Написание вашего первого приложения Django, часть 3

Это учебное пособие начинается с того места, где остановился Учебное пособие 2. Мы продолжаем приложение для веб-опросов и сосредоточимся на создании общедоступного интерфейса - «просмотров».

Где получить помощь:

Если у вас возникли проблемы с прохождением этого учебника, перейдите в раздел «Получение справки» в разделе часто задаваемых вопросов.

Обзор

Представление - это "тип" веб-страницы в вашем приложении Django, который обычно выполняет определенную функцию и имеет определенный шаблон. Например, в приложении для блога у вас могут быть следующие представления:

- Домашняя страница блога - отображает последние несколько записей.
- Страница «подробности» записи - страница постоянной ссылки для одной записи.
- Годовая страница архива - отображает все месяцы с записями в данном году.
- Страница архива на основе месяца - отображает все дни с записями в данном месяце.
- Страница дневного архива - отображает все записи в данный день.
- Действие комментариев - обрабатывает размещение комментариев к данной записи.

В нашем приложении для голосования у нас будут следующие четыре мнения:

- Страница «индекс» вопроса - отображает последние несколько вопросов.
- Страница "Подробности" вопроса - отображает текст вопроса без результатов, но с формой для голосования.
- Страница «результаты» вопроса - отображает результаты по конкретному вопросу.
- Голосование - обрабатывает голосование за конкретный выбор по конкретному вопросу.

В Джанго веб-страницы и другой контент доставляются просмотрами. Каждое представление представлено функцией Python (или методом, в случае представлений на основе классов). Django выберет представление, изучив запрошенный URL-адрес (точнее, часть URL-адреса после доменного имени).

Now in your time on the web you may have come across such beauties as **`ME2/Sites/dirmod.htm?sid=&type=gen&mod=Core+Pages&gid=A6CD4967199A42D9B65B1B`**. You will be pleased to know that Django allows us much more elegant *URL patterns* than that.

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

Чтобы получить от URL-адреса к представлению, Django использует то, что известно как «URLconfs». URLconf сопоставляет шаблоны URL-адресов с представлениями.

Это учебное пособие содержит основные инструкции по использованию URLconfs, и для получения дополнительной информации вы можете обратиться к диспетчеру URL.

Написание большего количества просмотров

Теперь давайте добавим еще несколько просмотров в **`polls/views.py`**. Эти взгляды немного отличаются, потому что они принимают аргумент:

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)
```

Подключите эти новые представления к модулю **`polls.urls`**, добавив следующие вызовы **`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/
    path('<int:question_id>/results/', views.results,
name='results'),
    # ex: /polls/5/vote/
    path('<int:question_id>/vote/', views.vote,
name='vote'),
]
```

Take a look in your browser, at "/polls/34/". It'll run the **`detail()`** method and display whatever ID you provide in the URL. Try "/polls/34/results/" and "/polls/34/vote/" too – these will display the placeholder results and voting pages.

When somebody requests a page from your website – say, "/polls/34/", Django will load the **`mysite.urls`** Python module because it's pointed to by the **`ROOT_URLCONF`** setting. It finds the variable named **`urlpatterns`** and traverses the patterns in order. After finding the match at **`'polls/'`**, it strips off the matching text (**`"polls/"`**) and sends the remaining text – **`"34/"`** – to the 'polls.urls' URLconf for further processing. There it matches **`'<int:question_id>/'`**, resulting in a call to the **`detail()`** view like so:

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

The **`question_id=34`** part comes from **`<int:question_id>`**. Using angle brackets " captures" part of the URL and sends it as a keyword argument to the view function. The **`:question_id>`** part of the string defines the name that will be used to identify the matched pattern, and the **`<int:`** part is a converter that determines what patterns should match this part of the URL path.

Напишите представления, которые на самом деле что-то делают

Каждое представление отвечает за одну из двух вещей: возврат объекта **`HttpResponse`**, содержащего содержимое запрошенной страницы, или за исключение, такое как **`Http404`**. Остуть зависит от вас.

Ваше представление может читать записи из базы данных или нет. Он может использовать систему шаблонов, такую как Django - или стороння система шаблонов Python - или нет. Он может генерировать PDF-файл, выводить XML, создавать ZIP-файл на лету, все, что вы хотите, используя любые библиотеки Python, которые вы хотите.

Все, чего хочет Джанго, это **`HttpResponse`**. Или исключение.

Поскольку это удобно, давайте использовать собственный API базы данных Django, который мы рассмотрели в Учебнике 2. Вот один удар в новом представлении **`index()`**, который отображает последние 5 вопросов опроса в системе, разделенных запятыми, в соответствии с датой публикации:

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
```

Однако здесь есть проблема: дизайн страницы жестко запрограммирован в представлении. Если вы хотите изменить внешний вид страницы, вам придется отредактировать этот код Python. Итак, давайте использовать систему шаблонов Django, чтобы отделить дизайн от Python, создав шаблон, который может использовать представление.

First, create a directory called **`templates`** in your **`polls`** directory. Django will look for templates in there.

Your project's **`TEMPLATES`** setting describes how Django will load and render templates. The default settings file configures a **`DjangoTemplates`** backend whose **`APP_DIRS`** option is set to **`True`**. By convention **`DjangoTemplates`** looks for a "templates" subdirectory in each of the **`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`**.

Шаблоны именные интервалы

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.

Поместите следующий код в этот 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

Чтобы сделать учебник короче, во всех примерах шаблонов используется неполный HTML. В своих собственных проектах вы должны использовать полные HTML-документы.

Now let's update our **`index`** view in **`polls/views.py`** to use the template:

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))
```

That code loads the template called **`polls/index.html`** and passes it a context. The context is a dictionary mapping template variable names to Python objects.

Загрузите страницу, указав браузеру на «/polls/», и вы увидите маркированный список, содержащий вопрос «Как дела» из учебника 2. Ссылка указывает на страницу сведений о вопросе.

Ярлык: `render()`

It's a very common idiom to load a template, fill a context and return an **`HttpResponse`** object with the result of the rendered template. Django provides a shortcut. Here's the full **`index()`** view, rewritten:

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)
```

Note that once we've done this in all these views, we no longer need to import **`loader`** and **`HttpResponse`** (you'll want to keep **`HttpResponse`** if you still have the stub methods for **`detail`**, **`results`**, and **`vote`**).

The **`render()`** function takes the request object as its first argument, a template name as its second argument and a dictionary as its optional third argument. It returns an **`HttpResponse`** object of the given template rendered with the given context.

Повышение ошибки 404

Теперь давайте рассмотрим подробное представление вопроса - страницу, на которой отображается текст вопроса для данного опроса. Вот мнение:

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})
```

Новая концепция здесь: Представление вызывает исключение **`Http404`**, если вопрос с запрошенным идентификатором не существует.

We'll discuss what you could put in that **`polls/detail.html`** template a bit later, but if you'd like to quickly get the above example working, a file containing just:

polls/templates/polls/detail.html

```
{{ question }}
```

пока вы начнете.

Ярлык: `get_object_or_404()`

It's a very common idiom to use **`get()`** and raise **`Http404`** if the object doesn't exist. Django provides a shortcut. Here's the **`detail()`** view, rewritten:

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})
```

The **`get_object_or_404()`** function takes a Django model as its first argument and an arbitrary number of keyword arguments, which it passes to the **`get()`** function of the model's manager. It raises **`Http404`** if the object doesn't exist.

Философия

Why do we use a helper function **`get_object_or_404()`** instead of automatically catching the **`ObjectDoesNotExist`** exceptions at a higher level, or having the model API raise **`Http404`** instead of **`ObjectDoesNotExist`**?

Потому что это свяжет слой модели со слоем вида. Одной из главных целей дизайна Django является поддержание свободной связи. Некоторые контролируемые соединения вводятся в модуль **`django.shortcuts`**.

There's also a **`get_list_or_404()`** function, which works just as **`get_object_or_404()`** – except using **`filter()`** instead of **`get()`**. It raises **`Http404`** if the list is empty.

Используйте систему шаблонов

Back to the **`detail()`** view for our poll application. Given the context variable **`question`**, here's what the **`polls/detail.html`** template might look like:

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>
```

The template system uses dot-lookup syntax to access variable attributes. In the example of **`{{ question.question_text }}`**, first Django does a dictionary lookup on the object **`question`**. Failing that, it tries an attribute lookup – which works, in this case. If attribute lookup had failed, it would've tried a list-index lookup.

Method-calling happens in the **`{% for %}`** loop: **`question.choice_set.all`** is interpreted as the Python code **`question.choice_set.all()`**, which returns an iterable of **`Choice`** objects and is suitable for use in the **`{% for %}`** tag.

Подробнее о шаблонах см. в руководстве по шаблонам.

Удаление жестко запрограммированных URL-адресов в шаблонах

Помните, когда мы писали ссылку на вопрос в шаблоне **`polls/index.html`**, ссылка была частично жестко запрограммирована следующим образом:

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

The problem with this hardcoded, tightly-coupled approach is that it becomes challenging to change URLs on projects with a lot of templates. However, since you defined the name argument in the **`path()`** functions in the **`polls.urls`** module, you can remove a reliance on specific URL paths defined in your url configurations by using the **`{% url %}`** template tag:

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

Это работает путем поиска определения URL-адреса, указанного в модуле **`polls.urls`**. Вы можете точно увидеть, где определено URL-имя 'detail' ниже:

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

If you want to change the URL of the polls detail view to something else, perhaps to something like **`polls/specifics/12/`** instead of doing it in the template (or templates) you would change it in **`polls/urls.py`**:

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

Имена URL-адресов с интервалом имен

The tutorial project has just one app, **`polls`**. In real Django projects, there might be five, ten, twenty apps or more. How does Django differentiate the URL names between them? For example, the **`polls`** app has a **`detail`** view, and so might an app on the same project that is for a blog. How does one make it so that Django knows which app view to create for a url when using the **`{% url %}`** template tag?

The answer is to add namespaces to your URLconf. In the **`polls/urls.py`** file, go ahead and add an **`app_name`** to set the application namespace:

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'),
]
```

Now change your **`polls/index.html`** template from:

polls/templates/polls/index.html

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

чтобы указать на выносной элемент с пространством имен:

polls/templates/polls/index.html

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

Когда вам удобно писать представления, прочитайте часть 4 этого учебника, чтобы узнать основы обработки форм и общих представлений.

< Написание вашего первого приложения Django, часть 2
Написание вашего первого приложения Django, часть 4 >

Поддержите Django!

Содержание

Просмотр

- Предыдущий: Написание вашего первого приложения Django, часть 2
- Далее: Написание вашего первого приложения Django, часть 4
- Содержание
- Общий индекс
- Индекс модуля Python

Вы находитесь здесь:

- Документация Django 3.2
  - Введение
    - Написание вашего первого приложения Django, часть 3

Получение помощи

Часто задаваемые вопросы
Попробуйте FAQ — в нем есть ответы на многие распространенные вопросы.

Индекс, индекс модуля или оглавление
Удобно при поиске конкретной информации.

список рассылки django-users
Найдите информацию в архивах списка рассылки django-users или оставьте вопрос.

#django IRC-канал
Задайте вопрос в IRC-канале #django или выполните поиск в журналах IRC, чтобы узнать, задавался ли он раньше.

Трекер билетов
Сообщите об ошибках с документацией Django или Django в нашем трекере билетов.

Скачать:

Автономный (Django 3.2): HTML | PDF | ePub
Предоставлено Read the Docs.

Дополнительная информация
О Django
Начало работы с Django
Организация команды
Django Software Foundation
Правила поведения
Заявление о разнообразии

Принять участие
Присоединиться к группе
Внести свой вклад в Django
Отправить ошибку
Сообщить о проблеме безопасности

Подписывайтесь на нас
GitHub
Twitter
Новости RSS
Список рассылки пользователей Django

Поддержать нас
Спонсор Джанго
Официальный магазин товаров
Улыбка Amazon
Программа благотворительности на рабочем месте

django Хостинг по rackspace Дизайн по threespot & andrevv

Получение помощи
Язык: ru
Версия документации: 3.2