Contenidos a Trabajar

1. Crear formularios a partir de modelos
   - ModelForm
   - Tipos de campo
   - Validación en un ModelForm
   - Sobreescribiendo el método clean()
   - El método sabe ()
   - Forma herencia

Crear formularios a partir de modelos

ModelForm

clase ModelForm

Si está creando una aplicación basada en una base de datos, es probable que tenga formularios que se correspondan estrechamente con los modelos de Django. Por ejemplo, es posible que tenga un modelo **BlogComment** y desee crear un formulario que permita a las personas enviar comentarios. En este caso, sería redundante definir los tipos de campo en su formulario, porque ya ha definido los campos en su modelo.

Por esta razón, Django proporciona una clase auxiliar que te permite crear una clase **Form** a partir de un modelo de Django.

Por ejemplo:

```
>>> from django.forms import ModelForm
>>> from myapp.models import Article

# Create the form class.
>>> class ArticleForm(ModelForm):
...     class Meta:
...         model = Article
...         fields = ['pub_date', 'headline', 'content', 'reporter']

# Creating a form to add an article.
>>> form = ArticleForm()

# Creating a form to change an existing article.
>>> article = Article.objects.get(pk=1)
>>> form = ArticleForm(instance=article)
```

Tipos de campo

La clase generada **Form** tendrá un campo de formulario para cada campo de modelo especificado, en el orden especificado en el atributo **fields**.

Cada campo de modelo tiene un campo de formulario predeterminado

correspondiente. Por ejemplo, un **CharField** en un modelo se representa como un **CharField** en un formulario. Un modelo **ManyToManyField** se representa como un **MultipleChoiceField.**

En la documentación oficial tendrá todas las conversiones que se realizan: https://docs.djangoproject.com/en/3.2/topics/forms/modelforms/

Validación en un ModelForm

Hay dos pasos principales involucrados en la validación de un **ModelForm**:

1. Validando el formulario
2. Validando la instancia del modelo

Al igual que la validación de formularios normal, la validación de formularios modelo se activa implícitamente al llamar **is_valid()** o acceder al atributo **errors** y explícitamente al llamar **full_clean()**, aunque normalmente no usará este último método en la práctica.

La validación de **Model** (**Model.full_clean()**) se activa desde el paso de validación del formulario, justodespués de llamar al método **clean()** del formulario.

Sobreescribiendo el método clean()

Puede sobreescribir el método **clean()** en un formulario modelo para proporcionar una validación adicional de la misma manera que en un formulario normal.

Una instancia de formulario de modelo adjunta a un objeto de modelo contendrá un atributo **instance** que da acceso a sus métodos a esa instancia de modelo específica.

El método save()

Cada **ModelForm** también tiene un método **save()**. Este método crea y guarda un objeto de base de datos a partir de los datos vinculados al formulario. Una subclase de **ModelForm** puede aceptar una instancia de modelo existente como argumento de palabra clave **instance**; si se proporciona, **save()** actualizará esa instancia. Si no se proporciona, **save()** creará una nueva instancia del modelo especificado:

```
>>> from myapp.models import Article
>>> from myapp.forms import ArticleForm

# Create a form instance from POST data.
>>> f = ArticleForm(request.POST)

# Save a new Article object from the form's data.
>>> new_article = f.save()

# Create a form to edit an existing Article, but use # POST data to populate the
form.
>>> a = Article.objects.get(pk=1)
>>> f = ArticleForm(request.POST, instance=a)
>>> f.save()
```

Tenga en cuenta que si el formulario no ha sido validado, llamando **save()** lo hará marcando **form.errors**. Se generará **ValueError** si los datos en el formulario no se validan, es decir, si **form.errors** se evalúa como **True**.

Si un campo opcional no aparece en los datos del formulario, la instancia del modelo resultante usa el campo del modelo **default**, si lo hay, para ese campo. Este comportamiento no se aplica a los campos que usan **CheckboxInput**, **CheckboxSelectMultiple** o **SelectMultiple**(o cualquier widget personalizado cuyo método **value_omitted_from_data()**siempre devuelva **False**) ya que una casilla de verificación sin marcar y sin seleccionar **<select multiple>** no aparece en los datos del envío de un formulario HTML. Use un campo de formulario personalizado o un widget si está diseñando una API y desea el comportamiento de respaldo predeterminado para un campo que usa uno de estos widgets.

Este método **save()** acepta un argumento **commit** de palabra clave opcional, que acepta **True** o **False**. Si llama **save()** con **commit=False**, devolverá un objeto que aún no se ha guardado en la base de datos. En este caso, depende de usted llamar **save()** a la instancia del modelo resultante. Esto es útil si desea realizar un procesamiento personalizado en el objeto antes de guardarlo, o si desea utilizar una de las opciones de guardado de modelos especializadas. **commit** es **True** por defecto.

Otro efecto secundario del uso **commit=False** se ve cuando su modelo tiene una relación de muchos a muchos con otro modelo. Si su modelo tiene una relación de

muchos a muchos y especifica **commit=False** cuándo guarda un formulario, Django no puede guardar inmediatamente los datos del formulario para la relación de muchos a muchos. Esto se debe a que no es posible guardar datos de muchos a muchos para una instancia hasta que la instancia exista en la base de datos.

Seleccionando los campos a usar

Se recomienda encarecidamente que establezca explícitamente todos los campos que deben editarse en el formulario utilizando el atributo **fields**. Si no lo hace, puede generar fácilmente problemas de seguridad cuando un formulario permite inesperadamente que un usuario establezca ciertos campos, especialmente cuando se agregan nuevos campos a un modelo. Dependiendo de cómo se represente el formulario, es posible que el problema ni siquiera sea visible en la página web.

El enfoque alternativo sería incluir todos los campos automáticamente o eliminar solo algunos. Se sabe que este enfoque fundamental es mucho menos seguro y ha dado lugar a vulnerabilidades graves en los principales sitios web (p. ej ., GitHub).

Forma herencia

Al igual que con los formularios básicos, puede ampliarlos y reutilizar los **ModelForms** heredándolos. Esto es útil si necesita declarar campos adicionales o métodos adicionales en una clase principal para usar en varios formularios derivados de modelos.

Para más información sobre los campos seleccionados, como personalizar los widgets elegidos, herencia, las validaciones o los errores, por favor diríjase a la documentación oficial en

https://docs.djangoproject.com/en/3.2/topics/forms/modelforms/