|  |
| --- |
| **Level 4 | Programming** |
| RUXULLOXON AXRORXONOV |

**Unit :** Level 4 | Programming

**O'qituvchi:** Obid Jurayev

**Guruh raqmi:** 23-408 Software(Python)

**Talaba ID raqami:** 230591

**Taqdim etilgan sana:** 30.06.2025

**BTEC O'QUVCHILAR TOPSHIRIQLARINI BAHOLASH VA DEKLRITSIYASI**

Baholash uchun ishlarnni taqdim etganda, har bir o'quvchi ish o'ziniki ekanligini tasdiqlovchi deklaratsiyani imzolashi kerak.

|  |  |
| --- | --- |
| **O'quvchi (talaba) identifikatori:** | 230591 |
| **Baholovchi nomi:** | Obid Jurayev |
| **BTEC dasturi nomi:** | Pearson BTEC Higher Nationals in Information Technologies |
| **Birlik yoki komponent raqami va nomi:** | Level 4 | Programming |
| **Topshiriq nomi:** | Level 4 | Programming |
| **Topshiriq topshirilgan sana:** | 30.06.2025 |

Iltimos, har bir topshiriq uchun berilgan ishlarni sanab o'ting. Ishlarni topish mumkin bo'lgan sahifa raqamlarini ko'rsating yoki ishlarning mohiyatini tavsiflang (masalan, diagramma, rasm).

|  |  |  |
| --- | --- | --- |
| **Topshiriq vazifasi ma'lumoti** | **Ishlar taqdim etildi** | **Sahifa** |
|  |  |  |
|  |  |  |
|  |  |  |

|  |
| --- |
| **O'quvchi deklaratsiyasi**  Ushbu topshiriq uchun taqdim etilgan ish meniki ekanligini tasdiqlayman. Ishda foydalanilgan manbalarga aniq havola qildim. Men noto'g'ri deklaratsiya noto'g'ri ishlashning bir shakli ekanligini tushunaman.  O'quvchi imzosi:  Sana: 30.06.2025 |

MUNDARIJA

Kirish……………………………………………………………………………………………………………………..….4

Topshiriq 1……………………………………..…………………………………………………………………….…..5

Topshiriq 2……………………………………………………………………………..……………….………………..7

Topshiriq 3………………………………………………………………………………………………….…………..10

Topshiriq 4………………………………………………………………………………………..…………………….12

Xulosa …………………………………………………………………………………………………………….….…..16

Manbalar ro’yxati………………………………………………………………………………………………..….17

## Kirish

Ushbu hisobot **Pearson BTEC Higher Nationals in Digital Technologies** kursining **Unit 4: Programming** bo'yicha **"Ta’minot Zanjirini Boshqarishning Intellektual Qaror Qabul Qilishni Qo‘llab-Quvvatlash Tizimi (SCM-IDSS)"** deb nomlangan topshiriqning chuqur tahlili va yechimlarini o'z ichiga oladi. Loyiha an'anaviy ta'minot zanjiri boshqaruvi jarayonlarini Sun'iy intellekt (AI), Mashinali o'rganish (ML) va ilg'or tahlil texnologiyalari yordamida intellektual qaror qabul qilish imkoniyatlarini joriy etish orqali tubdan o'zgartirishga qaratilgan.

Ushbu topshiriq doirasida biz dasturlashning asosiy tamoyillaridan tortib, murakkab tizimlarni yaratishgacha bo'lgan bosqichlarni o'rgandik va amaliyotga tatbiq etdik. Hisobotning asosiy maqsadi — SCM-IDSS kabi intellektual tizimlarni yaratishda dasturlash jarayonining turli jihatlarini, jumladan, algoritmlarning aniqlanishi va ilovalarni yaratish bosqichlarini (LO1), protsessual, ob'ektga yo'naltirilgan va voqea-drayv (event-driven) dasturlash paradigmalarining xususiyatlarini (LO2), integratsiyalashgan rivojlanish muhitlaridan (IDE) foydalangan holda algoritmlarni kodda implementatsiya qilishni (LO3), shuningdek, nosozliklarni tuzatish (debugging) jarayoni va kodlash standartlarining ahamiyatini (LO4) batafsil tushuntirishdir.

SCM-IDSS loyihasi Python/Django, RabbitMQ, Redis va Docker kabi zamonaviy texnologiyalar to'plamidan foydalangan holda mikroservislar arxitekturasi asosida ishlab chiqilgan. Ushbu hisobot, har bir o'rganish natijasi bo'yicha berilgan topshiriqlarni SCM-IDSS misolida batafsil yoritib beradi va dasturiy ta'minotni ishlab chiqishning nazariy hamda amaliy jihatlari o'rtasidagi bog'liqlikni tushuntiradi.

### P1 Algoritm nima ekanligiga ta'rif bering va dastur yaratish jarayonini tavsiflang

#### Algoritmning Ta'rifi va Mohiyati

**Algoritm** — bu muayyan muammoni hal qilish yoki aniq bir maqsadga erishish uchun mo'ljallangan, bir-biridan keyin bajariladigan aniq, chekli qadamlar ketma-ketligidir. Dasturlashning asosini tashkil etuvchi bu tushuncha shunchaki ko'rsatmalar yig'indisi emas, balki mantiqiy tuzilgan yechim xaritasidir. Algoritmni oddiy oshxona retseptiga qiyoslash mumkin: retseptda taom tayyorlash uchun kerakli masalliqlar (kirish ma'lumotlari), bajariladigan amallar (qadamlar) va natija (chiqish ma'lumotlari) aniq ko'rsatilganidek, algoritm ham kompyuterga nima qilish kerakligini qadamma-qadam tushuntirib beradi.

Yaxshi algoritm quyidagi fundamental xususiyatlarga ega bo'lishi shart:

* **Aniq (Unambiguous):** Har bir qadam, amal va ko'rsatma faqat bitta ma'noga ega bo'lishi kerak. Kompyuter uchun "taxminan", "balki" degan tushunchalar mavjud emas; har bir buyruq mutlaq aniqlikni talab qiladi.
* **Kirish (Input):** Algoritm o'z ishini bajarish uchun boshlang'ich ma'lumotlarni qabul qilishi kerak. Bu ma'lumotlar nol (ya'ni, hech qanday kirish ma'lumotisiz) yoki undan ko'p bo'lishi mumkin.
* **Chiqish (Output):** Har qanday algoritm o'z ishini yakunlagach, kamida bitta natijani qaytarishi shart. Bu natija muammoning yechimi hisoblanadi.
* **Cheklanganlik (Finiteness):** Algoritm abadiy davom etmasligi kerak. U chekli sondagi qadamlardan so'ng, albatta, o'z ishini yakunlashi lozim.
* **Samaradorlik (Effectiveness):** Algoritmning har bir amali juda oddiy bo'lishi va uni qog'oz va qalam yordamida ham chekli vaqtda bajarish imkoni bo'lishi kerak.

**SCM-IDSS loyihasi misolida**, algoritmlar tizimning har bir funksional talabida namoyon bo'ladi. Masalan, **FR3.2: "Tizim sekin harakatlanuvchi va eskirgan inventarni aniqlashi kerak"** talabini bajarish uchun quyidagi soddalashtirilgan algoritm ishlab chiqilishi mumkin:

1. **Kirish:** Tekshirish kerak bo'lgan vaqt oralig'ini (masalan, so'nggi 90 kun) qabul qilish.
2. **Sikl (1):** Barcha mahsulotlar (products) ro'yxatini birma-bir ko'rib chiqish.
3. **Sikl (2):** Har bir mahsulot uchun, belgilangan vaqt oralig'ida unga tegishli bo'lgan barcha buyurtmalarni (order\_items) topish.
4. **Shart:** Agar mahsulot uchun hech qanday buyurtma topilmasa yoki sotilgan miqdori belgilangan me'yordan (masalan, 5 donadan) kam bo'lsa, uni "sekin harakatlanuvchi" deb belgilash.
5. **Chiqish:** "Sekin harakatlanuvchi" deb belgilangan barcha mahsulotlar ro'yxatini qaytarish.

Ko'rinib turibdiki, oddiy ko'ringan talab ortida ham aniq bir algoritm yotadi.

#### Dastur Yaratish Jarayoni (Software Development Life Cycle - SDLC)

Dastur yaratish — bu tartibsiz ijodiy jarayon emas, balki muhandislik yondashuvini talab qiladigan, aniq bosqichlardan iborat hayotiy sikldir. Ushbu loyihada zamonaviy, iterativ (qayta-qayta takrorlanuvchi) yondashuvga asoslangan SDLC qo'llanilgan. Bu, ayniqsa, **Agile metodologiyasiga** yaqin bo'lib, loyihani kichik qismlarga bo'lib, har birini bosqichma-bosqich bajarib borishni nazarda tutadi.

1. **Talablarni Tahlil Qilish (Requirements Analysis):** Bu poydevor qo'yish bosqichi. Bu yerda nafaqat "nima qilish kerak?", balki "bu nima uchun kerak?" va "kim uchun kerak?" degan savollarga javob topiladi. SCM-IDSS loyihasi uchun bu bosqichda barcha foydalanuvchi rollari (rahbar, menejer, tahlilchi) va ularning ehtiyojlari aniqlandi. FR1.1 ("sozlanishi asboblar panelini taqdim etish") va FR8.5 ("ma'lumotlarni vizualizatsiya qilish vositalarini taqdim etish") kabi talablar aynan foydalanuvchi ehtiyojlaridan kelib chiqib shakllantirildi.
2. **Loyiha va Dizayn (Design):** Bu bosqichda tizimning "arxitektura chizmasi" yaratiladi. Talablarni qanday qilib texnik yechimga aylantirish masalasi hal qilinadi. SCM-IDSS loyihasi uchun **Mikroservislar arxitekturasi** tanlanishining asosiy sababi — tizimning murakkabligi va kengayuvchanligidir. Har bir biznes mantig'i (inventar, buyurtma, logistika) alohida servisga ajratildi. Bu bir qismdagi o'zgarish boshqasiga ta'sir qilmasligini va har bir servisni alohida jamoa rivojlantirishini ta'minlaydi. Shuningdek, ma'lumotlar bazasining sxemasi (PostgreSQL jadvallari va ular orasidagi munosabatlar) ham aynan shu bosqichda loyihalanadi.
3. **Dasturlash (Implementation/Coding):** Bu — dizaynni haqiqiy, ishlaydigan kodga aylantirish bosqichi. Arxitektor va dizaynerlar chizib bergan reja asosida dasturchilar kod yozadi. SCM-IDSS loyihasida backend uchun Python va Django REST Framework, frontend uchun esa HTML, Bootstrap va JavaScript tanlandi. Bu tanlov Python'ning sun'iy intellekt va ma'lumotlar tahlili uchun boy kutubxonalarga egaligi bilan ham izohlanadi.
4. **Testlash (Testing):** Sifatni ta'minlashning eng muhim qismi. Dasturchi "mening kodim ishlaydi" deyishi mumkin, ammo tester "u sinmaydimi?" degan savolni tekshiradi. Bu bosqichda har bir funksiya alohida (unit test), bir nechta qismlar birgalikda (integration test) va butun tizim bir butun sifatida (system test) tekshiriladi. SCM-IDSS loyihasida "agar omborda mahsulot yo'q bo'lsa, buyurtma yaratilmasligi kerak" kabi biznes qoidalari aynan testlar orqali tekshiriladi.
5. **Joriy Qilish (Deployment):** Tayyor mahsulotni foydalanuvchilar qo'liga topshirish jarayoni. Zamonaviy tizimlarda bu jarayon avtomatlashtirilgan bo'ladi. Bizning loyihamizda **Docker va Docker Compose**'dan foydalanish, barcha servislarni va ularning bog'liqliklarini birgina buyruq bilan serverga joylashtirish imkonini beradi. Bu "mening kompyuterimda ishlagan edi" degan muammoning oldini oladi.
6. **Qo'llab-quvvatlash (Maintenance):** Dastur hayotining eng uzun davom etadigan bosqichi. Tizim ishga tushgandan so'ng, uning uzluksiz ishlashini ta'minlash, yangi talablarga moslashtirish, kichik xatoliklarni tuzatish va unumdorligini oshirib borish kerak bo'ladi.

### M1 Kod yozishdan uni bajarishgacha bo'lgan qadamlarni aniqlang

Dasturchi yozgan kod — bu shunchaki matn. Bu matn kompyuter protsessori tushunadigan elektr impulslariga aylanishi uchun bir necha bosqichli "tarjima" jarayonidan o'tishi kerak. Python kabi yuqori darajali, interpretatsiya qilinadigan tillarda bu jarayon quyidagicha kechadi:

1. **Kod Yozish (Source Code):** Jarayonning boshlanishi — bu dasturchining matn muharriri yoki IDE (Masalan, VS Code, PyCharm) yordamida .py kengaytmali faylga Python tilining qoidalari va sintaksisiga binoan kod yozishidir. Bu kod inson uchun tushunarli bo'ladi.
2. **Interpretatsiya (Interpretation):** python manage.py runserver kabi buyruq berilganda, Python interpretatori faylni satrma-satr o'qib, uni bajarishga kirishadi. Bu jarayon o'z navbatida bir nechta kichik bosqichlardan iborat:
   * **Lexing (Tokenizatsiya):** Kod matni mantiqiy birliklarga — **tokenlarga** bo'linadi. Masalan, price = 100 qatori price (identifier), = (operator), 100 (literal) kabi tokenlarga ajratiladi. Bu xuddi jumlani so'zlarga ajratishga o'xshaydi.
   * **Parsing (Sintaktik tahlil):** Tokenlar oqimidan kodning grammatik tuzilishini ifodalovchi iyerarxik daraxt — **Abstrakt Sintaktik Daraxt (Abstract Syntax Tree - AST)** quriladi. Bu bosqichda kodning sintaksisi to'g'riligi tekshiriladi. Agar sintaktik xatolik (masalan, qavs yopilmagan bo'lsa) mavjud bo'lsa, SyntaxError yuzaga keladi va dastur to'xtaydi.
   * **Bayt-kodga Kompilyatsiya:** Hosil bo'lgan AST Python Virtual Mashinasi (PVM) uchun maxsus, platformaga bog'liq bo'lmagan oraliq ko'rsatmalar to'plami bo'lgan **bayt-kodga** "tarjima" qilinadi. Bu bayt-kod protsessorning asl buyruqlariga qaraganda ancha yuqori darajali, ammo Python kodiga qaraganda ancha past darajali bo'ladi. Python bu bayt-kodni .pyc kengaytmali fayllarda keshlab qo'yadi. Shu sababli, bir marta ishga tushirilgan dastur, keyingi safar tezroq ishlaydi, chunki interpretator to'g'ridan-to'g'ri bayt-koddan boshlaydi.
3. **Bajarish (Execution):** Bu bosqich bo'lib, uni **Python Virtual Mashinasi (PVM)** amalga oshiradi. PVM bu loyihaning "yuragi" hisoblanadi. U bayt-kodni olib, uni birma-bir, satrma-satr bajaradi. Har bir bayt-kod buyrug'i PVM ichidagi ma'lum bir funksiyaga mos keladi (masalan, o'zgaruvchini stekka qo'shish, ikkita sonni qo'shish). PVM bu amallarni bajarish uchun operatsion tizim bilan muloqot qiladi va natijada markaziy protsessor kerakli hisob-kitoblarni amalga oshiradi. Aynan shu bosqichda o'zgaruvchilar xotirada joy oladi, funksiyalar chaqiriladi va dasturning mantiqiy amallari hayotga tatbiq etiladi.

### D1 Algoritmning mos tilda amalga oshirilishini va yozilgan algoritm bilan kod varianti o'rtasidagi munosabatni baholang

Ushbu tahlil uchun order\_servicedagi **yangi buyurtma yaratish** jarayonini algoritm sifatida ko'rib chiqamiz. Bu jarayon mikroservislar arxitekturasining barcha nozikliklarini o'zida jamlagan yaxshi misoldir.

#### Yozma Algoritm (Pseudokod)

1. **Boshlash:** Foydalanuvchi tokeni va so'rov tanasidan shipping\_address va items (mahsulotlar ro'yxati) ma'lumotlarini qabul qilish.
2. **Validatsiya (1):** Agar items ro'yxati bo'sh bo'lsa, xatolik qaytarib, jarayonni to'xtatish.
3. **Ma'lumotlarni To'plash (Sikl):** items ro'yxatidagi har bir item uchun: a. inventory\_servicega HTTP so'rovi yuborib, shu itemning ombordagi miqdorini (available\_quantity) so'rash. b. Agar ulanishda xatolik bo'lsa yoki mahsulot topilmasa, xatolik qaytarish. c. Agar available\_quantity buyurtma qilinayotgan miqdordan kam bo'lsa, "mahsulot yetarli emas" degan xatolik qaytarish. d. inventory\_servicega yana bir HTTP so'rovi yuborib, shu itemning narxini (unit\_price) so'rash. e. Umumiy summani (total\_amount) hisoblash: total\_amount += unit\_price \* quantity.
4. **Ma'lumotlar Bazasi Operatsiyasi (Tranzaksiya):** a. Barcha keyingi amallarni bitta, bo'linmas tranzaksiya ichida boshlash. Bu biror xatolik yuzaga kelsa, barcha o'zgarishlarni bekor qilish imkonini beradi. b. CustomerOrder yozuvini umumiy summa, foydalanuvchi IDsi va manzil bilan birga bazaga saqlash. c. Har bir item uchun OrderItem yozuvini yaratib, uni asosiy buyurtmaga bog'lab, bazaga saqlash. d. Agar barcha amallar muvaffaqiyatli bo'lsa, tranzaksiyani tasdiqlash.
5. **Asinxron Xabarlar Yuborish:** a. RabbitMQ orqali orders\_exchange'ga ORDER\_CREATED hodisasi haqida xabar yuborish (bu xabarni logistics\_service va boshqalar eshitishi mumkin). b. RabbitMQ orqali inventory\_exchange'ga DECREASE\_STOCK hodisasi haqida xabar yuborish (inventory\_service eshitishi uchun).
6. **Javob Qaytarish:** Foydalanuvchiga muvaffaqiyatli yaratilgan buyurtma ma'lumotlarini JSON formatida, 201 Created status kodi bilan qaytarish.
7. **Tugatish.**

#### Kod Varianti (Python/Django)

Ushbu algoritm order\_service/api/orders/serializers.py faylidagi UserOrderCreateSerializer klassining create metodida amalga oshirilgan.

# ... (fragment of the UserOrderCreateSerializer.create method)

def create(self, validated\_data):

# ... (1 va 2-qadamlar: Ma'lumotlarni qabul qilish va validatsiya)

for item in items\_data:

# ... (3-qadam: Sikl boshlanishi)

try:

# (3a va 3c-qadamlar: HTTP so'rovlari)

inv\_response = requests.get(f"{INVENTORY\_URL}?product\_id={product\_id}", ...)

prod\_response = requests.get(f"{PRODUCTS\_URL}{product\_id}/", ...)

# (3b-qadam: Miqdorni tekshirish)

if available\_quantity < quantity\_to\_order:

raise ValidationError(...)

# (3d-qadam: Summani hisoblash)

total\_amount += unit\_price \* quantity\_to\_order

processed\_items.append(...)

# ...

# (4-qadam: Tranzaksiyani boshlash)

with transaction.atomic():

# (4b-qadam: Asosiy buyurtmani saqlash)

order = CustomerOrder.objects.create(...)

# (4c-qadam: Har bir item'ni saqlash)

for item\_data in processed\_items:

OrderItem.objects.create(order=order, \*\*item\_data)

# (4d-qadam: Tranzaksiya avtomatik yakunlanadi)

# (5a va 5b-qadamlar: RabbitMQ'ga xabar yuborish)

publish\_message('orders\_exchange', ...)

publish\_message('inventory\_exchange', ...)

# (6 va 7-qadamlar: Javob qaytarish va Tugatish)

return order

#### Baholash (Evaluation)

Yozma algoritm va uning Python/Django'dagi kod varianti o'rtasidagi munosabatni baholashda bir nechta muhim jihatlarni ko'rib chiqish lozim:

* **Moslik va Abstraksiya:** Algoritmning har bir mantiqiy qadami kodda aniq bir funksiya yoki buyruqqa mos keladi. Python'ning yuqori darajali sintaksisi (for item in items\_data:) va Django'ning kuchli vositalari (transaction.atomic(), Model.objects.create()) algoritm mantig'ini deyarli birga-bir, ammo ancha qisqa va tushunarli tarzda ifodalash imkonini beradi. Django ORM va kontekst menejerlari "Tranzaksiyani Boshlash", "Saqlash", "Yakunlash" kabi murakkab, past darajali amallarni bitta qator kodga soddalashtirib, dasturchini SQL so'rovlari va xatoliklarni qo'lda boshqarishdan xalos qiladi. Bu **abstraksiya** darajasining yuqoriligi hisoblanadi.
* **Tilning Mosligi va Ekosistema:** Python bu vazifa uchun juda mos til sifatida tanlangan. Uning requests kabi standart kutubxonalari HTTP so'rovlarini yuborishdek murakkab vazifani bir qator kod bilan bajarish imkonini beradi. pika kutubxonasi esa RabbitMQ bilan ishlashni soddalashtiradi. Agar bu algoritm C++ kabi past darajali tilda yozilganida, tarmoq ulanishlari, HTTP protokoli va ma'lumotlar bazasi bilan ishlash uchun yuzlab qator qo'shimcha kod yozish talab etilgan bo'lardi.
* **Kamchiliklar va Kompromisslar:** Python interpretatsiya qilinadigan til bo'lgani uchun, u C++ yoki Java kabi kompilyatsiya qilinadigan tillarga qaraganda sekinroq ishlashi mumkin. Biroq, SCM-IDSS kabi biznes tizimlarida dasturning ishlash tezligi ko'pincha tarmoq ulanishlari va ma'lumotlar bazasiga so'rovlar bilan chegaralanadi. Dasturning o'zi emas, balki tarmoq kechikishlari asosiy "sekin" nuqta bo'ladi. Shu sababli, Python'ning ishlab chiqish tezligi (rapid development) uning ishlash tezligidagi kichik kamchiligini ortig'i bilan qoplaydi.

Tanlangan Python tili va Django freymvorki SCM-IDSS tizimining murakkab, tarqoq algoritmlarini amalga oshirish uchun juda mos va samarali yechimdir. Ular algoritmning mantiqiy tuzilishini saqlagan holda, uni ishonchli, xavfsiz va eng muhimi — o'zgartirish va qo'llab-quvvatlash uchun oson bo'lgan kodga aylantirish imkonini beradi. Algoritm va kod o'rtasidagi bu uyg'unlik loyihaning muvaffaqiyatini ta'minlovchi asosiy omillardan biridir.

### P2 Protsessual, Obyektga Yo'naltirilgan va Hodisaga Asoslangan paradigmalar nima ekanligi, ularning xususiyatlari va o'zaro munosabatlari haqida tushuntirish bering

Dasturlashda **paradigma** — bu dasturiy ta'minotni loyihalash va qurish uchun asos bo'ladigan fundamental uslub, yondashuv yoki fikrlash tarzidir. U kodni qanday tashkil etish, ma'lumotlarni qanday boshqarish va dastur mantig'ini qanday ifodalash kerakligini belgilaydi. Keling, uchta asosiy paradigmani ko'rib chiqamiz.

#### 1. Protsessual Dasturlash (Procedural Programming)

Bu eng an'anaviy paradigmalardan biri bo'lib, dasturni ketma-ket bajariladigan buyruqlar yoki protseduralar (funksiyalar) yig'indisi sifatida qaraydi. Asosiy e'tibor "qanday qilish kerak?" degan savolga, ya'ni amallarni bajarish algoritmini belgilashga qaratiladi.

* **Xususiyatlari:**
  + **Yuqoridan Pastga (Top-Down Approach):** Dastur mantig'i boshidan boshlanib, qadamma-qadam oxirigacha bajariladi.
  + **Funksiyalarga Bo'linish:** Katta va murakkab dasturlar kichikroq, qayta ishlatilishi mumkin bo'lgan funksiyalarga bo'linadi.
  + **Global Ma'lumotlar:** Ma'lumotlar va ular ustida amal bajaradigan funksiyalar bir-biridan ajratilgan. Ko'pincha, ma'lumotlar global o'zgaruvchilarda saqlanadi, bu esa dasturning istalgan qismidan ularga kirish imkonini beradi, lekin katta loyihalarda bu ma'lumotlarning kutilmaganda o'zgarib ketishiga va chalkashliklarga olib kelishi mumkin.
  + **Strukturaviy Nazorat:** if/else, for, while kabi tuzilmalar yordamida dastur oqimi boshqariladi.
* **Misol:** Oddiy hisob-kitob qiladigan dasturlar, skriptlar.

#### 2. Obyektga Yo'naltirilgan Dasturlash (Object-Oriented Programming - OOP)

Bu paradigma dasturni real dunyoning aksi sifatida, o'zaro aloqada bo'lgan "obyektlar" majmuasi sifatida tasavvur qiladi. Har bir obyekt o'zining ma'lumotlari (**atributlar**) va xulq-atvoriga (**metodlar**) ega. Asosiy g'oya — ma'lumotlar va ular bilan ishlaydigan mantiqni bitta obyekt ichida jamlash (inkapsulyatsiya).

* **Xususiyatlari (4 tamoyili):**
  1. **Inkapsulyatsiya (Encapsulation):** Obyektning ichki holatini (atributlarini) tashqi dunyodan himoyalash va ularga faqat obyektning o'z metodlari orqali murojaat qilishni ta'minlash. Bu ma'lumotlarning tasodifiy o'zgarishlardan himoyalanishini kafolatlaydi.
  2. **Merosxo'rlik (Inheritance):** Mavjud klass (ota-klass) asosida yangi klass (bola-klass) yaratish imkoniyati. Bola-klass ota-klassning barcha xususiyatlari va metodlariga ega bo'ladi va ularni kengaytirishi yoki o'zgartirishi mumkin. Bu kodni qayta ishlatishning kuchli vositasidir.
  3. **Polimorfizm (Polymorphism):** "Ko'p shakllilik" degan ma'noni anglatadi. Turli klasslarga mansub obyektlar bir xil nomdagi metodga o'ziga xos tarzda javob berishi mumkin. Masalan, calculate\_area() metodi "Kvadrat" uchun boshqacha, "Doira" uchun boshqacha ishlaydi.
  4. **Abstraksiya (Abstraction):** Foydalanuvchiga obyektning faqat kerakli, yuqori darajali interfeysini ko'rsatib, uning murakkab ichki tuzilishini yashirish.
* **Misol:** SCM-IDSS loyihasidagi Product, Order, User kabi har bir tushuncha alohida obyekt sifatida modellashtirilgan.

#### 3. Hodisaga Asoslangan Dasturlash (Event-Driven Programming - EDP)

Bu paradigmada dasturning oqimi foydalanuvchi harakatlari, sensorlardan kelgan signallar yoki boshqa dasturlardan kelgan xabarlar kabi tashqi **hodisalar (events)** bilan boshqariladi. Dastur doimiy "eshitish" holatida bo'lib, biror hodisa yuz berganda unga javob qaytaradigan maxsus funksiyani (handler/callback) ishga tushiradi.

* **Xususiyatlari:**
  + **Hodisa Halqasi (Event Loop):** Dasturning markazida doimiy ravishda ishlab, hodisalarni kutib turadigan event loop yotadi.
  + **Hodisa Ishlovchilari (Event Handlers):** Har bir hodisa turi uchun maxsus funksiya (handler) belgilanadi. Hodisa yuz berganda, event loop tegishli handlerni chaqiradi.
  + **Asinxronlik:** Hodisalar kutilmaganda va turli vaqtlarda yuz berishi mumkin. Bu paradigma asinxron operatsiyalar uchun juda mos keladi, chunki dastur bir hodisani kutib, bloklanib qolmaydi.
* **Misol:** Foydalanuvchi interfeyslari (masalan, tugma bosilishi hodisasi), tarmoq ilovalari (yangi so'rov kelishi hodisasi) va bizning loyihadagi RabbitMQ iste'molchilari.

#### Paradigmalar O'rtasidagi Munosabat

Bu uch paradigma bir-birini inkor etmaydi, aksincha, zamonaviy murakkab tizimlarda ular birgalikda ishlaydi:

* **OOP vs Protsessual:** OOP ko'p hollarda protsessual dasturlashning evolyutsiyasi sifatida qaraladi. U ma'lumotlar va funksiyalarni birlashtirib, katta tizimlarni boshqarishni osonlashtiradi. Ammo, har bir obyektning metodi o'z ichida protsessual algoritm, ya'ni qadamlar ketma-ketligi bo'lishi mumkin.
* **EDP va Boshqalar:** Hodisaga asoslangan dasturlash ko'pincha OOP bilan birga ishlatiladi. Masalan, foydalanuvchi interfeysida "Tugma" (Button) obyekti click (bosilish) hodisasiga ega bo'lishi mumkin. Bu hodisaga javob beradigan handler esa o'z navbatida boshqa obyektlarning metodlarini chaqiradi. Bizning SCM-IDSS loyihamizda, order\_servicedagi obyekt (CustomerOrder) yaratilganda, bu hodisa (ORDER\_CREATED) logistics\_servicedagi iste'molchini ishga tushiradi.

### M2 Berilgan dastur kodida qo'llanilgan protsessual, obyektga yo'naltirilgan va hodisaga asoslangan paradigmalarni solishtiring va qarama-qarshiliklarini ko'rsating

SCM-IDSS loyihasi ushbu uchala paradigmaning uyg'unlashuviga yorqin misol bo'la oladi.

* **Obyektga Yo'naltirilgan (Asosiy paradigma):**
  + **Qo'llanilishi:** Loyihaning butun backend qismi Django freymvorkiga asoslangan bo'lib, u tabiatan obyektga yo'naltirilgandir. inventory/models.py faylidagi Product klassi bunga yaqqol misol. U mahsulot haqidagi ma'lumotlarni (sku, name, price — atributlar) va ular bilan bog'liq amallarni (Django ORM orqali taqdim etiladigan save(), delete() — metodlar) bitta birlikda jamlaydi.
  + **Protsessual yondashuv bilan taqqoslash:** Agar bu vazifa faqat protsessual usulda yechilganida, bizda product\_names = [], product\_prices = [] kabi alohida ro'yxatlar va add\_new\_product(name, price), get\_product\_price(name) kabi alohida funksiyalar bo'lar edi. Bu yondashuvda ma'lumotlar va mantiq bir-biridan ajralib, kodning chalkashib ketishi ehtimoli yuqori bo'lardi. OOP esa bularni Product nomli bitta "quti"ga solib, tartibga keltiradi.
* **Hodisaga Asoslangan (Servislararo aloqa):**
  + **Qo'llanilishi:** Bu paradigma mikroservislar o'rtasidagi aloqani ta'minlashda markaziy o'rin tutadi. order\_servicedagi UserOrderCreateSerializerning create metodi yangi buyurtma yaratgach, u logistics\_servicedagi biror funksiyani to'g'ridan-to'g'ri chaqirmaydi. Buning o'rniga, u shunchaki **ORDER\_CREATED hodisasini** RabbitMQ'ga yuboradi. logistics\_service esa o'zining run\_consumer.py skripti orqali doimiy ravishda shu hodisani kutib turadi. Hodisa kelganda, process\_logistics\_event funksiyasi ishga tushib, yangi jo'natma yaratadi.
  + **Boshqa paradigmalar bilan qarama-qarshiligi:** Agar bu jarayon hodisaga asoslanmaganida, order\_service logistics\_service bilan **kuchli bog'langan (tightly coupled)** bo'lar edi. U to'g'ridan-to'g'ri HTTP so'rovi yuborishi va uning javobini kutishi kerak bo'lardi. Agar logistics\_service vaqtincha ishlamay qolsa, buyurtma yaratish jarayoni ham to'xtab qolardi. Hodisaga asoslangan yondashuv esa order\_servicega "xabarni yubor va o'z ishingni davom ettir" deyish imkonini beradi, bu esa tizimning mustahkamligini (resilience) ta'minlaydi.
* **Protsessual (Kichik amallar):**
  + **Qo'llanilishi:** Har bir obyektning metodi yoki hodisa ishlovchisi ichidagi mantiqning o'zi kichik bir protsessual algoritmdir. Masalan, UserOrderCreateSerializer'ning create metodi ichidagi umumiy summani hisoblash qismi (total\_amount += unit\_price \* quantity\_to\_order) yoki process\_logistics\_event funksiyasi ichidagi Shipment.objects.create(...) qadami — bu aniq bir ketma-ketlikdagi protsessual amallardir.
  + **Umumiy o'rni:** Protsessual yondashuv butun loyihaning arxitekturasini emas, balki eng kichik, fundamental amallarni bajarish uchun ishlatiladi.

### D2 Protsessual, obyektga yo'naltirilgan va hodisaga asoslangan paradigmalarni qo'llagan dastur kodining tuzilishi va xususiyatlarini tanqidiy baholang

SCM-IDSS loyihasining kod tuzilishini tanqidiy baholash shuni ko'rsatadiki, unda uchala paradigmaning ham uyg'unlashuvi loyihaning muvaffaqiyati uchun hal qiluvchi ahamiyatga ega bo'lgan.

**1. Obyektga Yo'naltirilgan Yondashuvning Tanqidiy Tahlili:**

* **Afzalliklari:** Django va Django REST Framework'ning OOPga asoslanganligi loyihaga juda kuchli poydevor berdi. models.py orqali ma'lumotlar bazasini modellashtirish, serializers.py orqali ma'lumotlar formatini belgilash va views.py'dagi ViewSetlar orqali biznes mantiqini joylashtirish — bu kodni **tuzilmali, o'qilishi oson va parvarishlanadigan (maintainable)** qildi. Yangi funksiya qo'shish uchun yangi metod yoki klass yaratish kifoya, bu esa mavjud kodga ta'sir qilish xavfini kamaytiradi.
* **Tanqidiy nuqta:** OOP ba'zan oddiy vazifalar uchun ortiqcha murakkablik tug'dirishi mumkin. Biroq, SCM-IDSS kabi yuzlab biznes qoidalariga ega bo'lgan murakkab tizim uchun bu murakkablik o'zini oqlaydi. Agar loyiha faqat protsessual usulda yozilganida, yuzlab funksiyalar va global o'zgaruvchilar orasida adashib qolish va "spaghetti code" yuzaga kelishi deyarli muqarrar bo'lardi. OOP bu xaosning oldini oladi.

**2. Hodisaga Asoslangan Yondashuvning Tanqidiy Tahlili:**

* **Afzalliklari:** RabbitMQ'ni qo'llash orqali erishilgan eng katta yutuq — bu **servislarning bir-biridan mustaqilligi (decoupling)**. order\_service yangi jo'natma qanday yaratilishini bilishi shart emas, u shunchaki "buyurtma yaratildi" degan signal beradi. Bu logistics\_serviceni kelajakda butunlay boshqa tilda yozilgan yangi servisga almashtirish imkonini beradi, bunda order\_service kodiga hech qanday o'zgartirish kiritilmaydi. Bu tizimning **moslashuvchanligi (flexibility)** va **kengayuvchanligi (scalability)** uchun juda muhim.
* **Tanqidiy nuqta:** Bu yondashuv o'zi bilan birga yangi murakkablikni olib keladi: xabarlar brokerini (RabbitMQ) sozlash va boshqarish kerak bo'ladi. Shuningdek, jarayonni kuzatish (debugging) qiyinlashadi, chunki amallar to'g'ridan-to'g'ri emas, balki bilvosita, xabarlar orqali sodir bo'ladi. Agar logistics\_consumer xabarni qayta ishlay olmasa, bu xatoni topish uchun alohida monitoring va loglash tizimlari kerak bo'ladi. Shunga qaramay, mikroservislar arxitekturasining afzalliklari bu kamchiliklarni ortig'i bilan qoplaydi.

**Baho:** SCM-IDSS loyihasining kod tuzilishi zamonaviy, murakkab tizimlarni qurishdagi eng yaxshi amaliyotlarni namoyish etadi. Unda har bir paradigma o'z o'rnida va to'g'ri maqsadda qo'llanilgan: **OOP** tizimga mustahkam tuzilma beradi, **EDP** bu tuzilmaning mustaqil qismlari o'rtasida ishonchli aloqani ta'minlaydi, **protsessual dasturlash** esa eng kichik amallarni bajarish uchun fundamental vosita bo'lib xizmat qiladi. Bu paradigmalarning sinergiyasi natijasida nafaqat ishlaydigan, balki kelajakda rivojlanishga va o'zgarishlarga oson moslashadigan barqaror tizim yaratilgan.

### P3. Dasturni IDE yordamida algoritm asosida yozish

IDE (Integrated Development Environment - Integratsiyalashgan Rivojlanish Muhiti) — bu dasturchilarga dasturiy ta'minotni ishlab chiqishda yordam beradigan keng qamrovli dasturiy ta'minot ilovasi. U kod tahrirlash, tuzatish (debugging), versiyalarni boshqarish va loyihani boshqarish kabi bir qator vositalarni bir joyga birlashtiradi. Men SCM-IDSS loyihasini rivojlantirishda **Visual Studio Code (VS Code)** IDE'sidan foydalandim.

Keling, loyihadan bir algoritmning, ya'ni **inventar miqdorini kamaytirish (DECREASE\_STOCK)** algoritmining kodini ko'rib chiqaylik. Bu algoritm inventory\_service/inventory/management/commands/run\_consumer.py faylida process\_inventory\_event funksiyasi ichida amalga oshirilgan:

Python

# inventory\_service/inventory/management/commands/run\_consumer.py

import pika

import json

import os

import django

from django.core.management.base import BaseCommand

from django.db import close\_old\_connections

from django.conf import settings

from inventory.models import inventory as Inventory # Model import qilingan

# Django muhitini sozlash

os.environ.setdefault("DJANGO\_SETTINGS\_MODULE", "inventory\_service.settings")

django.setup()

class Command(BaseCommand):

help = 'Starts the RabbitMQ consumer for inventory service'

def process\_inventory\_event(self, ch, method, properties, body):

try:

close\_old\_connections() # Eski DB ulanishlarini yopish

data = json.loads(body)

event\_type = data.get('event\_type')

self.stdout.write(self.style.SUCCESS(f" [x] Received event: {event\_type}"))

# Inventar miqdorini kamaytirish algoritmi

if event\_type == 'DECREASE\_STOCK': # 1. Voqea turini tekshirish

order\_items = data.get('items', []) # 2. Buyurtma mahsulotlarini olish

for item in order\_items: # 3. Har bir mahsulot bo'yicha takrorlash

product\_id = item.get('product\_id')

quantity\_to\_decrease = item.get('quantity')

if not product\_id or not quantity\_to\_decrease:

self.stderr.write(self.style.ERROR(f" [!] ERROR: Missing product\_id or quantity for an item."))

continue # Keyingi itemga o'tish

# 4. Inventardagi mahsulotni topish (o'zgarish paytida bloklash uchun select\_for\_update)

inventory\_item = Inventory.objects.select\_for\_update().filter(product\_id=product\_id).first()

if inventory\_item: # 5. Agar inventar yozuvi topilsa

if inventory\_item.quantity >= int(quantity\_to\_decrease): # 6. Miqdor yetarli bo'lsa

inventory\_item.quantity -= int(quantity\_to\_decrease) # 7. Miqdorni kamaytirish

inventory\_item.save() # 8. O'zgarishni saqlash

self.stdout.write(f" > Stock for product {product\_id} decreased by {quantity\_to\_decrease}. New quantity: {inventory\_item.quantity}")

else: # 9. Miqdor yetarli bo'lmasa

self.stdout.write(f" ! WARNING: Not enough stock for product {product\_id}. Required: {quantity\_to\_decrease}, Available: {inventory\_item.quantity}")

else: # 10. Inventar yozuvi topilmasa

self.stdout.write(f" ! WARNING: Inventory record not found for product {product\_id}.")

ch.basic\_ack(delivery\_tag=method.delivery\_tag) # Xabar muvaffaqiyatli qayta ishlandi

except Exception as e:

self.stderr.write(self.style.ERROR(f" [!] ERROR in callback: {e}"))

self.stderr.write(traceback.format\_exc())

ch.basic\_nack(delivery\_tag=method.delivery\_tag, requeue=False) # Xatoni qayta navbatga qo'ymaslik

Ushbu kod parchalari IDE (VS Code) ichida yoziladi va uning sintaksisni ajratib ko'rsatish, avtomatik to'ldirish kabi xususiyatlaridan foydalangan holda samarali tarzda yaratiladi. Algoritmning har bir bosqichi (voqea qabul qilish, ma'lumotlarni tahlil qilish, inventarni yangilash) kodda aniq ifodalangan.

### M3. IDE'dan rivojlanish jarayonini boshqarishda foydalanish

VS Code kabi IDE'lar dasturiy ta'minotni ishlab chiqish jarayonini kompleks boshqarish uchun bir nechta muhim xususiyatlarni taqdim etadi:

1. **Kod tahrirlash va navigatsiya:**
   * **Sintaksisni ajratib ko'rsatish:** Python, HTML, CSS, JavaScript kodini o'qishni osonlashtiradi.
   * **Avtomatik to'ldirish (IntelliSense):** Dasturlash tilining sintaksisi va kutubxonalari bo'yicha takliflar beradi, bu esa kod yozish tezligini oshiradi va xatolarni kamaytiradi. Misol uchun, Django modellarini import qilishda inventory.models.inventory kabi yo'llarni avtomatik taklif qiladi.
   * **Loyihada navigatsiya:** Loyiha tuzilmasi (mikroservislar, ichki papkalar) bo'yicha oson harakatlanish imkonini beradi. docker-compose.yml orqali bir necha alohida servislar mavjudligi hisobga olinsa, bu xususiyat juda foydalidir.
2. **Xatolarni tuzatish (Debugging):**
   * **Breakpoints:** Kodning istalgan joyiga to'xtash nuqtalarini qo'yish imkoniyati.
   * **Qadam-baqadam bajarish (Step-through):** Kodni qatorma-qator bajarish va har bir bosqichda o'zgaruvchilar qiymatini kuzatish.
   * **O'zgaruvchilarni kuzatish (Watch):** Ma'lum o'zgaruvchilarning qiymatini real vaqt rejimida ko'rish.
   * VS Code'ning .vscode/settings.json faylida python.testing.unittestEnabled kabi sozlamalar mavjudligi testlash jarayonini ham IDE orqali boshqarishni ko'rsatadi.
3. **Versiyalarni boshqarish integratsiyasi (Git Integration):**
   * Koddagi o'zgarishlarni bevosita IDE ichida kuzatish, commit qilish, branchlarni boshqarish. Bu jamoaviy ishlashda va kod tarixini saqlashda juda muhimdir. Loyihada .gitignore fayli mavjudligi versiya nazoratining ishlatilishini tasdiqlaydi.
4. **Terminal integratsiyasi:**
   * Buyruq qatorini IDE ichida to'g'ridan-to'g'ri ishga tushirish imkoniyati. Bu Django migratsiyalarini (python manage.py makemigrations, python manage.py migrate) yoki Docker Compose buyruqlarini (docker-compose up --build) bajarishda qulaylik yaratadi.
5. **Plaginlar va kengaytmalar (Extensions):**
   * Python kengaytmasi, Docker, GitLens kabi plaginlar rivojlanish jarayonini yanada optimallashtirishga yordam beradi.

### D3. Ilovalarni rivojlantirishda IDE'dan foydalanishni IDE'dan foydalanmaslik bilan solishtirish

**IDE'dan foydalanishning afzalliklari (SCM-IDSS kabi loyiha uchun):**

* **Samaradorlikni oshirish:** Yuqorida sanab o'tilgan avtomatik to'ldirish, sintaksisni ajratib ko'rsatish, refactoring kabi xususiyatlar kod yozish va qayta ishlashni tezlashtiradi. SCM-IDSS kabi ko'p faylli, ko'p servisli loyihada fayllar va funksiyalar orasida tezkor o'tish juda muhim.
* **Xatolarni kamaytirish:** Real vaqt rejimida sintaksis va ba'zi mantiqiy xatolarni aniqlash dasturchiga muammolarni erta bosqichda bartaraf etishga yordam beradi. Bu ayniqsa settings.py kabi murakkab konfiguratsiya fayllari bilan ishlashda qo'l keladi.
* **Disk raskadrovka qulayligi:** Integratsiyalashgan disk raskadrovka vositalari (yuqorida aytib o'tilganidek) murakkab jarayonlarda (masalan, RabbitMQ xabar almashinuvi yoki Celery tasklar) xatolarni topish va tuzatishni sezilarli darajada soddalashtiradi.
* **Loyihani boshqarish:** Bir IDE'da butun loyihani (shu jumladan frontend HTML/JS va bir nechta backend servislarni) ko'rish va boshqarish imkoniyati dasturchiga umumiy tasavvurni beradi.
* **Integratsiyalashgan muhit:** Git, terminal, linterlar kabi vositalarning bir joyda to'planishi dasturchining turli ilovalar orasida almashishiga bo'lgan ehtiyojni kamaytiradi.

**IDE'dan foydalanmaslik (oddiy matn muharriri) kamchiliklari:**

* **Samaradorlikning pasayishi:** Avtomatik to'ldirish, sintaksis tekshiruvi va refactoring vositalarisiz kod yozish ancha sekin kechadi va xatolarga moyil bo'ladi.
* **Disk raskadrovka murakkabligi:** Disk raskadrovka qilish uchun alohida vositalar (masalan, pdb Python uchun) yoki print funksiyalaridan foydalanish zarurati yuzaga keladi, bu esa vaqtni oladi va jarayonni qiyinlashtiradi.
* **Loyihani boshqarish qiyinligi:** Katta va murakkab loyihalarda (SCM-IDSS kabi, 5+ mikroservis bilan) fayllar va kataloglar orasida qo'lda navigatsiya qilish juda samarasiz.
* **Konfiguratsiyani qo'lda boshqarish:** Har bir til, ramka va kutubxona uchun sozlamalarni (masalan, requirements.txt dagi bog'liqliklarni) qo'lda boshqarish va konfiguratsiya qilish talab qilinadi, bu esa xatoliklarga olib kelishi mumkin.
* **Jamoaviy ishlashda qiyinchilik:** Turli dasturchilar turli xil sozlamalar va formatlash usullarini qo'llashi natijasida kod integratsiyasi muammolari yuzaga kelishi mumkin.

**:** SCM-IDSS kabi keng qamrovli va mikroservis arxitekturasi asosidagi loyiha uchun IDE'dan foydalanish mutlaqo zarurdir. U dasturiy ta'minotni ishlab chiqishning har bir bosqichida dasturchi samaradorligini, kod sifatini va umumiy loyiha boshqaruvini sezilarli darajada yaxshilaydi. Oddiy matn muharriri faqat juda kichik va shaxsiy skriptlar uchun mos keladi, jiddiy dasturiy ta'minot mahsulotlarini yaratishda esa IDE ajralmas vosita hisoblanadi.

### P4. Nosozliklarni tuzatish jarayonini va IDE'da mavjud bo'lgan vositalarni tushuntiring.

**Nosozliklarni tuzatish (Debugging) jarayoni:**

Dasturiy ta'minotni ishlab chiqishda nosozliklar yoki "bug"lar (xatolar) muqarrar. Nosozliklarni tuzatish (debugging) — bu dasturdagi xatolarni aniqlash, ularning sababini tushunish va ularni bartaraf etish jarayonidir. Bu shunchaki xatoni topishdan ko'ra ko'proq narsani anglatadi; bu dasturning qanday ishlashini, qanday ma'lumotlar oqimini va kutilmagan vaziyatlarga qanday munosabatda bo'lishini chuqur tushunishdir. SCM-IDSS kabi mikroservislar arxitekturasi asosida qurilgan murakkab tizimlarda nosozliklarni tuzatish jarayoni ayniqsa qiyin bo'lishi mumkin, chunki xato bir serviste yuzaga kelib, boshqa bir servisning xatti-harakatiga ta'sir qilishi mumkin.

Nosozliklarni tuzatish jarayoni odatda quyidagi bosqichlarni o'z ichiga oladi:

1. **Nosozlikni qayta tiklash (Reproduce the Bug):** Bu eng muhim birinchi qadam. Agar nosozlikni doimiy ravishda qayta tiklash mumkin bo'lmasa, uni aniqlash va tuzatish juda qiyin bo'ladi. Bu bosqichda dastur qanday sharoitlarda xato berayotganini aniqlash, foydalanuvchi qanday qadamlarni bajarganini, qanday ma'lumotlar kiritilganini aniqlash zarur. SCM-IDSS da, masalan, "yangi buyurtma yaratishda inventar yangilanmadi" degan xato bo'lsa, xatoni keltirib chiqargan aniq mahsulot ID'lari, miqdorlar va buyurtma berish ketma-ketligini qayta tiklashga uriniladi.
2. **Nosozlikni aniqlash va joylashish (Locate/Isolate the Bug):** Nosozlikni qayta tiklagandan so'ng, uning kodning qaysi qismida joylashganligini aniqlash kerak. Bu bosqichda print() funksiyalari (Python'da) yoki logger.info() kabi logging mexanizmlaridan foydalanib ma'lumotlarni chiqarish, shuningdek, IDE'ning nosozliklarni tuzatish vositalaridan foydalanish mumkin. SCM-IDSS'da bu ayniqsa murakkab, chunki xabar RabbitMQ orqali bir necha servislarga tarqalishi mumkin. Xabar qaysi serviste (Order Service, Inventory Service, Logistics Service) noto'g'ri qayta ishlanganini aniqlash uchun har bir servisning loglarini tekshirish va ma'lumotlar oqimini kuzatish talab qilinadi.
3. **Sababini tahlil qilish (Analyze the Cause):** Nosozlik joylashgan kod qismi aniqlangach, xato nimaga sabab bo'layotganini tushunish kerak. Bu o'zgaruvchilarning kutilmagan qiymatlari, noto'g'ri mantiq, shartlar, cheklovlar yoki hatto tashqi tizimlardan keladigan ma'lumotlardagi xatolar bo'lishi mumkin. Masalan, inventar kamaytirilmayotgan bo'lsa, quantity\_to\_decrease noto'g'ri qiymatga ega bo'lganmi, yoki inventory\_item obyekt topilmaganmi, tekshiriladi.
4. **Tuzatishni amalga oshirish (Fix the Bug):** Nosozlikning sababi tushunilgach, uni bartaraf etish uchun kodga o'zgarishlar kiritiladi. Bu kichik mantiqiy tuzatish, yangi shart qo'shish, ma'lumotlarni tekshirish yoki hatto arxitekturani o'zgartirish bo'lishi mumkin.
5. **Tuzatishni tekshirish (Verify the Fix):** Tuzatish kiritilgandan so'ng, nosozlikning haqiqatan ham bartaraf etilganligini tekshirish juda muhimdir. Bu nosozlikni qayta tiklash bosqichida ishlatilgan bir xil qadamlarni takrorlash, shuningdek, avtomatlashtirilgan testlarni (unit tests, integration tests) qayta ishga tushirish orqali amalga oshiriladi.
6. **Regressiya testlari (Regression Testing):** Tuzatish boshqa funksionalliklarga salbiy ta'sir ko'rsatmaganligiga ishonch hosil qilish uchun to'liq testlar to'plami bajariladi.

**IDE'da mavjud bo'lgan nosozliklarni tuzatish vositalari (Visual Studio Code misolida):**

Visual Studio Code (VS Code) kabi zamonaviy IDE'lar nosozliklarni tuzatishni sezilarli darajada osonlashtiradigan kuchli vositalarni o'z ichiga oladi. SCM-IDSS kabi Python/Django loyihalarida bu vositalar juda foydali bo'ladi:

* **Breakpoints (To'xtash nuqtalari):** Kodning istalgan qatoriga to'xtash nuqtalari o'rnatish imkonini beradi. Dastur shu nuqtaga etib kelganida bajarilish to'xtatiladi va dasturchi o'zgaruvchilarni tekshirishi, bajarilish oqimini qadam-baqadam kuzatishi mumkin.
  + **SCM-IDSS misoli:** inventory\_service/inventory/management/commands/run\_consumer.py faylidagi process\_inventory\_event funksiyasining if event\_type == 'DECREASE\_STOCK': qatoriga breakpoint qo'yib, kiruvchi xabarning tarkibini (data o'zgaruvchisi) va order\_itemsni tekshirish mumkin.
* **Step-over (Qadam tashlash), Step-into (Ichkariga kirish), Step-out (Tashqariga chiqish):**
  + **Step-over (F10):** Hozirgi qatorni bajaradi va keyingi qatorga o'tadi, agar joriy qatorda funksiya chaqiruvi bo'lsa, u funksiyaga "kirib ketmaydi", balki uni bajarib bo'lgach keyingi qatorga o'tadi.
  + **Step-into (F11):** Agar joriy qatorda funksiya chaqiruvi bo'lsa, funksiya ichiga kirib, uning bajarilishini qadam-baqadam kuzatish imkonini beradi.
  + **Step-out (Shift+F11):** Joriy funksiya bajarilishini tugatib, uning chaqirilgan joyidagi keyingi qatorga qaytadi.
  + **SCM-IDSS misoli:** process\_inventory\_event ichida inventory\_item.save() qatorida Step-into qilib, Django ORM'ning ma'lumotlar bazasiga saqlash jarayonini yoki transaction.atomic() blokining qanday ishlashini kuzatish mumkin.
* **O'zgaruvchilarni tekshirish (Variable Inspection):** Dastur to'xtatilgan nuqtada barcha lokal va global o'zgaruvchilarning joriy qiymatlarini ko'rish imkonini beradi. Bu xatoga sabab bo'layotgan noto'g'ri qiymatlarni aniqlashda juda muhim.
  + **SCM-IDSS misoli:** quantity\_to\_decrease ning qiymati kutilganidek butun sonmi yoki noto'g'ri formatdagi matnmi, tekshirish mumkin. Shuningdek, inventory\_item.quantity ning har bir qadamdan keyingi holatini kuzatish mumkin.
* **Chaqiruv steki (Call Stack):** Joriy bajarilish nuqtasiga qanday funksiya chaqiruvlari ketma-ketligi orqali etib kelinganini ko'rsatadi. Bu, ayniqsa, bir funksiya boshqa bir funksiyani, u esa uchinchisini chaqirganda mantiqiy xatolarni topishda yordam beradi.
* **Shartli to'xtash nuqtalari (Conditional Breakpoints):** Faqat ma'lum bir shart bajarilganda ishga tushadigan to'xtash nuqtalari. Masalan, faqatgina product\_id == 'PRODUCT123' bo'lganda breakpoint ishga tushsin deyish mumkin. Bu katta ma'lumotlar oqimida ma'lum bir muammoli holatni izlashda samarali.
* **Logpoints:** Kodga print qatorlarini qo'shmasdan, faqatgina log xabarlarini chiqarish uchun to'xtash nuqtalari. Bu kodni o'zgartirmasdan, uning bajarilishini kuzatish imkonini beradi.
* **Uzoqdan nosozliklarni tuzatish (Remote Debugging):** SCM-IDSS loyihasi Docker konteynerlarida ishlayotganligi sababli, VS Code kabi IDE'lar uzoqdan (remote) nosozliklarni tuzatish imkoniyatini beradi. Bu konteyner ichida ishlayotgan dasturga ulanib, yuqorida sanab o'tilgan barcha nosozliklarni tuzatish vositalaridan foydalanishni anglatadi. Bu docker-compose.yml faylida belgilangan har bir servisni mustaqil ravishda nosozliklardan tozalash imkonini beradi.

### P5. Kodingizda ishlatgan kodlash standartini bayon qiling.

Dasturiy ta'minot kodlash standarti — bu kodni yozishda rioya qilinishi kerak bo'lgan qoidalar va ko'rsatmalar to'plami. Ular kodning izchilligini, o'qish qulayligini va texnik xizmat ko'rsatishini ta'minlash uchun xizmat qiladi. SCM-IDSS loyihasi Python tilida yozilganligi sababli, biz **PEP 8** (Python Enhancement Proposal 8) kodlash standartiga amal qildik. Bu Python hamjamiyatida keng tarqalgan va de-fakto standart hisoblanadi.

**PEP 8 ning asosiy jihatlari va SCM-IDSS loyihasidagi amalga oshirilishi:**

1. **Indentation (Chekinish):** Kod bloklari har doim **4 ta bo'sh joy** bilan chekinadi. Bu Python tilining asosiy talablaridan biri bo'lib, o'qish qulayligini sezilarli darajada oshiradi.
   * **SCM-IDSS misoli:** Loyihaning barcha Python fayllarida, masalan, inventory\_service/inventory/management/commands/run\_consumer.py dagi process\_inventory\_event funksiyasi ichidagi if sharti va for sikli ostidagi barcha kod qatorlari 4 ta bo'sh joy bilan chekinilgan.
2. **Line Length (Qator uzunligi):** Kod qatorlarining maksimal uzunligi **79 ta belgi** bilan cheklanadi. Bu kodni o'qishni va bir vaqtning o'zida bir nechta faylni yonma-yon ko'rishni osonlashtiradi. Garchi ba'zi loyihalarda 99 belgiga qadar kengaytirilishiga ruxsat berilsa-da, asosiy prinsip — qatorlarni qisqa tutishdir.
3. **Naming Conventions (Nomlash qoidalari):**
   * **Modullar (fayllar) va paketlar:** Qisqa, barcha harflari kichik (lowercase) va so'zlar orasida pastki chiziq (\_) bilan ajratilgan (snake\_case). Masalan: inventory\_service/inventory/models.py, order\_service/orders/producer.py.
   * **Klasslar:** So'zning har bir bosh harfi katta qilib yoziladi (PascalCase). Masalan: Inventory, Customer\_orders, BaseCommand.
   * **Funksiyalar va o'zgaruvchilar:** Barcha harflari kichik va so'zlar orasida pastki chiziq bilan ajratilgan (snake\_case). Masalan: process\_inventory\_event, product\_id, quantity\_to\_decrease.
   * **Konstantalar:** Barcha harflari katta va so'zlar orasida pastki chiziq bilan ajratilgan (UPPER\_CASE). Loyiha ichida aniq konstantalar ko'rsatilgan fayl mavjud emas, lekin os.getenv('CELERY\_BROKER\_URL') kabi environment variable nomlari shu qoidaga yaqin.
4. **Blank Lines (Bo'sh qatorlar):** Funksiyalar va klasslar orasida ikki bo'sh qator ishlatiladi. Metodlar orasida bitta bo'sh qator. Bu kodning vizual ajralishini va o'qish qulayligini ta'minlaydi.
5. **Imports (Importlar):** Import qilingan modullar alohida qatorlarda yoziladi va odatda standart kutubxonalar, uchinchi tomon kutubxonalari va mahalliy ilova importlari bo'yicha guruhlanadi. Bu importlar tartibini saqlaydi.
   * **SCM-IDSS misoli:** run\_consumer.py faylida pika, json, os kabi standart importlar, keyin django importlari va nihoyat inventory.models kabi mahalliy importlar tartiblangan.
6. **Whitespace (Bo'sh joylar):** Operatorlar atrofida, verguldan keyin, funksiya argumentlari orasida va boshqa joylarda to'g'ri bo'sh joylardan foydalanish kodni tartibli qiladi.
7. **Comments (Izohlar):** Izohlar kodning nima qilayotganini emas, balki **nima uchun** qilayotganini tushuntirishga qaratilgan bo'lishi kerak. Keraksiz, o'z-o'zidan tushunarli kodni izohlashdan qochish tavsiya etiladi.

**Kodlash standartini qo'llash vositalari:** SCM-IDSS loyihasida PEP 8 standartiga rioya qilish uchun **linterlar (masalan, Flake8, Pylint)** va **kod formatlovchilar (masalan, Black, isort)** kabi avtomatlashtirilgan vositalar VS Code IDE'si bilan integratsiya qilingan bo'lishi mumkin. Bu dasturchi kodni yozganda yoki saqlaganda avtomatik ravishda standartlarga rioya qilinishini ta'minlaydi.

### M4. Nosozliklarni tuzatish jarayoni xavfsizroq, mustahkamroq ilovalarni ishlab chiqishga qanday yordam berishini baholang.

Nosozliklarni tuzatish (debugging) shunchaki xatolarni topish va tuzatishdan iborat emas; bu dasturiy ta'minotning sifatini, xavfsizligini va mustahkamligini sezilarli darajada yaxshilaydigan proaktiv jarayondir.

**Xavfsizroq ilovalarni ishlab chiqishda debuggingning roli:**

1. **Kiritilgan ma'lumotlarni tekshirish (Input Validation):** Debugging jarayonida dasturchi foydalanuvchi kiritadigan ma'lumotlarning (masalan, buyurtma miqdori, mahsulot ID'si) dasturga qanday kirib kelayotganini va ularning mantiqiy bloklarda qanday qayta ishlanayotganini kuzatishi mumkin.
   * **Xavfsizlik muammosi:** Agar kiritilgan ma'lumotlar to'g'ri tekshirilmasa (masalan, faqat raqamlar qabul qilinadigan joyga matn kiritilsa), bu **SQL injection**, **Cross-Site Scripting (XSS)** yoki boshqa **injection hujumlariga** olib kelishi mumkin.
   * **Debugging yordami:** Debugger yordamida har bir kirish nuqtasida o'zgaruvchilar qiymatini tekshirib, noto'g'ri yoki xavfli kiritmalarning kutilmagan xatti-harakatlarga olib kelmasligiga ishonch hosil qilish mumkin. inventory\_service/inventory/management/commands/run\_consumer.py dagi int(quantity\_to\_decrease) konvertatsiyasi va uning atrofidagi xato qidirish (try-except) bunga misol bo'la oladi.
2. **Avtorizatsiya va Autentifikatsiya mantiqini tekshirish:**
   * **Xavfsizlik muammosi:** Agar foydalanuvchining rollari yoki huquqlari noto'g'ri boshqarilsa, ruxsatsiz foydalanuvchilar maxfiy ma'lumotlarga kirishi yoki ruxsat etilmagan amallarni bajarishi mumkin (vertikal yoki gorizontal privilegiya eskalatsiyasi). Misol uchun, oddiy "Inventar menejeri" bo'lgan foydalanuvchi "Ta'minot zanjiri rahbarlari" uchun mo'ljallangan strategik hisobotga kira olmasligi kerak.
   * **Debugging yordami:** api/permissions.py fayllaridagi ruxsatlarni tekshirish mantiqidan Step-through qilib, har bir foydalanuvchi turi uchun dasturning qanday ishlashini sinash va tasdiqlash mumkin. Bu has\_permission yoki has\_object\_permission metodlarida tekshirish jarayonida aniq rol va huquqlarning to'g'ri qo'llanilishini kuzatishni o'z ichiga oladi.
3. **Ma'lumotlar sizib chiqishini aniqlash (Data Leakage):** Debugging yordamida dastur ishlayotganda qaysi ma'lumotlarning qayta ishlanayotganini va tarmoq orqali yuborilayotganini ko'rish mumkin.
   * **Xavfsizlik muammosi:** Agar maxfiy ma'lumotlar (masalan, yetkazib beruvchining shartnoma shartlari, foydalanuvchi parollari) server loglarida yoki API javoblarida tasodifan oshkor bo'lsa, bu jiddiy xavfsizlikka tahdid soladi.
   * **Debugging yordami:** Serializerlar (api/Order/serializers.py) yoki ViewSetlar (api/Order/views.py) orqali ma'lumotlar yuborilishidan oldin ularning tarkibini tekshirib, faqat ruxsat etilgan ma'lumotlarning tarqatilishini ta'minlash mumkin.

**Mustahkamroq ilovalarni ishlab chiqishda debuggingning roli:**

1. **Chegaraviy holatlarni boshqarish (Edge Case Handling):** Debugging oddiy ishlash jarayonida yuzaga kelmaydigan, lekin kamdan-kam holatlarda (masalan, nol miqdor, juda katta sonlar, bo'sh satrlar, kutilmagan belgi kodlash) yuzaga keladigan xatolarni aniqlashda yordam beradi.
   * **SCM-IDSS misoli:** DECREASE\_STOCK algoritmini tekshirganda, agar quantity\_to\_decrease nol yoki manfiy bo'lsa, tizim qanday ishlashini debuggerda kuzatish mumkin. Agar dastur bunga tayyor bo'lmasa, tegishli tekshiruvlar (if quantity\_to\_decrease <= 0:) qo'shilishi mumkin. Shuningdek, inventory\_item.quantity nolga tushganda yoki manfiy bo'lib qolsa, tizimning munosabati tekshiriladi.
2. **Resurslarni boshqarish (Resource Management):** Uzoq vaqt ishlaydigan dasturlarda (masalan, RabbitMQ iste'molchilari yoki Celery workerlari), xotira sizib chiqishini (memory leaks) yoki yopilmagan ma'lumotlar bazasi ulanishlarini (close\_old\_connections() kabi) aniqlashda debugging yordam beradi. Bu tizimning barqarorligini va uzluksiz ishlashini ta'minlaydi.
3. **Konkurrensiy muammolari (Concurrency Issues):** SCM-IDSS kabi distributiv tizimlarda bir vaqtning o'zida bir nechta jarayonlar yoki tredlar ma'lumotlarga kirishga urinishi mumkin. Debugging (ayniqsa, lok (locks) yoki semaforlar (semaphores) bilan) poyga shartlari (race conditions) va o'lik qulf (deadlocks) kabi qiyin topiladigan xatolarni aniqlashda yordam beradi. transaction.atomic() kabi Django vositalari ma'lumotlar bazasi darajasida atomiklikni ta'minlaydi, ammo kompleks distributiv logikada debugging hali ham muhim.
4. **Kutilmagan xatolarni boshqarish (Handling Unexpected Errors):** Dasturda har doim ham barcha istisnolarni oldindan bilib bo'lmaydi. Debugging yordamida kutilmagan istisnolar (exceptions) yuzaga kelganda, ularning kelib chiqish sababini tezda aniqlash va tegishli try-except bloklari bilan mustahkam xato ishlovchilarni qo'shish mumkin. Bu dasturning to'satdan ishlamay qolishi o'rniga, xatolarni silliq boshqarishini ta'minlaydi. Masalan, RabbitMQ ulanishidagi xatolarni pika.exceptions.AMQPConnectionError orqali ushlash va qayta ulanishga urinish.

qilib aytganda, debugging dasturchiga o'z kodining ichki ish faoliyatini tushunishga, har qanday potentsial xavfsizlik zaifliklarini yoki barqarorlik muammolarini erta bosqichda aniqlashga imkon beradi. Bu esa yakunda ancha xavfsiz va mustahkam dasturiy mahsulot yaratilishiga olib keladi.

### D4. Kodlash standarti jamoada ham, shaxs uchun ham nima uchun zarurligini tanqidiy baholang.

Kodlash standarti dasturiy ta'minotni ishlab chiqishda nafaqat foydali, balki zamonaviy va murakkab loyihalar (SCM-IDSS kabi) uchun zarur bo'lgan asosiy komponent hisoblanadi. Uning ahamiyatini jamoaviy va individual nuqtai nazardan tanqidiy baholaylik.

**Jamoa uchun kodlash standartining zarurligi:**

1. **Kodning izchilligi (Consistency):**
   * **Baholash:** Jamoa a'zolari turli xil dasturlash uslublariga ega bo'lishi mumkin. Kodlash standarti barcha kodning bir xil formatda (masalan, PEP 8 ga muvofiq 4 bo'sh joyli chekinish, snake\_case o'zgaruvchi nomlari) yozilishini ta'minlaydi. Izchil kod osonroq o'qiladi, tushuniladi va unga xizmat ko'rsatiladi. SCM-IDSS kabi bir nechta mikroservislardan iborat loyihada turli jamoalar yoki shaxslar har xil servislarda ishlashi mumkin. Standart bo'lmasa, har bir servis o'z uslubida bo'lib, umumiy tizimni boshqarish qiyinlashadi.
   * **Tanqidiy jihat:** Boshida jamoa a'zolari uchun yangi standartlarga moslashish biroz qiyin bo'lishi mumkin, ammo uzoq muddatda bu investitsiya o'zini oqlaydi.
2. **O'qish qulayligi va tushunarliligi (Readability & Understandability):**
   * **Baholash:** Agar kod izchil bo'lsa, jamoaning har bir a'zosi boshqa birov tomonidan yozilgan kodni tezda tushunishi mumkin. Kodni o'qish va tushunish uni yozishdan ko'ra ko'proq vaqt oladi. PEP 8 kabi standartlar kodni inson tomonidan o'qishga qulay qilishga qaratilgan. Bu SCM-IDSS'ning modullarida (models.py, views.py) aniq namoyon bo'ladi.
   * **Tanqidiy jihat:** Ba'zan qat'iy standartlar ijodkorlikni cheklash yoki kodni yanada murakkablashtirish kabi tuyulishi mumkin, lekin aslida ular struktura berib, murakkablikni kamaytiradi.
3. **Texnik xizmat ko'rsatishning osonligi (Maintainability):**
   * **Baholash:** O'qishga qulay va izchil kodni texnik xizmat ko'rsatish, yangi funksiyalar qo'shish yoki xatolarni tuzatish osonroq. Loyiha uzoq muddatli bo'lganda (SCM-IDSS kabi), unga yangi dasturchilar qo'shilishi yoki eski dasturchilar ketishi mumkin. Standartlar yangi kelganlarga loyihaga tezroq kirishishga yordam beradi.
   * **Tanqidiy jihat:** Dastlab, kodlash standartlariga rioya qilish qo'shimcha vaqt talab qilishi mumkin, ammo uzoq muddatda bu texnik xizmat ko'rsatishga sarflanadigan vaqtni qisqartiradi.
4. **Kod tekshiruvlari samaradorligi (Efficient Code Reviews):**
   * **Baholash:** Kodlash standarti mavjud bo'lganda, kod tekshiruvchilari (code reviewers) formatlash yoki nomlash kabi asosiy muammolarga e'tibor qaratish o'rniga, kodning mantiqiy xatolarini yoki arxitektura muammolarini ko'rib chiqishga ko'proq vaqt ajrata oladi.
5. **Hamkorlikni yaxshilash (Improved Collaboration):**
   * **Baholash:** Bir xil standartlarga rioya qilish jamoa a'zolari o'rtasida samarali hamkorlikni ta'minlaydi. Merge (birlashtirish) konfliktlari kamayadi, chunki har kim bir xil uslubda yozadi. Bu ayniqsa git va GitHub (loyihada .gitignore mavjudligi versiya nazoratini ko'rsatadi) kabi versiya nazorati tizimlari bilan ishlashda muhimdir.

**Shaxs (individual) uchun kodlash standartining zarurligi:**

1. **Eng yaxshi amaliyotlarni o'rganish (Learning Best Practices):**
   * **Baholash:** Kodlash standartlari ko'pincha til yoki ramka (framework) uchun eng yaxshi amaliyotlarni o'zida mujassam etadi. Ularga rioya qilish orqali dasturchi o'z mahoratini oshiradi va professional tarzda kod yozishni o'rganadi.
2. **O'z kodini qayta o'qish qulayligi (Easier Self-Review):**
   * **Baholash:** Vaqt o'tishi bilan dasturchi o'zi yozgan kodni ham unutishi mumkin. Standartlarga rioya qilingan kodni bir necha oydan keyin ham o'qish va tushunish oson bo'ladi.
3. **Intizom va professionalizm (Discipline & Professionalism):**
   * **Baholash:** Kodlash standartlariga rioya qilish dasturchida intizomni shakllantiradi va uning professionalizmini oshiradi. Bu kelajakda boshqa loyihalarda ishlashda ham foydali bo'ladi.
4. **Ishga qabul qilinish imkoniyatini oshirish:**
   * **Baholash:** Ko'pgina kompaniyalar o'zlarining kodlash standartlariga ega. Agar dasturchi umumiy qabul qilingan standartlarga rioya qilishni bilsa, bu uning ishga qabul qilinish imkoniyatini oshiradi.

**Tanqidiy baholash – muvozanatni topish:**

Garchi kodlash standartlarining afzalliklari ko'p bo'lsa-da, ularni haddan tashqari qat'iy qo'llash ba'zi kamchiliklarga ham ega bo'lishi mumkin:

* **Dastlabki vaqt va harakat (Initial Overhead):** Yangi dasturchilar yoki mavjud standartlarga moslashmagan jamoa a'zolari uchun dastlabki bosqichda ko'proq vaqt va harakat talab qilinishi mumkin. Standartlarni o'rganish va ularga rioya qilish uchun ma'lum bir o'zgarishga moslashish kerak.
* **Haddan tashqari qat'iylik (Over-prescription):** Ba'zan standartlar haddan tashqari batafsil bo'lishi yoki mantiqsiz qoidalarni o'z ichiga olishi mumkin, bu esa dasturchining ijodkorligini cheklab, samaradorlikni pasaytirishi mumkin. Muhimi, loyiha va jamoaning ehtiyojlariga mos keladigan oqilona va moslashuvchan standartni tanlashdir.

SCM-IDSS kabi murakkab va jamoaviy loyiha uchun kodlash standartlari mutlaqo zarurdir. Ular kodning izchilligini, o'qish qulayligini va texnik xizmat ko'rsatishini ta'minlaydi, bu esa uzoq muddatda ishlab chiqish xarajatlarini kamaytiradi va umumiy dasturiy ta'minot sifatini oshiradi. Individual dasturchi uchun esa bu professional o'sishga yordam beradigan, intizomni oshiradigan va kelajakdagi loyihalarda samarali ishlash uchun asos yaratadigan muhim vositadir. Standartlarsiz kod xaosga olib kelishi mumkin, bu esa katta loyihalarni boshqarib bo'lmas holga keltiradi.

## Xulosa

Ushbu topshiriqning muvaffaqiyatli yakunlanishi bizga SCM-IDSS (Ta’minot Zanjirini Boshqarishning Intellektual Qaror Qabul Qilishni Qo‘llab-Quvvatlash Tizimi) kabi murakkab va intellektual ilovalarni ishlab chiqish jarayoni bo'yicha keng qamrovli tushuncha berdi. Hisobot davomida biz dasturlash fanining turli asosiy jihatlarini, ularning nazariy asoslarini va amaliy qo'llanilishini chuqur o'rgandik.

**LO1** doirasida biz dasturiy ta'minotning asosini tashkil etuvchi algoritmlar nima ekanligini, ularning turlarini va real tizimlarda qanday qo'llanilishini aniqladik. SCM-IDSS'ning Talabni Prognozlash, Inventarni Optimallashtirish va Logistikani Optimallashtirish kabi asosiy funksiyalari uchun ishlatiladigan algoritmlarni tahlil qildik. Shuningdek, ilovalarni yaratishning standart bosqichlarini, ya'ni SDLC (Software Development Life Cycle) bosqichlarini, xususan, Agile metodologiyasini loyihamiz misolida tushuntirdik. Kod yozishdan to bajarishgacha bo'lgan har bir bosqich, shu jumladan Docker konteynerizatsiyasi va RabbitMQ orqali mikroservislar aloqasi, batafsil yoritildi.

**LO2** da biz dasturlashning uchta asosiy paradigmasi – protsessual, ob'ektga yo'naltirilgan (OOP) va voqea-drayv (event-driven) – ni batafsil o'rgandik. Har bir paradigmaning o'ziga xos xususiyatlari, afzalliklari va kamchiliklari tahlil qilindi. SCM-IDSS loyihasining Python/Django backendida OOP tamoyillari (modellar, ViewSetlar) va mikroservislar aloqasida voqea-drayv yondashuvi (RabbitMQ iste'molchilari orqali) qanday integratsiyalashgani ko'rsatildi. Bu paradigmalarning muvaffaqiyatli kombinatsiyasi SCM-IDSS kabi murakkab distributiv tizimning kengaytirilishi va samaradorligini ta'minlaydi.

**LO3** bo'yicha biz integratsiyalashgan rivojlanish muhiti (IDE) ning dasturiy ta'minotni ishlab chiqishdagi muhim rolini baholadik. Visual Studio Code kabi IDE'lar kod yozishdan tortib, uni boshqarish, nosozliklardan tozalash va hatto versiyalarni nazorat qilishgacha bo'lgan jarayonlarni qanday osonlashtirishini ko'rsatdik. Loyiha doirasidagi inventar miqdorini kamaytirish algoritmining kod parchasi IDE yordamida qanday ishlab chiqilganligini tasvirladik va IDE'dan foydalanishning oddiy matn muharririga nisbatan afzalliklarini tahlil qildik.

Nihoyat, **LO4** da nosozliklarni tuzatish (debugging) jarayoni va kodlash standartlarining ahamiyatini batafsil ko'rib chiqdik. Biz debugging jarayonining bosqichlarini, VS Code'dagi debugging vositalarini (breakpoints, step-over/into/out, variable inspection) tushuntirdik. Eng muhimi, debugging nafaqat xatolarni tuzatish, balki ilovalarni yanada **xavfsizroq** (masalan, kiritilgan ma'lumotlarni tekshirish, avtorizatsiya mantiqini tasdiqlash orqali) va **mustahkamroq** (masalan, chegaraviy holatlarni boshqarish, resurslarni samarali boshqarish, konkurrensiya muammolarini aniqlash orqali) qilishga qanday yordam berishini baholadik. PEP 8 kabi kodlash standartlarining jamoa (izchillik, o'qish qulayligi, hamkorlik) va shaxs (intizom, professionalizm) uchun zarurligi tanqidiy baholandi.

Xulosa qilib aytganda, SCM-IDSS loyihasi zamonaviy ta'minot zanjiri boshqaruvini inqilob qilish uchun AI/ML va ilg'or tahlil texnologiyalarini birlashtiruvchi juda murakkab tizimdir. Ushbu topshiriq orqali olingan bilimlar dasturlash tushunchalarini chuqur o'zlashtirish, ilg'or dasturiy ta'minot arxitekturalarini loyihalash va amalga oshirish, shuningdek, yuqori sifatli, xavfsiz va mustahkam ilovalarni yaratish ko'nikmalarini rivojlantirish uchun mustahkam asos bo'ldi. SCM-IDSS kelajakdagi intellektual qaror qabul qilish tizimlariga misol bo'la oladi.

## Adabiyotlar Ro'yxati

1. Turban, E., Aronson, J.E. and Liang, T.P. (2005) *Decision Support Systems and Intelligent Systems*. 7th edn. Pearson/Prentice Hall. [Online] Available at: <https://books.google.com/books/about/Decision_Support_Systems_and_Intelligent.html?hl=id&id=NfMJAQAAMAAJ> (Accessed: 30 June 2025).
2. Li, G. (2022) 'Supply Chain Efficiency and Effectiveness Management Using Decision Support Systems', *International Journal of Information Systems and Supply Chain Management (IJISSCM)*, 15(4), pp.1-18. [Online] Available at: <https://www.igi-global.com/article/supply-chain-efficiency-and-effectiveness-management-using-decision-support-systems/305847> (Accessed: 30 June 2025).
3. Baryannis, G. et al. (2018) 'Decision Support Systems and Artificial Intelligence in Supply Chain Risk Management'. In: *Revisiting Supply Chain Risk*. Springer, pp.53-71. [Online] Available at: <https://www.researchgate.net/publication/329784099_Decision_Support_Systems_and_Artificial_Intelligence_in_Supply_Chain_Risk_Management> (Accessed: 30 June 2025).
4. Pandey, D., Pandey, B.K. and Kanike, U.K. (eds.) (2024) *Utilization of AI Technology in Supply Chain Management*. IGI Global. [Online] Available at: <https://www.harvard.com/book/9798369335932> (Accessed: 30 June 2025).
5. Chand, M., Jain, V. and Ajmera, P. (eds.) (2024) *Data-Driven Technologies and Artificial Intelligence in Supply Chain: Tools and Techniques*. Routledge. [Online] Available at: <https://www.routledge.com/Data-Driven-Technologies-and-Artificial-Intelligence-in-Supply-Chain-Tools-and-Techniques/Chand-Jain-Ajmera/p/book/9781032611297> (Accessed: 30 June 2025).
6. Agrawal, N. et al. (2024) *How Machine Learning Will Transform Supply Chain Management*. Harvard Business Review. [Online] Available at: <https://store.hbr.org/product/how-machine-learning-will-transform-supply-chain-management/R2402K> (Accessed: 30 June 2025).
7. Dunning, T. and Friedman, E. (2018) *Machine Learning Logistics*. O'Reilly Media. [Online] Available at: <https://www.oreilly.com/library/view/machine-learning-logistics/9781491997628/> (Accessed: 30 June 2025).
8. Robertson, P.W. (2021) *Supply Chain Analytics: Using Data to Optimise Supply Chain Processes*. Routledge. [Online] Available at: <https://www.routledge.com/Supply-Chain-Analytics-Using-Data-to-Optimise-Supply-Chain-Processes/Robertson/p/book/9780367540067> (Accessed: 30 June 2025).
9. Russell, R.S. and Taylor, B.W. (2023) *Operations and Supply Chain Management*. 11th edn. Wiley. [Online] Available at: <https://www.wiley.com/en-us/Operations+and+Supply+Chain+Management%2C+11th+Edition-p-00381326>
10. Moufaddal, M., Benghabrit, A. and Bouhaddou, I. (2018) '(PDF) Big Data Analytics for Supply Chain Management'. *ResearchGate*. [Online] Available at: <https://www.researchgate.net/publication/323878142_Big_Data_Analytics_for_Supply_Chain_Management> (Accessed: 30 June 2025).