O‘ZBEKISTON RESPUBLIKASI
OLIY TA’LIM, FAN VA INNOVATSIYALAR VAZIRLIGI

U.T.Subxonqulov, D.N.Xamroyeva, U.N.Xamroyev

PYTHON

DASTURLASH TILI

DARSLIK

“KAMOLOT” nashriyoti
Buxoro – 2024

UO‘K: 004.43
KBK: 32.973.2
S91

U.T.Subxonqulov, D.N.Xamroyeva, U.N.Xamroyev, Python dasturlash tili. / [Matn]: darslik / U.T.Subxonqulov, D.N.Xamroyeva, U.N.Xamroyev - Buxoro : “BUXORO DETERMINANTI”MCHJning Kamolot nashriyoti, 2024. - 176 b.

Taqrizchilar:
BuxMTI “Axborot kommunikatsiya texnologiyalari kafedrasi p.f.d. (DSc), professori: F.R. Muradova
BuxDU “Axborot tizimlari va raqami texnologiyalar” kafedrasi dotsenti: t.f.f.d (PhD) T.R. Shafiyev

ISBN: 978-9910-734-96-0

Oliy ta’lim, fan va innovatsiyalar vazirligi Buxoro davlat universitetining 2024-yil 29-apreldagi 281-soni buyrug‘iga asosan nashr etishga ruxsat berildi. Ro‘yxatga olish raqami 281-9

MUNDARIJA:

ОГЛАВЛЕНИЕ:

TABLE OF CONTENTS:

ANNOTATSIYA

Ushbu darslikda Python dasturlash tilida dasturlash, web loyihalarini va obyektga yo'naltirilgan dasturlash texnologiyalarini, dasturlash muhitida ishlash kabi bilimlar yoritilgan. Pythonda OOP, sinflarda konstruktor, vorislik tushunchalari, MVT texnologiyasi imkoniyatlari yoritib berilgan. Djangoda modellar tayyorlash, Djangoda standart user modeli va uni loyihaga qayta moslash, formlar tayyorlash, loyiha qurish va registration, login, logout, Djangoda loyihani testlash texnologiyasi keltirilgan. Shuningdek, Djangoda tayyorlangan modellar uchun admin panelda search va filter funksiyalarini tayyorlash, Mixins va messages va generic kutubxonasi imkoniyatlari batafsil tushuntirilgan. Python dasturlash tilida Django frameworklarni ishlab chiqish, loyiha yaratish va undan foydalanish imkoniyatlari bayon qilingan. Shuningdek, keltirilgan ma'lumotlar misollar orqali asoslangan.

АННОТАЦИЯ

В данном учебнике рассматривается понятие алгоритма, его основные свойства и виды, оценка эффективности алгоритмов, алгоритмы сортировки: алгоритмы сортировки со сложностью с категориями выборки и размещения, сортировка по методу обмена, сортировки по методу Шейкерна. Также представлены и пояснены методы разработки рекурсивных алгоритмов и рекурсивных функций, методы поиска: бинарный поиск, поиск Фибоначчи, поиск по бинарному дереву, алгоритм Рабина-карпа, простые алгоритмы работы с графами.

ANNOTATION

This textbook covers the concept of an algorithm, its main properties and types, evaluation of the effectiveness of algorithms, sorting algorithms: sorting algorithms with complexity in the selection and placement category, sorting in the exchange method, Scheiker methods of sorting. Also cited are methods for developing algorithms, recursive and recursive functions, search methods: binary search, Fibonacci search, Binary Tree Search, Rabin-karp algorithm, simple algorithms working with graphs, and annotated.

KIRISH

Mamlakatimiz bugungi kunda jadal sur’atlar bilan texnologik taraqqiyot va hayotning o‘sib borayotgan dinamikasi, bir tomondan, odamlarning samarali ta’limga bo‘lgan ehtiyojlarini oshirishga, ikkinchi tomondan, uni olishning yangi usullariga olib keldi. Bu o‘rindaalbatta oli ta’lim tizimida yuqori malakali va raqobatbardosh kadrlarni tayyorlash bu tizimning eng muhim vazifalaridan hisoblanadi. Uzluksiz ta’lim tizimining keng ko‘lamli joriy etilishi, ta’lim muassasalarining mustaqilligini mustahkamlash sharoitida mutaxassislar tayyorlashda jiddiy o‘zgarishlar ro‘y berdi. Bu kadrlar tayyorlash tizimining ko‘plab tarkibiy qismlarini: maqsadlari, vazifalari, mazmuni, usullari va tashkiliy shakllarini yangi texnologiyalar va o‘quv vositalari asosida jiddiy o‘zgartirishni taqozo etdi. Natijada, ta’lim tizimini modernizatsiya qilish va ta’lim muassasalarini boshqarish muammolarini hal qilishning eng muhim innovatsion yondashuvlaridan biri ta’limni axborotlashtirish bosh muammoga aylandi.

Ana shu maqsadda qabul qilingan, O‘zbekiston Respublikasi Prezidentining “2017-2021 yillarda O‘zbekiston Respublikasini rivojlantirishning beshta ustuvor yo‘nalishlari bo‘yicha Harakatlar strategiyasi to‘g‘risida”gi Farmoni, “Oliy ma’lumotli mutaxassislar tayyorlash sifatini oshirishda iqtisodiyot sohalari va tarmoqlarining ishtirokini yanada kengaytirish chora-tadbirlari to‘g‘risida”gi 2017-yil 27-iyul PQ-3151-sonli qarori, hamda “Oliy ta’lim muassasalarida ta’lim sifatini oshirish va ularni mamlakatda amalga oshirilayotgan keng qamrovli islohotlarda faol ishtirokini ta’minlash bo‘yicha qo‘shimcha chora-tadbirlar to‘g‘risida”gi 2018- yil 5-iyun PQ-3775-sonli qarorida belgilangan vazifalarni amalga oshirish aniq qilib belgilab berilgan.

“Bugun O‘zbekiston barcha sohalarda o‘zini namoyon qilmoqda. Axborot texnologiyalariga o‘z vaqtida e’tibor qaratishimiz yaxshi natijalar bermoqda. Bitta dasturiy mahsulot davlatga korrupsiya, byurokratiyani yo‘q qilish va odamlar uchun qulayliklar yaratishda katta yordamdir. Yodingizda bo‘lsin, sizning ishingiz juda muhim”, - dedi Shavkat Mirziyoyev.

Buning natijasida ilmiy-texnik taraqqiyot talablariga muvofiq yangi ta’lim shakllari ishlab chiqilmoqda, o‘quvchi-talabalarning ta’lim faoliyatini tashkil etishning pedagogik vositasi o‘zgarib bormoqda. Ta’lim mazmunini tanlash tamoyillarini o‘zgartirish masalalari birinchi o‘ringa chiqmoqda, o‘quv jarayonida o‘quvchi-talabalarning faolligini amalda tatbiq etishga yordam beradigan yangi o‘qitish usullari qo‘llanilmoqa.

Ta’lim muammolarini hal qilishda talabalarning ijodiy va ilmiy mustaqilligi, faolligini rivojlantirishda oliy o‘quv yurtlarining roli ortib bormoqda. Ta’lim samaradorligini oshirish, ta’lim mazmunini puxta ilmiy tanlab olish va pedagogik jarayonni takomillashtirish yo‘llari bo‘yicha izlanishni talab qiladi. Bu esa ta’lim muassasalarida o‘quv jarayonini optimallashtirish kabi muammolarni hal qilish bilan bog‘liqdir. Shunday qilib, ta’lim tizimida dasturlash asoslari yuzasidan bilim va malakalarni shakllantirish va uni rivojlantirish, mazkur jarayonga nisbatan tizimli, kompleks yondashuvni taqozo etadi.

Demak, ta’lim olayotgan o‘quvchi-talabalarni davr talabiga javob bera oladigan yetuk mutaxassis, komil inson bo‘lib tarbiyalanishlarida, yaratuvchanlik tamoyili asosida yangi g‘oyalarga tayangan holdi dasturlash tillaridan kreativ xususiyatlarini shakllantirish maqsadida dasturlash tillarini o‘qitish katta ahamiyat kasb etadi.

Ushbu darslik 60610200 -Axborot tizimlari va texnologiyalari bakalavriat ta’lim yo‘nalishida tahsil olayotgan talabalarning o‘zlashtirishi lozim bo‘lgan bilimlari asosida tuzilgan bo‘lib, talabalarning egallashi kerak bo‘lgan bilimlar va ko‘nikmalar mazmunini o‘z ichiga oladi.

Zamonaviy dunyoda, hayotni axborot texnologiyalarisiz tasavvur qilishning iloji yo‘q. Ular bizning hayotimizga qat’iy kirishdi, axborot texnologiyalari insoniyat hayotining barcha sohalarida qo'llaniladi, ayniqsa ikki tomonlama muhim rol o'ynaydi. Axborot texnologiyalari insoniyatning barcha to'plangan tajribasini amaliy foydalanish uchun mos formatlangan shaklda aks ettiradi. Shuningdek, u ijtimoiy jarayonlarni amalga oshirish uchun ilmiy bilim va materialistik tajribani jamlaydi, shu bilan birga mehnat, vaqt, energiya va moddiy xarajatlarni tejaydi.

Django framework python dasturlash tilida ishlab chiqilgan frameworklardan hisoblanadi, django framework ni asosiy sayti djangoproject.com hisoblanadi. Ushbu framework doimiy tarzda o'zgarib boradigan frameworklar sirasiga kiradi. Django MVT moduliga asoslangan framework hisoblanadi. MVT(Model, View, Template) bu sizga loyiha yaratishizda juda avfzallik beradi.

Django MVT(Modelni ko'rish shabloni) dizayn namunasiga amal qiladi. Model - Siz taqdim qilmoqchi bo'lgan ma'lumotlar, ma'lumotlar bazasi bilan o'zaro ishlashda qulaylik beradi. View - foydalanuvchi so'rovi asosida tegishli shablon va tarkibni qaytaradigan so'rovlar ustida ishlash uchun foydalaniladi. Template - matn fayli (HTML fayli kabi), veb-sahifaning tartibini o'z ichiga olgan, ma'lumotlarni qanday ko'rsatish kerakligini ta'minlash uchun foydalaniladi. “Python dasturlash tili” darsligi kirish, 15 ta mavzu, mundarija hamda glossariy, foydanilgan adabiyotlar va internet resurlari ro‘yxatidan iborat. Darslikda dasturlar sharti, kerakli loyihalar, modellar va ularning natijalari ko‘rinishida berilgan.

Darslikning umumiy hajmi 205 betdan iborat bo‘lib, ko‘plab Django loyihalari va ular ustida amallar bajarish imkoniyatlari ko‘rsatib o‘tilgan.

Darslik ba’zi kamchiliklardan xoli bo‘lmasligi mumkin. Uni yanada mukammal bo‘lishi yuzasidan bildirilgan fikr-mulohazalarni mamnuniyat bilan qabul qilamiz.

1-§. PYTHONDA OOP TUSHUNCHALARI.

Boshqa umumiy maqsadli tillar singari, Python ham obyektga yo‘naltirilgan til hisoblanadi. Python - obyektga yo‘naltirilgan dasturlash tilidir. Bu bizga obyektga yo‘naltirilgan yondashuv yordamida dasturlarni ishlab chiqishga imkon beradi. Pythonda biz osongina sinflar va obyektlarni yaratishimiz va ulardan foydalanishimiz mumkin.

Obyektga yo‘naltirilgan dasturlash tizimining asosiy prinsiplari quyida keltirilgan:

- ✓ Object (Obyekt)
- ✓ Class (Sinf)
- ✓ Method (metod, usul)
- ✓ Inheritance (Meros olish)
- ✓ Polymorphism (Polimorfizm)
- ✓ Data Abstraction (Ma’lumotlarni olish)
- ✓ Encapsulation (Inkapsulyatsiya)

Object (Obyekt). Obyekt - bu holat va xulq-atvor, xususiyatlarga ega bo‘lgan shaxs hisoblanadi. Bu sichqoncha, klaviatura, stul, stol, ruchka va boshqa turdagi har qanday haqiqiy obyekt bo‘lishi mumkin. Pythondagi hamma narsa obyekti bo‘lib, deyarli hamma narsada atributlar va metodlar mavjud. Barcha funksiyalar funksiya manba kodida belgilangan doc qatorini qaytaradigan o‘rnatilgan doc atributiga ega.

Class (Sinf). Sinf obyektlar to‘plami sifatida aniqlanishi mumkin. Bu ba’zi bir o‘ziga xos atributlar va usullarga ega bo‘lgan mantiqiy shaxs. Masalan: agar sizda ishchilar sinfingiz bo‘lsa, unda atribut va usulni, ya’ni elektron pochta identifikatori, ism, yosh, ish haqi va boshqalarni o‘z ichiga olishi kerak.

Method (metod, usul). Metod - bu obyekt bilan bog‘liq bo‘lgan funksiya. Pythonda metod faqat sinf misollari uchun xos emas. Har qanday obyekt turi metodlariga ega bo‘lishi mumkin.

Inheritance (Meros olish). Merosxo‘rlik - bu haqiqiy dunyo meros tushunchasini simulyatsiya qiladigan obyektga yo‘naltirilgan dasturlashning eng muhim jihati. Bola obyekti ota-onaning barcha xususiyatlarini va xatti-harakatlarini egallashini belgilaydi. Merosdan foydalanib, biz boshqa sinfning barcha xususiyatlari va xatti-harakatlaridan foydalanadigan sinfni yaratishimiz mumkin. Yangi sinf hosil bo‘lgan sinf yoki bola klasi, xossalari olingan sinf esa asosiy sinf

yoki ota-ona sinfi sifatida tanilgan. Bu kodning qayta ishlatilishini ta’minlaydi.

Polymorphism (Polimorfizm). Polimorfizm tarkibida ikkita “poli” va “morflar” so‘zlari mavjud. Poli ko‘p, morflar esa shakllar degan ma’noni anglatadi. Polimorfizm bilan biz bitta vazifani har xil usulda bajarish mumkinligini tushunamiz.

Encapsulation (Inkapsulyatsiya). Inkapsulyatsiya – obyektga yo‘naltirilgan dasturlashning muhim jihati hisoblanadi. U metodlar va o‘zgaruvchilarga kirishni cheklash uchun ishlatiladi. Inkapsulyatsiya kod va ma’lumotlar tasodifan o‘zgartirilishidan bir-birining ichida birlashtiriladi.

Data abstraction (Ma’lumotlarni abstraktsiya qilish). Ma’lumotlarni ajralish va inkapsulyatsiya qilish ikkalasi ham ko‘pincha sinonim sifatida ishlatiladi, chunki ma’lumotlar abstraktsiyasiga inkapsulyatsiya orqali erishiladi. Abstraktsiya ichki tafsilotlarni yashirish va faqat funksionallikni ko‘rsatish uchun ishlatiladi. Biron bir narsani mavhumlashtirish, bu narsa funksiyalar yoki butun dastur bajaradigan ishlarning mohiyatini o‘z ichiga olishi uchun narsalarga nom berishni anglatadi.

Obyektga yo‘naltirilgan dasturlash - bu muammolarni yechishga qaratilgan yondashuv va hisoblash obyektlar yordamida amalga oshiriladigan joyda qo‘llaniladi. Protsedurali dasturlash hisoblashlarni bosqichma-bosqich bajarish bo‘yicha ko‘rsatmalar ro‘yxatidan foydalanadi.

Nazorat savollari:

1. OOP nima?
2. Python siniflari va obyektlari haqida qisqa ma’lumot bering.
3. Python dasturlash tilida klasslar va obyekt metodlarini qanday tuzish mumkin?
4. Pythonda sinf qanday yaratiladi?
5. Pythonda obyekt qanday yaratiladi?
6. Pythonda "inheritance"(meroslash) qanday amalga oshiriladi?

2-§. SINFLARDA KONSTRUKTOR TUSHUNCHASI.

Aytaylik, sinf binoning prototipidir. Bino polga, eshiklarga, derazalarga va hokazolarga oid barcha ma’lumotlarni o‘z ichiga oladi, biz ushbu detallarga asoslanib, xohlagancha bino yasay olamiz. Demak, binoni sinf sifatida ko‘rish mumkin va biz shu sinfning shuncha obyektini yaratishimiz mumkin. Boshqa tomondan, obyekt sinfning misoli. Obyektni yaratish jarayonini **instantatsiya** deb atash mumkin. Darslikning ushbu qismida biz pythonda sinflar va obyektlarni yaratishni muhokama qilamiz. Atributga sinf obyekti yordamida qanday erishish mumkinligi haqida ham gaplashamiz.

Python – obyektga yo‘naltirilgan dasturlash tili. Pythonda deyarli barcha narsa obyekt hisoblandi. Ularning o‘z xususiyatlari va funksiyalari bor. Sinflar esa obyekt konstruktorlari hisoblanadi. Ular bilan obyektlar tuziladi.

Sinf hosil qilish. Sinf hosil qilish uchun **class** kalit so‘zi ishlatiladi. Hozir biz “Son” degan sinf hosil qilamiz. Shu sinf nomini print so‘zi bilan ekranga chiqarish buyrug‘ini bersak, shu sinf mavjudligi haqida ma’lumot chiqadi:

```
class Son:
x = 5
print(Son)
```

Obyekt hosil qilish. Sinflar obyekt konstruktorlari ekanligini aytgan edik. Hozir yuqorida hosil qilgan sinfimiz orqali yangi obyekt hosil qilamiz. Uning nomi s1 bo‘lsin.

```
class Son:
x = 5
s1 = Son()
print(s1.x)
5
```

__init__() funksiyasi. Yuqoridagi misollarimizdagi sinf va obyektlar bilan shunchaki sodda ko‘rinishda tanishib chiqdik. Ammo ular haqiqiy dasturlar tuzishga yaroqsiz. Sinflarning mohiyatini tushunish uchun init() ichki funksiyasini bilishimiz lozim. Har bir sinf tuzilgan paytda init() funksiyasi mavjud bo‘ladi. init() funksiyasi obyektlar tuzilayotgan paytda ularning xususiyatlariga qiymatlarni yoki bajarilishi kerak bo‘lgan operatsiyalarni biriktiradi. Hozir Ishchi degan sinf hosil qilamiz va unda ism va yosh ko‘rsatkichlariga qiymatlar

o‘zlashtirish uchun init() funksiyasidan foydalanamiz. Keyin init() funksiyasi har safar yangi obyekt tuzilganda avtomatik tarzda ishlaydi. Eslatib o‘tamiz, init() funksiyasini yozayotganda har ikkala tarafdan ham ikkitadan (__) tag chiziq yoziladi. Yuqoridagi misollarimizdagi sinf va obyektlar bilan shunchaki sodda ko‘rinishda tanishib chiqdik. Ammo ular haqiqiy dasturlar tuzishga yaroqsiz. Sinflarning mohiyatini tushunish uchun init() ichki funksiyasini bilishimiz lozim.

Har bir sinf tuzilgan paytda init() funksiyasi mavjud bo‘ladi. init() funksiyasi obyektlar tuzilayotgan paytda ularning xususiyatlariga qiymatlarni yoki bajarilishi kerak bo‘lgan operatsiyalarni biriktiradi.

Sinflarda konstruktor tushunchasi. Konstruktor - bu sinfning instansiya a’zolarini initsializatsiya qilish uchun ishlatiladigan maxsus metod(funksiya) turi hisoblanadi.

Konstruktorlar ikki xil bo‘lishi mumkin:

- parametrlangan konstruktor;
- parametrlanmagan konstruktor.

Ushbu sinf obyektini yaratganimizda konstruktor ta’rifi bajariladi. Shuningdek, konstruktorlar obyekt uchun biron bir ishga tushirish vazifasini bajarish uchun yetarli resurslar mavjudligini tasdiqlaydilar.

Pythonda konstruktor yaratish. Pythonda init metodi sinf konstruktorini simulyatsiya qiladi. Biz init ta’rifiga qarab, sinf obyektini yaratishda istalgan sonli argumentlarni berishimiz mumkin. Bu asosan sinf atributlarini ishga tushirish uchun ishlatiladi. Har bir sinf konstruktorga ega bo‘lishi kerak, hatto u oddiygina konstruktorga tayansa ham.

Misol:

```
class Employee:
def init (self,name,id):
self.id = id; self.name = name;
def display (self):
print("ID: %d \nName: %s"%(self.id,self.name))
emp1 = Employee("Usmon",101)
emp2 = Employee("Yunus",102)
emp1.display();
emp2.display();
```

Natija:

```
ID: 101
```

Name: Usmon
ID: 102
Name: Yunus
Misol:

```
class fff():
  def __init__(self, name, breed):
    self.name = name
  self.breed = breed
  def __repr__(self):
    return f"Fff(name={self.name}, breed={self.breed})"
max = fff("Max", "Golden Retriever")
pax = fff("Pax", "Labrador")
print(max)
print(pax)
```

1-qatorda biz fff nom yordamida yangi sinfni e'lon qilamiz. Keyin biz __init__ deb nomlangan funksiyaga duch kelamiz. Har bir Python sinfida bu mavjud, chunki u standart konstruktordir. Bu funksiya obyekt holatini ishga tushirish uchun ishlatiladi, shuning uchun u yangi yaratilgan obyektning o'zgaruvchilariga qiymatlar beradi. Konstruktorning argumentlari sifatida bizda name, the breed, va maxsus kalit so'z self mavjud.

Sinf kodining ichida self kalit so'z sinfning joriy nusxasini ifodalaydi. Bu shuni anglatadiki, har safar biz sinf misoliga tegishli bo'lgan ma'lum bir usul yoki o'zgaruvchiga kirishni istasak, self kalit so'zdan foydalanishimiz kerak.

Usulning birinchi qatoriga qarang:

__init__- self.name = name.

So'z bilan aytganda, bu Python tarjimoniga shunday deydi: "Yaxshi, biz yaratayotgan obyekt nomi (self.name) bo'ladi va bu nom argument ichida name". Xuddi shu narsa breed argument uchun ham sodir bo'ladi. Shuning uchun biz shu yerda to'xtashimiz mumkin edi. Bu sinfni aniqlash uchun ishlatiladigan asosiy sxema. Ushbu parchaning bajarilishiga o'tishdan oldin, __init__ dan keyin qo'shilgan usulni ko'rib chiqaylik.

Ikkinchi usul __repr__ deyiladi. Pythonda __repr__usul sinf obyektini satr sifatida ifodalaydi. Odatda, agar biz buni aniq belgilamasak, Python uni o'ziga xos tarzda amalga oshiradi. Agar biz aniq usulni aniqlamasak, __repr__funksiyani chaqirganda, Python

obyektning xotira ko‘rsatkichini qaytaradi. Buning o‘rniga, agar biz moslashtirilgan usulni aniqlasak, bizda obyektni qayta qurish uchun str()__repr__ ham ishlatilishi mumkin.
Yuqoridagi kodga o‘zgartirish kiritamiz:

```
class Fff:
  def __init__(self, name, breed):
    self.name = name
    self.breed = breed
max = Fff("Max", "Golden Retriever")
pax = Fff("Max", "Golden Retriever")
print(max)
print(pax)
print(max == pax)
```

Agar biz ushbu kodni saqlasak va ishga tushirsak, biz quyidagilarni olamiz:

```
<__main__.Fff object at 0x0000026BD792CF08>
<__main__.Fff object at 0x0000026BD792CFC8>
False
```

Kutib turing, qanday qilib ular bir xil ism va bir zotga ega bo‘lsa, ular ikkita teng it bo‘lmasligi mumkin? Keling, avval tuzilgan diagramma yordamida buni tasavvur qilaylik.

Birinchidan, biz print(max)ni ishga tushirganimizda, Python __repr__ usulning maxsus ta’rifi yo‘qligini ko‘radi va u __repr__usulning sukut bo‘yicha amalga oshirilishidan foydalanadi. Ikki obyekt max va pax, ikki xil obyektdir. Ha, ular bir xil nom va bir xil zotga ega, ammo ular Fff sinfning turli xil namunalari. Aslida, ular turli xil xotira joylariga ishora qiladilar, chunki biz chiqishning dastlabki ikki qatoridan ko‘ramiz. Bu fakt obyekt va sinf o‘rtasidagi farqni tushunish uchun juda muhimdir.

Agar biz birinchi kod misolini bajarsak, biz maxsus __repr__ usulni qo‘llaganimizda chiqishdagi farqni ko‘rishimiz mumkin:

```
Fff(name=Max, breed=Golden Retriever)
Fff(name=Pax, breed=Labrador)
```

Yangi usulni aniqlash. Aytaylik, biz max obyekt nomini olishni xohlaymiz. Bu holda name atribut ommaviy bo‘lganligi sababli, biz uni max.name atributga kirish orqali olishimiz mumkin. Xo‘sh, u holda biz sinfimiz ichida deb nomlangan usulni yaratamiz. Keyin, sinf ta’rifidan tashqari, biz oddiygina max.get_nickname() usulni chaqiramiz:

```
class Fff:
  def __init__(self, name, breed):
    self.name = name
    self.breed = breed
  def get_nickname(self):
    return f"{self.name}, the {self.breed}"
  def __repr__(self):
    return f"Fff(name={self.name}, breed={self.breed})"
max = Fff("Max", "Golden Retriever")
pax = Fff("Pax", "Labrador")
print(max.name)
print(max.get_nickname())
```

Agar biz ushbu qismni ishga tushirsak, biz quyidagi natijani olamiz:

```
> python snippet.py
Max
Max, the Golden Retriever
Background Image
Progress Bar
00:00 00:00
```

Kirish modifikatorlari: public, protected va private. Kirish modifikatorlarini ko‘rib chiqaylik. OOP tillarida kirish modifikatorlari sinflar, usullar yoki atributlarning mavjudligini o‘rnatish uchun ishlatiladigan kalit so‘zlardir. Bu C++ va Javada boshqacha holat, bunda kirish modifikatorlari til tomonidan aniqlangan kalit so‘zlardir. Pythonda bunday operator yo‘q. Pythondagi kirish modifikatorlari kirishni boshqarish kafolati emas, balki konventsiyadir.

Kod namunasi bilan bu nimani anglatishini ko‘rib chiqaylik:

```
class BankAccount:
  def __init__(self, number, openingDate):
    # public access
    self.number = number
    # protected access
    self._openingDate = openingDate
    # private access
    self.__deposit = 0
```

Biz BankAccount nomli sinf yaratamiz. Har qanday yangi obyekt uchta atributga ega bo‘lishi kerak: raqam, ochilish sanasi va boshlang‘ich depozit 0 ga o‘rnatilgan .

Python konventsiyasiga ko‘ra, bitta pastki chiziq protected a’zolar uchun prefiks sifatida ishlatiladi, ikkita pastki chiziq esa private a’zolar uchun. Bunday amal nimani anglatadi? Quyidagi kodni tushunishga harakat qilaylik:

```
account = BankAccount("ABXX", "01/01/2022")
print(account.number)
print(account._openingDate)
print(account.__deposit)
```

Agar biz ushbu kodni ishga tushirishga harakat qilsak, biz quyidagicha natija olamiz:

```
> python snippet.py
ABXX
01/01/2022
Traceback (most recent call last):
File "snippet.py", line 14, in <module>
 print(account.__deposit)
AttributeError: 'BankAccount' object has no attribute '__deposit'
```

Number accountini chop etishimiz mumkin, chunki u umumiy atributdir. openingDate konventsiyaga ko‘ra tavsiya etilmagan bo‘lsa ham, biz chop etishimiz mumkin. Biz depozitni chop eta olmaymiz.

Depozit atributi bo‘lsa, uning qiymatini o‘qish yoki o‘zgartirishning to‘g‘ri usuli get()va set() usullari bo‘lishi kerak. Bunga misolni ko‘rib chiqaylik:

```
class BankAccount:
  def __init__(self, number, openingDate):
    self.number = number
    self._openingDate = openingDate
    self.__deposit = 0
  def getDeposit(self):
    return self.__deposit
  def setDeposit(self, deposit):
    self.__deposit = deposit
    return True
account = BankAccount("ABXX", "01/01/2022")
print(account.getDeposit())
```

```
print(account.setDeposit(100))
print(account.getDeposit())
```

Yuqoridagi kodda biz ikkita yangi usulni aniqlaymiz. Birinchisi getDeposit chaqiriladi, ikkinchisi esa setDeposit. Ularning nomlaridan ko‘rinib turibdiki, ular merosni olish yoki o‘rnatish uchun ishlatiladi. Bu o‘qilishi yoki o‘zgartirilishi kerak bo‘lgan barcha atributlar uchun olish va sozlash usullarini yaratish uchun OOP konventsiyasidir. Shunday qilib, sinfdan tashqarida ularga to‘g‘ridan-to‘g‘ri kirish o‘rniga, biz buni amalga oshirish usullarini qo‘llaymiz.

Osonlik bilan taxmin qilishimiz mumkinki, ushbu kodni bajarish quyidagi natijani beradi:

```
> python snippet.py
0
True
100
```

Nazorat savollari:

1. Pythonda klassларn yurituvchilari (constructors) haqida qisqa ma’lumot bering.
2. Klass oyinlarini (inheritance) qanday ishlatish mumkin?
3. Pythonda polimorfizm (polymorphism) nima?
4. Abstrakt sinflar va metodlar Pythonda qanday tuziladi?
5. Klasslar va obyektlar orasidagi farq haqida ma’lumot bering.

3-§. SINFLARDA VORISLIK TUSHUNCHASI.

Vorislik - bu atama sinflarga xosdir. **Vorislik** deb, bir sinfdagi barcha funksiya va xususiyatlarni boshqa bir sinf o‘ziga o‘zlashtirishiga aytiladi. Funksiyalari meros qilib olinadigan sinf **ona sinf** deyiladi. Meros qilib olingan funksiyalarni o‘ziga o‘zlashtiradigan sinf **voris sinf** deyiladi.

Ona sinf hosil qilish. Istalgan sinf ona sinf bo‘lishi mumkin. Shu sababli ona sinfni hosil qilish xuddi oddiy sinfni hosil qilish kabidir. Masalan, Odam nomli sinf hosil qilamiz. Unda ism va familiya parametrlari va tanish degan funksiyasi bo‘ladi. So‘ngra shu sinf orqali x obyekt hosil qilamiz:

```
class Odam:
def init (self, ism, familiya): self.ism = ism
self.familiya = familiya
def tanish(self):
print(self.ism, self.familiya)
x = Odam ("Adizova", "Zuhro") x.tanish()
```

Adizova Zuhro

Voris sinf hosil qilish. Voris sinf hosil qilish uchun yangi sinf tuzilayotganda ona sinfni parametr sifatida kiritamiz. Shunda voris sinf ona sinfdan barcha xususiyatlarni o‘zlashtiradi.

Talaba degan sinf hosil qilamiz. Odam sinfi uning ona sinfi bo‘ladi. Qavslar ichida ona sinfni kiritamiz va uning barcha xususiyatlarini voris sinf o‘zlashtiradi. Qo‘shimcha parametr qo‘shish shart emas, ammo sinf hosil qilayotganda ichi bo‘sh bo‘lishi ham mumkin emas. Agar hech narsa yozishni istamasak xatolik yuz bermasligi uchun pass kalit so‘zini qo‘shib qo‘yamiz:

```
# Endi voris sinf ya'ni bola sinfni hosil qilamiz
class Talaba (Odam): pass
x = Talaba ("Ali", "Adizov")
x.tanish()
```

__init__() funksiyasini qo‘shish. Avvalgi misolimizda voris sinf hosil qilganimizda pass kalit so‘zi bilan cheklanib qo‘ya qoldik. Shu sababli voris sinf barcha funksiyalarni avtomatik tarzda o‘zlashtirgan edi. Endi voris sinfga init() funksiyasi bilan parametrlarini joylashtiramiz. Bunda voris sinf ona sinfdagi init() funksiyasidan emas o‘zidagidan foydalanadi.

```
class Odam:
```

```
def init (self, ism, familiya): self.ism = ism
self.familiya = familiya
def tanish(self):
print(self.ism, self.familiya)
# Endi voris sinf ya'ni bola sinfni hosil qilamiz class Talaba (Odam):
def init (self, ism, familiya):
self.ism = ism self.familiya = familiya
x = Talaba (" ", " ") x.tanish()
# Endi voris sinf ya'ni bola sinfni hosil qilamiz
class Talaba (Odam):
def init (self, ism, familiya): Odam. init (self, ism, familiya)
x = Talaba (" ", " ")
x.tanish()
```

super() funksiyasi

```
class Odam:
def init (self, ism, familiya): self.ism = ism
self.familiya = familiya
def tanish(self):
print(self.ism, self.familiya)
# Endi voris sinf ya'ni bola sinfni hosil qilamiz
class Talaba (Odam):
def init (self, ism, familiya): super(). init (ism, familiya)
x = Talaba ("A", " ")
x.tanish()
```

Sinflar bilan ishlash uchun maxsus super() funksiyasi ham mumkin. Bu funksiya ona sinfdagi barcha funksiya va parametrlarni voris sinfga o‘zlashtiradi.

Parametr qo‘shish. Voris sinf hosil qilingach unga yana qo‘shimcha parametr qo‘shmoqchi bo‘lsak quyidagicha amalga oshirish mumkin. Hozir yil parametrini qo‘shamiz:

```
# Endi voris sinf ya'ni bola sinfni hosil qilamiz
class Talaba (Odam):
def init (self, ism, familiya): super(). init (ism, familiya) self.yil =
2002
x = Talaba ("", "")
print(x.yil)
```

Yuqoridagi misolimizda yangi parametrni qo‘shib unga qiymat berdik. Endi init() funksiyasining o‘ziga yil parametrini qo‘shib unga

o‘zlashtiramiz. Shundan so‘ng, uning qiymatini yangi obyekt hosil qilayotganda o‘zimiz kirtishimiz kerak bo‘ladi.

```
class Odam:
def init (self, ism, familiya): self.ism = ism
self.familiya = familiya
def tanish(self):
print(self.ism, self.familiya)
# Endi voris sinf ya'ni bola sinfni hosil qilamiz
class Talaba (Odam):
def init (self, ism, familiya, yil): super(). init (ism, familiya) self.yil =
2002
x = Talaba (" ", " ", 1996) print(x.yil)
1996
```

Funksiya qo‘shish. Voris sinfga qo‘shimcha funksiyalar ham qo‘shish mumkin. Natijada u ona sinfdan o‘zlashtirgan funksiyalari va biz qo‘shgan qo‘shimcha funksiyalarga ega bo‘ladi. Hozir voris sinfga tugilgan() funksiyasini qo‘shamiz. Bu funksiya talabaning tug‘ilgan yili haqida ma’lumot beradi:

```
class Odam:
def init (self, ism, familiya): self.ism = ism
self.familiya = familiya
def tanish(self):
print(self.ism, self.familiya)
# Endi voris sinf ya'ni bola sinfni hosil qilamiz
class Talaba (Odam):
def init (self, ism, familiya, yil): super(). init (ism, familiya) self.yil =
1996
def tugilgan(self):
print("Men" , self.yil , " - yilda tug‘ilganman")
x = Talaba ("Ali", "Adizov", 1996) x.tugilgan()
Men 1996 - yilda tug‘ilganman
```

Nazorat savollari:

1. Vorislik nima?
2. Qanday voris sinf hosil qilinadi?
3. Ona sinf qanday hosil qilinadi?
4. Super() funksiyasi nima?
5. Parametr qanday hosil qilinadi?

4-§. MVT TEXNOLOGIYASI DJANGO FRAMEWORK.

Django nima? Django - bu Python veb-ramkasi bo‘lib, u veb-ishlab chiqishdagi umumiy amaliyotlarni soddalashtiradi va umumiy rivojlanish ehtiyojlarini qo‘llab-quvvatlaydigan barqaror kutubxonalar ekotizimini taklif qilish orqali ko‘p qiyinchiliklarni bartaraf etadi.

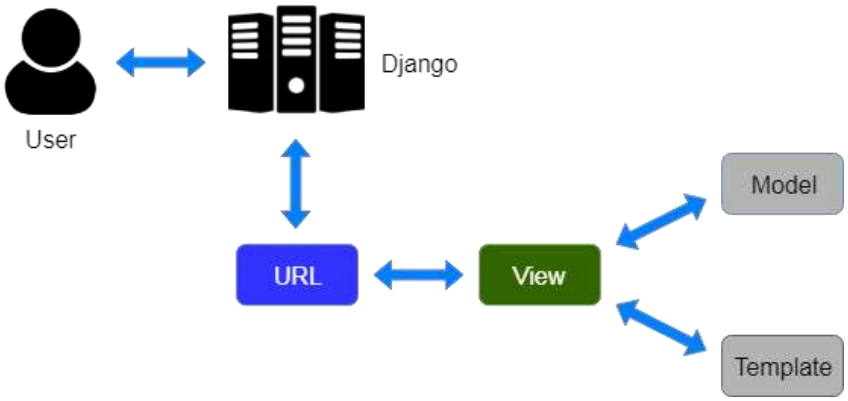
REST nima? REST yoki vakillik holatini o‘tkazish - bu kompyuter tizimlari o‘rtasida standartlarni ta’minlash uchun arxitektura uslubi. MDN Web docs buni quyidagicha tushuntiradi: RESTning asosiy g‘oyasi shundan iboratki, resurs, masalan, hujjat yaxshi tan olingan, tilga agnostik va ishonchli standartlashtirilgan mijoz/server o‘zaro ta’siri orqali uzatiladi. Xizmatlar ushbu cheklovlarga rioya qilganda RESTful deb hisoblanadi. Ushbu uslubdan so‘ng, frontend/mijoz tomoni va backend/server tomonidagi kod bir-biridan mustaqil ravishda, qarama-qarshi xizmatning ishlashini o‘zgartirmasdan amalga oshirilishi mumkin. Turli foydalanuvchilar bir xil REST so‘nggi nuqtalarini urishlari, bir xil harakatlarni bajarishlari va bir xil javoblarni olishlari mumkin. REST tizimi doirasida mijozlar yoki brauzerlar ma’lumotlarni olish yoki tahrirlash uchun so‘rov yuboradilar va server so‘rovni oladi va javobni qaytaradi. REST tizimidagi ma’lumotlar nuqtasi resurs bilan o‘zaro ishlash uchun ishlatiladigan 4 ta HTTP funksiyalari qaysi operatsiyani bajarish kerakligini belgilaydi: GET, POST (yangi resurs yaratish), PATCH/PUT (resursni tahrirlash), DELETE.

REST API (shuningdek, RESTful API nomi bilan ham tanilgan) – REST(API) arxitektura uslubining cheklovlariga mos keladigan amaliy dasturlash interfeysi. Ushbu APIlarda yo‘llar mijozga ularning so‘rovi (ya'ni ma'lum bir ma’lumot resurs qismini topish) tufayli nima sodir bo‘layotganini tushunishga yordam berish uchun mo‘ljallangan. Brauzer foydalanuvchi tomonidan RESTful API orqali so‘rov yuborilgandan so‘ng, u so‘rovchiga resurs holatining ko‘rinishini uzatadi. Ushbu ma’lumot HTTP orqali bir nechta formatlardan birida, odatda JSON (Javascript Object Notation) da taqdim etiladi.

Django REST Framework (DRF). REST APIlarini noldan yaratish vaqt va kuch talab etadi, ammo Django API yaratishni tezroq boshlash uchun funksionallikni ta’minlaydigan o‘rnatiladigan kengaytmani taklif qiladi.

MVT (Model View Templetes) dasturiy ta’minotni loyihalash namunasidir. Bu model ko‘rinishi va shablonning uchta muhim komponenti to‘plamidir. Model ma’lumotlar bazasi bilan ishlashga yordam beradi. Bu ma’lumotlarga ishlov beradigan ma’lumotlarga kirish qatlami hisoblanadi. Garchi Django MVC namunasiga amal qilsada, lekin o‘zining konventsiyalarini saqlaydi. Shunday qilib, boshqaruv ramkaning o‘zi tomonidan boshqariladi.Alohida kontroller yo‘q va to‘liq dastur Model ko‘rinishi va shablonga asoslangan. Shuning uchun u **MVT ilovasi** deb ataladi.

MVT asosidagi boshqaruv oqimini tasvirlangan holati:



Bu yerda foydalanuvchi Djangoga resurs so‘raydi, Django boshqaruvchi sifatida ishlaydi va URL manzilidagi mavjud manbani tekshiradi. Agar URL xaritalar bo‘lsa, model va shablon bilan o‘zaro aloqada bo‘lgan ko‘rinish deyiladi. Django foydalanuvchiga javob qaytaradi va javob sifatida shablonni yuboradi.

Djangoning dizayn namunasi. Python ramkasi MVC - model, ko‘rinish, kontroller - dasturiy ta’minot dizayni naqshiga amal qiladi, chunki u odatda Ruby, Java, C, C++ va boshqa ko‘plab tillarda qo‘llaniladi. Asosiy mantiq amal qilsada, Django o‘rniga MVT - model, ko‘rinish, shablon - arxitektura, o‘rganish va tushunishga arziydigan ba’zi asosiy farqlarga tayanadi. MVC yoki MVT kabi dizayn arxitekturasiga ega bo‘lish ma’lumotlar taqdimotini ma’lumotlarni

o‘zaro ta’sir qiluvchi va qayta ishlaydigan komponentlardan ajratishga e’tibor beradi.

Model - ilovadagi ma’lumotlar uchun interfeys, u ma’lumotlar bazasi (ko‘pincha Postgres yoki MYSQL kabi relyatsion ma’lumotlar bazasi) bilan ifodalangan butun ilovaning mantiqiy tuzilishini ta’minlaydi.

View - brauzerdan HTTP so‘rovlarini qabul qiladi va HTTP javoblarini qaytaradi. U Model va Shablon o‘rtasidagi ko‘prik vazifasini bajaradi va ma’lumotlar bazasidan tegishli ma’lumotlarni olish asosida shablonni ko‘rsatadi.

Templetes - HTML kodi. React kabi frontend ramka/kutubxona bilan ishlashda muhim emas. MVC naqshida Shablon Ko‘rinish bilan solishtiriladi.

MVC namunasi bilan solishtirganda, Django ramkasining o‘zi MVC naqshida kontroller nima uchun javobgar bo‘lishini boshqaradi. MVT oqimining ushbu bosqichma-bosqich tushuntirishi buni yaxshi tushuntiradi:

1. Foydalanuvchi Djangoga resurs uchun URL so‘rovini yuboradi.
2. Keyin Django ramkasi URL resursini qidiradi.
3. Agar URL yo‘li Ko‘rinishga bog‘lansa, u holda o‘sha ko‘rinish chaqiriladi.
4. Keyin Ko‘rinish Model bilan o‘zaro ishlaydi va ma’lumotlar bazasidan tegishli ma’lumotlarni oladi.
5. Keyin ko‘rinish mos shablonni olingan ma’lumotlar bilan birga foydalanuvchiga qaytaradi.

Django modeli. Djangoda model - bu muhim maydonlar va usullarni o‘z ichiga olgan sinf. Har bir model sinfi ma’lumotlar bazasidagi bitta jadvalga mos keladi. Django Model django.db.models. modelning quyi sinfidir va model sinfining har bir maydoni ma’lumotlar bazasi maydonini(ustun) ifodalaydi. Django bizga xaritalangan jadvaldan yozuvni yaratish, olish, yangilash va o‘chirish imkonini beruvchi ma’lumotlar bazasi abstraktsiyasi APIsini taqdim

etadi. Model models.py faylida aniqlanadi. Ushbu fayl bir nechta modellarni o‘z ichiga olishi mumkin.

Misol ko‘rib chiqaylik, biz ikkita nom va familiya maydoniga ega bo‘lgan xodim modelini yaratamiz.

```
from django.db import models
class Employee(models.Model):
first_name = models.CharField(max_length=30)
last_name = models.CharField(max_length=30)
```

Birinchi ism va familiya maydonlari sinf atributlari sifatida belgilanadi va har bir atribut ma’lumotlar bazasi ustuniga mos keladi. Ushbu model ma’lumotlar bazasida quyidagi ko‘rinishdagi jadval yaratadi.

```
CREATE TABLE appname_employee (
"id" INT NOT NULL PRIMARY KEY,
"first_name" varchar(30) NOT NULL,
"last_name" varchar(30) NOT NULL
);
```

Yaratilgan jadvalda avtomatik yaratilgan id maydoni mavjud. Jadval nomi ilova nomi va model nomining kombinatsiyasi bo‘lib, keyinchalik o‘zgartirilishi mumkin.

Ro‘yxatdan o‘tish/Modeldan foydalanish. Modelni yaratgandan so‘ng, settings.py ichidagi INSTALLED_APPS ilovasida modelni ro‘yxatdan o‘tkazing.

```
INSTALLED_APPS = [
#...
'appname',
#...
]
```

Django model maydonlari. Model sinfida belgilangan maydonlar xaritalangan jadvalning ustunlar nomidir. Maydonlar nomi tozalash, saqlash yoki o‘chirish kabi python kalit so‘zlari bo‘lmasligi kerak. Django turli xil o‘rnatilgan maydon turlarini taqdim etadi (1-jadval).

1-jadval

Maydon nomi	Sinf	Izoh
Avtomatik maydon	Class AutoField (**variantlar)	Bu avtomatik ravishda ortib boruvchi IntegerField.
BigAutoField	BigAutoField sinfi (**variantlar)	Bu 64 bitli butun xuddi AutoFieldga o‘xshaydi, bundan tashqari u 1 dan 9223372036854775807 gacha bo‘lgan raqamlarga mos kelishi kafolatlanadi.
BigIntegerField	sinf BigIntegerField (**variantlar)	Bu IntegerField-ga o‘xshab 64 bitli butun sondir, bundan tashqari - 9223372036854775808 dan 9223372036854775807 gacha bo‘lgan raqamlar mos kelishi kafolatlanadi.
BinaryField	sinf BinaryField (**variantlar)	Ikkilik ma’lumotlarni saqlash uchun maydon.
BooleanField	sinf BooleanField (**variantlar)	Haqiqiy/noto‘g‘ri maydon. Ushbu maydon uchun standart CheckboxInput hisoblanadi.
CharField	class DateField(auto_now=False, auto_now_add=False, **variantlar)	Bu Pythonda datetime.date misoli bilan ifodalangan sana.
DateTimeField	class DateTimeField(auto_now=False, auto_now_add=False, **variantlar)	Bu Pythonda datetime.date misoli bilan ifodalangan sana.
DateTimeField	class DateTimeField(auto_now=False, auto_now_add=False, **variantlar)	U Pythonda datetime misoli bilan ifodalangan sana va vaqt uchun ishlatiladi.
DecimalField	class DecimalField(max_digits=Yo‘q, decimal_places=Yo‘q, **variantlar)	Bu Pythonda O‘nlik misol bilan ifodalangan aniqlikdagi o‘nlik sondir.
Davomiylik maydoni	sinf DurationField(**variantlar)	Vaqt davrlarini saqlash uchun maydon.
EmailField	sinf EmailField(max_length=254, **variantlar)	Bu qiymatning haqiqiy elektron pochta manzili ekanligini tekshiradigan CharField.
FileField	class FileField(upload_to=Yo‘q, max_length=100, **variantlar)	Bu faylni yuklash maydoni.
FloatField	FloatField sinfi (**variantlar)	Bu Pythonda float misoli bilan ifodalangan suzuvchi nuqtali raqam.
ImageField	class ImageField(upload_to=Yo‘q, height_field=Yo‘q, width_field=Yo‘q, max_length=100, **variantlar)	U FileField-dan barcha atributlar va usullarni meros qilib oladi, lekin yuklangan obyektning haqiqiy rasm ekanligini ham tasdiqlaydi.
IntegerField	sinf IntegerField(**variantlar)	Bu butun sonli maydon. -2147483648 dan 2147483647 gacha bo‘lgan qiymatlar Django tomonidan qo‘llab-quvvatlanadigan barcha ma’lumotlar bazalarida xavfsizdir.

NullBooleanField	NullBooleanField sinfi (**variantlar)	BooleanField kabi, lekin variantlardan biri sifatida NULL ga ruxsat beradi.
PositiveIntegerField	PositiveIntegerField sinfi (**variantlar)	IntegerField kabi, lekin ijobiy yoki nol (0) bo‘lishi kerak. 0 dan 2147483647 gacha bo‘lgan qiymatlar Django tomonidan qo‘llab-quvvatlanadigan barcha ma’lumotlar bazalarida xavfsizdir.
SmallIntegerField	Class SmallIntegerField(**variantlar)	Bu IntegerField-ga o‘xshaydi, lekin faqat ma'lum (ma’lumotlar bazasiga bog‘liq) nuqta ostidagi qiymatlarga ruxsat beradi.
TextField	class TextField (**variantlar)	Katta matn maydoni. Bu maydon uchun standart shakl vidjeti Textarea hisoblanadi.
Vaqt maydoni	sinf TimeField(auto_now=False, auto_now_add=False, **variantlar)	Pythonda datetime.time misoli bilan ifodalangan vaqt.

Nazorat savollari:

1. Django frameworki nima?
2. MVT texnologiyasi qanday ishlaydi?
3. Djangoda model qanday tuziladi va qanday ishlatiladi?
4. View funksiyalarini Djangoda qanday tuzish mumkin?
5. Django templating tili haqida qisqa ma’lumot bering.
6. Djangoda ishchi direktoriyalar nima?
7. Django routelash qanday ishlaydi?
8. Django frameworkida ma’lumotlar bazasi bilan ishlash qanday amalga oshiriladi?

5-§. DJANGODA MODELLAR TAYYORLASH.

Django ilovasini yaratish. Django modullilik falsafasiga mos kelish uchun loyiha doirasida blog veb-saytini yaratish uchun zarur bo‘lgan barcha fayllarni o‘z ichiga olgan Django ilovasini yaratamiz. Biz Python va Djangoda ishlashni boshlaganimizda, Python virtual muhitimizni faollashtirishimiz va ilovamizning asosiy katalogiga o‘tishimiz kerak. U yerda ushbu buyruqni bajaramiz:

```
python manage.py startapp blogsite
```

Bu bizning ilovamiz bilan birga katalogni yaratadi blogsite.

Darslikning bu qismida loyiha uchun quyidagi katalog tuzilishiga ega:

```
my_blog_app/
└── blog
    ├── blog
    │   ├── __init__.py
    │   ├── __pycache__
    │   │   ├── __init__.cPython38.pyc
    │   │   ├── settings.cPython38.pyc
    │   │   ├── urls.cPython38.pyc
    │   │   └── wsgi.cPython38.pyc
    │   ├── asgi.py
    │   ├── settings.py
    │   ├── urls.py
    │   └── wsgi.py
    ├── blogsite
    │   ├── __init__.py
    │   ├── admin.py
    │   ├── apps.py
    │   ├── migrations
    │   │   └── __init__.py
    │   ├── models.py
    │   ├── tests.py
    │   └── views.py
    └── manage.py
```

Biz e’tibor qaratadigan blogsite fayl katalogdagi models.py fayl bo‘ladi.

Xabarlar modelini qo‘shish. Avval biz models.py faylini ochishimiz va tahrirlashimiz kerak, shunda unda model yaratish uchun

Post kodi mavjud. Model Post quyidagi ma’lumotlar bazasi maydonlarini o‘z ichiga oladi:
title - Blog postining sarlavhasi.
slug - Veb-sahifalar uchun haqiqiy URL manzillar saqlanadigan va yaratilgan.
content - Blog postining matn mazmuni.
created_on - post yaratilgan sana.
Author - Xabarni yozgan odam.
models.py endi fayl joylashgan katalogga o‘tiladi.
cd ~/my_blog_app/blog/blogsite
Terminalingizda fayl mazmunini ko‘rsatish uchun buyruqdan foydalaning.
cat models.py: Fayl quyidagi kodga ega bo‘lishi kerak, u modellarni import qiladi va ushbu faylga nima joylashishini izoh bilan birga tavsiflaydi models.py.

```
#models.py
from django.db import models
# Create your models here.
```

Matn muharriridan foydalanib, models.py faylga quyidagi kodni qo‘shamiz. Biz nano matn muharriri sifatida foydalanamiz, lekin siz xohlagan matn muharriridan foydalanishingiz mumkin.

Ushbu fayl ichida API modellarini import qilish uchun kod allaqachon qo‘shilgan, biz davom etishimiz va keyingi sharhni o‘chirishimiz mumkin.

```
#models.py
from django.db import models
from django.template.defaultfilters import slugify
from django.contrib.auth.models import User
from django.urls import reverse
```

Keyin, biz chaqiradigan model sinfiga Post sinf usulini quyidagi ma’lumotlar bazasi maydonlari bilan qo‘shamiz: title, slug, content, created_onva author.

```
#models.py
...
class Post(models.Model):
    title = models.CharField(max_length=255)
    slug = models.SlugField(unique=True, max_length=255)
    content = models.TextField()
```

```
    created_on = models.DateTimeField(auto_now_add=True)
    author = models.TextField()
```

Keyinchalik, biz URL manzilini yaratish va postni saqlash funksiyasini qo‘shamiz. Bu juda muhim, chunki bu bizning noyob postimizga mos keladigan noyob havola yaratadi.

```
#models.py
...
    def get_absolute_url(self):
      return reverse('blog_post_detail', args=[self.slug])
    def save(self, *args, **kwargs):
        if not self.slug:
            self.slug = slugify(self.title)
        super(Post, self).save(*args, **kwargs)
```

Modelga postlarni qanday qilib buyurtma qilish va veb-sahifada ko‘rsatish kerakligini ko‘rib chiqamiz. Buning uchun, o‘rnatilgan ichki Meta sinfga qo‘shiladi. Meta sinfi odatda ma’lumotlar bazasi maydonini aniqlash bilan bog‘liq bo‘lmagan boshqa muhim model mantig‘ini o‘z ichiga oladi.

```
#models.py
...
    class Meta:
        ordering = ['created_on']
        def __unicode__(self):
            return self.title
```

Nihoyat, biz ushbu faylga Comment modelni qo‘shamiz.

Name - fikrni joylashtirgan shaxsning ismi.
Email - fikrni joylashtirgan shaxsning elektron pochta manzili.
Text - izoh matnining o‘zi.
Post - fikr bildirilgan post.
created_on - sharh yaratilgan vaqt.

```
#models.py
class Comment(models.Model):
    name = models.CharField(max_length=42)
    email = models.EmailField(max_length=75)
    website = models.URLField(max_length=200, null=True,
blank=True)
    content = models.TextField()
    post = models.ForeignKey(Post, on_delete=models.CASCADE)
```

```
    created_on = models.DateTimeField(auto_now_add=True)
```

Bu nuqtada models.py to‘liq bo‘ladi. models.py faylingiz quyidagilarga mos kelishiga ishonch hosil qiling :

models.py

```
from django.db import models
from django.template.defaultfilters import slugify
from django.contrib.auth.models import User
from django.urls import reverse
class Post(models.Model):
    title = models.CharField(max_length=255)
    slug = models.SlugField(unique=True, max_length=255)
    content = models.TextField()
    created_on = models.DateTimeField(auto_now_add=True)
    author = models.TextField()
    def get_absolute_url(self):
        return reverse('blog_post_detail', args=[self.slug])
        def save(self, *args, **kwargs):
        if not self.slug:
            self.slug = slugify(self.title)
        super(Post, self).save(*args, **kwargs)
    class Meta:
        ordering = ['created_on']
        def __unicode__(self):
            return self.title
class Comment(models.Model):
    name = models.CharField(max_length=42)
    email = models.EmailField(max_length=75)
    website = models.URLField(max_length=200, null=True,
blank=True)
    content = models.TextField()
    post = models.ForeignKey(Post, on_delete=models.CASCADE)
    created_on = models.DateTimeField(auto_now_add=True)
```

models.py faylni o‘rnatgandan so‘ng, biz settings.py faylimizni yangilashni davom ettirishimiz mumkin.

Sozlamalarni yangilash. Endi biz ilovamizga modellar qo‘shdik, endi biz blogsite qo‘shgan ilovaning mavjudligi haqida loyihamizga xabar berishimiz kerak. INSTALLED_APPS buni settings.py bo‘limga

qo‘shamiz. Bu yerda faylingizni, masalan, nano bilan oching: nano settings.py

Fayl ochiq holda, quyida ko‘rsatilganidek, ilovangizni fayl bo‘limiga blogsite qo‘shing .settings.py

```
# Application definition
INSTALLED_APPS = [
    'blogsite',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Qo‘shilgan ilova bilan blogsite faylni saqlashingiz va undan chiqishingiz mumkin.

Migratsiyani amalga oshirish. Biz Post va Comment modellarini qo‘shganimizdan keyingi qadam bu o‘zgarishlarni qo‘llashdir, shunda bizning MySQL ma’lumotlar bazasi sxemasi ularni taniydi va kerakli jadvallarni yaratadi. Birinchidan, buyruq yordamida model o‘zgarishlarimizni individual migratsiya makemigrations commits fayllariga to‘plashimiz kerak. Ushbu fayllar Git kabi versiyalarni boshqarish tizimidagi fayllarga o‘xshaydi.

Migratsiyalarni qo‘shganimizdan keyin fayl o‘zgaradi. cd ~/my_blog_app/blog dan foydalanib blog katalogiga o‘ting.

Keyin terminal oynasida quyidagi chiqishni olishingiz kerak:

Output

```
Migrations for 'blogsite':
  blogsite/migrations/0001_initial.py
    - Create model Post
    - Create model Comment
```

Agar faylda nima borligini o‘qishni istasangiz, u joylashgan katalogdan ishga tushirish kerak.

Migratsiya faylini yaratganimiz uchun, migrate buyruq yordamida ma’lumotlar bazasiga ushbu fayllar tavsiflangan o‘zgarishlarni qo‘llashimiz kerak. Lekin avval showmigrations buyruq yordamida qaysi migratsiyalar mavjudligini tekshirib ko‘raylik.

```
python manage.py showmigrations
```

Output
admin
 [X] **0001**_initial
 [X] **0002**_logentry_remove_auto_add
 [X] **0003**_logentry_add_action_flag_choices
auth
 [X] **0001**_initial
 [X] **0002**_alter_permission_name_max_length
 [X] **0003**_alter_user_email_max_length
 [X] **0004**_alter_user_username_opts
 [X] **0005**_alter_user_last_login_null
 [X] **0006**_require_contenttypes_0002
 [X] **0007**_alter_validators_add_error_messages
 [X] **0008**_alter_user_username_max_length
 [X] **0009**_alter_user_last_name_max_length
 [X] **0010**_alter_group_name_max_length
 [X] **0011**_update_proxy_permissions
blogsite
 [ ] **0001**_initial
contenttypes
 [X] **0001**_initial
 [X] **0002**_remove_content_type_name
sessions
 [X] **0001**_initial

Siz va 0001_initial modellari bilan Post Comment yaratilganidan tashqari barcha migratsiyalar tekshirilganini sezasiz.

Endi quyidagi buyruq yordamida migratsiyalarni amalga oshirganimizdan so‘ng qaysi bayonotlar bajarilishini tekshiramiz. U migratsiya va migratsiya nomini dalil sifatida oladi:

python manage.py sqlmigrate blogsite 0001_initial

Quyida sahna ortida amalga oshirilayotgan haqiqiy SQL so‘rovi ko‘rsatilgan.

Output

```
--
-- Create model Post
--
CREATE TABLE `blogsite_post` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `title` varchar(255) NOT NULL, `slug`
```

varchar(255) NOT NULL UNIQUE, `content` longtext NOT NULL, `created_on` datetime(6) NOT NULL, `author` longtext NOT NULL);
--
-- Create model Comment
--
CREATE TABLE `blogsite_comment` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `name` varchar(42) NOT NULL, `email` varchar(75) NOT NULL, `website` varchar(200) NULL, `content` longtext NOT NULL, `created_on` datetime(6) NOT NULL, `post_id` integer NOT NULL);
ALTER TABLE `blogsite_comment` ADD CONSTRAINT `blogsite_comment_post_id_de248bfe_fk_blogsite_post_id`
FOREIGN KEY (`post_id`) REFERENCES `blogsite_post` (`id`);
Endi python manage.py migrate MySQL ma’lumotlar bazasiga qo‘llanilishi uchun migratsiyalarni amalga oshiramiz.
Biz quyidagi chiqishni olamiz:
Output
Operations to perform:
  Apply all migrations: admin, auth, blogsite, contenttypes, sessions
Running migrations:
  Applying blogsite.0001_initial... OK
Migratsiyalar muvaffaqiyatli bajarildi.

Django hujjatlarida ko‘rsatilganidek, MySQL bilan Django migratsiyasiga uchta ogohlantirish mavjudligini yodda tutish kerak. Sxemani o‘zgartirish bu operatsiyalar atrofidagi tranzaktsiyalarni qo‘llab-quvvatlamaslik hisoblanadi. Boshqacha qilib aytganda, agar migratsiya muvaffaqiyatli qo‘llanilmasa, boshqa ko‘chirishga urinish uchun kiritilgan o‘zgarishlarni qo‘lda olib tashlashingiz kerak bo‘ladi. Muvaffaqiyatsiz ko‘chirishda biron bir o‘zgartirish kiritilgunga qadar oldingi nuqtaga qaytish mumkin emas. Ko‘pgina sxemalarni o‘zgartirish operatsiyalari uchun MySQL jadvallarni to‘liq qayta yozadi. Eng yomon holatda, vaqtning murakkabligi ustunlarni qo‘shish yoki olib tashlash uchun jadvaldagi qatorlar soniga mutanosib bo‘ladi. MySQLda ustunlar, jadvallar va indekslar uchun nom uzunligi bo‘yicha kichik cheklovlar mavjud. Bundan tashqari, barcha ustunlar va indeks qopqoqlarining birlashtirilgan hajmi bo‘yicha cheklov mavjud. Ba’zi boshqa backendlar Djangoda yaratilgan yuqori chegaralarni qo‘llab-quvvatlasada, bir xil indekslar MySQL backend bilan yaratilmaydi.

Django bilan foydalanishni ko‘rib chiqayotgan har bir ma’lumotlar bazasi uchun har birining afzalliklari va kamchiliklarini ko‘rib chiqish kerak bo’ladi.

Ma’lumotlar bazasi sxemasini tekshirish. Migratsiya tugallangandan so‘ng, biz Django modellarimiz orqali yaratgan MySQL jadvallarini muvaffaqiyatli yaratishni tekshirishimiz kerak. Buning uchun MySQLga kirish uchun terminalda quyidagi buyruqni bajaring.

mysql blog_data -u djangouser

Endi blog_data ma’lumotlar bazasini tanlang.

DATABASES.SQL da ko‘rsatishingiz mumkin.

USE blog_data;

Keyin jadvallarni ko‘rish uchun quyidagi buyruqni kiriting.

SHOW TABLES;

Ushbu SQL so‘rovi quyidagilarni ko‘rsatishi kerak:

Output

```
+--------------------------+
| Tables_in_blog_data      |
+--------------------------+
| auth_group               |
| auth_group_permissions   |
| auth_permission          |
| auth_user                |
| auth_user_groups         |
| auth_user_user_permissions |
| blogsite_comment         |
| blogsite_post            |
| django_admin_log         |
| django_content_type      |
| django_migrations        |
| django_session           |
+--------------------------+
```

Jadvallar orasida blogsite_comment va blogsite_post mavjud. Bu biz o‘zimiz yaratgan modellardir. Ularda biz belgilagan maydonlar mavjudligini tasdiqlaylik.

DESCRIBE blogsite_comment;

Output

```
+------------+-------------+------+-----+---------+---------------+
```

```
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int          | NO   | PRI | NULL    | auto_increment |
| name       | varchar(42)  | NO   |     | NULL    |                |
| email      | varchar(75)  | NO   |     | NULL    |                |
| website    | varchar(200) | YES  |     | NULL    |                |
| content    | longtext     | NO   |     | NULL    |                |
| created_on | datetime(6)  | NO   |     | NULL    |                |
| post_id    | int          | NO   | MUL | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

DESCRIBE blogsite_post;

Output

```
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int          | NO   | PRI | NULL    | auto_increment |
| title      | varchar(255) | NO   |     | NULL    |                |
| slug       | varchar(255) | NO   | UNI | NULL    |                |
| content    | longtext     | NO   |     | NULL    |                |
| created_on | datetime(6)  | NO   |     | NULL    |                |
| author     | longtext     | NO   |     | NULL    |                |
+-----------+--------------+------+----+--------+-------------+
6 rows in set (0.00 sec)
```

Biz ma’lumotlar bazasi jadvallari Django model migratsiyalarimizdan muvaffaqiyatli yaratilganligini tasdiqladik. Django va Python muhitini tark etishga tayyor bo‘lgach, deactivate buyruqni bajarishingiz mumkin:

Deactivate: Dasturlash muhitini o‘chirib qo‘yish sizni terminalning buyruq satriga qaytaradi.

Xulosa qilib aytganda, ushbu mavzuda blog veb-ilovasining asosiy funksiyalari uchun modellar muvaffaqiyatli qo‘shildi. Bunda modelda kodlashni, qanday ishlashini va Djangoni haqiqiy ma’lumotlar bazasi MySQL jadvallariga migrations tarjima qilish jarayoni yoritib berildi.

Nazorat savollari

1. Djangoda modellar nima?
2. Djangoda model tayyorlash uchun qanday jarayonlar o‘tkaziladi?
3. Modelga doimiy ma’lumotlar qo‘shish uchun avvalgi qadamlar qanday?
4. Djangoda model maydonlarining turli turlari nima?
5. Modeldagi ma’lumotlarni bazaga saqlash uchun qanday yo‘l?
6. Djangoda modelega validatorlarni qanday qo‘shish mumkin?
7. Model orqali ma’lumotlarni o‘qish va yangilash uchun metodlar mavjudmi?
8. Boshqaruv panelida modelga chiroyli interfeys yaratish uchun qanday qo‘llanma mavjud?

6-§. DJANGODA TEMPLATESLAR VA JINJA2 FORMATI.

Veb-ramka sifatida Django HTMLni dinamik ravishda yaratish uchun qulay usulga muhtoj. Eng keng tarqalgan yondashuv shablonlarga tayanadi. Shablon kerakli HTML chiqishining statik qismlarini hamda dinamik tarkib qanday kiritilishini tavsiflovchi maxsus sintaksisni o‘z ichiga oladi. Shablonlar bilan HTML sahifalarini yaratishni yuqoridagi mavzularda ko‘rib o‘tgan edik.

Django loyihasi bir yoki bir nechta shablon dvigatellari bilan sozlanishi mumkin. Django o‘zining Django shablon tili (DTL) deb nomlanuvchi shablon tizimi va mashhur alternativ Jinja2 uchun o‘rnatilgan backendlarni yuboradi. Boshqa shablon tillari uchun backendlar uchinchi tomonlardan mavjud bo‘lishi mumkin.

Django serverdan qat'iy nazar shablonlarni yuklash va ko‘rsatish uchun standart APIni belgilaydi. Yuklash ma'lum identifikator uchun shablonni topish va uni oldindan qayta ishlashdan, odatda uni xotiradagi tasvirga kompilyatsiya qilishdan iborat. Renderlash shablonni kontekst ma’lumotlari bilan interpolyatsiya qilish va natijada olingan qatorni qaytarishni anglatadi.

Django shablon tili Djangoning shablon tizimidir. Django 1.8 ga qadar u yagona o‘rnatilgan variant edi. Bu juda yaxshi shablon kutubxonasi, garchi u o‘ziga xos fikrga ega va bir nechta o‘ziga xos xususiyatlarga ega. Agar sizda boshqa serverni tanlash uchun jiddiy sabab bo‘lmasa, DTL dan foydalanishingiz kerak, ayniqsa siz ulanadigan dastur yozayotgan bo‘lsangiz va shablonlarni tarqatmoqchi bo‘lsangiz django.contrib.admin kabi shablonlarni o‘z ichiga olgan Django ilovalari DTL dan foydalanadi. django.template shablon dvigatellari uchun umumiy qo‘llab-quvvatlash va Django shablon tilini amalga oshirish nomlar maydonida bo‘ladi.

Django shablonlari Django shablon tilidan foydalangan holda belgilangan matn hujjati yoki Python qatoridir. Ba’zi konstruktsiyalar shablon mexanizmi tomonidan tan olinadi va talqin qilinadi. Asosiylari o‘zgaruvchilar va teglardir.

Shablon kontekst bilan berilgan. Renderlash o‘zgaruvchilarni kontekstda qidiriladigan qiymatlari bilan almashtiradi va teglarni bajaradi. Qolgan hamma narsa avvalgidek chiqariladi. Django shablon tilining sintaksisi to‘rtta konstruktsiyani o‘z ichiga oladi.

O‘zgaruvchilar. O‘zgaruvchi kontekstdan qiymat chiqaradi, bu qiymatlarga kalitlarni xaritalash dictga o‘xshash obyektdir. O‘zgaruvchilar figurali qavslar bilan o‘ralgan:

My first name **is** {{ first_name }}. My last name **is** {{ last_name }}.

{'first_name': 'Ali', 'last_name': 'Aliyev'}

My first name **is** Ali. My last name **is** Aliyev.

Lug‘atni qidirish, atributlarni qidirish va ro‘yxat indekslarini qidirish nuqta belgisi bilan amalga oshiriladi:

{{ my_dict.key }}

{{ my_object.attribute }}

{{ my_list.**0** }}

Agar o‘zgaruvchi chaqiriladigan bo‘lsa, shablon tizimi uni hech qanday argumentsiz chaqiradi va chaqiriladigan o‘rniga uning natijasidan foydalanadi.

Teglar. Teglar kontentni chiqarishi, boshqaruv tuzilmasi bo‘lib xizmat qilishi, masalan, “if” iborasi yoki “for” sikli, ma’lumotlar bazasidan tarkibni olishi yoki hatto boshqa shablon teglariga kirishni yoqishi mumkin.

Teglar foizlar o‘rtasida yoziladi: **{% %}**

{% csrf_token %}

Aksariyat teglar argumentlarni qabul qiladi:

{% cycle 'odd' 'even' %}

Ba’zi teglar boshlang‘ich va tugatish teglarini talab qiladi:

{% **if** user.is_authenticated %}Hello, {{ user.username }}.{% endif %}

O‘rnatilgan teglar ma’lumotnomasi, shuningdek, maxsus teglarni yozish bo‘yicha ko‘rsatmalar mavjud.

Filtrlar. Filtrlar o‘zgaruvchilar va teg argumentlarining qiymatlarini o‘zgartiradi.

Ular shunday ko‘rinadi:

{{ django|title }}

ning konteksti bilan ushbu shablon quyidagilarga xizmat qiladi:

{'Django: 'the web framework for perfectionists with deadlines'}

The Web Framework For Perfectionists With Deadlines

Ba’zi filtrlar argument oladi:

{{ my_date|date:"Y-m-d" }}

O‘rnatilgan filtrlar ma’lumotnomasi, shuningdek , maxsus filtrlarni yozish bo‘yicha ko‘rsatmalar mavjud .

Fikrlar. Sharhlar quyidagicha ko‘rinadi:

```
{# this won't be rendered #}
```

Teg ko‘p qatorli sharhlarni taqdim etadi. {% comment %} django.template.Engine Django shablon tizimining namunasini qamrab oladi. To‘g‘ridan-to‘g‘ri yaratishning asosiy sababi Engine Django shablon tilini Django loyihasidan tashqarida ishlatishdir. django.template.backends.django.DjangoTemplatesdjango.template.Engine

Andoza. django.template.Template kompilyatsiya qilingan shablonni ifodalaydi. Shablonlar Engine.get_template() yoki Engine.from_string() bilan olinadi.

Kontekst. django.template.Context kontekst ma’lumotlariga qo‘shimcha ravishda ba’zi meta ma’lumotlarga ega. Template.render() shablonni ko‘rsatish uchun uzatiladi. django.template.RequestContextContext joriy ma’lumotlarni saqlaydigan va shablon kontekstli protsessorlarini boshqaradigan kichik HttpRequest sinfdir. Umumiy API ekvivalent tushunchaga ega emas. Kontekst ma’lumotlari tekislikda uzatiladi dict va agar kerak bo‘lsa, oqim HttpRequest alohida uzatiladi. Shablonlarni yuklovchilar Template shablonlarni joylashtirish, ularni yuklash va obyektlarni qaytarish uchun javobgardir. Django bir nechta o‘rnatilgan andoza yuklagichlarini taqdim etadi va maxsus shablon yuklagichlarini qo‘llab-quvvatlaydi.

Kontekst protsessorlari. Kontekst protsessorlari - bu oqimni HttpRequest argument sifatida qabul qiladigan va dictrenderlash kontekstiga qo‘shiladigan ma’lumotlarni qaytaradigan funksiyalardir. Ularning asosiy qo‘llanilishi har bir ko‘rinishda kodni takrorlamasdan kontekstga barcha shablonlar tomonidan umumiy ma’lumotlarni qo‘shishdir. Django ko‘plab o‘rnatilgan kontekst protsessorlarini taqdim etadi va siz o‘zingizning qo‘shimcha kontekst protsessorlaringizni ham amalga oshirishingiz mumkin.

Shablon dvigatellarini qo‘llab-quvvatlash. Konfiguratsiya. Shablonlar dvigatellari TEMPLATES sozlamalar bilan sozlangan. Bu har bir dvigatel uchun bittadan konfiguratsiyalar ro‘yxatidir.

```
TEMPLATES = [
    {
```

```
        "BACKEND":
"django.template.backends.django.DjangoTemplates",
        "DIRS": [],
        "APP_DIRS": True,
        "OPTIONS": {
            # ... some options here ...
        },
    },
]
```

BACKEND - bu Django shablonining backend APIni amalga oshiradigan shablon dvigatel sinfiga nuqtali Python yo‘lidir.

Ko‘pgina dvigatellar shablonlarni fayllardan yuklaganligi sababli, har bir dvigatel uchun yuqori darajadagi konfiguratsiya ikkita umumiy sozlamalarni o‘z ichiga oladi:

- **DIRS** qidiruv tizimi shablon manba fayllarini qidirishi kerak bo‘lgan kataloglar ro‘yxatini belgilaydi.
- **APP_DIRS** vosita o‘rnatilgan ilovalar ichida shablonlarni izlashi kerakligini aytadi. Har bir backend shablonlari saqlanishi kerak bo‘lgan ilovalar ichidagi pastki katalog uchun an'anaviy nomni belgilaydi.

Kamdan kam bo‘lsada, bir xil backendning bir nechta nusxalarini turli xil variantlar bilan sozlash mumkin. Bunday holda, har bir **NAME** dvigatel uchun o‘ziga xoslikni belgilashingiz kerak .

OPTIONS backend uchun maxsus sozlamalarni o‘z ichiga oladi.

Foydalanish. Modul **django.template.loader** shablonlarni yuklash uchun ikkita funksiyani belgilaydi.

get_template(template_name , yordamida = Yo‘q)

Bu funksiya shablonni berilgan nom bilan yuklaydi va Template obyektni qaytaradi. Qaytish qiymatining aniq turi shablonni yuklagan backendga bog‘liq. Har bir backend o‘z sinfiga ega. Template. get_template() har bir shablon dvigatelini muvaffaqiyatli bo‘lgunga qadar tartibda sinab ko‘radi. Agar shablon topilsa, lekin noto‘g‘ri sintaksisga ega bo‘lsa, u TemplateSyntaxError ni qaytaradi. Shablonlarni qanday qidirish va yuklash har bir dvigatelning konfiguratsiyasiga bog‘liq.

select_template(template_name_list , yordamida = Yo‘q)

select_template()ga **get_template()** o‘xshaydi, faqat shablon nomlari ro‘yxatini oladi. U har bir nomni tartibda sinab ko‘radi va mavjud bo‘lgan birinchi shablonni qaytaradi. Agar shablonni yuklash

muvaffaqiyatsiz tugasa, belgilangan quyidagi ikkita istisno **django.template** qaytarilishi mumkin:
istisno TemplateDoesNotExist (*xabar , sinab ko‘rilgan = Yo‘q , backend = Yo‘q , zanjir = Yo‘q*)

Ushbu istisno shablon topilmasa paydo bo‘ladi. U disk raskadrovka sahifasida o‘chirilgandan keyingi shablonni to‘ldirish uchun quyidagi ixtiyoriy argumentlarni qabul qiladi:

Backend: istisno paydo bo‘lgan shablon backend misoli.

Tried: shablonni topishda sinab ko‘rilgan manbalar ro‘yxati. Bu o‘z ichiga olgan kortejlar ro‘yxati sifatida formatlangan.

Qidiruv algoritmiga misol keltiramiz:

```
TEMPLATES = [
    {
        "BACKEND":
"django.template.backends.django.DjangoTemplates",
        "DIRS": [
            "/home/html/example.com",
            "/home/html/default",
        ],
    },
    {
        "BACKEND": "django.template.backends.jinja2.Jinja2",
        "DIRS": [
            "/home/html/jinja2",
        ],
    },
]
```

Agar siz get_template('story_detail.html')ga murojaat qilsangiz, Django quyidagi tartibda fayllarni qidiradi:

- /home/html/example.com/story_detail.html('Djangodvigatel)
- /home/html/default/story_detail.html('Djangodvigatel)
- /home/html/jinja2/story_detail.html('Jinja2dvigatel)

Agar siz select_template(['story_253_detail.html', 'story_detail.html']) murojaat qilsangiz , Django faylni quyidagicha qidiradi:

- **/home/html/example.com/story_253_detail.html('Django dvigatel)**
- **/home/html/default/story_253_detail.html**(**'Django** dvigatel)

- **/home/html/jinja2/story_253_detail.html**('**Jinja2** dvigatel)
- **/home/html/example.com/story_detail.html**('**Django** dvigatel)
- **/home/html/default/story_detail.html**('**Django** dvigatel)
- **/home/html/jinja2/story_detail.html**('**Jinja2** dvigatel)

Django mavjud shablonni topganda, u qidirishni to‘xtatadi.

django.template.loader.select_template(): Ko‘proq moslashuvchanlik uchun foydalaniladi. select_template() dan moslashuvchan shablonni yuklash uchun foydalanishingiz mumkin . Misol uchun, agar siz yangilik yozgan bo‘lsangiz va ba'zi hikoyalarda maxsus shablonlarga ega bo‘lishni istasangiz, shunga o‘xshash narsadan foydalaning. Bu sizga shaxsiy hikoya uchun maxsus shablondan foydalanishga imkon beradi. Shablonlarni o‘z ichiga olgan har bir katalog ichidagi pastki kataloglarda shablonlarni tashkil qilish mumkin va afzalroqdir. Konventsiya har bir Django ilovasi uchun pastki katalog yaratishdan iborat bo‘lib, kerak bo‘lganda o‘sha pastki kataloglar ichida kichik kataloglar mavjud. Barcha shablonlarni bitta katalogning ildiz darajasida saqlash tartibsiz bo‘ladi. Pastki katalogdagi shablonni yuklash uchun slashdan foydalaning, masalan:

get_template("news/story_detail.html")

Yuqoridagi kabi bir xil **TEMPLATES** operatsiyadan foydalanib, bu quyidagi andozalarni yuklashga harakat qiladi:

- /home/html/example.com/news/story_detail.html('Djangodvigatel)
- /home/html/default/news/story_detail.html('Djangodvigatel)
- /home/html/jinja2/news/story_detail.html('Jinja2dvigatel)

Bundan tashqari, shablonlarni yuklash va ko‘rsatishning takroriy holatini kamaytirish uchun Django jarayonni avtomatlashtiradigan yorliq funksiyasini taqdim etadi.

render_to_string(shablon_nomi , kontekst = Yo‘q , so‘rov = Yo‘q , foydalanish = Yo‘q)

render_to_string() kabi get_template() shablonni yuklaydi va render() darhol uning usulini chaqiradi. Bu quyidagi dalillarni oladi.

template_name yuklash va ko‘rsatish uchun shablon nomi hisoblanadi. Agar bu shablon nomlari ro‘yxati bo‘lsa, Django shablonni topish uchun select_template() o‘rniga get_template()dan foydalanadi.

Misol:

```
from django.template.loader import render_to_string
rendered = render_to_string("my_template.html", {"foo": "bar"})
```

Nihoyat, biz to‘g‘ridan-to‘g‘ri tuzilgan dvigatellardan foydalanishimiz mumkin:
Engines: Shablon dvigatellari quyidagilarda
mavjud **django.template.engines**:

```
from django.template import engines
django_engine = engines["django"]
template = django_engine.from_string("Hello {{ name }}!")
```

Qachon **APP_DIRS True** bo‘lsa , dvigatellar o‘rnatilgan ilovalarning pastki katalogidan **Jinja2** shablonlarni qidiradi. Eng muhim kirish - **OPTIONS** bu **'environment'**. Bu Jinja2 muhitini qaytaradigan chaqiruvga olib boradigan nuqtali Python yo‘lidir. Bu standart 'jinja2.Environment' hisoblanadi. Django ushbu chaqiruvni chaqiradi va boshqa variantlarni kalit so‘z argumentlari sifatida uzatadi. Bundan tashqari, Django bir nechta variant uchun Jinja2dan farq qiladigan standart sozlamalarni qo‘shadi:

- **'autoescape':True**
- **'loader'DIRS**: va uchun sozlangan yuklagich **APP_DIRS**
- **'auto_reload':settings.DEBUG**
- **'undefined':DebugUndefined if settings.DEBUG else Undefined**

Jinja2 OPTIONS dvigatellari ham quyidagilarni qabul qiladi:
'context_processors': shabloni so‘rov bilan ko‘rsatilganda kontekstni to‘ldirish uchun foydalaniladigan chaqiriladiganlarga nuqtali Python yo‘llari ro‘yxatidir. Ushbu chaqiriladiganlar so‘rov obyektini o‘z argumenti sifatida qabul qiladi va **dict** kontekstga birlashtiriladigan elementlarni qaytaradi.
Bu bo‘sh ro‘yxatga sukut bo‘yicha **Jinja2** shablonlari bilan kontekst protsessorlaridan foydalanish tavsiya etilmaydi. Kontekst protsessorlari Django shablonlari bilan foydalidir, chunki Django shablonlari argumentlar bilan chaqiruv funksiyalarini qo‘llab-quvvatlamaydi. Jinja2da bunday cheklov yo‘qligi sababli, kontekst protsessor sifatida foydalanadigan jinja2.Environment funksiyani quyida tavsiflanganidek shablonda mavjud bo‘lgan global o‘zgaruvchilarga qo‘yish tavsiya etiladi. Keyin shablonda ushbu funksiyani chaqirishingiz mumkin:

```
{{ function(request) }}
```

Ba’zi Django shablonlari kontekst protsessorlari belgilangan qiymatni qaytaradi. Jinja2 shablonlari uchun bu bilvosita qatlam kerak emas, chunki siz to‘g‘ridan-to‘g‘ri jinja2.Environmentga doimiylarni

qo‘shishingiz mumkin. Jinja2 uchun kontekst protsessorlarini qo‘shish uchun asl foydalanish holati:

- ✓ so‘rovga bog‘liq qimmat hisob-kitob qilish;
- ✓ har bir shablonda natija kerak;
- ✓ natijani har bir shablonda bir necha marta ishlatish.

Agar ushbu shartlarning barchasi bajarilmasa, shablonga funksiyani o‘tkazish Jinja2 dizayniga ko‘proq mos keladi. Standart konfiguratsiya maqsadli ravishda minimal darajada saqlanadi. Agar shablon so‘rov bilan taqdim etilsa (masalan, render()dan foydalanilganda), backend kontekstga Jinja2 global request, csrf_input va csrf_tokenni qo‘shadi. Djangoga xos APIlardan foydalanish uchun ularni muhitda sozlashingiz kerak.

Misol:

```
from django.templatetags.static import static
from django.urls import reverse
from jinja2 import Environment
def environment(**options):
    env = Environment(**options)
    env.globals.update(
        {
            "static": static,
            "url": reverse,
        }     )
    return env
```

Keyin Jinja2 shablonlarida quyidagi konstruksiyalardan foydalanishingiz mumkin:

```
<img src="{{ static('path/to/company-logo.png‘) }}" alt="Company Logo">
<a href="{{ url('admin:index') }}">Administration</a>
```

Teglar va filtrlar tushunchalari Django shablon tilida ham, Jinja2 da mavjud, ammo ular boshqacha qo‘llaniladi. Jinja2 shablonlarda chaqiriladiganlarga argumentlarni uzatishni qo‘llab-quvvatlaganligi sababli, yuqoridagi misolda ko‘rsatilganidek, Django shablonlarida shablon yorlig‘i yoki filtrni talab qiladigan ko‘plab funksiyalarga Jinja2 shablonlarida funksiyani chaqirish orqali erishish mumkin. Jinja2 global nom maydoni shablon kontekst protsessorlariga bo‘lgan ehtiyojni yo‘q qiladi. Django shablon tilida Jinja2 testlarining ekvivalenti yo‘q.

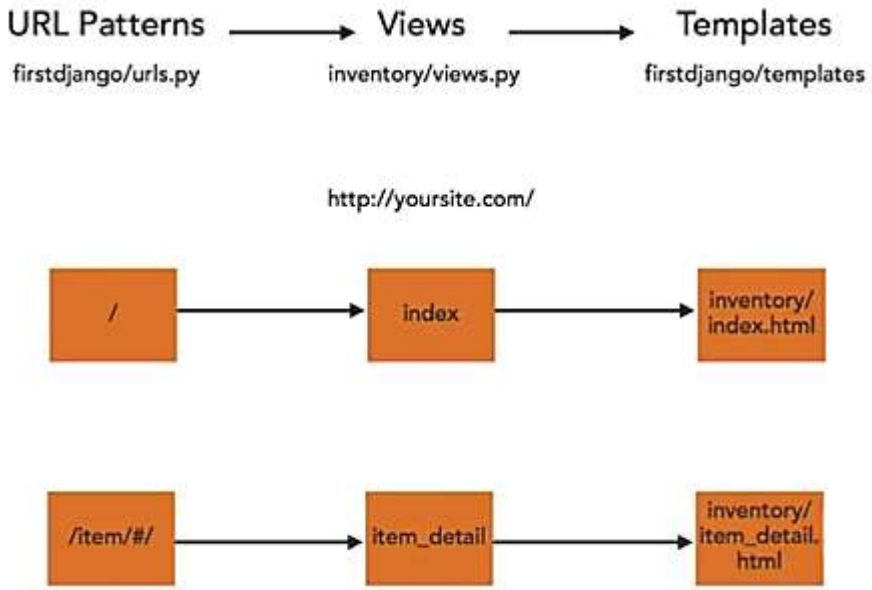
Nazorat savollari:

1. Django templateslari nima?
2. Jinja2 formati qanday ishlaydi?
3. Djangoda template fayllarini qayerga saqlash mumkin?
4. Jinja2 syntaxida qanday ma’lumotlarni chiqarish mumkin?
5. Django templatega o‘zgaruvchilar va ma’lumotlar qanday o‘tkaziladi?
6. Jinja2da shartlar (if-else) qanday ishlaydi?
7. Jinja2da sikl operatori (for loop) qanday ishlaydi?
8. Django template larni ishlatib, web sahifalarni qanday o‘zgartirish mumkin?

7-§. DJANGODA VIEWSLAR BILAN ISHLASH.

models.py - har bir ko‘rinish faylda biz aniqlagan modellardan models.py so‘rov qilish va kerak bo‘lganda ma’lumotlar bazasi bilan ishlash uchun foydalanishi mumkin.

Shablonlar - har bir ko‘rinish HTML qanday ko‘rinishda bo‘lishini ko‘rsatish qatlamiga yordam berish uchun mos keladigan shablonga tayanadi. Har bir shablon qo‘shimcha shablon sintaksisi bilan HTMLdan iborat alohida fayldir.



Qo‘shimcha qism - bu **Django formasi** foydalanuvchilar kiritgan ma’lumotlarni tekshirish va uni Python obyektlariga aylantirish usulidir. Djangoning shakl funksionalligi tekshirish va tozalash ishlarining katta qismini xavfsiz tarzda soddalashtirishi va avtomatlashtirishi mumkin. Shaklni modelga bog‘lashingiz mumkin. Shaklni modelga bog‘lashning afzalligi shundaki, o‘zgartirishlar kiritilganda shakl avtomatik ravishda yangilanadi, shakl avtomatik ravishda ma’lumotlarni tekshirishi va saqlashi mumkin.

forms.py- asosiy ko‘rinishlarga ma’lumot qo‘shadigan foydalanuvchi ma’lumot kiritishi mumkin bo‘lgan shakllarni yaratadi.

Ko‘rinishlarni amalga oshirish. Fayl **view.py** veb-saytingiz uchun “tarkib” vazifasini bajaradi. U veb-saytda qaysi sahifalar bo‘lishini va har bir veb-sahifaga uzatiladigan ma’lumotlarning mantiqiyligini belgilaydi. Djangoda veb-sahifalar va kontent ko‘rishlar bo‘yicha yetkazib beriladi.

Django ko‘rinishining ikki xil turi mavjud: *funksiyaga asoslangan ko‘rinishlar va sinfga asoslangan ko‘rinishlar*. Funksiya

ko‘rinishlari juda oddiy. Biroq, sinf ko‘rinishlari ko‘proq funksionallikni keltirib chiqaradi va ko‘plab o‘rnatilgan Django funksiyalaridan foydalanadi.

Funksiyaga asoslangan ko‘rinishlar

Boshlang‘ich views.py fayli quyidagicha ko‘rinadi:

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.
```

Ko‘rsatilishi kerak bo‘lgan har bir shablon uchun siz kerakli ma’lumotlarni uzatuvchi va shablonga yo‘naltiruvchi funksiyani qo‘shasiz. Xabarli bitta sahifali juda oddiy views.py fayli quyidagicha ko‘rinishi mumkin:

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.
def index(request):
    return HttpResponse("Hello World!")
```

Ko‘proq kontent qo‘shish orqali ko‘rishlar soniga ko‘proq qo‘shishimiz mumkin. Inventarzatsiya modelidan ma’lumotlarni uzatuvchi yanada murakkab views.py fayli quyidagicha ko‘rinadi:

```
from django.shortcuts import render
from django.http import Http404
from inventory.models import Item
def index(request):
    items = Item.objects.exclude(amount=0)
    return render(request, 'inventory/index.html', {
        'items': items,
    })

def item_detail(request, id):
    try:
        item = Item.objects.get(id=id)
    except Item.DoesNotExist:
        raise Http404('This item does not exist')
    return render(request, 'inventory/item_detail.html', {
        'item': item,
    })
```

Funksiya item_detail sahifada so‘ralgan element uchun item_detail.html element tafsilotlarini ko‘rsatish uchun tuzilgan. E'tibor bering, agar kiritilgan element raqami mavjud bo‘lmasa, unda ba'zi istisnolar mavjud.

Sinfga asoslangan ko‘rishlar. O‘rnatilgan va foydalanishga tayyor bo‘lgan sinfga asoslangan ko‘rinishlarning har xil turlari mavjud. Djangoning o‘rnatilgan ko‘rinishlariga ba’zi misollar:

- **ListView klassi** - Belgilangan model uchun barcha obyektlar ro‘yxatini ko‘rsatadigan ko‘rinish.
  - **ListView** klassi qidiradigan standart shablon - bu ko‘rinish ishlatilayotgan modelning nomi `model_list.html`. `model`
- **DetailView klassi** - bitta model namunasi maydonlarini o‘z ichiga olgan ko‘rinish.
  - **DetailView** klassi qidiradigan standart shablon - bu ko‘rinish orqali kiradigan modelning nomi `model_detail.html`. `model`
- **CreateView klassi - obyektni *yaratish*** , tasdiqlash xatolari (agar mavjud bo‘lsa) bilan shaklni qayta ko‘rsatish va obyektni saqlash uchun shaklni ko‘rsatadigan ko‘rinish.
  - **CreateView** klassi qidiradigan standart shablon - bu ko‘rinish ishlatilayotgan modelning nomi `model_form.html`. `model`
- **UpdateView klassi - Mavjud obyektni *tahrirlash*** , tasdiqlash xatolari bilan shaklni qayta ko‘rsatish (agar ular mavjud bo‘lsa) va obyektdagi o‘zgarishlarni saqlash uchun shaklni ko‘rsatadigan ko‘rinish.
  - **UpdateView** klassi qidiradigan standart shablon - bu ko‘rinish ishlatilayotgan modelning nomi `model_form.html`. `model`
- **DeleteView klassi** - Tasdiqlash sahifasini ko‘rsatadigan va keyin mavjud obyektni ***o‘chiradigan*** ko‘rinish.
  - **DeleteView** klassi qidiradigan standart shablon - bu ko‘rinish ishlatilayotgan modelning nomi `model_confirm_delete.html`. `model`

Django o‘rnatilgan sinfga asoslangan ko‘rinishlarining to‘liq ro‘yxati uchun hujjatlarga qaraladi. Misol quyida ko‘rsatilgan:

```
from django.shortcuts import render
from django.views.generic import (View, TemplateView, ListView,
DetailView,CreateView, UpdateView, DeleteView)
from . import models
```

```
# CLASS BASED VIEWS
# IndexView
class IndexView(TemplateView):
    template_name = 'index.html'
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['injectme'] = 'BASIC INJECTION!'
        return context
# List View
class SchoolListView(ListView):
    # Set the called object to 'schools'; default is 'school_list'
    context_object_name = 'schools'
    model = models.School

# Detail View
class SchoolDetailView(DetailView):
    # Set the called object to 'school_detail'; default is 'school'
    context_object_name = 'school_detail'
    model = models.School
    template_name = 'basic_app/school_detail.html'

# Update an existing School
class SchoolUpdateView(UpdateView):
    # Define which fields can be updated by the user
    fields = ('name', 'principal')
    # Define the model that this UpdateView is tied to
    model = models.School

# Delete an existing school
class SchoolDeleteView(DeleteView):
    model = models.School
    success_url = reverse_lazy("basic_app:list")
```

URL namunalarini yangilash kerak. urls.py faylda Djangoga views.py fayldan (yoki ilova ichidagi boshqa joylardan) kelayotgan sahifa so‘rovlarini qanday yo‘naltirishini ko‘rsatuvchi URL namunalari mavjud.

Ilova ko‘rinishlari faylini import qilish

URL namunalari to‘g‘ri ishlashi uchun urls.py modul va marshrutlashda ishtirok etuvchi boshqa modul o‘rtasida muvofiqlashtirish bo‘lishi kerak. Birinchi qadam bu modullarni import qilishdir.

Quyida turli ko‘rinish turlarini bog‘lash uchun ushbu qadamlarning qisqacha tavsifi keltirilgan:

- **Funksiya ko‘rinishlari:**

➢ Importga qo‘shing: `from my_app import views`

➢ URL namunalariga URL qo‘shing: `url('', views.home, name='home')`

- **Sinfga asoslangan ko‘rinishlar:**

➢ Import qo‘shing: `from other_app.views import Home`

➢ URL namunalariga URL qo‘shing: `url('', Home.as_view(), name='home')`

- **Boshqa URL konf faylini o‘z ichiga oladi:**

➢ include() funksiyasini import qiling: `from django.conf.urls import url, include`

➢ URL namunalariga URL qo‘shing: `url('blog/', include('blog.urls'))`

URL namunalarini sozlash. Djangoda marshrutlash URLlarni Python usullariga moslashtiradi va bu usullarni Python sinflarida chaqiradi. Ro‘yxatdagi har bir URL namunasi to‘rtta parametrga ega:

1. So‘rovga mos keladigan URL namunasi mavjud marshrut.
2. Agar mos keladigan bo‘lsa, qo‘ng‘iroq qilish uchun ko‘rinish
3. Maqsadli ko‘rinishga o‘tkaziladigan kalit so‘z argumentlari yoki kvarglar**.**
4. Ushbu naqsh shablonlarda ko‘rsatiladigan nom.

Django 2.0 dan oldin marshrut muntazam ifodalar yordamida ifodalangan. Django 2.0 soddalashtirildi, shunda marshrutlar endi oddiy simli naqsh bilan ifodalanishi mumkin.

Format bu `url('route_pattern', destination_method, name = 'optional_name')`.

```
from django.conf.urls import include, url
from django.contrib import admin
urlpatterns = [
    url('', views.index, name='index'),
    url('info/', views.info, name='info‘),
```

]

Yuqoridagi misolda, agar kimdir urlsite.comURLda boshqa hech narsa bo‘lmagan holda kirsa, u **indeks** ko‘rinishiga o‘tadi. Agar kiritilgan URL o‘z ichiga olgan bo‘lsa, u **ma’lumot** /info/ ko‘rinishiga o‘tadi.

Muhim: *url elementlarining tartibi muhim, chunki urlpattern funksiya yuqoridan boshlanadi va moslikni topgunga qadar belgilangan barcha URL manzillar bo‘ylab aylanadi. Hech qanday moslik topilmasa, u bizni 404 sahifasiga yuboradi.*

Quyida yana bir misol:

```
# urls.py file
from django.conf.urls import include, url
from django.contrib import admin
from inventory import views

urlpatterns = [
    url('', views.index, name='index'),
    url('item/<int:pk>/', views.item_detail, name='item_detail'),
    url('admin/', include(admin.site.urls)),
]
# view.py file
from django.shortcuts import render
from django.http import HttpResponse
def index(request):
    return HttpResponse('<p>In index view</p>')
def item_detail(request, id):
    return HttpResponse('<p>In item_detail view with id
{0}</p>'.format(id)
```

“Indeks” deb nomlangan url namunasi **indeks** funksiyasiga va “item.detail” url namunasi fayldagi **item_**detail views.py funksiyasiga qanday tegishli ekanligiga e’tibor bering.

Agar modulda *True* DEBUG ga o‘rnatilgan bo‘lsa, foydalanuvchi aniqlanmagan URL manziliga o‘tishga harakat qilganda Django sahifasi topilmaganini ko‘rasiz. Agar *False* settings.py DEBUGga o‘rnatilgan bo‘lsa, siz kutgan oddiy 404 sahifani ko‘rasiz.

Oddiy ifodalar. Ilgari Django URL-manzillarni sharhlash uchun faqat oddiy iboralardan foydalangan. Django 2.0 joriy etilishi bilan ular

endi talab emas, balki variant hisoblanadi. Siz hali ham muntazam iboralarni ishlatishingiz mumkinligi sababli, ushbu bo‘lim ma’lumot olish uchun kiritilgan.

Muntazam ifoda - bu satrlarni qidirish tartibini belgilaydigan belgilar ketma-ketligidir.

Muhim: Django bilan bir nechta qoidalarni yodda tutish kerak:

1. Oddiy iboralar katta-kichik harflarga sezgir!
2. Django mos keladigan birinchi naqshni tanlaydi, shuning uchun, birinchi navbatda aniqroq naqshlar ro‘yxatga kiritilganligiga ishonch hosil qiling.

MOS TRINGLAR:

- Ducky satrning istalgan joyida “ducky” so‘ziga mos keladi.
- ^admin/ “admin/” bilan boshlanadigan har qanday satrga mos keladi, lekin satrning boshqa joyida admin so‘ziga mos kelmaydi.
- suffix$ “qo‘shimchasi” so‘zi bilan tugaydigan har qanday satrga mos keladi, chunki $ ketma-ketlikning oxirini anglatadi.
- ^$ bo‘sh satrga mos keladi, chunki u o‘rtasida hech narsa bilan boshlanmaydi va tugamaydi.

Mos keladigan naqshlar:

Raqamlar:

- \d bitta raqamli belgiga mos keladi. Bu “d” belgisi bilan hech qanday aloqasi yo‘q. Bu raqamni anglatadi va teskari qiyshiq chiziq har qanday raqam belgisini anglatadi;
- \d\d yoki \d{x} elementlarning x soniga mos keladi;
- \d{1,2} elementlarning minimal va maksimal soniga mos keladi;
- \D raqam bo‘lmagan raqamga mos keladi;
- \d+ bir yoki bir nechta raqamli belgilarni bildiradi, shuning uchun bu satrning endi raqamlar bo‘lmagan nuqtasiga qadar istalgan raqamga mos keladi. Shunday qilib, 12 untsiya qatorida bu 1 va 2 belgilarga mos keladi;
- \d* nol yoki undan ortiq raqamni bildiradi;
- \d? raqam ixtiyoriy ekanligini bildiradi.

Raqamlar bilan aralashtirilgan so‘z belgilari:

- asdf\d{2} "asdf" harflaridan keyin ikkita raqamni bildiradi
  - asdf42 mos keladi;

- asdf42asdf42 mos kelmaydi;
- (asdf\d{2} "asdf" harflarini, keyin ikki marta ikki raqamni bildiradi,
- asdf42 mos kelmaydi;
- asdf42asdf42 mos keladi;

So‘z belgilari (alfa-raqam va "_"):

- \w "so‘z" belgisiga yoki alfa-raqamga mos keladi (az, AZ, 0-9, _)
- \W so‘z bo‘lmagan belgiga mos keladi

Parametrlar:

- ?P<name> parametr nomi ko‘rsatilganligini bildiradi (burchak qavslari ichida <>)

Diapazonlar:

- [] (kvadrat qavslar) diapazonlarni yaratishga imkon beradi.
  - [a-z] a dan z gacha kichik harflarni bildiradi.
  - [az] maxsus a yoki z degan ma'noni anglatadi.
  - [a-zA-Z] kichik harf a dan z gacha va katta harf A dan Z gacha degan ma’noni anglatadi.

URL manzillarini modulli qilish. Django loyihasi uchun standart tuzilmada loyiha papkasidagi asosiy faylda URL naqshlari mavjud bo‘lsada url.py, loyihani iloji boricha modulli saqlash eng yaxshi amaliyot hisoblanadi.

Buni URL namunalarimizdagi include() funksiyadan foydalanib qilishimiz mumkin. Asosan, loyiha darajasidagi asosiy url.py faylda har bir ilova ichidagi ilova url.py fayliga ishora qiluvchi URL namunasi mavjud. Ushbu ilova url.py fayllari faqat shu ilova ichidagi URL manzillari uchun URL namunalarini o‘z ichiga oladi, bu boshqa loyihalarda yana ishlatilishi mumkin bo‘lgan modulli tuzilmani ta'minlaydi.

Ushbu mantiqdan kelib chiqqan holda, **loyiha darajasidagi** urls.py fayl quyidagicha ko‘rinadi:

```
""" first_project URL Configuration """

from django.urls import path, include
from django.contrib import admin
```

```
from first_app import views
urlpattersn = [
    path('', views.index, name='index'),
    path('first_app/', include('first_app.urls')),
    path('admin/', admin.site.urls),
]
```

Va **dastur darajasidagi** `urls.py` fayl quyidagicha ko‘rinadi:

```
""" first_app URL configuration """
from django.urls import path
from first_app import views
urlpatterns = [
path('', views.index, name='index'),
]
```

Shablonlar va shablon komponentlari. Django shablonlari Django loyihalari uchun veb-sahifalardir. Django shablonlari kerakli HTML va CSS ning statik qismlarini hamda dinamik tarkibni qo‘shadigan ba’zi maxsus sintaksislarni o‘z ichiga oladi. Ko‘rinish renderni chaqirganda, u shablonga ma’lumotlarni uzatadi va shablon foydalanuvchiga ko‘rsatish uchun HTMLni yaratadi.

Loyiha tuzilishi. Django shablonlari alohida papkada saqlanadi. Odatda, loyiha ichidagi **har bir** `/templates/` ilova uchun pastki papkaga ega asosiy papka mavjud. Standart papka tuzilishi quyidagicha ko‘rinadi:

```
/templates
  /app1
  /app2
```

Oddiy Django loyihasi asosiy loyiha papkasi ostidagi andozalar papkasiga va dastur papkasi bilan bir xil darajada bo‘lishi mumkin. Biroq, ko‘plab ishlab chiquvchilar modulli va qayta foydalanish uchun narsalarni saqlash g‘oyasini saqlab qolish uchun har bir ilova ichida shablonlar papkasiga ega bo‘ladilar.

Shablon komponentlari. Django shablonlari sintaksisi to‘rt qismdan iborat:

- `{{ variable }}` - **O‘zgaruvchining** qiymati o‘zgaruvchi nomi qo‘sh figurali qavs ichida ishlatilganda ko‘rsatiladi.

- {% tag %} – **Shablon yorlig‘i** figurali qavslar ichida foiz bilan olinadi va for loop, ifs, strukturaviy elementlar va boshqa funksiyalar bilan ishlatiladi.
- {{ variable|filter }} - **Shablon filtri** o‘zgaruvchilar va teg argumentlarining qiymatlarini o‘zgartirish uchun ishlatilishi mumkin.
- {% comment %} - **Sharh yorlig‘i** ko‘p qatorli sharhlarni taqdim etadi.

Django shablon tilidan foydalanishdan maqsad kelajakdagi o‘zgarishlarni osonlashtirishdir. Misol uchun, URLni qattiq kodlashdan ko‘ra, shablon tegidan foydalanib, biz URL namunasini keyinroq o‘zgartirishimiz mumkin va shablonlarimizda ishlatadigan havolalar hali ham to‘g‘ri bo‘ladi.

O‘rnatilgan shablon teglarining to‘liq ro‘yxati va ularning funksiyalarini Django hujjatlaridagi o‘rnatilgan andoza teglari sahifasida topish mumkin.

Shablon meros. Shablon komponentlariga kirishdan oldin shablon merosini muhokama qilishimiz kerak. Django shablonini meros qilib olish bilan siz shablonlar qatlamlarini yaratishingiz mumkin, bunda ota-shablon veb-saytdagi barcha sahifalar uchun kod va uslubni o‘z ichiga oladi va bolalar shablonlari faqat shu sahifaga tegishli ma’lumotlarni olib keladi.

- Masalan, asosiy navigatsiya, veb-sayt CSS havolalari yoki asosiy fayl bo‘lishi mumkin.
- Shuningdek, bizda uslubga ega bo‘lgan va veb-saytda ishlatiladigan barcha foydalanuvchi sahifalari uchun tashkil etilgan foydalanuvchi bazasi fayli bo‘lishi mumkin. Asosiy fayl uning ota-onasi bo‘ladi.
- Keyin login.html va logout.html kabi har bir foydalanuvchi sahifasi uchun turli shablonlarni ko‘ramiz. Foydalanuvchi bazasi fayli ushbu andozalar uchun asosiy fayl bo‘ladi.
- Bizning katalogimiz quyidagicha ko‘rinishi mumkin:

/templates
/users
base_user.html
login.html
logout.html
base_home.html

Shablon teglari. Shablon teglari - bu HTML shabloningizda qo‘shimcha ishlarni bajarishga imkon beruvchi kichik yorliqlar. Ba’zilari mantiqiy ishlaydi, ba’zilari esa veb-sahifangizda ishlatilishi mumkin bo‘lgan tashqi ma’lumotlarni yuklaydi.

EXTENDS teg. Kengaytma yorlig‘i ushbu shablonning asosiy shablonni kengaytirishini bildiradi. Undan asosiy shablonning nisbiy yo‘li va nomini berish orqali foydalaniladi.

```
{% extends "base.html %}
```

Teg yo‘l shablonni yuklovchining ildiz katalogiga nisbatan ekanligini taxmin qiladi. Sizning katalogingiz shunday ko‘rinadi deb faraz qiling:

```
templates/
   template.html
   base2.html
   app1/
      base3.html
base1.html
```

Ushbu yo‘llarning barchasi to‘g‘ri bo‘ladi:

```
{% extends "./base2.html" %}
{% extends "../base1.html" %}
{% extends "./app1/base3.html" %}
```

LOAD teg. Yana bir keng tarqalgan qo‘llaniladigan shablon tegi **yuklash** yorlig‘idir. Bu teg shablondan foydalanish uchun maxsus shablon tegini yuklaydi.

```
{% load library library2 %}
```

Ko‘pincha siz foydalanilgan statik teglar kutubxonasini ko‘rasiz .

```
{% load static %}
```

Global prognoz bilan siz foydalanilayotgan xalqaro teglar va filtrlarni ham ko‘rasiz.

```
{% load static i18n %}
```

BLOK yorlig‘i. Takrorlashni kamaytirish uchun Django loyihalari har bir shablonda ishlatiladigan elementlar, masalan, metateglar, har qanday global CSS yoki JavaScript va navigatsiya paneli kabi strukturaviy elementlardan iborat asosiy shablonni amalga oshiradi. Asosiy shablonni veb-saytdagi barcha sahifalar qanday ko‘rinishini ko‘rsatadigan kontur sifatida ko‘rish mumkin.

Blok yorlig‘i turli sahifalarda har xil bo‘lishi mumkin bo‘lgan muayyan komponentlar uchun “ushlab turish joyini” belgilash imkonini beradi**.** Bu shunday ko‘rinadi:

```
{% block <name> %}
{% endblock <name> %}
```

Blok tegidan foydalanishning asosiy sababi kodingizni DRY saqlashdir. Siz asosiy blokingiz atrofida barcha uslublar va tuzilmalarni moslashtirishingiz mumkin, keyin boshqa shablonlarda blokga murojaat qilishingiz mumkin. Shunday qilib, siz butun saytingizda `<div>`, … va hokazo va uslublar sinfini faqat bir marta belgilaysiz. `<ul>` quyidagi misolda, blok belgilangan sinf va uslub atributlariga ega bo‘lganligi sababli, “**content** `<div>`” deb nomlangan blok ichidagi hamma narsa bir xil formatlanadi: ko‘k matnli konteynerda markazlashtirilgan.

```
<div class="container center" style="color:blue;">
  {% block content%}{% endblock content%}
</div>
```

Bu qanday ishlaydi? Tushuntirish uchun `<title>` elementidan foydalanamiz. Asosiy shablon `<head>` bo‘limida joylashgan `<title>` elementi foydalanuvchilar saytga kirganlarida brauzerining yorlig‘ida nimani ko‘rishini boshqaradi. Ba’zi saytlar buni faqat bir marta belgilaydi va o‘z saytidagi har bir sahifa uchun bir xil sarlavhani ko‘rsatadi. Biroq, kerak bo‘lganda, ushbu sarlavhani o‘zgartirishingiz mumkin. Django blok tegidan foydalanib, siz ushbu qiymatni har bir sahifada ko‘rsatilishi mumkin bo‘lgan o‘zgaruvchiga aylantirishingiz mumkin.

Quyidagi misolda, fayl sahifaning `base.html` bo‘limida ushbu qatorni o‘z ichiga oladi va keyingi sahifalar ularning mazmuniga mos keladigan sarlavhani o‘zgartiradi

```
<!-- base.html -->
<title>{% block title %}MySite{% endblock title %}</title>

<!-- about.html -->
{% extends base.html %}
{% block title %}About{% endblock title %}
```

Endi saytga foydalanuvchining brauzer yorlig‘i bosh sahifada “Mening saytim” va “haqida” sahifasida “haqida” ko‘rsatiladi.

Blok yorlig‘i o‘zgaruvchiga o‘xshash ishlaydi. Buning farqi shundaki, siz faqat bitta qiymatdan ko‘ra kod bloklarini kiritishingiz mumkin. Yana bir farq shundaki, siz bloklarni bloklarga joylashtirishingiz mumkin.

Barcha veb-sayt sahifalari bir xil tuzilishga ega va ma’lum elementlarni (masalan, yon panel) almashadi, lekin foydalanuvchi sahifalarida blog sahifalariga qaraganda kamroq elementlar bo‘lishi mumkin.

Misol:

```
<!-- website base (base.html) -->
...
<div class="breadcrumbs">
  <ul>
    {% block breadcrumbs %}
      <li><a href="{% url 'home' %}">Home</a></li>
      {% block app-crumbs %}{% endblock app-crumbs %}
    {% endblock breadcrumbs %}
  </ul>
</div>
...
<!-- user application base (user_base.html) -->
{% extends base.html %}
...
  {% block app-crumbs %}
    <li><a href="{% url 'users:list' %}">Users</a></li>
    {% block page-crumbs %}{% endblock page-crumbs %}
  {% endblock app-crumbs %}
...
<!-- page within user application -->
  {% extends users/user_base.html %}
...
  {% block page-crumbs %}
    <li><a href="{% url 'users:login' %}">Login</a></li>
  {% endblock page-crumbs %}
...
```

Mulohazalar

1. Blok **yorlig‘i** {% endblock %} nomini talab qilmaydi, lekin men ularni eng yaxshi amaliyot sifatida belgilashni afzal ko‘raman. Bu o‘qishni osonlashtiradi, ayniqsa sizda bir nechta va ichki bloklar mavjud bo‘lsa.
2. Blok yorlig‘i nafaqat boshqa shablonlarga kod qo‘shish imkoniyatini beradi, balki siz kodni olib tashlashingiz ham mumkin. Agar sizda asosiy shablonda blok tegi bo‘lsa, lekin uni ma’lum bir sahifada ko‘rishni istamasangiz, uni {% block %} va {% endblock %} iboralari orasiga hech narsa qo‘ymasdan kiritishingiz mumkin. Bu tegning sahifadan olib tashlanishiga olib keladi, chunki u Djangoga blokni hech narsa bilan almashtirishni aytadi.

Quyidagi misolda asosiy sahifa va haqida sahifada sahifa sarlavhasi bo‘ladi, lekin rasm galereyasida bo‘lmaydi.

```
<!-- base.html -->
{% block page-title %}
  <h1>My Site</h1>
{% endblock page-title %}
<!-- about.html -->
{% extends base.html %}
{% block page-title %}
  <h1>About</h1>
{% endblock page-title %}
```

Agar kontentni qayta-qayta takrorlayotganingizni sezsangiz, **blok tegidan** foydalanishni o‘ylab ko‘ring. Bu veb-sahifalaringiz tartibini formatlash va tartibga solishni oson va muammosiz qiladi.

Shaxsiy shablon teglari. Django yuqorida muhokama qilingan turli xil o‘rnatilgan shablon teglarini taklif qiladi. Biroq, Django sizga maxsus harakatlarni amalga oshirish uchun o‘z shablon teglarini yaratishga ham imkon beradi. Shaxsiy shablon teglari shablonlaringizga Django shablon teglarining asosiy to‘plamiga kirmaydigan funksiyalarni qo‘shish kerak bo‘lganda juda qulay bo‘ladi.

Django o‘z shablon teglarini osongina yaratish imkonini beruvchi ba’zi yordamchi funksiyalarni taqdim etadi.

- **Simple_tag** - ma’lumotlarni qayta ishlaydi va oqimni qaytaradi. Endi natijalarni saqlash ham mumkin.

- **Inclusion_tag** - ma’lumotlarni qayta ishlaydi va lug‘atni o‘z ichiga olgan ko‘rsatilgan shablonni qaytaradi.

Shablon teglari Django ilovalari ichida bo’lishi kerak. Ularni to‘g‘ri sozlash uchun quyidagilarni bajaring:

1. Ilova templatetags/ papkasida chaqiriladigan katalog yarating.
2. Unga bo‘sh `init.py` fayl qo‘shing.
3. Xuddi shu papkada boshqa fayl yarating va unga `<app-name>_tags.py` deb nom bering . Quyida blog saytida chop etilgan postlar sonini qaytaradigan shablon tegini yaratuvchi fayl namunasi keltirilgan.

```
# /blog/templatetags/blog_tags.py
from django import template
from ..models import Post
register = template.Library()
@register.simple_tag
def total_posts():
    return Post.published.count()
```

HTML shablonlarida foydalanish uchun avval shablon tegini yuklang, so‘ngra unga bodydagi teg bilan kiring.

```
{% load blog_tags %}
{% load staticfiles %}
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>{% block title %}{% endblock title %}</title>
  </head>
  <body>
    <h2>My blog</h2>
    <p>
      This is my blog.
      I've written {% total_posts %} posts so far.
    </p>
  </body>
</html>
```

Filtrlar. Shablon filtridan foydalanishdan maqsad chiqishni oddiy formatlash imkonini berishdir. Bu bizga tavsifni qanday

kiritilganligidan qat’iy nazar, har doim bir xil formatda ko‘rsatish imkonini beradi.

Misol uchun, quyidagi buyruq kichik harf bilan kiritilgan yoki kiritilmaganidan qat’iy nazar, har bir so‘z uchun bosh harflar bilan elementning sarlavhasini ko‘rsatadi.

```
<!-- In inventory/item_detail.html -->
<h3>{{ item.title|capfirst }}</h3>
```

O‘rnatilgan filtrlarning to‘liq ro‘yxatini ko‘rish uchun Django hujjatlaridagi o‘rnatilgan filtr ma’lumot sahifasiga qarang.

Sana filtri. Sana filtri shablonlarda mashhur formatlash vositasidir. Quyida eng ko‘p ishlatiladigan format satrlari keltirilgan. Batafsil ma’lumot uchun o‘rnatilgan sana formatlarini ko‘ring (2-jadval) .

Misol:

```
{{ value|date:"D d M Y" }} <!-- Note: Capitalization is important! -->
```

2-jadval

Belgini formatlash	**Tavsif**	**Misol chiqarish**
a	'am' yoki 'pm'	"man"
A	"AM" yoki "PM".	'AM'
d	Oyning kuni, bosh nol bilan 2 ta raqam.	'01' dan '31' gacha
D	Haftaning kuni, matnli, 3 ta harf.	"Jum"
F	Oy, matnli, uzun.	"yanvar"
g	Bosh nolsiz soatlik, 12 soatlik format.	"1" dan "12" gacha
h	Soatlik, 12 soatlik format.	"01" dan "12" gacha
H	Soat, 24 soat formati.	'00' dan '23' gacha
i	Daqiqalar.	'00' dan '59' gacha
I	Yozgi vaqt, amal qiladimi yoki yo‘qmi.	'1' yoki '0'
j	Nolsiz oy kuni.	'1' dan '31' gacha
l	Haftaning kuni, matnli, uzun.	'juma'
L	Bu kabisa yili bo‘ladimi-yo‘qligi uchun mantiqiy.	To‘g‘ri yoki noto‘g‘ri
m	Oy, bosh nol bilan 2 ta raqam.	"01" dan "12" gacha
M	Oy, matnli, 3 ta harf.	"yanvar"
n	Boshidagi nollarsiz oy.	"1" dan "12" gacha

Belgini formatlash	Tavsif	Misol chiqarish
P	Vaqt, 12 soatlik soatlar, daqiqalar va 'am'/'pm', agar ular nolga teng bo‘lsa, qolgan daqiqalar va tegishli bo‘lsa, "yarim tun" va "peshin" maxsus qatorlari. Xususiy kengaytma.	'soat 1', '13:30', 'yarim tun', 'peshin', '12:30'
s	Soniyalar, bosh nol bilan 2 ta raqam.	'00' dan '59' gacha
S	Oy kuni uchun inglizcha tartib qo‘shimchasi, 2 belgi.	"st", "nd", "rd" yoki "th"
t	Belgilangan oydagi kunlar soni.	28 dan 31 gacha
T	Ushbu mashinaning vaqt mintaqasi.	'EST', 'MDT'
w	Hafta kuni, bosh nolsiz raqamlar.	"0" (yakshanba) dan "6" (shanba)
V	ISO-8601 haftalik yil soni, haftalar dushanba kuni boshlanadi.	1, 53
y	Yil, 2 ta raqam.	'99'
Y	Yil, 4 ta raqam.	'1999'
z	Yil kuni.	0 dan 365 gacha

Uzunlik filtri. Tez-tez ishlatiladigan filtr `length` filtrdir. Uzunlik filtri yordamida siz qiymat uzunligini qaytarishingiz mumkin, bu satr, ro‘yxat va hokazolarda foydalanish mumkin.

```
{{ value|length }}
```

Filtrni kesish. Foydali ikkita filtr - bu `truncatechars` va `truncatewords` filtrlar. Ushbu filtrlar argumentni qabul qiladi va faqat satr o‘zgaruvchisi tarkibidagi birinchi juda ko‘p belgilar yoki so‘zlarni ko‘rsatishga imkon beradi.

```
{{ value|truncatechars:4 }} <!-- Only shows the first four characters. -->
{{ value|truncatewords:4 }} <!-- Only shows the first four words. -->
```

Shablon sozlamalarini yangilash. Birinchi ko‘rinish va shablon(lar)ni yaratgandan so‘ng, keyingi qadam o‘zgaruvchidagi elementni o‘z ichiga olgan ro‘yxatga andozalar topiladigan katalogni qo‘shish orqali o‘zgaruvchi uchun `settings.py` faylini yangilashdir.

Eslatma: *Agar shablonlar bir nechta kataloglarda tuzilgan bo‘lsa, har bir katalog mos ravishda settings.py faylida sozlanishi kerak.*

Esda tutingki, bu sizning andozalar papkangiz joylashgan joyga qarab o‘rnatilishi kerak.

```
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': ['app1/templates', app1/templates, ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
 'django.template.context_processors.debug‘,
 'django.template.context_processors.request',
 'django.contrib.auth.context_processors.auth',
 'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Shablonning barcha ajoyib komponentlari, teglari va filtrlari bilan shabloningizda narsalarni amalga oshirish jozibador, ammo bu aslida ilovangizni sekinlashtiradi.

HTML shablonlaringizdagi `{% for %}` sikllar va `{% if %}` bayonotlarni ishlatish o‘rniga, ushbu ishlov berishni python kodingizga o‘tkazing va modellar, shakllar va/yoki ko‘rinishlaringizda funksiyalar yarating.

Umumiy konventsiyalar

- Shablon nomlarida, blok nomlarida va shablonlardagi boshqa nomlarda chiziqcha (`_`) ustiga pastki chiziq qo‘ying. Ko‘pchilik Django foydalanuvchilari ushbu konventsiyaga rioya qilishadi. Nega? Xo‘sh, chunki Python obyektlari nomlarida pastki chiziqqa ruxsat berilgan, lekin tire taqiqlangan.
- Oxirgi blokga blok tegining nomini kiriting. Hech qachon faqat {% endblock %} yozmang, butun `{% endblock javascript %}` ni kiriting .
- Boshqa shablonlar tomonidan chaqirilgan shablonlarga prefiks qo‘shiladi va ko‘pincha "qisman" deb ataladi. Bu {% include %} orqali

chaqiriladigan andozalar yoki maxsus shablon `{% extends %}` teglari uchun amal qiladi.

Nazorat savollari:

1. Djangoda viewslar nima?
2. Views funksiyasini qanday tuzish mumkin?
3. Django CBV (Class-Based Views) nima?
4. Class-Based Viewsni qanday tuzish mumkin?
5. Views funksiyasi qanday ma’lumotlarni qabul qilishi mumkin?
6. Class-Based Viewsda kichikroq kodni qanday yozish mumkin?
7. Djangoda views funksiyasidan ma’lumotlarni qaytarish qanday bo‘ladi?
8. Django viewsida ma’lumotlarni tekshirish (validation) qanday qo‘llaniladi?

8-§. DJANGODA SETTINGS SOZLAMALARI VA URLLAR.

Ilova uchun URL-manzillarni loyihalash uchun siz norasmiy ravishda **URLconf** (URL konfiguratsiyasi) deb nomlangan Python modulini yaratasiz. Bu modul sof Python kodi bo‘lib, Python funksiyalariga (sizning qarashlaringiz) URL yo‘l ifodalari o‘rtasidagi xaritadir. Ushbu xaritalash kerak bo‘lganda qisqa yoki uzoq bo‘lishi mumkin. U boshqa xaritalarga murojaat qilishi mumkin va bu sof Python kodi bo‘lgani uchun uni dinamik ravishda qurish mumkin.

Django shuningdek, URL-manzillarni faol tilga ko‘ra tarjima qilish usulini taqdim etadi.

Django so‘rovni qanday ko‘rib chiqadi? Agar foydalanuvchi Djangoda ishlaydigan saytingizdan sahifa so‘rasa, tizim qaysi Python kodini bajarishni aniqlash uchun quyidagi algoritmga amal qiladi:

1. Django foydalanish uchun URLconf modulini aniqlaydi. Odatda, bu sozlamaning qiymati **ROOT_URLCONF**, lekin agar kiruvchi **HttpRequest** obyekt **urlconf** (o‘rta dastur tomonidan o‘rnatilgan) atributga ega bo‘lsa , uning qiymati **ROOT_URLCONF** sozlama o‘rniga ishlatiladi.
2. Django ushbu Python modulini yuklaydi va o‘zgaruvchini **urlpatterns** qidiradi. Bu **django.urls.re_path()** va/yoki **django.urls.path()** misollar ketma -ketligi bo‘lishi kerak.
3. Django har bir URL namunasi bo‘ylab tartibda ishlaydi va so‘ralgan URLga mos keladigan **path_info** birinchisida to‘xtaydi.
4. URL namunalaridan biri mos kelsa, Django Python funksiyasi (yoki sinfga asoslangan ko‘rinish) bo‘lgan berilgan ko‘rinishni import qiladi va chaqiradi. Ko‘rinish quyidagi argumentlardan o‘tadi:
   - **HttpRequest** ning misoli.
   - Agar mos keladigan URL namunasi nomli guruhlarni o‘z ichiga olmagan bo‘lsa, muntazam ifodadagi mosliklar pozitsion argumentlar sifatida taqdim etiladi.
   - Kalit so‘z argumentlari taqdim etilgan yo‘l ifodasi bilan mos keladigan har qanday nomlangan qismlardan iborat bo‘lib, yoki **kwargs**ga ixtiyoriy **django.urls.path()** argumentida ko‘rsatilgan har qanday **django.urls.re_path()** argumentlari tomonidan bekor qilinadi.
5. Hech qanday URL namunasi mos kelmasa yoki ushbu jarayonning istalgan nuqtasida istisno paydo bo‘lsa, Django tegishli xatolarni qayta ishlash ko‘rinishini chaqiradi.

Misol: URLconf namunasi:

```
from django.urls import path
from . import views
urlpatterns = [
    path("articles/2003/", views.special_case_2003),
    path("articles/<int:year>/", views.year_archive),
    path("articles/<int:year>/<int:month>/", views.month_archive),
path("articles/<int:year>/<int:month>/<slug:slug>/",
views.article_detail),
]
```

Eslatmalar:

- URL manzilidan qiymat olish uchun figurali qavslardan foydalaning.
- Qabul qilingan qiymatlar ixtiyoriy ravishda konvertor turini o‘z ichiga olishi mumkin. Masalan, butun son parametrini olish uchun **<int:name>dan** foydalaning. Agar konvertor kiritilmagan bo‘lsa, /belgidan tashqari har qanday satr mos keladi.
- Yetakchi chiziq qo‘shishning hojati yo‘q, chunki har bir URLda shunday bo‘ladi. Masalan, bu **articles**emas, balki **/articles**.

Misol so‘rovlar:

- So‘rov **/articles/2005/03/** ro‘yxatdagi uchinchi yozuvga mos keladi. Django funksiyani
- chaqiradi.**views.month_archive(request, year=2005, month=3)**
- **/articles/2003/**ro‘yxatdagi birinchi naqshga mos keladi, ikkinchisiga emas, chunki naqshlar tartibda tekshiriladi va birinchisi birinchi bo‘lib o‘tadi. Bu kabi maxsus holatlarni kiritish uchun buyurtmadan bemalol foydalaning. Bu yerda Django **views.special_case_2003(request)** funksiyani chaqiradi
- **/articles/2003**bu naqshlarning birortasiga mos kelmaydi, chunki har bir naqsh URL qiyshiq chiziq bilan tugashini talab qiladi.
- **/articles/2003/03/building-a-Djangosite/** yakuniy naqshga mos keladi. Django funksiyani chaqiradi . **views.article_detail(request, year=2003, month=3, slug="building-a-Djangosite")**

Yo‘lni o‘zgartirgichlar. Quyidagi yo‘l konvertorlari sukut bo‘yicha mavjud:

- **Str** - Har qanday bo‘sh bo‘lmagan qatorga mos keladi, yo‘l ajratuvchidan tashqari, '/'. Ifodaga konvertor kiritilmagan bo‘lsa, bu standart hisoblanadi.
- **Int** - Nolga yoki istalgan musbat songa mos keladi. a ni qaytaradi **int**.
- **Slug** - ASCII harflari yoki raqamlaridan, shuningdek defis va pastki chiziq belgilaridan tashkil topgan har qanday slug satrga mos keladi. Masalan, **building-your-1st-Djangosite**.
- **Uuid** - Formatlangan UUID ga mos keladi. Bitta sahifaga bir nechta URL-manzillar ko‘rsatilishiga yo‘l qo‘ymaslik uchun chiziqlar qo‘shilishi va harflar kichik bo‘lishi kerak. Masalan, **075194d3-6885-417e-a8a8-6c931e272f00**. Bir misolni qaytaradi **UUID**.
- **Path** - Har qanday bo‘sh bo‘lmagan qatorga, jumladan, yo‘l ajratuvchiga mos keladi '/'. Bu sizga URL yo‘lidagi kabi segmentga emas, balki to‘liq URL yo‘liga mos kelish imkonini beradi **str**.

Maxsus yo‘l konvertorlarini ro‘yxatdan o‘tkazish. Murakkab mos keladigan talablar uchun siz o‘zingizning yo‘l konvertorlaringizni belgilashingiz mumkin.

Konverter - bu quyidagilarni o‘z ichiga olgan sinf:

- Sinf **regex** atributi, qator sifatida.
- Mos keladigan satrni ko‘rish funksiyasiga o‘tkazilishi kerak bo‘lgan turga aylantirishni boshqaradigan usul . Berilgan qiymatni o‘zgartira olmasa, u ko‘tarilishi kerak . A mos kelmaydi deb talqin qilinadi va natijada, agar boshqa URL namunasi mos kelmasa, foydalanuvchiga 404 javobi yuboriladi. **to_python(self, value)ValueErrorValueError**
- Python turini URL manzilida ishlatiladigan satrga aylantirish usuli . Berilgan qiymatni o‘zgartira olmasa, u ko‘tarilishi kerak . A mos kelmaydi deb talqin qilinadi va natijada boshqa URL namunasi mos kelmaguncha ko‘tariladi

.to_url(self, value)ValueErrorValueErrorreverse()NoReverseMatch

Masalan:

```
class FourDigitYearConverter:
    regex = "[0-9]{4}"
    def to_python(self, value):
        return int(value)
    def to_url(self, value):
        return "%04d" % value
```

Quyidagi yordamida URLconf-da maxsus konvertor sinflarini ro‘yxatdan o‘tkazing **register_converter()**:

```
from django.urls import path, register_converter
from . import converters, views
register_converter(converters.FourDigitYearConverter, "yyyy")
urlpatterns = [
    path("articles/2003/", views.special_case_2003),
    path("articles/<yyyy:year>/", views.year_archive),
    ...,
]
```

Muntazam iboralardan foydalanish. Agar yo‘llar va konvertorlar sintaksisi URL namunalarini aniqlash uchun etarli bo‘lmasa, siz oddiy iboralardan ham foydalanishingiz mumkin. Buning uchun **re_path()**o‘rniga foydalaning **path()**.

Python muntazam iboralarida nomlangan muntazam ifoda guruhlari sintaksisi **(?P<name>pattern)**, bu yerda **name**guruh nomi va **pattern** mos keladigan namunadir.

Bu oddiy iboralar yordamida qayta yozilgan avvalgi URLconf misoli:

```
from django.urls import path, re_path
from . import views
urlpatterns = [
    path("articles/2003/", views.special_case_2003),
    re_path(r"^articles/(?P<year>[0-9]{4})/$", views.year_archive),
    re_path(r"^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/$",
views.month_archive),
    re_path(
        r"^articles/(?P<year>[0-9]{4})/(?P<month>[0-
9]{2})/(?P<slug>[\w-]+)/$",
        views.article_detail,
    ),
]
```

Bu avvalgi misol bilan taxminan bir xil narsani amalga oshiradi, bundan mustasno:

- Mos keladigan aniq URL manzillar biroz cheklangan. Misol uchun, 10000 yil endi mos kelmaydi, chunki yil butun sonlari to‘liq to‘rtta raqamdan iborat bo‘lishi bilan cheklanadi.

- Har bir olingan argument muntazam ifoda qanday mos kelishidan qat'i nazar, ko‘rinishga satr sifatida yuboriladi.

path() Foydalanishdan foydalanishga yoki teskarisiga o‘tishda **re_path()** ko‘rish argumentlarining turi o‘zgarishi mumkinligini bilish ayniqsa muhimdir va shuning uchun siz qarashlaringizni moslashtirishingiz kerak bo‘lishi mumkin.

Nomsiz oddiy ifoda guruhlaridan foydalanish. Nomlangan guruh sintaksisi kabi, masalan **(?P<year>[0-9]{4})**, qisqaroq nomsiz guruhdan ham foydalanishingiz mumkin, masalan **([0-9]{4})**.

Bu foydalanish ayniqsa tavsiya etilmaydi, chunki bu moslikning mo‘ljallangan ma'nosi va ko‘rinish argumentlari o‘rtasida tasodifiy xatolarni kiritishni osonlashtiradi.

Ikkala holatda ham, berilgan regex ichida faqat bitta uslubdan foydalanish tavsiya etiladi. Ikkala uslub aralashganida, nomsiz guruhlar e'tiborga olinmaydi va faqat nomli guruhlar ko‘rish funksiyasiga o‘tkaziladi.

Ichki argumentlar. Muntazam ifodalar ichki argumentlarga ruxsat beradi va Django ularni hal qiladi va ko‘rinishga o‘tkazadi. Orqaga o‘tishda Django barcha ichki olingan argumentlarni e'tiborsiz qoldirib, barcha tashqi olingan argumentlarni to‘ldirishga harakat qiladi. Ixtiyoriy ravishda sahifa argumentini oladigan quyidagi URL namunalarini ko‘rib chiqing:

```
from django.urls import re_path
urlpatterns = [
    re_path(r"^blog/(page-([0-9]+)/)?$", blog_articles),  # bad
    re_path(r"^comments/(?:page-(?P<page_number>[0-9]+)/)?$",
comments),  # good
]
```

Ikkala naqsh ham ichki argumentlardan foydalanadi va hal qiladi: masalan, ikkita pozitsion argumentga **blog/page-2/**mos keladi : va . Ikkinchi naqsh kalit so‘z argumenti 2 ga o‘rnatilgan bo‘ladi. Bu holatda tashqi argument tutib bo‘lmaydigan argumentdir .**blog_articlespage-2/2commentscomments/page-2/page_number(?:...)**

Ko‘rinishni **blog_articles**o‘zgartirish uchun eng tashqi olingan argument kerak **page-2/**yoki bu holda hech qanday argument yo‘q, lekin **comments**argumentsiz yoki qiymati bilan teskari o‘zgartirilishi mumkin **page_number**.

Ichki o‘rnatilgan argumentlar ko‘rish argumentlari va URL o‘rtasida kuchli bog‘lanish hosil qiladi **blog_articles**: ko‘rinish **page-2/**faqat ko‘rinishni qiziqtiradigan qiymat o‘rniga URLning bir qismini ( ) oladi. Bu bog‘lanish teskari o‘girilishda yanada aniqroq bo‘ladi, chunki teskari ko‘rinishda biz sahifa raqami o‘rniga URL qismini o‘tkazishimiz kerak.

Oddiy iboraga argument kerak bo‘lsa, lekin ko‘rinish uni e'tiborsiz qoldirganda, qoida tariqasida, faqat ko‘rinish ishlashi kerak bo‘lgan qiymatlarni oling va yozilmaydigan argumentlardan foydalaning.

URLconf nimani qidiradi

URLconf oddiy Python qatori sifatida so‘ralgan URL-ga qarshi qidiradi. Bunga GET yoki POST parametrlari yoki domen nomi kirmaydi.

Misol uchun, so‘rovda **https://www.example.com/myapp/**URLconf ni qidiradi **myapp/**.

So‘rovda **https://www.example.com/myapp/?page=3**URLconf qidiradi **myapp/**.

URLconf so‘rov usuliga qaramaydi. Boshqacha qilib aytganda, barcha so‘rov usullari - **POST**, **GET**, **HEAD**, va boshqalar - bir xil URL uchun bir xil funksiyaga yo‘naltiriladi.

Ko‘rish argumentlari uchun standart parametrlarni belgilash. Ko‘rinishlaringiz argumentlari uchun standart parametrlarni belgilash qulay hiyladir. Mana misol URLconf va ko‘rish:

```
# URLconf
from django.urls import path
from . import views
urlpatterns = [
    path("blog/", views.page),
    path("blog/page<int:num>/", views.page),
]
# View (in blog/views.py)
def page(request, num=1):
    # Output the appropriate page of blog entries, according to num.
```

Yuqoridagi misolda ikkala URL namunasi ham bir xil ko‘rinishga ishora qiladi – **views.page**– lekin birinchi naqsh URL manzilidan hech narsa olinmaydi. Agar birinchi naqsh mos kelsa, funksiya **page()**

o‘zining standart argumentini, uchun ishlatadi **num**. **1** Agar ikkinchi naqsh mos kelsa, olingan qiymatdan **page()**foydalaniladi .**num**

urlpatternsDjango birinchi marta kiritilgan ro‘yxatdagi muntazam ifodalarni qayta ishlaydi. Keyingi so‘rovlar URL hal qiluvchi orqali keshlangan konfiguratsiyadan foydalanadi.

urlpatterns o‘zgaruvchisi sintaksisi

urlpatterns va/yoki misollar ketma -ketligi bo‘lishi kerak: **path()re_path()**

Xatolik. Django so‘ralgan URL uchun moslikni topa olmasa yoki istisno paydo bo‘lganda, Django xatolarni qayta ishlash ko‘rinishini ishga tushiradi.

Ushbu holatlar uchun ishlatiladigan ko‘rinishlar to‘rtta o‘zgaruvchi bilan belgilanadi. Ularning standart qiymatlari ko‘pgina loyihalar uchun etarli bo‘lishi kerak, ammo ularning standart qiymatlarini bekor qilish orqali keyingi sozlash mumkin.

To‘liq tafsilotlar uchun xato ko‘rinishlarini sozlash bo‘yicha hujjatlarga qarang. Bunday qiymatlar URLconf ildizingizda o‘rnatilishi mumkin. Ushbu o‘zgaruvchilarni boshqa URLconf-da o‘rnatish hech qanday ta'sir qilmaydi.

Qiymatlar qo‘ng‘iroq qilish mumkin bo‘lgan qiymatlar yoki xato holatini boshqarish uchun chaqirilishi kerak bo‘lgan ko‘rinishga to‘liq Python import yo‘lini ifodalovchi satrlar bo‘lishi kerak.

Jumladan, boshqa URLconflar. Istalgan vaqtda siz **urlpatterns**boshqa URLconf modullarini "qo‘shishingiz" mumkin. Bu, aslida, boshqa URL manzillari ostidagi bir qator “ildiz”.

Masalan, Django veb-saytining o‘zi uchun URLconf dan parcha . U bir qator boshqa URLconflarni o‘z ichiga oladi:

```
from django.urls import include, path
urlpatterns = [
    # ... snip ...
    path("community/", include("aggregator.urls")),
    path("contact/", include("contact.urls")),
    # ... snip ...
]
```

Django **include()**ga duch kelganda , u URLning o‘sha nuqtaga to‘g‘ri kelgan qismini kesib tashlaydi va qolgan qatorni keyingi ishlov berish uchun kiritilgan URLconf ga yuboradi.

Yana bir imkoniyat - misollar ro‘yxatidan foydalanib, qo‘shimcha URL naqshlarini kiritish **path()**. Misol uchun, ushbu URLconf ni ko‘rib chiqing:

```
from django.urls import include, path
from apps.main import views as main_views
from credit import views as credit_views
extra_patterns = [
    path("reports/", credit_views.report),
    path("reports/<int:id>/", credit_views.report),
    path("charge/", credit_views.charge),
]
urlpatterns = [
    path("", main_views.homepage),
    path("help/", include("apps.help.urls")),
    path("credit/", include(extra_patterns)),
]
```

Ushbu misolda /credit/reports/URL manzili Django ko‘rinishida ishlaydi credit_views.report().

Bu bitta naqsh prefiksi qayta-qayta ishlatiladigan URLconfs dan ortiqchalikni olib tashlash uchun ishlatilishi mumkin. Misol uchun, ushbu URLconf ni ko‘rib chiqing:

```
from django.urls import path
from . import views
urlpatterns = [
    path("<page_slug>-<page_id>/history/", views.history),
    path("<page_slug>-<page_id>/edit/", views.edit),
    path("<page_slug>-<page_id>/discuss/", views.discuss),
    path("<page_slug>-<page_id>/permissions/", views.permissions),
]
```

Buni umumiy yo‘l prefiksini faqat bir marta ko‘rsatish va farq qiluvchi qo‘shimchalarni guruhlash orqali yaxshilashimiz mumkin:

```
from django.urls import include, path
from . import views
urlpatterns = [
    path(
        "<page_slug>-<page_id>/",
        include(
            [
```

```
            path("history/", views.history),
            path("edit/", views.edit),
            path("discuss/", views.discuss),
            path("permissions/", views.permissions),
        ]
    ),
  ),
]
```

Qabul qilingan parametrlar. Kiritilgan URLconf asosiy URLconfs dan olingan har qanday parametrlarni oladi, shuning uchun quyidagi misol amal qiladi:

```
from django.urls import include, path
from . import views
urlpatterns = [
  path(
    "<page_slug>-<page_id>/",
    include(
      [
        path("history/", views.history),# In settings/urls/main.py
from django.urls import include, path
urlpatterns = [
  path("<username>/blog/", include("foo.urls.blog")),
]
# In foo/urls/blog.py
from django.urls import path
from . import views
urlpatterns = [
  path("", views.blog.index),
  path("archive/", views.blog.archive),
]

            path("edit/", views.edit),
            path("discuss/", views.discuss),
            path("permissions/", views.permissions),
        ]
    ),
  ),
]
```

Yuqoridagi misolda, olingan **“username”** o‘zgaruvchi kutilganidek, kiritilgan URLconf ga uzatiladi.

Funksiyalarni ko‘rish uchun qo‘shimcha parametrlarni o‘tkazish. URLconf-larda Python lug‘ati sifatida ko‘rish funksiyalaringizga qo‘shimcha argumentlarni o‘tkazish imkonini beruvchi ilgak mavjud. Funksiya **path()**ko‘rish funksiyasiga o‘tish uchun qo‘shimcha kalit so‘z argumentlari lug‘ati bo‘lishi kerak bo‘lgan ixtiyoriy uchinchi argumentni olishi mumkin.

Masalan:

```
from django.urls import path
from . import views
urlpatterns = [
    path("blog/<int:year>/", views.year_archive, {"foo": "bar"}),
]
```

Ushbu misolda, so‘rov uchun **/blog/2005/**Django ga qo‘ng‘iroq qiladi .**views.year_archive(request, year=2005, foo='bar')**

Ushbu uslub metadata va variantlarni ko‘rishlarga o‘tkazish uchun sindikatsiya tizimida qo‘llaniladi .

Mojarolar bilan shug‘ullanish. Nomlangan kalit so‘z argumentlarini saqlaydigan, shuningdek, qo‘shimcha argumentlar lug‘atida bir xil nomdagi argumentlarni uzatuvchi URL namunasi bo‘lishi mumkin. Bu sodir bo‘lganda, URL manzilida olingan argumentlar o‘rniga lug‘atdagi argumentlar ishlatiladi.

include()ga qo‘shimcha imkoniyatlar o‘tkazilmoqda

Xuddi shunday, siz qo‘shimcha parametrlarga o‘tishingiz mumkin **include()**va kiritilgan URLconf ning har bir qatoriga qo‘shimcha parametrlar o‘tadi.

Masalan, ushbu ikkita URLconf to‘plami funksional jihatdan bir xil:

Birini o‘rnating:

```
# main.py
from django.urls import include, path
urlpatterns = [
    path("blog/", include("inner"), {"blog_id": 3}),
]
# inner.py
from django.urls import path
from mysite import views
urlpatterns = [
```

```
    path("archive/", views.archive),
    path("about/", views.about),
]
```
Ikkichini o‘rnating:
```
# main.py
from django.urls import include, path
from mysite import views
urlpatterns = [
    path("blog/", include("inner")),
]
# inner.py
from django.urls import path
urlpatterns = [
    path("archive/", views.archive, {"blog_id": 3}),
    path("about/", views.about, {"blog_id": 3}),
]
```
E’tibor bering, qo‘shimcha parametrlar *har doim kiritilgan URLconf ning har bir* qatoriga uzatiladi , chiziq ko‘rinishi ushbu variantlarni haqiqiy deb qabul qiladimi yoki yo‘qmi. Shu sababli, ushbu usul faqat kiritilgan URLconf-dagi har bir ko‘rinish siz o‘tayotgan qo‘shimcha variantlarni qabul qilishiga ishonchingiz komil bo‘lsa foydali bo‘ladi.

URL manzillarining teskari ruxsati. Django loyihasi ustida ishlashda umumiy ehtiyoj - yaratilgan tarkibga (ko‘rishlar va aktivlar URL-manzillari, foydalanuvchiga ko‘rsatilgan URL-manzillar va boshqalar) joylashtirish yoki serverdagi navigatsiya oqimini boshqarish uchun URL-manzillarni yakuniy shaklda olish imkoniyati. yon (yo‘naltirishlar va boshqalar)

Ushbu URL-manzillarni qattiq kodlashdan qochish tavsiya etiladi (mehnat talab qiladigan, kengaytirilmaydigan va xatolarga moyil strategiya). URLconf tomonidan tasvirlangan dizaynga parallel bo‘lgan URL manzillarini yaratish uchun maxsus mexanizmlarni ishlab chiqish ham xavflidir, bu esa vaqt o‘tishi bilan eskirgan URL-manzillarni ishlab chiqarishga olib kelishi mumkin.

Boshqacha qilib aytganda, kerak bo‘lgan narsa QURUQ mexanizmdir. Boshqa afzalliklar qatorida, bu eskirgan URL manzillarini qidirish va almashtirish uchun barcha loyiha manba kodini

ko‘rib chiqishga hojat qoldirmasdan, URL dizaynining evolyutsiyasiga imkon beradi.

URL-manzilni olish uchun bizda mavjud bo‘lgan asosiy ma’lumot uni qayta ishlash uchun mas'ul bo‘lgan ko‘rinishning identifikatsiyasi (masalan, nomi) hisoblanadi. To‘g‘ri URLni qidirishda ishtirok etishi kerak bo‘lgan boshqa ma’lumotlar bo‘laklari ko‘rish argumentlarining turlari (pozitsion, kalit so‘z) va qiymatlaridir.

Django shunday yechimni taqdim etadiki, URL mapper URL dizaynining yagona ombori hisoblanadi. Siz uni URLconf bilan oziqlantirasiz va keyin u har ikki yo‘nalishda ham ishlatilishi mumkin:

- Foydalanuvchi/brauzer so‘ragan URL manzilidan boshlab, u URLdan olingan qiymatlari bilan kerakli argumentlarni taqdim etuvchi to‘g‘ri Django ko‘rinishini chaqiradi.
- Tegishli Django ko‘rinishini va unga uzatiladigan argumentlar qiymatlarini aniqlashdan boshlab, bog‘langan URL manzilini oling.

Birinchisi, biz oldingi bo‘limlarda muhokama qilgan foydalanish. *Ikkinchisi , URL manzillarining teskari o‘lchamlari , teskari URL moslashuvi , teskari URL izlash* yoki oddiygina *URLni teskari o‘zgartirish* deb ataladigan narsadir .

Django URL manzillari kerak bo‘lgan turli qatlamlarga mos keladigan URL teskarisini amalga oshirish uchun vositalarni taqdim etadi:

- Shablonlarda: **url**shablon tegidan foydalanish.
- Python kodida: Funksiyadan foydalanish **reverse().**
- Django namunalarining URL manzillarini qayta ishlash bilan bog‘liq yuqori darajadagi kodda: Usul **get_absolute_url()**.

Misollar. Ushbu URLconf yozuvini yana bir bor ko‘rib chiqing:

```
from django.urls import path
from . import views
q1urlpatterns = [
    # ...
    path("articles/<int:year>/", views.year_archive, name="news-year-archive"),
    # ...
]
```

Ushbu dizaynga ko‘ra, nnnn yiliga mos keladigan arxiv uchun URL **/articles/<nnnn>/**.

Siz ularni shablon kodida olishingiz mumkin:

```
<a href="{% url 'news-year-archive' 2012 %}">2012 Archive</a>
{# Or with the year in a template context variable: #}
<ul>
{% for yearvar in year_list %}
<li><a href="{% url 'news-year-archive' yearvar %}">{{ yearvar }}
Archive</a></li>
{% endfor %}
</ul>
```

yoki Python kodida:

```
from django.http import HttpResponseRedirect
from django.urls import reverse
def redirect_to_year(request):
    # ...
    year = 2006
    # ...
    return HttpResponseRedirect(reverse("news-year-archive",
args=(year,)))
```

Agar biron sababga ko‘ra yillik maqola arxivlari uchun kontent chop etiladigan URL manzillarini o‘zgartirishga qaror qilingan bo‘lsa, URLconfdagi yozuvni o‘zgartirish kifoya qiladi.

Ko‘rishlar umumiy xususiyatga ega bo‘lgan ba'zi stsenariylarda URL manzillar va ko‘rinishlar o‘rtasida ko‘p-bir munosabat mavjud bo‘lishi mumkin. Bunday hollarda ko‘rish nomi URL manzillarini teskari o‘zgartirish vaqti kelganda uning uchun yetarli identifikator emas. Django tomonidan taqdim etilgan yechim haqida bilish uchun keyingi bo‘limni o‘qing.

URL namunalarini nomlash. URL manzilini o‘zgartirishni amalga oshirish uchun yuqoridagi misollarda bo‘lgani kabi **nomlangan URL namunalaridan foydalanishingiz kerak bo‘ladi.** URL nomi uchun ishlatiladigan satr siz yoqtirgan belgilarni o‘z ichiga olishi mumkin. Siz haqiqiy Python nomlari bilan cheklanmagansiz.

URL shablonlarini nomlashda, boshqa ilovalarning nom tanlashiga mos kelmaydigan nomlarni tanlang. Agar siz URL shabloniga qo‘ng‘iroq qilsangiz **comment** va boshqa dastur xuddi shu ishni qilsa, topadigan URL **reverse()** loyihangiz ro‘yxatida oxirgi bo‘lgan naqshga bog‘liq **urlpatterns**.

Ilova nomidan olingan bo‘lishi mumkin bo‘lgan URL nomlariga prefiks qo‘yish (masalan, **myapp-comment**o‘rniga **comment**) to‘qnashuv ehtimolini kamaytiradi.

Agar ko‘rinishni bekor qilmoqchi bo‘lsangiz, ataylab boshqa ilova bilan *bir xil URL nomini* tanlashingiz mumkin . Misol uchun, umumiy foydalanish holati ni bekor qilishdir **LoginView**. Django qismlari va ko‘pgina uchinchi tomon ilovalari ushbu ko‘rinishda nomi bilan URL namunasi bor deb taxmin qilinadi **login**. Agar sizda shaxsiy kirish ko‘rinishi bo‘lsa va uning URL manziliga nom bersangiz **login**, u kiritilgandan keyin (agar u umuman kiritilgan bo‘lsa) **reverse()** o‘zingizning shaxsiy ko‘rinishingizni topadi.

.**urlpatternsdjango.contrib.auth.urls**

Agar ular argumentlarida farq qilsa, bir nechta URL namunalari uchun bir xil nomdan foydalanishingiz mumkin. URL nomiga qo‘shimcha ravishda **reverse()** argumentlar soni va kalit so‘z argumentlari nomlariga mos keladi. Yo‘l konvertorlari ham **ValueError** mos kelmasligini ko‘rsatish uchun ko‘tarilishi mumkin, batafsil ma’lumot uchun Maxsus yo‘l konvertorlarini ro‘yxatdan o‘tkazish bo‘limiga qarang.

URL nom maydonlari. URL nom bo‘shliqlari turli ilovalar bir xil URL nomlaridan foydalansa ham nomlangan URL naqshlarini noyob tarzda o‘zgartirishga imkon beradi . Bu uchinchi tomon ilovalari uchun har doim nomlar maydoni URL manzillaridan foydalanish yaxshi amaliyotdir (darslikda qilganimizdek). Xuddi shunday, agar ilovaning bir nechta nusxalari o‘rnatilgan bo‘lsa, u URL manzillarini o‘zgartirishga ham imkon beradi. Boshqacha qilib aytadigan bo‘lsak, bitta ilovaning bir nechta nusxalari nomlangan URL manzillarini baham ko‘rishi sababli, nomlar bo‘shliqlari ushbu nomlangan URL manzillarini ajratish usulini ta’minlaydi.

URL nom maydonidan to‘g‘ri foydalanadigan Django ilovalari ma'lum bir sayt uchun bir necha marta joylashtirilishi mumkin. Masalan, administratorning bir nechta nusxalarini joylashtirish imkonini beruvchi sinf **django.contrib.admin** mavjud. Keyingi misolda biz ikkita turli auditoriyaga (mualliflar va noshirlar) bir xil funksiyalardan foydalanishimiz uchun o‘quv darsliksidan so‘rovnoma ilovasini ikki xil joyda joylashtirish g‘oyasini muhokama qilamiz. **AdminSite** URL nom maydoni ikki qismdan iborat bo‘lib, ikkalasi ham satrlardan iborat:**ilova nom maydoni**

Bu o‘rnatilayotgan ilova nomini tavsiflaydi. Bitta ilovaning har bir nusxasi bir xil dastur nom maydoniga ega bo‘ladi. Masalan, Django administrator ilovasi biroz taxmin qilinadigan dastur nom maydoniga ega **'admin'**.

Bu ilovaning muayyan namunasini aniqlaydi. Namuna nomlari butun loyihangiz bo‘ylab noyob bo‘lishi kerak. Biroq, misol nom maydoni ilova nom maydoni bilan bir xil bo‘lishi mumkin. Bu ilovaning standart nusxasini belgilash uchun ishlatiladi. Misol uchun, standart Django administrator misolida misol nom maydoni mavjud **'admin'**.

Ismlar maydoni URL manzillari operator yordamida belgilanadi **':'**. Masalan, administrator ilovasining asosiy indeks sahifasi yordamida havola qilinadi **'admin:index'**. Bu ning nom maydonini **'admin'** va nomli URL manzilini bildiradi **'index'**.

Nom maydonlarini ham joylashtirish mumkin. Nomlangan URL yuqori darajadagi nomlar maydonida aniqlangan nomlar maydonida **'sports:polls:index'** nomlangan naqshni qidiradi. **'index''polls''sports'**

Ismlar oralig‘idagi URL manzillarini o‘zgartirish.

Yechish uchun nom maydoni URL manzili berilganda 'polls:index', Django to‘liq nomni qismlarga ajratadi va keyin quyidagi qidiruvni amalga oshiradi:

1. Birinchidan, Django mos keladigan dastur nom maydonini qidiradi (bu misolda, 'polls'). Bu o‘sha ilovaning namunalari ro‘yxatini beradi.
2. Agar joriy ilova aniqlangan bo‘lsa, Django o‘sha misol uchun URL hal qiluvchini topadi va qaytaradi. **current_app** Joriy dasturni funksiyaga argument bilan ko‘rsatish mumkin **reverse()**.

Shablon **url** yorlig‘i joriy ilova sifatida hozirda hal qilingan ko‘rinishning nom maydonidan foydalanadi **RequestContext**. Atributga joriy ilovani o‘rnatish orqali ushbu standartni bekor qilishingiz mumkin **request.current_app**.

3. Agar joriy ilova bo‘lmasa, Django standart dastur namunasini qidiradi. Standart dastur namunasi ilova nomlar maydoniga mos keladigan misol nom maydoniga ega bo‘lgan misoldir (bu misolda chaqirilgan misol).**polls'polls'**

4. Agar standart dastur namunasi bo‘lmasa, Django uning namunasi qanday bo‘lishidan qat'i nazar, ilovaning oxirgi o‘rnatilgan nusxasini tanlaydi.
5. Agar taqdim etilgan nom maydoni 1-bosqichdagi dastur nom maydoniga mos kelmasa, Django nom maydonini misol nom maydoni sifatida to‘g‘ridan-to‘g‘ri qidirishga harakat qiladi .

Agar ichki o‘rnatilgan nomlar bo‘shliqlari mavjud bo‘lsa, bu qadamlar faqat ko‘rinish nomi hal etilmaguncha nomlar maydonining har bir qismi uchun takrorlanadi. Keyin ko‘rish nomi topilgan nomlar maydonida URL manzilida hal qilinadi.

Polls Ushbu rezolyutsiya strategiyasini amalda ko‘rsatish uchun darslikdagi ilovaning ikkita misolini ko‘rib chiqing : biri chaqirilgan **'author-polls'** va biri chaqirilgan **'publisher-polls'**. Faraz qilaylik, biz ushbu ilovani so‘rovnomalarni yaratish va ko‘rsatishda misol nomlari maydonini hisobga oladigan qilib kengaytirdik.

urls.py

```
from django.urls import include, path
urlpatterns = [
    path("author-polls/", include("polls.urls", namespace="author-
polls")),
    path("publisher-polls/", include("polls.urls",
namespace="publisher-polls")),
]
```

polls/urls.py

```
from django.urls import path
from . import views
app_name = "polls"
urlpatterns = [
    path("", views.IndexView.as_view(), name="index"),
    path("<int:pk>/", views.DetailView.as_view(), name="detail"),
    ...,
]
```

Ushbu sozlamadan foydalanib, quyidagi qidiruvlarni amalga oshirish mumkin:

- Agar misollardan biri joriy bo‘lsa - deylik, biz misolda batafsil sahifani ko‘rsatgan bo‘lsak **'author-polls'**- **'polls:index'** misolning indeks sahifasiga o‘tadi **'author-polls'**; ya'ni quyidagi ikkalasi ham natija beradi **"/author-polls/"**.

Sinfga asoslangan ko‘rinish usulida:
reverse("polls:index",
current_app=self.request.resolver_match.namespace)
va shablonda:
{% **url** 'polls:index' %}

- Agar joriy misol bo‘lmasa - aytaylik, agar biz saytning boshqa joyida sahifani ko‘rsatayotgan bo‘lsak - **'polls:index'**oxirgi ro‘yxatdan o‘tgan nusxasini hal qiladi **polls**. Standart misol (misol nom maydoni) bo‘lmaganligi sababli **'polls'**, ro‘yxatga olingan oxirgi nusxasi **polls**ishlatiladi. **'publisher-polls'** Bu oxirgi marta e’lon qilinganidan beri bo‘ladi **urlpatterns**.
- **'author-polls:index'**har doim misolning indeks sahifasiga o‘tadi **'author-polls'**(va shunga o‘xshash uchun **'publisher-polls'**).

Agar birlamchi misol - ya'ni nomli misol ham bo‘lsa, **'polls'** yuqoridagi yagona o‘zgarish joriy nusxa mavjud bo‘lmagan holatda bo‘ladi (yuqoridagi ro‘yxatdagi ikkinchi element). Bu holda **'polls:index'** oxirgi marta e'lon qilingan misol o‘rniga standart misolning indeks sahifasiga o‘tadi **urlpatterns**.

URL nom maydonlari va kiritilgan URLconfs. Kiritilgan URLconf-larning ilova nom maydonlari ikki usulda belgilanishi mumkin.

Birinchidan, siz **app_name** kiritilgan URLconf modulida **urlpatterns**atribut bilan bir xil darajadagi atributni o‘rnatishingiz mumkin. **include()**Siz o‘z ro‘yxatiga emas, balki haqiqiy modulni yoki modulga string havolasini o‘tkazishingiz kerak **urlpatterns**.

polls/urls.py

```
from django.urls import path
from . import views
app_name = "polls"
urlpatterns = [
    path("", views.IndexView.as_view(), name="index"),
    path("<int:pk>/", views.DetailView.as_view(), name="detail"),
    ...,
]
```

urls.py

```
from django.urls import include, path
urlpatterns = [
    path("polls/", include("polls.urls")),
```

]

Belgilangan URL manzillar **polls.urls** ilova nom maydoniga ega bo‘ladi **polls**.

Ikkinchidan, siz o‘rnatilgan nomlar maydoni ma’lumotlarini o‘z ichiga olgan obyektni kiritishingiz mumkin. Agar siz yoki misollar **include()** ro‘yxati bo‘lsa , ushbu obyektdagi URL manzillar global nomlar maydoniga qo‘shiladi. Shu bilan birga, siz o‘z ichiga olgan 2-tube ham mumkin: **path()re_path()include()**

(<list of path()/re_path() instances>, <application namespace>)

Masalan:

```
from django.urls import include, path
from . import views
polls_patterns = (
    [
        path("", views.IndexView.as_view(), name="index"),
        path("<int:pk>/", views.DetailView.as_view(), name="detail"),
    ],
    "polls",
)
urlpatterns = [
    path("polls/", include(polls_patterns)),
]
```

Bu berilgan ilova nom maydoniga belgilangan URL namunalarini o‘z ichiga oladi. Misol nom maydoni **namespace**to argumenti yordamida belgilanishi mumkin **include()**. Agar misol nom maydoni ko‘rsatilmagan bo‘lsa, u birlamchi URLconf ilovasining nom maydoniga o‘tadi. Bu, shuningdek, ushbu nom maydoni uchun standart namuna bo‘lishini anglatadi.

Nazorat savollari:

1. Djangoda settings.py fayl nima?
2. Settings sozlamalarini o‘zgartirish uchun qanday qilib tuzish mumkin?
3. Django proyektida boshqa sozlamalar fayllarini qanday qo‘llash mumkin?
4. Djangoda ALLOWED_HOSTS sozlamasi nima uchun kerak?
5. STATIC_URL va STATIC_ROOT sozlamalari qanday ishlaydi?
6. MEDIA_URL va MEDIA_ROOT sozlamalari nima uchun kerak?
7. Djangoda URL lar qanday tuziladi?
8. URL lar ro‘yxatida qanday shablonlar ishlatish mumkin?

9-§. DJANGODA STANDART USER MODELI VA UNI LOYHAGA QAYTA MOSLASH.

Obyektlar autentifikatsiya tizimining asosiy qismidir. Ular odatda saytingiz bilan o‘zaro aloqada bo‘lgan odamlarni ifodalaydi va kirishni cheklash, foydalanuvchi profillarini ro‘yxatdan o‘tkazish, kontentni yaratuvchilar bilan bog‘lash va h.k. kabi narsalarni yoqish uchun ishlatiladi. Django autentifikatsiya tizimida faqat bitta toifali foydalanuvchilar mavjud, ya’ni administrator foydalanuvchilar shunchaki foydalanuvchi 'superusers' obyektlari 'staff'. foydalanuvchi obyektlarining turli sinflari emas, balki maxsus atributlar to‘plami.

Standart foydalanuvchining asosiy atributlari quyidagilardir:

username
password
email
first_name
last_name

Foydalanuvchilarni yaratish. Foydalanuvchilarni yaratishning eng oson yo‘li kiritilgan yordamchi create_user() funksiyadan foydalanishdir:

```
>>> from django.contrib.auth.models import User
>>> user = User.objects.create_user("john",
"lennon@thebeatles.com", "johnpassword")
# to the database. You can continue to change its attributes
# if you want to change other fields.
>>> user.last_name = "Lennon"
>>> user.save()
```

Agar sizda Django administratori o‘rnatilgan bo‘lsa, foydalanuvchilarni interaktiv tarzda ham yaratishingiz mumkin .

Superusers yaratish. Buyruq yordamida superuserlarni yarating createsuperuser:

```
$ python manage.py createsuperuser --username=joe --
email=joe@example.com
```

Sizdan parol so‘raladi. Birini kiritganingizdan so‘ng, foydalanuvchi darhol yaratiladi. –username. Agar yoki parametrlarini o‘chirib qo‘ysangiz --email, u sizdan o‘sha qiymatlarni so‘raydi.

Parollarni o‘zgartirish. Django foydalanuvchi modelida xom (aniq matnli) parollarni saqlamaydi, faqat xesh (to‘liq ma’lumot uchun parollar qanday boshqarilishi haqidagi hujjatlarga qarang). Shu sababli,

foydalanuvchining parol atributini to‘g‘ridan-to‘g‘ri manipulyatsiya qilishga urinmang. Shuning uchun foydalanuvchi yaratishda yordamchi funksiyadan foydalaniladi. Foydalanuvchi parolini o‘zgartirish uchun sizda bir nechta variant mavjud: manage.py changepassword *username*buyruq satridan foydalanuvchi parolini o‘zgartirish usulini taklif qiladi. U sizdan ikki marta kiritishingiz kerak bo‘lgan foydalanuvchining parolini o‘zgartirishni taklif qiladi. Agar ikkalasi ham mos kelsa, yangi parol darhol o‘zgartiriladi. Agar siz foydalanuvchini taqdim qilmasangiz, buyruq foydalanuvchi nomi joriy tizim foydalanuvchisiga mos keladigan parolni o‘zgartirishga harakat qiladi.

Shuningdek, parolni quyidagi yordamida dasturiy tarzda o‘zgartirishingiz mumkin set_password():

```
>>> from django.contrib.auth.models import User
>>> u = User.objects.get(username="john")
>>> u.set_password("new password")
>>> u.save()
```

Agar sizda Django administratori o‘rnatilgan bo‘lsa, autentifikatsiya tizimining administrator sahifalarida foydalanuvchi parollarini ham o‘zgartirishingiz mumkin. Django shuningdek, foydalanuvchilarga o‘z parollarini o‘zgartirishga ruxsat berish uchun ishlatilishi mumkin bo‘lgan ko‘rinishlar va shakllarni taqdim etadi. Foydalanuvchi parolini o‘zgartirish uning barcha seanslarini o‘chirib qo‘yadi. Tafsilotlar uchun parolni o‘zgartirish bo‘yicha sessiya bekor qilinishiga qarang .

Foydalanuvchilarning autentifikatsiyasi. authenticate(so‘rov = Yo‘q , ** hisob ma’lumotlari) authenticate() Hisob ma’lumotlari to‘plamini tekshirish uchun foydalaning. U hisob ma’lumotlarini kalit so‘z argumentlari sifatida oladi usernameva passwordstandart holatda ularni har bir autentifikatsiya backendiga nisbatan tekshiradi va Useragar hisob ma’lumotlari backend uchun haqiqiy bo‘lsa, obyektni qaytaradi. Agar hisob ma’lumotlari hech qanday backend uchun haqiqiy bo‘lmasa yoki backend ko‘tarilsa PermissionDenied, u qaytadi None. Masalan:

```
from django.contrib.auth import authenticate
user = authenticate(username="john", password="secret")
if user is not None:
    # A backend authenticated the credentials
```

```
    ...
else:
    # No backend authenticated the credentials
    ...
```

requestixtiyoriy bo‘lib, u autentifikatsiyaning backends usuli HttpRequestbo‘yicha uzatiladi. authenticate()

Eslatma: Bu hisobga olish ma’lumotlari to‘plamini autentifikatsiya qilishning past darajadagi usuli; masalan, tomonidan ishlatiladi RemoteUserMiddleware. Agar siz o‘zingizning autentifikatsiya tizimingizni yozmaguningizcha, undan foydalana olmaysiz. Aksincha, foydalanuvchiga kirish usulini izlayotgan bo‘lsangiz, dan foydalaning LoginView.

Ruxsatlar va avtorizatsiya. Django o‘rnatilgan ruxsat tizimi bilan birga keladi. Bu ma'lum foydalanuvchilar va foydalanuvchilar guruhlariga ruxsat berish usulini taqdim etadi. U Django administrator sayti tomonidan qo‘llaniladi, lekin siz uni o‘z kodingizda ishlatishingiz mumkin.

Django administrator sayti quyidagi ruxsatlardan foydalanadi:

Obyektlarni ko‘rish huquqi ushbu turdagi obyektlar uchun “ko‘rish” yoki “o‘zgartirish” ruxsatiga ega foydalanuvchilar uchun cheklangan.

“Qo‘shish” shaklini ko‘rish va obyektni qo‘shish uchun ruxsat ushbu turdagi obyekt uchun “qo‘shish” ruxsatiga ega foydalanuvchilar uchun cheklangan.

O‘zgartirishlar ro‘yxatini ko‘rish, “o‘zgartirish” shaklini ko‘rish va obyektni o‘zgartirish uchun kirish ushbu turdagi obyekt uchun "o‘zgartirish" ruxsatiga ega foydalanuvchilar uchun cheklangan.

Obyektni o‘chirish uchun ruxsat ushbu turdagi obyekt uchun “o‘chirish” ruxsatiga ega foydalanuvchilar uchun cheklangan.

Ruxsatlar nafaqat obyekt turiga, balki muayyan obyekt namunasiga ham o‘rnatilishi mumkin. has_view_permission()Sinf tomonidan taqdim etilgan , va usullaridan foydalanib , bir xil turdagi turli has_add_permission()xil obyektlar uchun ruxsatlarni sozlash mumkin.

has_change_permission()has_delete_permission()ModelAdmin

User obyektlar ikkita va undan ko‘p maydonga ega: groups va user_permissions. Obyektlar o‘zlarining tegishli obyektlariga boshqa Django modeliUser kabi kirishlari mumkin :

myuser.groups.set([group_list])
myuser.groups.add(group, group, ...)
myuser.groups.remove(group, group, ...)
myuser.groups.clear()
myuser.user_permissions.set([permission_list])
myuser.user_permissions.add(permission, permission, ...)
myuser.user_permissions.remove(permission, permission, ...)
myuser.user_permissions.clear()

Birlamchi ruxsatlar. django.contrib.auth Sozlamangiz ro‘yxatida bo‘lsa , INSTALLED_APPS o‘rnatilgan ilovalaringizdan birida belgilangan har bir Django modeli uchun to‘rtta standart ruxsat – qo‘shish, o‘zgartirish, o‘chirish va ko‘rish – yaratilganligini ta’minlaydi. Ushbu ruxsatnomalar ishga tushirilganda yaratiladi ; ga qo‘shgandan keyin birinchi marta ishga tushirganingizda , avval o‘rnatilgan barcha modellar, shuningdek, o‘sha paytda o‘rnatilgan har qanday yangi modellar uchun standart ruxsatlar yaratiladi. Keyinchalik, u har safar ishga tushganingizda yangi modellar uchun standart ruxsatlarni yaratadi (ruxsatlarni yaratuvchi funksiya signalga ulangan). manage.py migratemigratedjango.contrib.auth INSTALLED_APPS manage.py migratepost_migrate
Modelga Permission kamdan-kam hollarda to‘g‘ridan-to‘g‘ri kirish mumkin.

Guruhlar. django.contrib.auth.models.Groupmodellar foydalanuvchilarni turkumlashning umumiy usuli bo‘lib, siz ushbu foydalanuvchilarga ruxsatlar yoki boshqa yorliqlarni qo‘llashingiz mumkin. Foydalanuvchi istalgan miqdordagi guruhlarga tegishli bo‘lishi mumkin.

Guruhdagi foydalanuvchi avtomatik ravishda ushbu guruhga berilgan ruxsatlarga ega. Misol uchun, agar guruh ruxsatga ega bo‘lsa , ushbu guruhdagi har qanday foydalanuvchi ushbu ruxsatga ega bo‘ladi. Site editorscan_edit_home_page

Ruxsatlardan tashqari, guruhlar foydalanuvchilarni yorliq yoki kengaytirilgan funksiyalarni berish uchun toifalarga ajratishning qulay usulidir. Misol uchun, siz guruh yaratishingiz mumkin va siz ularga saytingizning faqat a'zolar qismiga kirish huquqini beradigan kod yozishingiz yoki ularga faqat a'zolar uchun elektron pochta xabarlarini yuborishingiz mumkin.'Special users'

Ruxsatlarni dasturiy ravishda yaratish. Maxsus ruxsatlar model sinfida aniqlanishi mumkin bo‘lsa-da Meta, siz to‘g‘ridan-to‘g‘ri ruxsatlarni ham yaratishingiz mumkin. Masalan, quyidagi model can_publish uchun ruxsatni yaratishingiz mumkin :BlogPostmyapp

```
from myapp.models import BlogPost
from django.contrib.auth.models import Permission
from django.contrib.contenttypes.models import ContentType
content_type = ContentType.objects.get_for_model(BlogPost)
permission = Permission.objects.create(
    codename="can_publish",
    name="Can Publish Posts",
    content_type=content_type)
```

Agar proksi modeli uchun ruxsatlar yaratmoqchi bo‘lsangiz , tegishlisini olish uchun for_concrete_model=False quyidagiga o‘ting:

```
ContentTypeManager.get_for_model()ContentType
content_type = ContentType.objects.get_for_model( BlogPostProxy,
for_concrete_model=False)
```

Ruxsat keshlash. Foydalanuvchi obyektidagi ruxsatlarni keshlar ModelBackend birinchi marta ruxsatlarni tekshirish uchun olinishi kerak. Bu odatda so‘rov-javob sikli uchun to‘g‘ri keladi, chunki ruxsatlar odatda qo‘shilgandan so‘ng darhol tekshirilmaydi (masalan, adminstratorda). Agar siz ruxsatlarni qo‘shsangiz va ularni darhol test yoki ko‘rinishda tekshirsangiz, eng oson yechim foydalanuvchini ma’lumotlar bazasidan qayta olishdir. Masalan:

```
from django.contrib.auth.models import Permission, User
from django.contrib.contenttypes.models import ContentType
from django.shortcuts import get_object_or_404
from myapp.models import BlogPost
def user_gains_perms(request, user_id):
    user = get_object_or_404(User, pk=user_id)
    # any permission check will cache the current set of permissions
    user.has_perm("myapp.change_blogpost")
    content_type = ContentType.objects.get_for_model(BlogPost)
    permission = Permission.objects.get(
        codename="change_blogpost",
        content_type=content_type,    )
    user.user_permissions.add(permission)
    # Checking the cached permission set
```

```
    user.has_perm("myapp.change_blogpost")  # False
    # Request new instance of User
    # Be aware that user.refresh_from_db() won't clear the cache.
    user = get_object_or_404(User, pk=user_id)
    # Permission cache is repopulated from the database
    user.has_perm("myapp.change_blogpost")  # True
```

Proksi-server modellari. Proksi-serverlar aniq modellar bilan bir xil ishlaydi. Ruxsatlar proksi-server modelining o‘ziga xos kontent turi yordamida yaratiladi. Proksi-serverlar o‘zlari kichik sinfga kiruvchi aniq modelning ruxsatlarini meros qilib olmaydilar:

```
class Person(models.Model):
    class Meta:
        permissions = [("can_eat_pizzas", "Can eat pizzas")]
class Student(Person):
    class Meta:
        proxy = True
        permissions = [("can_deliver_pizzas", "Can deliver pizzas")]
>>> # Fetch the content type for the proxy model.
>>> content_type = ContentType.objects.get_for_model(Student,
for_concrete_model=False)
>>> student_permissions =
Permission.objects.filter(content_type=content_type)
>>> [p.codename for p in student_permissions]
['add_student', 'change_student', 'delete_student', 'view_student',
'can_deliver_pizzas']
>>> for permission in student_permissions:
...     user.user_permissions.add(permission)
>>> user.has_perm("app.add_person")
False
>>> user.has_perm("app.can_eat_pizzas")
False
>>> user.has_perms(("app.add_student", "app.can_deliver_pizzas"))
True
```

Veb-so‘rovlarda autentifikatsiya. Django request objects autentifikatsiya tizimini ulash uchun seanslar va o‘rta dasturlardan foydalanadi.

Ular request.user joriy foydalanuvchini ifodalovchi har bir so‘rov uchun atributni taqdim etadi. Agar joriy foydalanuvchi tizimga

kirmagan bo‘lsa, bu atribut ning namunasiga o‘rnatiladi AnonymousUser, aks holda uning namunasi bo‘ladi User.
is_authenticated Siz ularni quyidagicha ajratishingiz mumkin :

```
if request.user.is_authenticated:
    # Do something for authenticated users.
    ...
else:
    # Do something for anonymous users.
    ...
```

Foydalanuvchini qanday tizimga kiritish kerak. Agar sizda autentifikatsiya qilingan foydalanuvchi bo‘lsa, siz joriy seansga biriktirmoqchi bo‘lsangiz - bu funksiya bilan amalga oshiriladi login().

login(so‘rov , foydalanuvchi , backend = Yo‘q)

Foydalanuvchini ko‘rinishdan tizimga kiritish uchun foydalaning login(). HttpRequestU obyekt va obyektni oladi User login()Django seans ramkasidan foydalanib, foydalanuvchi identifikatorini sessiyada saqlaydi.

Esda tutingki, foydalanuvchi tizimga kirgandan so‘ng anonim seans davomida to‘plangan har qanday ma’lumotlar sessiyada saqlanadi.

authenticate()Ushbu misol ikkalasini ham qanday ishlatishingiz mumkinligini ko‘rsatadi login():

```
from django.contrib.auth import authenticate, login
def my_view(request):
    username = request.POST["username"]
    password = request.POST["password"]
    user = authenticate(request, username=username,
password=password)
    if user is not None:
        login(request, user)
        # Redirect to a success page.
        ...
    else:
        # Return an 'invalid login' error message.
```

Autentifikatsiya serverini tanlash. Foydalanuvchi tizimga kirganda, foydalanuvchi identifikatori va autentifikatsiya qilish uchun foydalanilgan backend foydalanuvchi sessiyasida saqlanadi. Bu xuddi shu autentifikatsiya serveriga kelajakdagi so‘rovda foydalanuvchi

ma’lumotlarini olish imkonini beradi. Seansda saqlash uchun autentifikatsiya serveri quyidagicha tanlanadi: Agar mavjud bo‘lsa, ixtiyoriy argumentning qiymatidan foydalaning backend.

Qanday qilib foydalanuvchi tizimdan chiqish kerak?

logout(so‘rov) orqali tizimga kirgan foydalanuvchini tizimdan chiqish uchun o‘z ko‘rinishida django.contrib.auth.login()foydalaning . django.contrib.auth.logout()U HttpRequestobyektni oladi va qaytish qiymatiga ega emas. Misol:

```
from django.contrib.auth import logout
def logout_view(request):
    logout(request)
    # Redirect to a success page.
```

E’tibor bering, logout()agar foydalanuvchi tizimga kirmagan bo‘lsa, hech qanday xatolik yuzaga kelmaydi.

logout()ga qo‘ng‘iroq qilganingizda, joriy so‘rov uchun seans ma’lumotlari butunlay tozalanadi. Barcha mavjud ma’lumotlar o‘chiriladi. Bu boshqa shaxsning tizimga kirishi va avvalgi foydalanuvchining sessiya ma’lumotlariga kirishi uchun bir xil veb-brauzerdan foydalanishiga yo‘l qo‘ymaslik uchundir. Agar siz tizimdan chiqqaningizdan so‘ng darhol foydalanuvchi uchun mavjud bo‘ladigan seansga biror narsa qo‘ymoqchi bo‘lsangiz, qo‘ng‘iroq qilganingizdan so‘ng django.contrib.auth.logout() buni qiling .

Tizimga kirgan foydalanuvchilarga kirishni cheklash. Sahifalarga kirishni cheklashning asosiy usuli - kirish sahifasini tekshirish request.user.is_authenticated va yo‘naltirish:

```
from django.conf import settings
from django.shortcuts import redirect
def my_view(request):
    if not request.user.is_authenticated:
        return redirect(f"{settings.LOGIN_URL}?next={request.path}")
    # yoki xato xabarini ko‘rsating:
from django.shortcuts import render
def my_view(request):
    if not request.user.is_authenticated:
        return render(request, "myapp/login_error.html")
    # ...
```

Dekorator login_required _

login_required(redirect_field_name = 'keyingi' , login_url = Yo‘q)

Yorliq sifatida siz qulay login_required()dekoratordan foydalanishingiz mumkin:

```
from django.contrib.auth.decorators import login_required
@login_required
def my_view(request):
```

login_required()quyidagilarni amalga oshiradi:

Agar foydalanuvchi tizimga kirmagan bo‘lsa, settings.LOGIN_URLso‘rovlar qatoridagi joriy mutlaq yo‘lni o‘tkazib, yo‘naltiring. Misol: /accounts/login/?next=/polls/3/.

Agar foydalanuvchi tizimga kirgan bo‘lsa, ko‘rinishni odatdagidek bajaring.

Odatiy bo‘lib, muvaffaqiyatli autentifikatsiyadan so‘ng foydalanuvchi qayta yo‘naltirilishi kerak bo‘lgan yo‘l so‘rovlar qatori deb nomlangan parametrda saqlanadi “next”. Agar siz ushbu parametr uchun boshqa nomdan foydalanishni xohlasangiz, login_required()ixtiyoriy redirect_field_name parametrni oladi:

```
from django.contrib.auth.decorators import login_required
@login_required(redirect_field_name="my_redirect_field")
def my_view(request):
```

Shuni yodda tutingki, agar siz qiymat bersangiz redirect_field_name, katta ehtimol bilan login shabloningizni ham sozlashingiz kerak bo‘ladi, chunki qayta yo‘naltirish yo‘lini saqlaydigan shablon kontekst o‘zgaruvchisi (standart) redirect_field_nameemas, balki uning qiymatidan kalit sifatida foydalanadi. “next”

login_required()ixtiyoriy login_url parametrni ham oladi.

Misol:

```
from django.contrib.auth.decorators import login_required
@login_required(login_url="/accounts/login/")
def my_view(request):
```

E’tibor bering, agar siz parametrni ko‘rsatmasangiz va kirish ko‘rinishi to‘g‘ri bog‘langanligiga login_urlishonch hosil qilishingiz kerak. settings.LOGIN_URL Masalan, standart sozlamalardan foydalanib, URLconf-ga quyidagi qatorlarni qo‘shing:

```
from django.contrib.auth import views as auth_views
path("accounts/login/", auth_views.LoginView.as_view())
```

Shuningdek, settings.LOGIN_URL, ko‘rish funksiyasi nomlari va nomlangan URL naqshlarini ham qabul qiladi . Bu sozlamani

yangilamasdan turib URLconf ichida kirish ko‘rinishini erkin tarzda qayta ko‘rsatish imkonini beradi.

Eslatma: Dekorator foydalanuvchidagi bayroqni login_required tekshirmaydi, lekin sukut bo‘yicha nofaol foydalanuvchilarni rad etadi.is_active AUTHENTICATION_BACKENDS

Agar siz Django administratori uchun maxsus ko‘rinishlarni yozayotgan bo‘lsangiz (yoki o‘rnatilgan ko‘rinishlar foydalanadigan bir xil avtorizatsiya tekshiruviga muhtoj bo‘lsangiz), dekoratorni django.contrib.admin.views.decorators.staff_member_required() ga foydali alternativ topishingiz mumkin login_required().

Aralash LoginRequiredMixin _

Sinfga asoslangan ko‘rinishlardan foydalanganda, login_requireddan foydalanish bilan bir xil xatti-harakatlarga erishishingiz mumkin LoginRequiredMixin.

Agar ko‘rinish ushbu aralashdan foydalansa, autentifikatsiya qilinmagan foydalanuvchilar tomonidan barcha so‘rovlar kirish sahifasiga yo‘naltiriladi yoki parametrga qarab HTTP 403 Taqiqlangan xato ko‘rsatiladi raise_exception.

AccessMixinRuxsatsiz foydalanuvchilar bilan ishlashni sozlash uchun har qanday parametrni o‘rnatishingiz mumkin :

```
from django.contrib.auth.mixins import LoginRequiredMixin
class MyView(LoginRequiredMixin, View):
    login_url = "/login/"
    redirect_field_name = "redirect_to"
```

Eslatma: Xuddi dekorator sifatida , bu miksin foydalanuvchidagi bayroqni login_requiredtekshirmaydi, lekin sukut bo‘yicha nofaol foydalanuvchilarni rad etadi.is_activeAUTHENTICATION_BACKENDS

Sinovdan o‘tgan tizimga kirgan foydalanuvchilarga kirishni cheklash. Muayyan ruxsatlar yoki boshqa testlar asosida kirishni cheklash uchun siz avvalgi paragrafda tavsiflangan amalni bajarasiz.

Sinovni request.user to‘g‘ridan-to‘g‘ri ko‘rinishda bajarishingiz mumkin. Masalan, ushbu ko‘rinish foydalanuvchining kerakli domenda elektron pochta manziliga ega ekanligini tekshiradi va agar bo‘lmasa, kirish sahifasiga yo‘naltiradi:

```
from django.shortcuts import redirect
def my_view(request):
    if not request.user.email.endswith("@example.com"):
```

```
    return redirect("/login/?next=%s" % request.path)
```
user_passes_test(test_func , login_url = Yo‘q , redirect_field_name = "keyingi")

Yorliq sifatida siz user_passes_test qo‘ng‘iroq qilinadigan narsa qaytib kelganda qayta yo‘naltirishni amalga oshiradigan qulay dekoratordan foydalanishingiz mumkin False:

```
from django.contrib.auth.decorators import user_passes_test
def email_check(user):
  return user.email.endswith("@example.com")
@user_passes_test(email_check)
def my_view(request):
```

user_passes_test()talab qilinadigan argumentni oladi: obyektni oladigan User va True foydalanuvchi sahifani ko‘rishga ruxsat berilsa, qaytib keladigan chaqiruv. user_passes_test()Anonim emasligini avtomatik ravishda tekshirmasligini unutmang.

user_passes_test()ikkita ixtiyoriy argumentni oladi:

login_url

Sinovdan o‘ta olmagan foydalanuvchilar qayta yo‘naltiriladigan URL manzilini belgilash imkonini beradi. Bu tizimga kirish sahifasi bo‘lishi mumkin va settings.LOGIN_URLagar siz ko‘rsatmagan bo‘lsangiz, u birlamchi bo‘ladi. redirect_field_name

Xuddi shunday login_required(). NoneAgar siz testdan o‘tmagan foydalanuvchilarni “keyingi sahifa” mavjud bo‘lmagan tizimga kirmaydigan sahifaga yo‘naltirayotgan bo‘lsangiz, uni URL manzilidan olib tashlashingiz mumkin .

Masalan:

```
@user_passes_test(email_check, login_url="/login/")
def my_view(request):
```

Sinfga asoslangan ko‘rinishlardan foydalanganda buni amalga oshirish uchun foydalanishingiz mumkin UserPassesTestMixin.

test_func(). test_func()Amalga oshirilgan testni taqdim etish uchun sinf usulini bekor qilishingiz kerak . AccessMixin Bundan tashqari, ruxsatsiz foydalanuvchilar bilan ishlashni sozlash uchun har qanday parametrni o‘rnatishingiz mumkin :

```
from django.contrib.auth.mixins import UserPassesTestMixin
class MyView(UserPassesTestMixin, View):
  def test_func(self):
    return self.request.user.email.endswith("@example.com")
```

get_test_func()

Shuningdek, siz get_test_func()miksin tekshiruvlari uchun boshqa nomdagi funksiyadan foydalanishi uchun usulni bekor qilishingiz mumkin (o‘rniga test_func()).

StackingUserPassesTestMixin. Amalga oshirish usuli tufayli User PassesTestMixinsiz ularni meros ro‘yxatiga joylashtira olmaysiz. Quyidagilar ishlamaydi:

```
class TestMixin1(UserPassesTestMixin):
    def test_func(self):
        return self.request.user.email.endswith("@example.com")
class TestMixin2(UserPassesTestMixin):
    def test_func(self):
        return self.request.user.username.startswith("django")
class MyView(TestMixin1, TestMixin2, View):
```

Agar TestMixin1 qo‘ng‘iroq qilsa super() va bu natijani hisobga olsa, TestMixin1 endi mustaqil ishlamaydi.

Dekorator permission_required.

permission_required(perm , login_url = Yo‘q , rise_exception = False)

Foydalanuvchining ma'lum bir ruxsati borligini tekshirish nisbatan keng tarqalgan vazifadir. Shu sababli, Django bu holat uchun yorliq beradi: dekorator permission_required().:

```
from django.contrib.auth.decorators import permission_required
@permission_required("polls.add_choice")
def my_view(request):
```

Xuddi has_perm()usul kabi, ruxsat nomlari ham shaklni oladi (ya'ni, ilovadagi modelga ruxsat olish uchun)."<app label>.<permission codename>"polls.add_choicepolls

Dekorator shuningdek, ruxsatlarning takrorlanishini ham olishi mumkin, bu holda foydalanuvchi ko‘rinishga kirish uchun barcha ruxsatlarga ega bo‘lishi kerak.

E’tibor bering, permission_required() ixtiyoriy login_urlparametr ham oladi:

```
from django.contrib.auth.decorators import permission_required
@permission_required("polls.add_choice", login_url="/loginpage/")
def my_view(request):
```

Dekoratorda bo‘lgani kabi login_required(), login_url sukut bo‘yicha settings.LOGIN_URL.

Agar raise_exceptionparametr berilgan bo‘lsa, dekorator kirish sahifasiga yo‘naltirish o‘rniga 403 (HTTP Taqiqlangan) ko‘rinishini PermissionDenied taklif qiladi.

Agar siz foydalanmoqchi bo‘lsangiz raise_exception, lekin foydalanuvchilarga avval tizimga kirish imkoniyatini bersangiz, login_required()dekoratorni qo‘shishingiz mumkin:

```
from django.contrib.auth.decorators import login_required,
permission_required
@login_required
@permission_required("polls.add_choice", raise_exception=True)
def my_view(request):
```

LoginViewBu, shuningdek , tizimga redirect_authenticated_user =True kirgan foydalanuvchi barcha kerakli ruxsatlarga ega bo‘lmaganida, qayta yo‘naltirish davrining oldini oladi .

Aralash PermissionRequiredMixin _

Sinfga asoslangan ko‘rinishlarga ruxsatnoma tekshiruvlarini qo‘llash uchun foydalanishingiz mumkin PermissionRequiredMixin:

sinf PermissionRequiredMixin

Ushbu aralash, xuddi permission_required dekorator kabi, ko‘rinishga kirayotgan foydalanuvchi barcha ruxsatlarga ega yoki yo‘qligini tekshiradi. Ruxsatni (yoki ruxsatlarning takrorlanishini) permission_requiredparametr yordamida belgilashingiz kerak:

```
from django.contrib.auth.mixins import PermissionRequiredMixin
class MyView(PermissionRequiredMixin, View):
    permission_required = "polls.add_choice"
    # Or multiple of permissions:
    permission_required = ["polls.view_choice",
"polls.change_choice"]
```

AccessMixinRuxsatsiz foydalanuvchilar bilan ishlashni sozlash uchun istalgan parametrni o‘rnatishingiz mumkin .

Siz ushbu usullarni ham bekor qilishingiz mumkin:

get_permission_required()

Miksin tomonidan ishlatiladigan ruxsat nomlarining takrorlanishini qaytaradi. Atributning standart sozlamalari permission_required, agar kerak bo‘lsa, kortejga aylantiriladi.

has_permission()

Joriy foydalanuvchining bezatilgan ko‘rinishni bajarishga ruxsati bor-yo‘qligini bildiruvchi mantiqiy qiymatni qaytaradi.

has_perms()Odatiy bo‘lib, bu tomonidan qaytarilgan ruxsatnomalar ro‘yxati bilan qo‘ng‘iroq natijasini qaytaradi get_permission_required().

Sinfga asoslangan ko‘rinishlarda ruxsatsiz so‘rovlarni qayta yo‘naltirish.

Sinfga asoslangan ko‘rinishlarda kirish cheklovlarini boshqarishni yengillashtirish uchun AccessMixin kirish taqiqlanganda ko‘rinishning harakatini sozlash uchun foydalanish mumkin. Haqiqiyligi tekshirilgan foydalanuvchilarga HTTP 403 taqiqlangan javob bilan kirish taqiqlanadi. Anonim foydalanuvchilar kirish sahifasiga yo‘naltiriladi yoki atributga qarab HTTP 403 Taqiqlangan javobni ko‘rsatadi raise_exception.

sinf AccessMixin. login_url uchun standart qaytarish qiymati get_login_url(). None Qaysi holatda standart qiymatlar get_login_url()ga qaytadi settings.LOGIN_URL. permission_denied_message uchun standart qaytarish qiymati get_permission_denied_message(). Birlamchi bo‘sh satr uchun.

redirect_field_name uchun standart qaytarish qiymati get_redirect_field_name(). Birlamchi "next".

raise_exception

Agar bu atribut ga o‘rnatilgan bo‘lsa True, PermissionDeniedshartlar bajarilmasa, istisno paydo bo‘ladi. Qachon False(standart), anonim foydalanuvchilar kirish sahifasiga yo‘naltiriladi.

get_login_url()

Sinovdan o‘ta olmagan foydalanuvchilar qayta yo‘naltiriladigan URL manzilini qaytaradi. login_urlAgar o‘rnatilgan bo‘lsa yoki boshqacha tarzda qaytariladi settings.LOGIN_URL.

get_permission_denied_message()

Qachon raise_exceptionbo‘lsa True, bu usul foydalanuvchiga ko‘rsatish uchun xato ishlov beruvchiga yuborilgan xato xabarini boshqarish uchun ishlatilishi mumkin. permission_denied_messageSukut bo‘yicha atributni qaytaradi.

get_redirect_field_name()

Muvaffaqiyatli kirishdan so‘ng foydalanuvchi yo‘naltirilishi kerak bo‘lgan URL manzilini o‘z ichiga olgan so‘rov parametri nomini qaytaradi. Agar siz buni ga o‘rnatsangiz None, so‘rov parametri qo‘shilmaydi. redirect_field_nameSukut bo‘yicha atributni qaytaradi .

handle_no_permission() ning qiymatiga qarab raise_exception, usul PermissionDeniedistisnoni keltirib chiqaradi yoki foydalanuvchini ga yo‘naltiradi login_url, ixtiyoriy ravishda redirect_field_nameagar u o‘rnatilgan bo‘lsa.

Parolni o‘zgartirishda seansni bekor qilish. Agar siz o‘z usulini AUTH_USER_MODELmeros qilib olsa yoki amalga oshirsa , autentifikatsiya qilingan seanslar ushbu funksiya tomonidan qaytarilgan xeshni o‘z ichiga oladi. Bunday holda , bu parol maydonining HMAC hisoblanadi. Django har bir so‘rov uchun sessiyadagi xesh so‘rov davomida hisoblanganga mos kelishini tekshiradi. Bu foydalanuvchiga parolni o‘zgartirish orqali barcha seanslaridan chiqish imkonini beradi. AbstractBaseUserget_session_auth_hash()AbstractBaseUser

Djangoga kiritilgan standart parolni o‘zgartirish ko‘rinishlari PasswordChangeViewva user_change_passwordadministratordagi ko‘rinish django.contrib.auth, o‘z parolini o‘zgartirgan foydalanuvchi tizimdan chiqmasligi uchun sessiyani yangi parol xeshi bilan yangilang. Agar sizda maxsus parolni o‘zgartirish ko‘rinishi bo‘lsa va shunga o‘xshash xatti-harakatga ega bo‘lishni istasangiz, funksiyadan foydalaning update_session_auth_hash() .

update_session_auth_hash(so‘rov , foydalanuvchi)

Ushbu funksiya joriy so‘rovni va yangi seans xeshi olinadigan yangilangan foydalanuvchi obyektini oladi va seans xeshini mos ravishda yangilaydi. Shuningdek, u o‘g‘irlangan seans cookie-faylini bekor qilish uchun seans kalitini aylantiradi.

Misol:

```
from django.contrib.auth import update_session_auth_hash
def password_change(request):
    if request.method == "POST":
        form = PasswordChangeForm(user=request.user,
data=request.POST)
        if form.is_valid():
            form.save()
            update_session_auth_hash(request, form.user)
    else:
```

Eslatma: get_session_auth_hash() ga asoslanganligi sababli SECRET_KEY, yangi sirni ishlatish uchun saytingizni yangilashda mavjud seanslarni bekor qilmaslik uchun maxfiy kalit qiymatlari

aylantirilishi kerak. SECRET_KEY_FALLBACKSTafsilotlar uchun qarang .

Autentifikatsiya ko‘rinishlari. Django login, chiqish va parollarni boshqarish uchun foydalanishingiz mumkin bo‘lgan bir nechta ko‘rinishlarni taqdim etadi. Ular aktsiyalarni tasdiqlash shakllaridan foydalanadi , lekin siz o‘zingizning shakllaringizda ham o‘tishingiz mumkin.

Django autentifikatsiya ko‘rinishlari uchun standart shablonni taqdim etmaydi. Siz foydalanmoqchi bo‘lgan ko‘rinishlar uchun o‘zingizning shablonlaringizni yaratishingiz kerak. Shablon konteksti har bir ko‘rinishda hujjatlashtirilgan, qarang: Barcha autentifikatsiya ko‘rinishlari .

Ko‘rinishlardan foydalanish. Loyihangizda bu qarashlarni amalga oshirishning turli usullari mavjud. django.contrib.auth.urls Eng oson yo‘li, taqdim etilgan URLconf-ni o‘zingizning URLconf-ga qo‘shishdir , masalan:

```
urlpatterns = [
    path("accounts/", include("django.contrib.auth.urls")),
]
```

Bunga quyidagi URL namunalari kiradi:

```
accounts/login/ [name='login']
accounts/logout/ [name='logout']
accounts/password_change/ [name='password_change']
accounts/password_change/done/ [name='password_change_done']
accounts/password_reset/ [name='password_reset']
accounts/password_reset/done/ [name='password_reset_done']
accounts/reset/<uidb64>/<token>/ [name='password_reset_confirm']
accounts/reset/done/ [name='password_reset_complete']
```

Ko‘rinishlar osonroq murojaat qilish uchun URL nomini beradi. Nomlangan URL naqshlaridan foydalanish bo‘yicha batafsil ma’lumot uchun URL hujjatlariga qarang .

Agar siz URL manzillaringizni ko‘proq nazorat qilishni istasangiz, URLconfda ma’lum bir ko‘rinishga murojaat qilishingiz mumkin:

```
from django.contrib.auth import views as auth_views
urlpatterns = [
    path("change-password/",
auth_views.PasswordChangeView.as_view()),
```

```
]
```

Ko‘rinishlarda ko‘rinishning harakatini o‘zgartirish uchun foydalanishingiz mumkin bo‘lgan ixtiyoriy argumentlar mavjud. Misol uchun, agar siz ko‘rinishda foydalanadigan shablon nomini o‘zgartirmoqchi bo‘lsangiz, argumentni taqdim etishingiz mumkin template_name. Buning usuli URLconf da kalit so‘z argumentlarini taqdim etishdir, ular ko‘rinishga uzatiladi. Masalan:

```
urlpatterns = [
    path(
        "change-password/",

auth_views.PasswordChangeView.as_view(template_name="change-
password.html"),
    ),
]
```

Barcha ko‘rinishlar sinfga asoslangan bo‘lib , bu sizga ularni pastki sinflarga ajratish orqali osongina sozlash imkonini beradi.

Barcha autentifikatsiya ko‘rinishlari

Bu barcha ko‘rinishlarni o‘z ichiga olgan ro‘yxat django.contrib.auth. Amalga oshirish tafsilotlari uchun ko‘rinishlardan foydalanishga qarang .

sinf LoginView

Nomlangan URL naqshlaridan foydalanish bo‘yicha batafsil ma’lumot uchun URL hujjatlariga qarang .

Usullar va atributlar. template_name Foydalanuvchi tizimga kirishda foydalaniladigan ko‘rinish uchun ko‘rsatiladigan shablon nomi. Standart registration/login.html.

next_page

Kirishdan keyin qayta yo‘naltiriladigan URL. Birlamchi LOGIN_REDIRECT_URL.

redirect_field_name

GETTizimga kirgandan keyin yo‘naltiriladigan URL manzili bo‘lgan maydon nomi . Birlamchi next. get_default_redirect_url()Agar berilgan GETparametr uzatilsa , URL manzilini bekor qiladi .

authentication_form

Autentifikatsiya uchun foydalanish uchun chaqiriladigan (odatda shakl sinfi). Birlamchi AuthenticationForm.

extra_context

Shablonga uzatiladigan standart kontekst ma’lumotlariga qo‘shiladigan kontekst ma’lumotlarining lug‘ati.

redirect_authenticated_user

Kirish sahifasiga autentifikatsiya qilingan foydalanuvchilar kirish yoki yo‘qligini nazorat qiluvchi mantiqiy parametr go‘yo ular endigina muvaffaqiyatli kirgandek qayta yo‘naltiriladi. Standart False.

Ogohlantirish. redirect_authenticated_userni yoqsangiz, boshqa veb-saytlar veb-saytingizdagi rasm fayllariga qayta yo‘naltirish URL manzillarini so‘rash orqali o‘z tashrif buyuruvchilari saytingizda autentifikatsiya qilinganligini aniqlashi mumkin bo‘ladi. “ Ijtimoiy tarmoqlarda barmoq izlari ” ma’lumotlari sizib chiqishining oldini olish uchun barcha rasmlar joylashtiring.

Yoqish, shuningdek, agar parametr ishlatilmasa, dekoratordan redirect_authenticated_userfoydalanganda qayta yo‘naltirish sikliga olib kelishi mumkin .permission_required()raise_exception

success_url_allowed_hosts

setga qo‘shimcha ravishda request.get_host(), tizimga kirgandan so‘ng qayta yo‘naltirish uchun xavfsiz bo‘lgan hostlar . Birlamchi bo‘sh set.

get_default_redirect_url()

Tizimga kirgandan keyin yo‘naltiriladigan URL manzilini qaytaradi. Standart dastur hal qilinadi va next_pageagar o‘rnatilgan bo‘lsa yoki LOGIN_REDIRECT_URLboshqacha tarzda qaytariladi.

Mana nima LoginView qiladi:

GET orqali chaqirilsa, u bir xil URL manziliga POST yuboradigan kirish formasini ko‘rsatadi. Bu haqida birozdan keyin.

Agar POSTfoydalanuvchi tomonidan taqdim etilgan hisob ma’lumotlari orqali qo‘ng‘iroq qilinsa, u foydalanuvchini tizimga kirishga harakat qiladi. Kirish muvaffaqiyatli bo‘lsa, ko‘rinishda ko‘rsatilgan URL manziliga yo‘naltiriladi next. Agar nexttaqdim etilmagan bo‘lsa, u ga yo‘naltiriladi settings.LOGIN_REDIRECT_URL(birlamchi qiymat /accounts/profile/). Agar tizimga kirish muvaffaqiyatli bo‘lmasa, u kirish formasini qayta ko‘rsatadi. registration/login.htmlSukut bo‘yicha chaqirilgan login shablonini html bilan ta'minlash sizning javobgarligingizdir . Ushbu shablon to‘rtta shablon kontekstli o‘zgaruvchilardan o‘tadi:

form: Formni ifodalovchi obyekt AuthenticationForm.

next: Muvaffaqiyatli tizimga kirgandan so‘ng qayta yo‘naltiriladigan URL. Bu so‘rovlar qatorini ham o‘z ichiga olishi mumkin.
siteSite: Sozlamaga muvofiq joriy SITE_ID. RequestSiteAgar sizda sayt ramkasi o‘rnatilmagan bo‘lsa, bu sayt nomi va domenini joriy dan oladigan misoliga o‘rnatiladi HttpRequest.
site_name: uchun taxallus site.name. Agar sizda sayt ramkasi o‘rnatilmagan bo‘lsa, bu qiymatga o‘rnatiladi request.META['SERVER_NAME']. Saytlar haqida ko‘proq ma’lumot olish uchun "saytlar" ramkasiga qarang .

Agar siz shablonga qo‘ng‘iroq qilmaslikni xohlasangiz , parametrni qo‘shimcha argumentlar orqali URLconf-dagi usulga registration/login.htmlo‘tkazishingiz mumkin . Misol uchun, bu URLconf qatori o‘rniga :template_nameas_viewmyapp/login.html
path("accounts/login/",
auth_views.LoginView.as_view(template_name="myapp/login.html")
),

GETdan foydalanib, tizimga kirgandan so‘ng qayta yo‘naltiriladigan URL manzilini o‘z ichiga olgan maydon nomini ham belgilashingiz mumkin redirect_field_name. Odatiy bo‘lib, maydon chaqiriladi next.

registration/login.htmlBu yerda siz boshlang‘ich nuqta sifatida foydalanishingiz mumkin bo‘lgan namunaviy shablon. base.htmlSizda blokni belgilaydigan shablon mavjud deb taxmin qilinadi content:

```
{% extends "base.html" %}
{% block content %}
{% if form.errors %}
<p>Your username and password didn't match. Please try again.</p>
{% endif %}
{% if next %}
    {% if user.is_authenticated %}
    <p>Your account doesn't have access to this page. To proceed,
    please login with an account that has access.</p>
    {% else %}
    <p>Please login to see this page.</p>
    {% endif %}
{% endif %}
<form method="post" action="{% url 'login' %}">
{% csrf_token %}
```

```
<table>
<tr>
   <td>{{ form.username.label_tag }}</td>
   <td>{{ form.username }}</td>
</tr>
<tr>
   <td>{{ form.password.label_tag }}</td>
   <td>{{ form.password }}</td>
</tr>
</table>
<input type="submit" value="login">
<input type="hidden" name="next" value="{{ next }}">
</form>
{# Assumes you set up the password_reset view in your URLconf #}
<p><a href="{% url 'password_reset' %}">Lost password?</a></p>
{% endblock %}
```

Agar sizda moslashtirilgan autentifikatsiya bo‘lsa (Autentifikatsiyani sozlash bo‘limiga qarang) atributni o‘rnatish orqali maxsus autentifikatsiya formasidan foydalanishingiz mumkin authentication_form. Ushbu shakl requesto‘z usulida kalit so‘z argumentini qabul qilishi __init__()va get_user()autentifikatsiya qilingan foydalanuvchi obyektini qaytaradigan usulni taqdim etishi kerak (bu usul faqat forma tekshiruvidan so‘ng chaqiriladi).

sinf LogoutView

So‘rovlar bo‘yicha foydalanuvchini tizimdan chiqaradi POST.

URL nomi: logout

Atributlar: next_page

Chiqishdan keyin qayta yo‘naltiriladigan URL. Birlamchi LOGOUT_REDIRECT_URL.

template_name

Foydalanuvchi tizimdan chiqqandan keyin ko‘rsatiladigan shablonning to‘liq nomi. Birlamchi registration/logged_out.html.

redirect_field_name

GET Tizimdan chiqqandan keyin qayta yo‘naltiriladigan URL manzilini o‘z ichiga olgan maydon nomi. Birlamchi 'next'. next_pageAgar berilgan GETparametr uzatilsa , URL manzilini bekor qiladi .

extra_context

Shablonga uzatiladigan standart kontekst ma’lumotlariga qo‘shiladigan kontekst ma’lumotlarining lug‘ati.
success_url_allowed_hosts

setga qo‘shimcha ravishda request.get_host(), tizimdan chiqqandan so‘ng qayta yo‘naltirish uchun xavfsiz bo‘lgan A xostlari. Birlamchi bo‘sh set.
Shablon konteksti:
title: "Chiqish" qatori lokallashtirilgan.
siteSite: Sozlamaga muvofiq joriy SITE_ID. RequestSiteAgar sizda sayt ramkasi o‘rnatilmagan bo‘lsa, bu sayt nomi va domenini joriy dan oladigan misoliga o‘rnatiladi HttpRequest.
site_name: uchun taxallus site.name. Agar sizda sayt ramkasi o‘rnatilmagan bo‘lsa, bu qiymatga o‘rnatiladi request.META['SERVER_NAME']. Saytlar haqida ko‘proq ma’lumot olish uchun "saytlar" ramkasiga qarang .
logout_then_login(so‘rov , login_url = Yo‘q)

So‘rovlar bo‘yicha foydalanuvchini tizimdan chiqaradi POST, keyin kirish sahifasiga yo‘naltiradi.
URL nomi: Standart URL ko‘rsatilmagan

Ixtiyoriy argumentlar: login_url: Qayta yo‘naltiriladigan kirish sahifasining URL manzili. settings.LOGIN_URLAgar ta'minlanmagan bo‘lsa, birlamchi .
sinf PasswordChangeView
URL nomi: password_change
Foydalanuvchiga o‘z parolini o‘zgartirishga ruxsat beradi.
Atributlar: template_name

Parolni o‘zgartirish shaklini ko‘rsatish uchun foydalaniladigan shablonning to‘liq nomi. registration/password_change_form.htmlAgar ta'minlanmagan bo‘lsa, birlamchi .
success_url

Muvaffaqiyatli parol o‘zgartirilgandan so‘ng qayta yo‘naltiriladigan URL. Birlamchi 'password_change_done'.
form_class

Kalit so‘z argumentini qabul qilishi kerak bo‘lgan maxsus "parolni o‘zgartirish" shakli user. Shakl foydalanuvchi parolini o‘zgartirish uchun javobgardir. Birlamchi PasswordChangeForm.
extra_context

Shablonga uzatiladigan standart kontekst ma’lumotlariga qo‘shiladigan kontekst ma’lumotlarining lug‘ati.
Shablon konteksti:
form: Parolni o‘zgartirish shakli (form_classyuqoriga qarang).
sinf PasswordChangeDoneView
URL nomi: password_change_done
Foydalanuvchi parolini o‘zgartirgandan so‘ng ko‘rsatiladigan sahifa.
Atributlar: template_name
Foydalanish uchun shablonning to‘liq nomi. registration/password_change_done.htmlAgar ta'minlanmagan bo‘lsa, birlamchi .
extra_context
Shablonga uzatiladigan standart kontekst ma’lumotlariga qo‘shiladigan kontekst ma’lumotlarining lug‘ati.
sinf PasswordResetView
URL nomi: password_reset
Parolni tiklash uchun ishlatilishi mumkin bo‘lgan bir martalik foydalanish havolasini yaratish va ushbu havolani foydalanuvchining ro‘yxatdan o‘tgan elektron pochta manziliga yuborish orqali foydalanuvchiga o‘z parolini tiklash imkonini beradi. Agar quyidagi shartlar bajarilsa, bu ko‘rinish elektron pochta xabarini yuboradi: Ko‘rsatilgan elektron pochta manzili tizimda mavjud. Talab qilingan foydalanuvchi faol (User.is_active) True.
So‘ralgan foydalanuvchi foydalanish mumkin bo‘lgan parolga ega. Ishlamaydigan parol bilan belgilangan foydalanuvchilarga (qarang set_unusable_password()) LDAP kabi tashqi autentifikatsiya manbasidan foydalanganda noto‘g‘ri foydalanishning oldini olish uchun parolni tiklashni so‘rashga ruxsat berilmaydi.
Agar ushbu shartlardan birortasi bajarilmasa , elektron pochta xabari yuborilmaydi, lekin foydalanuvchi ham xato xabarini olmaydi. Bu potentsial tajovuzkorlarga ma’lumotlarning oqib ketishining oldini oladi. Agar siz bu holatda xato xabarini taqdim qilmoqchi bo‘lsangiz, siz subklassga kirishingiz PasswordResetFormva form_classatributdan foydalanishingiz mumkin.
Eslatma: Shuni yodda tutingki, elektron pochta xabarini yuborish qo‘shimcha vaqt talab etadi, shuning uchun siz mavjud elektron pochta manzilini qayta o‘rnatish so‘rovining davomiyligi va mavjud

bo‘lmagan elektron pochta manzilini tiklash so‘rovining davomiyligi o‘rtasidagi farq tufayli elektron pochta manzillarini sanash vaqti hujumiga qarshi himoyasiz bo‘lishingiz mumkin. . Qo‘shimcha xarajatlarni kamaytirish uchun siz elektron pochta xabarlarini sinxron ravishda yuborish imkonini beruvchi uchinchi tomon paketidan foydalanishingiz mumkin, masalan, Djangomailer .

Atributlar:

template_name

Parolni tiklash shaklini ko‘rsatish uchun foydalaniladigan shablonning to‘liq nomi. registration/password_reset_form.htmlAgar ta'minlanmagan bo‘lsa, birlamchi .

form_class

Parolni tiklash uchun foydalanuvchining elektron pochta manzilini olish uchun foydalaniladigan shakl. Birlamchi PasswordResetForm.

email_template_name

Parolni tiklash havolasi bilan elektron pochta xabarini yaratishda foydalaniladigan shablonning to‘liq nomi.
Registration/password_reset_email.htmlAgar ta'minlanmagan bo‘lsa, birlamchi .

subject_template_name

Parolni tiklash havolasi bilan xat mavzusi uchun foydalanish uchun shablonning to‘liq nomi.
Registration/password_reset_subject.txtAgar ta'minlanmagan bo‘lsa, birlamchi .

token_generator

Bir martalik havolani tekshirish uchun sinfning namunasi. Bu sukut bo‘yicha bo‘ladi default_token_generator, bu ning namunasidir django.contrib.auth.tokens.PasswordResetTokenGenerator.

success_url

Muvaffaqiyatli parolni tiklash so‘rovidan keyin qayta yo‘naltiriladigan URL. Birlamchi 'password_reset_done'.

from_email

Yaroqli elektron pochta manzili. Odatiy bo‘lib Django dan foydalanadi DEFAULT_FROM_EMAIL.

extra_context

Shablonga uzatiladigan standart kontekst ma’lumotlariga qo‘shiladigan kontekst ma’lumotlarining lug‘ati.

html_email_template_name

Parolni tiklash havolasi bilan matn/html ko‘p qismli elektron pochta xabarini yaratish uchun foydalaniladigan shablonning to‘liq nomi . Odatiy bo‘lib, HTML xat yuborilmaydi.

extra_email_context

Elektron pochta shablonida mavjud bo‘ladigan kontekst ma’lumotlarining lug‘ati. U quyida keltirilgan standart shablon kontekst qiymatlarini bekor qilish uchun ishlatilishi mumkin, masalan domain.

Shablon konteksti:

formform_class: Foydalanuvchi parolini tiklash uchun shakl (yuqoriga qarang).

Elektron pochta shablonining konteksti:

email: uchun taxallususer.email

user: Joriy User, shakl maydoniga ko‘ra email. Faqat faol foydalanuvchilar o‘z parollarini tiklashlari mumkin ( ).User.is_active is True

site_name: uchun taxallus site.name. Agar sizda sayt ramkasi o‘rnatilmagan bo‘lsa, bu qiymatga o‘rnatiladi request.META['SERVER_NAME']. Saytlar haqida ko‘proq ma’lumot olish uchun "saytlar" ramkasiga qarang .

domain: uchun taxallus site.domain. Agar sizda sayt ramkasi o‘rnatilmagan bo‘lsa, bu qiymatga o‘rnatiladi request.get_host().

protocol: http yoki https

uid: 64-bazada kodlangan foydalanuvchining asosiy kaliti.

token: Qayta tiklash havolasi haqiqiyligini tekshirish uchun token.

Namuna registration/password_reset_email.html(elektron pochta asosiy shabloni):

Someone asked for password reset for email {{ email }}. Follow the link below:

{{ protocol}}://{{ domain }}

{% url 'password_reset_confirm' uidb64=uid token=token %}

Xuddi shu shablon konteksti mavzu shabloni uchun ishlatiladi. Mavzu bir qatorli oddiy matn qatori bo‘lishi kerak.

URL nomi: password_reset_done

Foydalanuvchiga parolni tiklash uchun havola yuborilganidan keyin ko‘rsatiladigan sahifa. PasswordResetView Agarda aniq URL manzili bo‘lmasa, bu ko‘rinish sukut bo‘yicha chaqiriladi success_url.

Eslatma: Agar taqdim etilgan elektron pochta manzili tizimda mavjud bo‘lmasa, foydalanuvchi faol emas yoki paroli ishlatib bo‘lmaydigan bo‘lsa, foydalanuvchi hali ham ushbu ko‘rinishga yo‘naltiriladi, lekin elektron pochta xabari yuborilmaydi.

Atributlar:

template_name

Foydalanish uchun shablonning to‘liq nomi. registration/password_reset_done.htmlAgar ta'minlanmagan bo‘lsa, birlamchi .

extra_context

Shablonga uzatiladigan standart kontekst ma’lumotlariga qo‘shiladigan kontekst ma’lumotlarining lug‘ati.

sinf PasswordResetConfirmView

URL nomi: password_reset_confirm

Yangi parolni kiritish shaklini taqdim etadi.

URL manzilidagi kalit so‘z argumentlari:

uidb64: 64-bazada kodlangan foydalanuvchi identifikatori.

token: Parolning haqiqiyligini tekshirish uchun belgi.

Atributlar:

template_name

Parolni tasdiqlash ko‘rinishini ko‘rsatish uchun shablonning to‘liq nomi. Standart qiymat registration/password_reset_confirm.html.

token_generator

Parolni tekshirish uchun sinfning namunasi. Bu sukut bo‘yicha bo‘ladi default_token_generator, buning namunasidir django.contrib.auth.tokens.PasswordResetTokenGenerator.

post_reset_login

Parolni muvaffaqiyatli tiklashdan so‘ng foydalanuvchi avtomatik ravishda autentifikatsiya qilinishi kerakligini ko‘rsatadigan mantiqiy ko‘rsatkich. Birlamchi False.

post_reset_login_backend

Foydalanuvchini autentifikatsiya qilishda foydalanish uchun autentifikatsiya serveriga nuqtali yo‘l, agar post_reset_loginbo‘lsa True. Agar sizda bir nechta AUTHENTICATION_BACKENDSkonfiguratsiya mavjud bo‘lsagina talab qilinadi. Birlamchi None.

form_class

Parolni o‘rnatish uchun foydalaniladigan shakl. Birlamchi SetPasswordForm.
success_url

Parolni tiklashdan so‘ng qayta yo‘naltirish uchun URL. Birlamchi password_reset_complete'.
extra_context

Shablonga uzatiladigan standart kontekst ma’lumotlariga qo‘shiladigan kontekst ma’lumotlarining lug‘ati.
reset_url_token

Token parametri parolni tiklash URL manzillarining komponenti sifatida ko‘rsatiladi. Birlamchi 'set-password'.
Shablon konteksti:
formform_class: Yangi foydalanuvchi parolini o‘rnatish uchun shakl (yuqoriga qarang).
validlinkuidb64: Mantiqiy, Havola (va birikmasi token) haqiqiy yoki hali foydalanilmagan bo‘lsa, rost.
sinf PasswordResetCompleteView
URL nomi: password_reset_complete

Foydalanuvchiga parol muvaffaqiyatli o‘zgartirilganligi haqida xabar beradigan ko‘rinishni taqdim etadi.
Atributlar:
template_name

Ko‘rinishni ko‘rsatish uchun shablonning to‘liq nomi. Birlamchi registration/password_reset_complete.html.
extra_context

Shablonga uzatiladigan standart kontekst ma’lumotlariga qo‘shiladigan kontekst ma’lumotlarining lug‘ati.
Yordamchi funksiyalar
redirect_to_login(keyingi , login_url = Yo‘q , redirect_field_name = 'keyingi')

Kirish sahifasiga yo‘naltiradi va muvaffaqiyatli kirishdan keyin boshqa URL manziliga qaytadi.
Kerakli dalillar:
next: Muvaffaqiyatli tizimga kirgandan so‘ng qayta yo‘naltiriladigan URL.
Ixtiyoriy argumentlar:
login_url: Qayta yo‘naltiriladigan kirish sahifasining URL manzili. settings.LOGIN_URLAgar ta'minlanmagan bo‘lsa, birlamchi .

redirect_field_nameGET: tizimdan chiqqandan keyin qayta yo‘naltiriladigan URL manzilini o‘z ichiga olgan maydon nomi . nextBerilgan GETparametr o‘tgan bo‘lsa, bekor qiladi .

O‘rnatilgan shakllar. Agar siz o‘rnatilgan ko‘rinishlardan foydalanishni istamasangiz, lekin bu funksiya uchun shakllarni yozishingiz shart emasligi uchun qulaylikni istasangiz, autentifikatsiya tizimi quyidagi manzilda joylashgan bir nechta o‘rnatilgan shakllarni taqdim etadi django.contrib.auth.forms:

Eslatma: O‘rnatilgan autentifikatsiya shakllari ular ishlayotgan foydalanuvchi modeli haqida ma'lum taxminlarni keltirib chiqaradi. Agar siz maxsus foydalanuvchi modelidan foydalanayotgan bo‘lsangiz , autentifikatsiya tizimi uchun o‘z shakllaringizni belgilashingiz kerak bo‘lishi mumkin. Qo‘shimcha ma’lumot olish uchun shaxsiy foydalanuvchi modellari bilan o‘rnatilgan autentifikatsiya shakllaridan foydalanish bo‘yicha hujjatlarga qarang .

sinf AdminPasswordChangeForm

Foydalanuvchi parolini o‘zgartirish uchun administrator interfeysida foydalaniladigan shakl.

userBirinchi pozitsion argument sifatida ni oladi .

sinf AuthenticationForm

Foydalanuvchini tizimga kirish uchun ariza. O‘zining birinchi pozitsion argumentini oladi request, u kichik sinflar tomonidan foydalanish uchun shakl misolida saqlanadi.

confirm_login_allowed(foydalanuvchi) odatiy bo‘lib, bayrog‘i ga o‘rnatilgan Authentication Formfoydalanuvchilarni rad etadi . Qaysi foydalanuvchilar tizimga kirishi mumkinligini aniqlash uchun ushbu xatti-harakatni maxsus siyosat bilan bekor qilishingiz mumkin. Buni usulni pastki sinflarga ajratadigan va bekor qiluvchi maxsus shakl bilan bajaring . Agar foydalanuvchi tizimga kirmasa, bu usul a ko‘tarishi kerak

.is_activeFalseAuthenticationFormconfirm_login_allowed()Validation Error

Masalan, barcha foydalanuvchilarga "faol" holatidan qat'iy nazar tizimga kirishiga ruxsat berish uchun:

```
from django.contrib.auth.forms import AuthenticationForm
class
AuthenticationFormWithInactiveUsersOkay(AuthenticationForm):
    def confirm_login_allowed(self, user):
```

```
pass
```
(Bunday holatda, siz faol bo‘lmagan foydalanuvchilarga ruxsat beruvchi autentifikatsiya serveridan ham foydalanishingiz kerak bo‘ladi, masalan AllowAllUsersModelBackend.)
Yoki faqat ayrim faol foydalanuvchilarga tizimga kirishiga ruxsat berish uchun:

```
class PickyAuthenticationForm(AuthenticationForm):
    def confirm_login_allowed(self, user):
        if not user.is_active:
            raise ValidationError(
                _("This account is inactive."),
                code="inactive",
            )
        if user.username.startswith("b"):
            raise ValidationError(
                _("Sorry, accounts starting with 'b' aren't welcome here."),
                code="no_b_users",
            )
```

sinf PasswordChangeForm
Foydalanuvchiga o‘z parolini o‘zgartirishga ruxsat berish shakli.
sinf PasswordResetForm
Foydalanuvchi parolini tiklash uchun bir martalik foydalanish havolasini yaratish va elektron pochta orqali yuborish uchun shakl.
send_mail(mavzu_shablon_nomi , email_shablon_nomi , kontekst , from_email , to_email , html_email_template_name = Yo‘q)
ni yuborish uchun argumentlardan foydalanadi EmailMultiAlternatives. Elektron pochta foydalanuvchiga qanday yuborilishini sozlash uchun bekor qilinishi mumkin.
Parametrlar:
subyekt_shablon_nomi - mavzu uchun shablon.
email_template_name - elektron pochtaning asosiy qismi uchun shablon.
kontekst - kontekst subject_template, email_templateva html_email_template(agar bo‘lmasa None) ga uzatiladi.
from_email – jo‘natuvchining elektron pochtasi.
to_email - so‘rovchining elektron pochtasi.
html_email_template_name - HTML korpusi uchun shablon; sukut bo‘yicha None, bu holda oddiy matnli elektron pochta xabari

yuboriladi. Odatiy bo‘lib, elektron pochta kontekstiga o‘tadigan bir xil o‘zgaruvchilar bilan save()to‘ldiradi .contextPasswordResetView
sinf SetPasswordForm

Foydalanuvchiga eski parolni kiritmasdan o‘z parolini o‘zgartirish imkonini beruvchi shakl.
sinf UserChangeForm

Foydalanuvchi ma’lumotlari va ruxsatlarini o‘zgartirish uchun administrator interfeysida foydalaniladigan shakl.
sinf BaseUserCreationForm

ModelFormYangi foydalanuvchi yaratish uchun agar foydalanuvchi yaratish shaklini sozlashingiz kerak bo‘lsa, bu tavsiya etilgan asosiy sinfdir.

U uchta maydonga ega: username(foydalanuvchi modelidan), password1, va password2. password1U buni va mos kelishini tekshiradi password2, yordamida parolni tasdiqlaydi validate_password()va yordamida foydalanuvchi parolini o‘rnatadi set_password().

sinf UserCreationForm baseUserCreationForm dan meros oladi. Shunga o‘xshash foydalanuvchi nomlari bilan chalkashmaslikka yordam berish uchun ariza faqat har qanday holatda farq qiladigan foydalanuvchi nomlariga ruxsat bermaydi.

Eski versiyalarda User CreationForm maxsus foydalanuvchi modeli uchun ko‘pdan ko‘pga shakl maydonlari saqlanmadi. Eski versiyalarda faqat har bir holatda farq qiladigan foydalanuvchi nomlariga ruxsat beriladi.

Shablonlardagi autentifikatsiya ma’lumotlari. Hozirda tizimga kirgan foydalanuvchi va ularning ruxsatlari siz foydalanganda shablon kontekstidaRequestContext taqdim etiladi .

Texnikaviylik. RequestContextTexnik jihatdan, bu o‘zgaruvchilar, agar siz foydalansangiz va 'django.contrib.auth.context_processors.auth'kontekst protsessori yoqilgan bo‘lsa, faqat shablon kontekstida mavjud bo‘ladi . U standart yaratilgan sozlamalar faylida joylashgan. Qo‘shimcha ma’lumot uchun RequestContext hujjatlariga qarang .

Foydalanuvchilar. Shablonni ko‘rsatishda RequestContexthozirda tizimga kirgan foydalanuvchi, misol User yoki AnonymousUsermisol, shablon o‘zgaruvchisida saqlanadi :{{ user }}

```
{% if user.is_authenticated %}
  <p>Welcome, {{ user.username }}. Thanks for logging in.</p>
```

```
{% else %}
  <p>Welcome, new user. Please log in.</p>
{% endif %}
```

RequestContext Agar ishlatilmayotgan bo‘lsa, bu shablon kontekst o‘zgaruvchisi mavjud emas .

Ruxsatlar. Hozirda tizimga kirgan foydalanuvchi ruxsatlari shablon o‘zgaruvchisida saqlanadi . Bu shablonga mos ruxsatnoma proksi-serverining namunasidir . {{ perms }}django.contrib.auth.context_processors.PermWrapper

Bitta atributli qidiruvni mantiqiy sifatida baholash proksi-server hisoblanadi. Masalan, tizimga kirgan foydalanuvchining ilovada ruxsati borligini tekshirish uchun :{{ perms }}User.has_module_perms()foo

```
{% if perms.foo %}
```

Ikki darajali atribut qidirishni mantiqiy sifatida baholash proksi-server hisoblanadi User.has_perm(). Masalan, tizimga kirgan foydalanuvchining ruxsati borligini tekshirish uchun foo.add_vote:

```
{% if perms.foo.add_vote %}
```

Shablondagi ruxsatlarni tekshirishning to‘liqroq misoli:

```
{% if perms.foo %}
  <p>You have permission to do something in the foo app.</p>
  {% if perms.foo.add_vote %}
    <p>You can vote!</p>
  {% endif %}
  {% if perms.foo.add_driving %}
    <p>You can drive!</p>
  {% endif %}
{% else %}
  <p>You don't have permission to do anything in the foo app.</p>
{% endif %}
```

Shuningdek, ruxsatlarni bayonotlar bo‘yicha ko‘rib chiqish mumkin . Masalan:{% if in %}

```
{% if 'foo‘ in perms %}
  {% if 'foo.add_vote' in perms %}
    <p>In lookup works, too.</p>
  {% endif %}
{% endif %}
```

Administratorda foydalanuvchilarni boshqarish. Ikkalasi ham mavjud django.contrib.admin va django.contrib.auth o‘rnatilgan bo‘lsa, administrator foydalanuvchilarni, guruhlarni va ruxsatlarni ko‘rish va boshqarishning qulay usulini taqdim etadi. Foydalanuvchilar har qanday Django modeli kabi yaratilishi va o‘chirilishi mumkin. Guruhlar yaratilishi va ruxsatlar foydalanuvchilar yoki guruhlarga berilishi mumkin. Administrator ichida qilingan modellarga foydalanuvchi tahrirlari jurnali ham saqlanadi va ko‘rsatiladi.

Foydalanuvchilarni yaratish. Asosiy administrator indeks sahifasining "Auth" bo‘limida "Foydalanuvchilar" havolasini ko‘rishingiz kerak. “Foydalanuvchi qo‘shish” administrator sahifasi standart administrator sahifalaridan farq qiladi, chunki u foydalanuvchining qolgan maydonlarini tahrirlashdan oldin foydalanuvchi nomi va parolni tanlashni talab qiladi.

Shuningdek, diqqat qiling: agar siz Django administrator sayti yordamida foydalanuvchi qayd yozuvi foydalanuvchi yaratish imkoniyatiga ega bo‘lishini istasangiz, ularga foydalanuvchilar qo‘shish va foydalanuvchilarni o‘zgartirishga ruxsat berishingiz kerak (ya’ni, “Foydalanuvchi qo‘shish” va “Foydalanuvchini o‘zgartirish” ruxsatlari) . Agar hisob foydalanuvchilarni qo‘shishga ruxsati bo‘lsa, lekin ularni o‘zgartirmasa, u hisob foydalanuvchilarni qo‘sha olmaydi. Nega? Chunki agar siz foydalanuvchilarni qo‘shishga ruxsatingiz bo‘lsa, siz o‘z navbatida boshqa foydalanuvchilarni o‘zgartirishi mumkin bo‘lgan super foydalanuvchilarni yaratish huquqiga egasiz. Shunday qilib, Django ozgina xavfsizlik chorasi sifatida ruxsatlarni qo‘shish va o‘zgartirishni talab qiladi.

Foydalanuvchilarga ruxsatlarni boshqarishga qanday ruxsat berishingiz haqida o‘ylab ko‘ring. Agar siz super foydalanuvchi bo‘lmagan foydalanuvchiga foydalanuvchilarni tahrirlash imkoniyatini bersangiz, bu oxir-oqibat ularga superuser maqomini berish bilan bir xil bo‘ladi, chunki ular foydalanuvchilarning, shu jumladan o‘zlarining ham ruxsatlarini oshirishlari mumkin!

Parollarni o‘zgartirish. Foydalanuvchi parollari administratorda ko‘rsatilmaydi (ma’lumotlar bazasida saqlanmaydi), lekin parolni saqlash tafsilotlari ko‘rsatiladi. Ushbu ma’lumot ekranida administratorlarga foydalanuvchi parollarini o‘zgartirish imkonini beruvchi parolni o‘zgartirish shakliga havola kiritilgan.

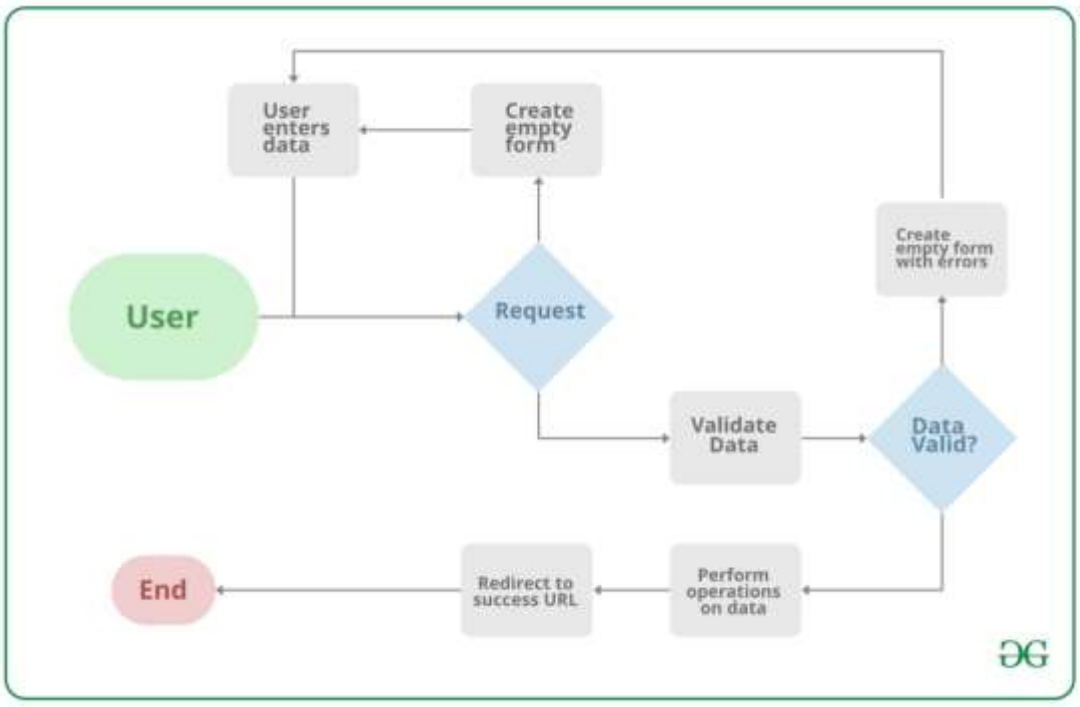
Nazorat savollari:

1. Djangoda standart User modeli nimaga xizmat qiladi?
2. Djangoda User modelini qanday ishlatish mumkin?
3. Djangoda User modelini moslashgan loyihada qanday ishlaymiz?
4. Djangoda User modelidagi ma’lumotlarni qanday o‘qish va yangilash mumkin?
5. Djangoda foydalanuvchi rolini qanday boshqarish mumkin?
6. Djangoda foydalanuvchining autentifikatsiyasini qanday amalga oshirish mumkin?
7. Djangoda foydalanuvchi parolini qanday tiklash mumkin?
8. Djangoda foydalanuvchi registratsiyasini qanday amalga oshirish mumkin?

10-§. DJANGODA FORMLAR TAYYORLASH.

Form klassi yaratilganda , eng muhim qismi shakl maydonlarini aniqlashdir. Har bir maydonda bir nechta boshqa ilgaklar bilan birga maxsus tekshirish mantig‘i mavjud. Django shakllari bilan bog‘liq turli xil xususiyatlar va texnikalar bilan bir shaklda foydalanish mumkin bo‘lgan turli sohalar atrofida aylanadi. Shakllar asosan foydalanuvchidan ma’lumot olish va ma’lumotlar bazalarida mantiqiy operatsiyalar uchun ma’lumotlardan foydalanish uchun ishlatiladi. Masalan, foydalanuvchini uning ismi, elektron pochtasi, paroli va boshqalar sifatida kiritish orqali ro‘yxatdan o‘tkazish.
Django Django formalarida belgilangan maydonlarni HTML kiritish maydonlariga xaritalaydi. Django shakllardagi ishning uchta alohida qismini boshqaradi:

- ma’lumotlarni ko‘rsatishga tayyor qilish uchun tayyorlash va qayta tuzish
- ma’lumotlar uchun HTML shakllarini yaratish
- mijozdan taqdim etilgan shakllar va ma’lumotlarni qabul qilish va qayta ishlash




```
from django import forms
# creating a form
class GeeksForm(forms.Form):
        title = forms.CharField()
        description = forms.CharField()
```

Django formasini yaratish. Djangoda shakl yaratish model yaratishga mutlaqo o‘xshaydi, shaklda qanday maydonlar va qaysi turdagi mavjud bo‘lishini belgilash kerak. Masalan, ro‘yxatga olish formasini kiritish uchun Ism (CharField), Roll Number (IntegerField) va boshqalar kerak bo‘lishi mumkin.

```
from django import forms
    class FormName(forms.Form):
     # each field would be mapped as an input field in HTML
    field_name = forms.Field(**options)
# import the standard Django Forms
from django import forms
# creating a form
class InputForm(forms.Form):
        first_name = forms.CharField(max_length = 200)
        last_name = forms.CharField(max_length = 200)
        roll_number = forms.IntegerField(
                                help_text = "Enter 6 digit roll
number"
                                )
        password = forms.CharField(widget = forms.PasswordInput())
```

Django forma maydonlarida ishlab chiquvchining ishini yengillashtirish uchun bir nechta o‘rnatilgan usullar mavjud, ammo ba'zida foydalanuvchi interfeysini (UI) sozlash uchun narsalarni qo‘lda amalga oshirish kerak bo‘ladi. Shaklda Django forma maydonlarini ko‘rsatish uchun ishlatilishi mumkin bo‘lgan 3 ta o‘rnatilgan usul mavjud.

- {{ form.as_table }} ularni <tr> teglari bilan o‘ralgan jadval kataklari sifatida ko‘rsatadi
- {{ form.as_p }} ularni <p> teglariga o‘ralgan holda ko‘rsatadi
- {{ form.as_ul }} ularni <li> teglariga o‘ralgan holda ko‘rsatadi

Ushbu shaklni ko‘rinishga aylantirish uchun views.py saytiga o‘ting va quyidagi tarzda home_view yarating.

```
from django.shortcuts import render
from .forms import InputForm
# Create your views here.
def home_view(request):
        context ={}
        context['form']= InputForm()
```

```
        return render(request, "home.html", context)
```

Ko‘rinishidan, forms.py da yuqorida yaratilgan forma sinfining namunasini yaratish kifoya. Endi shablonlarni tahrirlaymiz > home.html

```
<form action = "" method = "post">
        {% csrf_token %}
        {{form }}
        <input type="submit" value=Submit">
</form>
```

Modellardan Django formasini yaratish. Django ModelForm - bu modelni to‘g‘ridan-to‘g‘ri Django shakliga aylantirish uchun ishlatiladigan sinf. Agar siz ma’lumotlar bazasiga asoslangan ilova yaratayotgan bo‘lsangiz, ehtimol sizda Django modellari bilan yaqindan bog‘langan shakllar bo‘ladi. Endi loyihamiz tayyor bo‘lgach, models.py da model yarating.

```
# import the standard Django Model
# from built-in library
from django.db import models
        # declare a new model with a name "GeeksModel"
class GeeksModel(models.Model):
                # fields of the model
        title = models.CharField(max_length = 200)
        description = models.TextField()
        last_modified = models.DateTimeField(auto_now_add = True)
        img = models.ImageField(upload_to = "images/")
                        # renames the instances of the model
                # with their title name
        def __str__(self):
                return self.title
```

Ushbu model uchun to‘g‘ridan-to‘g‘ri shakl yaratish uchun forms.py saytiga kiring va quyidagi kodni kiriting:

```
# import form class from django
from django import forms
# import GeeksModel from models.py
from .models import GeeksModel
# create a ModelForm
class GeeksForm(forms.ModelForm):
        # specify the name of model to use
        class Meta:
                model = GeeksModel
                fields = "__all__"
```

Endilikda http://127.0.0.1:8000/ ga kiriladi.

Nazorat savollari:

1. Djangoda formalar nima?
2. Djangoda formalar tayyorlash uchun qanday qo‘llanmalar mavjud?
3. Djangoda ModelForm nima?
4. ModelForm ni qanday tuzish mumkin?
5. Djangoda formaga qanday maydonlar qo‘shish mumkin?
6. Djangoda formani ma’lumotlar bilan to‘ldirish va qaytarish qanday qo‘llaniladi?
7. Djangoda formani stil qilish (styling) mumkinmi?
8. Djangoda formaga validatorlarni qanday qo‘shish mumkin?

11-§. DJANGODA LOYIHA QURISH VA REGISTRATION, LOGIN, LOGOUT.

Django o‘rnatilgan foydalanuvchi ro‘yxatga olish shakli bilan birga keladi. Biz uni faqat ehtiyojlarimizga moslashtirishimiz kerak (ya’ni, ro‘yxatdan o‘tish paytida elektron pochta manzilini yig‘ish).

Ro‘yxatga olish shaklini yarating

env > mening saytim > asosiy > (Yangi fayl) forms.py

```
from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User
# Create your forms here.
class NewUserForm(UserCreationForm):
        email = forms.EmailField(required=True)
        class Meta:
                model = User
                fields = ("username", "email", "password1",
"password2")
        def save(self, commit=True):
                user = super(NewUserForm, self).save(commit=False)
                user.email = self.cleaned_data['email']
                if commit:
                        user.save()
                return user
```

UserCreationForm Django oldindan tuzilgan modelga ulanadigan oldindan tuzilgan ro‘yxatga olish formasi bilan birga keladi User. Biroq, UserCreationForm faqat foydalanuvchi nomi va parolni talab qiladi (password1 boshlang‘ich parol va password2 parolni tasdiqlash).

Oldindan tuzilgan shaklni sozlash uchun, avvalo , ilovalar katalogida *forms.py* nomli yangi fayl yarating .

Ushbu yangi fayl models.py va *views.py* bilan bir xil katalogda yaratilgan .

Keyin chaqirilgan yangi sinf ichida UserCreationForm ga qo‘ng‘iroq qiling NewUserFormva deb nomlangan boshqa maydonni qo‘shing email. Elektron pochtani foydalanuvchiga saqlang.

UserCreationFormga kerak bo‘lganda qo‘shimcha maydonlarni qo‘shing .

Register.html faylini yaratish. *env > mening saytim > asosiy > andozalar > asosiy > (Yangi fayl) register.html*

```
{% extends "main/header.html" %}
{% block content %}
{% load crispy_forms_tags %}
<!--Register-->
<div class="container py-5">
        <h1>Register</h1>
        <form method="POST">
                {% csrf_token %}
                {{ register_form|crispy }}
                <button class="btn btn-primary"
type="submit">Register</button>
        </form>
        <p class="text-center">If you already have an account, <a
href="/login">login</a> instead.</p>
</div>
{% endblock %}
```

Keyin ro‘yxatdan o‘tish shakli uchun HTML shablonini yarating. *Biz bu shablonni register.html* deb nomlaymiz . Shakl ba'zi qo‘shilgan CSS uchun Bootstrap va Django crispy shakllaridan foydalanadi. U shuningdek , “Django shablon teglarini kengaytirish va qo‘shish” bo‘limida topilgan *header.html* faylini kengaytiradi .

Ilovaga ro‘yxatdan o‘tish URL manzilini qo‘shing. *env > mening saytim > asosiy > urls.py*

```
from django.urls import path
from . import views
app_name = "main"
urlpatterns = [
   path("", views.homepage, name="homepage"),
   ...
   path("register", views.register_request, name="register")]
```

Endi ilovaning URL manzillariga ro‘yxatga olish yo‘lini qo‘shing, shunda biz unga ko‘rinishlarda murojaat qilishimiz mumkin. URL manzillari, shuningdek , Django Web App Beginners Cheat Sheet da joylashgan bosh sahifa URL namunasini ham o‘z ichiga oladi .

Ko‘rinishlarga registr funksiyasini qo‘shing. *env > mening saytim > asosiy > views.py*

```
from django.shortcuts import  render, redirect
from .forms import NewUserForm
from django.contrib.auth import login
from django.contrib import messages
def register_request(request):
        if request.method == "POST":
                form = NewUserForm(request.POST)
                if form.is_valid():
                        user = form.save()
                        login(request, user)
                        messages.success(request, "Registration
successful." )
                        return redirect("main:homepage")
                messages.error(request, "Unsuccessful registration.
Invalid information.")
        form = NewUserForm()
        return render (request=request,
template_name="main/register.html",
context={"register_form":form})
```

NewUserFormforms.py va dan import *qiling* . logindjango.contri b.authKeyn deb nomlangan yangi ko‘rish funksiyasini yozing register_request.

Funksiyada ikkita if/else iborasi mavjud . Birinchisi, shaklning joylashtirilganligini tekshiradi, ikkinchisi esa shaklning haqiqiyligini tekshiradi. Agar ikkalasi ham rost bo‘lsa, shakl ma’lumotlari foydalanuvchi ostida saqlanadi, foydalanuvchi tizimga kiradi va foydalanuvchi muvaffaqiyatli xabarni ko‘rsatuvchi bosh sahifaga yo‘naltiriladi.

Aks holda, ariza noto‘g‘ri bo‘lsa, xato xabari ko‘rsatiladi. Agar so‘rov birinchi navbatda POST bo‘lmasa, ya'ni birinchi if bayonoti "False" bo‘lsa, registr shablonida bo‘sh shaklni ko‘rsating.

E’tibor bering, agar siz Django loyihangizga xabarlar qo‘shmoqchi bo‘lsangiz, Xabarlar ramkasini yoqishingiz va *views.py* ning yuqori qismidagi xabarlarni import qilishingiz kerak .

Ro‘yxatdan o‘tish foydalanuvchisi funksiyasini sinab ko‘ring. Ro‘yxatdan o‘tish funksiyasi tugallangandan so‘ng brauzer oynasida http://127.0.0.1:8000/ register , ro‘yxatga olish URL manziliga o‘tishingiz mumkin.

Test foydalanuvchi hisobini yarating. To‘g‘ri bajarilgan bo‘lsa, siz "Ro‘yxatdan o‘tish muvaffaqiyatli" xabarini ko‘rsatadigan bosh sahifaga yo‘naltirilasiz. Endi Django ma'muriyatida foydalanuvchi ma’lumotlarini tekshiramiz.

Superuser yarating. *Terminal/Buyruqning satri*

```
(env) C:\Users\Owner\Desktop\Code\env\mysite>py manage.py
createsuperuser
Username (leave blank to use 'owner'): owner
Email address:
Password: *****
Password (again): *****
Superuser created successfully.
(env) C:\Users\Owner\Desktop\Code\env\mysite>py manage.py
runserver
```

Agar sizda hali yo‘q bo‘lsa, Django ma'muriyatiga kirish uchun super user hisobini yarating. Biz foydalanuvchi ma’lumotlar bazasiga to‘g‘ri qo‘shilganligini tekshiramiz.

Buyruqni py manage.py createsuperuserWindows buyruq satrida va python3 manage.py createsuperuserMac terminalida ishga tushiring. Keyin foydalanuvchi nomi va parolni to‘ldiring.

Foydalanuvchi Django administratorida ro‘yxatga olinganligini tekshiring. http://127.0.0.1:8000/admin/ URL manziliga o‘ting , u yerda siz Django ma'muriyati loginini ko‘rasiz. Agar siz hali ham test foydalanuvchisi sifatida tizimga kirgan bo‘lsangiz, "Siz testuser1 sifatida autentifikatsiya qilingansiz, lekin bu sahifaga kirish huquqiga ega emassiz. Boshqa hisobga kirishni xohlaysizmi?" degan ogohlantirish xabarini olasiz.

Superuser hisobingiz bilan administratorga kiring va "Foydalanuvchilar" tugmasini bosing. U yerda siz barcha foydalanuvchi nomlari va elektron pochta xabarlari ro‘yxatini va ularning xodimlarining holatini ko‘rishingiz kerak.

Foydalanuvchi logini. Endi siz biz yaratgan ko‘rish funksiyasi foydalanuvchini hisob yaratishda avtomatik ravishda tizimga kiritganini payqagan bo‘lishingiz mumkin. Biz foydalanuvchining erkin kirish imkoniyatiga ega bo‘lishini xohlaymiz. Shunday qilib, bizga kirish shablonini, URL manzili va ko‘rish funksiyasi kerak.

login.html faylini yarating. *env > mening saytim > asosiy > andozalar > asosiy > (Yangi fayl) login.html*

```
{% extends "main/header.html" %}
{% block content %}
{% load crispy_forms_tags %}
<!--Login-->
<div class="container py-5">
  <h1>Login</h1>
  <form method="POST">
    {% csrf_token %}
    {{ login_form|crispy }}
    <button class="btn btn-primary" type="submit">Login</button>
  </form>
  <p class="text-center">Don't have an account? <a
href="/register">Create an account</a>.</p>
</div>
{% endblock %}
```

HTML shablonining asosiy tuzilishi HTML registriga o‘xshaydi. Faqatgina farqlar - shakl va pastki qismdagi havola.

Kirish URL manzilini qo‘shing. *env > mening saytim > asosiy > urls.py*

```
from django.urls import path
from . import views
app_name = "main"
urlpatterns = [
    path("", views.homepage, name="homepage"),
    ...
    path("register", views.register_request, name="register"),
    path("login", views.login_request, name="login")
]
```

URL manzillariga kirish yo‘lini qo‘shing.

Ko‘rinishlarga kirish funksiyasini qo‘shing.*env > mening saytim > asosiy > views.py*

```
from django.shortcuts import  render, redirect
from .forms import NewUserForm
from django.contrib.auth import login, authenticate #add this
from django.contrib import messages
from django.contrib.auth.forms import AuthenticationForm #add
this
def register_request(request):
```

```
        ...
def login_request(request):
        if request.method == "POST":
                form = AuthenticationForm(request,
data=request.POST)
                if form.is_valid():
                        username = form.cleaned_data.get('username')
                        password = form.cleaned_data.get('password')
                        user = authenticate(username=username,
password=password)
                        if user is not None:
                                login(request, user)
                                messages.info(request, f"You are now
logged in as {username}.")
                                return redirect("main:homepage")
                        else:
                                messages.error(request,"Invalid username
or password.")
                else:
                        messages.error(request,"Invalid username or
password.")
        form = AuthenticationForm()
        return render(request=request,
template_name="main/login.html", context={"login_form":form})
```

Endi *views.py* saytiga qayting va authenticateimport formasi ro‘yxatiga qo‘shing django.contrib.auth, so‘ng faylning yuqori qismidan AuthenticationFormimport qiling.django.contrib.auth.forms

AuthenticationForm - bu foydalanuvchiga kirish uchun oldindan tuzilgan Django formasidir.

Kirish funksiyangizni yozish uchun Django funksiyasidan foydalanadigan if/else iborasini qo‘shing authenticate(). Bu funksiya foydalanuvchi hisob ma’lumotlarini (foydalanuvchi nomi va parol) tekshirish va backendda saqlangan to‘g‘ri Foydalanuvchi obyektini qaytarish uchun ishlatiladi.

Agar server hisob ma’lumotlarini autentifikatsiya qilgan bo‘lsa, funksiya login()autentifikatsiya qilingan foydalanuvchiga kirish uchun Djangoni ishga tushiradi. Aks holda, agar foydalanuvchi

autentifikatsiya qilinmagan bo‘lsa, u foydalanuvchiga noto‘g‘ri foydalanuvchi nomi yoki parolni kiritganligi haqida xabar qaytaradi.

Ikkinchi else bayonoti, agar shakl noto‘g‘ri bo‘lsa, u shunga o‘xshash xato xabarini qaytaradi. Yakuniy else bayonoti, agar so‘rov POST bo‘lmasa, login HTML shablonidagi bo‘sh shaklni qaytaring.

Foydalanuvchiga kirish funksiyasini sinab ko‘ring. Endi login URL manziliga o‘ting, http://127.0.0.1:8000/login va test foydalanuvchingizga kiring. Agar siz to‘g‘ri foydalanuvchi nomi va parolni kiritgan bo‘lsangiz, muvaffaqiyat haqida xabar olasiz va tizimga kirasiz. Hozircha login faqat Djangoning o‘rnatilgan UserCreationForm orqali mavjud, biroq siz ro‘yxatdan o‘tish oqimlarini tezda boshqarish va yaratish uchun Djangoalluth-dan osongina foydalanishingiz mumkin . Bu shuni anglatadiki, foydalanuvchilar o‘zlarining ijtimoiy media akkauntlari yordamida foydalanuvchi hisobini yaratishlari mumkin. Siz ushbu o‘quv darsliksini shu yerda kuzatishingiz mumkin .

Foydalanuvchi tizimdan chiqish. Biz hal qilishimiz kerak bo‘lgan oxirgi narsa - foydalanuvchi tizimdan chiqish. Biz chiqish havolasini navigatsiya paneliga joylashtiramiz, lekin u faqat autentifikatsiya qilingan bo‘lsa (ya'ni tizimga kirgan bo‘lsa) foydalanuvchi tomonidan ko‘rinadi.

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="#">Navbar</a>
  <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbarText" aria-
controls="navbarText" aria-expanded="False" aria-label="Toggle
navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarText">
    <ul class="navbar-nav mr-auto">
      {% if user.is_authenticated %}
          <li class="nav-item">
        <a class="nav-link" href="/logout">Logout</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Welcome, {{user.username}}</a>
      </li>
```

```
      {% else %}
      <li class="nav-item">
        <a class="nav-link" href="/login">Login</a>
      </li>
      {% endif %}
    </ul>
  </div>
</nav>
```

Django shablon o‘zgaruvchisi {{ user }}joriy kirgan foydalanuvchini saqlaydi va ularning ruxsatlari shablon kontekstida mavjud.

Biz qilishimiz kerak bo‘lgan yagona narsa, agar foydalanuvchi autentifikatsiya qilingan bo‘lsa, tizimdan chiqish havolasini va uning foydalanuvchi nomini, aks holda kirish havolasini ko‘rsatadigan if/else iborasini qo‘shishdir. Bu Bootstrap hujjatlaridagi asosiy navigatsiya paneli. Agar siz uni yanada moslashtirmoqchi bo‘lsangiz, 10-moddaga qarang Custom Bootstrap Navbars .

Chiqish URL manzilini qo‘shing. *env > mening saytim > asosiy > urls.py*

```
from django.urls import path
from . import views
app_name = "main"
urlpatterns = [
    path("", views.homepage, name="homepage"),
    ...
    path("register", views.register_request, name="register"),
    path("login", views.login_request, name="login"),
    path("logout", views.logout_request, name= "logout"),
]
```

Endi ilovaning URL manzillariga chiqish URL manzilini qo‘shing.

Chiqish funksiyasini qo‘shing. *env > mening saytim > asosiy > views.py*

```
from django.shortcuts import  render, redirect
from .forms import NewUserForm
from django.contrib.auth import login, authenticate, logout #add this
from django.contrib import messages
```

```
from django.contrib.auth.forms import AuthenticationForm
def register_request(request):
        ...
def login_request(request):
        ...
def logout_request(request):
        logout(request)
        messages.info(request, "You have successfully logged out.")
        return redirect("main:homepage")
```

Nihoyat, ko‘rishlar fayliga chiqish funksiyasini qo‘shing. Funksiya logout_request Django funksiyasidan logout()foydalanuvchini o‘z hisobidan chiqish va chiqish URL manzili so‘ralganda ularni bosh sahifaga yo‘naltirish uchun foydalanadi.

Nazorat savollari:

1. Djangoda loyiha qurish uchun qadamlar nimalardir?
2. Djangoda foydalanuvchi ro‘yhatdan o‘tkazish (registration) qanday amalga oshiriladi?
3. Foydalanuvchini tizimga kirish (login) uchun Django qanday qo‘llaniladi?
4. Foydalanuvchi tizimdan chiqish (logout) uchun qanday jarayonlar o‘tkaziladi?
5. Django loyihasida foydalanuvchining kirish va chiqishini qanday amalga oshirish mumkin?
6. Djangoda foydalanuvchining kirish holatini tekshirish (authentication) qanday amalga oshiriladi?
7. Foydalanuvchining kirish ma’lumotlarini saqlash uchun Django qanday asboblar taklif qiladi?
8. Django loyihada foydalanuvchining rolini belgilash va boshqarish mumkin?

12-§. DJANGODA LOYIHANI TESTLASH.

Sinov turlari. Birlik va integratsiya testlarning ikkita asosiy turidir:

- *Birlik testlari* - bu ma'lum bir funksiyani sinab ko‘radigan izolyatsiya qilingan testlar.
- *Integratsiya testlari* , shu bilan birga, foydalanuvchi xatti-harakatlariga va butun ilovalarni sinab ko‘rishga qaratilgan kattaroq testlardir. Boshqacha qilib aytganda, integratsiya testi o‘zini to‘g‘ri tutishiga ishonch hosil qilish uchun kod funksiyalarining turli qismlarini birlashtiradi.

Birlik testlariga e'tibor qarating. Bulardan KO‘P yozing. Ushbu testlarni yozish va disk raskadrovka qilish va integratsiya testlariga nisbatan ancha oson va qancha ko‘p bo‘lsa, integratsiya testlari shunchalik kam bo‘ladi. Birlik sinovlari tez bo‘lishi kerak. Sinovlarni tezlashtirishning bir necha usullarini ko‘rib chiqamiz.

Ya’ni, integratsiya testlari ba'zida birlik testlari bilan qamrab olingan bo‘lsa ham, zarur bo‘ladi, chunki integratsiya testlari kod regressiyalarini aniqlashga yordam beradi .

Umuman olganda, testlar Muvaffaqiyat (kutilgan natijalar), Muvaffaqiyatsizlik (kutilmagan natijalar) yoki xatoga olib keladi. Siz nafaqat kutilgan natijalarni sinab ko‘rishingiz, balki sizning kodingiz kutilmagan natijalarni qanchalik yaxshi ishlashini ham sinab ko‘rishingiz kerak.

Agar u sinishi mumkin bo‘lsa, uni sinab ko‘rish kerak. Bunga modellar, ko‘rinishlar, shakllar, andozalar, validatorlar va boshqalar kiradi.

1. Har bir test odatda faqat bitta funksiyani sinab ko‘rishi kerak.
2. Oddiy qilib qo‘ying. Boshqa testlar ustiga test yozishni xohlamaysiz.
3. Ishlab chiqarishga o‘tishdan oldin, kod repodan PULLed yoki PUSHed va bosqichlash muhitida sinovlarni o‘tkazing.
4. Djangoning yangi versiyasiga yangilashda:
   - lokal darajada yangilash,
   - test to‘plamini ishga tushiring,
   - xatolarni tuzatish,
   - Repo va sahnalashtirishga PUSH, keyin esa
   - kodni jo‘natishdan oldin bosqichma-bosqich sinovdan o‘tkazing.

Sinovlaringizni loyihangizga mos keladigan tarzda tuzing. *Men har bir ilova uchun barcha testlarni tests.py* fayliga qo‘yishni va

testlarni o‘zim sinab ko‘rayotgan narsalarim bo‘yicha guruhlashni ma’qul ko‘raman, masalan, modellar, ko‘rinishlar, shakllar va boshqalar.

Shuningdek, tests.py faylini butunlay chetlab o‘tishingiz (o‘chirib tashlashingiz) va testlaringizni har bir ilovada shu tarzda tuzishingiz mumkin:

```
└── app_name
    └── tests
        ├── __init__.py
        ├── test_forms.py
        ├── test_models.py
        └── test_views.py
```

Va nihoyat, har bir ilova papkasida *tests.py* faylini joylashtirgan holda, butun loyiha tuzilmasini aks ettiruvchi alohida test papkasini yaratishingiz mumkin .

Kattaroq loyihalar oxirgi tuzilmalardan birini qo‘llashi kerak. Agar siz kichikroq loyihangiz oxir-oqibat kattaroq narsaga aylanishini bilsangiz, oxirgi ikkita tuzilmadan birini qo‘llash yaxshidir. Men birinchi va uchinchi tuzilmalarni ma'qul ko‘raman, chunki ularning barchasi bitta skriptda ko‘rish mumkin bo‘lsa, har bir ilova uchun testlarni ishlab chiqish osonroq.

Sinov to‘plamini yozish va ishga tushirishda yordam berish uchun quyidagi paketlar va kutubxonalardan foydalaning:

- Djangowebtest : oxirgi foydalanuvchi tajribasiga mos keladigan funktsional testlar va tasdiqlarni yozishni ancha osonlashtiradi. Shablonlar va ko‘rinishlarni to‘liq qamrab olish uchun ushbu testlarni Selenium testlari bilan birlashtiring.
- qamrov : testlar samaradorligini o‘lchash uchun ishlatiladi, testlar bilan qamrab olingan kod bazangiz foizini ko‘rsatadi. Agar siz endigina birlik testlarini o‘rnatishni boshlayotgan bo‘lsangiz, qamrov nima sinovdan o‘tkazilishi kerakligi haqida takliflar berishga yordam beradi. Sinovni o‘yinga aylantirish uchun qamrov ham ishlatilishi mumkin: masalan, men har kuni testlar qamrab olgan kod foizini oshirishga harakat qilaman.
- Djangodiscover-runner : agar siz ularni boshqacha tarzda tashkil qilsangiz (masalan, *tests.py* dan tashqarida) testlarni topishga yordam beradi. Shunday qilib, agar siz testlarni yuqoridagi misoldagi kabi

alohida papkalarga ajratsangiz, testlarni topish uchun Discover-runner-dan foydalanishingiz mumkin.

- factory_boy , model_mommy va mock : barchasi sinov uchun kerakli ma'lumotlarni to'ldirish uchun armatura yoki ORM o'rniga ishlatiladi . Har ikkala armatura va ORM sekin bo'lishi mumkin va modelingiz o'zgarganda yangilanishi kerak.

Ushbu asosiy misolda biz sinovdan o'tamiz:

- modellar,
- ko'rishlar,
- shakllari va
- API.

Davom etish uchun Github repo ni yuklab oling.

Sozlash va o'rnatish. Modullarni o'rnating va uni INSTALLED_APPS ilovangizga qo'shing:

```
$ pip install coverage==3.6
```

Hisobot natijalarini ko'rish uchun *django15/htmlcov/index.html* ni oching. Hisobotning pastki qismiga o'ting. Virtualenv jildidagi barcha qatorlarni o'tkazib yuborishingiz mumkin. Hech qachon o'rnatilgan Python funksiyasi yoki kutubxonasi bo'lgan narsalarni sinab ko'rmang, chunki ular allaqachon sinovdan o'tgan. Hisobot ishga tushirilgandan so'ng uni tozalash uchun virtualenvni jilddan ko'chirishingiz mumkin.

Keling, modellarni sinab ko'rishdan boshlaylik.

Sinov modellari. Qamrov hisobotida "nima bo'lishidan qat'iy nazar/modellar" uchun havolani bosing. Siz ushbu ekranni ko'rishingiz kerak:


```
1  from django.db import models
2
3  class Whatever(models.Model):
4          title = models.CharField(max_length=200)
5          body = models.TextField()
6          created_at = models.DateTimeField(auto_now_add=True)
7
8          def __unicode__(self):
9                  return self.title
```

« index coverage.py v3.6

Asosan, ushbu hisobot biz kirish nomini sinab ko‘rishimiz kerakligini ko‘rsatadi. Oddiy.

tests.py ni oching va quyidagi kodni qo‘shing:

```
from django.test import TestCase
from whatever.models import Whatever
from django.utils import timezone
from django.core.urlresolvers import reverse
from whatever.forms import WhateverForm
# models test
class WhateverTest(TestCase):
    def create_whatever(self, title="only a test", body="yes, this is only a test"):
        return Whatever.objects.create(title=title, body=body, created_at=timezone.now())
    def test_whatever_creation(self):
        w = self.create_whatever()
        self.assertTrue(isinstance(w, Whatever))
        self.assertEqual(w.__unicode__(), w.title)
```

Bu yerda nima bo‘lyapti? Biz Whateverobyektni yaratdik va yaratilgan sarlavha kutilgan sarlavhaga mos kelishini sinab ko‘rdik - bu shunday qildi.

Eslatma: Funksiya nomlari bilan boshlanganiga ishonch hosil qiling test_, bu nafaqat umumiy konventsiya, balki Djangodiscover-runner testni topishi uchun ham. Shuningdek, modelingizga qo‘shgan *barcha usullar uchun testlarni yozing.*

Qayta ishlash qamrovi:

```
$ coverage run manage.py test whatever -v 2
```

Sinovdan o‘tganligini ko‘rsatadigan quyidagi natijalarni ko‘rishingiz kerak:

```
test_whatever_creation (whatever.tests.WhateverTest) ... ok
----------------------------------------------------
Ran 1 test in 0.002s
OK
```

Keyin qamrov hisobotiga yana qarasangiz, modellar endi 100% bo‘lishi kerak.

Ko‘rishlarni sinab ko‘rish

Ko‘rinishlarni sinab ko‘rish ba'zan qiyin bo‘lishi mumkin. Men odatda holat kodlarini tekshirish uchun birlik testlaridan, shuningdek, AJAX, Javascript va boshqalarni sinab ko‘rish uchun Selenium Webdriver-dan foydalanaman.

tests.py dagi WhateverTest sinfiga quyidagi kodni qo‘shing :

```
# views (uses reverse)

    def test_whatever_list_view(self):
        w = self.create_whatever()
        url = reverse("whatever.views.whatever")
        resp = self.client.get(url)

        self.assertEqual(resp.status_code, 200)
        self.assertIn(w.title, resp.content)
```

Bu yerda biz mijozdan URL-manzilni olamiz, natijalarni o‘zgaruvchida saqlaymiz respva keyin tasdiqlarimizni sinab ko‘ramiz. Birinchidan, biz javob kodi 200 yoki yo‘qligini tekshiramiz va keyin haqiqiy javobni sinab ko‘ramiz. Siz quyidagi natijalarni olishingiz kerak:

```
test_whatever_creation (whatever.tests.WhateverTest) ... ok
test_whatever_list_view (whatever.tests.WhateverTest) ... ok


----------------------------------------------------
Ran 2 tests in 0.052s

OK
```

Hisobotingizni qayta ishga tushiring. Endi siz quyidagi natijalarni ko‘rsatadigan "nima bo‘lishidan qat'iy nazar/ko‘rishlar" uchun havolani ko‘rishingiz kerak:

Bundan tashqari, biror narsa muvaffaqiyatsiz bo‘lishiga ishonch hosil qilish uchun testlarni yozishingiz mumkin. Misol uchun, agar foydalanuvchi yangi obyekt yaratish uchun tizimga kirishi kerak bo‘lsa, u obyektni yarata olmasa, sinov muvaffaqiyatli bo‘ladi.
Keling, tez selen testini ko‘rib chiqaylik:

```
# views (uses selenium)

import unittest
from selenium import webdriver
class TestSignup(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
    def test_signup_fire(self):
        self.driver.get("http://localhost:8000/add/")
        self.driver.find_element_by_id('id_title').send_keys("test title")
        self.driver.find_element_by_id('id_body').send_keys("test body")
        self.driver.find_element_by_id('submit').click()
        self.assertIn("http://localhost:8000/", self.driver.current_url)
    def tearDown(self):
        self.driver.quit
if __name__ == '__main__':
```

```
    unittest.main()
$ pip install selenium==2.33.0
```

Sinovlarni bajaring. Firefox yuklanishi kerak (agar u o‘rnatilgan bo‘lsa) va sinovni o‘tkazing. Keyin biz to‘g‘ri sahifa taqdim etilgandan keyin yuklanganligini tasdiqlaymiz. Shuningdek, yangi obyekt ma’lumotlar bazasiga qo‘shilganligini tekshirishingiz mumkin.

Sinov shakllari. Quyidagi usullarni qo‘shing:

```
def test_valid_form(self):
    w = Whatever.objects.create(title='Foo‘, body='Bar')
    data = {'title': w.title, 'body': w.body,}
    form = WhateverForm(data=data)
    self.assertTrue(form.is_valid())
def test_invalid_form(self):
    w = Whatever.objects.create(title='Foo‘, body='')
    data = {'title': w.title, 'body': w.body,}
    form = WhateverForm(data=data)
    self.assertFalse(form.is_valid())
```

JSON-dan shakl uchun ma’lumotlarni qanday yaratayotganimizga e'tibor bering. Bu armatura.

Endi sizda 5 ta o‘tish testi bo‘lishi kerak:

```
test_signup_fire (whatever.tests.TestSignup) ... ok
test_invalid_form (whatever.tests.WhateverTest) ... ok
test_valid_form (whatever.tests.WhateverTest) ... ok
test_whatever_creation (whatever.tests.WhateverTest) ... ok
test_whatever_list_view (whatever.tests.WhateverTest) ... ok

Ran 5 tests in 12.753s

OK
```

Shaklning o‘zida tekshirgichlar asosida ma'lum xato xabari ko‘rsatilishini tasdiqlovchi testlarni ham yozishingiz mumkin.

API sinovdan o‘tkazilmoqda

Birinchidan, APIga ushbu URL orqali kirishingiz mumkin: http://localhost:8000/api/whatever/?format=json . Bu oddiy sozlash, shuning uchun testlar ham juda oddiy bo‘ladi.

lxml va zararsizlantirilgan XML ni o‘rnating:

```
$ pip install lxml==3.2.3
$ pip install defusedxml==0.4.1
```

Quyidagi test holatlarini qo‘shing:

```
from tastypie.test import ResourceTestCase
class EntryResourceTest(ResourceTestCase):
    def test_get_api_json(self):
        resp = self.api_client.get('/api/whatever/', format='json')
        self.assertValidJSONResponse(resp)
    def test_get_api_xml(self):
        resp = self.api_client.get('/api/whatever/', format='xml')
        self.assertValidXMLResponse(resp)
```

Siz shunchaki har bir holatda javob olishingiz kerak bo‘ladi.

Nazorat savollari:

1. Djangoda loyihani testlash nima uchun kerak?
2. Django testlari qanday yoziladi?
3. Django testlari uchun qo‘llanmalar yoki kutubxanalar mavjudmi?
4. Test funksiyalarini Djangoda qanday nomlash mumkin?
5. Djangoda faqat bazi modellarni test qilish mumkinmi?
6. Djangoda test holatini tekshirish uchun qanday asboblar mavjud?
7. Djangoda test javoblari qanday tekshiriladi?
8. Django test holatini avtomatlashtirish va integratsiya testlari qanday ishlaydi?

13--§. DJANGODA TAYYORLANGAN MODELLAR UCHUN ADMIN PANELDA SEARCH VA FILTER FUNKSIYALARINI TAYYORLASH.

Agar siz loyihangizning turli jihatlarini qidirish yoki filtrlashni xohlasangiz, Django administrator sahifasi juda foydali.

Katta miqyosli loyihalar uchun **qidiruv** va **filtrlash** qobiliyatlari tezkor qidirish, tezroq yangilash va samaradorlikni oshirish imkonini beradi.

Aytaylik, biz allaqachon veb-saytimiz uchun Django administrator indeksini o‘rnatdik. Qidiruv va ko‘rsatish xususiyatlarini o‘z ichiga olmasdan, administrator konsoli bizga faqat barcha foydalanuvchilar ro‘yxatini ko‘rsatadi.

Indeks juda cheklangan ma’lumotlarni beradi va foydalanuvchilarni qidirishda yordam beradigan qidiruv paneli mavjud emas. Bu bir nechta foydalanuvchilar bilan ishlashda katta muammo bo‘lishi mumkin.

Qo‘shimcha ma’lumotlarni qidirish va ko‘rsatishga ruxsat berish uchun biz bilan bog‘langan maydonlar list_displayendi har bir foydalanuvchi uchun administrator konsolida ko‘rinadi.

quyidagilarni qo‘shish orqali administrator kodini yangilashimiz kerak:

```
list_display = ('email', 'username', 'date_joined', 'last_login',
'is_admin', 'is_staff')
search_fields = ('email', 'username')
```

Ma’lumotlar bazasi bilan bog‘langan maydonlar uchun so‘rovni amalga oshiradigan qidiruv paneli ham paydo bo‘ladi search_display.

Endi administrator konsolida foydalanuvchilar haqida ko‘proq ma’lumotga egamiz. Bundan tashqari, foydalanuvchi nomi va elektron pochta manzillari asosida foydalanuvchilarni osongina qidirishimiz mumkin.

Filtrlash. Qidiruv foydali xususiyatdir. Biroq, Django ramkasi qidirilayotgan element qaysi maydonga tegishli ekanligini bilmaydi. Qidiruvingizga moslashish uchun u ro‘yxatda keltirilgan barcha maydonlarni qidiradi search_fields. Bu sekin va mashaqqatli jarayon.

ListFilterUshbu muammoni bartaraf etish uchun siz qo‘llashni tanlagan filtrlar asosida qidiruv maydonini kamaytiradigan dasturni amalga oshirishimiz mumkin.

```
list_filter = ('username',)
```
Ro‘yxat filtrlarini to‘g‘ridan-to‘g‘ri faollashtirishingiz mumkin:

Eslatma: Bu faqat mavjud modellar uchun ro‘yxat filtrlarini faollashtiradi.

Ba’zida bizning atributlarimiz faqat ma'lum miqdordagi toifalarga ega. Bunday hollarda, biz intuitivroq ko‘rinishi uchun atributlar uchun variantlarni bog‘lashga moyilmiz. Gender atributiga oid vaziyatni ko‘rib chiqing.

Biz filtrimizni quyidagicha qo‘shishimiz mumkin:

```
choices_gender = [
    (0, 'male'),
    (1, 'female'),
]
```

Nazorat savollari

1. Djangoda loyihani testlash nima uchun kerak?
2. Django testlari qanday yoziladi?
3. Django testlari uchun qo‘llanmalar yoki kutubxanalar mavjudmi?
4. Test funksiyalarini Djangoda qanday nomlash mumkin?
5. Djangoda faqat bazi modellarni test qilish mumkinmi?
6. Djangoda test holatini tekshirish uchun qanday asboblar mavjud?
7. Djangoda test javoblari qanday tekshiriladi?
8. Django test holatini avtomatlashtirish va integratsiya testlari qanday ishlaydi?

14--§. DJANGODA MIXINS VA MESSAGES KUTUBXONALARI.

Xabarlar doirasi. Ko‘pincha veb-ilovalarda ariza yoki foydalanuvchi kiritishining boshqa turlarini qayta ishlagandan so‘ng foydalanuvchiga bir martalik bildirishnoma xabarini (shuningdek, "flesh xabar" deb ham ataladi) ko‘rsatishingiz kerak.

Buning uchun Django anonim va autentifikatsiya qilingan foydalanuvchilar uchun cookie-fayl va seansga asoslangan xabar almashishni to‘liq qo‘llab-quvvatlaydi. Xabarlar tizimi xabarlarni vaqtinchalik bitta so‘rovda saqlash va ularni keyingi so‘rovda (odatda keyingi so‘rovda) ko‘rsatish uchun olish imkonini beradi. **Level** har bir xabar uning ustuvorligini belgilaydigan o‘ziga xos belgi bilan belgilanadi (masalan, **info**, **warning**, yoki **error**).

Xabarlarni yoqish. Xabarlar o‘rta dastur sinfi va tegishli kontekst protsessori orqali amalga oshiriladi .

settings.py tomonidan yaratilgan standart xabar funksiyasini yoqish uchun zarur bo‘lgan barcha sozlamalarni o‘z ichiga oladi:**Djangoadmin startproject**

- 'django.contrib.messages' ichida joylashgan INSTALLED_APPS.
- MIDDLEWARE'django.contrib.sessions.middleware.SessionMidd leware'va o‘z ichiga oladi 'django.contrib.messages .middleware. MessageMiddleware'.

Odatiy saqlash serveri seanslarga tayanadi . Shuning uchun Session Middleware faollashtirilgan bo‘lishi va Message Middlewareichida oldin paydo bo‘lishi kerak MIDDLEWARE.

- Sozlamangizda belgilangan backend opsiyasi 'context_processors'ni o‘z ichiga oladi .DjangoTemplatesTEMPLATES'django.contrib.messages.contex t_processors.messages'

Agar siz xabarlardan foydalanishni istamasangiz, 'django.contrib.messages'o‘zingizning dan INSTALLED_APPS, MessageMiddleware qatorni dan MIDDLEWAREva messages kontekst protsessorini dan olib tashlashingiz mumkin TEMPLATES.

Xabar mexanizmini sozlash. Xabarlar ramkasi vaqtinchalik xabarlarni saqlash uchun turli xil backendlardan foydalanishi mumkin. Django uchta o‘rnatilgan saqlash sinfini taqdim etadi **django.contrib.messages**:

class storage.session.SessionStorage

Bu sinf so‘rov sessiyasi ichidagi barcha xabarlarni saqlaydi. Shuning uchun u Django **contrib.sessions** ilovasini talab qiladi.

class storage.cookie.CookieStorage

Bu sinf so‘rovlar bo‘ylab bildirishnomalarni saqlab turish uchun xabar ma’lumotlarini cookie faylida (manipulyatsiyani oldini olish uchun maxfiy xesh bilan imzolangan) saqlaydi. Agar cookie ma’lumotlarining hajmi 2048 baytdan oshsa, eski xabarlar o‘chiriladi.

class storage.fallback.FallbackStorage

Bu sinf avval dan foydalanadi va bitta cookie fayliga sig‘maydigan xabarlar uchun CookieStorage foydalanishga qaytadi. SessionStorage Bu Django contrib.sessions ilovasini ham talab qiladi.

Bu xatti-harakat imkon qadar sessiyaga yozishdan qochadi. U umumiy holatda eng yaxshi ishlashni ta'minlashi kerak.

FallbackStorage standart saqlash sinfidir. Agar u sizning ehtiyojlaringizga mos kelmasa, MESSAGE_STORAGE to‘liq import yo‘liga o‘rnatib, boshqa saqlash sinfini tanlashingiz mumkin, masalan:

MESSAGE_STORAGE
"django.contrib.messages.storage.cookie.CookieStorage"

sinf storage.base.BaseStorage

O‘z saqlash sinfingizni yozish uchun BaseStorage sinfni pastki sinfga kiriting django.contrib.messages.storage.base va _get va _store usullarini amalga oshiring.

Xabar darajalari. Xabarlar ramkasi Python logging moduliga o‘xshash sozlanishi darajadagi arxitekturaga asoslangan. Xabar darajalari xabarlarni turlari bo‘yicha guruhlash imkonini beradi, shuning uchun ularni ko‘rinish va shablonlarda filtrlash yoki boshqacha ko‘rsatish mumkin. To‘g‘ridan-to‘g‘ri import qilinadigan o‘rnatilgan darajalar **django.contrib.messages** quyidagilardir:

Doimiy	**Maqsad**
DEBUG	Ishlab chiqarishni joylashtirishda e’tiborga olinmaydigan (yoki o‘chiriladigan) ishlab chiqish bilan bog‘liq xabarlar
INFO	Foydalanuvchi uchun ma’lumotli xabarlar

SUCCESS	Harakat muvaffaqiyatli bajarildi, masalan, “Profilingiz muvaffaqiyatli yangilandi”
WARNING	Muvaffaqiyatsizlik yuz bermadi, lekin yaqin orada bo‘lishi mumkin
ERROR	Harakat muvaffaqiyatli bo‘lmadi **yoki** boshqa xatolik yuz berdi

Sozlama minimal yozilgan darajani o‘zgartirish uchun ishlatilishi mumkin yoki har bir so‘rov bo‘yicha o‘zgartirilishi MESSAGE_LEVEL mumkin. Bundan pastroq darajadagi xabarlarni qo‘shishga urinishlar e'tiborga olinmaydi.

Xabar teglari. Xabar teglari - bu xabar darajasining satrli ko‘rinishi va to‘g‘ridan-to‘g‘ri ko‘rinishga qo‘shilgan har qanday qo‘shimcha teglar (batafsilroq ma’lumot uchun quyidagi qo‘shimcha xabar teglarini qo‘shish bo‘limiga qarang). Teglar satrda saqlanadi va bo‘shliqlar bilan ajratiladi. Odatda xabar teglari xabar turiga qarab xabar uslubini sozlash uchun CSS sinflari sifatida ishlatiladi. Odatiy bo‘lib, har bir darajada o‘z doimiysining kichik harfli versiyasi bo‘lgan bitta teg mavjud:

Doimiy daraja	**Teg**
DEBUG	Debug
INFO	Info
SUCCESS	Success
WARNING	warning
ERROR	Error

Xabar darajasi uchun standart teglarni o‘zgartirish uchun (o‘rnatilgan yoki moslashtirilgan), sozlamani **MESSAGE_TAGS**o‘zgartirmoqchi bo‘lgan darajalarni o‘z ichiga olgan lug‘atga o‘rnating. Bu standart teglarni kengaytirganda, siz faqat bekor qilmoqchi bo‘lgan darajalar uchun teglarni taqdim etishingiz kerak:

```
from django.contrib.messages import constants as messages
MESSAGE_TAGS = {
   messages.INFO: "",
   50: "critical",}
```

Ko‘rinishlar va shablonlarda xabarlardan foydalanish

add_message(*so‘rov* , *daraja* , *xabar* , *extra_tags = ''* , *fail_silently = False*)

Xabar qo‘shish. Xabar qo‘shish uchun qo‘ng‘iroq qiling:

```
from django.contrib import messages
messages.add_message(request, messages.INFO, "Hello world.")
```

Ba’zi yorliq usullari tez-tez ishlatiladigan teglar bilan xabarlarni qo‘shishning standart usulini ta'minlaydi (ular odatda xabar uchun HTML sinflari sifatida taqdim etiladi):

```
messages.debug(request, "%s SQL statements were executed." % count)
messages.info(request, "Three credits remain in your account.")
messages.success(request, "Profile details updated.")
messages.warning(request, "Your account expires in three days.")
messages.error(request, "Document deleted.")
```

Xabarlarni ko‘rsatish. get_messages(*so‘rov*)

Shabloningizda quyidagi kabi foydalaning:

```
{% if messages %}
<ul class="messages">
    {% for message in messages %}
    <li{% if message.tags %} class="{{ message.tags }}"{% endif %}>{{ message }}</li>
    {% endfor %}
</ul>
{% endif %}
```

Agar kontekst protsessoridan foydalanayotgan bo‘lsangiz, shabloningiz bilan ko‘rsatilishi kerak **RequestContext**. Aks holda, **messages**andoza kontekstida ishonch hosil qiling.

Agar siz faqat bitta xabar borligini bilsangiz ham, ketma-ketlikni takrorlashingiz kerak **messages**, chunki aks holda xabarlar xotirasi keyingi so‘rov uchun tozalanmaydi.

Kontekst protsessori, shuningdek, **DEFAULT_MESSAGE_LEVELS** xabar darajasidagi nomlarning raqamli qiymatiga mos keladigan o‘zgaruvchini taqdim etadi:

```
{% if messages %}
<ul class="messages">
    {% for message in messages %}
    <li{% if message.tags %} class="{{ message.tags }}"{% endif %}>
        {% if message.level == DEFAULT_MESSAGE_LEVELS.ERROR %}Important: {% endif %}
```

```
        {{ message }}
    </li>
    {% endfor %}
</ul>
{% endif %}
```

Shablonlardan tashqari siz foydalanishingiz mumkin **get_messages()**:

```
from django.contrib.messages import get_messages
storage = get_messages(request)
for message in storage:
    do_something_with_the_message(message)
```

Masalan, siz barcha xabarlarni o‘rniga JSONResponseMixin da qaytarishingiz mumkin TemplateResponseMixin.

get_messages() konfiguratsiya qilingan xotira orqa qismining nusxasini qaytaradi.

class Message *_class* storage.base.Message

Shablondagi xabarlar ro‘yxatini aylanib chiqsangiz, siz sinfning namunalari olasiz **Message**. Ular faqat bir nechta xususiyatlarga ega:

- **message**: Xabarning haqiqiy matni.
- **level**: Xabar turini tavsiflovchi butun son (yuqoridagi xabarlar darajalari bo‘limiga qarang).
- **tags**: Bo‘shliqlar bilan ajratilgan barcha xabar teglarini (**extra_tags**va) birlashtirgan satr.**level_tag**
- **extra_tags**: Bo‘shliqlar bilan ajratilgan ushbu xabar uchun maxsus teglarni o‘z ichiga olgan qator. U sukut bo‘yicha bo‘sh.
- **level_tag**: Darajaning satr tasviri. Odatiy bo‘lib, u bog‘langan konstanta nomining kichik harfli versiyasidir, lekin agar kerak bo‘lsa, bu sozlamadan foydalanib o‘zgartirilishi mumkin **MESSAGE_TAGS**.

Shaxsiy xabar darajalarini yaratish. Xabarlar darajalari butun sonlardan boshqa narsa emas, shuning uchun siz o‘zingizning darajangiz konstantalarini belgilashingiz va ulardan foydalanuvchilarning ko‘proq moslashtirilgan fikr-mulohazalarini yaratish uchun foydalanishingiz mumkin, masalan:

```
CRITICAL = 50
def my_view(request):
    messages.add_message(request, CRITICAL, "A serious error occurred.")
```

Maxsus xabar darajalarini yaratishda siz mavjud darajalarni ortiqcha yuklamaslik uchun ehtiyot bo‘lishingiz kerak. O‘rnatilgan darajalar uchun qiymatlar:

Doimiy daraja	Qiymat
DEBUG	10
INFO	20
SUCCESS	25
WARNING	30
ERROR	40

Agar siz HTML yoki CSS-da maxsus darajalarni aniqlashingiz kerak bo‘lsa, sozlama orqali xaritani taqdim etishingiz kerak **MESSAGE_TAGS**.

Eslatma: Agar siz qayta foydalanish mumkin bo‘lgan dastur yaratayotgan bo‘lsangiz, faqat o‘rnatilgan xabar darajalaridan foydalanish tavsiya etiladi va har qanday maxsus darajalarga tayanmaslik tavsiya etiladi.

Har bir so‘rov uchun qayd etilgan minimal darajani o‘zgartirish

Minimal qayd etilgan daraja har bir so‘rov bo‘yicha **set_level** usul orqali o‘rnatilishi mumkin:

```
from django.contrib import messages
# Change the messages level to ensure the debug message is added.
messages.set_level(request, messages.DEBUG)
messages.debug(request, "Test message...")
# In another request, record only messages with a level of WARNING and higher
messages.set_level(request, messages.WARNING)
messages.success(request, "Your profile was updated.")  #ignored
messages.warning(request, "Your account is about to expire.")
#recorded
# Set the messages level back to default.
messages.set_level(request, None)
```

Xuddi shunday, joriy samarali darajani quyidagi bilan olish mumkin **get_level**:

```
from django.contrib import messages
current_level = messages.get_level(request)
```

Minimal qayd etilgan daraja qanday ishlashi haqida qo‘shimcha ma’lumot olish uchun yuqoridagi Xabar darajalariga qarang.

Qo‘shimcha xabar teglarini qo‘shish. Xabar teglarini to‘g‘ridan-to‘g‘ri boshqarish uchun siz qo‘shimcha usullardan biriga qo‘shimcha teglarni o‘z ichiga olgan qatorni taqdim etishingiz mumkin:

```
messages.add_message(request, messages.INFO, "Over 9000!",
extra_tags="dragonball")
messages.error(request, "Email box full", extra_tags="email")
```

Qo‘shimcha teglar ushbu darajadagi standart tegdan oldin qo‘shiladi va bo‘sh joy ajratiladi.

Xabarlar tizimi o‘chirilganda jimgina bajarilmaydi. Agar siz qayta foydalanish mumkin bo‘lgan ilova (yoki boshqa kod bo‘lagi) yozayotgan bo‘lsangiz va xabar almashish funksiyasini qo‘shmoqchi bo‘lsangiz, lekin foydalanuvchilaringiz xohlamasa, uni yoqishni talab qilmoqchi bo‘lmasangiz, qo‘shimcha kalit so‘z argumentini yuborishingiz mumkin **fail_silently=True**. har qanday **add_message**usullar oilasi. Masalan:

```
messages.add_message(
    request,
    messages.SUCCESS,
    "Profile details updated.",
    fail_silently=True,
)
messages.info(request, "Hello world.", fail_silently=True)
```

Eslatma: Sozlama faqat xabarlar ramkasi o‘chirilganda va usullar oilasidan birini ishlatishga harakat qilganda yuzaga keladigan narsalarni **fail_silently=True**yashiradi . Boshqa sabablarga ko‘ra yuzaga kelishi mumkin bo‘lgan muvaffaqiyatsizliklarni yashirmaydi.

Sinfga asoslangan ko‘rinishlarga xabarlar qo‘shish.

class views.SuccessMessageMixin

FormView Asoslangan sinflarga muvaffaqiyat xabari atributini qo‘shadi

get_success_message(***tozalangan_ma’lumotlar*** **)**

cleaned_data

String formatlash uchun ishlatiladigan shakldan tozalangan ma’lumotlar

Misol views.py :

```
from django.contrib.messages.views import SuccessMessageMixin
from django.views.generic.edit import CreateView
from myapp.models import Author
```

```
class AuthorCreateView(SuccessMessageMixin, CreateView):
    model = Author
    success_url = "/success/"
    success_message = "%(name)s was created successfully"
```

get_success_message() dan tozalangan ma’lumotlar sintaksisi **form**yordamida string interpolyatsiyasi uchun mavjud **%(field_name)s**. ModelForms uchun saqlangan maydonlarga kirish kerak bo‘lsa, usulni **object**bekor qiling

ModelForms uchun view.py misoli :

```
from django.contrib.messages.views import SuccessMessageMixin
from django.views.generic.edit import CreateView
from myapp.models import ComplicatedModel
class ComplicatedCreateView(SuccessMessageMixin, CreateView):
    model = ComplicatedModel
    success_url = "/success/"
    success_message = "%(calculated_field)s was created successfully"
    def get_success_message(self, cleaned_data):
        return self.success_message % dict(
            cleaned_data,
            calculated_field=self.object.calculated_field,
        )
```

Xabarlarning amal qilish muddati. Xabarlar saqlash nusxasi takrorlanganda (va javob qayta ishlanganida o‘chiriladi) tozalanishi uchun belgilanadi.

Xabarlarni o‘chirib tashlamaslik uchun siz **False** takrorlashdan keyin xabarlar xotirasini o‘rnatishingiz mumkin:

```
storage = messages.get_messages(request)
for message in storage:
    do_something_with(message)
storage.used = False
```

Parallel so‘rovlarning harakati. Cookie-fayllar (va shuning uchun seanslar) ishlash usuli tufayli, bir mijoz parallel ravishda xabarlarni o‘rnatuvchi yoki qabul qiluvchi bir nechta so‘rovlarni yuborsa, cookie yoki seanslardan foydalanadigan har qanday backendlarning xatti-harakati aniqlanmaydi . Misol uchun, agar mijoz bitta oynada (yoki yorliqda) xabar yaratadigan so‘rovni boshlasa, so‘ngra boshqa oynada birlashtirilgan xabarlarni oladigan boshqa oynada, birinchi oyna qayta yo‘naltirilgunga qadar, xabar birinchi

oynada emas, ikkinchi oynada paydo bo‘lishi mumkin. kutilishi mumkin bo‘lgan oyna.

Muxtasar qilib aytganda, bitta mijozdan bir vaqtning o‘zida bir nechta so‘rovlar ishtirok etganda, xabarlar ularni yaratgan bir oynaga yetkazilishi yoki ba'zi hollarda umuman bo‘lishi kafolatlanmaydi. E'tibor bering, bu odatda ko‘pgina ilovalarda muammo emas va har bir oyna/yorliq o‘z ko‘rish kontekstiga ega bo‘lgan HTML5da muammo bo‘lmaydi.

Sozlamalar.

Bir nechta sozlamalar sizga xabarning harakatini boshqarish imkonini beradi:

- MESSAGE_LEVEL
- MESSAGE_STORAGE
- MESSAGE_TAGS

Cookie-fayllardan foydalanadigan backendlar uchun cookie-fayl sozlamalari seans cookie sozlamalaridan olinadi:

- SESSION_COOKIE_DOMAIN
- SESSION_COOKIE_SECURE
- SESSION_COOKIE_HTTPONLY

Miksinlar ko‘p tillarda uchraydigan dizayn namunasidir. Pythonda, garchi til lokal miksinlarni qo‘llab-quvvatlamasa ham, ular Pythonning ko‘p meros modeli yordamida amalga oshiriladi.

Aralashmalar merosxo‘rlikdan farq qiladi. Meroslangan sinfda qamrovni toraytirmoqchi bo‘lsangiz, meros foydali bo‘ladi. Miksinlar bir nechta ortogonal xususiyatlarni birlashtirish kerak bo‘lganda bo‘shliqni to‘ldiradi. Qo‘llanish doirasi toraymaydi, shuning uchun an'anaviy meros foydali emas.

Mixins paradigmasi:

- mixinlar odatda obyektdan meros bo‘ladigan sinflardir (agar siz Django yadro ishlab chiqaruvchisi bo‘lmasangiz)
- miksinlar tor doirada, chunki ular bitta mas'uliyatga ega. Ular bitta narsani qilishadi va buni juda yaxshi qilishadi.
- miksinlar plagin funksiyasini ta'minlaydi
- miksinlar meros orqali ishlayotgan bo‘lsa-da, ular subtyping munosabatini yaratmaydi. Bu shuni anglatadiki, miksinlar SOLIDdan Liskoff tamoyiliga amal qilmaydi. Liskoff printsipi: Asosiy sinf misollari olingan sinflar misollari bilan almashtirilishi

mumkin va bu mantiqiy bo‘ladi. Ushbu bayonot miksinlar uchun amal qilmaydi. Shuning uchun biz miksinlar subtiplanish munosabatini yaratmaydi deymiz.

- miksinlar uzaytirilishi uchun mo‘ljallanmagan : class SubMixin(BaseMixin):

Pythonning ko‘p merosi hal qilish tartibini aniqlashni dahshatli tushga aylantiradi va har qanday nozik muammolar yuzaga kelishi mumkin. Agar usulni aniqlash tartibida sikllar mavjud bo‘lsa, Python kompilyatsiya xatosini beradi. Agar siz tasodifan yozgan bo‘lsangiz, class ComposedView(AccessMixin, PermissionRequiredMixin, BaseView)MRO sikllari va shuning uchun kompilyatsiya xatosi sabab bo‘lishi mumkin, chunki buyurtma muhim.

- miksinlarni yaratish uchun mo‘ljallanmagan

Miksinlar modulli dizaynlarni yaratishda juda yaxshi. Python bir nechta merosni qo‘llab-quvvatlaganligi sababli, bu juda yaxshi:
class ComposedView(Mixin1, BaseView)
Mixin1 BaseView-ni kengaytirganda, keyin ComposedView tomonidan kengaytirilgan holda buni o‘qing.

Buyurtma berish masalalari . BaseView-ni meros daraxtining ildizida saqlang va boshqa aralashmalarning har biri funksiyaning BIR maxsus qismini kengaytiradi yoki qo‘shadi. Aniqrog‘i, Mixin2, Mixin1 va BaseView qo‘shgan barcha narsalarni qo‘shadi (ehtiyot bo‘lmasa, uning ustiga yozish ham mumkin). Ehtiyot bo‘ling, miksinlarni zanjirlashda turli xil aralashmalar bir xil asosiy holatni tahrirlash bilan tugamaydi, aks holda ular bir-birining ma’lumotlarini qayta yozadi. Bu mikserlardan foydalanganda eng muhim bo‘lgan g‘amxo‘rlikdir. Bir miksin boshqa miksin o‘zgartirishlarini ustini yazmasi uchun har bir miksin nima qo‘shtirishini bilib olishingiz kerak.

Misol:

```
import PermissionDenied from django.core.exceptions
class LoginRequiredMixin ( object ):
def dispatch ( self , request , *args , **kwargs ) :
if not required.user . is_authenticated() :
upgrade Permission denied
```

return super (LoginRequiredMixin , self) . send (request, *args, **kwargs)

Bu yerda bizda foydalanuvchining tizimga kirganligini tekshirish uchun miksin mavjud. Bu miksin aynan shu imzo bilan jo‘natish usuliga ega bo‘lgan bazaviy ko‘rinishni kengaytiradi degan taxmin bilan yozilgan. Biz jo‘natish usulini kengaytirishni tanlaymiz, chunki bu so‘rovga javob berishning hayotiy siklida birinchi usullardan biri hisoblanadi.

Oxirida super usulga e’tibor bering. Agar foydalanuvchi chinakam tizimga kirgan bo‘lsa, ushbu super usul kengaytirilgan asosiy ko‘rinishni jo‘natishni boshlaydi.

Yana bir amaliy misol. Keling, TemplateViewkimning yagona maqsadi berilgan html shablonini ko‘rsatish ekanligini olaylik. Miksinni qo‘shish orqali ushbu TemplateViewga faqat tizimga kirgan foydalanuvchi kirishi mumkin.

django.views.generic **import TemplateView from**
Class AboutView (LoginRequiredMixin, TemplateView) :
template_name = "about.html"

AboutView ikkala LoginRequiredMixin va TemplateView-dan meros bo‘lib o‘tadi.

Endi agar siz ushbu kirish talabini TemplateView o‘rniga DetailView-ga qo‘shmoqchi bo‘lsangiz, class AboutView(LoginRequiredMixin, TemplateView):bo‘ladiclass AboutView(LoginRequiredMixin, DetailView):

Kirish funksiyasini amalga oshirish uchun DetailView-ga endi kodni o‘zgartirish kerak bo‘lmaydi.

Django sinfiga asoslangan ko‘rinishlar uchun miksinlarni yozish. Miksinlar sinfga asoslangan ko‘rinishlarga tabiiy mos keladi, xuddi dekoratorlar funksiyaga asoslangan ko‘rinishlarga tabiiy ravishda mos keladi. Sinfga asoslangan ko‘rinish uchun mikslarni yozish uchun ko‘pincha ko‘rinishning hayot aylanishi haqida ozgina bilish yaxshidir. Keling, hayot aylanishining ba'zi usullarini tartibda ko‘rib chiqaylik:

1. Agar foydalanuvchi tizimga kirgan bo‘lsa, ruxsati bor yoki yo‘qligini tekshirish kerak bo‘lsa mixins **jo‘natishni bekor qiladi.** Bu chaqirilgan birinchi usullardan biri va view qaysi so‘rov usuli (olish, post va h.k.) ishlatilishini aniqlaydi.

2. aralashmalar kontekstga yangi ma’lumotlarni qo‘shish uchun **get_context_data() ni bekor qiladi.** Ushbu usul kalit so‘z argumentlarini shablon kontekstiga o‘tkazadi. Shunday qilib, bu ko‘rinishlarga qo‘shimcha kontekst qo‘shishning bir usuli.
3. mixlar **get_template_names()** ni bekor qiladi . Bu tomonidan chaqirilgan usul render_to_responseva bu usul barcha shablon nomlarini sanab o‘tadi.

Eslatma: render_to_responsekabi ko‘plab yorliqlar mavjud bo‘lgan asl versiya render.

Shunday qilib , miksindagi **get_template_names() ni** bekor qilish , agar Django loyihangizning o‘ziga xos usuli bo‘lsa, shablon nomlarini aniqlashda ko‘proq moslashuvchanlikni beradi.

Djangobracestez-tez ishlatiladigan django miksinlari to‘plami bo‘lib, ular uch toifaga bo‘lingan.

1. Mikslarga kirish
2. Aralashmalarni shakllantirish
3. Boshqa aralashmalar

Dekoratorlar. Funksiyaga asoslangan ko‘rinishlarda juda keng tarqalgan **login_required(), user_passes_test(), permission_required()**

Sinfga asoslangan ko‘rinishlarda siz ularning ekvivalent aralashmalarini olasiz.

LoginRequiredMixin, UserPassesTestMixin, Permission RequiredMixin

Djangoda miksindan foydalanishga qarshi andozalar. Mos kelmaydigan aralashmalardan ehtiyot bo‘ling, juda ko‘p mikslarni zanjirband qiling, mikslarni haddan tashqari oshiring, aks holda ijro oqimini aniqlash dahshatli tush bo‘ladi. 100 qatorli ko‘rish funksiyalari va funksiyalar bo‘yicha tonnalab dekorativlarga qarshi maslahat berganimizdek, biz ham stsenariylarga qarshi maslahat beramiz, masalan, agar biror narsa ishlashi uchun 6 ta aralash kerak bo‘lsa, siz noto‘g‘ri ish qilyapsiz va ko‘rish qatlamingizga haddan tashqari ko‘p mantiqni joylashtiryapsiz. Taxminan 3 ta aralashtirish OK. Zanjirlash kabi PermissionRequiredMixin va LoginRequiredMixin juda keng tarqalgan, chunki ular ikkalasi ham jo‘natish bilan ishlaydi, lekin bir-biriga xalaqit bermaydi. Biri avtorizatsiya bilan, ikkinchisi autentifikatsiya bilan shug‘ullanadi. Ehtimol, get_context_data()

da ishlash uchun boshqa miksin qo‘shishingiz mumkin. Qanchalik ko‘p aralash qo‘shsangiz, ko‘rish qatlamiga shunchalik ko‘p mantiq qo‘shiladi va mantiq va ijro jarayonini tushunish qiyinlashadi. 4 miksin mutlaq itaruvchi hisoblanadi.

Nazorat savollari:

1. Djangoda mixins nima?
2. Mixin'larni qanday tuzish mumkin?
3. Mixin'larni nima uchun va qanday holatlar uchun ishlatish mumkin?
4. Djangoda messages kutubxonasini qanday ishlatish mumkin?
5. Messages kutubxonasida qaysi xabar turlari mavjud?
6. Django loyihasida foydalanuvchiga xabar berish qanday qo‘llaniladi?
7. Messages kutubxonasida qanday xabarlar o‘tkaziladi va qo‘llaniladi?
8. Mixinlarni andozalash (override) qilish va ma’lumotlarni o‘zgartirish qanday amalga oshiriladi?

15-§. DJANGODA GENERIC KUTUBXONASI.

Jarayon indeks sahifasini yaratishga o‘xshaydi, biz buni avvalgi darslikda ko‘rsatdik. Biz hali ham URL xaritalari, ko‘rinishlari va shablonlarini yaratishimiz kerak bo‘ladi. Asosiy farq shundaki, batafsil sahifalar uchun biz URL manzilidagi naqshlardan ma’lumot olish va uni ko‘rinishga o‘tkazish kabi qo‘shimcha qiyinchiliklarga duch kelamiz. Ushbu sahifalar uchun biz butunlay boshqacha ko‘rinishni namoyish qilamiz: umumiy sinfga asoslangan ro‘yxat va batafsil ko‘rinishlar. Ular kerakli ko‘rish kodini sezilarli darajada kamaytirishi mumkin, bu ularni yozish va saqlashni osonlashtiradi.

Darslikning yakuniy qismida umumiy sinfga asoslangan ro‘yxat ko‘rinishlaridan foydalanganda ma’lumotlaringizni qanday qilib sahifalash mumkinligi ko‘rsatiladi.

Kitob ro‘yxati sahifasi. Kitoblar ro‘yxati sahifasida URL manzilidan foydalanilgan sahifadagi barcha mavjud kitob yozuvlari ro‘yxati ko‘rsatiladi: catalog/books/. Sahifada har bir yozuv uchun sarlavha va muallif ko‘rsatiladi, sarlavha tegishli kitob tafsilotlari sahifasiga giperhavola bo‘ladi. Sahifa saytdagi boshqa barcha sahifalar bilan bir xil tuzilishga va navigatsiyaga ega bo‘ladi va shuning uchun biz avvalgi darslikda yaratgan asosiy shablonni **(base_generic.html) kengaytira olamiz.**

URL xaritalash

/catalog/urls.py ni oching va quyida 'books/'ko‘rsatilganidek, yo‘lni o‘rnatgan qatorda nusxa ko‘chiring. Xuddi indeks sahifasida bo‘lgani kabi, bu funksiya URL manziliga (**'kitoblar/'**path()) mos keladigan naqshni , URL mos keladigan bo‘lsa chaqiriladigan ko‘rish funksiyasini ( ) va ushbu xaritalash uchun nomni belgilaydi. views.BookListView.as_view()

Buferga nusxalash

```
urlpatterns = [
    path('', views.index, name='index'),
    path('books/', views.BookListView.as_view(), name='books'),
]
```

Oldingi darslikda muhokama qilinganidek, URL allaqachon mos kelishi kerak /catalog, shuning uchun ko‘rinish aslida URL uchun chaqiriladi: /catalog/books/.

Ko‘rish funksiyasi avvalgidan boshqacha formatga ega - buning sababi, bu ko‘rinish aslida sinf sifatida amalga oshiriladi. Biz o‘zimizni

noldan yozishdan ko‘ra, biz ushbu ko‘rinish funksiyasini bajarishni xohlagan narsaning ko‘pini bajaradigan mavjud umumiy ko‘rinish funksiyasidan meros bo‘lamiz.

Django sinfiga asoslangan ko‘rinishlar uchun biz sinf usulini chaqirish orqali tegishli ko‘rinish funksiyasiga kiramiz as_view(). Bu sinfning namunasini yaratish va kiruvchi HTTP so‘rovlari uchun to‘g‘ri ishlov berish usullari chaqirilganligiga ishonch hosil qilish bo‘yicha barcha ishlarni bajaradi.

Ko‘rish (sinf asosida). Biz kitoblar ro‘yxati ko‘rinishini oddiy funksiya sifatida osongina yozishimiz mumkin (xuddi oldingi indeks ko‘rinishimiz kabi), u barcha kitoblar uchun ma’lumotlar bazasini so‘raydi va keyin ro‘yxatni render()belgilangan shablonga o‘tkazish uchun qo‘ng‘iroq qiladi. Buning o‘rniga biz sinfga asoslangan umumiy ro‘yxat ko‘rinishidan ( ) foydalanamiz ListView- mavjud ko‘rinishdan meros bo‘lgan sinf. Umumiy ko‘rinish bizga kerak bo‘lgan ko‘pgina funksiyalarni allaqachon amalga oshirganligi va Django eng yaxshi amaliyotiga amal qilganligi sababli, biz kamroq kod, kamroq takrorlash va oxir-oqibat kamroq texnik xizmat ko‘rsatish bilan yanada mustahkamroq ro‘yxat ko‘rinishini yaratishimiz mumkin.

catalog/views.py ni oching va quyidagi kodni faylning pastki qismiga nusxalash:

```
from django.views import generic
class BookListView(generic.ListView):
    model = Book
```

Yuqoridagi standart xatti-harakatni o‘zgartirish uchun atributlarni qo‘shishingiz mumkin. book_listMisol uchun, agar bir xil modeldan foydalanadigan bir nechta ko‘rinishga ega bo‘lishingiz kerak bo‘lsa, boshqa shablon faylini belgilashingiz mumkin yoki shablonni ishlatish uchun intuitiv bo‘lmasa, boshqa shablon o‘zgaruvchisi nomidan foydalanishni xohlashingiz mumkin. Ehtimol, eng foydali variant - qaytarilgan natijalar to‘plamini o‘zgartirish/filtrlash - shuning uchun barcha kitoblarni ro‘yxatga olish o‘rniga, siz boshqa foydalanuvchilar tomonidan o‘qilgan eng yaxshi 5 ta kitobni ro‘yxatlashingiz mumkin.

```
class BookListView(generic.ListView):
    model = Book
    context_object_name = 'book_list'   # your own name for the list as a template variable
```

```
    queryset = Book.objects.filter(title__icontains='war')[:5] # Get 5
books containing the title war
    template_name = 'books/my_arbitrary_template_name_list.html'  #
Specify your own template name/location
```

Sinfga asoslangan ko‘rinishlardagi usullarni bekor qilish. Bu yerda biz buni qilishimiz shart bo‘lmasa-da, siz ba'zi sinf usullarini ham bekor qilishingiz mumkin. get_queryset() Misol uchun, qaytarilgan yozuvlar ro‘yxatini o‘zgartirish usulini bekor qilishimiz mumkin . Bu querysetavvalgi kod fragmentida qilganimiz kabi atributni o‘rnatishdan ko‘ra ko‘proq moslashuvchan (garchi bu holatda haqiqiy foyda yo‘q):

```
class BookListView(generic.ListView):
    model = Book
    def get_queryset(self):
        return Book.objects.filter(title__icontains='war')[:5] # Get 5
books containing the title war
```

get_context_data()Shablonga qo‘shimcha kontekst o‘zgaruvchilarni o‘tkazish uchun ham bekor qilishimiz mumkin (masalan, kitoblar ro‘yxati sukut bo‘yicha uzatiladi). some_dataQuyidagi fragment kontekstga " " nomli o‘zgaruvchini qanday qo‘shishni ko‘rsatadi (keyin u shablon o‘zgaruvchisi sifatida mavjud bo‘ladi).

```
class BookListView(generic.ListView):
    model = Book
    def get_context_data(self, **kwargs):
        # Call the base implementation first to get the context
        context = super(BookListView, self).get_context_data(**kwargs)
        # Create any data and add it to the context
        context['some_data'] = 'This is just some data'
        return context
```

Buni amalga oshirayotganda, yuqorida qo‘llanilgan namunaga amal qilish muhimdir:

- Avval bizning supersinfimizdan mavjud kontekstni oling.
- Keyin yangi kontekst ma’lumotlarini qo‘shing.
- Keyin yangi (yangilangan) kontekstni qaytaring.

Eslatma: Nima qilish mumkinligi haqida ko‘plab misollar uchun o‘rnatilgan sinfga asoslangan umumiy ko‘rinishlarni (Django docs) tekshiring .

Ro‘yxat ko‘rinishi shablonini yaratish. HTML faylini yarating va quyidagi matndan nusxa oling. Yuqorida muhokama qilinganidek, bu umumiy sinfga asoslangan ro‘yxat ko‘rinishida kutilgan standart shablon faylidir (Bookismli ilovada nomlangan model uchun catalog).

Umumiy ko‘rinishlar uchun shablonlar xuddi boshqa shablonlarga o‘xshaydi (garchi, albatta, shablonga uzatilgan kontekst/ma’lumotlar har xil bo‘lishi mumkin). *Indeks* shablonida bo‘lgani kabi , biz asosiy shablonni birinchi qatorda kengaytiramiz va keyin nomli blokni almashtiramiz content.

```
{% extends "base_generic.html" %}
{% block content %}
  <h1>Book List</h1>
  {% if book_list %}
    <ul>
      {% for book in book_list %}
      <li>
        <a href="{{ book.get_absolute_url }}">{{ book.title }}</a>
        ({{book.author}})
      </li>
      {% endfor %}
    </ul>
  {% else %}
    <p>There are no books in the library.</p>
  {% endif %}
{% endblock %}
```

Ko‘rinish kontekstni (kitoblar ro‘yxatini) sukut bo‘yicha as object_listva book_listtaxalluslar bilan uzatadi; yoki ishlaydi.

Shartli bajarish. Belgilanganligini va bo‘sh emasligini tekshirish uchun if, else, va andoza teglaridan foydalanamiz . Agar bo‘sh bo‘lsa, bandda ro‘yxatga olinadigan kitoblar yo‘qligini tushuntiruvchi matn ko‘rsatiladi. Agar bo‘sh bo‘lmasa, biz kitoblar ro‘yxatini takrorlaymiz. endifbook_listbook_listelsebook_list

```
{% if book_list %}
  <!-- code here to list the books -->
{% else %}
  <p>There are no books in the library.</p>
{% endif %}
```

Yuqoridagi shart faqat bitta holatni tekshiradi, lekin siz qo‘shimcha shartlarni elifshablon yorlig‘i (masalan {% elif var2 %},) yordamida sinab ko‘rishingiz mumkin. Shartli operatorlar haqida ko‘proq ma’lumot olish uchun qarang: if , ifequal/ifnotequal va ifchanged o‘rnatilgan shablon teglari va filtrlarida (Django Docs).

Looplar uchun. Shablon quyida ko‘rsatilganidek, kitoblar ro‘yxatini aylanib chiqish uchun for va and template teglaridan foydalanadi. Har bir iteratsiya shablon o‘zgaruvchisini joriy ro‘yxat elementi uchun ma’lumot bilan endfor to‘ldiradi .book

```
{% for book in book_list %}
  <li><!-- code here get information from each book item --></li>
{% endfor %}
```

{% empty %} Agar kitoblar ro‘yxati bo‘sh bo‘lsa, nima bo‘lishini aniqlash uchun shablon tegidan ham foydalanishingiz mumkin (garchi bizning shablonimiz shartli shartdan foydalanishni tanlagan bo‘lsa ham):

```
<ul>
  {% for book in book_list %}
    <li><!-- code here get information from each book item --></li>
  {% empty %}
    <p>There are no books in the library.</p>
  {% endfor %}
</ul>
```

Bu yerda ishlatilmasa ham, Django siklida siz iteratsiyani kuzatish uchun foydalanishingiz mumkin bo‘lgan boshqa o‘zgaruvchilarni ham yaratadi. Misol uchun, siz forloop.lastsikl oxirgi marta ishga tushirilganda shartli ishlov berish uchun o‘zgaruvchini sinab ko‘rishingiz mumkin.

O‘zgaruvchilarga kirish. Loop ichidagi kod har bir kitob uchun ro‘yxat elementini yaratadi, unda ham sarlavha (hali yaratilmagan tafsilotlar ko‘rinishiga havola sifatida) va muallif ko‘rsatiladi.

```
<a href="{{ book.get_absolute_url }}">{{ book.title }}</a>
({{book.author}})
```

Biz "nuqta belgisi" (masalan, va) yordamida bog‘langan kitob yozuvining *maydonlariga* kiramiz , bu yerda elementdan keyingi

matn maydon nomidir (modelda aniqlanganidek). book.titlebook.authorbook

Shuningdek, biz shablonimiz ichidan modeldagi *funksiyalarni* chaqirishimiz mumkin - bu holda biz Book.get_ absolute_url()bog‘langan tafsilot yozuvini ko‘rsatish uchun foydalanishingiz mumkin bo‘lgan URL manzilini olish uchun qo‘ng‘iroq qilamiz. Bu funksiyada argumentlar bo‘lmasa ishlaydi (argumentlarni uzatishning hech qanday usuli yo‘q!)

Eslatma: Shablonlarda funksiyalarni chaqirishda “nojo‘ya ta’sirlardan” biroz ehtiyot bo‘lishimiz kerak. Bu yerda biz shunchaki ko‘rsatish uchun URL manzilini olamiz, lekin funksiya deyarli hamma narsani qila oladi - biz shablonimizni ko‘rsatish orqali ma’lumotlar bazasini (masalan) o‘chirishni xohlamaymiz!

Asosiy shablonni yangilang. Asosiy shablonni oching va quyida ko‘rsatilganidek**,** barcha kitoblar uchun URL havolasiga {% url 'books' %} kiriting. Bu barcha sahifalarda havolani faollashtiradi (biz buni "kitoblar" URL mapperini yaratganimizdan keyin muvaffaqiyatli o‘rnatishimiz mumkin).

```
<li><a href="{% url 'index' %}">Home</a></li>
<li><a href="{% url 'books' %}">All books</a></li>
<li><a href="">All authors</a></li>
```

Siz hali kitoblar ro‘yxatini tuza olmaysiz, chunki bizda hali ham bog‘liqlik yo‘q — kitob tafsilotlari sahifalari uchun URL xaritasi, alohida kitoblarga giperhavolalar yaratish uchun zarur. Keyingi bo‘limdan keyin biz ro‘yxat va batafsil ko‘rinishlarni ko‘rsatamiz.

Kitob tafsilotlari sahifasi. Kitob tafsilotlari sahifasida ma'lum bir kitob haqidagi ma’lumotlar ko‘rsatiladi, unga URL manzili orqali kirish mumkin catalog/book/<id>(bu yerda <id>kitobning asosiy kaliti). Modeldagi maydonlarga qo‘shimcha ravishda Book(muallif, xulosa, ISBN, til va janr) mavjud nusxalar ( ) tafsilotlarini ham sanab o‘tamiz, BookInstancesshu jumladan holat, kutilayotgan qaytish sanasi, nashr va identifikator. Bu bizning o‘quvchilarimizga nafaqat kitob haqida ma’lumot olish, balki uning mavjud yoki qachon mavjudligini tasdiqlash imkonini beradi.

URL xaritalash. /catalog/urls.py ni oching va quyida ko‘rsatilgan " **book-detail** " nomli yo‘lni qo‘shing . Bu path()funksiya naqsh, bog‘langan umumiy sinfga asoslangan batafsil ko‘rinish va nomni belgilaydi.

```
urlpatterns = [
    path('', views.index, name='index'),
    path('books/', views.BookListView.as_view(), name='books'),
    path('book/<int:pk>', views.BookDetailView.as_view(),
name='book-detail'),
]
```

Kitob tafsilotlari yo‘li uchun URL namunasi biz ko‘rmoqchi bo‘lgan kitobning maxsus identifikatorini olish uchun maxsus sintaksisdan foydalanadi. Sintaksis juda oddiy: burchakli qavslar URLning suratga olinadigan qismini belgilaydi, ko‘rinish olingan ma’lumotlarga kirish uchun foydalanishi mumkin bo‘lgan o‘zgaruvchining nomini o‘z ichiga oladi. Misol uchun, **<something>** , belgilangan naqshni oladi va qiymatni "bir narsa" o‘zgaruvchisi sifatida ko‘rinishga o‘tkazadi. Ma’lumotlar turini (int, str, slug, uuid, path) belgilaydigan konvertor spetsifikatsiyasi bilan o‘zgaruvchi nomidan oldin ixtiyoriy ravishda qo‘yishingiz mumkin .

Bunday holda biz '<int:pk>'kitob identifikatorini olish uchun foydalanamiz, u maxsus formatlangan satr bo‘lishi kerak va uni parametr sifatida ko‘rinishga o‘tkazamiz pk(asosiy kalit uchun qisqacha). Bu Kitob modelida ta'riflanganidek, kitobni ma’lumotlar bazasida yagona saqlash uchun foydalaniladigan identifikator.

Eslatma: Yuqorida aytib o‘tilganidek, bizning mos URL manzilimiz aslida catalog/book/<digits>(chunki biz **katalog** ilovasidamiz, /catalog/deb taxmin qilinadi).

Ogohlantirish: Umumiy sinfga asoslangan batafsil ko‘rinish **pk** nomli parametrdan o‘tishni *kutadi* . Agar siz o‘zingizning funksiya ko‘rinishini yozayotgan bo‘lsangiz, o‘zingiz yoqtirgan parametr nomidan foydalanishingiz mumkin yoki haqiqatan ham ma’lumotni nomsiz argumentga yuborishingiz mumkin.

Taqdim etilgan naqsh mosligi oddiy va *har qanday* satr yoki butun sonni path()olishni istagan (juda keng tarqalgan) holatlar uchun foydalidir . Agar sizga aniqroq filtrlash kerak bo‘lsa (masalan, faqat ma'lum miqdordagi belgilarga ega bo‘lgan satrlarni filtrlash uchun), re_path() usulidan foydalanishingiz mumkin .

Bu usul xuddi Regular iborapath() yordamida naqsh belgilash imkonini beruvchidan tashqari qo‘llaniladi . Misol uchun, oldingi yo‘l quyida ko‘rsatilgandek yozilishi mumkin edi:

re_path(r'^book/(?P<pk>\d+)$', views.BookDetailView.as_view(), name='book-detail'),

Muntazam iboralar naqshni xaritalashning ajoyib vositasidir. Ochig‘ini aytganda, ular juda noaniq va yangi boshlanuvchilar uchun qo‘rqitishi mumkin. Quyida juda qisqa primer mavjud!

Bilish kerak bo‘lgan birinchi narsa shundaki, muntazam iboralar odatda raw string literal sintaksisi yordamida e'lon qilinishi kerak (ya'ni ular ko‘rsatilgandek qo‘shib qo‘yilgan:

Shakl mosligini e’lon qilish uchun sintaksisning asosiy qismlarini bilishingiz kerak:

Belgi	**Ma'nosi**
^	Matn boshini moslang
$	Matn oxirini moslang
\d	Raqamni moslang (0, 1, 2, … 9)
\w	So‘z belgisini, masalan, alifbodagi katta yoki kichik harf, raqam yoki pastki chiziq belgisini (_) moslang.
+	Oldingi belgilarning bir yoki bir nechtasini moslang. Masalan, bir yoki bir nechta raqamga mos kelish uchun siz dan foydalanasiz \d+. Bir yoki bir nechta "a" belgilarini moslashtirish uchun siz foydalanishingiz mumkina+
*	Oldingi belgining nol yoki undan ko‘pini moslang. Masalan, siz foydalanishingiz mumkin bo‘lgan hech narsa yoki so‘z bilan mos kelish uchun\w*
( )	Qavslar ichidagi naqsh qismini oling. Har qanday olingan qiymatlar nomsiz parametrlar sifatida ko‘rinishga o‘tkaziladi (agar bir nechta naqshlar olingan bo‘lsa, tegishli parametrlar qo‘lga kiritishlar e'lon qilingan tartibda taqdim etiladi).
(?P< *ism* >...)	Namunani (... bilan ko‘rsatilgan) nomlangan o‘zgaruvchi sifatida (bu holda "ism") oling. Olingan qiymatlar ko‘rsatilgan nom bilan ko‘rinishga o‘tkaziladi. Shuning uchun sizning ko‘rinishingiz bir xil nomdagi parametrni e'lon qilishi kerak!
[ ]	To‘plamdagi bitta belgiga mos keling. Masalan, [abc] "a" yoki "b" yoki "c" da mos keladi. [-\w] '-' belgisiga yoki istalgan so‘z belgisiga mos keladi.

Ko‘pgina boshqa belgilar tom ma'noda olinishi mumkin!

Keling, naqshlarning bir nechta haqiqiy misollarini ko‘rib chiqaylik:

Naqsh	**Tavsif**
r'^book/(?P<pk>\d+)$'	Bu bizning URL mapperimizda ishlatiladigan RE. U book/satr boshida bo‘lgan (**^book/**), keyin bir yoki bir nechta raqamga (\d+) ega bo‘lgan va keyin

	tugaydigan (satr oxiridan oldin raqamli bo‘lmagan belgilarsiz) satrga mos keladi. Shuningdek, u barcha raqamlarni **(?P<pk>\d+)** ushlaydi va ularni 'pk' nomli parametrdagi ko‘rinishga o‘tkazadi. **Olingan qiymatlar har doim satr sifatida uzatiladi!** Masalan, bu ga mos keladi book/1234va o‘zgaruvchini pk='1234'ko‘rinishga yuboradi.
r'^book/(\d+)$'	Bu avvalgi holat bilan bir xil URL manzillariga mos keladi. Olingan ma’lumotlar ko‘rinishga nomsiz argument sifatida yuboriladi.
r'^book/(?P<stub>[-\w]+)$'	book/Bu satr boshida joylashgan satrga mos keladi *(* **^book/**), so‘ngra '-' yoki so‘z belgisi (**[-\w]+**) bo‘lgan bir yoki bir nechta belgilarga ega va keyin tugaydi. Shuningdek, u ushbu belgilar to‘plamini ushlaydi va ularni "stub" deb nomlangan parametrdagi ko‘rinishga o‘tkazadi. Bu "stub" uchun juda odatiy naqsh. Stublar ma’lumotlar uchun URL-do‘st so‘zga asoslangan asosiy kalitlardir. Agar kitobingizning URL manzili ko‘proq ma’lumotli bo‘lishini istasangiz, stubdan foydalanishingiz mumkin. Masalan, /catalog/book/the-secret-gardeno‘rniga /catalog/book/33.

Bitta o‘yinda bir nechta naqshlarni suratga olishingiz va shuning uchun URL manzilida juda ko‘p turli xil ma’lumotlarni kodlashingiz mumkin.

Eslatma: Qiyinchilik sifatida, ma'lum bir yil, oy, kunda chiqarilgan barcha kitoblar va unga mos keladigan RE ro‘yxati uchun URL manzilini qanday kodlash mumkinligini ko‘rib chiqing.

URL xaritalaringizda qo‘shimcha parametrlarni o‘tkazish. Biz bu yerda foydalanmagan, lekin siz uchun qimmatli bo‘lgan xususiyatlardan biri shundaki, siz ko‘rinishga qo‘shimcha variantlarni o‘z ichiga olgan lug‘atnipath() o‘tkazishingiz mumkin (funksiya uchun uchinchi nomlanmagan argumentdan foydalanib). Agar siz bir nechta manbalar uchun bir xil ko‘rinishdan foydalanmoqchi bo‘lsangiz va har bir holatda uning xatti-harakatlarini sozlash uchun ma’lumotlarni uzatmoqchi bo‘lsangiz, bu yondashuv foydali bo‘lishi mumkin.

Misol uchun, quyida ko‘rsatilgan yo‘lni hisobga olgan holda, /myurl/halibut/Django so‘rovi uchun qo‘ng‘iroq qiladi views.my_view(request, fish='halibut', my_template_name='some_path').
path('myurl/<fish>', views.my_view, {'my_template_name': 'some_path'}, name='aurl'),

Eslatma: Har ikkala nomli olingan naqshlar va lug‘at variantlari *nomli* argumentlar sifatida ko‘rinishga o‘tkaziladi . Agar siz suratga olish naqsh va lug‘at kaliti uchun **bir xil nomdan** foydalansangiz , lug‘at opsiyasi ishlatiladi.

Ko‘rish (sinf asosida). catalog/views.py ni oching va quyidagi kodni faylning pastki qismiga nusxalash:

```
class BookDetailView(generic.DetailView):
    model = Book
```

Agar yozuv mavjud bo‘lmasa nima bo‘ladi? Agar so‘ralgan yozuv mavjud bo‘lmasa, umumiy sinfga asoslangan batafsil ko‘rinish avtomatik ravishda siz uchun istisno yaratadi Http404- ishlab chiqarishda bu avtomatik ravishda tegishli “resurs topilmadi” sahifasini ko‘rsatadi, agar xohlasangiz, uni sozlashingiz mumkin.

Bu qanday ishlashi haqida bir oz tasavvurga ega bo‘lish uchun, quyida keltirilgan kod qismi, agar siz umumiy sinfga asoslangan batafsil ko‘rinishdan foydalanmasangiz, **sinfga** asoslangan ko‘rinishni funksiya sifatida qanday amalga oshirishingiz mumkinligini ko‘rsatadi .

```
def book_detail_view(request, primary_key):
    try:
        book = Book.objects.get(pk=primary_key)
    except Book.DoesNotExist:
        raise Http404('Book does not exist')
    return render(request, 'catalog/book_detail.html', context={'book': book})
```

Ko‘rinish birinchi navbatda modeldan maxsus kitob yozuvini olishga harakat qiladi. Agar bu bajarilmasa, ko‘rinish Http404kitob "topilmadi" degan istisnoni ko‘rsatishi kerak. Yakuniy qadam, odatdagidek, parametrdagi (lug‘at sifatida) render()shablon nomi va kitob ma’lumotlari bilan qo‘ng‘iroq qilishdir.context

Agar siz umumiy ko‘rinishdan foydalanmasangiz, buni qilishning yana bir usuli bu funksiyani chaqirishdir get_object_or_404(). Http404Bu yozuv topilmasa, istisno qilish uchun yorliqdir .

```
from django.shortcuts import get_object_or_404
def book_detail_view(request, primary_key):
    book = get_object_or_404(Book, pk=primary_key)
    return render(request, 'catalog/book_detail.html', context={'book': book})
```

Tafsilotlarni ko‘rish shablonini yaratish. HTML faylini yarating va unga quyidagi tarkibni bering. Yuqorida muhokama qilinganidek, bu umumiy sinfga asoslangan tafsilot ko‘rinishida kutilgan standart shablon fayl nomidir (Bookismli ilovada nomlangan model uchun catalog).

```
{% extends "base_generic.html" %}
{% block content %}
  <h1>Title: {{ book.title }}</h1>
  <p><strong>Author:</strong> <a href="">{{ book.author }}</a></p>
  <!-- author detail link not yet defined -->
  <p><strong>Summary:</strong> {{ book.summary }}</p>
  <p><strong>ISBN:</strong> {{ book.isbn }}</p>
  <p><strong>Language:</strong> {{ book.language }}</p>
  <p><strong>Genre:</strong> {{ book.genre.all|join:", " }}</p>
  <div style="margin-left:20px;margin-top:20px">
    <h4>Copies</h4>
    {% for copy in book.bookinstance_set.all %}
      <hr />
      <p
        class="{% if copy.status == 'a' %}text-success{% elif copy.status == 'm' %}text-danger{% else %}text-warning{% endif %}">
        {{ copy.get_status_display }}
      </p>
      {% if copy.status != 'a' %}
        <p><strong>Due to be returned:</strong> {{ copy.due_back }}</p>
      {% endif %}
      <p><strong>Imprint:</strong> {{ copy.imprint }}</p>
      <p class="text-muted"><strong>Id:</strong> {{ copy.id }}</p>
```

```
    {% endfor %}
  </div>
{% endblock %}
```

Eslatma: Yuqoridagi shablondagi muallif havolasi bo‘sh URL manziliga ega, chunki biz hali havola qilish uchun muallif tafsilotlari sahifasini yaratmaganmiz. Tafsilotlar sahifasi mavjud bo‘lgandan so‘ng, biz uning URL manzilini ushbu ikkita yondashuvdan biri bilan olishimiz mumkin:

- urlShablon tegidan “muallif-tafsiloti” URL manzilini (URL xaritalagichida belgilangan) teskari o‘zgartirish uchun foydalaning va unga kitob uchun muallif misolini o‘tkazing:

```
<a href="{% url 'author-detail' book.author.pk %}">{{ book.author }}</a>
```

- Muallif modeli get_absolute_url()usulini chaqiring (bu bir xil teskari operatsiyani bajaradi):

```
<a href="{{ book.author.get_absolute_url }}">{{ book.author }}</a>
```

Ikkala usul ham bir xil ishni samarali bajarsa-da, get_absolute_url()afzalroqdir, chunki u yanada izchil va barqaror kod yozishga yordam beradi (har qanday o‘zgarishlar faqat bitta joyda amalga oshirilishi kerak: muallif modeli).

Biroz kattaroq bo‘lsa-da, bu shablondagi deyarli hamma narsa avvalroq tasvirlangan:

- Biz asosiy shablonimizni kengaytiramiz va "kontent" blokini bekor qilamiz.
- Muayyan kontentni ko‘rsatish yoki ko‘rsatmaslikni aniqlash uchun biz shartli ishlov berishdan foydalanamiz.
- forObyektlar ro‘yxati bo‘ylab aylanish uchun biz sikllardan foydalanamiz .
- Biz kontekst maydonlariga nuqta belgisi yordamida kiramiz (chunki biz batafsil umumiy ko‘rinishdan foydalanganmiz, kontekst nomi bilan atalgan book; biz “ object” dan ham foydalanishimiz mumkin)

Biz ilgari ko‘rmagan birinchi qiziqarli narsa bu funksiyadir book.bookinstance_set.all(). Bu usul Django tomonidan ma’lum BookInstancebir Book.

```
{% for copy in book.bookinstance_set.all %}
  <!-- code to iterate across each copy/instance of a book -->
{% endfor %}
```

Bu usul kerak, chunki siz ForeignKey(birdan ko‘pgacha) maydonni faqat munosabatlarning "ko‘p" tomonida (BookInstance) e'lon qilasiz. Boshqa ("bitta") modeldagi munosabatlarni e'lon qilish uchun hech narsa qilmasangiz, u () Bookbog‘langan yozuvlar to‘plamini olish uchun hech qanday maydonga ega emas. Ushbu muammoni bartaraf etish uchun Django siz foydalanishingiz mumkin bo‘lgan tegishli nomdagi "teskari qidirish" funksiyasini yaratadi. Funksiya nomi e'lon qilingan model nomini kichik harf bilan yozish orqali tuziladi ForeignKey, keyin esa (ya'ni, yaratilgan _setfunksiya).Bookbookinstance_set()

Eslatma: Bu yerda biz barcha yozuvlarni olish uchun foydalanamiz all()(standart). filter()Koddagi yozuvlar to‘plamini olish uchun usuldan foydalanishingiz mumkin bo‘lsa-da, siz buni to‘g‘ridan-to‘g‘ri shablonlarda qila olmaysiz, chunki siz funksiyalarga argumentlarni ko‘rsata olmaysiz.

Agar siz buyurtmani (sinfga asoslangan ko‘rinishingiz yoki modelingiz bo‘yicha) belgilamasangiz, quyidagi kabi ishlab chiqish serveridagi xatolarni ham ko‘rasiz:

```
[29/May/2017 18:37:53] "GET /catalog/books/?page=1 HTTP/1.1" 200 1637
/foo/local_library/venv/lib/python3.5/site-packages/django/views/generic/list.py:99: UnorderedObjectListWarning: Pagination may yield inconsistent results with an unordered object_list: <QuerySet [<Author: Ortiz, David>, <Author: H. McRaven, William>, <Author: Leigh, Melinda>]>
  allow_empty_first_page=allow_empty_first_page, **kwargs)
```

Buning sababi, paginator obyekti sizning asosiy ma’lumotlar bazasida ba'zi ORDER BY bajarilishini ko‘rishni kutadi. Busiz, qaytarilayotgan yozuvlar aslida to‘g‘ri tartibda ekanligiga ishonch hosil qilib bo‘lmaydi!

Siz parametrdan foydalana olmaysiz sort_by() va o‘tkaza olmaganingiz uchun (yuqorida tavsiflangani kabi filter()) uchta variantdan birini tanlashingiz kerak bo‘ladi:

1. Modelingizga orderingichki deklaratsiya qo‘shing .class Meta
2. querysetMaxsus sinfga asoslangan ko‘rinishga atribut qo‘shing order_by(), .

3. get_querysetMaxsus sinfga asoslangan ko‘rinishga usul qo‘shish va shuningdek order_by(), .

class MetaAgar siz a for modeli bilan ishlashga qaror qilsangiz Author(ehtimol, sinfga asoslangan ko‘rinishni sozlash kabi moslashuvchan emas, lekin etarlicha oson), siz shunday bir narsaga erishasiz:

```
class Author(models.Model):
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    date_of_birth = models.DateField(null=True, blank=True)
    date_of_death = models.DateField('Died', null=True, blank=True)
    def get_absolute_url(self):
        return reverse('author-detail', args=[str(self.id)])
    def __str__(self):
        return f'{self.last_name}, {self.first_name}'
    class Meta:
        ordering = ['last_name']
```

Albatta, maydon bo‘lishi shart emas last_name: u boshqa har qanday bo‘lishi mumkin. Va nihoyat, unumdorlik bilan bog‘liq muammolarga yo‘l qo‘ymaslik uchun ma’lumotlar bazasida indeksi (noyob yoki yo‘q) bo‘lgan atribut/ustun bo‘yicha saralashingiz kerak. Albatta, bu yerda kerak bo‘lmaydi (biz, ehtimol, juda kam kitoblar va foydalanuvchilar bilan o‘zimizdan oldindamiz), lekin kelajakdagi loyihalar uchun buni yodda tutish kerak. Shablondagi ikkinchi qiziqarli (va aniq bo‘lmagan) narsa - bu yerda biz har bir kitob misoli uchun holat matnini ko‘rsatamiz ("mavjud", "xizmat" va boshqalar). Aqlli o‘quvchilar shuni ta'kidlashadiki, BookInstance.get_status_display() biz status matnini olish uchun ishlatadigan usul kodning boshqa joyida ko‘rinmaydi.

```
<p class="{% if copy.status == 'a' %}text-success{% elif copy.status
== 'm' %}text-danger{% else %}text-warning{% endif %}">
 {{ copy.get_status_display }} </p>
```

Bu funksiya avtomatik ravishda yaratiladi, chunki tanlov BookInstance.status maydoni. Django avtomatik ravishda modeldagi har bir tanlov maydoni uchun " " usul yaratadi , bu maydonning joriy qiymatini olish uchun ishlatilishi mumkin.get_FOO_display()Foo

Shu nuqtada, biz kitoblar ro‘yxatini va kitob tafsilotlari sahifalarini ko‘rsatish uchun kerak bo‘lgan hamma narsani yaratishimiz kerak edi. Serverni ishga tushiring (python3 manage.py runserver) va brauzeringizni oching http://127.0.0.1:8000/.

Ogohlantirish: Muallif yoki muallif tafsilotlari havolalarini hali bosmang – ularni tanlovda yaratasiz!

Kitoblar ro‘yxatini ko‘rsatish uchun “**Barcha kitoblar”** havolasini bosing .

Keyin kitoblaringizdan biriga havolani bosing. Agar hamma narsa to‘g‘ri sozlangan bo‘lsa, siz quyidagi skrinshotga o‘xshash narsani ko‘rishingiz kerak.

Sahifalar. Agar sizda bir nechta yozuvlar bo‘lsa, bizning kitoblar ro‘yxati sahifamiz yaxshi ko‘rinadi. Biroq, o‘nlab yoki yuzlab yozuvlarga kirganingizda, sahifani yuklash asta-sekin ko‘proq vaqt

talab etadi (va oqilona ko‘rib chiqish uchun juda ko‘p tarkibga ega). Ushbu muammoni hal qilish - har bir sahifada ko‘rsatiladigan elementlar sonini kamaytirish, ko‘rinishlar ro‘yxatiga sahifalash qo‘shish.

Django sahifalash uchun mukammal o‘rnatilgan yordamga ega. Bundan ham yaxshiroq, bu umumiy sinfga asoslangan ro‘yxat ko‘rinishlariga kiritilgan, shuning uchun uni yoqish uchun ko‘p ish qilishingiz shart emas!

Ko‘rishlar. katalog/views.py ni oching va paginate_byquyida ko‘rsatilgan qatorni qo‘shing.

```
class BookListView(generic.ListView):
    model = Book
    paginate_by = 10
```

Ushbu qo‘shimcha bilan siz 10 dan ortiq yozuvga ega bo‘lishingiz bilan ko‘rinish shablonga yuboradigan ma’lumotlarni sahifalashni boshlaydi. Turli sahifalarga GET parametrlari yordamida kirish mumkin — 2-sahifaga kirish uchun URL manzilidan foydalanasiz /catalog/books/?page=2.

Shablonlar. Endi ma’lumotlar sahifalangan bo‘lsa, natijalar to‘plami bo‘ylab harakatlanish uchun shablonga yordam qo‘shishimiz kerak. Biz barcha ro‘yxat ko‘rinishlarini sahifalashtirmoqchi bo‘lishimiz sababli, biz buni asosiy shablonga qo‘shamiz.

{% block content %}{% endblock %}dan keyin darhol quyidagi sahifalash blokiga nusxa ko‘chiring {% endblock %}. Kod avval joriy sahifada sahifalash yoqilganligini tekshiradi. *Agar shunday bo‘lsa, u tegishli ravishda keyingi* va *oldingi* havolalarni (va joriy sahifa raqamini) qo‘shadi .

```
{% block pagination %}
    {% if is_paginated %}
        <div class="pagination">
            <span class="page-links">
                {% if page_obj.has_previous %}
                    <a href="{{ request.path }}?page={{
page_obj.previous_page_number }}">previous</a>
                {% endif %}
                <span class="page-current">
                    Page {{ page_obj.number }} of {{
page_obj.paginator.num_pages }}.
```

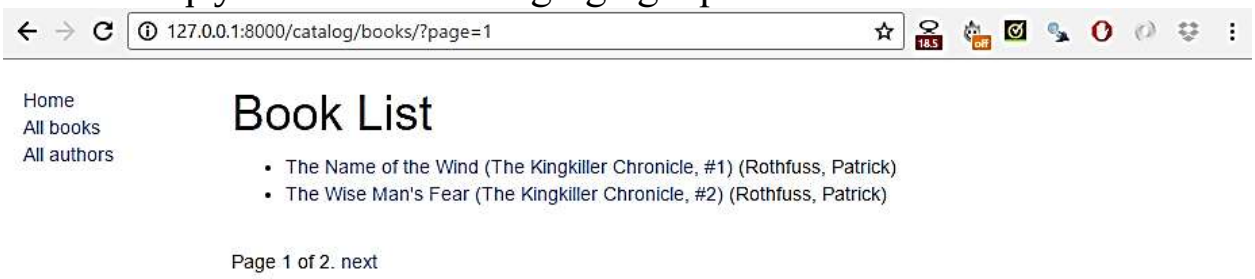
```
            </span>
            {% if page_obj.has_next %}
               <a href="{{ request.path }}?page={{
page_obj.next_page_number }}">next</a>
            {% endif %}
          </span>
        </div>
    {% endif %}
  {% endblock %}
```

Agar joriy sahifada sahifalash qo‘llanilsa, mavjud bo‘ladigan Paginatorpage_obj obyekti . U joriy sahifa, oldingi sahifalar, qancha sahifa borligi va hokazolar haqidagi barcha ma’lumotlarni olish imkonini beradi.

{{ request.path }}Biz sahifalash havolalarini yaratish uchun joriy sahifa URL manzilini olish uchun foydalanamiz . Bu foydali, chunki u biz sahifalashayotgan obyektdan mustaqil.

Quyidagi skrinshotda sahifalash qanday ko‘rinishi ko‘rsatilgan - agar siz ma’lumotlar bazasiga 10 dan ortiq sarlavha kiritmagan bo‘lsangiz, katalog/views.py paginate_by faylidagi qatorda ko‘rsatilgan raqamni kamaytirish orqali uni osonroq sinab ko‘rishingiz mumkin . Quyidagi natijani olish uchun biz uni ga o‘zgartirdik.paginate_by = 2

Sahifalar havolalari pastki qismida ko‘rsatiladi, keyingi/oldingi havolalar qaysi sahifada ekanligingizga qarab ko‘rsatiladi.



Vazifa muallif tafsilotlarini yaratish va loyihani yakunlash uchun zarur bo‘lgan ko‘rinishlar ro‘yxatini yaratishdir. Ular quyidagi URL manzillarida taqdim etilishi kerak:

- catalog/authors/- Barcha mualliflar ro‘yxati.
- catalog/author/<id>— Muayyan muallif uchun asosiy kalit maydoni nomli batafsil ko‘rinish<id>

URL mappers va ko‘rinishlar uchun talab qilinadigan kod Bookbiz yuqorida yaratgan ro‘yxat va batafsil ko‘rinishlar bilan

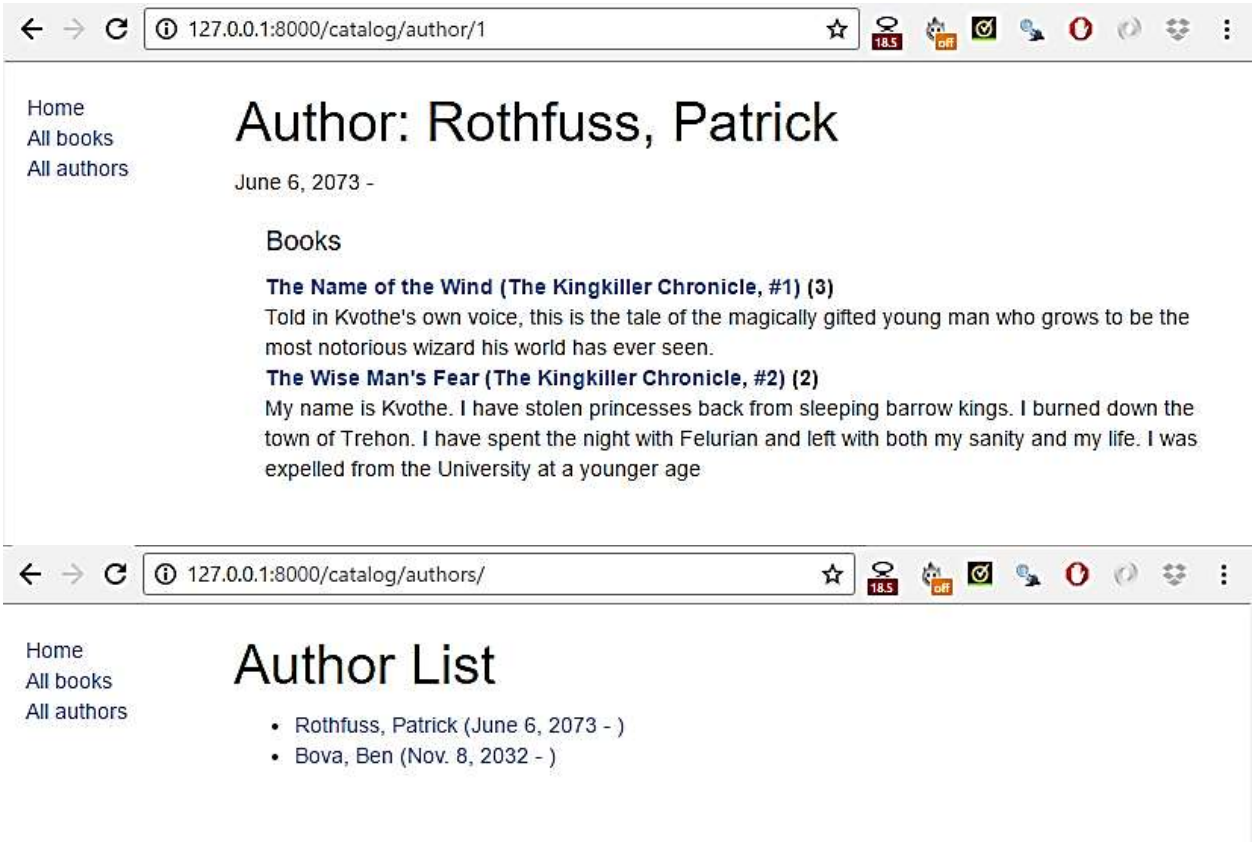
deyarli bir xil bo‘lishi kerak. Shablonlar boshqacha bo‘ladi, lekin o‘xshash xatti-harakatlarga ega.

Eslatma:

- Mualliflar ro‘yxati sahifasi uchun URL mapper yaratganingizdan so‘ng, asosiy shablondagi “ Barcha mualliflar” havolasini ham yangilashingiz kerak bo‘ladi. “Barcha kitoblar” havolasini yangilaganimizdek, xuddi shu jarayonni bajaring.
- Muallif tafsilotlari sahifasi uchun URL mapper yaratganingizdan so‘ng, kitob tafsilotlarini ko‘rish shablonini yangilashingiz kerak, shunda muallif havolasi sizning yangi muallif tafsilotlari sahifangizga ishora qiladi (bo‘sh URL bo‘lishdan ko‘ra). Buni amalga oshirishning tavsiya etilgan usuli get_absolute_url()quyida ko‘rsatilganidek, muallif modelini chaqirishdir.

```
<p>
<strong>Author:</strong>
<a href="{{ book.author.get_absolute_url }}">{{ book.author }}</a>
</p>
```

Tugatganingizdan so‘ng, sahifalaringiz quyidagi skrinshotlarga o‘xshash bo‘lishi kerak.



Nazorat savollari

1. Djangoda Generic kutubxonasi (Generic Views) nimaga xizmat qiladi?
2. Generic kutubxonasida qaysi kichik funksiyalar mavjud?
3. Generic Viewlar qanday tuziladi?
4. Generic kutubxonasida ma’lumotlarni o‘qish uchun qanday viewlar mavjud?
5. Generic Viewlar orqali modellarni qanday qo‘llab-quvvatlash mumkin?
6. Generic Viewlar qanday bo‘lib ma’lumotlarni filtirlash (filtering) uchun qo‘llaniladi?
7. Generic Viewlarda ma’lumotlarni yangilash (updating) uchun qanday funksiyalar mavjud?
8. Djangoda Generic Viewlar bilan qanday bog‘liq ma’lumotlar kutubxonalari mavjud?

GLOSSARIY

1. **Algoritm -** ma’lum bir turga oid masalalarni yechishda ishlatiladigan amallarning muayyan tartibda bajarilishi haqidagi aniq qoida (dastur).
2. **Blok-sxema (oqim diagrammasi) -** ish jarayonini aks ettiruvchi diagramma turidir. Sxemani algoritmning diagrammatik tasviri, vazifani bosqichma-bosqich hal qilish usuli sifatida ham aniqlash mumkin.
3. **Dastur:**
   - ❖ biror-bir faoliyat, ishning mazmuni va rejasi;
   - ❖ siyosiy partiyalar, tashkilotlar, alohida arboblar faoliyatining asosiy qoidalari va maqsadlari bayoni;
   - ❖ o‘quv fani mazmunining qisqacha izohi;
4. **Dasturlash -** kompyuterlar va boshqa mikroprotsessorli elektron mashinalar uchun dasturlar tuzish, sinash va o‘zgartirish jarayonidan iborat. Odatda dasturlash yuqori saviyali dasturlash tillari (PHP,Java,C++,Python) vositasida amalga oshiriladi. Bu dasturlash tillarining semantikasi odam tiliga yaqinligi tufayli dastur tuzish jarayoni ancha oson kechadi.
5. **Funksiya** - (lotincha: functio — „bajarish“, „amalga oshirish“) - muayyan til, til birligi, lisoniy shaklning u yoki bu vazifani bajarish qobiliyati; tilning kishilik jamiyatidagi roli, vazifasi; til tizimining barcha sathlarida uning birliklari o‘rtasidagi bog‘liqlik yoki munosabatlar.
6. **Massiv** - bir xil ma'lumot turiga ega bo'lib bir nechta o'zgaruvchini har birini alohida e'lon qilish o'rniga bir o'zgaruvchiga bir nechta qiymat saqlash uchun ishlatiladi.
7. **Obyektga yo`naltirilgan dasturlash**(Object Oriented Programming) - bu biror bir maqsadga yo'naltirilgan dasturlash degan ma'noni anglatadi.(OOP)
8. **Protseduraviy dasturlash** - bu ma'lumotlarga ishlov beradigan protseduralar yoki funksiyalarni yozish, obyektga yo'naltirilgan dasturlash esa ma'lumot va funksiyalarni o'z ichiga olgan obyektlarni yaratish haqida.
9. **Parametr** - (yunoncha: parametron — o‘lchaydigan) matematik formula va ifodalarda qo‘llaniladigan kattalik.

10. Rekursiya - funksiya o‘ziga o‘zi to‘g‘ridan-to‘g‘ri yoki qandaydir vosita orqali murojaat qilish jarayoni.

11. Assembler dasturlash tili - (yoki Assembly tili, yoki belgili mashina kodi), binar(ya'ni ikkilik) tilga eng yaqin til bo'lib, unda tildagi ko‘rsatmalar va arxitektura ko‘rsatmalari o‘rtasida juda kuchli muvofiqlik mavjud. Assembler dasturlash tilida odatda bitta mashina ko'rsatmasi bo'ladi, lekin konstantalar, izohlar, assembler yo'naltirmalari, belgili teglar, xotira joylari, registerlar va makroslarni ham qo'llab-quvvatlaydi.

12. Fortran - bu umumiy maqsadli kompilyatsiya qilingan imperativ dasturlash tili bo'lib, u ayniqsa sonli hisoblash va ilmiy hisoblash uchun juda mos keladi.

13. Paskal (inglizcha: Pascal) —Ushbu dasturlash tili boshqa bir qator tillar uchun asos bo‘lib xizmat qiladi. Paskal imperativ va Protsedurali dasturlash tili hisoblanadi.

14. C (talaffuzi: si) — kompilyatsiyalanuvchi statik dasturlash tili bo‘lib, 1969—1973-yillarda Bell laboratoriyasi xodimi Dennis Ritchie tomonidan yaratilgan. Ushbu dasturlash tili B tilining takomillashgan ko‘rinishi sifatida yaratilgan.

15. Kompilyator – bu o‘zgartirish degan ma’noni beradi. Ya’ni dasturlash tilida yozilgan dastur(C++ bo‘lsa, *.c, *.cpp)ni kompyuter tushunadigan tilga o‘zgartirib, uni ishlashini ta’minlaydi. Bu degani dastur kompyuterda to‘liq ishlaydi. Bundan ko‘rinib turibdiki, C++ da dastur tuzish uchun kompilyator o‘rnatish zarur.

16. Kompilyatsiya – o‘zgaruvchi jarayon, ya’ni yuqori pog‘onali dastur kodlari(misol uchun C++ da tuzilgan kod)ni quyi pog‘onali ishlovchi kodga aylantirish jarayoni.

17. Abstraksiya – bu identifikatorlardan farqli bo‘lgan istalgan dasturlash tili ifodasi

18. Izoh - bu dasturchilar o'rtasida qayta tushuntirish uchun hojat qolmasligi uchun dastur qismiga sharh sifatida qoldiriladigan izohlar hisoblanadi.

19. O'zgaruvchi - xotiraning nomlangan qismi bo’lib, o’zida ma’lum bir toifadagi qiymatlarni saqlaydi. O`zgaruvchining nomi va qiymatlari bo’ladi. O`zgaruvchining nomi orqali qiymat saqlanayotgan xotira qismiga murojaat qilinadi. Dastur ishlashi jarayonida o`zgaruvchining qiymatini o`zgartirish mumkin. Har qanday o`zgaruvchini ishlatishdan oldin, uni e’lon qilish lozim.

20. Xotira manzili - bu kompyuterda saqlanadigan joy nomi hisoblanadi. Xotira manzilini olish uchun & dan foydalanib olishimiz mumkin. Bundan tashqari o'zgaruvchi xotira manzilini olish uchun ham foydalanish mumkin.

21. Enum – yangi ma’lumotlar turlarini yaratish va qiymatlari o’zgarmas bo`lgan o’zgaruvchilar to’plami.

22. O`zgarmas - bu konstanta hisoblanadi. C++ dasturlash tilida o'zgarmaslarni e'lon qilish uchun const kalit so'zidan foydalanib ushbu o'zgaruvchini o'zgarmas deb e'lon qilinadi.

23. Ifoda - ma’lumotlar bo‘yicha amallarni bajarish tartibini belgilaydi, ular operatorlardan (o‘zgarmaslar, o‘zgaruvchilar, funksiyalar), qavslar va amal belgilaridan iborat.

24. Operator -o'zgaruvchilar va qiymatlar bo'yicha amallarni bajarish uchun ishlatiladi.

25. Encapsulation(Enkapsulatsiya)- ning ma'nosi , "sezgir" ma'lumotlar foydalanuvchilardan yashirilganligiga ishonch hosil qilishdir.

26. Massiv indekslari - massiv elementlarini bir biridan farqlash va ularga murojaat qilishda ishlatiladi. Ular 0 dan boshlanadi: [0] birinchi element. [1] ikkinchi element va boshqalar.

27. Meros - bu odatda bitta sinf dan qayta-qayta foydalanish imkoniyatini beradi. Yangi sinf yaratishda sinf attributlaridan foydalanish imkoniyati.

28. Polimorfizm - ko'p shakllar ma`nosini anglatadi. Umuman olganda, polimorfizm sinflar ierarxiyasi mavjud bo'lganda paydo bo'ladi va ular meros bilan bog'liq. C ++ polimorfizmi shuni anglatadiki, a'zo funksiyasini chaqirish ushbu funksiyani chaqiradigan obyekt turiga qarab boshqa funksiyani bajarishga olib keladi.

**29. Global o`zgaruvchilar -** ham asosiy dasturda, ham funksiyada ishlatish mumkin bo'lgan o`zgaruvchilar global o'zgaruvchilar deyiladi. Global o'zgaruvchilar asosiy dasturda e`lon qilishi kerak.

**30. Lokal o`zgaruvchilar -** faqat funksiyada ishlatish mumkin bo'lgan o'zgaruvchilarga lokal o'zgaruvchilar deyiladi. Ular funksiya ichida e`lon qilinadi.

31. Spetsifikator - sinf a'zolariga (atributlari va usullari) qanday kirish mumkinligini belgilaydi.

32. **Modellashtirish tili** - har qanday sun'iy til ifodalash uchun ishlatilishi mumkin ma`lumot yoki bilim tuzilishi, bu izchil qoidalar to'plami bilan belgilanadi. Qoidalar tarkibidagi tarkibiy qismlarning ma'nosini izohlash uchun ishlatiladi.

33. **Konstruktor -** sinf bilan bir xil nomga ega, u doimo public bo'ladi va u hech qanday qiymat qaytarmaydi.

34. **Konstruktor parametrlari** - konstruktorlar parametrlarni (odatdagi funksiyalar kabi) ham olishi mumkin, bu esa atributlar uchun boshlang'ich qiymatlarni o`zlashtirishda foydali bo`lishi mumkin.

35. **Istisnolar –** dastur yozish davomida keltirib chiqaradigan xatoliklarni bartaraf etish ishlatiladi va uch guruhga bo'linadi.

36. **try -** agar kiritilgan blok kodi to'g'ri bo'lgan holda boshlanishi kerak bo'ladigan kalit so'zi.

37. **throw** - maxsus xatolikni kutish uchun ishlatish mumkin

38. **catch** - agar yuqoridagi kalit so'zlar tarkibidagi blok kod xatolik yuzaga kelsa ushbu kalit so'zdan foydalaniladi**.**

39. **Interpretator** - dasturlash tilida yozilgan kodni bosqichma-bosqich mashina kodiga aylantirib, tahlil qiladi va berilgan buyruqlarni ketma-ketlikda bajaradi. Agar xatolik sodir bólsa, ósha zahoti xabar beradi.

40. **Preprotsessorlar** - ko'rsatmalar bo'lib, ular kompilyatorga haqiqiy kompilyatsiya boshlanishidan oldin ma'lumotni qayta ishlash bo'yicha ko'rsatmalar beradi.

41. **Return -** funksiyani tugatadi va boshqaruvni chaqirilayotgan funksiyaga qaytaradi (yoki boshqaruv asosiy funksiyadan uzatilsa operatsion tizimga qaytaradi).

42. **RAM (random access memory -tasodifiy kirish xotirasi)** - bu kompyuterning qisqa muddatli xotirasi bo'lib, u yerda protsessor hozirda foydalanayotgan ma'lumotlar saqlanadi. Sizning kompyuteringiz RAM xotirasiga qattiq disk, SSD yoki boshqa uzoq muddatli saqlash qurilmasidagi ma'lumotlarga qaraganda tezroq kirishi mumkin, shuning uchun RAM hajmi tizim ishlashi uchun juda muhimdir.

43. **Ichki sinf (nested class)** - boshqa sinf ichida joylashgan sinf. Odatda, ichki sinflar faqat tashqi sinf obyekti ichida mavjud bo`lishi mumkin bo'lgan obyektlarni tavsiflash uchun ishlatiladi.

FOYDALANILGAN ADABIYOTLAR RO‘YXATI

1. Mirziyoyev Sh.M. Yangi O‘zbekiston taraqqiyot strategiyasi. Toshkent-2022, 245-bet.
2. Васильев А.Н. Python на примерах. Практический курс по программированию. Наука и техника, Санкт-Петербург, 2016, 235-243стр.
3. Azamatov A.R., Boltayev B. Algoritmlash va dasturlash asoslari // O‘quv qo‘llanma – Toshkent : “Cho‘lpon”, 2013. – 232 b.
4. Aripov M.M., Otaxanov N.A. Dasturlash asoslari // O‘quv qo‘llanma. Toshkent: “Tafakkur bo‘stoni”, 2015. – 240 b.
5. Gabor Szabo. 1000 Python Examples. - 2020, PP.140-165.
6. Mengliyev Sh.A., Abdug‘aniyev O.A., Shonazarov S.Q., To‘rayev D. Sh. Python dasturlash tili. Termiz-2021.
7. Thomas H. Cormen. Intruduction to algorithms. Third Edition. Massachusetts Institute of Technology. The MIT Press. London 2009. P.1292.
8. Fabio Nelli. Python Data Analytics. Rome, Library of Congress. 2018. 576 p. https://doi.org/10.1007/978-1-4842-3913-1
9. Vasilyev A.N. Python na primerax. – Sankt Peterburg: Nauka i texnika, 2018, -430 s.
10. Algorithms, Fourth Edition (Deluxe): Book and 24-Part Lecture Series 1st Edition , Addison-Wesley Professional , USA, 2015.
11. Chris Roffey. Computer science. Programming book for Python. – USA:Cambridge university press. 2017, – p .204.
12. Chris Roffey. Python basics. Coding club. Level 1,2. – USA:Cambridge university press. 2012, – p.85.
13. Ro‘ziyev R.A. Dasturlash asoslari // O‘quv qo‘llanma. ¬ Toshkent, 2020. – 159 b.
14. Nazirov Sh. A., Qobulov R.V. Obyektga mo‘ljallangan dasturlash // Kasb-hunar kollejlari uchun o‘quv qo‘llanma. – T.: G‘afur G‘ulom nomidagi nashriyotmatbaa ijodiy uyi, 2007. – 184 b.
15. Mirsanov U.M., Toxirov F.J., Norbekov A.O., Djurayeva D.R. Dasturlash. // O‘quv qo‘llanma. Toshkent, 2021. – 152 b.
16. https://metanit.com/python
17. https://pythonworld.ru/tipy-dannyx-v-python/slovari-dict-funkcii-i-metody-slovarej.html
18. https://ai.mohirdev.uz/
19. https://www.w3resource.com/
20. https://www.sololearn.com/
21. https://leetcode.com/
22. https://www.hackerrank.com/
23. https://www.tiobe.com/tiobe-index/
24. acmp.ru
25. https://docs.python.org/

U.T.Subxonqulov, D.N.Xamroyeva, U.N.Xamroyev

PYTHON

DASTURLASH TILI

DARSLIK

Muharrir: **E.Eshov**
Tex. muharrir: **D.Abduraxmonova**
Musahhih: **M.Shodiyeva**
Badiiy rahbar: **M.Sattorov**

Nashriyot litsenziyasi № 022853. 04.03.2022.
Original maketdan bosishga ruxsat etildi: 10.11.2024. Bichimi 60x84. Kegli 16 shponli. “Times New Roman” garnitura 1/16.
Ofset bosma usulida. Ofset bosma qog‘ozi.
Bosma tabog‘i 11. Adadi 100. Buyurtma № 347

KAMOLOT

“BUXORO DETERMINANTI” MCHJ
bosmaxonasida chop etildi.
Buxoro shahar Namozgoh ko‘chasi 24 uy
Tel:. + 998 91 310 27 22