Django: MVC web framework in Python (Win)

You can find this digital handout and the slides online at www.marcolussetti.com/workshops/django2017

Editor used by the speaker: Atom by Github, available at www.atom.io. Packages installed: autocomplete-python, language-python, python-indent, atom-django, django-templates, kite [minimap, file-icons].

You could also use Pycharm by JetBrains (free for students at https://www.jetbrains.com/pycharm/download). The speaker will go over the setup for a Django project in PyCharm quickly at the beginning of the tutorial section.

Setup a Python3 virtual environment

Windows

Download Python 3.6.1 from https://www.python.org/ftp/python/3.6.1/python-3.6.1.exe

Tick the box that says "Add Python 3.6 to PATH"

Open the Windows Command Prompt ("cmd").

Note: :: (double colon) indicates a comment, and should be ignored. In future boxes, the linux comment # will be used, just think of it as a double colon. Also, when the comments differ between versions (namely dir on windows and ls on linux/mac), I have tried to note so in the comment.

```
::Initial setup of virtualenvwrapper
pip install virtualenvwrapper-win :: installs virtualenvwrapper
mkvirtualenv djangoev :: creates new virtual environment
:: workon djangoenv :: This is to work on an existing virtual environment, not needed
the first time
```

Linux (Python3 default)

Assumes Python3 is already installed.

Open the Terminal/Console

```
#Initial setup of virtualenvwrapper
#Install virtualenvwrapper globally with sudo, or use pip install --user
virtualenvwrapper
sudo pip install virtualenvwrapper  # installs virtualenvwrapper
mkvirtualenv djangoev  # creates new virtual environment
# workon djangoenv # This is to work on an existing virtual environment, not needed
the first time
```

Ubuntu (Python2.7 default)

Open the Terminal/Console

```
#Install pip for Python3
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python3-pip

#Initial setup of virtualenvwrapper
sudo -H pip3 install virtualenvwrapper  # installs virtualenvwrapper
VIRTUALENVWRAPPER_PYTHON =/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
mkvirtualenv djangoev  #creates new virtual environment
#workon djangoenv # This is to work on an existing virtual environment
```

Create a Django project

```
pip install django  # installs django to the virtual environment
pip freeze > requirements.txt  # Save the requirements
# pip install -r requirements.txt #To install all requirements
django-admin startproject djangoproject  # create a new django project
cd djangoproject  # enter the directory of the project
ls # dir on windows # Let's look at the list of things we have in the directory
ls djangoproject  # dir on windows # continued from above
python manage.py startapp djangoapp  # create an inner app
ls djangoapp # dir on windows # let's look at contents of the app
python manage.py runserver  # Let's start the development http server
```

To look at your new "project", go to http://localhost:8000

To close the server and continue working on this, use Ctrl-C

Git

If you wish to use git version control for this seminar, you can use https://www.gitignore.io/api/django to generate a decent .gitignore file

Simple HTTP response

In **djangoapp/views.py**, add lines to the following effect

```
from django.html import HttpResponse
def index(request):
    return HttpResponse("Welcome to the Django Test App" )
```

In **djangoapp/urls.py**

```
from django.conf.urls import url
from . import views

urlpatterns = [
  url(r'^$', views.index, name="index"), #Connect http://localhost:8000/djangoapp with
the index view
]
```

In **djangoproject/urls.py**

```
#Add include to the imports from django.conf.urls
from django.conf.urls import url, include

urlpatterns = [
  url(r'^djangoapp/', include('djangoapp.urls')), #Include the urls from the djangoapp
app
]
```

Now if we just go to http://localhost:8000 we'll get a 404, and we'll get a list of the URL patterns Django tried to match our request to as we are in debug mode. Let's try http://localhost:8000/djangoapp

Models & Database

To change the type of database, edit **djangoproject/settings.py**, but we'll keep sqlite3 for now.

Let's look at models.

```
from django.db import models

#Create a new feedback class (think of it as a table if you need to)
class Feedback(models.Model):
    name = models.CharField(max_length=254, help_text="Name of the user")
    email = models.EmailField(max_length=254, help_text="Email address")
    date = models.DateTimeField(auto_now_add=True) #Add this later as the migration
for it is pretty cool
    ip = models.GenericIPAddressField(protocol='both', blank=True) #protocol='both'
allows for ipv4 and ipv6
    feedback = models.TextField(null=True, blank=True)
```

Add the app to the **djangoproject/settings.py**

```
INSTALLED_APPS = [
    'djangoapp', #Add the new djangoapp
    'django.contrib.admin',
]
```

Run migrations

```
python manage.py makemigrations djangoapp  # creates migrations
python manage.py sqlmigrate djangoapp  0001 # shows SQL output of the migration (based
on DB)
python manage.py migrate  # actually carries out the migrations
```

Testing the interactive shell `python manage.py shell`

```
from djangoapp.models import Feedback
Feedback.objects.all() # But we have no values yet!
new_feedback = Feedback(name="Marco Lussetti", email="marco@marcolussetti.com",
ip="127.0.0.1")
new_feedback # Doesn't really return anything useful, we'll see soon how to fix this
new_feedback.save() # Save new feedback to database
Feedback.objects.all() # Note it now returns a value
```

Now to clarify what the object is like, let's edit the model at **djangoproject/models.py** again

```
from django.db import models

class Feedback(models.Model):
    name = models.CharField(max_length=254, help_text="Name of the user")
    email = models.EmailField(max_length=254, help_text="Email address")
    date = models.DateTimeField(auto_now=True, auto_now_add=True)
    ip = models.GenericIPAddressField(protocol='both', blank=True)
    feedback = models.TextField(null=True, blank=True)

    # Return the value of the name when queried
    def __str__(self):
        return self.name
```

In the `python manage.py shell`

```
from djangoapp.models import Feedback
Feedback.objects.all() # Note that it returns the value of name because that's what we
defined as __str__ in the Model
#Let's add another feedback item
Feedback(name="Marco Rossetti", email="marco@marcolussetti.com", ip="127.0.0.2",
feedback="Terrible form, it doesn't even exist yet!").save()
Feedback.objects.all() # Should show both objects
Feedback.objects.filter(ip="127.0.0.1")
```

Django Admin Interface

`python manage.py createsuperuser`

After creating the user, run the server with `python manage.py runserver` and browse to http://localhost:8000/admin and log in.

You won't be able to see the feedback items, so let's add that to **djangoapp/admin.py**

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

Now you can check and it will show the data, but you will just see the name. Note that if you edit an item, it shows edits to it if done from the admin interface if you click on history in the details view. Let's make it more useful display-wise.

Edit **djangoapp/admin.py**

```
from .models import Feedback

class FeedbackAdmin(admin.ModelAdmin):
    search_fields = ['name', 'email']
    list_display = ['name', 'ip', 'email', 'date', 'get_feedback']


    def get_feedback(self, obj):
        return obj.name
    get_feedback.admin_order_field = 'date'
admin.site.register(Feedback, FeedbackAdmin)
```

Writing actual (or semi-actual) views

Create some views

Let's add a view that shows a get parameter using pretty urls

djangoapp/views.py

```
def feedback(request, feedback_id):
    return HttpResponse("You are looking at feedback for item %s."  % feedback_id)
```

djangoapp/urls.py

```
urlpatterns = [
    url(r'^$', views.index, name="index"),
    url(r'^(?P<feedback_id>[0-9]+)/$' , views.feedback, name='feedback'), #regex stands
for any digit
]
```

Let's now add an index that shows some data from the database: the names of the last 3 users that submitted feedback.

djangoapp/views.py

```
from .models import Feedback
def index(request):
    latest_feedbacks = Feedback.objects.order_by("-date")[:3] #Print the last 3 items
by date, descending
    output = ", ".join([f.name for f in latest_feedbacks]) #join them into a string
separated by commas
    return HttpResponse(output)
```

Templates at long last!

Create the folder templates inside djangoapp

djangoapp/templates/index.html

```
{% if latest_feedback_list %}
    <ol>
        {% for feedback in latest_feedback_list %}
            <li><a href="/djangoapp/{{ feedback.id }}/">{{ feedback.name }}</a></li>
        {% endfor %}
    </ol>
{% else %}
    <p>No feedback received</p>
{% endif %}
```

djangoapp/views.py

```
from django.shortcuts import render

def index(request):
    latest_feedbacks = Feedback.objects.order_by("-date")[:3] #Print the last 3 items
by date, descending
    context = {'latest_feedback_list' : latest_feedbacks}
    return render(request, 'index.html', context)
```

Let's also add a 404 error for items that do not exist

djangoapp/views.py

```
from django.http import Http404

def feedback(request, feedback_id):
    try:
        feedback_item = Feedback.objects.get(id=feedback_id)
    except Feedback.DoesNotExist:
        raise Http404("Feedback item does not exist!")
    return render(request, 'feedback.html', {'feedback' : feedback_item})

#Shortcut syntax, use above or below
from django.shortcuts import get_object_or_404
def feedback(request, feedback_id):
    feedback_item = get_object_or_404(Feedback, id=feedback_id)
    return render(request, 'feedback.html', {'feedback' : feedback_item})
```

djangoapp/templates/feedback.html (temporary template)

```
<h1>{{ feedback.name }}</h1>
<h2>Date: {{ feedback.date }}</h2>
<p>Contents: {{feedback.feedback}} </p>
```

Creating a form to receive feedback

djangoapp/forms.py

```
from django import forms

class FeedbackForm(forms.Form):
    name = forms.CharField(label='Name:', max_length=240)
    email = forms.EmailField(label="Email:", max_length=240)
    ip = forms.GenericIPAddressField()
    feedback = forms.CharField(widget=forms.Textarea, required=False)
```

djangoapp/urls.py

```
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^$', views.index, name="index"),
    url(r'^(?P<feedback_id>[0-9]+)/$' , views.feedback, name="feedback"),
    url(r'^form/$', views.form, name="form"),
]
```

djangoapp/views.py

```
from .forms import FeedbackForm

def form(request):
    if request.method == 'POST':
        form = FeedbackForm(request.POST)
        if form.is_valid():
            new_feedback = Feedback(name=form.cleaned_data['name'],
                                    email=form.cleaned_data['email'],
                                    ip=form.cleaned_data['ip'],
                                    feedback=form.cleaned_data['feedback'])
            new_feedback.save()
            return HttpResponse("<h1>Thank you for your feedback</h1>" )
    else:
        form = FeedbackForm()
    return render(request, 'form.html', {'form': form})
```

djangoapp/templates/form.html

```
<form action="/djangoapp/form/"  method="post">
    {% csrf_token %}
        <table>
        {{ form }}
        </table>
    <input type="submit" value="Submit" />
</form>
```