Servicios y Aplicaciones Telemáticas (2011-12)

Ingeniería de Telecomunicación (URJC)

Jesus M. González Barahona, Gregorio Robles Martínez

jgb@gsyc.es
http://identi.ca/jgbarah http://twitter.com/jgbarah
GSyC, Universidad Rey Juan Carlos

Septiembre 2011

Universidad

Prácticas: Introducción a Django

Enfoques comunes de desarrollo web

- Frameworks de desarrollo web
  - PHP, JavaEE, Python+HttpServer...
- Entornos de desarrollo web completos
  - Django (Python), `http://djangoproject.org`
  - Ruby on Rails (Ruby), `http://rubyonrails.org/`
  - CakePHP (PHP), `http://cakephp.org/`
  - Grails (Groovy, sovre JVM), `http://grails.org/`
  - RIFE (Java), `http://rifers.org/`
- Plataformas extensibles
  - CMS: Joomla, Drupal...
  - Portal: Plone/Zope, Liferay Portal...
  - Plataformas de propósito específico: Moodle, Wordpress...

¿Qué es Django?

- Entorno integrado de desarrollo de aplicaciones web
- Herramientas para gestionar la aplicación
- Framework (armazón) para presentación de la aplicación
- Acceso a base de datos (correspondencia objeto-relacional)
- Seguridad (XSS, SQL Injection, ...)
- Componentes listos para usar (gestión de usuarios, sesiones, interfaz administración,...)
- Cache, internacionalización, plantillas, etc.

`http://docs.djangoproject.com/en/dev`

Django: conceptos principales

- Objetivo principal: desarrollo muy rápido
  - Entorno integrado y completo
  - Cambios en caliente
  - Descripciones de error muy descriptivas
  - Convenciones preferible a configuración
  - Evitar duplicación a toda costa (DRY, don't repeat yourself)
- Desarrollo dirigido por el modelo
  - Se comienza por el diseño del modelo de datos

Preparativos

- Usaremos la versión 1.3.1
- Disponible para Linux, *BSD, Windows, MacOS, etc.
- Descarga e instalación en $DJANGO

  `tar xvzf Django-1.3.1.tar.gz`
- Preparación de entorno (necesario si no se ha instalado en path "habituales"):

  ```
  export PATH=$DJANGO/django/bin:$PATH
  export PYTHONPATH=$DJANGO:$PYTHONPATH
  ```
- Comprobación:

  `django-admin.py --version`

`http://docs.djangoproject.com/en/dev/topics/install/#installing-an-official-release`

Armazón para proyecto y aplicación

- Creación (primero proyecto, luego aplicación)

```
$ cd dir-practica
$ django-admin.py startproject myproject
$ cd myproject
$ python manage.py startapp myfirstapp
```

- Más opciones de manage.py

```
$ python manage.py --help
```

- Ejecución de la aplicación (`http://localhost:1234`)

```
$ python manage.py runserver 1234
```

Ficheros creados en el armazón

- Proyecto:
  - `__init__.py`: fichero vacío, directorio debe ser considerado un paquete Python
  - `manage.py`: herramienta para gestionar el proyecto
  - `settings.py`: configuración del proyecto
  - `urls.py`: URLs de las aplicaciones del proyecto
- Aplicación:
  - `models.py`: definición de las clases del modelo de datos
  - `views.py`: vistas (código invocado para cada recurso)

Fichero settings.py

- Fichero de configuración, en Python
- Configuración de la base de datos (usaremos SQLite3)

```
ENGINE = 'django.db.backends.sqlite3'
NAME = 'myproject.sqlite'
USER = ''
PASSWORD = ''
HOST = ''
PORT = ''
```

- Aplicaciones instaladas

```
INSTALLED_APPS = (
    'myproject.myfirstapp',
)
```

- Otros: zona horaria, codificación, directorio de plantillas...

Declaración de urls

- En el fichero urls.py
- Usa expresiones regulares para asociar URLs (sin parámetros) a vistas
- Ejemplo:

```
urlpatterns = patterns('',
  url(r'^$',
      'myproject.myfirstapp.views.say_main',),
  url(r'^hello',
      'myproject.myfirstapp.views.say_hello',),
  url(r'^bye/(.*)',
      'myproject.myfirstapp.views.say_bye_to',),
  url(r'^number/(?P<number>[\d]+)',
      'myproject.myfirstapp.views.say_number',),
```

Views

- Código invocado para una URL o conjunto de URLs
- Debe ser un método (o un objeto)
- Los métodos se definen en el fichero myfirstapp/views.py
- Ejemplo:

```
from django.http import HttpResponse
def say_main(request):
    return HttpResponse('<h1>My Application</h1>')
def say_hello(request):
    return HttpResponse('Hello!')
def say_bye_to(request, name):
    return HttpResponse('Bye %s'%name)
def say_number(request, number=0):
    return HttpResponse('Number: %s'%number)
```

Gestión de datos persistentes

- Django hace corresponder un objeto Python con cada tabla
- Cada aplicación tiene su models.py
  - Una clase por cada entidad (tabla) del modelo
  - Un campo por cada dato (columna) de la entidad
  - Ejemplo:

```
class MyFirstAppData(models.Model):
    name = models.CharField(max_length=200)
    birthday = models.DateTimeField()
```

- Creación de tablas

```
$ python manage.py syncdb
```

Definición del modelo

- Tipos de campos:
  - CharField(maxlength)
  - TextField()
  - IntegerField()
  - DateField()
  - BooleanField()
- Relaciones:
  - ForeignKey(othermodel)
  - ManyToManyField('self', symmetrical=False)

`http://docs.djangoproject.com/en/dev/ref/models/fields/`

Otras acciones de gestión del proyecto

- Ejecución en el contexto de Python con acceso al código de la aplicación

  ```
  % python manage.py shell
  ```

- Validacion de modelos de datos

  ```
  % python manage.py validate
  ```

- Exportación de datos de la base de datos

  ```
  % python manage.py dumpdata
  ```

- Importación de datos en la base de datos

  ```
  % python manage.py loaddata
  ```

Consultas a la base de datos

- Métodos para realizar consultas a la base de datos
- Acceso a entradas de la base de datos mediante el objeto 'objects'
  (ej. MyFirstAppData.objects)
- Métodos:
  - MyFirstAppData.save()
  - MyFirstAppData.objects.all()
  - MyFirstAppData.objects.filter(campo=valor)
  - MyFirstAppData.objects.get(campo=valor)
    Excepción si no lo encuentra

La shell de Django

Acceso a la API de los objetos de nuestro proyecto

```
% python manage.py shell
>>> from myproject.myfirstapp.models import MyFirstAppData
>>> MyFirstAppData.objects.all()
[]
>>> p = MyFirstAppData(name="Jesus",
                       birthday="2009-05-05")
>>> p.save()
>>> p.id
1
>>> MyFirstAppData.objects.filter(name="Jesus")
...
>>> MyFirstAppData.objects.get(pk=1).name
```

Acceso al modelo desde las vistas

- Las vistas pueden usarse para leer y modificar al modelo

```
from django.http import HttpResponse,HttpResponseNotFou
from content.models import Pages

def show_content(request, resource):
    try:
        record = Pages.objects.get(name=resource)
        return HttpResponse(record.page)
    except Pages.DoesNotExist:
        return HttpResponseNotFound(
            'Page not found: /%s.' % resource
            )
```

Usuarios

- INSTALLED_APPS (en settings.py) ha de incluir:
  - django.contrib.auth
  - django.contrib.contenttypes
- Hay que crear las tablas pertinentes (manage.py syncdb)

Admin site

Versión simple:

- INSTALLED_APPS (en settings.py) ha de incluir:
  - django.contrib.admin
  - django.contrib.sessions (dependencia del anterior)
  - ...y lo necesario para usuarios
- Hay que crear las tablas pertinentes (manage.py syncdb)
- Enganche en urls.py

```
from django.contrib import admin
admin.autodiscover()
...
(r'^admin/', include(admin.site.urls)),
```

Admin site (2)

Ahora, proporcionemos interfaz para nuestra tabla Pages:

- Crea en el directorio de la aplicación Django de gestión de contenidos el fichero admin.py
- Registra en él los modelos a manejar:

```
from django.contrib import admin
from cms_users.content.models import Pages

admin.site.register(Pages)
```

- Prueba que ahora puedes manejar esta tabla desde el sitio de administración

Login

- Utilizamos view predefinida (entiende GET y POST)
- En urls.py:

```
url(r'^login', 'django.contrib.auth.views.login'),
```

- Necesita una plantilla registration/login.html:
  - En settings.py:

    ```
    TEMPLATE_DIRS = ('templates')
    ```

  - Creación de templates/registration/login.html

Info detallada: "User authentication in Django"

templates/registration/login.html

```
<html><body>
<form method="post" action="/login">
<table>
  <tr><td>Username</td>
      <td>{{ form.username }}</td></tr>
  <tr><td>Password</td>
      <td>{{ form.password }}</td></tr>
</table>
<input type="submit" value="login" />
</form>
</body></html>
```

Acceso a información de usuario y logout

- Accedemos a información del objeto User, que tenemos en HTTPRequest
- En views.py:

```
def show_content(request, resource):
  if request.user.is_authenticated():
    logged = 'Logged in as ' + request.user.username
  else:
    logged = 'Not logged in.'
```

- Para logout, utilizamos view predefinida. En urls.py:

```
(r'^logout', 'django.contrib.auth.views.logout'),
```

Plantillas (templates)

- Ficheros de texto que pueden generar cualquier formato basado en texto (HTML, XML, CSV, etc.)
- Contienen:
  - Texto (que queda igual)
  - Variables (reemplazadas por su valor cuando se evalúan)
  - Filtros (modifican variables cuando se evalúan)
  - Etiquetas (controlan la lógica de la evaluación de la plantilla)
  - Comentarios {# Comentario #}
- Pueden extender (heredar de) otras plantillas
- Se colocan en los directorios de plantillas (TEMPLATE_DIRS en settings.py)

Plantillas: variables y filtros

- Variables:

```
{{ variable }}
```

- Filtros:

```
{{ variable|filtro|otrofiltro }}
```

- Filtro con argumentos:

```
{{ variable|filtro:30 }}
```

- Ejemplos de filtros:

```
{{ value|default:"nothing" }}
{{ value|length }}
{{ text|striptags }}
{{ text|truncatewords:30 }}
{{ text|escape|linebreaks }}
{{ list|join:", " }}.
```

Plantillas: etiquetas

- for

```
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% endfor %}
```

- if

```
{% if athlete_list %}
    Number of athletes: {{ athlete_list|length }}
{% else %}
    No athletes.
{% endif %}
```

Plantillas: etiquetas (2)

- ifequal, ifnotequal

```
{% ifequal athlete.name coach.name %}
    ...
{% endifequal %}
{% ifnotequal athlete.name "Joe" %}
    ...
{% endifnotequal %}
```

- block, extends: Herencia

Ejemplo de plantilla

```
{% extends "base.html" %}
{% block title %}{{ section.title }}{% endblock %}
{% block content %}
<h1>{{ section.title }}</h1>
{% for story in story_list %}
<h2>
  <a href="{{ story.get_absolute_url }}">
    {{ story.headline|upper }}
  </a>
</h2>
<p>{{ story.tease|truncatewords:"100" }}</p>
{% endfor %}
{% endblock %}
```

Plantillas: uso en vistas

Directorios con plantillas: TEMPLATE_DIRS en settings.py

```
from django.template.loader import get_template
from django.template import Context

def show_annotated_content(request, resource):
    ...
    template = get_template("annotated.html")
    return HttpResponse(template.render(
      Context({'user': user,
               'resource': resource,
               'page': page})))
```

Plantillas: uso en urls.py

```
from django.conf.urls.defaults import *
from django.views.generic.simple import direct_to_template

urlpatterns = patterns('',

    url(r'^about$', direct_to_template, {
        'template': 'about.html'
    }),
)
```

Modelos: relación muchos a uno (ForeignKey)

```
class Manufacturer(models.Model):
    # ...
class Car(models.Model):
    manufacturer = models.ForeignKey(Manufacturer)
    # ...

# Creating
m = Manufacturer(name='Seat')
c = Car(name='Toledo')
m.save(); c.save()
# Relationship
c.manufacturer = m
# Obtaining
```

Modelos: relación muchos a muchos (ManyToManyField)

```
class Topping(models.Model):
    # ...
class Pizza(models.Model):
    # ...
    toppings = models.ManyToManyField(Topping)
```

Modelos: relación muchos a muchos (ManyToManyField) (2)

```
pb = Pizza(name='Barbecue')
pq = Pizza(name='4 Cheese')
b = Topping(name='Barbecue sauce')
m = Topping(name='Mozzarela')
pb.save(); pp.save(); b.save(); m.save()
pb.toppings.add(b, m)
pq.toppings.add(m)
pq.toppings.create(name='Rochefort')
m.pizza_set.all()
pb.toppings.all()
Pizza.objects.filter(toppings__name='Mozzarela')
```

Ficheros estáticos con Django

- Los ficheros estáticos no se deberían servir con Django...
- (lo hace mucho mejor un servidor web como Apache o Cherokee)
- ...pero se pueden servir
- django.views.static.serve()

```
(r'^css/(?P<path>.*)$', 'django.views.static.serve',
        {'document_root': 'sfiles/css'}),
```

Generador de canales

- Django viene con módulos para generar canales RSS y Atom
- View de alto nivel que genera el canal (feed):

```
(r'^feeds/(?P<url>.*)/$', 'django.contrib.syndication.v
    {'feed_dict': feeds}),
```

- Hay que proporcionar un diccionario con la correspondencia canal a objeto Feed:

```
feeds = {
    'latest': LatestEntries,
    'categories': LatestEntriesByCategory,
}
```

Generador de canales: objetos Feed

- Representan los datos de un canal:

```
from django.contrib.syndication.feeds import Feed
from content.models import Pages

class LatestEntries(Feed):
    title = "My CMS contents"
    link = "/feed/"
    description = "Contents of my CMS."

    def items(self):
        return Pages.objects.order_by('-pub_date')[:5]
```

- También hay que definir plantillas (templates) para <title> y <description> de cada item del canal RSS

Internacionalización

- Cadenas de traducción en código Python

```
from django.utils.translation import ugettext as _

def my_view(request):
    output = _("Welcome to my site.")
    return HttpResponse(output)

def my_view(request, m, d):
    output = _('Today is %(month)s, %(day)s.') %
        {'month': m, 'day': d}
    return HttpResponse(output)
```

Internacionalización (2)

- Cadenas de traducción en plantillas

```
<title>{% trans "This is the title." %}</title>

{% blocktrans %}
This string will have {{ value }} inside.
{% endblocktrans %}
```

Internacionalización (3)

- Traducciones en los lenguajes requeridos

```
django-admin.py makemessages -l es
```

- Activar el soporte para locale en Django

```
MIDDLEWARE_CLASSES = (
 'django.contrib.sessions.middleware.SessionMiddleware'
 'django.middleware.locale.LocaleMiddleware',
 'django.middleware.common.CommonMiddleware',
)
```

Referencias

- Documentación de Django
  `http://docs.djangoproject.com/en/dev`
- Libro de Django
  `http://www.djangobook.com`
- Documentación de Python
  `http://www.python.org/doc/`
- Tutorial sobre Django (e introducción a Django)
  `http://docs.djangoproject.com/en/dev/intro`