Arkadaşlar merhaba,

Bu videomuzda ise modeller oluşturacağız ve bu modeller ile işlemler yapacağız. Model tanımlarken yaptığımız işlemleri açıklayacağım fakat çok fazla detaya girmeyeceğim. Bu noktada önemli olan genel yapıyı öğrenebilmek. İleriki konularda zaten detaylı olarak anlatacağız.

----------- model oluşturma ve migration ----------

Şimdi ilk modelimizi oluşturarak başlayalım. Modeller models.Model sınıfından türetilmektedir. Biliyoruzki bir sınıftan türetilen yeni sınıf diğer sınıfın özelliklerine sahip olur. Yani burda oluşturduğumuz Products sınıfı Model sınıfının özelliklerini miras alacaktır.

class Product(models.Model):

Product modelimize alanlarımızı ekleylim.

Ürünümüz için isim alanı ekleylim. En fazla 200 karakter alabilsin.

name = models.CharField(max\_length=**200**)

Açıklama alanı ekleyelim. Uzun yazılar olacağı için TextField seçelim.

content = models.TextField()

Birde eklenme tarihi ekleyelim. auto\_now\_add=True eklediğimizde. Ürün oluşturulduğunda otomatik olarak oluşturulma tarihini veritabanına eklemektedir.

created = models.DateTimeField(auto\_now\_add=True)

Model oluşturduğumuzda veritabanına tablo appadi\_modeladi olarak oluşturulur. Eğer kendi istediğimiz ismi vermek istersek Meta Sınıfı altında tanımlayabiliriz.

class Meta:  
 db\_table = 'urunler'

Veri yapımızı tanımlamış olduk. Migrations klasörünün veri yapılarının veri tabanına aktarılmasına yarayan yapılar olduğunu söylemiştik. Şimdi bizim veri yapımızdaki değişikliği bu klasöre taşımamız gerekli.

Bunun için;

**python manage.py makemigrations**

diye bir komut çalıştırıyoruz. Bu proje içerisindeki tüm modelleri tarıyor ve migration dosyaları oluşturuyor.

Konutu çalıştırdık ve model dosyalarında herhangi bir değişiklik bulamadı.

Aslında biz yeni bir model oluşturduk. Neden böyle bir şey yapmış olabilir ?

Daha önceki konularda belirttiğim gibi bir uygulamayı aktifleştirmek için settings.py dosyasında INSTALLED APPS’a eklememiz gerekmektedir.

**python manage.py makemigrations**

Evet gördüğünüz gibi migration dosyamızı oluşturdu. Şimdi bu migration dosyalarındaki değişiklikleri veritabanı ile eşleyelim.

Bunun için;

**python manage.py migrate**

konumutu çalıştırıyoruz.

Biz sadece Products veri yapısını oluşturduk fakat burada bir sürü migrate işlemi yapıldı. Django nun hazır oturum yönetimi ve yönetici paneli ile birlikte geldiğini söylemiştik. Bu tablolar bu modüller ve Django genel ayarları için oluşturulan tablolardır.Burada gördüğünüz gibi bizim tablomuzda oluşmuş. Veritabanından kontrol ettiğimizde de oluştuğunu görebiliyoruz.

Biz id diye bir alan oluşturmadık ama veri tabanına id alanı gelmiş. Eğer model alanları içerisinde hiçbir primary\_key alanı olmaz ise Django otomatik olarak id alanını primary\_key olarak tanımlamaktadır. İstersek biz bunu manuel oluşturabiliriz.

benzersiz\_key = models.AutoField(primary\_key=True)

------------------ yönetici paneli ------------------

Hazır yönetici panelinin olduğundan bahsetmiştik şimdi yönetici paneline giriş yapalım. URL adresimizin sonunda admin yazarak login sayfasını getirebiliriz.

Giriş için kullanıcı adı ve şifre oluşturmamız gerekmektedir.

Bunun için;

**python manage.py createsuperuser**

Komutunu giriyoruz ve kullanıcı adı ve email ve şifre belirliyoruz. Şifre validasyon kurallarının yerini settings.py de göstermiştim. Oraya yeni validayon kuralları girebilir veya istediklerini kaldırabilirsiniz.

Oluşturduğumuz kullanıcı adı ve şifre ile giriş yapalım. Gördüğünüz gibi hazır oturum yönetimi için Kullanıcı ve Gruplar oluşturulmuş. Bu bölümlerde kullanıcı ve grup oluşturabilir ve hem kullanıcı hemde grup bazında yetkilendirme yapabilirsiniz. Oturum Yönetimi ve Yetkilendirmeyi ileriki konularda anlatacağım.

Bizim eklediğimiz Learning uygulaması buraya gelmemiş, getirmek için admin.py den tanıtmamız gerekmektedir.

**admin.site.register(Product)**

Admin panelini kontrol ettiğimizde Learning uygulaması altında Products geldiğini görüyoruz.

Şimdi bir tane kayıt ekleyelim. created alanı auto add olduğu için gelmedi. Bu name ve content direk alan adlarından geldi bunları değiştirmek için Modelimizde verbose\_name vermemiz gerekmektedir.

Name -> ürün ismi

name = models.CharField(verbose\_name='Ürün İsmi'**,** max\_length=**200**)

Content -> ürün açıklaması yapalım

content = models.TextField(verbose\_name='Ürün Açıklaması')

ve şimdi yönetim panelinde bir tane kayıt ekleyelim.

PYÇARM’da değişiklikleri save etmeye gerek yoktur. Kendisi otomatik olarak save yapıp sanal sunucuyu yeniden çalıştırmaktadır.

Model adınıda Products olarak görünüyor model adını da değiştirebiliriz. Bu Meta sınıfında verbose\_name ile yapmalıyız. Verbose\_name\_plural ise çoğul halini yazmalıyız. Boş bırakırsa ingizlicedeki çoğul eki olan s takısını sonuna getirir.

class Meta:  
 db\_table = 'urunler'  
 verbose\_name = 'Ürün'  
 verbose\_name\_plural = 'Ürünler'

Listele kısmında id numarası ile geldiğini görüyoruz. Buraya ürün ismini getirebiliriz. Bunun için Product modeline bir sihirli method yazmamız gerekmektedir.

def \_\_str\_\_(self):  
 return self.name

Yani modellere methodlar yazabiliyoruz. Daha sonra farklı methodlarda yazacağız. Şimdilik method yazılabildiği bilmemiz yeterli.

Uygulama adımız Learnng olarak kaldı. Bunuda öğrenme olarak değiştielim.

Apps.py. Geliyoruz ve verbose\_name tanımlıyoruz.

class LearningConfig(AppConfig):  
 name = 'learning'  
 verbose\_name = 'Öğrenme'

Daha sonra bu ayarların aktifleşmesi için \_\_init\_\_.py dosyasında tanıtmamız lazım.

default\_app\_config = 'learning.apps.LearningConfig'

Default uygulama ayarları olarak learning içindeki apps çindeki LearningConfig dosyadığını atadığımızı söylüyoruz.

Ürünlere birde sahiplik bilgisi ekleyelim ürünü eklerken kullanıcı listesinden sahibi seçilebilsin. Bunun için modelimize author alanı ekleyip kullanıcı tablosu ile ilişkisini kurmalıyız.

author = models.ForeignKey(User**,** on\_delete=models.CASCADE)

İleriki konularda veritabanı ilişkileri hakkında detaylı bilgiler vereceğim.

Daha önce verdiğimiz tablo adınıda kaldıralım. Standart djangonun verdiği isimlendirme ile devam etsin.

class Meta:  
 #db\_table = 'urunler'

Değişikliklerin veritabanına yansıması için sırasıyla makemigration yapıp daha sonra migrate yapmalıyız.

**python manage.py makemigrations**

**python manage.py migrate**

Serverı çalıştırıp ürün eklemede gelip gelmediğini kontrol edelim.

Bu derste model oluşturma, modellerin temel ayarları, yönetici paneli ve özelleştirilmesini öğrendik. Bir sonraki derste model alanlarını öğreneceğiz.