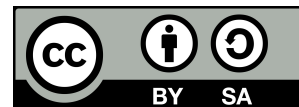


# CURSO PYTHON

## APLICACIONES WEB CON DJANGO



Autor: Jon Vadillo  
[www.jonvadillo.com](http://www.jonvadillo.com)



# Contenidos

1. Introducción y **fundamentos básicos**
2. Crea tu primer **proyecto** en Django
3. Crea tu primera **aplicación** en django
4. El **modelo** en Django, **acceso a datos** y la aplicación de administrador
5. **Vistas** y **plantillas** en Django
6. **Vistas basadas en clases** (Class Based Views)
7. **Formularios** en Django

## 7. Formularios en Django

# Formularios

The image shows a web browser window with the address bar displaying "127.0.0.1:8000/catalog/book/create/". The page layout includes a left sidebar with navigation links and a main content area with form fields.

**Left Sidebar:**

- Home
- All books
- All authors
- User: superman
- My Borrowed
- Logout
- Staff
- All borrowed

**Main Content Area:**

**Title:**

**Author:**

**Summary:**

Enter a brief description of the book

**ISBN:**

13 Character ISBN number

**Genre:**

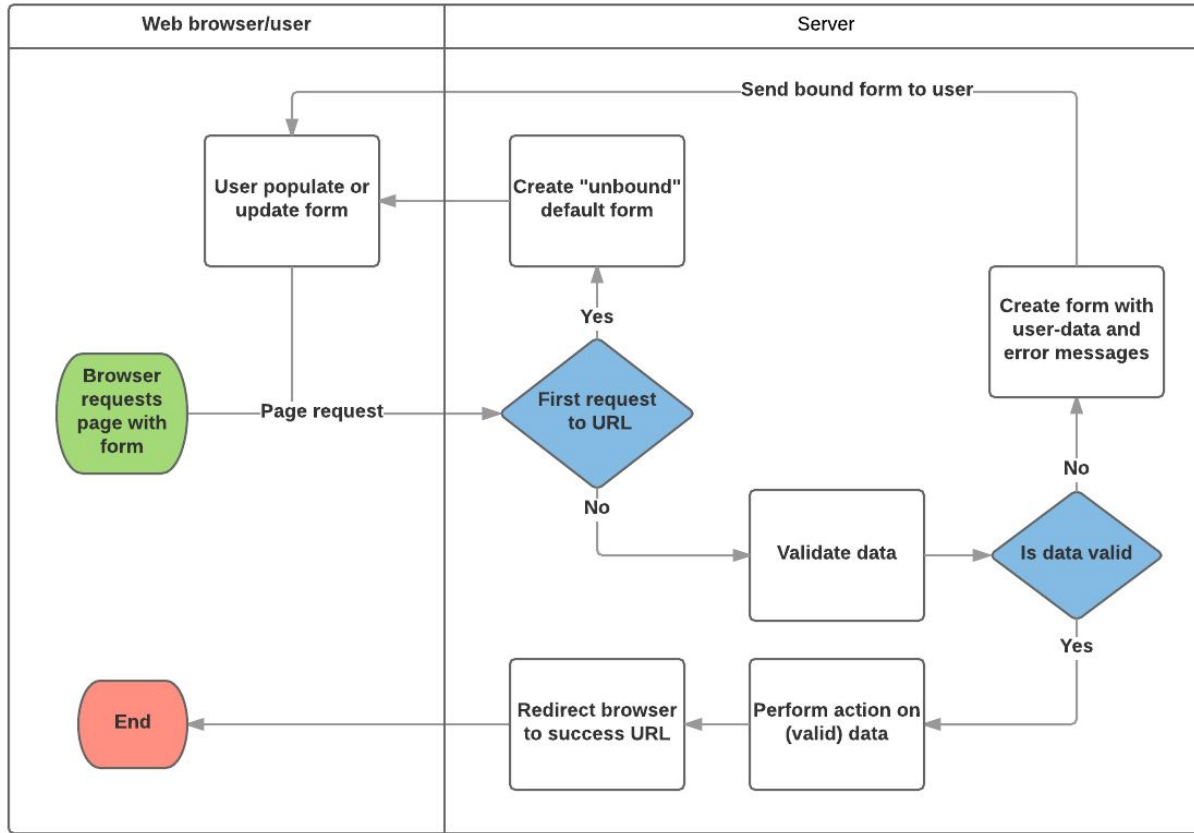
Science Fiction  
Fantasy  
Western  
French Poetry

Select a genre for this book

**Language:**

# Formularios

- Django provee de una serie de **herramientas y librerías** para crear formularios, procesar los envíos de información y generar las respuestas.
- Se centra en tres trabajos en torno a los formularios:
  - a. **Preparación** de los datos
  - b. **Presentación** HTML
  - c. **Recepción y procesamiento** de los formularios y los datos enviados por el cliente.



Source: <https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Forms>

# La clase Form

- Al igual que el modelo en Django define la estructura y lógica de los datos, la clase **Form** refleja el **contenido, la representación y el funcionamiento de un formulario**.
- Cumple las siguientes funciones:
  - a. **Generar el código HTML** de un formulario
  - b. **Procesar y validar los datos** de un formulario



# La clase Form

forms.py

```
from django import forms

class MyForm(forms.Form):
    # Crear un <input> con un label y una restricción de 100 caracteres
    nombre = forms.CharField(label='Introduce tu nombre', max_length=100)
    # Crear un <input> con type="email"
    email = forms.EmailField(label='Introduce tu email', max_length=20)
```

## forms.py

```
class LoginForm(forms.Form):  
    nombre = forms.CharField(label='Introduce tu nombre', max_length=100)  
    email = forms.EmailField(label='Introduce tu email', max_length=20)
```

## login.html

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

## views.py

```
def loginform(request):  
    form = MyForm()  
    return render(  
        request, 'login.html', {'form':form}  
    )
```



```
<label for="nombre">Introduce tu nombre: </label>  
<input id="nombre" type="text" name="nombre" maxlength="100" required>  
<label for="email">Introduce tu email: </label>  
<input id="email" type="text" name="email" maxlength="20" required>
```

**Nota:** Añade {% csrf\_token %} a todos los formularios de Django con método POST. Reducirá las posibilidades de hackeo por parte de usuarios malignos.

## unbound / bound

- Un elemento Form puede tener dos estados: bound y unbound
  - a. **Unbound**: el formulario está vacío, **no tiene asociados datos** introducidos por el usuario (por ejemplo, cuando mostramos por primera vez una página con un formulario vacío).
  - b. **Bound**: el formulario **tiene asociados datos asociados** (por ejemplo, cuando recibimos el formulario enviado y lo validamos).

# Lógica

```
def get_name(request):  
    if request.method == 'POST':  
        # Crear una instancia del formulario y asociar los datos recibidos:  
        form = LoginForm(request.POST)  
        # Validar los datos:  
        if form.is_valid():  
            # Procesar los datos mediante form.cleaned_data  
            # ...  
            # redireccionar a una nueva URL:  
            return HttpResponseRedirect('/dashboard/')  
        # Si el metodo es GET (o cualquier otro) crear un formulario vacío  
    else:  
        form = NameForm()  
        # Renderizar la plantilla con el formulario (con los datos anteriores o vacío)  
    return render(request, 'name.html', {'form': form})
```

## Acceder a la información

- Una vez el formulario es enviado y pasa la validación (`is_valid()` devuelve **True**) los datos quedan disponibles en el diccionario `form.cleaned_data`.

```
if form.is_valid():
    nombre = form.cleaned_data['nombre']
    email = form.cleaned_data['email']
    # ...
    # Realizar las operaciones deseadas
    # ...
    return HttpResponseRedirect('/dashboard/')

```

# Hands on!

Crea una nueva aplicación formada por dos vistas:

- Una vista que muestre un formulario con distintos campos
- Otra segunda vista que reciba los datos introducidos por el usuario en el formulario y los muestre por pantalla.



## Opciones para renderizar el formulario

- `{{ form.as_table }}` mostrará el formulario en **etiquetas** `<tr>` de tabla
- `{{ form.as_p }}` mostrará los elementos en **párrafos** `<p>`
- `{{ form.as_ul }}` mostrará los elementos en **etiquetas** `<li>` de una lista

```
<p><label for="nombre">Introduce tu nombre:</label>
  <input id="nombre" type="text" name="nombre" maxlength="100" required></p>
<p><label for="email">Introduce tu email:</label>
  <input type="email" name="email" id="email" maxlength="50" required></p>
```

# Renderizar manualmente el formulario

- Cada atributo estará disponible mediante `{{ form.nombre_campo }}`

```
{{ form.non_field_errors }}  
<div class="clase-formulario-A" >  
    {{ form.nombre.errors }}  
    <label for="{{ form.nombre.id_for_label }}">Introduce tu nombre:</label>  
    {{ form.nombre }}  
</div>  
<div class="clase-formulario-A" >  
    {{ form.email.errors }}  
    <label for="{{ form.email.id_for_label }}">Introduce tu email:</label>  
    {{ form.email }}  
</div>
```



## Generar un label completo

- Es posible crear el `<label>` completo utilizando `label_tag()`

```
<div class="clase-form">
  {{ form.nombre.errors }}
  {{ form.nombre.label_tag }}
  {{ form.nombre }}
</div>
```

# Visualización de errores

- La etiqueta `{{ form.non_field_errors }}` buscará errores en cada campo del formulario.
- Mostrar los errores de un campo: `{{ form.name_of_field.errors }}`
- El código generado es el siguiente (es posible personalizar más el output):

```
<ul class="errorlist">  
  <li>Sender is required.</li>  
</ul>
```

# Iterar por cada campo

- Es posible iterar por los campos mediante la etiqueta `{% for %}`
- `{{field.label}}` muestra el label
- `{{field.label_tag}}` muestra el label dentro de una etiqueta `<label>`
- `{{field.value}}` muestra el valor del campo
- `{{field.help_text}}` es el texto de ayuda asociado al campo

```
{% for field in form %}
    <div class="fieldWrapper">
        {{ field.errors }}
        {{ field.label_tag }} {{ field }}
        {% if field.help_text %}
            <p class="help">
                {{ field.help_text|safe }}
            </p>
        {% endif %}
    </div>
{% endfor %}
```

# Hands on!

Modifica la aplicación anterior para mostrar de forma más personalizada los campos del formulario, sus labels correspondientes y los posible errores.



# ModelForm

- Los formularios de tipo **ModelForm** adquieren la definición de **los campos desde el modelo**.
- Incluyen de serie **métodos adicionales** para guardar los formularios en base de datos.

# ModelForm

```
# Clase que define el modelo
class Autor(models.Model):
    nombre = models.CharField(max_length=100)
    apellidos = models.CharField(max_length=200)
    fecha_nacimiento = models.DateField(blank=True, null=True)

    def __str__(self):
        return self.name

# Formulario asociado a la clase
class AutorForm(ModelForm):
    class Meta:
        model = Autor
        fields = ['nombre', 'apellidos', 'fecha_nacimiento']
```

# Abreviaciones

- Utilizar **todos** los campos mediante `__all__`
- Utilizar **todos los campos a excepción de los indicados** mediante `exclude`

```
from django.forms import ModelForm

class AutorForm(ModelForm):
    class Meta:
        model = Author
        fields = '__all__'
```

```
class PartialAutorForm(ModelForm):
    class Meta:
        model = Author
        exclude = ['apellidos']
```

# Hands on!

Actualiza la aplicación anterior para utilizar ModelForm y las ventajas que ofrece.





# Sources

- Documentación oficial: <https://www.djangoproject.com>
- Mozilla MDN Web Docs: <https://developer.mozilla.org>