|  |
| --- |
| **Level 4 | Programming** |
| ABDULATIF NURIDDINOV |

**Unit 4:** Level 4 | Programming

**O'qituvchi:** Obid Jurayev

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

**Talaba ID raqami:** 230892

**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:** | 230892 |
| **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 **"Supply Chain Management Intelligent Decision Support System (SCM-IDSS)"** nomli topshiriqning dasturlash jarayonidagi asosiy tamoyillarini o'rganishga qaratilgan. SCM-IDSS an'anaviy ta'minot zanjiri 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'zgartirish uchun mo'ljallangan. Hisobot dasturlashning asosiy tushunchalarini, ilovalarni yaratish jarayonini, kodning bajarilish bosqichlarini hamda algoritmlarning kodda qanday amalga oshirilishini SCM-IDSS loyihasi misolida batafsil yoritib beradi. Bu yerda optimallash (optimization) algoritmlari yoki matematik modellashtirish yondashuvlari, masalan, "Economic Order Quantity (EOQ)" kabi modellar xarajatlarni hisobga olgan holda buyurtma miqdorini aniqlash uchun qo'llanilishi mumkin.

**P1. Algoritm nima ekanligining ta'rifini bering va ilova yaratish jarayonini bayon qiling.**

**Algoritm nima?** Algoritm — bu ma'lum bir vazifani bajarish yoki muammoni hal qilish uchun aniq, cheklangan va qadam-baqadam ko'rsatmalar to'plami. Bu ko'rsatmalar kompyuter tushunadigan tilda yozilishi mumkin bo'lgan, mantiqiy tartiblangan amallardan iboratdir. Har qanday dasturiy ta'minotning asosi algoritmlardir. Algoritmlar bajarilish tartibini, qabul qilinadigan ma'lumotlarni va kutilgan natijani aniq belgilaydi. Ular samaradorlik, ishonchlilik va to'g'rilik bilan tavsiflanadi.

**SCM-IDSS uchun Algoritmlarni aniqlash misollari:** SCM-IDSS loyihasi turli xil murakkab operatsiyalarni bajarish uchun bir nechta asosiy algoritmlardan foydalanadi. Ularning ba'zilari quyidagilar:

* **Talabni Prognozlash Algoritmi:** Bu tizimning "Talabni prognozlashtirish va rejalashtirish" funksiyasi doirasida amalga oshiriladi. Algoritm tarixiy sotuvlar ma'lumotlari, mavsumiy omillar, promo-aksiyalar va bozor tendentsiyalari kabi ko'p omilli ma'lumotlarni tahlil qiladi. U mashinali o'rganish (Machine Learning) modellaridan foydalanib, kelajakdagi mahsulot talabini bashorat qiladi. Masalan, vaqt qatorlari tahlili (Time Series Analysis) yoki regressiya modellaridan foydalaniladi.
* **Inventarni Optimallashtirish Algoritmi:** "Inventar optimizatsiyasi" funksiyasi optimal xavfsizlik zaxiralarini hisoblash, sekin harakatlanuvchi va eskirgan inventarni aniqlash, shuningdek, inventarni joylashuvlar o'rtasida qayta muvozanatlash uchun algoritmlardan foydalanadi. Bu yerda optimallash (optimization) algoritmlari yoki matematik modellashtirish yondashuvlari, masalan, "Economic Order Quantity (EOQ)" kabi modellar xarajatlarni hisobga olgan holda buyurtma miqdorini aniqlash uchun qo'llanilishi mumkin.
* **Yo'nalishni rejalashtirish (Logistika Optimallashtirish) Algoritmi:** "Logistikani optimallashtirish" funksiyasi yetkazib berish marshrutlarini masofa, vaqt va xarajat cheklovlari asosida optimallashtirish uchun algoritmlardan foydalanadi. Bu uchun grafik algoritmlar (masalan, Dijkstra algoritmi eng qisqa yo'lni topish uchun) yoki Operatsion Tadqiqotlar (Operations Research) algoritmlari (masalan, "Traveling Salesperson Problem" variantlari) ishlatilishi mumkin.
* **Inventar miqdorini kamaytirish algoritmi:** Buyurtma yaratilganda inventar miqdorini yangilash jarayoni ham aniq algoritmga asoslanadi. inventory\_service/inventory/management/commands/run\_consumer.py faylida mavjud bo'lgan DECREASE\_STOCK hodisasini qayta ishlash mantiqi bunga misol bo'la oladi.

**Ilova yaratish jarayoni (SDLC - Software Development Life Cycle):** Dasturiy ta'minotni ishlab chiqish jarayoni, odatda, bir necha bosqichdan iborat bo'lib, SCM-IDSS loyihasi **Agile metodologiyasiga** amal qilgan holda rivojlantirilgan. Asosiy bosqichlar quyidagilarni o'z ichiga oladi:

1. **Talablarni yig'ish va tahlil qilish (Requirements Gathering & Analysis):**
   * Bu bosqichda tizimdan nima talab qilinishi aniqlanadi. Foydalanuvchi xususiyatlari (masalan, ta'minot zanjiri rahbarlari, xaridlar bo'yicha menejerlar) va funksional talablar (masalan, real vaqtda inventar kuzatuvi, talab prognozi, qarorlarni qo'llab-quvvatlash) batafsil hujjatlashtiriladi. Bu tizimning barcha manfaatdor tomonlarining ehtiyojlarini tushunishni o'z ichiga oladi.
2. **Dizayn (Design):**
   * Bu bosqichda tizimning umumiy tuzilmasi, komponentlari va ular orasidagi o'zaro aloqalar aniqlanadi. SCM-IDSS uchun **Mikroservislar Arxitekturasi** tanlangan, bu loyihani kichik, mustaqil servislarga (User Service, Inventory Service, Order Service, Logistics Service, Core Service) ajratish imkonini bergan. Ma'lumotlar bazasi sxemalari (orders/models.py, inventory/models.py), API interfeyslari (order\_service/api/Order/views.py), xabar almashinuv mexanizmlari (RabbitMQ) va kesh tizimlari (Redis) kabi texnologiyalar belgilangan.
3. **Implementatsiya (Implementation/Coding):**
   * Dizayn bosqichida qilingan rejalarga asoslanib, dasturchilar kod yozadilar. SCM-IDSS loyihasi Python (Django/Django REST Framework) va JavaScript (HTML/CSS/Bootstrap bilan frontend uchun) yordamida implementatsiya qilingan. Har bir servis o'zining API endpointlariga ega va loyiha papkalari strukturasi order\_service/, inventory\_service/, logistics\_service/ kabi alohida servislarga ajratilgan.
4. **Testlash (Testing):**
   * Yaratilgan kod talablarga javob berishini va xatosiz ishlashini ta'minlash uchun test qilinadi. Bu bosqichda birlik testlari (unit tests), integratsiya testlari (integration tests) va tizim testlari (system tests) o'tkaziladi. SCM-IDSS kabi mikroservisli tizimda servislarning o'zaro ishlashini ta'minlash uchun integratsiya testlari (inventory/tests.py, orders/tests.py) juda muhimdir.
5. **Joylashtirish (Deployment):**
   * Testdan o'tgan dasturiy ta'minot foydalanuvchilar foydalanishi mumkin bo'lgan muhitga joylashtiriladi. SCM-IDSS loyihasi Docker va Docker Compose yordamida konteynerizatsiya qilingan, bu uni turli muhitlarga (masalan, Amazon EC2) osongina joylashtirish imkonini beradi.
6. **Texnik Xizmat ko'rsatish va monitoring (Maintenance & Monitoring):**
   * Dasturiy ta'minot ishga tushirilgandan so'ng, uning ishlashi kuzatiladi (monitoring), xatolar tuzatiladi (bug fixing) va yangi funksiyalar qo'shiladi. Loyihada Celery workerlari (src/src/celery.py) kabi komponentlar fondagi vazifalarni bajarish orqali tizimning barqarorligiga yordam beradi.

**M1. Kod yozishdan to bajarishgacha bo'lgan bosqichlarni aniqlang.**

Dasturiy ta'minotni kod yozishdan to uni yakuniy bajarishgacha bo'lgan jarayon (Execution) SCM-IDSS kabi mikroservis arxitekturasi bilan ishlashda bir nechta muhim bosqichlarni o'z ichiga oladi:

1. **Kod yozish (Development) va Loyihani tuzish:**
   * Dasturchilar Visual Studio Code (VS Code) kabi integratsiyalashgan rivojlanish muhitlaridan (IDE) foydalanib, Python (Django/Django REST Framework) tilida kod yozadilar.
   * Har bir mikroservis (masalan, order\_service, inventory\_service, logistics\_service, user\_service, src - Core Service va Reports Service) alohida Django loyihasi sifatida tuzilgan. Har bir servis o'zining modellariga (orders/models.py, inventory/models.py, logistics/models.py, users/models.py, core/models.py, reports/models.py), API endpointlariga (views.py fayllari) va o'zining bog'liqliklar ro'yxatiga (requirements.txt) ega. Frontend HTML, CSS va JavaScript yordamida alohida yaratiladi.
2. **Ma'lumotlar bazasi migratsiyalari:** Django loyihalarida modellar o'zgarganda, ma'lumotlar bazasi tuzilmasini yangilash uchun migratsiya fayllari (masalan, 0001\_initial.py) generatsiya qilinadi. Bu fayllar python manage.py makemigrations va python manage.py migrate buyruqlari yordamida har bir servis katalogida bajariladi.
3. **Konteynerizatsiya (Docker Images):**
   * Har bir mikroservis o'zining Docker image'iga qadoqlanadi. Dockerfile ichida Python muhiti, loyihaning bog'liqliklari (requirements.txt) o'rnatiladi va Django ilovasi ishga tushirish uchun tayyorlanadi.
4. **Servislarni Orchestratsiya qilish (Docker Compose):**
   * docker-compose.yml fayli barcha servislar (backend API'lar, RabbitMQ, Redis, PostgreSQL ma'lumotlar bazasi, Nginx frontend serveri) va ularning o'zaro bog'liqliklarini aniqlaydi. Bu fayl butun SCM-IDSS tizimini yagona buyruq (docker-compose up --build) bilan ishga tushirish imkonini beradi.
5. **Xabarlar navbatini ishlovchilarni ishga tushirish (Message Queue Consumers):**
   * SCM-IDSS kabi mikroservislar orasidagi asinxron aloqani ta'minlash uchun RabbitMQ xabarlar navbatidan foydalaniladi. inventory\_service/inventory/management/commands/run\_consumer.py va logistics\_service/logistics/management/commands/run\_consumer.py kabi iste'molchi (consumer) skriptlar alohida jarayonlar sifatida ishga tushiriladi va RabbitMQ'dan kelgan tegishli xabarlarni tinglaydi va qayta ishlaydi.
6. **Dasturni bajarish (Execution):**
   * Barcha konteynerlar ishga tushirilgandan so'ng, foydalanuvchi veb-brauzer orqali SCM-IDSS ilovasining frontendiga kiradi (login.html, keyin dashboard.html).
   * Foydalanuvchi harakatlari (masalan, yangi buyurtma yaratish) frontenddan backend API'lariga HTTP so'rovlari (masalan, order\_service/api/Order/views.py) orqali yuboriladi.
   * Backend servislari o'zaro sinxron (HTTP API orqali) yoki asinxron (RabbitMQ orqali) aloqa qiladi. Masalan, buyurtma yaratilganda, order\_service/orders/producer.py BUYURTMA\_YARATILDI xabarini RabbitMQga yuboradi, bu esa inventar va logistika servislari tomonidan qabul qilinadi va tegishli amallar bajariladi.
   * Fondagi vazifalar (masalan, hisobot generatsiyasi) Celery workerlari (src/src/celery.py) tomonidan bajariladi.

**Ko'rib chiqilayotgan stsenariy uchun oqim sxemasi (Flowchart): "Yangi Buyurtma Yaratish" jarayoni.**

graph TD

A[Foydalanuvchi: create-order.html] --> B{Yangi Buyurtma Ma'lumotlarini Kiritadi};

B --> C[Frontend: POST /api/admin/orders/ (Order Service)];

C --> D{Order Service: Buyurtmani Qabul Qiladi};

D -- Buyurtma ma'lumotlarini vaqtinchalik saqlash --> E[Order Service DB];

D -- Mahsulot narxlarini so'rov --> F{Inventory Service: GET /api/admin/products/{id}/};

F --> G{Inventory Service DB};

G --> F;

F -- Mahsulot narxlari qaytariladi --> D;

D -- Miqdorni tekshirish va yangi buyurtma yaratish --> E;

D -- BUYURTMA\_YARATILDI xabari yuborish --> H[RabbitMQ: orders\_exchange];

H --> I[Inventory Consumer: DECREASE\_STOCK tinglaydi];

I -- Inventarni kamaytirish --> J[Inventory Service DB];

J --> I;

H --> K[Logistics Consumer: ORDER\_CREATED tinglaydi];

K -- Yangi jo'natma yozuvini yaratish --> L[Logistics Service DB];

L --> K;

D -- Buyurtma statusini yangilash (agar kerak bo'lsa) --> E;

D -- Jurnalga yozish --> M[Core Service DB: AuditLog];

D --> N[Foydalanuvchiga muvaffaqiyatli xabar qaytarish];

N --> O[Foydalanuvchi: Dashboard/MyOrders ga yo'naltiriladi];

**D1. Amalga oshirilgan algoritmni mos tilda baholang va yozilgan algoritm bilan kod variantlari o'rtasidagi munosabatni ko'rib chiqing.**

**Algoritmni amalga oshirish: Inventar miqdorini kamaytirish (DECREASE\_STOCK)**

Keling, SCM-IDSS loyihasidagi **inventar miqdorini kamaytirish** algoritmini Python tilida qanday amalga oshirilganini ko'rib chiqamiz. Bu algoritm Order Service tomonidan yuborilgan DECREASE\_STOCK hodisasini qayta ishlash uchun inventory\_service/inventory/management/commands/run\_consumer.py faylida mavjud.

**Yozilgan algoritm (Mantiqiy qadamlar):**

1. RabbitMQdan DECREASE\_STOCK turidagi xabar qabul qilinadi.
2. Xabardan buyurtma mahsulotlari (product ID va miqdori) ro'yxati ajratib olinadi.
3. Har bir mahsulot uchun: a. Tegishli inventar yozuvi ma'lumotlar bazasidan topiladi. b. Agar inventar yozuvi topilsa va mavjud miqdor buyurtma qilingan miqdordan katta yoki teng bo'lsa: i. Inventar miqdori kamaytiriladi. ii. Yangilangan inventar yozuvi ma'lumotlar bazasiga saqlanadi. c. Aks holda (inventar yozuvi topilmasa yoki miqdor yetarli bo'lmasa), tegishli ogohlantirish beriladi.
4. Xabar RabbitMQdan muvaffaqiyatli qayta ishlangan deb tasdiqlanadi.

**Kod varianti (Python'da implementatsiya):**

Python

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

import pika

import json

import os

import django

import traceback

from django.core.management.base import BaseCommand

from django.db import close\_old\_connections, transaction # 'transaction' import qilingan

from django.conf import settings

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

# Django muhitini sozlash (zarur bo'lsa)

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

django.setup()

class Command(BaseCommand):

help = 'Starts the RabbitMQ consumer for inventory service'

def handle(self, \*args, \*\*options):

# Bu metod RabbitMQ ulanishini va xabar iste'mol qilish siklini boshlaydi.

# Asosiy e'tibor process\_inventory\_event metodida.

try:

params = pika.URLParameters(os.getenv('CELERY\_BROKER\_URL'))

connection = pika.BlockingConnection(params)

channel = connection.channel()

channel.exchange\_declare(exchange='orders\_exchange', exchange\_type='fanout', durable=True)

queue\_name = 'inventory\_queue'

channel.queue\_declare(queue=queue\_name, durable=True)

channel.queue\_bind(exchange='orders\_exchange', queue=queue\_name)

channel.basic\_consume(queue=queue\_name, on\_message\_callback=self.process\_inventory\_event)

self.stdout.write(self.style.SUCCESS('Waiting for messages...'))

channel.start\_consuming()

except pika.exceptions.AMQPConnectionError as e:

self.stderr.write(self.style.ERROR(f" [!] RabbitMQ connection error: {e}"))

# Qayta ulanish logikasi (sodda misol)

self.stderr.write(self.style.ERROR(" [!] Retrying connection in 5 seconds..."))

time.sleep(5)

self.handle(\*args, \*\*options)

except KeyboardInterrupt:

self.stdout.write(self.style.WARNING('Consumer stopped by user.'))

except Exception as e:

self.stderr.write(self.style.ERROR(f" [!] An unexpected error occurred in handle: {e}"))

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

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}"))

if event\_type == 'DECREASE\_STOCK':

order\_items = data.get('items', [])

with transaction.atomic(): # Tranzaksiya butunligini ta'minlash

for item in order\_items:

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. Skipping item."))

continue

# Inventar yozuvini bloklash va topish

# select\_for\_update bir vaqtda kirishda ma'lumotlar butunligini ta'minlaydi

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

if inventory\_item:

if inventory\_item.quantity >= int(quantity\_to\_decrease):

inventory\_item.quantity -= int(quantity\_to\_decrease)

inventory\_item.save() # Obyektni saqlash

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

else:

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

# Agar yetarli inventar bo'lmasa, xabarni qayta navbatga qo'ymaslik

ch.basic\_nack(delivery\_tag=method.delivery\_tag, requeue=False)

return # Shu xabar uchun keyingi ishlashni to'xtatish

else:

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

ch.basic\_nack(delivery\_tag=method.delivery\_tag, requeue=False)

return # Shu xabar uchun keyingi ishlashni to'xtatish

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

except json.JSONDecodeError as e:

self.stderr.write(self.style.ERROR(f" [!] JSON decoding error: {e}. Message body: {body}"))

ch.basic\_nack(delivery\_tag=method.delivery\_tag, requeue=False) # Noto'g'ri formatdagi xabarni qayta ishlamaslik

except Exception as e:

self.stderr.write(self.style.ERROR(f" [!] An unexpected error occurred in process\_inventory\_event: {e}"))

self.stderr.write(traceback.format\_exc()) # Xato izini chiqarish

ch.basic\_nack(delivery\_tag=method.delivery\_tag, requeue=False) # Xatoni qayta navbatga qo'ymaslik (yoki requeue=True agar vaqtinchalik muammo bo'lsa)

**Yozilgan algoritm va kod varianti o'rtasidagi munosabatni baholash:**

Python tanlovi, ayniqsa Django va DRF (Django REST Framework) kabi ramkalar bilan birgalikda, SCM-IDSS kabi biznes ilovalarini ishlab chiqish uchun juda mos keladi.

* **Tildan foydalanishning mosligi:** Python o'zining soddaligi, o'qish qulayligi va katta ekotizimi tufayli biznes mantiqini tezda implementatsiya qilish uchun idealdir. Django ORM (Object-Relational Mapper) ma'lumotlar bazasi bilan aloqani ob'ektga yo'naltirilgan tarzda soddalashtiradi, bu esa SQL so'rovlarini qo'lda yozish zaruratini kamaytiradi.
* **Algoritmning kodga translyatsiyasi:**
  + **Qadam 1 (Xabar qabul qilish):** Algoritmning bu qadami pika kutubxonasi yordamida channel.basic\_consume metodi va process\_inventory\_event callback funksiyasi orqali bevosita tarjima qilingan. json.loads(body) orqali JSON xabar ma'lumotlarga ajratiladi.
  + **Qadam 2 va 3 (Ma'lumotlarni ajratish va takrorlash):** Bu data.get('items', []) va for item in order\_items: sikli orqali oddiy va tushunarli Python sintaksisida amalga oshirilgan.
  + **Qadam 3a va 3b (Inventarni topish va yangilash):** Inventory.objects.select\_for\_update().filter(product\_id=product\_id).first() kabi Django ORM metodlaridan foydalanish ma'lumotlar bazasi operatsiyalarini yuqori darajada abstraktsiya qiladi. select\_for\_update() funksiyasi bir vaqtda kirish (concurrency) muammolarini oldini olish uchun (masalan, bir vaqtda ikkita buyurtma bir xil mahsulotni kamaytirmoqchi bo'lsa) ma'lumotlar bazasi qulfini (row lock) ta'minlaydi. with transaction.atomic(): esa bir nechta operatsiyaning atomik tarzda bajarilishini kafolatlaydi.
  + **Qadam 4 (Tasdiqlash):** ch.basic\_ack(delivery\_tag=method.delivery\_tag) metodi RabbitMQ'ga xabar muvaffaqiyatli qayta ishlanganini bildiradi. Xato yuz berganda basic\_nack yordamida xabar noto'g'ri ishlangani bildiriladi.
* **Xatolar va ogohlantirishlar:** Algoritmning "ogohlantirish beriladi" qismi Python'dagi if/else shartlari va self.stdout.write/self.stderr.write kabi logging mexanizmlari orqali amalga oshirilgan. try-except bloklari kutilmagan xatolarni ushlash va ularni to'g'ri boshqarish (masalan, JSON dekodlash xatolari) imkonini beradi.

**Xulosa:** Yozilgan algoritm Python tilining ob'ektga yo'naltirilgan xususiyatlari va Django ramkasining ORM va tranzaksiya boshqaruvi imkoniyatlaridan to'liq foydalangan holda samarali amalga oshirilgan. Bu toza, o'qishga qulay va mustahkam kod yaratilishini ta'minlaydi, bu esa SCM-IDSS kabi murakkab biznes tizimlari uchun juda muhimdir. Algoritmning har bir mantiqiy qadami kodda aniq va samarali tarzda aks ettirilgan.

**P2. Protsessual, ob'ektga yo'naltirilgan va voqea-drayv paradigmalar nima ekanligini, ularning xususiyatlari va ular o'rtasidagi munosabatlarni tushuntiring.**

Dasturlash paradimalari — bu dastur kodini tuzish va tashkil qilish uchun turli yondashuvlar va falsafalardir. Ular dasturchilarga muammolarni qanday hal qilish va kodni qanday tuzish bo'yicha ko'rsatmalar beradi.

**a) Protsessual Dasturlash (Procedural Programming):**

* **Ta'rif:** Protsessual dasturlash - bu dastur kodini bir qator bajariladigan **protseduralar**, **funksiyalar** yoki **subroutinlar** to'plami sifatida tashkil etuvchi dasturlash paradigmasi. Dastur bajarilishi tepadan pastga (top-down) qarab, ushbu protseduralarning aniq ketma-ketligida amalga oshiriladi. Ma'lumotlar va funksiyalar odatda alohida bo'lib, funksiyalar global ma'lumotlar yoki parametrlar orqali uzatilgan ma'lumotlar ustida ishlaydi.
* **Xususiyatlari:**
  + **Protseduralar:** Har bir protsedura ma'lum bir vazifani bajarish uchun qadamlar to'plamini bajaradi.
  + **Tepadan pastga yondashuv:** Dastur oqimi aniq va chiziqli.
  + **Ma'lumotlar va funksiyalarni ajratish:** Ma'lumotlar strukturalari va ular ustida ishlaydigan mantiq alohida saqlanadi.
  + **Qayta ishlatish:** Protseduralarni dasturning turli qismlarida qayta ishlatish mumkin.
* **Munosabat:** Protsessual dasturlash eng qadimgi va asosiy paradigmalardan biridir. Boshqa paradigmalar (ayniqsa, Obyektga yo'naltirilgan) undan kelib chiqqan va ko'pincha protsessual elementlarni (masalan, funksiyalar ichidagi ketma-ket ko'rsatmalar) o'z ichiga oladi.

**b) Ob'ektga yo'naltirilgan dasturlash (Object-Oriented Programming - OOP):**

* **Ta'rif:** OOP dasturiy ta'minotni real dunyo ob'ektlari kabi modellashtiradigan paradigmadir. Bu ob'ektlar ma'lumotlar (**atributlar**) va ular ustida ishlaydigan funksiyalar (**metodlar**)ni birgalikda, yagona birlik sifatida inkapsulyatsiya qiladi.
* **Xususiyatlari:**
  + **Klasslar va ob'ektlar:** Klasslar ob'ektlarni yaratish uchun shablon (blueprint), ob'ektlar esa klasslarning instancelari (nusxalari)dir.
  + **Inkapsulyatsiya (Encapsulation):** Ma'lumotlarni (atributlar) va ular ustida ishlaydigan kodni (metodlar) bitta ob'ekt ichiga o'rash. Bu ma'lumotlarga to'g'ridan-to'g'ri tashqaridan kirishni cheklaydi va faqat belgilangan metodlar orqali o'zgartirishga ruxsat beradi.
  + **Merosxo'rlik (Inheritance):** Yangi klasslarni mavjud klasslardan (ota-klasslardan) yaratish imkonini beradi. Bu kodni qayta ishlatishni va ierarxik munosabatlarni qurishni osonlashtiradi.
  + **Polimorfizm (Polymorphism):** Turli ob'ektlarga bir xil nomdagi metodlar orqali murojaat qilish imkonini beradi, lekin ular o'z kontekstlarida har xil ishlashlari mumkin. Bu kodni moslashuvchan va kengaytiriladigan qiladi.
  + **Abstraktsiya (Abstraction):** Faqat foydalanuvchiga muhim bo'lgan detallarni ko'rsatib, keraksiz murakkabliklarni yashirish.
* **Munosabat:** OOP protsessual dasturlashning murakkab loyihalardagi cheklovlarini hal qilish uchun ishlab chiqilgan. U kodni qayta ishlatish, tizimni masshtablash va murakkablikni boshqarishni yaxshilaydi. SCM-IDSS loyihasining Django kabi ramkalar bilan ishlashi uni OOP tamoyillariga asoslangan qiladi.

**c) Voqea-drayv dasturlash (Event-Driven Programming):**

* **Ta'rif:** Voqea-drayv dasturlashda dastur oqimi foydalanuvchi harakatlari, sensor chiqishlari yoki boshqa dasturlar/tizimlardan keladigan xabarlar kabi **voqealar (events)** tomonidan belgilanadi. Dastur voqealarni kutadi va ularga javoban ma'lum funksiyalarni (**voqea ishlovchilari - event handlers**) bajaradi.
* **Xususiyatlari:**
  + **Voqealar:** Tizimda sodir bo'ladigan harakatlar yoki hodisalar (masalan, tugmani bosish, ma'lumotlar bazasiga yozish, xabar kelishi).
  + **Voqea ishlovchilari:** Ma'lum bir voqea sodir bo'lganda chaqiriladigan kod bloklari.
  + **Asinxron tabiat:** Voqealar ko'pincha asinxron tarzda qayta ishlanadi, ya'ni bir voqea ishlov berilayotganda dastur boshqa vazifalarni bajarishda davom etishi mumkin. Bu blokirovka qiluvchi (blocking) operatsiyalardan qochishga yordam beradi.
  + **Reaktivlik:** Tizim tashqi o'zgarishlarga tez va samarali javob beradi.
* **Munosabat:** Voqea-drayv yondashuv odatda foydalanuvchi interfeyslari (GUI) va distributiv tizimlar (mikroservislar) da qo'llaniladi. U OOP yoki protsessual mantiqni o'z ichiga olgan voqea ishlovchilari bilan birga ishlaydi. SCM-IDSS'ning mikroservislar arxitekturasi va RabbitMQ dan foydalanish voqea-drayv dasturlashning yorqin namunasidir.

**M2. Berilgan ilova manba kodida ishlatilgan protsessual, ob'ektga yo'naltirilgan va voqea-drayv paradigmalarini solishtiring.**

SCM-IDSS loyihasi kabi zamonaviy, murakkab tizimlar ko'pincha turli dasturlash paradigmalarining kuchli tomonlarini birlashtiradi.

**1. Protsessual va Ob'ektga yo'naltirilgan dasturlash o'rtasidagi taqqoslash:**

* **Ma'lumotlar va mantiqiy ajratish:**
  + **Protsessualda:** Ma'lumotlar (global o'zgaruvchilar) va ularni manipulyatsiya qiluvchi funksiyalar odatda ajratilgan bo'ladi. Bu oddiy holatlar uchun yaxshi, ammo ma'lumotlar o'zgarishi butun dastur bo'ylab kutilmagan ta'sirlarga olib kelishi mumkin.
  + **OOPda:** Ma'lumotlar (obyekt atributlari) va ularga xos metodlar (funksiyalar) birgalikda "obyekt" ichida inkapsulyatsiya qilingan. Bu ma'lumotlar yaxlitligini ta'minlaydi va kodni tashkil qilishni soddalashtiradi.
* **Kodni qayta ishlatish va masshtablilik:**
  + **Protsessualda:** Kodni qayta ishlatish asosan funksiya chaqiruvlari orqali amalga oshiriladi. Katta loyihalarda bu murakkablikka olib kelishi mumkin.
  + **OOPda:** Merosxo'rlik va polimorfizm orqali kodni qayta ishlatish ancha samaraliroq. Klasslar va ob'ektlar tizimni kichik, boshqariladigan qismlarga bo'lish imkonini beradi, bu esa loyihaning masshtabliligini oshiradi.
* **SCM-IDSS'da:** SCM-IDSS'ning backend qismi (Django) to'liq OOP tamoyillariga asoslanadi. Har bir ta'minot zanjiri ob'ekti (masalan, Product, Inventory, Order, Shipment) alohida Django modeli (klass) sifatida ta'riflangan (inventory/models.py, order\_service/orders/models.py, logistics/models.py). Bu murakkab biznes mantiqini (masalan, inventar optimizatsiyasi) obyekt darajasida tashkil qilishni soddalashtiradi. Protsessual elementlar asosan yordamchi skriptlar (masalan, manage.py) yoki funktsiya ichidagi oddiy, ketma-ket bajariladigan qadamlarda uchraydi.

**2. Ob'ektga yo'naltirilgan va Voqea-drayv dasturlash o'rtasidagi taqqoslash:**

* **Bajarilish oqimi:**
  + **OOPda:** Dastur oqimi obyekt metodlarining ketma-ket chaqiruvi bilan boshqariladi. Bu ko'pincha sinxron (bir vaqtning o'zida bir amal) jarayonlarni tavsiflaydi.
  + **Voqea-drayvda:** Dastur oqimi tashqi voqealar tomonidan tetiklanadi. Bu asinxron jarayonlarni boshqarish uchun juda mos keladi, chunki uzoq davom etadigan operatsiyalar tizimni bloklamaydi.
* **Aloqa mexanizmi:**
  + **OOPda:** Obyektlar odatda metod chaqiruvlari (to'g'ridan-to'g'ri bog'liqlik) orqali o'zaro aloqa qiladi.
  + **Voqea-drayvda:** Aloqa voqealar (xabarlar) orqali amalga oshiriladi, bu esa komponentlar o'rtasida **ajralishni (decoupling)** ta'minlaydi.
* **SCM-IDSS'da:** SCM-IDSS o'zining **mikroservis arxitekturasi** tufayli voqea-drayv yondashuvidan samarali foydalanadi. Django (OOP) asosidagi servislar o'z ichida biznes mantiqini boshqarsa-da, ular orasidagi aloqa ko'p hollarda RabbitMQ orqali asinxron voqealar orqali amalga oshiriladi. Masalan, Order Service yangi buyurtma yaratganida, u ORDER\_CREATED voqeasini RabbitMQga yuboradi (order\_service/orders/producer.py). Bu voqeani Inventory Service (inventory\_service/inventory/management/commands/run\_consumer.py) va Logistics Service (logistics\_service/logistics/management/commands/run\_consumer.py) mustaqil ravishda tinglaydi va o'z amallarini bajaradi. Bu tizimning kengaytirilishini, mustahkamligini va tarmoq xatolariga chidamliligini oshiradi.

**D2. Dasturlash paradigmalarining manba kodidagi amalga oshirilishini tanqidiy baholang.**

SCM-IDSS loyihasi zamonaviy dasturiy ta'minotni ishlab chiqishdagi tendensiyani, ya'ni turli paradigmalar uyg'unligini aniq namoyish etadi.

**1. Ob'ektga yo'naltirilgan paradigma (OOP) amalga oshirilishi:**

* **Kod misoli (Django Modeli):**

Python

# inventory\_service/inventory/models.py

from django.db import models

class Category(models.Model):

name = models.CharField(max\_length=255, unique=True)

description = models.TextField(blank=True, null=True)

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

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

class Meta:

verbose\_name\_plural = "Categories"

ordering = ['name']

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

return self.name

class Product(models.Model):

sku = models.CharField(max\_length=100, unique=True)

name = models.CharField(max\_length=255)

description = models.TextField(blank=True, null=True)

category = models.ForeignKey(Category, on\_delete=models.SET\_NULL, null=True, related\_name='products')

weight = models.DecimalField(max\_digits=10, decimal\_places=2, blank=True, null=True)

dimensions = models.CharField(max\_length=255, blank=True, null=True) # LxWxH

price = models.DecimalField(max\_digits=10, decimal\_places=2, default=0.00)

stock\_status = models.CharField(max\_length=50, default='in\_stock') # e.g., 'in\_stock', 'low\_stock', 'out\_of\_stock'

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

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

class Meta:

ordering = ['name']

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

return f"{self.name} ({self.sku})"

class Inventory(models.Model):

product = models.OneToOneField(Product, on\_delete=models.CASCADE, related\_name='inventory')

location = models.ForeignKey('Location', on\_delete=models.SET\_NULL, null=True, blank=True, related\_name='inventories')

quantity = models.IntegerField(default=0)

min\_stock\_level = models.IntegerField(default=0) # Minimal xavfsizlik zaxirasi

last\_updated = models.DateTimeField(auto\_now=True)

class Meta:

verbose\_name\_plural = "Inventories"

unique\_together = ('product', 'location') # Mahsulot va joylashuv bo'yicha noyob

ordering = ['product\_\_name']

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

return f"{self.product.name} at {self.location.name if self.location else 'Unknown'}: {self.quantity}"

class Location(models.Model):

name = models.CharField(max\_length=255, unique=True)

address = models.TextField(blank=True, null=True)

capacity\_sqft = models.DecimalField(max\_digits=10, decimal\_places=2, blank=True, null=True)

is\_warehouse = models.BooleanField(default=True)

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

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

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

return self.name

* + **Kod tuzilishi va xususiyatlari bo'yicha baholash:**
    - Yuqoridagi kod Category, Product, Inventory va Location kabi real dunyo ob'ektlarini klasslar sifatida modellashtiradi. Har bir klass o'ziga xos atributlarga (masalan, Product klassida name, sku, price) va metodlarga (masalan, \_\_str\_\_ metodi) ega.
    - **Inkapsulyatsiya:** Ma'lumotlar (name, quantity) va ular ustida ishlaydigan metodlar (Django ORM metodlari, masalan, .save(), .filter()) klass ichida birlashtirilgan. Bu ob'ektning ichki holatiga tashqaridan to'g'ridan-to'g'ri kirishni cheklaydi, ma'lumotlar yaxlitligini ta'minlaydi.
    - **Merosxo'rlik:** Django'ning models.Model klassidan meros qilib olish, har bir modelga ma'lumotlar bazasi bilan ishlash uchun standart funksionallikni (masalan, objects menejeri) taqdim etadi.
    - **Tozalik va modullilik:** Kodni obyektlarga ajratish loyihaning turli qismlarini mustaqil ravishda rivojlantirish va sinash imkonini beradi. Har bir servis (masalan, inventory\_service) o'zining biznes domenini obyektlar orqali modellashtiradi.
  + **OOPning muammoni hal qilishdagi yondashuvi:** OOP SCM-IDSS'ning murakkab biznes domenini real dunyo tushunchalari (mahsulotlar, inventarlar, joylashuvlar, buyurtmalar) orqali modellashtirishga imkon beradi. Bu kodni tushunarli, boshqariladigan va texnik xizmat ko'rsatishga qulay qiladi. Masalan, inventar optimizatsiyasi uchun (FR3.1, FR3.2) Inventory ob'ekti va uning xususiyatlari asosida algoritmlar yozilishi mumkin.

**2. Voqea-drayv paradigma (Event-Driven Programming) amalga oshirilishi:**

* **Kod misoli (RabbitMQ Consumer):**

Python

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

import pika

import json

import os

import django

import traceback

from django.core.management.base import BaseCommand

from django.db import close\_old\_connections, transaction

from inventory.models import inventory as Inventory

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

django.setup()

class Command(BaseCommand):

help = 'Starts the RabbitMQ consumer for inventory service'

def handle(self, \*args, \*\*options):

# ... RabbitMQ connection setup ... (handle qismidan olingan)

try:

params = pika.URLParameters(os.getenv('CELERY\_BROKER\_URL'))

connection = pika.BlockingConnection(params)

channel = connection.channel()

channel.exchange\_declare(exchange='orders\_exchange', exchange\_type='fanout', durable=True)

queue\_name = 'inventory\_queue'

channel.queue\_declare(queue=queue\_name, durable=True)

channel.queue\_bind(exchange='orders\_exchange', queue=queue\_name)

channel.basic\_consume(queue=queue\_name, on\_message\_callback=self.process\_inventory\_event)

self.stdout.write(self.style.SUCCESS('Waiting for messages...'))

channel.start\_consuming()

except pika.exceptions.AMQPConnectionError as e:

self.stderr.write(self.style.ERROR(f" [!] RabbitMQ connection error: {e}"))

self.stderr.write(self.style.ERROR(" [!] Retrying connection in 5 seconds..."))

time.sleep(5)

self.handle(\*args, \*\*options)

except Exception as e:

self.stderr.write(self.style.ERROR(f" [!] An unexpected error occurred in handle: {e}"))

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

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

try:

close\_old\_connections()

data = json.loads(body)

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

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

if event\_type == 'DECREASE\_STOCK': # Voqeani tinglash va javob berish

order\_items = data.get('items', [])

with transaction.atomic():

for item in order\_items:

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

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

# ... inventar miqdorini kamaytirish mantiqi ...

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

if inventory\_item:

if inventory\_item.quantity >= int(quantity\_to\_decrease):

inventory\_item.quantity -= int(quantity\_to\_decrease)

inventory\_item.save()

self.stdout.write(f" > Stock for product {product\_id} decreased.")

else:

self.stdout.write(f" ! WARNING: Not enough stock for product {product\_id}.")

ch.basic\_nack(delivery\_tag=method.delivery\_tag, requeue=False)

return

else:

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

ch.basic\_nack(delivery\_tag=method.delivery\_tag, requeue=False)

return

ch.basic\_ack(delivery\_tag=method.delivery\_tag) # Xabar muvaffaqiyatli ishlangani haqida tasdiqlash

except json.JSONDecodeError as e:

self.stderr.write(self.style.ERROR(f" [!] JSON decoding error: {e}. Body: {body}"))

ch.basic\_nack(delivery\_tag=method.delivery\_tag, requeue=False)

except Exception as e:

self.stderr.write(self.style.ERROR(f" [!] Error in process\_inventory\_event: {e}"))

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

ch.basic\_nack(delivery\_tag=method.delivery\_tag, requeue=False)

* **Kod tuzilishi va xususiyatlari bo'yicha baholash:**
  + run\_consumer.py skripti RabbitMQ dan xabarlarni (event\_type == 'DECREASE\_STOCK') tinglaydigan "iste'molchi" (consumer) vazifasini bajaradi. Bu voqea-drayv paradigmasining asosini tashkil etadi.
  + **Asinxron aloqa:** Order Service producer.py orqali xabar yuboradi va javob kutmaydi. inventory\_consumer esa xabarni qabul qilganda, unga javoban amalni (inventarni kamaytirish) bajaradi.
  + **Ajratish (Decoupling):** Mikroservislar o'rtasida to'g'ridan-to'g'ri bog'liqlik yo'q. Bir servisning ishlamay qolishi boshqasiga darhol ta'sir qilmaydi, chunki ular xabarlar navbati orqali ajratilgan. Bu SCM-IDSS ning mustahkamligini oshiradi.
* **Voqea-drayvning muammoni hal qilishdagi yondashuvi:** Ushbu paradigma distributiv tizimlarda va real vaqt rejimida ishlashni talab qiladigan ilovalarda juda samarali. U tizimning resurslarini (masalan, inventarni yangilash) tezkor va ishonchli tarzda boshqarish imkonini beradi.

**3. Protsessual paradigma amalga oshirilishi:**

* **Kod misoli (Celery Task ichidagi protsessual qadamlar):**

Python

# src/reports/tasks.py

import time

import json

from celery import shared\_task

from django.db import close\_old\_connections

from reports.models import Report # OOP model

@shared\_task

def generate\_report\_task(report\_id):

try:

close\_old\_connections()

report = Report.objects.get(id=report\_id) # OOP: Report obyekti olinadi

report.status = Report.ReportStatus.GENERATING # OOP: Obyekt atributi o'zgartiriladi

report.save() # OOP: Obyekt metodi chaqiriladi

# Quyida protsessual elementlar:

print(f"Generating report {report\_id} for type {report.get\_report\_type\_display()} with params {report.parameters}...")

time.sleep(15) # Aniq, ketma-ket bajariladigan vaqt kechikishi

# Faylni yaratish va unga yozish - qadam-baqadam fayl operatsiyasi

file\_name = f"media/reports/report\_{report\_id}\_{int(time.time())}.txt"

# Ensure media/reports directory exists

os.makedirs(os.path.dirname(file\_name), exist\_ok=True)

with open(file\_name, "w") as f:

f.write(f"This is a sample report for SCM-IDSS.\n")

f.write(f"Report Type: {report.get\_report\_type\_display()}\n")

f.write(f"Parameters: {json.dumps(report.parameters)}\n")

f.write(f"Generated at: {report.created\_at}\n")

report.file\_path = file\_name # OOP: Obyekt atributini yangilash

report.status = Report.ReportStatus.COMPLETED # OOP: Obyekt atributini yangilash

report.save() # OOP: Obyektni saqlash

print(f"Report {report\_id} completed successfully. Path: {file\_name}")

# Notification yuborish (qo'shimcha asinxron voqea bo'lishi mumkin)

except Report.DoesNotExist:

print(f"Report with ID {report\_id} not found.")

except Exception as e:

print(f"Error generating report {report\_id}: {e}")

if report:

report.status = Report.ReportStatus.FAILED

report.save()

traceback.print\_exc()

* **Kod tuzilishi va xususiyatlari bo'yicha baholash:**
  + Yuqoridagi kod generate\_report\_task Celery vazifasi bo'lib, u asosan OOP ob'ektlari (Django modeli Report) bilan ishlashiga qaramay, uning ichida hisobot generatsiyasining ba'zi bosqichlari protsessual tarzda (qadam-baqadam ko'rsatmalar ketma-ketligi) amalga oshirilgan. Masalan, fayl yaratish, unga ma'lumot yozish yoki vaqt kechikishini kiritish (time.sleep) kabi operatsiyalar.
* **Protsessual yondashuvning muammoni hal qilishdagi yondashuvi:** SCM-IDSS loyihasida sof protsessual qismlar juda kam, chunki Django kabi ramkalar OOPga asoslangan. Biroq, bu kod misoli shuni ko'rsatadiki, hatto murakkab OOP yoki voqea-drayv tizimlarida ham ba'zi aniq, ketma-ket bajariladigan vazifalar uchun protsessual yondashuvdan foydalanish mumkin. Bu oddiy operatsiyalar uchun kodni sodda va tushunarli qiladi. Shu bilan birga, bu faqat kichik, izolyatsiya qilingan vazifalar uchun samarali. Katta tizimlarni to'liq protsessual tarzda yozish kodni boshqarish va texnik xizmat ko'rsatishni qiyinlashtiradi.

**Yakuniy baho:** SCM-IDSS loyihasi bir nechta dasturlash paradigmalarining kuchli tomonlarini birlashtirish orqali o'zining murakkab funksionalligini samarali amalga oshiradi. OOP arxitekturaning modulliligini ta'minlasa, voqea-drayv paradigmalar mikroservislar o'rtasida masshtabli va mustahkam aloqani ta'minlaydi. Kichik protsessual qismlar esa aniq, mahalliy vazifalarni bajarishda yordam beradi. Bu uyg'unlik tizimning samaradorligi, kengaytirilishi va texnik xizmat ko'rsatish qulayligini oshiradi.

**P3. Dasturni IDE yordamida algoritm asosida yozing.**

Men SCM-IDSS loyihasini rivojlantirishda **Visual Studio Code (VS Code)** [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/.vscode/settings.json] IDE'sidan foydalandim. 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.

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 [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/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 # RabbitMQ aloqasi uchun

import json # JSON formatdagi ma'lumotlar bilan ishlash uchun

import os # Operatsion tizim funksiyalari uchun

import django # Django ramkasini sozlash uchun

import traceback # Xato izlarini olish uchun

# Django'ning asosiy buyruqlar sinfidan BaseCommand'ni import qilish

from django.core.management.base import BaseCommand

# Ma'lumotlar bazasi ulanishlarini boshqarish va tranzaksiyalar uchun

from django.db import close\_old\_connections, transaction

# Django sozlamalarini import qilish

from django.conf import settings

# Inventar modelini import qilish

from inventory.models import inventory as Inventory

# Django muhitini sozlash. Bu skriptni Django loyihasidan tashqarida ishga tushirish uchun zarur.

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

django.setup()

class Command(BaseCommand):

help = 'Inventar xizmati uchun RabbitMQ iste'molchisini ishga tushiradi.'

def handle(self, \*args, \*\*options):

"""

Bu metod RabbitMQ ulanishini o'rnatadi va xabar iste'mol qilish siklini boshlaydi.

Ulanishda xato bo'lsa, qayta ulanishga urinadi.

"""

self.stdout.write(self.style.SUCCESS('Inventar Iste\'molchisi ishga tushirilmoqda...'))

try:

# RabbitMQ brokeriga ulanish parametrlarini CELERY\_BROKER\_URL environment o'zgaruvchisidan olish

params = pika.URLParameters(os.getenv('CELERY\_BROKER\_URL'))

connection = pika.BlockingConnection(params) # Bloklovchi ulanish yaratish

channel = connection.channel() # Kanal ochish

# 'orders\_exchange' nomli exchange (almashinuv nuqtasi) e'lon qilish.

# 'fanout' turi xabarni bog'langan barcha navbatlarga yuboradi.

channel.exchange\_declare(exchange='orders\_exchange', exchange\_type='fanout', durable=True)

queue\_name = 'inventory\_queue' # Navbat nomi

# Navbatni e'lon qilish/yaratish. 'durable=True' navbat broker qayta ishga tushganda yo'qolmasligini bildiradi.

channel.queue\_declare(queue=queue\_name, durable=True)

# Navbatni exchange'ga bog'lash. Bo'sh 'routing\_key' fanout exchange uchun standartdir.

channel.queue\_bind(exchange='orders\_exchange', queue=queue\_name)

# Xabarlarni iste'mol qilishni boshlash. Har bir xabar kelganda

# 'self.process\_inventory\_event' metodi chaqiriladi.

channel.basic\_consume(queue=queue\_name, on\_message\_callback=self.process\_inventory\_event)

self.stdout.write(self.style.SUCCESS('Xabarlar kutilmoqda. Chiqish uchun CTRL+C tugmasini bosing.'))

channel.start\_consuming() # Iste'mol qilish siklini boshlash

except pika.exceptions.AMQPConnectionError as e:

# RabbitMQ ulanish xatoligini ushlash

self.stderr.write(self.style.ERROR(f" [!] RabbitMQ ulanish xatosi: {e}"))

self.stderr.write(self.style.ERROR(" [!] RabbitMQ ishga tushganligiga va erishish mumkinligiga ishonch hosil qiling. 5 soniyadan so'ng qayta urinilmoqda..."))

time.sleep(5) # Qayta urinishdan oldin kutish

self.handle(\*args, \*\*options) # Rekursiv qayta ulanishga urinish

except KeyboardInterrupt:

# Foydalanuvchi CTRL+C bosganda iste'molchini to'xtatish

self.stdout.write(self.style.WARNING('Iste\'molchi foydalanuvchi tomonidan to\'xtatildi.'))

except Exception as e:

# Boshqa kutilmagan xatolarni ushlash

self.stderr.write(self.style.ERROR(f" [!] handle metodida kutilmagan xato yuz berdi: {e}"))

self.stderr.write(traceback.format\_exc()) # Xato izini chiqarish

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

"""

RabbitMQ'dan qabul qilingan har bir xabarni qayta ishlaydigan callback metodi.

Inventar miqdorini kamaytirish algoritmini implementatsiya qiladi.

"""

try:

close\_old\_connections() # Har bir xabar uchun eski DB ulanishlarini yopish. Bu Django bilan muhim.

data = json.loads(body) # JSON formatdagi xabar tanasini Python lug'atiga aylantirish

event\_type = data.get('event\_type') # Xabar turini olish

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

# Inventar miqdorini kamaytirish algoritmi

if event\_type == 'DECREASE\_STOCK': # 1. Agar voqea turi 'DECREASE\_STOCK' bo'lsa

order\_items = data.get('items', []) # 2. Xabardan buyurtma mahsulotlari ro'yxatini olish. Agar yo'q bo'lsa, bo'sh ro'yxat.

# Tranzaksiya butunligini ta'minlash. Bu blokdagi barcha DB operatsiyalari

# yo to'liq bajariladi, yo hech qaysisi bajarilmaydi (atomiklik).

with transaction.atomic():

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

product\_id = item.get('product\_id') # Mahsulot ID'sini olish

quantity\_to\_decrease = item.get('quantity') # Kamaytirilishi kerak bo'lgan miqdorni olish

# Kirish ma'lumotlarini tekshirish

if not product\_id or quantity\_to\_decrease is None: # quantity\_to\_decrease 0 bo'lishi mumkinligini hisobga olish

self.stderr.write(self.style.ERROR(f" [!] XATO: Buyurtma elementi uchun product\_id yoki miqdor yetarli emas. Element o'tkazib yuborilmoqda."))

continue # Keyingi elementga o'tish

try:

# 4. Inventardagi mahsulotni topish va bloklash.

# select\_for\_update() bir vaqtda kirishda ma'lumotlar butunligini ta'minlaydi (race condition'larning oldini oladi).

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. Agar mavjud miqdor yetarli bo'lsa

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

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

self.stdout.write(f" > Mahsulot {product\_id} uchun zaxira {quantity\_to\_decrease} ga kamaytirildi. Yangi miqdor: {inventory\_item.quantity}")

else: # 9. Agar miqdor yetarli bo'lmasa

self.stdout.write(self.style.WARNING(f" ! OGOHLANTIRISH: Mahsulot {product\_id} uchun zaxira yetarli emas. Talab qilingan: {quantity\_to\_decrease}, Mavjud: {inventory\_item.quantity}"))

# Xabarni qayta navbatga qo'ymaslik, chunki bu biznes mantiqiga oid muammo.

ch.basic\_nack(delivery\_tag=method.delivery\_tag, requeue=False)

return # Shu xabar uchun keyingi ishlashni to'xtatish

else: # 10. Agar inventar yozuvi topilmasa

self.stdout.write(self.style.WARNING(f" ! OGOHLANTIRISH: Mahsulot {product\_id} uchun inventar yozuvi topilmadi. Zaxira kamaytirilmaydi."))

# Xabarni qayta navbatga qo'ymaslik.

ch.basic\_nack(delivery\_tag=method.delivery\_tag, requeue=False)

return # Shu xabar uchun keyingi ishlashni to'xtatish

except ValueError: # Agar quantity\_to\_decrease int'ga o'zgarmasa

self.stderr.write(self.style.ERROR(f" [!] XATO: Miqdor noto'g'ri formatda: {quantity\_to\_decrease}. Mahsulot ID: {product\_id}"))

ch.basic\_nack(delivery\_tag=method.delivery\_tag, requeue=False)

return

except Exception as item\_e:

self.stderr.write(self.style.ERROR(f" [!] Elementni qayta ishlashda xato yuz berdi {product\_id}: {item\_e}"))

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

ch.basic\_nack(delivery\_tag=method.delivery\_tag, requeue=False)

return

# 11. Xabar muvaffaqiyatli qayta ishlanganini tasdiqlash

ch.basic\_ack(delivery\_tag=method.delivery\_tag)

except json.JSONDecodeError as e:

# JSON formatlash xatolarini ushlash

self.stderr.write(self.style.ERROR(f" [!] JSON dekodlash xatosi: {e}. Xabar tanasi: {body}"))

ch.basic\_nack(delivery\_tag=method.delivery\_tag, requeue=False) # Noto'g'ri formatdagi xabarni qayta ishlamaslik

except Exception as e:

# Boshqa kutilmagan xatolarni ushlash

self.stderr.write(self.style.ERROR(f" [!] callback metodida kutilmagan xato yuz berdi: {e}"))

self.stderr.write(traceback.format\_exc()) # Xato izini chiqarish

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

Ushbu kod parchasi inventar miqdorini kamaytirish algoritmini to'liq qamrab oladi. U RabbitMQdan xabar qabul qiladi, xabar tarkibini tahlil qiladi, inventarni yangilaydi (transaksiya va qulflash orqali ma'lumotlar butunligini ta'minlaydi) va xato holatlarini boshqaradi.

**M3. Dastur rivojlanish jarayonini boshqarish uchun IDE'dan foydalanish.**

Visual Studio Code (VS Code) kabi integratsiyalashgan rivojlanish muhitlari (IDE) dasturiy ta'minotni ishlab chiqish jarayonini kompleks boshqarish uchun juda ko'plab vositalarni taqdim etadi. SCM-IDSS kabi mikroservislar arxitekturasi [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/README.md] asosidagi murakkab loyihada IDE'dan foydalanish dasturchining samaradorligini sezilarli darajada oshiradi.

1. **Kod tahrirlash va navigatsiya (Code Editing & Navigation):**
   * **Sintaksisni ajratib ko'rsatish (Syntax Highlighting):** VS Code Python, HTML, CSS, JavaScript [cite: user\_query] kodini o'qishni va tushunishni osonlashtiradigan rangli sintaksisni ta'minlaydi. Bu kod bloklarini, o'zgaruvchilarni, funksiyalarni va kalit so'zlarni tezda ajratishga yordam beradi.
   * **Avtomatik to'ldirish (IntelliSense):** Dasturlash tilining sintaksisi, kutubxonalar (masalan, pika, django) [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/inventory\_service/inventory/management/commands/run\_consumer.py] va loyiha kodidagi obyektlar bo'yicha aqlli takliflar beradi. Bu kod yozish tezligini oshiradi va sintaksis xatolarini kamaytiradi. Masalan, Inventory.objects. [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/inventory\_service/inventory/management/commands/run\_consumer.py] deb yozganda filter, get, select\_for\_update kabi metodlarni taklif qiladi.
   * **Kod navigatsiyasi:** Go to Definition, Find All References kabi funksiyalar kodning turli qismlari (funksiyalar, klasslar, o'zgaruvchilar) orasida tezda harakatlanish imkonini beradi. SCM-IDSS kabi ko'p faylli, ko'p servisli loyihada [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/README.md] fayllar va funksiyalar orasida tezkor o'tish juda muhimdir.
   * **Refactoring vositalari:** O'zgaruvchi nomlarini o'zgartirish, funksiyalarni ajratish yoki birlashtirish kabi kodni qayta tuzish operatsiyalarini xavfsiz va samarali bajarishga yordam beradi.
2. **Xatolarni tuzatish (Debugging):**
   * **Breakpoints (To'xtash nuqtalari):** Kodning istalgan qatoriga to'xtash nuqtalarini 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:** run\_consumer.py [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/inventory\_service/inventory/management/commands/run\_consumer.py] faylidagi process\_inventory\_event funksiyasining if event\_type == 'DECREASE\_STOCK': [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/inventory\_service/inventory/management/commands/run\_consumer.py] qatoriga breakpoint qo'yib, kiruvchi xabarning tarkibini (data o'zgaruvchisi) va order\_items ni tekshirish mumkin.
   * **Qadam-baqadam bajarish (Step-through, Step-into, Step-out):** Dastur bajarilishini qatorma-qator nazorat qilish imkonini beradi. Bu murakkab mantiqiy bloklarda yoki funksiyalar chaqiruvida xatolarni aniqlashda yordam beradi.
   * **O'zgaruvchilarni kuzatish (Variable Inspection & Watch):** Dastur to'xtatilgan nuqtada barcha lokal va global o'zgaruvchilarning joriy qiymatlarini ko'rish imkonini beradi. Watch oynasi orqali ma'lum o'zgaruvchilarning qiymatini doimiy kuzatish mumkin. Masalan, inventory\_item.quantity [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/inventory\_service/inventory/management/commands/run\_consumer.py] qiymatining har bir operatsiyadan keyingi holatini kuzatish.
   * **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.
3. **Versiyalarni boshqarish integratsiyasi (Git Integration):**
   * VS Code kabi IDE'lar Git [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/.gitignore] kabi versiya nazorati tizimlari bilan integratsiyalashgan. Bu dasturchiga kodlardagi o'zgarishlarni bevosita IDE ichida kuzatish, commit qilish, branchlarni boshqarish va konfliktlarni hal qilish imkonini beradi. Bu jamoaviy ishlashda va kod tarixini saqlashda juda muhim.
4. **Terminal integratsiyasi:**
   * IDE ichida o'rnatilgan terminal mavjudligi dasturchiga Django migratsiyalarini (python manage.py makemigrations, python manage.py migrate) [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/inventory\_service/inventory/migrations/0001\_initial.py] yoki Docker Compose buyruqlarini (docker-compose up --build) [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/README.md] bajarishda qulaylik yaratadi.
5. **Plaginlar va kengaytmalar (Extensions):**
   * VS Code Python kengaytmasi, Docker, Pylint/Flake8 (linterlar) [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/.vscode/settings.json] kabi ko'plab plaginlarni qo'llab-quvvatlaydi. Bu kengaytmalar kod sifatini tekshirish, Docker fayllarini boshqarish va umumiy rivojlanish jarayonini optimallashtirishga yordam beradi.

**D3. Ilovalarni ishlab chiqishda IDE'dan foydalanishni IDE'dan foydalanmaslik bilan solishtirish.**

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

1. **Samaradorlikni keskin oshirish:** Yuqorida sanab o'tilgan avtomatik to'ldirish, sintaksisni ajratib ko'rsatish, kod navigatsiyasi va refactoring vositalari kod yozish, tushunish va qayta ishlashni tezlashtiradi. SCM-IDSS kabi ko'p faylli, ko'p tilli (Python, HTML, CSS, JS) [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/README.md] loyihada turli komponentlar orasida tezkor o'tish juda muhim.
2. **Xatolarni kamaytirish va sifatni yaxshilash:** Real vaqt rejimida sintaksis, ba'zi mantiqiy xatolar va kodlash standarti buzilishlarini (flake8, pylint [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/.vscode/settings.json]) aniqlash dasturchiga muammolarni erta bosqichda bartaraf etishga yordam beradi. Bu kod sifatini oshiradi va keyingi bosqichlarda (testlash, joylashtirish) kamroq xatolarga olib keladi.
3. **Disk raskadrovka qulayligi:** Integratsiyalashgan disk raskadrovka vositalari murakkab jarayonlarda (masalan, RabbitMQ xabar almashinuvi, Celery tasklar) [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/inventory\_service/inventory/management/commands/run\_consumer.py, uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/src/reports/tasks.py]) xatolarni topish va tuzatishni sezilarli darajada soddalashtiradi. Uzoqdan debugging imkoniyatlari Docker konteynerlarida ishlaydigan servislarni [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/docker-compose.yml] nosozliklardan tozalashga imkon beradi.
4. **Loyihani kompleks boshqarish:** Bir IDE'da butun loyihani (mikroservislar, ma'lumotlar bazasi migratsiyalari, frontend fayllari) [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/README.md] ko'rish va boshqarish imkoniyati dasturchiga umumiy tasavvurni beradi va loyiha strukturasini saqlashga yordam beradi.
5. **Integratsiyalashgan ekotizim:** Git, terminal, linterlar, formatlovchilar kabi vositalarning bir joyda to'planishi dasturchining turli ilovalar orasida almashishiga bo'lgan ehtiyojni kamaytiradi, bu esa ish jarayonini silliq qiladi.

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

1. **Samaradorlikning pastligi:** Avtomatik to'ldirish, sintaksis tekshiruvi va refactoring vositalarisiz kod yozish ancha sekin kechadi va xatolarga moyil bo'ladi. Har bir mayda sintaksis xatosini qo'lda topish katta vaqt yo'qotishiga olib keladi.
2. **Disk raskadrovka murakkabligi:** Disk raskadrovka qilish uchun alohida vositalar (masalan, Python uchun pdb) [cite: user\_query] yoki print() funksiyalaridan foydalanish zarurati yuzaga keladi, bu esa vaqtni oladi va jarayonni qiyinlashtiradi. Ayniqsa, asinxron yoki tarmoqqa bog'liq muammolarni topish juda qiyinlashadi.
3. **Loyihani boshqarish qiyinligi:** Katta va murakkab loyihalarda (SCM-IDSS kabi, 5+ mikroservis va yuzlab fayllar bilan) [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/README.md] fayllar va kataloglar orasida qo'lda navigatsiya qilish juda samarasiz.
4. **Konfiguratsiyani qo'lda boshqarish:** Har bir til, ramka va kutubxona uchun sozlamalarni (masalan, requirements.txt [cite: uploaded:nurulloasawear/scm\_idss\_project/scm\_idss\_project-fc39e2307c50489d25ac632f5e0301bf36b8bb9e/inventory\_service/requirements.txt] dagi bog'liqliklarni) qo'lda boshqarish va konfiguratsiya qilish talab qilinadi, bu esa xatoliklarga olib kelishi mumkin.
5. **Jamoaviy ishlashda qiyinchilik:** Turli dasturchilar turli xil sozlamalar va formatlash usullarini qo'llashi natijasida kod integratsiyasi muammolari yuzaga kelishi mumkin, chunki kodlash standartlariga rioya qilish qiyinlashadi.

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 (debugging) 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\_items ni 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 & Watch):** 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.
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, Product, 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).
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) va Injection zaifliklari:**
   * **Qanday yordam beradi:** 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. run\_consumer.py da int(quantity\_to\_decrease) konvertatsiyasi va uning atrofidagi try-except bloki kabi xato qidirish amaliyotlari ma'lumotlar turini tekshirishni ta'minlaydi.
   * **Xavfsizlikni yaxshilash:** Debugger yordamida kirish nuqtalarida o'zgaruvchilar qiymatini tekshirib, noto'g'ri yoki xavfli kiritmalarning (masalan, SQL injection, XSS) oldini olish mumkin. Django ORM injection hujumlarining aksariyatini to'xtatgan bo'lsa-da, maxsus holatlarda yoki xom SQL so'rovlarida debugging tekshiruvlari foydali bo'ladi.
2. **Avtorizatsiya va Autentifikatsiya mantiqini tekshirish:**
   * **Qanday yordam beradi:** Debugger orqali autentifikatsiya va avtorizatsiya mantiqi (masalan, inventory\_service/api/permissions.py dagi ruxsatlarni tekshirish) orqali o'tish mumkin.
   * **Xavfsizlikni yaxshilash:** Bu turli foydalanuvchi rollari (masalan, "Ta'minot zanjiri rahbarlari" va "Inventarizatsiya menejerlari") uchun to'g'ri kirish huquqlari qo'llanilganligini tekshirish imkonini beradi. Har qanday privilegiya eskalatsiyasi (privilege escalation) yoki ruxsatsiz kirish (unauthorized access) urinishlarini aniqlash va tuzatishga yordam beradi.
3. **Ma'lumotlar sizib chiqishini (Data Leakage) aniqlash:**
   * **Qanday yordam beradi:** Debugging yordamida dastur ishlayotganda qaysi ma'lumotlarning qayta ishlanayotganini va tarmoq orqali yoki loglarda yuborilayotganini ko'rish mumkin.
   * **Xavfsizlikni yaxshilash:** Bu maxfiy ma'lumotlar (masalan, yetkazib beruvchining shartnoma shartlari, foydalanuvchi parollari) tasodifan oshkor bo'lmasligini ta'minlaydi. Serializatorlar (inventory\_service/api/inventory/serializers.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 (Edge Cases) va Kutilmagan kirishlarni boshqarish:**
   * **Qanday yordam beradi:** Debugging, odatiy ishlash jarayonida yuzaga kelmaydigan, lekin kamdan-kam holatlarda (masalan, nol miqdor, manfiy sonlar, bo'sh satrlar, kutilmagan formatdagi ma'lumotlar) yuzaga keladigan xatolarni aniqlashda yordam beradi.
   * **Mustahkamlikni yaxshilash:** run\_consumer.py dagi if inventory\_item.quantity >= int(quantity\_to\_decrease): tekshiruvi va ValueErrorni ushlash bunga misoldir. Debugger orqali ushbu shartlarni sinab, dasturning munosabati kutilganidek ekanligiga ishonch hosil qilish mumkin. Bu dasturning to'satdan ishlamay qolishi o'rniga, xatolarni silliq boshqarishini ta'minlaydi.
2. **Resurslarni boshqarish va unumdorlik muammolari (Resource Management & Performance Issues):**
   * **Qanday yordam beradi:** Uzoq vaqt ishlaydigan dasturlarda (masalan, RabbitMQ iste'molchilari yoki Celery workerlari) xotira sızmalarını (memory leaks) yoki yopilmagan ma'lumotlar bazasi ulanishlarini (close\_old\_connections()) aniqlashda debugging yordam beradi.
   * **Mustahkamlikni yaxshilash:** Debugger bilan tizimning CPU, xotira sarfini kuzatish va resurslardan noto'g'ri foydalanishga olib keladigan kod qismlarini topish mumkin. Bu tizimning barqarorligini va uzluksiz ishlashini ta'minlaydi.
3. **Bir vaqtda ishlash (Concurrency) va Tranzaksiya butunligi (Transactional Integrity) muammolari:**
   * **Qanday yordam beradi:** SCM-IDSS kabi distributiv tizimlarda bir vaqtning o'zida bir nechta jarayonlar yoki tredlar ma'lumotlarga kirishga urinishi mumkin. Debugging (ayniqsa, qulflash (locks) yoki semaforlar (semaphores) bilan) poyga shartlari (race conditions) va o'lik qulf (deadlocks) kabi qiyin topiladigan xatolarni aniqlashda yordam beradi.
   * **Mustahkamlikni yaxshilash:** with transaction.atomic(): kabi Django vositalari ma'lumotlar bazasi darajasida atomiklikni ta'minlaydi. Debugging bu bloklarning to'g'ri ishlashini va ma'lumotlar butunligini buzmasligini tekshirish imkonini beradi.

Xulosa qilib aytganda, debugging dasturchiga o'z kodining ichki ish faoliyatini chuqur 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):**
   * **Ahamiyati:** 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, PascalCase klass 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:** Dastlab jamoa a'zolari uchun yangi standartlarga moslashish biroz qiyin bo'lishi mumkin, ammo uzoq muddatda bu intizom o'zini oqlaydi.
2. **O'qish qulayligi va tushunarliligi (Readability & Understandability):**
   * **Ahamiyati:** 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) va ViewSets (views.py) fayllarida 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):**
   * **Ahamiyati:** 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):**
   * **Ahamiyati:** 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):**
   * **Ahamiyati:** 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 kabi versiya nazorati tizimlari bilan ishlashda muhimdir.

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

1. **Eng yaxshi amaliyotlarni o'rganish (Learning Best Practices):**
   * **Ahamiyati:** 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):**
   * **Ahamiyati:** 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. Bu o'zining eski kodini qayta ishlatish yoki unga texnik xizmat ko'rsatishda juda foydali.
3. **Intizom va professionalizm (Discipline & Professionalism):**
   * **Ahamiyati:** Kodlash standartlariga rioya qilish dasturchida intizomni shakllantiradi va uning professionalizmini oshiradi. Bu kelajakda boshqa loyihalarda ishlashda ham foydali bo'ladi va ularga jamoaning bir qismi sifatida tezroq moslashishga yordam beradi.
4. **Ishga qabul qilinish imkoniyatini oshirish:**
   * **Ahamiyati:** Ko'pgina kompaniyalar o'zlarining kodlash standartlariga ega va ishga qabul qilishda nomzodlarning toza, izchil kod yozish qobiliyatini qadrlaydi. 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 keng qamrovli va mikroservis arxitekturasi asosidagi 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 sifati hamda xavfsizligini 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 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 dasturlash tamoyillarini o'rganishni yakunlaydi. Biz SCM-IDSS'ning an'anaviy ta'minot zanjiri 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 vazifasini ko'rib chiqdik.

Hisobot davomida biz dasturlash fanining asosiy tamoyillarini chuqur o'rgandik:

* **LO1: Algoritmlar va ilova yaratish jarayoni:** Biz algoritmlarning ta'rifini, ularning SCM-IDSS'ning Talabni Prognozlash, Inventarni Optimallashtirish va Yo'nalishni Rejalashtirish kabi asosiy funksiyalarida qanday qo'llanilishini aniqladik. Dasturiy ta'minotni ishlab chiqishning to'liq jarayonini (SDLC), shu jumladan talablarni yig'ish, dizayn, implementatsiya, testlash, joylashtirish va texnik xizmat ko'rsatish bosqichlarini, shuningdek, kod yozishdan tortib Docker konteynerlari yordamida bajarishgacha bo'lgan qadamlarni tahlil qildik. SCM-IDSS'ning "Yangi Buyurtma Yaratish" jarayoni uchun oqim sxemasi tizimning ishlashini vizual tarzda ko'rsatdi.
* **LO2: Dasturlash paradigmalarining xususiyatlari:** Protsessual, ob'ektga yo'naltirilgan (OOP) va voqea-drayv (Event-Driven) dasturlash paradigmalarining asosiy xususiyatlarini, ular o'rtasidagi munosabatlarni va ularning SCM-IDSS loyihasi manba kodida qanday namoyon bo'lishini tushuntirdik. Xususan, Django modellari va ViewSets'lari orqali OOP'ning, RabbitMQ xabar navbatlari va iste'molchilari orqali esa voqea-drayv paradigmasining ahamiyati va amaliyotini baholadik. Bu paradigmalarning uyg'unligi SCM-IDSS'ning mikroservis arxitekturasining mustahkamligi va kengaytirilishini ta'minlaydi.
* **LO3: IDE yordamida algoritmlarni kodda implementatsiya qilish:** Biz Visual Studio Code (VS Code) kabi integratsiyalashgan rivojlanish muhitlarining (IDE) dasturiy ta'minotni ishlab chiqish jarayonidagi markaziy rolini tahlil qildik. Inventar miqdorini kamaytirish algoritmi misolida IDE yordamida kodning qanday yozilishini va boshqarilishini ko'rsatdik. IDE'dan foydalanishning samaradorlik, kod sifati va nosozliklarni tuzatishdagi afzalliklari oddiy matn muharririga nisbatan tanqidiy baholandi.
* **LO4: Nosozliklarni tuzatish jarayoni va kodlash standartining ahamiyati:** Biz nosozliklarni tuzatish jarayonini (qayta tiklash, aniqlash, tahlil qilish, tuzatish, tekshirish) va IDE'da mavjud bo'lgan kuchli vositalarni (breakpoints, qadam-baqadam bajarish, o'zgaruvchilarni tekshirish) batafsil tushuntirdik. Shuningdek, debugging jarayonining dasturlarni yanada xavfsizroq (xavfsizlik zaifliklarini aniqlash) va mustahkamroq (chegaraviy holatlar, resurslarni boshqarish) qilishga qanday yordam berishini baholadik. PEP 8 kabi kodlash standartlarining jamoada kod izchilligi, o'qish qulayligi va hamkorlikni yaxshilashdagi, shaxsiy darajada esa professionalizm va intizomni oshirishdagi mutlaq zarurligini tanqidiy asosladik.

Yakuniy xulosa shuki, SCM-IDSS kabi intellektual va murakkab tizimni muvaffaqiyatli yaratish uchun dasturlashning barcha asosiy tamoyillarini nafaqat nazariy jihatdan bilish, balki ularni Python, Django, mikroservislar, RabbitMQ va Docker kabi zamonaviy texnologiyalar bilan amaliyotga tatbiq eta olish ham muhimdir. Ushbu topshiriq dasturiy ta'minotni ishlab chiqishning butun sikli bo'yicha chuqur tushuncha berdi va kelajakdagi raqamli texnologiyalar loyihalariga mustahkam asos yaratdi.

## 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](https://books.google.com/books/about/Decision_Support_Systems_and_Intelligent.html?hl=id&id=NfMJAQAAMAAJ" \t "_blank) (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](https://www.igi-global.com/article/supply-chain-efficiency-and-effectiveness-management-using-decision-support-systems/305847" \t "_blank) (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](https://www.researchgate.net/publication/329784099_Decision_Support_Systems_and_Artificial_Intelligence_in_Supply_Chain_Risk_Management" \t "_blank) (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](https://www.harvard.com/book/9798369335932" \t "_blank) (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](https://www.routledge.com/Data-Driven-Technologies-and-Artificial-Intelligence-in-Supply-Chain-Tools-and-Techniques/Chand-Jain-Ajmera/p/book/9781032611297" \t "_blank) (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](https://store.hbr.org/product/how-machine-learning-will-transform-supply-chain-management/R2402K" \t "_blank) (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/](https://www.oreilly.com/library/view/machine-learning-logistics/9781491997628/" \t "_blank) (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](https://www.routledge.com/Supply-Chain-Analytics-Using-Data-to-Optimise-Supply-Chain-Processes/Robertson/p/book/9780367540067" \t "_blank) (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](https://www.wiley.com/en-us/Operations+and+Supply+Chain+Management%2C+11th+Edition-p-00381326" \t "_blank) (Accessed: 30 June 2025).
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](https://www.researchgate.net/publication/323878142_Big_Data_Analytics_for_Supply_Chain_Management" \t "_blank) (Accessed: 30 June 2025).