# Source Code of App: core

## admin.py

```
from django.contrib import admin

# Register your models here.
```

## apps.py

```
from django.apps import AppConfig


class CoreConfig(AppConfig):
    default_auto_field = "django.db.models.BigAutoField"
    name = "core"
```

## context_processors.py

```
from .models import Restaurant

def user_restaurant(request):
    if request.user.is_authenticated:
        restaurant = Restaurant.objects.filter(owner=request.user).first()
    else:
        restaurant = None
    return {'restaurant': restaurant}
```

## forms.py

```
from django import forms
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm
from .models import *
from django_select2.forms import ModelSelect2MultipleWidget

class RegisterForm(UserCreationForm):
    email = forms.EmailField(required=True)

    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']

class ProfileForm(forms.ModelForm):
    email = forms.EmailField(required=True)
    username = forms.CharField(required=True)

    class Meta:
        model = Profile
        fields = ['first_name', 'last_name', 'profile_picture', 'email', 'username']  # เพิ่ม profile_picture

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # ตั้งค่าเริ่มต้นสำหรับ email และ username จาก instance ของ user
        self.fields['email'].initial = self.instance.user.email
        self.fields['username'].initial = self.instance.user.username

    def save(self, commit=True):
        # บันทึกข้อมูลฟอร์ม โดยใช้ commit=False เพื่อให้สามารถแก้ไขโปรไฟล์และ user ได้ก่อน
        profile = super().save(commit=False)

        # ดึงข้อมูล user ที่เชื่อมโยงกับโปรไฟล์
        user = profile.user

        # อัปเดตข้อมูล user จากข้อมูลใหม่ในฟอร์ม
        user.email = self.cleaned_data['email']
        user.username = self.cleaned_data['username']

        # ตรวจสอบว่ามีการอัปโหลดรูปภาพใหม่หรือไม่
        if self.cleaned_data.get('profile_picture'):
            profile.profile_picture = self.cleaned_data['profile_picture']

        # บันทึกข้อมูลทั้งโปรไฟล์และผู้ใช้
        if commit:
            user.save()  # บันทึกข้อมูล user ก่อน
            profile.save()  # บันทึกข้อมูลโปรไฟล์

        return profile

class RestaurantForm(forms.ModelForm):
    class Meta:
        model = Restaurant
        fields = [
            'name', 'description', 'location', 'contact_info',
            'opening_hours', 'social_media', 'images', 'categories', 'latitude', 'longitude'
        ]
        widgets = {
            'categories': ModelSelect2MultipleWidget(
                queryset=RestaurantCategory.objects.all(),
                search_fields=['name__icontains'],  # ทำให้สามารถค้นหาได้
                attrs={'data-placeholder': 'เลือกประเภท...'},
            ),
            'latitude': forms.NumberInput(attrs={'step': 'any', 'placeholder': 'กรุณากรอกละติจูด'}),
            'longitude': forms.NumberInput(attrs={'step': 'any', 'placeholder': 'กรุณากรอกลองจิจูด'})
        }

class RestaurantImageForm(forms.ModelForm):
    class Meta:
        model = RestaurantImage
        fields = ['image']
        widgets = {
            'image': forms.ClearableFileInput(attrs={
                'class': 'border rounded-lg p-2 w-full',
                'accept': 'image/*',
            }),
        }



class FoodForm(forms.ModelForm):
    class Meta:
        model = Food
        fields = ['name', 'description', 'price', 'category', 'image']

    # ใช้ MultipleChoiceField หรือ ModelMultipleChoiceField เพื่อรองรับหลายหมวดหมู่
    category = forms.ModelMultipleChoiceField(
        queryset=Category.objects.all(),
        widget=forms.CheckboxSelectMultiple,  # ใช้ Checkbox เพื่อให้เลือกหลายหมวดหมู่ได้
        required=True
```

```
    )


class FoodFilterForm(forms.Form):
    category = forms.ModelMultipleChoiceField(
        queryset=Category.objects.all(),
        required=False,
        label="ประเภทอาหาร",
        widget=forms.SelectMultiple(attrs={
            'class': 'select2',  # ใช้ Select2
            'id': 'category-select',
        }),
    )
    min_price = forms.DecimalField(
        required=False,
        min_value=0,
        label="ราคาขั้นต่ำ",
        widget=forms.NumberInput(attrs={
            'step': '0.01',
            'class': 'form-input',
        }),
    )
    max_price = forms.DecimalField(
        required=False,
        min_value=0,
        label="ราคาสูงสุด",
        widget=forms.NumberInput(attrs={
            'step': '0.01',
            'class': 'form-input',
        }),
    )
    rating = forms.ChoiceField(
        required=False,
        label="คะแนนร้านอาหาร",
        choices=[('', 'ทุกคะแนน'), ('5', '5 ดาว'), ('4', '4 ดาว'), ('3', '3 ดาว'), ('2', '2 ดาว'), ('1', '1 ดาว')],
        widget=forms.Select(attrs={
            'class': 'form-input',
        }),
    )
    distance = forms.DecimalField(
        required=False,
        min_value=0,
        label="ระยะทาง (กิโลเมตร)",
        widget=forms.NumberInput(attrs={
            'step': '0.1',
            'class': 'form-input',
        }),
    )


class ForumForm(forms.ModelForm):
    class Meta:
        model =Forum
        fields = ['title','content','image']


class ForumCommentForm(forms.ModelForm):
    class Meta:
        model = ForumComment
        fields = ['content']  # ใช้เฉพาะฟิลด์สำหรับเนื้อหา
        widgets = {
            'content': forms.Textarea(attrs={
                'class': 'form-control bg-gray-100 border border-gray-300 rounded-lg p-2 w-full',
                'placeholder': 'Edit your comment...',
                'rows': 4,
            }),
        }

class ReviewForm(forms.ModelForm):
    class Meta:
        model = Review
        fields = ['rating', 'comment']
        widgets = {
            'rating': forms.NumberInput(attrs={
                'min': 1,
                'max': 5,
                'step': 1,
                'class': 'form-control',
                'placeholder': 'คะแนน (1-5)'
            }),
            'comment': forms.Textarea(attrs={
                'class': 'form-control',
                'placeholder': 'ความคิดเห็นของคุณ...'
            })
        }
```

## models.py

```
from django.db import models
from django.contrib.auth.models import User

# โมเดลโปรไฟล์
class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    first_name = models.CharField(max_length=255)
    last_name = models.CharField(max_length=255)
    profile_picture = models.ImageField(upload_to='profile_pictures/', null=True, blank=True)
    latitude = models.FloatField(null=True, blank=True)  # เพิ่มฟิลด์ latitude
    longitude = models.FloatField(null=True, blank=True)  # เพิ่มฟิลด์ longitude

    def __str__(self):
        return f'{self.user.username}'


# โมเดลร้านอาหาร
class RestaurantCategory(models.Model):
    name = models.CharField(max_length=255)

    def __str__(self):
        return self.name

class Restaurant(models.Model):
    owner = models.OneToOneField(User, on_delete=models.CASCADE)
    name = models.CharField(max_length=255)
    description = models.TextField()
    location = models.TextField(max_length=255)
    contact_info = models.CharField(max_length=255)
    opening_hours = models.TextField()
    social_media = models.TextField(null=True, blank=True)
    images = models.ImageField(upload_to='restaurant_images/', null=True, blank=True)
    categories = models.ManyToManyField(RestaurantCategory, related_name='restaurants')  # เปลี่ยนชื่อ field
    saved_by = models.ManyToManyField(User, related_name='saved_restaurants', blank=True)  # ฟิลด์สำหรับบันทึกร้านอาหาร

    latitude = models.FloatField(null=True, blank=True)
```

```
    longitude = models.FloatField(null=True, blank=True)

    def __str__(self):
        return self.name


class RestaurantImage(models.Model):
    restaurant = models.ForeignKey(Restaurant, related_name='additional_images', on_delete=models.CASCADE)
    image = models.ImageField(upload_to='restaurant_additional_images/',null=True, blank=True)

    def __str__(self):
        return f'{self.restaurant.name} - Image'


# หมวดหมู่หลัก
class Category(models.Model):
    name = models.CharField(max_length=255)

    def __str__(self):
        return self.name

# หมวดหมู่ย่อยที่เชื่อมกับหมวดหมู่หลัก

# โมเดลอาหาร ที่เชื่อมกับร้านอาหาร
class Food(models.Model):
    restaurant = models.ForeignKey(Restaurant, on_delete=models.CASCADE, related_name='foods')  # เชื่อมกับร้านอาหาร
    name = models.CharField(max_length=255)
    description = models.TextField(null=True, blank=True)
    price = models.DecimalField(max_digits=6, decimal_places=2)
    category = models.ManyToManyField(Category, related_name='foods')  # เปลี่ยนเป็น ManyToManyField
    image = models.ImageField(upload_to='food_images/', null=True, blank=True)  # รูปภาพอาหาร

    def __str__(self):
        return f'{self.name} - {self.restaurant.name}'


# โมเดลความคิดเห็นร้านอาหาร
class Review(models.Model):
    restaurant = models.ForeignKey(Restaurant, on_delete=models.CASCADE)  # เชื่อมกับร้านอาหาร
    user = models.ForeignKey(User, on_delete=models.CASCADE)  # เชื่อมกับผู้ใช้
    rating = models.IntegerField()  # คะแนนที่ผู้ใช้ให้ (1-5 ดาว)
    comment = models.TextField()  # ข้อความแสดงความคิดเห็น
    created_at = models.DateTimeField(auto_now_add=True)  # วันที่และเวลาที่แสดงความคิดเห็น

    def __str__(self):
        return f'Review by {self.user.username} on {self.restaurant.name}'



# โมเดลกระทู้
class Forum(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=255)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    image = models.ImageField(upload_to='forum_images/', null=True, blank=True)
    saved_by = models.ManyToManyField(User, related_name='saved_forums', blank=True)  # เพิ่มฟิลด์นี้

    def __str__(self):
        return self.title

# โมเดลความคิดเห็นในกระทู้
class ForumComment(models.Model):
    forum = models.ForeignKey(Forum, on_delete=models.CASCADE, related_name='comments')
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Comment by {self.user} on {self.forum}"




class LikeDislikeFood(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='like_dislike_foods')
    food = models.ForeignKey('Food', on_delete=models.CASCADE, related_name='likes_dislikes')
    liked = models.BooleanField()  # True = ชอบ, False = ไม่ชอบ
    timestamp = models.DateTimeField(auto_now_add=True)  # บันทึกเวลาที่ชอบ/ไม่ชอบ
    source = models.CharField(max_length=20, choices=[('random', 'Random'), ('recommend', 'Recommend')], default='random')

    def __str__(self):
        return f"{self.user.username} {'ชอบ' if self.liked else 'ไม่ชอบ'} {self.food.name} ({self.source})"
```

## tests.py

```
from django.test import TestCase

# Create your tests here.
```

## urls.py

```
from django.contrib.auth import views as auth_views
from django.urls import path, include
from core import views
from django.conf import settings
from django.conf.urls.static import static

from core.views import app_to_pdf

urlpatterns = [
    path('login/', auth_views.LoginView.as_view(template_name='core/login.html'), name='login'),
    path('register/', views.register, name='register'),
    path('logout/', auth_views.LogoutView.as_view(next_page='login'), name='logout'),
    path('', views.home, name='home'),  # หน้าแรก

    # ลืมรหัสผ่าน
    path('password_reset/', auth_views.PasswordResetView.as_view(template_name='core/resetpassword/password_reset.html'), name='password_reset'),
    path('password_reset/done/', auth_views.PasswordResetDoneView.as_view(template_name='core/resetpassword/password_reset_done.html'), name='password_reset_done'),
    path('reset///', auth_views.PasswordResetConfirmView.as_view(template_name='core/resetpassword/password_reset_confirm.html'), name='password_reset_confirm'),
    path('reset/done/', auth_views.PasswordResetCompleteView.as_view(template_name='core/resetpassword/password_reset_complete.html'), name='password_reset_complete'),

    # หน้าโปรไฟล์
    path('profile/', views.profile_view, name='profile_view'),  # ชื่อ profile_view
    # แก้ไขโปรไฟล์
    path('profile/edit/', views.profile_edit, name='profile_edit'),  # ชื่อ profile_edit

    # สุ่มอาหาร
    path('random-food/', views.random_food, name='random_food'),

    # ร้านอาหาร
    path('restaurants/', views.restaurant_list, name='restaurant_list'),
    path('restaurant/create/', views.restaurant_create, name='restaurant_create'),
```

```
    path('restaurant//', views.restaurant_detail, name='restaurant_detail'),
    path('restaurant//edit/', views.restaurant_edit, name='restaurant_edit'),
    path('restaurant//delete/', views.delete_restaurant, name='delete_restaurant'),
    path('restaurant//delete_image//', views.delete_image, name='delete_image'),

    path('restaurant//add_image/', views.add_restaurant_image, name='add_restaurant_image'),


    # เพิ่ม URL สำหรับการจัดการเมนูอาหาร
    path('restaurant//add_food/', views.add_food, name='add_food'),  # เพิ่มเมนูอาหาร
    path('restaurant//edit_food//', views.edit_food, name='edit_food'),  # แก้ไขเมนูอาหาร
    path('restaurant//delete_food//', views.delete_food, name='delete_food'),
    path('choose_food//', views.choose_food, name='choose_food'),# ลบเมนูอาหาร
    path('foods/', views.food_list, name='food_list'),  # เพิ่มบรรทัดนี้

    # กระทู้
    path('forum/', views.forum_list, name='forum'),
    path('forum/create/', views.create_forum, name='forum_create'),
    path('forum//', views.forum_detail, name='forum_detail'),
    #path('forum//edit/', views.edit_forum, name='edit_forum'),
   # path('forum//delete/', views.delete_forum, name='delete_forum'),
    path('forum/my/', views.my_forums, name='my_forums'),
    path('forum//edit/', views.forum_edit, name='forum_edit'),
    path('forum//delete/', views.forum_delete, name='forum_delete'),
    path('forum//comment/', views.add_comment, name='add_comment'),  # ส่งความคิดเห็น
    path('comments//delete/', views.delete_comment, name='delete_comment'),
    path('comments//edit/', views.edit_comment, name='edit_comment'),
    path("select2/", include("django_select2.urls")),
    path('restaurant//review/', views.add_review, name='add_review'),
    path('review//delete/', views.delete_review, name='delete_review'),
    path('review//edit/', views.edit_review, name='edit_review'),
    path('restaurant//toggle-save/', views.toggle_save_restaurant, name='toggle_save_restaurant'),
    path('forum//toggle_save/', views.toggle_save_forum, name='toggle_save_forum'),


    path('admin-dashboard/', views.admin_dashboard, name='admin_dashboard'),
    path('manage-users/', views.manage_users, name='manage_users'),  # หน้าแสดงผู้ใช้ทั้งหมด
    path('edit-user//', views.edit_user, name='edit_user'),  # หน้าแก้ไขข้อมูลผู้ใช้
    path('delete-user//', views.delete_user, name='delete_user'),  # ลบผู้ใช้
    path('manage-restaurants/', views.manage_restaurants, name='manage_restaurants'),
    path('superuser/edit-restaurant//', views.admin_edit_restaurant, name='edit_restaurant'),
    path('superuser/restaurant//delete/', views.admin_delete_restaurant, name='admin_delete_restaurant'),
    path('manage-forums/', views.manage_forums, name='manage_forums'),  # หน้าจัดการกระทู้
    path('superuser/edit-forum//', views.admin_edit_forum, name='edit_forum'),  # แก้ไขกระทู้
    path('superuser/delete-forum//', views.admin_delete_forum, name='delete_forum'),  # ลบกระทู้
    path('recommend-food/', views.recommend_food, name='recommend_food'),
    path('food//', views.food_detail, name='food_detail'),
    path('update-location/', views.update_location, name='update_location'),
    path('export-code//', app_to_pdf, name='export_code'),
    path('feedback-food//', views.feedback_food, name='feedback_food'),

]
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

## views.py

```
import numpy as np
from django.contrib.messages import success
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth import login
from django.utils.html import strip_tags
from django.views.decorators.csrf import csrf_exempt, csrf_protect
from django.views.decorators.http import require_http_methods
import re

from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler

from .forms import *
import random
from .models import *
from django.contrib.auth.decorators import login_required
from django.http import HttpResponseRedirect, JsonResponse, HttpResponseForbidden
from django.urls import reverse
from django.contrib import messages
from django.http import HttpResponse
from django.template.loader import render_to_string
from django.utils import timezone
import logging
from urllib.parse import unquote
from django.db.models import Avg
from django.db.models import Q
from django.contrib import messages
from django.contrib.auth.models import User
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required, user_passes_test
from django.urls import reverse
from django.core.paginator import Paginator
from sklearn.ensemble import RandomForestClassifier
from geopy.distance import distance
import json
from geopy.distance import geodesic
# หน้าแรก
def home(request):
    # จำกัดจำนวนรายการเป็น 6 รายการแรกในแต่ละประเภท
    restaurants = Restaurant.objects.all()[:6]
    foods = Food.objects.all()[:6]
    forums = Forum.objects.all()[:6]

    return render(request, 'core/home.html', {
        'restaurants': restaurants,
        'foods': foods,
        'forums': forums,
    })

# ลงทะเบียนผู้ใช้
def register(request):
    if request.method == 'POST':
        form = RegisterForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('home')
    else:
        form = RegisterForm()
    return render(request, 'core/register.html', {'form': form})


# แสดงโปรไฟล์
@login_required
def profile_view(request):
    profile, created = Profile.objects.get_or_create(user=request.user)
```

```
    # ดึงรายการอาหารที่ชอบและไม่ชอบ
    liked_foods = LikeDislikeFood.objects.filter(user=request.user, liked=True)
    disliked_foods = LikeDislikeFood.objects.filter(user=request.user, liked=False)

    # ดึงร้านอาหารที่ผู้ใช้บันทึกไว้
    saved_restaurants = Restaurant.objects.filter(saved_by=request.user)

    # ดึงกระทู้ที่ผู้ใช้บันทึกไว้
    saved_forums = Forum.objects.filter(saved_by=request.user)

    return render(request, 'core/profile_view.html', {
        'profile': profile,
        'liked_foods': liked_foods,
        'disliked_foods': disliked_foods,
        'saved_restaurants': saved_restaurants,  # ร้านอาหารที่บันทึก
        'saved_forums': saved_forums,  # กระทู้ที่บันทึก
    })


def calculate_distance(user_lat, user_lon, restaurant_lat, restaurant_lon):
    user_location = (user_lat, user_lon)
    restaurant_location = (restaurant_lat, restaurant_lon)
    # คำนวณระยะทางระหว่างผู้ใช้และร้านอาหาร (ระยะทางเป็นกิโลเมตร)
    return distance(user_location, restaurant_location).km

# แก้ไขโปรไฟล์
@login_required
def profile_edit(request):
    profile = request.user.profile
    if request.method == 'POST':
        form = ProfileForm(request.POST, request.FILES, instance=profile)
        if form.is_valid():
            form.save()
            return redirect('profile_view')
    else:
        form = ProfileForm(instance=profile)
    return render(request, 'core/profile_edit.html', {'form': form, 'profile': profile})


# ฟังก์ชันสำหรับสุ่มอาหาร

def random_food(request):
    form = FoodFilterForm(request.GET or None)
    food = None
    avg_rating = None

    user_lat = request.user.profile.latitude
    user_lon = request.user.profile.longitude

    foods = Food.objects.all().select_related('restaurant')

    # ✅ ตรวจสอบการกด Like/Dislike
    if request.method == 'POST':
        if not request.user.is_authenticated:
            messages.error(request, "กรุณาเข้าสู่ระบบก่อน")
            return redirect('login')

        food_id = request.POST.get('food_id')
        try:
            food = Food.objects.get(id=food_id)
            action = request.POST.get('action')
            liked_status = action == 'like'

            existing_entry, created = LikeDislikeFood.objects.get_or_create(
                user=request.user, food=food,
                defaults={'liked': liked_status, 'timestamp': timezone.now()}
            )

            if not created:
                existing_entry.liked = liked_status
                existing_entry.timestamp = timezone.now()
                existing_entry.save()

            messages.success(request, f"คุณ{'ชอบ' if liked_status else 'ไม่ชอบ'}อาหาร {food.name} แล้ว!")
            return redirect(request.path)
        except Food.DoesNotExist:
            messages.error(request, "ไม่พบอาหารที่คุณเลือก กรุณาลองใหม่")
            return redirect(request.path)

    # ✅ ตรวจสอบค่าที่รับมาจากฟอร์ม
    if request.GET and form.is_valid():
        category = form.cleaned_data.get('category')
        min_price = form.cleaned_data.get('min_price')
        max_price = form.cleaned_data.get('max_price')
        rating = form.cleaned_data.get('rating')
        distance = form.cleaned_data.get('distance')

        print("✅ Request GET:", request.GET)
        print("✅ All Foods Before Filtering:", list(foods.values('name', 'price', 'restaurant__name')))

        # ✅ กรองอาหารที่ถูก Dislike โดยผู้ใช้
        disliked_foods = LikeDislikeFood.objects.filter(user=request.user, liked=False).values_list('food_id', flat=True)
        foods = foods.exclude(id__in=disliked_foods)
        print("✅ Foods After Removing Disliked:", list(foods.values('name')))

        # ✅ กรองตามหมวดหมู่
        if category:
            foods = foods.filter(category__in=category)
            print("✅ Foods After Category Filter:", list(foods.values('name', 'category__name')))

        # ✅ แจ้งเตือนหากไม่มีอาหารในหมวดหมู่ที่เลือก
        if not foods.exists():
            messages.warning(request, "ไม่มีอาหารในหมวดหมู่นี้")
            return render(request, 'core/random_food.html', {'food': None, 'form': form})

        # ✅ กรองช่วงราคา
        if min_price:
            foods = foods.filter(price__gte=min_price)
        if max_price:
            foods = foods.filter(price__lte=max_price)
        print("✅ Foods After Price Filter:", list(foods.values('name', 'price')))

        # ✅ กรองตามคะแนนร้านอาหาร
        foods = foods.annotate(average_rating=Avg('restaurant__review__rating'))
        if rating:
            foods = foods.filter(average_rating__gte=rating)
        print("✅ Foods After Rating Filter:", list(foods.values('name', 'average_rating')))

        # ✅ กรองร้านที่มีพิกัดเท่านั้น
        foods_with_lat_lon = foods.filter(restaurant__latitude__isnull=False, restaurant__longitude__isnull=False)
        print("✅ Foods with lat/lon:", list(foods_with_lat_lon.values('name', 'restaurant__latitude', 'restaurant__longitude')))

        # ✅ ถ้าไม่มีพิกัด ให้แจ้งเตือนและลองใช้ Fallback
        if not foods_with_lat_lon.exists():
            messages.warning(request, "ไม่มีร้านที่มีพิกัดบนแผนที่")
```

```
            print("⚠️ ไม่มีร้านที่มีพิกัดเลย ใช้ค่าพิกัดผู้ใช้เป็น fallback")
            return render(request, 'core/random_food.html', {'food': None, 'form': form})

        # 🔹 กรองตามระยะทาง
        filtered_foods = []
        if distance:
            for food in foods_with_lat_lon:
                restaurant_location = (food.restaurant.latitude, food.restaurant.longitude)
                user_location = (user_lat, user_lon)
                distance_to_restaurant = geodesic(user_location, restaurant_location).kilometers
                print(f"📏 Distance to {food.name}: {distance_to_restaurant} km")
                if distance_to_restaurant <= float(distance):
                    filtered_foods.append(food)

            foods = Food.objects.filter(id__in=[food.id for food in filtered_foods])
        print("🍽 Foods After Distance Filter:", list(foods.values('name')))

        # 🔹 แจ้งเตือนหากไม่มีอาหารที่ตรงกับเงื่อนไข
        if not foods.exists():
            messages.warning(request, "ไม่มีอาหารที่ตรงกับเงื่อนไขการกรอง")
            return render(request, 'core/random_food.html', {'food': None, 'form': form})

        # 🔹 สุ่มอาหารจากที่กรองแล้ว
        if foods.exists():
            food = random.choice(list(foods))
            print("🎲 Random food selected:", food.name)

        # 🔹 คำนวณคะแนนเฉลี่ยของร้านที่สุ่มได้
        if food:
            avg_rating = Review.objects.filter(restaurant=food.restaurant).aggregate(Avg('rating'))['rating__avg']
            avg_rating = round(avg_rating, 1) if avg_rating else None

        # 🔹 คำนวณระยะทางของร้านที่สุ่มได้
        if food and food.restaurant.latitude and food.restaurant.longitude:
            restaurant_location = (food.restaurant.latitude, food.restaurant.longitude)
            user_location = (user_lat, user_lon)
            food.restaurant.distance = geodesic(user_location, restaurant_location).kilometers
        else:
            food.restaurant.distance = None

    return render(request, 'core/random_food.html', {
        'food': food,
        'form': form,
        'avg_rating': avg_rating if food else None,
        'distance': food.restaurant.distance if food and food.restaurant.distance else None,
    })

@login_required
def choose_food(request, food_id):
    food = get_object_or_404(Food, id=food_id)
    # สร้าง ChosenFood สำหรับผู้ใช้คนปัจจุบัน
    LikeDislikeFood.objects.create(user=request.user, food=food)
    return redirect('home')

# รายการร้านอาหาร
def restaurant_list(request):
    search_query = request.GET.get('search', '')
    rating_filter = request.GET.get('rating', '')
    distance_filter = request.GET.get('distance', '')
    category_filter = request.GET.get('category', '')

    # ดึงข้อมูลร้านอาหารทั้งหมด
    restaurants = Restaurant.objects.all()

    # การค้นหาตามชื่อและคำอธิบายร้าน
    if search_query:
        restaurants = restaurants.filter(
            Q(name__icontains=search_query) | Q(description__icontains=search_query)
        )

    # การกรองตามประเภทของร้านอาหาร
    if category_filter:
        restaurants = restaurants.filter(categories__id=category_filter)

    # ดึงพิกัดของผู้ใช้ (ต้องมีในโปรไฟล์ของผู้ใช้)
    user_lat = request.user.profile.latitude
    user_lon = request.user.profile.longitude

    # คำนวณระยะทางและเพิ่มฟิลด์ distance ให้แต่ละร้าน
    restaurant_list_with_ratings = []
    for restaurant in restaurants:
        # คำนวณคะแนนเฉลี่ยดาว
        average_rating = Review.objects.filter(restaurant=restaurant).aggregate(Avg('rating'))['rating__avg']
        restaurant.average_rating = average_rating if average_rating else 0  # กำหนดให้เป็น 0 ถ้าคะแนนเฉลี่ยไม่เป็นที่ต้องการ

        # ตรวจสอบว่าร้านมีพิกัด latitude, longitude หรือไม่
        if restaurant.latitude and restaurant.longitude:
            # คำนวณระยะทางจากพิกัดของร้านและผู้ใช้
            restaurant_location = (restaurant.latitude, restaurant.longitude)
            user_location = (user_lat, user_lon)
            distance = geodesic(user_location, restaurant_location).kilometers
            restaurant.distance = distance  # เพิ่มระยะทางเข้าไปในร้าน
        else:
            restaurant.distance = None  # ถ้าไม่มีพิกัดให้ระยะทางเป็น None

        restaurant_list_with_ratings.append(restaurant)

    # การกรองตามคะแนนเฉลี่ย
    if rating_filter:
        restaurant_list_with_ratings = [restaurant for restaurant in restaurant_list_with_ratings if restaurant.average_rating >= float(rating_filter)]

    # การกรองตามระยะทาง
    if distance_filter:
        restaurant_list_with_ratings = [restaurant for restaurant in restaurant_list_with_ratings if restaurant.distance and restaurant.distance <= float(distance_filter)]

    return render(request, 'core/food/restaurant_list.html', {
        'restaurants': restaurant_list_with_ratings,
        'categories': RestaurantCategory.objects.all(),  # ส่งรายการประเภทร้านอาหาร
    })

def restaurant_detail(request, id):
    restaurant = get_object_or_404(Restaurant, id=id)
    foods = Food.objects.filter(restaurant=restaurant)
    categories = restaurant.categories.all()
    reviews = Review.objects.filter(restaurant=restaurant)  # ดึงรีวิวร้านอาหาร

    # ตรวจสอบว่าผู้ใช้ได้รีวิวร้านอาหารนี้หรือยัง
    user_reviewed = False
    if request.user.is_authenticated:  # ตรวจสอบว่า user ล็อกอินอยู่หรือไม่
        user_reviewed = reviews.filter(user=request.user).exists()

    # ตรวจสอบว่าผู้ใช้ได้บันทึกร้านนี้หรือยัง
    is_saved = False
    if request.user.is_authenticated:
        is_saved = restaurant.saved_by.filter(id=request.user.id).exists()
```

```
    # คำนวณคะแนนเฉลี่ยดาวของร้านอาหาร
    average_rating = reviews.aggregate(average=Avg('rating'))['average']

    return render(request, 'core/food/restaurant_detail.html', {
        'restaurant': restaurant,
        'foods': foods,
        'categories': categories,
        'reviews': reviews,  # ส่งข้อมูลรีวิวไปยังเทมเพลต
        'user_reviewed': user_reviewed,  # ส่งตัวแปรตรวจสอบการรีวิวไปยังเทมเพลต
        'average_rating': average_rating,  # ส่งคะแนนเฉลี่ยไปยังเทมเพลต
        'is_saved': is_saved,  # ส่งสถานะการบันทึกไปยังเทมเพลต
    })


# สร้างร้านอาหารใหม่
@login_required
def restaurant_create(request):
    # ตรวจสอบว่าผู้ใช้มีร้านอาหารอยู่แล้วหรือไม่
    if Restaurant.objects.filter(owner=request.user).exists():
        restaurant = Restaurant.objects.get(owner=request.user)
        return redirect('restaurant_detail', restaurant.id)

    if request.method == 'POST':
        form = RestaurantForm(request.POST, request.FILES)
        if form.is_valid():
            restaurant = form.save(commit=False)
            restaurant.owner = request.user
            restaurant.save()
            return redirect('restaurant_detail', restaurant.id)
    else:
        form = RestaurantForm()

    return render(request, 'core/food/restaurant_create.html', {'form': form})


@login_required
def restaurant_edit(request, id):
    restaurant = get_object_or_404(Restaurant, id=id)

    # ตรวจสอบว่าเป็นเจ้าของร้านหรือเป็น Superuser
    if request.user != restaurant.owner and not request.user.is_superuser:
        messages.error(request, "คุณไม่มีสิทธิ์แก้ไขร้านอาหารนี้!")
        return redirect('restaurant_detail', id=restaurant.id)

    if request.method == 'POST':
        form = RestaurantForm(request.POST, request.FILES, instance=restaurant)

        if form.is_valid():
            restaurant = form.save(commit=False)  # ยังไม่บันทึกใน DB

            # อัปเดตหมวดหมู่ (categories)
            category_ids = request.POST.getlist('categories')  # รับค่าหมวดหมู่จาก form
            categories = RestaurantCategory.objects.filter(id__in=category_ids)  # ค้นหา Category objects
            restaurant.save()  # บันทึกข้อมูลหลักของร้าน
            restaurant.categories.set(categories)  # ตั้งค่าหมวดหมู่

            # อัปเดตรูปภาพใหม่หากมีการอัปโหลด
            if 'images' in request.FILES:
                restaurant.images = request.FILES['images']
                restaurant.save()

            messages.success(request, "แก้ไขข้อมูลร้านอาหารเรียบร้อยแล้ว!")
            return redirect('restaurant_detail', id=restaurant.id)
        else:
            messages.error(request, "กรุณาตรวจสอบข้อมูลให้ถูกต้อง.")

    else:
        form = RestaurantForm(instance=restaurant)

    return render(request, 'core/food/restaurant_edit.html', {
        'form': form,
        'restaurant': restaurant,
    })

@login_required
def add_restaurant_image(request, restaurant_id):
    restaurant = get_object_or_404(Restaurant, id=restaurant_id, owner=request.user)

    if request.method == 'POST':
        form = RestaurantImageForm(request.POST, request.FILES)
        if form.is_valid():
            RestaurantImage.objects.create(restaurant=restaurant, image=form.cleaned_data['image'])
            messages.success(request, "เพิ่มรูปบรรยากาศร้านเรียบร้อยแล้ว!")
            return redirect('restaurant_detail', id=restaurant.id)  # แก้ตรงนี้
    else:
        form = RestaurantImageForm()

    return render(request, 'core/food/add_image.html', {
        'form': form,
        'restaurant': restaurant
    })




@login_required
@require_http_methods(["DELETE", "POST"])
def delete_restaurant(request, pk):
    """
    View สำหรับลบร้านอาหาร รองรับทั้ง DELETE และ POST method
    """
    # ดึงข้อมูลร้านอาหาร
    restaurant = get_object_or_404(Restaurant, pk=pk)

    # ตรวจสอบว่า request.user เป็นเจ้าของร้าน
    if restaurant.owner != request.user:
        return JsonResponse({'error': 'คุณไม่มีสิทธิ์ในการลบร้านอาหารนี้'}, status=403)

    # ทำการลบร้านอาหาร
    restaurant.delete()

    # ตรวจสอบว่าเป็น HTMX request หรือไม่
    if request.headers.get('HX-Request'):
        return HttpResponse(status=204)  # HTMX ชอบ response 204 สำหรับการลบ
    else:
        # ถ้าไม่ใช่ HTMX request ให้ redirect ไปยัง restaurant_list
        return HttpResponseRedirect(reverse('restaurant_list'))


@login_required
def delete_image(request, restaurant_id, image_id):
    restaurant = get_object_or_404(Restaurant, id=restaurant_id, owner=request.user)
    image = get_object_or_404(RestaurantImage, id=image_id, restaurant=restaurant)
```

```
    # ลบรูปภาพ
    image.delete()
    messages.success(request, "ลบรูปบรรยากาศร้านเรียบร้อยแล้ว!")

    # เปลี่ยนพารามิเตอร์เป็น id
    return redirect('restaurant_detail', id=restaurant.id)


@login_required
def add_food(request, restaurant_id):
    # ดึงข้อมูลร้านอาหารที่เป็นเจ้าของโดย user
    restaurant = get_object_or_404(Restaurant, id=restaurant_id, owner=request.user)

    if request.method == 'POST':
        form = FoodForm(request.POST, request.FILES)
        if form.is_valid():
            food = form.save(commit=False)
            food.restaurant = restaurant  # เชื่อมอาหารกับร้านอาหาร
            food.save()  # บันทึกข้อมูลอาหาร

            # บันทึก ManyToManyField
            form.save_m2m()

            # เพิ่มข้อความแจ้งเตือนสำเร็จ
            messages.success(request, "เพิ่มเมนูอาหารเรียบร้อยแล้ว!")
            return redirect('restaurant_detail', id=restaurant.id)  # ต้องใส่ return
        else:
            messages.error(request, "เกิดข้อผิดพลาด กรุณาตรวจสอบข้อมูล.")
    else:
        # ดึงค่า queryset สำหรับ categories
        form = FoodForm()
        form.fields['category'].queryset = Category.objects.all()

    return render(request, 'core/food/add_food.html', {'form': form, 'restaurant': restaurant})

@login_required
def edit_food(request, restaurant_id, food_id):
    """
    View สำหรับแก้ไขเมนูอาหารที่เชื่อมกับร้านอาหาร
    """
    # ดึงข้อมูลร้านอาหารและเมนูที่ต้องการแก้ไข
    restaurant = get_object_or_404(Restaurant, id=restaurant_id, owner=request.user)
    food = get_object_or_404(Food, id=food_id, restaurant=restaurant)

    if request.method == 'POST':
        form = FoodForm(request.POST, request.FILES, instance=food)
        if form.is_valid():
            form.save()
            messages.success(request, "แก้ไขเมนูอาหารเรียบร้อยแล้ว!")
            return redirect('restaurant_detail', id=restaurant.id)  # กลับไปที่หน้ารายละเอียดร้านอาหาร
        else:
            messages.error(request, "มีข้อผิดพลาด กรุณาตรวจสอบข้อมูลที่กรอก")
    else:
        form = FoodForm(instance=food)

    return render(request, 'core/food/edit_food.html', {
        'form': form,
        'restaurant': restaurant,
        'food': food
    })

# ฟังก์ชันสำหรับลบเมนูอาหาร
@login_required()
def delete_food(request, restaurant_id, food_id):
    restaurant = get_object_or_404(Restaurant, id=restaurant_id)
    food = get_object_or_404(Food, id=food_id, restaurant=restaurant)

    # ตรวจสอบว่าผู้ใช้งานเป็นเจ้าของร้านอาหารหรือไม่
    if request.user == restaurant.owner:
        food.delete()

    return redirect('restaurant_detail', id=restaurant_id)
def food_list(request):
    search_query = request.GET.get('search', '')  # ค้นหาชื่ออาหาร
    category_filter = request.GET.getlist('category')  # รับค่าหมวดหมู่จาก select2 ที่เลือกหลายค่า
    rating_filter = request.GET.get('rating', '')  # กรองคะแนน
    distance_filter = request.GET.get('distance', '')  # กรองระยะทาง
    min_price = request.GET.get('min_price', '')  # รับค่าราคาต่ำสุด
    max_price = request.GET.get('max_price', '')  # รับค่าราคาสูงสุด

    # ดึงข้อมูลอาหารทั้งหมด
    foods = Food.objects.all()

    # ค้นหาตามชื่ออาหาร
    if search_query:
        foods = foods.filter(name__icontains=search_query)

    # กรองตามหมวดหมู่
    if category_filter:
        foods = foods.filter(category__id__in=category_filter)  # ใช้ __id__in เพื่อกรองหลายหมวดหมู่

    # กรองตามราคาต่ำสุดและสูงสุด
    if min_price:
        foods = foods.filter(price__gte=min_price)  # กรองอาหารที่ราคามากกว่าหรือเท่ากับราคาต่ำสุด
    if max_price:
        foods = foods.filter(price__lte=max_price)  # กรองอาหารที่ราคาน้อยกว่าหรือเท่ากับราคาสูงสุด

    # ดึงพิกัดของผู้ใช้ (ต้องมีในโปรไฟล์ของผู้ใช้)
    user_lat = request.user.profile.latitude
    user_lon = request.user.profile.longitude

    food_list_with_ratings = []
    for food in foods:
        # คำนวณคะแนนเฉลี่ยดาวของร้านอาหาร
        average_rating = Review.objects.filter(restaurant=food.restaurant).aggregate(Avg('rating'))['rating__avg']
        food.restaurant.average_rating = average_rating if average_rating else 0  # กำหนดให้เป็น 0 ถ้าคะแนนเฉลี่ยไม่เป็นที่ต้องการ

        # ตรวจสอบว่าร้านมีพิกัด latitude, longitude หรือไม่
        if food.restaurant.latitude and food.restaurant.longitude:
            # คำนวณระยะทางจากพิกัดของร้านและผู้ใช้
            restaurant_location = (food.restaurant.latitude, food.restaurant.longitude)
            user_location = (user_lat, user_lon)
            distance = geodesic(user_location, restaurant_location).kilometers
            food.restaurant.distance = distance  # เพิ่มระยะทางเข้าไปในร้าน
        else:
            food.restaurant.distance = None  # ถ้าไม่มีพิกัดให้ระยะทางเป็น None

        food_list_with_ratings.append(food)

    # การกรองตามคะแนนเฉลี่ย
    if rating_filter:
        food_list_with_ratings = [food for food in food_list_with_ratings if food.restaurant.average_rating >= float(rating_filter)]

    # การกรองตามระยะทาง
    if distance_filter:
```

```
        food_list_with_ratings = [food for food in food_list_with_ratings if food.restaurant.distance and food.restaurant.distance <= float(distance_filter)]

    # ส่งข้อมูลไปยังเทมเพลต
    return render(request, 'core/food/food_list.html', {
        'foods': food_list_with_ratings,
        'categories': Category.objects.all(),  # ส่งหมวดหมู่ทั้งหมดไปยังเทมเพลต
    })


# ฟอรัม (กระทู้)
def forum_list(request):
    forums = Forum.objects.all().order_by('-created_at')  # ดึงข้อมูลเรียงตามวันที่สร้าง
    return render(request, 'core/community/forum.html', {'forums': forums})


@login_required
def create_forum(request):
    if request.method == "POST":
        form = ForumForm(request.POST, request.FILES)
        if form.is_valid():
            forum = form.save(commit=False)
            forum.user = request.user  # ผูกกระทู้กับผู้ใช้งาน
            forum.save()
            return redirect('forum')  # กลับไปยังหน้ารายการกระทู้
    else:
        form = ForumForm()

    return render(request, 'core/community/create_forum.html', {'form': form})


@login_required
def forum_detail(request, forum_id):
    """แสดงกระทู้และความคิดเห็น"""
    forum = get_object_or_404(Forum, id=forum_id)
    comments = forum.comments.order_by('-created_at')

    if request.method == 'POST':
        content = request.POST.get('content', '').strip()
        if content:
            ForumComment.objects.create(
                forum=forum,
                user=request.user,
                content=content
            )
            messages.success(request, "ความคิดเห็นของคุณถูกบันทึกแล้ว!")
        else:
            messages.error(request, "กรุณากรอกข้อความก่อนแสดงความคิดเห็น")
        return redirect('forum_detail', forum_id=forum_id)

    return render(request, 'core/community/forum_detail.html', {'forum': forum, 'comments': comments})



def my_forums(request):
    forums = Forum.objects.filter(user=request.user)  # แสดงกระทู้ของผู้ใช้ที่ล็อกอิน
    return render(request, 'core/community/forum.html', {'forums': forums})


@login_required
def forum_edit(request, forum_id):
    forum = get_object_or_404(Forum, id=forum_id)

    # ตรวจสอบสิทธิ์: เจ้าของกระทู้หรือ Superuser เท่านั้นที่สามารถแก้ไขได้
    if request.user != forum.user and not request.user.is_superuser:
        return redirect('forum_detail', forum_id=forum.id)  # ถ้าไม่ใช่เจ้าของหรือ superuser ให้ redirect กลับ

    if request.method == 'POST':
        form = ForumForm(request.POST, request.FILES, instance=forum)
        if form.is_valid():
            form.save()
            return redirect('forum_detail', forum_id=forum.id)
    else:
        form = ForumForm(instance=forum)

    return render(request, 'core/community/forum_edit.html', {'form': form, 'forum': forum})


@login_required
def forum_delete(request, pk):
    forum = get_object_or_404(Forum, pk=pk)

    # ตรวจสอบสิทธิ์: เจ้าของกระทู้หรือ Superuser เท่านั้นที่สามารถลบได้
    if request.user != forum.user and not request.user.is_superuser:
        return redirect('forum_detail', forum_id=forum.id)  # ถ้าไม่ใช่เจ้าของหรือ superuser ให้ redirect กลับ

    if request.method == 'POST':
        forum.delete()
        return redirect('my_forums')  # กลับไปที่หน้ารายการกระทู้

    return render(request, 'core/community/forum_delete.html', {'forum': forum})


logger = logging.getLogger(__name__)

@login_required
@csrf_protect
def add_comment(request, forum_id):
    """เพิ่มความคิดเห็นใหม่"""
    if request.method == 'POST':
        forum = get_object_or_404(Forum, id=forum_id)
        raw_content = request.POST.get('content', '').strip()

        # ล้างข้อความ
        cleaned_content = strip_tags(raw_content)
        if not cleaned_content:
            return JsonResponse({'error': 'กรุณากรอกข้อความที่ต้องการแสดงความคิดเห็น'}, status=400)

        # สร้างความคิดเห็น
        comment = ForumComment.objects.create(
            forum=forum,
            user=request.user,
            content=cleaned_content
        )

        # โหลดความคิดเห็นใหม่
        comments = forum.comments.select_related('user__profile').order_by('-created_at')

        # Render partial template
        html = render_to_string(
            'core/partials/forum_comments.html',
            {'comments': comments, 'request': request},
            request=request
        )

        # Decode URL ที่อาจถูก encode
```

```
        sanitized_html = unquote(html)
        return JsonResponse({'html': sanitized_html.strip()})

    return JsonResponse({'error': 'Invalid request method'}, status=400)

'''@login_required
def load_comments(request, forum_id):
    """โหลดความคิดเห็นทั้งหมด"""
    forum = get_object_or_404(Forum, id=forum_id)
    comments = forum.comments.select_related('user__profile').order_by('-created_at')

    # Render partial template
    html = render_to_string(
        'core/partials/forum_comments.html',
        {'comments': comments},
        request=request
    )

    # Debug HTML
    logger.debug(f"Rendered Comments HTML: {html}")

    return HttpResponse(html, content_type='text/html; charset=utf-8')  # กำหนด encoding เพื่อป้องกันปัญหาภาษาแปลก ๆ'''

@login_required
def delete_comment(request, comment_id):
    comment = get_object_or_404(ForumComment, id=comment_id)

    #  อนุญาตให้ลบได้หากเป็นเจ้าของความคิดเห็นหรือเป็น superuser
    if comment.user != request.user and not request.user.is_superuser:
        return HttpResponseForbidden("คุณไม่มีสิทธิ์ลบความคิดเห็นนี้")

    forum_id = comment.forum.id
    comment.delete()
    return redirect('forum_detail', forum_id=forum_id)

@login_required
def edit_comment(request, comment_id):
    """View สำหรับการแก้ไขความคิดเห็น"""
    comment = get_object_or_404(ForumComment, id=comment_id)

    #  อนุญาตให้แก้ไขได้หากเป็นเจ้าของความคิดเห็นหรือเป็น superuser
    if comment.user != request.user and not request.user.is_superuser:
        return HttpResponseForbidden("คุณไม่มีสิทธิ์แก้ไขความคิดเห็นนี้")

    if request.method == "POST":
        new_content = request.POST.get('content', '').strip()
        if new_content:
            comment.content = new_content
            comment.save()
            return redirect('forum_detail', comment.forum.id)  # กลับไปที่หน้า Forum เดิม
        else:
            return HttpResponseForbidden("ไม่สามารถบันทึกความคิดเห็นว่างได้")

    # Render แบบ GET เพื่อแสดงแบบฟอร์ม
    return render(request, 'core/community/edit_comment.html', {'comment': comment})

@login_required
def add_review(request, restaurant_id):
    restaurant = get_object_or_404(Restaurant, id=restaurant_id)

    # ตรวจสอบว่าผู้ใช้ได้รีวิวร้านอาหารนี้ไปแล้วหรือไม่
    if Review.objects.filter(restaurant=restaurant, user=request.user).exists():
        messages.error(request, "คุณได้รีวิวร้านนี้ไปแล้ว")
        return redirect('restaurant_detail', id=restaurant.id)  # กลับไปยังหน้ารายละเอียดร้าน

    if request.method == 'POST':
        form = ReviewForm(request.POST)
        if form.is_valid():
            review = form.save(commit=False)
            review.user = request.user  # เชื่อมโยงรีวิวกับผู้ใช้ที่ล็อกอิน
            review.restaurant = restaurant  # เชื่อมโยงรีวิวกับร้านอาหาร
            review.save()
            messages.success(request, "รีวิวของคุณถูกบันทึกแล้ว")
            return redirect('restaurant_detail', id=restaurant.id)  # กลับไปยังหน้ารายละเอียดร้าน
    else:
        form = ReviewForm()

    return render(request, 'core/food/add_review.html', {
        'form': form,
        'restaurant': restaurant,
    })

@login_required
def delete_review(request, review_id):
    review = get_object_or_404(Review, id=review_id)
    if request.user == review.user or request.user.is_superuser:
        review.delete()
    return redirect('restaurant_detail', id=review.restaurant.id)

@login_required
def edit_review(request, review_id):
    review = get_object_or_404(Review, id=review_id)

    # อนุญาตให้แก้ไขได้เฉพาะเจ้าของรีวิวหรือ Super Admin
    if request.user != review.user and not request.user.is_superuser:
        messages.error(request, "คุณไม่มีสิทธิ์แก้ไขรีวิวนี้")
        return redirect('restaurant_detail', id=review.restaurant.id)

    if request.method == 'POST':
        form = ReviewForm(request.POST, instance=review)
        if form.is_valid():
            form.save()
            messages.success(request, "รีวิวของคุณถูกแก้ไขเรียบร้อยแล้ว!")
            return redirect('restaurant_detail', id=review.restaurant.id)
    else:
        form = ReviewForm(instance=review)

    return render(request, 'core/food/edit_review.html', {'form': form, 'review': review})


@login_required
def toggle_save_restaurant(request, restaurant_id):
    restaurant = get_object_or_404(Restaurant, id=restaurant_id)

    if request.user in restaurant.saved_by.all():
        # ยกเลิกการบันทึก
        restaurant.saved_by.remove(request.user)
        messages.success(request, 'ยกเลิกการบันทึกร้านอาหารเรียบร้อยแล้ว!')
    else:
        # บันทึกร้านอาหาร
        restaurant.saved_by.add(request.user)
        messages.success(request, 'บันทึกร้านอาหารเรียบร้อยแล้ว!')

    return redirect('restaurant_detail', id=restaurant_id)
```

```
@login_required
def toggle_save_forum(request, forum_id):
    forum = get_object_or_404(Forum, id=forum_id)

    if request.user in forum.saved_by.all():
        # ยกเลิกการบันทึก
        forum.saved_by.remove(request.user)
        messages.success(request, 'ยกเลิกการบันทึกกระทู้เรียบร้อยแล้ว!')
    else:
        # บันทึกกระทู้
        forum.saved_by.add(request.user)
        messages.success(request, 'บันทึกกระทู้เรียบร้อยแล้ว!')

    return redirect('forum_detail', forum_id=forum_id)

# ฟังก์ชันการแสดงแดชบอร์ด Admin
@login_required
def admin_dashboard(request):
    if request.user.is_superuser:  # ตรวจสอบว่า user เป็น superuser หรือไม่
        total_users = User.objects.count()
        total_restaurants = Restaurant.objects.count()
        total_forums = Forum.objects.count()

        # □ ดึงข้อมูล Feedback เฉพาะจากหน้าแนะนำ (recommend)
        total_likes = LikeDislikeFood.objects.filter(liked=True, source="recommend").count()
        total_dislikes = LikeDislikeFood.objects.filter(liked=False, source="recommend").count()

        return render(request, 'core/adminpage/admin_dashboard.html', {
            'total_users': total_users,
            'total_restaurants': total_restaurants,
            'total_forums': total_forums,
            'total_likes': total_likes,
            'total_dislikes': total_dislikes
        })
    else:
        return render(request, 'core/adminpage/access_denied.html')  # หากไม่ใช่ superuser แสดงหน้าไม่อนุญาต



@login_required
def manage_users(request):
    if request.user.is_superuser:
        # ดึงข้อมูลผู้ใช้ทั้งหมด
        users = User.objects.all()

        # ตรวจสอบว่ามีการค้นหาหรือไม่
        search = request.GET.get('search', '')  # ดึงค่าจากฟอร์มค้นหา

        if search:
            # ถ้ามีการค้นหาให้กรองตามชื่อผู้ใช้หรืออีเมล
            users = users.filter(username__icontains=search) | users.filter(email__icontains=search)

        # ตรวจสอบฟิลเตอร์สถานะผู้ใช้
        status = request.GET.get('status', '')
        if status == 'active':
            users = users.filter(is_active=True)
        elif status == 'inactive':
            users = users.filter(is_active=False)

        return render(request, 'core/adminpage/manage_users.html', {'users': users})

    else:
        messages.error(request, "คุณไม่มีสิทธิ์เข้าถึงหน้านี้")
        return redirect('home')  # เปลี่ยนเป็นหน้าหลักของแอปพลิเคชัน


@login_required
def edit_user(request, user_id):
    if request.user.is_superuser:
        user = get_object_or_404(User, id=user_id)
        if request.method == "POST":
            # อัปเดตข้อมูลผู้ใช้ที่เลือก เช่น อัปเดตบทบาท
            user.is_active = request.POST.get('is_active') == 'on'
            user.is_staff = request.POST.get('is_staff') == 'on'
            user.save()
            messages.success(request, f"ข้อมูลของ {user.username} ได้รับการอัปเดตแล้ว")
            return redirect('manage_users')  # เปลี่ยนไปหน้า manage users
        return render(request, 'core/adminpage/edit_user.html', {'user': user})
    else:
        messages.error(request, "คุณไม่มีสิทธิ์เข้าถึงหน้านี้")
        return redirect('home')  # เปลี่ยนเป็นหน้าหลักของแอปพลิเคชัน

@login_required
def delete_user(request, user_id):
    if request.user.is_superuser:
        user = get_object_or_404(User, id=user_id)
        user.delete()
        messages.success(request, f"ผู้ใช้ {user.username} ถูกลบออกจากระบบแล้ว")
        return redirect('manage_users')
    else:
        messages.error(request, "คุณไม่มีสิทธิ์ลบผู้ใช้นี้")
        return redirect('home')


@login_required
def manage_restaurants(request):
    if request.user.is_superuser:
        # ดึงข้อมูลคำค้นหาจากฟอร์ม
        query = request.GET.get('search', '')

        # ค้นหาจากชื่อร้านและรายละเอียด
        if query:
            restaurants = Restaurant.objects.filter(
                Q(name__icontains=query) | Q(description__icontains=query)
            )
        else:
            restaurants = Restaurant.objects.all()

        return render(request, 'core/adminpage/manage_restaurants.html', {'restaurants': restaurants})
    else:
        messages.error(request, "คุณไม่มีสิทธิ์เข้าถึงหน้านี้")
        return redirect('home')  # เปลี่ยนเป็นหน้าหลักของแอปพลิเคชัน

@login_required
def admin_edit_restaurant(request, restaurant_id):
    if request.user.is_superuser:
        restaurant = get_object_or_404(Restaurant, id=restaurant_id)

        if request.method == "POST":
            form = RestaurantForm(request.POST, instance=restaurant)
            if form.is_valid():
                form.save()
                messages.success(request, "แก้ไขร้านอาหารเรียบร้อยแล้ว!")
                return redirect('manage_restaurants')
        else:
```

```
            form = RestaurantForm(instance=restaurant)
        return render(request, 'core/adminpage/edit_restaurant.html', {'form': form, 'restaurant': restaurant})
    else:
        messages.error(request, "คุณไม่มีสิทธิ์เข้าถึงหน้านี้")
        return redirect('home')


def is_superuser(user):
    return user.is_superuser

@login_required
@user_passes_test(is_superuser)
def admin_delete_restaurant(request, restaurant_id):
    if request.method == 'POST':
        restaurant = get_object_or_404(Restaurant, id=restaurant_id)
        restaurant.delete()
        messages.success(request, "ลบร้านอาหารเรียบร้อยแล้ว!")
        return redirect('manage_restaurants')
    return redirect('manage_restaurants')


@login_required
def manage_forums(request):
   if request.user.is_superuser:
        # ดึงกระทู้ทั้งหมด
        forums = Forum.objects.all().order_by('-created_at')  # เรียงจากใหม่ไปเก่า

        # ค้นหากระทู้
        search_query = request.GET.get('search', '')
        if search_query:
            forums = forums.filter(
                Q(title__icontains=search_query) |  # ค้นหาจากชื่อ
                Q(user__username__icontains=search_query)  # ค้นหาจากชื่อผู้ใช้
            )

        # Pagination (ถ้าต้องการ)
        paginator = Paginator(forums, 10)  # แสดง 10 รายการต่อหน้า
        page_number = request.GET.get('page')
        page_obj = paginator.get_page(page_number)

        context = {
            'forums': page_obj,
            'search_query': search_query,
        }

        return render(request, 'core/adminpage/manage_forums.html', context)
   else:
        messages.error(request, "คุณไม่มีสิทธิ์ในการเข้าถึงหน้านี้")
        return redirect('home')

@login_required
def admin_edit_forum(request, forum_id):
    if request.user.is_superuser:
        forum = get_object_or_404(Forum, id=forum_id)

        if request.method == 'POST':
            form = ForumForm(request.POST, instance=forum)
            if form.is_valid():
                form.save()
                messages.success(request, 'แก้ไขกระทู้เรียบร้อย!')
                return redirect('manage_forums')
        else:
            form = ForumForm(instance=forum)

        return render(request, 'core/adminpage/edit_forum.html', {'form': form, 'forum': forum})
    else:
        messages.error(request, "คุณไม่มีสิทธิ์ในการเข้าถึงหน้านี้")
        return redirect('home')

@login_required
@require_http_methods(["POST"])
def admin_delete_forum(request, forum_id):
    if request.user.is_superuser:
        forum = get_object_or_404(Forum, id=forum_id)
        forum.delete()
        messages.success(request, 'ลบกระทู้เรียบร้อยแล้ว!')
        return redirect('manage_forums')
    else:
        messages.error(request, "คุณไม่มีสิทธิ์ในการลบกระทู้นี้")
        return redirect('home')


def food_detail(request, id):
    food = get_object_or_404(Food, id=id)
    return render(request, 'core/food_detail.html', {'food': food})

@login_required
def recommend_food(request):
    user = request.user
    liked_foods = LikeDislikeFood.objects.filter(user=user)

    if not liked_foods.exists():
        messages.warning(request, "คุณยังไม่มีข้อมูลการชอบ/ไม่ชอบอาหาร กรุณาให้คะแนนอาหารก่อน!")
        return render(request, 'core/recommend_food.html', {'recommendations': []})

    foods = Food.objects.all()
    food_data, food_labels, food_ids = [], [], []
    category_list = set()

    for log in liked_foods:
        food = log.food
        category_names = [c.name for c in food.category.all()]
        category_list.update(category_names)

        # 📍 คำนวณระยะทางระหว่างผู้ใช้กับร้านอาหาร
        if food.restaurant.latitude and food.restaurant.longitude:
            user_location = (user.profile.latitude, user.profile.longitude)
            restaurant_location = (food.restaurant.latitude, food.restaurant.longitude)
            distance = geodesic(user_location, restaurant_location).kilometers
        else:
            distance = 100  # 📍 ถ้าไม่มีพิกัด ให้กำหนดค่า Default 100 กม.

        # 📍 ใช้ aggregate() เพื่อดึงคะแนนเฉลี่ยของร้าน
        avg_rating = Review.objects.filter(restaurant=food.restaurant).aggregate(Avg('rating'))['rating__avg']
        avg_rating = avg_rating if avg_rating else 0  # 📍 ถ้ายังไม่มีรีวิว ให้กำหนดค่าเป็น 0

        food_data.append([food.price, distance, avg_rating, *category_names])
        food_labels.append(1 if log.liked else 0)
        food_ids.append(food.id)

    if len(food_data) < 5:
        messages.warning(request, "ข้อมูลที่เคยให้คะแนนมีน้อยเกินไป แนะนำให้ให้คะแนนอาหารเพิ่มก่อน!")
        return render(request, 'core/recommend_food.html', {'recommendations': []})

    # 📍 One-Hot Encoding หมวดหมู่อาหาร
```

```
    encoder = OneHotEncoder(handle_unknown='ignore')
    category_array = np.array(list(category_list)).reshape(-1, 1)
    encoder.fit(category_array)

    transformed_data = []
    for food in food_data:
        price, distance, avg_rating = food[:3]
        category_vector = encoder.transform(np.array(food[3:]).reshape(-1, 1)).toarray().sum(axis=0)
        transformed_data.append([price, distance, avg_rating] + list(category_vector))

    # □ Train-Test Split
    scaler = MinMaxScaler()
    scaled_data = scaler.fit_transform(transformed_data)
    X_train, X_test, y_train, y_test = train_test_split(scaled_data, food_labels, test_size=0.2, random_state=42)

    # □ เทรนโมเดล RandomForest
    clf = RandomForestClassifier(n_estimators=100, random_state=42)
    clf.fit(X_train, y_train)

    # □ ทำนายค่าใน Test Set
    y_pred = clf.predict(X_test)

    # □ คำนวณ Accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f"□ Accuracy ของโมเดล: {accuracy:.4f}")

    # □ คัดกรองอาหารที่ยังไม่เคยถูกให้คะแนน
    unseen_foods = foods.exclude(id__in=food_ids)
    if not unseen_foods.exists():
        messages.warning(request, "ไม่มีอาหารที่ยังไม่ได้ให้คะแนน")
        return render(request, 'core/recommend_food.html', {'recommendations': []})

    unseen_data = []
    unseen_food_list = []

    for food in unseen_foods:
        category_names = [c.name for c in food.category.all()]

        # □ คำนวณระยะทาง
        if food.restaurant.latitude and food.restaurant.longitude:
            user_location = (user.profile.latitude, user.profile.longitude)
            restaurant_location = (food.restaurant.latitude, food.restaurant.longitude)
            distance = geodesic(user_location, restaurant_location).kilometers
        else:
            distance = 100  # □ ใช้ค่า Default ถ้าไม่มีพิกัด

        # □ ใช้ aggregate() เพื่อดึงคะแนนเฉลี่ยของร้าน
        avg_rating = Review.objects.filter(restaurant=food.restaurant).aggregate(Avg('rating'))['rating__avg']
        avg_rating = avg_rating if avg_rating else 0  # □ กำหนดค่า Default

        category_vector = encoder.transform(np.array(category_names).reshape(-1, 1)).toarray().sum(axis=0)
        unseen_data.append([food.price, distance, avg_rating] + list(category_vector))
        unseen_food_list.append(food)

    scaled_unseen_data = scaler.transform(unseen_data)

    # □ ใช้โมเดลทำนาย
    predictions = clf.predict_proba(scaled_unseen_data)[:, 1]
    sorted_indices = np.argsort(predictions)[::-1]
    recommended_foods = [unseen_food_list[i] for i in sorted_indices[:5]]

    return render(request, 'core/recommend_food.html', {
        'recommendations': recommended_foods
    })

@login_required
def feedback_food(request, food_id):
    user = request.user
    try:
        food = Food.objects.get(id=food_id)

        if request.method == 'POST':
            action = request.POST.get('action')
            liked_status = action == 'like'

            # □ บันทึก Feedback เฉพาะจากหน้า Recommend
            existing_entry = LikeDislikeFood.objects.filter(user=user, food=food, source='recommend').first()
            if existing_entry:
                existing_entry.liked = liked_status
                existing_entry.save()
            else:
                LikeDislikeFood.objects.create(user=user, food=food, liked=liked_status, source='recommend')

            messages.success(request, f"คุณ{'ชอบ' if liked_status else 'ไม่ชอบ'}อาหาร {food.name} แล้ว!")
            return redirect('recommend_food')  # □ กลับไปที่หน้าแนะนำอาหาร

    except Food.DoesNotExist:
        messages.error(request, "ไม่พบอาหารที่เลือก กรุณาลองใหม่")
        return redirect('recommend_food')


@login_required
def update_location(request):
    if request.method == 'POST':
        data = json.loads(request.body)
        latitude = data.get('latitude')
        longitude = data.get('longitude')

        # อัปเดตข้อมูลตำแหน่งใน Profile
        profile = Profile.objects.get(user=request.user)
        profile.latitude = latitude
        profile.longitude = longitude
        profile.save()

        return JsonResponse({'status': 'success'}, status=200)

    return JsonResponse({'status': 'failed'}, status=400)



import os
import pdfkit
from django.http import HttpResponse
from django.conf import settings

# กำหนด path ของ wkhtmltopdf (Windows ใช้ path นี้, Linux/Mac อาจไม่ต้อง)
PDFKIT_CONFIG = pdfkit.configuration(wkhtmltopdf=r"C:\Program Files\wkhtmltopdf\bin\wkhtmltopdf.exe")

def app_to_pdf(request, app_name):
    app_path = os.path.join(settings.BASE_DIR, app_name)
    pdf_filename = f"{app_name}.pdf"

    if not os.path.exists(app_path):
        return HttpResponse("App not found", status=404)
```

```
    html_content = f"
```

## Source Code of App: {app_name}

```
"

    for root, dirs, files in os.walk(app_path):
        for file in files:
            if file.endswith((".py", ".html", ".css", ".js")):  # แปลงเฉพาะไฟล์ที่ต้องการ
                file_path = os.path.join(root, file)
                with open(file_path, "r", encoding="utf-8") as f:  #  อ่านไฟล์เป็น UTF-8
                    html_content += f"
```

### {file}

```
{f.read()}
"

    #  ใช้ pdfkit พร้อมกำหนดให้ใช้ encoding UTF-8
    options = {'encoding': 'UTF-8'}
    pdfkit.from_string(html_content, pdf_filename, configuration=PDFKIT_CONFIG, options=options)

    with open(pdf_filename, "rb") as pdf:
        response = HttpResponse(pdf.read(), content_type="application/pdf")
        response["Content-Disposition"] = f'attachment; filename="{pdf_filename}"'
        return response
```

## __init__.py

## generate_dbml.py

```
from django.core.management.base import BaseCommand
from django.apps import apps
from django.db import models


class Command(BaseCommand):
   help = 'Generate DBML file from Django models'


   def handle(self, *args, **kwargs):
       # สร้างตัวแปรสำหรับไฟล์ DBML
       dbml_content = ""


       # ดึงโมเดลทั้งหมดจากทุกแอปในโปรเจกต์
       models_list = []
       for app_config in apps.get_app_configs():
           models_list.extend(app_config.get_models())


       # สร้าง DBML content สำหรับแต่ละตาราง
       for model in models_list:
           table_name = model._meta.model_name
           dbml_content += f"Table {table_name} {{\n"


           # สร้างคอลัมน์ตามฟิลด์ในโมเดล
           for field in model._meta.fields:
               field_name = field.name
               field_type = field.get_internal_type().lower()
               if isinstance(field, models.CharField):
                   dbml_content += f"  {field_name} varchar\n"
               elif isinstance(field, models.TextField):
                   dbml_content += f"  {field_name} text\n"
               elif isinstance(field, models.IntegerField):
                   dbml_content += f"  {field_name} integer\n"
               elif isinstance(field, models.DateTimeField):
                   dbml_content += f"  {field_name} timestamp\n"
               elif isinstance(field, models.ForeignKey):
                   related_model = field.related_model
                   dbml_content += f"  {field_name}_id integer [note: 'foreign key to {related_model._meta.model_name}']\n"


           dbml_content += "}\n\n"


       # สร้างความสัมพันธ์ระหว่างตาราง
       for model in models_list:
           for field in model._meta.fields:
               if isinstance(field, models.ForeignKey):
                   from_table = model._meta.model_name
                   to_table = field.related_model._meta.model_name
                   dbml_content += f"Ref: {from_table}.{field.name}_id > {to_table}.id // many-to-one\n"


       # เขียนข้อมูล DBML ลงไฟล์
       with open('generated_schema.dbml', 'w') as file:
           file.write(dbml_content)


       self.stdout.write(self.style.SUCCESS('Successfully generated DBML file'))
```

## 0001_initial.py

```
# Generated by Django 5.1.1 on 2024-09-26 12:14

import django.db.models.deletion
from django.conf import settings
from django.db import migrations, models


class Migration(migrations.Migration):
    initial = True

    dependencies = [
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
    ]

    operations = [
        migrations.CreateModel(
            name="Restaurant",
            fields=[
                (
                    "id",
                    models.BigAutoField(
                        auto_created=True,
                        primary_key=True,
                        serialize=False,
```

```
                    verbose_name="ID",
                ),
            ),
            ("name", models.CharField(max_length=255)),
            ("description", models.TextField()),
            ("location", models.CharField(max_length=255)),
            ("contact_info", models.CharField(max_length=255)),
            ("opening_hours", models.TextField()),
            ("social_media", models.TextField(blank=True, null=True)),
            ("images", models.TextField(blank=True, null=True)),
        ],
    ),
    migrations.CreateModel(
        name="Profile",
        fields=[
            (
                "id",
                models.BigAutoField(
                    auto_created=True,
                    primary_key=True,
                    serialize=False,
                    verbose_name="ID",
                ),
            ),
            ("first_name", models.CharField(max_length=255)),
            ("last_name", models.CharField(max_length=255)),
            (
                "profile_picture",
                models.CharField(blank=True, max_length=255, null=True),
            ),
            (
                "user",
                models.OneToOneField(
                    on_delete=django.db.models.deletion.CASCADE,
                    to=settings.AUTH_USER_MODEL,
                ),
            ),
        ],
    ),
    migrations.CreateModel(
        name="Review",
        fields=[
            (
                "id",
                models.BigAutoField(
                    auto_created=True,
                    primary_key=True,
                    serialize=False,
                    verbose_name="ID",
                ),
            ),
            ("rating", models.IntegerField()),
            ("comment", models.TextField()),
            ("created_at", models.DateTimeField(auto_now_add=True)),
            (
                "restaurant",
                models.ForeignKey(
                    on_delete=django.db.models.deletion.CASCADE,
                    to="core.restaurant",
                ),
            ),
            (
                "user",
                models.ForeignKey(
                    on_delete=django.db.models.deletion.CASCADE,
                    to=settings.AUTH_USER_MODEL,
                ),
            ),
        ],
    ),
    migrations.CreateModel(
        name="Thread",
        fields=[
            (
                "id",
                models.BigAutoField(
                    auto_created=True,
                    primary_key=True,
                    serialize=False,
                    verbose_name="ID",
                ),
            ),
            ("title", models.CharField(max_length=255)),
            ("content", models.TextField()),
            ("created_at", models.DateTimeField(auto_now_add=True)),
            (
                "user",
                models.ForeignKey(
                    on_delete=django.db.models.deletion.CASCADE,
                    to=settings.AUTH_USER_MODEL,
                ),
            ),
        ],
    ),
    migrations.CreateModel(
        name="Comment",
        fields=[
            (
                "id",
                models.BigAutoField(
                    auto_created=True,
                    primary_key=True,
                    serialize=False,
                    verbose_name="ID",
                ),
            ),
            ("content", models.TextField()),
            ("created_at", models.DateTimeField(auto_now_add=True)),
            (
                "user",
                models.ForeignKey(
                    on_delete=django.db.models.deletion.CASCADE,
                    to=settings.AUTH_USER_MODEL,
                ),
            ),
            (
                "thread",
                models.ForeignKey(
                    on_delete=django.db.models.deletion.CASCADE, to="core.thread"
                ),
            ),
        ],
    ),
]
```

## 0002_alter_profile_profile_picture.py

```
# Generated by Django 5.1.1 on 2024-10-05 13:49

from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0001_initial"),
    ]

    operations = [
        migrations.AlterField(
            model_name="profile",
            name="profile_picture",
            field=models.ImageField(
                blank=True, null=True, upload_to="profile_pictures/"
            ),
        ),
    ]
```

## 0003_category_subcategory_food.py

```
# Generated by Django 5.1.1 on 2024-10-14 08:40

import django.db.models.deletion
from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0002_alter_profile_profile_picture"),
    ]

    operations = [
        migrations.CreateModel(
            name="Category",
            fields=[
                (
                    "id",
                    models.BigAutoField(
                        auto_created=True,
                        primary_key=True,
                        serialize=False,
                        verbose_name="ID",
                    ),
                ),
                ("name", models.CharField(max_length=255)),
            ],
        ),
        migrations.CreateModel(
            name="SubCategory",
            fields=[
                (
                    "id",
                    models.BigAutoField(
                        auto_created=True,
                        primary_key=True,
                        serialize=False,
                        verbose_name="ID",
                    ),
                ),
                ("name", models.CharField(max_length=255)),
                (
                    "category",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE,
                        related_name="subcategories",
                        to="core.category",
                    ),
                ),
            ],
        ),
        migrations.CreateModel(
            name="Food",
            fields=[
                (
                    "id",
                    models.BigAutoField(
                        auto_created=True,
                        primary_key=True,
                        serialize=False,
                        verbose_name="ID",
                    ),
                ),
                ("name", models.CharField(max_length=255)),
                ("description", models.TextField()),
                ("price", models.DecimalField(decimal_places=2, max_digits=6)),
                (
                    "image",
                    models.ImageField(blank=True, null=True, upload_to="food_images/"),
                ),
                (
                    "category",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE,
                        related_name="foods",
                        to="core.category",
                    ),
                ),
                (
                    "restaurant",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE,
                        related_name="foods",
                        to="core.restaurant",
                    ),
                ),
                (
                    "subcategory",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE,
                        related_name="foods",
                        to="core.subcategory",
                    ),
                ),
            ],
        ),
    ]
```

## 0004_restaurant_owner.py

```
# Generated by Django 5.1.1 on 2024-10-14 09:04

import django.db.models.deletion
from django.conf import settings
from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0003_category_subcategory_food"),
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
    ]

    operations = [
        migrations.AddField(
            model_name="restaurant",
            name="owner",
            field=models.OneToOneField(
                default=1,
                on_delete=django.db.models.deletion.CASCADE,
                related_name="restaurant",
                to=settings.AUTH_USER_MODEL,
            ),
            preserve_default=False,
        ),
    ]
```

## 0005_alter_restaurant_images_alter_restaurant_owner.py

```
# Generated by Django 5.1.1 on 2024-10-14 09:40

import django.db.models.deletion
from django.conf import settings
from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0004_restaurant_owner"),
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
    ]

    operations = [
        migrations.AlterField(
            model_name="restaurant",
            name="images",
            field=models.ImageField(
                blank=True, null=True, upload_to="restaurant_images/"
            ),
        ),
        migrations.AlterField(
            model_name="restaurant",
            name="owner",
            field=models.OneToOneField(
                on_delete=django.db.models.deletion.CASCADE, to=settings.AUTH_USER_MODEL
            ),
        ),
    ]
```

## 0006_restaurantimage.py

```
# Generated by Django 5.1.1 on 2024-10-14 10:49

import django.db.models.deletion
from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0005_alter_restaurant_images_alter_restaurant_owner"),
    ]

    operations = [
        migrations.CreateModel(
            name="RestaurantImage",
            fields=[
                (
                    "id",
                    models.BigAutoField(
                        auto_created=True,
                        primary_key=True,
                        serialize=False,
                        verbose_name="ID",
                    ),
                ),
                ("image", models.ImageField(upload_to="restaurant_additional_images/")),
                (
                    "restaurant",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE,
                        related_name="additional_images",
                        to="core.restaurant",
                    ),
                ),
            ],
        ),
    ]
```

## 0007_alter_restaurantimage_image.py

```
# Generated by Django 5.1.1 on 2024-10-14 11:50

from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0006_restaurantimage"),
    ]

    operations = [
        migrations.AlterField(
            model_name="restaurantimage",
            name="image",
            field=models.ImageField(
                blank=True, null=True, upload_to="restaurant_additional_images/"
            ),
        ),
    ]
```

## 0008_chosenfood.py

```
# Generated by Django 5.1.1 on 2024-10-30 19:06

import django.db.models.deletion
from django.conf import settings
from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0007_alter_restaurantimage_image"),
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
    ]

    operations = [
        migrations.CreateModel(
            name="ChosenFood",
            fields=[
                (
                    "id",
                    models.BigAutoField(
                        auto_created=True,
                        primary_key=True,
                        serialize=False,
                        verbose_name="ID",
                    ),
                ),
                ("chosen_at", models.DateTimeField(auto_now_add=True)),
                (
                    "food",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE, to="core.food"
                    ),
                ),
                (
                    "user",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE,
                        to=settings.AUTH_USER_MODEL,
                    ),
                ),
            ],
        ),
    ]
```

## 0009_rename_comment_thread_comment_thread_image.py

```
# Generated by Django 5.1.1 on 2024-11-20 14:07

from django.conf import settings
from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0008_chosenfood"),
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
    ]

    operations = [
        migrations.RenameModel(
            old_name="Comment",
            new_name="Thread_comment",
        ),
        migrations.AddField(
            model_name="thread",
            name="image",
            field=models.ImageField(blank=True, null=True, upload_to="thread_images/"),
        ),
    ]
```

## 0010_remove_thread_comment_thread_and_more.py

```
# Generated by Django 5.1.1 on 2024-11-20 14:12

import django.db.models.deletion
from django.conf import settings
from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0009_rename_comment_thread_comment_thread_image"),
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
    ]

    operations = [
        migrations.RemoveField(
            model_name="thread_comment",
            name="thread",
        ),
        migrations.RemoveField(
            model_name="thread_comment",
            name="user",
        ),
        migrations.CreateModel(
            name="Forum",
            fields=[
                (
                    "id",
                    models.BigAutoField(
                        auto_created=True,
                        primary_key=True,
                        serialize=False,
                        verbose_name="ID",
                    ),
                ),
                ("title", models.CharField(max_length=255)),
                ("content", models.TextField()),
                ("created_at", models.DateTimeField(auto_now_add=True)),
                (
                    "image",
                    models.ImageField(blank=True, null=True, upload_to="forum_images/"),
                ),
                (
                    "user",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE,
                        to=settings.AUTH_USER_MODEL,
                    ),
                ),
            ],
        ),
        migrations.CreateModel(
```

```
            name="Forum_comment",
            fields=[
                (
                    "id",
                    models.BigAutoField(
                        auto_created=True,
                        primary_key=True,
                        serialize=False,
                        verbose_name="ID",
                    ),
                ),
                ("content", models.TextField()),
                ("created_at", models.DateTimeField(auto_now_add=True)),
                (
                    "Forum",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE, to="core.forum"
                    ),
                ),
                (
                    "user",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE,
                        to=settings.AUTH_USER_MODEL,
                    ),
                ),
            ],
        ),
        migrations.DeleteModel(
            name="Thread",
        ),
        migrations.DeleteModel(
            name="Thread_comment",
        ),
    ]
```

## 0011_remove_food_subcategory_remove_food_category_and_more.py

```
# Generated by Django 5.1.1 on 2024-12-04 07:43

from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0010_remove_thread_comment_thread_and_more"),
    ]

    operations = [
        migrations.RemoveField(
            model_name="food",
            name="subcategory",
        ),
        migrations.RemoveField(
            model_name="food",
            name="category",
        ),
        migrations.DeleteModel(
            name="SubCategory",
        ),
        migrations.AddField(
            model_name="food",
            name="category",
            field=models.ManyToManyField(related_name="foods", to="core.category"),
        ),
    ]
```

## 0012_likedislikefood_delete_chosenfood.py

```
# Generated by Django 5.1.1 on 2024-12-10 05:09

import django.db.models.deletion
from django.conf import settings
from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0011_remove_food_subcategory_remove_food_category_and_more"),
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
    ]

    operations = [
        migrations.CreateModel(
            name="LikeDislikeFood",
            fields=[
                (
                    "id",
                    models.BigAutoField(
                        auto_created=True,
                        primary_key=True,
                        serialize=False,
                        verbose_name="ID",
                    ),
                ),
                ("liked", models.BooleanField()),
                ("timestamp", models.DateTimeField(auto_now_add=True)),
                (
                    "food",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE,
                        related_name="likes_dislikes",
                        to="core.food",
                    ),
                ),
                (
                    "user",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE,
                        related_name="like_dislike_foods",
                        to=settings.AUTH_USER_MODEL,
                    ),
                ),
            ],
        ),
        migrations.DeleteModel(
            name="ChosenFood",
        ),
    ]
```

## 0013_restaurantcategory_restaurant_categories.py

```
# Generated by Django 5.1.1 on 2024-12-13 08:50

from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0012_likedislikefood_delete_chosenfood"),
    ]

    operations = [
        migrations.CreateModel(
            name="RestaurantCategory",
            fields=[
                (
                    "id",
                    models.BigAutoField(
                        auto_created=True,
                        primary_key=True,
                        serialize=False,
                        verbose_name="ID",
                    ),
                ),
                ("name", models.CharField(max_length=255)),
            ],
        ),
        migrations.AddField(
            model_name="restaurant",
            name="categories",
            field=models.ManyToManyField(
                related_name="restaurants", to="core.restaurantcategory"
            ),
        ),
    ]
```

## 0014_forumcomment_delete_forum_comment.py

```
# Generated by Django 5.1.1 on 2024-12-15 07:24

import django.db.models.deletion
from django.conf import settings
from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0013_restaurantcategory_restaurant_categories"),
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
    ]

    operations = [
        migrations.CreateModel(
            name="ForumComment",
            fields=[
                (
                    "id",
                    models.BigAutoField(
                        auto_created=True,
                        primary_key=True,
                        serialize=False,
                        verbose_name="ID",
                    ),
                ),
                ("content", models.TextField()),
                ("created_at", models.DateTimeField(auto_now_add=True)),
                (
                    "forum",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE,
                        related_name="comments",
                        to="core.forum",
                    ),
                ),
                (
                    "user",
                    models.ForeignKey(
                        on_delete=django.db.models.deletion.CASCADE,
                        to=settings.AUTH_USER_MODEL,
                    ),
                ),
            ],
        ),
        migrations.DeleteModel(
            name="Forum_comment",
        ),
    ]
```

## 0015_alter_food_description.py

```
# Generated by Django 5.1.1 on 2024-12-18 12:58

from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0014_forumcomment_delete_forum_comment"),
    ]

    operations = [
        migrations.AlterField(
            model_name="food",
            name="description",
            field=models.TextField(blank=True, null=True),
        ),
    ]
```

## 0016_restaurant_saved_by.py

```
# Generated by Django 5.1.1 on 2025-01-25 03:59

from django.conf import settings
from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0015_alter_food_description"),
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
    ]

    operations = [
```

```
        migrations.AddField(
            model_name="restaurant",
            name="saved_by",
            field=models.ManyToManyField(
                blank=True,
                related_name="saved_restaurants",
                to=settings.AUTH_USER_MODEL,
            ),
        ),
    ]
```

## 0017_forum_saved_by.py

```
# Generated by Django 5.1.1 on 2025-01-27 09:25

from django.conf import settings
from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0016_restaurant_saved_by"),
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
    ]

    operations = [
        migrations.AddField(
            model_name="forum",
            name="saved_by",
            field=models.ManyToManyField(
                blank=True, related_name="saved_forums", to=settings.AUTH_USER_MODEL
            ),
        ),
    ]
```

## 0018_restaurant_latitude_restaurant_longitude.py

```
# Generated by Django 5.1.1 on 2025-02-07 07:43

from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0017_forum_saved_by"),
    ]

    operations = [
        migrations.AddField(
            model_name="restaurant",
            name="latitude",
            field=models.FloatField(blank=True, null=True),
        ),
        migrations.AddField(
            model_name="restaurant",
            name="longitude",
            field=models.FloatField(blank=True, null=True),
        ),
    ]
```

## 0019_profile_latitude_profile_longitude.py

```
# Generated by Django 5.1.1 on 2025-02-07 07:58

from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0018_restaurant_latitude_restaurant_longitude"),
    ]

    operations = [
        migrations.AddField(
            model_name="profile",
            name="latitude",
            field=models.FloatField(blank=True, null=True),
        ),
        migrations.AddField(
            model_name="profile",
            name="longitude",
            field=models.FloatField(blank=True, null=True),
        ),
    ]
```

## 0020_alter_restaurant_location.py

```
# Generated by Django 5.1.1 on 2025-02-07 11:07

from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0019_profile_latitude_profile_longitude"),
    ]

    operations = [
        migrations.AlterField(
            model_name="restaurant",
            name="location",
            field=models.TextField(max_length=255),
        ),
    ]
```

## 0021_likedislikefood_source.py

```
# Generated by Django 5.1.1 on 2025-02-20 04:19

from django.db import migrations, models


class Migration(migrations.Migration):
    dependencies = [
        ("core", "0020_alter_restaurant_location"),
    ]

    operations = [
        migrations.AddField(
            model_name="likedislikefood",
```

```
            name="source",
            field=models.CharField(
                choices=[("random", "Random"), ("recommend", "Recommend")],
                default="random",
                max_length=20,
            ),
        ),
    ]
```

## __init__.py

## custom_tags.py

```
from django import template

register = template.Library()

@register.filter
def range_filter(value):
    return range(value)
```

## __init__.py