**Django nedir?**

Django, (framework). Bir web iskeleti, web sitesi geliştirmeyi hızlandıran ve kolaylaştıran bir grup bileşendir. So it is The library of libraries

Bir web sitesi geliştirdiğinizde benzer türde bileşenlere her zaman ihtiyacınız olur: kullanıcılar için kimlik denetimi (üye olma, üye girişi, üye çıkışı), web siteniz için bir yönetim paneli, formlar, dosyaları yüklemek için bir yol, vs.

Django'nun **url çözücüsü** (urlresolver) tarafından yapılıyor (web sitesi adresine URL -Uniform Resource Locator- deniyor, dolayısıyla url çözücü ismi mantıklı oluyor). Çok akıllı değildir -kalıpların bir listesini alır ve URL'yi eşleştirmeye çalışır. Django kalıpları üstten alta doğru denetler ve eşleşen bir şey varsa isteği ilgili işleve (view) aktarır. Tüm ilginç şeyler view fonksiyonunda yapılır:

Öyleyse bir **virtual environment** (diğer adıyla virtualenv) kuralım. Virtualenv Python/Django kurulumunuzu her proje için ayrı tutup izole eder. Bu, bir web sitesine yapacağınız değişikliklerin diğer geliştirdiklerinize yansımayacağı anlamına gelir.

manage.py site yönetimine yardımcı olan bir komut dosyasıdır. Bu dosya sayesinde, başka herhangi bir şey kurmadan bilgisayarımızda bir web sunucusunu başlatabileceğiz.

settings.py dosyası, web sitesinizin ayarlarını içerir.

Bir mektubu nereye götüreceğini kontrol eden postacının hakkında konuştuğumuzu hatırlıyor musun? urls.py dosyası urlresolver tarafından kullanılan kalıpların bir listesini içerir.

**Web sunucusunu başlatmak**

**manage.py** dosyasının bulunduğu dizinde olmalıyız.

**python manage.py runserver**  komutunu çalıştırarak web sunucusunu başlatabilirsiniz:

--

**Peki bir nesne nedir?**

Nesne bir özellikler ve hareketler bütünüdür. Eğer bir kediyi modellemek istiyorsak, Kedi nesnesini oluştururuz ve bu nesne şöyle özelliklere sahip olur: renk, yas, ruh\_hali. Ayrıca Kedi bazı hareketlere sahiptir: miyavla ya da beslen (bu durumda kediye biraz KediMamasi vermemiz gerekir ki o da kendine ait özellikleri olan başka bir nesne olarak tanımlanabilir, bu durumda onun özellikleri arasında da tat yer alabilir). Yani aslında ana fikir, gerçek nesneleri kod içinde özellikleri (nesne özellikleri) ve hareketleri (metodlar) ile tanımlamak.

Öyleyse blog gönderilerini nasıl modelleyeceğiz? Bir blog tasarlamak istiyoruz, degil mi? Cevaplamamız gereken soru: Bir blog gönderisi nedir? Özellikleri ne olmalıdır? Tabii ki blog gönderimizin içeriği için yazı ve bir de başlık lazım, değil mi? Kimin yazdığını da bilsek iyi olur, dolayısı ile bir de yazara ihtiyacımız var. Son olarak, gönderinin ne zaman yaratıldığını ve yayınlandığını da bilmek isteyebiliriz.

**Post**

**--------**

**Title, text, author, created\_date, published\_date**

Bir blog gönderisi ile ne tür şeyler yapılabilir? Gönderiyi yayınlayan bir method olması güzel olurdu, değil mi? Bu yüzden publish metoduna ihtiyacımız olacak.

**Django'da modeller özel bir çeşit nesnedir ve V**eri tabanı**na kaydedilir.** Veri tabanında, kullanıcılar, blog gönderileri, vs. ile ilgili bilgileri saklarız. Verilerimizi depolamak için SQLite veri tabanını kullanacağız. Veri tabanındaki bir modeli, sütunları (alan adı) ve satırları (veri) olan bir hesap çizelgesi olarak düşünebilirsiniz**.**

### **Uygulama oluşturma**

Her şeyi derli toplu tutmak için, projemizin içinde ayrı bir uygulama oluşturacağız. Her şeyin en başından düzenli olması çok iyidir. Bir uygulama oluşturmak için aşağıdaki komutu konsolda çalıştırmamız gerekiyor:

(myvenv) C:\Users\Name\djangogirls> python manage.py startapp blog

Uygulamamızı oluşturduktan sonra, Django'ya bunu kullanmasını söylememiz lazım. Bunu mysite/settings.py içinde INSTALLED\_APPS'ı içine 'blog', yazacağız:

**mysite/settings.py**

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'blog',

]

* str'nin her iki tarafında da 2 tane alt çizgi (\_) kullandığımızdan emin olalım. İki alt çizgi Python dilinde sık kullanılır, bazen "dunder" olarak da anılır ("double-underscore"un kısaltması).

models.CharField - belirli bir uzunluktaki metinleri tanımlamak için kullanılır.

models.TextField - bu da uzun metinleri tanımlar.

models.DateTimeField - bu da gün ve saati tanımlamada kullanılır.

models.ForeignKey - başka bir modele referans tanımlar.

### **Modeller için veri tabanında tablo oluşturma**

Son adımımız yeni modelimizin veri tabanına eklenmesini sağlamak. Öncelikle Django'nun modelimizde bazı değişiklikler yaptığımızı bilmesini sağlamalıyız. (Daha yeni oluşturduk!) Komut satırına gidelim ve python manage.py makemigrations blog yazalım.

**Not:** Düzenlediğiniz dosyaları kaydetmeyi unutmayın. Aksi takdirde, bilgisayarınız önceki sürümü çalıştırarak beklenmedik hatalar verebilir.

Django bize veritabanımıza uygulayabileceğimiz bir taşıma (migration) dosyası oluşturdu.

**python manage.py migrate blog**

Django admin

Modellediğimiz gönderilere ekleme, düzenleme ya da silme işlemi yapmak için Django admini kullanacağız. Şimdi blog/admin.py dosyasını açarak içeriğini aşağıdaki şekilde değiştirelim:

**blog/admin.py**

**from** django.contrib **import** admin

**from** .models **import** Post

admin.site.register(Post)

Gördüğünüz gibi, bir önceki bölümde tanımladığımız gönderi modelini **admin.py** dosyamıza dahil **(import)** ettik. Admin sayfasında modelimizi görünür kılabilmek için, modeli admin.site.register(Post) ile kaydetmemiz gerekiyor.

Giriş yapabilmek için, sitedeki her şey üzerinde kontrolü olan superuser - bir kullanıcı hesabı oluşturmanız gerekiyor. Komut satırına aşağıdakileri yazıp devam edin.

(myvenv) C:\Users\Name\djangogirls> python manage.py createsuperuser

# **Yayına alın!**

**GİT**

Hatırlatma: Kullanıcı adı seçerken Türkçe karakter kullanmayın.

$ git init

Initialized empty Git repository in ~/djangogirls/.git/

$ git config --global user.name "Adınız"

$ git config --global user.email you@example.com

Git bu dizindeki tüm dizin ve dosyalardaki değişiklikleri kaydedecek, ama takip **etmemesini** istediğimiz bazı dosyalar var. Bunu dizinin dibinde .gitignore adında bir dosya oluşturarak yapıyoruz. Editörünüzü açın ve aşağıdaki içeriklerle yeni bir dosya yaratın(txt sonra txt’sini silersin):

.gitignore

\*.pyc

\*~

\_\_pycache\_\_

myvenv

db.sqlite3

/static

.DS\_Store

Ve onu .gitignore ismi ile ilgili yere kaydedin.

komut-satırı

$ git status

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitignore

blog/

manage.py

mysite/

requirements.txt

nothing added to commit but untracked files present (use "git add" to track)

Ve son olarak değişikliklerimizi kaydediyoruz. Komut satırına gidin ve aşağıdaki komutları çalıştırın:

komut-satırı

$ git add --all .

$ git commit -m "Django Girls uygulamam, ilk commit"

[...]

13 files changed, 200 insertions(+)

create mode 100644 .gitignore

[...]

create mode 100644 mysite/wsgi.py

## Kodunu GitHub'a yollama (Push)

Create a repo on Github. Şimdi bilgisayarınızdaki Git reposunu Github'daki repo ile ilişkilendirmemiz gerekiyor.

Aşağıdakini komut satırına yazın **(**< your-github-username>**)** kısmını Github hesabını yarattığınız sırada kullandığınız kullanıcı adı ile değiştirin, büyüktür küçüktür işaretlerini eklemeyin):

komut-satırı

$ git remote add origin https://github.com/<github-kullanıcı-adınız>/my-first-blog.git

$ git push -u origin master

(fatal falan verirse zaten var diye: git remote rm origin)

Github kullanıcı adı ve şifrenizi girin, ardından aşağıdakine benzer bir şey görmeniz gerekiyor:

Username for 'https://github.com': zeynep

Password for 'https://zeynep@github.com':

Counting objects: 6, done.

Writing objects: 100% (6/6), 200 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To https://github.com/zeynep/my-first-blog.git

\* [new branch] master -> master

Branch master set up to track remote branch master from origin.

Kodunuz artık Github'da. Hemen gidip bak! İyi bir çevrede olduğunu göreceksiniz - [Django](https://github.com/django/django" \t "_blank), the [Django Girls Tutorial](https://github.com/DjangoGirls/tutorial" \t "_blank), ve daha birçok harika açık kaynak yazılım projesi de kodlarını Github'da tutuyor. :)

### …or create a new repository on the command line

echo "# deneme\_1" >> README.md

git init

git add README.md

git commit -m "first commit"

git remote add origin https://github.com/yunusemre002/deneme\_1.git

git push -u origin master

### …or push an existing repository from the command line

git remote add origin https://github.com/yunusemre002/deneme\_1.git

git push -u origin master

### …or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](https://github.com/yunusemre002/deneme_1/import)

# **Blogumuzun PythonAnywhere üzerinde kurulumu**

Localde değişim yap 🡪 Githuba at 🡪 Canlıya çek. (Pythonanywhere)

1. [www.pythonanywhere.com](http://www.pythonanywhere.com) sitesine git hesap aç. **Not** Burada kullanıcı adınızı seçerken, bloğunuzun URL'sinin YourUserName.pythonanywhere.com, şeklini alacak

## PythonAnywhere API token oluşturma

Bu sadece bir kez yapmanız gereken bir şey. PythonAnywhere'e kaydolduğunuzda, dashboard sayfasına yönlendirileceksiniz. "Account" sayfanızın sağ üstüne yakın link'i bulun ve "API token" isimli sekmeyi seçerek "Create new API token" butonuna basın.

## Sitemizin PythonAnywhere üzerinde yapılandırılması

Ana [PythonAnywhere Dashboard](https://www.pythonanywhere.com/" \t "_blank) a logosuna tıklayarak dönelim ve bir "Bash" console başlatalım – bu bilgisayarınızdaki komut satırının PythonAnywhere versiyonudur.

PythonAnywhere command-line

**$ pip3.6 install --user pythonanywhere**

**… Succesfully …**

**$ pa\_autoconfigure\_django.py** [https://github.com/<github-kullanıcı-adınız>/my-first-blog.git](https://github.com/%3cgithub-kullanıcı-adınız%3e/my-first-blog.git)

* Kodunuz GitHub'dan çekiliyor. Tıpkı kendi bilgisayarındaki gibi PythonAnywhere üzerinde bir virtualenv oluşturuluyor. Yayına almak için gerekli ayarlar (settings) dosyası güncelleniyor. Yine manage.py migrate komutu ile PythonAnywhere üzerinde veritabanı oluşturuluyor. Sabit dosyaların (bunları daha sonra öğreneceğiz) oluşturuluyor. Ve web uygulamanızın API sinin PythonAnywhere tarafından sunulması için ayarlar yapılıyor

1. PythonAnywhere üzerinde yaratılan veritabanı ile kendi bilgisayarımızda yarattığımız veritabanı birbirinden tamamen ayrı -- dolayısı ile bu iki veritabanında saklanmış olan postlar ve kullanıcılar da farklı olabilir.Bu nedenle, aynen kendi bilgisayarımızda yapmış olduğumuz gibi, createsuperuser ile bir admin kullacısını oluşturmamız gerekiyor. PythonAnywhere otomatik olarak virtualenv i başlatmış olduğu için, bu kullanıcıyı hemen oluşturabiliriz:

PythonAnywhere komut satırı

**(<kullanici-adiniz>.pythonanywhere.com) $ python manage.py createsuperuser**

## Şimdi Canlı Yayındasınız!

Web siteniz şimdi İnternet üzerinden erişilebilir olmalı! PythonAnywhere "Web" tabına tıklayarak linki kopyalayın. Bu linki istediğiniz herkes ile paylaşabilirsiniz :)

**Not** Basit tutmak amacıyla siteyi yayına alırken güvenlik açısından ideal olmayan bir kaç seçim yaptık. Güvenlik tavsiyeleri için [Django deployment checklist](https://docs.djangoproject.com/en/2.0/howto/deployment/checklist/" \t "_blank) i inceleyin.

Not Üçlü tırnaklar (''' or """) arasındaki satırlara docstrings denir

HTML "HyperText Markup Language" (HiperMetin İşaretleme Dili) anlamına gelir. **HyperText** (HiperMetin) sayfalar arası bağlantıları destekleyen türden bir metin demektir. **Markup** (işaretleme), bir belgeyi alıp onu kodlarla işaretleyerek, nasıl yorumlanacağını (tarayıcıya) söylemek demektir. HTML kodu **etiketler** ile oluşturulur, etiketlerin her biri < ile başlar ve > ile biter. Bu etiketler biçimlendirme **öğelerini** temsil eder.

## Bir şey daha: deployment (yayına alma) zamanı!

### Kodumuzu commit ve push komutları ile Github'a yükleyelim

Cmd

$ git status

$ git add --all .

* **Not** --all demek eğer git'in silinmiş dosyaları varsa onları tanımayı sağlar (varsayılan olarak sadece yeni/değiştirilmiş dosyaları tanır). Hatırlatma: . içinde olduğumuz klasör anlamına gelir.
* Dosyalarımızı yüklemeden önce git'in hangilerini yükleyeceğine (git'in yükleyeceği dosyalar yeşil gösterilir) bakalım:

$ git status

Commit için değişiklikleri açıklayan bir mesaj yazalım.

$ git commit -m "Site için HTML dosyasını değiştirdim."

* Bunu bir kez yapıp, değişikliklerimizi GitHub'a yükledik (ittik):

$ git push

### Pull ile yeni kodu PythonAnywhere e alıp web uygulamasını tekrar yükleyelim

* [PythonAnywhere consoles page](https://www.pythonanywhere.com/consoles/) sayfasını ve **Bash console** u açalım (ya da yeni bir tane açalım). Sonra da çalıştıralım:

PythonAnywhere komut satırı

$ cd ~/<your-pythonanywhere-username>.pythonanywhere.com

$ git pull

[...]

* (Açı parantezleri olmadan <your-pythonanywhere-username>'i gerçek PythonAnywhere kullanıcı isminizle değiştirmeyi unutmayın).
* Ve kodunuzun indirilmesini izleyin. Kontrol etmek istersen, **Files** sekmesine gidip kodunu PythonAnywhere üzerinde görebilirsin (Konsol sayfasındaki menü butonundan diğer PythonAnywhere sayfalarına ulaşabilirsin).
  + Son olarak, [Web sekmesi](https://www.pythonanywhere.com/web_app_setup/)ne gidip uygulamanızın **Yenile** butonuna basın.

**Django ORM ve QuerySets (SorguSetleri)**

* Django'nun veritabanına nasıl bağlandığını ve veriyi nasıl sakladığını öğreneceğiz.
* QuerySet (SorguSeti), esas olarak, verilen bir modele ait nesnelerin listesidir. QuerySet veritabanından veri okumamıza, veriyi filtrelememize ve sıralamamıza imkan sağlar.

# **Template içinde dinamik veri**

* Bu tam olarak view'lerin yapmasını beklediğimiz şey: modelleri ve template'leri bağlamak. post\_list view 'ımızda göstermek istediğimiz modelleri alıp template'e iletmemiz gerekecek. view ile template içinde neyin (hangi modelin) gösterileceğine karar veriyoruz.
* Django şablonunda (template) bir değişken (variable) yazdırmak için, değişken adını çift kıvrımlı parantez içinde şu şekilde kullanırız:

blog/templates/blog/post\_list.html

{{ posts }}

{% for post in posts %}

<div>

<p>published: {{ post.published\_date }}</p>

<h1><a href="">{{ post.title }}</a></h1>

<p>{{ post.text|linebreaksbr }}</p>

</div>

{% endfor %}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**$ git status**

**[...]**

**$ git add --all .**

**$ git status**

**[...]**

**$ git commit -m "Veritabanından gönderileri göstermek için template'ler değiştirildi."**

**[...]**

**$ git push**

* [**PythonAnywhere**](https://www.pythonanywhere.com/consoles/)**'e bağlanalım ve Bash konsolu'na gidelim (veya yeni bir konsol açalım) ve şunu çalıştıralım:**

**PythonAnywhere komut-satırı**

**$ cd $USER.pythonanywhere.com**

**$ git pull**

**[...]**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***