Beşinci Bölüm

Şablonlar ve Kullanıcı Arayüzleri

Django Şablonlarına Giriş

Django şablonları, Django web çerçevesinin temel unsurlarıdır ve sunum katmanı ile uygulamanın çekirdek mantığı arasında net bir ayrım sağlarken dinamik HTML üretimine olanak tanır. Bu ayrım, sürdürülebilirliği artırır ve geliştiricilerin statik ve dinamik içeriği sorunsuz bir şekilde entegre etmesine olanak tanır.

Django Şablonlarının Amacı

Django şablonları, HTTP isteklerine yanıt olarak dinamik bir şekilde içerik üretir. Django mimarisinin sunum katmanında yer alan şablonlar, içeriğin nasıl yapılandırıldığını ve kullanıcıya nasıl sunulduğunu yönetir. Veri işleme ve sunum arasında belirgin bir ayrım sağlarlar ve bu da web uygulaması geliştirmeyi kolaylaştırmak için çok önemlidir.

Django Şablon Dilinin Temel Özellikleri

Django Şablon Dili (DTL), sunum mantığını oluşturmak için özel olarak tasarlanmış kapsamlı bir araç seti sunar.

web sayfaları dinamik olarak. DTL, geliştiricilerin Python kodunu doğrudan şablonlara eklemeden dinamik verileri HTML düzenlerine örmelerine olanak tanır.

Şablonlardaki Etiketler ve Filtreler

DTL'deki **etiketler** şablon içindeki mantığı yönetir, döngüleri, koşulları ve verileri oluşturmak için gereken diğer mantık işlemlerini ele alır:

```
<ul>
{% for book in book_list %}
    <li>{{ book.title }}</li>
{% endfor %}
</ul>
```

Filtreler, şablonlar içindeki değişkenlerin değiştirir ve

{{ }} işaretleyiciler:

```
<p>{{ "Hello World"|lower }}</p>
```

Bu filtre, **lower**, sayfada işlenmeden önce metni küçük harfe dönüştürür.

Şablon Kalıtımı

Şablon kalıtımı, geliştiricilerin ortak öğelere ve yapıya sahip bir temel şablon oluşturmasına olanak tanıyan güçlü bir DTL özelliğidir; bu şablon daha sonra farklı sayfalar için özel içerik sağlayan alt şablonlarla genişletilebilir.

Temel şablon (**base.html**):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>{% block title %}My Website{% endblock %}</title>
</head>
<body>
    <header>
        <h1>Welcome to My Website</h1>
    </header>
    <div>
        {% block content %}{% endblock %}
    </div>
    <footer>
        <p>Thank you for visiting!</p>
    </footer>
</body>
</html>
```

Çocuk şablonu:

```
{% extends "base.html" %}

{% block title %}Home Page{% endblock %}

{% block content %}
<p>This is the home page.</p>
{% endblock %}
```

Django'da Şablonları Yapılandırma

Şablon ayarları, şablonların nasıl yönetileceğini belirleyen Django **settings.py** dosyasında belirtilir:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Şablon Geliştirmek için En İyi Uygulamalar

- **Endişelerin net bir şekilde ayrılmasını sağlayın**: Şablonların iş mantığı yerine ağırlıklı olarak sunuma odaklandığından emin olun.
- **Kodun yeniden kullanımından yararlanın**: Yeniden kullanılabilir kod parçacıkları için include ve ortak yapısal düzenler için inheritance kullanın.
- **Güvenliği artırın**: XSS (siteler arası komut dosyası oluşturma) güvenlik açıklarına karşı koruma sağlamak için her zaman kullanıcı tarafından oluşturulan içeriğin kaçtığından emin olun.

Sonuç

Django şablonları, dinamik ve ilgi çekici web uygulamaları oluşturmayı amaçlayan geliştiriciler için temeldir. Etkili kullanıcı arayüzü yönetimini ve sofistike veri sunumunu kolaylaştırırlar. Django şablonlarını anlamak ve ustalıkla kullanmak, etkileyici kullanıcı deneyimleri sunmak ve temiz, düzenli bir kod tabanını korumak için çok önemlidir.

Şablon Kalıtımı

Şablon kalıtımı, Django'da web uygulama arayüzlerinin organizasyonunu ve bakımını geliştiren önemli bir . Tanımlayarak

Temel bir şablondaki ortak yapısal öğeler, Django geliştiricilerin bu yapıyı birden fazla sayfada genişletmesine olanak tanıyarak, gereksizliği önlerken tüm uygulama genelinde uyumlu görünüm sağlar.

Django'nun Şablon Kalıtımını Anlamak

Django'daki şablon kalıtımı, sitenizin düzeninin tüm evrensel öğelerini (üstbilgiler, altbilgiler, gezinme ve diğer paylaşılan bileşenler) içeren temel bir şablon oluşturulmasına olanak tanır.

Temel Şablon

Temel şablon, türetilmiş şablonlar tarafından geçersiz kılınabilen bloklar halinde tanımlanmış statik ve dinamik öğeler içeren web sitesi için bir çerçeve görevi görür.

Temel şablon örneği (**base.html**):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}Default Title{% endblock %}</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <h1>{% block header %}Welcome{% endblock %}</h1>
        <nav>
            <ul>
                <li><a href="/">Home</a></li>
                <li><a href="/about/">About</a></li>
                <li><a href="/contact/">Contact</a></li>
            </ul>
        </nav>
    </header>
    <main>
        {% block content %}Default content goes here{% endblock %}
    </main>
    <footer>
        {% block footer %}Copyright © 2023 My Website{% endblock %}
    </footer>
</body>
</html>
```

Bu şablon, **alt** şablonlarda sayfanın belirli alanlarını özelleştirmek için geçersiz kılınabilecek birkaç blok (**başlık**, **üstbilgi**, **içerik**, **altbilgi**) tanımlar.

Çocuk Şablonlarının Uygulanması

Alt şablonlar temel şablondan yararlanır ve bloklarını belirli sayfalara uyarlanmış özel içeriklerle özelleştirir.

"Hakkında" sayfası alt şablonu örneği:

```
{% extends "base.html" %}

{% block title %}About Us{% endblock %}

{% block content %}
    <h2>About Us</h2>
    <p>Learn about our mission and values on this page.</p>
{% endblock %}
```

Bu şablon **base.html**'i genişletir ve "Hakkımızda" sayfasıyla ilgili belirli bilgileri yansıtmak için **başlık** ve **içerik** bloklarını değiştirir.

Şablon Kalıtımının Faydaları

1. **Sayfalar Arasında Tutarlı Tasarım:** Temel bir şablonla, tüm sayfalar ortak bir tasarım çerçevesini paylaşarak tek tip bir kullanıcı deneyimini teşvik eder.
2. **Azaltılmış Bakım Çabası:** Navigasyon menüleri veya altbilgiler gibi evrensel olarak paylaşılan bileşenlere yönelik güncellemeler tek bir yerden yapılabilir ve bakım süreci kolaylaştırılabilir.
3. **Sayfa Tasarımında Geliştirilmiş Esneklik:** Sayfalar ortak bir düzeni paylaşsa da, önceden tanımlanmış blokları geçersiz kılarak benzersiz içerik ve stilleri gerektiği gibi görüntüleyebilirler.

En İyi Uygulamalar

- **Açıklayıcı blok adları kullanın:** Blokları açık ve mantıklı bir şekilde adlandırmak, şablon yapısının anlaşılmasına ve korunmasına yardımcı olur.
- **Genel temel şablonları koruyun:** Temel şablonların, çeşitli alt şablonlar tarafından değişiklik yapılmadan genişletilebilecek kadar çok yönlü olmasını sağlayın.
- **Şablonları iyi organize edin:** Şablon dosyalarını proje dizininizde net bir şekilde yapılandırmak, özellikle büyük projelerde bunları etkili bir şekilde yönetmenize yardımcı olur.

Sonuç

Şablon kalıtımı, Django'da bir web uygulaması genelinde tutarlı tasarım modellerinin yönetimini basitleştiren güçlü bir stratejidir. Merkezi bir temel şablon kullanarak ve bunu farklı sayfalara genişleterek, geliştiriciler tasarım bütünlüğünü sağlayabilir ve karmaşık uygulamaların bakımını kolaylaştırabilir. Şablon kalıtımını etkili bir şekilde kullanmak, Django'da ölçeklenebilir ve bakımı yapılabilir web uygulamaları oluşturmayı amaçlayan geliştiriciler için çok önemlidir.

Statik Dosyaları Kullanma (CSS, JavaScript)

Django'da CSS, JavaScript ve resimler gibi statik dosyaları etkili bir şekilde yönetmek, hem görsel olarak çekici hem de işlevsel olarak sağlam web uygulamaları geliştirmek için çok önemlidir. Django, web sitelerinin tasarımı ve etkileşimi için gerekli olan bu kaynakları işlemek için kapsamlı bir çerçeve sağlar.

Django'da Statik Dosyalara Genel Bakış

Statik dosyalar, sunucu tarafından dinamik olarak oluşturulmayan ve stil sayfaları, komut dosyaları ve medya dosyalarını içeren varlıklardır. Django'nun bu dosyaları yönetme sistemi, kurulumlarını kolaylaştırır ve özellikle geliştirme ortamlarından üretim ortamlarına geçiş sırasında sorunsuz bir şekilde entegre edilmelerini sağlar.

Django'da Statik Dosyaları Yapılandırma

Statik dosyaları düzgün bir şekilde yönetmek için Django, settings.py dosyasında bu dosyaların nasıl ele alınacağını tanımlayan özel ayarlar gerektirir:

- **STATIC_URL**: Statik dosyalara erişmek için kullanılan URL önekini belirtir.
- **STATICFILES_DIRS**: Django'nun her uygulamada standart 'static' klasörlerinin ötesinde ek statik dosyalar arayacağı dosya sistemi dizinlerini listeler.
- **STATIC_ROOT**: Dağıtım için **collectstatic** komutu kullanılarak statik dosyaların toplandığı yolu tanımlar.

settings.py içinde örnek yapılandırma:

```
STATIC_URL = '/static/'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'extra_static'),
]
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
```

Bu kurulum şunları sağlar:

- **STATIC_URL** statik dosyalar için web üzerinden erişilebilir bir yol sağlar.
    - **STATICFILES_DIRS** Django'nun statik dosyaları belirtilen dizinlerden sunmasını sağlar.
    - **STATIC_ROOT**, üretime hazırlık aşamasında statik dosyalar için merkezi konumdur.

Geliştirme Sırasında Statik Dosyaların Sunulması

Geliştirme aşamasında, Django statik dosyaları doğrudan her uygulamanın statik dizininden ve **STATICFILES_DIRS** içinde belirtilenlerden sunabilir. Bu işlem, **runserver** komutu kullanıldığında Django'nun geliştirme sunucusu tarafından otomatik olarak gerçekleştirilir.

Statik dosyaların bir şablona dahil edilmesi aşağıdaki şekilde yapılabilir:

```
{% load static %}
<link rel="stylesheet" href="{% static 'css/style.css' %}">
<script src="{% static 'js/app.js' %}"></script>
```

Statik dosyaların yollarına **STATIC_URL** için **{% static %}** şablon etiketi kullanılır ve doğru URL'lerin oluşturulmasını sağlar.

Statik Dosyaların Üretim için Hazırlanması

Statik dosyaları üretime hazır hale getirmek için **collectstatic** komutu tüm **statik** varlıkları **STATIC_ROOT** dizininde toplayarak dağıtımlarını optimize eder:

```
python manage.py collectstatic
```

Bu adım, statik dosyaları tek bir konumda birleştirerek üretim sunucusundan verimli bir şekilde teslim edilmeye hazır hale getirir.

Statik Dosya Yönetimi için En İyi Uygulamalar

1. **Versiyonlama**: Önbelleğe almayı yönetmek ve kullanıcıların statik dosyaların en güncel sürümlerine erişmesini sağlamak için dosya sürümleme stratejilerini kullanın.
2. **Optimizasyon**: Boyutlarını en aza indirmek için CSS ve JavaScript dosyalarını sıkıştırın, bu da web sitesi yükleme hızlarını artırmaya yardımcı olur.
3. **Organizasyonel Yapı**: Yönetim ve erişilebilirlik kolaylığı sağlamak için statik dosyaları her uygulamanın statik dizini içinde belirlenmiş klasörlerde iyi organize edin.

Sonuç

Statik dosyaların doğru yönetimi, Django kullanan web uygulamalarının başarılı bir şekilde geliştirilmesinin ayrılmaz bir parçasıdır. Geliştiriciler, Django'yu bu dosyaları işleyecek şekilde doğru bir şekilde yapılandırarak uygulamanın geliştirme aşamasından üretime sorunsuz bir şekilde geçmesini ve yüksek kaliteli bir kullanıcı deneyimi sunmasını sağlayabilir. Statik kaynakların etkin yönetimi, performansı optimize etmek ve Django tabanlı uygulamaların işlevselliğini artırmak için gereklidir.

Pratik Örnekler: Ana Sayfa Oluşturma

Bir ana sayfa oluşturmak, özellikle Django çerçevesini kullanırken web geliştirmede temel bir adımdır. Bu kılavuz, Django'nun temel bileşenlerini entegre ederek basit bir ana sayfa oluşturma sürecini detaylandırmaktadır: URL yapılandırmaları, görünümler ve şablonlar.

Adım 1: Proje ve Uygulama Kurulumu

Öncelikle Django'nun kurulu olduğundan emin olun. **django-admin startproject myproject** komutuyla yeni bir proje başlatın ve ardından ana sayfayı barındırmak için yeni bir uygulama oluşturun:

```
python manage.py startapp home
```

Bu, views.py, models.py gibi temel dosyaları ve ana sayfayı oluşturmak için gerekli olan şablonlar için bir dizin oluşturacaktır.

Adım 2: URL'leri Yapılandırma

Django tarafından tanınmasını sağlamak için yeni uygulamayı **settings.py**'deki **INSTALLED_APPS**'ekleyin:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'home',  # Ensure your app is listed here
]
```

Trafiği uygulamanıza yönlendirmek için **urls.py**'de ana URL modelini tanımlayın:

```
from django.urls import path, include  # Ensure to include the necessary import

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('home.urls')),  # Route the base URL to the home app
]
```

Ardından, bir **urls.py** dosyası oluşturarak uygulamanızda URL yapılandırmasını ayarlayın:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.homepage_view, name='homepage'),  # Define the URL for the homepage
]
```

Adım 3: Görünümün Uygulanması

Uygulamanızın **views.py dosyasında**, ana sayfayı oluşturmak için görünüm işlevini tanımlayın:

```
from django.shortcuts import render

def homepage_view(request):
    return render(request, 'home/homepage.html', {
        'title': 'Welcome to My Site!',
        'message': 'Hello, welcome to my fabulous site!'
    })
```

Bu görünüm homepage.html dosyasını işler ve başlık ve mesaj içeren bir bağlam iletir.

Adım 4: Ana Sayfa Şablonunun Oluşturulması

Uygulamanızın içinde bir **templates/home** dizini oluşturun ve bu dizini **homepage.html** ile doldurun. Bu yapılandırılmış yaklaşım, Django'nun şablonun belirli bir uygulama ile ilişkisini tanımasına yardımcı olur.

İşte temel bir **homepage.html** düzeni:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ title }}</title>
</head>
<body>
    <h1>{{ title }}</h1>
    <p>{{ message }}</p>
</body>
</html>
```

Adım 5: Ana Sayfanızı Çalıştırma

Ana sayfanızı çalışırken görmek için Django geliştirme sunucusunu başlatın:

```
python manage.py runserver
```

Belirtilen başlık ve mesajla birlikte yeni oluşturduğunuz ana sayfanızı görmek için web tarayıcınızda **http://127.0.0.1:8000/** adresini ziyaret edin.

Sonuç

Django'da bir ana sayfa geliştirmek, URL yönlendirme, görünüm kurulumu ve şablon oluşturma işlemlerinin entegre edilmesini içerir. Bu temel kurulum, hem işlevsel hem de daha karmaşık projeler için uyarlanabilir bir ana sayfa geliştirmek için bir şablon sağlar. Geliştiriciler bu ilk adımları anlayarak

Django uygulamalarını geliştirerek bu temel üzerine etkili bir şekilde inşa edebilir ve genişletebilir.