A quick look at

“Writing your first Django app”

Ralf Lämmel, http://softlang.wikidot.com/

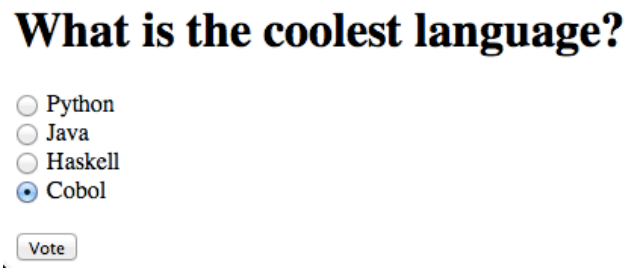
Version 9 *March 2014*

Acknowledgment: The step-to-to development section is entirely based on the Django documentation https://docs.djangoproject.com and specifically the tutorial “Writing your first Django app” (https://docs.djangoproject.com/en/1.6/intro/tutorial01/). The point of this document is to describe the development of the app in a concise manner, while also showing exactly what file edits need to be performed. This document does not, in any way, replace or compete with the Django documenation. In fact, you are much more likely to understand what you are doing, if you follow the original documentation. However, if you just want to really quickly get going and reproduce the file system of (some version of) the app, then this document may be helpful.

Table of Contents

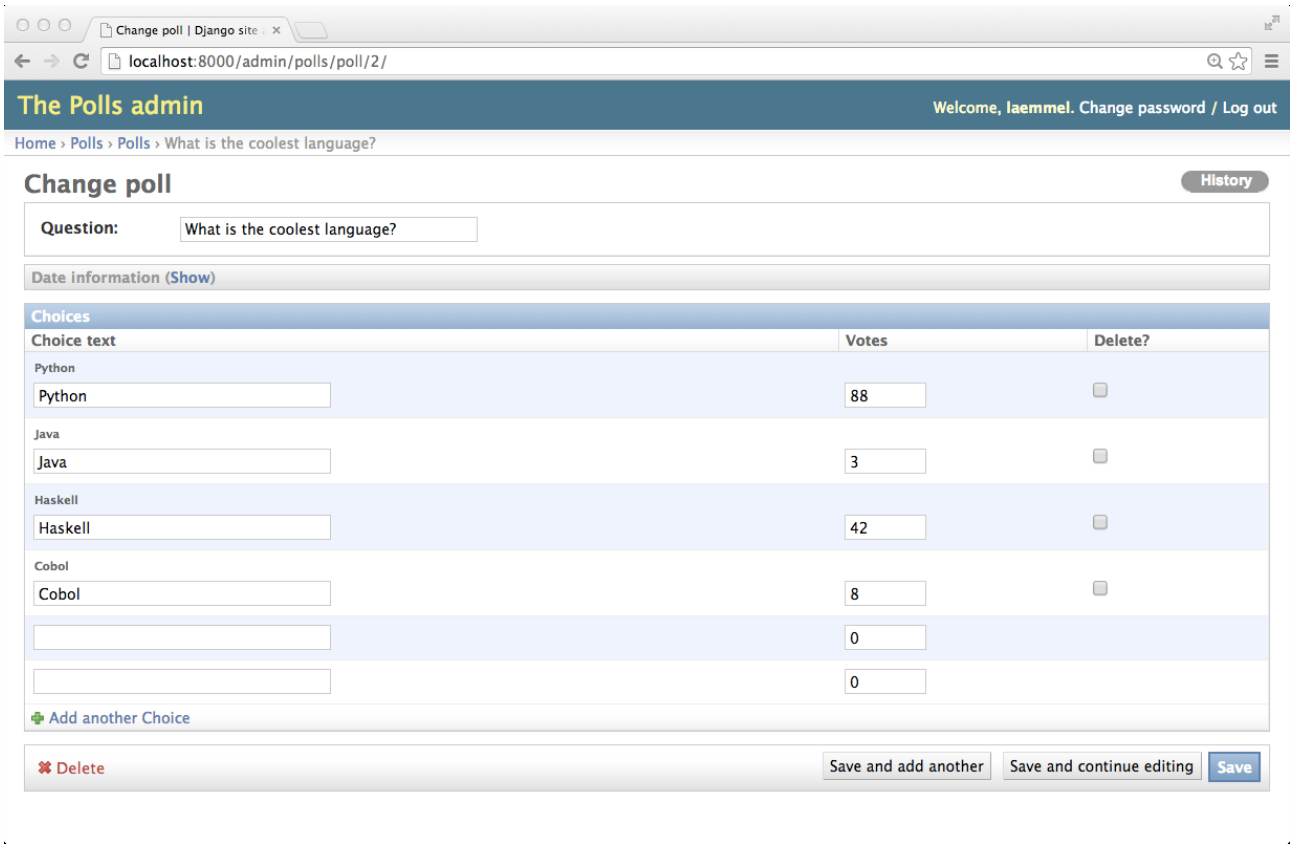
I. Preamble

Let's develop a polls application for polls like this:



Users would be participating in the poll and see the number of votes eventually.

Administrators (admins) would be setting up and editing up polls like this:



II. Conventions

- Code and program input as well as output in shown in Lucida Grande.
- File names are underlined.
- Code portions to be added are shown with yellow green background.
- Code portions also repeated as context are shown with yellow background.
- Program input is shown with red background.
- Program output is shown with gray background.
- Elisions (in repeating previous code or skipping output) are shown as "**...**".

We need to run commands at the command line ("bash" or alikes) all the time.
Typically, we run Python in batch or interactive mode.

III. Step-by-step development

1. Create src directory

At the command line, you run these commands:

```
mkdir src
cd src
```

2. Test Django installation

At the command line, you run these commands:

```
python --version
Python 2.7.5
python -c "import django; print(django.get_version())"
1.6.2
```

In what follows, these versions are assumed.
Differences are to be expected, if you use different versions.

3. Create a project

You are sitting in the src directory and run this command:

```
django-admin.py startproject mysite
```

This command may just finish silently and create various files. Have a look, perhaps.

4. Start server

At the command line, you run these commands:

```
cd mysite
python manage.py runserver

Validating models...

0 errors found
February 12, 2014 - 22:39:06
Django version 1.6.2, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

As this server runs in the current shell, you need another shell inside the mysite directory.

5. Prepare the database

At the command line, you run this command:

```
python manage.py syncdb
Creating tables ...
Creating table django_admin_log
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_groups
Creating table auth_user_user_permissions
Creating table auth_user
Creating table django_content_type
Creating table django_session

You just installed Django's auth system, which means you don't have any
superusers defined.
Would you like to create one now? (yes/no): yes
Username (leave blank to use 'laemmel'):
Email address: rlaemmel@gmail.com
Password:
Password (again):
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
```

6. Examine database catalog

Run the following command and query the database catalog as shown:

```
sqlite3 db.sqlite3
SQLite version 3.7.13 2012-07-17 17:46:21
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE "auth_group" (
    "id" integer NOT NULL PRIMARY KEY,
    "name" varchar(80) NOT NULL UNIQUE
);
CREATE TABLE "auth_group_permissions" (
...
```

7. Create an app

At the command line, you run this command:

```
python manage.py startapp polls
```

8. Edit the model

Put the following content into file polls/models.py.

```
from django.db import models

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

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

These are class fields describing the data model.

Add a method as well:

```
from django.db import models
import datetime
from django.utils import timezone

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    def was_published_recently(self):
        return self.pub_date >= timezone.now() - datetime.timedelta(days=1)
...
```

9. Install the app

Add 'polls' to INSTALLED_APPS in mysite/settings.py.

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'polls',
)
```

10. Examine SQL schema for model

At the command line, you run this command:

```
python manage.py sql polls
```

```
BEGIN;
CREATE TABLE "polls_poll" (
    "id" integer NOT NULL PRIMARY KEY,
    "question" varchar(200) NOT NULL,
    "pub_date" datetime NOT NULL
)
;
CREATE TABLE "polls_choice" (
    "id" integer NOT NULL PRIMARY KEY,
    "poll_id" integer NOT NULL REFERENCES "polls_poll" ("id"),
    "choice_text" varchar(200) NOT NULL,
    "votes" integer NOT NULL
)
;

COMMIT;
```

11. Upgrade the database

At the command line, you run this command:

```
python manage.py syncdb
Creating tables ...
Creating table polls_poll
Creating table polls_choice
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
```

12. Play with the API

Run the following command and enter Python code as shown:

```
python manage.py shell
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from polls.models import Poll, Choice
>>> Poll.objects.all()
[]
>>> from django.utils import timezone
>>> p = Poll(question="What's new?", pub_date=timezone.now())
>>> p.save()
>>> p.id
1
>>> p.question
```

```
"What's new?"
>>> p.pub_date
datetime.datetime(2014, 2, 12, 23, 11, 42, 979360, tzinfo=<UTC>)
>>> p.question = "What's up?"
>>> p.save()
>>> Poll.objects.all()
[<Poll: Poll object>]
```

13. Improve data representation

Edit file polls/models.py as shown:

```
from django.db import models
...

class Poll(models.Model):
    ...
    def __unicode__(self):  # Python 3: def __str__(self):
        return self.question

class Choice(models.Model):
    # ...
    def __unicode__(self):  # Python 3: def __str__(self):
        return self.choice_text
```

Accordingly, the data representation has changed:

```
python manage.py shell
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from polls.models import Poll, Choice
>>> Poll.objects.all()
[<Poll: What's up?>]
```

14. Register model with admin

Put the following content into file polls/admin.py:

```
from django.contrib import admin
from polls.models import Poll, Choice

admin.site.register(Poll)
admin.site.register(Choice)
```

Run the admin in the browser again. Polls and choices can be edited now.

15. Customize admin

Put the following content into file polls/admin.py:

```
from django.contrib import admin
from polls.models import Poll, Choice

class ChoiceInline(admin.TabularInline):
    model = Choice
    extra = 2

class PollAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,               {'fields': ['question']}),
        ('Date information', {'fields': ['pub_date'], 'classes': ['collapse']}),
    ]
    inlines = [ChoiceInline]
    list_display = ('question', 'pub_date', 'was_published_recently')
    list_filter = ['pub_date']
    search_fields = ['question']

admin.site.register(Poll, PollAdmin)
admin.site.register(Choice)
```

Add a few attributes to polls/models.py:

```
class Poll(models.Model):
    # ...
    was_published_recently.admin_order_field = 'pub_date'
    was_published_recently.boolean = True
    was_published_recently.short_description = 'Published recently?'
```

Try out the admin: http://127.0.0.1:8000/admin/

16. Customize templates of admin

Create a templates directory:

```
mkdir templates
```

Create a templates/admin directory:

```
mkdir templates/admin
```

Edit mysite/settings.py to register the directory:

```
TEMPLATE_DIRS = [os.path.join(BASE_DIR, 'templates')]
```

Figure out the location of the Django source files:

```
python -c "
import sys
sys.path = sys.path[1:]
import django
print(django.__path__)"
['/Library/Python/2.7/site-packages/django']
find /Library/Python/2.7/site-packages/django -name base_site.html
.../contrib/admin/templates/admin/base_site.html
```

Copy base_site.html to the local templates/admin directory:

```
cp .../contrib/admin/templates/admin/base_site.html templates/admin/
cp .../contrib/admin/templates/admin/index.html templates/admin/
```

Edit file templates/admin/base_site.html:

```
{% extends "admin/base.html" %}
{% load i18n %}

{% block title %}{{ title }} | {% trans 'Django site admin' %}{% endblock %}

{% block branding %}
<h1 id="site-name">{% trans 'The Polls admin' %}</h1>
{% endblock %}

{% block nav-global %}{% endblock %}
```

(We could also edit file templates/admin/index.html.)

17. Create a trivial view

Edit file polls/views.py:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world. You're at the poll index.")
```

Edit file polls/urls.py:

```
from django.conf.urls import patterns, url
from polls import views
```

```
urlpatterns = patterns('',
    # ex: /polls/
    url(r'^$', views.index, name='index')
)
```

Edit file mysite/urls.py:

```
from django.conf.urls import patterns, include, url
from django.contrib import admin

admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'mysite.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
    url(r'^polls/', include('polls.urls')),
)
```

Try out the view: http://localhost:8000/polls/

18. Organize all the views of the app

Add this code to polls/views.py:

```
def detail(request, poll_id):
    return HttpResponse("You're looking at poll %s." % poll_id)

def results(request, poll_id):
    return HttpResponse("You're looking at the results of poll %s." % poll_id)

def vote(request, poll_id):
    return HttpResponse("You're voting on poll %s." % poll_id)
```

Revise polls/urls.py as follows:

```
from django.conf.urls import patterns, url
from polls import views

urlpatterns = patterns('',
    # ex: /polls/
    url(r'^$', views.index, name='index'),
    # ex: /polls/5/
```

```
    url(r'^(?P<poll_id>\d+)/$', views.detail, name='detail'),
    # ex: /polls/5/results/
    url(r'^(?P<poll_id>\d+)/results/$', views.results, name='results'),
    # ex: /polls/5/vote/
    url(r'^(?P<poll_id>\d+)/vote/$', views.vote, name='vote'),
)
```

19. Implement a view

Replace the content of polls/views.py with the following content:

```
from django.http import HttpResponse
from polls.models import Poll

def index(request):
    latest_poll_list = Poll.objects.order_by('-pub_date')[:5]
    output = ', '.join([p.question for p in latest_poll_list])
    return HttpResponse(output)
...
```

Arguably the view is rendered poorly; consider this:

```
Who are you?, What's new?
```

20. Implement a view with a template

We are going to reimplement the index view.
Create a directory for the templates of the app:

```
mkdir templates/polls
```

Put the following text into file templates/polls/index.html:

```
{% if latest_poll_list %}
    <ul>
    {% for poll in latest_poll_list %}
        <li><a href="/polls/{{ poll.id }}/">{{ poll.question }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```

Edit file polls/views.py to revise the definition of index specifically:

```
from django.http import HttpResponse
from polls.models import Poll
from django.shortcuts import render

def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    context = {'latest_poll_list': latest_poll_list}
    return render(request, 'polls/index.html', context)
...
```

21. Implement a view with a 404 error

Put the following text into file templates/polls/detail.html:

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

Edit file polls/views.py to revise the definition of detail specifically:

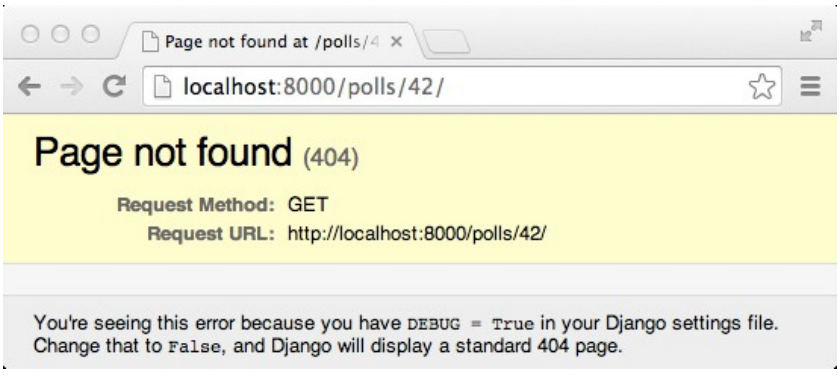
```
...
from django.http import Http404

def index(request):
    ...

def detail(request, poll_id):
    try:
        poll = Poll.objects.get(pk=poll_id)
    except Poll.DoesNotExist:
        raise Http404
    return render(request, 'polls/detail.html', {'poll': poll})
...
```

You are ready to render a question: http://localhost:8000/polls/2/

Here is how a missing poll is rendered:



22. Implement a view with a form

Put the following text into file templates/polls/detail.html:

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

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

<form action="{% url 'vote' poll.id %}" method="post">
{% csrf_token %}
{% for choice in poll.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 %}
<p/>
<input type="submit" value="Vote" />
</form>
```

Participation in the poll is rendered as it was shown in the preamble of this document.

23. Implement a view as a handler

Edit file polls/views.py to revise the definition of vote specifically:

```
from django.shortcuts import get_object_or_404, render
from django.http import HttpResponseRedirect, HttpResponse
from django.core.urlresolvers import reverse
from polls.models import Choice, Poll
...
def vote(request, poll_id):
    p = get_object_or_404(Poll, pk=poll_id)
    try:
        selected_choice = p.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        # Redisplay the poll voting form.
        return render(request, 'polls/detail.html', {
            'poll': p,
            '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=(p.id,)))
```

Put the following text into file templates/polls/results.html:

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

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

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

Edit file polls/views.py to revise the definition of results specifically:

```
def results(request, poll_id):
    poll = get_object_or_404(Poll, pk=poll_id)
    return render(request, 'polls/results.html', {'poll': poll})
```

The result of the vote is rendered as it was shown in the preamble of this document.