Arkadaşlar merhaba,

Artık uygulamamızı yazmaya başlayabiliriz.

Yazmaya başlamadan önce neler yapacağımızı belirlememiz lazım. Daha sonra veritabanı tasarımını yapmalı ve modellerimizi yazmalıyız. Daha sonra view ve templateleri hazırlayabiliriz.

* Bir firmamız olsun farklı sektörlerde faaliyet göstersin. Her faaliyet gösterdiği alan için ayrı distribütörlük oluşturalım ve bunlar sisteme ürün eklesinler.
* Bu distribütörlerin bayileri bu ürünleri sipariş edebilsin.
* Ürünleri listelemek ve sipariş verebilmek için bayi girişi yapılması zorunlu olsun
* Distribütör ve bayilere yetkili atanabilsin. Bu yetkililer şirket personeli ekleyebilsin ve izinlerini güncelleyebilsin.
* Ürünlerin markaları ve kategorileri olsun
* Sayfaların bazıları üye girişi yapılmadan görüntülenebilsin.
* Kullanıcı birden fazla bayide ve/veya distribütörde yetkili olabilsin.
* Distrübütör bazı işlemleri yönetici sayfasından, bazı işlemleri web sayfasından gerçekleştirilsin.
* Bayi işlemleri ise web sayfası üzerinden gerçekleştirilsin.
* Kullanıcı giriş yaptığında rol sayfası açılsın ve işlem yapmak istediği bayiyi seçsin. Daha sonra yapılan işlemler bu bayi için gerçekleşsin.
* Bayilerin distribütörlere sistem üzerinden ödeme yapsın ve distribütörlerle olan borç/alacak bakiyesini takip edebilsin.
* Distribütörler bayilerden gelen ödemeleri onaylasın. Bayilere olan borç / alacak bakitesini takip edebilsin.
* Bayi sipariş verdiğinde sipariş formu oluşturulsun. Distribütör ürünleri nakliye şirketine verdiğinde sipariş tamamlansın ve sipariş faturası hazırlansın.
* Bayiler kendi faturalarını sistemden kontrol edebilsin.
* Siparşler nakliyeye verilene kadar bayiler siparişlerini iptal edebilsin. Nakliyeye verildikten sonra sipariş iptal edilemesin.
* Distribitörler siparişleri her koşulda iptal edebilsin.
* Sitemizde firmamız ile ilgili bilgilerin yer aldığı statik sayfalar olsun.

Bir distribütör ve bayileri olduğunu düşünelim. Distribütör ürünlerini sisteme ekleyecek bayiler ise web arayüzü üzerinden sipariş verebilecekler. Birden fazla Distribütörler eklenebilecek şekilde sağlayalım yani bir distribütörün birden fazla bayisi bir bayininde birden fazla distribütörü olabilsin. Model ilişkilerini hatırlarsak distribütör ve bayiler arasında çoka çok ilişki olduğunu anladınız sanırım.

Projemize yeni bir uygulama ekleyelim ve modellerimizi oluşturmaya başlayalım. Uygulama adımız project olsun.

**python manage.py startapp project**

views ve modellerimiz için klasörlerimizi ve \_\_init\_\_.py dosyalarını oluşturalım. views.py ve models.py yi silelim.

Modellerimizi oluşturabiliriz. Distribütörler için Distributor ve Bayiler için Dealer modellrini oluşturalım. Company.py oluşturalım ve modellerimizi bu sayfa içerisinde oluşturalım. Modellerimiz Model sınıfından türetiliyordu.

from django.db import models

class Dealer(models.Model):  
 pass  
  
  
class Distributor(models.Model):  
 pass

\_\_init\_\_.py de modellerimizi tanıtalım.

from .company import Dealer**,** Distributor

Model alanlarımızı oluşturmaya başlayalım. Distrübütörler ve Bayiler ikiside şirket genel olarak aynı alanlara sahip olacaklar. Biz bunları ayrı ayrı oluşturabileceğimiz gibi, Company diye bir sınıf oluşturup ordan da türetebiliriz. Türettiğimizde Dealer ve Distributor, Company sınıfındaki alanlara sahip olacaktır. Bu sınıfların ortak özelliklerini Company sınıfında toplayalım.

from django.db import models  
from django.contrib.auth.models import User  
  
class Company(models.Model):  
 name = models.CharField(max\_length=**50**)  
 content = models.TextField()  
 slug = models.SlugField(unique=True)  
 author = models.ForeignKey(User**,** on\_delete=models.CASCADE)  
 created = models.DateTimeField(auto\_now\_add=True)  
 active = models.BooleanField(default=True)

class Meta:  
 abstract = True

class Dealer(Company):  
 pass  
  
class Distributor(Company):  
 pass

Daha önceki videolarımızda name alanından slug alanını otomatik olarak doldurmak için save() metodunu override yapmıştık. Bu modellerimizde de slug alanımız name alanından otomatik olarak doldurulsun. Daha önceki save() metodunu kopyalayalım ve buraya alalım.

Slugların unique lüğünü tespit etmek için Product modelimizde o sluga ait kayıt varmı diye bakmıştık. Fakat şu anda abstract sınıftayız ve unique alanı Dealer veya Distrubitor modelinde olabilir. Bunun için abstract sınıfta hangisi ile işlem yapıldığını tespit etmemiz lazım. Bunun için self.\_\_class\_\_ kullanırız.

def save(self**,** force\_insert=False**,** force\_update=False**,** using=None):  
 if not self.id:  
 self.slug = slugify(self.name)  
  
 for slug\_id in itertools.count(**1**):  
 if not self.\_\_class\_\_.objects.filter(slug=self.slug).exists():  
 break  
 self.slug = '%s-%d' % (self.slug**,** slug\_id)  
  
 super(self.\_\_class\_\_**,** self).save()

Şimdi Distributor ve Dealer arasında çoka çok ilişkimizi kullanalım. Çoka çok ilişkiler için many to many field kullanıyorduk. related\_name ve related\_query\_name ise ters ilişki kurmak için yapılıyordu. Eğer biz bu alanı abstract sınıfımızda kullansaydık hem Distributor hem de Dealer modelinde tanımlayacaktı ve sorun oluşacaktı.

class Distributor(Company):  
 dealers = models.ManyToManyField(Dealer**,** related\_name='distributors'**,** related\_query\_name='distributor')

Distribütör ve Delaer arasında ilişki kurduğumuzda veritabanında otomatik olarak project\_dealer\_distributor tablosu oluşur. Bu tabloya id, dealer\_id ve product\_id alanları eklenir. Biz bu tabloya başka alanlarda eklemek isteybiliriz. Örneğin ilişkinin kurulduğu tarih veya bayilik seviyesi gibi. Bu gibi durumlar için through argümanı kullanılır. Değer olarak ilişki kurulacak Model girilir.

class Distributor(Company):  
 dealers = models.ManyToManyField(Dealer**,** through=DistributorDealer**,** related\_name='distributors'**,** related\_query\_name='distributor')

class DistributorDealer(models.Model):  
 distributor = models.ForeignKey(Distributor**,** on\_delete=models.CASCADE)  
 dealer = models.ForeignKey(Dealer**,** on\_delete=models.CASCADE)  
 created = models.DateTimeField(auto\_now\_add=True)

Burada gördüğünüz gibi DistributorDealer ilişkili diğer modellerden sonra tanıdıldığı için Dealer ve Distributor modellerini bulamadı bu gibi durumlarda direk yol verebiliriz aksi taktirde NameError hatası ile karşılaşırız.

class DistributorDealer(models.Model):  
 distributor = models.ForeignKey('project.Distributor'**,** on\_delete=models.CASCADE)  
 dealer = models.ForeignKey('project.Dealer'**,** on\_delete=models.CASCADE)  
 created = models.DateTimeField(auto\_now\_add=True)

Bizim şu an için başka alan kullanmamıza gerek yok onun için throug argümanını kaldıralım ve modelimizi silelim.

Uygulamamızın settings.py girelim ve modellerimizi veritabanı ile eşleyelim. Tablolarımız oluşturuldu veritabanından kontrol edebiliriz. Delaer, distributor ve delaerdistributor tablolarımız oluşturulmuş.

**python manage.py makemigrations  
python manage.py migrate**

Distribütörler sisteme ürün eklemesi gerekmektedir. Yani bizim bir Ürün modelimiz olması gerekli. Ürünlerinde markaları olsun. Ayakkabıdan düşünürsek bir düstribütör birden fazla markada ürün satabilir. Bunun için Product ve Brand modellerimizi oluşturmak için product.py dosyamızı oluşturalım ve modellerimizi ekleyelim. Eklediğimiz modelleri \_\_init\_\_.py de tanıtalım.

from django.db import models  
  
class Brand(models.Model):  
 pass  
  
class Product(models.Model):  
 pass

from .company import Dealer**,** Distributor**,** DistributorDealer**,** Company  
from .product import Brand**,** Product

Model alanlarımızı oluşturmaya başlayabiliriz.

from django.db import models  
from .company import Distributor

class Brand(models.Model):  
 name = models.CharField(max\_length=**25**)

slug = models.SlugField(unique=True)  
 created = models.DateTimeField(auto\_now\_add=True)  
  
  
class Product(models.Molde):  
 name = models.CharField(max\_length=**50**)

slug = models.SlugField(unique=True)  
 content = models.TextField()  
 brand = models.ForeignKey(Brand**,** on\_delete=models.CASCADE)   
 distributor = models.ForeignKey(Distributor**,** on\_delete=models.CASCADE)  
 created = models.DateTimeField(auto\_now\_add=True)

updated = models.DateTimeField(auto\_now=True)   
 active = models.BooleanField(default=True)

Modellerimizin nerdeyse tamamında slug alanı kullanıyoruz ve slugın name alanından otomatik doldurulmasını sağlıyoruz. Brand ve Product için de aynısını yapmamız gerekli. Bunun yerine bir ade Mixin oluşturabilir ve tüm modellere slug için gerekli metodların ve alanların eklenmesini sağlayabiliriz.

Mixinler birden çok sınıfta ortak olarak kullanılan yapılar için bize yardımcı olur. Viewlerimizde kullandığımız ListView bir mixindir mesela. Biz Slug için oluşturacağımız mixini sadece bu app de değil tüm projemizde kullanabiliriz onun için ana dizine utils diye bir klasör açalım ve genel olarak tüm proje genelinde kullanacağımız nesnelerimizi buraya ekleyelim. mixins klasörünü açalım. Model için mixin oluşturacağımız için models.py dosyası oluşturalım. Dosyanın ismi önemli değil ben daha kolay ulaşmak için bu şekilde tercih ettim.

from django.db import models  
from django.template.defaultfilters import slugify  
import itertools  
  
  
class SlugMixin(models.Model):  
 name = models.CharField(max\_length=**255**)  
 slug = models.SlugField(unique=True)  
  
 class Meta:  
 abstract = True  
  
 def save(self**,** force\_insert=False**,** force\_update=False**,** using=None):  
 if not self.id:  
 self.slug = slugify(self.name)  
  
 for slug\_id in itertools.count(**1**):  
 if not self.\_\_class\_\_.objects.filter(slug=self.slug).exists():  
 break  
 self.slug = '%s-%d' % (self.slug**,** slug\_id)  
  
 super(SlugMixin,self).save()

Mixinimizi oluşturduk. Şimdi bunu kullanacağımız modellerde güncelleme işlemini yapalım. Company sınıfında kullandığımız mixin otomatik olarak bu sınıftan türeyen sınıflara da aktarılacaktır.

class Company(SlugMixin):

class Brand(SlugMixin):

class Product(SlugMixin):

Evet şimdi geldiğimiz duruma bakalım.

Bizim distribütör ve bayilerimiz var. Bunlar bir biri arasında çoka çok ilişkili. Distribütörün Ürünleri var. Bu ürünlerin Markaları var.

Bayilerimiz distrübütörün ürünlerinden sipariş edecekler. Sipariş edebilmesi için ürünleri stokta bulunmalıdır. Biz bu uygulamamızda stoğa giren malların log kayıtlarını tutmayacağız sadece bir alanda stoktaki miktarı takip edelim. Bunun için Product modeline stock\_count diye bir alan açalım. Sipariş yapılabilmesi için ürün fiyatları da olması gerekli bunuda price alanı olarak ekleyelim.

class Product(SlugMixin):  
 name = models.CharField(max\_length=**50**)  
 content = models.TextField()  
 brand = models.ForeignKey(Brand**,** on\_delete=models.CASCADE)  
 distributor = models.ForeignKey(Distributor**,** on\_delete=models.CASCADE)  
 created = models.DateTimeField(auto\_now\_add=True)  
 active = models.BooleanField(default=True)  
 price = models.FloatField()  
 stock\_count = models.PositiveIntegerField()

Siparişlerin tutulacağı bir Model oluşturmamız gerekli. Bir siparişte birden fazla ürün sipariş edilebilmeli bunun için bir tane Order modeli birtane de sipariş edilen ürünlerin tutulacağı OrderItem modeli oluşturalım.

Orders.py diye bir dosya oluşturalım ve Modelimizi burada oluşturalım. Modelimizi \_\_init\_\_ dosyamızda tanıtalım.

from django.db import models  
  
  
class Order(models.Model):  
 pass  
  
  
class OrderItem(models.Model):  
 pass

Order modelimizde sipariş yapan bayiyi, sipariş tarihini, sipariş güncelleme tarihini, sipariş iptal durumunu ve sipariş nakliye durumunu tutalım.

from django.db import models  
from .company import Dealer, Distributor  
from .product import Product  
  
  
class Order(models.Model):  
 dealer = models.ForeignKey(Dealer**,** on\_delete=models.CASCADE)  
 order\_transfered = models.BooleanField(default=False)  
 order\_cancelled = models.BooleanField(default=False)  
 created = models.DateTimeField(auto\_now\_add=True)  
 updated = models.DateTimeField(auto\_now=True)  
  
  
class OrderItem(models.Model):  
 order\_id = models.ForeignKey(Order**,** on\_delete=models.CASCADE)  
 product\_id = models.ForeignKey(Product**,** on\_delete=models.CASCADE)  
 item\_count = models.PositiveIntegerField()  
 item\_price = models.FloatField()

Son olarak distribütör ve bayi arasında gerçekleşen para transferlerinin olacağı bir tablo oluşturalım. Bu tablodan bayilerin ne kadar ödeme yaptığını hesaplayacağız. Siparişlerden tutan ödemeleri buradan çıkardığımızda borç / alacak tutarı hesaplanacak. Bunun için Payment adında bir model oluşturalım ve \_\_init\_\_ tanıtalım.

choices argümanı alana daha önceden tanmlanmış seçimli öğeler atamak için kullanılır. Bu argümanı girildiğinde ilgili alan için form tarafında seçim kutusu (selectbox) görüntülenir. Şimdi model tarafında nasıl tanımladığımıza bakalım. .

Payment modülünde Ödeme Kabülü alanını choices olarak ayarlaylım ve ödeme kabülü olarak KABUL EDİLDİ, RED EDİLDİ, İPTAL EDİLDİ, BEKLEMEDE kullanılabilsin. Öncelikle seçeneklerimizi oluşturalım. Seçenekler tupple olarak oluşturulur. Seçim kutusunu düşünürseniz her bir seçinin key ve değeri oluyor. Bizde tupple ile bunları tanımlıyoruz. Kabül tipleri değerler, baş harfleri de keyimiz oluyor. Veritabanına keyleri kayıt edeceğiz.

Ödeme Kabülü seçeneklerini oluşturduk. Şimdi bunu bir alana kayıt etmemiz lazım. Keylerimiz text olduğu için CharField alanını kullanabiliriz. Tüm keyler bir harften oluştuğu için max\_length olarak 1 verelim ve seçim yapılmaz ise default olarak ‘B’ yani BEKLEMEDE seçilmiş olsun.

class Payment(models.Model):  
  
 PAYMENT\_CHOICES = (  
 ('K'**,** 'KABUL EDİLDİ')**,** ('R'**,** 'RED EDİLDİ')**,** ('I'**,** 'İPTAL EDİLDİ')**,** ('B'**,** 'BEKLEMEDE')  
 )  
  
 dealer = models.ForeignKey(Dealer**,** on\_delete=models.CASCADE**,** verbose\_name='Bayi')  
 distributor = models.ForeignKey(Distributor**,** on\_delete=models.CASCADE**,** verbose\_name='Distribütör')  
 amount = models.FloatField(verbose\_name='Miktar')  
 payment\_accepted = models.CharField(max\_length=**1,** choices=PAYMENT\_CHOICES**,** default='B')  
  
 class Meta:  
 verbose\_name = 'Ödeme'  
 verbose\_name\_plural = 'Ödemeler'  
  
 def \_\_str\_\_(self):  
 return self.dealer.name

Ürünleri kategorize etmek için kategori modelimizi oluşturalım.

class Category(SlugMixin):  
 name = models.CharField(max\_length=**50,** verbose\_name='Ad')  
 products = models.ManyToManyField(Product**,** related\_name='products'**,** related\_query\_name='product'**,** blank=True)  
  
 class Meta:  
 verbose\_name = 'Kategori'  
 verbose\_name\_plural = 'Kategoriler'  
  
 def \_\_str\_\_(self):  
 return self.name

Yazdığımız modelleri veritabanı ile eşleyelim.

**python manage.py makemigrations  
python manage.py migrate**

Modellerimizi tanımlamış olduk. View ve Templateler oluşturulurken Modellerimizde şu an için düşünemediğimiz yerlerin güncellenmesini yaparız. Bir sonraki videomuzda yönetim panelini yapılandırırmaya başlayabiliriz.