

Django İle Python Arka Yüz Yazılım Geliştirme

Teknik Bilimler
Meslek Yüksekokulu

Öğr. Gör. Furkan DURMUŞ



Bu Haftanın Ders Kazanımları



Model Kavramı

Field Tipleri

Yazarlar Sayfası



Model Kavramı

```
require 'capybara/re Rails'
```

```
1 Capybara.javascript_driver = :webkit
```

```
2 Category.delete_all; Category.create
```

```
3 Shoulda::Matchers.configure do |config|
```

```
4   config.integrate do |with|
```

```
5     with.test_framework :rspec
```

```
6     with.library :rails
```

```
7   end
```

```
8 end
```

```
9 # Add additional requires below this line
```

```
10 # Requires supporting ruby files with support/ and its subdirectories. This will be
```

```
11 # spec/support/ and its subdirectories. This will be run as spec files by default. This
```

```
12 # in _spec.rb will both be required and run. It is recommended that you can
```

```
13 # in _spec.rb will both be required and run. It is recommended that you can
```



Neden veritabanına ihtiyaç var?



Db uygulama



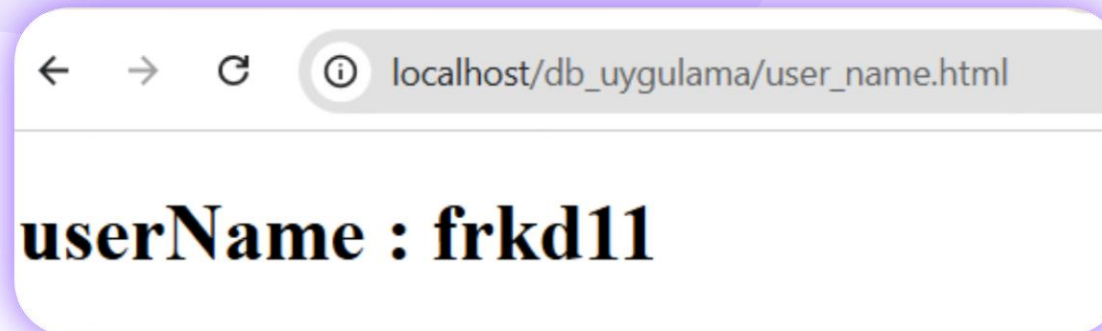
```
user_name.html X username.txt user_name.php
user_name.html > html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <h1>userName : frkd11</h1>
10 </body>
11 </html>
```

Üstteki kodun çıktısı ne olur ?

Db uygulama



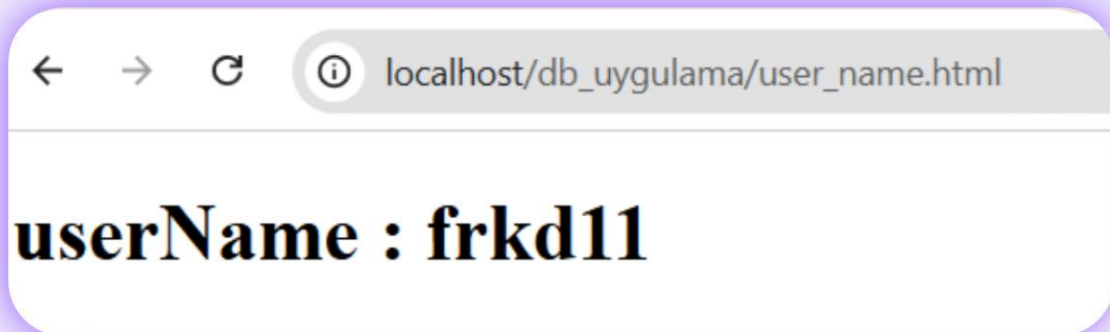
```
user_name.html X username.txt user_name.php
user_name.html > html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <h1>userName : frkd11</h1>
10 </body>
11 </html>
```



Db uygulama



```
user_name.html X username.txt user_name.php
user_name.html > html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <h1>userName : frkd11</h1>
10 </body>
11 </html>
```



userName'ı güncellemek için kaynak kaynak kodunu güncelleme gerekiyor.

Db uygulama

```
user_name.php > ...
1 <?php
2
3 // username.txt dosyasından bir kullanıcı adı oku
4 $dosyaAdi = 'username.txt';
5
6 ?>
7 <!DOCTYPE html>
8 <html lang="tr">
9 <head>
10 <meta charset="UTF-8">
11 <meta name="viewport" content="width=device-width, initial-scale=1.0">
12 <title>Kullanıcı Adı</title>
13 </head>
14 <body>
15 <h1>
16 <?php
17 $username = trim(file_get_contents($dosyaAdi));
18 echo 'Kullanıcı adı: ' . htmlspecialchars($username, ENT_QUOTES, 'UTF-8');
19 ?>
20 </h1>
21 </body>
</html>
```

username.txt

1	frknd12
---	---------

username.txt

1	frknd123
---	----------

localhost/db_uygulama/user_name.php

Kullanıcı adı: frknd12

localhost/db_uygulama/user_name.php

Kullanıcı adı: frknd123

userName'i bir kaynaktan okuyor böylece kaynak koda erişmem gerekiyor.



Djangoda Model Nedir ?



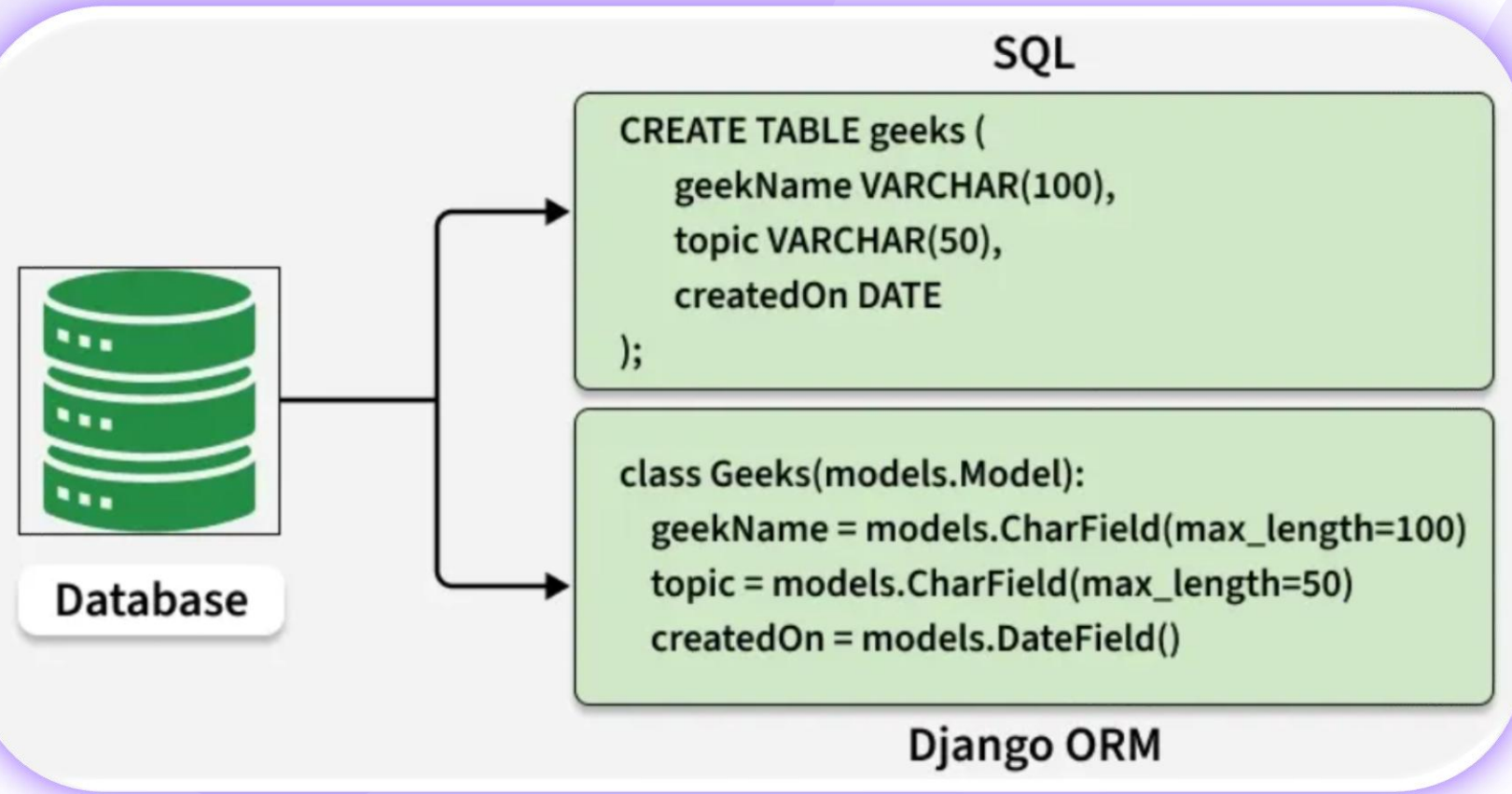


Djangoda Model Nedir ?

Model, uygulamada saklamak istediğimiz verinin yapısını tanımladığımız Python sınıfıdır.



Model Tablo ilişkisi



Django modeli ↔ Veritabanı tablosu

- Model **sınıfı** → veritabanında bir **tablo**
- Model **alanları** → tabloda **sütunlar**
- Model **nesnesi** → tabloda bir **satır**



Model Kavramı



```
core > models.py > BlogPost
1  from django.db import models
2
3  class BlogPost(models.Model):
4      title = models.CharField(max_length=200)
5      content = models.TextField()
6      created_at = models.DateTimeField(auto_now_add=True)
```

Örnek model



Model Kavramı

```
core > models.py > BlogPost
1  from django.db import models
2
3  class BlogPost(models.Model):
4      title = models.CharField(max_length=200)
5      content = models.TextField()
6      created_at = models.DateTimeField(auto_now_add=True)
```

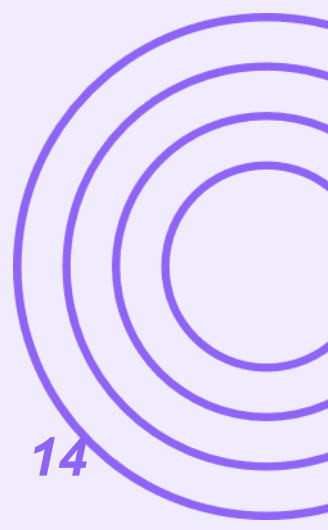
id	title	content	created_at
1	Django'ya Giriş	Django güçlü bir frameworktür...	2026-02-18 10:00
2	Template Kullanımı	Template yapısı sayesinde...	2026-02-18 11:30

Model Kavramı

```
core > models.py > BlogPost
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```

id	title	content	created_at
1	Django'ya Giriş	Django güçlü bir frameworktür...	2026-02-18 10:00
2	Template Kullanımı	Template yapısı sayesinde...	2026-02-18 11:30

Tablo adı =>



Model Kavramı

```
core > models.py > BlogPost
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```

id	title	content	created_at
1	Django'ya Giriş	Django güçlü bir frameworktür...	2026-02-18 10:00
2	Template Kullanımı	Template yapısı sayesinde...	2026-02-18 11:30

Tablo adı => BlogPost
Sutunlar =>

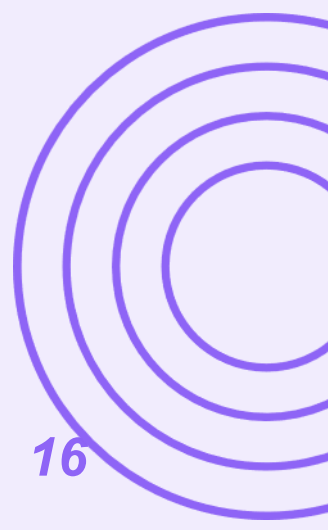


Model Kavramı

```
core > models.py > BlogPost
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```

id	title	content	created_at
1	Django'ya Giriş	Django güçlü bir frameworktür...	2026-02-18 10:00
2	Template Kullanımı	Template yapısı sayesinde...	2026-02-18 11:30

Tablo adı => BlogPost
 Sutunlar => title, content, created_at

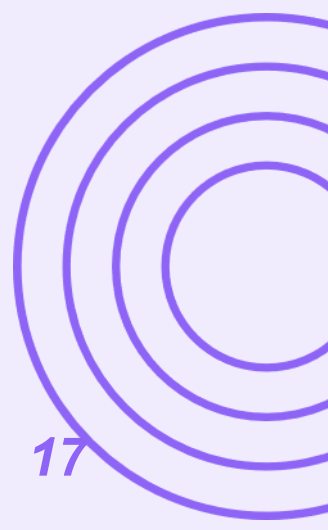


Model Kavramı

```
core > models.py > BlogPost
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```

id	title	content	created_at
1	Django'ya Giriş	Django güçlü bir frameworktür...	2026-02-18 10:00
2	Template Kullanımı	Template yapısı sayesinde...	2026-02-18 11:30

- Tablo adı => BlogPost
- Sutunlar => title, content, created_at
- Nesneler =>

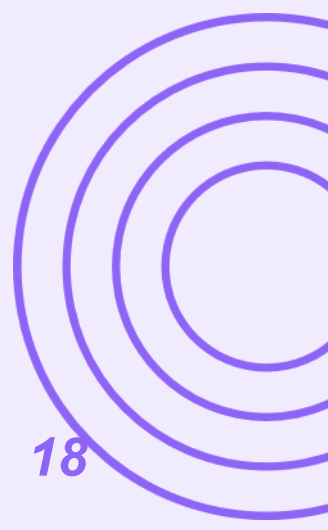


Model Kavramı

```
core > models.py > BlogPost
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```

id	title	content	created_at
1	Django'ya Giriş	Django güçlü bir frameworktür...	2026-02-18 10:00
2	Template Kullanımı	Template yapısı sayesinde...	2026-02-18 11:30

- Tablo adı => BlogPost
- Sutunlar => title, content, created_at
- Nesneler => Her satır



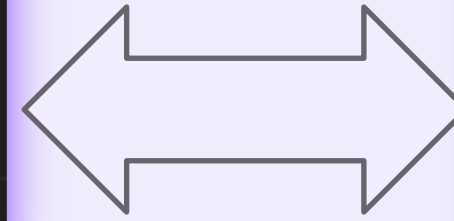
Model Kavramı

```
core > models.py > BlogPost
1  from django.db import models
2
3  class BlogPost(models.Model):
4      title = models.CharField(max_length=200)
5      content = models.TextField()
6      created_at = models.DateTimeField(auto_now_add=True)
```

Madem veritabanı işlemi yapıyoruz, neden SQL yazmıyoruz da Python class yazıyoruz?

Model Kavramı

```
core > models.py > BlogPost
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```



```
1 CREATE TABLE post (
2     id INTEGER PRIMARY KEY,
3     title VARCHAR(200),
4     content TEXT,
5     created_at DATETIME
6 );
```

Madem veritabanı işlemi yapıyoruz, neden SQL yazmıyoruz da Python class yazıyoruz?

- Django, veritabanı ile doğrudan uğraşmayı kolaylaştırır. Biz veriyi Python tarafında model olarak tanımlarız, Django bunu arka planda veritabanına uygun hale getirir.

Model Kavramı - ORM

```
core > models.py > BlogPost
1  from django.db import models
2
3  class BlogPost(models.Model):
4      title = models.CharField(max_length=200)
5      content = models.TextField()
6      created_at = models.DateTimeField(auto_now_add=True)
```

ORM = Object Relational Mapping

Veritabanı tablolarını ilgili programla dilinin objelerine çeviren sistemdir.

Django da ORM zorunlu mudur?

Model Kavramı - ORM



```
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```

ORM

```
from django.db import connection

def get_posts():
    with connection.cursor() as cursor:
        cursor.execute("""
            CREATE TABLE blog_blogpost (
                id BIGSERIAL PRIMARY KEY,
                title VARCHAR(200) NOT NULL,
                content TEXT NOT NULL,
                created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP
            );
        """)
```

Raw SQL

Django da ORM zorunlu **değildir**.



Model Kavramı - ORM



```
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```

ORM

ORM Avantajları :

```
from django.db import connection

def get_posts():
    with connection.cursor() as cursor:
        cursor.execute("""
            CREATE TABLE blog_blogpost (
                id BIGSERIAL PRIMARY KEY,
                title VARCHAR(200) NOT NULL,
                content TEXT NOT NULL,
                created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP
            );
        """)
```

Raw SQL



Model Kavramı - ORM



```
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```

ORM

```
from django.db import connection

def get_posts():
    with connection.cursor() as cursor:
        cursor.execute("""
            CREATE TABLE blog_blogpost (
                id BIGSERIAL PRIMARY KEY,
                title VARCHAR(200) NOT NULL,
                content TEXT NOT NULL,
                created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP
            );
        """)
```

Raw SQL

ORM Avantajları :

> Daha hızlı geliştirmeye açıktır.



Model Kavramı - ORM



```
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```

ORM

```
from django.db import connection

def get_posts():
    with connection.cursor() as cursor:
        cursor.execute("""
            CREATE TABLE blog_blogpost (
                id BIGSERIAL PRIMARY KEY,
                title VARCHAR(200) NOT NULL,
                content TEXT NOT NULL,
                created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP
            );
        """)
```

Raw SQL

ORM Avantajları :

- > Daha hızlı geliştirmeye açıktır.
- > Kod daha okunaklıdır.



Model Kavramı - ORM



```
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```

ORM

```
from django.db import connection

def get_posts():
    with connection.cursor() as cursor:
        cursor.execute("""
            CREATE TABLE blog_blogpost (
                id BIGSERIAL PRIMARY KEY,
                title VARCHAR(200) NOT NULL,
                content TEXT NOT NULL,
                created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP
            );
        """)
```

Raw SQL

ORM Avantajları :

- > Daha hızlı geliştirmeye açıktır.
- > Kod daha okunaklıdır.
- > Daha güvenlidir.



Model Kavramı - ORM

```
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```

ORM

```
from django.db import connection

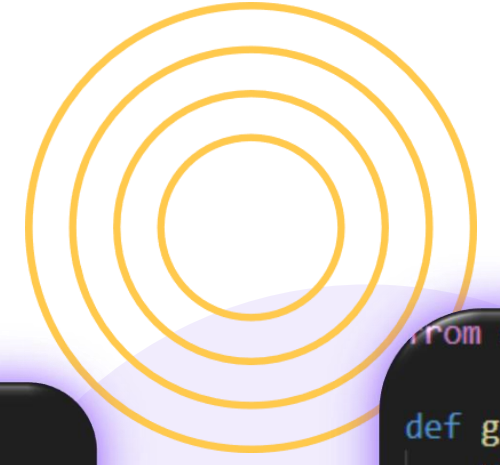
def get_posts():
    with connection.cursor() as cursor:
        cursor.execute("""
            CREATE TABLE blog_blogpost (
                id BIGSERIAL PRIMARY KEY,
                title VARCHAR(200) NOT NULL,
                content TEXT NOT NULL,
                created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP
            );
        """)
```

Raw SQL

ORM Avantajları :

- > Daha hızlı geliştirmeye açıktır.
- > Kod daha okunaklıdır.
- > Daha güvenlidir.
- > Db bağımsızdır.

Model Kavramı



```
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```

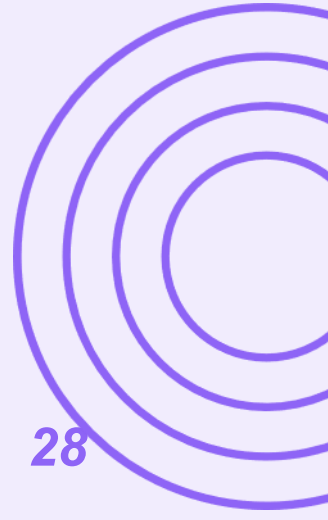
ORM

```
from django.db import connection

def get_posts():
    with connection.cursor() as cursor:
        cursor.execute("""
            CREATE TABLE blog_blogpost (
                id BIGSERIAL PRIMARY KEY,
                title VARCHAR(200) NOT NULL,
                content TEXT NOT NULL,
                created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP
            );
        """)
```

Raw SQL

Hangisi daha performanslıdır ?



Model Kavramı



```
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```

ORM

```
from django.db import connection

def get_posts():
    with connection.cursor() as cursor:
        cursor.execute("""
            CREATE TABLE blog_blogpost (
                id BIGSERIAL PRIMARY KEY,
                title VARCHAR(200) NOT NULL,
                content TEXT NOT NULL,
                created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP
            );
        """)
```

Raw SQL

Hangisi daha performanslıdır ?

Raw SQL daha hızlıdır çünkü doğrudan veritabanına gider ve ekstra katman yoktur; ancak ORM daha yavaştır çünkü SQL'e çeviri yapan bir ara katman içerir ama geliştirme açısından ORM her zaman daha pratiktir.



Model oluşturmadan önce



1- Model hangi varlığı temsil ediyor?

Örneğin: Blog yazısı mı, kategori mi, yorum mu?





Model oluşturmadan önce



1- Model hangi varlığı temsil ediyor?

2- Hangi alanlara ihtiyaç var?

Örneğin BlogPost için:

başlık

içerik

tarih

yazar

yayın durumu



Model oluşturmada önce



- 1- Model hangi varlığı temsil ediyor?
- 2- Hangi alanlara ihtiyaç var?
- 3- Hangi alan zorunlu?**
Başlık zorunlu gibi.



Model oluşturmadan önce



- 1- Model hangi varlığı temsil ediyor?
- 2- Hangi alanlara ihtiyaç var?
- 3- Hangi alan zorunlu?
- 4- Veri tipi ne olmalı?**

Başlık kısa metin, içerik uzun metin, tarih ise datetime olabilir.



Model oluşturmada önce



- 1- Model hangi varlığı temsil ediyor?
- 2- Hangi alanlara ihtiyaç var?
- 3- Hangi alan zorunlu?
- 4- Veri tipi ne olmalı?
- 5- Başka modellerle ilişkisi var mı?**

Her yazının bir yazarı vardır.



models.py Rolü



Django uygulamasında modeller ilgili app'in models.py dosyasında tanımlanır.

```
blog > models.py > ...
1  from django.db import models
2
3  class BlogPost(models.Model):
4      title = models.CharField(max_length=200)
5      content = models.TextField()
6      created_at = models.DateTimeField(auto_now_add=True)
7
8
```

ORM kullanabilmek için "**from django.db import models**" yazılmalı ardından ilgili class tanımlanmalıdır.

models.py



Django uygulamasında modeller ilgili app'in models.py dosyasında tanımlanır.



```
blog > models.py > ...
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
7
8
```

Bir app içindeki model başka bir app'de import edilerek kullanılabilir.



```
core > views.py > ...
1 from django.shortcuts import render
2 import blog.models
3
4 def anasayfa(request):
5     posts = blog.models.BlogPost.objects.all()
6     return render(request, "home.anasayfa", {"posts": posts})
7
8
```



Models Kavrama Soruları



Model nedir?





Models Kavrama Soruları



Model nedir?

Model, veritabanındaki veriyi temsil eden Python sınıfıdır.



Models Kavrama Soruları



Model nedir?

Bir model veritabanında neye karşılık gelir?



Models Kavrama Soruları



Model nedir?

Bir model veritabanında neye karşılık gelir?

Bir model, veritabanında bir tabloya karşılık gelir.



Models Kavrama Soruları



Model nedir?

Bir model veritabanında neye karşılık gelir?

Model alanları tabloda neyi temsil eder?



Models Kavrama Soruları



Model nedir?

Bir model veritabanında neye karşılık gelir?

Model alanları tabloda neyi temsil eder?

Model alanları, tabloda sütunları (columns) temsil eder.



Models Kavrama Soruları



Model nedir?

Bir model veritabanında neye karşılık gelir?

Model alanları tabloda neyi temsil eder?

Post modelindeki bir kayıt veritabanında nasıl görünür?



Models Kavrama Soruları



Model nedir?

Bir model veritabanında neye karşılık gelir?

Model alanları tabloda neyi temsil eder?

Post modelindeki bir kayıt veritabanında nasıl görünür?

Post modelindeki her kayıt, tabloda bir satır (row) olarak görünür.





Models Kavrama Soruları



Model nedir?

Bir model veritabanında neye karşılık gelir?

Model alanları tabloda neyi temsil eder?

Post modelindeki bir kayıt veritabanında nasıl görünür?

Django neden doğrudan SQL yerine model kullanıyor?



Models Kavrama Soruları



Model nedir?

Bir model veritabanında neye karşılık gelir?

Model alanları tabloda neyi temsil eder?

Post modelindeki bir kayıt veritabanında nasıl görünür?

Django neden doğrudan SQL yerine model kullanıyor?

Django, geliştirmeyi kolaylaştırmak ve veritabanı işlemlerini soyutlamak için model kullanır.





```
require 'capybara/re Rails'
```

```
1 Capybara.javascript_driver = :webkit
```

```
2 Category.delete_all; Category.create
```

```
13 Shoulda::Matchers.configure do |config|
```

```
14 config.integrate do |with|
```

```
15 with.test_framework :rspec
```

```
16 with.library :rails
```

```
17 end
```

```
18 end
```

```
19 # Add additional requires below this line. For example, you may want to require
```

```
20 # Requires supporting ruby files with support/ and its subdirectories. For example,
```

```
21 # spec/support/ and its subdirectories. This will be required by default.
```

```
22 # run as spec files by default. This will be required by default.
```


```
23 # in _spec.rb will both be required by default. This will be required by default.
```

```
24 # It is recommended that you configure the support directory in your
```


Field Tipleri

Field Tipleri

Field nedir?



```
blog > models.py > ...
1  from django.db import models
2
3  class BlogPost(models.Model):
4      title = models.CharField(max_length=200)
5      content = models.TextField()
6      created_at = models.DateTimeField(auto_now_add=True)
7
8
```



Field Tipleri

Field nedir?

```
blog > models.py > ...
1  from django.db import models
2
3  class BlogPost(models.Model):
4      title = models.CharField(max_length=200)
5      content = models.TextField()
6      created at = models.DateTimeField(auto_now_add=True)
7
8
```

Field Tipleri

Field nedir?

Field, model içinde tanımlanan ve veritabanında bir sütuna karşılık gelen veri alanıdır.

```
blog > models.py > ...  
1  from django.db import models  
2  
3  class BlogPost(models.Model):  
4      title = models.CharField(max_length=200)  
5      content = models.TextField()  
6      created_at = models.DateTimeField(auto_now_add=True)  
7  
8
```

Field Tipleri

Yandaki koddan incelediğiniz üzere BlogPost class'ındaki field tipleri nelerdir?

```
blog > models.py > ...
1  from django.db import models
2
3  class BlogPost(models.Model):
4      title = models.CharField(max_length=200)
5      content = models.TextField()
6      created_at = models.DateTimeField(auto_now_add=True)
7
8
```

Field Tipleri

Yandaki koddan incelediğiniz üzere BlogPost class'ındaki field tipleri nelerdir?

```
blog > models.py > ...  
1 from django.db import models  
2  
3 class BlogPost(models.Model):  
4     title = models.CharField(max_length=200)  
5     content = models.TextField()  
6     created_at = models.DateTimeField(auto_now_add=True)  
7  
8
```

Field neden önemlidir?



```
log > models.py > ...  
1 from django.db import models  
2  
3 class BlogPost(models.Model):  
4     title = models.CharField(max_length=200)  
5     content = models.TextField()  
6     created_at = models.DateTimeField(auto_now_add=True)  
7  
8
```

Yanlış field seçimi şu sorunlara yol açar:

- veri sığmaz (max_length hatası),



Field neden önemlidir?



```
log > models.py > ...
1 from django.db import models
2
3 class BlogPost(models.Model):
4     title = models.CharField(max_length=200)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
7
8
```

Yanlış field seçimi şu sorunlara yol açar:

- veri sığmaz (max_length hatası),
- performans düşer,



Field neden önemlidir?



```
log > models.py > ...  
1 from django.db import models  
2  
3 class BlogPost(models.Model):  
4     title = models.CharField(max_length=200)  
5     content = models.TextField()  
6     created_at = models.DateTimeField(auto_now_add=True)  
7  
8
```

Yanlış field seçimi şu sorunlara yol açar:

- veri sığmaz (max_length hatası),
- performans düşer,
- veri türü yanlış olur (tarih yerine string gibi),



Field neden önemlidir?



```
blog > models.py > ...  
1 from django.db import models  
2  
3 class BlogPost(models.Model):  
4     title = models.CharField(max_length=200)  
5     content = models.TextField()  
6     created_at = models.DateTimeField(auto_now_add=True)  
7  
8
```

Yanlış field seçimi şu sorunlara yol açar:

- veri sığmaz (max_length hatası),
- performans düşer,
- veri türü yanlış olur (tarih yerine string gibi),
- Veri doğrulaması zorlaşır.



CharField

Kısa metinleri tutar.

```
class BlogPost(models.Model):  
    title = models.CharField(max_length=200)
```

CharField

Kısa metinleri tutar.

- max_length zorunludur.
- veritabanında genelde VARCHAR olarak tutulur.


```
class BlogPost(models.Model):  
    title = models.CharField(max_length=200)
```

CharField

Kısa metinleri tutar.

Ne zaman kullanılır?

- Başlık
- İsim
- Kategori
- Kısa açıklamalar vb.



```
class BlogPost(models.Model):  
    title = models.CharField(max_length=200)
```



TextField

Uzun metinleri tutar.

```
class BlogPost(models.Model):  
    content = models.TextField()
```

TextField

Uzun metinleri tutar.

- max_length zorunlu değildir.
- Büyük metinler için optimize edilmiştir.
- veritabanında genelde TEXT olarak tutulur.

```
class BlogPost(models.Model):  
    content = models.TextField()
```

TextField

Uzun metinleri tutar.

Ne zaman kullanılır?

- blog yazısı içeriği
- Açıklamalar
- Yorumlar vb.

```
class BlogPost(models.Model):  
    content = models.TextField()
```

IntegerField

Tam sayıları tutar.

```
class BlogPost(models.Model):  
    ⚡ author_count = models.IntegerField()
```

`models.IntegerField(default = 5)` ile varsayılan değer atanabilir .

IntegerField

Tam sayıları tutar.
Ne zaman kullanılır?

- Görüntülenme sayısı
- Yaş
- Adet vb.

```
class BlogPost(models.Model):  
    ⚡ author_count = models.IntegerField()
```



BooleanField

True / False değer tutar.

```
✓ class BlogPost(models.Model):  
    published = models.BooleanField(default=False)
```

BooleanField

True / False değer tutar.

default ile varsayılan değer verilebilir.

```
✓ class BlogPost(models.Model):  
    published = models.BooleanField(default=False)
```

BooleanField

True / False değer tutar.
Ne zaman kullanılır?

- yayınlandı mı?
- aktif mi?
- admin mi? Vb.

```
✓ class BlogPost(models.Model):  
    published = models.BooleanField(default=False)
```



DateTimeField / DateField / TimeField



Tarih ve saat tutar.

```
✓ class BlogPost(models.Model):  
    created_at = models.DateTimeField(auto_now_add=True)
```



DateTimeField / DateField / TimeField



Tarih ve saat tutar.
Ne zaman kullanılır?

- oluşturulma tarihi
- güncellenme tarihi vb.

```
✓ class BlogPost(models.Model):  
    created_at = models.DateTimeField(auto_now_add=True)
```



DateTimeField / DateField / TimeField



Tarih ve saat tutar.

```
class BlogPost(models.Model):  
    created_at = models.DateTimeField(auto_now_add=True)  
    updated_at = models.DateTimeField(auto_now=True)
```

- `auto_now_add = True` => sadece ilk oluşturulurken tarih atar.
- `auto_now = True` => her güncellemede değişir.



Field

ÖNEMLİ!

Varsayılan olarak Django alanları zorunlu kabul eder; opsiyonel yapmak için blank, null veya default ile açıkça belirtmek gerekir.

```
✓ class BlogPost(models.Model):  
    created_at = models.DateTimeField(auto_now_add=True)
```

```
✓ class BlogPost(models.Model):  
    published = models.BooleanField(default=False)
```

```
class BlogPost(models.Model):  
    author_count = models.IntegerField(default=0)
```

```
class BlogPost(models.Model):  
    content = models.TextField(blank=True, null=True)
```

```
class BlogPost(models.Model):  
    title = models.CharField(max_length=200, blank=True)
```

Veritabanına veri kaydedilirken alanın boş olup olamayacağı açıkça belirtilmelidir.

Diğer Field Tipleri



EmailField – URLField – SlugField – ImageField – FileField

İhtiyaç duyulduğunda kullanılan ve ilgili veri tipine uygun doğrulama (validation) işlemlerini otomatik olarak yapan özel alanlardır.



Field Parametreleri

nullable=True

Veritabanında boş olabilir.

```
class BlogPost(models.Model):  
    content = models.TextField(blank=True, nullable=True)
```

Field Parametreleri

blank=True

Formda boş bırakılabilir.

```
class BlogPost(models.Model):  
    content = models.TextField(blank=True, null=True)
```

Field Parametreleri



unique=True

```
class BlogPost(models.Model):  
    email = models.EmailField(unique=True)
```

Aynı değerden sadece bir tane olabilir.





Field Parametreleri

max_length

CharField için zorunlu.

```
class BlogPost(models.Model):  
    title = models.CharField(max_length=200, blank=True)
```

Field Tipleri

```
class BlogPost(models.Model):  
    title = models.CharField(max_length=200)  
    content = models.TextField()  
    created_at = models.DateTimeField(auto_now_add=True)  
    is_published = models.BooleanField(default=True)  
    views = models.IntegerField(default=0)
```

Model – Field Tam Örnek



Yazarlar Sayfası

```
require 'capybara/reils'
```

```
1 Capybara.javascript_driver = :webkit
```

```
2 Category.delete_all; Category.create
```

```
13 Shoulda::Matchers.configure do |config|
```

```
14 config.integrate do |with|
```

```
15 with.test_framework :rspec
```

```
16 with.library :rails
```

```
17 end
```

```
18 end
```

```
19 # Add additional requires below this
```

```
20 # Requires supporting ruby files with support/
```

```
21 # spec/support/ and its subdirectories. These
```

```
22 # run as spec files by default. This will
```

```
23 # in _spec.rb will both be required and
```

```
24 # It is recommended that you configure
```

Yazarlar Sayfası



Kampus Blog Ana Sayfa İletişim Hakkımızda Yazarlar Blog Kullanıcı

yazı ara... Ara

Yazarlarımız

Blogumuza katkı sağlayan değerli yazarlarımız

Ahmet Yılmaz
Backend Developer
Python ve Django ile web uygulamaları geliştiriyor.
Açık kaynak projelerle ilgileniyor.
[Profili Gör](#)

Ayşe Demir
Frontend Developer
Modern web teknolojileri ile kullanıcı dostu arayüzler tasarlamayı seviyor.
[Profili Gör](#)

Mehmet Kaya
Full Stack Developer
Web uygulamaları geliştiren ve yeni teknolojileri öğrenmeye meraklı bir yazılımcı.
[Profili Gör](#)

Zeynep Arslan
UI / UX Designer
Kullanıcı deneyimi ve modern tasarım prensipleri üzerine çalışıyor.
[Profili Gör](#)

Ali Can
Data Analyst
Veri analizi ve makine öğrenmesi konularında projeler geliştiriyor.
[Profili Gör](#)

Elif Şahin
Software Developer
Web geliştirme ve yazılım mimarisi üzerine çalışmalar yapmaktadır.
[Profili Gör](#)

f t G i in

© 2026 Copyright

Model **kullanmadan** core app'i altında Yazarlar sayfası oluşturunuz, templates, urls.py, view.py'ye gibi alanlara ilgili tanımlamaları yapınız. **yazarlar.html'in içi boş olsun.**



Yazarlar Sayfası

```
✓ core
  > __pycache__
  > migrations
  ✓ templates
    <> anasayfa.html
    <> hakkimizda.html
    <> iletisim.html
    <> yazarlar.html
```

Core / templates altında yazarlar.html oluşturunuz.

Yazarlar Sayfası

```
core > views.py > ...
1  from django.shortcuts import render
2
3  def anasayfa(request):
4      return render(request, "anasayfa.html")
5
6
7  def iletisim(request):
8      return render(request, "iletisim.html")
9
10 def hakkimizda(request):
11     return render(request, "hakkimizda.html")
12
13 def yazarlar(request):
14     return render(request, "yazarlar.html")
15
16
```

Core / views.py içerisine yazarlar view tanımlamasını yapınız.

Yazarlar Sayfası

```
core > urls.py > ...
1  from django.urls import path
2  from . import views
3
4
5  urlpatterns = [
6      path('', views.anasayfa, name='anasayfa'),
7      path('iletisim/', views.iletisim, name='iletisim'),
8      path('hakkimizda/', views.hakkimizda, name='hakkimizda'),
9      path('yazarlar/', views.yazarlar, name='yazarlar'),
10 ]
11
12
```

Core / urls.py içerisine yazarlar url ve view tanımlamasını yapınız.

Yazarlar Sayfası



Templates / partials / navbar.html içerisine yazarlar tab'ını ekleyiniz.

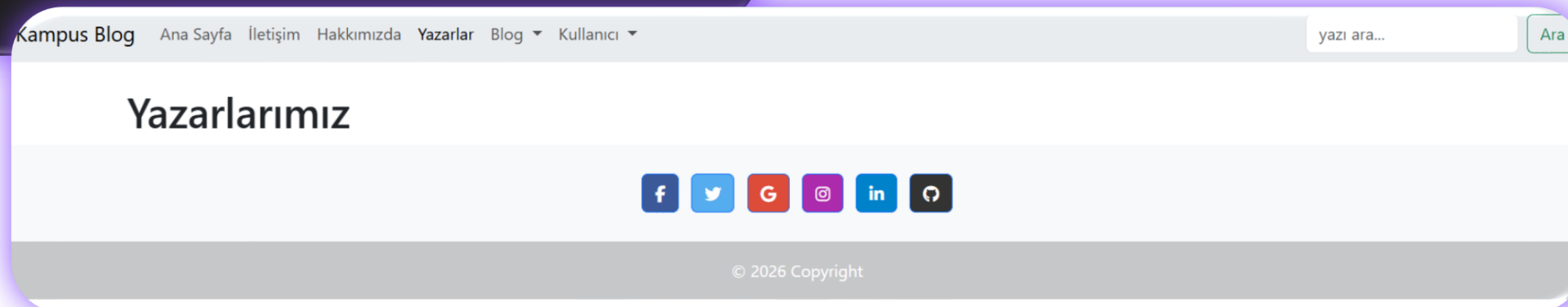
```
templates
├── partials
│   ├── footer.html
│   ├── navbar.html
│   └── base.html
```

```
<li class="nav-item "><a class="nav-link {% if request.resolver_match.url_name == 'yazarlar' %}active{% endif %}" href="{% url 'yazarlar' %}">Yazarlar</a></li>
```

Yazarlar Sayfası

```
core > templates > <> yazarlar.html > ...
1   {% extends "base.html" %}
2   {% load static %}
3
4   {% block baslik %}Yazarlar{% endblock %}
5   {% block icerik %}
6
7   <h1>Yazarlarımız</h1>
8
9   {% endblock %}
10
11
```

Core / Templates / yazarlar.html içerisine temel blokları ekleyiniz ve alttaki görünüm elde ediniz.



Yazarlar Sayfası Model Tanımlaması

```
core > models.py > ...
1  from django.db import models
2
3  class Author(models.Model):
4      full_name = models.CharField(max_length=100)
5      title = models.CharField(max_length=100)
6      bio = models.TextField()
7      profile_url = models.URLField(blank=True, null=True)
8
9  def __str__(self):
10     return self.full_name
11
```

Core / modelys.py içerisine üstteki gibi Author model'ini tanımlayınız.

Yazarlar Sayfası Model Tanımlaması

```
def __str__(self):  
    return self.full_name
```

`def __str__(self) :` ile model çağırıldığı zaman ekrana ne yazılacağı belirlenir.

O class'tan üretilen nesne print edildiğinde, Django objenin kendisini değil, `__str__` metodunun döndürdüğü değeri gösterir.

Yazarlar Sayfası Model Tanımlaması

```
core > views.py > ...
1  from django.shortcuts import render
2  import core.models
3
4  def yazarlar(request):
5      author = core.models.Author(
6          full_name="Ahmet Faruk DURSUN",
7          title="Backend Developer",
8          bio="Python ve Django ile web uygulamaları geliştiriyor.",
9          profile_url="#"
10     )
11
12     context = {
13         "author": author
14     }
15
16     return render(request, "yazarlar.html", context)
17
```

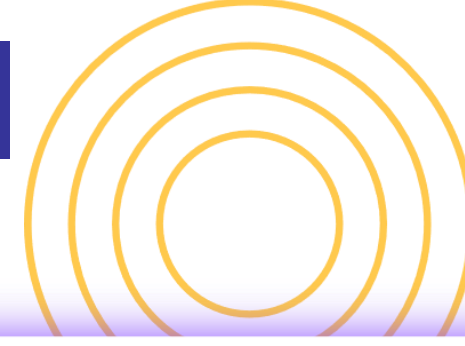
Models 'i import edelim ve models'deki Author'dan bir nesne türetelim. Views.py içindeki yazarlar def'ini üstteki gibi güncelleyelim.

Yazarlar Sayfası Model Tanımlaması

```
core > templates > <> yazarlar.html > ...  
• 1 {% extends "base.html" %}  
2 {% load static %}  
3  
4 {% block baslik %}Yazarlar{% endblock %}  
5 {% block icerik %}  
6  
7 <h1>{{ author.full_name }}</h1>  
8 <p><strong>Unvan:</strong> {{ author.title }}</p>  
9 <p><strong>Hakkında:</strong> {{ author.bio }}</p>  
10  
11 {% endblock %}  
12 |
```

yazarlar.html sayfasını üstteki gibi güncelleyip projeyi çalıştıralım.

Yazarlar Sayfası Model Tanımlaması



Kampus Blog Ana Sayfa İletişim Hakkımızda Yazarlar Blog ▾ Kullanıcı ▾

yazı ara...

Ara

Ahmet Faruk DURSUN

Unvan: Backend Developer

Hakkında: Python ve Django ile web uygulamaları geliştiriyor.



© 2026 Copyright

Sonuç

Yazarlar Sayfası

View içerisinde veriyi ön yüze tam olarak nerede gönderdik ?



Yazarlar Sayfası

View içerisinde veriyi ön yüze tam olarak nerede gönderdik ?

```
context = {  
    "yazarlar": yazarlar  
}  
  
return render(request, "yazarlar.html", context)
```

Yazarlar Sayfası

```
context = {  
    "yazarlar": yazarlar  
}  
  
return render(request, "yazarlar.html", context)
```

Context'in tipi nedir ?

Yazarlar Sayfası

```
context = {  
    "yazarlar": yazarlar  
}  
  
return render(request, "yazarlar.html", context)
```

Render fonksiyonu ile önyüze gönderilen veri her zaman **dictionary** tipinde olmak zorundadır.

Yazarlar Sayfası

```
context = {  
    "yazarlar": yazarlar  
}  
  
return render(request, "yazarlar.html", context)
```

```
return render(request, "yazarlar.html", {"yazarlar": yazarlar})
```

Bu **dictionary**'i ister değişkene atayıp ister doğrudan dict olarak render içinde gönderebiliriz.

Yazarlar Sayfası

```
context = {  
  "yazarlar": yazarlar,  
  "bloglar" :bloglar,  
  "title": "Yazarlar"  
}  
  
return render(request, "yazarlar.html", context)
```

```
core > templates > <> yazarlar.html > ...  
1   {% extends "base.html" %}  
2   {% load static %}  
3  
4   {% block baslik %}{{ title }}{% endblock %}  
5   {% block icerik %}  
6
```

Birden fazla farklı veriyi önyüze gönderebilirim.

Yazarlar Sayfası



```
core > templates > <> yazarlar.html > ...
1   {% extends "base.html" %}
   {% load static %}

   {% block baslik %}{{ title }}{% endblock %}
   {% block icerik %}

6
```

```
context = {
    "yazarlar": yazarlar,
    "title": "Yazarlar"
}

return render(request, "yazarlar.html", context)
```

View'dan önyüze gönderilen verilere **anahtar adı** ile {{ veri }} şeklinde ulaşılabilir.
Üstteki şekilde hem view hem html'i düzenleyiniz.

Yazarlar Sayfası

```
def yazarlar(request):
    yazarlar = [
        core.models.Author(
            full_name="Ahmet Faruk DURSUN",
            title="Backend Developer",
            bio="Python ve Django ile web uygulamaları geliştiriyor.",
            profile_url="#"
        ),
        core.models.Author(
            full_name="Tuncay ALTUN",
            title="Frontend Developer",
            bio="React ve Vue.js ile kullanıcı arayüzleri oluşturuyor.",
            profile_url="#"
        )
    ]

    context = {
        "yazarlar": yazarlar
    }

    return render(request, "yazarlar.html", context)
```

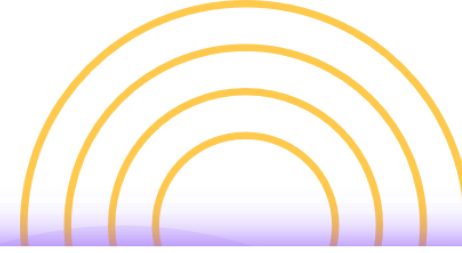
Core/views.py içerisindeki yazarlar def'ini üstteki gibi güncelleyiniz.

Yazarlar Sayfası

```
core > templates > <> yazarlar.html > ...
1   {% extends "base.html" %}
2   {% load static %}
3
4   {% block baslik %}{{ title }}{% endblock %}
5   {% block icerik %}
6
7       {% for yazar in yazarlar %}
8
9           <h1>{{ yazar.full_name }}</h1>
10          <p><strong>Unvan:</strong> {{ yazar.title }}</p>
11          <p><strong>Hakkında:</strong> {{ yazar.bio }}</p>
12
13      {% endfor %}
14
15  {% endblock %}
16
```

Core/templayes/yazarlar.html'i üstteki gibi güncelleyiniz.

Yazarlar Sayfası



Kampus Blog Ana Sayfa İletişim Hakkımızda Yazarlar Blog ▾ Kullanıcı ▾

yazı ara...

Ara

Ahmet Faruk DURSUN

Unvan: Backend Developer

Hakkında: Python ve Django ile web uygulamaları geliştiriyor.

Tuncay ALTUN

Unvan: Frontend Developer

Hakkında: React ve Vue.js ile kullanıcı arayüzleri oluşturuyor.



© 2026 Copyright

Sonuç

Yazarlar Sayfası

View içerisindeki **yazarlar** fonksiyonuna 4 yazar nesnesi daha ekleyiniz.



Yazarlar Sayfası

Yazarlar sayfasını yandaki gibi
yada kendi tasarımınızla
güncelleyiniz.

```
updates > yazarlar.html > ...
1 {% extends "base.html" %}
2 {% load static %}
3
4 {% block baslik %}{{ title }}{% endblock %}
5 {% block icerik %}
6
7 <div class="container mt-5">
8
9   <div class="text-center mb-5">
10     <h1>Yazarlarımız</h1>
11     <p class="text-muted">Blogumuza katkı sağlayan değerli yazarlarımız</p>
12   </div>
13
14   <div class="row g-4">
15     {% for yazar in yazarlar %}
16     <div class="col-md-4">
17       <div class="card text-center shadow-sm h-100">
18         <div class="card-body">
19           <i class="bi bi-person-circle" style="font-size:90px;"></i>
20           <h5 class="card-title mt-3">{{ yazar.full_name }}</h5>
21           <p class="text-muted">{{ yazar.title }}</p>
22           <p class="card-text">
23             {{ yazar.bio }}
24           </p>
25           <a href="{{ yazar.profile_url }}" class="btn btn-outline-primary btn-sm">Profili Gör</a>
26         </div>
27       </div>
28     </div>
29     {% endfor %}
30   </div>
31
32 </div>
33
34 {% endblock %}
```

Pin selection to current chat prompt (Ctrl+Alt+X) | Don't show this again (Alt+I)

Yazarlar Sayfası

Kampus Blog Ana Sayfa İletişim Hakkımızda Yazarlar Blog Kullanıcı

yazı ara... Ara

Yazarlarımız

Blogumuza katkı sağlayan değerli yazarlarımız

Ahmet Yılmaz
Backend Developer
Python ve Django ile web uygulamaları geliştiriyor.
Açık kaynak projelerle ilgileniyor.
Profili Gör

Ayşe Demir
Frontend Developer
Modern web teknolojileri ile kullanıcı dostu arayüzler tasarlamayı seviyor.
Profili Gör

Mehmet Kaya
Full Stack Developer
Web uygulamaları geliştiren ve yeni teknolojileri öğrenmeye meraklı bir yazılımcı.
Profili Gör

Zeynep Arslan
UI / UX Designer
Kullanıcı deneyimi ve modern tasarım prensipleri üzerine çalışıyor.
Profili Gör

Ali Can
Data Analyst
Veri analizi ve makine öğrenmesi konularında projeler geliştiriyor.
Profili Gör

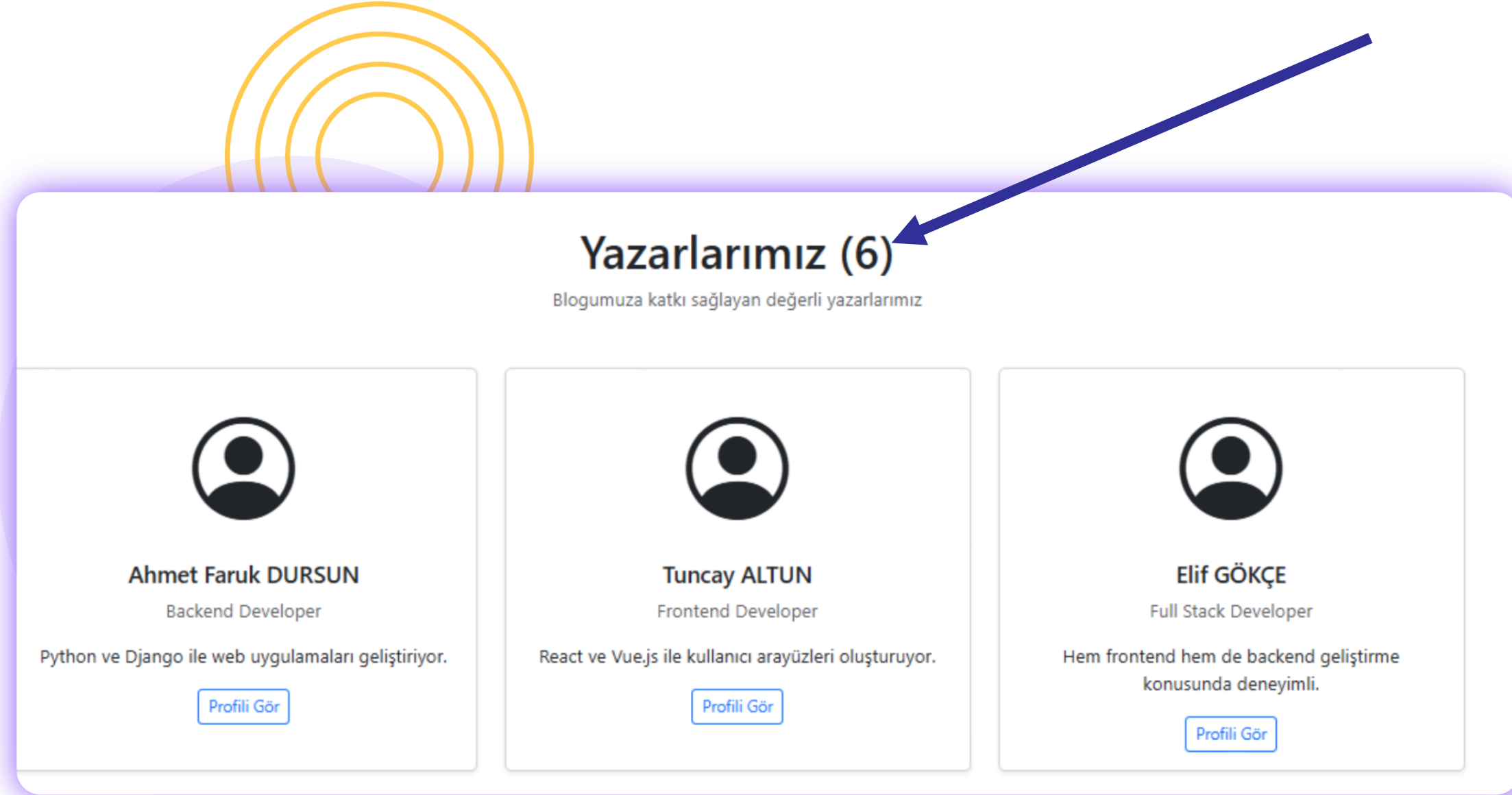
Elif Şahin
Software Developer
Web geliştirme ve yazılım mimarisi üzerine çalışmalar yapmaktadır.
Profili Gör

f t G i in

© 2026 Copyright

Yazarlar Sayfası

Toplam yazar sayısı yandaki gibi dinamik olarak gözükecek şekilde hem view hem html'de gerekli güncellemeyi yapınız.



The screenshot displays a section titled 'Yazarlarımız (6)' with a subtitle 'Blogumuza katkı sağlayan değerli yazarlarımız'. It features three author cards, each with a profile picture, name, title, a short bio, and a 'Profili Gör' button. A blue arrow points to the '(6)' in the title, indicating a dynamic count.

Yazarın Adı	Unvanı	Bio
Ahmet Faruk DURSUN	Backend Developer	Python ve Django ile web uygulamaları geliştiriyor.
Tuncay ALTUN	Frontend Developer	React ve Vue.js ile kullanıcı arayüzleri oluşturuyor.
Elif GÖKÇE	Full Stack Developer	Hem frontend hem de backend geliştirme konusunda deneyimli.

Yazarlar Sayfası

```
context = {  
    "yazarlar": yazarlar,  
    "title": "Yazarlar",  
    "toplam_yazar": len(yazarlar),  
}  
  
return render(request, "yazarlar.html", context)
```

```
<div class="text-center mb-5">  
    <h1>Yazarlarımız ({{ toplam_yazar }})</h1>  
    <p class="text-muted">Blogumuza katkı sağlayan değerli yazarlarımız</p>  
</div>
```

Yazarlarımız (6)

Blogumuza katkı sağlayan değerli yazarlarımız



Ahmet Faruk DURSUN
Backend Developer
Python ve Django ile web uygulamaları geliştiriyor.
[Profili Gör](#)



Tuncay ALTUN
Frontend Developer
React ve Vue.js ile kullanıcı arayüzleri oluşturuyor.
[Profili Gör](#)



Elif GÖKÇE
Full Stack Developer
Hem frontend hem de backend geliştirme konusunda deneyimli.
[Profili Gör](#)

Son