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 zaman zaman modeller konusuna dönüş yapacağız.

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

Şimdi ilk modelimizi oluşturarak başlayalım. Modelimizin adı Product olsun. Bunun için model klasörünün altında bir adet learnings.py dosyası oluşturuyorum. 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):

Modelimizi \_\_init\_\_.py dosyasından tanıtalım.

from .learnings import Product

Şimdi serverımızı çalıştıralım. Gördüğünüz gibi hata ile karşılaşıldı. Çünkü Product modelimizi tamamlamadık. Bir class oluşturdunuz fakat sonra kullanmak istiyorsunuz diyelim. Bu gibi durumlarda hata vermemesi için pass ile geçebilirsiniz.

class Product(models.Model):

pass

Serverımızı tekrar çalıştırıp deneyelim. Evet serverımız sorunsuz çalıştı.

Product modelimize alanlarımızı ekleylim. Bu eklediğimiz alanlar veritabanımızda bulunan product tablosundaki sütunlar olacaktır. Alanlarımızı oluşturmaya başlamadan önce kullanabileceğimiz alan türleri ve özelliklerine bakalım.

Model alanları Field sınıfından türemektedir. Model alanları Field sınıfından türetilmektedir. Oluşturduğumuzun alanlar Field sınıfının özelliklerini taşıyacaktır. Field sınıfından alabildiği parametrelere bakalım.

Tüm alanlar bu parametreleri alabilmektedir. Fakat bazıları için sınırlandırma getirilmiş olabilir veya bazı alanlar bunlar haricinde farklı parametreler alabilir.

def \_\_init\_\_(self**,** verbose\_name=None**,** name=None**,** primary\_key=False**,** max\_length=None**,** unique=False**,** blank=False**,** null=False**,** db\_index=False**,** rel=None**,** default=NOT\_PROVIDED**,** editable=True**,** serialize=True**,** unique\_for\_date=None**,** unique\_for\_month=None**,** unique\_for\_year=None**,** choices=None**,** help\_text=''**,** db\_column=None**,** db\_tablespace=None**,** auto\_created=False**,** validators=()**,** error\_messages=None):

Örneğin IntegerField sınıfında max\_length kullanıldığında uyarı vermektedir. IntegerField için max\_length ekleyip nasıl bir uyarı verdiğine bakalım.

integer\_field = models.IntegerField(max\_length=**10**)

max\_length integer field ile kullanıldığında yok sayılmaktadır. İPUCU : Alandan max\_length silidiniz.

**WARNINGS:**

**learning.LearningFields.tamsayi: (fields.W122) 'max\_length' is ignored when used with IntegerField.**

**HINT: Remove 'max\_length' from field**

Bu özellikleri modelleri oluştururken anlatacağım. Uygulama üzerinde kullanmak teorik bilgi şeklinde öğrenmektense daha yararlı oluyor.

Alan türlerine başlamadan önce veritabanında alanların alabileceği min – max değerleri de öğrenelim. Small Integer -32.768 – 32.767 arası tam sayı alabilir. Integer -2.147.483.648 – 2.147.483.647 arası tam sayı alabilir. Big Integer ise yaklaşık -9 x 1018 ie 9 x 1018 arası tam sayı alır. Positive olanlar ise eksi değer alamazlar aldıkları değerler 0 dan başlar.

BigIntegerField zaten bütün değerleri alabiliyor. Neden hepsinde bunu kullanmıyoruz diye aklınıza soru gelebilir. SmallIntegerField her bir alan için 2 byte yer kaplarken BigIntegerField 8 byte yer kaplamaktadır. Gereksiz yere BigIntegerField kullanmak veritabanınıj büyükmesine neden olur. Veritabanlarımız büyüdükçe sorgu performansı da düşer. Alanın alabileceği maksimum değer hangi aralıkta kalıyorsa o alan türü kullanılmalıdır.

Kullanabileceğimiz alan türleri bunlar olup,

**SAYI ALAN TÜRLERİ :**

AutoField, BigAutoField, SmallIntegerField, IntegerField, BigIntegerField, PositiveIntegerField, PositiveSmallIntegerField, FloatField, DecimalField

**TEXT ALAN TÜRLERİ:**

CharField, TextField, EmailField, GenericIPAddressField, SlugField, URLField

**TARİH ALAN TÜRLERİ:**

DateField, DateTimeField, TimeField

**DOSYA ALAN TÜRLERİ:**

FileField, FilePathField, ImageField

**İLİŞKİSEL ALAN TÜRLERİ:**

ForeignKey, ManyToManyField, OneToOneField

**DİĞER ALAN TÜRLERİ :**

BinaryField, UUIDField, BooleanField

istersek kendimiz yeni alan türü ekleyebiliriz. Yine bu alan türlerini modeller ile uygulama yaparken öğreneceğiz.

Şimdi model alanlarımızı oluşturmaya başlayabiliriz.

Ürünümüz için isim alanı ekleylim. En fazla 200 karakter alabilsin. CharField255 karaktere kadar stringler için kullanılan alan tipidir. max\_length argümanı tanımlanması zorunludur. max\_length argümanı alanın alabileceği maksimum karakter sayısını belirler.

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

Açıklama alanı ekleyelim. Uzun yazılar olacağı için TextField seçelim.

content = models.TextField()

TextField ,Uzun stringler için kullanılan alan tipidir. max\_length argümanı seçilirse otomatik oluşturulan form için geçerli olur.

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

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

auto\_now=True argümanı ise save methodu çağırıldığında alanı güncel tarih ile değiştirir. auto\_now\_add, auto\_now ve default argümanı birlikte kullanılamaz.

Ürünün aktif/pasif olduğunu belirleyebilmek için bir adet alan ekleyelim. Alan türü olarak BooleanField seçebiliriz. BooleanField, True / False ve None değeri alabilir. Varsayılan olarak None seçilidir. default argümanına girilen değer eğer alanın değeri boş ise veritabanına eklenecek olan varsayılan değerdir.

active = models.BooleanField(default=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. Django bu alanı AutoField olarak oluşturmaktadır. AutoField’ler integer alan tipindedir. Alan adını değiştirmek istersek veya AutoField aldığı değerden fazla kaydımız olacağı için BigAutoField kullanmak istersek kendimiz oluşturabiliriz. Bu alanların değerleri otomatik olarak her kayıt eklenişinde artmaktadır. Bu alanlarda primary\_key argümanının True olarak seçilmesi zorunludur.

İstersek biz bunu manuel oluşturabiliriz.

benzersiz\_key = models.BigAutoField(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. verbose\_name alanın admin sayfasıi form gibi çıktılarda görüntülecek adıdır. Eğer girilmezse alanadi kullanılır.

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 Learning 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.