Contenidos a Trabajar

1. Formularios
   - Formularios HTML
     - Get y Post
     - El papel de Django en los formularios
     - La Clase Form de Django
2. Procesamiento de formularios
3. Construyendo un formulario
   - La Clase Form
   - La Vista
   - La Plantilla

Formularios (forms)

A menos que planee crear sitios web y aplicaciones que no hagan nada más que publicar contenido y no acepten aportes de sus visitantes, necesitará comprender y usar formularios.

Django proporciona una gama de herramientas y bibliotecas para ayudar a crear formularios para aceptar la entrada de los visitantes del sitio y luego procesar y responder a la entrada.

Formularios HTML

En HTML, un formulario es una colección de elementos internos **<form>...</form>** que permiten a un visitante hacer cosas como ingresar texto, seleccionar opciones, manipular objetos o controles, etc., y luego enviar esa información al servidor.

Algunos de estos elementos de interfaz de formulario (entrada de texto o casillas de verificación) están integrados en el propio HTML. Otros son mucho más complejos; una interfaz que muestra un selector de fecha o le permite mover un control deslizante o manipular controles normalmente utilizará JavaScript y CSS, así como elementos **<input>** de formulario HTML para lograr estos efectos.

Además de sus elementos **<input>**, un formulario debe especificar dos cosas:

- *dónde* : la URL a la que se deben devolver los datos correspondientes a la entrada del usuario
- *cómo* : el método HTTP por el que deben devolverse los datos

Como ejemplo, el formulario de inicio de sesión para el administrador de Django contiene varios elementos **<input>**: uno **type="text"** para el nombre de usuario, uno **type="password"** para la contraseña y otro **type="submit"** para el botón «Iniciar sesión». También contiene algunos campos de texto ocultos que el usuario no ve, que Django usa para determinar qué hacer a continuación.

También le dice al navegador que los datos del formulario deben enviarse a la URL especificada en el atributo **action= /admin/** del **<form>** y que deben enviarse utilizando el mecanismo HTTP especificado porel atributo **method= post**.

Cuando se activa el elemento **<input type="submit" value="Log in">**, los datos se devuelven a **/admin/**

GETy POST

GETy **POST** son los únicos métodos HTTP que se pueden usar cuando se trata de formularios.

El formulario de inicio de sesión de Django se devuelve utilizando el método **POST**, en el que el navegador agrupa los datos del formulario, los codifica para su transmisión, los envía al servidor y luego recibe su respuesta.

GET, por el contrario, agrupa los datos enviados en una cadena y los usa para componer una URL. La URL contiene la dirección donde se deben enviar los datos, así como las claves y valores de los datos. Puede ver esto en acción si realiza una búsqueda en la documentación de Django, que producirá una URL del formulario **https://docs.djangoproject.com/search/?q=forms&release=1**.

GETy **POST** se utilizan normalmente para diferentes propósitos.

Cualquier solicitud que pueda usarse para cambiar el estado del sistema, por ejemplo, una solicitud que termine realizando cambios en la base de datos, debe usar **POST**. **GET** debe usarse solo para solicitudes que no afecten el estado del sistema.

GET tampoco sería adecuado para un formulario de contraseña, porque la contraseña aparecería en la URL y, por lo tanto, también en el historial del navegador y en los registros del servidor, todo en texto sin formato. Tampoco sería adecuado para grandes cantidades de datos, ni para datos binarios, como una imagen. Una aplicación web que utiliza solicitudes **GET** de formularios de administración es un riesgo para la seguridad: puede ser fácil para un atacante imitar la solicitud de un formulario para obtener acceso a partes confidenciales del sistema. **POST**, junto con otras protecciones como la **protección CSRF** de Django ofrece máscontrol sobre el acceso.

Por otro lado, **GET** es adecuado para cosas como un formulario de búsqueda web, porque las URL que representan una solicitud **GET** se pueden marcar, compartir o volver a enviar fácilmente.

El papel de Django en los formularios

El manejo de formularios es un negocio complejo. Considere el administrador de Django, donde es posible que sea necesario preparar numerosos elementos de datos de varios tipos diferentes para mostrarlos en un formulario, representarlos

como HTML, editarlos usando una interfaz conveniente, devolverlos al servidor, validarlos y limpiarlos, y luego guardarlos o pasarlos para su posterior procesamiento.

La funcionalidad de formularios de Django puede simplificar y automatizar grandes porciones de este trabajo, y también puede hacerlo de manera más segura de lo que la mayoría de los programadores podrían hacer en el código que escribieron ellos mismos.

Django maneja tres partes distintas del trabajo relacionado con los formularios:

- preparar y reestructurar datos para que estén listos para renderizar
- crear formularios HTML para los datos
- recibir y procesar formularios enviados y datos del cliente

Es *posible* escribir código que haga todo esto manualmente, pero Django puede encargarse de todo por ti.

Formularios en Django

Hemos descrito brevemente los formularios HTML, pero un HTML **<form>** es solo una parte de la maquinarianecesaria.

En el contexto de una aplicación web, "formulario" puede referirse a ese HTML **<form>**, al Django **Form** que lo produce, a los datos estructurados que se devuelven cuando se envían, o a la colección de trabajo de principio a fin de estas partes.

La clase Form de Django

En el corazón de este sistema de componentes está la clase **Form** de Django. De la misma manera que un modelo de Django describe la estructura lógica de un objeto, su comportamiento y la forma en que se nos representan sus partes, una clase **Form** describe una forma y determina cómo funciona y aparece.

Los campos de un formulario son en sí mismos clases; administran los datos del formulario y realizan la validación cuando se envía un formulario. Por ejemplo, **DateField** y a **FileField** manejan tipos de datos muy diferentes y tienen que hacer cosas diferentes con ellos.

Un campo de formulario se representa para un usuario en el navegador como un «widget» HTML, una pieza de maquinaria de interfaz de usuario. Cada tipo de campo tiene una clase de Widget predeterminada adecuada, pero se pueden anular según sea necesario.

Procesamiento de formularios

Cuando renderizamos un objeto en Django, generalmente:

1. obtenerlo en la vista (obtenerlo de la base de datos, por ejemplo)
2. pasarlo al contexto de la plantilla
3. expandirlo a marcado HTML usando variables de plantilla

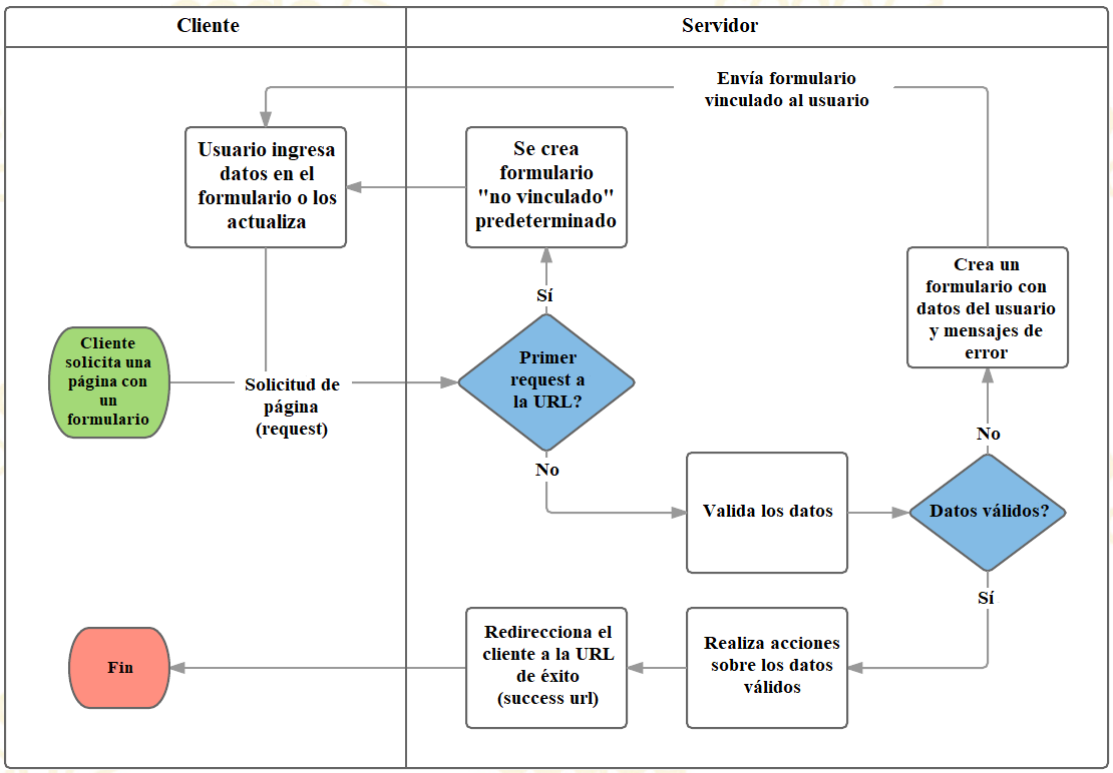
La representación de un formulario en una plantilla implica casi el mismo trabajo que la representación de cualquier otro tipo de objeto, pero existen algunas diferencias clave.

Tiene perfecto sentido representar un formulario vacío; eso es lo que hacemos cuando queremos que elusuario lo complete.

Cuando creamos una instancia de un formulario, podemos optar por dejarlo vacío o rellenarlo previamente,por ejemplo, con:

- datos de una instancia de modelo guardada (como en el caso de los formularios de administraciónpara editar)
- datos que hemos recopilado de otras fuentes
- datos recibidos de un envío de formulario HTML anterior

El último de estos casos es el más interesante, porque es lo que hace posible que los usuarios no solo lean unsitio web, sino que también le envíen información.


Cliente
Servidor
Envía formulario vinculado al usuario
Usuario ingresa datos en el formulario o los actualiza
Se crea formulario "no vinculado" predeterminado
Crea un formulario con datos del usuario y mensajes de error
Sí
Cliente solicita una página con un formulario
Solicitud de página (request)
Primer request a la URL?
No
Valida los datos
Datos válidos?
No
Sí
Fin
Redirecciona el cliente a la URL de éxito (success url)
Realiza acciones sobre los datos válidos

Construyendo un formulario

La clase Form

Ya sabemos cómo queremos que se vea nuestro formulario HTML. Nuestro punto de partida para ello en Django es este:

formularios.py

```
from django import forms
class NameForm (Forms.Form):
    your_name = forms.CharField(label= ´Your name´, max_length=100)
```

Esto define una clase **Form** con un solo campo (**your_name**). Hemos aplicado una etiqueta amigable para los humanos al campo, que aparecerá en el **<label>**cuando se represente (aunque en este caso, la **label** que especificamos es en realidad la misma que se generaría automáticamente si la hubiéramos omitido).

La longitud máxima permitida del campo está definida por **max_length**. Esto hace dos cosas. Pone una **maxlength="100"**en el **<input>** HTML (por lo que el navegador debe evitar que el usuario ingrese más de esa cantidad de caracteres en primer lugar). También significa que cuando Django reciba el formulario del navegador, validará la longitud de los datos.

Una instancia **Form** tiene un método **is_valid()** que ejecuta rutinas de validación para todos sus campos. Cuando se llama a este método, si todos los campos contienen datos válidos, hará lo siguiente:

- Devolver **True**
- Colocar los datos del formulario en su atributo **cleaned_data**.

El formulario completo, cuando se represente por primera vez, se verá así:

```
<label for="your_name">Your name: </label>

<input id="your_name" type="text" name="your_name" maxlength="100" required>
```

Tenga en cuenta que **no** incluye las etiquetas **<form>** ni un botón de envío. Tendremos que proporcionarlos nosotros mismos en la plantilla.

La vista

Los datos del formulario enviados a un sitio web de Django son procesados por una vista, generalmente la misma vista que publicó el formulario. Esto nos permite reutilizar parte de la misma lógica.

Para manejar el formulario, necesitamos instanciarlo en la vista de la URL donde queremos que se publique:

vistas.py

```
from django.http import
HttpResponseRedirect from django.shortcuts
import render
from .forms import NameForm
def get_name(request):
    # if this is a POST request we need to process the form data
    if request.method == 'POST':
        # create a form instance and populate it with data from the request:
        form = NameForm(request.POST)
        # check whether it's valid:
        if form.is_valid():
            # process the data in form.cleaned_data as
            required # ...
            # redirect to a new URL:
            return HttpResponseRedirect('/thanks/')
    # if a GET (or any other method) we'll create a blank form
    else:
        form = NameForm()
    return render(request, 'name.html', {'form': form})
```

Si llegamos a esta vista con una solicitud **GET**, creará una instancia de formulario vacía y la colocará en el contexto de la plantilla para que se represente. Esto es lo que podemos esperar que suceda la primera vez que visitamos la URL.

Si el formulario se envía mediante una solicitud **POST**, la vista volverá a crear una instancia de formulario y la completará con los datos de la solicitud: **form = NameForm(request.POST)** esto se denomina «vincular datos al formulario» (ahora es un formulario *vinculado*).

Llamamos al método **is_valid()** del formulario; si no es **True**, volvemos a la plantilla con el formulario. Esta vez, el formulario ya no está vacío (*sin vincular*), por lo que el formulario HTML se completará con los datos enviados anteriormente, donde se puede editar y corregir según sea necesario.

Si **is_valid()** es **True**, ahora podremos encontrar todos los datos del formulario validado en su atributo **cleaned_data**. Podemos usar estos datos para actualizar la base de datos o realizar otro procesamiento antes de enviar una redirección HTTP al navegador para indicarle a dónde ir a continuación.

La plantilla

No necesitamos hacer mucho en nuestra plantilla name.html:

```
<form action="/your-name/" method="post">
        {% csrf_token %}
        {{ form }}
        <input type="submit" value="Submit">
</form>
```

Todos los campos del formulario y sus atributos serán desempaquetados en marcado HTML de **{{ form }}** porel lenguaje de plantilla de Django.

Ahora tenemos un formulario web en funcionamiento, descrito por Django **Form**, procesado por una vista yrepresentado como HTML **<form>**.

Eso es todo lo que necesita para comenzar, pero el marco de formularios pone mucho más a su alcance. Una vez que comprenda los conceptos básicos del proceso descrito anteriormente, debe estar preparado para comprender otras características del sistema de formularios y listo para aprender un poco más sobre la maquinaria subyacente.