

Django Web Çerçevesi

Mete Alpaslan KATIRCIOĞLU, Emre YÜCE

mete.alpaslan@portakalteknoloji.com, emre.yuce@portakalteknoloji.com

Özet: Django Python Programlama Dili için hazırlanmış ve BSD lisansı ile lisanslanmış yüksek seviyeli bir web çatısıdır. Basit kurulumu ve kullanımı, detaylı hata raporu sayfaları ve sunduğu yeni arayüz kodlama yöntemleriyle diğer sunucu yazılımı ve çatılardan kendini ayırmaktadır. Django projesinin temel hedefi karmaşık bir yapıda olan ve bir veritabanı kullanan web uygulamalarının gerçekleştirimini kolaylaştırmaktır. Django yeniden kullanılabilirlik, modülerlik, hızlı geliştirme süreci ve DRY prensiplerini sonuna kullanma politikasına sahip bir yapıda tasarlanmaktadır. Bu yazı Django Web Uygulama Çatısını kullanarak web servislerinin geliştirme sürecini MVC web mimarisini anlatmaktadır.

Anahtar Sözcükler: Django Web Framework, Python, Apache

1.Giriş

Django Python Programlama Dili için hazırlanmış ve BSD lisansı ile lisanslanmış, modüler, pragmatik bir tasarımla hızlı bir şekilde web uygulamaları geliştirmenize olanak sağlayan açık kaynak kodlu bir platformdur.

Django Web Uygulama Çatısında yeniden kullanılabilirlik, modülerlik, hızlı geliştirme süreci ve DRY(Don't Repeat Yourself) prensiplerini sonuna kadar kullanma politikasına sahip bir yapıda tasarlanmasıyla uygulama geliştirme sürecini olabildiğince otomatize etmesi ve böylece tekrarları ortadan kaldırması en büyük artılarından.

Django Çerçevesinin temel bileşenlerinin sunduğu en önemli özellikler;

- Modellerin yaratılması için nesne ile ilişkisel eşleme
- Son kullanıcılar için tasarlanmış hoş görünümlü yönetici arabirimi
- Zarif URL tasarımı
- Tasarımı kolay şablon dili
- Önbelleğe alma sistemidir.

Django mimari olarak MVC (Model-View-Controller) yapısını kullanarak geliştirme sürecinde verinin tanımlanması ve erişilmesi için yazılan kodun (model) istek yönlendirme mantığından (controller) yani kullanıcı arayüzünden(view) ayrılmasını sağlayan bir yazılım geliştirme yöntemi tanımlar. Web Çatısının bileşenlerinin birbiriyle Djangodaki gibi gevşek bir şekilde bağlandığı çalışma ortamında projedeki geliştirici uygulamanın bir bölümündeki adres ilgili yapmak istediği değişiklikleri altındaki implemantasyona etkilemeden yapabilirken, bir taraftan da tasarımcı sayfaların görünümünü Python kodlarına dokunmadan değiştirebilme ya da veritabanı yöneticisinin veritabanı tablolarını yeniden isimlendirebilme fırsatı olur. Django hızlı bir geliştirme süreci sağlar.

2.Django Projeleri ve Uygulamaları

Bir Django Projesini başlatmak için aşağıda anlatıldığı gibi django-admin startproject komutunu kullanın:

Liste 1. Bir projenin başlatılması

```
$ django-admin.py
startproject ab08
```

Eğer sunum dizinin içeriğini listelersek:

Liste 2. Bir projenin içeriği

```
$ ls -l sunum/
__init__.py
manage.py
settings.py
urls.py
```

- `__init__.py` dosyası bu klasörle birlikte bundan sonra oluşturacağınız tüm uygulama modüllerinde de yer alacak boş bir dosyadır. Bulduğunuz klasörün aynı zamanda bir Python paketi olduğunu gösteren `__init__.py`, aynı zamanda paketin ilk çalıştırılan dosyasıdır.
- `manage.py`, bundan sonraki tüm aşamalarda yeni uygulama modülü yaratırken; veritabanı tablolarını, ilişkilerini yaratırken kullanacağımız dosyadır.
- `urls.py` dosyasını istek yapılan adresin görünümle(view) eşleştirme yapabilmemiz için gereklidir.
- `settings.py` dosyasıyla yarattığımız projenin yapılandırılmasının yapabilirsiniz.

Liste 3. `manage.py startapp`'nin kullanılması

```
$ python manage.py startapp bildiri
```

Bu komut, modelleriniz için bir Python modülü ve görünümünüz için bir Python modülü olan temel bir uygulama bir yaratır. “bildiri”

dizini aşağıdaki dosyaları içerecektir:

Liste 4. Bir projenin ‘bildiri’ uygulamasının içeriği

```
$ ls -l ab08/bildiri
__init__.py
models.py
views.py
```

- `models.py` dosyası veritabanı tablolarının tanımlandığı Python sınıfları içerir. Bu sınıfları kullanarak tekrarlayan SQL ifadeleri yerine Python kodlarıyla veritabanınızdaki kayıtlar üzerinde işlemler yapabilirsiniz.
- `views.py` dosyasıyla veritabanınızdaki kayıtlar üzerinde işlem yaptığınız ve sonucunda web arayüzünde göstereceğiniz fonksiyonlar tanımlayabilirsiniz.

Django'nun yeni bir uygulamadan haberdar olması için, `settings.py` dosyasında `INSTALLED_APPS` alanına bir giriş eklemeniz gerekir. Bu “bildiri” uygulaması için, `ab08.bildiri` dizisi eklenmelidir:

Liste 5. `settings.py`'ye bir giriş eklenmesi

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'ab08.bildiri',
)
```

3. Bir Model Yaratılması

Django, bir Python nesne arabirimi yoluyla dinamik veritabanı arabirimi erişimini destekleyen kendi nesne ile ilişkisel eşleyici (ORM; object-relational mapper) kitaplığıyla birlikte sağlar.

ORM şu anda PostgreSQL, MySQL, SQLite ve Microsoft® SQL veritabanları için destek sağlar. Bu örnekte veritabanı arka ucu olarak SQLite kullanılmıştır. SQLite, hiçbir yapılandırma gerektirmeyen ve diskte basit bir dosya olarak yer alan basit bir veritabanıdır. SQLite'ı kullanmak için, ayar araçlarını kullanarak yalnızca pysqlite kitaplığını kurun.

Liste 6. Veritabanının settings.py'de yapılandırılması

```
DATABASE_ENGINE = 'sqlite3'
DATABASE_NAME = '/path/to/ab08/database.db'
DATABASE_USER = ''
DATABASE_PASSWORD = ''
DATABASE_HOST = ''
DATABASE_PORT = ''
```

Bu “bildiri” uygulamasının “conflocation”(Konferans şehri) ve “paper”(makale) olmak üzere 2 tip nesnesi olacaktır. Bir conflocation nesnesi yıl ve şehri isteğe bağlı alanları içerir, paper nesnesi de bildiri sahibi bildiri başlığı ve özeti, bildiri sahibinin e-posta adresi ve bildiri yükleme alanlarını içerir.

Liste 7. bildiri/models.py modülü

`__str__` yöntemi, bir nesnenin dize gösterimini döndüren Python ürününün içerdiği özel bir sınıf yöntemidir. Django Admin aracında nesnelere görüntülerken bu yöntemi yaygın olarak kullanır.

Modele ilişkin veritabanı şemasını görmek için `manage.py`'nin `sql` komutunu çalıştırın ama henüz şema canlandırılmayacaktır (enact).

```
from django.db import models

class confLocation(models.Model):
    year = models.DateField()
    state = models.CharField(maxlength=50)
class Admin:
    pass
def __str__(self):
    return "%s, %s" % (self.year, self.state)
class paper(models.Model):
    id = models.AutoField('ID', primary_key=True)
    author = models.CharField(maxlength=50)
    title = models.CharField(maxlength=50)
    summary = models.CharField(maxlength=50)
    email = models.EmailField()
    upload = models.FileField(upload_to='files')
    location = models.ForeignKey(confLocation)
class Admin:
    pass
def __str__(self):
    return "(%s) %s" % (self.id, self.title)
```

Liste 8. manage.py sql komutunu kullanarak veritabanı şemasının görüntülenmesi

Modeli başlatmak ve kurmak için, `syncdb` veritabanını eşitle komutunu çalıştırın:

Liste 9. syncdb komutunun kullanılarak modelin kurulması ve başlatılması

```
$ python ab08/manage.py syncdb
```

`syncdb` komutunun sizden bir süper kullanıcı hesabı yaratmanızı ister, bunun nedeni, temel kullanıcı doğrulaması işlevini sağlayan `django.contrib.auth` uygulamasının, `INSTALLED_APPS` ayarlarınızda varsayılan değer olarak sağlanmasıdır.

```
$ python ab08/manage.py sql bildiri
BEGIN;
CREATE TABLE "bildiri_paper" (
"id" integer$ python ab08/manage.py sql
bildiri
BEGIN;
CREATE TABLE "bildiri_paper" (
"id" integer NOT NULL PRIMARY KEY,
"author" varchar(50) NOT NULL,
"title" varchar(50) NOT NULL,
"summary" varchar(50) NOT NULL,
"email" varchar(75) NOT NULL,
"upload" varchar(100) NOT NULL,
"location_id" integer NOT NULL
);
CREATE TABLE "bildiri_conflocation" (
"id" integer NOT NULL PRIMARY KEY,
"year" date NOT NULL,
"state" varchar(50) NOT NULL
);
COMMIT; NOT NULL PRIMARY KEY,
"author" varchar(50) NOT NULL,
"title" varchar(50) NOT NULL,
"summary" varchar(50) NOT NULL,
"email" varchar(75) NOT NULL,
"upload" varchar(100) NOT NULL,
"location_id" integer NOT NULL
);
CREATE TABLE "bildiri_conflocation" (
"id" integer NOT NULL PRIMARY KEY,
"year" date NOT NULL,
"state" varchar(50) NOT NULL
);
COMMIT;
```

Not: Sisteminizin süper kullanıcısı değil, Django süper kullanıcısı olduğu unutmayın.

4.Admin (Yönetici) Aracı

Django'nun tercih edilmesinde en büyük rolü taşıyan özelliği, sahip olduğu hoş görünümlü yönetici arabirimidir. Bu araç, son kullanıcılar düşünülerek tasarlanmıştır. Bu araç projelerinize büyük bir veri girişi aracı sağlar.

Admin aracı, Django ile birlikte sağlanan bir uygulamadır. Kullanabilmeniz için bunun da jobs uygulaması gibi, önce kurulması gerekir. Bunun için ilk adım, uygulamanın modülünü (`django.contrib.admin`) INS-

TALLED_APPS ayarına eklemektir:

Liste 10. settings.py'nin değiştirilmesi

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'ab08.bildiri',
    'django.contrib.admin',
)
```

Admin aracının /admin URL'sinden kullanılmasını sağlamak için, yalnızca projenizin urls.py dosyasında sağlanan satırı devre dışı bırakın. Bir sonraki bölümde URL yapılandırması daha ayrıntılı biçimde açıklanacaktır.

Liste 11. Admin aracının urls.py yoluyla sağlanması

```
from django.conf.urls.defaults import *

urlpatterns = patterns('',
    (r'^admin/',
    include('django.contrib.admin.urls.admin')),
)
```

5.URL Düzeninizin Tasarlanması

Django URL dağıtım sistemi, URL dizesi desenlerini views (görünüm)ler olarak adlandırılan Python yöntemleriyle eşleyen normal anlatım yapılandırma modüllerini kullanır. Bu sistem, URL'lerin temel koddan tamamen ayrıştırılmalarına olanak tanıyarak en üst düzeyde denetim ve esneklik sağlar.

Bir urls.py modülü, URL yapılandırmaları için varsayılan başlangıç noktası olarak yaratılır ve

tanımlanır (settings.py modülündeki ROOT_URLCONF değeri yoluyla). Bir URL yapılan-dırma dosyası için tek gereksinim, urlpatterns olarak adlandırılan desenleri tanımlayan bir nesne içermesi zorunluluğudur.

Bildiri uygulaması, aşağıdaki URL eşlemeleri yoluyla erişilebilecek bir dizin ve ayrıntılı görünümle başlayacaktır:

- /βιλδιρι dizin görünümü: Bildirileri görüntüler.

Liste 12. ab08/urls.py: URL'lerin projeye geri bağlanması

```
from django.conf.urls.defaults import *
urlpatterns = patterns('',
    (r'^admin/',
     include('django.contrib.admin.urls.admin')),
    (r'^bildiri/',
     include('ab08.bildiri.urls')),)
```

Bu noktada test sunucunuzu kullanarak dizin sayfasına (http://localhost:8000/jobs) erişmeye çalışırsanız, çağrılan görünüm (djproject.jobs.views.index) henüz var olmadığı için bir hata iletisi alırsınız.

6. Görünümlerin Uygulanması

Görünüm, bir istek nesnesini kabul eden bir Python yöntemidir ve aşağıdakilerden sorumludur:

- Herhangi bir iş mantığı (doğrudan ya da dolaylı olarak)
- Şablona ilişkin verileri kapsayan bir içerik sözlüğü
- Şablonun bir içerikle oluşturulması
- Oluşturulan sonuçları çerçeveye geri taşıyan yanıt nesnesi

Django'da, bir URL istendiğinde çağrılan Python yöntemi bir view (görünüm) olarak adlan-

dırılır ve view tarafından yüklenen ve oluşturulan sayfaya template (şablon) adı verilir. Bu nedenle, Django ekibi Django'dan bir MVT (model-görünüm-şablon) çerçevesi olarak bahseder. Diğer yandan, TurboGears, MVC kısaltmasına tam olarak uyması için, yöntemlerini controllers (denetleyiciler) ve bunların oluşturduğu şablonları views (görünümler) olarak adlandırır. Bunlar aynı işlevleri içerdiğinden, fark büyük ölçüde anlama dayalıdır.

Liste 13. bildiri/views.py

```
from django.http import
HttpResponse

from django.template import
Context, loader

from bildiri.models import
paper

def index(request):

    list = paper.objects.order_
by('title')

    tpl = loader.get_
template('bildiri/list.html')

    context = Context({'list' :
list })

    return HttpResponse(tmpl.
render(context))
```

7. Şablonların Yaratılması

Django, hızlı oluşturma ve kullanım kolaylığı için tasarlanan basit bir şablon dili sağlar. Django şablonları, {{ variables }} (değişkenler) ve {% tags %} (etiketler) ile yerleşik olan düz metin kullanılarak yaratılır. Değişkenler değerlendirilir ve bunların yerini temsil ettiği değerler alır. Etiketler temel denetim mantığı için kullanılır. Şablonlar, HTML, XML, CSV ve düz metin de dahil olmak üzere

herhangi bir metne dayalı biçimi oluşturmak için kullanılabilir.

Gerçekleştirilecek ilk adım, şablonların bulunduğu yeri tanımlamaktır. Karmaşıklıktan kaçınmak için, `djproject` altında bir “`templates`” (şablonlar) dizini oluşturun ve bunun yolunu `TEMPLATE_DIRS` settings.py girişine ekleyin:

Liste 14. settings.py’de bir templates dizininin oluşturulması

```
TEMPLATE_DIRS = ('/path/to/ab08/templates/',)
```

Django şablonları, site tasarımcılarının, içeriği her bir şablonda yinelemeksizin bir örnek bir görünüm yaratmalarını sağlayan `template inheritance` kavramını destekler. Bu kavramı, blok etiketli bir iskelet ya da temel belge tanımlayarak kullanabilirsiniz. Bu blok etiketleri, içerikli sayfa şablonları tarafından doldurulur.

Liste 15. İskelet belge, templates/base.html

```
<!DOCTYPE html PUBLIC
“-//W3C//DTD XHTML 1.0
Transitional//EN”
“http://www.w3.org/
TR/xhtml1/DTD/xhtml1-
transitional.dtd”>
<html xmlns=“http://
www.w3.org/1999/xhtml”
xml:lang=“en” lang=“en”>
  <head>
    <title>Akademik Bilişim
Konferansı: {% block title %}
Page{% endblock %}</title>
    {% block extrahead %}{%
endblock %}
  </head>
  <body>
    {% block content %}{%
endblock %}
  </body>
</html>
```

Şimdi, görünüm tarafından yüklenecek ve oluşturulacak sayfa şablonunu yaratın. `Bildiriler/list.html` şablonu `list` sayesinde başta yinelenir, `index` (dizin) görünümü sayesinde içeriğe ulaşır ve her bir kaydın ayrıntı sayfasına bir bağlantı içerir.

Liste 16. templates/bildiriler/list.html şablonu

```
{% extends “base.html” %}

{% block extrahead %}
  <style>
    body {
      font-style: arial;}
    h1 {
      tex{% extends “base.
html” %}

{% block extrahead %}
  <style>
    body {
      font-style: arial;}
    h1 {
      text-align: center;}
    .bildiri .title {
      font-size: 120%;
      font-weight: bold;}
    .bildiri .posted {
      font-style: italic;}
  </style>
{% endblock %}t-align:
center;
  }
  .bildiri .title {
    font-size: 120%;
    font-weight: bold;}
  .bildiri .posted {
    font-style: italic;}
  </style>
{% endblock %}
```

Django řablon dili, sınırlı iřlevsel yetenekler iřerecek řekilde tasarlanmıřtır. Bu sınırlandırma, řablonların programcı olmayan kiřiler için kolay anlaşılır olmasını ve programcılar için de iř mantığını ait olmadığı bir yer olan sunu düzeyine yerleřtirmemelerini sađlar.

8. Kaynakça

- Django Book
- Django Project Documentation
- Django Wiki