# Создаём своё первое приложение с Django, часть 4

Мы продолжим разрабатывать приложение для голосования и сосредоточимся на обработке форм и упрощении нашего кода.

## Создадим простую форму

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;
- Значение action формы равно {% url 'polls:vote' question.id %}, еще добавлено method="post". Использование method="post" (вместо method="get") очень важно, так как отправка формы изменяет данные. Для форм, изменяющих данные используйте method="post". Этот совет не относится к Django, это хорошая практика для Web-приложений;
- forloop.counter содержит номер итерации цикла тега;
- Так как мы создаем POST форму (которая может привести к изменениям данных), нам следует побеспокоится о Cross Site Request Forgeries. Благодаря Django это очень просто. В общем, все POST формы, которые отправляются на внутренний URL, должны использовать тег {% csrf_token %}.

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

```
polls/urls.py

url(r'^(?P<question_id>[0-9]+)/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.shortcuts import get_object_or_404, render
from django.http import HttpResponseRedirect, HttpResponse
from django.core.urlresolvers 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:

- 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);

После успешной обработки POST запроса, вы должны возвращать HttpResponseRedirect. Это не относится конкретно к Django, это просто хорошая практика при разработке Web-приложений.

- 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 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 главе 3, request является экземпляром HttpRequest.

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 главы 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.

## Общие представления: меньше кода - меньше проблем

Представления detail() и results() до глупости просты – и, как упоминалось выше, избыточны. Представление index(), которое показывает список опросов, также довольно стандартное.

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.

Давайте используем их в нашем приложении. Необходимо выполнить следующие действия:

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.conf.urls import url

from . import views

urlpatterns = [
    url(r'^$', views.IndexView.as_view(), name='index'),
    url(r'^(?P<pk>[0-9]+)/$', views.DetailView.as_view(), name='detail'),
    url(r'^(?P<pk>[0-9]+)/results/$', views.ResultsView.as_view(), name='results'),
    url(r'^(?P<question_id>[0-9]+)/vote/$', 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.shortcuts import get_object_or_404, render
from django.http import HttpResponseRedirect
from django.core.urlresolvers 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
```

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".

В предыдущей части учебника в шаблоне использовался контекст с переменными question и latest_question_list. Для DetailView переменная question добавляется автоматически – так как мы используем модель Question, Django автоматически генерирует подходящее название для переменного контекста. Для ListView переменная будет называться question_list. Чтобы переопределить это название, мы использовали атрибут context_object_name указав latest_question_list. Как вариант, вы можете изменить шаблон и использовать question_list, но проще указать Django какое название переменной контекста использовать.

Запустите сервер и протестируйте наше приложение.

Когда вы освоите формы и общие представления, переходите к следующей главе: *«Создаём своё первое приложение с Django, часть 5»*, чтобы узнать о тестировании нашего приложения.