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

U.T. Subxonqulov

PYTHON

DASTURLASH TILI

O‘QUV QO‘LLANMA

“KAMOLOT” nashriyoti
Buxoro–2024

UO‘K: 004.43.378.2
KBK: 32.973.26
S91

Subxonqulov Umidjon To‘xtamurod o‘g‘li, Python dasturlash tili / [Matn]: o‘quv qo‘llanma / U.T.Subxonqulov - Buxoro : “BUXORO DETERMINANTI” MCHJning Kamolot nashriyoti, 2024. - 132 b.

ISBN: 978-9910-729-58-4

Taqrizchilar:

G.I.Atayeva	Buxoro davlat universiteti “Axborot tizimlari va raqamli texnologiyalari” kafedrasi
F.R.Muradova	Buxoro muhandislik texnologiya instituti “Axborot kommunikatsiya texnologiyalari kafedrasi p.f.d. (DSc), professori

Oliy ta’lim, fan va innovatsiyalar vazirligi Buxoro davlat universitetining 2024-yil 02-iyuldagi 452- soni buyrug‘iga asosan nashr etishga ruxsat berildi. Ro‘yxatga olish raqami 452-13.

MUNDARIJA

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

Этот учебник охватывает такие знания, как программирование на языке программирования Python, веб-проекты и технологии объектно-ориентированного программирования, работа в среде программирования. ООП в Python, конструктор в классах, концепции наследования, возможности технологии MVT. В Django представлена технология создания моделей, стандартная модель пользователя в Django и ее адаптация к проекту, подготовка форм, создание проекта и регистрация, логин, логотип, тестирование проекта в Django. Кроме того, для моделей, созданных в django, админ-панель содержит подробное объяснение подготовки функций поиска и фильтрации, миксов и сообщений, а также возможностей библиотеки generic. Язык программирования Python описывает возможности разработки, создания и использования фреймворков Django. Также приведенная информация основана на примерах.

ANNOTATION

This textbook covers knowledge such as programming in the Python programming language, web projects, and Object-Oriented programming technologies, working in a programming environment. In Python, the capabilities of OOP, constructor in classes, succession concepts, MVT technology are highlighted. There are models making in Django, Standard user model in Django and re-fitting it to the project, forms making, project building and project testing technology in registration, login, logout, Django. Also detailed are the preparation of search and filter functions in the admin panel for models made in Django, Mixins and messages, and generic library capabilities. The Python programming language describes the possibilities for developing, creating, and using Django frameworks. Also, the information provided is based on examples.

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‘rinda albatta oliy 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 o‘quv qo‘llanma 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 koʻrish shabloni) dizayn namunasiga amal qiladi. Model - Siz taqdim qilmoqchi boʻlgan ma'lumotlar, ma'lumotlar bazasi bilan oʻzaro ishlashda qulaylik beradi. View - foydalanuvchi soʻrovi asosida tegishli shablon va tarkibni qaytaradigan soʻrovlar ustida ishlash uchun foydalaniladi. Template - matn fayli (HTML fayli kabi), veb-sahifaning tartibini oʻz ichiga olgan, ma'lumotlarni qanday koʻrsatish kerakligini ta'minlash uchun foydalaniladi. “Python dasturlash tili” oʻquv qoʻllanmasi kirish, 15 ta mavzu, mundarija hamda glossariy, foydanilgan adabiyotlar va internet resurlari roʻyxatidan iborat. Oʻquv qoʻllanmada dasturlar sharti, kerakli loyihalar, modellar va ularning natijalari koʻrinishida berilgan.

Oʻquv qoʻllanmaning umumiy hajmi 145 betdan iborat boʻlib, koʻplab Django loyihalari va ular ustida amallar bajarish imkoniyatlari koʻrsatib oʻtilgan.

Oʻquv qoʻllanma baʼzi kamchiliklardan xoli boʻlmasligi mumkin. Uni yanada mukammal boʻlishi yuzasidan bildirilgan fikr-mulohazalarni mamnuniyat bilan qabul qilamiz.

Ma’ruza №1

Mavzu: Python OOP tushunchalari.

Reja:

1. Ob’ekt tushunchasi
2. Pythonda sinf va obyektlar
3. Classning methodlari

1. Ob’ekt tushunchasi

Boshqa umumiy maqsadli tillar singari, python dasturlash tili ham ob'ektga yo‘naltirilgan til hisoblanadi. Bu bizga ob'ektga yo‘naltirilgan yondashuv orqali dasturlarni ishlab chiqishimizga imkon beradi. Python-da biz osongina sinflar va obyektlarni yaratishimiz va ulardan foydalanishimiz mumkin.

Ob'ektga yo‘naltirilgan dasturlash tizimining asosiy printsiplari quyida keltirilgan:

Object (Ob'ekt)
Class (Sinf)
Method (metod, usul)
Inheritance (Meros olish)
Polymorphism (Polimorfizm)
Data Abstraction (Ma'lumotlarni olish)
Encapsulation (Inkapsulyatsiya)

Object (Ob'ekt)

Ob'ekt - bu holat va xulq-atvor, xususiyatlarga ega bo‘lgan shaxs. Bu sichqoncha, klaviatura, stul, stol, ruchka va boshqa turdagi har qanday haqiqiy ob'ekt bo‘lishi mumkin.

Python-dagi hamma narsa ob'yekt bo‘lib, deyarli hamma narsada atributlar va metodlar mavjud. Barcha funksiyalar funksiya manba kodida belgilangan **__doc__** qatorini qatorini qaytaradigan o‘rnatilgan doc atributiga ega.

Class (Sinf)

Sinf ob'ektlar 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 u atribut va usulni, ya'ni elektron pochta identifikatori, ism, yosh, ish haqi va boshqalarni o‘z ichiga olishi kerak.

Sintaksis

```
class ClassName:
 <Bayonot-1>
 .
 .
 <Bayonot-N>
```

Method (metod, usul)

Metod - bu ob'ekt bilan bog'liq bo‘lgan funksiya. Python-da metod faqat sinf misollari uchun xos emas. Har qanday ob'ekt turi metodlariga ega bo‘lishi mumkin.

Inheritance (Meros olish)

Merosxo‘rlik - bu haqiqiy dunyo meros tushunchasini simulyatsiya qiladigan ob'ektga yo‘naltirilgan dasturlashning eng muhim jihati. Bola ob'ekti 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 klassi, xossalari olingan sinf esa asosiy sinf yoki ota-parent classi 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. Masalan, sizda sinf hayvonlari bor, va barcha hayvonlar gapirishadi. Ammo ular boshqacha gapirishadi. Bu erda "gapirish" harakati ma'noda polimorf va hayvonga bog'liq. Shunday qilib, mavhum "hayvon" tushunchasi aslida "gapirmaydi", lekin aniq hayvonlar (it va mushuklar kabi) "gapirish" harakatini aniq amalga oshiradilar.

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 bir ichida birlashtiriladi.

Data abstraction (Ma'lumotlarni abstraktsiya qilish)

Ma'lumotlarni ajralish va inkapsulyatsiya qilish ikkalasi ham ko‘pincha sinonim sifatida ishlatiladi. Ikkalasi ham deyarli sinonimdir, chunki ma'lumotlar abstraktsiyasiga inkapsulyatsiya orqali erishiladi.

Abstraktsiya ichki tafsilotlarni yashirish va faqat funksionallikni ko‘rsatish uchun ishlatiladi. Biron bir narsani mavhumlashtirish, bu narsa funktsiyalar yoki butun dastur bajaradigan ishlarning mohiyatini o‘z ichiga olishi uchun narsalarga nom berishni anglatadi.

2. Pythonda sinf va obyektlar

Biz allaqachon muhokama qilganimizdek, sinf virtual ob'ekt bo‘lib, uni ob'ektning rejasi sifatida ko‘rish mumkin. Sinf paydo bo‘lganida obyekt paydo bo‘ldi. Keling, buni bir misol orqali tushunaylik.

Bir talabani qisqacha tahlil qilaylik, talaba deyilayotgani uni inson ekanligidan darak va u insonlarga xos parametrlari bor ya’ni ismi, familyasi, otasining ismi, tug‘ilgan yili, jinsi, bo‘yi, vazni va xokazolar endi biz butkul boshqa talabaga qaraymiz va unda ham xuddi shunday parametrlar borligini faqatgina qiymatlari boshqacharoq (masalan bo‘yi balandroq yoki vazni kamroq va x.k) ekanligini ko‘ramiz. Bu ikki talaba aloxida aloxida obyektlar ammo ularning mavjud ma’lumotlari aynan bir gruhga mos kelishini ko‘ramiz xuddi bir qolibdan chiqqani kabi va biz manashu qolibga sinf deb qaraymiz shu bilan yuqoridagi ikki talaba talabalar sinfiga kiradi.

Boshqa tomondan, ob'ekt sinfning misoli. Ob'ektni yaratish jarayonini instantatsiya deb atash mumkin.

O‘quv qo‘llanmasining ushbu qismida biz python-da sinflar va ob'ektlarni yaratishni muhokama qilamiz. Atributga sinf ob'ekti yordamida qanday erishish mumkinligi haqida ham gaplashamiz.

Sinflar obyekt konstruktorlari hisoblanadi. Ular bilan obyektlar tuziladi.

Sinf hosil qilish

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

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

<class '__main__.Son'>

Obyekt hosil qilish

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

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

5

3. Classning methodlari

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.

```
class Ishchi:
    def __init__(self, ism, yosh):
        self.ism = ism
        self.yosh = yosh
p1 = Ishchi ("Abbosbek", 20)
print(p1.ism)
print(p1.yosh)
```

Abbosbek
20

Obyekt funksiyalari

Obyektlar ham funksiyaga ega bo‘lishi mumkin. Bu funksiyalar sinf ichida tuziladi va obyektlar tomonida ishlatiladi. Masalan, obyekt o‘zini tanishtirish funksiyasini tuzamiz:

```
class Ishchi:
    def __init__(self, ism, yosh):
        self.ism = ism
        self.yosh = yosh
    def tanish(self):
        print("Mening ismim "+ self.ism)
p1 = Ishchi ("Abbosbek", 20)
p1.tanish()
```

```
Mening ismim Abbosbek
```

Self parametri

self parametri sinfga tegishli o‘zgaruvchilarga murojaat qila olish uchun ishlatiladi. U o‘ziga xos yo‘llovchi vositadir. U aynan self deb nomlanishi shart emas, boshqa nomlarni ishlatish ham mumkin. Faqat u sinfdagi istalgan funksiyaning ilk parametri sifatida yozilishi shart.

Hozir yuqoridagi misolimizdagi self parametrlarini abc deb o‘zgartiramiz va natija o‘zgarmaydi.

```
class Ishchi:
    def __init__(abc, ism, yosh):
        abc.ism = ism
        abc.yosh = yosh
    def tanish(abc):
        print("Mening ismim "+ abc.ism)
p1 = Ishchi ("Abbosbek", 20)
p1.tanish()
```

```
Mening ismim Abbosbek
```

Obyekt xususiyatini o‘zgartirish

Biror obyektning xususiyatlarini osongina o‘zgartirishimiz mumkin. Masalan, dastlab tuzgan obyektimiz 22 yosh bo‘lsa, so‘ng uni 25 yoshga o‘zgartiramiz:

```
class Ishchi:
    def __init__(abc, ism, yosh):
        abc.ism = ism
        abc.yosh = yosh
p1 = Ishchi ("Abbosbek", 20)
```

```
p1.yosh = 25
print(p1.yosh)
```
25

Obyekt xususiyatini o‘chirish

Obyekt xususiyatlarini o‘chirish ham mumkin. Hozir obyektimizdagi yosh xususiyatini o‘chiramiz.

So‘ng uni ekranga chiqarish buyrug’ini beramiz. Dastur ishga tushgach xatolik haqida xabar beriladi.

```
class Ishchi:
    def __init__(abc, ism, yosh):
        abc.ism = ism
        abc.yosh = yosh
p1 = Ishchi ("Abbosbek", 20)
del p1.yosh
print(p1.yosh)
```
AttributeError: 'Ishchi' object has no attribute 'yosh'

Obyektni o‘chirish

Obyektni o‘chirish uchun del kalit so‘zini obyekt nomi bilan qo‘llaymiz. Natijada obyekt butkul o‘chib ketadi.

Quyidagi kodimizda ham xatolik haqida xabar beriladi. Sababi, biz o‘chib ketgan obyektni ekranga chiqarmoqchi bo‘lyabmiz:

```
class Ishchi:
    def __init__(abc, ism, yosh):
        abc.ism = ism
        abc.yosh = yosh
p1 = Ishchi ("Abbosbek", 20)
del p1
print(p1)
```
NameError: name 'p1' is not defined

Nazorat savollari

1. Ob’ekt nima?
2. Pythonda sinflar qayday tartibda yaratiladi?
3. Method nima va qanday vazifalarni bajaradi?
4. Classning methodlari haqida ma’lumot bering.
5. Inheritance (Meros olish) haqida ma’lumot bering.
6. Polymorphism (Polimorfizm) haqida ma’lumot bering.
7. Encapsulation (Inkapsulyatsiya) haqida ma’lumot bering.
8. Data abstraction (Ma'lumotlarni abstraktsiya qilish) haqida ma’lumot bering.

Ma’ruza №2

Mavzu: Sinflarda konstruktor tushunchasi.

Reja:

1. Konstruktorning mavjud xolatlari
2. Python ichki sinf vazifalari
3. Dunder methodlar

1. Konstruktorning mavjud xolatlari

Konstruktor - bu sinfning instansiya a'zolarini initsializatsiya qilish uchun ishlatiladigan maxsus metod (funktsiya) turidir soda qilib aytganda sinfga tegishli obyekt atributlarini qabul qiluvchi va obyektning atributlari bilan bog‘liqligini taminlovchi fuksiya.

Konstruktorlar ikki xil bo‘lishi mumkin:

Parametrlangan konstruktor

Parametrlanmagan konstruktor

Sinfga tegishli ob'ektini yaratganimizda konstruktor aftomatik ishga tushadi. Shuningdek, konstruktorlar ob'ekt uchun biron bir ishga tushirish vazifasini bajarish uchun yetarli atributlar mavjudligini tasdiqlaydilar.

Python-da konstruktor yaratish

Pythonda __init__ metodi sinf konstruktorini simulyatsiya qiladi. Ushbu usul sinfga tegishli obyekt yaratiganda avtomatik chaqiriladi. Biz __init__ ta'rifiga qarab, sinf ob'ektini yaratishda istalgan 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.

Fuqaro sinfining atributlarini ishga tushirish uchun quyidagi misolni ko‘rib chiqing.

Fuqaro:

```
class Fuqaro:
        def __init__(self,name,id):
                self.id = id; self.name = name;
        def display (self):
                print("ID: %d \nName: %s"%(self.id,self.name))

emp1 = Fuqaro ("John",101)
emp2 = Fuqaro ("David",102)
emp1.display();
emp2.display();
```

```
ID: 101
Name: John
ID: 102
Name: David
```

Misol: Sinfga tegishli nechta obyekt yaratilganini hisoblash

```
class Student:
 count = 0
        def __init__(self):
                Student.count = Student.count + 1

s1=Student()
s2=Student()
s3=Student()
print("The number of students:",Student.count)
```

```
The number of students: 3
```

Pythonning parametrlanmagan konstruktorga misoli

```
class Student:
        def __init__(self):
                print("This is non parametrized constructor")
        def show(self,name):
                print("Salom",name)
student = Student()
student.show("Abbosbek")
```

```
Salom Abbosbek
```

Pythonning parametrlangan konstruktorga misol

```
class Student:
        def __init__(self, name):
                print("This is parametrized constructor")
                self.name = name
        def show(self):
                print("Hello",self.name)
student = Student("John")
student.show()
```

```
Hello John
```

2. Python ichki sinf vazifalari

Sinfda aniqlangan ichki funktsiyalar quyida tavsiflangan.

1 getattr (obj, name, default) Ob'ektning atributiga kirish uchun ishlatiladi.
2 setattr (obj, name, value) U ob’ektning o‘ziga xos atributiga ma’lum bir qiymatni belgilash uchun ishlatiladi.
3 delattr (obj, name) U ma’lum bir atributni o‘chirish uchun ishlatiladi.
4 hasattr (obj, name) Ob’ektda o‘ziga xos atribut bo‘lsa, u haqiqiy qiymatni
qaytaradi.

Misol:

```
class Student:
        def __init__(self,name,id,age):
                self.name = name;
                self.id = id;
                self.age = age
s = Student("John",101,22)
print(getattr(s,'name'))
setattr(s,"age",23)
print(getattr(s,'age'))
print(hasattr(s,'id'))
delattr(s,'age')
print(s.age)
```

```
John
23
True
AttributeError: 'Student' object has no attribute 'age'
```

O‘rnatilgan sinf atributlari

Boshqa atributlar bilan bir qatorda, python class-ida sinf haqida ma'lumot beradigan ba'zi bir o‘rnatilgan sinf atributlari mavjud.

O‘rnatilgan sinf atributlari quyida keltirilgan:

__dict__ - Bu sinf nomlari maydoni haqidagi ma'lumotlarni o‘z ichiga olgan lug'atni taqdim etadi.
__doc__ - U sinf hujjatiga ega bo‘lgan qatorni o‘z ichiga oladi
__name__ - U sinf nomiga kirish uchun ishlatiladi.
__module__ - Ushbu sinf aniqlangan modulga kirish uchun foydalaniladi.
__bases__ - Unda barcha asosiy sinflarni o‘z ichiga olgan korniş mavjud.

Misol:

```
class Student:
        def __init__(self,name,id,age):
                self.name = name;
                self.id = id;
                self.age = age

        def display_details(self):
                print("Name:%s,                    ID:%d,
age:%d"%(self.name,self.id))

        s = Student("John",101,22)
        print(s.__doc__)
        print(s.__dict__)
        print(s.__module__)
        {'name': 'John', 'id': 101, 'age': 22}
        __main__
```

3. Dunder methodlar

Dunder methodlar nomi orqali murojaat qilinmaydigan methodlar bo‘lib ularni ishga tushirish uchun belgi va amallardan foydalaniladi shunga ko‘ra ifoda yoki belgiga asosan birqancha gruhlarga ajraladi.

Arifmetik amallarga javob beruvchi dunder methodlar:

- __sub__(self,p2) (-)
- __isub__(self,p2) (-=)
- __mul__(self,p2) (*)
- __imul__(self,p2) (*=)
- __truediv__(self,p2)(\ )
- __itruediv__(self,p2) (\=)
- __floordiv__(self,p2) (\\ )
- __ifloordiv__(self,p2) (\=)
- __add__(self,p2) (+)
- __iadd__(self,p2) (+=)

```
class point:
   x = None
   y = None

   def __init__(self, x , y):
```

```
    self.x = x
    self.y = y

def __str__(self):
    s = f'({self.x},{self.y})'
    return s

def __add__(self,p2):
    print("In add")
    x = self.x + p2.x
    y = self.y + p2.y
    return point(x,y)

def __iadd__(self,p2):
    self.x += p2.x
    self.y += p2.y
    return self

def __isub__(self,p2):
    self.x -= p2.x
    self.y -= p2.y
    return self

def __imul__(self,p2):
    self.x *= p2.x
    self.y *= p2.y
    return self

def __itruediv__(self,p2):
    self.x /= p2.x
    self.y /= p2.y
    return self

def __ifloordiv__(self,p2):
    self.x //= p2.x
    self.y //= p2.y
    return self
```

```
    def __call__(self):
        print(f"Called Point {self.x},{self.y}")
```

munosabat belgilari asosida ishga tushuvchi dunder methodlar:

- __eq__(self,p2) (==)
- __lt__(self,p2) (<)
- __le__(self,p2) (<=)

```
class Avto:
    num_avtos = 0
    """Avtomobil klassi"""
    def __init__(self,make,model,rang,yil,narh):
        """Avtomobilning xususiyatlari"""
        self.make = make
        self.model = model
        self.rang = rang
        self.yil = yil
        self.narh = narh
        Avto.num_avtos += 1

    @classmethod
    def get_num_avto(cls):
        return cls.num_avtos

    def __str__(self):
        """Obyekt haqida ma'lumot"""
        return f"Avto: {self.make} {self.model}. {self.narh}$"

    def __repr__(self):
        """Obyekt haqida ma'lumot"""
        return f"Avto: {self.make} {self.model}. {self.narh}$"

    def __eq__(self,boshqa_avto):
        return self.narh == boshqa_avto.narh

    def __lt__(self,boshqa_avto):
        return self.narh<boshqa_avto.narh
```

```
    def __le__(self,boshqa_avto):
        return self.narh<=boshqa_avto.narh

    def get_info(self):
        return f"{self.rang} {self.make} {self.model}.{self.yil}-yil.
Narhi:{self.narh}$"

class AvtoSalon:
    """Avtosalon klassi"""
    def __init__(self,name):
        self.name = name
        self.avtolar = []

    def __repr__(self):
        return f"{self.name} avtosaloni"

    def __len__(self):
        return len(self.avtolar)

    def __getitem__(self,index):
        return self.avtolar[index]

    def __setitem__(self,index,value):
        if isinstance(value,Avto):
            self.avtolar[index]=value

    def __add__(self,qiymat):
        if isinstance(qiymat,AvtoSalon):
            yangi_salon =  AvtoSalon(f"{self.name} {qiymat.name}")
            yangi_salon.avtolar = self.avtolar + qiymat.avtolar
            return yangi_salon
        elif isinstance(qiymat,Avto):
            self.add_avto(qiymat)
        else:
            print(f"AvtoSalon ga {type(qiymat)} qo‘shib bo‘lmaydi")

    def __call__(self,*param):
        if param:
```

```
            for avto in param:
                self.add_avto(avto)
        else:
            return [avto for avto in self.avtolar]

    def add_avto(self,*qiymat):
        for avto in qiymat:
            if isinstance(avto,Avto):
                self.avtolar.append(avto)
            else:
                print("Avto obyketini kiriting")

    def get_list(self):
        return [avto for avto in self.avtolar]

avto1 = Avto("GM","Malibu","Qora",2020,40000)
avto2 = Avto("GM","Lacetti","Oq",2020,20000)
avto3 = Avto("Toyota",'Carolla',"Silver",2018, 45000)
avto4 = Avto("Mazda", "6", 'Qizil',2015,35000)
avto5 = Avto("Volkswagen","Polo",'Qora',2015,30000)
avto6 = Avto("Honda","Accord","Oq",2017,42000)

salon1 = AvtoSalon("MaxAvto")
salon1(avto1,avto2,avto3)
#print(salon1())

salon2 = AvtoSalon("Avto Lider")
salon2(avto4,avto5,avto6)
#print(salon2())
salon3=salon1+salon2
print(salon3())

#print(avto1.get_info())
```

Nazorat savollari

1. Konstruktor nima va uning vazifasi ayting.
2. Parametrlangan konstruktorga misol keltiring.
3. Parametrlanmagan konstruktorga misol keltiring.

4. Python ichki sinf imkoniyatlaridan qanday foydalaniladi?
5. Dunder methodlarga qanday murojaat qilinadi?
6. Dunder methodlarga qanday gruhlarga ajratiladi.
7. Dunder methodlar ahamiyatini tushuntiring.
8. Init qanday methodlar gruhlariga kiradi.

Ma’ruza №3

Mavzu: Sinflarda vorislik tushunchasi.

Reja:

1. Vorislik tushunchasi
2. super() funksiyasidan foydalanish
3. Voris sinfga funksiya qo‘shish

1. 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 **parent class** deyiladi. Meros qilib olingan funksiyalarni o‘ziga o‘zlashtiradigan sinf voris sinf deyiladi. Parent class hosil qilish Istalgan sinf parent class bo‘lishi mumkin. Shu sababli parent classni hosil qilish xuddi oddiy sinfni hosil qilish kabidir.

Hozir Odam deb nomlangan 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 ("Abbosbek", "Ibragimov")
x.tanish()
```

Abbosbek Ibragimov

Voris sinf hosil qilish

Voris sinf hosil qilish uchun yangi sinf tuzilayotganda parent classni parametr sifatida kiritamiz. Shunda voris sinf parent classdan barcha xususiyatlarni o‘zlashtiradi. Hozir **Talaba** deb nomlangan sinf hosil qilamiz. Odam sinfi uning parenti (ota) sinfi bo‘ladi. Qavslar ichida parent classni kirittamiz 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 hechnarsa yozishni istamasak xatolik yuz bermasligi uchun pass kalit so‘zini qo‘shib qo‘yamiz:

```
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):
    pass
x = Talaba ("Asadbek", "Suvonov")
x.tanish()
```

Asadbek Suvonov

Yuqoridagi Odam sinfi shu kunga qadar yaratib kelayotgan odatiy sinflarimiz bilan bir xil biroq Talaba sinfida Odam sinfi nomi parameter sifatida qavsda ko‘rsatib o‘tilmoqda va shuning o‘zi Odam sinfini Parent sinfga aylantiradi, shu bilan birgalikda Talaba sinfi Odam sinfidan vorislik oladi. Endi x o‘zgaruvchiga Talaba sinfiga tegishli obyekt yaratdik vaxolangki Talaba sinfida konstruktor ham tanish methodi ham mavjud emas bularning barchasi Odam sinfidan merosxo‘rlik asosida olinmoqda.

__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 parent classdagi __init__() funksiyasidan emas o‘zidagi konsturoktordan 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 ("Asadbek", "Suvonov")
x.tanish()
```
Asadbek Suvonov

Ammo parent classdagi __init__() funksiyasidan foydalanmoqchi bo‘lsak, voris sinfdagi __init__() funksiyasi ichiga parent classning shu funksiyasini yozamiz:

```
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):
        Odam.__init__(self, ism, familiya)
x = Talaba ("Asadbek", "Suvonov")
x.tanish()
```
Asadbek Suvonov

2. super() funksiyasidan foydalanish

Sinflar bilan ishlash uchun maxsus super() funksiyasidan foydalanish ham mumkin. Bu funksiya parent classdagi barcha funksiya va parametrlarni voris sinfga o‘zlashtiradi:

```
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 ("Asadbek", "Suvonov")
```

```
x.tanish()
```
Asadbek Suvonov

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:

```
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)
        self.yil = 2002
x = Talaba ("Asadbek", "Suvonov")
print(x.yil)
```
2002

Yuqoridagi misolimizda yangi parametrni qo‘shgan zahotimiz 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 ("Asadbek", "Suvonov", 2002)
print(x.yil)
```
2002

Ushbu voris sinfga etibor bersak unda ajdodiga nisbatan atributlarining ko‘pligini ko‘ramiz bu vaziyatda ajdod sinfga kerak bo‘lgan miqdordagi atributlar yuboriladi qolgan atributlarni saqlash va ular bilan ish olib boorish vazifasi voris sinfga qoladi.

3. Voris sinfga funksiya qo‘shish

Voris sinfga qo‘shimcha funksiyalar ham qo‘shish mumkin. Natijada u parent classdan 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 = 2002

    def tugilgan(self):
        print("Men" , self.yil , " - yilda tug'ilganman")
x = Talaba ("Asadbek", "Suvonov", 2002)
x.tugilgan()
```

```
Men 2002 - yilda tug'ilganman
```

Nazorat savollari

1. Vorislik nima va u qayerlarda qo‘llaniladi?
2. Parent sinf deb qanday sinflarga aytiladi?
3. Voris sinf qanday yaratiladi?
4. super( ) qanday funksiya?
5. Voris sinfga methodlat yozishga misollar keltiring.
6. Parent va voris sinflarda bir xil nomli methodlar bo‘lsa ish qaytartibda boradi.
7. Parent sinfning dunder methodlaridan foydalanishga misollar keltiring.
8. Meros xo‘rlikda konstruktorlarning o‘zaro bog‘liqligi tushuntiring.

Ma’ruza №4

Mavzu: MVT texnologiyasi django framework.

Reja:

1. Django haqida
2. MVT texnologiyasi
3. Djangoda app va url dispatcher
4. Djangoni o‘rnatish, loyiha yaratish va app hosil qilish.
5.

1. Django haqida

Django - python dasturlash tilida veb ilovalar ishlab chiqish uchun ajoyib framework. Framework python dasturlash tilida yozilgan. Django frameworki birinchi bor 2005-yil 21-iyulda ishga tushirilgan. Loyihani ishlab chiquvchilari Django Software Foundationdir va loyiha ular tomonidan qo‘llab-quvvatlanadi. Dasturchilar: Rasl Keyt Magi, Adrian Holovaty, Saymon Vilson, Yakob Kaplan Moss, Uilson Mayner. Framework krossplatformalikdir (ya'ni barcha operatsion tizimlarni qo‘llab quvvatlaydi). Djangoda yaratilgan sayt bir yoki bir nechta ilovalardan tashkil topishi mumkin. Bu esa frameworkning arxitekturasini boshqalaridan farqli jihatidir (Misol uchun:Ruby on Rails).

Frameworkning ishlash prinsplaridan biri bu - DRY (Don't repeat yourself- "o‘z-o‘zingni takrorlanma"). Bu degani, siz sayt yozish davomida siz "velosiped"ni qaytadan ixtiro qilmaysiz, u siz uchun tayyor. Ya'ni bir kodni qaytadan yozishga hojat yo‘q, bu esa veb ilovaning yaratilish protsessini tezlashtiradi.

Boya aytib o‘tganimdek, Django 2005-yilda ishlab chiqildi va shu vaqtdan boshlab bosqichma-bosqich eng yaxshi frameworklar qatoridan o‘rin egalladi. Framework dasturchilarning muammolarini bir necha daqiqa mobaynida hal etib bermoqda. Django veb ilovalarning yozilish jarayonini sezilarli darajada yengillashtirdi va veb ilovalar yaratish uchun ajoyib, soddalashtirilgan yo‘lni taqdim eta oldi. Masshtablilik va qayta foydalanish imkoniyati: Django kodni qayta ishlatish va modullikni targ'ib qiladi, bu esa dasturchilarga ularni qayta foydalanish mumkin bo‘lgan modullar yoki ilovalarga bo‘lish orqali kengaytiriladigan ilovalarni yaratishga imkon beradi.

ORM va ma'lumotlar bazasini abstraktsiya qilish: Django‘ning Ob'ekt bilan bog'liq xaritalash (ORM) qatlami ma'lumotlar bazasini

boshqarishning murakkabliklarini mavhumlashtiradi, bu esa dasturchilarga Python ob'ektlari va so‘rovlari yordamida ma'lumotlar bazalari bilan ishlash imkonini beradi.
URL yo‘naltirish: Django moslashuvchan URL marshrutlash tizimini taqdim etadi, bu esa ishlab chiquvchilarga ilovaning turli qismlari uchun toza va foydalanuvchilar uchun qulay URL manzillarini aniqlash imkonini beradi.

Template Engine: Django shablon mexanizmi HTML shablonlarini loyihalash va renderlash jarayonini soddalashtiradi, bu esa dinamik veb-sahifalarni yaratishni osonlashtiradi.

Autentifikatsiya va avtorizatsiya: Django foydalanuvchi autentifikatsiyasi va avtorizatsiyasi uchun mustahkam o‘rnatilgan vositalarni taklif etadi, bu esa ilovaning turli qismlariga xavfsiz kirishni nazorat qilishni ta'minlaydi.

Administrator interfeysi: Django ishlab chiquvchilarga qo‘shimcha kod yozmasdan ilova ma'lumotlarini boshqarish imkonini beruvchi intuitiv boshqaruv interfeysini taqdim etadi.

Django loyihasini ishga tushirish SQL-dan foydalanmasdan Python-da ilovangizning butun ma'lumotlar modelini yaratishga imkon beradi. Ob'ektga aloqador xaritalash vositasidan (ORM) foydalanib, Django to‘liq Python muhitida ishlashni osonlashtirish uchun an'anaviy ma'lumotlar bazasi strukturasini Python sinflariga aylantiradi. Django-MySQL JSON ma'lumotlar turini va tegishli funktsiyalarni qo‘llab-quvvatlaydi.

Django‘da ma'lumotlar bazasi jadvallari Python sinflariga aylanadi. Veb-ilovalar Django modellari orqali ma'lumotlarga kirish va boshqarish. Ma'lumotlar bazasi maydonlari oddiygina sinf atributlariga aylantiriladi. Agar siz Python-da sinf atributlari ta'rifi bilan tanish bo‘lsangiz, Django ma'lumotlar bazasini osongina loyihalashingiz va boshqarishingiz mumkin.

Django Web Framework ilovangiz ma'lumotlar bazasi bilan to‘liq integratsiya qilish uchun yorliqni taklif qiladi. U CRUD (yaratish, o‘qish, yangilash, o‘chirish) funksiyalarini, HttpResponse va saytlararo skriptlarni taqdim etadi, foydalanuvchilarni boshqarish imkoniyatlarini ta'minlaydi, dasturiy ta'minotni boshqarish funktsiyalarini taklif qiladi va boshqalar. Siz paketlarni import qilasiz, ma’lumotlar bazasiga ulanasiz va so‘ngra mahsulotingizni noyob qiladigan ilovangiz qismlarini ishlab chiqishga qaytasiz.

2. MVT texnologiyasi

MVT atamasi djangoning asosini tashkil etuvchi uchqismining nomlaridan olinga bo‘lib ular Model, View va Template lardir. Ularga birmabir toxtaladigan bo‘lsak.

View

Nomlanishi «ko‘rinish» ma’nosini bersa ham, aslida, views ko‘rinishga javob beradigan qism emas. Views django loyihaning «miyasi» sifatida ishlaydi. Mantiqiy qismlar shu yerda yoziladi. Qolgan deyarli barcha qismlar shu qism orqali biriktiriladi va view ularni nazorat qiladi. Serverga kelgan request(so‘rov)larga view javob beradi. Masalan, djangoda yozilgan saytga browser orqali «<saytnomi>/login/» qismiga request yuborsak, url dispatcher(bu haqida pastroqda) requestni tutib olib, «login/» qismi uchun javobgar viewga yuboradi. View esa requestni analiz qilib, kerakli vazifalarni bajaradi va bizga response(javob) yuboradi.

Model

Hozirda deyarli barcha web loyihalar ma’lumotlar bazasidan foydalanadi. Pythonda yozilgan serverimiz ma’lumotlar bazasi(deylik, PostgreSQLda bo‘lsin) bilan aloqa o‘rnatishi uchun bizga o‘rtada «ko‘prik» kerak bo‘ladi. Django ORM biz uchun mana shu vazifani bajaradi.

Object relational mapper(ORM) – boshqa tur(masalan, ma’lumotlar bazasi)dagi ma’lumotlarni biror dasturlash tiliga obyekt sifatida o‘tkazib, foydalanish imkonini beruvchi qism.

Model esa Django ORMning bir qismi. Demak, u ham bizning serverimizni database bilan bog’lashga xizmat qiladi.

Agar siz SQL tillaridan xabardor bo‘lsangiz, ma’lumotlar SQL oilasidagi ma’lumotlar bazasida jadval ko‘rinishida saqlanishidan xabardor bo‘lsangiz kerak. SQLda biz column(ustun)lardan qolip sifatida foydalanamiz. Row(qator)larda esa o‘sha qoliplardan foydalanib yaratilgan ma’lumotlar bo‘ladi.

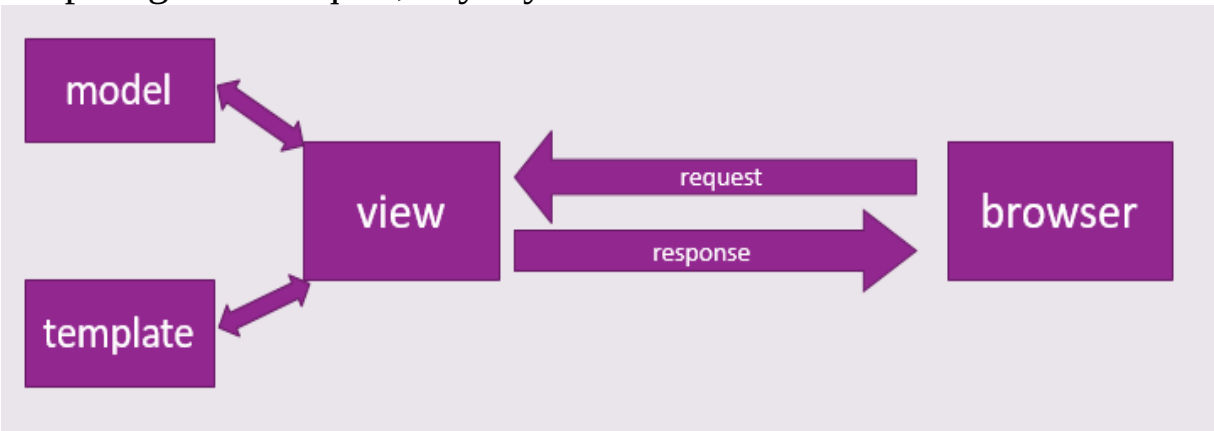
Djangoda bitta model bitta class orqali ifodalanib, u bitta «qolip», ya’ni jadvalni hosil qiladi. Uning ichidagi fieldlar databaseda ustunlarga to‘g’ri keladi. Bu model classdan olingan obyektlar esa row, ya’ni qatorlarni bildiradi.

Template

Bizga response qaytarilganda browserga keladigan ma’lumot HTML, CSS va JavaScript fayllari ko‘rinishida keladi(F12 ni bosib

ko‘ring). Bu ma’lumotlar tarkibi shartli ravishda ikki xil bo‘ladi: Tayyor shablon(qolip) va o‘zgarib turadigan ma’lumotlar. Masalan, youtubeda alohida 2 ta oynada 2 xil video ochib ko‘ring. E’tibor bering, sahifa dizayni, pageda qismlarning joylashgan joyi, rangi, Tepadagi «Youtube» yozuvi bir xil. Bu template(qolip) deyiladi. Lekin, ikkita pageda video, video nomi, tavsiya qilingan videolar va h.k har xil. Bu o‘zgarib turadigan ma’lumotlar.

Har bir ma’lumot uchun alohida HTML kod yoza olmaymiz. Shunchaki, qolib yozib olamiz va kerakli ma’lumotlarni bu qolipdagi kerakli joylarga «yopishtirib» chiqamiz. Bu jarayon rendering deyiladi. Tepada view bizga response qaytarishini bilib oldik. Ana shu response(aniqrog’i HTTP response) esa kerakli ma’lumotlarni tayyor templatege render qilib, keyin yuboriladi.

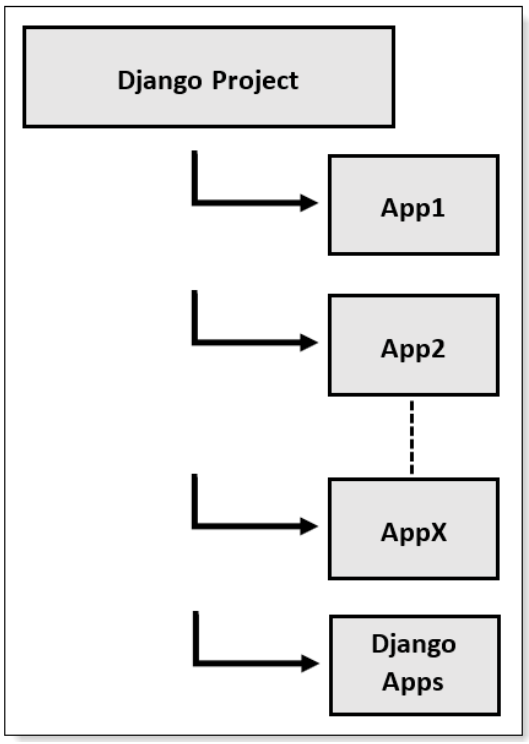


3. Djangoda app va url dispatcher

App

Loyiha kattalashgan sari uni alohida qismlarga ajratib, boshqarishga oson(managable) qilishga ehtiyoj tug’iladi. Bunda har bir qismning aniq vazifasi bo‘lishi kerak.

Bu vazifani djangoda app’lar bajaradi. Har bir app loyihaning o‘z vazifasiga ega bo‘lgan qismi. Masalan, Online bozor uchun backend yozganimizda to‘lovlar uchun alohida app, mahsulotlar uchun alohida app, foydalanuvchilar uchun alohida app va h.k. qilishimiz mumkin. Shunda loyihamiz tushunishga va boshqarishga osonroq bo‘ladi. Djangoda har bitta app o‘zining MVT dizayniga ega. Ya’ni har bir appning ichida alohida biz yuqorida ko‘rib chiqqan model, view va template bor.



URL dispatcher

Views requestlarni qabul qilib, response qaytarishini bilib oldik. Lekin, unga qaysi request aynan qaysi viewga murojat qilishini biladigan qism – url dispatcher kerak. Aynan mana shu qism birinchi bo‘lib clientdan kelgan requestni kutib oladi va request kelgan URL manziliga qarab kerakli viewga o‘tkazib yuboradi. Demak, url dispatcher requestlarni kerakli viewga yetkazib beradi. Url dispatcher url pattern yoki «path»lardan tashkil topgan. Qisqasi, dispatcher requestlarni kutib oladigan manzillar. Masalan, «login/» bu url pattern. Unga bitta view biriktiriladi va har safar client «<sayt nomi>/login/» qismiga request yuborganda dispatcher uni biriktirilgan viewga yuboradi. Vazifasiga qarab url patternlarni 2 turga ajratish mumkin: to‘g’ridan-to‘g’ri biror viewga yuboradigan va boshqa url dispatcherga yuboradigan.

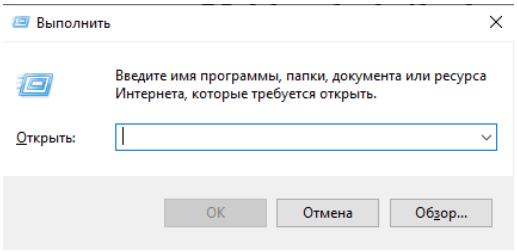
Project yaratilgan vaqtda loyiha konfiguratsiyalari turgan folderda url dispatcher avtomatik yaratiladi. Qulaylik uchun odatda, har bir app ichida ham alohida dispatcher yaratilib, bosh dispatcherga ulanadi.

Demak, o‘rganganlarimizni bir joyga yig’ib olsak. Siz browser orqali serverga request yubordingiz. Serverdagi dispatcher requestni tutib oladi va kerakli viewga yuboradi. View esa kerakli amallarni bajarib, zarur bo‘lsa models orqali ma’lumotlar bazasiga murojaat

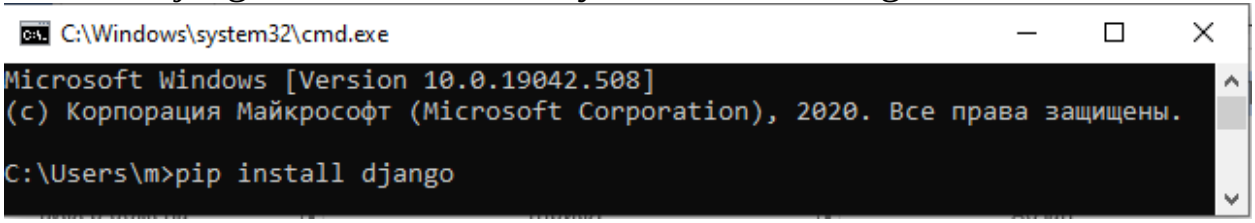
qiladi. Keyin ma'lumotlarni kerakli templatega qo'yib, sizga response ko'rinishida yuboradi.

4. Djangoni o'rnatish, loyiha yaratish va app hosil qilish.

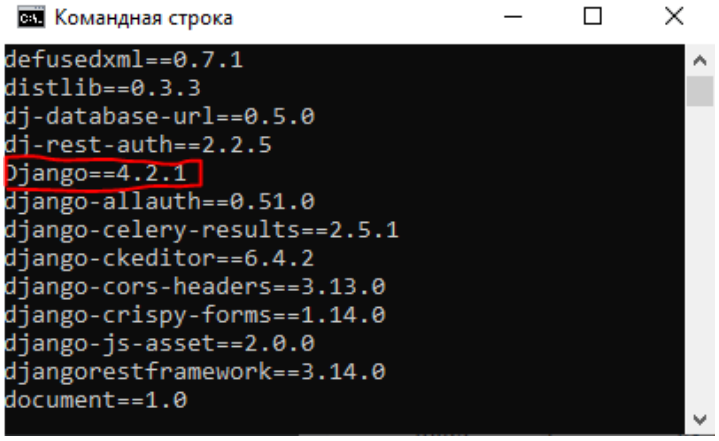
Keling endi Django frameworkini o'rnatib ko'raylik. Buning uchun avvalambor python.org rasmiy sahifasidan python dasturlash tili muhitini yuklab olamiz va o'rnatamiz. O'rnatishda etibor berish lozim bo'lgan bir qism bo'lib quyidagi rasmda belgilanganidek ikki bo'limni belgilash lozim.



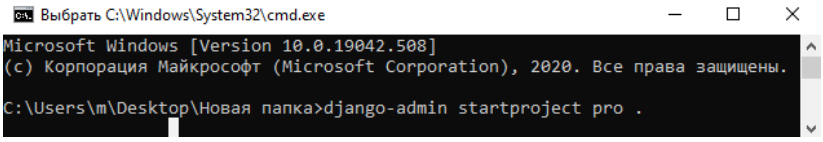
Ushbu belgilash orqali pythonning pip kutubxonalar paketini o'rnatishga ruxsat berasiz keyinchalik pip orqali o'rnatiluvchi barcha framework va modular uchun zarur. Shundan so'ng Install Now bandi bosiladi va ornatish boshlanadi. O'rnatish yakunlangach klaviatuadan windows+r kombinatsiyasini bosib выполнить oynasi ochiladi va uyerga cmd komandasi yozilib ok tugmasi bosiladi. Ekranda Командная строка oynasi ochiladi va biz djangoni o'rnatish uchun pip install django komandasini yozib enter tugmasini bosamiz.



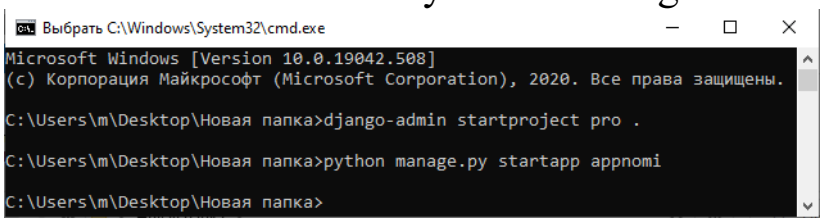
Yuqoridagi jarayonda internet yoniq xolda bo‘lishi lozim chunki o‘rnatishda fayllar internet orqali olinadi. O‘rnatish yakunlangach djangoni o‘rnatilish mofoqiyatli yakunlanganligini bilsh uchun “pip freeze” komandasini yozib enter tugmasini bosamiz. Kutubxonalar ro‘yxatida django ko‘rinishi o‘rnatish mofoqiyatli bo‘lganini anglatadi django nomidan so‘ng kelgan 4.2.1 bu versiya nomi u sizda boshqacharoq bo‘lishi mumkin chunki vaqt o‘tishi bilan versiyalar yangilanadi.

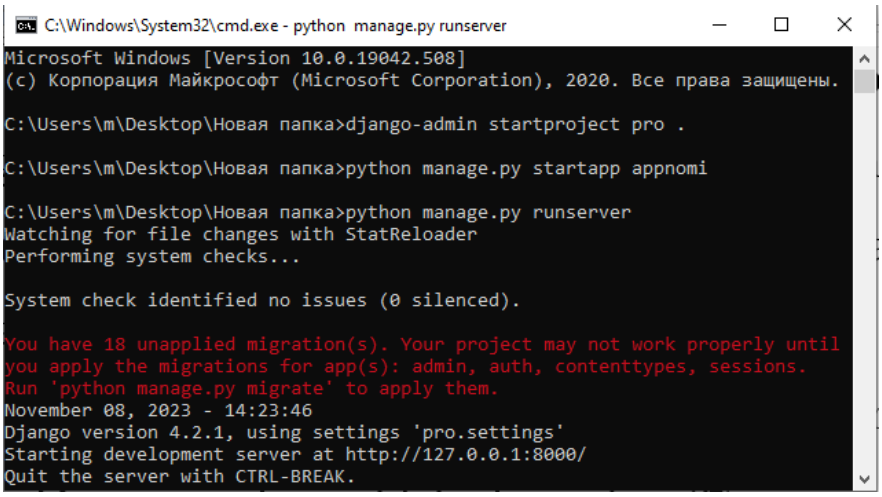


Endi oynani yopamiz va proyekt tayyorlash uchun bir papka yaratib uni ochib olamiz va ochilgan papkaning manzil qatoridagi yozuvlarni o‘chirib tashlab cmd komandasini yozib enter tugmasini bosamiz. Bizga yana Командная строка oynasi ochib beriladi bus afar u biz yaratgan papka adresiga kirgan xolda ochiladi. Endi proyekt yaratish uchun “django-admin startproject pro .” komandasini yozib enter tugmasi bosiladi.

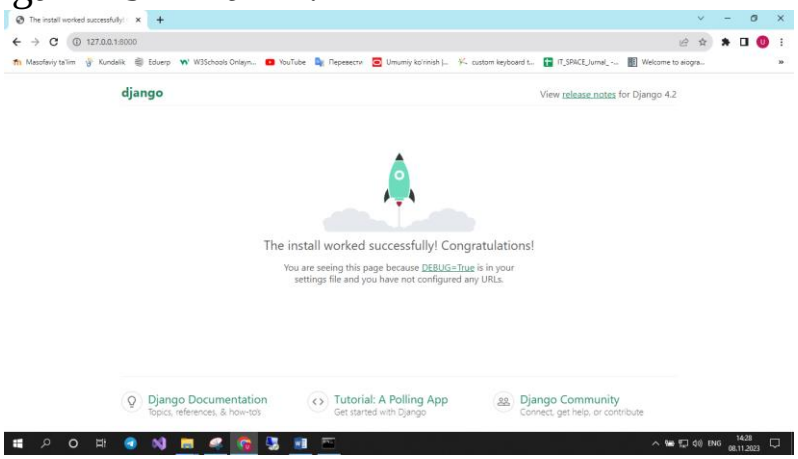


Ushbu proyekt uchun app yaratish uchun shu Командная строка ga quyidagi komandani yozasiz “python manage.py startapp appnomi” ayni vaqtda pro nomli proyektimiz va appnomi nomli qism dasturimiz bor djangoning to‘g‘ri ishlayotganini tekshirish uchun proyektimizni ishga tushurib ko‘ramiz. Buning uchun Командная строка ga “python manage.py runserver” komandasini yozib enter tugmasini bosamiz.





Yuqoridagi ko‘rinishda oyna hosil bo‘lishi django to‘g‘ri o‘rnatilganligi va siz proyektni to‘g‘ri yaratganingizni anglatadi django frameworkining o‘zida virtual web server mavjud bo‘lib ayni vaqtda localhost ning 8000 portida loyigani isga tushurdi. Biz uni ko‘rish uchun kompyuterimizdagi istalgan brouzerdan http://127.0.0.1:8000 adresiga kirib ko‘ramiz.



Nazorat savollari

1. Django qachon ishlab chiqarilgan?
2. Django qaysi davlatda ishlab chiqarilgan?
3. MVT texnologiyasini tushuntirib bering
4. ORM nima?
5. App va url nima?
6. Django qanday o‘rnatiladi?
7. Django proyekt qanday yaratiladi?
8. App qanday hosil qilinadi?

Ma’ruza №5

Mavzu: Djangoda modellar tayyorlash.

Reja:

1. Model qanday ishlaydi?
2. ORM (Object-Relational Mapping)
3. Models parent class fildlari

1. Model qanday ishlaydi?

Barchangizni yodingizda bo‘lsa python obyektga yo‘naltirilgan dasturlash tili edi va bu jarayon djangoda ham keng qo‘llaniladi. Oldingi darsimizdan sizga ma’lumki model ma’lumotlarni saqlash, o‘zgartirish, o‘cherish, saralash va shu kabi vazifalar uchun javob beradi. Model qismida ham biz python dasturlash tili asosida kodlar yozamiz va yana ham soddaroq qilib aytsak sinflar shakllantiramiz sinf atributlari ma’lumotlar omborining ustunlari sifatida ko‘riladi, methodlari esa ma’lumotlar omboriga tasir ko‘rsatmaydi.

Hozirda deyarli barcha web loyihalar ma’lumotlar bazasidan foydalanadi. Pythonda yozilgan serverimiz ma’lumotlar bazasi(deylik, PostgreSQLda bo‘lsin) bilan aloqa o‘rnatishi uchun bizga o‘rtada «ko‘prik» kerak bo‘ladi. Django ORM biz uchun mana shu vazifani bajaradi.

Object relational mapper(ORM) – boshqa tur(masalan, ma’lumotlar bazasi)dagi ma’lumotlarni biror dasturlash tiliga obyekt sifatida o‘tkazib, foydalanish imkonini beruvchi qism.

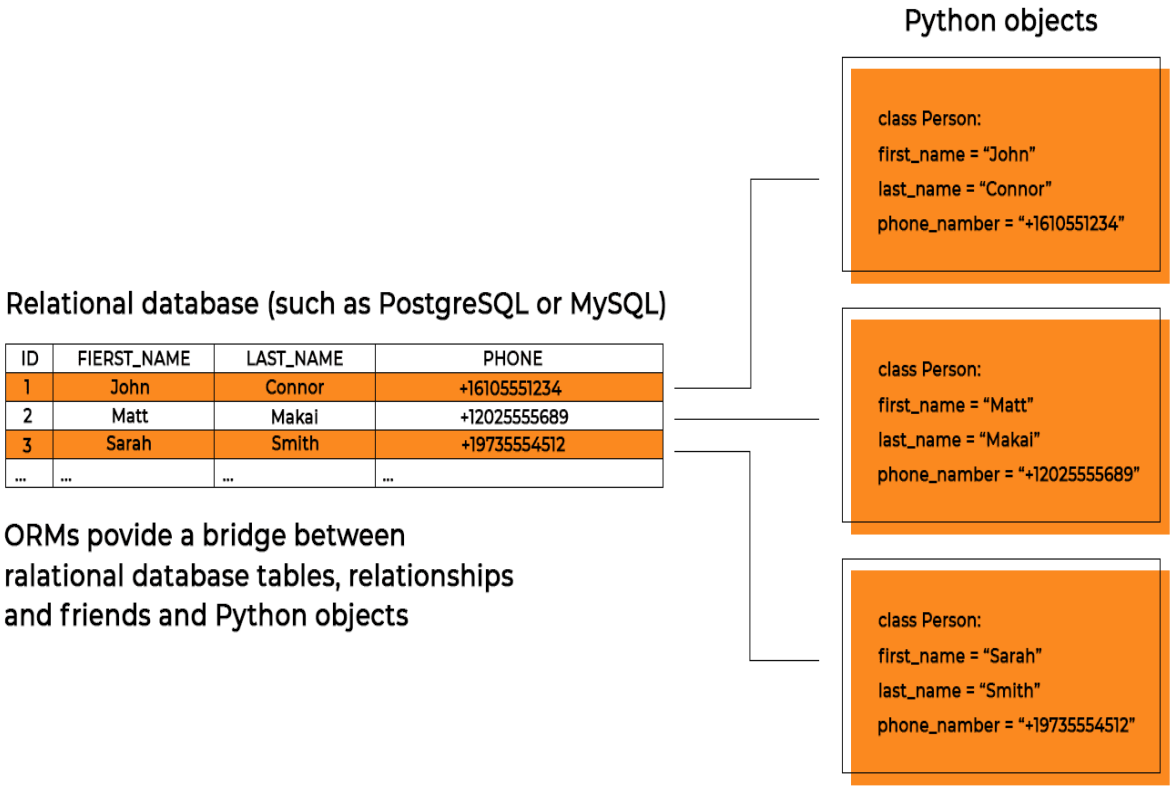
Model esa Django ORMning bir qismi. Demak, u ham bizning serverimizni database bilan bog’lashga xizmat qiladi.

Agar siz SQL tillaridan xabardor bo‘lsangiz, ma’lumotlar SQL oilasidagi ma’lumotlar bazasida jadval ko‘rinishida saqlanishidan xabardor bo‘lsangiz kerak. SQLda biz column(ustun)lardan qolip sifatida foydalanamiz. Row(qator)larda esa o‘sha qoliplardan foydalanib yaratilgan ma’lumotlar bo‘ladi.

Djangoda bitta model bitta class orqali ifodalanib, u bitta «qolip», ya’ni jadvalni hosil qiladi. Uning ichidagi fieldlar databaseda ustunlarga to‘g’ri keladi. Bu model classdan olingan obyektlar esa row, ya’ni qatorlarni bildiradi.

2. ORM (Object-Relational Mapping)

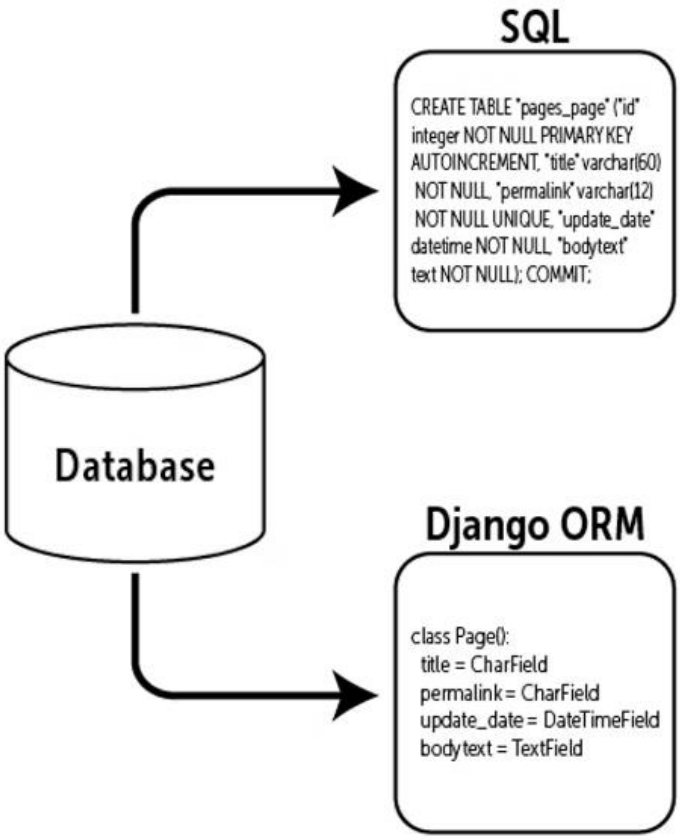
Djangoda ma'lumotlar ombori(ma'lumotlar bazasi) bilan ishlash uchun o‘zining ORMini ishlatadi va obyektning modelini pythondagi klasslar yordamida aniqlaydi, shu orqali ma'lumotlar ombori sxemasi shakillanadi:



Djangoda ORMning ishlash sxemasi.

ORM - "Object-Relational Mapping". Dasturlashning bu texnologiyasi ma'lumotlar omobori bilan uzviy bog'liq bo‘lib, obyektga yo‘naltirilgan dasturlash tilllari konsepsiyasi asosida ishlaydi va virtual "obyektlar ma'lumotlar omborini" yaratadi.

Shuni quvonarliki, freymvorkdagi ORM sababli dasturchi uchun ma'lumotlar ombori bilan ishlay oladigan maxsus tilni bilish talab etilmaydi! ORM kodni minimal ravishda o‘zgartirgan holatda ma'lumotlar omboriga kirish imkoniyatini beradi.



3. Models parent class fildlari

Biz model yozar ekanmiz u to‘g‘ridan to‘g‘ri ORMga bog‘lanib qolmaydi bu bog‘liqlikni tashkil etish uchun django frameworkining maxsus Models sinfidan merosxo‘rlik asosida voris sinflar yaratamiz va shu orqali Djangoning ORM bilan bog‘liqlik taminlanadi, bundan tashqari merosxo‘rlik asosida yaratilgan voris sinfga shunchaki atribut elon qilish bilan ma’lumotlar omboriga ustunlar hosil qila olmaymiz buning uchun maxsus models sinfida paramertlar bor va ular bilan hozir tanishib chiqamiz.

AutoField

Class AutoField (**variantlar)

U avtomatik ravishda ko‘payadigan butun son maydonini yaratadi.

BigAutoField

Class BigAutoField (**variantlar)

AutoFieldga o‘xshab, u 64 bitli butun son va 9223372036854775807 gacha raqamlarni saqlashi mumkin.

BigIntegerField

Class BigIntegerField (**variantlar)

U -9223372036854775807 dan 9223372036854775807 gacha bo‘lgan 64 bitli butun sonlarni saqlashi mumkin.

BinaryField

Class BinaryField (**variantlar)

Bu xom ikkilik ma'lumotlarni, shu jumladan baytlarni saqlashi mumkin.

BooleanField

Class BooleanField (**variantlar)

U True va False kabi mantiqiy qiymatlarni saqlash uchun ishlatiladi. Mantiqiy maydonning standart qiymati, agar aniqlanmagan bo‘lsa, False.

CharField

Class CharField(max_length = None, **variantlar)

CharField do‘konlari - bu kichik va katta o‘lchamdagi satrlarni saqlashi mumkin bo‘lgan satr maydoni. Juda katta o‘lchamlar uchun TextField dan foydalanish kerak.

DateField

class DateField(auto_now=False, auto_now_add=False, **variantlar)

U Python'da datetime.date misolida ko‘rsatilgan sanani saqlaydi.

DateTimeField

class DateTimeField(auto_now=False, auto_now_add=False, **variantlar)

Pythonda datetime.datetime bilan ifodalangan sana va vaqtni saqlaydi.

DecimalField

class DecimalField(max_digits=Yo‘q, decimal_places=Yo‘q, **variantlar)

U o‘nlik sonlarni belgilangan aniqlikgacha saqlashi mumkin. O‘nli kasrlar sonini decimal_places argumenti bilan aniqlash mumkin.

EmailField

sinf EmailField(*max_length=254 , **variantlar*)

U CharField-ga ham o‘xshaydi, lekin kiritilgan matn elektron pochta manzili yoki yo‘qligini tekshirish uchun qo‘shimcha xususiyatga ega.

FileField

class FileField(*upload_to=Yo‘q, max_length=100, **variantlar*)

Ushbu maydonda foydalanuvchi fayllarni yuklashi mumkin.

FloatField

Class FloatField (**variantlar)

FloatField suzuvchi nuqta raqamlarini saqlashi mumkin.

ImageField

class ImageField(*upload_to=Yo‘q, height_field=Yo‘q, width_field=Yo‘q, max_length=100, **variantlar*)

Bu, shuningdek, FileField turidir, lekin unga qo‘shimcha ravishda ImageField kirish fayli Tasvir yoki yo‘qligini ham tekshiradi.

IntegerField

Class IntegerField(**variantlar)

IntegerField butun sonlarni saqlashi mumkin. -2147483648 dan 2147483648 gacha bo‘lgan butun sonlar hech qanday muammosiz do‘kon bo‘lishi mumkin. Ulardan kattaroq butun sonlar to‘lib ketishiga olib kelishi mumkin.

JSONField

sinf JSONField (*kodlovchi=Yo‘q, dekoder=Yo‘q, **variantlar*)

Bu JSON kodlangan ma'lumotlar uchun maydon. Ushbu ma'lumotlar Pythonda lug'at yoki ro‘yxat shaklida taqdim etiladi.

NullBooleanField

Class NullBooleanField (***variantlar*)

Bu maydon mantiqiy maydonga o‘xshaydi, null = True.

PositiveIntegerField

Class PositiveIntegerField (**variantlar)

PositiveIntegerField IntegerField kabi butun sonlarni saqlaydi, lekin kiritilgan qiymat ijobiy bo‘lsa, tasdiqlaydi. 0 dan 2147483648 gacha bo‘lgan butun sonlarni osongina saqlash mumkin.

PositiveSmallIntegerField

Class PositiveSmallIntegerField *(**variantlar*)

Bu PositiveIntegerField-ga o‘xshaydi, lekin faqat 0 dan 32767 gacha bo‘lgan butun sonlarga ruxsat beradi.

TextField

Class TextField *(**variantlar*)

U katta matnni saqlaydi. TextField uchun standart vidjet matn qutisi hisoblanadi.

TimeField

Class TimeField(*auto_now=False, auto_now_add=False, **variantlar*)

Bu Vaqt maydoni. Python-da u datetime sifatida ifodalanadi. Vaqt.

Ushbu maydonlardan tashqari, muayyan holatlarda ishlatilishi mumkin bo‘lgan boshqa ko‘plab sohalar ham mavjud.

Model maydonlaridan foydalanishga misol

birinchi_ism = modellar.CharField(maksimal_uzunlik = 50)

Bu xaritalangan jadvalda maksimal uzunligi 50 bo‘lgan VARCHAR ma'lumotlar turi bilan first_name ustunini yaratadi.

Yosh = IntegerField()
Bu ma'lumotlar turi sifatida butun sonli Age ustunini yaratadi.

Relational Fields

Yuqorida ko‘rsatilgan maydonlar turli xil toifalarga tegishli ma’lumotlar omborida ustunlar yaratishga moslashgan biroq ma’lumotlar omborida ish olib brogan foydalanuvchilar biladiki ma’lumotlar omborida jadvallar o‘rtasida bog‘lanishlarning uch xil turi mavjud va djangoda ham bu bog‘lanishlardan jud keng foydalaniladi va django ham munosabatlarni ifodalovchi maydonlar to‘plamiga ega.

Foreign Key

Class ForeignKey (*to, on_delete, **options*)
U Ko‘pdan-birga aloqasi bo‘lgan maydonni yaratadi. Bu ikkita argumentni talab qiladi, ya'ni model bog'langan sinf va on_delete parametri.

ManytoManyField

Class ManyToManyField(*to, **options*)
Bu ko‘p-ko‘p munosabatlardir. Shuningdek, u bitta argumentni, ya'ni model bog'langan sinfni talab qiladi.

OnetoOneField

class OneToOneField(*to, on_delete, parent_link=False, **options*)
Model bog'langan sinf bo‘lgan bitta pozitsion argumentni talab qiladigan birma-bir munosabat.

Yuqoridagilar asosida xabar va maqolalar e`lon qilib boruvchi web ilovaning modelini tuzamiz

```
1 from django.db import models
2 from django.contrib.auth.models import User
3 class Post(models.Model):
4     title = models.CharField(max_length=200)
5     img = models.ImageField(upload_to='post_images/')
6     content = models.TextField()
7     created_at = models.DateTimeField(auto_now=True)
8                     author = models.ForeignKey(User,
on_delete=models.CASCADE)
9     def __str__(self):
10         return self.title
```

1-Qatorda django frameworkidan models moduli chaqirib olindi.

2- Djangoning maxsus User sinfi chaqirildi u foydalanuvchilarni saqlovchi ma’lumotlar jadvaliga tegishli, bu sinf haqida kelasi darslarimizda chuqurroq o‘rganamiz.

3- Post nomli voris sinf yaratdik va u Models modulidan Model sinfiga bog‘landi va bu ORM bilan bog‘liqlikni ham taminlaydi, natijada ma’lumotlar omborida Post nomli jadval yaratiladi.

4- Title atributi e’lon qilinayapti va unga models modulidan charfield parametri beriliyapri va unga max_length=200 komandasi berildi bu qatordagi dastur kodi asosida ma’lumotlar omborining Post jadvalida title ustuni hosil qilinadi uning ma’lumot toifasi char tipida bo‘ladi shu bilan birgalikda ushbu ustunning har bir yacheykasiga ko‘pi bilan 200 ta belgi yozishga ruxsat etildi.

5- Img nomli atribut e’lon qilindi va unga models modulidan Imagefield komandasi tanlandi, va unga upload_to='post_images/' parametri berildi bu qator asosida ma’lumotlar omboridagi Post jadvaliga Img ustuni yaratiladi va u o‘zidagi yacheykalarga suratli ma’lumotlar olishga moslashadi upload_to='post_images/' parametri asosida yuklanadigan suratlarni post_images papkasiga saqlaydi.

6- Content nomli atribut e’lon qilindi, unga models modulidan Textfield komandasi berildi. Bu qator asosida ma’lumotlar omboridagi Post jadvalida content nomli ustun shakllantiriladi va u o‘zidagi yacheykalarga keng hajmli matnlarni qabul qiladi.

7- Created_at nomli atribut e’lon qilindi, unga models modulidan DataTimeField komandasi berildi va uning avto_now parametri true qiymat berildi. Ushbu dastur qatori asosida ma’lumotlar omboridagi Post jadvalida created_at nomli ustun shakllantiriladi va u o‘zidagi yacheykalarga vaqt haqidagi ma’lumotlarni qabul qiladi yani yil, oy, kun, soat, minut, sekund va milli sekundlarni o‘z ichiga oladi, avto_now = true parametri esa ma’lumot to‘ldirish jarayonida bu ustun aftomatik ravishda to‘ldirilishini anglatadi.

8- author = models.ForeignKey(User, on_delete=models.CASCADE) ushbu qatordagi dastur kodi esa ma’lumotlar omboridagi Post jadvalida author nomli ustun shakllantiriladi va u o‘zidagi yacheykalarga jadvallarni o‘zaro bog‘lash imkoniyati asosida birga ko‘p bog‘lanish hosil qilgan xolda djangoning standart User madeli bilan bog‘lanish hosil etadi, undagi on_delete=models.CASCADE parametri esa User jadvalidagi obyektlar Post jadvali bilan bog‘liq bo‘lsa o‘chirib tashlashga ruxsat

bermaydi agar User jadvalidan biror obyektni o‘chirib tashlashni istasangiz avvalambor u bila bog‘liq bo‘lgan Post jadvalidagi barcha obyektlarni o‘chirib tashlashga to‘g‘ri keladi shundagina User jadvalidagi obyektni o‘chirishga erishishingiz mumkin.

9- Bu qatorda oldingi darslarda o‘tilgan dunder method yozilmoqda.

10- Bu qatordagi kod asosida str dunder methodimiz Post sinfimizdagi obyektni chop etishga bo‘lgan harakatlarga title atributidagi ma’lumotni qaytaradi.

Nazorat savollari

1. Model ma’lumotlar ombori bilan qanday aloqaga kirishadi?
2. ORM nima?
3. Fieldlarning vazifasini ayting.
4. Talabani obyekt sifatida qarab unga mos model tuzing.
5. Sonli toifalarga javob beruvchi nechta Fieldni bilasiz?
6. Mantiqiy toifaga asoslangan Fieldlarga misol keltiring.
7. Jadvallarni o‘zaro bog‘lanishi uchun qanday fieldlar mavjud?
8. Faylli ma’lumotlar uchun javob beruvchi fieldlar haqida ma’lumot bering.

Ma’ruza №6

Mavzu: Djangoda templateslar va jinja2 formati.

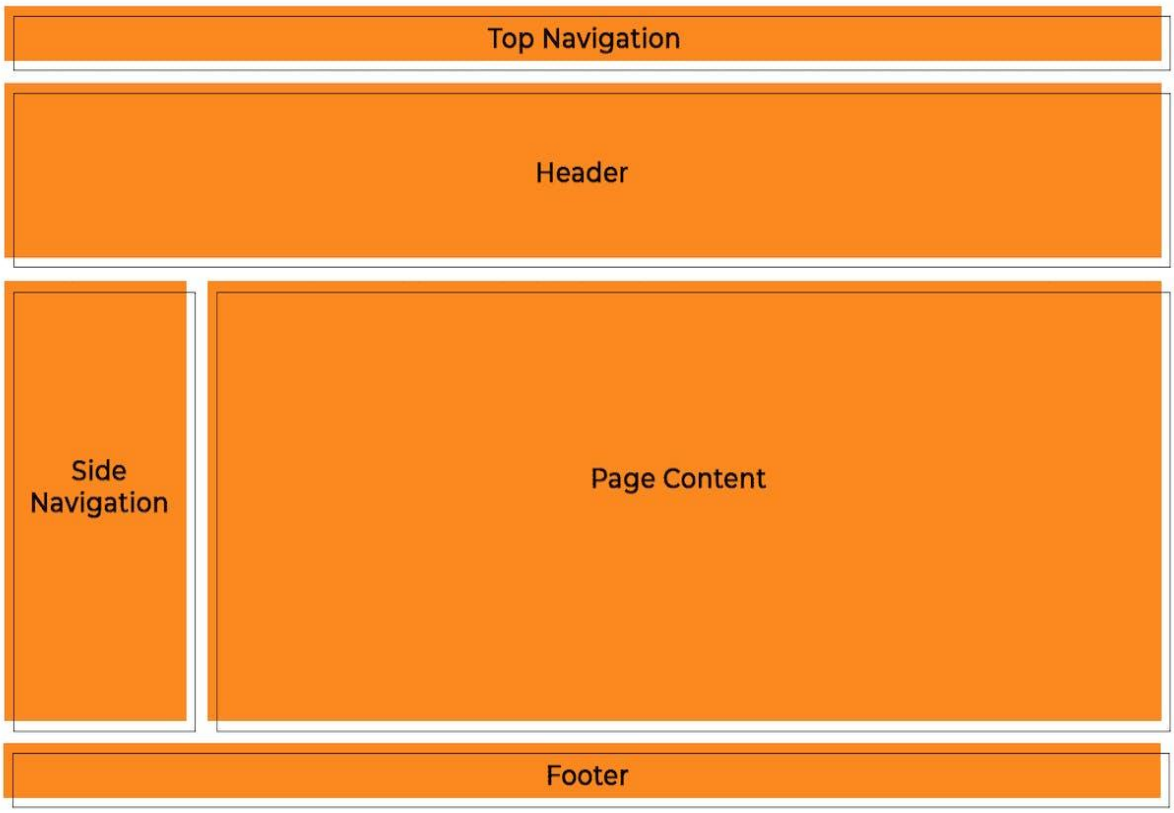
Reja:

1. Template qolib haqida
2. Jinja2 formati
3. Xabar va maqolalar e`lon qilib boruvchi web ilovaning template fayllari

Bizga response qaytarilganda browserga keladigan ma’lumot HTML, CSS va JavaScript fayllari ko‘rinishida bo‘ladi (brauzerdan istalgan web sahifani oching va F12 ni bosib ko‘ring). Bu ma’lumotlar tarkibi shartli ravishda ikki xil bo‘ladi: Tayyor shablon(qolip) va o‘zgarib turadigan ma’lumotlar.

Shablonlar haqida

Django frameworki shablonlar uchun o‘zining kuchli maxsus belgilash tiliga ega. Shablonlar o‘zida html kodlarni jamlagan va ular yordamida ma'lumot taqdim etiladi. Saqlangan fayl statik yoki dinamik bo‘lishi mumkin. Shablonlar o‘zlarida hech qanday biznes logikalarni saqlamaydi. Shunchaki ma'lumotlarni taqdim etadi.



Masalan, youtubeda alohida 2 ta oynada 2 xil video ochib ko‘ring. E’tibor bering, sahifa dizayni, pageda qismlarning joylashgan joyi, rangi, Tepadagi «Youtube» yozuvi bir xil. Bu template(qolip) deyiladi. Lekin, ikkita pageda video, video nomi, tavsiya qilingan videolar va h.k har xil. Bu o‘zgarib turadigan ma’lumotlar.

Har bir ma’lumot uchun alohida HTML kod yoza olmaymiz. Shunchaki, qolib yozib olamiz va kerakli ma’lumotlarni bu qolipdagi kerakli joylarga «yopishtirib» chiqamiz. Bu jarayon rendering deyiladi. Tepada view bizga response qaytarishini bilib oldik. Ana shu response(aniqrog’i HTTP response) esa kerakli ma’lumotlarni tayyor templatega render qilib, keyin yuboriladi. Render jarayonida esa qolibda belgilangan joylarga view tomonidan yuborilgan ma’lumotlar joylashtirilib chiqiladi.

1. Jinja2 formati

Jinja shabloni oddiygina matnli fayldir. Jinja har qanday matnga asoslangan formatni yaratishi mumkin (HTML, XML, CSV, LaTeX va boshqalar). Jinja shablonida ma'lum kengaytma bo‘lishi shart emas: .html, .xml, yoki boshqa kengaytmalarini juda yaxshi qo‘llay oladi.
Shablonda o‘zgaruvchilar yoki ifodalar mavjud bo‘lib, ular shablon ko‘rsatilganda qiymatlar bilan almashtiriladi, va teglar shablon mantiqini boshqaradi. Shablon sintaksisi ko‘p jihatdan Django va Python-dan ilhomlangan.

Quyida standart Jinja konfiguratsiyasidan foydalangan holda bir nechta asoslarni ko‘rsatadigan minimal shablon mavjud. Tafsilotlarni keyinroq ushbu hujjatda ko‘rib chiqamiz:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>My Webpage</title>
</head>
<body>
    <ul id="navigation">
    {% for item in navigation %}
        <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
    {% endfor %}
    </ul>

    <h1>My Webpage</h1>
    {{ a_variable }}

    {# a comment #}
</body>
</html>
```

Quyidagi misol standart konfiguratsiya sozlamalarini ko‘rsatadi. Ilova ishlab chiqaruvchisi sintaksis konfiguratsiyasini shunga o‘xshash narsaga o‘zgartirishi mumkin.{% foo %}.

Bir necha turdagi chegaralovchilar mavjud. Standart Jinja chegaralagichlari quyidagicha tuzilgan:

{% ... %}Bayonotlar uchun ya’ni bu usulda yozilganda qavslar oralig‘idagi ma’lumotlar dasturlash tilida bajariluvchi komandalar sifatida qaraladi.

{{ ... }}ma’lumotlarni chop etishga moslashtirilgan qavslar qachonki ma’lumot ko‘rsatilishini xoxlasangiz ushbu qavslardan foydalanasiz.

{# ... #}andoza chiqishiga kiritilmagan Sharhlar uchun

Jinja formati asosida fayllar tayyorlar ekanmiz yuqorida ko‘rsatilgan qavslar asosida python kodlarini yoza veramiz render vaqtida qavslar ichida yozilgan python kodlari ishga tushuriladi va shunda fayl ichida python kodlarini ishga tushurish uchun bir muammoli vaziyat yuzaga keladi bu python sikli va shartga asoslangan dastur qismlarini tab masofa orqali boshqarishidir jinja formatida buning uchun yechim qilingan bo‘lib sikl tugagan joyni belgilash lozim

masalan for operatordan foydalansak {% for I in array %}element-1 element-2 … element-n {% endfor %} bunday vaziyatlarga quyida birqancha misollar keltirdik.
{% if a = = 0 %} teg-1 teg-2 … teg-n {% endif %}

Asosiy shablon

Base deb ataydigan ushbu shablon base.html oddiy ikki ustunli sahifa uchun foydalanishingiz mumkin bo‘lgan oddiy HTML skelet hujjatini belgilaydi. Bo‘sh bloklarni tarkib bilan to‘ldirish "qism" shablonlarining vazifasidir. Base.html go‘yoki daftar yuzini himoyalovchi muqova kabidir har safar uning ichiga turli daftarlarni qoyish mumkin bo‘lgani kabi base.html ni ham boshqa bir fayllar bilan to‘ldirish mumkin.

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% block head %}
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}{% endblock %} - My Webpage</title>
    {% endblock %}
</head>
<body>
    <div id="content">{% block content %}{% endblock %}</div>
    <div id="footer">
        {% block footer %}
        &copy; Copyright 2008 by <a href="http://domain.invalid/">you</a>.
        {% endblock %}
    </div>
</body>
</html>
```

Bu misolda teglarni qism shablonlari to‘ldirishi mumkin bo‘lgan to‘rtta blokni belgilaydi. Barcha **blok** tegi shablon mexanizmiga shablondagi o‘sha joy egalarini bekor qilishi mumkinligini aytadi.{% block %}

Block teglar boshqa bloklar ichida bo‘lishi mumkin, lekin ular blok haqiqatda ko‘rsatilganmi yoki yo‘qmi, qat'iy nazar har doim bajariladi.

Qism uchun shablon

Qism shablonlari quyidagicha ko‘rinishi mumkin:

```
{% extends "base.html" %}
{% block title %}Index{% endblock %}
{% block head %}
    {{ super() }}
    <style type="text/css">
        .important { color: #336699; }
    </style>
{% endblock %}
{% block content %}
    <h1>Index</h1>
    <p class="important">
      Welcome to my awesome homepage.
    </p>
{% endblock %}
```

Bu yerda teg kalit hisoblanadi. Bu shablon mexanizmiga ushbu shablon boshqa shablonni "kengaytirishi" haqida xabar beradi, ya’ni qism shablonimiz base.html ni to‘ldiradi. {% extends %} komandasi orqali qaysi faylni to‘ldirishga ixtisoslashgan qism fayl ekanligi aytiladi. (yuqoridagi suratning 1-qatoriga qarang) shundan so‘ng blocklarga berilgan nomlar asosida qismlar aniqlanadi va mos o‘rinlarga qismlar ko‘chiriladi. Bunda base.html ichidagi {% block title %} bo‘lagi qism faylidagi {% block title %} qismi bilan almashtiriladi shu tariqa base.html da joylashgan barcha qismlar qism faylidagi qismlar bilan mos ravishga almashtiriladi.

2. Xabar va maqolalar e`lon qilib boruvchi web ilovaning template fayllari

Ilgarigi darslarimizda xabar va maqolalar e`lon qilib boruvchi web ilovaning modelini tayyorlab olgan edik endilikda shu proyektimiz uchun template fayllar yaratamiz biz buning uchun html fayllardan foydalanamiz va ularda jinja formatlarni qo‘llaymiz.

base.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link href="{% static 'abc/main.css' %}" rel="stylesheet" type="text/css">
   <title>Document</title>
  </head>
<body>
   {% block content %}
   {% endblock content %}
</body>
</html>
```

Base.html faylimiz asosiy fayl hisoblanadi qolgan barcha fayllar unga farzand sifatida yaratiladi. {% load static %} qatori template fayllariga yuboriluvchi static fayllari bilan bog‘liqlikni hosil qilishga mo‘ljallangan.

Index.html

```
{% extends "base.html" %}
<h1>list</h1>
{% block content %}
{% for post in posts %}
<a href="{% url 'maqola:detail' post.id %}" >{{post.title}}</a><div></div>
<div ><span> author {{post.author}} created date {{post.created_at}}</span></div>
<img class="list" src="{{post.img.url}}" alt="{{post.title}}">
<p >{{post.content | slice:300 }} ...</p>
{% endfor %}
{% endblock content %}
```

Base.html va index.html fayllari birlashgan xolda bosh sahifani hosil qiladi. |slice:300 komandasining vazifasi belgilar soni 300 tadan ko‘p bo‘lsa faqa boshlang‘ich 300 tasini chiqarish buyrug‘i bu bilan maqolaning qisqacha bayonini tashkil etamiz.

Detail.html

```
{% extends "base.html" %}
<h1>detail</h1>
{% block content %}
<h3>{{posts.title}}</h3>
```

```
<div><span>author {{posts.author}} created date {{posts.created_at}} </span> </div>
<img class="list" src="{{posts.img.url}}" alt="{{posts.title}}">
<p class="ttt">{{posts.content}}</p>

{% endblock content %}
```

Base.html va detail.html fayllari birlashgan xolda tanlangan bir maqolaning to‘liq bayonini hosil qiladi.

Nazorat savollari

1. Qolib tushunchasini izohlang.
2. Jinja2 formati haqida gapirib bering.
3. Jinja formatida necha xil qavs bor.
4. Jinja formatida sikllar hosil qiling.
5. Jinja formatida shart operatoridan foydalaning.
6. Jinja formati orqali fayllarni birlashtiring.
7. Jinja formati static fayllar bilan bog‘liqlik qanday taminlanadi.
8. Xabar va maqolalar e`lon qilib boruvchi web ilovaning template fayllarida qaysi bir standart jinja chegaralagichlaridan foydalanildi?

Ma’ruza №7

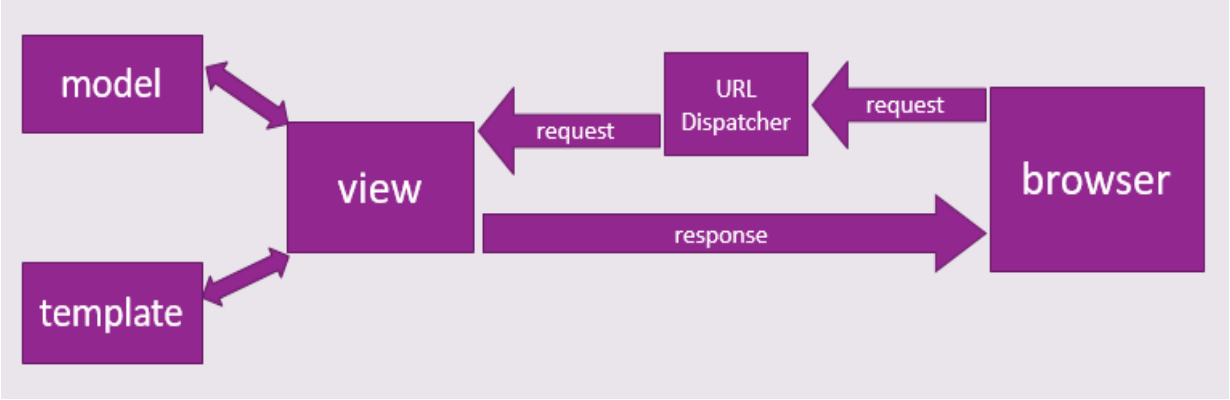
Mavzu: Djangoda Viewslar bilan ishlash.

Reja:

1. View haqida tushuncha
2. Djangoning View parent sinfi
3. Xabar va maqolalar e`lon qilib boruvchi web ilovaning Views fayllari

1. View haqida tushuncha

Nomlanishi «ko‘rinish» ma’nosini bersa ham, aslida, views ko‘rinishga javob beradigan qism emas. Views django loyihaning «miyasi» sifatida ishlaydi. Mantiqiy qismlar shu yerda yoziladi. Qolgan deyarli barcha qismlar shu qism orqali biriktiriladi va view ularni nazorat qiladi. Serverga kelgan request(so‘rov)larga view javob beradi. Masalan, djangoda yozilgan saytga browser orqali «<saytnomi>/login/» qismiga request yuborsak, url dispatcher(bu haqida pastroqda) requestni tutib olib, «login/» qismi uchun javobgar viewga yuboradi. View esa requestni analiz qilib, kerakli vazifalarni bajaradi va bizga response(javob) yuboradi.



Djangoning View nomli parent sinfi mavjud bo‘lib unga voris sinflar hosil qilish orqali turli xil view classlar yarata olamiz. Ularda request methodlar nomiga asoslangan (Masalan:GET, POST) methodlar tayyorlab shu orqali foydalanuvchilarga turli ma’lumotlar uzatishimiz va turli ma’lumotlarni qabul qilib olishimiz qayta ishlashimiz o‘chirishimiz mumkin.

GET method asosan malumotlarni olishga qaratiladi va u orqali tahrirlash, yaratish, o‘chirish kabi ishlarni amalga oshirilmaydi.

Browserning izlash qatoriga turli xil url mazillar orqali murojaatlar qilganimizda browser get methoddan foydalanadi.

POST method ma’lumotlar ustida tahrirlash, o‘zgartirish, yangi yaratish, o‘chirish kabi ishlarni olib boruvchi method hisoblanib bunday so‘rovlarda url mazilning o‘zi yetarli emas hafsizlikni taminlash uchun tokin, session kabi hafsizlik kalitlari ham jo‘natilishi lozim va shu orqali tizim hafsizligi taminlanadi.

Request methodlari ro‘yxatiga etibor beradigan bo‘lsak ular 5ta bo‘lib (get, post, put, patch, delete) faqat get ochiq so‘rovdir va qolgan barcha methodlar hafsizlik maqsadida yopiq so‘rovlardir.

2. Djangoning View parent sinfi

Djangoning views kutubxonasida joylashgan View parent sinfiga voris sifatida yaratilgan sinflar MVT ning views qismiga javob beruvchilar bo‘lib hisoblanadi va ular birqancha bo‘lishi mumkin. URl Dispatcher esa so‘rov mazmuniga ko‘ra ularning qaybiriga yo‘naltirishni hal qiladi.

```
from django.urls import path

from myapp.views import MyView

urlpatterns = [
    path('mine/', MyView.as_view(), name='my-view'),
]
```

Yuqorida urls.py fayli keltirilgan bo‘lib browserdan kelgan ‘mine/’ so‘rovigagina MyView nomli view classni chaqiradi.

```
from django.http import HttpResponse
from django.views import View

class MyView(View):

    def get(self, request, *args, **kwargs):
        return HttpResponse('Hello, World!')
```

Yuqorida views.py fayli keltirilmoqda va unda get method inobatga olingan bo‘lib url manzilga get method orqali so‘rov yuborilgan bo‘lsagina browserga ‘Hello, World!’ matni jo‘natiladi. Yuqorida keltirilgan urls.py fayliga etibor bersak MyView sinfidan

as_view() methodi chaqirilgan ammo views.py faylidagi MyView sinfi tuzulishiga qarasak as_view() methodi yozilmaganini ko‘ramiz demak as_view() methodi parent sinfda yani View sinfida joylashgan bo‘ladi.

3. Xabar va maqolalar e`lon qilib boruvchi web ilovaning Views fayllari

Oldingi darslarimizda tayyorlab kelayotgan Xabar va maqolalar e`lon qilib boruvchi web ilovaning Views fayllarini tayyorlab oladigan bo‘lsak. U yerda ikki xil xolat mavjud edi va ular bosh sahifa (Templatelarda index.html) va aynan bir maqolaning to‘liq bayoni beriluvchi yana bir sahifadan (Templatelarda detail.html) iborat demak har ikkalasi turli vaziyatlarda ishlaganligi sababli har biriga alohida View classlar tayyorlaymiz.

```
from django.shortcuts import render
from .models import Post
from django.views import View

class PostView(View):
    def get(self, request):
        posts = Post.objects.all()
            return render(request, 'maqola/index.html', {'posts':
posts})

class PostDetailView(View):
    def get(self, request, id):
        post = Post.objects.get(id=id)
            return render(request, 'maqola/detail.html', {'posts':
post})
```

Yuqorida views.py faylimiz ketirilgan bo‘lib endilikda uning faoliyatida model va templatelarning o‘zaro birlashishiga guvoh bo‘lamiz. Yuqoridagi uch qatorda foydalaniladigan fayllar chaqirib olingan va ular endilikda shu views.py faylimizda ishlatiladi.

Render funksiyasining vazifasi murojaat bo‘lgan foydalanuvchiga aytilayotgan malumotlarni yo‘naltirishdan iborat.

Post modelining chaqirilishi ushbu madel bilan to‘g‘ridan to‘g‘ri ishlash imkonini beradi.

PostView classning get methodidagi **posts = Post.objects.all()** buyrug‘i post modeliga asoslanib ma’lumotlar omboridagi post jadvalining barcha malumotlarini olishni anglatadi va posts o‘zgaruvchisida bu ma’lumotlarni jamlaydi. Shundan so‘ng templatega joylashtirish jinja2 formati orqali amalga oshiriladi.

Foydalanuvchiga bosh sahifa ko‘rsatiladi va unda maqola nomi, qisqacha bayoni, surati, yozilgan vaqti kabi malumotlar ko‘rsatilib o‘tiladi. Foydalanuvchi maqola nomida joylashgan linkni bosish orqali serverga ushbu maqolaning to‘liq bayoni kerakligi haqida so‘rov jo‘natadi va shunda PostDetailView classti ishga tushib ketadi undagi **post = Post.objects.get(id=id)** qatori post modeliga asoslanggan ma’lumotlar jadvalidan idsi (takrorlanmas tartib raqami) aytilayotgan maqola idsiga tengini ajratib oladi. Post o‘zgaruvhisiga yuklaydi va rebder da templatega qoliblanadi.

Nazorat savollari

1. MVT ning View tushunchasi haqida gapirib bering.
2. Views ning asosiy vazifasi nima?
3. Request methodlar haqida gapirib bering.
4. Ochiq so‘rovlar qaysi request method yordamida yoboriladi?
5. View parent classning vazifasi.
6. as_view() methodi haqida gapiring.
7. Xabar va maqolalar e`lon qilib boruvchi web ilovaning Views fayllaridagi dastur kodlarini tariflang.
8. Post modelidagi ma’lumotlarni olish uchun qanday komandalar bor?

Ma’ruza №8

Mavzu: Djangoda settings sozlamalari va urllar.

Reja:

1. Settings.py fayli sozlamalari
2. Urls.py fayllari (URL Dispatcher)
3. Xabar va maqolalar e`lon qilib boruvchi web ilovaning sittings.py sozlamalari va url manzillari

1. Settings.py fayli sozlamalari

Djangoda qurilayotgan proyektda maxsus sozlamalar uchun ajratilgan fayl mavjud va u settings deb nomlangan unda yaratilgan applarni ro‘yxatga kiritish media fayllar uchun sozlamalar, static fayllar uchun sozlamalar, electron pochta bilan bog‘lanish sozlamalari, vaqt mintaqalarini sozlash va boshqa birqancha sozlamalarni tayyorlash mumkin.

BASE_DIR = Path(__file__).resolve(strict=True).parent.parent

BASE_DIR ning vazifasi katta bo‘lib u fayllar va papkalarni serverdan ro‘yxatdan o‘tkazishga yordam beradi. Siz python manage.py runserver komandasini berganingizda projectingiz serverda turgandek ishlaydi. Serverdan ro‘yxatdan o‘tkazilmagan fayl va papkalarni django ishlata olmaydi (bunday fayl mavjud emas mazmunida xatolik qaytariladi).

SECRET_KEY = 'i8cw-a79ns1xp8dz)&y_ei=%=u(zu3^tqmsf)qwz9^n-!hqj8j'

SECRET_KEY ning vazifasi shundaki u prodakshinga yordam beradi va buni boshqalar bilishi taqiqlanadi.

DEBUG = True

DEBUG ning vazifasi shundaki agar u True qiymat qabul qilsa sizga xatolaringizni ko‘rsatib turadi(brouzer va CMD da) agar False bo‘lsa siz biror bir xato ish qilsangiz serverni o‘sha ondayoq to‘xtaydi(siz yana runserver qilishingizga to‘g'ri keladi). Agar siz projectni ohirgacha bitirib hostga qo‘ymaguningizgacha True tursin.

ALLOWED_HOSTS = []

ALLOWED_HOSTS ning vazifasi shundaki [] qavslar ichiga domen yozasiz va brouzerdan o‘sha domen yozilganda sizning saytingiz chiqadi (agar siz hostga qo‘ygan bo‘lsangiz va host o‘sha domenga ulangan bo‘lsa) agarda boshqa tizimlardan so‘rovlar kelsa tizim javob qaytarmaydi.

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

INSTALLED_APPS ning vazifasi django applarini ishlatish(uyg'otish). Bu mavzuga yana qaytamiz...

```
ROOT_URLCONF = 'library.urls'
```

ROOT_URLCONF ning vazifasi django projecti birinchi bo‘lib qaysi bir URL larga asosan ishlashini belgilaydi.

```
TEMPLATES = [
  {
    'BACKEND':
'django.template.backends.django.DjangoTemplates',
    'DIRS': [],
    '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',
      ],
    },
  },
]
```

TEMPLATES ning vazifasi html fayllarni qaysi papkadaligini serverdan ro‘yxatga o‘tkazadi. (kelasi mavzularda ishlatamiz)

```
WSGI_APPLICATION = 'library.wsgi.application'
```

WSGI_APPLICATION ning vazifasi get_wsgi_application() funksiyani uyg'otish (ishga tushirish).

```
DATABASES = {
  'default': {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': BASE_DIR / 'db.sqlite3',
```

```
    }
}
```

DATABASES ning vazifasi ma'lumotlarni saqlash bo‘lib bunda qanday nomli faylga va nima orqali o‘sha faylga tasir o‘tkazishni ko‘rsatadi. (postgresql, sql, va Mysql ga o‘xshash malumotlar omborini ulaydi).

LANGUAGE_CODE = 'en-en'

LANGUAGE_CODE ning vazifasi projectning tilini aniqlab beradi. Siz en-en desangiz ingiliz tilini tanlagan bo‘lasiz (ikkinchi en esa buyukbiritaniya shevasi). Xatto-ki o‘zbek tili ham mavjud 'uz'. Tilni o‘zgartirganingizda admin page va debug pagelar o‘zgarganini ko‘rasiz (hali beri admin pageni ko‘rmagan bo‘lsangiz keyinchalik o‘zgartirib koring)

TIME_ZONE = 'UTC'

TIME_ZONE ning vazifasi serverdagi vaqtni aniqlash. UTC=> "Universal Time Coordinated" bo‘lib biz(uzb)ning vaqti 'UTC+5'.

STATIC_URL = '/static/'

Bu static fayllarni serverdan ro‘yxatdan o‘tkazadi. Static fayllarga html fayllardan boshqa fayllar kiradi(CSS, JS, Rasmlar...).

Bulardan tashqari loyiha darajasidagi sozlamalar shu qismda saqlanadi va siz ham qandaydir loyiga darajasida boshqaruv olib borsangiz shu yerda kodlarini yozasiz.

2. Urls.py fayllari (URL Dispatcher)

Djangoda yangi proyekt yaratilganda uning ichida asosiy urls.py fayli mavjud bo‘ladi va unda aftomatik tarzda djangoning standart admin paneliga yo‘naltirish bilanhosil bo‘ladi. Applar ichida hosil qilinuvchi barcha urls.py fayllari yuqoridagi asosiy urls fayli bilan bog‘lanadi va bu include funksiyasi orqali amalga oshiriladi.

```
"""library URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
   https://docs.djangoproject.com/en/3.1/topics/http/urls/
Examples:
Function views
   1. Add an import:  from my_app import views
   2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
   1. Add an import:  from other_app.views import Home
   2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
   1. Import the include() function: from django.urls import include, path
   2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
   path('admin/', admin.site.urls),
   path('', include('lib_app.urls', namespace='app'))
]
```

Yuqorida proyektimizning asosiy urls.py fayli ko‘rsatilgan bo‘lib unda belgilangan qatorda ‘lib_app’ nomli appning ichida joylashgan urls.py fayliga include orqali yo‘naltirilganligiga guvoh bo‘lamiz. Appning ichidagi urls.py faylida esa yo‘naltirish viewlarga asosan amalga oshadi.

```
from django.urls import path
from . import views
app_name = 'lib_app'
urlpatterns = [
path('', views.home, name='home')
]
```

Appda yaratilgan qo‘shimcha urls.py fayli yuborilgan so‘rov asosida kerakli viewlarni ishga tushiradi

3. Xabar va maqolalar e`lon qilib boruvchi web ilovaning sittings.py sozlamalari va url manzillari

Oldingi darslarimizdan ko‘rib kelayotgan proyektimiz Xabar va maqolalar e`lon qilib boruvchi web ilovaning settings.py fayli quyidagicha ko‘rinishda

```
""""
```

```
Django settings for config project.

Generated by 'django-admin startproject' using Django 3.2.8.

For more information on this file, see
https://docs.djangoproject.com/en/3.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.2/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See
https://docs.djangoproject.com/en/3.2/howto/deployment/checklist
/

# SECURITY WARNING: keep the secret key used in production
secret!
SECRET_KEY = 'django-insecure-ybi(s%mq$tab-
kwo1mci9pi)u)nzed3nzu9r0=y!oii96t@!p5'

# SECURITY WARNING: don't run with debug turned on in
production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
```

```
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'maqola',
    'user',
]
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
ROOT_URLCONF = 'config.urls'
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / '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',
            ],
        },
    },
]
WSGI_APPLICATION = 'config.wsgi.application'
# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases
DATABASES = {
```

```
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
# Password validation
# https://docs.djangoproject.com/en/3.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
# Internationalization
# https://docs.djangoproject.com/en/3.2/topics/i18n/
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = True
```

```
MEDIA_URL = '/media/'
MEDIA_ROOT = 'file_img'

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.2/howto/static-files/

STATIC_URL = '/static/'
STATICFILES_DIRS = [BASE_DIR / 'static']
# Default primary key field type
# https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Yuqorida belgilangan qatorlar o‘zgatirish kiritilgan qatorlar birinchi bo‘lib app e’lon qilingan qatorni uchratasiz keyin esa template fayllar saqlanuvchi papka elon qilingan qatorni uchratasiz pastroqda esa media fayllar sozlamalarini va so‘ngida static fayllar sozlamalarini ko‘rasiz

Urls.py fayllari proyektimizda quyidagicha shakllanadi.

```
from django.contrib import admin
from django.urls import path, include
from django.conf.urls.static import static
from django.conf import settings

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('maqola.urls')),
]

urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

Yuqorida proyektimizning asosiy urls.py fayli unda so‘ngi qator to‘g‘ridan to‘g‘ri settings.py faylidan medialar haqidagi malumotlar bilan ishlayotganligini ko‘rasiz.

```
from django.urls import path
from .views import PostView, PostDetailView
```

```
app_name = 'maqola'
urlpatterns = [
    path('', PostView.as_view(), name='index'),
    path('detail/<int:id>/',                PostDetailView.as_view(),
name='detail'),
]
```

Ushbu dastur kodi maqola nomli appning ichidagi urls.py faylida joylashgan va u ikkita oldingi darslarda tayyorlagan viewlarimizga yo‘naltirmoqda

Nazorat savollari

1. Templates papkasi qanday sozlanadi?
2. Loyihaning app lari qayerda ro‘yxatga qo‘shilishi lozim?
3. Templates papkasi qayerda ro‘yxatdan o‘tkaziladi?
4. Loyiha uchun vaqt hududi qanday tuzatiladi?
5. Static fayllar uchun urls.py faylida qanday manzil sozlanadi?
6. Asosiy urls.py faylida qanday ma’lumot aftomatik yaratilgan bo‘ladi?
7. applarning urls.py fayllariga yo‘naltirishda qaysi funksiyadan foydalaniladi?
8. Xabar va maqolalar e`lon qilib boruvchi web ilovani ishga tushiring

Ma’ruza №9
Mavzu: Djangoda standart User modeli va uni loyhaga qayta moslash.

Reja:

1. User modeli haqida
2. Loyihaga mos foydalanuvchilarga model tayyorlash
3. Settings.py faylida User modeli o‘rniga yangi modelni sozlash

1. User modeli haqida

User modeli django proyekt bilan birgalikda hosil bo‘ladi uning parametrlari ko‘p emas ular quyidagilar

Username
FirstName
LastName
Email
Password

Yuqoridagi ma’lumotlar foydalanuvchining minimum ma’lumotlari hisoblanadi va shu orqali foydalanuvchilarni ro‘yxatga oladi va xizmat ko‘rsatadi. Ammo bu minimum ma’lumotlar ko‘plab tizimlarda yetarli bo‘lmaydi va ularga qo‘shimcha parametrlarga ehtiyoj tug‘iladi.

User modeli tuzilishiga qaraydigan bo‘lsak

```
class User(AbstractUser):
    """
    Users within the Django authentication system are represented by this
    model.

    Username and password are required. Other fields are optional.
    """
    class Meta(AbstractUser.Meta):
        swappable = 'AUTH_USER_MODEL'
```

User modelining tuzulishiga etibor bersak ma’lumot ko‘p emasligini u ham voris sinf ekanligini ko‘ramiz. Userning parenti AbstractUser ekanligini va barcha parametrlar unda joylashganini ko‘ramiz. Shu sababli AbstractUser sinfini ko‘zdan kechiramiz.

```
class AbstractUser(AbstractBaseUser, PermissionsMixin):
    """
```

```
    An abstract base class implementing a fully featured User
model with
    admin-compliant permissions.
    Username and password are required. Other fields are optional.
    """
    username_validator = UnicodeUsernameValidator()

    username = models.CharField(
        _('username'),
        max_length=150,
        unique=True,
        help_text=_('Required. 150 characters or fewer. Letters,
digits and @/./+/-/_ only.'),
        validators=[username_validator],
        error_messages={
            'unique': _("A user with that username already exists."),
        },
    )
    first_name = models.CharField(_('first name'),
max_length=150, blank=True)
    last_name = models.CharField(_('last name'), max_length=150,
blank=True)
    email = models.EmailField(_('email address'), blank=True)
    is_staff = models.BooleanField(
        _('staff status'),
        default=False,
        help_text=_('Designates whether the user can log into this
admin site.'),
    )
    is_active = models.BooleanField(
        _('active'),
        default=True,
        help_text=_(
            'Designates whether this user should be treated as active. '
            'Unselect this instead of deleting accounts.'
        ),
    )
```

```
    date_joined = models.DateTimeField(_('date joined'),
default=timezone.now)

    objects = UserManager()

    EMAIL_FIELD = 'email'
    USERNAME_FIELD = 'username'
    REQUIRED_FIELDS = ['email']

    class Meta:
        verbose_name = _('user')
        verbose_name_plural = _('users')
        abstract = True

    def clean(self):
        super().clean()
        self.email = self.__class__.objects.normalize_email(self.email)

    def get_full_name(self):
        """
        Return the first_name plus the last_name, with a space in
between.
        """
        full_name = '%s %s' % (self.first_name, self.last_name)
        return full_name.strip()

    def get_short_name(self):
        """Return the short name for the user."""
        return self.first_name

    def email_user(self, subject, message, from_email=None,
**kwargs):
        """Send an email to this user."""
        send_mail(subject, message, from_email, [self.email],
**kwargs)
```

AbstractUserda aslida User sinfi boryog‘I 5 ta atributdan iborat emasligini va ular birqancha va foydalanuvchilar harakatining mazmunan huquqiy asoslarini shaklantirilganini ko‘ramiz. Bulardan

Username unikal bo‘lishi to‘ldirish shart o‘zida 150 tagacha belgi sig‘dira olishi.
first_name foydalanuvchi ismini kiritish uchun atribut kiritmaslik ham mumkin.
Last_name foydalanuvchi familyasi uchun atribut bo‘sh bo‘lishi mumkin.
Email electron pochta uchun atribut bo‘sh bo‘lishi mumkin.
Is_staff mantiqiy atributi foydalanuvchining admin panelga kirishga huquqi bor yoki yoqligini belgilaydi.
Is_active foydalanuvchi akkaunti aktiv yoki aktiv emasligini anglatuvchi mantiqiy atribut agar aktiv bo‘lmasa tizimdan foydalana olmaydi.
Is_super foydalanuvchi tizim adminstratori ekanligiga aniqlik kirituvchi mantiqiy atribut.
Password foydalanuvchi parolini saqlovchi atribut u bilan birga ikta method ham qo‘shilib ishlaydi bular parolni shifrlovchi va tekshirish uchun taqdim etilayotgan variantni mosligini tekshuvchi methodlardir.
Yuqoridagi kodlardan o‘z tizimimiz uchun foydalanuvchilarni ro‘yxatga oluvchi va ularga xizmat ko‘rsatuvchi yangi sinf yaratishda merosxo‘rlik asosida foydalanamiz.

2.Loyihaga mos foydalanuvchilarga model tayyorlash

Ko‘rilayotgan loyihada foydalanuvchilarning qanday ma`lumotlarini kiritishga qarab User model parametrlari yetarli bo‘lmay qolganda foydalanuvchilar ma`lumotlarini saqlovchi yangi model tayyorlashga ehtiyoj paydo bo‘ladi. Modelning atributlarini loyihamiz asosida joylashtiramiz .

```
from django.contrib.auth.models import AbstractUser
from django.db import models

# Create your models here.

class CustomUser(AbstractUser):
    age = models.PositiveIntegerField(null=True,blank=True)
```

djangoning User modeli o‘rniga uning vazifalarini bajaruvchi model tayyorlar ekanmiz u qaysi sinfga voris bo‘lgan bo‘lsa biz ham xuddi shu sinfga voris sinf tayyorlaymiz va bu orqali ko‘plab xizmatlarni merosxo‘rlik asosida o‘zlashtiramiz. Yuqoridagi CustomUser sinfini

AbstractUserga voris qildik va bitta age nomli atribut qatorini yozdik bu bilan foydalanuvchi yoshini ham kiritish imkoniga ega bo‘lamiz.

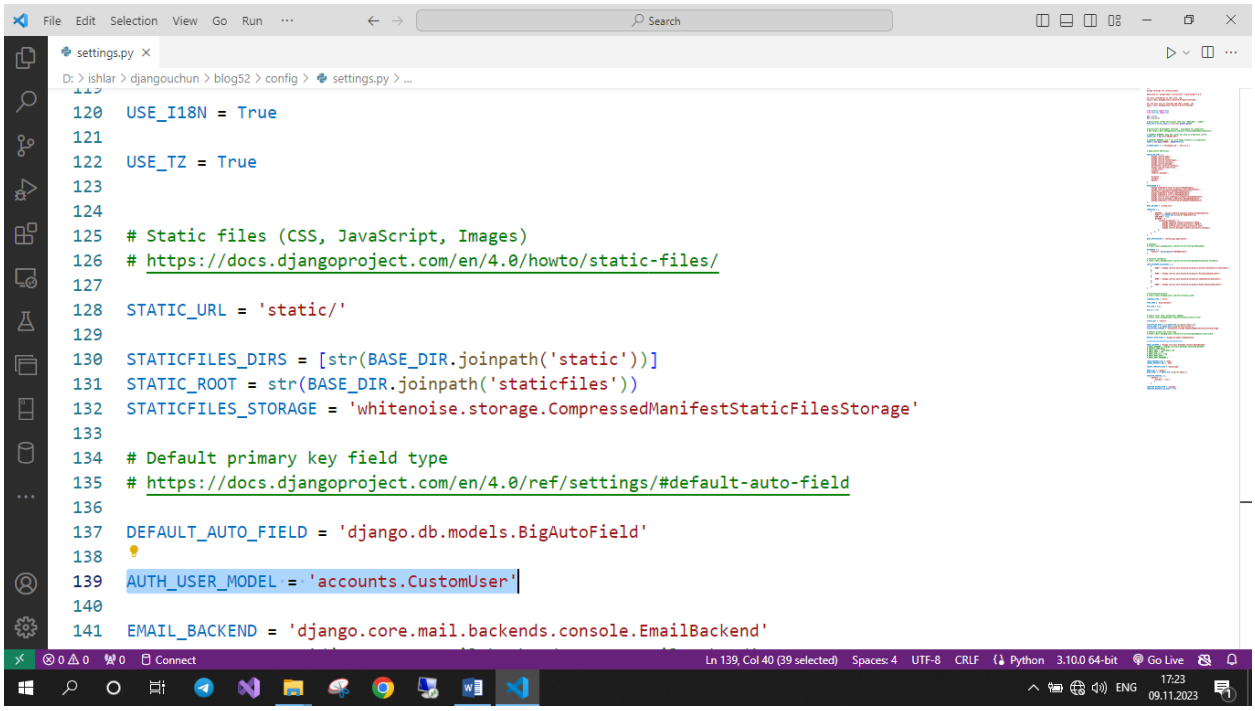
3. Settings.py faylida User modeli o‘rniga yangi modelni sozlash

Oldingi darslarimizda aytganimizdek loyiga darajasidagi sozlamalarni settings.py faylida amalga oshiramiz.

Foydalanuvchilarga xizmat ko‘rsatuvchi User sinfini quyidagicha o‘zgartiramiz:

settings.py faylining so‘ngi qatoriga tushamiz va quydagi dastur kodini yozamiz.

AUTH_USER_MODEL = 'accounts.CustomUser'



Nazorat savollari

1. User modelining parent classti nomini ayting
2. Modelda surat ustuniga qanday fild ishlatiladi?
3. Settings.py faylida user model o‘rniga tayyorlangan model qanday sozlanadi?

Ma’ruza №10

Mavzu: Djangoda Formlar Tayyorlash.

Reja:

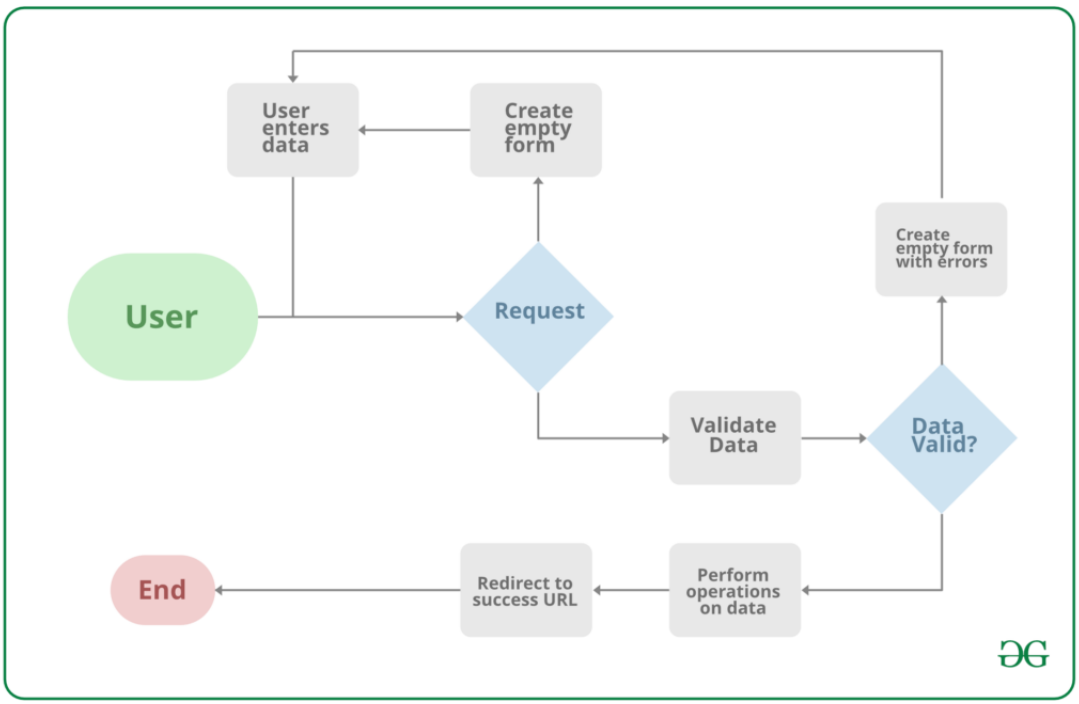
1. Form haqida
2. Django formasini yaratish
3. Modellardan Django formasini yaratish

1. Form haqida

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



10.1-rasm.

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

2. **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):
    field_name = forms.Field(**options)
from django import forms
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.

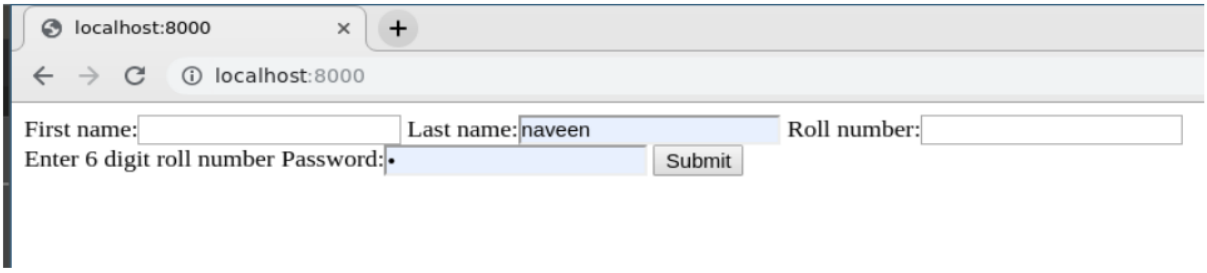
- {{ form.as_table }} ularni <tr> teglari bilan o‘ralgan jadval kataklari sifatida ko‘rsatadi
- {{ form.as_p }} ularni <p> teglariga o‘ralgan holda ko‘rsatadi
- {{ form.as_ul }} ularni <li> teglariga o‘ralgan holda ko‘rsatadi

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
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>
```



10.2-rasm.

3. 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.

```
from django.db import models
class GeeksModel(models.Model):
        title = models.CharField(max_length = 200)
        description = models.TextField()
        last_modified = models.DateTimeField(auto_now_add = True)
        img = models.ImageField(upload_to = "images/")
        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:

```
from django import forms
from .models import GeeksModel
class GeeksForm(forms.ModelForm):
        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?

Ma’ruza №11

Mavzu: Djangoda loyiha qurish va registration, login, logout.

Reja:

1. Foydalanuvchilarni ro‘yxatga olish
2. Foydalanuvchi logini
3. Foydalanuvchi tizimdan chiqish

1. Foydalanuvchilarni ro‘yxatga olish

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
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.

2. 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 .

3. 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?

Ma’ruza №12

Mavzu: Djangoda loyihani testlash.

Reja:

1. Sinov turlari
2. Sozlash va o‘rnatish
3. Sinov modellari

1. 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.

2. Sozlash va o‘rnatish.

Modullarni o‘rnating va uni INSTALLED_APPS ilovangizga qo‘shing:

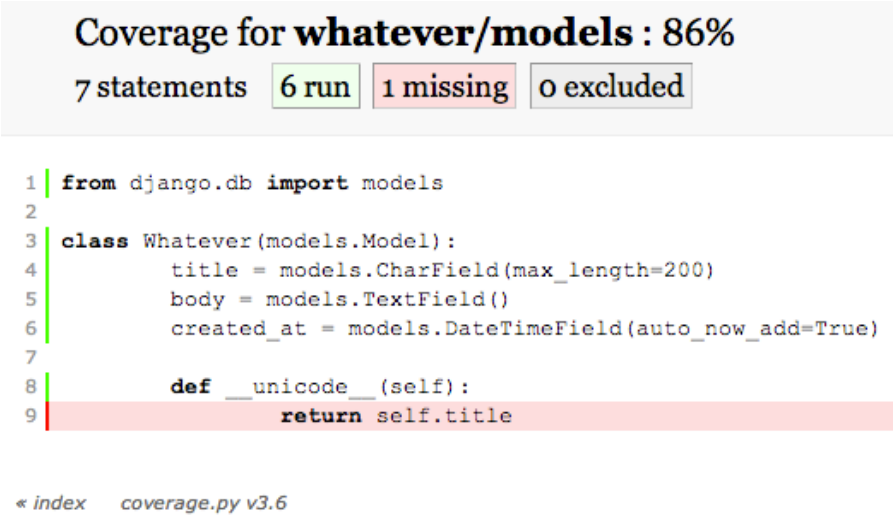
```
$ pip install coverage==3.6
```

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.

3. Sinov modellari.

Qamrov hisobotida "nima bo‘lishidan qat'iy nazar/modellar" uchun havolani bosing. Siz ushbu ekranni ko‘rishingiz kerak:



12.1-ram.

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
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:

12.2-rasm.

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:

```
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?

Ma’ruza №13

Mavzu: Djangoda tayyorlangan modellar uchun admin panelda search va filter funksiyalarini tayyorlash.

Reja:

1. Search va filter funksiyalari haqida
2. Filterlash jarayoni

1. search va filter funksiyalari haqida

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.

2. Filterlash jarayoni

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?

Ma’ruza №14

Mavzu: Djangoda mixins va messages kutubxonalari.

Reja:

1. Xabarlar doirasi
2. Xabar mexanizmini sozlash
3. Xabar darajalari

1. 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.context_ processors.messages'

Agar siz xabarlardan foydalanishni istamasangiz, 'django.contrib.messages' o‘zingizning dan INSTALLED_APPS, MessageMiddleware qatorni dan MIDDLEWARE va messages kontekst protsessorini dan olib tashlashingiz mumkin. TEMPLATES.

2. 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.

3. 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:

14.1-jadval.

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 foydalanuv-

chilarning 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
messages.set_level(request, messages.DEBUG)
messages.debug(request, "Test message...")
messages.set_level(request, messages.WARNING)
messages.success(request, "Your profile was updated.")
messages.warning(request, "Your account is about to expire.")
#recorded
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 Composed View (Access Mixin, Permission Required Mixin, 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. **Login Required Mixin, User Passes Test Mixin, PermissionRequiredMixin**

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?

Ma’ruza №15

Mavzu: Djangoda generic kutubxonasi.

Reja:

1. Generic kutubxonasi haqida
2. Tafsilotlarni ko‘rish shablonini yaratish
3. Shablonlar

1. Generic kutubxonasi haqida

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'
    queryset = Book.objects.filter(title__icontains='war')[:5]
    template_name = 'books/my_arbitrary_template_name_list.html'
```

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_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):
        context = super(BookListView, self).get_context_data(**kwargs)
        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:

15.1-jadval.

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:

15.2-jadval.

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})
```

2.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

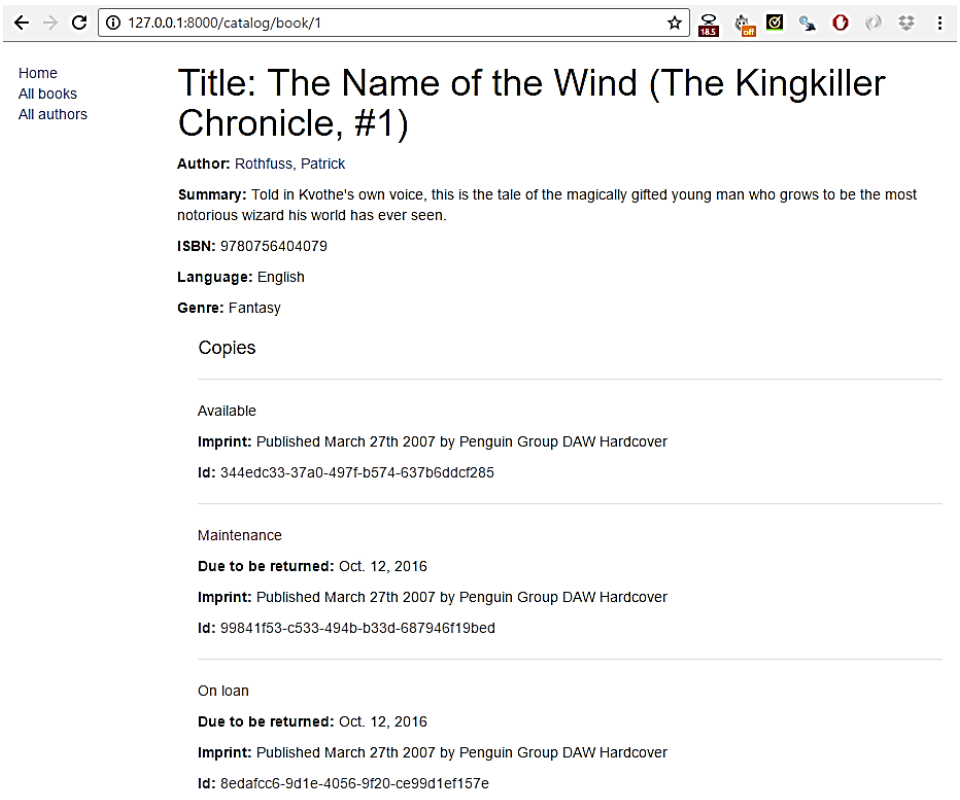
Shu nuqtada, biz kitoblar ro‘yxatini va kitob tafsilotlari sahifalarini ko‘rsatish uchun kerak bo‘lgan hamma narsani yaratishimiz kerak edi. Serverni ishga tushiring (python3 manage.py runserver) va brauzeringizni oching http://127.0.0.1:8000/.

Ogohlantirish: Muallif yoki muallif tafsilotlari havolalarini hali bosmang – ularni tanlovda yaratasiz!

Kitoblar ro‘yxatini ko‘rsatish uchun “**Barcha kitoblar”** havolasini bosing .

15.1-rasm.

Keyin kitoblaringizdan biriga havolani bosing. Agar hamma narsa to‘g‘ri sozlangan bo‘lsa, siz quyidagi skrinshotga o‘xshash narsani ko‘rishingiz kerak.



15.2-rasm.

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.

3. 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 .

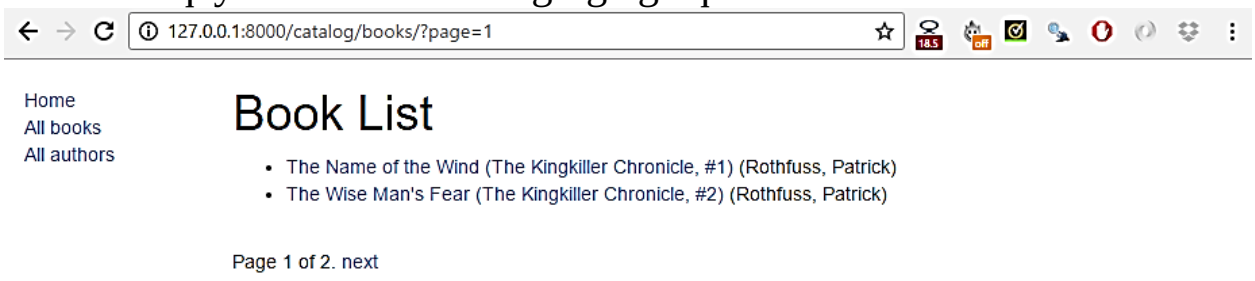
```
{% block pagination %}
  {% if is_paginated %}
    <div class="pagination">
      <span class="page-links">
        {% if page_obj.has_previous %}
          <a href="{{ request.path }}?page={{
page_obj.previous_page_number }}">previous</a>
        {% endif %}
        <span class="page-current">
          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.



15.3-rasm.

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.

15.4-rasm.

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. Algorithms, Fourth Edition (Deluxe): Book and 24-Part Lecture Series 1st Edition , Addison-Wesley Professional , USA, 2015.
10. Chris Roffey. Computer science. Programming book for Python. – USA:Cambridge university press. 2017, – p .204.
11. Chris Roffey. Python basics. Coding club. Level 1,2. – USA:Cambridge university press. 2012, – p.85.
12. Ro‘ziyev R.A. Dasturlash asoslari // O‘quv qo‘llanma. ¬ Toshkent, 2020. – 159 b.
13. 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.
14. Mirsanov U.M., Toxirov F.J., Norbekov A.O., Djurayeva D.R. Dasturlash. // O‘quv qo‘llanma. Toshkent, 2021. – 152 b.
15. "Python Programming: An Introduction to Computer Science" - John Zelle
16. "Automate the Boring Stuff with Python" - Al Sweigart
17. "Python Crash Course" - Eric Matthes
18. "Fluent Python" - Luciano Ramalho

19. "Learning Python" - Mark Lutz
20. "Effective Python: 90 Specific Ways to Write Better Python" - Brett Slatkin
21. "Python Cookbook" - David Beazley and Brian K. Jones
22. "Think Python: How to Think Like a Computer Scientist" - Allen B. Downey
23. "Learning Python Design Patterns" - Gennadiy Zlobin
24. "Python for Data Analysis" - Wes McKinney
25. https://metanit.com/python
26. https://pythonworld.ru/tipy-dannyx-v-python/slovari-dict-funkcii-i-metody-slovarej.html
27. https://ai.mohirdev.uz/
28. https://www.w3resource.com/
29. https://www.sololearn.com/
30. https://leetcode.com/
31. https://www.hackerrank.com/
32. https://www.tiobe.com/tiobe-index/
33. acmp.ru
34. https://docs.python.org/
35. https://praktikum.mohirdev.uz/

Qaydlar uchun

U.T. Subxonqulov

PYTHON DASTURLASH TILI

O‘QUV QO‘LLANMA

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: 12.07.2024. Bichimi 60x84. Kegli 16 shponli. “Times New Roman” garnitura 1/16. Elektrografik usulda. Oddiy bosma qog‘ozi.
Bosma tabog‘i 8. Adadi 100. Buyurtma № 425

KAMOLOT

“BUXORO DETERMINANTI” MCHJ
bosmaxonasida chop etildi.
Buxoro shahar Namozgoh ko‘chasi 24 uy
Tel:. + 998 91 310 27 22