Üçüncü Bölüm

Modeller ve Veritabanlarına Derinlemesine Dalış

Django'da Modelleri Anlama

Django'da modeller, uygulamanızın veritabanıyla nasıl etkileşime gireceğini belirledikleri, veri yapılarını tanımladıkları, veri bütünlüğünü sağladıkları ve veritabanı işlemlerini basitleştirdikleri için çok önemlidir. Django modellerini derinlemesine anlamak, uygulamanızın veri katmanını etkili bir şekilde yönetmek ve hem ölçeklenebilirlik hem de sağlamlık sağlamak için hayati önem taşır.

Django Modellerini Keşfetmek

Django'daki modeller, veritabanı tablolarına karşılık gelen Python sınıflarıdır. Genellikle her Django uygulamasındaki models.py dosyasında bulunurlar. Modellerin bazı temel görevleri şunlardır:

1. **Yapıyı Tanımlama:** Modeller, metin, sayı ve tarih gibi alanlar belirleyerek veritabanı şemasını tanımlar.
2. **Veri Bütünlüğünün Sağlanması:** Modeller, doğrulama kuralları ve kısıtlamaları uygulayarak veri doğruluğunun ve güvenliğinin korunmasına yardımcı olur.
3. **Veritabanı İşlemlerini Kolaylaştırma:** Django'nun ORM'si (ObjectRelational Mapper) sayesinde modeller, doğrudan SQL etkileşimleri olmadan veritabanı işlemleri yapmanıza olanak tanır.

Basit Bir Model Oluşturma

Django'da bir model tanımlamak için django.db.models'den Model sınıfını genişletirsiniz. İşte basit bir modelin nasıl oluşturulacağına dair bir örnek:

```python
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=100)
    publication_date = models.DateField()
```

Bu Book modeli, kitap adı (title), yazar (author) ve yayın tarihi (publication_date) için alanlar içerir ve CharField ile DateField gibi veri alanı türlerini kullanır.

Alanları Seçeneklerle Özelleştirme

Django, model alanlarının davranışlarını özelleştirmek için çeşitli seçenekler belirlemenize olanak tanır:

- **Max_length:** Karakter alanlarının maksimum uzunluğunu belirler.
- **Null:** Alanın veritabanında NULL değer almasına izin verir.
- **Blank:** Formlarda alanın boş bırakılmasına izin verilip verilmeyeceğini belirler.
- **Choices:** Alan değerlerini belirli seçeneklerle sınırlandırır.

Kitap modelini, belirli seçeneklere sahip bir tür alanı içerecek şekilde şu şekilde değiştirebilirsiniz:

```python
class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=100)
    publication_date = models.DateField()
    genre = models.CharField(
        max_length=100,
        choices=[
            ('fiction', 'Fiction'),
            ('non-fiction', 'Non-Fiction'),
            ('romance', 'Romance'),
            ('mystery', 'Mystery'),
            ('sci-fi', 'Sci-Fi'),
        ],
        default='fiction'
    )
```

Model Kalıtımı

Django, model kalıtımını destekleyerek geliştiricilerin mevcut modelleri genişleterek özel sürümler oluşturmasına olanak tanır. İşte temel bir modeli genişletmenin bir örneği:

```
class SpecialBook(Book):
    critic_rating = models.IntegerField()
```

SpecialBook, Book'tan miras alır ve ek bir critic_rating alanı ekler.

Modellerle Etkileşim

Django'nun ORM'sini kullanarak, modelleriniz aracılığıyla çeşitli veritabanı işlemleri gerçekleştirebilirsiniz. Book modeliyle şu şekilde etkileşim kurabilirsiniz:

```
# Creating a new book instance
new_book = Book(title='Sample Book', author='Author Name', publication_date='2023-01-01')
new_book.save()

# Retrieving all books
all_books = Book.objects.all()

# Filtering books by author
books_by_author = Book.objects.filter(author='Author Name')
```

Sonuç

Django modelleri, herhangi bir Django uygulamasının temel taşıdır ve veri yapısını, bütünlüğünü ve etkileşimlerini ele almak için merkezi bir öneme sahiptir. Modellerin doğru şekilde anlaşılması ve kullanılması, geliştiricilerin ölçeklendirmeye hazır, verimli ve güçlü web uygulamaları oluşturmalarını sağlar. Django modellerinde uzmanlaşarak, uygulamalarınızın yalnızca işlevsel olmasını değil, aynı zamanda performans ve sürdürülebilirlik açısından optimize edilmesini de sağlayabilirsiniz.