**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 804 «Теории вероятностей и математического моделирования»

Дисциплина: «Web-технологии разработки прикладных программных систем»

**Курсовая работа**

Студент: Лебеденко А.В.

Группа: М8О-104М-23

Преподаватель: Азанов В.М.

Дата: 27.12.2023

Оценка:

Москва, 2023

**Что такое Django?**

Django (*/ˈdʒæŋɡoʊ/ джанго*) -- бесплатный и свободный фреймворк для веб-приложений, написанный на Python. Фреймворк - это набор компонентов, которые помогают разрабатывать веб-сайты быстро и просто.

Каждый раз при разработке веб-сайтов требуются похожие компоненты: способ аутентифицировать пользователей (вход, выход, регистрация), панель управления сайтом, формы, инструменты для загрузки файлов и т. д.

К счастью для нас, другие люди обратили внимание на возникновение однотипных проблем при веб-разработке, так что они объединились и создали фреймворки (Django и другие), которые предлагают нам готовые шаблоны для использования.

Фреймворки существуют, чтобы облегчить процесс разработки и позволить нам не изобретать колесо.

**Зачем нам нужен фреймворк?**

Чтобы понять, для чего же нам нужен Django, нам нужно ближе познакомиться с серверами. Во-первых, сервер должен узнать о том, что мы ждем от него веб-страницу.

Представь себе почтовый ящик (порт), который проверяется на наличие новых писем (запросов). Это делает веб-сервер. Когда письмо приходит, сервер читает его и отправляет ответ с веб-страничкой. Однако, чтобы что-то отправить нам надо это что-то иметь. И Django как раз и отвечает за создание контента, который будет отправлен в ответе.

**Что происходит, когда кто-то запрашивает веб-сайт у твоего сервера?**

Когда на сервер приходит запрос, он переадресуется Django, который пытается сообразить что же конкретно от него просят. Для начала он берет адрес веб-страницы и пробует понять -- что же нужно сделать. Эту часть процесса в Django выполняет **urlresolver** (адрес веб-сайта называется URL - Uniform Resource Locator - Единый указатель ресурсов, так что название *urlresolver*, resolver == определитель, имеет определенный смысл). Он не слишком умен, поэтому просто берет список шаблонов и пытается сопоставить их с URL. Django сверяет шаблоны сверху вниз и, если что-то совпадает, он переправляет запрос соответствующей функции (которая называется *view*).

Представь себе почтальона с письмом. Она идет вниз по улицы и сверяет номера домов с адресом на письме. Если они совпадают, то она оставляет письмо. Так же работает и urlresolver!

Но самые интересные вещи происходят в функции *view*: мы, например, можем обращаться к базе данных за определенной информацией. Может быть пользователь попросил изменить какую-нибудь информацию? Как будто в письме написано: "Пожалуйста, поменяйте описание моей работы." Функция *view* может проверить имеете ли вы разрешение делать это, а затем обновит описание работы и отправит обратно ответ: "Готово!". Затем функция *view* сгенерирует ответ и Django сможет отправить его веб-браузеру пользователя.

**Создание проекта Blog на Django**

Первый шаг — создать новый проект Django. Попросту, это значит что мы запустим несколько стандартных скриптов из поставки Django, который создадут для нас скелет проекта. Это просто куча каталогов и файлов, которые мы будем использовать позже.

В консоли Mac OS или Linux нужно запустить следующую команду; не забудьте добавить точку . в конце:

(myvenv) ~/djangogirls$ django-admin startproject mysite .

В Windows:

(myvenv) C:\Users\Name\djangogirls> django-admin startproject mysite .

django-admin.py это скрипт, который создаст необходимую структуру директорий и файлы для нас. Ты должна теперь иметь следующую структуру проекта:

djangogirls

├───manage.py

└───mysite

settings.py

urls.py

wsgi.py

\_\_init\_\_.py

manage.py это другой скрипт, который помогает с управлением сайтом. С помощью него мы сможем запустить веб-сервер на твоем компьютере без установки дополнительных программ.

Файл settings.py содержит настройки для твоего веб-сайта.

Файл urls.py содержит список шаблонов, по которым ориентируется urlresolver.

**Изменяем настройки**

Давай внесем изменения в mysite/settings.py. Открой файл в текстовом редакторе, который ты выбрала ранее.

Было бы неплохо установить корректный часовой пояс на нашем сайте. Перейди к [списку часовых поясов википедии](http://en.wikipedia.org/wiki/List_of_tz_database_time_zones) и скопируй название своего часового пояса (TZ). (например, Europe/Moscow)

В файле settings.py найди строку, содержащую TIME\_ZONE, и модифицируй её в соответствии со своим часовым поясом:

TIME\_ZONE = 'Europe/Moscow'

Замени "Europe/Moscow" на соответствующий часовой пояс

Нам также необходимо добавить в настройки информацию о расположении статических файлов (мы познакомимся со статичными файлами и CSS в следующих главах). Спустись в *конец* файла и после переменной STATIC\_URL добавь новую - STATIC\_ROOT:

STATIC\_URL = '/static/'

STATIC\_ROOT = os.path.join(BASE\_DIR, 'static')

**Н**астро**йка базы данных**

Существует множество различных баз данных, которые могут хранить данные для твоего сайта. Мы будем использовать стандартную -- sqlite3.

Она уже выбрана по умолчанию в файле mysite/settings.py:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

Чтобы создать базу данных для нашего блога, набери в командной строке следующее: python manage.py migrate (мы должны быть в директории djangogirls, где расположен файл manage.py). Если все прошло успешно, то ты увидишь следующий результат:

(myvenv) ~/djangogirls$ python manage.py migrate

Operations to perform:

Synchronize unmigrated apps: messages, staticfiles

Apply all migrations: contenttypes, sessions, admin, auth

Synchronizing apps without migrations:

Creating tables...

Running deferred SQL...

Installing custom SQL...

Running migrations:

Rendering model states... DONE

Applying contenttypes.0001\_initial... OK

Applying auth.0001\_initial... OK

Applying admin.0001\_initial... OK

Applying contenttypes.0002\_remove\_content\_type\_name... OK

Applying auth.0002\_alter\_permission\_name\_max\_length... OK

Applying auth.0003\_alter\_user\_email\_max\_length... OK

Applying auth.0004\_alter\_user\_username\_opts... OK

Applying auth.0005\_alter\_user\_last\_login\_null... OK

Applying auth.0006\_require\_contenttypes\_0002... OK

Applying sessions.0001\_initial... OK

И мы закончили! Пришло время запустить веб-сервер и посмотреть работает ли наш веб-сайт!

Ты должна быть в директории, где расположен файл manage.py (в нашем случае -- djangogirls). Запустим веб-сервер из командной строки: python manage.py runserver:

(myvenv) ~/djangogirls$ python manage.py runserver

Если вы используете Windows, и команда падает с ошибкой UnicodeDecodeError, используйте вместо неё другую:

(myvenv) ~/djangogirls$ python manage.py runserver 0:8000

Теперь тебе нужно проверить работает ли веб-сайт - открой браузер (Firefox, Chrome, Safari, Internet Explorer или любой другой) и набери следующий адрес:

http://127.0.0.1:8000/

Веб-сервер займёт командную строку, пока вы его не остановите. Чтобы и дальше иметь возможность набирать команды, откройте ещё одно окно терминала и активируйте в нём виртуальное окружение. Чтобы остановить веб-сервер, перейдите обратно в окно, в котором он работает и нажмите CTRL + C - кнопки Control и C вместе (в Windows, может потребоваться нажать клавиши Ctrl + Break).

Модель в Django

Зная, что представляет из себя объект, мы можем создать Django модель для записи в блоге.

Модель в Django это объект определенного свойства - он хранится в базе данных. База данных представляет собой совокупность различных данных. Это то место, где ты будешь хранить информацию о своих пользователях, записях в блоге и т.д. Мы будем использовать базу данных SQLite для хранения информации. Это стандартная база данных в Django -- её вполне хватит для наших нужд сейчас.

Ты можешь представить модель в базе данных как электронную таблицу с колонками (полями) и строками (данными).

Создание приложения

Для аккуратности мы создадим отдельное приложение в нашем проекте. Очень удобно иметь хорошо организованное рабочее место с самого начала. Для создания приложения нам понадобиться набрать следующую инструкцию в командной строке (из директории djangogirls, где находится файл manage.py):

(myvenv) ~/djangogirls$ python manage.py startapp blog

Обрати внимание, что в нашем проекте появилась новая папка blog, которая содержит некоторые файлы. Структура нашего проекта, таким образом, будет выглядеть так:

djangogirls

├── mysite

| \_\_init\_\_.py

| settings.py

| urls.py

| wsgi.py

├── manage.py

└── blog

├── migrations

| \_\_init\_\_.py

├── \_\_init\_\_.py

├── admin.py

├── models.py

├── tests.py

└── views.py

После того как приложение создано, нам нужно сообщить Django, что теперь он должен его использовать. Мы сделаем это через файл mysite/settings.py. Нам нужно найти INSTALLED\_APPS и добавить к списку 'blog', прямо перед ). Конечный результат должен выглядеть следующим образом:

INSTALLED\_APPS = (

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'blog',

)

Создание модели записи в блоге

В файле blog/models.py мы определяем все Модели - модель записи для блога также пойдет сюда.

Открой файл blog/models.py, удали весь текст и вставь на его место следующий код:

from django.db import models

from django.utils import timezone

class Post(models.Model):

author = models.ForeignKey('auth.User')

title = models.CharField(max\_length=200)

text = models.TextField()

created\_date = models.DateTimeField(

default=timezone.now)

published\_date = models.DateTimeField(

blank=True, null=True)

def publish(self):

self.published\_date = timezone.now()

self.save()

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

return self.title

Смотрится страшно, да? Но не волнуйся, мы объясним что значит каждая строка кода!

Строки, начинающиеся с from или import, открывают доступ к коду из других файлов. Так что, вместо того, чтобы копировать и вставлять один и тот же код во все файлы, мы можешь сослаться на него при помощи from ... import ....

class Post(models.Model): -- эта строка определяет нашу модель (объект).

class это специальное ключевое слово для определения объектов.

Post это имя нашей модели, мы можем поменять его при желании (специальные знаки и пробелы использовать нельзя). Всегда начинай имена классов с прописной буквы.

models.Model означает что объект Post является моделью Django, так Django поймет, что он должен сохранить его в базу данных.

Дальше мы задаем свойства, о которых уже говорили: title, text, created\_date, published\_date и author. Чтобы это сделать нам нужно определиться с типом полей (это текст? число? дата? ссылка на другой объект? например, на пользователя?).

models.CharField -- так мы определяем текстовое поле с ограничением на количество символов.

models.TextField - так определяется поле для неограниченно длинного текста. Выглядит подходящим для содержимого поста, верно?

models.DateTimeField -- дата и время.

models.ForeignKey -- ссылка на другую модель.

Мы не будем объяснять каждую запятую, поскольку на это уйдет слишком много времени. Ознакомься с официальной документаций Django — если хочешь узнать больше о полях моделей и о том как определять разные объекты, то эта ссылка может помочь: (<https://docs.djangoproject.com/en/1.8/ref/models/fields/#field-types>).

Что насчет def publish(self):? Это как раз метод публикации для записи, о котором мы говорили. def означает, что создаётся функция/метод, а publish — это название этого метода. Можно изменить имя метода, если хочешь. Существует правило для имён функций: нужно использовать строчные буквы, а пробелы заменять на подчёркивания. Например метод, вычисляющий среднюю цену может называться calculate\_average\_price.

Методы часто возвращают что-то. Например, метод \_\_str\_\_. В наше случае, после вызова метода \_\_str\_\_() мы получим текст (**строку**) с заголовком записи.

Если тема моделей тебе до сих пор непонятна -- не стесняйся обратиться к тренеру! Мы знаем, что она действительно сложна, особенно когда ты изучаешь параллельно объекты и функции. Но мы надеемся, что все это кажется тебе теперь не магией!

Создаем таблицы моделей в базе данных

Последним шагом будет добавление нашей модели в базу данных. Сначала мы должны дать Django знать, что сделали изменения в нашей модели (мы её только что создали!). Набери python manage.py makemigrations blog. Должно получиться примерно так:

(myvenv) ~/djangogirls$ python manage.py makemigrations blog

Migrations for 'blog':

0001\_initial.py:

- Create model Post

Django создал для нас файл с миграцией для базы данных. Набери python manage.py migrate blog, результат должен быть следующим:

(myvenv) ~/djangogirls$ python manage.py migrate blog

Operations to perform:

Apply all migrations: blog

Running migrations:

Rendering model states... DONE

Applying blog.0001\_initial... OK

Администрирование Django

Чтобы добавлять, редактировать и удалять записи, для которых мы только сделали модель, нам потребуется использовать права администратора в Django.

Давай откроем файл blog/admin.py и заменим его содержимое на:

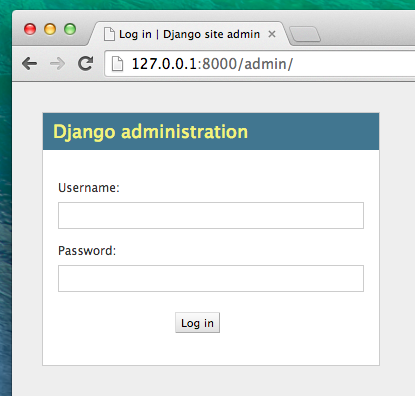
from django.contrib import admin

from .models import Post

admin.site.register(Post)

Как ты можешь заметить, мы импортировали (включили) модель Post, которая была определена в предыдущей главе. Чтобы наша модель стала доступна на странице администрирования, нам нужно зарегистрировать её при помощи admin.site.register(Post).

Хорошо, теперь нам нужно взглянуть на модель Post. Не забудь запустить веб-сервер командой python manage.py runserver. Перейди в браузер и набери адрес [http://127.0.0.1:8000/admin/](http://127.0.0.1:8000/admin/" \t "_blank) Ты увидишь страницу авторизации:



Чтобы залогиниться, тебе сначала нужно создать *суперпользователя (англ. superuser)* - пользователя, который имеет полный доступ к управлению сайтом. Вернись к командной строке, набери python manage.py createsuperuser, и нажми Enter. При появлении запроса введи имя пользователя (строчными буквами, без пробелов), адрес электронной почты и пароль. Не беспокойся, если пароль не появляется на экране по мере ввода, так и задумано. Просто напечатай его и нажмите Enter, чтобы продолжить. Результат должен выглядеть следующим образом (имя пользователя и почта соответственно будут твоими):

(myvenv) ~/djangogirls$ python manage.py createsuperuser

Username: admin

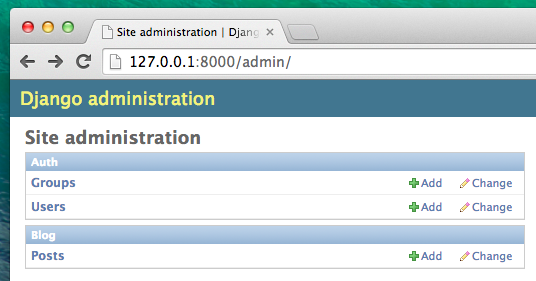
Email address: [[email protected]](https://ryangallen.gitbooks.io/cdn-cgi/l/email-protection)

Password:

Password (again):

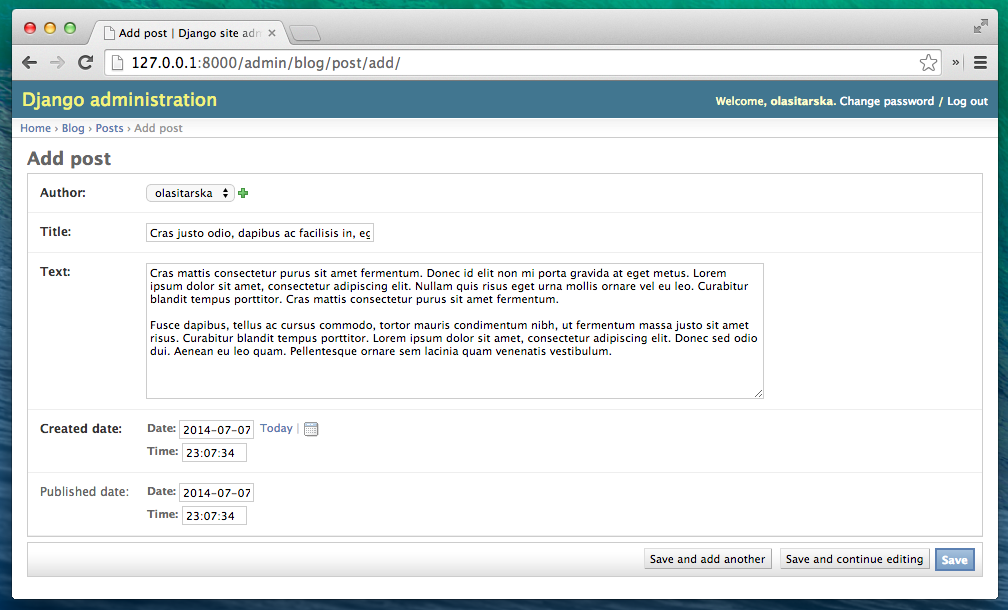
Superuser created successfully.

Вернись в браузер и войди в систему при помощи имени пользователя и пароля, которые ты только что выбрала. Ты должна попасть в панель управления Django.



Перейди к разделу Posts и немного поэкспериментируй с ним. Добавь пять или шесть постов. Не беспокойся о содержании - можешь просто скопировать и вставить текст из этого учебника, чтобы сэкономить время :).

Убедись, что выбрала для двух или трех записей (но не больше) дату публикации. Это пригодится позднее.

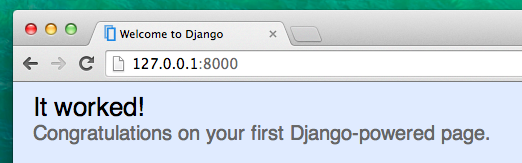


URL-адреса Django

Мы собираемся сделать нашу первую веб-страничку — домашнюю страницу твоего блога! Но для начала, давай чуть ближе познакомимся с URL-адресами в Django.

Что такое URL-адрес?

URL — это просто адрес в интернете. Ты можешь увидеть URL каждый раз, когда посещаешь веб-сайт — он отображается в адресной строке твоего браузера (да! 127.0.0.1:8000 это URL-адрес! И https://djangogirls.com — тоже URL):



Любая страница в Интернете нуждается в собственном URL-адресе. Таким образом ваше приложение точно знает, что показать пользователю, который открывает конкретный URL-адрес. В Django мы используем кое-что под названием URLconf (англ. URL configuration, конфигурация URL). URLconf — это набор шаблонов, которые Django попробует сравнить с полученным URL, чтобы выбрать правильный метод для отображения (view).

Динамически изменяющиеся данные в шаблонах

Отдельные части уже размещены в нужных местах: модель Post, определенна в файле models.py, post\_list - в файле views.py и добавлен шаблон. Но как нам отобразить записи в шаблоне HTML-страницы? Ведь именно этого нам и нужно добиться: взять определенный контент (модели, сохраненные в базе данных) и аккуратно отобразить их в шаблоне, верно?

Для этого и предназначены представления: соединять между собой модели и шаблоны. В post\_list представлениям нужно будет взять модели, которые мы хотим отобразить и передать их шаблону. Таким образом, в представлениях мы определяем что (какая модель) будет отображена в шаблоне.

Хорошо, так как мы этого добьемся?

Нам нужно открыть файл blog/views.py. Таким образом представление post\_list будет выглядеть следующим образом:

from django.shortcuts import render

def post\_list(request):

return render(request, 'blog/post\_list.html', {})

Помнишь мы говорили о включении кода из других файлов? Теперь нам нужно включить модель, которую мы определили в файле models.py. Мы добавим строку from .models import Post следующим образом:

from django.shortcuts import render

from .models import Post

Точка перед from означает текущую директорию или текущее приложение. Поскольку views.py и models.py находятся в одной директории, мы можем использовать точку . и имя файла (без расширения .py). Затем мы импортируем модель (Post).

Но что дальше? Чтобы получить реальную запись из модели Post, нам нужен QuerySet.

QuerySet

Ты уже должна быть знакома с принципом работы QuerySets. Мы говорили о нем в главе [Django ORM (QuerySets)](https://ryangallen.gitbooks.io/django-girls/content/ru/django_orm/index.html).

Сейчас нас интересует список записей блога, отсортированных по published\_date, согласна? Мы уже делали это в главе о QuerySets!

Post.objects.filter(published\_date\_\_lte=timezone.now()).order\_by('published\_date')

Теперь нам нужно включить эту строку кода в файлblog/views.py, добавив её в функцию def post\_list(request):

from django.shortcuts import render

from django.utils import timezone

from .models import Post

def post\_list(request):

posts = Post.objects.filter(published\_date\_\_lte=timezone.now()).order\_by('published\_date')

return render(request, 'blog/post\_list.html', {})

Пожалуйста, обрати внимание, что мы создали переменную для QuerySet: posts. Можешь думать о неё как об имени для нашего QuerySet. Теперь мы можем обращаться к нему, используя имя.

Кроме того, код использует функцию timezone.now(), так что нужно также импортировать timezone.

Нам осталось передать QuerySet posts в шаблон (мы расскажем о том, как отобразить его на странице в следующей главе).

В функции render мы уже имеем параметр request (т.е. все что мы получим от пользователя в качестве запроса через Интернет) и файл шаблона 'blog/post\_list.html'. Последний параметр, который выглядит как {}, это место, куда мы можем добавить что-нибудь для использования в шаблоне. Мы должны задавать имена, передаваемым шаблону вещам (мы будем придерживаться 'posts' прямо сейчас :)). В итоге параметр будет выглядеть следующим образом: {'posts': posts}. Обрати внимание, что часть перед : является строкой; её нужно обернуть в кавычки ''.

В результате файл blog/views.py должен выглядеть следующим образом:

from django.shortcuts import render

from django.utils import timezone

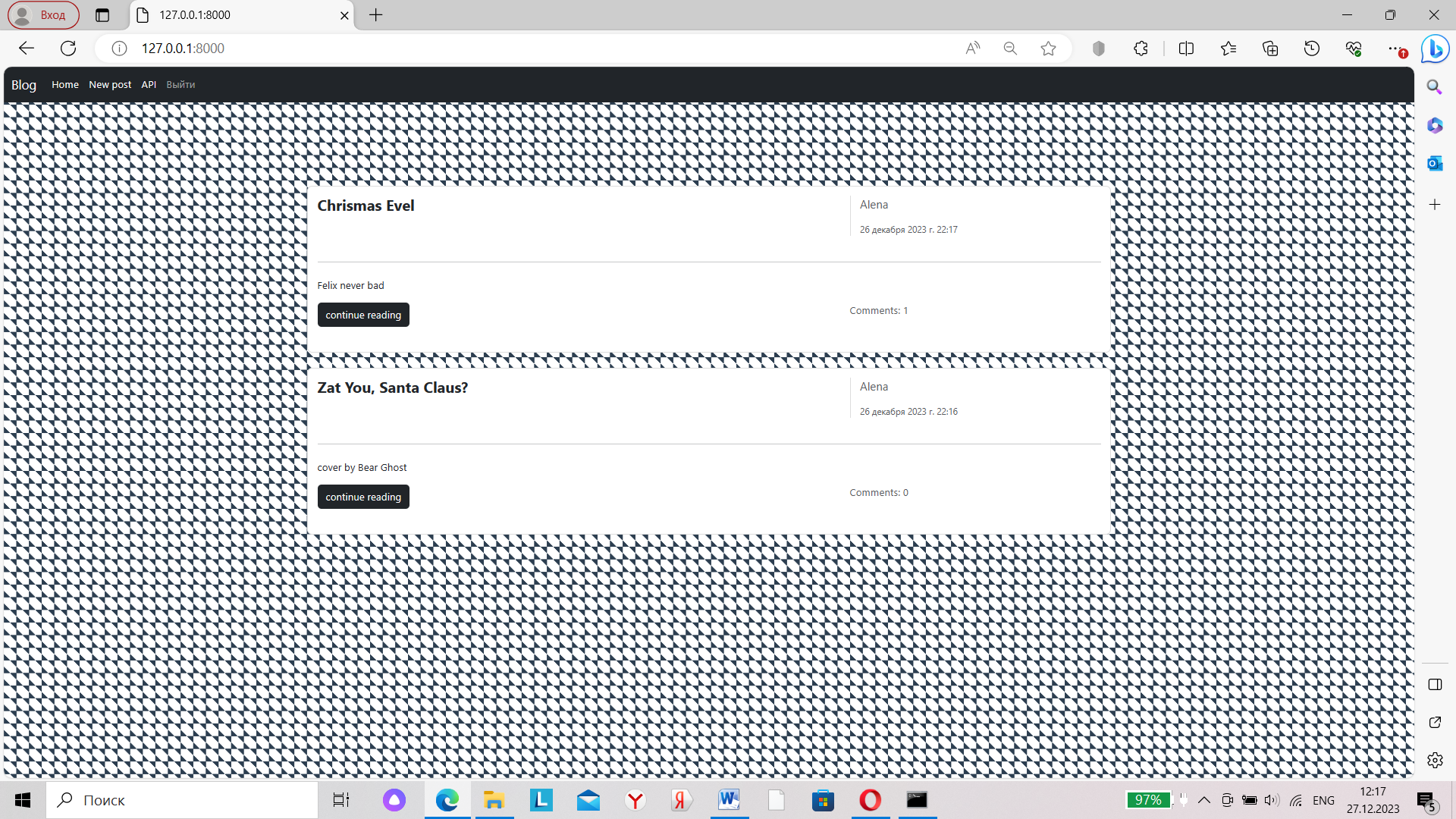
from .models import Post

def post\_list(request):

posts = Post.objects.filter(published\_date\_\_lte=timezone.now()).order\_by('published\_date')

return render(request, 'blog/post\_list.html', {'posts': posts})

Проект



Листинг

