Django İle Python Arka Yüz Yazılım Geliştirme

Teknik Bilimler
Meslek Yüksekokulu

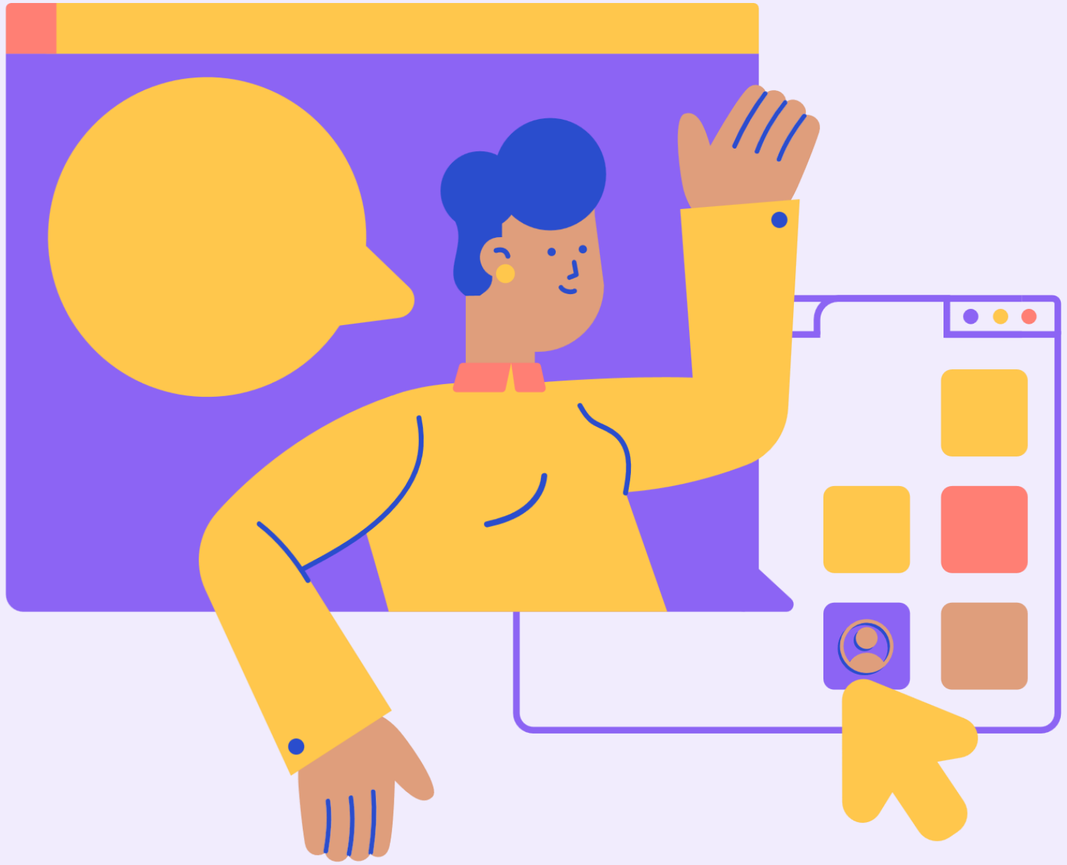
Öğr. Gör. Furkan DURMUŞ

Bu Haftanın Ders Kazanımları

- Model Kavramı
- Field Tipleri
- Yazarlar Sayfası

Model Kavramı

Neden veritabanına ihtiyaç var?



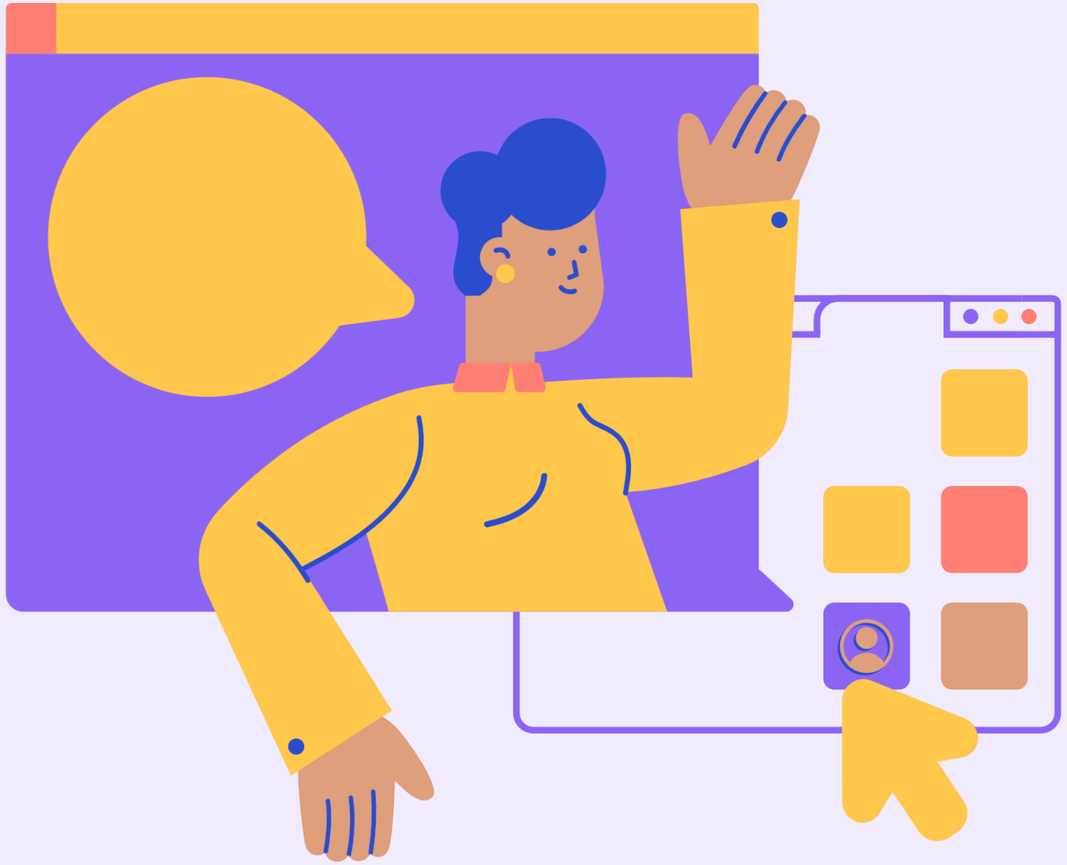
Db uygulama

user_name.html | 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

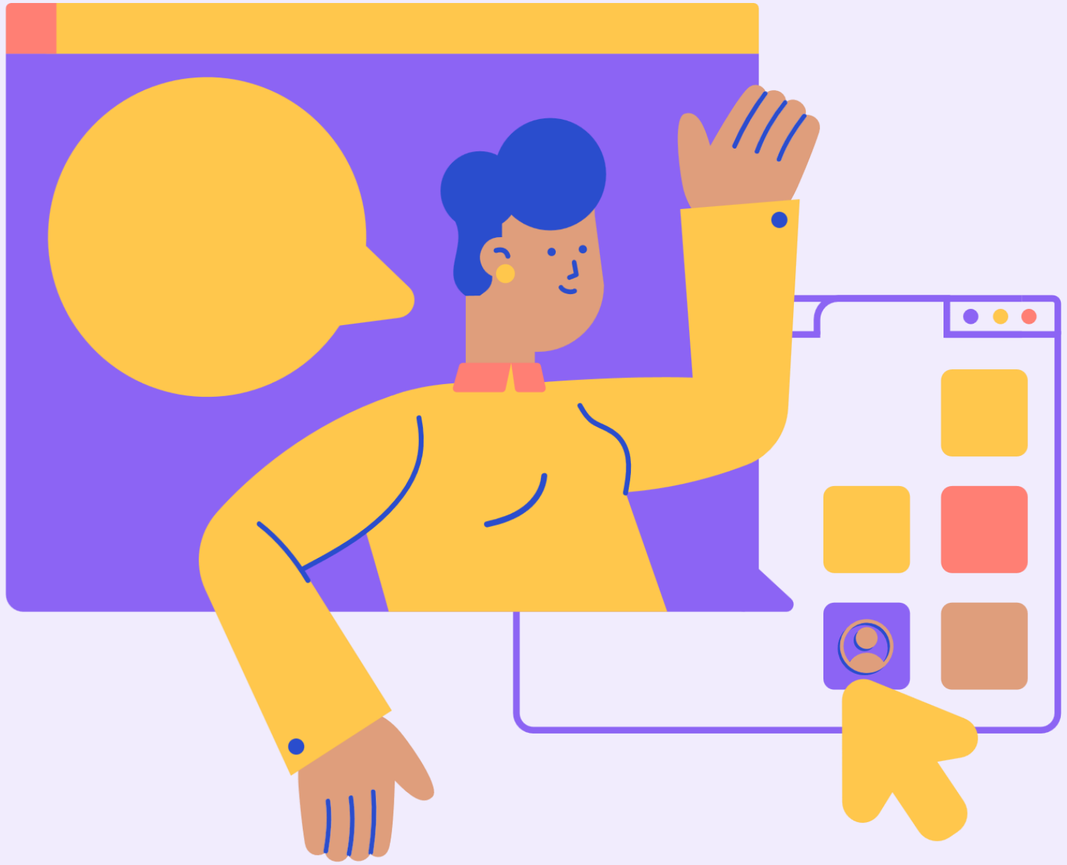
user_name.html | username.txt | user_name.php

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>
```

localhost/db_uygulama/user_name.html

userName : frkd11



Db uygulama

user_name.html | 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>
```

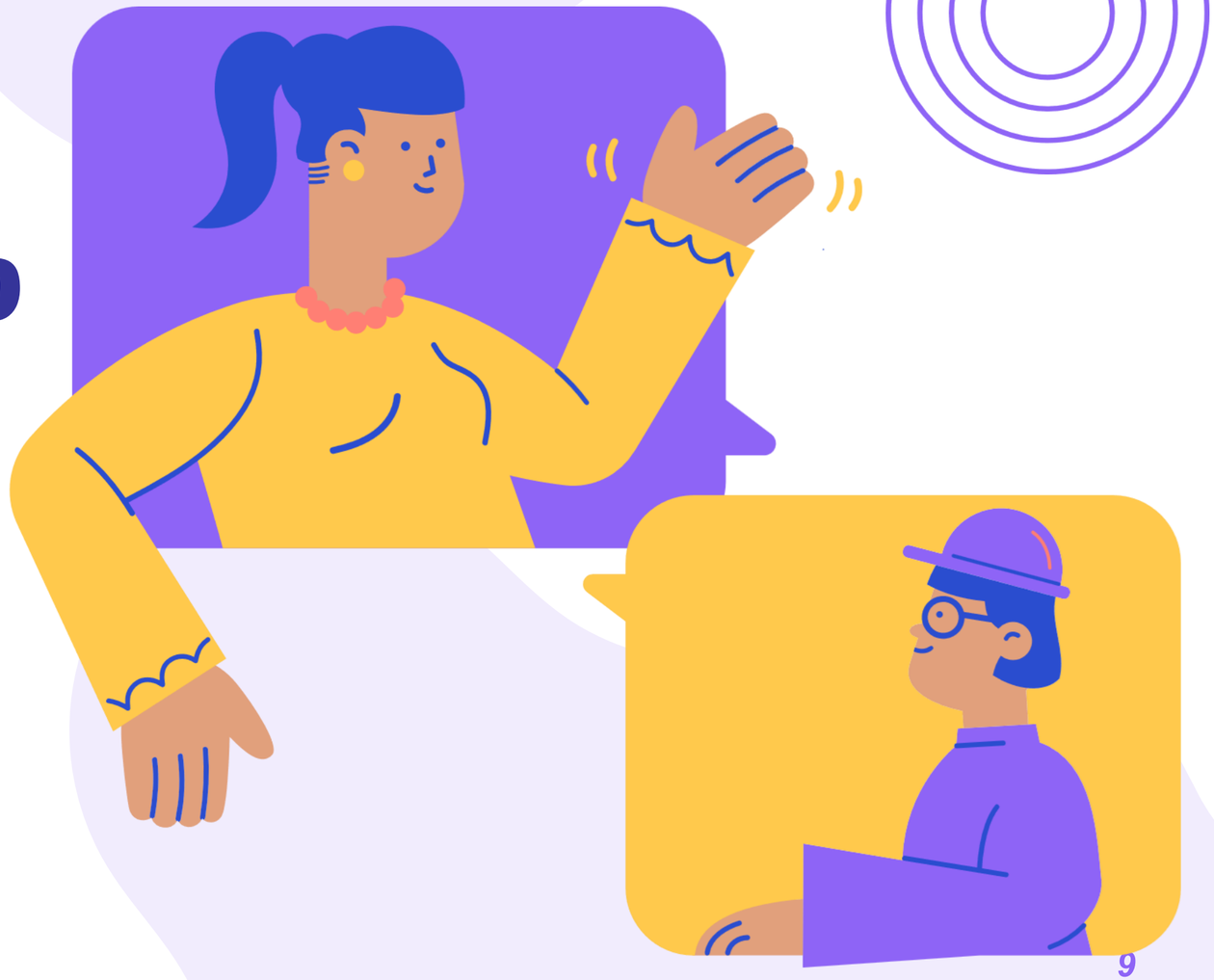
userName'i güncellemek için kaynak kaynak kodunu güncelleme gerekiyor.

Db uygulama

```php
<?php

// username.txt dosyasından bir kullanıcı adı oku
$dosyaAdi = 'username.txt';

?>
<!DOCTYPE html>
<html lang="tr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Kullanıcı Adı</title>
</head>
<body>
    <h1>
    <?php
        $username = trim(file_get_contents($dosyaAdi));
        echo 'Kullanıcı adı: ' . htmlspecialchars($username, ENT_QUOTES, 'UTF-8');
    ?>
    </h1>
</body>
</html>
```

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

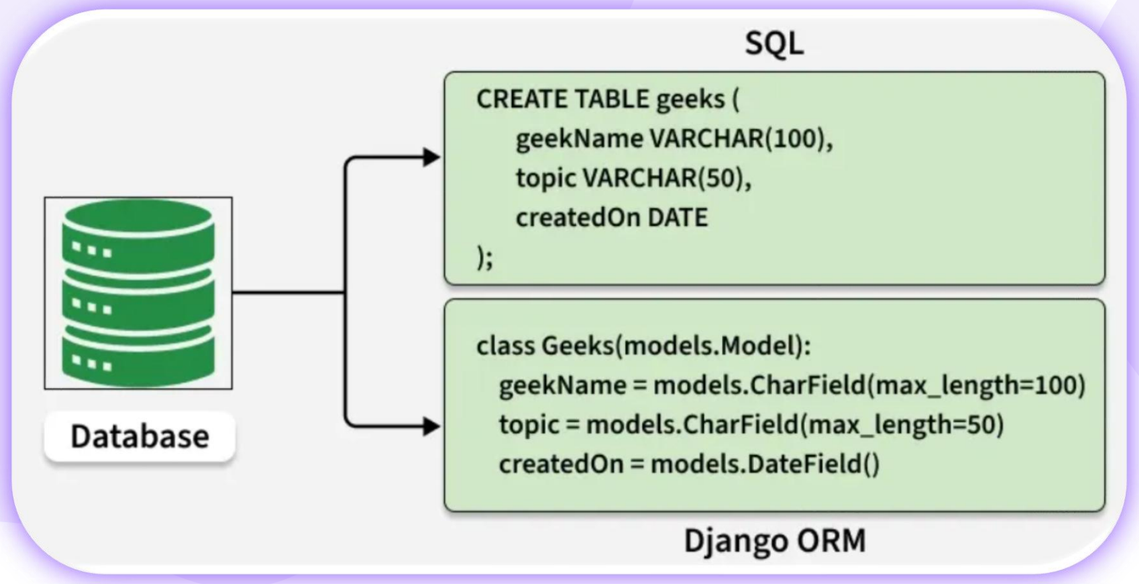
Djangoda Model Nedir ?



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

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

Örnek model

Model Kavramı

core > models.py > BlogPost

```python
from django.db import models

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

```python
from django.db import models

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

```python
from django.db import models

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

```python
from django.db import models

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

```python
from django.db import models

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

```python
from django.db import models

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

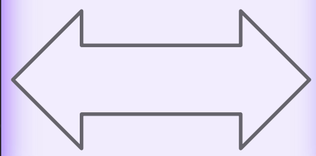
```python
from django.db import models

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

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```



```sql
CREATE TABLE post (
    id INTEGER PRIMARY KEY,
    title VARCHAR(200),
    content TEXT,
    created_at DATETIME
);
```

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

```python
from django.db import models

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

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

ORM

```python
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

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

ORM

```python
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 :

Model Kavramı - ORM

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

ORM

```python
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

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

ORM

```python
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

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

ORM

```python
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

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

ORM

```python
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

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

ORM

```python
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

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

ORM

```python
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 oluşturmadan 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 oluşturmadan 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 > ...

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

```

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

models.py

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

blog > models.py > ...

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

```

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

core > views.py > ...

```python
from django.shortcuts import render
import blog.models

def anasayfa(request):
    posts = blog.models.BlogPost.objects.all()
    return render(request, "home.anasayfa", {"posts": posts})

```

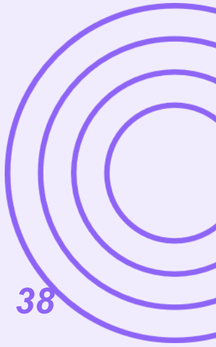
Models Kavrama Soruları

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.

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 > ...

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

```

Field Tipleri

Field nedir?

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

blog > models.py > ...

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

```

Field Tipleri

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

blog > models.py > ...

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

```

Field Tipleri

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

blog > models.py > ...

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

```

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),

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,

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),

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.

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

- max_length zorunludur.
- veritabanında genelde VARCHAR olarak tutulur.

CharField

Kısa metinleri tutar.

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

Ne zaman kullanılır?

- Başlık
- İsim
- Kategori
- Kısa açıklamalar vb.

TextField

Uzun metinleri tutar.

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

TextField

Uzun metinleri tutar.

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

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

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.

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

BooleanField

True / False değer tutar.

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

BooleanField

True / False değer tutar.

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

default ile varsayılan değer verilebilir.

BooleanField

True / False değer tutar.

Ne zaman kullanılır?

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

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

DateTimeField / DateField / TimeField

Tarih ve saat tutar.

```python
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.

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

DateTimeField / DateField / TimeField

Tarih ve saat tutar.

```python
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.

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

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

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

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

```python
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

null=True

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

Veritabanında boş olabilir.

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)
```

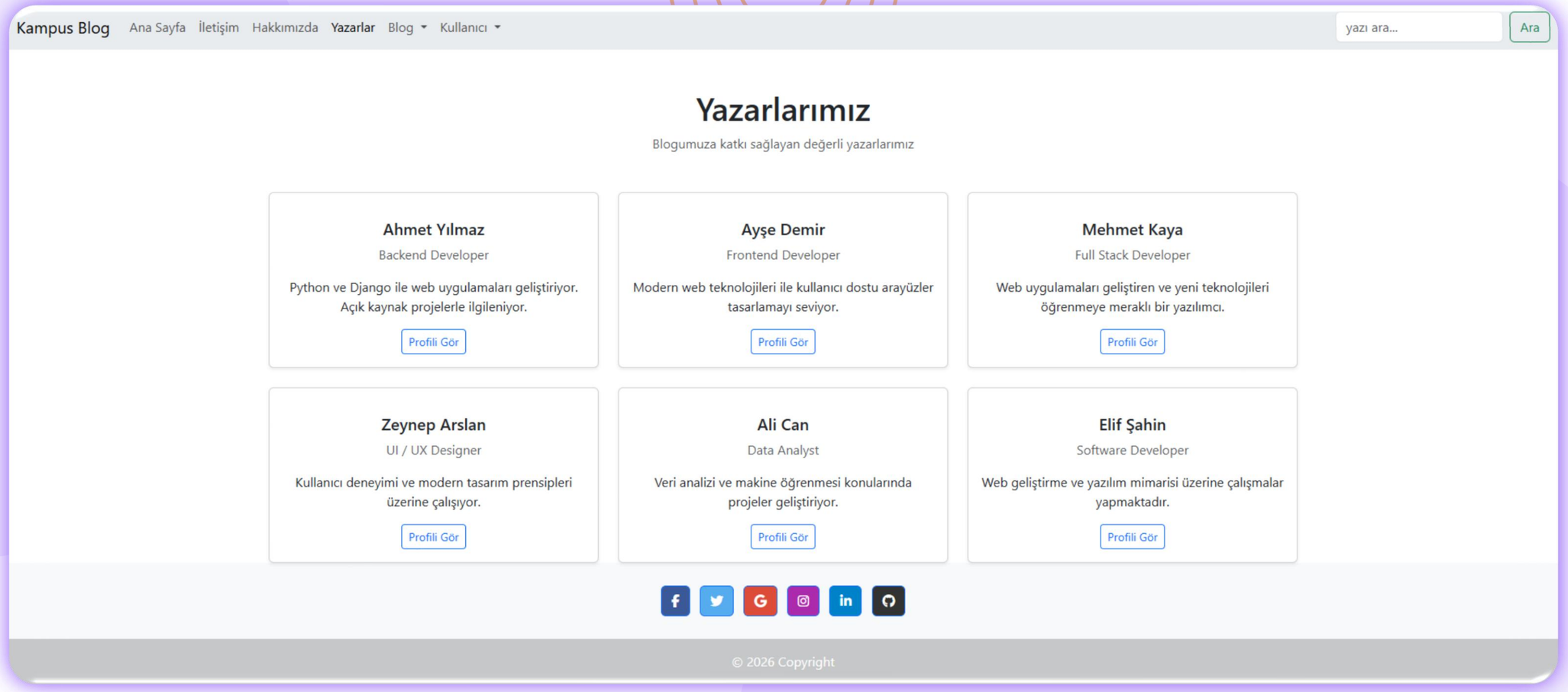
Field Tipleri

```python
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

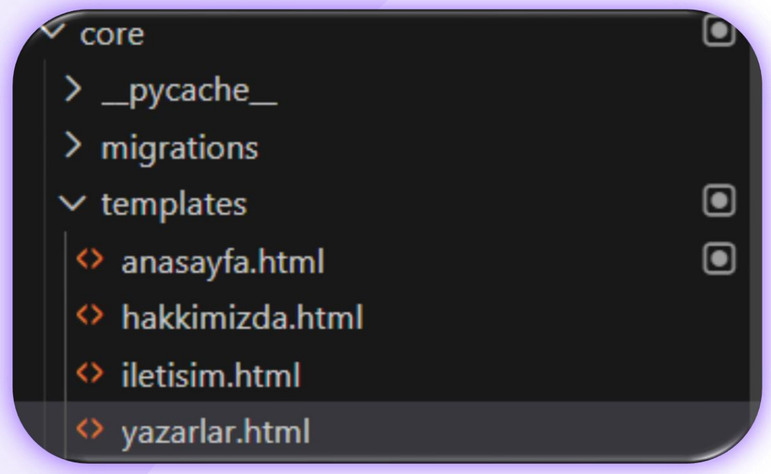
Yazarlar Sayfası

Yazarlar Sayfası



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 / templates altında yazarlar.html oluşturunuz.

Yazarlar Sayfası

core > views.py > ...

```python
from django.shortcuts import render

def anasayfa(request):
    return render(request, "anasayfa.html")


def iletisim(request):
    return render(request, "iletisim.html")

def hakkimizda(request):
    return render(request, "hakkimizda.html")

def yazarlar(request):
    return render(request, "yazarlar.html")

```

Core / views.py içerisine yazarlar view tanımlamasını yapınız.

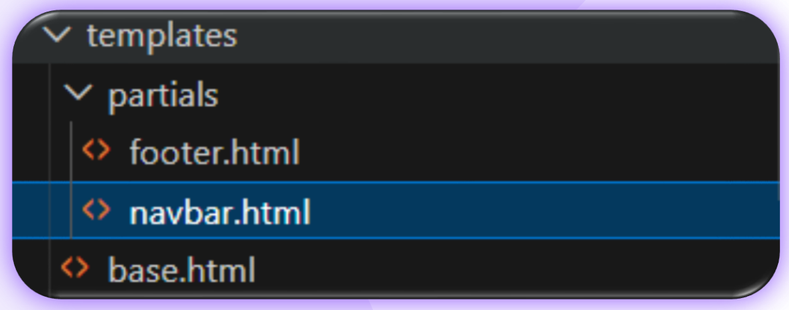
Yazarlar Sayfası

core > urls.py > ...

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


urlpatterns = [
    path('', views.anasayfa, name='anasayfa'),
    path('iletisim/', views.iletisim, name='iletisim'),
    path('hakkimizda/', views.hakkimizda, name='hakkimizda'),
    path('yazarlar/', views.yazarlar, name='yazarlar'),
]

```

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

Yazarlar Sayfası



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

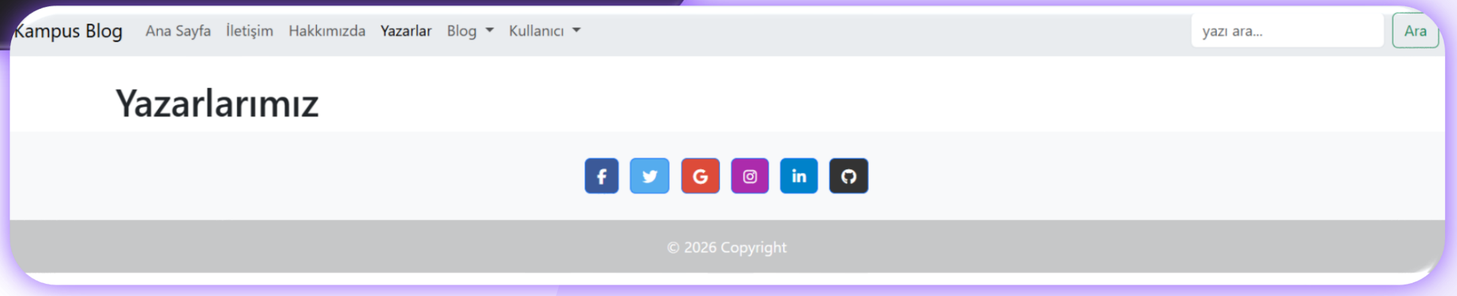
```
<li class="nav-item "><a class="nav-link {% if request.resolver_match.url_name == 'yazarlar' %}active{% endif %}" href="{% url 'yazarlar' %}">Yazarlar</a></li>
```

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 > ...

```python
from django.db import models

class Author(models.Model):
    full_name = models.CharField(max_length=100)
    title = models.CharField(max_length=100)
    bio = models.TextField()
    profile_url = models.URLField(blank=True, null=True)

    def __str__(self):
        return self.full_name

```

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 > ...

```python
from django.shortcuts import render
import core.models

def yazarlar(request):
    author = core.models.Author(
        full_name="Ahmet Faruk DURSUN",
        title="Backend Developer",
        bio="Python ve Django ile web uygulamaları geliştiriyor.",
        profile_url="#"
    )

    context = {
        "author": author
    }

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

```

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

Sonuç

Yazarlar Sayfası

View içerisinden veriyi ön yüze tam olarak nerede gönderdik ?

Yazarlar Sayfası

View içerisinden 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

```
context = {
    "yazarlar": yazarlar,
    "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
```

View'dan önyüze gönderilen verilere **anahtar adı** ile {{ veri }} şeklinde ulaşılabilir.

Üstteki şekilde hem view hem html'i düzenleyiniz.

Yazarlar Sayfası

```python
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.

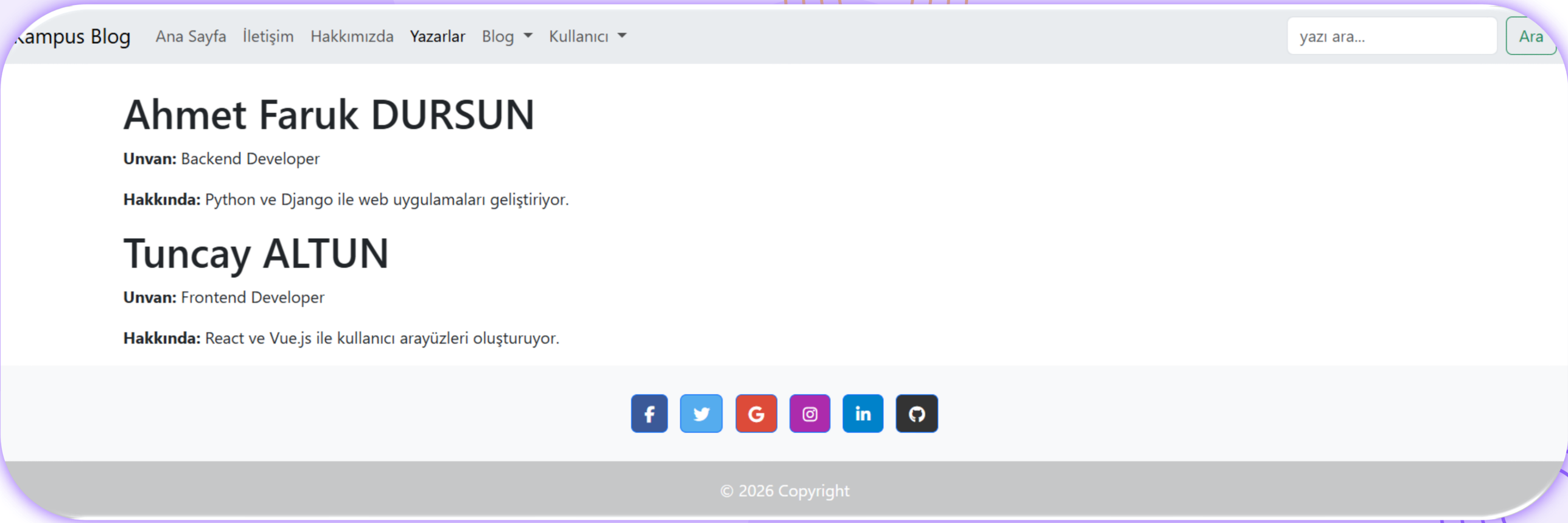
Yazarlar Sayfası

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



Yazarlar Sayfası

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

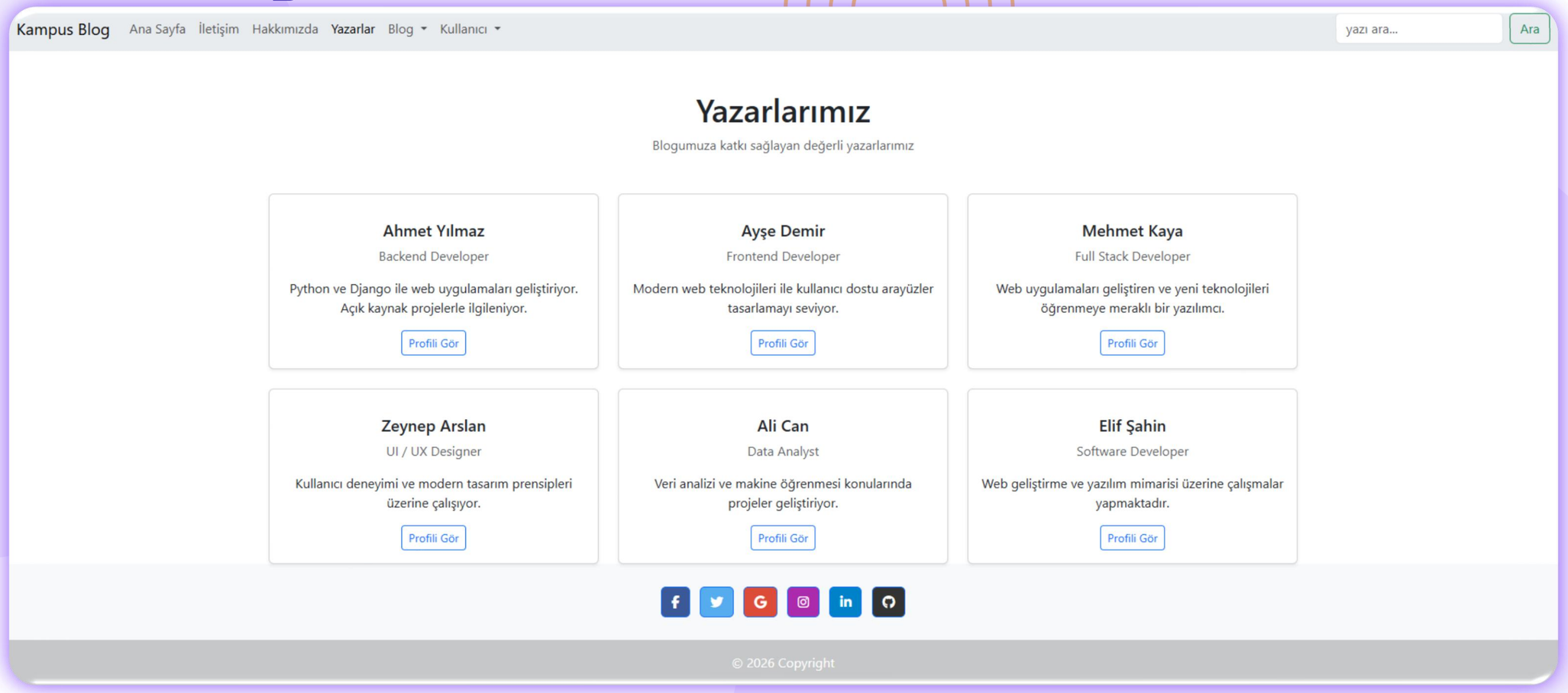
Yazarlar Sayfası

Yazarlar sayfasını yandaki gibi yada kendi tasarımınızla güncelleyiniz.

plates > yazarlar.html > ...

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

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

<div class="container mt-5">

    <div class="text-center mb-5">
        <h1>Yazarlarımız</h1>
        <p class="text-muted">Blogumuza katkı sağlayan değerli yazarlarımız</p>
    </div>

    <div class="row g-4">
        {% for yazar in yazarlar %}
        <div class="col-md-4">
            <div class="card text-center shadow-sm h-100">
                <div class="card-body">
                    <i class="bi bi-person-circle" style="font-size:90px;"></i>
                    <h5 class="card-title mt-3">{{ yazar.full_name }}</h5>
                    <p class="text-muted">{{ yazar.title }}</p>
                    <p class="card-text">
                        {{ yazar.bio }}
                    </p>
                    <a href="{{ yazar.profile_url }}" class="btn btn-outline-primary btn-sm">Profili Gör</a>
                </div>
            </div>
        </div>
        {% endfor %}
    </div>

</div>

{% endblock %}      Pin selection to current chat prompt (Ctrl+Alt+X) | Don't show this again (Alt+/)
```

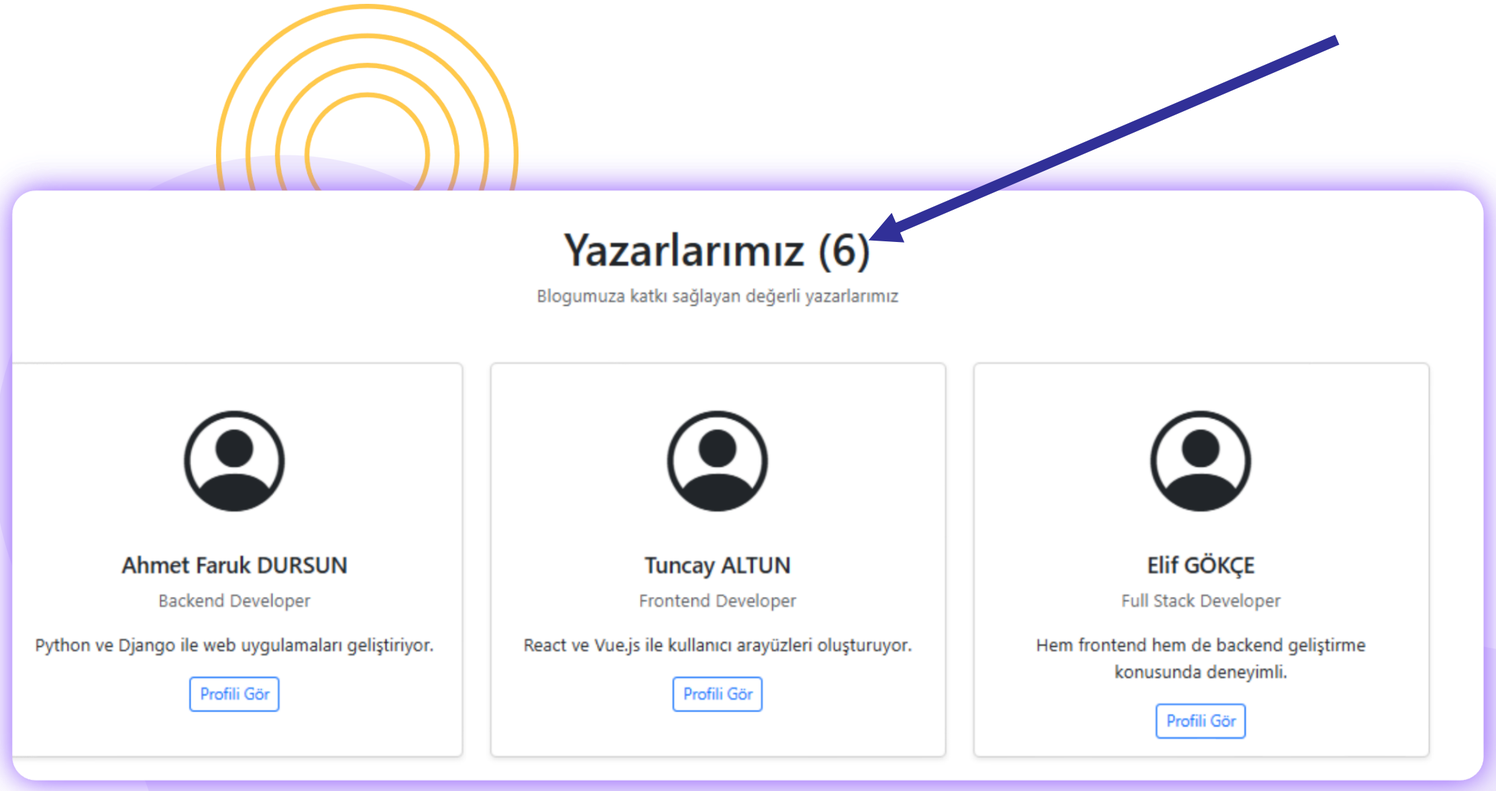
Yazarlar Sayfası



Sonuç

Yazarlar Sayfası

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



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>
```

Son