Formlar

Django'daki formlar, otomatik form oluşturma, veri doğrulama ve güvenlik sağlayarak form oluşturmanın tekrarlanabilirliğinin çoğunu soyutlar. Formlar Django içinde sınıflar olarak tanımlanır, hangi modeli kullandıkları ve hangi alanları içerdikleri belirtilir.

Blog için bir form şöyle görünebilir:

```
from django import forms
from .models import Post

class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ('title', 'text',)
```

Administrative Interface

Django'nun en ünlü özelliklerinden biri otomatik olarak oluşturulan yönetici arayüzüdür. Bu sağlam, dinamik sistem, geliştiricilerin uygulama içinde tanımlanan modellerin şemasına göre uyarlanmış içeriği doğrudan yönetmelerine olanak tanır.

Geliştiriciler bu temel bileşenleri anlayarak kapsamlı, ölçeklenebilir ve sağlam web uygulamaları geliştirmek için Django'nun işlevselliğinden tam olarak faydalanabilirler. Django'nun yapılandırılmış framework'ü hızlı geliştirmeyi destekler ve uygulamaların hem ölçeklenebilir hem de sürdürülebilir olmasını sağlayarak onu her ölçekteki web geliştirme projeleri için en uygun seçenek haline getirir.

MVC ve MVT Mimarisi

Web geliştirmede, bir uygulamanın mimarisini etkili bir şekilde yönetmek, verilerini, iş mantığını ve kullanıcı arayüzlerini ele almak için çok önemlidir. Bu konuda yardımcı olan iki önemli mimari model MVC (Model- View-Controller) ve MVT'dir (Model-View-Template). Her iki çerçeve de bir uygulama içindeki endişeleri ayırmak için yapılandırılmıştır, böylece modülerliğini, sürdürülebilirliğini ve ölçeklenebilirliğini geliştirir.

MVC Mimarisine Genel Bakış

MVC, bir uygulamayı üç temel bileşene bölen (bunlar; model, görünüm ve denetleyici), programlama sorumluluklarını üç ayrı bölüme ayırarak karmaşık uygulamaların yönetimini basitleştiren, zamanla test edilmiş bir mimari modeldir.

Bu sadece kod yönetimini kolaylaştırmakla kalmaz, aynı zamanda geliştirmeyi de kolaylaştırır.

- Model: Model, uygulamanın verilerini ve ilgili iş mantığını yöneten merkezi bileşendir. Verileri ve verilere erişildiği ve verilerin güncellendiği koşulları temsil eder. Tipik olarak Model, veritabanı ile etkileşimleri yönetir, veri doğrulaması yapar veri alma ve güncelleme görevlerini yerine getirir.
- View: View bileşeni, verilerin kullanıcıya gösterilmesi ve kullanıcı girdilerinin toplanması dahil olmak üzere uygulamanın tüm kullanıcı arayüzü mantığıyla görevlendirilmiştir. Görünüm, model verilerini kullanıcı dostu bir biçime dönüştürür ve kullanıcılarla görsel düzeyde arayüz oluşturur.
- Controller: Denetleyici, Model ve Görünüm arasında bir kanal görevi görür, model nesnelerine veri akışını yönetir ve veriler değiştiğinde görünümü günceller. Kullanıcı girdilerini alır ve bunları model veya görünüm için komutlara dönüştürür.

İşte MVC'yi göstermek için basit bir örnek:

```
# Model
class BookModel:
    def __init__(self, title, author):
        self.title = title
        self.author = author

# View
class BookView:
    def display_book_details(self, title, author):
        print(f"Book: {title} by {author}")
```

```python
# Controller
class BookController:
    def __init__(self, model, view):
        self.model = model
        self.view = view

    def set_book_title(self, title):
        self.model.title = title

    def get_book_title(self):
        return self.model.title

    def set_book_author(self, author):
        self.model.author = author

    def get_book_author(self):
        return self.model.author

    def update_view(self):
        self.view.display_book_details(self.model.title, self.model.author)

# Usage
model = BookModel("1984", "George Orwell")
view = BookView()
controller = BookController(model, view)
controller.update_view()
```