HACETTEPE ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ

WEB MİMARİSİNİN TEMELLERİ DERSİ

DÖNEM ÖDEVİ

İsim	**: Aybars YÜKSEL, Oğuz ÖZSANCAKTAR**
Öğrenci No.	**: 21228951, 21228621**
Konu	**: Django Tutorial**
Ders Danışmanları	**: Gönenç ERCAN**
	Aydın KAYA
	Ali Seydi KEÇELİ

İçindekiler

Resimler Tablosu

1. GİRİŞ

Projemiz için Django web çatısını seçtik. Django açık kaynak ve Python dili üzerine kurulmuş, kolay ve hızlı web uygulamaları geliştirmeyi sağlayan güçlü bir web uygulama çatısıdır. Birçok özellik halihazırda gelir. Github istatistiklerine göre Django bütün web çatıları arasında en popüleridir.

1.1. Neden Django

Django Python üzerine geliştirilmiş, dolayısıyla Python'un hızlı yazılması ve kolay anlaşılır olması gibi "Zen of Python" başlığı altında listelenen birçok avantajını miras alır. Bunlardan bazılarını örnek verecek olursak:

- "Hatalar sessizce geçiştirilmemeli" gizlenen hataları bulmayı kolaylaştırır.
- "Uygulamayı açıklaması zorsa, kötü bir fikirdir", "okunabilirlik önemlidir" ve "basit olan kompleks olandan iyidir" Django ve paketlerinin harika dokümantasyonlara ve yazılmış kodların yüksek okunabilirliğe sahip olmasında önemli payı vardır.

Djangonun ana sloganı "The Web framework for perfectionists with deadlines", çok hızlı ve yüksek kalitede kod geliştirmeye olanak sağladığının altını çizmektedir. Bu argümanı destekleyen bir çok avantajından en önemlilerini sıralarsak:

- Geniş bir topluluğunun olması ve çok fazla paketin halihazırda bulunması. Buna örnek olarak "django-allauth" paketiyle hızlı şekilde sosyal ağlarla uygulamayı bağlama çözümü üretilebilmesini gösterebiliriz.
- "Dont repeat yourself" felsefesi, gereksiz veri ve kod fazlalığından uzak olması.
- Pythonun "introspection" gibi birçok yeteneğine sahip olması.
- "Loose coupling" felsefesi: farklı katmanlar arasındaki bağlılığın minimize edilmesi.
- Yeni bir takım üyesinin projeye hızlı ve kolay adapte olabilmesi.
- Özelleştirilebilir admin paneli, doğrulama ve güvenlik katmanları, ORM(kolay veritabanı işlemleri), yüksek esneklik kabiliyeti gibi özelliklerin hazır olarak gelmesi.
- Son derece esnetilebilir olması
- Yüksek trafikle baş edebilmek için mongoDB, redis gibi teknolojilerle kolayca entegre edilebilmesi.

Bu sebepler Django ile uygulama geliştirmeyi son derece düşük maliyetli yapar. Son dönemde popüler bir yaklaşım olan MVP(minimum viable product), uygulamanın başlangıç için minimum fonksiyonlarla hayata geçirilmesi yaklaşımıyla uyumlu olması, Django'yu yeni girişimler arasında çok popüler bir tercih haline getirmiştir.

Biz de bu tutorialda MVP yaklaşımı üzerinden Django'nun nasıl kullanılacağını anlatmaya çalışacağız.

1.2. Django'nun Yapısı

Django, MVC(Model-View-Controller) mimari desenine çok benzeyen MTV (Model-Template-View) desenini kullanır.

Model : Uygulamanın veriye erişim katmanıdır. Veriye nasıl erişileceği, validasyon ve veriler arasındaki ilişkiler gibi veriyle alakalı her şey bu kısımda yer alır. Formlardaki alanlardan veri tabanındaki tablolara kadar her şey modellerden üretilir.

Template : Uygulamanın sunum katmanıdır. Bu katma, web sayfasında veya diğer tüm dökümanlarda verilerin nasıl gösterilmesi gerektiği gibi sunum kısmıyla ilgili kararları içerir.

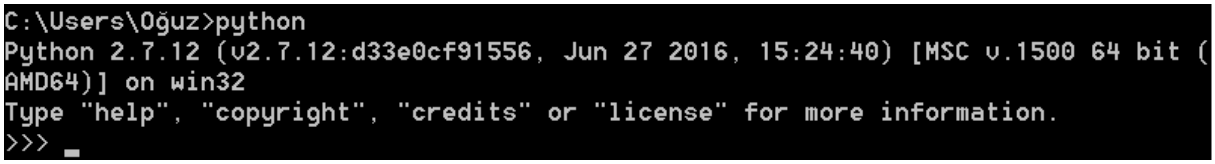
View : Uygulamanın iş mantığını içeren kısımdır. Bu katman, uygun template ve modellere erişimle ilgili mantıksal kısmı içerir. MVC'deki controller'a denk gelmektedir. Model kısmı uygulamanın iskeleti ise view kısmı da uygulamanın beynidir.

2. TUTORIAL

2.1. Kurulumlar

2.1.1. Python Kurulumu

Django, Python ile oluşturulmuş bir framework olduğu için Django kullanmak için öncelikle Python kurmak gerekmektedir. Bilgisayarınızda Python'ın kurulu olup olmadığını komut satırına **python** yazarak test edebilirsiniz.



Resim 1. Python Kurulum Kontrolü

Python kurulu olduğu takdirde size resimdeki gibi mevcut Python sürümü dönecektir.

Python kurulu değil ise öncelikle kendi sitesinden uygun sürümü indirip kurmanız gerekir. **https://www.python.org/downloads/**

Kurulum sırasında "Add Python X.X to PATH" seçeneğini seçerseniz Python otomatik olarak path'e eklenecektir. (X.X sürüm için kullanılmıştır)

Kurulumdan sonra tekrar komut satırına **python** yazarak kurulumu kontrol edebilirsiniz.

Python'u kurduktan sonra Python için hazırlanmış bir paket yönetim sistemi olan PIP'i kurmanız gerekmektedir. Pip'in açılımı "Pip Installs Packages" olup bir recursive acronym'dir.

Komut satırına **python –m pip install –U pip** yazarak pip'in kurulumunu gerçekleştirebilirsiniz.

2.1.2. Python Sanal Ortamı ve Django Kurulumu

Django'yu yüklemeden önce kod ortamını düzenli tutmak için son derece yararlı bir araç olan virtualenv yükleyeceğiz. Virtualenv Python/Django kurulumunuzu her proje için ayrı tutup izole eder. Bu, bir uygulamada yapacağınız değişikliklerin diğer geliştirdiklerinize yansımayacağı anlamına gelir.

Öncelikle komut satırına **pip install virtualenv** yazarak virtualenv kurulumunu gerçekleştirin.

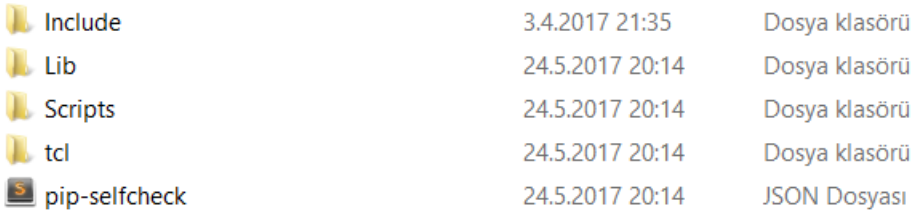
Sonra projenizi başlatacağınız bir klasör oluşturun. Klasörün adresinde boşluk ya da özel karakter olmamasına dikkat edin.

Oluşturduğunuz proje klasörünün içinde komut satırını açın ve **virtualenv deneme** yazın. Buradaki "deneme" sanal ortamınızın ismi olup önemli değildir. Bu komutu yazdıktan sonra aşağıdakine benzer bir komut satırı ile karşılaşacaksınız.

```
E:\Django\deneme>virtualenv deneme
New python executable in E:\Django\deneme\deneme\Scripts\python.exe
Installing setuptools, pip, wheel...done.
```

Resim 2. Sanal Ortam Oluşturma

Sanal ortam oluşturulduğunda aşağıdaki gibi bir klasör (bu örnekte "deneme" klasörü) otomatik olarak oluşturulmuş olacaktır.



Resim 3. Sanal Ortam Klasörü İçeriği

Burada "\Lib" klasörüne bakarsanız virtualenv'in yeni bir Python kurulumu yapmış olduğunu göreceksiniz. Bu sayede yeni projenizde diğer projeleri etkilemeden çalışabileceksiniz.

Bu yeni Python sanal ortamını çalıştırmak için komut satırına **deneme\Scripts\activate** yazın.

```
E:\Django>deneme\Scripts\activate

(deneme) E:\Django>
```

Resim 4. Python Sanal Ortamına Geçiş

Daha sonra bu yeni sanal ortamda **pip install django** yazarak Django kurulumunu gerçekleştirin. Dilerseniz ifadenini sonuna **==1.11.1** ve benzeri şekilde kurmak istediğiniz Django sürümünü de ekleyebilirsiniz. Eklemezseniz en güncel sürüm kurulacaktır.

```
E:\Django>deneme\Scripts\activate

(deneme) E:\Django>pip install django
Collecting django
  Using cached Django-1.11.1-py2.py3-none-any.whl
Collecting pytz (from django)
  Using cached pytz-2017.2-py2.py3-none-any.whl
Installing collected packages: pytz, django
Successfully installed django-1.11.1 pytz-2017.2

(deneme) E:\Django>
```

Resim 5. Django Kurulumu

Bu adımla birlikte kurulum için gerekli olan adımları tamamlamış olduk.

2.2. Proje Başlatma

Kurulumlar tamamlandıktan sonra yeni bir Django projesi başlatmak için komut satırına **django-admin startproject mysite** yazın. Buradaki "mysite" proje adını belirtmektedir. Bunu yazdığınızda proje dosyanız otomatik olarak oluşturulacaktır.

Daha sonra veri tabanını ayarlamak için komut satırından yeni oluşturulan proje dosyasına girin ve **python manage.py migrate** yazın. Bu komut yeni bir SQLite veri tabanı oluşturacaktır. Django default olarak SQLite kullanmaktadır.

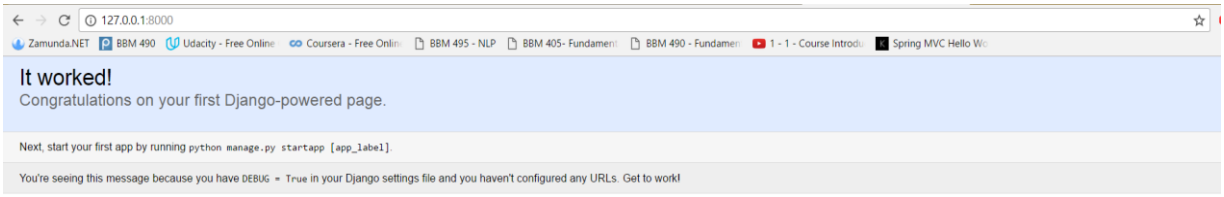
```
(deneme) E:\Django>cd mysite

(deneme) E:\Django\mysite>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying sessions.0001_initial... OK

(deneme) E:\Django\mysite>
```

Resim 6. Veri Tabanı Ayarı

Daha sonra komut satırına **python manage.py runserver** yazdığınızda aşağıdaki gibi bir ekran görüntüsüyle karşılaşacaksınız ve "Django development server" çalışmaya başlayacaktır. Serveri **http://127.0.0.1:8000/** adresinden ziyaret edebilirsiniz.

```
(deneme) E:\Django\mysite>python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
May 24, 2017 - 20:37:17
Django version 1.11.1, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Resim 7. Django Uygulama Çalıştırma



Resim 8. Django "Hoş Geldin" Sayfası

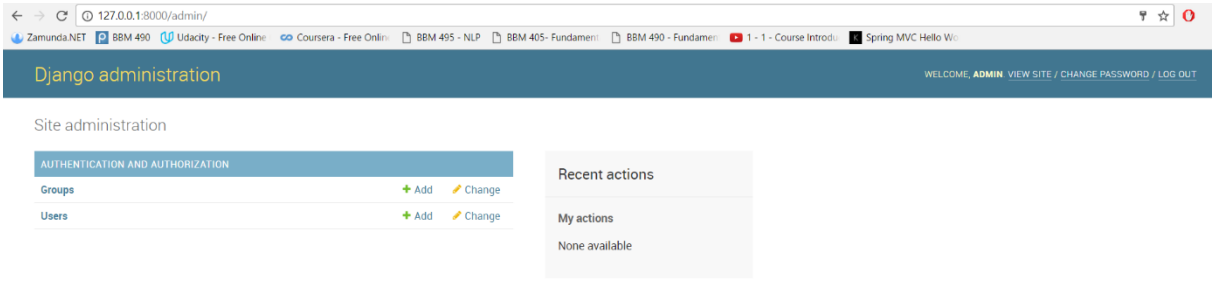
2.3. Django Admin Paneli

Önceki kısımda **django-admin startproject mysite** yazdığınızda Django otomatik olarak default bir admin paneli oluşturdu. Bu panele bağlanmak için superuser oluşturmamız lazım. Bunun için komut satırına **python manage.py createsuperuser** yazıyoruz. Sizden oluşturulacak admin için kullanıcı adı, email adresi ve şifre isteyecek. Bunları girdiğimiz zaman adminimiz oluşmuş olacak.

```
(deneme) E:\Django\mysite>python manage.py createsuperuser
Username (leave blank to use 'sentor'): admin
Email address: admin@example.com
Password:
Password (again):
Superuser created successfully.
```

Resim 9. Django Admin Oluşturma

Şimdi tekrar **python manage.py runserver** diyerek serveri çalıştırabilir ve **http://127.0.0.1:8000/admin/** adresinden admin paneline bağlanabiliriz. Çıkan ekranda kullanıcı adı ve şifremizle giriş yapmalıyız.



Resim 10. Django Admin Paneli

2.4. Model Oluşturma

Django'da model oluşturmadan önce projenin içinde ayrı bir uygulama oluşturuyoruz. Bu şekilde her şey en başından derli toplu oluyor. Bunun için komut satırına **python manage.py startapp hizmet** yazın. Burada "hizmet" uygulama ismi olmaktadır. Farklı bir şey de yazabilirsiniz.

Uygulama oluştuktan sonra Django'ya bunu kullanmasını söylememiz gerekiyor. Bunu **mysite/settings.py** dosyası ile yapıyoruz. **INSTALLED_APPS** kısmını bulup oraya "hizmet"i ekliyoruz.

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'hizmet',
]
```

Resim 11.Hizmet'i setting.py Dosyasına Ekleme

Daha sonra model oluşturmak için **hizmet/models.py** dosyasına istediğimiz modelleri tanımlayabiliriz.

```
1  # -*- coding: utf-8 -*-
2  from __future__ import unicode_literals
3
4  from django.db import models
5
6  # Create your models here.
7
8  class Urun(models.Model):
9      isim = models.TextField()
10     fiyat = models.FloatField()
11     aciklama = models.TextField()
12
13     def __str__(self):
14         return self.isim
15
```

Resim 12. Urun Modeli Oluşturma

Yukarıdaki resimde **hizmet/models.py** dosyasında "Urun" isminde bir model oluşturuldu. Ürünleriminiz ismi, fiyatı ve açıklaması bulunmaktadır. __str__() fonksiyonu da bize ürünün ismini döndüren bir fonksiyondur.

Şimdi oluşturduğumuz modeli veri tabanına eklememiz gerekmektedir. Bunun için önce Django'ya yeni bir model oluştuğunu haber vermemiz gerekiyor. Bunun için komut satırına **python manage.py makemigrations hizmet** yazın.

```
(deneme) E:\Django\mysite>python manage.py makemigrations hizmet
Migrations for 'hizmet':
  hizmet\migrations\0001_initial.py
    - Create model Urun
```

Resim 13. Model Migrasyon Dosyası Oluşturma

Django bize veri tabanına uygulayabileceğimiz bir taşıma(migrasyon) dosyası oluşturmuş oldu. Komut satırına **python manage.py migrate hizmet** yazarak modelimizi veri tabanına ekliyoruz.

```
(deneme) E:\Django\mysite>python manage.py migrate hizmet
Operations to perform:
  Apply all migrations: hizmet
Running migrations:
  Applying hizmet.0001_initial... OK
```

Resim 14. Modeli Veri Tabanına Ekleme

Şimdi "Urun" modelimiz veri tabanına eklenmiş oldu.

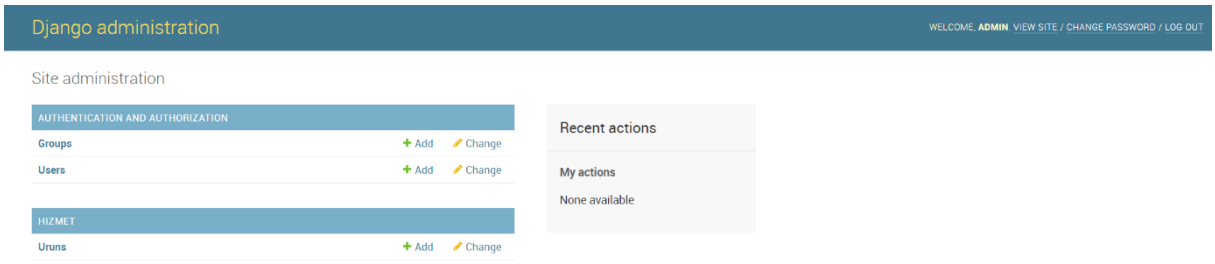
Admin panelinden Urunlere ulaşmak için **hizmet/admin.py** dosyasında değişiklik yapmamız gerekmektedir.

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.contrib import admin
from .models import Urun

admin.site.register(Urun)

# Register your models here.
```

Resim 15. Admin Paneline Model Ekleme

Yukarıdaki resimde görüldüğü gibi tanımladığımın Urun modelini **admin.py** dosyasına import ettik. Modeli admin panelinde görünür yapmak için de **admin.site.register(Urun)** kısmını ekledik.

Şimdi serveri çalıştırıp admin paneline girdiğinizde Uruns adı altında modelimizin oluştuğunu göreceksiniz.



Resim 16. Yeni Modelimiz ile Admin Paneli

Artık admin panelini kullanarak yeni ürün ekleyebilir; mevcut ürünleri değiştirebilir ya da silebilirsiniz.

2.5. View Oluşturma

Websitemizde 4 tane basit sayfamız olacak. Bu sayfalara erişimin sağlanması için önce 2 dosyada url ayarlarını yapacağız, sonra view dosyamızı düzenleyeceğiz, en son html dosyalarımızı yapılandıracağız.

Sitemizin görüntülenebilmesi için url bağlantılarını gerekli **urls.py** dosyasının url patterns kısmına **url(r'', include('hizmet.urls'))** aşağıdaki gibi ekledik.

```
18
19      urlpatterns = [
20          url(r'^admin/', admin.site.urls),
21          url(r'', include('hizmet.urls'))
22      ]
23
```

Resim 17. urls.py Dosyası

hizmet klasörünün altına aşağıdaki **hizmet/urls.py** dosyasını ekliyoruz:

```
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^$', views.anasayfa, name='anasayfa'),
    url(r'anasayfa', views.anasayfa, name='anasayfa'),
    url(r'hakkimizda', views.hakkimizda, name='hakkimizda'),
    url(r'urunler', views.urunler, name='urunler'),
    url(r'iletisim', views.iletisim, name='iletisim'),
]
```

Resim 18. hizmet/urls.py Dosyası

view dosyamıza görüntülenecek html dosyalarının isimlerini verdik. **hizmet/url.py** dosyasından çağırılan fonksiyonları **hizmet/views.py** dosyasının içinde aşağıdaki gibi tanımlayacağız:

```python
# -*- coding: utf-8 -*-
from __future__ import unicode_literals
from django.shortcuts import render
from .models import Urun

# Create your views here.

def anasayfa(request):
    return render(request, 'anasayfa.html', {})

def hakkimizda(request):
    return render(request, 'hakkimizda.html', {})

def urunler(request):
    urunler = Urun.objects.order_by('isim')
    return render(request, 'urunler.html', {'urunler': urunler})

def iletisim(request):
    return render(request, 'iletisim.html', {})
```

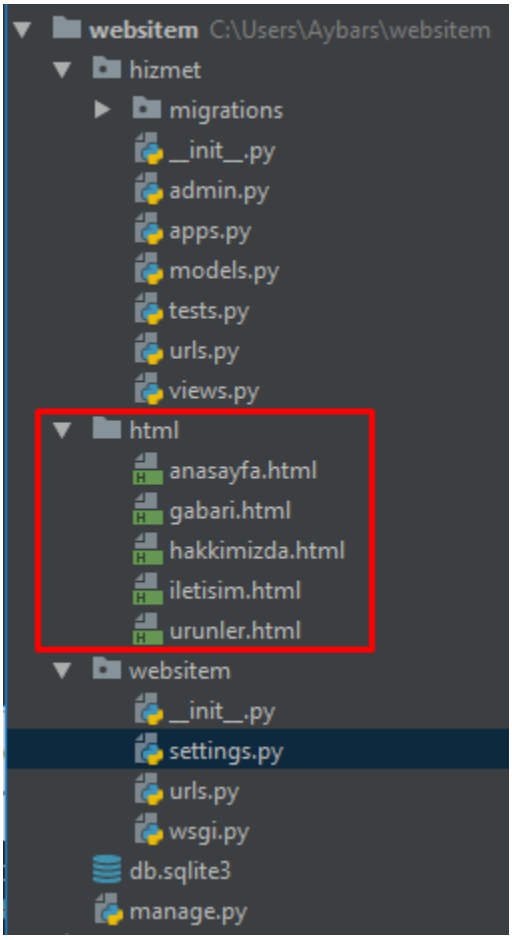
Resim 19. hizmet/views.py Dosyası

Burada dikkat edilmesi gereken iki nokta şudur:

1. Buradaki python fonksiyonları **hizmet/url.py** dosyasından çağrılmaktadır, ve her bir sayfa bu fonksiyonlar aracılığıyla çağrılan html dökümanları ile görüntülenecek.
2. **from .models import Urun** satırı ile yarattığımız modelimizi tanımladıktan sonra, **urunler** fonksiyonunun içinde yine **urunler** ismiyle admin panelinden yarattığmız objeleri çekiyoruz. Ve sayfamıza bu çektiğimiz objeleri yolluyoruz.

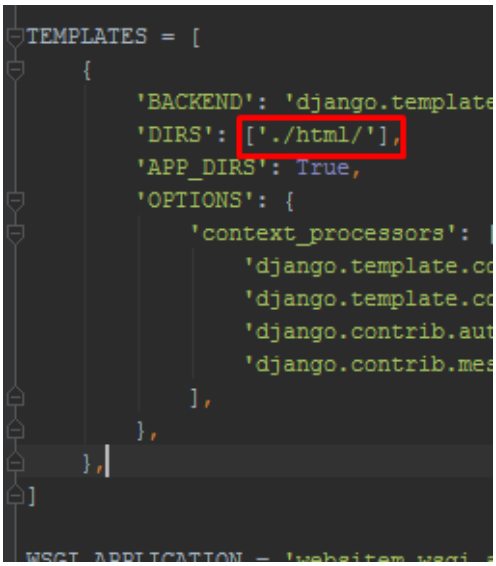
2.6. Template Oluşturma

Html dosyalarımızın yeri şu şekilde:



Resim 20. hml Klasörü

Bu dosyaların yerinin Django tarafından bulunabilmesi için, **settings.py** dosyamızın içindeki **TEMPLATES** dizininin **DIRS** değişkenine bu dosyaların **html** klasörünün altında olduğunu aşağıdaki gibi belirtmeliyiz:



Resim 21. setting.py Templates Ayarı

Html dosyalarının içerikleri hepsini buraya yazmanın zorluğundan dolayı, rehberimizin en sonunda paylaşacağımız github linkinden ulaşılabilecek. Linkteki html klasörünün altından içeriği görüp kopyalayabilirsiniz. Son olarak html dosyalarında kullandığımız template mantığını ve ürünlerin dinamik olarak nasıl çağrılması gerektiğini anlatacağız.

gabari.html dosyası diğer 4 sayfa için ortak kullanılacak iskelet html kodlarını barındırmaktadır. Bu yapıyı sağlamak için sayfadan sayfaya değişmesini istediğimiz bölümü <body></body> etiketlerinin içinde istediğimiz yere aşağıdaki gibi yazmalıyız:

```
{% block content %}
{% endblock %}
```

Resim 22. Block Oluşturma I

Aynı zamanda oluşturacağımız diğer bütün html sayfalarını aşağıdaki kod bloğunun içine yazmalıyız:

```
{% extends 'gabari.html' %}
{% block content %}
    <p>buraya herhangi bir içerik yazabilirsiniz</p>
{% endblock content %}
```

Resim 23. Block Oluşturma II

Yukarıdaki kodun eklendiği her html sayfası, django tarafından derlenerek **gabari.html** iskeletiyle kaplanarak yayınlanacaktır.

Örnek olarak, **urunler.html** sayfamız şu şekilde:

```
{% extends 'gabari.html' %}
{% block content %}
{% for urun in urunler %}
        <div class="jumbotron">
            <h1>{{ urun.isim }}</h1>
            <p>{{ urun.fiyat|linebreaks }}</p>
            <p>{{ urun.aciklama}}</p>
        </div>
{% endfor %}
{% endblock content %}
```

Resim 24. View'de for Döngüsü

Burada **{% for urun in urunler %}** ve **{% endfor %}** yapılarını kullandık. Hatırlarsak **urunler.html** sayfasını çağıran **hizmet/views.py** dosyası altından çağıran **urunler** fonksiyonu, **urunler** değişkenini göndererek bu sayfayı çağırıyordu. Bu değişkeni html içinde bu şekilde kullandık, bu sayede bütün **Urun** objeleri bir döngü içinde sayfaya yazılacak. **Urun** objesinin taşıdığı değerlere erişmek için **urun.isim, urun.fiyat** ve **urun.aciklama** değişkenlerini kullandık.

3. SONUÇ

MVP yaklaşımı önce basit bir site oluşturup üzerinden geri dönüş almak üzerine temellendirilmiştir. Django da, tutorialda gördüğünüz gibi, MVP yaklaşımı için çok uygun bir frameworktür. Çok kısa sürede bir internet sitesi oluşturabilirsiniz.

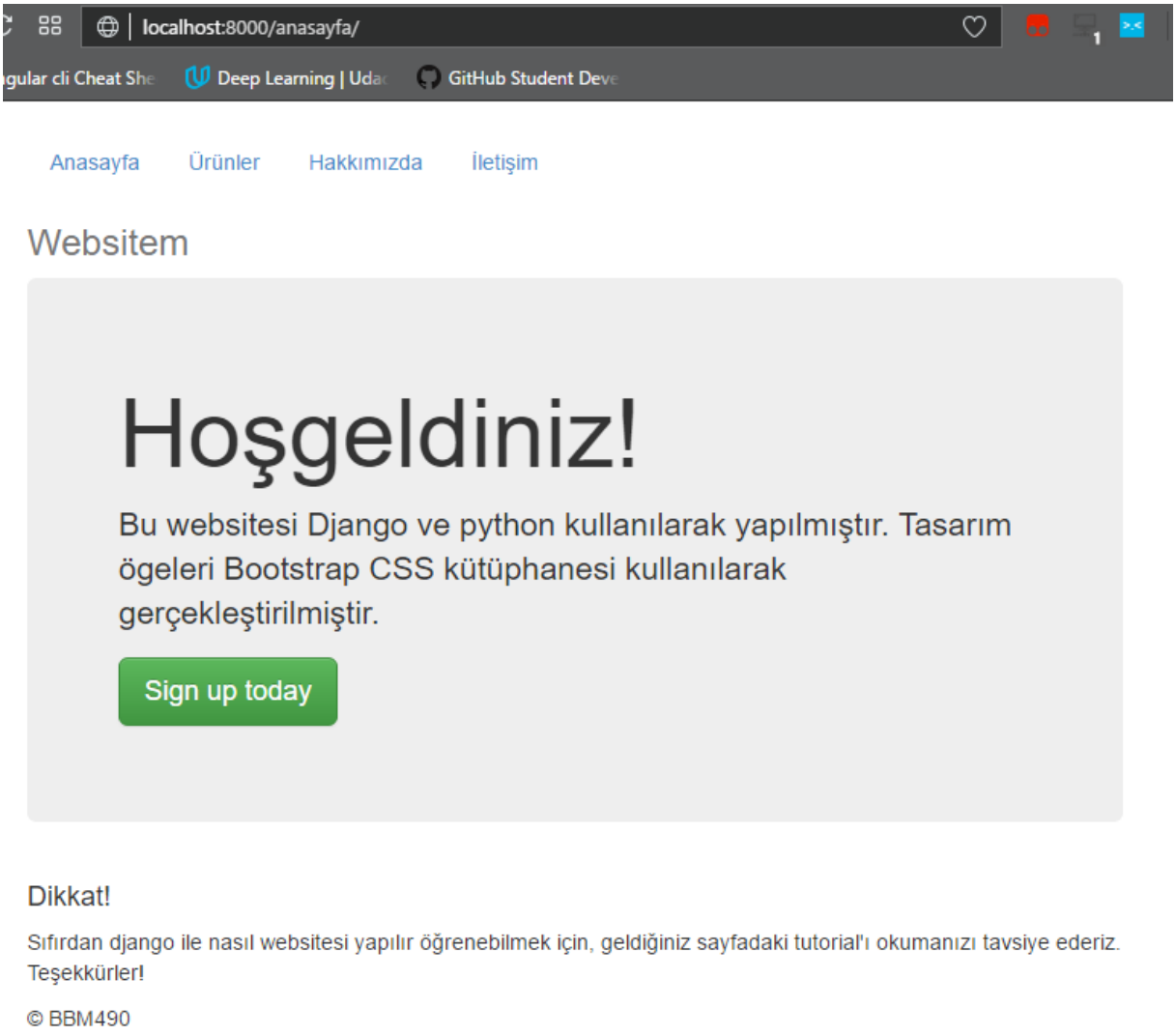
Bu tutorialdan yola çıkarak kendi sitenizin temellerini atabilirsiniz. Ayrıca Django'nun geniş topluluğu sayesinde bir çok hazır paketi sitenize direk dahil edebilirsiniz. Aşağıdaki siteden çok sayıda Django paketine ulaşabilirsiniz:

https://djangopackages.org/

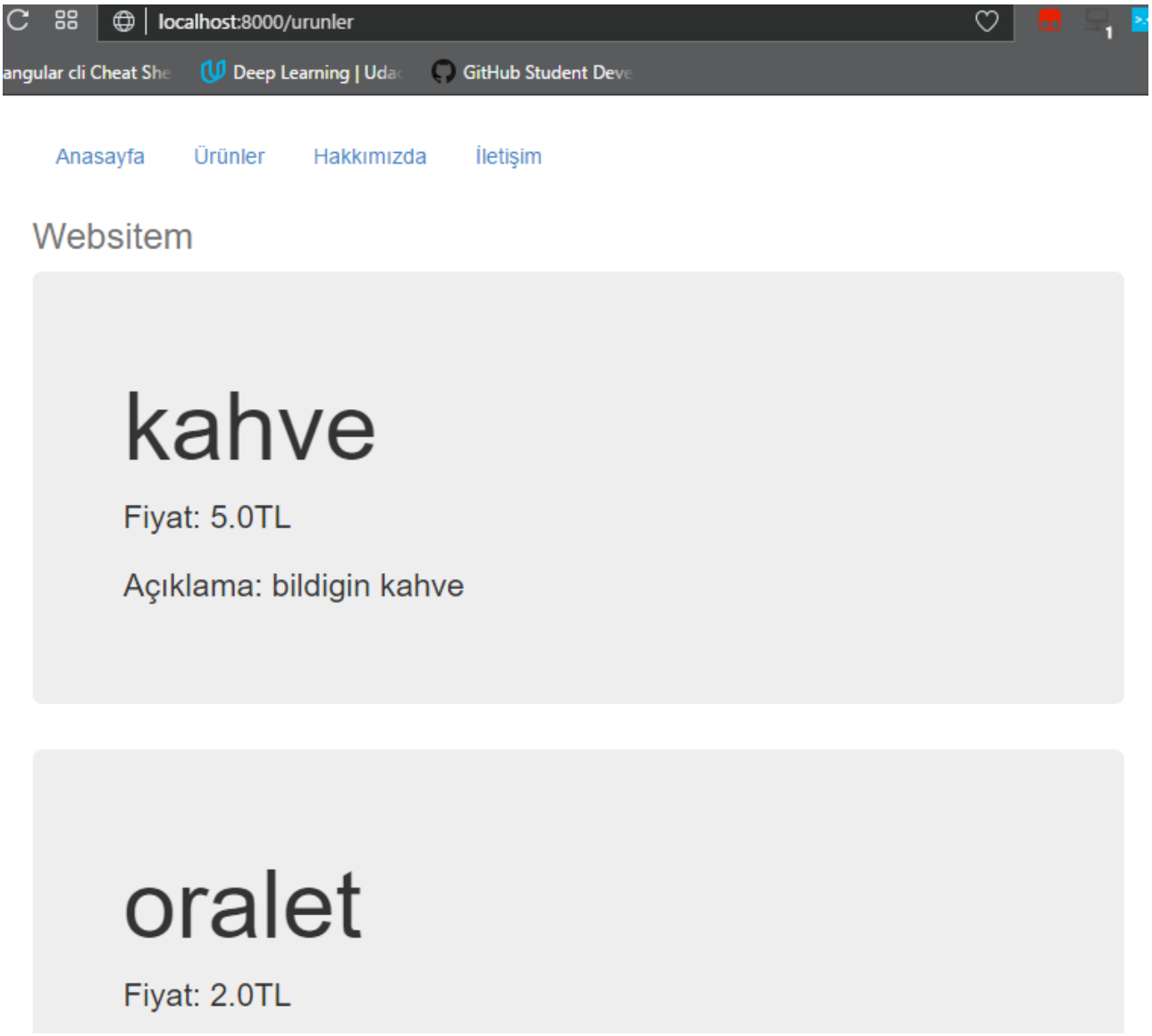
Projede geçen kodlara aşağıdaki adresten ulaşabilirsiniz:

https://github.com/rhycrea/django-rehberi

Hazırladığımız sitenin ekran görüntüleri:



Resim 25. Site Anasayfa



Resim 26. Site Ürünler Sayfası