------------------ DERS1 : KURS BİLGİSİ ------------------

Merhaba,

Ben Yunus Emre Çevik,

Eğitimin bu bölümünde sizlere JENGO hakkında bilgiler aktaracağım.

Eğitimi bitirdiğimizde umuyorumki sizlerde istediklerinizi hayata geçirebilecek web tabanlı uygulamalar yazar duruma geleceksiniz.

Bu Django eğitiminden en iyi seviyede yararlanmak için;

Orta düzeyde; Pyton, veritabanları ve nesne yönelimli programlama

Temel düzeyde html / css hakkında bilgi sahibi olmanızda fayda vardır.

Tecrübelerimden edindiğim kadarıyla video ile birlikte aynı anda koda dökmeye çalışmak teorik bilginin tam öğrenilmesine engel oluyor.Sizlere tavsiyem öncelikle videoyu sonuna kadar izleyin.. Daha sonra videoyu kapatıp kursta öğrendiklerinizi kodlara dökmeye çalışın.Takıldığınız noktalarda videoyu ileri – geri sararak hatırlamanıza yardımcı olabilirsiniz.

Takıldığınız ve anlamadığınız yerleri bizlere sorma konusunda lütfen tereddüt etmeyin. Günlük olarak sizden gelen sorular kontrol edilmekte ve 24 saat içinde yanıtlamaya özen gösterilmektedir. Bununla birlikte takıldığınız yerleri google da arayıp kendinizin çözmesininde öğrenmenize yardımcı olacağını unutmayın. Sorunları çözmek insana gerçekten öğrenme için motivasyon ve kendine güven sağlamaktadır.

Kurulumlar kullanılan sistemlere ve bilgisayarda kurulu olan uygulamalara göre farklı hatalara sebep olabiliyor. Bu yüzden kurulumda başarısız olduğunuzda öncelikle hata kodunu Google da aramanızı ve kendiniz çözemeye çalışmanızı rica ediyorum. Her konuda biz elimizden geldiğince yardımcı olmaya çalışırız fakat kurumlardaki hataların bizim tarafımızdan çözüme ulaştırılabilmesi gerçekten zor oluyor.

Bu eğitimi tamamladığımızda neler öğreneceğiz ve eğitim boyunca nasıl bir metodoloji izleyeceğize değinelim.

JENGO çok kapsamlı bir framework. Biz bu framework içerisindeki işimize en çok yarayacak bilgileri öğreneceğiz. İşimize yarayacak derken detaya girmeyecek, yüzelsel bilgiler verecek diye düşünnmeyin

emin olun udemy üzerindeki en kapsamlı Django eğitimlerinden biri olacaktır. Öğrendiğimiz bu teorik bilgileri paralel olararak pratiğe dökecek uygulama geliştireceğiz.

Önceki konuları eğer takip etmişseniz hatırlayacağınız gibi PAY KÜTİ FAYF ile stok takip programı

hazırlanmıştı. Biz bu kursumuzda uygulama olarak bayilerimizin ulaşabilmesi için web arayüzü yapacağız.

Yani distribütör için masaüstü programı bayirler için ise web arayüzü hazırlanmış olacak. Dersler eş zamanlı olarak hazırlandığı için veritabanı alanları bir biri ile uyumlu olmayacak fakat bayiler için nasıl bir web arayüzü hazırlanabilir öğrenmiş olacağız.

------------------ DERS2 : DJANGO NEDİR ------------------

Django nedir?

Django, python dili ile yazılmış MVT ( model, view, template) mimari yapısına sahip bir web frameworküdür.

Burada öncelikle frameworkü konusuna biraz değinmek gerekiyor.Framework, yazılım geliştiriciler için hazırlanan, en çok kullanılan kütüphaneleri içerisinde bulunduran yapılardır. Bu yapılar daha sonra farklı kütüphaneleri de sisteme entegre etmenizi sağlar.

Django PEKICIZ org adresinden kullanmak istediğiniz paketleri bulabilir ve frameworkümüze dahil edebiliriz.

Nasıl dahil edileceğini ileriki konularda göreceğiz.

Bir fotoğraf galerisi oluşturacağımzı düşünelim.Fotoğraf yüklemek, fotoğrafı düzenlemek (boyutlandırma, kırpma, sağa-sola çevirme, renk değiştirme) vb. işlemlerimiz var. Bu işlemleri yapabilmek için her bir işleme özel methodlar yazmamız gerekmektedir.

Bu işlemler için diğer yazılımcıların daha önceden hazırlamış oldukları paketler mevcut. Bu paketleri yükleyerek bu işlemleri yapacak kod bloklarını tekrar yazmamıza gerek kalmıyor. İlgili methodu çağırarak bu işlemleri kolayca yapabiliyoruz.Bu sayede daha az kod yazarak daha iyi web uygulamaları geliştirebiliriz.

Burada daha iyiden kasıt görüntüsel olarak değil de, daha çok yazılımın;

* daha güvenlikli olması
* daha stabil çalışması
* kodların diğer yazılımcılar tarafından okunabilir olması anlamına gelmektedir.

Şöyle düşünelim. Biz bir program yazdık bitirdik. İşlerde iyi gidiyor, yazılımımız tuttu! Gözlerde $ işaretleri belirmeye başladı. Ama artık işlere yetişemez olduk. Bir yandan yazılıma devam etmemiz gerekiyor bir yandan pazarlama faaliyetleri çok zamanımızı alıyor. Yazılımı geliştirmek için yanımıza bir yardımcı almamız gerekiyor.

Kodlarımızı bilindik bir framework kullanmadan hele de kara düzen yazdıysak öncelikle yazılımcının

sistemi öğrenebilmesi için tüm kodları incelemesi gerekecek. Ama bir framework kullandığımızda yeni arkadaş zaten bizim mevcutta kullandığımız kütüphaneler ile daha önce çalıştığı için daha hızlı bir şekilde olaya entegre olabilecek. Frameworkü bilmiyorsa bile detaylı dökümantasyonlar, eğitimler, makaleler vb. sayesinde kısa sürede adepte olabilecektir.

Peki neden Django kullanmalıyız.

* En popüler Python web frameworkü. İşinizi kolaylaştıracak istemediğiniz kadar kütüphanesi mevcut
* Basit kurulum ve kullanım
* Python syntax açısından kolay bir dil. Okunabilir ve düzenli kod yazmaya zorluyor.
* Hazır kullanıcı sistemi ve admin paneli ile birlikte geliyor.
* Detaylı hata ayıklamaları yapılan hataların bulunmasını kolaylaştırıyor.
* Temel güvenlik problemlerinin (xss, csrf, sql injection) engellenmesini sağlıyor. (Cross site scripting, Cross site request forgery, Sql Injection

Sonraki videomuzda Django kurulumu gerçekleştirecek ve yavaş yavaş uygulamamızı yazmaya başlayacağız.

------------------ DERS3 : KURULUM VE PROJE OLUŞTURMA – KURULUM ------------------

Arkadaşlar merhaba,

Bu bölümde Django kurulumu gerçekleştireceğiz ve Django hakkında genel bilgiler vereceğim.

Ben geliştirme ortamı olarak bu eğitim boyunca PAYÇARM, veritabanı olarak ise teorik bilgileri anlatırken SQL Lite, uygulama geliştirirken MYSQL kullanacağım. Siz isterseniz farklı veritabanları da kullanabilirsiniz.

Django kurulumu için bilgisayarınızda python ve pip kurulu olması gerekmektedir. Daha önceki konularda python, pip, PAYÇARM nasıl kurulduğu anlatılmıştı. Şimdi Django kurulumuna başlayabiliriz.

Eğer PAYÇARM Professional Edition kullanıyorsanız direk Django projesi oluşturabilirsiniz. Ben sıfırdan sanal ortam ve Django kurulumunu göstereceğim.

Öncelikle terminali açıyoruz.

**mkdir project** (ile klasör oluşturuyoruz)

**cd project** (oluşturduğumuz klasöre girelim)

**virtualenv stocksproject** (projemiz için yeni sanal ortam yaratalım)

**.\Scripts\activate** Sanal ortamı aktifleştirmek için ilgili klasöre gidiyoruz ve komutunu çalıştırıyoruz.

virtual envoriement ile sanal bir ortam oluşturur ve projelerinizi bu ortam içinde çalıştırmamıza yarar. Bu sanal ortam projeyi dış ortamdaki paket, sürüm değişiklikleri vs gibi problemlerden korur. Aktifleştirildiğinde gördüğünüz gibi başında sanal ortam ismi gelmektedir.

**pip freeze**

komutu ile kurulu paketleri kontrol edelim. Gördüğünüz gibi mevcutta paket bulunmamaktadır.

**pip install Django**

komutu ile Django kurulumu yapabiliriz.

**pip freze**

kurulu paketlere tekrar baktığımızda Django geldiğini görebiliriz.

Eski bir versiyon için

**pip install Django==2.2**

İki tane eşittir işaret ve sonrasına versiyon numarası ile de kurulum yapabiliriz.

**django-admin –version**

yazarak kurulan Django versiyonunu kontrol edebiliriz.

Şimdi Django projemizi oluşturabiliiz.

**django-admin startproject stocks**

Yazıyoruz. Buradaki stoks proje adımız oluyor.

------------------ DERS3 : KURULUM VE PROJE OLUŞTURMA – PROJE OLUŞTURMA ------------------

Evet şimdi oluşturduğumuz projeyi PAYÇARM içine alalım. Pycharm -> Open -> Django Proje mizi seçiyoruz. Burada dikkat edelim Proje klasörünü değil (Project deği) Django Projesini (stocks) seçmemiz gerekmektedir.

Şimdi PAYÇARM için virtualenv ayarlarını yapmamız gerekiyor.Settings -> Project: stocks -> Project Interpreter den ayar simgesine tıklıyoruz. Proje klasörünün içindeki Scripts içerisinden python.exe seçiyoruz. Gördüğünüz üzere kurulu paketlerimiz buraya geldi.Daha önce terminalden Django projemizi oluşturmuştuk. Proje yapısına baktığımızda içerisine bir adet stocks adında klasör ve manage.py dosyası eklendi.

manage.py dosyası terminal ekranından komutlarımızı çalıştırmaya yarayacaktır. Bu dosya ile herhangi bir işlem yapmayacağız.

\_\_init\_\_ py dosyası bulunduğu klasörün bir PAYTON modülü olduğunu göstermektedir.

settings.py Django ayarlarının bulunduğu dosyadır. Bunun içerisinden genel Django ayarlarını yapıyoruz.

urls.py dosyasından browserden istek yapıldığında hangin kodun çalıştırılması gerekiyorsa oraya yönlendirme yapıyoruz.

wsgi.py – sunucumuzun ayarlarının yapıldığı dosyadır.

Şimdi web sunucumuz çalışıyormu kontrol edelim. Bunun için manage.py ile runserver komutunu çalıştırıyoruz.Django otomatik olarak kendini 8000 portunda çalıştırıyor.

**python manage.py runserver**

ctrl + c yaparak sunucuyu durdurabiliriz.

Eğer 8000 portunda başka bir şey çalıştırıyorsak portu elle vererek başka portta da çalıştırabiliriz.

**python manage.py runserver 8081**

Açılan karşılama sayfası gördüğünüz gibi İngilizce. Biraz önce Django ayarlarını settings.py den ayarlayabildiğimizi söylemiştik.

------------------ DERS3 : KURULUM VE PROJE OLUŞTURMA – AYARLAR ------------------

Şimdi settings.py dosyasını biraz inceleyelim.

Burada **DEBUG = True** görüyoruz.

Bu sistem hataların ekrana basılmasını sağlamaktadır. Bu sayede hatalarımızı görebilir ve gerekli yerleri düzenleyebiliriz. Bu development yani geliştirme süreci bittiğinde False olarak değiştirilmelidir. Aksi taktirde güvenlik açığı oluşturacaktır.

**INSTALLED\_APPS** da hali hazırda kurulan uygulamalar gösterilmektedir. Biz paket indirdiğimizde veya uygulama yazdığımızda

Aktifleştirmek için bu alana eklememiz gerekmektedir.

**MIDDLEWARE**, ara katman anlamına gelmektedir. Ne işe yarar dersek gelen HTTP isteklerini filtrelemeye yarar. Örnek vermek gerekirse;

Tarayıcıdan bir url girmeye çalıştığınızda önce Middleware devreye girer kullanıcının giriş yapıp yapmadığına bakar. Eğer kullanıcı girişi başarısızsa giriş sayfasına yönlendirir, başarılı ise devam etmesine izin verir.

Kullanıcı bir form gönderdi. Formdaki alanları kontrol eder. Formdan gelen güvenlik ihlali sağlayacak veri varsa onları filtreler ve kalan veriyi işlemesi için bir sonraki aşamaya gönderir.

**ROOT\_URLCONF**  Django ya istek geldiğinde url maping için hang dosyanın kullanılacağını ayarlamaya yarar.

**TEMPLATES** : Template motoru ayar değişikliklerinin yapıldığı bölüm

**DATABASES** : Veritabanı ayar değişikliklerinin yapıldığı bölüm

**AUTH\_PASSWORD\_VALIDATORS** : Kullanılacak kullanıcı şifreleri için gerekli olan validasyon ayarları

**LANGUAGE\_CODE**: Django dil ayarı tr yaparak türkçeleştirebiliriz.

**TIME\_ZONE** : Zaman ayarı. Europe/Istanbul yaparak türkiye saatini ayarlayabiliriz.

**USE\_I18N, USE\_L10N, USE\_TZ** : Localizasyon ayarlarıdır.

**STATIC\_URL**: statik dosyaların konunlandıracağı yeri gösterir.

Bu videomuzda Django kurulumunu gerçekleştirdik ve genel proje dosyalarını inceledik.

Bir sonraki videomuzda uygulama kavramı ve uygulamanın nasıl oluşturulacağını öğreneceğiz.

------------------ DERS4 : UYGULAMA OLUŞTURMA VE TEMEL BİLGİLER ------------------

Merhaba arkadaşlar,

Bir önceki videomuzda Django kurulumunu gerçekleştirdik ve genel proje dosyalarını inceledik.

Bu videomuzda uygulama kavramı ve uygulamanın nasıl oluşturulacağını öğreneceğiz.

Öncelikle bir tane learning adında uygulama oluşturalım.

**python manage.py startapp learning**

Uygulama oluşturduğumuzda Django otomatik olarak bazı klasör ve dosyaları oluşturmaktadır. Şimdi bu dosyalar ne işe yaradığına kısaca bakalım. İleriki konularda ayrıntılarına gireceğiz. Şimdilik ne işe yaradıklarını bilmemiz yeterli.

**\_\_init\_\_.py :** dosyası bulunduğu klasörün bir PAYTON modülü olduğunu göstermektedir.

**admin.py :** uygulama için django admin panelini özelleştirmeye yarayan dosyadır.

**apps.py :** uygulamanın configürasyon ayarları için kullanılan dosyadır.

**migrations/ :** models.py de yer alan veri yapılarımızın veritabanına taşınmasını sağlayan dosyadır.

**models.py :** veri yapılarımızın tanımlandığı dosyadır. Veritabanında kullanmak istediğimiz veri yapılarını bu dosyalar içerisinde tutarız.

**tests.py :** birim testlerinin yazıldığı dosyadır. Testler kodumuzun çalıştığını kontrol ederler. Bu sayede her değişiklikte manuel test yapmamıza gerek kalmaz.

**views.py :** Tarayıcıdan bir adrese girildiğinde projenin urls.py dosyasından bu dosyanın içerisindeki methodlara istek gönderilir. Bu methodların içerisindeki kodlar çalışarak tarayıcıya bir çıktı response edilir. Daha sonra tarayıcı o çıktıyı ekrana basar.

Dosyalar ile ilgili temel bilgiler bu şekilde olup, uygulama aşamasına geçildiğinde daha iyi anlaşılacaktır.

Bu dosyaları ve klasörleri kullanmak zorunda değilsiniz. Ben büyük projeler için daha kolay okunabilirlik olması nedeniyle view ve modeller için ayrı klasörler kullanıyorum. İki kullanım içinde örnekler yapacağım. Öncelikle klasörler kullanarak oluşturulan yapıyı inceleyelim.

Eğer uygulamalarınız büyükse views.py ve models.py dosyalarının içerisinde çok fazla kod oluşmakta

bu da dosyada yapılan düzenlemeleri ve okunabilirliği zorlaştırmaktadır.

Uygulamada 10 tane model ve her modelinde yaklaşık 30 satır olduğunu düşünürsek models.py dosyası

300 satırdan oluşacaktır. Bu da dosya içerisinde istediğimiz modeli bulmayı ve düzenlemeyi zorlaştıracaktır.

Ben bunun yerine models ve views klasör halinde tutup içerisine her model ve view için dosya oluşturuyorum. Models/ klasörüne baktığımızda product ve product\_detail için iki ayrı dosya oluşturdum ve içerisinde ilgili veri yapılarımı ekledim.

Bu videomuzda uygulama oluşturduk ve uygulama dosyalarını inceledik.

Bir sonraki videomuzda nasıl model oluşturacağımızı öğreneceğiz ve model özelliklerini inceleyeceğiz.

----------- DERS5 : MODEL KAVRAMI VE TEMEL BİLGİLER – MODEL OLUŞTURMA VE MIGRATION ----------

Arkadaşlar merhaba,

Bir önceki videomuzda uygulama oluşturduk ve uygulama dosyalarını inceledik.

Bu videomuzda ise modeller oluşturacağız ve bu modeller ile işlemler yapacağız.

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)

------------------ DERS5 : MODEL KAVRAMI VE TEMEL BİLGİLER – YÖNETİCİ PANELİ ------------------

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 düzenlenmesini öğrendik.

Bir sonraki dersimizde django shell ile veritabanına eklenen verileri nasıl çekeceğimizi öğreneceğiz.

------- DERS6 : DJANGO SHELL VE VERİTABANI İŞLEMLERİ – EKLEME, GÜNCELLEME, SİLME, LİSTELEME-------

Merhaba arkadaşlar,

Bir önceki konumuzda model kavramına giriş yaptık ve model oluşturup genel özeliklerini öğrendik. Şu anda temel bilgiler anlatıyorum. Konular ilerledikçe anlaşılması daha zor olacak onun için temel bilgileri lütfen tekrar edelim.

Bu videomuzda veritabanına nasıl veri ekleyeceğimizi, güncelleyeceğimizi ve listeleyebileceğimizi öğreneceğiz. İlk aşamada veritabanındaki verilere ulaşmak için django shell kullanacağız. Daha sonra benzer işlemleri views dosyalarımızda da yapacağız. Kullanacağımız metodlar aynı olacak.

Django shelll açmak için terminalde:

**python manage.py shell**

yazıp entere basıyoruz.

Öncelike veritabanı ilişki kurabilmek için modeli import etmemiz gerekmektedir. product ve user modelimizi shelle import edelim.

**from learning.models import Product**

**from django.contrib.auth.models import User**

------------------ veritabanına ekleme ve güncelleme ------------------

Veritabanına veri eklemek ve güncellemek için Model sınıfının save() metodunu kullanıyoruz.

Bir adet kullanıcı ekleyelim.

**user = User(username=’editor’, email=’editor@learning.udm’)**

Kullanıcı bilgilerini user nesnesine atamış olduk. Bunları veritabanına işlemek için save() methodunu kullanmalıyız.

**user.save()**

create() methodunu kullanarak da veritabanına ekleme yapabiliriz.

**User.objects.create(username=’personel’, email=’personel@learning.udm’)**

Veritabanından kontrol ettiğimizde eklenmiş olduğunu görüyoruz.

Kullanıcı ismini değiştirelim. Hala user nesnemizde kayıt ettiğimiz kullanıcı bilgileri durmaktadır.

**user.username = ‘yonetici’**

**user.save()**

Bir tanede ürün eklemesi yapalım. Ürünlerde bildiğiniz gibi User modeli ile ilişkikli author alanı bulunmaktadır.

**product = Product(name=’Shell Product’, content=’Shellden yeni ürün ekliyoruz’, author=user)**

Evet şimdi product değişkenine ürün nesnemizi atadık. save() yaparak veritabanına kayıt edebiliriz.

Veritabanındaki verileri güncellemenin diğer bir yolu da update() methodunu kullanmak. 1 numaralı kullanıcının email adresini güncelleyelim. update() methodu geriye güncellenen nesne sayısını döndürür.

**User.objects.filter(id=1).update(email=’update@learning.udm’)**

------------------ veritabanına veri çekme ------------------

Veritabanından veri çekmek için Model Manager (model yöneticisi) ile QuerySet yani sorgu seti oluştururuz. Her modelin en az bir adet Manager’ı vardır ve modelismi.objects olarak ulaşılır.

**Product.objects**

QuerySet nedir?

Querysetler veritabanındaki bir nesne kolleksiyonunu temsil eder. Mesela Ürünler.

QuerySetlerin sıralanabilir., filtrelenebilir. Bu işlemler için Manager metodlarını kullanırız. Şimdi bu metodları inceleyelim.

Tüm nesneleri listelemek için **all()** methodunu kullanırız. Veritanbanındaki kullanıcıları users değişkenine atadık. Şimdi bunları ekrana bastıralım.

**users = User.objects.all()**

**for user in users:**

**print(user)**

Sorgu setinin ilk nesnesini getirmek için **first()** methodunu kullanırız. İlk nesnenin bilgilerini user değişkenine atayalım ve daha sonra elemanlarına ulaşalım.

**user = User.objects.first()**

**user.username**

**user.is\_staff**

Sorgu setinin son nesnesini getirmek için **last()** methodunu kullanırız.

**User.objects.last()**

Eğer kaydı nasıl çekeceğimizi biliyorsak **get()** kullanırız. Sadece bir tane kayıt gelmesi gerekmektedir.

Eğer birden fazla kayıt gelirse [**MultipleObjectsReturned**](https://docs.djangoproject.com/en/2.1/ref/exceptions/#django.core.exceptions.MultipleObjectsReturned) hatası döner.

Eğer kayıt gelmez ise [**DoesNotExist**](https://docs.djangoproject.com/en/2.1/ref/models/instances/#django.db.models.Model.DoesNotExist) hatası verir. Bunun için genellikle unique (benzersiz) alanlar için veya primary\_key ile birlikte kullanılır.

**User.objects.get(id=1)**

Sorgu setindeki kayıt tarihi en son olan nesneyi getirmek için latest() methodu kullanılır

**User.objects.latest(date\_joined)**

Sorgu setindeki kayıt tarihi en önce olan nesneyi getirmek için earliest() methodu kullanılır

**User.objects.earliest(date\_joined)**

Sorgu setinde kayıt olup olmadığını tespit etmek için exists() methodu kullanılır. Eğer kayıt varsa True yoksa False döner.

**User.objects.filter(id=1).exists()**

------------------ veritabanına veriyi çek, eğer veri yoksa ekle ------------------

**get\_or\_create()** metodu ile veritabanından bir veriyi çekmeye çalıştığımızda eğer varsa çekeriz yoksa çekmeye çalıştığımız veriyi veri tabanına ekletebiliriz. get\_or\_create() geriye tupple olarak döner.

obj olarak eklenen veya çekilen nesne bilgisini döndürür

created olarak da eğer eklendiyse True, çekildiyse False döndürür.

defaults kısmında eklenecek olan diğer alanları belirtiriz.

**obj, created = User.objects.get\_or\_create(**

**username='emre',**

**defaults={'email': ‘emre@learning.udm’},**

**)**

------------------ veritabanındaki veriyi güncelle, eğer veri yoksa ekle ------------------

**update\_or\_create()** methodu ile veritabanından bir veriyi güncellemeye çalıştığımızda eğer veri varsa güncellenir yoksa güncellemeye çalıştığımız veri veritabanına ekletebiliriz. update\_or\_create() geriye tupple olarak döner.

obj olarak eklenen veya çekilen nesne bilgisini döndürür

created olarak da eğer eklendiyse True, güncellendiyse False döndürür.

defaults kısmında eklenecek olan diğer alanları belirtiriz.

**obj, created = User.objects.update\_or\_create(**

**username ='Emre',**

**defaults={'email': ‘emre@learning.udm’},**

**)**

------------------ veritabanından toplu veri çekme ------------------

in\_bulk() methodu ile veritabanındaki kayıtları primary key veya benzersiz alan ile eşleştirerek çekebiliriz. dict olarak döner.

User tablosundaki tüm kayıtları getirir.

**User.objects.in\_bulk()**

User tablosundaki primary keyi 1 ve 2 olan kayıtlar

**User.objects.in\_bulk([1,2])**

Blog tablosunda slug alanı filtered\_word olan kayıtları getirir.

**Blog.objects.in\_bulk(['filtered\_word'], field\_name='slug')**

------------------ veritabanına toplu ekleme ------------------

bulk\_create(obj) methodu ile verilen nesne listesi bir tane sorgu çalıştırılarak veritabanına eklenir.

**User.objects.bulk\_create([**

**User(username='Emre'),**

**User(username ='Emre'),**

**], 1)**

------------------ veritabanında toplu güncelleme ------------------

bulk\_update() methodu ile verilen nesne listesi bir tane sorgu çalıştırılarak veritabanına eklenir. Veritabanından User modelindeki tüm nesneleri çekelim. Daha sonra 0 ve 1. Elemanlarının is\_staff bilgisini değiştirelim ve is\_staff bilgisini toplu olarak güncelleyelim.

**objects = User.objects.in\_bulk()**

**objects[0].is\_staff = 1**

**objects[1].is\_staff = 1**

**User.objects.bulk\_update(update.values(), ['is\_staff'])**

------------------ veritabanından veri silme ------------------

Veritabanından veri silmek için delete() methodunu kullanırız. Bu kod veritabanındaki primary\_key 2 olan kaydı siler ve geriye silinen nesne sayısını döndürür.

**user = User.objects.get(pk=2)**

**user.delete()**

Tüm ürünleri silmek için ise bu şekilde bir kod çalıştırabiliriz. Fakat veritabanına yeniden veri eklememiz gerekeceği için çalıştırmıyorum. Siz deneyebilirsiniz.

**products = Product.objects.all()**

**products.delete()**

------------------ veritabanından verileri sayma ------------------

count() methodu sorgu setindeki nesne sayısını döndürür.

**User.objects.count()**

veritabanındaki toplam kullanıcı sayısı

**User.objects.filter(is\_staff=1).count()**

veritabanındaki is\_staff=1 olan kullanıcı sayısı

Eğer QuerySet oluşturulmuşsa count() methodunun çalıştırması ekstra performans kaybında neden olur böyle durumlarda len() kullanılması daha uygundur.

**users = User.objects.all()**

**user.count()** # yanlış kullanım

**len(users)** # doğru kullanım

Bu videomuzda veritabanına nasıl veri ekleyeceğimizi, güncelleyeceğimizi ve listeleyebileceğimizi öğrendik.

Bir sonraki videomuzda ise sorgu setlerindeki kayıtların nasıl filtreleneceğini öğreneceğiz.

------------------ DERS6 : DJANGO SHELL VE VERİTABANI İŞLEMLERİ – FİLTRELEME ------------------

Merhaba arkadaşlar,

Bir önceki videomuzda veritabanına nasıl veri ekleyeceğimizi, güncelleyeceğimizi ve listeleyebileceğimizi öğrendik.

Bu videomuzda ise sorgu setlerindeki kayıtların nasıl filtreleneceğini öğreneceğiz.

Tüm kullanıcıları değilde bazı filtrelerden geçen kullanıcıları listelemek için filter() ve exclude() kullanılır.

filter içerisinde verilen parametrelere ile eşleşen kayıtları getirir. Örneğimizde veritabanında is\_staff bir olan kayıtlar sorgu setine eklenecektir.

**User.objects.filter(is\_staff=1)**

Birden fazla parametrede girebiliriz. Parametreler birbirine AND ile bağlanır. sql sorugusuna tercüme edersek WHERE is\_staff=1 AND active=1

**User.objects.filter(is\_staff=1, active=1)**

exclude ise filter tam tersidir. verilen paratmere ile eşleşmeyen kayıtlar getirilir. Örneğimizde veritabanında is\_staff bir olmayan ve is\_superuser=1 olmayan kayıtlar sorgu setine eklenecektir.

**User.objects.exclude(is\_staff=1, is\_superuser=1)**

filter ve exclude zincirleme olarak kullanıma uygundur. Örneğimizde aktif alanı 1 olan is\_staff alanı 1 olmayan ve is\_superuser alanı 0 olan kayıtlar gelecektir.

**User.objects.filter(active=1).exlude(is\_staff=1).filter(is\_superuser=0)**

İki exludenin zincirleme kullanılmasında olay biraz farklı. Yukarıda bu örneği tek exclude içerisinde vermiştik ve is\_staff bir olmayan ve is\_superuser=1 olmayan kayıtları getirdiğini söylemiştik. Zincirleme kullandığımızda ise is\_staff bir olmayan veya is\_superuser=1 olmayan kayıtları getirir. Bu ve / veya ayrımı önemli.

**User.objects.exclude(is\_staff=1).exclude(is\_superuser=1)**

DJANGO bize filter(), exclude() ve get() methodları için alan araması imkanı sunar.

Biraz önce örneklerimizde is\_staff = 1 olan kayıtları getirmiştik. 1 den büyük olan kayıtları da getirebiliriz. Veya 1 içeren kayıtları. Bu İngilizce olarak field lookups diye geçmektedir.

Field lookupslar anahtar kelimenin yanına iki tane alt çizgi ile kullanılır. Kullanabileceğimiz fields lookuplara hızlı bir şekilde bakalım.

exact tam eşleşen kayıtları listeler

**User.objects.get(username\_\_exact = ‘emre’)** # where username=emre

**User.objects.get(username\_\_exact =None)** # where username is null

iexact tam eşleşen kayıtları listeler, büyük-küçük harfe duyarlı değildir

**User.objects.get(username\_\_iexact = ‘emre’)** # EmRe emre

Büyüktür ve küçüktür filtrelemeleri için lt ve gt kullanılır.

1 den küçük (less then) : **User.objects.filter(is\_staf\_\_lt = 1)**

1 den küçük veya eşit : **User.objects.filter(is\_staf\_\_lte = 1)**

1 den büyük (greater then) : **User.objects.filter(is\_staf\_\_gt = 1)**

1 den büyük veya eşit : **User.objects.filter(is\_staf\_\_gte = 1)**

sql tercümesi LIKE olan işlemler için (başlayan, biten, içeren) startswith, endswith, contains kullanılır

küçük-büyük harf duyarlı olmayacaksa başına i koyulur.

e ile başlayan :

**User.objects.filter(username\_\_startswith = ‘e’)** #sadece e

**User.objects.filter(username\_\_startswith = ‘e’)** # e ve E

e ile biten :

**User.objects.filter(username\_\_endsswith = ‘e’)** # sadece e

**User.objects.filter(username\_\_ endsswith = ‘e’)** # e ve E

e içeren :

**User.objects.filter(username\_\_contains = ‘e’)** # sadece e

**User.objects.filter(username\_\_ icontains = ‘e’)** # e ve E

range iki değer arasındaki kayıtları filtreler. sql tercümesi BETWEEN’dir. ID si 2 ve 5 arasındaki kullanıcıları listelemek için kullanırız.

**start = 2**

**end = 5**

**User.objects.filter(id\_\_range=(start,end))**

in bir soruda birden fazla değere sahip kayıtların listelenmesini sağlar. IN kullanımı bazı durumlarda performans açısından hayat kurtarınıcı olabilir. İlişkili tablolar ile çalışırken tek sorguda yapmak yerine birinci sorguda ilk tablodaki kayıtların ID lerini almak daha sonra bu ID leri ikinci tabloda filtre olarak kullanmak büyük verilerle çalışıyorsanız performans açısından çok önemlidir.

**User.objects.filter(id\_\_in=[1, 3, 4])**

isnull filtrelenen alanın boş olup olmasını sorgulamak için kullanılır

**User.objects.filter(email\_\_isnull=True)** # WHERE email IS NULL;

regex ve iregex düzenli ifadeler ile sorgu yapmayı sağlar. iregex de küçük – büyük harf duyarlılığı yoktur. veritabanına göre kullanılacak düzenli ifadelerde ufak nüsans farklılıkları vardır. Kullanıcı adında sadece harf içeren kayıtları çeken bir sorgu:

**User.objects.get(username\_\_regex= r'^[a-zA-Z]+$')**

date, datetime ve time alanları için yıl, çeyrek, ay, gün, hafta, saat bazında filtreleme yapılabilir. Bu field lookupsları ile birlikte lt, gt, lte ve gte kullanılabilir. Bunların kullanımları hep aynı olup birkaç tane örnek yapalım.

**User.objects.filter(date\_joined\_\_quarter=2) # yılın 2. çeyreğinde kayıt olan kullanıcılar**

**User.objects.filter(date\_joined \_\_year\_\_gte=2019) # 2019 yılından sonra kayıt olan kullanıcılar**

date: tarih (sadece datetime alanı için)

time: zaman (sadece datetime alanı için)

date ve datetime alanları için

year : yıl

month: ay

day: gün

week: haftay

week\_day: hafta günü (1 Pazar, 7 Cumartesi)

quarter: mevsim (1 ve 4 arası)

time ve datetime alanları için

hour: saat

minute: dakika

second: saniye

Bu videomuzda sorgu setlerindeki kayıtların nasıl filtreleneceğini öğrendik.

Bir sonraki videomuzda sorgu setlerini sıralama ve diğer methodları öğreneceğiz.

----- DERS6 : DJANGO SHELL VE VERİTABANI İŞLEMLERİ – SIRALAMA – DİĞER METHODLAR------

Merhaba arkadaşlar,

Bir önceki videomuzda sorgu setlerindeki kayıtların nasıl filtreleneceğimizi öğrendik.

Bu videomuzda ise sıralama ve diğer methodları öğreneceğiz.

Umarım derslerden sonra tekrar ediyor ve methodları öğreniyorsunuzdur. Bu methodların çoğunu ileriki konularda kullanacağız.Diğer konularda methodları kullanırkende ne işe yaradıklarına değineceğim fakat eksik bilgi ile yola devam edersek ileriki konularda sorun çekeriz. Lütfen öğrendiklerimiz ile bol pratik yapalım.

------------------ verileri sıralama ve limit ------------------

Sorgu setindeki verileri sıralamak için order\_by() methodunu kullanıyoruz.

Kullanıcı tablosundaki verileri kullanıcı adına göre artan sıralamak için order\_by method içinde parameter olarak username kullanmamız yeterli. Azalan sıralamada ise başına tire (eksi) işareti koyaraız.

**User.objects.order\_by('username’)**  # artan sıralama

**User.objects.order\_by('-username’)** # azalan sıralama

Parametre olarak ? girerek verileri rasgele sıralayabiliriz.

**User.objects.order\_by(‘?’)** # rasgele sıralama

İlişkili alana göre sıralama için parametre olarak öncelikle bizim tablomuzdaki alanı yazarız daha sonra iki tane alt çizgi ekler ilişki tablosundaki alan adını yazarız. User tablosundaki kullanıcı adına göre ürünleri sıralamasını söylüyoruz.

Bir önceki dersten hatırlarsınız fields lookupstada alttan iki çizgi ekliyorduk. Biz veri yapılarımızı tanımlarken alan adı olarak fields\_lookup da yer alan özellikleri aynı isimleri veremiyoruz. Django buna izin vermiyor.

İleriki konularda eğer unutmazsak model oluştururken bunu deneriz.

**Product.objects.order\_by(‘author\_\_username’)**

Eğer zincirleme olarak sıralama yaparsak zincirin en son halkasına göre sıralama yapılır. Diğerleri yok sayılır. Birden çok alana göre sıralama telk method içerisinde birden fazla parametre göndermeliyiz.

**User.objects.order\_by('username').order\_by('date\_joined’)** # sadece date\_joined göre sıralanır

**User.objects.order\_by('username’,’date\_joined’)**  # önce username sonra date\_joined göre sıralanır

Verileri tersten sıralamak için reverse() methodu kullanılır. Eğer veriler daha önce sıralanmamışsa hiçbir etkisi olmaz. Ürünler sorgu setini ters çevirelim.

**products = Product.objects.order\_by(‘name’)**

**products.reverse()**

Djangoda sorgu setlerinde indexing (indexleme), slicing (dilimleme) ve stepping (adımlama) desteklenir. Örneklerle inceleyelim Öncelikle sorgu setindeki 2. İndexte yer alan kaydı alalım. Unutmayalım index 0 dan başlamaktadır.

**Products.objects.all()[2]**

ilk 10 kaydı alabiliriz.

**Products.objects.all()[:10]**

6. ve 10. Arası kayıtları çekelim.

**Products.objects.all()[6:10]**

İlk 11. ve 20. arası kayıtlar arasından 2. Kayıtları alalım

**Products.objects.all()[10:20:2]**

Ürünleri isme göre sıralayıp 0. indexteki nesneyi getirelim.

**Product.objects.order\_by(‘name')[0]**

Bir önceki sorgu seti ile bu sorgu seti aynı sonucu getirir. Eğer kayıt bulunamazsa birincisi IndexError hatası ikincisi ise DoesNotExist hatası döndürür.

**Product.objects.order\_by(‘name')[0:1].get()**

------------------ diğer metodlar ------------------

distinct() metodu ile bir sorgu setindeki benzer kayıtları temizleyebiliriz.

**User.objects.distinct()**

Sorgu seti nesne olarak dönmektedir. values() methodu ile dict (sözlük) olarak döndürülmesi sağlanır.

**User.objects.values()**

Values methoduna parametre olarak alan isimleri girilerek istenilen alanlar filtrelenebilir.

**User.objects.values(‘username’, ‘email’)**

values\_list() methodu sorgu setinin tupple olarak döndürülmesini sağlar.

**User.objects.values\_list()**

values() methodu gibi values\_list() methodunda da istenilen alanlar filtrelenebilir. Flat=True kullanarak sadece değerlerin dönmesini sağlayabiliriz.

**User.objects.values\_list(‘username’, ‘email’)**

**User.objects.values\_list(‘id, flat=True)**

Spesifik bir alanın değerini alabiliriz. Örnek olarak 1. numaralı kullanıcının kullanıcı adını alabiliriz.

**User.objects.values\_list(‘username’, flat=True).get(pk=1)**

none() methodu boş bir queryset döndürmeye yarar

**User.objects.none()**

Birden çok veritabanı kullanıyorsak using() methodu ile istediğimiz veritabanından veri çekebiliriz. Önceki konularımızı hatırlarsan settings.py dosyasında veritabanı bilgilerimizi tutuyorduk. Orada default olarak belirlenen veritabanından veri çekmek için using() kullanımına gerek yoktur.

Farklı bir veri tabanından çekeceksek using() kullanırız. Mesela backup ismi ile yeni veri tabanı tanımladığımızı düşünelim.

**Blog.objects.using(‘backup’)**

Birincil veritabanımızdan using default kullanarak da veri çekebiliriz.

**Blog.objects.using(‘default’)**

**union()**

----------------------------------------------------------------------------------------------

Uses SQL’s UNION operator to combine the results of two or more QuerySets.

The UNION operator selects only distinct values by default. To allow duplicate values, use the all=True argument.

qs1 = Author.objects.values\_list('name)

qs2 = Entry.objects.values\_list('headline’)

qs1.union(qs2).order\_by('name’)

**intersection()**

----------------------------------------------------------------------------------------------

Uses SQL’s INTERSECT operator to return the shared elements of two or more QuerySets. For example:

qs1.intersection(qs2, qs3)

**difference()**

----------------------------------------------------------------------------------------------

Uses SQL’s EXCEPT operator to keep only elements present in the QuerySet but not in some other QuerySets.

qs1.difference(qs2, qs3)

**defer()**

----------------------------------------------------------------------------------------------

This is done by passing the names of the fields to not load to defer():  
Entry.objects.defer("headline", "body")

*# Defers both the body and headline fields.*

Entry.objects.defer("body").filter(rating=5).defer("headline")

Blog.objects.select\_related().defer("entry\_\_headline", "entry\_\_body")

If you want to clear the set of deferred fields, pass None as a parameter to defer():

*# Load all fields immediately.*

my\_queryset.defer(**None**)

**only()**

----------------------------------------------------------------------------------------------

Suppose you have a model with fields name, age and biography.

The following two querysets are the same, in terms of deferred fields:

Person.objects.defer("age", "biography")

Person.objects.only("name")

*# Final result is that everything except "headline" is deferred.*

Entry.objects.only("headline", "body").defer("body")

*# Final result loads headline and body immediately (only() replaces any*

*# existing set of fields).*

Entry.objects.defer("body").only("headline", "body")

dates() - **datetimes()**

----------------------------------------------------------------------------------------------

order, which defaults to 'ASC', should be either 'ASC' or 'DESC'. This specifies how to order the results.

Entry.objects.dates('pub\_date', 'year')   
Entry.objects.dates('pub\_date', 'day', order='DESC')

"year" returns a list of all distinct year values for the field.

"month" returns a list of all distinct year/month values for the field.

"week" returns a list of all distinct year/week values for the field. All dates will be a Monday.

"day" returns a list of all distinct year/month/day values for the field.

**extra()** methodu django query leri ile yapılamayacak çok karışık sorguların yapılması için eklenen bir methoddur. Django ekibi bu methodun kaldırılması için çalışmalar yürütmektedir. extra() methodu içerisine alınan parametreler sql injection güvenlik açığına karşı korumasızdır aldığınız verileri kontrol etmeniz gerekmektedir.

**raw()** methoduda extra() gibi karışık sorguların yapılması için eklenen bir başka method olup, bu da güvenlik açıklarına karşı korumasızdır.

Ben hiç bu methodları kullanma ihtiyacı duymadım. Bu eğitim dizisinde bu methodlar ile ilgili anlatım yapmayacağım. Eğer gerek olursa django dükümantasyonun bilgi alabilirsiniz.

**transaction da anlatalım**

----------------------------------------------------------------------------------------------

**select\_for\_update()**

ilişkilerde anlatalım

----------------------------------------------------------------------------------------------

annonate()

**select\_related()**

**prefetch\_related()**