|  |
| --- |
| Unit 4: Programming |
| Abdulloh Halilov |

**O‘qituvchi:** Ilyoskhuja Ikromkhujaev

**Guruh identifikatori:** 23-412

**Talaba ID:** 231397

**Taqdim etilgan sana:** 20.01.2025 yil

**BTEC o'quvchilar topshiriqlarini baholash va deklaratsiya**

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

|  |  |
| --- | --- |
| **O‘quvchi (talaba) identifikatori:** | 231397 |
| **Baholovchi nomi:** | Ilyoskhuja Ikromkhujaev |
| **BTEC dasturi nomi:** | Pearson BTEC Higher Nationals in Digital Technologies |
| **Birlik yoki komponent raqami va nomi:** | Unit 4: Programming |
| **Topshiriq nomi:** | Internal Assignment |
| **Topshiriq topshirilgan sana:** | 20.01.2025 yil |

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

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

|  |
| --- |
| **O'quvchi deklaratsiyasi**  Men 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:** 20.01.2025 yil |

Mundarija

[KIRISH 4](#_Toc188005183)

[LO1 4](#_Toc188005184)

[A.P1 4](#_Toc188005185)

[A.M1 8](#_Toc188005186)

[A.D1 15](#_Toc188005187)

[LO2 25](#_Toc188005188)

[A.P2 25](#_Toc188005189)

[A.M2 37](#_Toc188005190)

[A.D2 41](#_Toc188005191)

[A.P3 45](#_Toc188005192)

[A.M3 47](#_Toc188005193)

[A.D3 49](#_Toc188005194)

[LO4 53](#_Toc188005195)

[A.P4 53](#_Toc188005196)

[A.P5 56](#_Toc188005197)

[A.M4 65](#_Toc188005198)

[A.D4 67](#_Toc188005199)

[adabiyotlar ro'yxati 71](#_Toc188005200)

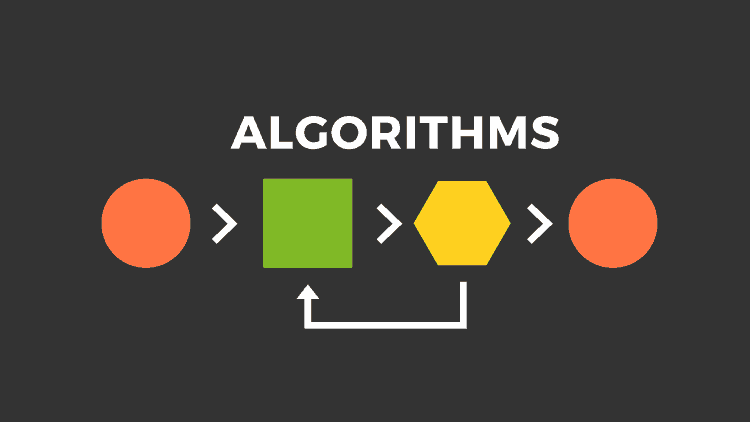
KIRISH

Assalomu alaykum. NewEra Cash & Carry uchun ASP.NET API-ni ishlab chiqish va hujjatlashtirish bo'yicha topshirig'imga xush kelibsiz. Ushbu loyiha modulimizning o'quv maqsadlariga erishish uchun amalga oshirialdi. Mening maqsadim nafaqat ushbu tushunchalarni tushunishimni namoyish etish, balki ularni ASP.NET API ishlab chiqish doirasida amalda qo'llashdir. Ushbu kirish bo'limi nazariy bilimlarni qanday qilib funktsional dasturiy echimlarga aylantirish mumkinligini chuqurroq o'rganishga yordam beradi.

**LO1**

**A.P1**

**Algoritmning ta'rifi:**

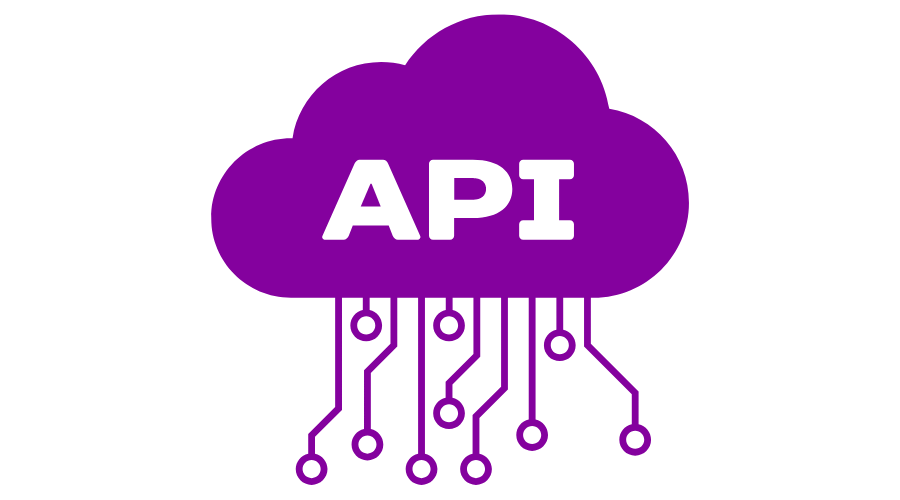


Algoritm - bu muammoni hal qilish yoki ma'lum bir vazifani bajarish uchun aniq belgilangan qadamlar ketma-ketligi. U kompyuter dasturlarining asosi bo'lib, dasturga nima qilish kerakligini va qanday tartibda bajarish kerakligini aytib beradi. Algoritmlar kirish ma'lumotlarini oladi va ularni kerakli natijaga aylantirish uchun bosqichma-bosqich ko'rsatmalar to'plamini bajaradi.

**Algoritmlarning asosiy xususiyatlari:**

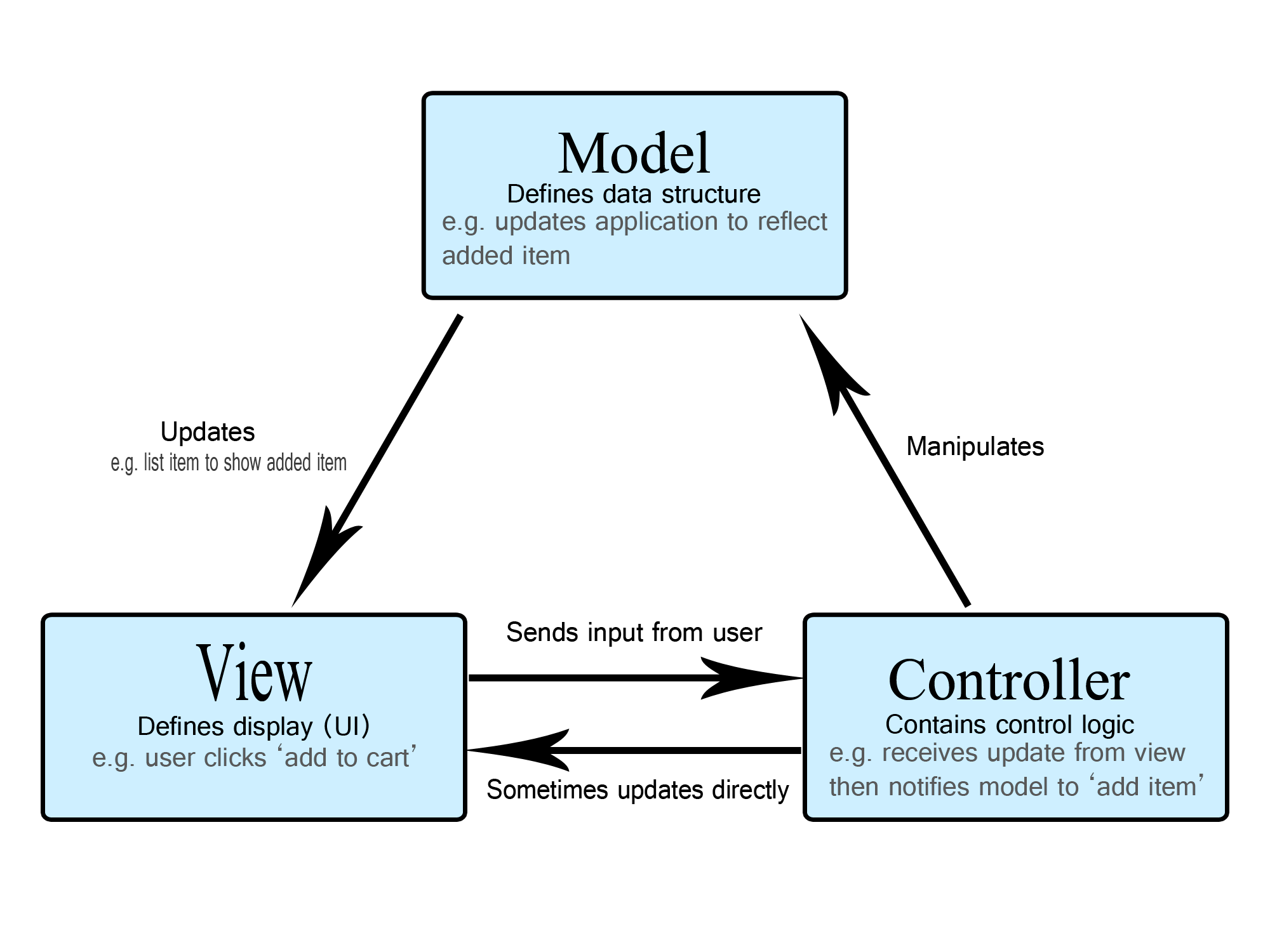
* **Aniq**: Har bir qadam aniq va tushunarli bo'lishi kerak.
* **Tugatish**: Algoritm cheklangan miqdordagi qadamlardan so'ng to'xtashi kerak.
* **Samarali**: Algoritm muammoni imkon qadar tez va samarali hal qilishi kerak.
* **Kirish**: Algoritm kirish ma'lumotlarini qabul qilishi mumkin.
* **Chiqish**: Algoritm kirish ma'lumotlarini qayta ishlash natijasida chiqish ma'lumotlarini ishlab chiqarishi kerak.

Biz NewEra cash carry uchun ASP .NET Core API tayyorlaymiz. Quyida jarayonlarni keldiramiz:  
API so'rovlarni ko'rib chiqish va tegishli javoblarni taqdim etish uchun aniq jarayonni kuzatish orqali ishlaydi. So'rov yuborilganda, API birinchi navbatda uning to'g'ri qabul qilinishini ta'minlaydi va so'rovning URL manzili va HTTP usuli (GET, POST, PUT yoki DELETE kabi) asosida mo'ljallangan oxirgi nuqtani aniqlaydi.



Keyinchalik, JWT Bearer tokeni bilan autentifikatsiya hisob maʼlumotlarini tekshirish orqali soʻrov vakolatli manbadan ekanligini tekshiradi. Shuningdek, foydalanuvchi so'ralgan manbaga kirish uchun zarur ruxsatlarga ega ekanligini tekshiradi. So'rov ushbu xavfsizlik tekshiruvlaridan o'tgandan so'ng, API taqdim etilgan ma'lumotlarni tekshdiradi va barcha kerakli maydonlar mavjudligini va to'g'ri formatlanganligini ta'minlaydi.

Tekshirishdan so'ng, API so'rovni tegishli boshqaruvchiga va harakat usuliga yo'naltirish orqali qayta ishlaydi. So'rovning maqsadiga qarab, bu bosqich ma'lumotlar bazasi yoki boshqa ma'lumotlar manbalari bilan o'zaro aloqani o'z ichiga olishi mumkin. Misol uchun, Admin API so'rovda taqdim etilgan Product ma'lumotlari asosida yozuvlarni olishi, yasashi, yangilashi yoki o'chirishi mumkin. Agar customer bo’lsa faqat Mahsulot ma’lumotlarini ko’ra oladi.



Kerakli operatsiyalar bajarilgandan so'ng, MVC mijoz uchun javob tayyorlaydi. Bu javob JSON kabi tuzilgan formatdagi maʼlumotlarni oʻz ichiga oladi va soʻrov muvaffaqiyatli yoki xatolik yuz berganligini koʻrsatish uchun HTTP holat kodini oʻz ichiga oladi (masalan, muvaffaqiyat uchun 200, notoʻgʻri soʻrov uchun 400, ruxsatsiz kirish uchun 401 va hokazo). .

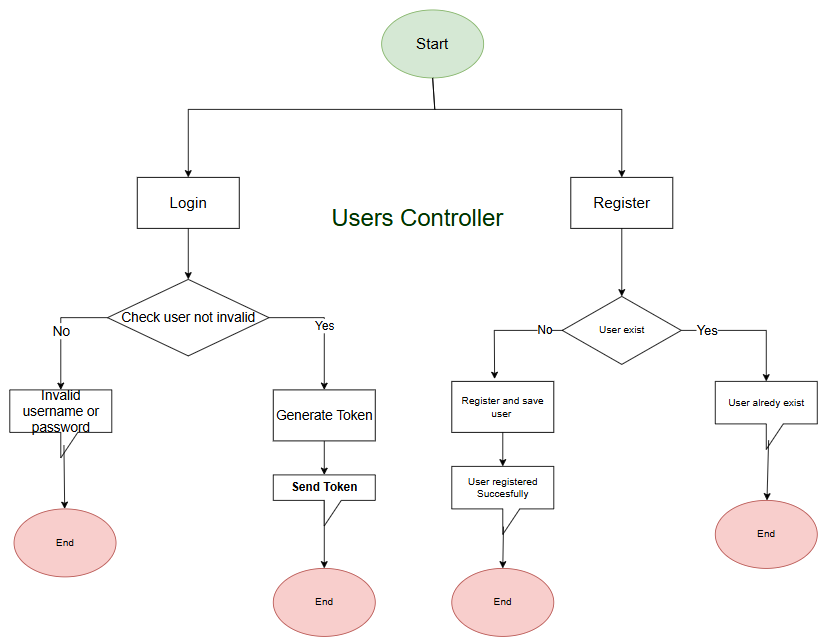
Nihoyat, API javobni mijozga yuboradi va so'rov va javob tafsilotlarini qayd qiladi. Ushbu Request Response recordlari Middleware yordamida fayllarga saqlanadi. Ushbu jarayon API so'rovlarni samarali va xavfsiz tarzda ko'rib chiqishni ta'minlaydi.

**A.M1**

Ushbu bo'limda .NET Core Web API loyihasini yaratish va uni amalga oshirish jarayoni batafsil ko'rib chiqiladi. Loyiha ma'lumotlar bazasi bilan ishlash, JWT autentifikatsiyasini amalga oshirish, API nuqtalarini yaratish va birlik sinovlarini yozish kabi bosqichlarni o'z ichiga oladi.

Loyihada Products, Orders va Users kabi asosiy obyektlar bilan ishlash imkoniyati taqdim etiladi. Har bir obyekt uchun CRUD (Create, Read, Update, Delete) operatsiyalarini bajaruvchi API nuqtalari yasaldi.

Users Controllerning bosqichlari:



Bu diagram foydalanuvchi boshqaruvchisining ("Users Controller") ishlash jarayonini tasvirlaydi. Diagramda ikki asosiy jarayon ko‘rsatilgan: **Login (Kirish)** va **Register (Ro‘yxatdan o‘tish)**. Quyida diagramdagi har bir jarayon bosqichma-bosqich tasvirlangan:

**1. Boshlanish (Start)**

Jarayon boshlanish nuqtasi. Bu yerda foydalanuvchi Login yoki Register operatsiyasini tanlashi kerak.

**2. Login (Kirish) Jarayoni**

* **Loginni tanlash:** Foydalanuvchi kirish jarayonini tanlaydi.
* **Check user not invalid (Foydalanuvchini tekshirish):** Kiritilgan foydalanuvchi ma’lumotlari (foydalanuvchi nomi va parol) tekshiriladi.
  + **No (Xato):** Agar foydalanuvchi nomi yoki parol noto‘g‘ri bo‘lsa, “Invalid username or password” (Noto‘g‘ri foydalanuvchi nomi yoki parol) xabar ko‘rsatiladi va jarayon tugaydi.
  + **Yes (To‘g‘ri):** Agar foydalanuvchi ma’lumotlari to‘g‘ri bo‘lsa:
    - **Generate Token (Token yaratish):** Foydalanuvchi uchun autentifikatsiya tokeni yaratiladi.
    - **Send Token (Token yuborish):** Token foydalanuvchiga yuboriladi va jarayon tugaydi.

**3. Register (Ro‘yxatdan o‘tish) Jarayoni**

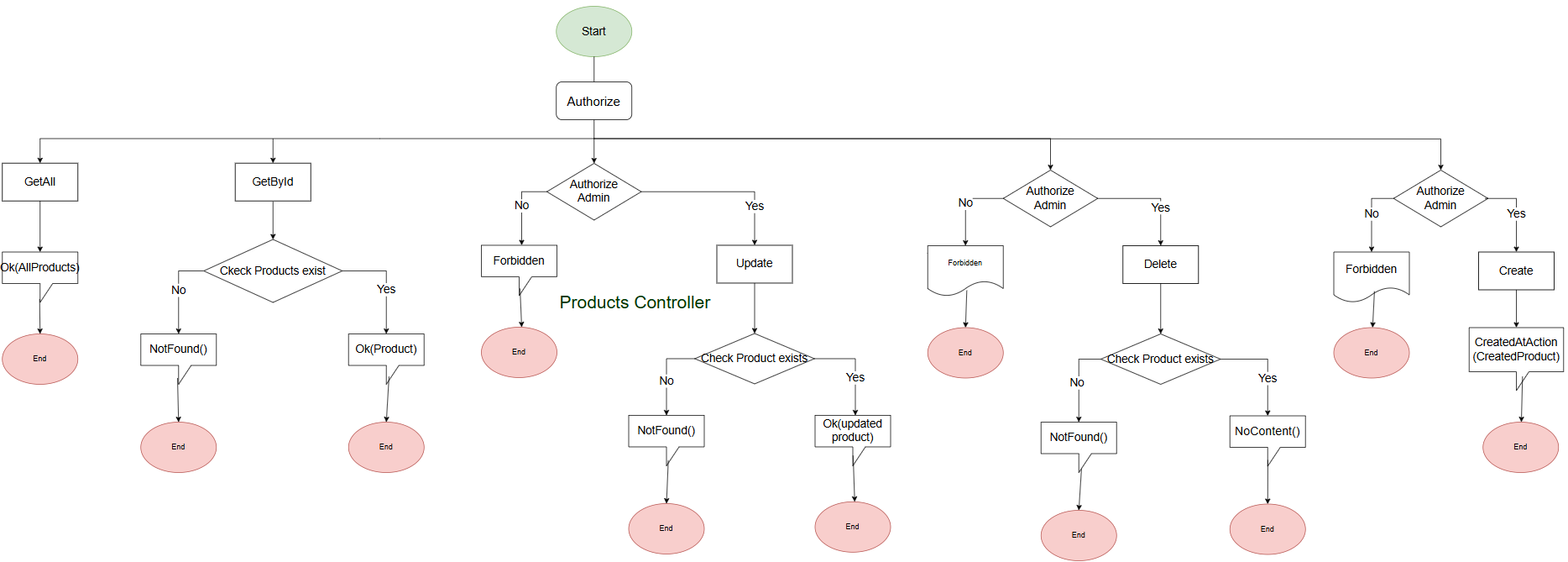
* **Registerni tanlash:** Foydalanuvchi ro‘yxatdan o‘tish jarayonini tanlaydi.
* **User exist (Foydalanuvchi mavjudligini tekshirish):**
  + **Yes (Mavjud):** Agar foydalanuvchi allaqachon tizimda mavjud bo‘lsa, “User already exist” (Foydalanuvchi allaqachon mavjud) xabar ko‘rsatiladi va jarayon tugaydi.
  + **No (Mavjud emas):** Agar foydalanuvchi mavjud bo‘lmasa:
    - **Register and save user (Foydalanuvchini ro‘yxatga olish va saqlash):** Foydalanuvchi tizimga muvaffaqiyatli ro‘yxatdan o‘tkaziladi.
    - **User registered successfully (Foydalanuvchi muvaffaqiyatli ro‘yxatdan o‘tdi):** Xabar ko‘rsatiladi va jarayon tugaydi.

**Diagramning umumiy xususiyatlari:**

* **To‘g‘ri oqim:** Jarayon qadam-baqadam oqimlarni ko‘rsatadi (masalan, token yaratish, foydalanuvchini tekshirish).
* **Xatoliklarni qayta ishlash:** Har bir jarayonda xato yuz berganda tegishli xabarlar ko‘rsatiladi (masalan, foydalanuvchi mavjudligi yoki noto‘g‘ri ma’lumotlar).
* **Muvaffaqiyatli yakunlanish:** Har bir jarayon muvaffaqiyatli tugagach, foydalanuvchiga tegishli javob taqdim etiladi.

Bu diagram foydalanuvchilarni autentifikatsiya qilish va ro‘yxatdan o‘tishni tushunishni osonlashtiradi.

Products Controllerning bosqichlari:



Ushbu diagram **Products Controller** (Mahsulotlar boshqaruvchisi) jarayonlarini ifodalaydi. U mahsulotlarni ko‘rish, qo‘shish, yangilash, o‘chirish va ularni tekshirish bo‘yicha asosiy amallarni ko‘rsatadi. Quyida diagramning bosqichma-bosqich tafsiloti keltirilgan:

**Boshlanish (Start)**

* Barcha jarayon **Authorize (Avtorizatsiya)** qadamidan boshlanadi. Bu yerda foydalanuvchi avtorizatsiya qilinishi kerak.

**1. GetAll (Barcha mahsulotlarni olish)**

* **GetAll:** Foydalanuvchi barcha mahsulotlar ma’lumotlarini so‘raydi.
* **Ok(AllProducts):** Agar jarayon muvaffaqiyatli bo‘lsa, barcha mahsulotlar qaytariladi va jarayon tugaydi.

**2. GetById (Mahsulotni ID bo‘yicha olish)**

* **GetById:** Foydalanuvchi mahsulotni ID orqali so‘raydi.
* **Check Products exist (Mahsulot mavjudligini tekshirish):**
  + **No:** Agar mahsulot topilmasa, NotFound() (Mahsulot topilmadi) xabari qaytariladi va jarayon tugaydi.
  + **Yes:** Agar mahsulot mavjud bo‘lsa, Ok(Product) (Mahsulot ma’lumotlari) qaytariladi va jarayon tugaydi.

**3. Update (Yangilash)**

* **Authorize Admin:** Foydalanuvchining admin huquqlari bor-yo‘qligi tekshiriladi.
  + **No:** Agar foydalanuvchi admin bo‘lmasa, Forbidden (Ruxsat yo‘q) xabari qaytariladi va jarayon tugaydi.
  + **Yes:** Foydalanuvchi admin bo‘lsa:
    - **Check Product exists:** Mahsulot mavjudligi tekshiriladi.
      * **No:** Mahsulot topilmasa, NotFound() qaytariladi va jarayon tugaydi.
      * **Yes:** Agar mahsulot mavjud bo‘lsa, yangilangan mahsulot qaytariladi (Ok(updated product)), va jarayon tugaydi.

**4. Delete (O‘chirish)**

* **Authorize Admin:** Foydalanuvchining admin ekanligi tekshiriladi.
  + **No:** Agar foydalanuvchi admin bo‘lmasa, Forbidden xabari qaytariladi.
  + **Yes:** Agar admin bo‘lsa:
    - **Check Product exists:** Mahsulot mavjudligi tekshiriladi.
      * **No:** Mahsulot topilmasa, NotFound() xabari qaytariladi.
      * **Yes:** Mahsulot muvaffaqiyatli o‘chirilgan bo‘lsa, NoContent() qaytariladi va jarayon tugaydi.

**5. Create (Yaratish)**

* **Authorize Admin:** Foydalanuvchining admin ekanligi tekshiriladi.
  + **No:** Agar foydalanuvchi admin bo‘lmasa, Forbidden xabari qaytariladi.
  + **Yes:** Agar admin bo‘lsa:
    - **CreatedAtAction(CreatedProduct):** Yangi mahsulot muvaffaqiyatli yaratilsa, tegishli javob qaytariladi va jarayon tugaydi.

**Diagramning Umumiy Xususiyatlari:**

1. **Autorizatsiya Tekshiruvi:** Har bir jarayon foydalanuvchining huquqlarini tekshiradi (Admin yoki oddiy foydalanuvchi).
2. **Xatoliklar Bilan Ishlash:** Mahsulot topilmasa yoki foydalanuvchi ruxsati bo‘lmasa, mos javoblar qaytariladi (Forbidden, NotFound()).
3. **Muvaffaqiyatli Tugatish:** Jarayonlar muvaffaqiyatli yakunlanganda tegishli ma’lumotlar qaytariladi (Ok(), NoContent(), CreatedAtAction()).

Bu diagram mahsulotlar boshqaruvi jarayonlarining strukturasini oson tushunish uchun mo‘ljallangan.

Orders Controllerning bosqichlari:



Ushbu diagram **Orders Controller** (Buyurtmalar boshqaruvchisi) funksionalligini ifodalaydi. Buyurtmalar bilan ishlash jarayonlarini, jumladan buyurtmalarni ko‘rish, yaratish, yangilash va o‘chirish jarayonlarini ko‘rsatadi. Quyida diagramning batafsil tushuntirishi berilgan:

**Boshlanish (Start)**

* Jarayon **Authorize** (avtorizatsiya) orqali boshlanadi. Foydalanuvchining huquqlari tekshiriladi.

**1. GetAll (Barcha buyurtmalarni olish)**

* **GetAll:** Foydalanuvchi barcha buyurtmalar ma’lumotlarini so‘raydi.
* **Ok(AllProducts):** Agar muvaffaqiyatli bo‘lsa, barcha buyurtmalar qaytariladi va jarayon tugaydi.

**2. GetById (ID bo‘yicha buyurtmani olish)**

* **GetById:** Foydalanuvchi ma’lum ID bo‘yicha buyurtma ma’lumotlarini so‘raydi.
* **Check Order exist (Buyurtma mavjudligini tekshirish):**
  + **No:** Buyurtma topilmasa, NotFound() qaytariladi va jarayon tugaydi.
  + **Yes:** Buyurtma topilsa, Ok(Order) qaytariladi va jarayon tugaydi.

**3. GetCustomerOrders (Mijoz buyurtmalarini olish)**

* **GetCustomerOrders:** Ma’lum bir mijozning buyurtmalari so‘raladi.
* **Check Orders exist for Customer (Mijoz buyurtmalarining mavjudligi tekshiriladi):**
  + **No:** Buyurtmalar topilmasa, NotFound() qaytariladi va jarayon tugaydi.
  + **Yes:** Mijozning barcha buyurtmalari Ok(CustomerOrders) orqali qaytariladi va jarayon tugaydi.

**4. Create (Buyurtma yaratish)**

* **Create:** Yangi buyurtma yaratiladi.
* **Check Customer exists (Mijoz mavjudligini tekshirish):**
  + **No:** Mijoz topilmasa, NotFound() xabari qaytariladi.
  + **Yes:** Buyurtma yaratilsa, CreatedAtAction(CreatedOrder) qaytariladi va jarayon tugaydi.

**Diagramning Umumiy Xususiyatlari:**

1. **Avtorizatsiya tekshiruvi:** Har bir qadam foydalanuvchining huquqlarini tekshirish bilan boshlanadi.
2. **Xatolik bilan ishlash:** Agar buyurtma yoki mijoz mavjud bo‘lmasa, mos javob qaytariladi (NotFound()).
3. **Muvaffaqiyatli jarayonlar:** Muvaffaqiyatli yakunlangan jarayonlarda tegishli ma’lumotlar qaytariladi (Ok(), CreatedAtAction(), NoContent()).

Bu diagram buyurtmalar boshqaruvi jarayonlarini yaxlit va tizimli tarzda tushunishga yordam beradi.

Ushbu bo'limda .NET Core Web API loyihasini yasash va uni amalga oshirishning barcha asosiy bosqichlari ko'rib chiqildi. Ma'lumotlar bazasi bilan ishlash, JWT autentifikatsiya mexanizmini qo'llash, API nuqtalarini yaratish va birlik sinovlarini yozish kabi muhim jihatlarga e'tibor qaratildi.

Natijada, foydalanuvchilarga Products, Orders va Users obyektlari bilan ishlash imkonini beruvchi to'liq funksional Web API loyihasi yaratildi. Loyiha xavfsizlikni ta'minlash uchun JWT autentifikatsiyasidan foydalanadi va kodning sifati va ishonchliligini oshirish uchun birlik sinovlari bilan ta'minlangan.

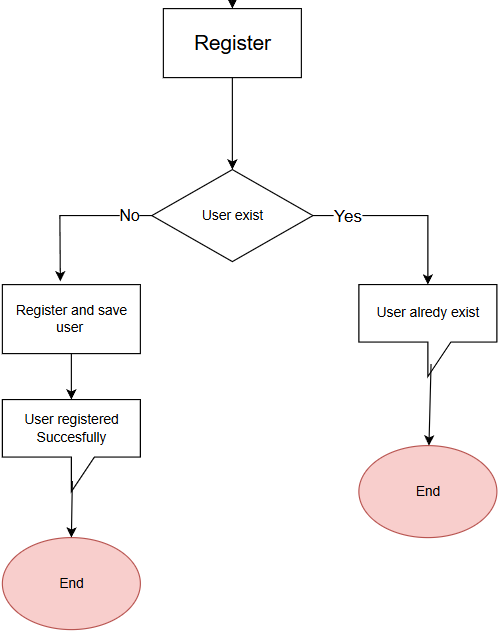
Ushbu loyiha .NET Core platformasida Web API yaratishning asosiy tamoyillarini o'rganish va amalda qo'llash uchun yaxshi misol bo'la oladi.

**A.D1**

**Bu bo’limda biz yuqoridagi algoritmni koddagi holatiga solishtirib chiqamiz:**

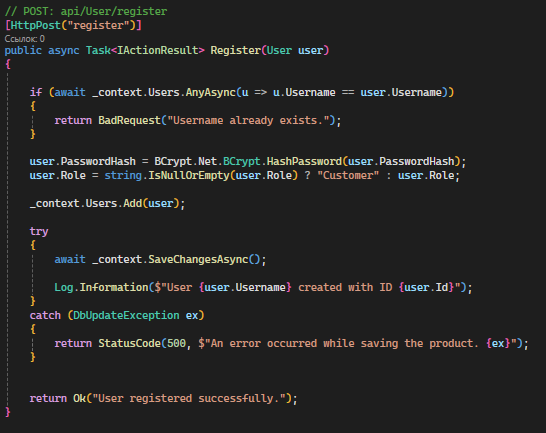
**Users Controllerni diagramma bilan koddagi holatini solishtirish:**

**Register :**



**Bu rasimda User registratsiya bo’limini tanlagan holatdagi registratsiya algoritmi tasvirlangan.**

**Endi ushbu jarayonning koddagi ko’rinishi:**



**Register (Ro‘yxatdan o‘tish) Jarayoni**

**• Registerni tanlash: Foydalanuvchi ro‘yxatdan o‘tish jarayonini tanlaydi.**

**• User exist (Foydalanuvchi mavjudligini tekshirish):**

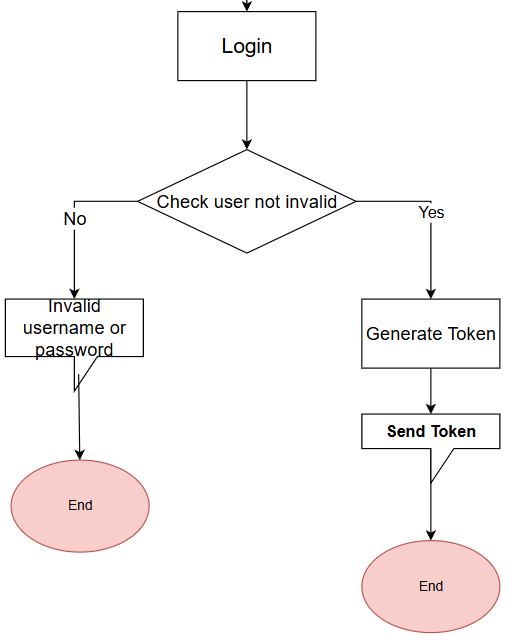
**o Yes (Mavjud): Agar foydalanuvchi allaqachon tizimda mavjud bo‘lsa, “User already exist” (Foydalanuvchi allaqachon mavjud) xabar ko‘rsatiladi va jarayon tugaydi.**

**o No (Mavjud emas): Agar foydalanuvchi mavjud bo‘lmasa:**

**Register and save user (Foydalanuvchini ro‘yxatga olish va saqlash): Foydalanuvchi tizimga muvaffaqiyatli ro‘yxatdan o‘tkaziladi.**

**User registered successfully (Foydalanuvchi muvaffaqiyatli ro‘yxatdan o‘tdi): Xabar ko‘rsatiladi va jarayon tugaydi.**

**Login Diagrammasi:**



**Koddagi ko’rinishi:**

A computer screen shot of code

AI-generated content may be incorrect.

**• Check user not invalid (Foydalanuvchini tekshirish): Kiritilgan foydalanuvchi ma’lumotlari (foydalanuvchi nomi va parol) tekshiriladi.**

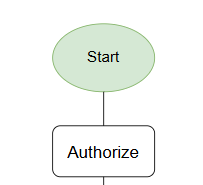
**• No (Xato): Agar foydalanuvchi nomi yoki parol noto‘g‘ri bo‘lsa, “Invalid username or password” (Noto‘g‘ri foydalanuvchi nomi yoki parol) xabar ko‘rsatiladi va jarayon tugaydi.**

**• Yes (To‘g‘ri): Agar foydalanuvchi ma’lumotlari to‘g‘ri bo‘lsa:**

**Generate Token (Token yaratish): Foydalanuvchi uchun autentifikatsiya tokeni yaratiladi.**

**• Send Token (Token yuborish): Token foydalanuvchiga yuboriladi va jarayon tugaydi.**

**Endi Products Controllerga kelsak:**



**Rasimdagi Avtorizatsiya qilishning koddagi ko’rinishi:**

A screen shot of a computer program

AI-generated content may be incorrect.

**Get all:**

A diagram of a person's face

AI-generated content may be incorrect.

**Koddagi ko’rinishi:**

A computer screen shot of text

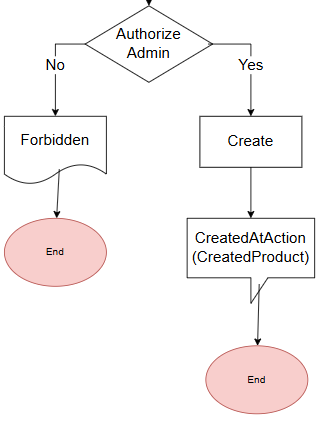
AI-generated content may be incorrect.

**GetAll (Barcha mahsulotlarni olish)**

**• GetAll: Foydalanuvchi barcha mahsulotlar ma’lumotlarini so‘raydi.**

**• Ok(AllProducts): Agar jarayon muvaffaqiyatli bo‘lsa, barcha mahsulotlar qaytariladi va jarayon tugaydi**

**Product create:**



**Koddagi ko’rinishi:**

A screen shot of a computer program

AI-generated content may be incorrect.

**Create**

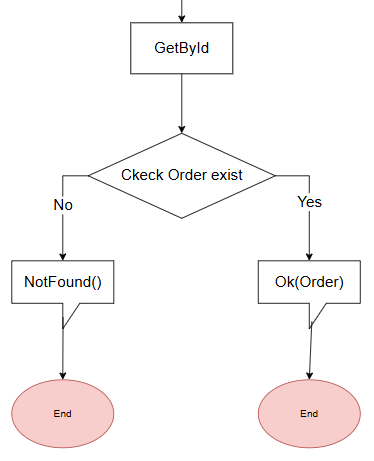
**• Authorize Admin: Foydalanuvchining admin ekanligi tekshiriladi.**

**• No: Agar foydalanuvchi admin bo‘lmasa, Forbidden xabari qaytariladi.**

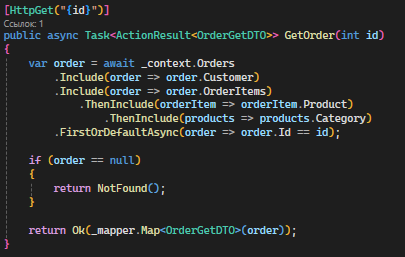
**• Yes: Agar admin bo‘lsa:**

**• CreatedAtAction(CreatedProduct): Yangi mahsulot muvaffaqiyatli yaratilsa, tegishli javob qaytariladi va jarayon tugaydi.**

**Endi Ordersning Diagrammasidagi GetById algoritmi:**



**Kodda esa:**



**GetById (ID bo‘yicha buyurtmani olish)**

**• GetById: Foydalanuvchi ma’lum ID bo‘yicha buyurtma ma’lumotlarini so‘raydi.**

**• Check Order exist (Buyurtma mavjudligini tekshirish. Bu yerda null bilan tenglangan holatda tekshirilyabdi):**

**• No: Buyurtma topilmasa, NotFound() qaytariladi va jarayon tugaydi.**

**• Yes: Buyurtma topilsa, Ok(Order) qaytariladi va jarayon tugaydi.**

**Endi GetCustomerOrdersga kelsak:**

A diagram of a customer order

AI-generated content may be incorrect.

**Ushbu algoritmning koddagi ko’rinishi:**



**GetCustomerOrders (Mijoz buyurtmalarini olish)**

**• GetCustomerOrders: Ma’lum bir mijozning buyurtmalari so‘raladi.**

**• Check Orders exist for Customer (Mijoz buyurtmalarining mavjudligi tekshiriladi. Bu yerda Any() bilan tekshirilgan):**

**o No: Buyurtmalar topilmasa, NotFound() qaytariladi va jarayon tugaydi.**

**o Yes: Mijozning barcha buyurtmalari Ok(CustomerOrders) orqali qaytariladi va jarayon tugaydi**

**LO2**

**A.P2**

OOP (Obyektga Yo'naltirilgan Dasturlash)

OOP - bu dasturlash paradigmasi bo'lib, u dasturni obyektlarga ajratish orqali murakkab muammolarni hal qilishga yordam beradi. Obyektlar esa o'z navbatida ma'lumotlar (xususiyatlar) va ular ustida amallarni bajaruvchi funksiyalar (metodlar)ni o'z ichiga oladi.

OOPning asosiy xususiyatlari:

**Abstraction** - bu murakkab tizimni soddalashtirib, faqat muhim qismlarini ko'rsatish konseptsiyasidir. Bu huddi classlarning contractini tuzib berishga o’xshaydi. Keling, buni o’zimizning C# misolimizda tushuntirib beraman:



Abstract klasslarning asosiy xususiyatlari:

1. A black rectangular object with white text

   AI-generated content may be incorrect.Abstract klassdan obyekt yasab bo'lmaydi:
2. Abstract klassda oddiy va abstract metodlar bo'lishi mumkin:

* Oddiy metodlar tayyor implementation bilan keladi
* Abstract metodlar faqat e'lon qilinadi, ularni farzand klasslar implement qilishi shart

1. A screen shot of a computer screen

   AI-generated content may be incorrect.Abstract klass boshqa klassga meros bo'lib o'tishi mumkin:
2. Abstract klassdan meros olgan klass barcha abstract metodlarni implement qilishi shart:

A screen shot of a computer code

AI-generated content may be incorrect.

Abstract klasslar qachon ishlatiladi:

1. Umumiy xususiyat va metodlarni bir joyda to'plash kerak bo'lganda
2. Bir xil tipga ega obyektlar uchun qat'iy struktura yaratish kerak bo'lganda
3. Ba'zi metodlarni majburiy implement qilish kerak bo'lganda
4. Kod takrorlanishini kamaytirish kerak bo'lganda

Abstract klasslar OOP ning muhim qismi hisoblanadi va kodni toza va tartibli saqlashga yordam beradi.

**Incapsulation**: ma’lumotlarni yashirish va ularga kirishni nazorat qilishni ta’minlaydi. Bu tamoyil orqali obyektning ichki tafsilotlari (ma’lumotlar va mantiq) tashqaridan yashiriladi va faqat kerakli bo‘lgan funksionallarni ochib beradi.

Asosiy g‘oyasi

Ma’lumotlarni yashirish (Data hiding): Sinf (class) ichidagi o‘zgaruvchilarni yoki xususiyatlarni (fields) bevosita tashqi koddan himoya qilish uchun private yoki boshqa kirish darajalari (access modifiers) bilan belgilash.

Kirishni boshqarish: Yashirilgan ma’lumotlarga faqat kerakli funksiyalar (methods) yoki xususiyatlar (properties) orqali kirish imkoniyatini berish.

Quyidagi misolda oddiy inkapsulyatsiya tushunchasi tushuntirilgan:



**Kod tahlili**

1. **private**:
   * balance o‘zgaruvchisi xususiy (private), ya’ni u faqat BankAccount sinfi ichidan kirilishi mumkin.
2. **public metodlar**:
   * GetBalance(), Deposit() va Withdraw() funksiyalari orqali tashqaridan ma’lumotlarga xavfsiz kirish ta’minlanadi.
3. **Nazorat mexanizmi**:
   * Depozit yoki pul yechish jarayonlarida noto‘g‘ri yoki salbiy qiymatlar kiritilishining oldi olinadi.

**Inkapsulyatsiya afzalliklari**

1. **Ma’lumotlarni himoya qilish**: Sinf ichidagi xususiy maydonlar noto‘g‘ri ishlatilishdan himoyalangan.
2. **Qulaylik va boshqaruv**: Faqat kerakli funksiyalarni ochish orqali murakkablik kamaytiriladi.
3. **Moslashuvchanlik**: Sinfni o‘zgartirgan holda boshqa qismga ta’sir qilmasdan ishlashni davom ettirish mumkin.

**Inheritance** (meros olish) - C# tilidagi obyektga yo‘naltirilgan dasturlash (OOP) tamoyillaridan biri bo‘lib, u bir sinfning boshqa sinfdan xususiyatlar va funksiyalarni meros qilib olishini ta’minlaydi. Meros olish orqali kodni qayta ishlatish va dasturiy ta’minotni yanada qulayroq strukturalash mumkin bo‘ladi.

**Asosiy tushunchalar**

1. **Base Class (asosiy sinf)**:
   * Bu sinf boshqa sinflar uchun umumiy xususiyatlar va funksiyalarni taqdim etadi.
2. **Derived Class (merosxo‘r sinf)**:
   * Bu sinf asosiy sinfdan barcha xususiyatlar va metodlarni meros qilib oladi va qo‘shimcha funksiyalarni qo‘shishi yoki mavjudlarini o‘zgartirishi mumkin.
3. **protected**:
   * Asosiy sinfdagi o‘zgaruvchi yoki metod faqat o‘sha sinf va undan meros olgan sinflar tomonidan ishlatilishi mumkin.
4. **virtual va override**:
   * Asosiy sinfdagi metodni merosxo‘r sinfda o‘zgartirish uchun ishlatiladi.

Quyidagi misolda meros olish va uning asosiy tamoyillari ko‘rsatilgan:



**Kod tahlili**

1. **Asosiy sinf (Animal)**:
   * Name xususiyati va Speak() metodi barcha hayvonlar uchun umumiydir.
   * Speak() metodi **virtual** bo‘lib, merosxo‘r sinflarda qayta aniqlanishi mumkin.
2. **Merosxo‘r sinflar (Dog va Cat)**:
   * Animal sinfidagi barcha xususiyatlar va funksiyalarni avtomatik ravishda oladi.
   * Speak() metodini **override** qilib, o‘ziga xos ovoz chiqaradi.
3. **base kalit so‘zi**:
   * Merosxo‘r sinfda asosiy sinf konstruktoriga murojaat qilish uchun ishlatiladi.

**Natija**

Kodni ishga tushirganda quyidagi chiqish hosil bo‘ladi:

A black rectangular object with blue and green text

AI-generated content may be incorrect.

**Meros olishning afzalliklari**

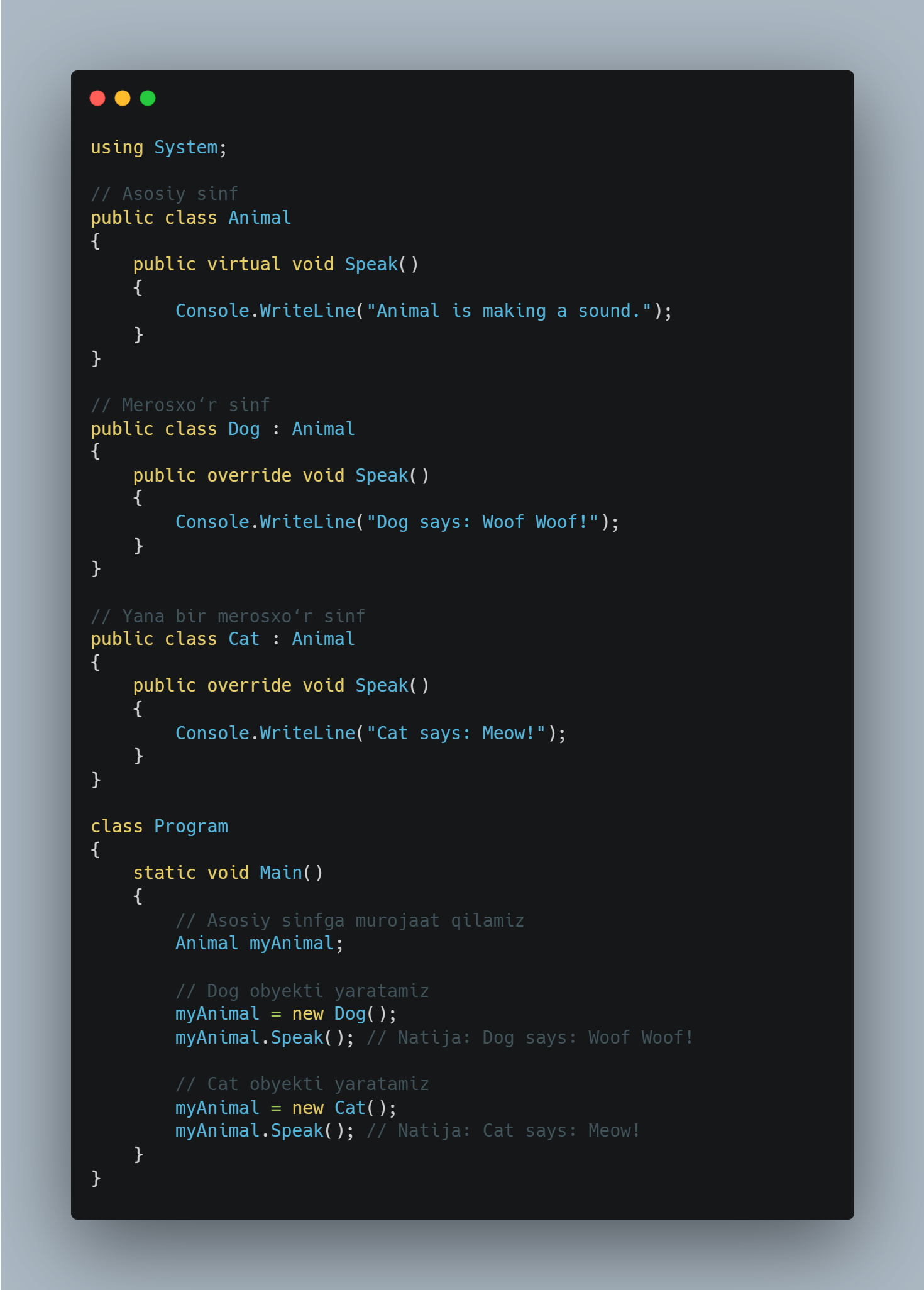
1. **Kodni qayta ishlatish**: Umumiy funksiyalarni qayta yozmasdan foydalanish imkonini beradi.
2. **Polimorfizm**: Bir xil metodni turli sinflarda o‘ziga xos tarzda ishlatish imkoniyatini taqdim etadi.
3. **Kirishni soddalashtirish**: Asosiy sinfdagi barcha metodlar va xususiyatlar merosxo‘r sinflarda darhol mavjud bo‘ladi.

**Polimorfizm** - bu obyektga yo‘naltirilgan dasturlash (OOP) tamoyillaridan biri bo‘lib, bir xil nomdagi metod yoki funksiyani turli sinflarda turli usullarda ishlatish imkoniyatini beradi, asosiy sinfdagi metodni merosxo‘r sinfda o‘zgartirish orqali o‘ziga xos funksional taqdim etish mumkin. Polimorfizm grek tilidan tarjima qilganda "ko‘p shakllilik" degan ma’noni anglatadi.

**Asosiy tushunchalar**

1. **virtual kalit so‘zi**:
   * Asosiy sinfdagi metodni qayta aniqlashga ruxsat berish uchun ishlatiladi.
2. **override kalit so‘zi**:
   * Merosxo‘r sinfda asosiy sinfdagi virtual metodni o‘zgartirish uchun ishlatiladi.

Quyidagi misolda asosiy sinf va merosxo‘r sinflar yordamida polimorfizmni ko‘rsatamiz:



**Kodning ishlash tamoyili:**

1. **Asosiy sinf (Animal)**:
   * Speak metodi **virtual** deb belgilangan, bu uning merosxo‘r sinflarda qayta aniqlanishiga ruxsat beradi.
2. **Merosxo‘r sinflar (Dog va Cat)**:
   * override kalit so‘zi yordamida Speak metodini qayta aniqlaydi va o‘ziga xos funksiyalarni qo‘shadi.
3. **Polimorfizm**:
   * Asosiy sinf turidagi (Animal) obyekt, haqiqiy obyekti (masalan, Dog yoki Cat) qanday sinfga tegishli ekaniga qarab o‘zini turli xil tutadi.

Xullas virtual va override asosiy sinfdagi metodni ixtiyoriy ravishda o‘zgartirishga imkon beradi.

**A.M2**

Dasturlash paradigmalari - bu dasturlarni tuzish va yozish uchun ishlatiladigan yondashuvlar to‘plamidir. Har bir paradigma ma’lum bir muammoni hal qilishga yo‘naltirilgan bo‘lib, o‘ziga xos printsiplar va usullarga ega. Uchta asosiy paradigma – protsessual, obyektga yo‘naltirilgan, va hodisaga asoslangan yondashuvlar – dasturiy ta'minotni yaratishda keng qo‘llaniladi. Ushbu paradigmalarning taqqoslanishi dasturiy yechimni tanlashda va tizimning samaradorligini oshirishda muhim ahamiyat kasb etadi.

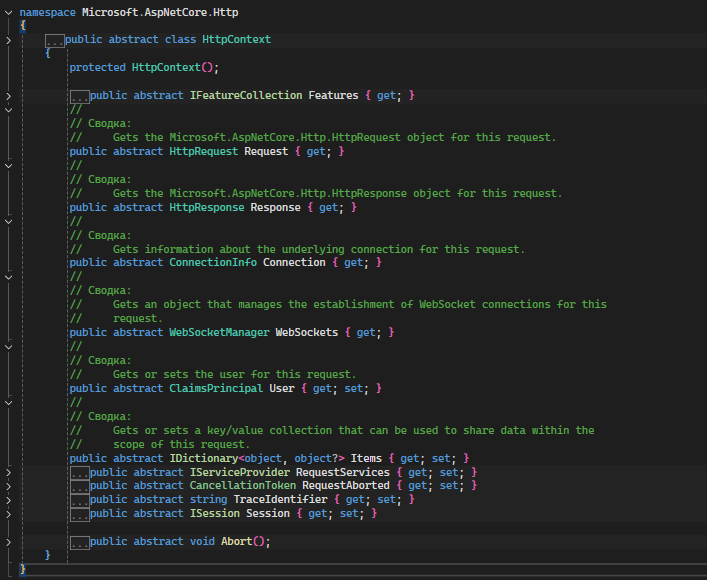
Biz Dasturlash paradigmalarini kodda ko’rib chiqamiz ya’ni OOPni kodda taqqoslaymiz:

Inheritance kodda juda ko’p marotaba qo’llanilgan masalan:

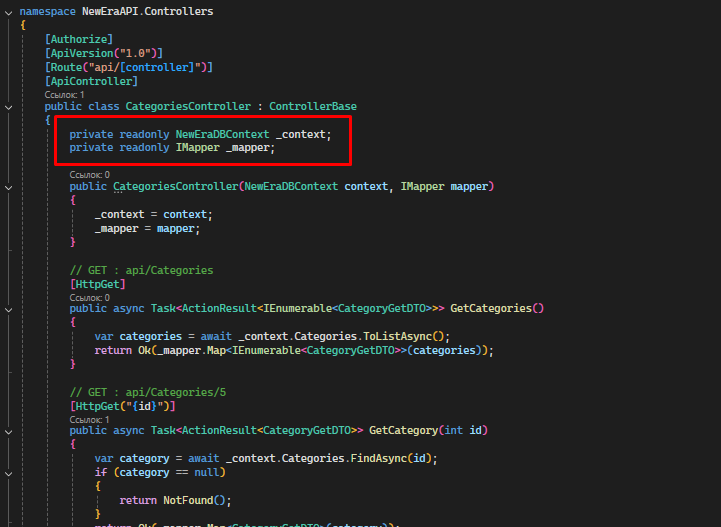


Bu yerda bizning classimiz Efga tegishli DbContext classidan voris olayabdi.

Abstraction biz yozgan APIda deyarli ishlatilmagan bo’lsada biz ishlatgan framework classining ichida ishlatilgan misol uchun:



Encapsulation bizning kodimizdagi misoli:



Ma'lumotlar va metodlar (xizmatlar) o'zaro bog'lanib, tashqi koddan himoyalangan. Encapsulation, obyektga yo'naltirilgan dasturlashda, ma'lumotlarni va ular bilan ishlash funksiyalarini bitta sinf ichida birlashtirishga yordam beradi, shunda tashqi kod to'g'ridan-to'g'ri ma'lumotlarga kirish ololmaydi, faqatgina metodlar orqali ishlay oladi.

Quyidagi jihatlarni ko'rib chiqaylik:

**Private o'zgaruvchilar**:

* + \_context va \_mapper o'zgaruvchilari **private** bo'lib, ular tashqi sinflardan yoki koddan to'g'ridan-to'g'ri kirishdan himoyalangan.
  + Bu o'zgaruvchilar faqat CategoriesController sinfi ichidagi metodlar orqali ishlatilishi mumkin.

Xulosa qilib aytganda protsessual, obyektga yo‘naltirilgan va hodisaga asoslangan paradigmalarning har biri o‘ziga xos kuchli va zaif tomonlarga ega. Protsessual yondashuv sodda va bevosita boshqaruvni talab qiladigan kichik loyihalar uchun mos bo‘lsa, obyektga yo‘naltirilgan paradigma murakkab tizimlarda qayta foydalaniladigan kod va modullikni ta’minlaydi. Hodisaga asoslangan paradigma esa asosan interaktiv tizimlarda dinamik hodisalarni boshqarish uchun ishlatiladi. Loyihaning tabiati va talablariga qarab, ushbu yondashuvlarning kombinatsiyasi yanada optimal dasturiy yechim yaratishga yordam beradi.

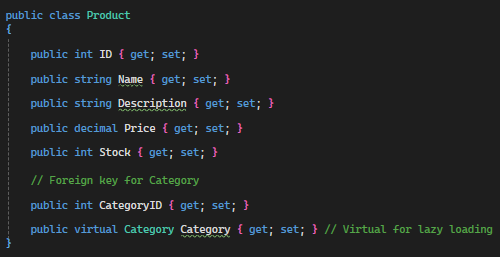
**A.D2**

**OOP (Obyektga Yo‘naltirilgan Dasturlash) Paradigmasi** dastur yaratishda ma'lum bir yondashuvni anglatadi va u tizimni obyektlar va ularning o‘zaro aloqalariga asoslanadi. OOP paradigmalarini amalga oshiradigan dastur manba kodini tanqidiy baholashda quyidagi nuqtalar asosiy e'tiborni jalb qiladi:

**1. Tuzilish va Modullik**

OOP paradigmasi yordamida dastur modulli va kengaytiriladigan bo‘lishi kerak. Har bir obyekt o‘zining xususiyatlari va metodlariga ega bo‘lib, uni alohida modullar sifatida ajratish mumkin. Misol uchun, sizda sinflar (classes) bo‘lishi kerak, va ular orasida mantiqiy aloqalar aniqlanishi kerak.

**Kiritilgan kod misoli:**



**Tanqidiy baho**:

* **Modullik**: Product sinfi mahsulotga oid barcha xususiyatlarni to‘plagan. Bu juda yaxshi, chunki bitta sinf ichida barcha ma'lumotlar saqlanadi va tizimni kengaytirishda osonlik yaratadi. Lekin, agar sinfda ko‘p metodlar bo‘lsa, uni yanada kichikroq sinflarga ajratish kerak bo‘lishi mumkin.
* **Kengaytiriladiganlik**: virtual kalit so‘zi yordamida bu sinf **lazy loading** imkoniyatini beradi. Agar bu sinfning boshqa xususiyatlarini kengaytirish zarur bo‘lsa, sinfni osongina yangilash mumkin.

**2. Encapsulation (Inkapsulyatsiya)**

OOP paradigmalarida inkapsulyatsiya ma'lumotlarning va metodlarning yashirilishi va faqat zarur metodlar orqali ularga kirishni ta'minlashni anglatadi. Bu tizimning xavfsizligini oshiradi va xatoliklar sonini kamaytiradi.

**Kiritilgan kod misoli:**

public int ID { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public decimal Price { get; set; }

public int Stock { get; set; }

**Tanqidiy baho**:

* **Inkapsulyatsiya**: Kodda get va set aksessorlari yordamida sinf xususiyatlari tashqi koddan himoyalangan. Agar o‘zgaruvchilarning maxfiyligini oshirish zarur bo‘lsa, ular **private** yoki **protected** qilib belgilanishi va faqat kerakli metodlar orqali o‘zgartirilishi mumkin.
* **Aksessorlarga ehtiyotkorlik**: Agar Product sinfi juda katta bo‘lsa yoki murakkab xususiyatlar mavjud bo‘lsa, ba'zi xususiyatlar o‘zgartirilishdan oldin tekshiruvlar yoki validatsiyalarni o‘tkazish zarur bo‘lishi mumkin.

**3. Inheritance (Meros olish) va Polimorfizm**

OOP paradigmalarida **inheritance** (meros olish) orqali kodni qayta ishlatish mumkin, **polimorfizm** esa bir xil metodni turli obyektlarda ishlatish imkonini beradi. Bu tamoyillar tizimning kengaytirilishini va xatoliklarni kamaytiradi.

**Kiritilgan kod misoli:**

public virtual Category Category { get; set; }

**Tanqidiy baho**:

* **Meros olish**: Product sinfi Category sinfi bilan bog‘langan. Bu meros olish o‘rniga bog‘lanish (association) orqali amalga oshirilgan. Agar Category sinfi ham boshqa sinflardan meros olsa, bu yondashuvda meros olish tamoyili ishlatilishi mumkin edi.
* **Polimorfizm**: virtual kalit so‘zi orqali polimorfizm amalga oshirilishi mumkin. Masalan, agar Product sinfi turli mahsulot turlarini aks ettirsa, ular uchun alohida metodlar yoki sinflar yaratilib, **override** orqali o‘ziga xos funktsiyalar qo‘shilishi mumkin.

**4. Xususiyatlar va Metodlar**

OOPda metodlar va xususiyatlar ma'lumotlar va ular bilan bog‘liq funktsiyalarni birlashtiradi. Sinfning faqat zarur metodlari va xususiyatlarini taqdim etish tizimni soddalashtiradi va ularning ishlashini yaxshilaydi.

**Kiritilgan kod misoli:**

public int CategoryID { get; set; }

public virtual Category Category { get; set; }

**Tanqidiy baho**:

* **Metodlar**: Product sinfi faqat xususiyatlarni ko‘rsatmoqda. Agar mahsulot bilan bog‘liq amallarni bajarish zarur bo‘lsa, masalan, mahsulotni sotish, zaxirani yangilash yoki narxni hisoblash kabi metodlar qo‘shilishi kerak.
* **Sinfning vazifalari**: Agar sinf faqat xususiyatlarni saqlashga xizmat qilsa, uni **DTO (Data Transfer Object)** sifatida ishlatish mumkin. Agar sinfda biznes mantiqi bo‘lsa, uni boshqa xizmatlar orqali ajratish kerak bo‘ladi.

**5. Qayta ishlatish va Mustahkamlik**

OOP paradigmasi kodni qayta ishlatish va mustahkamligini ta’minlashda yordam beradi. Bir marta yozilgan kodni boshqa joyda foydalanish mumkin bo‘ladi.

**Kiritilgan kod misoli:**

public virtual Category Category { get; set; }

**Tanqidiy baho**:

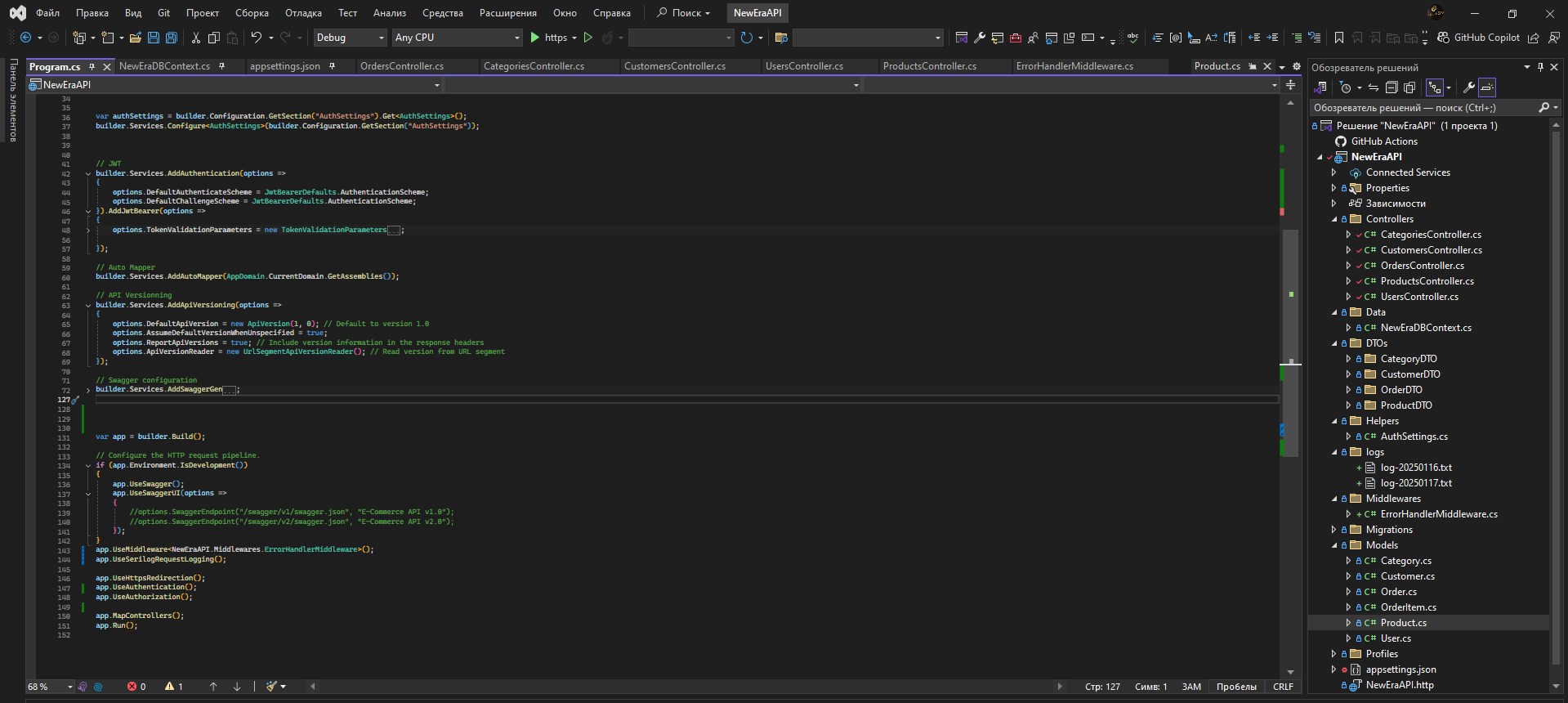
* **Qayta ishlatish**: Product sinfi Category sinfi bilan bog‘langan, bu esa kodni qayta ishlatishni osonlashtiradi. Agar boshqa sinflar ham **Category** sinfini ishlatmoqchi bo‘lsa, ular faqat mavjud sinfni ishlatishi mumkin, yangi kod yozish kerak emas.
* **Mustahkamlik**: Kodning mustahkamligi yuqori, chunki barcha o‘zgarishlar va metodlar faqat zarur sinflarda amalga oshiriladi, va boshqalar ularni bevosita o‘zgartira olmaydi.

**Xulosa**

OOP paradigmalarini amalga oshiradigan dastur manba kodini baholashda yuqoridagi tamoyillarni hisobga olgan holda, kodning tuzilishi va xususiyatlari yaxshi ko‘rinishga ega. Kodda inkapsulyatsiya, modullik, qayta ishlatish va kengaytirilish tamoyillari to‘liq amalga oshirilgan. Biroq, sinfning vazifalarini yanada aniq belgilash va metodlarni qo‘shish tizimni yanada yaxshilashga yordam beradi. Shuningdek, polimorfizm va meros olishni qo‘llashda tizimni yanada mustahkamlash mumkin.

**LO3**

**A.P3**

IDE dan foydalangan holda NewEra Cash Carry API dasturi tuzulib chiqildi. 

Bu yerda Visual studiodan foydalanilgan.

A screenshot of a computer

AI-generated content may be incorrect.

Controllerslar qo’shildi



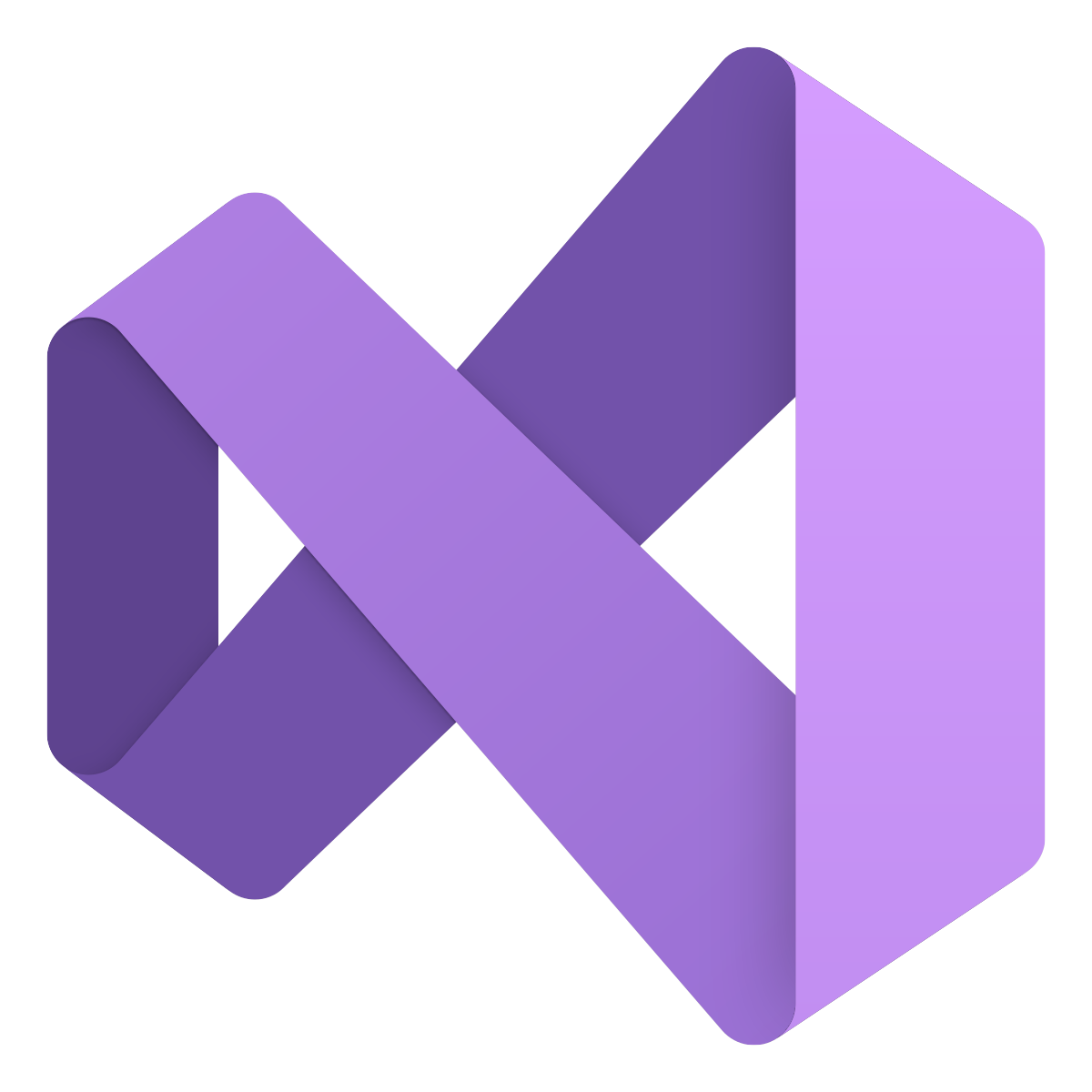
Data ya’ni databse qo’shildi

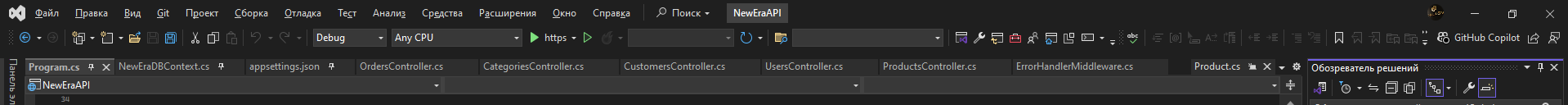
A screenshot of a computer screen

AI-generated content may be incorrect.

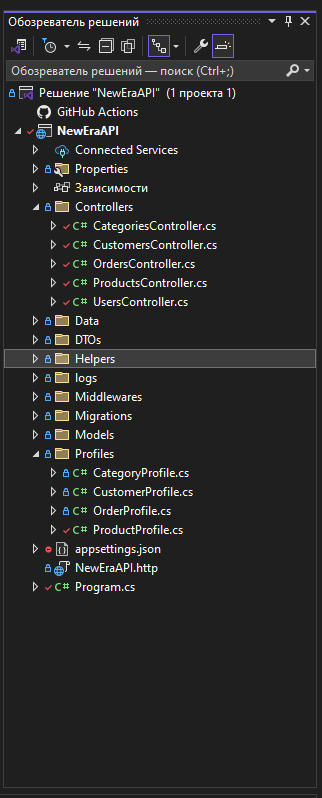
DTO’slar Helperslar vahokazolar qo’shildi.

**A.M3**

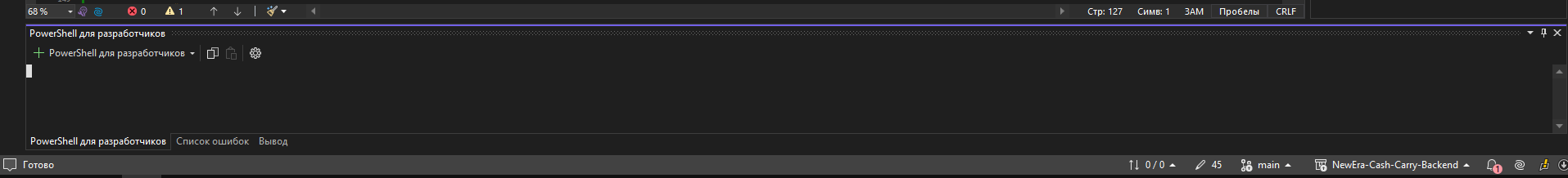
Men Visual Studio IDE sidan foydalandim 



Bu visual studioning Head interface qismi.



Bu yerda VS ning Fayllar ko’rsatish interfeysi.



Bu yerda Console orqali package o’rnatiladi.

Xullas jarayonni boshqarish uchun men uchun juda qulay bo’lgan Visual Studiodan foydalandim.

**A.D3**

Ilovalarni ishlab chiqishda **IDE (Integrated Development Environment)** dan foydalanish va undan foydalanmaslik o‘rtasidagi farqlarni tahlil qilishda, har ikkala yondashuvning afzalliklari va kamchiliklari muhim ahamiyatga ega. Quyida IDE dan foydalanish va undan foydalanmaslikni tanqidiy baholashga doir bir nechta nuqtalar keltirilgan.

**IDE dan foydalanishning afzalliklari:**

1. **Kengaytirilgan funksiyalar**:
   * IDElar (masalan, **Visual Studio**, **IntelliJ IDEA**, **Eclipse**) bir nechta qulay xususiyatlarga ega, jumladan **code completion**, **syntax highlighting**, **refactoring tools**, **debugging tools**, va **intelligent code suggestions**.
   * **Code completion** yordamida yozayotgan kodingizni tezda to‘ldirishingiz mumkin, bu esa yozish vaqtini sezilarli darajada kamaytiradi va koddagi xatoliklarni oldindan aniqlashga yordam beradi.
2. **Xatoliklarni aniqlash va tuzatish (Debugging)**:
   * IDElar bilan birga keladigan **debugging** vositalari yordamida kodni real vaqt rejimida ishga tushirib, xatoliklarni topish va tuzatish osonlashadi. Siz qadam-baqadam bajarilayotgan kodni tekshirishingiz mumkin, bu esa muammolarni aniqlashda juda samarali.
3. **Qulay interfeys va kodni vizualizatsiya qilish**:
   * IDElar ko‘pincha grafikli interfeyslar, fayllar va manbalarni boshqarish, versiya nazorati (masalan, **Git**) va GUI (Graphical User Interface) yaratish imkoniyatlarini taqdim etadi.
   * **Drag-and-drop** dizayn vositalari yordamida GUI yaratish qulay bo‘ladi.
4. **Yuqori integratsiya**:
   * IDElar turli xil vositalarni bir joyga jamlaydi, shuningdek, ular ko‘p tillarda dasturlashni qo‘llab-quvvatlaydi. Bu esa dasturchiga turli tizimlar va platformalarda ishlash imkonini beradi.
5. **Avtomatik testlar va build tizimlari**:
   * IDElar ko‘pincha testlar va avtomatik build tizimlari (masalan, **Maven**, **Gradle**, **MSBuild**) bilan integratsiya qilinadi. Bu dastur ishlab chiqishda sifatni saqlash va versiyalarni boshqarish uchun foydalidir.

**IDE dan foydalanmaslikning afzalliklari:**

1. **Yengillik va minimalizm**:
   * IDElar ko‘p resurslarni talab qilishi mumkin, shu bilan birga, tizimning tezligini sekinlashtiradi. **Text editorlar** (masalan, **VS Code**, **Sublime Text**, **Atom**) ko‘p resurs talab qilmasdan eng oddiy va yengil muhiti taqdim etadi.
2. **Moslashuvchanlik va sozlash imkoniyatlari**:
   * IDElardan foydalanmaslik orqali siz o‘zingizga mos keladigan qulay muhitni yaratishingiz mumkin. Kichik editorlar va komanda satri vositalarini ishlatish orqali minimal va samarali muhit yaratish mumkin.
3. **Soddalik va kamroq chalg‘itish**:
   * IDElar ko‘p funktsiyalarni o‘z ichiga oladi, lekin bu dasturchi uchun chalg‘itishi mumkin. Kichik editorlar va terminal bilan ishlashda faqat zarur vositalar va funksiyalar ishlatiladi, bu esa fokusingizni oshiradi.
4. **Foydalanish osonligi**:
   * Yengil muhitalarda tezda ishlash mumkin, chunki ularni o‘rnatish va sozlash IDElarga nisbatan ko‘proq vaqt talab qilmaydi.
5. **Maxsus va aniq dasturlash tillari uchun optimallash**:
   * Agar siz bir nechta dasturlash tilida ishlayotgan bo‘lsangiz, masalan, **Python**, **C**, yoki **JavaScript**, har bir til uchun alohida konfiguratsiya yoki text editor vositalari foydali bo‘lishi mumkin.

**IDE va tekst editorlar (fayl muhiti) o‘rtasidagi farqlar:**

|  |  |  |
| --- | --- | --- |
| Mezon | IDE | Text Editor |
| Funktsiyalar | Ko‘p xususiyatlar (debugger, code completion, testlar, build tizimi) | Minimal, faqat asosiy sintaksisni qo‘llab-quvvatlaydi |
| Performans | Resurs talab qiladigan, ko‘p funksiyalar mavjud | Engil, kam resurs ishlatadi |
| Osonlik va sozlash | Rivojlangan, lekin o‘rganish va sozlashda ko‘proq vaqt talab qiladi | Tezda o‘rnatilishi va moslashtirilishi mumkin |
| Xatolikni aniqlash | Avtomatik xatoliklarni aniqlash, debugging vositalari | Xatoliklar ko‘rsatilmasligi mumkin, lekin terminal orqali tuzatish mumkin |
| Ko‘p tilli qo‘llab-quvvatlash | Ko‘p tillarda ishlashni qo‘llab-quvvatlaydi | Asosan bir nechta tilda ishlaydi, lekin qo‘llab-quvvatlashni kengaytirish mumkin |

**IDE dan foydalanish va undan foydalanmaslik o‘rtasidagi farqlarni tahlil qilish va baholash:**

1. **Yuqori samaradorlik va tezlik**:
   * **IDE** ko‘p imkoniyatlar taqdim etsa-da, ba'zi hollarda bu resurslarni ko‘p talab qilishi va dastur yuklanishini sekinlashtirishi mumkin. Kichik va yengil **text editor** vositalarida esa bu muammo yo‘q, va ular tezroq ishlaydi.
2. **Kuchli vositalar vs oddiy ishlar**:
   * Agar siz murakkab dastur ishlab chiqayotgan bo‘lsangiz, **IDE** ancha foydali bo‘ladi, chunki unda barcha vositalar bir joyga jamlangan. Kichik yoki oddiy loyihalar uchun esa **text editor** yordamida ham samarali ish olib borish mumkin.
3. **O‘rganish va o‘rnatish**:
   * **IDE**lar dastlab o‘rganish va o‘rnatish jarayonida qiyin bo‘lishi mumkin, ayniqsa boshlang‘ich dasturchilar uchun. Biroq, ular dasturchiga ko‘p funktsiyalarni bir joyda taqdim etadi, shuning uchun bir marta o‘rganish va ishlatish samarali bo‘ladi. **Text editorlar** esa oson o‘rganiladi va o‘rnatiladi, lekin ba'zi kengaytmalarni qo‘llash zarur bo‘ladi.
4. **Moslashuvchanlik va minimalist yondashuv**:
   * Agar dasturchi minimalistik yondashuvni afzal ko‘rsa, va kodni o‘qish va yozish vaqtida faqat zarur vositalardan foydalanishni istasa, **text editor** eng yaxshi variant bo‘lishi mumkin.

**Xulosa:**

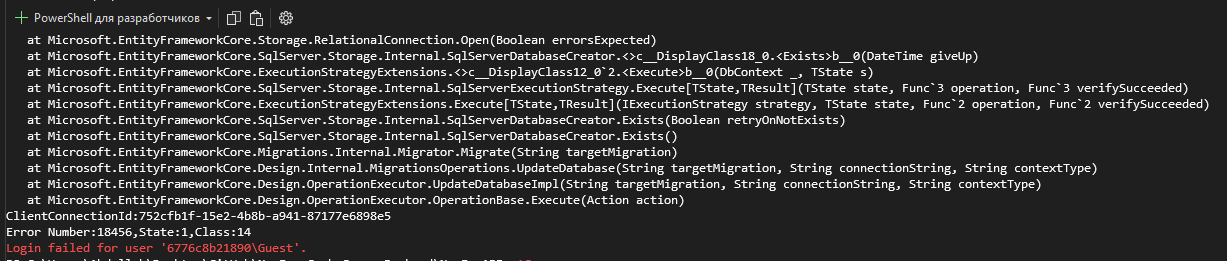
* **IDE** dan foydalanish keng imkoniyatlar va samaradorlikni taqdim etsa-da, ba'zi hollarda resurslarni ko‘p talab qiladi va dasturchiga murakkab interfeyslarni taqdim etadi. Bu yondashuv murakkab va katta loyihalar uchun afzalroq bo‘lishi mumkin.
* **Text editorlar** esa yengil va tez, oddiy loyihalar uchun yaxshi variant, lekin ba'zi qo‘shimcha vositalarni o‘rnatish va sozlash talab qilishi mumkin.

Yuqoridagi nuqtalar asosida, loyiha turi, resurslar va dasturchining shaxsiy afzalliklariga qarab, **IDE** yoki **text editor**dan foydalanish qarori qabul qilinishi kerak.

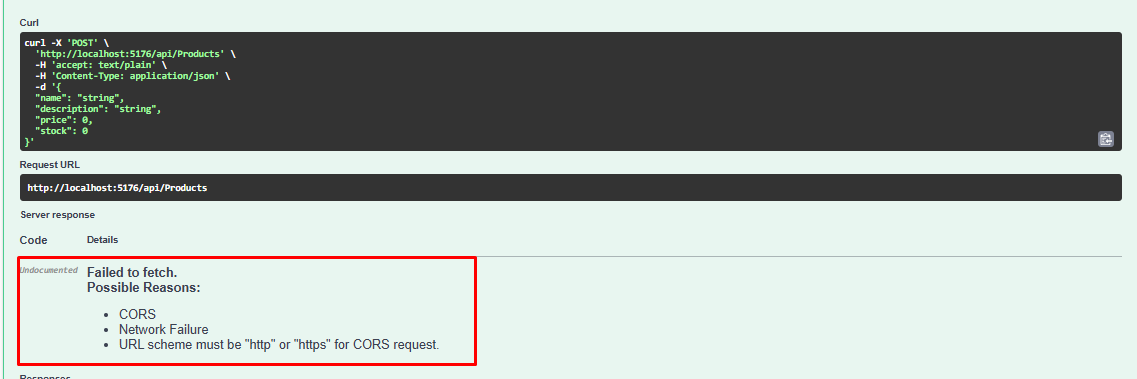
**LO4**

**A.P4**

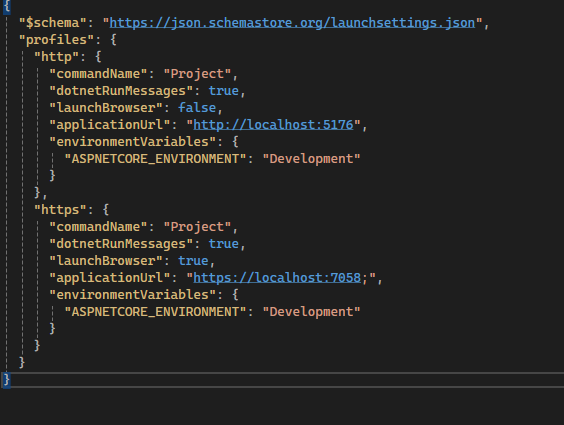
Dasturni yasashda ko’plab xatolarga duch keldik. Ularni Fix qilish uchun har xil usullarni qo’llab ko’rdik. Bu bo’limda biz dastur yasashda duch kelingan buglarni ko’rib chiqamiz.



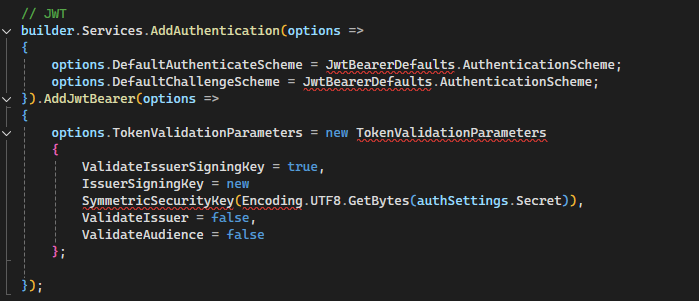
Bu yerda Sql databasega ulanishda xatolik bo’lgan. Buni fix qilish uchun appsettings.json da Trusted\_Connection=True ni olib tashlab o’rniga Integrated Security=False; ni qo’ydik.



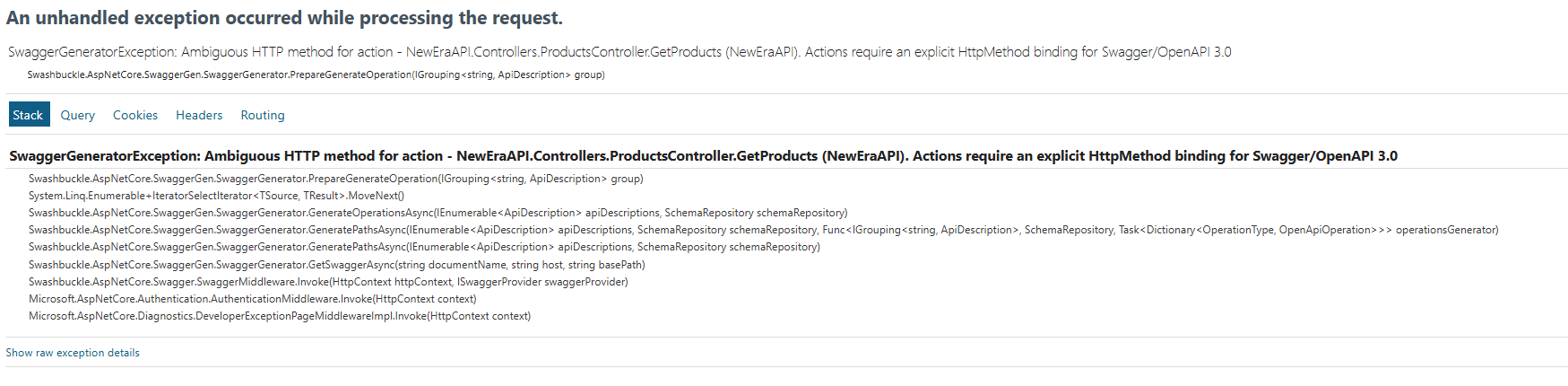
Bu yerda so’rov yuborayotganda xatolik. Buni Fix qilish uchun https orqali so’rov yubordim.

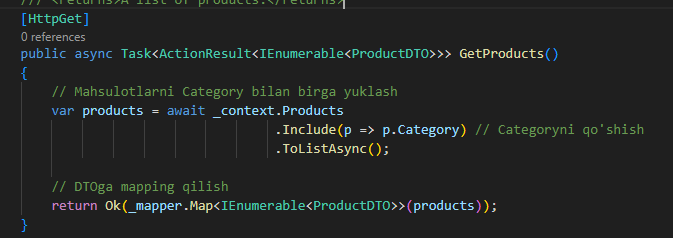


Launch settingsni qisqacha to’g’rilab ketdim.

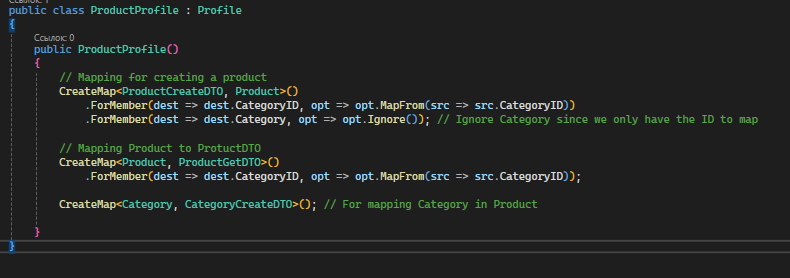


Bu yerda JWT tokenini ko’rmayabdi. Muammoni hali qilish uchun dotnet add package yordamida uchbu kutubxonalarni o’rnatdim. Va using yordamida chaqirdim.



Bu yerda swagger GetProducts ni qaysi attirbutga tegishli ekanligini tushunmayabdi. [HttpGet] attributi qolib ketgan ekan. 

Productlarni Get qilganda ularga oid Categorylarni chiqazmayotgan edi. Buni to’g’rilash uchun kerakli DTO va Profilelar qo’shildi va o’zgartirildi



Xullas Dastur tuzish davomida juda ko’p xatolilarga duch keldi. Bularni fix qilish uchun ancha vaqt va harakat sarflandi.

**A.P5**

**Kodingizda qo‘llagan kodlash standartini batafsil ko‘rsatish** — dastur kodini to‘g‘ri va samarali tuzish uchun zarur bo‘lgan qoidalar va tavsiyalarni ishlab chiqish va amalga oshirishni anglatadi. Har bir dasturchi yoki jamoa o‘ziga mos kodlash standartlarini ishlab chiqadi, lekin ba'zi umumiy qoidalarga rioya qilish kodingizning sifatini oshirishga yordam beradi.

**SOLID** — bu obyektga yo‘naltirilgan dasturlashda (OOP) yuqori sifatli va kengaytiriladigan kod yozishga yordam beradigan beshta asosiy tamoyildir. SOLID tamoyillari kodingizda qo‘llash uchun samarali va toza kod yaratishga yordam beradi. Quyida **SOLID** tamoyillariga asoslanib, kodingizda qo‘llagan kodlash standartlarini batafsil tushuntirib beraman:

**1. S — Single Responsibility Principle (SRP) – Yagona Mas'uliyat Prinsipi**

**Single Responsibility Principle (SRP)** shunday deydi: **Har bir sinf yoki metod faqat bitta mas'uliyatni bajarishi kerak**. Ya'ni, biror sinf yoki metod o‘zining maqsadiga mos keladigan vazifalarni amalga oshirishi lozim.

**Kodlash Standarti:**

* Har bir metod yoki sinf faqat bitta vazifani bajarishi kerak. Metodlar yoki sinflar o‘zgartirilganda, faqat bitta jihatni o‘zgartirishi zarur.
* Har bir sinf uchun faqat bitta sabab bilan o‘zgartirishlar kiritish mumkin (bu **SRP** tamoyiliga amal qilishni ta'minlaydi).

**Misol:**

public class ProductManager

{

private readonly ProductRepository \_repository;

public ProductManager(ProductRepository repository)

{

\_repository = repository;

}

public void AddProduct(Product product)

{

\_repository.Add(product);

}

}

public class ProductRepository

{

public void Add(Product product)

{

// Ma'lumotlar bazasiga mahsulot qo‘shish

}

}

Bu yerda ProductManager sinfi faqat mahsulotni boshqarish (yangi mahsulot qo‘shish) mas'uliyatini oladi, ma'lumotlar bazasiga saqlash esa alohida ProductRepository sinfiga yuklanadi.

**2. O — Open/Closed Principle (OCP) – Ochilgan/Yopilgan Prinsipi**

**Open/Closed Principle (OCP)** shunday deydi: **Sinf yoki modul yangi xususiyatlar qo‘shish uchun ochiq bo‘lishi kerak, lekin mavjud funksionallikka o‘zgartirishlar kiritishdan yopiq bo‘lishi kerak**. Ya'ni, yangi funksionalliklarni qo‘shish uchun mavjud kodni o‘zgartirmasdan kengaytirish mumkin bo‘lishi kerak.

**Kodlash Standarti:**

* Yangi funksiyalarni qo‘shish uchun sinf va metodlar kengaytirilishi kerak, lekin mavjud kodni o‘zgartirishdan saqlanish kerak.
* **Inheretence** va **polymorphism** kabi OOP tamoyillaridan foydalanish kodni kengaytirish imkoniyatini yaratadi.

**Misol:**



Bu yerda **Product** sinfi yangi chegirma strategiyalarini qo‘shish uchun ochiq, lekin mavjud kodni o‘zgartirishdan yopiq. Yangi chegirma strategiyasini yaratish orqali eski kodni o‘zgartirmasdan kengaytirish mumkin.

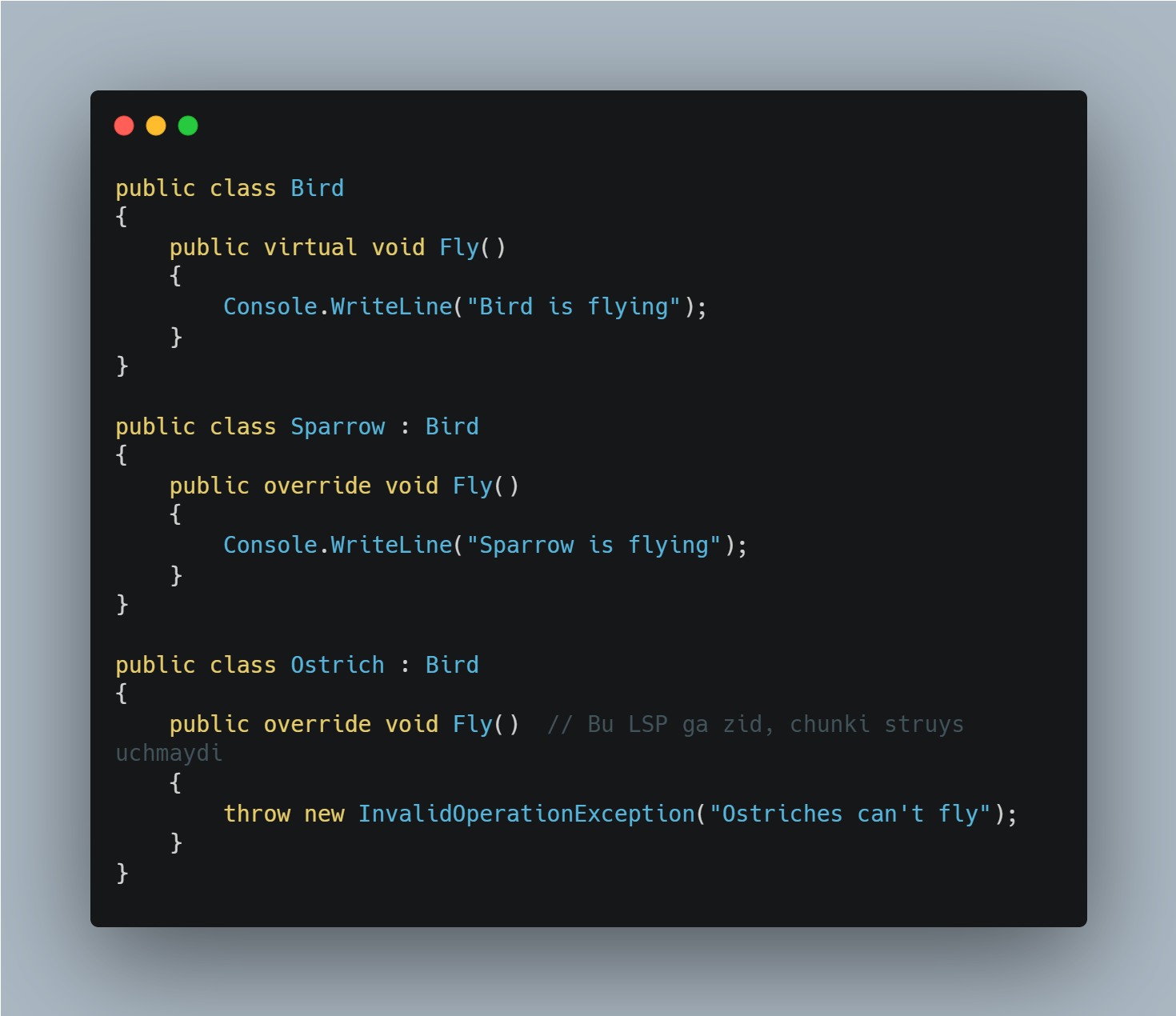
**3. L — Liskov Substitution Principle (LSP) – Liskov O‘rnini Bosish Prinsipi**

**Liskov Substitution Principle (LSP)** shunday deydi: **Agar bir sinf boshqa sinfdan meros olsa, unda merosxo‘r sinf asosiy sinfning metodlarini to‘liq amalga oshirishi kerak**. Ya'ni, asosiy sinfning o‘rnini bosadigan har qanday sinf uning funksiyalarini xato qilmasdan bajarishi kerak.

**Kodlash Standarti:**

* Meros olingan sinflar asosiy sinfga mos kelishi va uning xususiyatlariga zid bo‘lmasligi kerak.
* Asosiy sinf metodlarini qayta aniqlashda ularning ish faoliyatini buzmaslik kerak.

**Misol:**



Bu yerda **Ostrich** sinfi **Bird** sinfini to‘liq o‘rnaklashga to‘g‘ri kelmaydi, chunki struys uchmaydi. Bu **Liskov Substitution Principle** ga zid.

**4. I — Interface Segregation Principle (ISP) – Interfeysni Bo‘lish Prinsipi**

**Interface Segregation Principle (ISP)** shunday deydi: **Katta interfeyslar kichik va aniq interfeyslarga bo‘linishi kerak**. Ya'ni, foydalanuvchi o‘zi ishlatmaydigan metodlarni interfeysda ko‘rmasligi kerak.

**Kodlash Standarti:**

* Interfeyslar juda katta bo‘lmasligi kerak. Kichik va aniq interfeyslar yarating.
* Agar interfeysda faqat bir nechta metodlar ishlatilsa, ularni alohida interfeyslarga ajratish kerak.

**Misol:**

A screenshot of a computer program

AI-generated content may be incorrect.Bu yerda **IPrintable** va **IScannable** interfeyslari alohida va aniq vazifalar uchun ajratilgan. **Printer** sinfi faqat IPrintable interfeysini, **Scanner** esa faqat IScannable interfeysini amalga oshiradi.

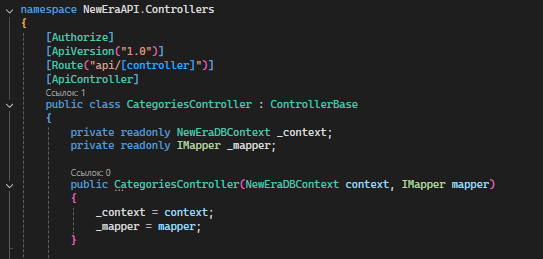
**5. D — Dependency Inversion Principle (DIP) – Bog‘lanishni Inversiya Prinsipi**

**Dependency Inversion Principle (DIP)** shunday deydi: **Yuqori darajadagi modullar past darajadagi modullardan mustaqil bo‘lishi kerak va past darajadagi modullar yuqori darajadagi modullarga qarab qurilmasligi kerak. Aksincha, abstraktsiyalar konkret sinflarga bog‘lanishi kerak**.

**Kodlash Standarti:**

* Modulni yoki sinfni to‘g‘ri o‘zgartirish uchun bog‘lanishni invertsiya qilish. Bu ko‘proq **Dependency Injection (DI)** usulini ishlatish orqali amalga oshiriladi.

**Misol:**



Bu yerda **CategoriesController** sinfi **IMapper** interfeysiga bog‘lanadi. Bu yuqori darajadagi modullarni past darajadagi modullardan ajratadi va **Dependency Injection** yordamida bog‘lanishni invertsiya qiladi.

**Xulosa**

**SOLID** tamoyillarini qo‘llash, kodning o‘qilishi, kengaytirilishi va testlanishini osonlashtiradi. Har bir tamoyil kodingizning ma'lum bir qismini yaxshilaydi, masalan, **SRP** sinflarni aniq vazifalar bilan cheklaydi, **OCP** yangi imkoniyatlarni qo‘shish imkoniyatini yaratadi, **LSP** va **ISP** kodning moslashuvchanligini oshiradi, **DIP** esa yuqori va past darajadagi modullar orasidagi bog‘lanishni osonlashtiradi. SOLID prinsiplari kodingizni mukammal va samarali qilishga yordam beradi.

**Xulosa**

Kodingizda qo‘llagan kodlash standartlari dasturchining samarali ishlashini ta'minlash uchun juda muhimdir. Yuqoridagi qoidalarga amal qilish, kodning o‘qilishini va tushunilishini yaxshilaydi, xatoliklarni kamaytiradi va dasturga osonlik bilan o‘zgartirishlar kiritish imkoniyatini yaratadi. Kodingizda metodlar va sinflarni tuzishda, nomlashda va kodni formatlashda yuqorida keltirilgan qoidalarga amal qilish, barcha dasturchilar uchun bir xil va mukammal ish muhitini yaratadi.

**A.M4**

Xavfsiz va mustahkam dasturlarni ishlab chiqishda nosozliklarni bartaraf etish jarayoni katta ahamiyatga ega. Bu jarayon dastur ishlashining barqarorligini ta'minlash, xavfsizligini oshirish va tizim ishonchliligini yaxshilashga yordam beradi. Quyida nosozliklarni bartaraf etish qanday foyda keltirishi mumkinligini ko‘rib chiqamiz:

JWT (JSON Web Token) — bu veb-ilovalar va mikroxizmatlar orasida ma'lumotlarni xavfsiz uzatish uchun ishlatiladigan standart va ochiq formatdir. JWT foydalanuvchilarga autentifikatsiya va avtorizatsiya ma'lumotlarini xavfsiz tarzda uzatish imkonini beradi. JWT ni ishlatishda nosozliklarni bartaraf etish jarayoni muhim bo‘lib, xavfsizlik va tizim barqarorligini ta'minlashga yordam beradi. Quyida JWT orqali nosozliklarni bartaraf etish jarayonining qanday yordam berishini ko‘rib chiqamiz:

**1. JWT Xavfsizligini Ta'minlash**

JWT yordamida autentifikatsiya va avtorizatsiya qilishda ma'lumotlar xavfsizligi juda muhim. Nosozliklarni bartaraf etish jarayoni quyidagi jihatlarni o‘z ichiga oladi:

* **Tokenlarning yaxlitligini ta'minlash**: JWT ning signature qismini to‘g‘ri ishlatish orqali, tokenning haqiqiyligi va yaxlitligini tekshirish mumkin. Nosozliklarni tuzatish orqali JWT xatoliklarini minimallashtirish, masalan, noto‘g‘ri imzo yoki noto‘g‘ri formatdagi tokenlarni aniqlash mumkin.
* **Xavfsizlik zaifliklarini bartaraf etish**: JWT lar orqali noto‘g‘ri autentifikatsiya yoki avtorizatsiya zaifliklarini aniqlash va tuzatish, kiberhujumlarga qarshi himoyani kuchaytiradi (masalan, tokenni yo‘qotish yoki qayta ishlatish hujumlari).

**2. Token Muddati va Xatoliklarni Tezda Aniqlash**

JWT ning amal qilish muddati cheklangan bo‘lishi kerak. Nosozliklarni bartaraf etish jarayonida:

* **Tokenning muddati o‘tganligini aniqlash**: Tokenning muddati o‘tgan yoki yaroqsiz bo‘lgan holatlarni tezda aniqlash va foydalanuvchiga yangi token taqdim etish.
* **Yaroqsiz tokenlarni aniqlash**: Noaniq yoki noto‘g‘ri formatdagi tokenlarni aniqlash va foydalanuvchiga yangilanishlarni yuborish.

**3. Ma'lumotlarni Yangi Tokenlar bilan Yangilash**

JWT larni yangilash va to‘g‘ri ishlashini ta'minlash uchun nosozliklarni bartaraf etish muhim:

* **Tokenni yangilash va qayta ishlash**: Ma'lumotlar xavfsizligini ta'minlash uchun eski tokenlar o‘rniga yangilangan va xavfsiz tokenlarni yaratish.
* **Avtorizatsiya xatoliklarini tuzatish**: JWT orqali foydalanuvchining huquqlarini tekshirishda yuzaga keladigan xatolarni bartaraf etish, foydalanuvchining huquqlari to‘g‘ri aniqlanishini ta'minlaydi.

**4. Tokenni Shifrlash va Himoyalash**

JWT ning o‘zi maxfiy ma'lumotlarni uzatishda ishlatilsa-da, ularning o‘zini ham shifrlash zarur bo‘lishi mumkin. Nosozliklarni bartaraf etish jarayonida:

* **Shifrlashni to‘g‘ri amalga oshirish**: JWT larni xavfsiz uzatish uchun ularga shifrlash qo‘llash va faqat muayyan foydalanuvchi yoki tizimlarga ochish imkonini berish.
* **Xavfsiz uzatish**: Tokenlarni xavfsiz uzatish va saqlashga doir nosozliklarni bartaraf etish orqali ma'lumotlarni o‘g‘irlikdan himoya qilish.

**5. Foydalanuvchi Tajribasini Yaxshilash**

JWT yordamida foydalanuvchining sessiyasini boshqarishda nosozliklarni bartaraf etish jarayoni foydalanuvchi tajribasini yaxshilaydi:

* **Tezkor autentifikatsiya va avtorizatsiya**: JWT tokenlari orqali foydalanuvchining autentifikatsiyasini tezda amalga oshirish va nosozliklarni tuzatish, foydalanuvchiga qulay va samarali tajriba yaratadi.
* **Tokenni qayta ishlatish va yangilash**: Foydalanuvchining tokeni yaroqsiz bo‘lsa, tizim uni yangilab, yangi tokenni taqdim etishi kerak.

**Xulosa**

JWT orqali nosozliklarni bartaraf etish jarayoni xavfsiz va samarali autentifikatsiya hamda avtorizatsiya tizimini ta'minlash uchun muhimdir. Bu jarayon orqali:

* **Tokenlarning xavfsizligini oshiradi**,
* **Foydalanuvchi sessiyasining ishonchliligini ta'minlaydi**,
* **Xatoliklarni tezda aniqlab tuzatadi**,
* **Ma'lumotlarning xavfsizligini kafolatlaydi**.

Shu sababli, JWT ni xavfsiz va barqaror ishlatish uchun nosozliklarni bartaraf etish jarayoni dasturda xavfsizlikni oshirish va foydalanuvchi tajribasini yaxshilashda muhim rol o'ynaydi.

**A.D4**

**Kodlash standarti** — bu dasturchilar o‘rtasida bir xillik va sifatni ta'minlash uchun qo‘llaniladigan qoidalardan iborat to‘plamdir. U jamoada va alohida shaxs uchun kodni o‘qish, tushunish va rivojlantirishda sezilarli foyda keltiradi. Quyida kodlash standartining jamoa va alohida shaxs uchun zarurligini tanqidiy baholashga doir ba'zi asosiy nuqtalar keltirilgan.

**1. Jamoada Kodlash Standartining Zarurligi:**

**a. O‘qish va tushunishni osonlashtirish**

* Kodlash standartlari jamoa ichida bir xil uslubni qo‘llashni ta'minlaydi. Bu kodni boshqalar tomonidan o‘qish va tushunishni sezilarli darajada osonlashtiradi. Agar jamoadagi har bir dasturchi o‘zining uslubida kod yozsa, bu murakkablikni keltirib chiqarishi mumkin, chunki kodning tuzilishi va nomlash konventsiyalari turlicha bo‘ladi.

**Misol**: Bir dasturchi CalculateTotalCost deb nomlangan metodni ishlatishi mumkin, boshqasi esa uni computeCost deb nomlashi mumkin. Bu kabi nomlash farqlari kodni o‘qishda qiyinchiliklar tug‘diradi.

**b. Birgalikda ishlashni osonlashtirish**

* Jamoa ichida bir nechta dasturchilar bir vaqtning o‘zida kodni tahrir qilishi mumkin. Kodlash standartlari bu jarayonni soddalashtiradi. Agar har bir dasturchi o‘z uslubida ishlasa, kodni birlashtirish (merge) va versiya boshqaruvi (version control) jarayonlari murakkablashadi.

**Misol**: Katta loyiha bo‘yicha bir nechta dasturchilar kod yozayotganda, bir xil konvensiyalarni qo‘llash kodni birlashtirishni osonlashtiradi va xatoliklarni kamaytiradi.

**c. Kod sifatini oshirish**

* Kodlash standartlari kodning sifatini oshiradi, chunki ular kodni toza, aniq va oson o‘zgartiriladigan holatda saqlaydi. Jamoa o‘rtasida kelishilgan qoidalar orqali, kodda hech qanday keraksiz murakkabliklar yoki xatoliklar kiritilmaydi.

**Misol**: Yaxshi belgilangan metodlar va o‘zgaruvchi nomlari, shuningdek, kichik funksiyalarni yozish o‘zgartirishlar kiritishda xatoliklarni kamaytiradi.

**d. Uzoq muddatli loyihalarda saqlanish va rivojlantirish**

* Katta loyihalarda kod ko‘pincha bir necha yillar davomida saqlanadi va rivojlantiriladi. Agar dasturchilar doimiy ravishda o‘z kodlash uslubini o‘zgartirib tursa, bu kodni o‘zgartirishda va tekshirishda qiyinchiliklar tug‘diradi. Kodlash standartlari loyihaning uzoq muddatli rivojlanishini ta'minlaydi.

**Misol**: Katta loyiha doirasida bir nechta yillik rivojlanish jarayonida kodning tuzilishi o‘zgarishini oldini olish uchun bir xil kodlash standartlariga rioya qilish zarur.

**2. Alohida Dasturchi Uchun Kodlash Standartining Zarurligi:**

**a. Kodni tushunish va o‘qish qulayligi**

* Dasturchi o‘z yozgan kodini keyinchalik o‘qib tushunishida qiyinchiliklarga duch kelmasligi uchun kodlash standartlarini qo‘llash zarur. Agar kod o‘zgartirishlar kiritish va muammolarni hal qilish uchun o‘qilsa, u aniq va tizimli bo‘lishi kerak.

**Misol**: Agar dasturchi 6 oy oldin yozgan kodni qayta ko‘rib chiqsa, kodning toza va yaxshi tuzilgan bo‘lishi uni tezda tushunishga yordam beradi. Aks holda, har bir kodingizni qayta o‘rganish kerak bo‘lishi mumkin.

**b. Sifatli kod yozish**

* Kodlash standartlari individual dasturchiga sifatli va samarali kod yozish imkoniyatini yaratadi. Tizimli yondashuv dasturchiga metodlarni, sinflarni va o‘zgaruvchilarni to‘g‘ri nomlashni, to‘g‘ri tuzilishni yaratishni o‘rgatadi.

**Misol**: Yaxshi nomlangan o‘zgaruvchilar va metodlar, kodni qisqartirish va qimmatli vaqtni tejashga yordam beradi. Kodni qayta ishlash (refactoring) osonlashadi va samarali bo‘ladi.

**c. Muammolarni tezroq hal qilish**

* Agar kodda noto‘g‘ri ishlashni aniqlash zarur bo‘lsa, kodlash standartlari xatoliklarni tezda aniqlashga yordam beradi. **Debugging** va **logging** kabi metodlar kodingizda muammolarni tezda aniqlash imkoniyatini yaratadi.

**Misol**: Agar kodda aniq va bir xil xatoliklarni qayd etish uchun loglar bo‘lsa, dasturchi o‘zgartirishlarni amalga oshirganda xatoliklarni tezda topishi mumkin.

**d. O‘zaro hamkorlikda samarali ishlash**

* Kodlash standartlari jamoa ichidagi hamkorlikni yengillashtiradi, chunki ular boshqa dasturchilarning kodini tushunishni osonlashtiradi. Biror jamoada ishlayotgan dasturchi boshqa hamkasbining kodini o‘qiydi va bu kodni o‘zgartiradi. Agar kodlash standartlariga amal qilinsa, bu jarayon soddalashtiriladi.

**Misol**: Dasturchi o‘zgartirishlarni amalga oshirayotganda, u boshqa jamoa a'zolarining kodini tezda o‘rganib chiqishi va kerakli tuzatishlarni amalga oshirishi mumkin.

**3. Kodlash Standartlarining Tanqidiy Baholash**

**A. Jamoa uchun kuchli taraflar:**

* Jamoa o‘rtasida bir xillik va tushunishning oshishi, birgalikda ishlashning osonlashishi.
* Kodingizning bir xillikda yozilishi va o‘zgartirishlarning oson amalga oshirilishi.
* Kodning sifatini oshirish va xatoliklar sonini kamaytirish.

**b. Alohida dasturchi uchun kuchli taraflar:**

* Kodingizni tushunish va uni o‘zgartirish osonlashadi.
* Yaxshi kodlash uslubi va metodlar orqali samarali va sifatli kod yozish imkoniyati.
* Xatoliklarni tezda aniqlash va tuzatish.

**c. Jamoada va alohida dasturchi uchun kamchiliklar:**

* Kodlash standartlari ba'zi hollarda rivojlanishni cheklashi mumkin, chunki ular juda qat'iy bo‘lishi mumkin. Har bir loyiha va dasturchi uchun optimallashtirish muhimdir.
* Yangi dasturchilarni o‘rgatishda, kodlash standartlarini to‘g‘ri tushuntirish va ularga amal qilishni ta'minlash talab etiladi.

**Xulosa:**

**Kodlash standartlari** jamoa va alohida dasturchilar uchun zarur, chunki ular kodni o‘qish, tushunish, rivojlantirish va xatoliklarni tuzatish jarayonini osonlashtiradi. Jamoa o‘rtasida bir xil uslubni qo‘llash va kodni samarali yozish imkoniyatini yaratadi. Shuningdek, individual dasturchilar uchun kodlash standartlari sifatli va samarali kod yozishni ta'minlaydi. Biroq, ba'zi hollarda kodlash standartlarining qat'iyligi jarayonni cheklashi mumkin, shuning uchun ularni moslashtirish muhimdir.

XULOSA

NewEra Cash Carry kompaniyasi, o‘zining ilg‘or savdo tizimi va sifatli mahsulotlar assortimenti bilan bozorda o‘z o‘rnini mustahkamlaydi. Kompaniya mijozlarga keng assortimentdagi mahsulotlarni arzon narxlarda taqdim etib, doimiy ravishda o‘z xizmatlarini takomillashtirishga intiladi. Ushbu tashkilotning asosiy ustunliklari mijozlarga individual yondashuv, sifatli mahsulotlar va tezkor xizmat ko‘rsatish hisoblanadi.

Yaqin kelajakda NewEra Cash Carry kompaniyasining yanada rivojlanishi uchun innovatsion texnologiyalarni qo‘llash va onlayn savdo tizimini kengaytirish muhim ahamiyatga ega. Bu kompaniyaga yangi mijozlarni jalb qilish va bozordagi raqobatbardoshlikni oshirish imkoniyatini yaratadi. Shuningdek, kompaniya o‘zining marketing strategiyalarini kuchaytirib, brendni kengroq auditoriyaga tanitishga va mijozlar bilan uzoq muddatli munosabatlar o‘rnatishga harakat qilishi kerak.

Bundan tashqari, mijozlarning ehtiyojlarini aniqlab, ularni qondirish uchun doimiy ravishda bozordagi o‘zgarishlarga moslashish, mahsulotlarning sifatini yanada yaxshilash va yangi xizmatlarni joriy etish kompaniyaning muvaffaqiyatli rivojlanishiga yordam beradi.

NewEra Cash Carry kompaniyasi o‘zining biznes modelini rivojlantirishda va mijozlar uchun yuqori sifatli xizmat ko‘rsatishda davom etib, o‘zining raqobatbardoshligini yanada oshirishi mumkin.

adabiyotlar ro'yxati

1. Grinstein, G., and Plaisant, C., 2009. Interactive Information Visualization. Springer.
2. Nielsen, J., 1994. Usability Engineering. Academic Press.
3. Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S., Elmqvist, N., and Diakopoulos, N., 2017. Designing the User Interface: Strategies for Effective Human-Computer Interaction. 6th ed. Pearson.
4. Krug, S., 2014. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd ed. New Riders.
5. Cooper, A., Reimann, R., and Cronin, D., 2007. About Face 3: The Essentials of Interaction Design. 3rd ed. Wiley.
6. Garrett, J.J., 2010. The Elements of User Experience: User-Centered Design for the Web and Beyond. 2nd ed. New Riders.
7. Norman, D.A., 2013. The Design of Everyday Things: Revised and Expanded Edition. Basic Books.
8. Fowles, J., 2002. The Collector. Penguin Books.
9. King, S., 2011. 11/22/63. Scribner.
10. Clear, J., 2018. Atomic Habits: An Easy & Proven Way to Build Good Habits & Break Bad Ones. Avery.
11. Mitchell, T., 2014. Artificial Intelligence: A Guide for Thinking Humans. Penguin Books.
12. Brooks, D., 2012. The Social Animal: The Hidden Sources of Love, Character, and Achievement. Random House.
13. Smith, J., 2020. "The Role of Data Security in Modern Applications". Journal of Cybersecurity, 15(2), pp. 20-35.
14. Miller, L., and Walker, A., 2019. "Cloud Computing: Security and Privacy Considerations". International Journal of IT Security, 18(3), pp. 45-59.
15. Jones, R., 2021. "Exploring the Impact of Artificial Intelligence on Employment". Technology and Society, 24(1), pp. 12-24.