Django Framework
Interfaces Web

José Ortiz Bejar

job@correo.fie.umich.mx

Universidad Michoacana de San Nicolás de Hidalgo

16 de septiembre de 2014

Interfaz Admin

Interfaz Pública

Sitio de administración

El objetivo es generar una interface para la administración del sitio, donde se pueda agregar, modificar o borrar contenido del sitio. Django proporciona una interfaz unificada para el/los administradores del sitio.

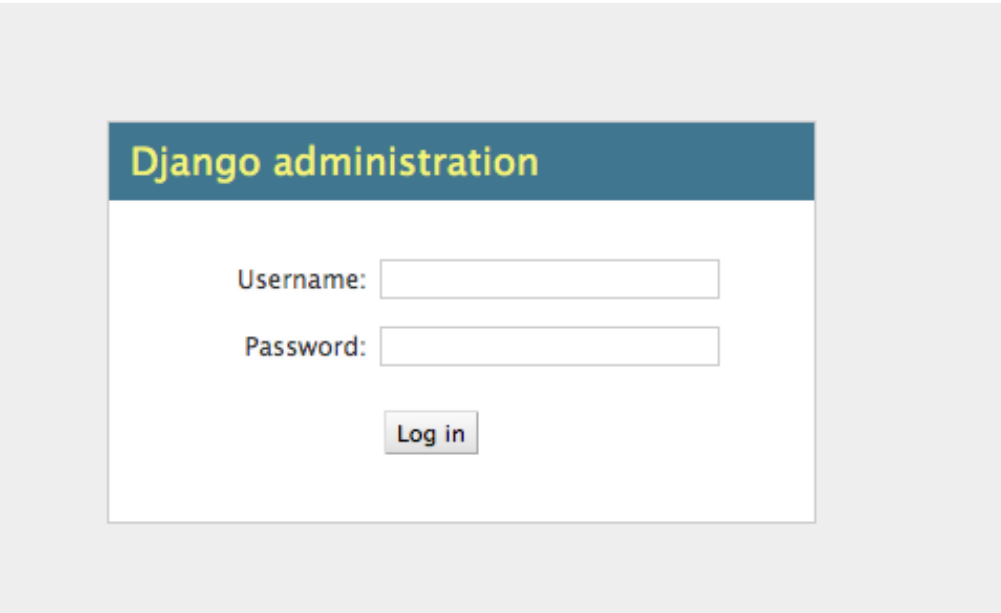
Activar el sitio de administración

Está deshabilitado por defecto. Para habilitarlo hay que hacer tres cosas:

- Editar *settings.py y descomentar* **django.contrib.admin** *en* **INSTALLED_APP**
- *Ejecutar* **pyhon manage.py syndb**
- *Editat* urls.py y descomentar las siguientes lineas **from django.contrib import admin admin.autodiscover() url(r'^admin/', include(admin.site.urls)),**

Probar el sitio de administración

- **python manage.py runserver**
- Abrir la url http://127.0.0.1/admin/
- Proporcionar el usuario y password que generamos cuando se ejecuto syndb por primera vez. Sino se hizo ejecutar :
- **python manage.py createsuperuser**



Administración Django

- Una vez inciada sesión, se pueden observar algunos tipos de contenido editable, incluyen grupos, usuarios y sitios.

Agregar el app encuestas a admin

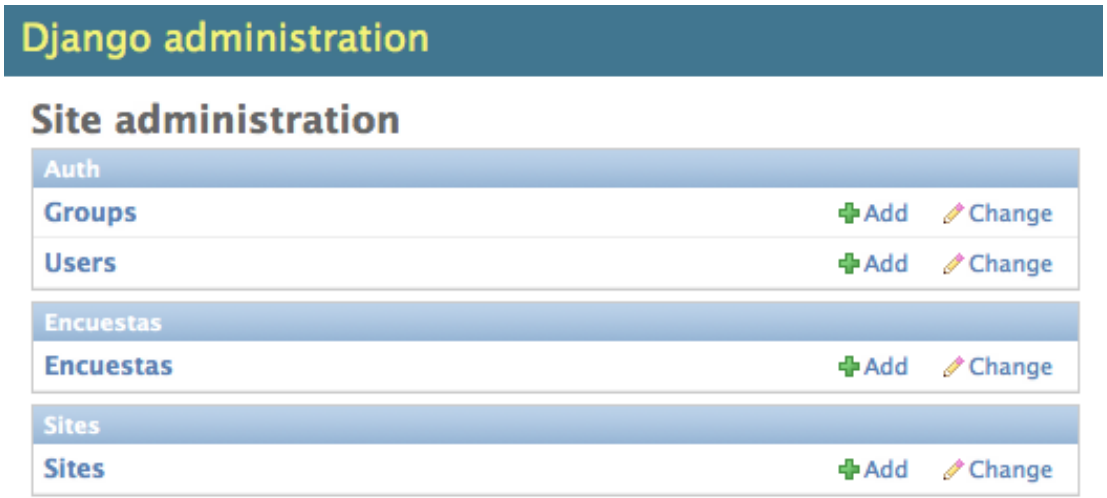
- Encuestas no se despliega en el sitio de administración
- Podemos generar una interface en admin para encuestas haciendo lo siguiente:
  - Crear un archivo *admin.py en el directorio encuestas.*
  - *Agregar el siguiente contenido:*

```
from encuestas.models import Encuesta
from django.contrib import admin

admin.site.register(Encuesta)
```

  - *Reiniciar el servidor*

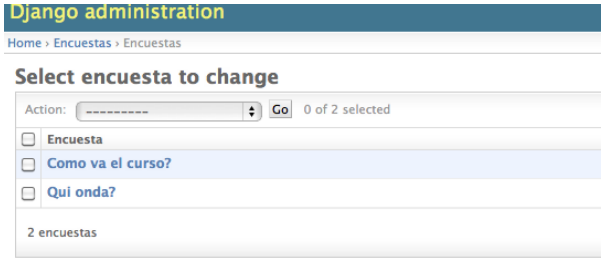
Explorar la funcionalidad

- Ahora ya aparece *encuestas en el index de admin.*



Explorar la funcionalidad

- Seleccionando "Encuestas"



- y una de las encuestas ya en la DB

Explorar la funcionalidad

- Se genera automáticamente un formulario para el objeto encuesta
- Los diferentes campos correspondes un widget html apropiado (DateTimeField, CharField). Admin realiza el mapeo.
- Tenemos controles para algunas acciones:
  - Save
  - Save and continue editing
  - Save and add another
  - Delete

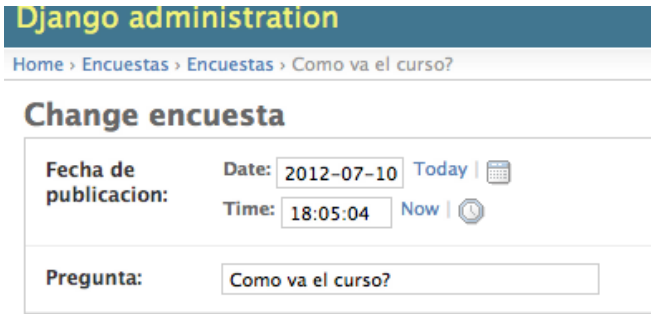
Personalizar formularios admin

- Los formularios se pueden personalizar mediante un conjunto de opciones que se proporcionan cuando se registra el objeto.
- Para personalizar un formulario se utiliza un modelo que se pasa como segundo parámetro a **admin.site.register()**

Personalizar formularios admin

- Por ejemplo para modificar el orden que se listan los elementos en el formulario

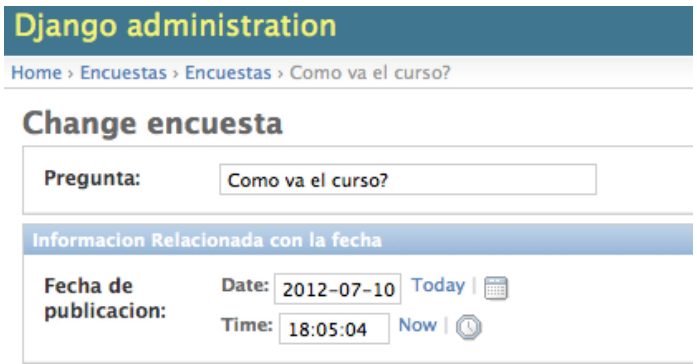
```
class EncuestaAdmin(admin.ModelAdmin):
    fields = ['fecha_pub', 'pregunta']

admin.site.register(Encuesta, EncuestaAdmin)
```



Personalizar formularios admin

- Si tuviera muchos campos, es posible dividirlos en fieldsets
- El primer elemento del tuple es el titulo para cada bloque

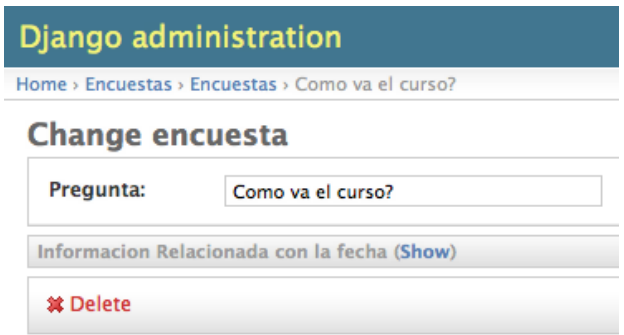
```
class EncuestaAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,{'fields': ['pregunta']}),
        ('Informacion Relacionada con la fecha',
         {'fields': ['fecha_pub']}),
        ]
```



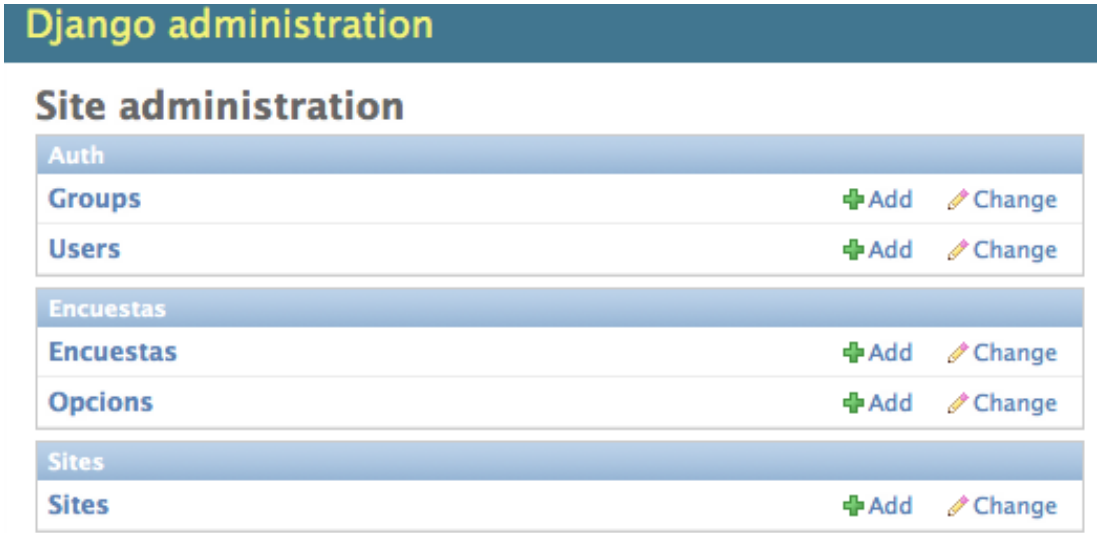
Personalizar formularios admin

- Se pueden asignar clases HTML a cada fieldset. Django provee la clase "collapse" que despliega un fieldset inicialmente colapsado

```
class EncuestaAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,{'fields': ['pregunta']}),
        ('Informacion Relacionada con la fecha',
         {'fields': ['fecha_pub'],
          'classes':['collapse']}),
        ]
```



Agregar Opciones para que sea desplegado en el index de admin



- Recuerde que hay que utilizar register

Agregar objetos relacionados

- Es mejor agregar las opciones directamente cuando se agrega la encuesta.
- Esto se hace extendiendo a **admin.StackedInline** y después agregandolo al **admin.ModelAdmin** que modifica al formulario de la encuesta.

```
class OpcionInline(admin.StackedInline):
    model = Opcion
    extra = 3
class EncuestaAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,{'fields': ['pregunta']}),
        ('Informacion Relacionada con la fecha',
         {'fields': ['fecha_pub'],
          'classes':['collapse']}),
        ]
        inlines = [OpcionInline]
```

Agregar Objetos relacionados

Ahora modifique la clase OpcionInline, para que extienda a **admin.TabularInline** en lugar de **admin.StackedInline**

Modificar el index de encuestas

```
class EncuestaAdmin(admin.ModelAdmin):
    # ...
    list_display = ('pregunta', 'fecha_pub')
```

- En list_display especificamos la lista de parámetros a listar.
- Aunque también se pueden incluir métodos

```
list_display = ('pregunta', 'fecha_pub', 'publicado_recientemente')
```

Modificar el index de encuestas

- Agregue los siguientes elementos a EncuestaAdmin y observe como se modifica el index de Encuestas.

```
class EncuestaAdmin(admin.ModelAdmin):
    # ...
    list_filter = ['fecha_pub']
    search_fields = ['pregunta']
    date_hierarchy = 'fecha_pub'
```

Interfaz Pública

Ahora nos enfocaremos en crear la interfaz pública , es decir las vistas(*views)*

- *Crear una página index,la cual despliegue las últimas encuestas*
- *Página de detalles, que muestre la pregunta de la encuesta y de opción para escribir una respuesta.*
- *Resultados, que despliegue los resultados de una encuesta.*
- *Definir la acción votar, la cual maneje como se maneja la elección de una opción particular.*

Diseñar las URLs

Cuando se crea un proyecto en Django automáticamente se genera un URLConf por defecto ubicado en *misitio/urls.py*, *y también se pone la variable de entorno* **ROOT_URLCONF** *en* settings.py

```
ROOT_URLCONF = 'misitio.urls'
```

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

admin.autodiscover()

urlpatterns = patterns('',
    url(r'^encuestas/$', 'encuestas.views.index'),
    url(r'^encuestas/(?P<encuesta_id>\d+)/$', 'encuestas.views.detalle'),
    url(r'^encuestas/(?P<encuesta_id>\d+)/resultados/$', 'encuestas.views.resultados'),
    url(r'^encuestas/(?P<encuesta_id>\d+)/votar/$', 'encuestas.views.votar'),
    url(r'^admin/', include(admin.site.urls)),
)
```

Significado de las URLs

- Cuando se solicita la url *encuestas/23 Django hace lo siguiente*
  - *Busca si hay una expresión regular ligada en* urlpatterns
  - Cuando encuentra **r'^encuestas/(?P<encuesta_id>\d+)/**, ejecuta la función *detalle*
  - *Ejecuta la función como:*

```
detalle(request=<HttpRequest object>, encuesta_id='23')
```

Mapeo de URLs

```
detalle(request=<HttpRequest object>, encuesta_id='23')
```

- El *encuesta_id=23 se obtiene de* (?P<encuesta_id> \d+)
- Utilizando paréntesis el patrón es capturado y enviado como argumento a la función
- ?P<encuesta_id> define el nombre con que será utilizado para identificar el patrón relacionado.
- d+ es la expresión regular que hace coincidencia con una secuencia de 1 o más dígitos.

Escribir las vistas

- Una vez definidas las urls es necesario definir las vistas.
- Si ahora iniciamos el servidor si definir las vistas y accesamos la url *http://127.0.0.1:8000/encuestas*

```
ViewDoesNotExist at /encuestas
Could not import encuestas.views.index. View does not exist in module encuestas.views.
```

Escribir vistas falsas

- La primera mapea a */encuestas/ y la segunda a* encuestas/Ida/
- La segunda vista recibe un parámetro y es el asociado con el que se obtiene de la expresión regular.
- Las vista son falsas solo dicen lo que hará la vista
- Escriba vistas falsas para las dos urls faltantes.

```
from django.http import HttpResponse

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

def detalle(request, encuesta_id):
    return HttpResponse("You're looking at encuesta %s." % encuesta_id)
```

Escribir vistas funcionales

Cada vista es reponsable de:

- Regresar un *HttpResponse*
- *o una excepcion (por ejemplo Htpp404)*

Todo los demás que ocurra en la vista es cuestión del desarrollador

- Puede consultar o no la BD
- Regresar un PDF
- Generar un archivo ZIP al vuelo
- etc

Escribir vistas funcionales

```
from encuestas.models import Encuesta
from django.http import HttpResponse

def index(request):
    lista_encuestas_recientes = Encuesta.objects.all().order_by('-fecha_pub')[:5]
    salida = '<br />'.join([p.pregunta for p in  lista_encuestas_recientes])
    return HttpResponse(salida)
```

Desventajas

- El diseño de la página es difícil de codificar
- Se tiene que hacer en Python

La solución es el sistema de plantillas de Django.

Vistas con plantillas

```
from django.template import Context, loader

def index(request):
    lista_ultimas_encuestas = Encuesta.objects.all().order_by('-pub_date')[:5]
     t = loader.get_template('encuestas/index.html')
     c = Context({
         'lista_ultimas_encuestas': lista_ultimas_encuestas,
     })
     return HttpResponse(t.render(c))
```

- El código carga(mediante *loader) la plantilla* index.html que se encuentran en "template_directory/encuestas/index.html
- Llena la plantilla con los datos del diccionario c y el método *render*

Vistas con plantillas

- Recargando la url *http://127.0.0.1/encuestas/*

```
TemplateDoesNotExist at /encuestas/
encuestas/index.html
```

Para corregir:

- Crear los directorios necesarios
- Editar **TEMPLATE_DIRS** en *settings.py para que apunte al directorio donde se creará index.html*

Vistas con plantillas

- Agregar el siguiente contenido

```
{% if lista_ultimas_encuestas %}
    <ul>
    {% for encuesta in lista_ultimas_encuestas %}
        <li><a href="/encuestas/{{ encuesta.id }}/">{{ encuesta.pregunta }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No hay encuestas aun.</p>
{% endif %}
```

Vistas con plantillas

En la plantilla podemos apreciar:

- Ses mezcla html y un lenguaje de plantilla
- Se utiliza {% instruccón %} para poner una instrurcción en el lenguaje de plantilla
- Se utiliza {{ variable }} para acceder a un dato de Context
  - {{ variable.algo }} algo puede referirse a un método, una clave o una variable.
  - En el caso del método, sólo pueden ser métodos sin parámetros

Shortcut para render y HttpResponse

Django define un conjunto de shortcuts. Como ejemplo analice el siguiente código:

```
from django.shortcuts import render_to_response

def index(request):
    lista_ultimas_encuestas = Encuesta.objects.all().order_by('-fecha_pub')[:5]
    return render_to_response('encuestas/index.html',
                              {'lista_ultimas_encuestas': lista_ultimas_encuestas})
```

- Primero se exporta de shortcuts el método *render_to_response*
- *El método recibe la ruta donde se almacena la plantilla, hace render con el diccionario que se proporciona como segundo parámetro, para finalmente regresar un* HttpResponse

Escribir

- Una plantilla y una vista para mostrar los detalles de cada encuesta.

Error 404

- Tratar de acceder a la siguiente dirección http://localhost:8000/encuestas/100/

```
DoesNotExist at /encuestas/100/
Encuesta matching query does not exist.
```

- El mensaje de error no es apropiado para una vista pública

Lanzar una excepción

```
from django.http import Http404
# ...
def detail(request, encuesta_id):
    try:
        p = Encuesta.objects.get(pk=encuesta_id)
    except Encuesta.DoesNotExist:
        raise Http404
    return render_to_response('encuesta/detalle.html', {'encuesta': p})
```

Shortcut

```
from django.shortcuts import render_to_response, get_object_or_404
# ...
def detail(request, encuesta_id):
    p = get_object_or_404(Encuesta, pk=encuesta_id)
    return render_to_response('encuestas/detalle.html', {'encuesta': p})
```

Desacoplar URLconf

- reescribir *urlpatterns en misitio/urls.py como:*

```
urlpatterns = patterns('',
    url(r'^encuestas/', include('encuestas.urls')),
    url(r'^admin/', include(admin.site.urls)),
)
```

- con lo anterior le decimos que las urls para todo lo que contenga encuestas será definido en *encuestas/urls.py* encuestas

Desacoplar URLconf

- Definir *urlpatterns en encuestas/urls.py como sigue:*

```
urlpatterns = patterns('encuestas.views',
    url(r'^$', 'index'),
    url(r'^(?P<encuesta_id>\d+)/$', 'detail'),
    url(r'^(?P<encuesta_id>\d+)/resultados/$', 'resultados'),
    url(r'^(?P<encuesta_id>\d+)/votar/$', 'votar'),
)
```

- la estrategia anterior hace fácil insertar y remover las rutas para las distintas aplicaciones
- Todas las aplicaciones se encargan de rutas relativas, lo que las hace independientes entre si.

Tarea

- Reescribir la vista y la plantilla que muestran los detalles de una encuesta, para que despliegue:
  - Los datos de la encuesta, fecha y pregunta,
  - las opciones asociadas a la encuesta
  - y una opción para regresar al listado de todas encuestas

Formularios

- Ahora se modificará la plantilla y la vista "detalle" para que nos permita votar la encuesta.

```
<h1>{{ encuesta.pregunta }}</h1>

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

<form action="/encuestas/{{ encuesta.id }}/votar/" method="post">
{% csrf_token %}
{% for opcion in encuesta.opcion_set.all %}
    <input type="radio" name="opcion" id="opcion{{ forloop.counter }}"
     value="{{ opcion.id }}" />
    <label for="opcion{{ forloop.counter }}">{{ opcion.opcion }}</label><br />
{% endfor %}
<input type="submit" value="Votar" />
</form>
```

Formularios

- Despliega un botón de radio por cada opción. El "value" de cada botón es el id de la opción
- La acción del formulario es la asociada con la url "/encuestas/ encuesta.id /votar/", y mediante el método "POST".
- *forloop.counter es una variable que existe dentro de los ciclos for e indica cuantas veces de ha ejecutado el ciclo.*
- *{% csrf_token %} es una etiqueta de plantilla que permite prevenir Cross Site Request Forgeries.*

Formularios

- La etiqueta {% csrf_token %} requiere información del objeto request, la cual no es accesible desde el contexto de la plantilla.
- Para corregir es necesario ajustar la vista detalle, como sigue:

```
def detalle(request, encuesta_id):
    p = get_object_or_404(Encuesta, pk=encuesta_id)
    return render_to_response('encuestas/detalle.html', {'encuesta': p},
                              context_instance=RequestContext(request))
```

Formularios

- Ahora es necesario el método para votar

```
def votar(request, encuesta_id):
    p = get_object_or_404(Encuesta, pk=encuesta_id)
    try:
        selected_opcion = p.opcion_set.get(pk=request.POST['opcion'])
    except (KeyError, Opcion.DoesNotExist):
        return render_to_response('encuestas/detalle.html', {
            'encuesta': p,
            'error_message': "You didn't select a opcion.",
        }, context_instance=RequestContext(request))
    else:
        selected_opcion.votos += 1
        selected_opcion.save()
        return HttpResponseRedirect(reverse('encuestas.views.resultados',
                                            args=(p.id,)))
```

Formularios

request.POST es un diccionario que contiene los datos proporcionados. request.POST 'opcion' Lanza una excepción si 'opcion' no es proporcionada, si eso ocurre se decide volver a desplegar 'detalle'.

Después de incrementar los votos en la opción elegida, se regresa un *HttpResponseRedirect en lugar de un* HttpResponse

Formularios

HttpResponseRedirect toma un sólo argumento: la URL a la cual se va a redirigir. Es una buena práctica de desarrollo Web, siempre regresar un *HttpResponseRedirect después de haber tratado exitosamente una petición que incluye datos POST.*

reverse() *Se utiliza el método* reverse() éste método permite proporcionar el nombre de la vista en lugar del la URL. En éste caso *reverse regresa algo como 'encuestas/2/resultados'*

Actividad

- Después de que alguien vota se direcciona a la url de resultados. Implemetar: La vista y la plantilla para resultados, la cual muestre información como la mostrada en la figura.