

Создаём своё первое приложение с Django, часть 3

Продолжаем начатое во [второй части](#) учебника. Мы продолжим разрабатывать приложение для голосования и сосредоточимся на создании страниц сайта – «представлений».



Where to get help:

If you're having trouble going through this tutorial, please head over to the [Getting Help](#) section of the FAQ.

Обзор

Представление – это «тип» страниц вашего приложения, которое является функцией для обработки запроса и использует шаблон для генерации страницы. Например, блог может состоять из следующих представлений:

- Главная страница – показывает несколько последних записей блога.
- Страница записи – страница отображения одной записи блога.
- Страница-архив записей по годам – показывает все месяца года и записи блога, сгруппированные по этим месяцам.
- Страница-архив записей по месяцам – показывает все дни месяца и записи блога, сгруппированные по этим дням.
- Страница-архив записей по дням – показывает все записи за указанный день.
- Форма комментариев – предоставляет возможность добавить комментарий к записи блога.

В нашем приложении для голосования мы реализуем следующие представления:

- Главная страница вопросов – показывает несколько последних вопросов.
- Страница вопроса – показывает вопрос без результатов но с формой для ответа.
- Страница результата опроса – показывает результаты опроса.
- Обрабатывает процесс опроса – обрабатывает ответ на вопрос.

In Django, web pages and other content are delivered by views. Each view is represented by a Python function (or method, in the case of class-based views). Django will choose a view by examining the URL that's requested (to be precise, the part of the URL after the domain name).

В наше время в интернете можно наткнуться на такие прелести как «ME2/Sites/dirmod.asp?sid=&type=gen&mod=Core+Pages&gid=A6CD4967199A42D9B65B1B». Рады вам сообщить, что Django позволяет использовать более элегантные *URL шаблоны* чем этот.

A URL pattern is the general form of a URL - for example: `/newsarchive/<year>/<month>/`.

Чтобы из URL-а получить представление, Django используется так называемый „URLconf“. URLconf определяет соответствие URL-шаблонов и представлений.

В этом разделе мы опишем основы использования настроек URL-ов, больше информации вы можете найти в разделе [Менеджер URL-ов](#).

Добавь парочку представлений

Теперь создадим еще парочку представлений в `polls/views.py`. Эти представления немного отличаются, так как принимают аргументы:

```
polls/views.py

def detail(request, question_id):
    return HttpResponse("You're looking at question %s." % question_id)

def results(request, question_id):
    response = "You're looking at the results of question %s."
    return HttpResponse(response % question_id)

def vote(request, question_id):
    return HttpResponse("You're voting on question %s." % question_id)
```

Привяжем наше представление новостей в модуле `polls.urls` добавив вызов `path()`:

```
polls/urls.py

from django.urls import path

from . import views

urlpatterns = [
    # ex: /polls/
    path('', views.index, name='index'),
    # ex: /polls/5/
    path('<int:question_id>/', views.detail, name='detail'),
    # ex: /polls/5/results/
```

Оглавление

- Создаём своё первое приложение с Django, часть 3
 - Обзор
 - Добавь парочку представлений
 - Добавим функционал в наши представления
 - Функция `render()`
 - Вызов 404 исключения
 - Функция `get_object_or_404()`
 - Использование системы шаблонов
 - Избавляемся от «хардкода» URL-ов в шаблонах
 - Пространства имен в названиях URL-ов

Предыдущий раздел

Создаём своё первое приложение с Django, часть 2

Следующий раздел

Создаём своё первое приложение с Django, часть 4

Эта страница

- Исходный текст

Быстрый поиск

[Искать](#)

Последнее обновление:

нояб. 23, 2021

```
path('<int:question_id>/results/', views.results, name='results'),
# ex: /polls/5/vote/
path('<int:question_id>/vote/', views.vote, name='vote'),
]
```

Откройте страницу «/polls/34/». Будет выполнена функция detail() и показан ID, который вы указали в URL. Откройте «/polls/34/results/» и «/polls/34/vote/» – вы увидите наши будущие страницы результатов и голосования.

При запросе страницы – например, «/polls/34/», Django загружает модуль `mysite.urls` т.к. он указан в `ROOT_URLCONF`. Находит переменную `urlpatterns` и перебирает все параметры переменных по порядку. Когда находит 'polls/', отрезает найденный текст ("polls/") и отправляет оставшийся – "34/" – в `URLconf` из „polls.urls” для последующей обработки. Там находит соответствие с `<int:question_id>/`, в результате чего вызывается представление `detail()`:

```
detail(request=<HttpRequest object>, question_id=34)
```

Аргумент `question_id='34'` получен из `<int:question_id>`. Использование скобок позволяет распознать часть URL и передать в представление. `int:question_id` определяет название переменной при передаче, `<int:` определяет какой шаблон должен использоваться при распознавании части URL.

Нет необходимости добавлять в URL всякий хлам, вроде `.html`, пока это вам не понадобится. В таком случае добавьте что-то вроде:

```
path('polls/latest.html', views.index),
```

Но не делайте так. Это глупо.

Добавим функционал в наши представления

Каждое представление должно выполнить одно из двух действий: вернуть экземпляр `HttpResponse` с содержимым страницы, или вызвать исключения такое как `Http404`. Все остальное на ваше усмотрение.

Ваше представление может обращаться к базе данных или нет. Может использовать систему шаблонов Django – или любую другую – или не использовать. Может генерировать PDF файл, возвращать XML, создавать ZIP архив «на лету», все что угодно, используя любые библиотеки Python.

Все что нужно Django – это `HttpResponse`. Или исключение.

Мы будем использовать API Django для работы с базой данных, которое мы рассматривали в [Части 1](#). Изменим `index()` так, чтобы оно отображало последние 5 вопросов разделенные запятой от самого нового к самому старому:

```
polls/views.py
from django.http import HttpResponse

from .models import Question

def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    output = ', '.join([q.question_text for q in latest_question_list])
    return HttpResponse(output)

# Leave the rest of the views (detail, results, vote) unchanged
```

Есть небольшая проблема: внешний вид страницы определяется в представлении. Если вы захотите изменить дизайн страницы, вам придется менять код. Давайте воспользуемся системой шаблонов Django, чтобы отделить представление от кода.

Для начала создайте каталог `templates` в каталоге приложения `polls`. Django будет искать шаблоны в этом каталоге.

Настройка `TEMPLATES` указывает Django как загружать и выполнять шаблоны. По умолчанию используется бэкэнд `DjangoTemplates`, с опцией `APP_DIRS` равной `True`. В этом случае `DjangoTemplates` проверяет подкаталог «`templates`» в приложениях, указанных в `INSTALLED_APPS`.

Within the `templates` directory you have just created, create another directory called `polls`, and within that create a file called `index.html`. In other words, your template should be at `polls/templates/polls/index.html`. Because of how the `app_directories` template loader works as described above, you can refer to this template within Django as `polls/index.html`.



Пространства имен для шаблонов

Now we *might* be able to get away with putting our templates directly in `polls/templates` (rather than creating another `polls` subdirectory), but it would actually be a bad idea. Django will choose the first template it finds whose name matches, and if you had a template with the same name in a *different* application, Django would be unable to distinguish between them. We need to be able to point Django at the right one, and the best way to ensure this is by *namespacing* them. That is, by putting those templates inside *another* directory named for the application itself.

Добавьте следующий код в шаблон:

```
polls/templates/polls/index.html
```

```
{% if latest_question_list %}
<ul>
  {% for question in latest_question_list %}
    <li><a href="/polls/{{ question.id }}/"/>{{ question.question_text }}</a></li>
  {% endfor %}
</ul>
{% else %}
  <p>No polls are available.</p>
{% endif %}
```



Примечание

To make the tutorial shorter, all template examples use incomplete HTML. In your own projects you should use [complete HTML documents](#).

Теперь изменим наше представление `index` в `polls/views.py`, чтобы использовать шаблон:

```
polls/views.py

from django.http import HttpResponse
from django.template import loader

from .models import Question

def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    template = loader.get_template('polls/index.html')
    context = {
        'latest_question_list': latest_question_list,
    }
    return HttpResponse(template.render(context, request))
```

Этот код загружает шаблон `polls/index.html` и передает ему контекст. Контекст - это словарь, содержащий название переменных шаблона и соответствующие им значения.

Загрузите страницу в браузере по адресу `/polls/`, вы должны увидеть список с опросом «What's up» из [Части 2](#). Ссылка ведет на страницу опроса.

Функция `render()`

Процесс загрузки шаблона, добавления контекста и возврат объекта `HttpResponse`, вполне тривиальный. Django предоставляет функцию для всех этих операций. Вот как будет выглядеть наш `index()`:

```
polls/views.py

from django.shortcuts import render

from .models import Question

def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    context = {'latest_question_list': latest_question_list}
    return render(request, 'polls/index.html', context)
```

Так как мы используем такой подход во всех наших представлениях, нет необходимости импортировать `loader` и `HttpResponse` (`HttpResponse` еще нужен, если остались старые `detail`, `results` и `vote`).

Функция `render()` первым аргументом принимает объект запроса, также название шаблона и необязательный словарь значений контекста. Возвращает объект `HttpResponse` содержащий выполненный шаблон с указанным контекстом.

Вызов 404 исключения

Теперь создадим страницу опроса, которая отображает вопрос и варианты ответа. Вот так будет выглядеть наше представление:

```
polls/views.py

from django.http import Http404
from django.shortcuts import render

from .models import Question
# ...
def detail(request, question_id):
    try:
        question = Question.objects.get(pk=question_id)
    except Question.DoesNotExist:
        raise Http404("Question does not exist")
    return render(request, 'polls/detail.html', {'question': question})
```

Представление вызывает исключение `Http404`, если вопрос с указанным ID не существует.

Содержимое шаблона `polls/detail.html` обсудим позже, но если хотите прямо сейчас, чтобы все заработало, вот его содержимое:

```
polls/templates/polls/detail.html

{{ question }}
```

чтобы можно было загрузить страницу.

Функция `get_object_or_404()`

Вызов `get()` и `Http404` при отсутствии объекта – обычные операции. Django предоставляет функцию, которая выполняет эти действия. Вот как будет выглядеть наше представление `detail()`:

```
polls/views.py

from django.shortcuts import get_object_or_404, render

from .models import Question
# ...
def detail(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/detail.html', {'question': question})
```

Функция `get_object_or_404()` первым аргументом принимает Django модель и произвольное количество именованных аргументов, которые передаются в метод `get()` менеджера модели. Если объект не найден, вызывается исключение `Http404`.



Философия

Зачем мы используем функцию `get_object_or_404()` вместо того, чтобы автоматически перехватывать исключения `ObjectDoesNotExist` уровнем выше, или вызывать на уровне API моделей исключение `Http404` вместо `ObjectDoesNotExist`?

Потому что это связывает уровень моделей с уровнем представления. Один из главных принципов проектирования Django – слабая связанность. Некоторая связанная функциональность находится в модуле `django.shortcuts`.

Существует также функция `get_list_or_404()`, которая работает аналогично `get_object_or_404()`, но использует `filter()` вместо `get()`. Вызывает `Http404`, если получен пустой список.

Использование системы шаблонов

Вернемся к представлению `detail()`. Вот как может выглядеть наш шаблон `polls/detail.html`, использующий контекстную переменную `question`:

```
polls/templates/polls/detail.html

<h1>{{ question.question_text }}</h1>
<ul>
{% for choice in question.choice_set.all %}
    <li>{{ choice.choice_text }}</li>
{% endfor %}
</ul>
```

Система шаблонов использует точку для доступа к атрибутам переменной. Например, для `{{ question.question_text }}` Django сначала пытается обратиться к `question` как к словарю. При неудаче ищется атрибут переменной, в данном случае он и используется. Если атрибут не найден, будет искаться индекс в списке.

В теге `{% for %}` выполняется вызов метода: `question.choice_set.all`, интерпретируется как код Python `question.choice_set.all()`, который возвращает итератор по `Choice` для использования в теге `{% for %}`.

Подробности о шаблонах смотрите в разделе о [Языке шаблонов Django](#).

Избавляемся от «хардкода» URL-ов в шаблонах

Помните, когда мы указывали ссылку в шаблоне `polls/index.html`, она была прописана прямо в коде:

```
<li><a href="/polls/{{ question.id }}">{{ question.question_text }}</a></li>
```

Проблема в том, что нам будет очень сложно поменять URL-ы в проекте с большим количеством шаблонов. Однако, так как мы указали названия при вызове `path()` в модуле `polls.urls`, мы можем ссылаться на шаблоны URL-ов используя шаблонный тег `{% url %}`:

```
<li><a href="{% url 'detail' question.id %}">{{ question.question_text }}</a></li>
```

Определение URL-а будет найдено в модуле `polls.urls`. Вот где определен наш URL с названием „detail“:

```
...
# the 'name' value as called by the {% url %} template tag
path('int:question_id/', views.detail, name='detail'),
...
```

Теперь, если вы захотите поменять URL, например на `polls/specifc/12/`, вам не придется менять все шаблоны, вы можете сделать это в `polls/urls.py`:

```
...
# added the word 'specifics'
path('specifics/int:question_id/', views.detail, name='detail'),
...
```

Пространства имен в названиях URL-ов

Наш проект содержит только одно приложение `polls`. В реальных проектах может быть 5, 10, 20 и больше приложений. Как же Django понимает где чей URL по его названию? Например, приложение `polls` содержит представление `detail`, аналогичное представление может быть и в приложении для блогов. Как же Django понимает для какого представления создается URL при использовании тега `{% url %}`?

Для этого используются пространства имен в URLconf. Изменим `polls/urls.py` и добавим пространство имен в `app_name`:

```
polls/urls.py

from django.urls import path

from . import views

app_name = 'polls'
urlpatterns = [
    path('', views.index, name='index'),
    path('<int:question_id>/', views.detail, name='detail'),
    path('<int:question_id>/results/', views.results, name='results'),
    path('<int:question_id>/vote/', views.vote, name='vote'),
]
```

Теперь поменяем в шаблоне `polls/index.html`:

```
polls/templates/polls/index.html

<li><a href="{% url 'detail' question.id %}">{{ question.question_text }}</a></li>
```

чтобы использовать пространство имен URL-ов:

```
polls/templates/polls/index.html

<li><a href="{% url 'polls:detail' question.id %}">{{ question.question_text }}</a></li>
```

When you're comfortable with writing views, read [part 4 of this tutorial](#) to learn the basics about form processing and generic views.

« previous | up | next »