Managers de Django

Materia:
Desarrollo de Aplicaciones WEB

Universidad:
UTT (Universidad Tecnologica de Tijuana)

Alumno:
Gerardo Lazcano Aguilar
Profesor:
Ray Brunett Parra Galaviz

Managers

¿Qué son?

Un Manager (o manejador) es la interfaz a través de la cual se proveen las operaciones de consulta o queries de la base de datos a los modelos de Django. Esto se refiere a ese *objects* que va después del nombre de tu modelo; *TuModelo.objects.all()* y *Tumodelo.objects.filter()*. Todos los modelos de Django tienen al menos un manager. Cada vez que usas el manejador de objetos (me referiré a él como manager de aquí en adelante) en una consulta a la base de datos usando el ORM de Django estás haciendo uso de su *object manager* predeterminado. Estos managers en Django pueden personalizarse para modificar los objetos que devuelve una consulta y podemos personalizarlos a nuestro gusto.

En Django, un gestor es una clase que proporciona operaciones de consulta de base de datos a un modelo de Django. Cada modelo de Django tiene al menos un gestor, que se define en la clase del modelo.

Lo más importante

- Los gestores son una interfaz que proporciona operaciones de consulta de base de datos a los modelos de Django.
- Cada modelo de Django tiene al menos un gestor, que se define en la clase del modelo.
- El gestor predeterminado de un modelo se llama objects.
- Los gestores personalizados se pueden utilizar para modificar el comportamiento de las consultas de base de datos.

Otros detalles importantes

- Los gestores se pueden heredar de clases base.
- Los gestores se pueden utilizar para crear consultas anidadas.
- Los gestores se pueden utilizar para crear consultas con paginación.

Ejemplos adicionales

Además de los ejemplos anteriores, aquí hay algunos ejemplos adicionales de cómo se pueden utilizar los gestores de Django:

- Para crear una consulta que devuelva solo los objetos de un modelo que coincidan con un determinado criterio, se puede utilizar el método filter().

```
# Obtener todos los objetos de Post que tengan el título "Mi publicación"
published_posts = Post.objects.filter(title="Mi publicación")
```

Para crear una consulta que devuelva solo los objetos de un modelo ordenados por un determinado campo, se puede utilizar el método order_by().

```
# Obtener todos los objetos de Post ordenados por su fecha de publicación
published_posts = Post.objects.order_by("-published_on")
```

Para crear una consulta que devuelva solo los objetos de un modelo que se hayan creado o actualizado recientemente, se puede utilizar los métodos created_at y updated_at.

```
# Obtener todos los objetos de Post creados en los últimos 7 días
```

```
published_posts = Post.objects.filter(created_at__gte=datetime.now() - timedelta(days=7))
```

Conclusión

Los gestores de Django son una herramienta poderosa que permite a los desarrolladores realizar consultas de base de datos de forma eficiente y flexible.

Ejemplos adicionales en español

- Modificando el manager por defecto

Python

```
from django.db import models


class Post(models.Model):
    title = models.CharField(max_length=255)
    content = models.TextField()

    # Cambiamos el nombre del manager por defecto
    stem = models.Manager()

    # Obtenemos todos los objetos de Post
    posts = Post.stem.all()
```

content_copy

- Agregando métodos a un manager personalizado

Python

```
from django.db import models
```

```
class Post(models.Model):
    title = models.CharField(max_length=255)
    content = models.TextField()

    # Creamos un manager personalizado
    class PublishedPostManager(models.Manager):
        def get_queryset(self):
            return super().get_queryset().filter(published=True)

    # Obtenemos todos los objetos de Post publicados
    published_posts = Post.PublishedPostManager.all()
```

content_copy

- Modificando los QuerySets iniciales del Manager

Python

```
from django.db import models

class Videogame(models.Model):
    title = models.CharField(max_length=255)
    company = models.CharField(max_length=255)

    # Creamos un manager personalizado
    class SquarenixVideogamesManager(models.Manager):
```

```
    def get_queryset(self):
        return super().get_queryset().filter(company='squarenix')

# Asignamos el manager personalizado al modelo
squarenix_videogames = SquarenixVideogamesManager()

# Obtenemos todos los videojuegos de Squarenix
squarenix_videogames = Videogame.squarenix_videogames.all()
```

Bibliografía

Django. (20 de 07 de 2023). *Managers*. Recuperado el 01 de 11 de 2023, de Django: https://docs.djangoproject.com/en/4.2/topics/db/managers/

Zepeda, E. (28 de Mayo de 2021). *Managers o manejadores personalizados en Django*. Recuperado el 01 de 11 de 2023, de Coffee bytes: https://coffeebytes.dev/managers-o-manejadores-personalizados-en-django/