



VALIDACIONES EN DJANGO

RICARDO RAMOS Z.

VALIDACIONES EN EL MODELO

La validación en el modelo asegura que los datos en la base de datos siempre sean válidos, independientemente de cómo se hayan introducido. Esto incluye datos ingresados desde formularios, la consola de Django, API externas o cualquier otro lugar donde se manipulen los modelos.

Ventajas

- **Centralización:** Las reglas de validación están definidas en un único lugar: el modelo. Esto garantiza que todos los datos almacenados cumplan con las restricciones.
- **Aplicación global:** Las validaciones funcionan para cualquier entrada de datos, no solo formularios. Por ejemplo, si creas o actualizas un registro con `Model.objects.create()` o `Model.save()`, las validaciones se aplicarán.
- **Integridad de datos:** Protege los datos de tu base de datos, evitando inconsistencias.

Limitaciones

- **Flexibilidad limitada en contextos específicos:** No puedes definir reglas de validación personalizadas para un formulario en particular. Las validaciones aplican para todos los casos de uso del modelo.
- **Errores genéricos:** Los errores de validación son menos específicos para el usuario final, ya que no se presentan en un formulario específico, sino en el contexto del modelo.

VALIDACIONES EN EL FORMULARIO

La validación en el formulario asegura que los datos proporcionados por el usuario cumplan con las reglas antes de ser procesados o almacenados. Esto permite personalizar la experiencia del usuario y presentar errores específicos en la interfaz.

Ventajas

- **Contexto específico:** Puedes adaptar las reglas de validación a un formulario en particular. Útil si necesitas reglas diferentes para distintas vistas (por ejemplo, un formulario de creación puede tener validaciones distintas de uno de edición).
- **Interacción con el usuario:** Los errores de validación se pueden mostrar directamente en la plantilla de forma clara y amigable. Puedes dar retroalimentación específica para cada campo.
- **Flexibilidad:** Los formularios pueden incluir lógica de validación que no está directamente relacionada con la base de datos (por ejemplo, verificar que dos campos coincidan, como contraseñas).

Limitaciones

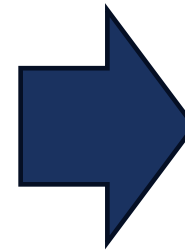
- **Alcance limitado:** Las validaciones en los formularios solo se aplican cuando el usuario interactúa con ellos. Si los datos se modifican desde la API, la consola o directamente en la base de datos, estas validaciones no se ejecutan.
- **Duplicación potencial:** Si tienes validaciones tanto en el modelo como en el formulario, puedes terminar duplicando lógica, lo que dificulta el mantenimiento.

CUADRO COMPARATIVO

Aspecto	Validar en el Modelo	Validar en el Formulario
Dónde se aplica	En toda la aplicación.	Solo en el contexto del formulario.
Propósito principal	Garantizar la integridad de los datos.	Mejorar la experiencia del usuario.
Uso en otras interfaces	Sí, aplica en API, consola, etc.	No, solo en la vista/formulario asociado.
Personalización por contexto	No, reglas globales para todos los casos.	Sí, puedes personalizar por formulario.
Nivel de interacción	Sin interacción directa con el usuario.	Diseñado para interactuar con el usuario.

VALIDACIÓN EN BASE AL MODELO

```
class Cliente(ModeloBaseGenerico):
    id_cliente = models.AutoField(primary_key=True)
    estatus_cliente = models.BooleanField("Estatus*", default=True,
                                         choices=ESTATUS_GEN)
    codigo_cliente = models.SmallIntegerField("Código", null=True, blank=True)
    nombre_cliente = models.CharField("Nombre Cliente*", max_length=50)
    domicilio_cliente = models.CharField("Domicilio Cliente*",
                                         max_length=50)
    codigo_postal = models.CharField("Código Postal*", max_length=5)
```



Validaciones en los atributos del modelo

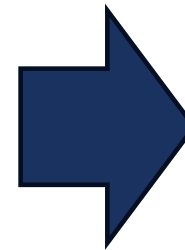
```
def clean(self):
    super().clean()

    # Diccionario contenedor de errores
    errors = {}

    # Convertir a string los valores de los campos previo a la validación
    telefono_str = str(self.telefono_cliente) if self.telefono_cliente else ''
    movil_cliente_str = str(self.movil_cliente) if self.movil_cliente else ''
    sub_cuenta_str = str(self.sub_cuenta) if self.sub_cuenta else ''

    try:
        validar_cuit(self.cuit)
    except ValidationError as e:
        # Agrego el error al diccionario errors
        errors['cuit'] = e.messages

    if not re.match(r'^\+?\d{0,14}$', telefono_str):
        errors.update({'telefono_cliente': 'Debe indicar sólo dígitos numéricos positivos, \
            mínimo 1 y máximo 15, el signo + y espacios.'})
```



Validaciones en el método clean() del modelo. Se pueden usar expresiones regulares

VALIDACIÓN GENÉRICA PARA LAS PLANTILLAS

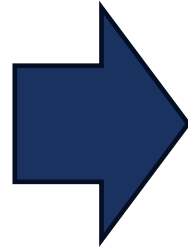
Las plantillas de crear o actualizar registros de los CRUDs, se hacen en base a la plantilla genérica maestro_form.html

maestro_form.html

include:
modal_errors.html
include:
modal_fields_requirements.html

Script

Que captura las excepciones de las validaciones, coloca los bordes de los elementos en rojo y coloca el foco en el primer elemento del error




Plantillas del modal que muestra los errores y el modal que muestra la ayuda para llenar los inputs del form



Script de JavaScript que se encarga de controlar las validaciones.

EXPRESIONES REGULARES

Son patrones de búsqueda que permiten verificar si un texto cumple con ciertas reglas. En Python, las expresiones regulares se manejan con el módulo `re`. En Django, puedes usarlas para validar campos de formularios cuando necesitas asegurarte de que los datos ingresados por el usuario sigan un formato específico. Los patrones básicos son:

Símbolo	Descripción	Ejemplo de patrón	Coincide con
<code>.</code>	Cualquier carácter (excepto nueva línea)	<code>a.b</code>	<code>"acb"</code> , <code>"a8b"</code>
<code>\d</code>	Dígito (0-9)	<code>\d{3}</code>	<code>"123"</code> , <code>"987"</code>
<code>\w</code>	Caracter alfanumérico (<code>a-z</code> , <code>0-9</code> , <code>_</code>)	<code>\w+</code>	<code>"hello"</code> , <code>"abc123"</code>
<code>\s</code>	Espacio en blanco (incluye tabulaciones)	<code>a\s+b</code>	<code>"a b"</code> , <code>"a b"</code>
<code>^</code>	Inicio de una cadena	<code>^Hola</code>	<code>"Hola mundo"</code> (al inicio)
<code>\$</code>	Final de una cadena	<code>mundo\$</code>	<code>"Hola mundo"</code> (al final)
<code>*</code>	Cero o más repeticiones	<code>ab*c</code>	<code>"ac"</code> , <code>"abc"</code> , <code>"abbc"</code>
<code>+</code>	Una o más repeticiones	<code>ab+c</code>	<code>"abc"</code> , <code>"abbc"</code>
<code>?</code>	Cero o una repetición	<code>ab?c</code>	<code>"ac"</code> , <code>"abc"</code>
<code>{n}</code>	Exactamente <code>n</code> repeticiones	<code>\d{4}</code>	<code>"1234"</code>
<code>[]</code>	Cualquier carácter dentro de los corchetes	<code>[aeiou]</code>	<code>"a"</code> , <code>"e"</code>
<code> </code>	 Alternativa (o lógico)	Alternativa (o lógico)	<code>`a</code>

EJEMPLO DE VALIDACIÓN DE EXPRESIONES REGULARES

Validar un número telefónico, debe empezar por el signo +, a continuación, uno, dos o tres dígitos, luego un espacio en blanco y luego, 9 o 10 dígitos, pero no pueden empezar con el dígito cero.

```
import re

def validar_telefono(telefono):
    # Actualización del patrón: código de país puede tener 1, 2 o 3 dígitos
    patron = r"^\+\d{1,3} [1-9]\d{8,9}$"
    if re.match(patron, telefono):
        return True
    else:
        return False
```


EXPLICACIÓN DE VALIDACIÓN DE EXPRESIONES REGULARES

Validar un número telefónico, debe empezar por el signo +, a continuación, uno, dos o tres dígitos, luego un espacio en blanco y luego, 9 o 10 dígitos, pero no pueden empezar con el dígito cero.

```
patron = r"^+\d{1,3} [1-9]\d{8,9}$"
```

El prefijo r evita que Python interprete los caracteres de escape, como \

Elemento	Descripción
<code>^</code>	Marca el inicio de la cadena.
<code>\+</code>	Coincide con el carácter literal <code>+</code> , obligatorio al inicio del número.
<code>\d{1,3}</code>	Coincide con 1, 2 o 3 dígitos consecutivos después del signo <code>+</code> (código de país).
<code> </code> (espacio)	Obliga a que haya un espacio después del código de país.
<code>[1-9]</code>	El primer dígito del número principal debe estar entre 1 y 9 (no puede ser 0).
<code>\d{8,9}</code>	Coincide con 8 o 9 dígitos consecutivos después del primer dígito (para un total de 9 o 10).
<code>\$</code>	Marca el final de la cadena, asegurando que no haya caracteres adicionales.