Приложение А. Текст программы

АННОТАЦИЯ

В данном программном документе приведён текст данные WEB-приложения хранения и размещения рецептов «РецептГид»

В разделе «Текст программы» указано наименование программы, область применения программы, модули программы в виде таблицы с указанием описания и размера каждого модуля, код программы.

СОДЕРЖАНИЕ

[1. ТЕКСТ ПРОГРАММЫ 3](#_Toc166003092)

[1.1. Наименование объекта 3](#_Toc166003093)

[1.2. Область применения объекта 3](#_Toc166003094)

[1.3. Модули 3](#_Toc166003095)

[1.4. Код программы 20](#_Toc166003096)

# ТЕКСТ ПРОГРАММЫ

## Наименование объекта

Наименование - « РецептГид».

## Область применения объекта

Приложение «РецептГид» предоставляет функционал для размещения и поиска рецептов блюд. Приложение включает в себя удобный интерфейс для добавления рецептов, возможность поиска по времени и категориям, что позволит пользователям легко находить и делиться любимыми блюдами

## Модули

В таблице 1 представлены модули WEB-приложения хранения и размещения рецептов. В WEB-приложении представлено 8 модулей с количеством кода – 2100. У пользователя 4 модуля – 573, а у работника – 191.

Таблица 1 – Модули

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название | Описание | Количество строк |
| 1 | 2 | 3 | 4 |
| 1 | user\_or\_mod | Определяет, является ли пользователь модератором или пользователем и возвращает результат проверки | 8 |
| 2 | adm | Определяет, является ли пользователь админом и возвращает результат проверки | 8 |
| 3 | mod | Определяет, является ли пользователь модератором и возвращает результат проверки | 15 |
| 4 | edit\_profile | Позволяет пользователю редактировать свою учетную запись и личную информацию | 14 |
| 5 | statistics | Отображает статистические данные о пользователях и рецептах | 60 |
| 6 | favorite\_recipe\_list | Выводит список рецептов, добавленных пользователем в избранное. | 12 |
| 7 | not\_verified\_recipes\_list | Выводит список рецептов, которые еще не были проверены модераторами. | 13 |
| 8 | recipe\_steps\_list | Предоставляет последовательность шагов для приготовления конкретного рецепта. | 11 |
| 9 | add\_recipe\_step | Добавляет новый шаг в процесс приготовления рецепта. | 20 |
| 10 | add\_or\_delete\_friend | Позволяет пользователю добавлять или удалять людей из своего списка подписок. | 16 |
| 11 | add\_or\_delete\_favorite\_recipe | Добавляет или удаляет рецепт из списка избранных у пользователя. | 17 |
| 12 | accept\_recipe | Принимает или отклоняет рецепт для публикации после его проверки модератором. | 31 |
| 13 | manage\_ingredients | Класс данных, представляет собой модель данных для управления оформления заказа, где можно очистить корзину, удалить определенный заказ и оформить заказ | 32 |
| 14 | manage\_tags | Позволяет добавлять, теги, в рецепт | 23 |
| 15 | BackupSQLView | Создает резервную копию базы данных в формате SQL. | 46 |
| 16 | user\_relationships | Выводит подписки или подписчиков определенного пользователя | 22 |
| 17 | BackupCSVView | Создает резервную копию данных в формате CSV. | 86 |
| 18 | upload\_sql\_file | Загружает файл SQL для восстановления базы данных или импорта данных. | 27 |
| 19 | admin\_home | Отображает главную страницу административной панели. | 10 |
| 20 | add\_tag | Добавляет новый тег в БД. | 14 |
| 21 | delete\_user | Блокирует учетную запись пользователя в системе. | 15 |
| 22 | undelete\_user | Разблокирует учетную запись пользователя в системе. | 10 |
| 23 | change\_user\_role | Изменяет роль пользователя (например, с обычного на администратора или модератора). | 15 |
| 24 | user\_register | Обрабатывает регистрацию нового пользователя в системе. | 12 |
| 25 | home | Отображает главную страницу сайта. | 16 |
| 26 | custom\_404\_view | Отображает пользовательскую страницу ошибки 404 (страница не найдена). | 9 |
| 27 | user\_profile | Показывает профиль конкретного пользователя с его данными и рецептами. | 25 |
| 28 | my\_profile | Позволяет пользователю просматривать свой личный профиль. | 15 |
| 29 | recipe\_detail | Отображает подробную информацию о конкретном рецепте. | 23 |
| 30 | create\_recipe | Позволяет пользователю создать новый рецепт и сохранить его в системе. | 17 |
| 31 | search\_recipe | Выполняет поиск рецептов по заданным критериям или ключевым словам. | 27 |
| 32 | user\_login | Обрабатывает процесс входа пользователя в систему. | 17 |
| 33 | user\_logout | Завершает сеанс пользователя и выводит его из системы. | 3 |

## Код программы

Ниже представлен код моделей, которые в последствии преобразуются в таблицы базы данных sqlite:

from django.db import models

from django.contrib.auth.models import AbstractUser

class User(AbstractUser):

description = models.TextField(blank=True, null=True)

photo = models.ImageField(upload\_to='user\_photos/', blank=True, null=True)

date\_of\_birth = models.DateField(blank=True, null=True)

rating = models.FloatField(default=0.0)

deletion\_reason = models.TextField(blank=True, null=True)

role = models.TextField(default="USER")

def \_\_str\_\_(self):

return self.username

class Recipe(models.Model):

title = models.CharField(max\_length=255)

user = models.ForeignKey(User, on\_delete=models.CASCADE)

header = models.CharField(max\_length=255)

cooking\_time = models.PositiveIntegerField()

description = models.TextField()

photo = models.ImageField(upload\_to='recipe\_photos/')

rating = models.FloatField(default=0.0)

deletion\_reason = models.TextField(default=None, blank=True, null=True)

servings = models.PositiveIntegerField()

is\_verified = models.BooleanField(default=False)

publication\_date = models.DateTimeField(auto\_now\_add=True)

kcal\_per\_100g = models.PositiveIntegerField()

protein\_per\_100g = models.FloatField()

fat\_per\_100g = models.FloatField()

carbohydrates\_per\_100g = models.FloatField()

def \_\_str\_\_(self):

return self.title

class Ingredient(models.Model):

name = models.CharField(max\_length=255)

kcal\_per\_100g = models.PositiveIntegerField()

protein\_per\_100g = models.FloatField()

fat\_per\_100g = models.FloatField()

carbohydrates\_per\_100g = models.FloatField()

def \_\_str\_\_(self):

return self.name

class RecipeStep(models.Model):

recipe = models.ForeignKey(Recipe, on\_delete=models.CASCADE)

step\_number = models.PositiveIntegerField()

photo = models.ImageField(upload\_to='recipe\_steps\_photos/', null=True, blank=True)

description = models.TextField()

class RecipeIngredient(models.Model):

ingredient = models.ForeignKey(Ingredient, on\_delete=models.CASCADE)

recipe = models.ForeignKey(Recipe, on\_delete=models.CASCADE)

quantity\_in\_grams = models.PositiveIntegerField()

class Tag(models.Model):

name = models.CharField(max\_length=255)

description = models.TextField()

def \_\_str\_\_(self):

return self.name

class RecipeTag(models.Model):

tag = models.ForeignKey(Tag, on\_delete=models.CASCADE)

recipe = models.ForeignKey(Recipe, on\_delete=models.CASCADE)

class Message(models.Model):

sender = models.ForeignKey(User, related\_name='sent\_messages', on\_delete=models.CASCADE)

recipient = models.ForeignKey(User, related\_name='received\_messages', on\_delete=models.CASCADE)

text = models.TextField()

timestamp = models.DateTimeField(auto\_now\_add=True)

class FavoriteRecipe(models.Model):

recipe = models.ForeignKey(Recipe, on\_delete=models.CASCADE)

user = models.ForeignKey(User, on\_delete=models.CASCADE)

class Review(models.Model):

user = models.ForeignKey(User, on\_delete=models.CASCADE)

rating = models.PositiveIntegerField(choices=[(i, str(i)) for i in range(1, 6)]) # Оценка от 1 до 5

title = models.CharField(max\_length=255)

text = models.TextField()

date\_written = models.DateTimeField(auto\_now\_add=True)

deletion\_reason = models.TextField(default='', null=True, blank=True)

def \_\_str\_\_(self):

return self.title

class Friend(models.Model):

requester = models.ForeignKey(User, related\_name='friend\_requests\_sent', on\_delete=models.CASCADE)

accepter = models.ForeignKey(User, related\_name='friend\_requests\_received', on\_delete=models.CASCADE)

class RecipeReview(models.Model):

review = models.ForeignKey(Review, on\_delete=models.CASCADE)

recipe = models.ForeignKey(Recipe, on\_delete=models.CASCADE)

Ниже представлен код сайта views-функций:

import csv

import os

import sqlite3

import datetime

from django.conf import settings

from django.contrib.auth import authenticate, logout, login

from django.contrib.auth.decorators import login\_required

from django.db import connection

from django.db.models import Q

from django.http import HttpResponse

from django.shortcuts import render, redirect, get\_object\_or\_404

from django.urls import reverse

from django.views import View

import matplotlib.pyplot as plt

from io import BytesIO

import base64

from main\_app.forms import \*

def user\_or\_mod(request):

try:

return request.user.role == "USER" or request.user.role == "MODER"

except:

return False

def adm(request):

try:

bool = request.user.role == "ADMIN"

return bool

except:

return False

def mod(request):

try:

bool = request.user.role == "MODER"

return bool

except:

return False

from django.db.models.signals import post\_save

from django.dispatch import receiver

@login\_required(login\_url='login')

def edit\_profile(request):

user = request.user # Получаем текущего пользователя

if request.method == 'POST':

form = UserChangeProfileForm(request.POST, request.FILES, instance=user)

if form.is\_valid():

form.save() # Сохраняем изменения в профиле

return redirect(reverse("user\_profile", args=[request.user.id])) # Перенаправляем на страницу профиля или другую страницу

else:

form = UserChangeProfileForm(instance=user) # Заполняем форму данными пользователя

return render(request, 'user/edit\_profile.html', {'form': form, 'user': user})

@login\_required(login\_url='login')

def statistics(request):

if adm(request):

# Получение данных о пользователях по ролям

roles\_count = User.objects.values('role').annotate(count=models.Count('id'))

roles = [role['role'] for role in roles\_count]

counts = [role['count'] for role in roles\_count]

# Создание графика для ролей

plt.figure(figsize=(10, 5))

plt.bar(roles, counts, color='skyblue')

# Настройка фона и текста

plt.gca().set\_facecolor('black') # Черный фон для графика

plt.title('Количество пользователей по ролям', color='blue') # Синий текст заголовка

plt.xlabel('Роли', color='blue') # Синий текст оси X

plt.ylabel('Количество пользователей', color='blue') # Синий текст оси Y

plt.tick\_params(axis='both', colors='blue') # Синий текст меток осей

# Сохранение графика в буфер

buf = BytesIO()

plt.savefig(buf, format='png', facecolor='black') # Указание черного фона при сохранении

plt.close()

buf.seek(0)

graph\_roles = base64.b64encode(buf.read()).decode('utf-8')

# Получение данных о рецептах по пользователям

recipes\_count = Recipe.objects.values('user\_\_username').annotate(count=models.Count('id'))

usernames = [recipe['user\_\_username'] for recipe in recipes\_count]

recipe\_counts = [recipe['count'] for recipe in recipes\_count]

# Создание графика для рецептов

plt.figure(figsize=(10, 5))

plt.bar(usernames, recipe\_counts, color='lightgreen')

# Настройка фона и текста

plt.gca().set\_facecolor('black') # Черный фон для графика

plt.title('Количество опубликованных рецептов у пользователей', color='blue') # Синий текст заголовка

plt.xlabel('Пользователи', color='blue') # Синий текст оси X

plt.ylabel('Количество рецептов', color='blue') # Синий текст оси Y

plt.tick\_params(axis='both', colors='blue') # Синий текст меток осей

# Сохранение графика в буфер

buf = BytesIO()

plt.savefig(buf, format='png', facecolor='black') # Указание черного фона при сохранении

plt.close()

buf.seek(0)

graph\_recipes = base64.b64encode(buf.read()).decode('utf-8')

# Передача графиков в контекст для отображения на странице

context = {

'graph\_roles': graph\_roles,

'graph\_recipes': graph\_recipes,

}

return render(request, 'admin/statistics.html', context)

else:

return redirect('page\_404')

@login\_required(login\_url='login')

def favorite\_recipe\_list(request):

if user\_or\_mod(request):

current\_user = request.user

favorite\_recipes = FavoriteRecipe.objects.filter(user=current\_user).select\_related('recipe')

recipes = [favorite.recipe for favorite in favorite\_recipes if

favorite.recipe.is\_verified and favorite.recipe.deletion\_reason == '']

context = {'recipes': recipes}

return render(request, 'user/favorite\_recipes.html', context)

else:

return redirect('page\_404')

@login\_required(login\_url='login')

def not\_verified\_recipes\_list(request):

if mod(request):

recipes = Recipe.objects.filter(

is\_verified=False

).filter(

Q(deletion\_reason='') | Q(deletion\_reason\_\_isnull=True)

).order\_by('-publication\_date')

context = {'recipes': recipes}

return render(request, 'user/not\_verified\_recipes.html', context)

else:

return redirect('page\_404')

@login\_required(login\_url='login')

def recipe\_steps\_list(request, recipe\_id):

if user\_or\_mod(request):

recipe = get\_object\_or\_404(Recipe, id=recipe\_id)

steps = RecipeStep.objects.filter(recipe=recipe).order\_by('step\_number')

context = {

'recipe': recipe,

'steps': steps

}

return render(request, 'user/recipe\_steps.html', context)

@login\_required(login\_url='login')

def add\_recipe\_step(request, recipe\_id):

if user\_or\_mod(request):

recipe = get\_object\_or\_404(Recipe, id=recipe\_id)

if recipe.user.id != request.user.id or recipe.is\_verified or recipe.deletion\_reason != None and recipe.deletion\_reason != '':

redirect('page\_404')

if request.method == 'POST':

form = RecipeStepForm(request.POST, request.FILES)

if form.is\_valid():

recipe\_step = form.save(commit=False)

recipe\_step.recipe = recipe

recipe\_step.save()

return redirect('add\_recipe\_step', recipe\_id=recipe.id)

else:

form = RecipeStepForm()

return render(request, 'user/add\_recipe\_step.html', {'form': form, 'recipe': recipe})

else:

return redirect('page\_404')

@login\_required(login\_url="login")

def add\_or\_delete\_friend(request, id):

if user\_or\_mod(request):

try:

accepter = User.objects.get(id=id)

friend = Friend.objects.filter(requester=request.user, accepter=accepter)

if friend.exists():

friend.delete()

return redirect(reverse("user\_profile", args=[id]))

Friend.objects.create(requester=request.user, accepter=accepter)

return redirect(reverse("user\_profile", args=[id]))

except User.DoesNotExist:

return redirect('page\_404')

else:

return redirect('page\_404')

@login\_required(login\_url="login")

def add\_or\_delete\_favorite\_recipe(request, id):

if user\_or\_mod(request):

try:

recipe = Recipe.objects.get(id=id)

favorite = FavoriteRecipe.objects.filter(recipe=recipe, user=request.user)

if favorite.exists():

favorite.delete()

return redirect(reverse("recipe\_detail", args=[id]))

FavoriteRecipe.objects.create(recipe=recipe, user=request.user)

return redirect(reverse("recipe\_detail", args=[id]))

except Recipe.DoesNotExist:

return redirect('page\_404')

else:

return redirect('page\_404')

@login\_required(login\_url='login')

def accept\_recipe(request, recipe\_id, is\_accept):

if mod(request):

recipe\_delete\_form = RecipeDeleteForm()

if request.method == 'GET':

is\_accept = is\_accept.lower() == 'true'

recipe = get\_object\_or\_404(Recipe, id=recipe\_id)

if (is\_accept):

recipe.is\_verified = True

recipe.deletion\_reason = ''

recipe.save()

return redirect(reverse("recipe\_detail", args=[recipe\_id]))

else:

context = {

'recipe': recipe,

'form': recipe\_delete\_form

}

return render(request, 'user/delete\_recipe.html', context)

else:

recipe = get\_object\_or\_404(Recipe, id=recipe\_id)

recipe\_delete\_form = RecipeDeleteForm(request.POST)

if recipe\_delete\_form.is\_valid():

# Обработка успешной валидации

recipe.deletion\_reason = recipe\_delete\_form.cleaned\_data['deletion\_reason']

recipe.save()

return redirect(reverse("home"))

else:

print(recipe\_delete\_form.errors) # Вывод ошибок

else:

return redirect('page\_404')

@login\_required(login\_url="login")

def manage\_ingredients(request, recipe\_id):

if user\_or\_mod(request):

recipe = get\_object\_or\_404(Recipe, id=recipe\_id)

if recipe.user.id != request.user.id or recipe.is\_verified or recipe.deletion\_reason != None and recipe.deletion\_reason != '':

redirect('page\_404')

if request.method == 'POST':

# Обработка формы для создания нового ингредиента

ingredient\_form = IngredientForm(request.POST)

recipe\_ingredient\_form = RecipeIngredientForm(request.POST)

if ingredient\_form.is\_valid():

ingredient\_form.save()

return redirect(reverse("manage\_ingredients", args=[recipe\_id]))

# После создания нового ингредиента можно добавить его к рецепту

if recipe\_ingredient\_form.is\_valid():

recipe\_ingredient = recipe\_ingredient\_form.save(commit=False)

recipe\_ingredient.recipe = recipe

recipe\_ingredient.save()

return redirect(reverse("manage\_ingredients", args=[recipe\_id]))

else:

ingredient\_form = IngredientForm()

recipe\_ingredient\_form = RecipeIngredientForm()

context = {

'recipe': recipe,

'ingredient\_form': ingredient\_form,

'recipe\_ingredient\_form': recipe\_ingredient\_form,

}

return render(request, 'user/manage\_ingredients.html', context)

@login\_required(login\_url="login")

def manage\_tags(request, recipe\_id):

if user\_or\_mod(request):

recipe = get\_object\_or\_404(Recipe, id=recipe\_id)

if recipe.user.id != request.user.id or recipe.is\_verified or recipe.deletion\_reason != None and recipe.deletion\_reason != '':

redirect('page\_404')

if request.method == 'POST':

recipe\_tag\_form = RecipeTagHiddenForm(request.POST)

if recipe\_tag\_form.is\_valid():

recipe\_tag = recipe\_tag\_form.save(commit=False)

recipe\_tag.recipe = recipe

recipe\_tag.save()

return redirect(reverse("manage\_tags", args=[recipe\_id]))

else:

recipe\_tag\_form = RecipeTagHiddenForm()

context = {

'recipe': recipe,

'recipe\_tag\_form': recipe\_tag\_form,

}

return render(request, 'user/manage\_tags.html', context)

@login\_required(login\_url='login')

def user\_relationships(request, user\_id, is\_subscriber):

if (user\_or\_mod(request)):

is\_subscriber = is\_subscriber.lower() == 'true'

user = get\_object\_or\_404(User, id=user\_id)

if is\_subscriber: # Если true, получаем подписчиков

followers = Friend.objects.filter(accepter=user)

context = {

'user': user,

'followers': followers,

}

return render(request, 'user/followers\_list.html', context)

else: # Если false, получаем подписки

following = Friend.objects.filter(requester=user).select\_related('accepter')

context = {

'user': user,

'following': following,

}

return render(request, 'user/following\_list.html', context)

else:

redirect('page\_404')

class BackupSQLView(View):

def get(self, request):

if adm(request):

response = HttpResponse(content\_type='application/sql')

response['Content-Disposition'] = 'attachment; filename=backup.sql'

with connection.cursor() as cursor:

# Получаем список всех таблиц в базе данных SQLite

cursor.execute("SELECT name FROM sqlite\_master WHERE type='table';")

tables = cursor.fetchall()

for table in tables:

table\_name = table[0]

if table\_name != "sqlite\_sequence":

# Получаем SQL команду создания таблицы

cursor.execute(f"SELECT sql FROM sqlite\_master WHERE name='{table\_name}';")

create\_table\_sql = cursor.fetchone()[0]

response.write(f"{create\_table\_sql};")

# Получаем данные из таблицы

cursor.execute(f"SELECT \* FROM {table\_name};")

rows = cursor.fetchall()

columns = [column[0] for column in cursor.description]

for row in rows:

formatted\_values = []

for value in row:

if isinstance(value, datetime.datetime):

# Форматируем datetime в стандартный вид

formatted\_value = value.strftime('%Y-%m-%d %H:%M:%S')

formatted\_values.append(f"'{formatted\_value}'")

elif isinstance(value, str):

# Экранируем строки

old = "'"

new = "''"

formatted\_values.append(f"'{value.replace(old, new)}'")

else:

# Для других типов (int, float и т.д.) просто добавляем значение

formatted\_values.append(str(value))

sql = f"INSERT INTO {table\_name} ({', '.join(columns)}) VALUES ({', '.join(formatted\_values)});"

response.write(sql)

return response

else:

return redirect('page\_404')

class BackupCSVView(View):

def get(self, request):

if adm(request):

response = HttpResponse(content\_type='text/csv')

response['Content-Disposition'] = 'attachment; filename=backup.csv'

writer = csv.writer(response)

writer.writerow(['Model', 'ID', 'Data']) # Заголовки

# Сохраняем данные пользователей

for user in User.objects.all():

writer.writerow(['User', user.id, user.username])

writer.writerow(['User Description', user.id, user.description])

writer.writerow(['User Date of Birth', user.id, user.date\_of\_birth])

writer.writerow(['User Rating', user.id, user.rating])

writer.writerow(['User Role', user.id, user.role])

# Сохраняем данные рецептов

for recipe in Recipe.objects.all():

writer.writerow(['Recipe', recipe.id, recipe.title])

writer.writerow(['Recipe Header', recipe.id, recipe.header])

writer.writerow(['Recipe Cooking Time', recipe.id, recipe.cooking\_time])

writer.writerow(['Recipe Description', recipe.id, recipe.description])

writer.writerow(['Recipe Rating', recipe.id, recipe.rating])

writer.writerow(['Recipe Servings', recipe.id, recipe.servings])

writer.writerow(['Recipe Is Verified', recipe.id, recipe.is\_verified])

writer.writerow(['Recipe Publication Date', recipe.id, recipe.publication\_date])

writer.writerow(['Recipe kcal per 100g', recipe.id, recipe.kcal\_per\_100g])

writer.writerow(['Recipe Protein per 100g', recipe.id, recipe.protein\_per\_100g])

writer.writerow(['Recipe Fat per 100g', recipe.id, recipe.fat\_per\_100g])

writer.writerow(['Recipe Carbohydrates per 100g', recipe.id, recipe.carbohydrates\_per\_100g])

for ingredient in Ingredient.objects.all():

writer.writerow(['Ingredient', ingredient.id, ingredient.name])

writer.writerow(['Ingredient kcal per 100g', ingredient.id, ingredient.kcal\_per\_100g])

writer.writerow(['Ingredient Protein per 100g', ingredient.id, ingredient.protein\_per\_100g])

writer.writerow(['Ingredient Fat per 100g', ingredient.id, ingredient.fat\_per\_100g])

writer.writerow(['Ingredient Carbohydrates per 100g', ingredient.id, ingredient.carbohydrates\_per\_100g])

# Сохраняем данные шагов рецепта

for step in RecipeStep.objects.all():

writer.writerow(['Recipe Step', step.id, step.description])

writer.writerow(['Step Number', step.id, step.step\_number])

# Сохраняем данные ингредиентов рецепта

for recipe\_ingredient in RecipeIngredient.objects.all():

writer.writerow(['Recipe Ingredient', recipe\_ingredient.id, f'Recipe: {recipe\_ingredient.recipe.id}, Ingredient: {recipe\_ingredient.ingredient.id}, Quantity: {recipe\_ingredient.quantity\_in\_grams}'])

# Сохраняем данные тегов

for tag in Tag.objects.all():

writer.writerow(['Tag', tag.id, tag.name])

writer.writerow(['Tag Description', tag.id, tag.description])

# Сохраняем данные тегов рецептов

for recipe\_tag in RecipeTag.objects.all():

writer.writerow(['Recipe Tag', recipe\_tag.id, f'Tag: {recipe\_tag.tag.id}, Recipe: {recipe\_tag.recipe.id}'])

# Сохраняем сообщения

for message in Message.objects.all():

writer.writerow(['Message', message.id,

f'Sender: {message.sender.id}, Recipient: {message.recipient.id}, Text: {message.text}, Timestamp: {message.timestamp}'])

# Сохраняем избранные рецепты

for favorite\_recipe in FavoriteRecipe.objects.all():

writer.writerow(['Favorite Recipe', favorite\_recipe.id,

f'Recipe: {favorite\_recipe.recipe.id}, User: {favorite\_recipe.user.id}'])

# Сохраняем отзывы

for review in Review.objects.all():

writer.writerow(['Review', review.id,

f'User: {review.user.id}, Rating: {review.rating}, Title: {review.title}, Text: {review.text}, Date Written: {review.date\_written}'])

# Сохраняем друзей

for friend in Friend.objects.all():

writer.writerow(

['Friendship', friend.id, f'Requester: {friend.requester.id}, Accepter: {friend.accepter.id}'])

for recipe\_review in RecipeReview.objects.all():

writer.writerow(

['Recipe Review', recipe\_review.id,

f'Recipe: {recipe\_review.recipe.id}', f'Review: {recipe\_review.review.id}']

)

return response

else:

return redirect('page\_404')

@login\_required(login\_url='login')

def upload\_sql\_file(request):

if adm(request):

if request.method == 'POST':

form = UploadSQLFileForm(request.POST, request.FILES)

if form.is\_valid():

sql\_file = request.FILES['sql\_file']

with open('temp.sql', 'wb+') as temp\_file:

for chunk in sql\_file.chunks():

temp\_file.write(chunk)

db\_path = settings.DATABASES['default']['NAME']

connection.close()

if os.path.exists(db\_path):

os.remove(db\_path)

conn = sqlite3.connect(settings.DATABASES['default']['NAME'])

cursor = conn.cursor()

with open('temp.sql', 'r') as file:

sql\_script = file.read()

cursor.executescript(sql\_script)

conn.commit()

conn.close()

os.remove('temp.sql')

return render(request, 'sql/upload\_success.html')

else:

return redirect('page\_404')

@login\_required(login\_url='login')

def admin\_home(request):

if adm(request):

tags = Tag.objects.all()

users = User.objects.all()

sql\_form = UploadSQLFileForm()

return render(request, 'admin/home.html', {'tags': tags, 'users': users, 'sql\_form': sql\_form})

else:

return redirect('page\_404')

@login\_required(login\_url='login')

def add\_tag(request):

if adm(request):

if request.method == 'POST':

form = TagForm(request.POST)

if form.is\_valid():

form.save()

return redirect('admin\_home')

else:

form = TagForm()

return render(request, 'admin/add\_tag.html', {'form': form})

else:

return redirect('page\_404')

@login\_required(login\_url='login')

def delete\_user(request, user\_id):

if adm(request) and user\_id != request.user.id:

user = get\_object\_or\_404(User, id=user\_id)

if request.method == 'POST':

form = UserDeletionForm(request.POST, instance=user)

if form.is\_valid():

form.save()

return redirect('admin\_home')

else:

form = UserDeletionForm(instance=user)

return render(request, 'admin/delete\_user.html', {'form': form, 'user': user})

else:

return redirect('page\_404')

@login\_required(login\_url='login')

def undelete\_user(request, user\_id):

if adm(request) and user\_id != request.user.id:

user = get\_object\_or\_404(User, id=user\_id)

user.deletion\_reason = ''

user.save()

return redirect('admin\_home')

else:

return redirect('page\_404')

@login\_required(login\_url='login')

def change\_user\_role(request, user\_id):

if adm(request) and user\_id != request.user.id:

user = get\_object\_or\_404(User, id=user\_id)

if request.method == 'POST':

form = UserRoleChangeForm(request.POST, instance=user)

if form.is\_valid():

form.save()

return redirect('admin\_home')

else:

form = UserRoleChangeForm(instance=user)

return render(request, 'admin/change\_user\_role.html', {'form': form, 'user': user})

else:

return redirect('page\_404')

def user\_register(request):

if request.method == 'POST':

form = CustomUserCreationForm(request.POST, request.FILES)

if form.is\_valid():

form.save()

return redirect('login')

else:

form = CustomUserCreationForm()

return render(request, 'auth/register.html', {'form': form})

@login\_required(login\_url='login')

def home(request):

if user\_or\_mod(request):

moder = False

if mod(request):

moder = True

recipes = Recipe.objects.filter(is\_verified=True, deletion\_reason='').order\_by('-publication\_date')

context = {'recipes': recipes, 'moder': moder}

return render(request, 'home.html', context)

else:

if adm(request):

return redirect('admin\_home')

return redirect('page\_404')

def custom\_404\_view(request):

return render(request, '404.html')

@login\_required(login\_url='login')

def user\_profile(request, pk):

if user\_or\_mod(request):

user = get\_object\_or\_404(User, id=pk)

your\_acc = False

in\_friends = Friend.objects.filter(requester=request.user.id, accepter=pk).exists()

podpisok = Friend.objects.filter(requester=pk)

podpischikov = Friend.objects.filter(accepter=pk)

if request.user.id == pk:

recipes = user.recipe\_set.filter().order\_by('-publication\_date')

your\_acc = True

else:

recipes = user.recipe\_set.filter(is\_verified=True, deletion\_reason='').order\_by('-publication\_date')

podpisok\_count = int(len(podpisok))

podpischikov\_count = int(len(podpischikov))

context = {'user': user, 'recipes': recipes, 'your\_acc': your\_acc,

'in\_friends': in\_friends, 'podpisok\_count': podpisok\_count, 'podpischikov\_count': podpischikov\_count

}

return render(request, 'user/user\_profile.html', context)

else:

return redirect('page\_404')

@login\_required(login\_url='login')

def my\_profile(request):

if user\_or\_mod(request):

user = get\_object\_or\_404(User, id=request.user.id)

podpischikov = Friend.objects.filter(accepter=request.user.id)

podpisok = Friend.objects.filter(requester=request.user.id)

podpischikov\_count = int(len(podpischikov))

podpisok\_count = int(len(podpisok))

recipes = user.recipe\_set.filter().order\_by('-publication\_date')

context = {'user': user, 'recipes': recipes, 'your\_acc': True, 'podpisok\_count': podpisok\_count, 'podpischikov\_count': podpischikov\_count}

return render(request, 'user/user\_profile.html', context)

else:

return redirect('page\_404')

@login\_required(login\_url='login')

def recipe\_detail(request, id):

if user\_or\_mod(request):

recipe = get\_object\_or\_404(Recipe, id=id)

ingredients = RecipeIngredient.objects.filter(recipe=recipe)

ingredient\_list = [(ingredient.ingredient.name, ingredient.quantity\_in\_grams) for ingredient in ingredients]

is\_fav = FavoriteRecipe.objects.filter(user=request.user.id, recipe=recipe).exists()

if not recipe.is\_verified:

if request.user.id == recipe.user.id or mod(request):

if mod(request):

context = {'recipe': recipe, 'ingredients': ingredient\_list, 'moder': True}

return render(request, 'user/recipe\_detail.html', context)

context = {'recipe': recipe, 'ingredients': ingredient\_list}

return render(request, 'user/recipe\_detail.html', context)

else:

return redirect('page\_404')

else:

context = {'recipe': recipe, 'ingredients': ingredient\_list, 'is\_fav': is\_fav}

return render(request, 'user/recipe\_detail.html', context)

else:

return redirect('page\_404')

@login\_required(login\_url='login')

def create\_recipe(request):

if user\_or\_mod(request):

if request.method == 'POST':

form = RecipeForm(request.POST, request.FILES)

if form.is\_valid():

recipe = form.save(commit=False)

recipe.user = request.user

recipe.save()

return redirect('manage\_ingredients', recipe\_id=recipe.id)

else:

form = RecipeForm()

return render(request, 'user/create\_recipe.html', {'form': form})

else:

return redirect('page\_404')

@login\_required(login\_url='login')

def search\_recipe(request):

if user\_or\_mod(request):

recipes = Recipe.objects.filter(is\_verified=True) # Получаем все рецепты по умолчанию

# Фильтрация по названию

title\_query = request.GET.get('title', '')

if title\_query:

recipes = recipes.filter(title\_\_icontains=title\_query)

# Фильтрация по времени приготовления

cooking\_time\_query = request.GET.get('cooking\_time', '')

if cooking\_time\_query.isdigit():

recipes = recipes.filter(cooking\_time\_\_lte=int(cooking\_time\_query))

# Фильтрация по тегам

tag\_ids = request.GET.getlist('tags')

if tag\_ids:

recipes = recipes.filter(recipetag\_\_tag\_\_id\_\_in=tag\_ids).distinct()

tags = Tag.objects.all() # Получаем все теги для отображения в форме

return render(request, 'user/search\_recipe.html', {'recipes': recipes, 'tags': tags, 'selected\_tags': tag\_ids, 'title\_query': title\_query,

'cooking\_time\_query': cooking\_time\_query})

else:

return redirect('page\_404')

def user\_login(request):

if request.method == 'POST':

username = request.POST['username']

password = request.POST['password']

user = authenticate(request, username=username, password=password)

if user is not None:

if user.deletion\_reason is not None and user.deletion\_reason != '':

return render(request, 'auth/login.html',{'error': 'Вы были заблокированы по причине:' + user.deletion\_reason})

login(request, user)

if user.role == 'ADMIN':

return redirect('admin\_home')

return redirect('home')

else:

return render(request, 'auth/login.html', {'error': 'Неверное имя пользователя или пароль'})

return render(request, 'auth/login.html')

def user\_logout(request):

logout(request)

return redirect('login')