Arkadaşlar Merhaba,

Bu videomuzda formlar oluşturacağız ve formlardan gelen verileri veritabanına kayıt edeceğiz. Django formlar ile çalışmak için bize hazır imkanlar sunmaktadır.

Formlar için forms adında klasör oluşturalım ve \_\_init\_\_ dosyamızı ekleyelim. Bir adet iletişim formu hazırlayalım. Bunun için form klasörü içerisine contact.py dosyası oluşturalım. Formlar djangonun forms kütüphanesindeki Form sınıfından türerler. Kütüphanemizi import edelim ve form sınıfımızı oluşturalım. \_\_init\_\_ dosyasında tanıtalım.

from django import forms  
  
class Contact(forms.Form):  
 pass

Formda kullandığımız alanları oluşturalım form alanlarıda model alanlarına benzer Modellerdeki çoğu alanların çoğu kullanılabilir durumdadır. TextField yerine CharField kullanılır. CharField model gibi karakter sınırlaması yoktur. Tüm alanların listesine django dökümantasyonundan ulaşabilirsiniz.

İletişim formu yapcağımız için 3 tane alan oluşturalım. Form doldurulup submit yapıldığında isim, email ve içerik gönderilsin.

CharField ve EmailField için input oluşturur. Bizim açıklama alanımız için TextArea oluşturması gerekiyor bunun için widgetler kullanılır.

from django import forms  
  
  
class Contact(forms.Form):  
  
 name = forms.CharField(  
 label='İsim'**,** max\_length=**50,** min\_length=**5,** required=True**,** initial='isminizi giriniz...'**,** help\_text='burası isim alanıdır.'**,** error\_messages={  
 'required': 'Lütfen isim giriniz'**,** 'max\_length': 'En fazla 50 karakter girebilirsiniz'**,** 'min\_length': 'En az 50 karakter girebilirsiniz'  
 }**,** disabled=False**,** widget=forms.TextInput(  
 attrs={  
 'class': 'special'**,** 'size': '40'**,** 'title': 'İsminiz'**,** 'required': True  
 }  
 )  
 )  
 email = forms.EmailField(label='Email Adresi')  
 content = forms.CharField(widget=forms.Textarea)

Biz bu formu template dosyamızda getir dediğimizde otomatik olarak html şablonunu da oluşturmaktadır. Şimdi view dosyanızda formumuzu oluşturalım ve template dosyamıza render edelim. Bunun için views altında contact.py dosyası oluşturuyorum ve formumuzu import ediyorum. View olarak function based view kullanarak yapalım. contact\_form diye bir method yazıyorum ve içerisinde formu oluşturuyorum. \_\_init\_\_ methodumuzu tanıtalım.

from learning.forms import Contact  
from django.shortcuts import render  
  
  
def contact\_form(request):  
  
 form = Contact()  
 return render(request=request**,** template\_name='contact/form.html'**,** context={'form': form})

from .contact import contact\_form

learning/contact/ gelen istekleri methodumuza yönlendirelim.

path('contact/'**,** views.contact\_form**,** name='contact')**,**

template dosyamız için templates/contact içine form.html ouşturalım. Formlar otomatik olarak üç şekilde getirilebilir. tablo olarak, liste etiketleri ile, p etiketi ile. Hepsini örnekleyelim. Farklı görünümler için css dosyalarını güncelleyebilir veya formu manuel oluşturabiliriz.

Öncelikle formumuzu açalım. Action tanımlaması yapmadığımızı fark etmişsinizdir. Bu durumda istek bulunan mevcut sayfaya aktarılacaktır.csrf token yazalım aksi taktirde hata verir. İlk başladğımızda django cunun entegre olarak xss, sql injection vs için güvenlik özellikleri sunduğunu söylemiştik. Csrf izin vermek isterseniz view metodunda @csrf\_exempt dekoratörü kullanabilirsiniz.

{% extends 'layout/base.html' %}  
  
{% block title %}  
 İletişim Formu  
{% endblock %}  
  
{% block top %}  
 <div class="jumbotron">  
 <div class="container">  
  
 <h1 class="display-3">İletişim!</h1>  
 <p>Bu sayfadan bizimle iletişime geçebilirsiniz.</p>  
 </div>  
 </div>  
{% endblock %}  
  
{% block content %}

<form method="post" action="">  
 {% csrf\_token %}  
 {{ form.as\_ul }}  
</form>

{% endblock %}

learning/contact/ istek attığımızda formun sayfamıza geldiğini görüyoruz. Form submit edildiğinde yapılacak işlemlere bakalım.

from learning.forms import Contact  
from django.shortcuts import render  
from django.http import HttpResponseRedirect  
from django.core.mail import send\_mail  
  
  
def contact\_form(request):  
  
 if request.method == "POST":  
 form = Contact(request.POST)  
 if form.is\_valid():  
  
 subject = form.cleaned\_data['name']  
 message = form.cleaned\_data['content']  
 sender = form.cleaned\_data['email']  
  
 recipients = ['info@example.com']  
  
 send\_mail(subject**,** message**,** sender**,** recipients)  
   
 return HttpResponseRedirect('/')  
 else:  
 form = Contact()  
  
 return render(request=request**,** template\_name='contact/form.html'**,** context={'form': form})

Bir formun verilerini doğrulamak için is\_valid() yöntemini kullanmanız gerekir. Eğer form.is\_valid() True ise veriler işlenir ve sonraki işlem yapılır, return ile URL’ye yönlendirme ile son bulur. form.is\_valid() öğesi false ise form verilerinin hataları olduğu anlamına gelir. Doğrulanan cleaned\_data olarak geri döner.

Form alanlarının Method alanlarına benzerliğinden söz etmiştik Django bize ModelForm sınıfları ile direkt olarak Modellerden sınıf oluşturmaya izin verir.

Ürün ekleme formu:

*learning/forms/product.py*

from django import forms  
from learning.models import Product  
  
  
class ProductForm(forms.ModelForm):  
 class Meta:  
 model = Product  
 fields = '\_\_all\_\_'

*learning/forms/\_\_init\_\_.py*

from .product import ProductForm

*learning/views/learning.py*

from learning.forms import ProductForm  
from django.http import HttpResponseRedirect  
  
  
def product\_form(request):  
  
 if request.method == "POST":  
 form = ProductForm(request.POST)  
 if form.is\_valid():  
 p = form.save(commit=False)  
 p.save()  
 return HttpResponseRedirect('/learning/product/detail/%i/' % p.id)  
 else:  
 form = ProductForm()  
  
 return render(request=request**,** template\_name='product/add\_product.html'**,** context={'form': form})

*learning/views/\_\_init\_\_.py*

from .product import product\_form

*learning/urls.py*

path('product/add/'**,** views.product\_form**,** name='add-product')**,**

*templates/product/add\_product.html*

{% extends 'layout/base.html' %}  
  
{% block title %}  
Ürün Ekle  
{% endblock %}  
  
{% block top %}  
 <div class="jumbotron">  
 <div class="container">  
  
 <h1 class="display-3">Ürün Ekle!</h1>  
 <p>Ürün ekleme sayfasıdır. </p>  
 </div>  
 </div>  
{% endblock %}  
  
{% block content %}  
  
<form method="post" action="">  
{% csrf\_token %}  
{{ form.as\_ul }}  
<input type="submit" value="Gönder">  
</form>  
{% endblock %}

Widgetleri kullanmak için

*learning/forms/product.py*

class ProductForm(ModelForm):  
 class Meta:  
 model = Product  
 fields = '\_\_all\_\_'  
 widgets = {  
 'name': Textarea(attrs={'cols': **80,** 'rows': **20**})**,** }

Ürün düzenleme formu:

def product\_edit\_form(request**,** pk=None):  
  
 if request.method == "POST":  
 instance = Product.objects.get(pk=pk)  
 form = ProductForm(request.POST**,** instance=instance)  
 if form.is\_valid():  
 p = form.save(commit=False)  
 p.save()  
 return HttpResponseRedirect('/learning/product/detail/%i/' % pk)  
 else:  
 p = Product.objects.get(pk=pk)  
 form = ProductForm(instance=p)  
  
 return render(request=request**,** template\_name='product/add\_product.html'**,** context={'form': form})

*learning/views/\_\_init\_\_.py*

from .product import product\_edit\_form

path('product/edit/<int:pk>/'**,** views.product\_edit\_form**,** name='edit-product')**,**