**# Django kurulması ve çalıştırılması**

python yüklendiği zaman (windows 10 işletim sistemi için) arayüzde pip komutunundan indirilmesi için bir tik işareti oluyor. Onu işaretleyip pip yükleyicisini kurmanız gerekmektedir.

**Django kurulumu**:

pip3 install django

komutunu terminal / cmd penceresine yazarak proje için gerekli olan django kütüphanesini direk olarak kurabilirsiniz.

**Django proje oluşturması**

django-admin startproject <django-proje-adi>

şeklinde komut girerek projemizin temelini oluşturmuş oluyoruz.

Daha sonra içine eklemek istediğimiz alt projeler için örneğin; account, basket gibi;

python3 manage.py startapp account

komutunu kullanıyoruz bu şekilde istediğimiz kadar alt uygulama oluşturabiliyoruz.

**# çalıştırılması**

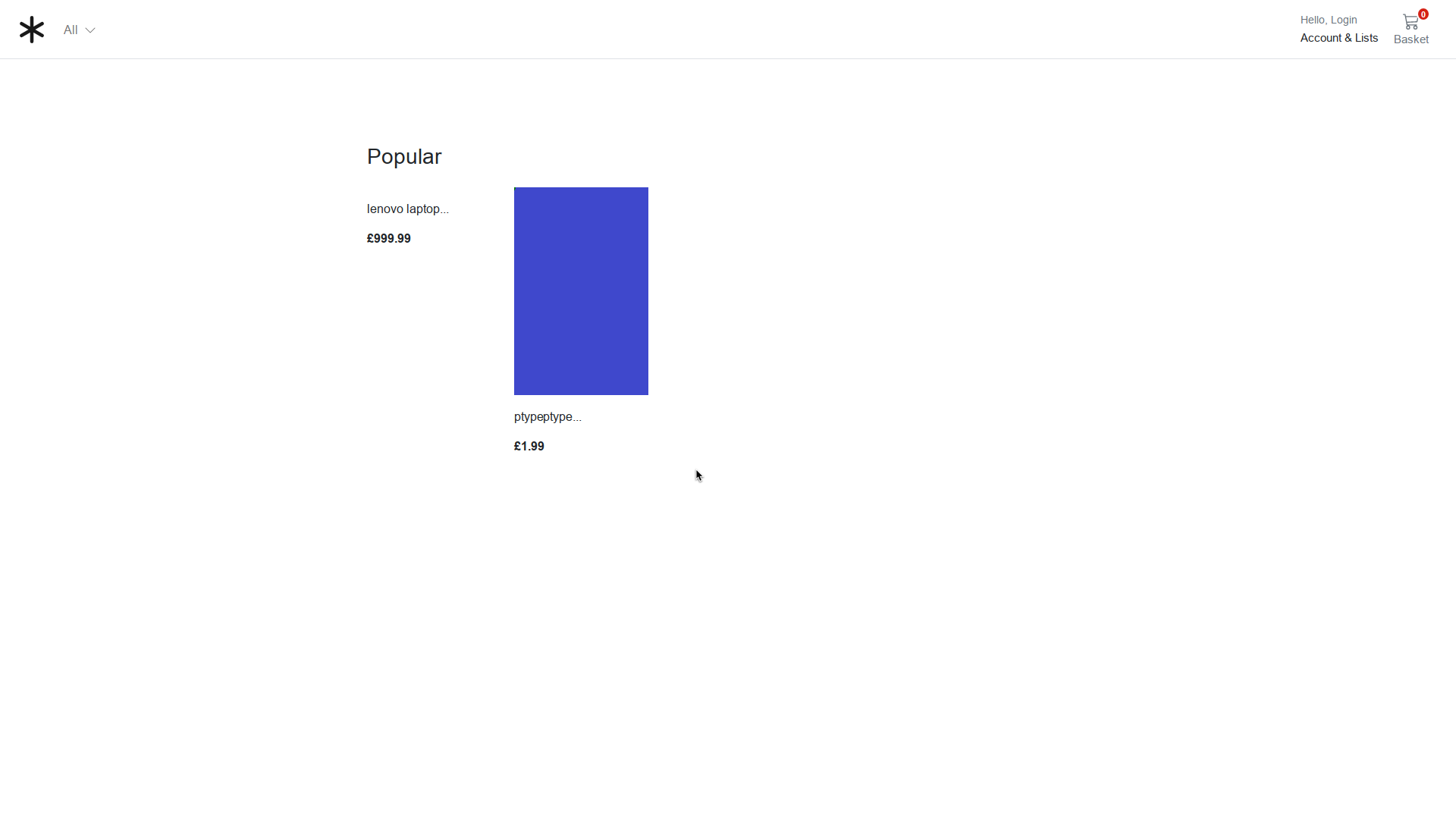
manage.py dosyasının 3 asıl alt komutu vardır. Bunlar; runserver, makemigrations, migrate’tir

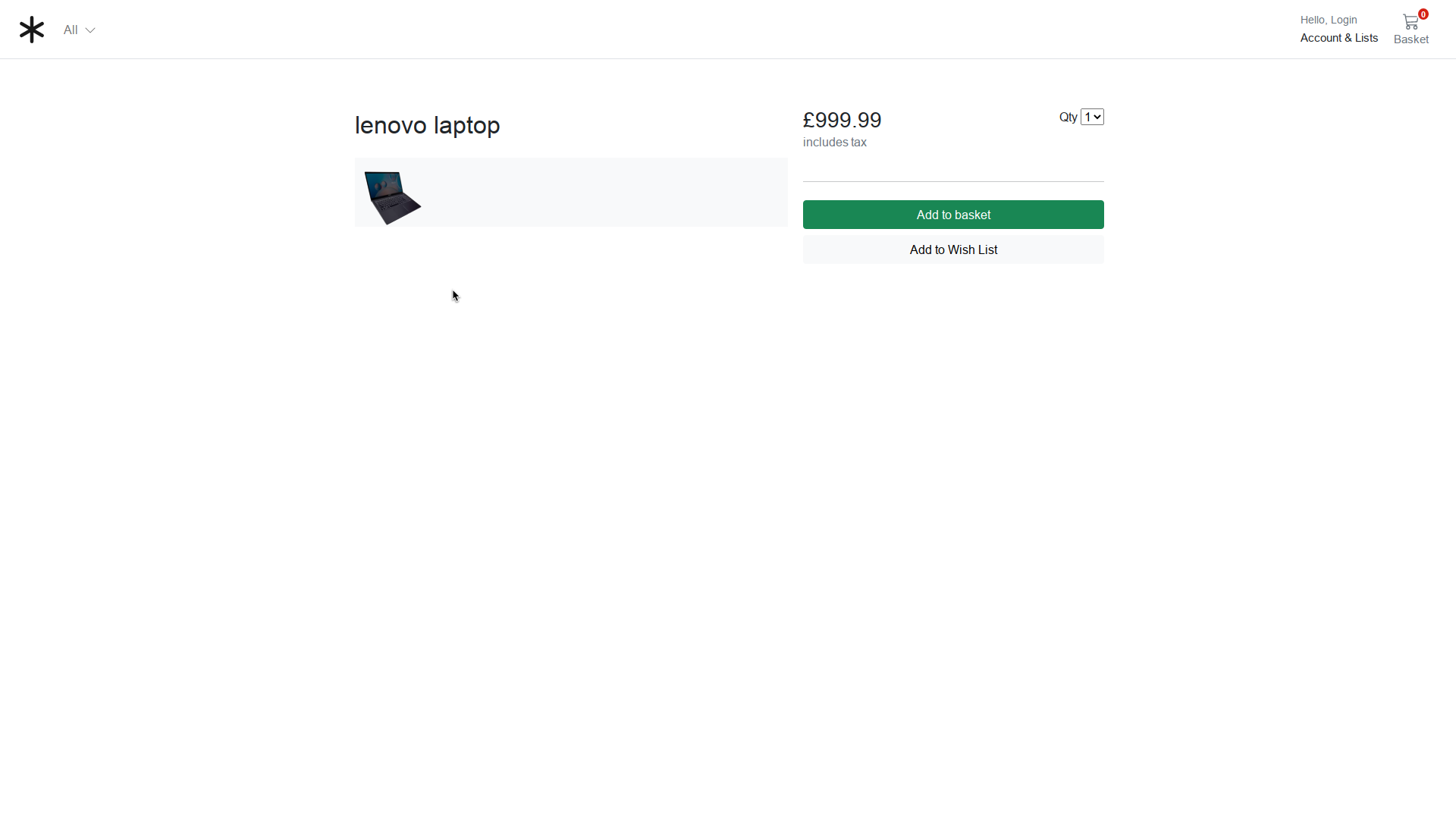
**runserver:** serverin başlatılması için kullanılır,

**makemigrations ve migrate:** yapılan değişikliklerin veritabanına aktarılmasına yarar.

python3 manage.py runserver komutu ile terminal üzerinden serverimizi başlatabiliriz. Terminal ekranında çıkan urlye tıkladığımızda ise direkt olarak varsayılan tarayıcı ile sistem açılacaktır.

**# sayfaların ekran görüntüleri**

****

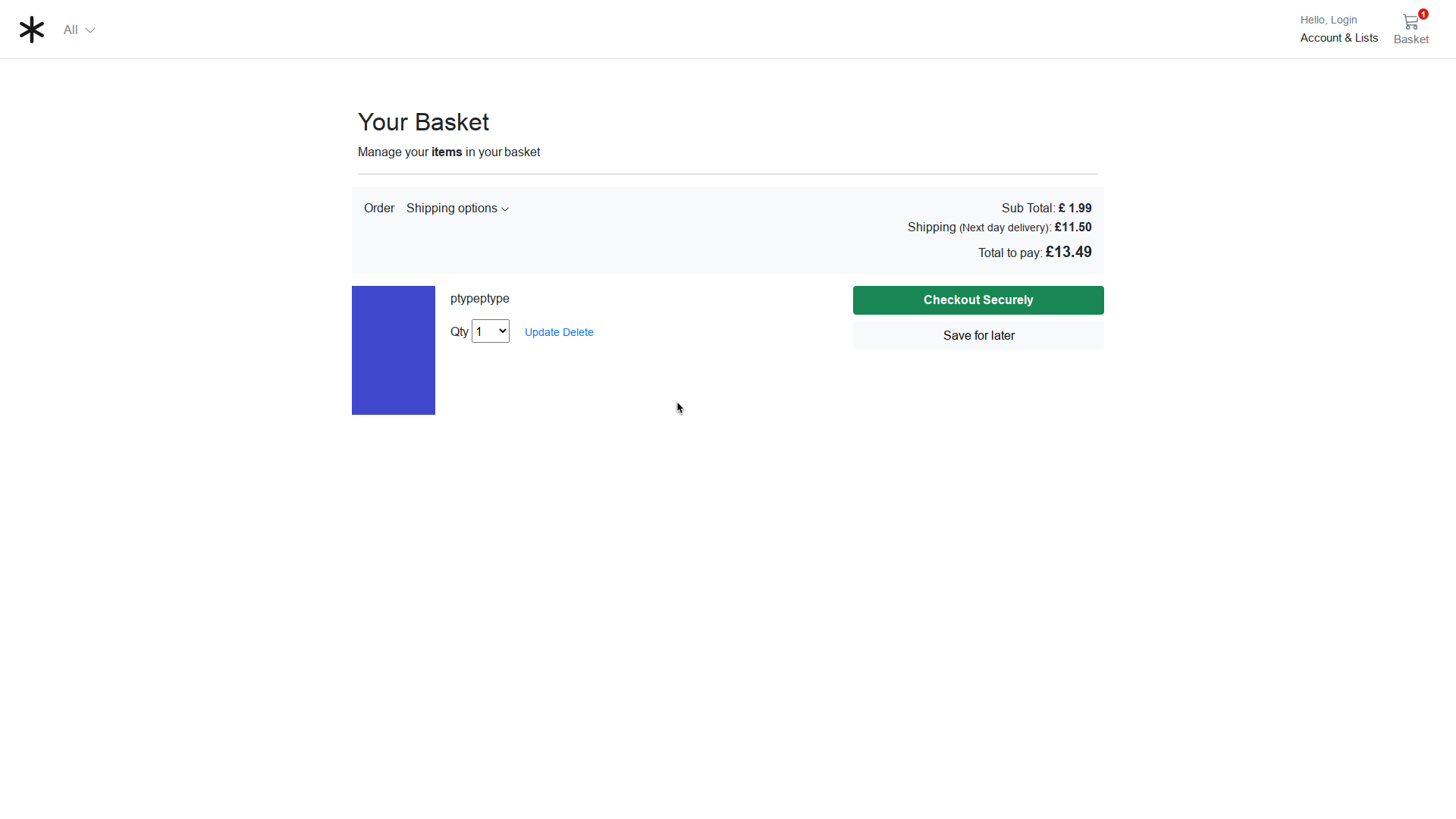
****

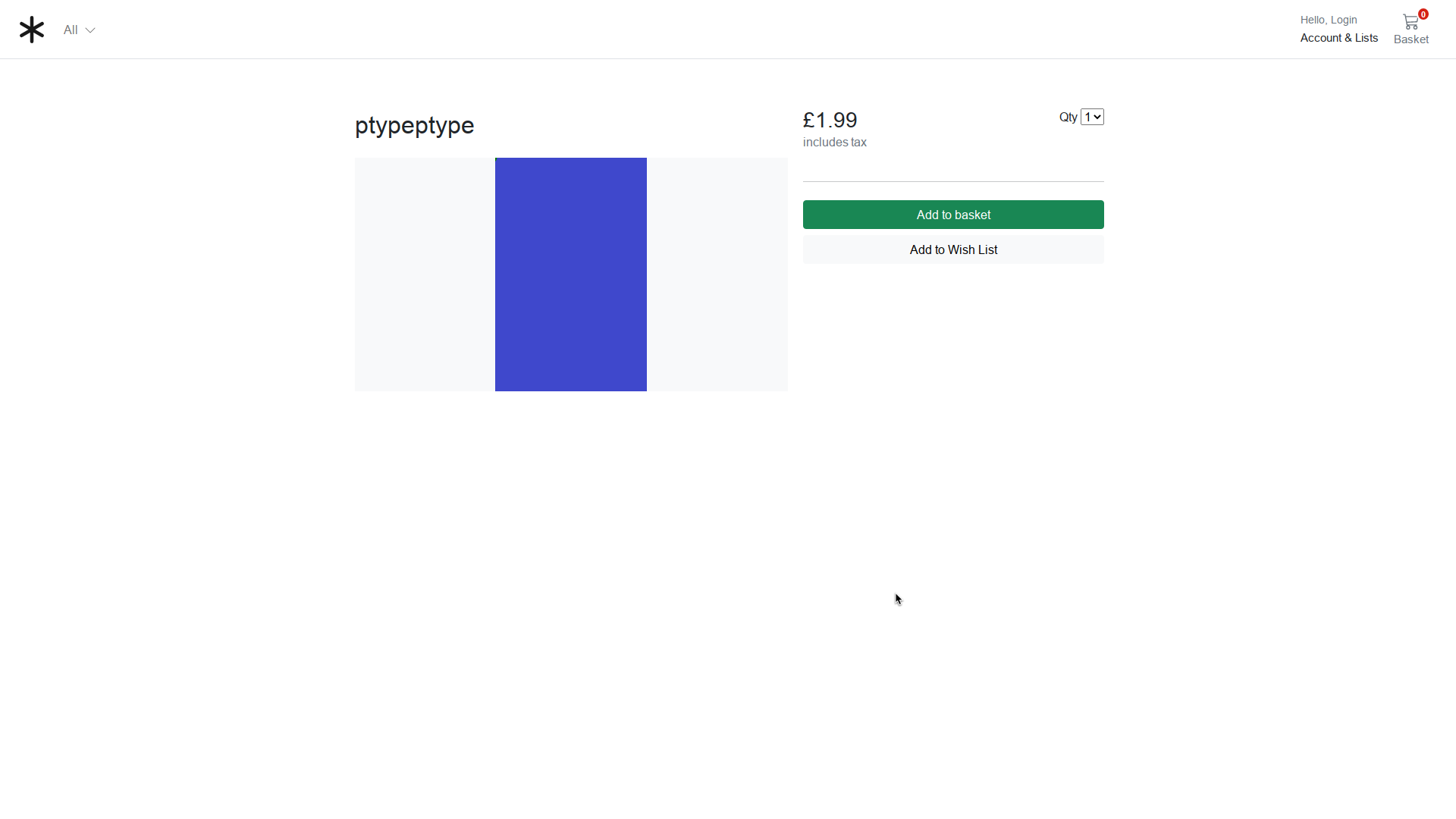
**# kullanıcı girişi çalışması**

account alt uygulaması içinde oluşturulan formda 46. satırda RegistrationForm içinde tanımlanmış olan fonksiyonlara göre html template i üzerinden renderlanmış olan girdi alanlarına istenilen değerler girildikten girilerek kaydı oluşturulmuş olan kullanıcının yine UserLoginForm içinde tanımlanmış olan input kısımları ile alınan bilgilerin, veritabanındaki kayıtlarla doğrulanması ile kullanıcı girişi tamamlanmaktadır.

UserLoginForm formundan tanımlı alanların views üzerindeki tanımlı fonksiyonla server bağlantısı sağlanmaktadır.

**# shopping cart**

****

****

**# shopping cart kodlarının nasıl ekleme yapma şekli**

Basket adında tanımlamış olduğumuz modelimizde ürün idleri üzerinden veritabanı üzerinden kategorilere ekleme mantığı ile çalışıyor. Yani views.py dosyasındaki basket\_add fonksiyonu product\_id sini alıp basket içindeki add isimli fonksiyona argüman olarak yollanılan değişkenleri veritabanına ekleme yapıyor

**# kodların**

from django.http import JsonResponse

from django.shortcuts import get\_object\_or\_404, render

from ecommerce.apps.catalogue.models import Product

from .basket import Basket

# sepet toplamını içeren fonksiyon

def basket\_summary(request):

basket = Basket(request)

return render(request, "basket/summary.html", {"basket": basket})

# elemanları sepete eklemeye yarayan fonksiyon

def basket\_add(request):

basket = Basket(request)

if request.POST.get("action") == "post":

product\_id = int(request.POST.get("productid"))

product\_qty = int(request.POST.get("productqty"))

product = get\_object\_or\_404(Product, id=product\_id)

basket.add(product=product, qty=product\_qty)

basketqty = basket.\_\_len\_\_()

response = JsonResponse({"qty": basketqty})

return response

# elemanları sepetten silmeye yarayan fonksiyon

def basket\_delete(request):

basket = Basket(request)

if request.POST.get("action") == "post":

product\_id = int(request.POST.get("productid"))

basket.delete(product=product\_id)

basketqty = basket.\_\_len\_\_()

baskettotal = basket.get\_total\_price()

response = JsonResponse({"qty": basketqty, "subtotal": baskettotal})

return response

# sepetteki elemanlara göre fiyatların güncellendiği fonksiyon

def basket\_update(request):

basket = Basket(request)

if request.POST.get("action") == "post":

product\_id = int(request.POST.get("productid"))

product\_qty = int(request.POST.get("productqty"))

basket.update(product=product\_id, qty=product\_qty)

basketqty = basket.\_\_len\_\_()

basketsubtotal = basket.get\_subtotal\_price()

response = JsonResponse({"qty": basketqty, "subtotal": basketsubtotal})

return response

import uuid

from django.contrib.auth.models import (

AbstractBaseUser,

BaseUserManager,

PermissionsMixin,

)

from django.core.exceptions import ValidationError

from django.core.mail import send\_mail

from django.core.validators import validate\_email

from django.db import models

from django.utils.translation import gettext\_lazy as \_

# kullanıcı hesaplarının düzenlemesi için yazılmış olan model

# bu model sınıfı projeye ürün eklemeye kısmında yetki verilecek olan kişilerin

# sisteme kayıtlarını sağlıyor

class CustomAccountManager(BaseUserManager):

def validateEmail(self, email):

try:

validate\_email(email)

except ValidationError:

raise ValueError(\_("You must provide a valid email address"))

# tüm her şey konusunda yetkisi olan kişinin yaratılmasını sağlayan fonksiyon

def create\_superuser(self, email, name, password, \*\*other\_fields):

other\_fields.setdefault("is\_staff", True)

other\_fields.setdefault("is\_superuser", True)

other\_fields.setdefault("is\_active", True)

if other\_fields.get("is\_staff") is not True:

raise ValueError("Superuser must be assigned to is\_staff=True")

if other\_fields.get("is\_superuser") is not True:

raise ValueError("Superuser must be assigned to is\_superuser=True")

if email:

email = self.normalize\_email(email)

self.validateEmail(email)

else:

raise ValueError(\_("Superuser Account: You must provide an email address"))

return self.create\_user(email, name, password, \*\*other\_fields)

# sadece ürün ekleme ve üzerlerinden değişim yapma izni olan kullanıcıların

# eklenmesini kolaylaştıran model fonksiyon

def create\_user(self, email, name, password, \*\*other\_fields):

if email:

email = self.normalize\_email(email)

self.validateEmail(email)

else:

raise ValueError(\_("Customer Account: You must provide an email address"))

email = self.normalize\_email(email)

user = self.model(email=email, name=name, \*\*other\_fields)

user.set\_password(password)

user.save()

return user

# müşterilerin hesap oluşturmasına yarayan kısım

# sahip olduğu tek yetki site üzerinden ürünlerin sepete eklenmesi

# ürünler üzerinden satın alma işlemi yapılması

class Customer(AbstractBaseUser, PermissionsMixin):

email = models.EmailField(\_("email address"), unique=True)

name = models.CharField(max\_length=150)

mobile = models.CharField(max\_length=20, blank=True)

is\_active = models.BooleanField(default=False)

is\_staff = models.BooleanField(default=False)

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

updated = models.DateTimeField(auto\_now=True)

objects = CustomAccountManager()

USERNAME\_FIELD = "email"

REQUIRED\_FIELDS = ["name"]

class Meta:

verbose\_name = "Accounts"

verbose\_name\_plural = "Accounts"

def email\_user(self, subject, message):

send\_mail(

subject,

message,

"l@1.com",

[self.email],

fail\_silently=False,

)

def \_\_str\_\_(self):

return self.name

# satın alma işlemi kısımlarında kullanıcının kart bilgileri oluşturma

# adres girme zip kod girme gibi kısımların kullanıcı bağlantılı olarak

# alındığı model

class Address(models.Model):

"""

Address

"""

id = models.UUIDField(primary\_key=True, default=uuid.uuid4, editable=False)

customer = models.ForeignKey(Customer, verbose\_name=\_("Customer"), on\_delete=models.CASCADE)

full\_name = models.CharField(\_("Full Name"), max\_length=150)

phone = models.CharField(\_("Phone Number"), max\_length=50)

postcode = models.CharField(\_("Postcode"), max\_length=50)

address\_line = models.CharField(\_("Address Line 1"), max\_length=255)

address\_line2 = models.CharField(\_("Address Line 2"), max\_length=255)

town\_city = models.CharField(\_("Town/City/State"), max\_length=150)

delivery\_instructions = models.CharField(\_("Delivery Instructions"), max\_length=255)

created\_at = models.DateTimeField(\_("Created at"), auto\_now\_add=True)

updated\_at = models.DateTimeField(\_("Updated at"), auto\_now=True)

default = models.BooleanField(\_("Default"), default=False)

class Meta:

verbose\_name = "Address"

verbose\_name\_plural = "Addresses"

def \_\_str\_\_(self):

return "{} Address".format(self.full\_name)

from django.contrib import messages

from django.contrib.auth import login, logout

from django.contrib.auth.decorators import login\_required

from django.contrib.sites.shortcuts import get\_current\_site

from django.http import HttpResponse, HttpResponseRedirect

from django.shortcuts import get\_object\_or\_404, redirect, render

from django.template.loader import render\_to\_string

from django.urls import reverse

from django.utils.encoding import force\_bytes, force\_text

from django.utils.http import urlsafe\_base64\_decode, urlsafe\_base64\_encode

from ecommerce.apps.catalogue.models import Product

from ecommerce.apps.orders.models import Order

from ecommerce.apps.orders.views import user\_orders

from .forms import RegistrationForm, UserAddressForm, UserEditForm

from .models import Address, Customer

from .tokens import account\_activation\_token

# istek listesi için tanımlanmış olan fonksiyon

# return satırındaki user\_wish\_list.html template dosyasına ürünleri ekler ve

# satın almaya hazır halde bekletir

@login\_required

def wishlist(request):

products = Product.objects.filter(users\_wishlist=request.user)

return render(request, "account/dashboard/user\_wish\_list.html", {"wishlist": products})

# wish list için ekleme yapacak olan butonun çalıştıracağı fonksiyon

# kullanıcı odaklı olarak

@login\_required

def add\_to\_wishlist(request, id):

product = get\_object\_or\_404(Product, id=id)

if product.users\_wishlist.filter(id=request.user.id).exists():

product.users\_wishlist.remove(request.user)

messages.success(request, product.title + " has been removed from your WishList")

else:

product.users\_wishlist.add(request.user)

messages.success(request, "Added " + product.title + " to your WishList")

return HttpResponseRedirect(request.META["HTTP\_REFERER"])

# KULLNICI HESAPLARI İŞLEMLERİ

# kullanıcı profilinin listelendiği template üzerine bilgilerin gönderildiği fonksiyon

@login\_required

def dashboard(request):

orders = user\_orders(request)

return render(request, "account/dashboard/dashboard.html", {"section": "profile", "orders": orders})

# kullanıcı profilini editlenmesi için gerekli olan sayfa ve

# bilgileri alacak olan fonksiyon

@login\_required

def edit\_details(request):

if request.method == "POST":

user\_form = UserEditForm(instance=request.user, data=request.POST)

if user\_form.is\_valid():

user\_form.save()

else:

user\_form = UserEditForm(instance=request.user)

return render(request, "account/dashboard/edit\_details.html", {"user\_form": user\_form})

# kullanıcının bilgilerinin ve hesabın komple silimnmesine yarayan fonksiyon

@login\_required

def delete\_user(request):

user = Customer.objects.get(user\_name=request.user)

user.is\_active = False

user.save()

logout(request)

return redirect("account:delete\_confirmation")

# kullanıcı hesabının onaylanması için gerekli olan linki

# oluşturan fonksiyon

# şimdilik terminal ekranına

def account\_register(request):

if request.user.is\_authenticated:

return redirect("account:dashboard")

if request.method == "POST":

registerForm = RegistrationForm(request.POST)

if registerForm.is\_valid():

user = registerForm.save(commit=False)

user.email = registerForm.cleaned\_data["email"]

user.set\_password(registerForm.cleaned\_data["password"])

user.is\_active = False

user.save()

current\_site = get\_current\_site(request)

subject = "Activate your Account"

message = render\_to\_string(

"account/registration/account\_activation\_email.html",

{

"user": user,

"domain": current\_site.domain,

"uid": urlsafe\_base64\_encode(force\_bytes(user.pk)),

"token": account\_activation\_token.make\_token(user),

},

)

user.email\_user(subject=subject, message=message)

return render(request, "account/registration/register\_email\_confirm.html", {"form": registerForm})

else:

return HttpResponse("Error handler content", status=400)

else:

registerForm = RegistrationForm()

return render(request, "account/registration/register.html", {"form": registerForm})

# aktivasyon linkine tıklandıktan sonra

# hesabı veritabanı üzerinden onaylanmış hesapların olduğu kategoriye

# taşıyan fonksiyon

def account\_activate(request, uidb64, token):

try:

uid = force\_text(urlsafe\_base64\_decode(uidb64))

user = Customer.objects.get(pk=uid)

except (TypeError, ValueError, OverflowError, user.DoesNotExist):

user = None

if user is not None and account\_activation\_token.check\_token(user, token):

user.is\_active = True

user.save()

login(request, user)

return redirect("account:dashboard")

else:

return render(request, "account/registration/activation\_invalid.html")

# Addresses

@login\_required

def view\_address(request):

addresses = Address.objects.filter(customer=request.user)

return render(request, "account/dashboard/addresses.html", {"addresses": addresses})

@login\_required

def add\_address(request):

if request.method == "POST":

address\_form = UserAddressForm(data=request.POST)

if address\_form.is\_valid():

address\_form = address\_form.save(commit=False)

address\_form.customer = request.user

address\_form.save()

return HttpResponseRedirect(reverse("account:addresses"))

else:

return HttpResponse("Error handler content", status=400)

else:

address\_form = UserAddressForm()

return render(request, "account/dashboard/edit\_addresses.html", {"form": address\_form})

@login\_required

def edit\_address(request, id):

if request.method == "POST":

address = Address.objects.get(pk=id, customer=request.user)

address\_form = UserAddressForm(instance=address, data=request.POST)

if address\_form.is\_valid():

address\_form.save()

return HttpResponseRedirect(reverse("account:addresses"))

else:

address = Address.objects.get(pk=id, customer=request.user)

address\_form = UserAddressForm(instance=address)

return render(request, "account/dashboard/edit\_addresses.html", {"form": address\_form})

@login\_required

def delete\_address(request, id):

address = Address.objects.filter(pk=id, customer=request.user).delete()

return redirect("account:addresses")

@login\_required

def set\_default(request, id):

Address.objects.filter(customer=request.user, default=True).update(default=False)

Address.objects.filter(pk=id, customer=request.user).update(default=True)

previous\_url = request.META.get("HTTP\_REFERER")

if "delivery\_address" in previous\_url:

return redirect("checkout:delivery\_address")

return redirect("account:addresses")

@login\_required

def user\_orders(request):

user\_id = request.user.id

orders = Order.objects.filter(user\_id=user\_id).filter(billing\_status=True)

return render(request, "account/dashboard/user\_orders.html", {"orders": orders})

**# tkinter arayüzü**

Tkinter Nedir

Tkinter, Python'un fiili standart GUI (Grafik Kullanıcı Arayüzü) paketidir. Yani Tkinter bir python kütüphanesidir.

Windows, diğer tüm GUI öğelerinin içinde bulunduğu kapsayıcılardır. Metin kutuları, etiketler ve düğmeler gibi bu diğer GUI öğeleri widget olarak bilinir . Widget'lar pencerelerin içinde bulunur.

Tkinter projesinde tanımlamaların hepsini el ile yapıyoruz. Labellar, butonlar, girdi alanları gibi kısımların herbiri el ile yapılıyor.

Proje girilen değerleri direk olarak sqlite veritabanına entegre etmektedir. Ve farklı uygulamalardan sqlite açıldığında içindeki verileri aynen görmek mümkündür.

Tkinter çalıştırmak için terminale;

pip3 install tkinter

yükleme işlemi tamamlandıktan sonra

python3 main.py komutları ile program çalışır hale gelecektir.