Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Ордена Трудового Красного Знамени
федеральное государственное бюджетное
образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

Кафедра МКиИТ

Проектирование клиент-серверных приложений

Лабораторная работа №4
“Создание страницы определённой записи”

Выполнил:
студент 3 курса,
группы БФИ2001
Лушин Е. А.

Москва 2023

Цель работы: научиться создавать страницу со статьями с возможностью перейти на каждую из статей кликнув на название нужной нам статьи.

Задание:

Создание страницы определённой записи

- Теперь, когда есть страница определенной записи, можно на странице списка постов каждый заголовок сделать ссылкой. Сделайте так, чтобы при клике по названию происходил переход на страницу указанной записи. Для этого достаточно воспользоваться тегом **<a>**, которому в атрибуте **href** указать адрес перехода.

Вёрстка обеих страниц в соответствии с макетом

- Создайте стили, которые соответствуют макету, для страницы определенной записи. Не забудьте добавить и стилизовать ссылку "**Все записи**".

Краткая теория

Модели в Django

Обычно представления создаются в файле views.py, однако это не задано жесткими рамками, название может быть абсолютно другим, однако принято называть его именно views.py, чтобы другие разработчики, читающие ваш код, сразу понимали, что в нем находится.

В качестве начального примера будет рассмотрено представление «Hello world».

Ниже приведен код функции вместе с командами импорта, который нужно поместить в файл views.py.

Простейшая функция-представление hello:

```
from django.http import HttpResponse
def hello(request):
    return HttpResponse("Hello world")
```

- Сначала импортируется класс HttpResponse, который находится в модуле django.http. Импортировать его необходимо, потому что он используется в коде функции ниже.
- Далее определяется функция представления hello.
- Любая функция представления принимает по меньшей мере один параметр, который принято называть request. Это объект, содержащий информацию о текущем веб-запросе, в ответ на который была вызвана функция; он является экземпляром класса django.http.HttpRequest. В данном примере мы не используем параметр request, тем не менее он должен быть первым параметром представления.
- Отметим, что имя функции представления не имеет значения, фреймворк Django не предъявляет каких-либо специфических требований к именам. Мы назвали ее hello просто потому, что это имя ясно показывает назначение представления, но могли бы назвать hello wonderful beautiful world или еще как-то. В следующем разделе будет показано, каким образом Django находит эту функцию.
- Сама функция состоит всего из одной строки: она просто возвращает объект HttpResponse, инициализированный строкой «Hello world».

Главный урок состоит в том, что представление — обычная функция на языке Python, которая принимает экземпляр класса HttpRequest в качестве первого параметра и возвращает экземпляр класса HttpResponse. Чтобы функция на Python могла считаться функцией представления, она должна обладать этими двумя свойствами.

Если сейчас для нового проекта выполнить команду, запускающую сервер Django:

python manage.py runserver

то появится сообщение «Welcome to Django» без каких бы то ни было следов представления «Hello world». Объясняется это тем, что проект, который вы только что создали, еще ничего не знает о представлении hello; необходимо явно сообщить Django, что при обращении к некоторому URL должно

активироваться это представление. (Если продолжить аналогию с публикацией статических HTMLстраниц, то сейчас файл был только создан, но еще не загружен в каталог на сервере.) Чтобы связать функцию представления с URL, в Django используется механизм конфигурации URL.

Можно сказать, что конфигурация URL — это оглавление веб-сайта, созданного с помощью Django. По сути дела, речь идет об установлении соответствия между URL и функцией представления, которая должна вызываться при обращении к этому URL. Django получает указания: «Для этого адреса URL следует вызвать эту функцию, а для этого — эту». Например, «При обращении к URL /foo/ следует вызвать функцию представления foo view(), которая находится в Python-модуле views.py».

Во время выполнения команды, создающей новый Django проект:

django-admin.py startproject project name here

сценарий автоматически создал конфигурацию URL: файл urls.py. По умолчанию она выглядит следующим образом (файл конфигурации URL, созданный автоматически):

```
from django.conf.urls.defaults import *
# Раскомментировать следующие две строки для активации
# административного интерфейса:
# from django.contrib import admin
# admin.autodiscover()
urlpatterns = patterns('',
# Пример:
# (r'^mysite/', include('mysite.foo.urls')),
# Раскомментировать строку admin/doc ниже и добавить
# 'django.contrib.admindocs' в INSTALLED APPS для активации
# документации по административному интерфейсу:
# (r'^admin/doc/', include('django.contrib.admindocs.urls')),
# Раскомментировать следующую строку для активации
# административного интерфейса:
```

 # (r'^admin/', include(admin.site.urls)),

)

В этой конфигурации URL по умолчанию некоторые часто используемые функции Django закомментированы, для их активации достаточно раскомментировать соответствующие строки. Если не обращать внимания на закомментированный код, то конфигурация URL сведется к следующему коду. Пример файла конфигурации URL без комментариев:

*from django.conf.urls.defaults import **

urlpatterns = patterns('',

)

- В первой строке импортируются все объекты из модуля django.conf.urls.defaults поддержки механизма конфигурации URL. В частности, импортируется функция patterns.
- Во второй строке производится вызов функции patterns, а возвращенный ею результат сохраняется в переменной urlpatterns. Функции patterns передается всего один аргумент — пустая строка. (С ее помощью можно задать общий префикс для функций представления.)

Главное здесь — переменная urlpatterns, которую Django ожидает найти в конфигурации URL. Она определяет соответствие между URL-адресами и обрабатывающим их кодом. По умолчанию конфигурация URL пуста, то есть приложение Django — чистая доска.

Чтобы добавить URL и представление в конфигурацию URL, достаточно включить кортеж, отображающий шаблон URL-адреса на функцию представления. Вот как подключается представление hello (пример добавления нового URL):

*from django.conf.urls.defaults import **

from mysite.views import hello

urlpatterns = patterns('',

('^hello/$', hello),

)

Было внесено два изменения:

1. Во-первых, импортировали функцию представления hello из модуля, где она находится, — mysite/views.py, полное имя которого, согласно синтаксису импорта, принятому в Python, транслируется в mysite.views. (Здесь пред- полагается, что mysite/views.py включен в путь, где интерпретатор Python пытается искать файлы.)
2. Далее в список шаблонов urlpatterns мы добавили строку ('^hello/$', hello). Такая строка называется шаблоном URL. Это кортеж Python, в котором первый элемент — строка с шаблоном (регулярное выражение), а второй — функция представления, соответствующая этому шаблону.

Тем самым мы сказали Django, что любой запрос к URL /hello/ должен обрабатываться функцией представления hello.

Имеет смысл подробнее остановиться на синтаксисе определения шаблона URL, так как он может показаться неочевидным. Вам требуется обеспечить совпадение с URL /hello/, а шаблон выглядит несколько иначе. И вот почему:

- Django удаляет символ слеша в начале любого поступающего URL и только потом приступает к сопоставлению с шаблонами URL. Поэтому начальный символ слеша не включен в образец. (На первый взгляд, это требование противоречит здравому смыслу, зато позволяет многое упростить, например включение одних шабонов URL в другие.)
- Шаблон включает знаки вставки (^) и доллара ($). В регулярных выражениях эти символы имеют специальное значение: знак вставки означает, что совпадение с шаблоном должно начинаться в начале строки, а знак доллара — что совпадение с шаблоном должно заканчиваться в конце строки.
- Этот синтаксис проще объяснить на примере. Если бы мы задали шаблон '^hello/' (без знака доллара в конце), то ему соответствовал бы любой URL, начинающийся с /hello/ (например /hello/foo и /hello/bar, а не только/hello/). Аналогично, если бы мы опустили знак вставки в начале

(например, 'hello/$'), то ему соответствовал бы любой URL, заканчивающийся строкой hello/, например /foo/bar/hello/. Если написать просто hello/ без знаков вставки и доллара, то подойдет вообще любой URL, содержащий строку hello/, например /foo/hello/bar). Поэтому мы включаем оба знака — вставки и доллара, чтобы образцу соответствовал только URL /hello/ и ничего больше.

- Как правило, шаблоны URL начинаются знаком вставки и заканчиваются знаком доллара, но полезно все же иметь дополнительную гибкость на случай, если потребуется более хитрое сопоставление.
- Но что произойдет, если пользователь обратится к URL /hello (без завершающего символа слеша)? Так как в образце URL завершающий символ слеша присутствует, то такой URL с ним не совпадет. Однако по умолчанию запрос к любому URL, который не соответствует ни одному шаблону URL и не заканчивается символом слеша, переадресуется на URL, отличающийся от исходного только добавленным в конце символом слеша. (Этот режим управляется параметром Django APPEND SLASH.)
- Если вы предпочитаете завершать все URL-адреса символом слеша (как большинство разработчиков Django), то просто включайте завершающий символ слеша в конец каждого шаблона URL и не изменяйте принятое по умолчанию значение True параметра APPEND SLASH. Если же вам боль- ше нравятся URL-адреса, не завершающиеся символом слеша, или если вы решаете этот вопрос для каждого URL в отдельности, то задайте для параметра APPEND SLASH значение False и разбирайтесь с символом слеша, как считаете нужным.

Теперь немного теории о том, как в целом работает вся система конфигурирования URL и представлений. Все начинается с файла параметров. При выполнении команды, запускающей сервер Django:

python manage.py runserver

сценарий ищет файл settings.py в том же каталоге, в котором находится файл manage.py. В этом файле хранятся всевозможные параметры настройки данного проекта Django, записанные заглавными буквами: TEMPLATE DIRS, DATABASE NAME и т. д. Самый важный параметр называется ROOT URLCONF. Он говорит Django, какой Python-модуль следует использовать в качестве конфигурации URL для данного веб-сайта.

Вспомните, что команда, создающая новый Django-проект:

django-admin.py startproject

создала два файла: settings.py и urls.py. В автоматически сгенерированном файле settings.py параметр ROOT URLCONF указывает на автоматически сгенерированный файл urls.py. Откройте settings.py и убедитесь сами; он должен выглядеть примерно так (пример переменной ROOT URLCONF в файле settings.py):

ROOT URLCONF = 'mysite.urls'

Это соответствует файлу mysite/urls.py. Когда поступает запрос к некоторому URL — например /hello/, — фреймворк Django загружает конфигурацию URL из файла, на который указывает параметр ROOT URLCONF. Далее он поочередно сравнивает каждый образец URL, представленный в конфигурации, с запрошенным URL, пока не найдет соответствие. Обнаружив соответствие, Django вызывает функцию представления, ассоциированную с найденным образцом, передавая ей в качестве первого параметра объект HttpRequest.

В примере первого представления выше вы видели, что такая функция должна возвращать объект HttpResponse. А Django сделает все остальное: превратит объект Python в веб-ответ с нужными HTTP-заголовками и телом (содержимым веб-страницы).

Вот перечень выполняемых шагов:

1. Поступает запрос к URL /hello/.
2. Django находит корневую конфигурацию URL, сверяясь с параметром ROOT URLCONF.

3. Django просматривает все образцы URL в конфигурации URL, пока не найдет первый, соответствующий URL /hello/.
4. Если соответствие найдено, вызывается ассоциированная с ним функция представления.
5. Функция представления возвращает объект HttpResponse.
6. Django преобразует HttpResponse в соответствующий HTTP-ответ, который визуализируется в виде веб-страницы.

Вот вы и познакомились с основами создания страниц с помощью Django. На самом деле все просто: нужно лишь написать функции представления и отобразить их на URL-адреса с помощью конфигурации URL.

Выполнение

Для выполнения этой лабораторной работы за основу был взят проект из предыдущей лабораторной работы.

Для того, чтобы можно было прочитать значение идентификатора (id) отдельного поста, укажим в файле urls.py в качестве адреса регулярное выражение.

Листинг 1. Содержимое файла urls.py

```
blog > blog > urls.py > ...
1   from articles import views
2   from django.contrib import admin
3   from django.urls import path
4
5   urlpatterns = [
6       path('admin/', admin.site.urls),
7       path('', views.archive, name= 'home'),
8       path('article/<int:article_id>', views.get_article, name= 'get_article')
9   ]
```

После того, как в регулярном выражении мы объявили именованную группу, в представление можно добавлять именованный аргумент, куда и будет передано значение из адреса, по которому перешел пользователь. Создадим в файле views.py из директории articles помимо уже написанной в предыдущей работе функции archieve ещё и функцию get_article для обработки исключений для статей с несуществующим id.

Листинг 2. Содержимое файла views.py

```
blog > articles > views.py > ...
1   from .models import Article
2   from django.shortcuts import render
3   from django.http import Http404
4
5   # Create your views here.
6
7   def archive(request):
8       return render(request, 'archive.html', {"posts": Article.objects.all()})
9
10  def get_article(request, article_id):
11      try:
12          post = Article.objects.get(id=article_id)
13          return render(request, 'article.html', {"post": post})
14      except Article.DoesNotExist:
15          raise Http404
```

Далее сделал каждый заголовок статьи ссылкой на саму страницу статьи. Для этого в тэг заголовка поместил тэг «a» с атрибутом «href» в котором будет указана ссылка на статью.

Листинг 3. Содержимое файла archive.html

```
blog > articles > templates > archive.html > ...
1   {% load static %}
2
3   <!DOCTYPE html>
4   <html>
5
6   <head>
7       <title>Архив всех статей</title>
8   </head>
9
10  <body>
11
12      <body>
13          <div class="header">
14              <img src="{% static 'img/dpb-logo.png' %}" alt="Логотип" type="image/png" />
15          </div>
16          <div class="archive">
17              {% for post in posts %}
18              <div class="one-post">
19                  <h2 class="post-title"><a href="http://127.0.0.1:8000/article/{{post.id}}">{{post.title}}</a></h2>
20                  <div class="article-info">
21                      <div class="article-author">{{ post.author.username }}</div>
22                      <div class="article-createddate">{{
23                          post.created_date }}</div>
24                  </div>
25              </div>
26              <p class="article-text">{{ post.get_excerpt }}</p>
27          </div>
28          {% endfor %}
29      </body>
30  </body>
31
32  </html>
```

Затем, сделал верстку двух страниц сайта в соответствии с макетами, представленными в методических указаниях к лабораторной работе, то есть

главной страницы, за которую отвечает файл «archive.html» и страницы с каждый отдельной статьей, за которую отвечает файл «article.html». За стили отвечает файл «article.css. На рисунках 1-2 представлены страницы сайта.

Листинг 4. Содержимое файла arcticle.html

```
blog > articles > templates > article.html > html
<!DOCTYPE html>
<html lang="en">

<head>
    <title>{{post.title}}</title>
    <link rel="stylesheet" href="{{STATIC_URL}}/static/article.css">
</head>

<body>
    <div class="archive">
        <div class="article-border">
            <p>
            <div class="article-header">
                <img class="article-image" src="{% static 'img/dpb-logo.png' %}" alt="Логотип" type="image/png" />
            </div>
            <a class="link" href="http://127.0.0.1:8000">Все статьи</a>
            </p>
        </div>
        <div class="post">
            <h1 class="post-title">{{post.title}}</h1>
            <div class="article-info">
                <h2>
                    <div class="article-author">{{post.author.username}}</div>
                </h2>
                <div class="article-createddate">{{post.created_date }}</div>
                <p class="article-text">{{post.text}}</p>
            </div>
        </div>
    </div>
</body>

</html>
```

Листинг 5. Содержимое файла arcticle.css

```
blog > articles > static > css > # article.css > ...
body {
    background: #1abc9c;
    font-family: Tahoma, Arial, sans-serif;
    color: #ffffff
}

img {
    display: block;
    width: 318px;
    margin-left: auto;
    margin-right: auto;
}

.archive {
    width: 960px;
    margin-left: auto;
    margin-right: auto;
}

```

```
post-title a {
    color: #ffffff;
}

.article-author {
    width: 50%;
    float: Left;
}

.article-createddate {
    text-align: right;
}

.article-image {
    display: block;
    width: 318px;
    margin-left: 0;
}

.link {
    color: white;
    font-weight: bold;
    position: absolute;
    right: 470px;
    top: 180px;
}

.article-border p {
    text-align: right;
}

.article-text {
    width: 960px;
    text-align: justify;
}

.article-created-data {
    text-align: right;
}
```

Рисунок 1 – Начальная страница

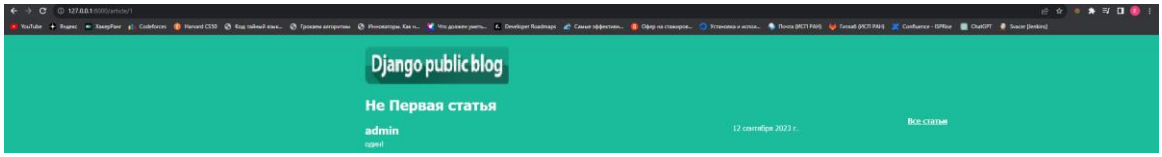

Рисунок 2 – Страница со статьёй

Вывод: в данной лабораторной работе я научился создавать страницу со статьями с возможностью перейти на каждую из статей кликнув на название нужной нам статьи.