Компьютерные технологии

Лекция № 7.

Web-программирование: Django

Часть 2.

Нижний
Новгород
2023 г.

URL-адреса Django

Любая страница в Интернете нуждается в собственном URL-адресе. Таким образом приложение точно знает, что показать пользователю, который открывает конкретный URL-адрес. В Django используется URLconf. Это набор шаблонов, которые Django попробует сравнить с полученным URL, чтобы выбрать правильный метод для отображения (view).

В файл mysite/urls.py была добавлена строчка

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

Таким образом, любому URL-адресу, начинающемуся с admin/, Django будет находить соответствующее *view* (представление). В этом случае охватывается большое количество различных URL-адресов, которые явно не прописаны в этом файле — так он становится более аккуратным и удобочитаемым.

Regex

В Django используются регулярные выражения. Регулярные выражения имеют множество правил, которые формируют поисковый шаблон. Например:

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

Есть адрес http://www.mysite.com/post/12345/, где 12345 — номер записи в блоге.

С помощью регулярных выражений можно создать шаблон, соответствующий url, который позволит извлекать из адреса номер: ***^post/(\d+)/$***

Создаем URL-адрес

Необходимо, чтобы 'http://127.0.0.1:8000/' возвращал домашнюю страничку блога со списком записей в нём.

Файл ***mysite/urls.py*** (на локальном компьютере) 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')),
]
```

Django теперь будет перенаправлять все запросы 'http://127.0.0.1:8000/' к ***blog.urls*** и искать там дальнейшие инструкции.

blog.urls

Создаем новый пустой файл ***blog/urls.py*** и добавляем в него следующие строки:

from django.conf.urls import url

from . import views

urlpatterns = [url(r'^$', views.post_list, name='post_list'),]

Импортируем функцию url Django и все views (представления) из приложения blog и связываем view под именем post_list с URL-адресом ^$.

Последняя часть name='post_list' — это имя URL, которое будет использовано, чтобы идентифицировать его. Оно может быть таким же, как имя представления (view).

Представления в Django

View, или *представление*, — это то место, где находиться «логика» работы приложения. Оно запросит информацию из модели, которую мы создали ранее, и передаст её в шаблон. Представления похожи на методы в Python.

Откроем файл **blog/views.py** и добавим представление:

```
from django.shortcuts import render
def post_list(request):
    return render(request, 'blog/post_list.html', {})
```

Функция (def) с именем post_list, которая принимает request в качестве аргумента и возвращает (return) результат работы функции render, которая соберёт шаблон страницы blog/post_list.html.

Введение в HTML

Шаблон — это файл, который можно использовать повторно для отображения различной информации в заданном формате; например, использовать шаблон, чтобы упростить написание письма, поскольку письма хоть и различаются по содержанию и получателю, но сохраняют общую структуру.

HTML — это простой код, который может быть интерпретирован браузером чтобы отобразить веб-страницу пользователю.

HTML — язык гипертекстовой разметки. **Гипертекст** — это тип текста, поддерживающий гиперссылки между страницами. Под **разметкой** понимается введение в текст документа кода, который будет говорить браузеру (в нашем случае), как интерпретировать веб-страницу. HTML код строится при помощи **тегов**, каждый из которых должен начинаться с < и заканчиваться >. Эти теги представляют **элементы** разметки.

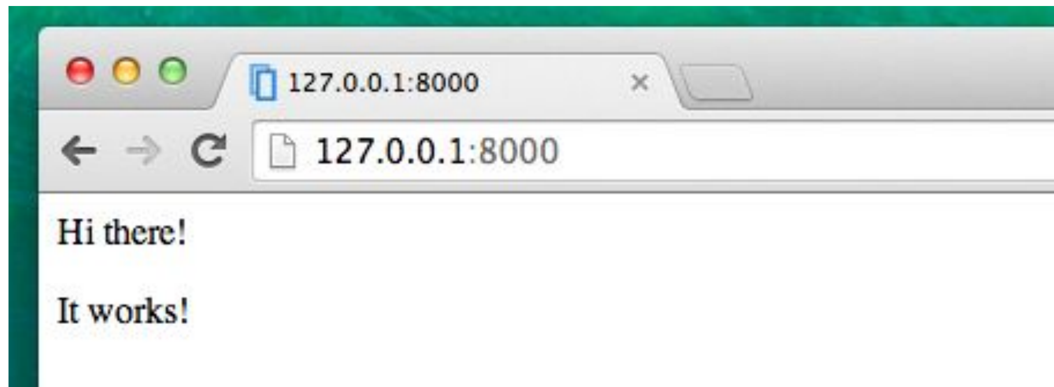
Создание шаблона

Шаблоны сохраняются в директории ***blog/templates/blog***. Создаем директорию ***templates*** внутри папки ***blog***, далее другую директорию ***blog*** внутри папки ***templates.***

Создаем файл ***post_list.html*** внутри директории ***blog/templates/blog*** и добавляем следующий код:

```
<html>
<p>Hi there!</p>
<p>It works!</p>
</html>
```

Переходим на http://127.0.0.1:8000/



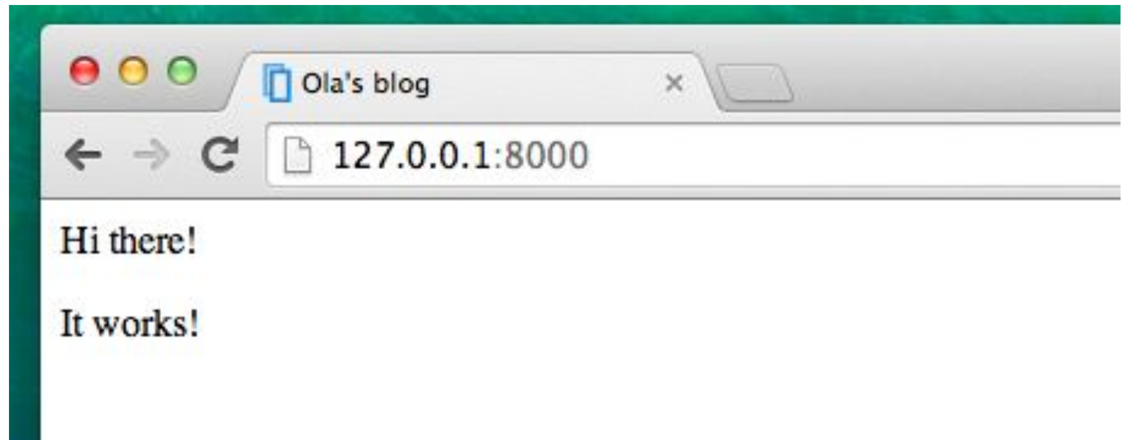
Head и body

- Наиболее базовой тег, <html>, всегда присутствует в начале веб-страницы, а </html> — в конце.
- <p> — это тег для параграфов; </p>, соответственно, закрывает каждый параграф.

Каждая HTML-страница также делится на два элемента: **head** и **body**.

- **head** — это элемент, содержащий информацию о документе, которая не отображается на экране.
- **body** — это элемент, который содержит всё, что будет отражено на веб-странице. Например:

```
<html>
  <head>
    <title>Ola's blog</title>
  </head>
  <body>
    <p>Hi there!</p>
    <p>It works!</p>
  </body>
</html>
```



Настраиваем шаблон

Примеры тегов:

- <h1>Заголовок</h1> — главный заголовок страницы;
- <h2>Подзаголовок</h2> — для заголовков второго уровня;
- <h3>Заголовок третьего уровня</h3> … и так далее, вплоть до <h6>;
- <p>Параграф</p>
- текст подчёркивает твой текст;
- текст — жирный шрифт;
-
 — переход на следующую строку (внутрь br тега нельзя ничего поместить);
- link создаёт ссылку;
- первый элемент
второй элемент создаёт список, такой же как этот!
- <div></div> определяет раздел страницы.

Настраиваем шаблон

Пример готового шаблона, **копируем** его содержимое в файл ***blog/templates/blog/post_list.html***

```
<html>
    <head>
        <title>Django blog</title>
    </head>
    <body>
        <div>
            <h1><a href="/">Django Blog</a></h1>
        </div>
        <div>
            <p>published: 14.06.2014, 12:14</p>
            <h2><a href="">Первый пост</a></h2>
            <p>Текст первого поста</p>
        </div>
        <div>
            <p>published: 14.06.2014, 12:14</p>
            <h2><a href="">Второй пост</a></h2>
            <p> Текст второго поста.</p>
        </div>
    </body>
</html>
```

QuerySet

Это список объектов заданной модели. QuerySet позволяет читать данные из базы данных, фильтровать и изменять их порядок.

Интерактивная консоль Django

Выведем на экран все записи в блоге. В своём локальном терминале набираем команду (myvenv) ***(myvenv) ~/djangopracticum$ python manage.py shell***

from blog.models import Post

Post.objects.all()

```
(myvenv) C:\Users\Olga\djangopracticum>python manage.py shell
Python 3.5.1 |Anaconda 2.5.0 (32-bit)| (default, Jan 29 2016, 15:46:01) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> Post.objects.all()
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'Post' is not defined
>>> from blog.models import Post
>>> Post.objects.all()
<QuerySet [<Post: Глубинное обучение с подкреплением пока не работает>, <Post: Многоэтапные (multi-stage builds)
оматизация удаления забытых транзакций>, <Post: Падение Stack Overflow: что случилось>]>
>>>
```

Создаём объект

Создать объект Post в базе данных можно следующим образом:

1. Импортируем модель User:

>>> from django.contrib.auth.models import User

2. Проверим какие пользователи есть в базе данных?

>>> User.objects.all()

3. Создадим его экземпляр:

me = User.objects.get(username='ola')

4. Создадим пост:

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

```
>>> from django.contrib.auth.models import User
>>> User.objects.all()
<QuerySet [<User: admin>]>
>>> me = User.objects.get(username='admin')
>>> Post.objects.create(author=me, title='Sample title', text='Test')
<Post: Sample title>
```

Фильтрация объектов

Важной особенностью QuerySets является возможность фильтровать объекты. Например:

>>> Post.objects.filter(author=me) (все посты автора me)

>>> Post.objects.filter(title__contains='title') (все записи со словом 'title' в поле title)

Список всех опубликованных записей:

>>> from django.utils import timezone

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

>>> post.publish()

>>> Post.objects.filter(published_date__lte=timezone.now()) []

```
>>> Post.objects.filter(author=me)
<QuerySet [<Post: Глубинное обучение с подкреплением пока не работает>,
оматизация удаления забытых транзакций>, <Post: Падение Stack Overflow:
>>> Post.objects.filter(title__contains='title')
<QuerySet [<Post: Sample title>]>
>>> from django.utils import timezone
>>> Post.objects.filter(published_date__lte=timezone.now())
<QuerySet []>
>>> post = Post.objects.get(title="Sample title")
>>> post.publish()
>>> Post.objects.filter(published_date__lte=timezone.now())
<QuerySet [<Post: Sample title>]>
```

Сортировка объектов

QuerySets позволяет сортировать объекты.

Сортируем по полю created_date:

>>> Post.objects.order_by('created_date')

Можем изменить порядок на противоположный, добавив - в начало условия:

>>> Post.objects.order_by('-created_date')

QuerySets можно **сцеплять**, создавая цепочки:

>>> Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')

```
>>> Post.objects.order_by('created_date')
<QuerySet [<Post: Падение Stack Overflow: что случилось>, <Post: Многоэтапные (multi-st
е с подкреплением пока не работает>, <Post: Sample title>, <Post: Автоматизация удалени
>>> Post.objects.order_by('-created_date')
<QuerySet [<Post: Автоматизация удаления забытых транзакций>, <Post: Sample title>, <Po
т>, <Post: Многоэтапные (multi-stage builds) сборки в Docker>, <Post: Падение Stack Ove
>>> Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
<QuerySet [<Post: Sample title>]>
>>> exit()
```

Динамически изменяющиеся данные в шаблонах

Необходимо отобразить записи в шаблоне HTML-страницы. Для этого нужны *представления*: соединять между собой модели и шаблоны. В post_list *представлению* нужно будет взять модели, которые необходимо отобразить, и передать их шаблону. В *представлениях* определяем, что будет отображена в шаблоне. В файле blog/views.py:

```
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})
```

Шаблоны Django

В HTML нельзя помещать код Python. **Теги шаблонов Django** позволяют нам вставлять Python в HTML, так что можно создавать динамические веб-сайты быстрее и проще.

Отображаем шаблон списка записей

Чтобы вставить переменную в шаблон Django, нам нужно использовать двойные фигурные скобки с именем переменной внутри: в файле ***blog/templates/blog/post_list.html*** заменим всё, начиная со второго <div> и вплоть до третьего </div> кодом:

{{ posts }}

Сохраняем файл и обновляем страницу, чтобы увидеть результат .

Шаблоны Django

Для того, чтобы отобразить список постов:

```
{% for post in posts %}
    {{ post }}
{% endfor %}
```

В ***blog/templates/blog/post_list.html*** элемент body будет выглядеть следующим образом:

```
<div>
    <h1><a href="/">Django Girls Blog</a></h1>
</div>
{% for post in posts %}
    <div>
        <p>published: {{ post.published_date }}</p>
        <h1><a href="">{{ post.title }}</a></h1>
        <p>{{ post.text|linebreaksbr }}</p>
    </div>
{% endfor %}
```

CSS. Каскадные таблицы стилей

Это специальный язык, используемый для описания внешнего вида и форматирования сайта, написанного на языке разметки (как HTML).

Bootstrap — один из наиболее популярных HTML и CSS фреймворков для разработки красивых 😊 сайтов: https://getbootstrap.com/

Установка Bootstrap

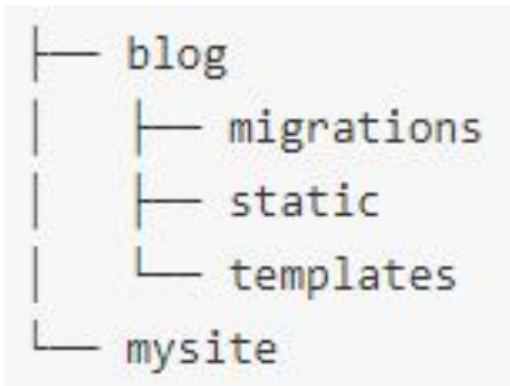
Для установки Bootstrap нужно добавить следующие строки в <head> .html файла (blog/templates/blog/post_list.html) и обновить страницу:

```
<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">
```

Статические файлы в Django

Статическими файлами называются все файлы CSS и изображения, т.е. файлы, которые не изменяются динамически, их содержание не зависит от контекста запроса и будет одинаково для всех пользователей.

Нам нужно добавить статические файлы для своего приложения blog. Мы сделаем это, создав папку ***static*** внутри каталога с нашим приложением:



Django будет автоматически находить папки ***static*** внутри всех каталогов твоих приложений и сможет использовать их содержимое в качестве статических файлов.

Добавление CSS

Создаем новую папку под названием ***css*** внутри папки ***static***, затем файл ***blog.css*** внутри папки ***css.*** Изменим цвет заголовка:

h1 a {

color: #FCA205;

}

h1 a — это CSS-селектор. В CSS файле определяем стили для элементов файла HTML. Элементы идентифицируются именами (то есть a, h1, body), атрибутом class или атрибутом id. Class и id – это имена, которые присваиваются элементам. Классы (class) определяют группы элементов, а идентификаторы (id) указывают на конкретные элементы. Например, следующий тег может быть идентифицирован CSS с использованием имени тега a, класса external_link или идентификатора link_to_wiki_page:

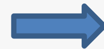
<a href="https://en.wikipedia.org/wiki/Django" class="external_link" id="link_to_wiki_page">

Добавление CSS

1. Скопируем и добавим код в файл djangopracticum/static/css/blog.css из blog.css

2. Далее переделаем код HTML, отображающий посты, используя классы. ***Заменяем*** в blog/templates/blog/***post_list.html*** (код в комментариях к слайду):

```
{% for post in posts %}
    <div class="post">
        <p>published: {{ post.published_date }}</p>
        <h1><a href="">{{ post.title }}</a></h1>
        <p>{{ post.text|linebreaksbr }}</p>
    </div>
{% endfor %}
```



```
<div class="content container">
    <div class="row">
        <div class="col-md-8">
            {% for post in posts %}
                <div class="post">
                    <div class="date">
                        <p>Опубликовано: {{ post.published_date }}</p>
                    </div>
                    <h1><a href="">{{ post.title }}</a></h1>
                    <p>{{ post.text|linebreaksbr }}</p>
                </div>
            {% endfor %}
        </div>
    </div>
</div>
```

Расширение шаблона

Базовый шаблон — это наиболее общая типовая форма страницы, которую можно расширить для отдельных случаев.

Создаем файл ***base.html*** в директории ***blog/templates/blog/*** и скопируем всё из post_list.html в base.html. Затем в файле ***base.html*** заменим всё между тегами ***<body>*** и ***</body>*** следующим кодом:

```
<body>
    <div class="page-header">
        <h1><a href="/">Django Girls Blog</a></h1>
    </div>
    <div class="content container">
        <div class="row">
            <div class="col-md-8">
            {% block content %}
            {% endblock %}
            </div>
        </div>
    </div>
</body>
```

Расширение шаблона

Далее открываем ***blog/templates/blog/post_list.html*** снова. И модифицируем его так, чтобы он выглядел следующим образом:

```
{% extends 'blog/base.html' %}
{% block content %}
    {% for post in posts %}
        <div class="post">
            <div class="date">
                {{ post.published_date }}
            </div>
            <h1><a href="">{{ post.title }}</a></h1>
            <p>{{ post.text|linebreaksbr }}</p>
        </div>
    {% endfor %}
{% endblock %}
```

Расширяем приложение

Необходимо создать страницу для отображения конкретной записи (добавим дополнительный код в файл models.py).

Добавим ссылку внутрь файла ***blog/templates/blog/post_list.html***

Заменим: ***<h1>{{ post.title }}</h1>***

<h1>{{ post.title }}</h1>

Создадим URL для страницы поста

Создадим URL в файле **blog/urls.py** и укажем Django на представление под названием post_detail, которое будет отображать пост целиком. Добавим строчку **url(r'^post/(?P<pk>\d+)/$', views.post_detail, name='post_**

```
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^$', views.post_list, name='post_list'),
    url(r'^post/(?P<pk>\d+)/$', views.post_detail, name='post_detail'),
]
```

Представление для страницы поста

В файле ***blog/urls.py*** был создан шаблон URL под названием ***post_detail***, который ссылался на представление под названием views.post_detail. Это значит, что Django ожидает найти функцию-представление с названием ***post_detail*** в ***blog/views.py***.

В файл ***blog/views.py*** рядом с другими строками, начинающимися с from добавляем:

```
from django.shortcuts import render, get_object_or_404
```

В конец же файла добавляем новое *представление*:

```
def post_detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    return render(request, 'blog/post_detail.html', {'post': post})
```

Создадим шаблон для страницы поста

Создадим файл ***post_detail.html*** в директории ***blog/templates/blog***

```
{% extends 'blog/base.html' %}
{% block content %}
    <div class="post">
        {% if post.published_date %}
            <div class="date">
                {{ post.published_date }}
            </div>
        {% endif %}
        <h1>{{ post.title }}</h1>
        <p>{{ post.text|linebreaksbr }}</p>
    </div>
{% endblock %}
```

Развёртывание!

Добавим изменения на PythonAnywhere.

На локальном хосте:

$ git status

$ git add --all .

$ git status

$ git commit -m "Added CSS for the site"

$ git push

В консоли на PythonAnywhere:

$ cd my-first-blog

$ git pull

И нажимаем **Reload** на вкладке Web.