

Sınıf Tabanlı Görünümler

- Sınıf tabanlı görünüm, Django'nun temel Görünüm sınıfını alt sınıflandırarak tanımlanır ve HTTP isteklerini işlemek için ortak yöntemlere erişim sağlar.
- Örneğin, bir HTTP GET isteğini işlemek için bir Get yöntemi tanımlanabilir ve veritabanından veri sorgulanarak bir HTML şablonuna yerleştirilebilir.

Genel Sınıf Tabanlı Görünümler

- Django, yaygın görevleri hızlandırmak için genel sınıf tabanlı görünüm sunar; örneğin, ListView ve DetailView gibi.
- Bu görünüm, geliştiricilerin yalnızca birkaç satır kod yazarak nesne detaylarını veya listelerini görüntülemelerini sağlar.

Avantajlar ve Dezavantajlar

- İşlev tabanlı görünüm, basit yazım ve anlaşılabilirlik sunar, ancak genişletme ve yeniden kullanım zorluğu vardır.
- Sınıf tabanlı görünüm, yeniden kullanılabilirlik ve genişletilebilirlik sağlar, ancak kodun okunabilirliğini zorlaştırabilir ve gizli kodlar içerebilir.

Kimlik Doğrulama ve Yetkilendirme

- Kimlik doğrulama, kullanıcıların kimliklerini doğrulamak için kullanıcı adı ve şifre gibi kimlik bilgilerini kullanır.
- Yetkilendirme, kullanıcıların belirli kaynaklara erişim izinlerini kontrol eder; sistem yöneticileri, farklı roller veya gruplar tanımlayarak erişim izinlerini yönetir.

Django Kullanıcı Modeli

- Django, kullanıcı bilgilerini yönetmek için bir Kullanıcı modeli sağlar; bu model, kullanıcı adı, şifre ve e-posta gibi temel bilgileri içerir.
- Geliştiriciler, uygulama özel kullanıcılar tanımlamak için Kullanıcı modelini genişletebilirler.

Kullanıcı Girişi ve Çıkışı

- Kullanıcı girişi için bir web sayfası şablonu oluşturulur ve kullanıcı adı ile şifre alınarak POST isteği ile doğrulama yapılır.
- Kullanıcı çıkışı, basit bir logout yöntemi ile gerçekleştirilir; bu işlem, oturum bilgilerini temizler.

Statik Dosyaların Yapılandırılması

- Statik dosyalar için farklı klasörler oluşturulmalı; örneğin, HTML şablonları, resimler veya CSS dosyaları için alt klasörler oluşturulmalıdır.
- Her uygulama için statik dosyaların ad alanını oluşturmak amacıyla, uygulama adıyla aynı isme sahip alt klasörler oluşturulmalıdır.

Django Projesinde Statik Dosyaların Yönetimi

- Uygulama kök klasöründe "static" adında bir klasör oluşturularak uygulama spesifik statik dosyalar saklanır.
- Ek statik dosyalar için, STATICFILES_DIRS listesi kullanılarak dizinler tanımlanabilir.

Üretim Ortamında Statik Dosyaların Dağıtımı

- Tüm statik dosyaların tek bir konumda toplanması için STATIC_ROOT ve STATIC_URL tanımlanmalıdır.
- collectstatic komutu kullanılarak, statik dosyalar belirtilen STATIC_ROOT klasörüne toplanır ve dosya yapısı oluşturulur.

Django Uygulamalarının Dağıtımı

- Django uygulamaları, web sunucularında çalışabilmesi için WSGI (Web Server Gateway Interface) gibi arayüzlere ihtiyaç duyar. WSGI, Python uygulamaları ile web sunucuları arasında iletişim sağlar.
- ASGI (Asynchronous Server Gateway Interface) ise asenkron kodu destekler ve Django uygulamaları için alternatif bir arayüz sunar.

Bulut Dağıtım Seçenekleri

- Uygulamanızı bulutta dağıtmak için sanal makineler (VM), platform olarak hizmet (PaaS) veya konteynerleştirme yöntemlerini kullanabilirsiniz. PaaS, kodunuzu bir Git deposuna iterek dağıtım sürecini otomatikleştirir.
- Konteynerleştirme, Docker kullanarak uygulamanızı paketlemenizi ve Kubernetes gibi araçlarla dağıtmanızı sağlar.

Üretim İçin En İyi Uygulamalar

- Üretim ortamında, PostgreSQL veya MySQL gibi sağlam veritabanları kullanmak önerilir. SQLite, yüksek eşzamanlılık veya yoğun trafik için uygun değildir.
- Güvenlik için veritabanı kimlik bilgilerini çevresel değişkenlerde saklayın ve HTTPS kullanarak güvenli iletişim sağlayın. Ayrıca, yük dengelemesi ve izleme hizmetleri ile uygulamanızın performansını artırabilirsiniz.