Web sayfaları header, footer, menu gibi her sayfada görüntülenen standart alanlara sahiptir. Biz şu ana kadar öğrendiğimiz bilgiler ile her view için bunları tekrar tekrar yazmalıyız.

Bu bölümleri oluşturup include ile şablonumuza entegre edebiliriz. Bu sayede her view için tekrar yazmamıza gerek kalmaz.

Yine templatelerde kullanacağımız block ve extends etiketleri ile bir şablonda, başka bir şablonun sadece belli kısımlarını değiştirerek kullanabiliriz.

Şimdi örnek üzerinden nasıl yapacağımıza bakalım. Ürün listeleme sayfamıza baktığımızda bu alanlara her sayfada ortak kullanılacak kısımlar getirebiliriz.

*templates/product/list.html*

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Ürün Listeleme Sayfası</title>  
</head>  
<body>  
  
<!--   
BURAYA HEADER GELSİN  
-->  
  
Bu sayfa ürün listeleme sayfasıdır.  
  
<table>  
 {% for product in products %}  
 <tr>  
 <td>Ürün Adı : {{ product.name | upper }}</td>  
 <td>Ürün Açıklaması : {{ product.content }}</td>  
 </tr>  
 {% endfor %}  
</table>  
  
<!--   
BURAYA FOOTER GELSİN  
-->  
</body>  
</html>

Bunun için öncelikle header ve footer sayfalarımızı oluşturmalıyız. Bunun için templates klasörü içerisine layout klasörü oluşturalım ve bunun içerisine şablonumuzun genel bölümlerini ekleyelim.

*templates/layout/header.html*

BURASI HEADER OLACAK  
<br><br>

*templates/layout/footer.html*

<br><br>

BURASI FOOTER OLACAK

Header ve footer alanlarına gelecek dosyalarımızı oluşturduk. bunları template dosyamıza çağırmak için include etiketinden yararlanırız. İnclude etiketi ve ilgili alana getirecek dosyamızın yolumuzu yazarız.

*templates/product/list.html*

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Ürün Listeleme Sayfası</title>  
</head>  
<body>  
  
{% include 'layout/header.html' %}  
<br><br>  
  
Bu sayfa ürün listeleme sayfasıdır.  
  
<table>  
 {% for product in products %}  
 <tr>  
 <td>Ürün Adı : {{ product.name | upper }}</td>  
 <td>Ürün Açıklaması : {{ product.content }}</td>  
 </tr>  
 {% endfor %}  
</table>  
  
<br><br>  
{% include 'layout/footer.html' %}  
  
</body>  
</html>

Bu header ve footer dosyalarımızı istediğimiz template dosyasına include ile getirebiliriz.

block ve extend yapısı ise bir template dosyasındaki bazı alanları değiştirerek kullanmamıza yarar. Örnek ile daha iyi anlayabiliriz. Layout klasörümüze bir adet base.html dosyası oluştralım.

*templates/Layout/base.html*

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>  
 {% block title %}  
 Base Title  
 {% endblock %}  
 </title>  
</head>  
<body>  
{% include 'layout/header.html' %}

{% block content %}  
 Base content  
{% endblock %}

{% include 'layout/footer.html' %}  
</body>  
</html>

*templates/product/list.html*

{% extends 'layout/base.html' %}

{% block content %}

Bu sayfa ürün listeleme sayfasıdır.  
  
<table>  
 {% for product in products %}  
 <tr>  
 <td>Ürün Adı : {{ product.name | upper }}</td>  
 <td>Ürün Açıklaması : {{ product.content }}</td>  
 </tr>  
 {% endfor %}  
</table>  
{% endblock %}

*templates/product/detail.html*

{% extends 'layout/base.html' %}  
  
{% block title %}  
Ürün Detay Sayfası  
{% endblock %}  
  
{% block content %}  
<table>  
 <tr>  
 <td>Ürün adı : </td>  
 <td>{{ product.name }}</td>  
 </tr>  
  
 <tr>  
 <td>Ürün açıklaması : </td>  
 <td>{{ product.content }}</td>  
 </tr>  
</table>  
{% endblock %}

*templates/product/archive.html*

{% extends 'layout/base.html' %}  
  
{% block title %}  
Ürün Arşiv Sayfası  
{% endblock %}  
  
{% block content %}  
Bu sayfa {{ year }} / {{ month }} için ürün arşiv sayfasıdır.  
<br>  
  
  
{% if products %}  
 <table>  
 {% for product in products %}  
 <tr>  
 <td><a href="{% url 'product-detail' product.id %}">Ürün Adı : {{ product.name | upper }}</a></td>  
 <td>Ürün Açıklaması : {{ product.content }}</td>  
 </tr>  
 {% endfor %}  
 </table>  
{% else %}  
 Ürün bulunamadı  
{% endif %}  
  
2019 / 5. ay arşivleri için <a href="{% url 'product-archive' **2019 5** %}">tıklayın</a>  
<br>  
  
Ürün listesi için <a href="{% url 'product-list'%}">tıklayın</a>

{% endblock %}

{{ block.super }} ile üst templatenin içeriğine ulaşabiliriz.

Şimdi sistemimize boostrap entegre ederek şablonumuzu oluşturalım ve sayfalarımızı daha güzel görünür hale getirelim. Bootsrap dosyalarını indirmeden önce Djangun statik dosyaları nasıl sakladığı hakkında bilgi sahibi olmamız gerekmektedir.

Statik dosyalar jpg, pdf, txt, css, js gibi kod içermeyen dosyalardır. Settings.py den static dosyaların saklandığı dizin için ayarlamalar yapılabilir.

STATIC\_ROOT: Fiziki olarak static dosyalarımızın bulunduğu yerdir**. python manage.py collectstatic** komutu ile birlikte proje uygulamaların statik dosyaları ve STATICFILES\_DIRS’a eklenen klasörler altında bulunan statik dosyalar ana statik dizininde toplanır.

STATIC\_URL : STATIC\_ROOT içerisinde yer alan dosyalara ulaşabilmek için tanımlanan adrestir.

STATICFILES\_DIRS : Static olarak kullanılacak klasörler seçilir.

# Static files (CSS, JavaScript, Images)  
# https://docs.djangoproject.com/en/2.2/howto/static-files/  
  
STATIC\_URL = '/static/'  
STATIC\_ROOT = os.path.join(BASE\_DIR**,** 'staticfiles')

STATICFILES\_DIRS = [  
 os.path.join(BASE\_DIR**,** 'static')**,**]

Statik dosyalar Debug=True ve Debug=False olmasına göre farklı şekillerde çalışır. Uygulamayı geliştirirken Debug=True, uygulamayı canlıya aldığımızda yani yayınladığımızda False olması gerektiğini söylemiştik.

Debug=True ise runserver komutuyla birlikte otomatik olarak statik dosyalar yüklenir. Bu dosyalara ulaşabilmek için ana URL dosyamıza static url ve rootun tanıtılması gerekir. Aksi taktirde dosyalara ulaşılamaz.

if settings.DEBUG:  
 urlpatterns += static(settings.STATIC\_URL**,** document\_root=settings.STATIC\_ROOT)

Debug=False olduğunda ise statik dosyalar yüklenmez. Collecstatic komutu ile dosyalar STATIC\_ROOT klasörüne yüklenir. Bu dosyalar python aracılığı ile ulaşılmaz NGINX veya APACHE üzerinden servis edilir. Bunun için STATIC\_ROOT klasörü webserver configürasyonunda ayarlanması gerekir.

Statik dosyaların nasıl kullanılacağını öğrenmiş olduk. Şimdi bootstrap yükleyebiliriz. Bootstrap için [https://getbootstrap.com](https://getbootstrap.com/) adresinden Download gelelim ve Download Source tıklayalım.

Examples bölümündeki hazır temalardan Framework altındaki Jumbotronu temasını kullanalım. Bunun için kullanacağımız css ve js dosyalarını static altına css ve js klasörü açarak altına ekleyelim.

bootstrap-4.3.1\site\docs\4.3\examples\jumbotron\jumbotron.css  
bootstrap-4.3.1\dist\css\bootstrap.min.css  
bootstrap-4.3.1\dist\js\bootstrap.min.js  
bootstrap-4.3.1\site\docs\4.3\assets\js\vendor\ jquery-slim.min.js

bootstrap-4.3.1\site\docs\4.3\examples\jumbotron\index.html yi açalım ve kaynak kodları base.html içine yükleyelim. {{ site.time | date: "%Y" }} template tagları içerdiği için kaldıralım.

Şimdi static içine yüklediğimiz dosyaları temamıza ekleyelim.

Base.html en üstüne static dosyalarını kullanabilmek için

{% load static %}

Yazıyoruz. Head tagları içerisine css dosyalarımızı getirelim.

<link href="{% static 'css/bootstrap.min.css' %}" rel="stylesheet">  
<link href="{% static 'css/jumbotron.css' %}" rel="stylesheet">

Sayfanın en altına da js dosyalarımızı getirelim.

<script src="{% static 'js/jquery-slim.mim.js' %}"></script>  
<script src="{% static 'js/bootstrap.min.js' %}"></script>

Header ve footer dosyalarımızı ayarlayarak base.html yi biraz sadeleştirelim. Navbar kısmını headera footer kısmınını footer bölümüne alalım.

<footer class="container">  
 <p>**&copy;** Company 2017</p>  
</footer>

Collectstatic yapıp dosyaları staticfiles içine kopyalalım. Debug = True yu Debug = False yaparak kontrolleri sağlayalım. Değişimi görelim. Gördüğünüz gibi Production’da static dosyalar getirilmiyor. Staticfiles için NGINX ve APACHE ayarları yaparak getirebilirsiniz. Biz development tarafında devam edeceğiz.

Temamızı kullanacağımız alanlara göre düzenleyelim. (Türkçeleştirme)

Navbarımızda olduğumuz sayfaya göre menüden ilgili alanının aktif olmasını sağlayalım. Şu anda her sayfada Ana Sayfa seçili gibi. Urls.py lerde routelara verdiğimiz isimleri ilgili url için request.resolver\_match.url\_name ile alabiliriz. Eğer girdiğimiz sayfanın name menünün ismi ile aynı ise active olarak seçelim.

<li class="nav-item {% if request.resolver\_match.url\_name == 'product-list' %}active{% endif %}">  
 <a class="nav-link" href="{% url 'product-list' %}">Ürünler</a>  
</li>

Sayfamızı yenileyip deneyelim. Ürünlere ve Arşive tıklandığında seçildiğini görüyoruz. Burada dikkat edilmesi gerekli olan bir nokta template içerisinde request.resolver\_match.url\_name kullanabilmemiz için request’in settings.py de TEMPLATES alanında context\_processors olarak tanımlanmış olması gerekmektedir.

'context\_processors': [  
 'django.template.context\_processors.debug',  
 'django.template.context\_processors.request',  
 'django.contrib.auth.context\_processors.auth',  
 'django.contrib.messages.context\_processors.messages',  
],

Contenxt processor olarak tanımlanmış şeyleri biz bütün temamızdan direk ulaşabiliriz. İleriki derslerde eğer gerek duyarsak kendi context processorlarımızı yazacağız.