Code:

```
from fastapi import FastAPI
from pydantic import BaseModel,conlist
from typing import List,Optional
import pandas as pd
from model import recommend,output_recommended_recipes


dataset=pd.read_csv('../Data/dataset.csv',compression='gzip')

app = FastAPI()


class params(BaseModel):
    n_neighbors:int=5
    return_distance:bool=False

class PredictionIn(BaseModel):
    nutrition_input:conlist(float, min_items=9, max_items=9)
    ingredients:list[str]=[]
    params:Optional[params]


class Recipe(BaseModel):
    Name:str
    CookTime:str
    PrepTime:str
    TotalTime:str
    RecipeIngredientParts:list[str]
    Calories:float
    FatContent:float
    SaturatedFatContent:float
    CholesterolContent:float
    SodiumContent:float
    CarbohydrateContent:float
    FiberContent:float
    SugarContent:float
    ProteinContent:float
    RecipeInstructions:list[str]

class PredictionOut(BaseModel):
    output: Optional[List[Recipe]] = None
```

```
@app.get("/")
def home():
    return {"health_check": "OK"}


@app.post("/predict/",response_model=PredictionOut)
def update_item(prediction_input:PredictionIn):

recommendation_dataframe=recommend(dataset,prediction_input.nutrition_input,prediction_inp
ut.ingredients,prediction_input.params.dict())
    output=output_recommended_recipes(recommendation_dataframe)
    if output is None:
        return {"output":None}
    else:
        return {"output":output}


import numpy as np
import re
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import NearestNeighbors
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer


def scaling(dataframe):
    scaler=StandardScaler()
    prep_data=scaler.fit_transform(dataframe.iloc[:,6:15].to_numpy())
    return prep_data,scaler

def nn_predictor(prep_data):
    neigh = NearestNeighbors(metric='cosine',algorithm='brute')
    neigh.fit(prep_data)
    return neigh

def build_pipeline(neigh,scaler,params):
    transformer = FunctionTransformer(neigh.kneighbors,kw_args=params)
    pipeline=Pipeline([('std_scaler',scaler),('NN',transformer)])
    return pipeline

def extract_data(dataframe,ingredients):
    extracted_data=dataframe.copy()
    extracted_data=extract_ingredient_filtered_data(extracted_data,ingredients)
```

```
    return extracted_data

def extract_ingredient_filtered_data(dataframe,ingredients):
    extracted_data=dataframe.copy()
    regex_string=''.join(map(lambda x:f'(?=.*{x})',ingredients))

extracted_data=extracted_data[extracted_data['RecipeIngredientParts'].str.contains(regex_strin
g,regex=True,flags=re.IGNORECASE)]
    return extracted_data

def apply_pipeline(pipeline,_input,extracted_data):
    _input=np.array(_input).reshape(1,-1)
    return extracted_data.iloc[pipeline.transform(_input)[0]]

def
recommend(dataframe,_input,ingredients=[],params={'n_neighbors':5,'return_distance':False}):
        extracted_data=extract_data(dataframe,ingredients)
        if extracted_data.shape[0]>=params['n_neighbors']:
            prep_data,scaler=scaling(extracted_data)
            neigh=nn_predictor(prep_data)
            pipeline=build_pipeline(neigh,scaler,params)
            return apply_pipeline(pipeline,_input,extracted_data)
        else:
            return None

def extract_quoted_strings(s):
    # Find all the strings inside double quotes
    strings = re.findall(r'"([^"]*)"', s)
    # Join the strings with 'and'
    return strings

def output_recommended_recipes(dataframe):
    if dataframe is not None:
        output=dataframe.copy()
        output=output.to_dict("records")
        for recipe in output:
            recipe['RecipeIngredientParts']=extract_quoted_strings(recipe['RecipeIngredientParts'])
            recipe['RecipeInstructions']=extract_quoted_strings(recipe['RecipeInstructions'])
    else:
        output=None
    return output


import requests
```

```
import json

class Generator:
    def
__init__(self,nutrition_input:list,ingredients:list=[],params:dict={'n_neighbors':5,'return_distance':
False}):
        self.nutrition_input=nutrition_input
        self.ingredients=ingredients
        self.params=params

    def set_request(self,nutrition_input:list,ingredients:list,params:dict):
        self.nutrition_input=nutrition_input
        self.ingredients=ingredients
        self.params=params

    def generate(self,):
        request={
            'nutrition_input':self.nutrition_input,
            'ingredients':self.ingredients,
            'params':self.params
        }
        response=requests.post(url='http://backend:8080/predict/',data=json.dumps(request))
        return response


import streamlit as st

st.set_page_config(
    page_title="Hello",
    page_icon="👋",
)

st.write("# Welcome to Diet Recommendation System! 👋")

st.sidebar.success("Select a recommendation app.")

st.markdown(
    """
    A diet recommendation web application using content-based approach with Scikit-Learn,
FastAPI and Streamlit.
    You can find more details and the whole project on my
[repo](https://github.com/zakaria-narjis/Diet-Recommendation-System).
    """
)
```

```
import os
import json
import numpy as np
import pandas as pd

from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image

# Flask utils
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer



from splinter import Browser
from bs4 import BeautifulSoup
import pandas as pd
import requests
import os


# Define a flask app
app = Flask(__name__)

# Model saved with Keras model.save()
#MODEL_PATH = os.path.join("models","keras_models",
"model-mobilenet-RMSprop0.0002-001-0.930507-0.647776.h5")
MODEL_PATH = os.path.join("models","keras_models",
"model-mobilenet-RMSprop0.0002-008-0.995584-0.711503.h5")

# Load your trained model
model = load_model(MODEL_PATH)
print("Model loaded successfully !! Check http://127.0.0.1:5000/")

with open(os.path.join("static","food_list", "food_list.json"), "r", encoding="utf8") as f:
    food_labels = json.load(f)
class_names = sorted(food_labels.keys())
label_dict = dict(zip(range(len(class_names)), class_names))

food_calories = pd.read_csv(os.path.join("static","food_list", "Food_calories.csv"))

def prepare_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
```

```
    # Preprocessing the image
    x = image.img_to_array(img) / 255
    x = np.expand_dims(x, axis=0)
    return x


@app.route("/", methods=["GET"])
def Home():
    # Main page
    #Food = mongo.db.collection.find_one()
    return render_template('Know_Before_You_Eat.html')




@app.route("/predict", methods=["GET", "POST"])
def upload():
    data = {}
    if request.method == "POST":
        # Get the file from post request
        f = request.files["image"]

        # Save the file to ./upload_image
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(basepath, "upload_image", secure_filename(f.filename))
        f.save(file_path)

        # Make prediction
        image = prepare_image(file_path)
        preds = model.predict(image)
        predictions = preds.argmax(axis=-1)[0]
        pred_label = label_dict[predictions]

        food_retrieve = food_calories[food_calories["name"]==pred_label]

        food_nutrional_min = food_retrieve["nutritional value min,kcal"]
        food_nutrional_min=np.array(food_nutrional_min)
        food_nutrional_min = str(food_nutrional_min)


        food_nutrional_max = food_retrieve["nutritional value max,kcal"]
        food_nutrional_max=np.array(food_nutrional_max)
        food_nutrional_max = str(food_nutrional_max)
```

```
Unit = food_retrieve["unit"]
Unit=np.array(Unit)
Unit = str(Unit)

Calories = food_retrieve["average cal"]
Calories=np.array(Calories)
Calories = str(Calories)

data = pred_label

if data=="beef carpaccio":
   data="carpaccio"
elif data=="cheese plate":
    data="cheese"
elif data=="chicken quesadilla":
    data="quesadilla"
elif data=="chicken wings":
    data="Buffalo wing"
elif data=="grilled salmon":
    data="Salmon#As_food"
elif data=="lobster roll sandwich":
    data="lobster roll"
elif data=="strawberry shortcake":
    data="Shortcake#Strawberry_shortcake"

path={'executable_path':'/usr/local/bin/chromedriver'}
browser=Browser('chrome',**path,headless=False)
# browser=Browser('chrome',path,headless=True)

if data=="tuna tartare":
    url="http://ahealthylifeforme.com/tuna-tartare-recipe/"
    browser.visit(url)
    html=browser.html
    soup=BeautifulSoup(html,"html.parser")
    var=soup.select_one('div.entry-content')
    description=var.select('p')
else:
    url="https://en.wikipedia.org/wiki/"
    browser.visit(url+data)
    html=browser.html
    soup=BeautifulSoup(html,"html.parser")
    var=soup.select_one('div.mw-parser-output')
    description=var.select('p')
    nutri=soup.select_one('table.infobox')
```

```
        if (data=="greek salad" or data=="oysters" or data=="smoked scallop" or data=="paella"):
            output=description[1].text
        elif data=="mussels" :
            output=description[2].text
        elif data=="Salmon#As_food":
            output=description[3].text
        else:
            if description[0].text!='\n':
                output=description[0].text
            elif description[0].text=='\n' and description[1].text!='\n':
                output=description[1].text
            elif description[1].text=='\n' and description[2].text!='\n':
                output=description[2].text
        output
        description = output
        browser.quit()


        return "<center><i><h4>" + pred_label.title()+" </h4></i>
"+"<b><h3>Probability</h3></b><h4>"+str(preds.max(axis=-1)[0]) + '\n' +
"</h4><br><br><b><h4 class=\"desc\">" +\
        description + "</h4><br><br>" +\
        "<div class=\"heading-section\"><h2 class=\"mb-4\"><span>Nutrional
Facts</span></h2></div><hr></hr>" + \
        "<h5><b>Nutrional Value - Min (kcal)  :  </b>" + food_nutrional_min + '\n' +
"<br><br>" + \
        "<b>Nutrional Value - Max (kcal)  :  </b>" + food_nutrional_max + '\n' +
"<br><br>" + \
        "<b> Avg Calories  :  </b>" + Calories + "<br><br>" + \
        "<b> Unit  :  </b>" + Unit + '\n' + "</h5></center> <br><br>" + \
        "<div id=\"Recipe\" class=\"heading-section\"><h2 class=\"mb-4\"><span>Recipe -
Cookbook </span></h2></div><hr></hr>" + \
        str(nutri)



    return None


if __name__ == "__main__":
    # Serve the app with gevent
    http_server = WSGIServer(("0.0.0.0", 5000), app)
    http_server.serve_forever()
```