Inheritance in django

Castro Soberanes Luis Alexis

Universidad Tecnológica de Tijuana.
TSU. Tecnologías de la información.
Entornos virtuales y negocios digitales.
Docente:Parra Galaviz Ray Brunett

Introduction

In the following document, you will find research on the application of inheritance in Django. This research addresses several relevant topics related to inheritance, including its definition. Additionally, it presents a detailed exploration of various functions associated with inheritance, accompanied by corresponding images to facilitate understanding.

After analyzing these functions, the document reveals the benefits of using inheritance in Django. Furthermore, it explores another important topic called 'polymorphism,' all of which contribute to a comprehensive understanding of the relevance of inheritance in Django.

What is Inheritance [1]

In Django, inheritance is a key feature that allows you to create and extend base models to reuse code and common field definitions across different models. This is achieved by creating a base class that other models can inherit its properties and fields from. Here's a description of how inheritance works in Django:

Base Class (Abstract Model):

To define a base class that serves as an abstract model, you should make use of Django's models.Model class and set the abstract option to True. This means that this class cannot be used to create instances directly in the database but can only be inherited by other models. [1]

```python
from django.db import models

class BaseModel(models.Model):
    common_field = models.CharField(max_length=100)
    # Other common fields and methods

    class Meta:
        abstract = True
```

Inheriting Models:

You can create other models that inherit from the base class and add additional fields specific to that model. Django will merge the fields from the base class with the additional fields defined in the child model. [1]

```python
class ChildModel(BaseModel):
    specific_field = models.IntegerField()
    # Other specific fields
```

Migrations:

When you define models that inherit from a base class, Django will generate the appropriate migrations to create the tables in the database. You can apply these migrations using makemigrations and migrate. [2]

```
python manage.py makemigrations
python manage.py migrate
```

Benefits:

Inheritance in Django allows code reusability and the creation of a more organized and maintainable database structure by defining common fields only once in the base class. It also simplifies the management and modification of common fields across multiple models, as changes only need to be made in one place (the base class). [1]

Polymorphism:

Inheritance in Django also enables the creation of polymorphic queries, meaning you can retrieve instances of child models through the base class and work with them in a unified way. [2]

Conclusion

From my perspective, I've come to the conclusion that inheritance is an essential feature that allows you to create a more organized and reusable database structure in Django. It involves defining a base model with common fields and then extending this base model in specific models. This simplifies the management of common fields across multiple models because you only need to define them once in the base class. It also enables the creation of polymorphic queries, making it easier to work with instances of specific models through the base class. Overall, in Django, inheritance is a powerful tool that promotes modularity and efficient maintenance in web application development

BIBLIOGRAFIAS IEEE

[1] [En línea]. Available: https://docs.djangoproject.com/en/4.1/topics/db/models/#model-inheritance.

[2] [En línea]. Available: https://pythondiario.com/2016/10/herencia-y-polimorfismo-en-python.html.