Создаём своё первое приложение с Django, часть 4

This tutorial begins where Tutorial 3 left off. We're continuing the Web-poll application and will focus on form processing and cutting down our code.

Where to get help:

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

Write a minimal form

Отредактируем шаблон страницы опроса («polls/detail.html») из предыдущего раздела учебника, и добавим HTML элемент `<form>`:

polls/templates/polls/detail.html

```
<h1>{{ question.question_text }}</h1>

{% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}

<form action="{% url 'polls:vote' question.id %}" method="post">
{% csrf_token %}
{% for choice in question.choice_set.all %}
    <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}">
    <label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br>
{% endfor %}
<input type="submit" value="Vote">
</form>
```

Краткий обзор:

- Данный шаблон отображает radio-поле для каждого варианта ответа вопроса. `value` поля содержит ID ответа. `name` каждого поля равно `"choice"`. Это означает, что при выборе поля и отправке формы, будет отправлены POST данные `choice=#`, где # равно ID выбранного ответа. Это основной принцип работы HTML форм.
- We set the form's `action` to `{% url 'polls:vote' question.id %}`, and we set `method="post"`. Using `method="post"` (as opposed to `method="get"`) is very important, because the act of submitting this form will alter data server-side. Whenever you create a form that alters data server-side, use `method="post"`. This tip isn't specific to Django; it's good Web development practice in general.
- `forloop.counter` содержит номер итерации цикла тега `for`
- Since we're creating a POST form (which can have the effect of modifying data), we need to worry about Cross Site Request Forgeries. Thankfully, you don't have to worry too hard, because Django comes with a helpful system for protecting against it. In short, all POST forms that are targeted at internal URLs should use the `{% csrf_token %}` template tag.

Теперь создадим представление Django, которое принимает данные отправленные формой и обрабатывает их. Вспомним, в Части 3 мы создали URLconf который содержит следующую строку:

polls/urls.py

```
path('<int:question_id>/vote/', views.vote, name='vote'),
```

Мы также создали функцию-«заглушку» `vote()`. Давайте создадим настоящую функцию представления. Добавьте следующее в `polls/views.py`:

polls/views.py

```
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import get_object_or_404, render
from django.urls import reverse

from .models import Choice, Question
# ...
def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice = question.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        # Redisplay the question voting form.
        return render(request, 'polls/detail.html', {
            'question': question,
            'error_message': "You didn't select a choice.",
        })
    else:
        selected_choice.votes += 1
        selected_choice.save()
        # Always return an HttpResponseRedirect after successfully dealing
        # with POST data. This prevents data from being posted twice if a
        # user hits the Back button.
        return HttpResponseRedirect(reverse('polls:results', args=(question.id,)))
```

Этот код содержит вещи, которые еще не объяснялись в учебнике:

Оглавление

- Создаём своё первое приложение с Django, часть 4
  - Write a minimal form
  - Общие представления: меньше кода - меньше проблем
    - Изменим URLconf
    - Изменим представления

Предыдущий раздел

Создаём своё первое приложение с Django, часть 3

Следующий раздел

Создаём своё первое приложение с Django, часть 5

Эта страница

- Исходный текст

Последнее обновление:

нояб. 23, 2021

- `request.POST` – это объект с интерфейсом словаря, который позволяет получить отправленные данные через название ключа. В нашем случае `request.POST['choice']` вернет ID выбранного варианта ответа в виде строки. `request.POST` всегда содержит строки.

  Заметим что Django также предоставляет `request.GET` для аналогичного доступа к GET данным – но мы используем `request.POST`, чтобы быть уверенным, что данные передаются только при POST запросе.

- `request.POST['choice']` вызовет исключение `KeyError`, если `choice` не находится в POST. Код обрабатывает исключение `KeyError` и показывает страницу опроса с ошибкой, если не был выбран вариант ответа.

- После увеличения счетчика ответов представление возвращает `HttpResponseRedirect` вместо `HttpResponse`. `HttpResponseRedirect` принимает один аргумент: URL, на который необходимо перенаправить пользователя (обратите внимание, как мы создаем URL).

  As the Python comment above points out, you should always return an `HttpResponseRedirect` after successfully dealing with POST data. This tip isn't specific to Django; it's good Web development practice in general.

- Мы используем функцию `reverse()` при создании `HttpResponseRedirect`. Это функция помогает избегать «хардкодинга» URL-ов в коде. Она принимает название URL-шаблона и необходимые аргументы для создания URL-а. В этом примере используется конфигурация URL-ов из Части 3 и `reverse()` вернет:

  ```
  '/polls/3/results/'
  ```

  … где `3` значение `question.id`. При запросе к этому URL-у вызовется представление `'results'` для отображения результатов опроса.

Как упоминалось в Части 3, `request` является экземпляром `HttpRequest`. Подробности о классе `HttpRequest` смотрите в разделе об объектах запроса и ответа.

После ответа на опрос, представление `vote()` перенаправит пользователя на страницу с результатами. Давайте создадим представление для этой страницы:

polls/views.py

```
from django.shortcuts import get_object_or_404, render


def results(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/results.html', {'question': question})
```

Это представление аналогично представлению `detail()` из Части 3. Единственная разница - используемый шаблон. Повторение кода мы исправим позже.

Теперь создадим шаблон `polls/results.html`:

polls/templates/polls/results.html

```
<h1>{{ question.question_text }}</h1>

<ul>
{% for choice in question.choice_set.all %}
    <li>{{ choice.choice_text }} -- {{ choice.votes }} vote{{ choice.votes|pluralize }}</li>
{% endfor %}
</ul>

<a href="{% url 'polls:detail' question.id %}">Vote again?</a>
```

Теперь откройте страницу `/polls/1/` в браузере и ответьте на опрос. Вы должны увидеть страницу с результатами, которые обновляются после каждого ответа на опрос. Если вы отправите форму, не ответив на вопрос, вы должны увидеть сообщение об ошибке.

Примечание

У кода представления `vote()` есть небольшие проблемы. Он сначала получает объект `selected_choice` из базы данных, затем рассчитывает новое значение `votes`, и сохраняет обратно в базу данных. Если два пользователя проголосуют *в один момент*, код может сработать неправильно: одно и то же значение, скажем 42, будет получено для `votes`. Затем для обоих пользователей получится и сохранится значение 43, хотя должно быть 44.

Это называется *состояние гонки*. Подробнее можете прочитать в разделе о как предотвратить состояние гоник, используя F().

Общие представления: меньше кода - меньше проблем

The `detail()` (from Tutorial 3) and `results()` views are very short – and, as mentioned above, redundant. The `index()` view, which displays a list of polls, is similar.

Эти представления выполняют стандартные операции Веб-приложений: получение данных из базы данных в соответствии с параметрами из URL, загрузка шаблона и возвращение результата выполнения шаблона. Для таких стандартных операций Django предоставляет набор «общих представлений(generic views)».

Общие представления позволяют создавать приложения практически не написав ни строчки кода Python.

Let's convert our poll app to use the generic views system, so we can delete a bunch of our own code. We'll have to take a few steps to make the conversion. We will:

1. Изменить URLconf.
2. Удалить старые и ненужные представления.
3. Создать новые представления из встроенных в Django общих представлений.

Подробности читайте далее.

Зачем мы переписываем код?

Скорее всего, вы будете использовать общие представления с самого начала без необходимости рефакторить проект. Этот учебник специально описывает создание представлений, чтобы описать основные концепции.

Вы должны знать основы математики для использования калькулятора.

Изменим URLconf

Первым делом откройте `polls/urls.py` и измените его следующим образом:

polls/urls.py

```
from django.urls import path

from . import views

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

Обратите внимание, мы поменяли `<question_id>` на `<pk>` во втором и третьем URL-ах.

Изменим представления

Теперь мы собираемся удалить старые представления `index`, `detail` и `results` и воспользуемся встроенными в Django общими представлениями. Для этого измените файл `polls/views.py`:

polls/views.py

```
from django.http import HttpResponseRedirect
from django.shortcuts import get_object_or_404, render
from django.urls import reverse
from django.views import generic

from .models import Choice, Question


class IndexView(generic.ListView):
    template_name = 'polls/index.html'
    context_object_name = 'latest_question_list'

    def get_queryset(self):
        """Return the last five published questions."""
        return Question.objects.order_by('-pub_date')[:5]


class DetailView(generic.DetailView):
    model = Question
    template_name = 'polls/detail.html'


class ResultsView(generic.DetailView):
    model = Question
    template_name = 'polls/results.html'


def vote(request, question_id):
    ... # same as above, no changes needed.
```

Мы используем здесь два общих представления: `ListView` и `DetailView`. Эти два типа представлений отображают две концепции: «отображение списка объектов» и «отображение подробностей о конкретном объекте».

- Каждое общее представление должно знать с какой моделью работать. Ее можно указать с помощью атрибута `model`.
- Представление `DetailView` принимает значение первичного ключа из URL с названием `"pk"`, поэтому мы изменили название параметра с `question_id` на `pk`.

По умолчанию представление `DetailView` использует шаблон `<app name>/<model name>_detail.html`. В нашем случае будет использоваться `"polls/question_detail.html"`. Атрибут `template_name` позволяет указать Django какой шаблон использовать. Мы указали `template_name` также для представления `results`, чтобы страница результата и подробностей использовали разные шаблоны, несмотря на то, что они используют представление `DetailView`.

Аналогично `ListView` использует шаблон `<app name>/<model name>_list.html`, мы использовали `template_name`, чтобы определить другой шаблон - `"polls/index.html"`.

1

In previous parts of the tutorial, the templates have been provided with a context that contains the `question` and `latest_question_list` context variables. For `DetailView` the `question` variable is provided automatically – since we're using a Django model (`Question`), Django is able to determine an appropriate name for the context variable. However, for ListView, the automatically generated context variable is `question_list`. To override this we provide the `context_object_name` attribute, specifying that we want to use `latest_question_list` instead. As an alternative approach, you could change your templates to match the new default context variables – but it's a lot easier to tell Django to use the variable you want.

Запустите сервер и протестируйте наше приложение.

Подробности об общих представлениях смотрите в соответствующем разделе.

Когда вы освоите формы и общие представления, переходите к пятой части учебника, чтобы узнать о тестировании нашего приложения.