Мы создадим виртуальное окружение под именем myvenv. В общем случаем команда будет выглядеть так:

python -m venv myvenv

Запусти виртуальное окружение, выполнив:

myvenv\Scripts\activate

Cоздание проекта

django-admin.exe startproject mysite .

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

TIME\_ZONE = 'Europe/Kiev'

Код языка состоит из сокращённого названия языка, например en для английского или ru для русского, и кода страны, например, ru для России или 'ch' для Швейцарии. Тебе понадобится эта настройка, если ты хочешь, чтобы все встроенные кнопки и уведомления от Django были на твоём языке. Таким образом, надпись на кропке «Cancel» будет переведена на заданный тобой язык

LANGUAGE\_CODE = 'ru-ru'

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

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

Когда наcтройка DEBUG имеет значение True, а настройка ALLOWED\_HOSTS пуста, имя хост твоего веб-сайта сверяется со списком ['localhost', '127.0.0.1', '[::1]']. Ни одно из значений не соответствует имени хоста на PythonAnywhere при публикации нашего приложения, поэтому нам необходимо изменить следующую настройку:

ALLOWED\_HOSTS = ['127.0.0.1', '<твоё\_имя\_пользователя>.pythonanywhere.com']

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).

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

Create app

python manage.py startapp blog

Появились доп папки

После того, как приложение создано, нам нужно сообщить Django, что теперь он должен его использовать. Мы сделаем это с помощью файла 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',

]

**Создаём 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':

blog/migrations/0001\_initial.py:

- Create model Post

**Примечание:** не забудь сохранить отредактированные файлы. Иначе твой компьютер выполнит команду с их предыдущей версией, и могут появиться неожиданные ошибки.

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 что создали модель, мы используем панель управления администратора Django.

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

После чего нужно опубликовать свой проект

Для начала неплохо было бы закинуть его на гит хаб

После на PythonAnywhere

## Regex

Ты спрашиваешь себя, как Django сопоставляет URL-адреса и представления? Ну, это не так просто. В Django используются так называемые регулярные выражения (англ. «regular expressions» — сокращённо regex). Регулярные выражения имеют множество (множество!) правил, которые формируют поисковый шаблон. Поскольку регулярные выражения являются довольно сложной темой, мы не будем рассматривать их подробно.

Если ты все еще хочешь разобраться в написании шаблонов, приведём пример — нам потребуется лишь ограниченное число правил для описания шаблона, а именно:

* ^ — начало текста;
* $ — конец текста;
* \d — цифра;
* + — чтобы указать, что предыдущий элемент должен быть повторен как минимум один раз;
* () — для получения части шаблона.

Всё остальное в шаблоне URL будет интерпретироваться буквально.

Файл mysite/urls.py должен выглядеть следующим образом:

from django.conf.urls import include, url

from django.contrib import admin

urlpatterns = [

url(r'^admin/', admin.site.urls),

url(r'', include('blog.urls')),

]

Создай новый пустой файл blog/urls.py. Отлично! Добавь в него следующие две строки:

from django.conf.urls import url

from . import views

urlpatterns = [

url(r'^$', views.post\_list, name='post\_list'),

]

## blog/views.py

Хорошо, давай откроем этот файл и посмотрим на его содержимое

def post\_list(request):

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

blog

└───templates

└───blog

Создание поста через консоль

from blog.models import Post

from django.contrib.auth.models import User

Post.objects.all()

User.objects.all()

me = User.objects.get(username='<User>')

Post.objects.create(author=me, title='Sample title', text='Test')

А может быть мы хотим получить все записи со словом 'title' в поле title?

Post.objects.filter(title\_\_contains='title')

Ты также можешь получить список всех опубликованных записей. Мы просто отфильтруем записи по полю published\_date:

from django.utils import timezone

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

К сожалению, пост, который мы добавили в консоли Python, ещё не опубликован. Мы можем изменить это! Сначала выберем запись, которую мы хотим опубликовать:

post = Post.objects.get(title="Sample title")

post.publish()

QuerySets позволяет сортировать объекты. Давай попробуем сортировку по полю created\_date

Post.objects.order\_by('created\_date')

Мы также можем изменить порядок на противоположный, добавив - в начало условия:

Post.objects.order\_by('-created\_date')

Что бы вывести посты на страницу, переходим в файл blog/views. Импортируем класс Post Из файла models

*from* .models *import* Post

В уже существующую функцию, нужно добавить строку

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

Выглядит это вот ак

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', {})

Нам осталось передать QuerySet posts в шаблон

**Post detail**

В url добавляем строку

url(r'^post/(?P<pk>\d+)/$', views.post\_detail, name='post\_detail')

в ссылку вставляем

<a *href=*"{% url 'post\_detail' *pk*=*post*.*pk* %}">

Во views

*def* post\_detail(request, pk):  
 post = get\_object\_or\_404(Post, pk=pk)  
 *return* render(request, 'blog/post\_detail.html', {'post' : post})

Ну и понятное дело, создать шаблон постов

Создание поста

Создать файл forms.py

*from* django *import* forms  
*from* .models *import* Post  
  
*class* PostForm(forms.ModelForm):  
  
 *class* Meta:  
 model = Post  
 fields = ('title', 'text')

in urls

url(r'^post/new/$', views.post\_new, name='post\_new')

шаблон для создания файла

{% extends 'blog/base.html' %}  
  
{% block *content* %}  
 <h1>New post</h1>  
 <form *method=*"POST" *class=*"post-form">{% csrf\_token %}  
 {{ *form*.*as\_p* }}  
 <button *type=*"submit" *class=*"save btn btn-default">Save</button>  
 </form>  
{% endblock %}

Во views

def post\_new(request):

form = PostForm()

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

Для редактирования записей

Необходимо добавить ссылку на страницу редактирования,

<a class="btn btn-default" href="{% url 'post\_edit' pk=post.pk %}"><span class="glyphicon glyphicon-pencil"></span></a>

Также добавить url

url(r'^post/(?P<pk>\d+)/edit/$', views.post\_edit, name='post\_edit'),

и добавить функцию в файл views

def post\_edit(request, pk):

post = get\_object\_or\_404(Post, pk=pk)

if request.method == "POST":

form = PostForm(request.POST, instance=post)

if form.is\_valid():

post = form.save(commit=False)

post.author = request.user

post.published\_date = timezone.now()

post.save()

return redirect('post\_detail', pk=post.pk)

else:

form = PostForm(instance=post)

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

что бы редактировать и создавать записи могли только авторизованые пользователи необхадимо ограничить показ определенных частей кода

{% if *user*.*is\_authenticated* %}

{% endif %}