Django The Web framework for perfectionists with deadlines.

Manual para hacer CRUD

Versión 1.0

TECNOLOGÍAS PARA DESARROLLOS EN INTERNET

Alumnos:
Lemus Pablo Arturo
Martinez Lorenzo Jiovanni
Pérez Márquez José Alejandro
Sánchez Alcántara Jesús Esteban

Índice

1. Introducción

Django es un framework de desarrollo web de código abierto escrito en Python que ahorra tiempo y hace que el desarrollo Web sea divertido. El framework fue nombrado en alusión al guitarrista de jazz Django Reinhardt. La meta fundamental de Django es facilitar la creación de sitios web complejos poniendo énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio de DRY (Don't repeat yourself).

Utilizando Django se pueden crear y mantener aplicaciones web de alta calidad con un mínimo esfuerzo. El objetivo de este manual es darse una idea sobre como podría uno puede usar Django, a través del CRUD que viene explicado con un ejemplo, y así pueda ser útil para otros proyectos.

2. ¿Por qué Django es una buena opción?

Son varias las ventajas que presenta este framework entre las que podemos destacar:

- En cuestión de poco tiempo con Django se puede tener una aplicación bastante funcional y, a medida que uno vaya aprendiendo puede llegar a ser capaz de reducir aún más el tiempo de desarrollo.
- Al desarrollar en Django, este te induce a utilizar buenas prácticas de desarrollo web. Utiliza además una variación de la arquitectura MVC (Model-View-Controller/Modelo-Vista-Controlador) llamada MTV (Model-Template-View/Modelo-Plantilla-Vista), todo esto bajo el pricipio de Una y sólo una vez o DRY Don't Repeat Yourself.
- Se tienen url's limpias (amigables) y modulos poco dependientes
- Tiene una documentación entendible ya que está muy bien redactada y completa, lo que hace que sea más sencillo de aprender que otros frameworks.

3. Objetivo

Dar a conocer Django como una opción de Framework para tu proyecto, y de esta manera ver si te es útil de acuerdo a tus necesidades.

Django es un framework para hacer aplicaciones web (todo a tu medida, te agilizan muchas cosas, pero puedes hacer las aplicaciones exactamente a tu medida).

Nota
Si no estás muy familiarizado con el modelo MVC, perderás algo de tiempo en tratar de comprender (o a veces hacerte "bolas"), debido a que Django se basa en un modelo llamado MVT.

4. Requisitos de instalación

Se necesitará trabajar bajo Python 2.7, dicha versión es la que viene por defecto en los sistemas Linux (Ubuntu, Debian, etc..). En la mayoría de aplicaciones web necesitamos una base de datos que soporte, en nuestro caso particular usaremos SQLite que es la que viene por defecto. Si no tiene mucha experiencia con bases de datos o simplemente está interesado en probar Django, esta es la opción más fácil. SQLite está incluido en Python, por lo que no tendrá que instalar nada más para soportar su base de datos, sin embargo, es posible que desee utilizar una base de datos más potente como PostgreSQL, aunque ya requiere su instalación aparte.

Trabajaremos sobre Django 1.10, por lo que la instalación desde la terminal será algo distinta a lo que conocemos (sudo apt-get install Django), ya que si hacemos esto, la versión que nos descargara será la 1.9.

Proseguimos con la instalación de Django 1.10, de la siguiente manera:

- Instalaremos primero PIP que es un manejador de paquetes en Python

  `sudo apt-get install python3-pip`

- Ya que terminamos con la instalación de PIP, continuamos con la siguiente instrucción:

  `pip install django==1.10`

5. Creando nuestra aplicación

Para crear nuestra primera aplicación, será necesario usar el entorno virtual (virtualenv).

- Para instalar virtualenv será necesaria la siguiente instrucción:

  `sudo pip install virtualenv`

- Creamos nuestro proyecto bajo virtualenv de la siguiente manera:

  `virtualenv ‘‘nombre_de_proyecto’’` , en nuestro caso será `‘‘proyecto’’`

- Ya que creamos “CRUD”, activamos el ambiente con la instrucción `bin/activate`

- Ahí crearemos nuestro proyecto llamado `‘‘misitio’’`, con la siguiente instrucción:

  ```
  django-admin startproject misitio
  ```

- Ahora, desde la raíz de nuestro proyecto crearemos una app llamada courses para almacenar todo la información de los cursos:

  `sudo python manage.py startapp cursos`

Luego agregaremos a `‘courses’` la variable `INSTALLED_APPS` dentro de `crud/settings.py`

```
INSTALLED_APPS = [
    'cursos',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Para proseguir con nuestro CRUD, será necesario comprender las siguientes secciones.

5.1. Modelo

El modelo define los datos almacenados, se encuentra en forma de clases de Python, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros, posee métodos también. Todo esto permite indicar y controlar el comportamiento de los datos.

5.2. Plantilla

La plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django, en sí no solamente crea contenido en HTML (también XML, CSS, Javascript, CSV, etc).

5.3. Vista

La vista se presenta en forma de funciones en Python, su propósito es determinar qué datos serán visualizados, entre otras cosas más. La vista también se encarga de tareas conocidas como el envío de correo electrónico, la autenticación con servicios externos y la validación de datos a través de formularios.

5.4. MVT Y MVC su relación

En MVC:

- M, la porción de acceso a la base de datos, es manejada por la capa de la base de datos de Django.
- V, la porción que selecciona qué datos mostrar y cómo mostrarlos, es manejada por la vista y las plantillas.
- C, la porción que delega a la vista dependiendo de la entrada del usuario, es manejada por el framework mismo siguiendo tu URLconf y llamando a la función apropiada de Python para la URL obtenida.

En MVT:

- M significa "Model"(Modelo), la capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos: cómo acceder a estos, cómo validarlos, cuál es el comportamiento que tiene, y las relaciones entre los datos.
- T significa "Template"(Plantilla), la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación: como algunas cosas son mostradas sobre una página web o otro tipo de documento.
- V significa "View"(Vista), la capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada: puedes pensar en esto como un puente entre el modelos y las plantillas.

5.5. Creación del Modelo

En este CRUD consideramos que nuestros cursos constan de un nombre, fecha de inicio y fecha de fin.

El archivo del modelo dentro de `cursos/models.py`, quedaría de la siguiente manera:

```
from __future__ import unicode_literals
from django.db import models

# Create your models here.

class Curso(models.Model):
    nombre = models.CharField(max_length=140) #nombre del curso
    fecha_inicio = models.DateTimeField() #fecha de inicio del curso
    fecha_fin = models.DateTimeField() #fecha de fin del curso
```

Ya que terminamos de definir nuestro modelo, crearemos las migraciones:

```
./manage.py makemigrations
```

Y luego hacemos las migraciones para crear la tabla en la Base de datos:

```
./manage.py migrate
```

Vistas

- Para listar los cursos en nuestra página usaremos `ListView`
- Para la creación de un nuevo curso usaremos `CreateView`
- Para editar un objeto ya existente mediante un formulario usaremos `UpdateView`
- Por último, para borrar un objeto usaremos `DeleteView`

El código queda de la siguiente manera: `cursos/view.py`

```
from django.shortcuts import render
from django.views.generic import ListView
from .models import Curso
from django.views.generic.edit import CreateView
from django.core.urlresolvers import reverse_lazy
from django.views.generic.edit import UpdateView
from django.views.generic.edit import DeleteView

# Create your views here.
class CourseList(ListView):
    model = Curso

class CourseDetail(ListView):
    model = Curso

class CourseCreation(CreateView):
    model = Curso
    success_url = reverse_lazy('cursos:ver')
    campos = ['nombre','fecha_inicio','fecha_fin']

class CourseUpdate(UpdateView):
    model = Curso
    success_url = reverse_lazy('cursos:ver')
    campos = ['nombre','fecha_inicio','fecha_fin']

class CourseDelete(DeleteView):
    model = Curso
    success_url = reverse_lazy('cursos:ver')
```

5.6. Url' s donde nuestras vistas serán llamadas

Extendemos el módulo principal de URLs dentro de `misitio/urls.py` de la siguiente forma:

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

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^cursos/', include('cursos.urls', namespace='cursos')),
]
```

Agregamos un archivo `urls.py` dentro de cursos, el código queda así:

```
from django.conf.urls import url

from .views import{
```

```
    CourseList,
    CourseDetail,
    CourseCreation,
    CourseUpdate,
    CourseDelete
}

urlpatterns = [
    url(r'^$', CourseList.as_view(), name='ver')
    url(r'^(?P<pk>\d+)$', CourseDetail.as_view(), name='detalle'),
    url(r'^nuevo$', CourseCreation.as_view(), name='nuevo')
    url(r'^editar/(?P<pk>\d+)$', CourseUpdate.as_view(), name='editar')
    url(r'^borrar/(?P<pk>\d+)$', CourseDelete.as_view(), name='borrar')
]
```

5.7. Templates

Necesitamos crear los templates HTML. Para esto dentro de `cursos`, crearemos los templates

`cursos/templates/cursos/lista_curso.html`

```
<h1>Cursos de Musica</h1>
<p>
    <a href="{% url "cursos:nuevo" %}">Agregar curso</a>
</p>
<ul>
    {% for curso in object_list %}
        <li>
            <p>{{ curso.nombre }}</p>
            <p>
                <a href="{% url "cursos:detalle" cursos.id %}">Ver</a> |
                <a href="{% url "cursos:editar" cursos.id %}">Editar</a> |
                <a href="{% url "cursos:borrar" cursos.id %}">Borrar</a>
            </p>
        </li>
    {% endfor %}
</ul>
```

Creamos el **form** del HTML (esto ya es convención)

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Submit" />
</form>
```

Para ver los detalles del curso creamos un archivo `curso_detail.html`

```
<h1>{{ curso.nombre }}</h1>
<p>Fecha de inicio: <i>{{ curso.fecha_inicio }}</i></p>
<p>Fecha de fin: <i>{{ curso.fecha_fin }}</i></p>
<p>
    <a href="{% url "cursos:ver" %}">Regresar</a>
</p>
```

Por último necesitamos la pantalla de confirmación para **borrar** un curso, en el archivo `borrarcurso.html`

```
<form method="post">{% csrf_token %}
    ¿Estás seguro que deseas borrar el curso "{{ curso.nombre }}"?
    <input type="submit" value="Submit" />
</form>
```

5.8. Correr el servidor

Para correr el servidor y verificar que se crearon satisfactoriamente nuestras plantillas es necesario poner la siguiente instrucción en la terminal:

```
./manage.py runserver
```

Posteriormente en nuestro navegador pondremos `localhost:8000/cursos` y de esta manera debería abrir una página de la siguiente manera:

Al pulsar “agregar curso” se despliega la siguiente página:

Al agregar un campo mal, se despliega la siguiente página:

Al crearse el curso, se añade en la página principal:

Al borrar el curso se despliega la siguiente página:

6. Referencias

- Tu primera app CRUD en Django con Class Based Views - Platzi , https://platzi.com/blog/django-class-based-views/, consultado el día [14 -septiembre-2016]
- 5.2. El patrón de diseño MTV, http://librosweb.es/libro/django_1_0/capitulo_5/el_patron_de_diseno_mtv.html, consultado el día [20-septiembre-2016]