Django Beginner’s Tutorial

Parts 1 and 2

`https://docs.djangoproject.com/en/2.2/intro/tutorial01/`
`https://docs.djangoproject.com/en/2.2/intro/tutorial02/`

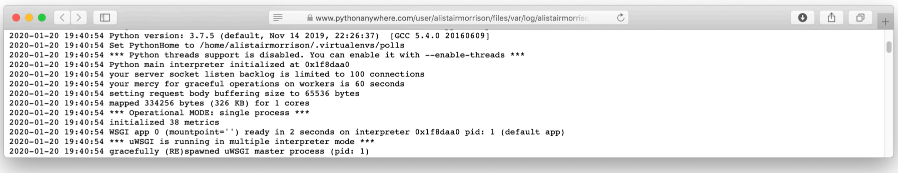
- **Making ‘Polls’ web application in Django**

- **Using PythonAnywhere**
- **Creating projects**
- **Creating views**
- **Mapping URLs**
- **Creating models**
- **Querying and modifying models**
- **The Django admin site**

Diagnostic information

- **To check what packages are installed in a given virtual environment:**
  - **run `pip list`**
  - **the command `django-admin version` can also tell you which version of Django you are running**

- **Log files can be viewed:**
  - `access.log` **(requests made to subdomain)**
  - `error.log` **(errors produced by application)**
  - `server.log` **(log of UNIX processes running on application)**


www.pythonanywhere.com/user/alistairmorrison/files/var/log/alistairmorriso

```
2020-01-20 19:40:54 Python version: 3.7.5 (default, Nov 14 2019, 22:26:37)  [GCC 5.4.0 20160609]
2020-01-20 19:40:54 Set PythonHome to /home/alistairmorrison/.virtualenvs/polls
2020-01-20 19:40:54 *** Python threads support is disabled. You can enable it with --enable-threads ***
2020-01-20 19:40:54 Python main interpreter initialized at 0x1f8daa0
2020-01-20 19:40:54 your server socket listen backlog is limited to 100 connections
2020-01-20 19:40:54 your mercy for graceful operations on workers is 60 seconds
2020-01-20 19:40:54 setting request body buffering size to 65536 bytes
2020-01-20 19:40:54 mapped 334256 bytes (326 KB) for 1 cores
2020-01-20 19:40:54 *** Operational MODE: single process ***
2020-01-20 19:40:54 initialized 38 metrics
2020-01-20 19:40:54 WSGI app 0 (mountpoint='') ready in 2 seconds on interpreter 0x1f8daa0 pid: 1 (default app)
2020-01-20 19:40:54 *** uWSGI is running in multiple interpreter mode ***
2020-01-20 19:40:54 gracefully (RE)spawned uWSGI master process (pid: 1)
```

Writing your first view

- Views *receive* HttpRequest, *return* HttpReponse objects
- In `views.py`, include the following code:

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("Hello, world")
```

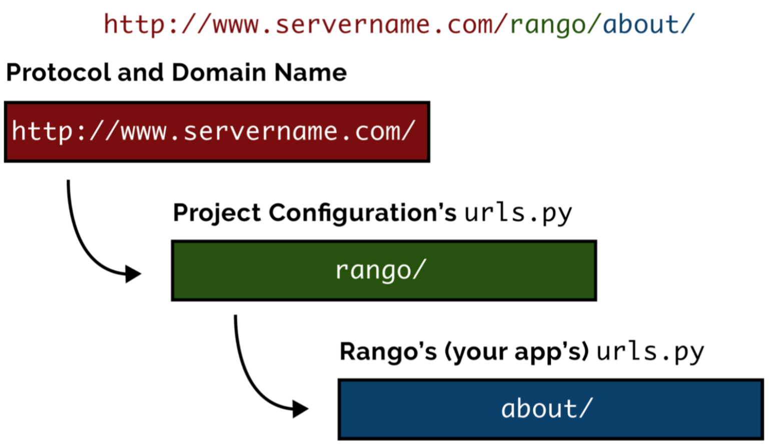
- In `urls.py` in the `mysite` folder, include the following code:

```
from django.urls import include, path
from django.contrib import admin

urlpatterns = [
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]
```

Directory Structure

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
    polls/
        __init__.py
        admin.py
        apps.py
        migrations/
            __init__.py
        models.py
        tests.py
        views.py
        urls.py
```



http://www.servername.com/rango/about/
Protocol and Domain Name
http://www.servername.com/
Project Configuration's urls.py
rango/
Rango's (your app's) urls.py
about/

An illustration of a URL, represented as a chain, showing how different parts of the URL following the domain are the responsibility of different `url.py` files.

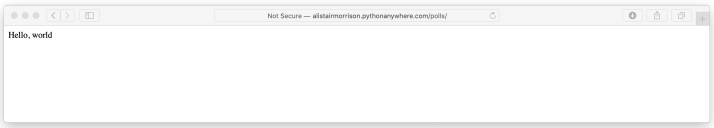
Writing your first view (cont)

- **Create `urls.py` in the `polls` folder, & write:**

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

urlpatterns = [
    path('', views.index, name='index'),
]
```

- **Path() function – path(route, view, kwargs=None, name)**
  - **Route: URL string; urlpatterns will go through list until it matches**
  - **View: When route found, Django calls the view with HttpRequest obj as argument**
  - **Name: allows URL to be referenced from elsewhere unambiguously (reverse name lookup); allows easy global alteration of URLs**


Not Secure — alistairmorrison.pythonanywhere.com/polls/

Hello, world

Setting up the model

- Run `python manage.py migrate` to create database tables
- Create your models. Add the following to `models.py`:

```
from django.db import models

class Question(models.Model):
   question_text = models.CharField(max_length=200)
   pub_date = models.DateTimeField('date published')

class Choice(models.Model):
   question = models.ForeignKey(Question,
                 on_delete=models.CASCADE)
   choice_text = models.CharField(max_length=200)
   votes = models.IntegerField(default=0)
```

- In `settings.py`, add `polls` to the `INSTALLED_APPS` list

7

Setting up the model (cont)

- Run `python manage.py makemigrations polls` to create migrations for the changes made to the model

```
Migrations for 'polls':
  0001_initial.py:
    - Create model Choice
    - Create model Question
    - Add field question to choice
```

- Then run `python manage.py migrate` again to apply the migrations to the database

```
Operations to perform:
  Apply all migrations: admin, contenttypes, polls, auth, sessions
Running migrations:
  Rendering model states... DONE
  Applying polls.0001_initial... OK
```

- Can run `python manage.py sqlmigrate polls 0001` to check the SQL code that is executed

8

Manually changing the model

- Execute `python manage.py shell` in the console

```
>>> from polls.models import Question, Choice
>>> Question.objects.all()
<Query set []>

>>> from django.utils import timezone
>>> q = Question(question_text="What's new?", pub_date=timezone.now())
>>> q.save()

>>> q.id
1
>>> q.question_text
"What's new?"
>>> q.pub_date
datetime.datetime(2020, 1, 16, 9, 15, 5, 775217, tzinfo=<UTC>)

>>> q.question_text = "What's up?"
>>> q.save()

>>> Question.objects.all()
<Query set [<Question: Question object>]>
```

9

A better representation of a Question object

- `Question: Question object` is not very illuminating
- We can add `__str__` methods (a bit like `toString`)
- Add the following to `models.py`:

```
from django.db import models

class Question(models.Model):
    # ...
    def __str__(self):
        return self.question_text

class Choice(models.Model):
    # ...
    def __str__(self):
        return self.choice_text
```

- `Question.objects.all()` returns `[<Question: What's up?>]`

10

A custom method for Question

- Determine if a Question has been published recently

- Add the following to `models.py`:

```
import datetime

from django.utils import timezone

class Question(models.Model):
    # ...
    def was_published_recently(self):
        return self.pub_date >= timezone.now() -
                                datetime.timedelta(days=1)
```

11

More queries and model changes

```
>>> from polls.models import Question, Choice
>>> Question.objects.filter(id=1)
<QuerySet [<Question: What's up?>]>
>>>
Question.objects.filter(question_text__startswith='What')
<QuerySet [<Question: What's up?>]>

# Get the question that was published this year.
>>> from django.utils import timezone
>>> current_year = timezone.now().year
>>> Question.objects.get(pub_date__year=current_year)
<Question: What's up?>

# ‘get’ expects 1 answer, no more, no less!
>>> Question.objects.get(id=2)
Traceback (most recent call last):
    ...
DoesNotExist: Question matching query does not exist.
```

12

More queries and model changes (2)

```
# Lookup by a primary key - identical to
# Question.objects.get(id=1).
>>> Question.objects.get(pk=1)
<Question: What's up?>

# Make sure our custom method worked.
>>> q = Question.objects.get(pk=1)
>>> q.was_published_recently()
True

# Give the Question some Choices.
>>> q = Question.objects.get(pk=1)

# Display any choices from the related object set
# -- none so far.
>>> q.choice_set.all()
<QuerySet []>

# Create three choices.
>>> q.choice_set.create(choice_text='Not much', votes=0)
<Choice: Not much>
```

13

More queries and model changes (3)

```
>>> q.choice_set.create(choice_text='The sky', votes=0)
<Choice: The sky>
>>> c = Choice(question=q, choice_text='Just hacking
again', votes=0)
>>> c.save()

# Choice objects have API access to their related
Question objects.
>>> c.question
<Question: What's up?>

# And vice versa: Question objects get access to Choice
# objects.
>>> q.choice_set.all()
<QuerySet [<Choice: Not much>, <Choice: The sky>,
<Choice: Just hacking again>]>
>>> q.choice_set.count()
3
```

14

More queries and model changes (4)

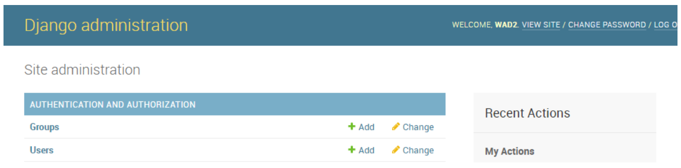
```
# Find all Choices for any question whose pub_date is
# in this year
>>> Choice.objects.filter(question__pub_date__year=
                          current_year)
<QuerySet [<Choice: Not much>, <Choice: The sky>,
<Choice: Just hacking again>]>

# Delete one of the choices
>>> c = q.choice_set.filter(choice_text__startswith=
                            'Just hacking')
>>> c.delete()
```

- For Rango, you will create a script to populate the database

The Django admin site

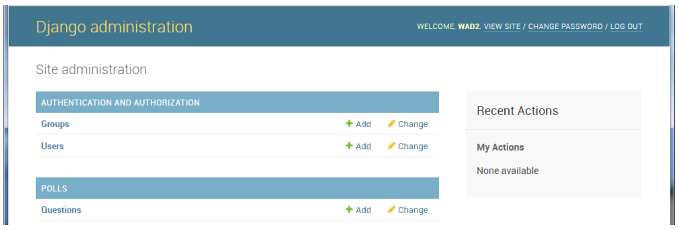
- Useful for managing your site content
- Create a superuser using `python manage.py createsuperuser`
- Now visit `https://<username>.pythonanywhere.com/admin/`



Make the polls app modifiable in the admin site

- **Add the following to `admin.py`:**

```
from .models import Question
admin.site.register(Question)
```



Django administration

WELCOME, WAD2. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups + Add ✎ Change

Users + Add ✎ Change

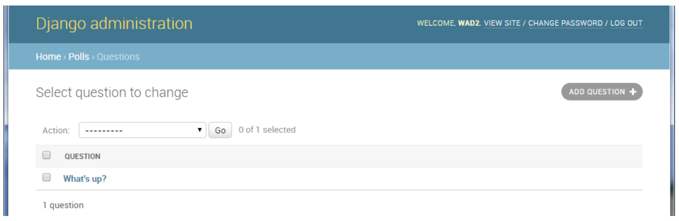
POLLS

Questions + Add ✎ Change

Recent Actions

My Actions

None available



Django administration

WELCOME, WAD2. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Polls › Questions

Select question to change

ADD QUESTION +

Action: --------- Go 0 of 1 selected

QUESTION

What's up?

1 question

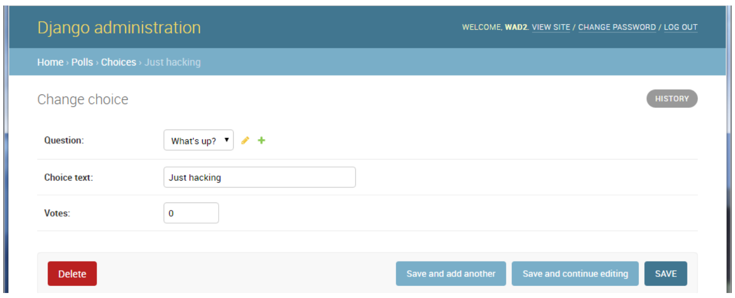
17

Make the polls app modifiable in the admin site

- **Change question text and publication date/time**
- **Add the following to `admin.py`:**

```
from .models import Choice
admin.site.register(Choice)
```

- **Add another choice to an existing question:**



Django administration

WELCOME, WAD2. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Polls › Choices › Just hacking

Change choice

HISTORY

Question:	What's up?
Choice text:	Just hacking
Votes:	0

Delete

Save and add another · Save and continue editing · SAVE

18

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
•Population scripts
•404 errors
•Removing hardcoded URLs
•Forms

19

The "Polls" application

• We will have the following views:

–Question "index" page – displays the latest few questions
–Question "detail" page – displays a question text, with no results but with a form to vote
–Question "results" page – displays results for a particular question
–Vote action – handles voting for a particular choice in a particular question

•In Django, web pages and other content are delivered by views

•Each view is represented by a simple Python function

•Django will choose a view by examining the URL that is requested

•To get from a URL to a view, Django uses 'URLconfs'
 – these map URL patterns to views

20

Writing more views

- Add the following code to `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)
```

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'),
]
```

Writing a view that does something

- Add the following code to `views.py`:

```
from .models import Question

def index(request):
    latest_questions = Question.objects.order_by
                            ('-pub_date')[:3]
    output = ', '.join([q.question_text for q in
                        latest_questions])
    return HttpResponse(output)
```

- Displays the latest 3 poll questions in the system, separated by commas, according to publication date

- Problem: page design is hard-coded in the view

- Solution: use a *template*

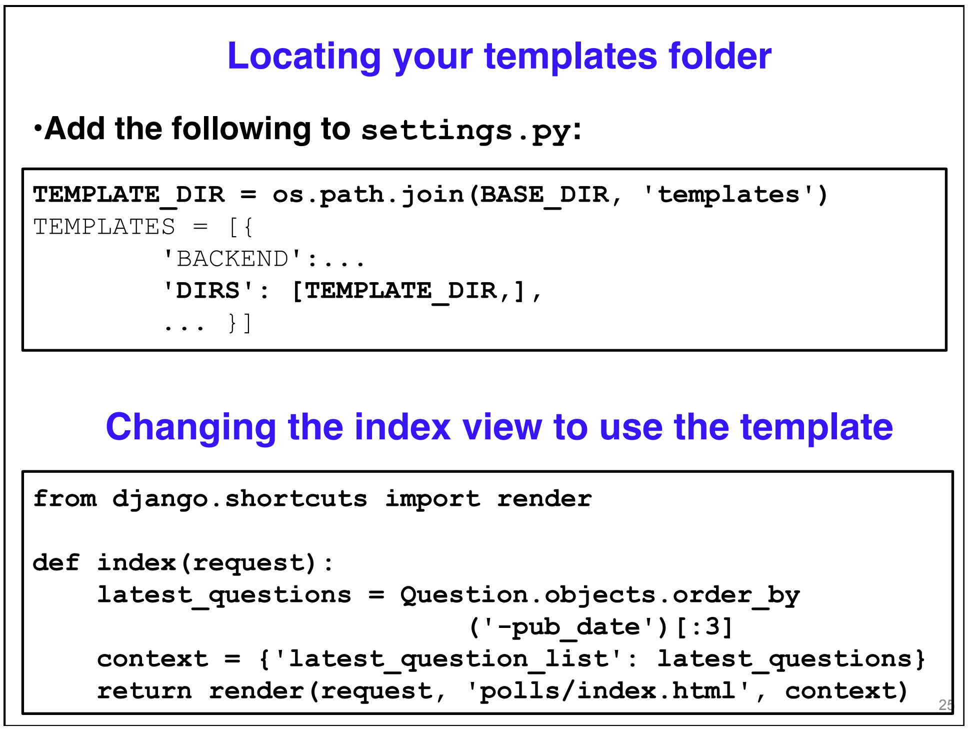
Writing a template for a view

•Create a `templates` folder within the `mysite` folder

•Create a `polls` sub-folder within the `templates` folder – to allow for multiple applications

•In that sub-folder, create a file `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 %}
```



Locating your templates folder

•**Add the following to `settings.py`:**

```
TEMPLATE_DIR = os.path.join(BASE_DIR, 'templates')
TEMPLATES = [{
        'BACKEND':...
        'DIRS': [TEMPLATE_DIR,],
        ... }]
```

Changing the index view to use the template

```
from django.shortcuts import render

def index(request):
    latest_questions = Question.objects.order_by
                           ('-pub_date')[:3]
    context = {'latest_question_list': latest_questions}
    return render(request, 'polls/index.html', context)
```

25