Formularios

Antonio Espín Herranz

Contenidos

- Objeto request. Propiedades y métodos.
- Procesamiento de formularios.
- Formularios basados en clases.
- Tipos de campos.
- Validaciones.

Objeto request

- Cuando implementamos una vista en django siempre el primer parámetro es un objeto de tipo: HttpRequest.

- A partir de este objeto podemos recuperar información del cliente, por ejemplo:
  - Cabeceras Http
  - Campos de un formulario
  - …

- Dispone de una serie de métodos y propiedades.

Objeto request: Atributos / métodos

Atributos o Métodos	Descripción	Ejemplo
request.path	La ruta completa, no incluye el dominio pero incluye, la barra inclinada.	"/hola/"
request.get_host()	El host (ejemplo: tu "dominio," en lenguaje común).	"127.0.0.1:8000" o "www.example.com"
request.get_full_path()	La ruta (path), mas una cadena de consulta (si está disponible).	"/hola/?print=true"
request.is_secure()	True si la petición fue hecha vía HTTPS. Si no, False.	True o False

- Además, dispone de un diccionario: **request.META** que contiene todas las cabeceras HTTP.
- Si se intenta acceder a una clave que NO existe se obtendrá una excepción: KeyError.
- Se deberían capturar las excepciones cuando accedamos directamente a una cabecera.

Ejemplo

```
def mostrar_navegador(request):
    try:
        ua = request.META['HTTP_USER_AGENT']
    except KeyError:
        ua = 'unknown'
    return HttpResponse("Tu navegador es %s" % ua)
```

PRACTICA Nº 6.1: Formularios1

Procesamiento de formularios

- Para recuperar información de los formularios request proporciona otros objetos:
  - GET / POST
    - Normalmente GET se utilizará para realizar consultas.
    - Y POST para modificar datos a través de un formulario.
- Para acceder a ellos:
  - request.GET / request.POST
- Vienen a ser como diccionarios Python.

Procesamiento de formularios

- **Django permite**:
  - El procesamiento de formularios **HTML**
  - Y ofrece la posibilidad de utilizar una **librería del framework (django.forms)** para la creación de formularios (vistas basadas en clases)

- **En general al procesar formularios** tendremos que hacer:
  - **Comprobaciones** para ver si nos han **enviado datos**, estas se pueden hacer sobre request.GET / request.POST
  - **Validaciones** de los datos.
  - Y **acceder al modelo** tanto para modificar datos como para mostrarlos.

Procesamiento de formularios: **Simple**

- **views.py**

```
from django.shortcuts import render
def buscador(request):
    return render(request, 'formulario_buscar.html')
```

- **formulario_buscar.html**

```
<html>
    <head>
        <title>Buscar</title>
    </head>
    <body>
    <form action="/resul/" method="get">
        <input type="text" name="q">
        <input type="submit" value="Buscar">
    </form>
    </body>
</html>
```

urls.py

```
path('buscar/', formularios.views.buscador),
path('resul/', formularios.views.resultados),
```

```
from django.http import HttpResponse
def resultados(request):
    if 'q' in request.GET and request.GET['q']:
        mensaje = 'Estas buscando: %s' % request.GET['q']
    else:
        mensaje = 'Haz subido un formulario vacio.'
    return HttpResponse(mensaje)
```

Notas

- Comprobación si hemos recibido datos del formulario:
  - if '**q**' in **request.GET** and request.GET['**q**']:
  - La q se relaciona con la propiedad name del HTML.
    - <input type="text" name="**q**">

- En el formulario se indica con action la petición a realizar que estará definida en la etiqueta form.

- Y el método **GET / POST**
  - <form action="/**resul**/" method="**get**">
  - path('**resul**/', formularios.views.**resultados**)

- Desde la vista podemos acceder al modelo para recuperar datos y pasarlos a una plantilla a través de un diccionario.

Procesamiento de formularios

- Los posibles **errores** de **validación** que tengamos en un **formulario** se deberían de mostrar en el mismo form, sino es incómodo para el usuario.
  - En las validaciones no solo será comprobar si está o no vacío un campo.
  - Habrá que validar formatos
  - Longitud de los datos.

Procesamiento de formularios: **mejor I**

- Este ejemplo: comprueba el campo q,
- Si está llama al modelo y le pasa los resultados de la consulta a una plantilla
- La consulta que realiza al modelo es: libros que contengan el término tecleado en q sin tener en cuenta las mayúsculas / minúsculas.
- Y el formulario buscar recibirá un indicador de si hay o no error
- OJO, en este caso el control de errores sería pobre, no distingue ningún tipo de error
- Por ejemplo, controlar la longitud de la consulta.

biblioteca/views.py

```
from django.http import HttpResponse
from django.shortcuts import render

from biblioteca.models import Libro

def formulario_buscar(request):
    return render(request, 'formulario_buscar.html')

def buscar(request):
    if 'q' in request.GET and request.GET['q']:
        q = request.GET['q']
        libros = Libro.objects.filter(titulo__icontains=q)
        return render(request, 'resultados.html',{'libros': libros, 'query': q})
    else:
        return render(request, 'formulario_buscar.html', {'error': True})
```

Procesamiento de formularios: **mejor II**

biblioteca/templates/formulario_buscar.html

```
<html>
<head>
    <title>Buscar</title>
</head>
<body>
    {% if error %}
        <p style="color: red;">Por favor introduce un término de búsqueda.</p>
    {% endif %}
    <form action="/buscar/" method="get">
        <input type="text" name="q">
        <input type="submit" value="Buscar">
    </form>
</body>
</html>
```

Una mejora I

- Modificamos la vista buscar para que compruebe si hay error o no.
- Si el usuario visita **/buscar/** sin parámetros GET se mostrará el formulario vacío y sin errores.
- Equivalente a cuando se realiza la primera petición.
- Con esto nos podemos **quitar la vista** de la página 11, **formulario_buscar** que simplemente pinta el formulario.

biblioteca/views.py

```
def buscar(request):
    error = False
    if 'q' in request.GET:
        q = request.GET['q']
        if not q:
            error = True
        else:
            libros = Libro.objects.filter(titulo__icontains=q)
            return render(request, 'resultados.html', {'libros': libros, 'query': q})

    return render(request, 'formulario_buscar.html', {'error': error})
```

Una mejora II

- Y a nivel de la plantilla del formulario:
- Se puede sustituir la declaración:
  - <form action="/buscar/" method="get">
- Por esta otra:
  - <form action="" method="get">
- Al dejarla en blanco, se envía la petición a la misma página.

PRACTICA Nº 6.2: Formularios2

Formularios basados en clases

- Django proporciona la librería **django.forms** para la creación y validación de formularios HTML.
- Por cada **formulario HTML** necesitamos una clase **Form**.
- Dentro de esta clase se definirán los campos que formarán parte del formulario.
  - ***Django tiene una clase para cada tipo de campo.***
- Ofrece **varias posibilidades para mostrarlos**:
  - Como tabla, lista, párrafo …

Formularios basados en clases

- **Los campos del formulario se definen de una forma muy similar a cuando definimos el modelo**.

- **Se puede indicar**:
  - Si son obligatorios
  - Longitud del campo
  - El tipo de Widget, por ejemplo, cambiar la caja de texto en un textarea.
  - Distinto tipo de campo como en la definición de modelos.

- Django añade **capacidades de validación**, y permite la **personalización** de **validaciones**.

Convenciones

- Se creará un módulo de Python a nivel de **aplicación** llamado **forms.py** que contendrá las clases de los formularios.

- Las clases tienen que heredar de **forms.Form**

- Hay que importar:
  - **from django import forms**

Ejemplo

 # mi_app/forms.py
from django import forms

```
class FormularioContactos(forms.Form):
        nombre= forms.CharField()
        email = forms.EmailField(required=False)
        datos= forms.CharField()
```

- Los formularios se pueden comprobar con:
  python manage.py shell
  from mi_app.forms import FormularioContactos
  f=FormularioContactos()
  print(f) → ***Por defecto es una tabla, f.as_table()***
  print(f.as_ul())
  print(f.as_p())
  print(f['**nombre**'])

- ***Django NO añade las cabeceras de la tabla ni del formulario, lo tendremos que especificar en la plantilla.***

Métodos / Propiedades del form

- Siendo **f un formulario**:
  - **f.is_bound**: True / False. Indica si es un formulario vinculado. Se le puede pasar un diccionario con datos a visualizar, con esto se está creando un formulario vinculado.
    - f = FormularioContactos({'asunto': 'Hola', 'email': 'adrian@example.com‘,'mensaje': '¡Buen sitio!'})
    - f.is_bound → **True**
  - **f.is_valid():** True / False. Lanza las validaciones del formulario.
  - **f.errors**: Los errores de cada campo del form.
    - Un diccionario. Las claves son los nombres de los campos.
  - **f.cleaned_data**: Devuelve un diccionario con todos los datos del formulario.
    - Las claves son los campos del form.
    - Solo añade los campos válidos.

Prueba

```
In [1]: from formularios.forms import *

In [2]: f=FormContacto()

In [3]: f.is_valid()
Out[3]: False

In [4]: f.is_bound
Out[4]: False

In [5]: f=FormContacto({'nombre':'Antonio'})

In [6]: f.is_bound
Out[6]: True

In [7]: f.is_valid()
Out[7]: False

In [8]: f.errors
Out[8]:
{'edad': ['This field is required.'],
 'fecha': ['This field is required.'],
 'observaciones': ['This field is required.'],
 'password': ['This field is required.'],
 'usuario': ['This field is required.']}

In [9]: f.errors['edad']
Out[9]: ['This field is required.']

In [11]: f.cleaned_data
Out[11]: {'email': '', 'nombre': 'Antonio'}

In [12]: _
```

La plantilla

mi_app/templates/formulario_contactos.html

```
<html>
<head>
    <title>Datos de contacto</title>
</head>
<body>
    <h1>Datos de contacto</h1>
    {% if form.errors %}
        <p style="color: red;">Por favor corrige lo siguiente:
        </p>
    {% endif %}

    <form action="" method="post"># action: vacío se reenvía así mismo.
    {% csrf_token %} # Cross-site request forgery, protege de ataques CSRF
    <table>
        {{ form.as_table }} # Lo muestra como una tabla
    </table>
    <input type="submit" value="Enviar">
    </form>
</body>
</html>
```

La vista

- En general, puede ser:

```
def some_view(request):
    if request.method == 'POST':
        form = SomeForm(request.POST)
        if form.is_valid():
            return HttpResponseRedirect('/thanks'/)
    else:
        form = SomeForm()
    return render(request, 'some_form.html', {'form': form})
```

Ejemplo

```
def contactos(request):
    if request.method == 'POST':
        form = FormularioContactos(request.POST)
        if form.is_valid():
            cd = form.cleaned_data
            # Trabajar con los datos recibidos!!
            print(cd['nombre'], cd['email'], cd.get('datos', '...'))

            # Va todo BIEN aviso al Usuario con una pantalla de información
            return HttpResponseRedirect('/contactos/gracias/')
    else:
        # Pasar datos de inicialización!
        # La petición viene por GET, se crea el form es la 1ª petición
        form = FormularioContactos(initial={'nombre': 'Anónimo'})

    # Si el form tiene algún error o se crea por 1ª vez se pinta
    return render(request, 'formulariocontactos.html', {'form': form})
```

Tipos de campos

- **CharField**():
  - Se traduce por <input type=“text” …
- CharField(widget=**forms.PasswordInput**())
  - Se traduce por <input type=“password”
- Si el widget es: forms.**TextArea**
  - Genera un campo de observaciones.
  - ***Cada campo lleva asociado un widget. Por defecto, se muestra este a no ser que se cambie en el parámetro.***
- **Parámetros**: (utilizar parámetros nominales)
  - min_length, max_length
  - required=False (por defecto es True)

Tipos de campos

- **IntegerField**()
  - Parámetros: **min_value** y **max_value**
  - **Normaliza a int**
- **FloatField():**
  - Normaliza a float.
- **EmailField**():
  - Para direcciones de correos.
- **URLField()**
  - Un campo de url.
- **BooleanField()**
  - Genera un checkBox

Tipos de campos

- **DateField**()
  - Normaliza a datetime.date object
  - Parámetro:
    - **input_formats**

    - Si el parámetro no se suministra los parámetros por defecto son:
    - ['%Y-%m-**%d**', *# '2006-10-25'*
    - '%m/**%d**/%Y', *# '10/25/2006'*
    - '%m/**%d**/%y'] *# '10/25/06'*

  - *Si en el fichero de settings se especifica USE_L10N=False*
  - *Se amplía el número de parámetros.*

- **TimeField()**
  - **Normaliza** a **datetime.time**
  - Parámetro**: input_formats**

- **DateTimeField()**
  - **Normaliza** a datetime.datetime
  - Parámetro: **input_formats**

Tipos de campos

- **ChoiceField**()
  - Genera un desplegable
  - Parámetro: **choices**, los elementos a cargar.
  - Será una lista o tupla de tuplas de 2 elementos.
    - RELEVANCE_CHOICES = ((1,'valor1'), (2,'valor2'), …)
  - relevance = forms.ChoiceField(choices = RELEVANCE_CHOICES, required=True)
- **MultipleChoiceField**
  - **Normaliza** a una **lista de string**.
  - Recibe el parámetro **choices** para pasar los elementos de la lista.

Tipos de campos

- Hay más tipos de campos en la doc:
  - https://docs.djangoproject.com/es/4.2/ref/forms/fields/

Parámetros de los campos del Form

- **error_messages:**
  - Añadir mensajes de error personalizados, a un campo.
  - Con un diccionario:

```
>>> name = forms.CharField(error_messages={'required': 'Please
enter your name'})
>>> name.clean('')
Traceback (most recent call last):
  ...
ValidationError: ['Please enter your name']
```

- **help_text**:
  - Mostrar un texto de ayuda al lado del campo.
- **disabled:**
  - deshabilita el campo en el form.

PRÁCTICA 6.3: Formularios con Clases

Parámetros de los campos del Form II

- **label**:
  - Al crear el campo se puede indicar la etiqueta.

- **widget**:
  - El control que va a utilizar para renderizar.
    - **En vez de input type='text' queremos un textarea.**

- **initial**:
  - Para dar un valor inicial al campo.
  - Rellena el campo value de html.

- **required:**
  - El campo es obligatorio. Por defecto es True. Sólo se indica cuando queremos que no sea obligatorio. **required=False**

Ejemplo

```
>>> import datetime
>>> class DateForm(forms.Form):
    ... day = forms.DateField(initial=datetime.date.today)
>>> print(DateForm())


<tr>
   <th>Day:</th>
   <td><input type="text" name="day"
   value="12/23/2008" required><td>
</tr>
```

Validaciones

- Se pueden añadir validadores personalizados.

- Dentro de la clase del form, si se añaden a la clase métodos que empiecen por clean_ y terminen por el nombre de un campo Django los lanzará automáticamente al validar el form.

- class MiForm(forms.Form):
  - nombre = CharField(required=False)
  - …
  - def clean_nombre(self):
    - aux = self.cleaned_data[‘nombre’]
    - … hacer nuestras comprobaciones.
    - Si consideramos que hay error, lanzamos:
    - ***raise forms.ValidationError("¡Se requieren mínimo 4 palabras!“, ‘invalid’)***
    - La función devolverá aux si va todo bien.

Renderizar los errores

- Al reenviar el form Django mostrará los errores pero se puede personalizar y dar estilos para mostrarlos por encima del form.

```
{% if form.errors %}
    {% for field in form %}
        {% for error in field.errors %}
            <div class="alert">
            <strong>{{ error|escape }}</strong>
            </div>
        {% endfor %}
    {% endfor %}
{% endif %}
```