We Deserve Healthy Food!" Nourish Your Body, Mind And Soul Using Our Application

Upload Here

Upload An Image Of Food. We Will Display It's Nutritional Values

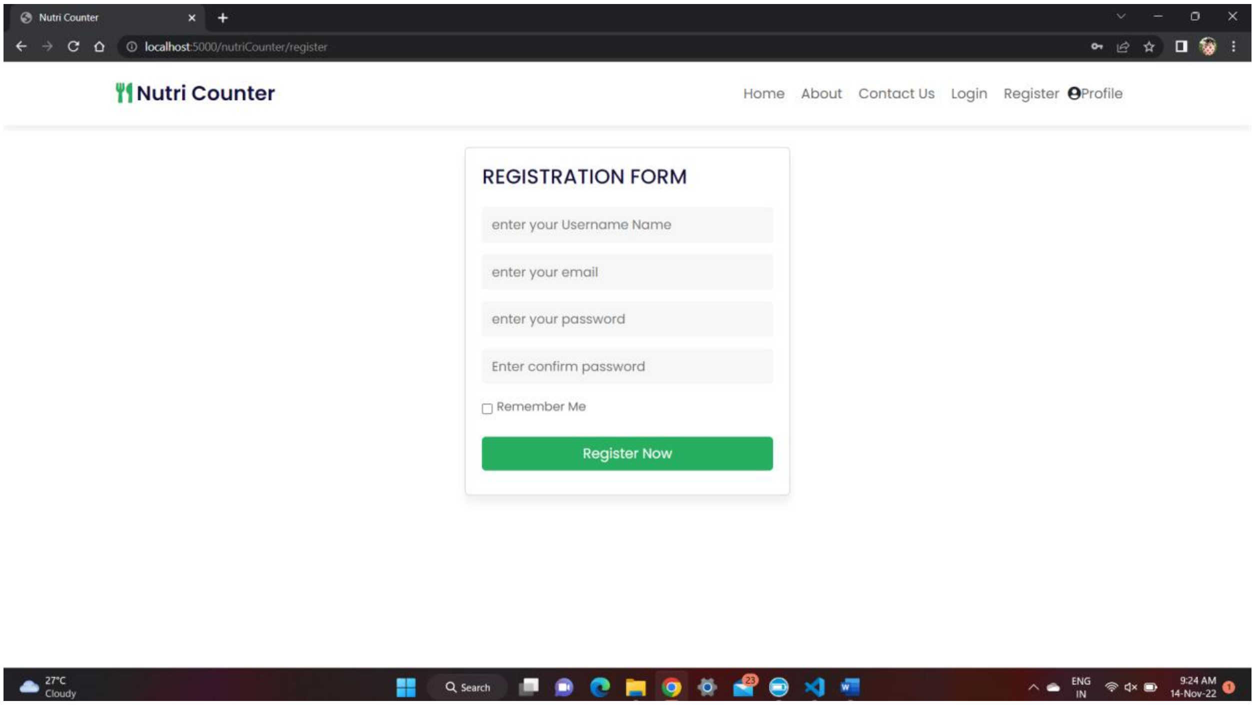
Why Choose Us?

LOGIN PAGE



Don't Have An Account? Create One

REGISTRATION PAGE



REGISTRATION FORM

enter your Username Name

enter your email

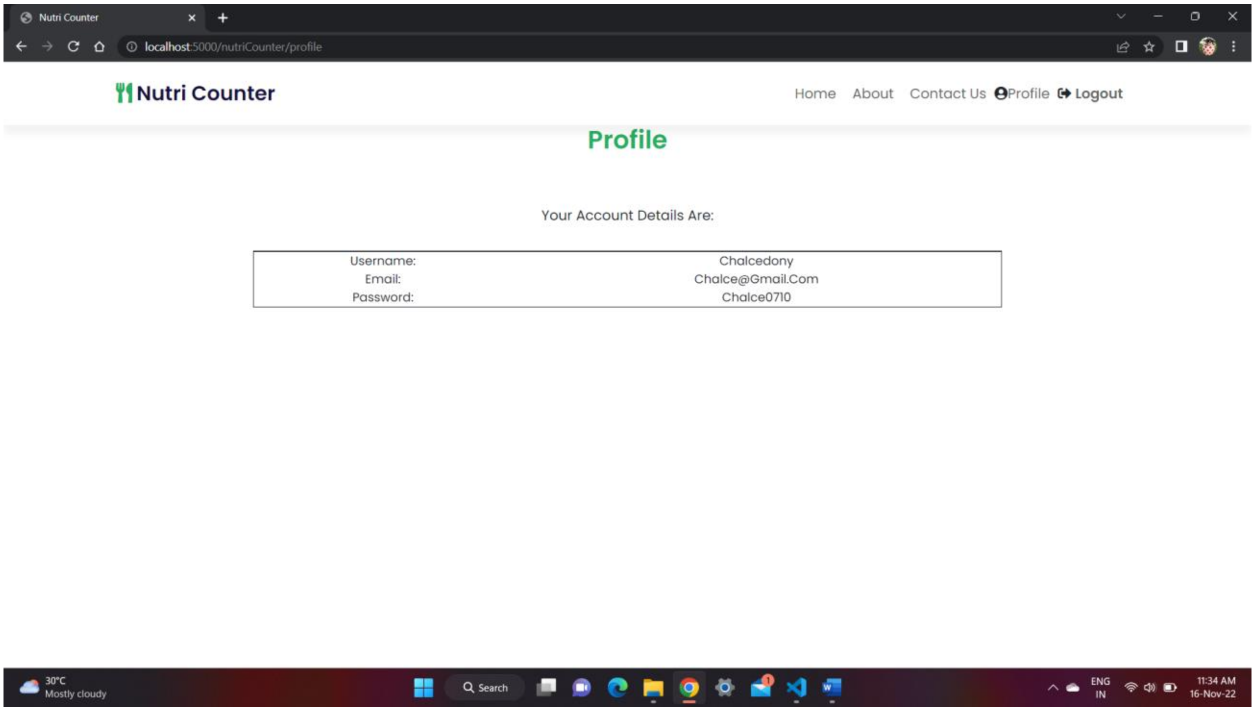
enter your password

Enter confirm password

□ Remember Me

Register Now

PROFILE

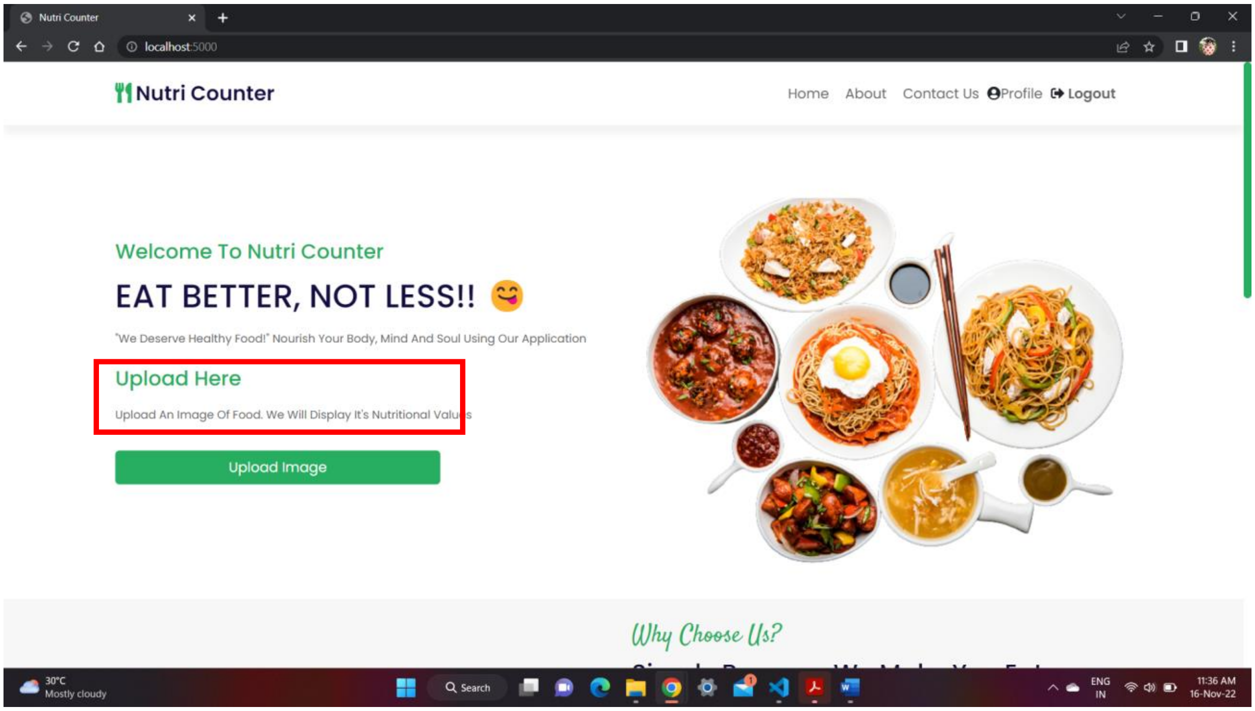


Profile

Your Account Details Are:

Username:	Chalcedony
Email:	Chalce@Gmail.Com
Password:	Chalce0710

UPLOAD



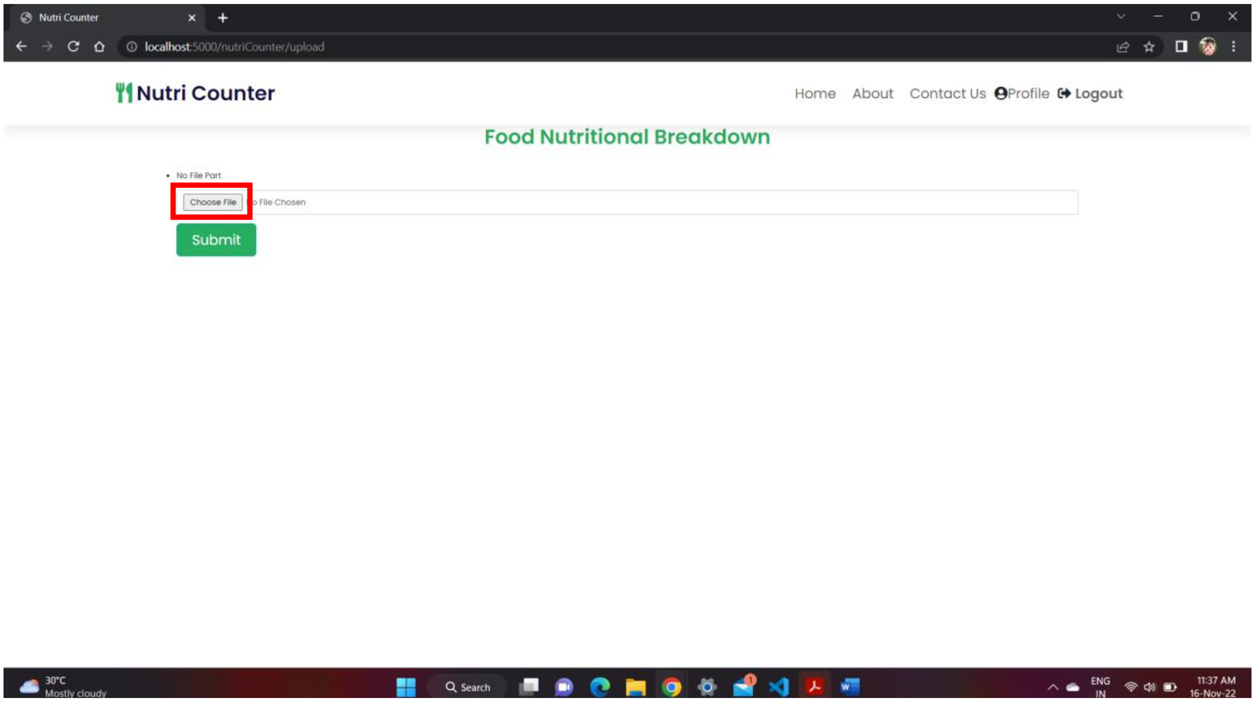
Welcome To Nutri Counter

EAT BETTER, NOT LESS!! 😋

"We Deserve Healthy Food!" Nourish Your Body, Mind And Soul Using Our Application

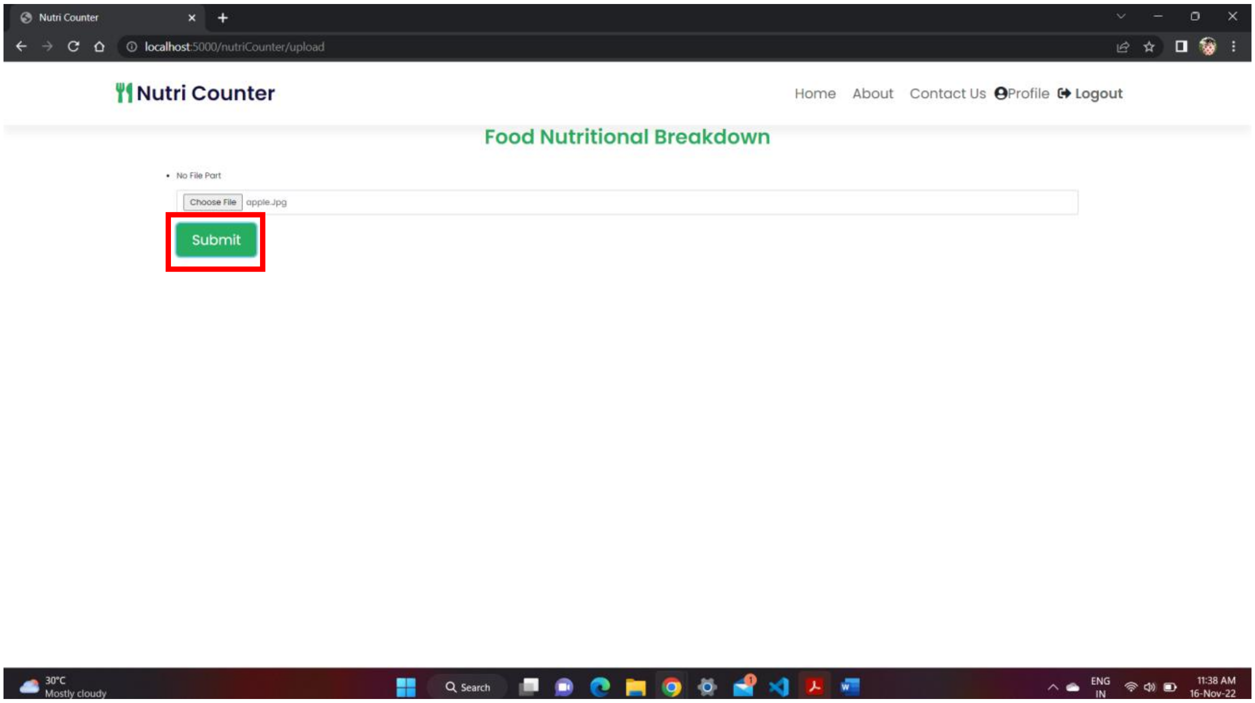
Upload Here

Upload An Image Of Food. We Will Display It's Nutritional Values



Food Nutritional Breakdown

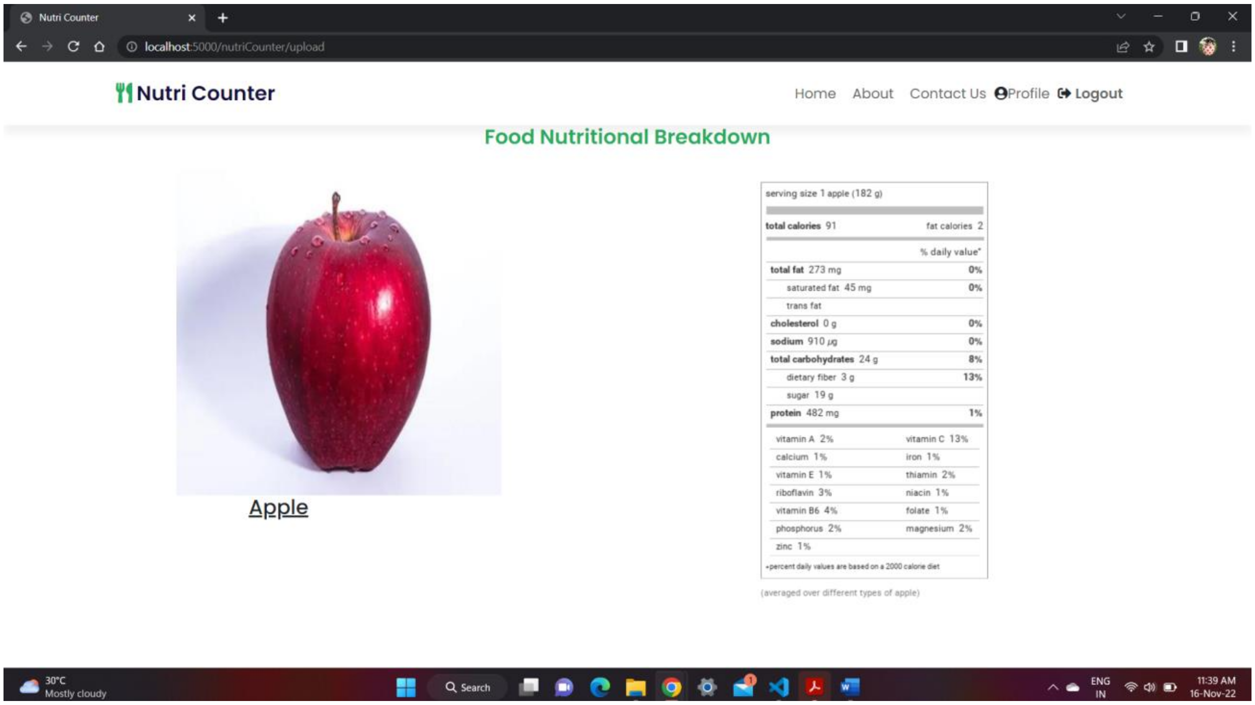
- No File Part



Food Nutritional Breakdown

- No File Part

apple.jpg



Food Nutritional Breakdown

Apple

serving size 1 apple (182 g)

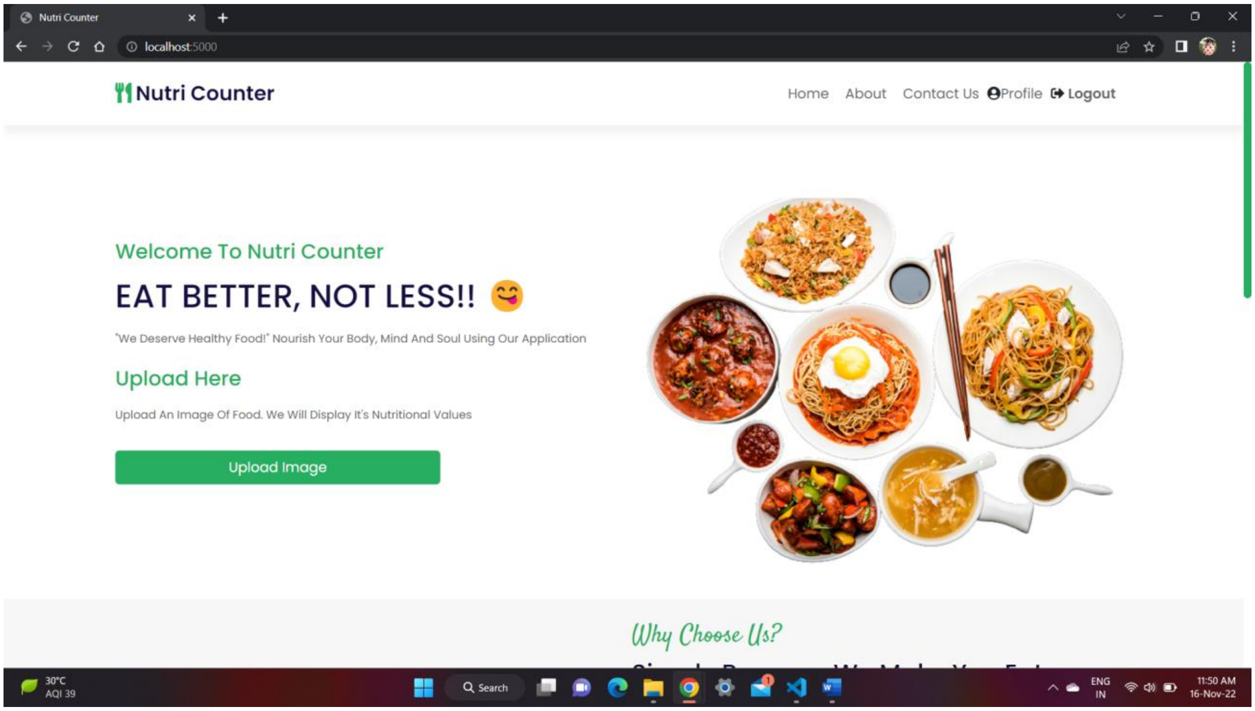
total calories 91	fat calories 2
	% daily value*
total fat 273 mg	0%
saturated fat 45 mg	0%
trans fat	
cholesterol 0 g	0%
sodium 910 μg	0%
total carbohydrates 24 g	8%
dietary fiber 3 g	13%
sugar 19 g	
protein 482 mg	1%

vitamin A 2%	vitamin C 13%
calcium 1%	iron 1%
vitamin E 1%	thiamin 2%
riboflavin 3%	niacin 1%
vitamin B6 4%	folate 1%
phosphorus 2%	magnesium 2%
zinc 1%	

*percent daily values are based on a 2000 calorie diet

(averaged over different types of apple)

LOGOUT



Welcome To Nutri Counter

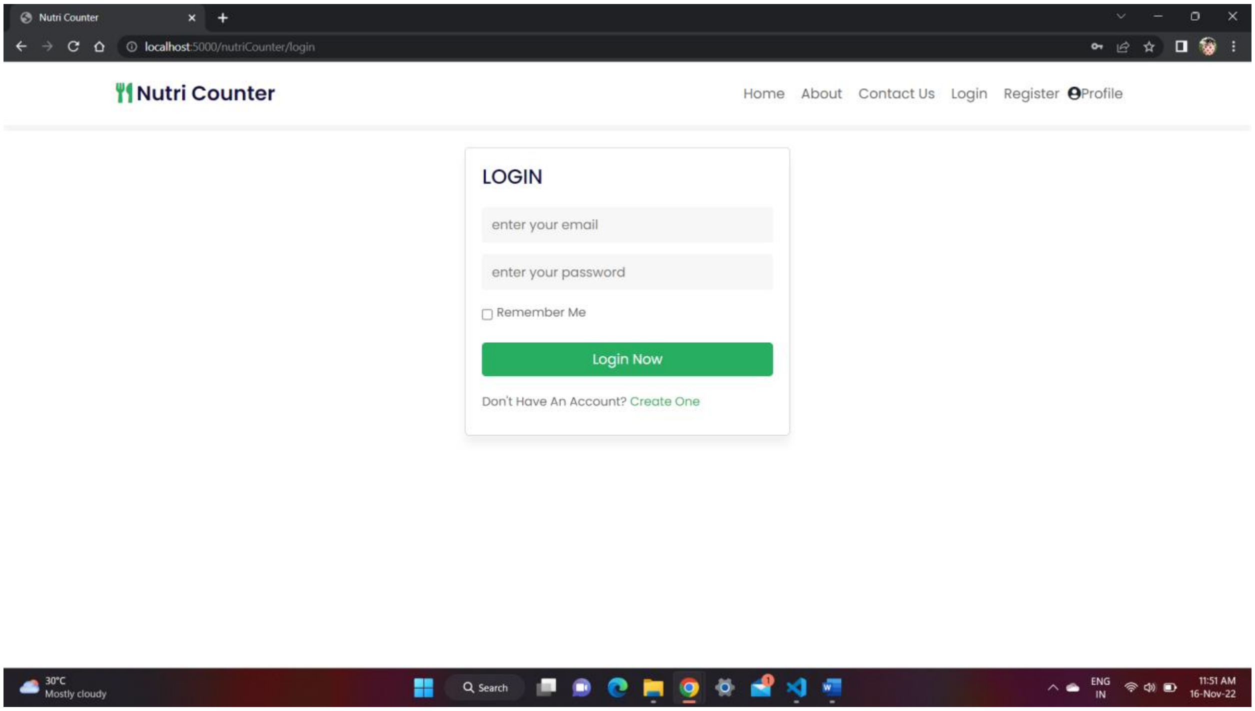
EAT BETTER, NOT LESS!! 😋

"We Deserve Healthy Food!" Nourish Your Body, Mind And Soul Using Our Application

Upload Here

Upload An Image Of Food. We Will Display It's Nutritional Values

Why Choose Us?



LOGIN

enter your email

enter your password

□ Remember Me

Don't Have An Account? Create One

6.2 CONCLUSIONS

This project has been a challenge that was fun to overcome. So many ideas incorporated into a single application makes up for a good challenge. The application is not 100% accurate as foods can look similar. The ingredients in the dishes cannot be distinguished in certain scenarios so the application is not able to predict correctly. Besides these constraints the application provides very good information.

6.3 FUTURE SCOPE

Enhancements that can be made in the future:

1. UI Improvement
2. Scaling the application to handle more requests

APPENDIX

SOURCE CODE

app.py

```
import os

from flask import Flask

from flask import render_template,request,redirect,url_for,flash,session

import re

import os

from werkzeug.utils import secure_filename

from PIL import Image

from ml_model import food_identifier

from food import nutrients

import ibm_db

import json

app = Flask(__name__)

app.secret_key = 'a'

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=b70af05b-76e4-4bca-
a1f5-
23dbb4c6a74e.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=32716;S
ECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=rqn22933
;PWD=v1K08EB1xJ6XXWmS",",")

@app.route('/nutriCounter/login', methods =['GET', 'POST'])

def login():

    global id

    msg = "

    if request.method == 'POST' and 'email' in request.form and 'password' in
request.form:

        email = request.form['email']

        password = request.form['password']

        sql = "SELECT * FROM accounts WHERE email = ? AND password
= ? "

        stmt = ibm_db.prepare(conn,sql)

        ibm_db.bind_param(stmt,1,email)

        ibm_db.bind_param(stmt,2,password)

        ibm_db.execute(stmt)

        account = ibm_db.fetch_assoc(stmt)

        if account:

     # Create session data, we can access this data in other route

            session['loggedin'] = True
```

```
                        session['id'] = request.form['email']

                        session['email'] = request.form['email']

         # Redirect to home page

                        return redirect(url_for('home'))

                else:

                        msg = 'Incorrect username / password !'

        return render_template('login.html', msg = msg)

@app.route('/')

def home():

   # Check if user is loggedin

        login=False

        if 'loggedin' in session:

      # User is loggedin show them the home page

                # User is not loggedin redirect to login page

                login=True

                return render_template('home.html', username=session['email'],
login=login)

        return render_template('home.html', login=login)
```

```
@app.route('/nutriCounter/logout')
def logout():
     # Remove session data, this will log the user out
    session.pop('loggedin', None)
    session.pop('id', None)
    session.pop('email', None)
    # Redirect to login page
    return redirect(url_for('login'))
@app.route('/nutriCounter/register', methods =['GET', 'POST'])
def register():
        msg = ''
        if request.method == 'POST' and 'username' in request.form and 'password' in request.form and 'email' in request.form :
                username = request.form['username']
                password = request.form['password']
                email = request.form['email']
                sql = "SELECT * FROM accounts WHERE username = ?"
                stmt = ibm_db.prepare(conn, sql)
                ibm_db.bind_param(stmt, 1, username)
                ibm_db.execute(stmt)
                account = ibm_db.fetch_assoc(stmt)
                if account:
                    msg = 'Account already exists !'
                elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
                    msg = 'Invalid email address !'
                elif not re.match(r'[A-Za-z0-9]+', username):
                    msg = 'Username must contain only characters and numbers !'
                elif not username or not password or not email:
                    msg = 'Please fill out the form !'
                else:
                    insert_sql = "INSERT INTO accounts(username,
email,password) VALUES (?,?,?)"
                    prep_stmt = ibm_db.prepare(conn, insert_sql)
                    ibm_db.bind_param(prep_stmt, 1, username)
                    ibm_db.bind_param(prep_stmt, 2, email)
                    ibm_db.bind_param(prep_stmt, 3, password)
                    ibm_db.execute(prep_stmt)
                    msg = 'You have successfully registered !'
```

```
                        # print(msg)

                        return redirect(url_for('login'))

        elif request.method == 'POST':

                msg = 'Please fill out the form !'

        return render_template('register.html', msg = msg)

@app.route('/nutriCounter/profile')

def profile():

   # Check if user is loggedin

        login=False

        if 'loggedin' in session:

      # We need all the account info for the user so we can display it on the profile
page

                login=True

                sql = "SELECT * FROM accounts WHERE email = '%s'"
%(session['id'])

                stmt = ibm_db.exec_immediate(conn, sql)

                account = ibm_db.fetch_assoc(stmt)

                while account != False:

                        acc = json.dumps(account)

                        res = json.loads(acc)
```

```
                               return render_template('profile.html', res=res,login=login)
    # User is not loggedin redirect to login page
          return redirect(url_for('login'))
UPLOAD_FOLDER = 'static/uploads/'
ALLOWED_EXTENSIONS = set(['png', 'jpg', 'jpeg', 'gif'])
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
@app.route('/nutriCounter/upload',methods=['GET'])
def upload():
    return render_template('uploads.html',login=True)
@app.route('/nutriCounter/upload', methods=['POST'])
def upload_image():
    if 'file' not in request.files:
        flash('No file part')
        return redirect(request.url)
    file = request.files['file']
    if file.filename == '':
        flash('No image selected for uploading')
        return redirect(request.url)
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        image = Image.open(file)
        # Get the current working directory
        cwd = os.path.dirname(os.path.abspath(__file__))
        file_path=os.path.join(cwd,UPLOAD_FOLDER,filename)
        resized_img = image.resize((400, 400))
        resized_img.save(file_path)
        food_name=food_identifier(file_path)
        nutr=nutrients(food_name)
        print(food_name)
        print(nutr)
        return render_template('uploads.html', filename=filename,
food=food_name,nutr=nutr,login=True)
    else:
        flash('Allowed image types are - png, jpg, jpeg, gif')
        return redirect(request.url)
@app.route('/display/<filename>')
```

```
def display_image(filename):
    return redirect(url_for('static', filename='uploads/' + filename), code=301)
if __name__ == "__main__":
    app.run(debug=True)
```

ml_model.py

```
from clarifai_grpc.channel.clarifai_channel import ClarifaiChannel
from clarifai_grpc.grpc.api import resources_pb2, service_pb2, service_pb2_grpc
from clarifai_grpc.grpc.api.status import status_pb2, status_code_pb2
import os
# Construct the communications channel
channel = ClarifaiChannel.get_grpc_channel()
# Construct the V2Stub object for accessing all the Clarifai API functionality
stub = service_pb2_grpc.V2Stub(channel)
metadata = (('authorization', 'Key ' + 'ab3c8d7291bc4f55a9891519e6aa5fc0'),)
userDataObject = resources_pb2.UserAppIDSet(user_id='clarifaiforme', app_id='main')
def food_identifier(string):
    with open(string, "rb") as f:
        file_bytes = f.read()
    post_model_outputs_response = stub.PostModelOutputs(
        service_pb2.PostModelOutputsRequest(
            user_app_id=userDataObject,  # The userDataObject is created in the
overview and is required when using a PAT
            model_id="food-item-recognition",  # This is model ID of the clarifai/main
General model.
        version_id="1d5fd481e0cf4826aa72ec3ff049e044",  # This is optional.
Defaults to the latest model version.
            inputs=[
                resources_pb2.Input(
                    data=resources_pb2.Data(
                        image=resources_pb2.Image(
                            base64=file_bytes
                        )
                    )
                )
            ]
        ),
        metadata=metadata
    )
    if post_model_outputs_response.status.code != status_code_pb2.SUCCESS:
        print("There was an error with your request!")
        print("\tCode: {}".format(post_model_outputs_response.outputs[0].status.code))
        print("\tDescription: {}".format(post_model_outputs_response.outputs[0].status.description))
        print("\tDetails: {}".format(post_model_outputs_response.outputs[0].status.details))
        raise Exception("Post model outputs failed, status: " + post_model_outputs_response.status.description)
    # Since we have one input, one output will exist here.
    output = post_model_outputs_response.outputs[0]
    print("Predicted concepts:")
    for concept in output.data.concepts:
        print("\t%s %.2f" % (concept.name, concept.value))
    return output.data.concepts[0].name
```

food.py

```
from io import BytesIO
from tkinter import Image
import wolframalpha
import requests
from IPython import display
from PIL import Image
def nutrients(food):
    food=food+"nutritional info"
    app_id="RLWKY9-EUT258WJ5X"
    client=wolframalpha.Client(app_id)
    res=client.query(food)
    url=res.pod[1].subpod.img.src
    response=requests.get(url)
    img=Image.open(BytesIO(response.content))
    ans=next(res.results).text
    return url
```

BIBLIOGRAPHIES

[1]https://www.w3schools.com/html/ - W3schools html guide

[2]https://www.w3schools.com/w3css/defaulT.asp - W3schools CSS

[3]https://cloud.ibm.com/docs/Db2onCloud?topic=Db2onCloud-getting-started – IBM Cloud DB2 Connection docs

[4]https://docs.clarifai.com/api-guide/api-overview/api-clients#client-installation-instructions – Clarifai's AI-Driven Food Detection Model Service API Connection docs