def generate\_diet\_plan(user\_profile):

first\_name = user\_profile.user.first\_name

dietary\_preferences = user\_profile.dietary\_preferences

health\_conditions = user\_profile.health\_conditions

# Example enhanced logic considering health conditions

if 'diabetes' in health\_conditions.lower():

plan\_details = f"{first\_name}, since you have diabetes, your diet plan includes low-sugar meals following a {dietary\_preferences} diet."

else:

plan\_details = f"Based on your preferences, {first\_name}, your diet plan is to follow a {dietary\_preferences} diet."

return plan\_details

def generate\_exercise\_plan(user\_profile):

first\_name = user\_profile.user.first\_name

activity\_level = user\_profile.activity\_level

health\_goals = user\_profile.health\_goals

# Example enhanced logic considering health goals

if 'weight loss' in health\_goals.lower():

plan\_details = f"{first\_name}, to achieve your goal of weight loss, follow a {activity\_level} exercise routine with additional cardio sessions."

else:

plan\_details = f"Based on your activity level, {first\_name}, your exercise plan is to follow a {activity\_level} exercise routine."

return plan\_details

# myapp/views.py

from django.http import JsonResponse

from django.views.decorators.csrf import csrf\_exempt

from django.contrib.auth.models import User

from .models import UserProfile, DietPlan, ExercisePlan, HealthRecommendation

import json

import requests

def get\_diet\_recommendation(user\_profile):

if user\_profile.health\_goals == "weight\_loss":

return DietPlan.objects.filter(name="Low-Carb Diet").first()

elif user\_profile.health\_goals == "muscle\_gain":

return DietPlan.objects.filter(name="High-Protein Diet").first()

elif user\_profile.dietary\_preferences == "vegetarian":

return DietPlan.objects.filter(name="Vegetarian Diet").first()

elif user\_profile.dietary\_preferences == "vegan":

return DietPlan.objects.filter(name="Vegan Diet").first()

return DietPlan.objects.filter(name="Balanced Diet").first()

def get\_exercise\_recommendation(user\_profile):

if user\_profile.activity\_level == "sedentary":

return ExercisePlan.objects.filter(name="Beginner Workout").first()

elif user\_profile.activity\_level == "moderately\_active":

return ExercisePlan.objects.filter(name="Intermediate Workout").first()

elif user\_profile.health\_goals == "weight\_loss":

return ExercisePlan.objects.filter(name="Cardio Workout").first()

elif user\_profile.health\_goals == "muscle\_gain":

return ExercisePlan.objects.filter(name="Strength Training").first()

return ExercisePlan.objects.filter(name="General Fitness Workout").first()

def get\_real\_time\_articles(user\_profile):

tags = user\_profile.health\_goals

response = requests.get(f'https://newsapi.org/v2/everything?q={tags}&apiKey=YOUR\_NEWS\_API\_KEY')

articles = response.json().get('articles', [])[:5] # Limit to 5 articles

return articles

def get\_real\_time\_images(user\_profile):

tags = user\_profile.health\_goals

response = requests.get(f'https://api.unsplash.com/search/photos?query={tags}&client\_id=YOUR\_UNSPLASH\_ACCESS\_KEY')

images = response.json().get('results', [])[:5] # Limit to 5 images

return images

def get\_real\_time\_videos(user\_profile):

tags = user\_profile.health\_goals

response = requests.get(f'https://www.googleapis.com/youtube/v3/search?part=snippet&q={tags}&key=YOUR\_YOUTUBE\_API\_KEY')

videos = response.json().get('items', [])[:5] # Limit to 5 videos

return videos

@csrf\_exempt

def create\_user(request):

if request.method == 'POST':

try:

data = json.loads(request.body.decode('utf-8'))

phone = data.get('phone')

email = data.get('email')

first\_name = data.get('first\_name')

last\_name = data.get('last\_name')

date\_of\_birth = data.get('date\_of\_birth')

gender = data.get('gender')

password = data.get('password')

weight = data.get('weight')

height = data.get('height')

activity\_level = data.get('activity\_level')

dietary\_preferences = data.get('dietary\_preferences')

health\_conditions = data.get('health\_conditions')

medical\_history = data.get('medical\_history')

health\_goals = data.get('health\_goals')

membership\_status = data.get('membership\_status')

user, created = User.objects.get\_or\_create(email=email, defaults={

'username': email,

'first\_name': first\_name,

'last\_name': last\_name,

})

if created:

user.set\_password(password)

user.save()

profile, created = UserProfile.objects.update\_or\_create(

user=user,

defaults={

'weight': weight,

'height': height,

'activity\_level': activity\_level,

'dietary\_preferences': dietary\_preferences,

'health\_conditions': health\_conditions,

'medical\_history': medical\_history,

'health\_goals': health\_goals,

'membership\_status': membership\_status,

}

)

diet\_recommendation = get\_diet\_recommendation(profile)

exercise\_recommendation = get\_exercise\_recommendation(profile)

articles = get\_real\_time\_articles(profile)

images = get\_real\_time\_images(profile)

videos = get\_real\_time\_videos(profile)

recommendation = HealthRecommendation.objects.create(

user=user,

diet\_plan=diet\_recommendation,

exercise\_plan=exercise\_recommendation

)

response\_data = {

'status': 200,

'msg': 'User created successfully',

'diet\_plan': {

'name': diet\_recommendation.name,

'description': diet\_recommendation.description,

'calories\_per\_day': diet\_recommendation.calories\_per\_day,

'protein\_per\_day': diet\_recommendation.protein\_per\_day,

'carbs\_per\_day': diet\_recommendation.carbs\_per\_day,

'fats\_per\_day': diet\_recommendation.fats\_per\_day,

} if diet\_recommendation else None,

'exercise\_plan': {

'name': exercise\_recommendation.name,

'description': exercise\_recommendation.description,

'frequency\_per\_week': exercise\_recommendation.frequency\_per\_week,

'duration\_per\_session': exercise\_recommendation.duration\_per\_session,

'intensity\_level': exercise\_recommendation.intensity\_level,

} if exercise\_recommendation else None,

'articles': articles,

'images': images,

'videos': videos

}

return JsonResponse(response\_data, status=201)

except json.JSONDecodeError:

return JsonResponse({'status': 400, 'msg': 'Invalid JSON'}, status=400)

return JsonResponse({'status': 405, 'msg': 'Invalid request method'}, status=405)