Django Beginner’s Tutorial

Parts 3 and 4

`https://docs.djangoproject.com/en/2.2/intro/tutorial03/`
`https://docs.djangoproject.com/en/2.2/intro/tutorial04/`

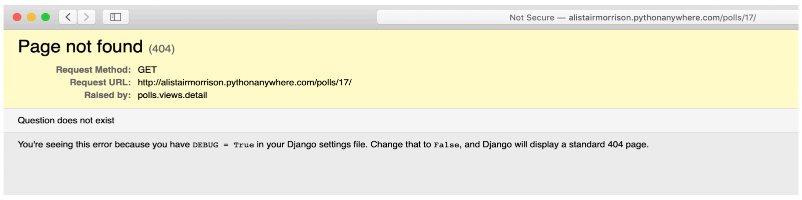
- More views ✓
- Templates ✓
- 404 errors
- Removing hardcoded URLs
- Forms
- Population scripts

Raising a 404 error

- Add the following to `views.py`:

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

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})
```


Not Secure — alistairmorrison.pythonanywhere.com/polls/17/

Page not found (404)

```
Request Method: GET
   Request URL: http://alistairmorrison.pythonanywhere.com/polls/17/
     Raised by: polls.views.detail
```

Question does not exist

You're seeing this error because you have `DEBUG = True` in your Django settings file. Change that to `False`, and Django will display a standard 404 page.

A shortcut: get_object_or_404()

- Add the following to `views.py`:

```
from django.shortcuts import get_object_or_404, render

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

A template for the detail view

- Create a `detail.html`:

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

Removing hardcoded URLs

- When we wrote the link to a question in the `index.html` template, the link was partially hardcoded like this:

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

- Becomes challenging to change URLs on projects with many templates

- Solution: use name argument in `path` functions in `urls.py` together with the `{% url %}` template tag

Mapping URLs

- Add the following code to `urls.py` in the `polls` folder:

```
from django.urls import path
from . import views

urlpatterns = [
    # e.g. /polls/
    path('', views.index, name='index'),

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

5

Removing hardcoded URLs

- When we wrote the link to a question in the `index.html` template, the link was partially hardcoded like this:

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

- Becomes challenging to change URLs on projects with many templates
- Solution: use name argument in `path` functions in `urls.py` together with the `{% url %}` template tag
- Replace the code above by:

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

6

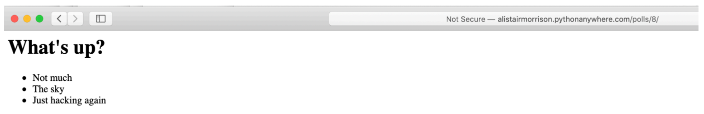
Removing hardcoded URLs (cont)

- The way this works is by looking up the URL definition as specified in the `urls.py`:

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

- To change a question URL, e.g., to `polls/specifics/12/`, instead of modifying templates, simply change `urls.py`:

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


Not Secure — alistairmorrison.pythonanywhere.com/polls/8/

What's up?

- Not much
- The sky
- Just hacking again

Namespacing URL names

- In big Django projects there may be many apps; several may have `detail` views, for example
- Differentiate between the apps using namespaces
- Add `app_name = 'polls'`
  in `urls.py` just before the `urlpatterns` assignment
- Then namespace template tags
- For example change index.html as follows:

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

Writing a simple form

- Add the following to `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”
    value="{{ choice.id }}" />
    <label> {{ choice.choice_text }}</label><br />
{% endfor %}
<input type="submit" value="Vote" />
</form>
```

9

Writing a (better) simple form

- (to allow users to also click on label, add matching id and for:)

```
<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>
```

10

Writing the view for the voting form

• Add the following to `views.py`:

```
from django.shortcuts import get_object_or_404, render
from django.http import HttpResponseRedirect, HttpResponse
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.",
        })
```

11

Writing the view for the voting form

•Add the following to `views.py`:

```
    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,)))
```

•`reverse()` in the `HttpResponseRedirect` constructor avoids having to hardcode a URL in the view function

•It is given the name of the view that we want to pass control to and the variable portion of the URL pattern that points to that view

•`reverse()` will return a string like `/polls/3/results/`

12

Writing a results view and template

- Add the following to `views.py`:

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

- Almost the same as the detail view!
- Add the following to `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>
```

Writing a population script

- Create file `populate_polls.py` in main proj dir:

```
import os
os.environ.setdefault('DJANGO_SETTINGS_MODULE',
                      'mysite.settings')
import django
django.setup()

from polls.models import Question, Choice
from datetime import datetime
from pytz import utc

def populate():
    question1_choices = [
        {"choice_text": "The sky", "votes": 5},
        {"choice_text": "Just hacking", "votes": 8},
        {"choice_text": "Not much", "votes": 2},]

    ... # similarly for questions 2-5
```

Writing a population script (cont)

```
questions = {"What's up?": {"choices":
             question1_choices, "pub_date":
             datetime(2020, 10, 17, 15, 30,
                      tzinfo=utc)},
             ...
             # similarly for questions 2-5
            }

for question, question_data in questions.items():
    q = add_question(question,
                     question_data["pub_date"])
    for c in question_data["choices"]:
        add_choice(q, c["choice_text"], c["votes"])
```

Writing a population script (cont)

```
def add_question(question_text, pub_date):
    question = Question.objects.get_or_create(
      question_text=question_text, pub_date=pub_date)[0]
    question.save()
    return question

def add_choice(question, choice_text, votes):
    choice = Choice.objects.get_or_create(
         question=question, choice_text=choice_text,
         votes = votes)[0]
    choice.save()
    return choice

# Start execution here!
if __name__ == '__main__':
    populate()
```