МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Розробка програмного забезпечення»

Група: 4РП-05

Дипломний проект

здобувача освіти денної форми навчання
РП.05.26.000.ДП

ШАВРІДІНА
АНАСТАСІЯ
ОЛЕКСАНДРІВНА

м. Одеса
2022 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНАХТ»

Спеціальність: **121 «Інженерія програмного забезпечення»**

Освітня програма: «**Розробка програмного забезпечення»**

Група: **4РП-05**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи) на тему:

Дослідження переваг застосування мікросервісної архітектури в порівнянні з монолітним додатком в розробках веб-орієнтованих систем побудованих на платформі ASP.NET Web API Core

Проектний матеріал складається з пояснювальної записки на 55 сторінках та графічного (презентаційного) матеріалу на 10 аркушах (слайдах).

Дипломник _____ (Шаврідіна А.О.)

Керівник _____ (Медведєв А.О.)

Консультанти:

з економічної частини _____ (Копайгородська Т.Г.)

з охорони праці_____ (Чорновол Н.І.)

з дотримання вимог ЄСКД _____ (Петрашова В.І.)

старший консультант _____ (Скорнякова О.В.)

До захисту допущений

Голова циклової комісії _____ (Скорнякова О.В.)

Завідувач відділення _____ (Суліма Ю.Ю.)

Захист «____»_____ 2022 р. Протокол ДКК №_____

Оцінка ДКК _____

Секретар ДКК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНАХТ»

Відділення **комп'ютерних систем** Комісія **КТ та ПІ**

Спеціальність **121 «Інженерія програмного забезпечення»**

Освітня програма **«Розробка програмного забезпечення»**

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР_____

_____

"_____"_____2022 р.

З А В Д А Н Н Я

на дипломний проект (роботу)

Здобувачеві (здобувачці) освіти Шавріділна Анастасія Олександрівна

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Дослідження переваг застосування мікросервісної архітектури в порівнянні з монолітним додатком в розробках веб-орієнтованих систем побудованих на платформі ASP.NET Web API Core

затверджена наказом по коледжу від "30" грудня 2021 р. № 306-А2-ОД

2. Термін здачі закінченого проекту (роботи) _____

3. Вихідні данні до проекту (роботи) Microsoft Visual Studio, Архітектура, .Net, C#, MS SQL LINQ, Postman, API, HTTP, Microsoft Edge, MySQL, СУБД, Docker

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

1. Огляд мікросервісної та монолітної архітектур. 2. Використання мікросервісної архітектури у веб-орієнтованої системі. 3. Економічний розрахунок. 4. Охорона праці.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Презентація (10 слайдів)

6. Консультанти по проекту (роботі), із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний	Медведєв А.О.		
Економічний	Копайгородська Т.Г.		
Охорона праці	Чорновол Н.І.		
Нормоконтроль	Петрашова В.І.		
Старший консультант	Скорнякова О.В.		

7. Дата видачі завдання _____

Керівник _____

(підпис)

Завдання прийняв до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту (роботи)	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	Визначення задач та цілей дипломного проекту. Обговорення тематики та розділів дипломного проекту.	16.05.2022	виконала
2	Робота над вступною частиною дипломного проекту.	17.05.2022	виконала
3	Робота над першим розділом липломного проекту. Розділ 1. Застосування мікросервісної архітектури у розробці веб-орієнтованої системи.	20.05.2022– 31.05.2022	виконала
4	Розділ 2. Економічний розрахунок.	01.06.2022 - 06.06.2022	виконала
5	Розділ 3. Охорона праці.	07.06.2022 - 12.06.2022	виконала
6	Розробка презентації до дипломної роботи.	12.06.2022	виконала
7	Робота над графічною частиною дипломного проекту. Оформлення пояснювальної записки та переліку використаних джерел.	13.06.2022-16.06.2022	виконала
8	Підготовка до захисту дипломного проекту. Підготовка доповіді.	17.06.2022– 20.06.2022	виконала

Дипломник _____

(підпис)

Керівник _____

(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 05.26.000 ДП ПЗ	*Дипломний проект*		
А4			*РП 05.26.000 ДП ПЗ*	*Пояснювальна*	*1*	
				записка		
				Презентація		

Ізм.	Арк.	№ докум.	Підпис	Дата	*РП 05..000*			
Розробила		Шаврідіна А.О.			Дослідження переваг застосування мікросервісної архітектури в порівнянні з монолітним додатком в розробках веб-орієнтованих систем побудованих на платформі ASP.NET Web API Core	Літ.	Аркуш	Аркушів
Перевірив		Медведєв А.О.				*Н Д П*	*3*	
						ВСП «ОТФК ОНТУ» гр.РП-05		
Н. Контр.		Петрашова В.І						
Затверд.		Скорнякова О.В.						

ЗМІСТ

ВСТУП

Розробка програмного забезпечення з кожним роком вимагає впровадження нових практик та підходів. Одним з таких нововведень є підхід гнучкої розробки програмного забезпечення з використанням мікросервісів, що представляє термін “Мікросервісна архітектура”. Від вибору архітектури залежать організація елементів програми та їх взаємодія, а також визначення вимог до програмного продукту. Обрана архітектура впливає на такі характеристики програмного забезпечення як : масштабованість, надійність, зручність тестування та супроводу.

Мікросервісна архітектура почала набувати розповсюдження з 2010 року. Вона вбачає в собі альтернативне рішення Монолітної архітектури.

Монолітна архітектура передбачає створення програм, компоненти яких є тісно зв’язанними. Проект розгортається на сервері й виконує всі операції як єдиний сервіс. Будь-які зміни, які мають бути внесені у програмне забезпечення, що впроваджене в користування, передбачають повну перебудову проекту.

Мікросервісна архітектура передбачає розробку програм, які розділені на мікросервіси. Кожен мікросервіс містить елемент програми, який характеризується вузькоспрямованою функціональною можливістю: кожен сервіс відповідальний за конкретний процес, згідно поставлених перед ним задач. Кожен сервіс розгортається незалежно від інших й містить все необхідне для самостійного існування. Управління сервісами відбувається через эдиний центр керування. Взаємодія між сервісами відбувається через прикладний програмний інтерфейс(API). Сервіси є слабо зв’язаними, тобто кожен сервіс є самостійним і внесення змін в один сервіс не передбачає зміни інших сервісів.

МЕТА ТА ЗАВДАННЯ

Метою дипломної роботи є порівняння мікросервісної архітектури з монолітною архітектурою при розробці веб-орієнтованих систем.

Для досягнення цієї мети необхідно вирішити та розробити ряд дій:

1. Провести дослідження щодо використання монолітної архітектури та мікросервісної архітектури. Визначити переваги та недоліки кожної архітектури.

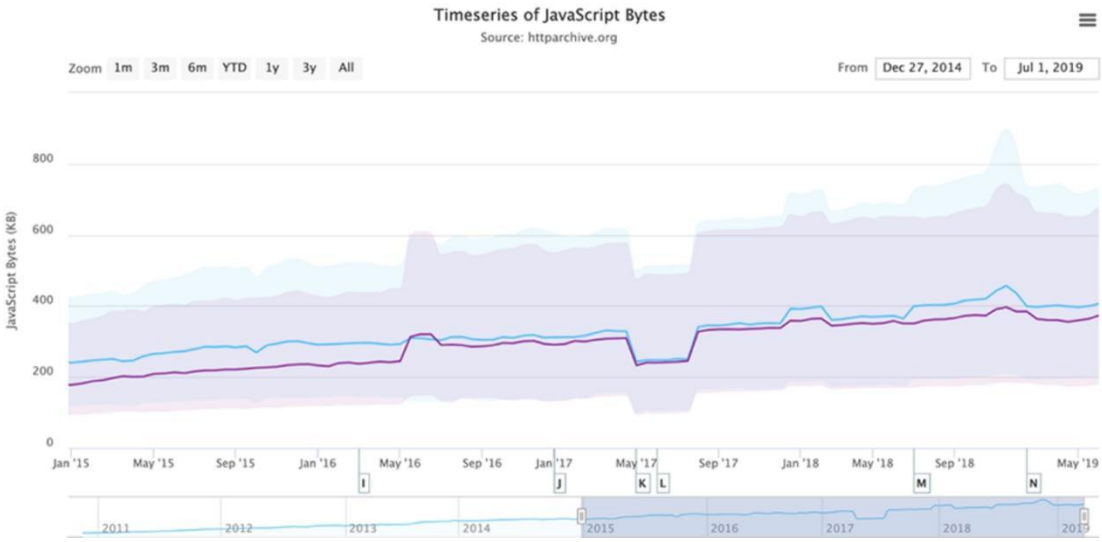
2. Зробити порівняльний аналіз монолітної та мікросервісної архітектури.

3. Розробити проект з використанням мікросервісної архітектури, що дозволить отримати необхідні навички роботи з розподілом програми на окремі мікросервіси, планування внутрішньої архітектури кожного мікросервісу та налаштування взаємодії всіх мікросерісів.

1 ЗАСТОСУВАННЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ У РОЗРОБЦІ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ

1.1 Дослідження актуальності проблеми

Створення веб-сайтів з використанням мікросервісної архітектури є новою тендецією у сфері розробки програмного забезпечення. Досить дового монолітна архітектура задовольняла усі потреби на стадії проектування, реалізації та розгортання програмного забезпечення. Якщо раніше більшість сайтів мали певний чітко визначений функціонал, що міг задовольняти потреби протягом великого відрізку часу, то зараз сайти потребують постійного оновлення та додавання нового функціоналу, щоб залишатися конкурентноспроможними на ринку.



1.1 Ріст об`єму програмного коду java script у веб-застосунках

Додавання, видалення та оновлення функціоналу у впроваджене в експлуатацію програмне забезпечення супроводжується багатьма проблемами та ризиками. В монолітній архітектурі модулі програми є дуже зв`язаними, тому, якщо внести якусь зміну в один модуль програми, це може спровокувати непередбачувану поведінку там, де був визваний програмний код з цього модуля. Прикладом може стати, наприклад, видалення функціоналу корзини з

сайту. Якщо видалити корзину з сайту, то помилки з`являться в усіх модулях програми, де була взаємодія з корзиною. Також виникнуть проблеми з базою данних, адже об`єктна модель корзини зв`язана з об`єктними моделями покупця, товару, замовлень і т.д.

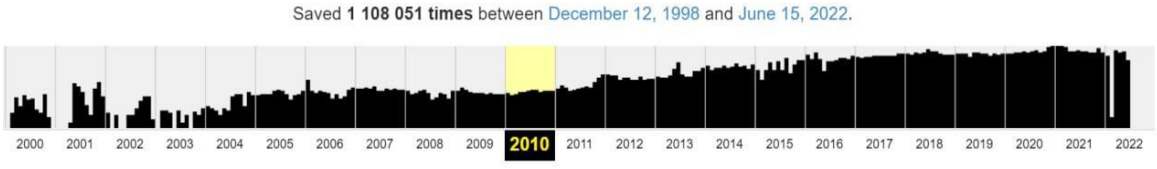
Мікросервісна архітекура дозволяє відокремити та зробити самостійним існування окремих модулів програми, шляхом винесення всього функціоналу в окремі контейнери і реалізації взаємодії між цими контейнерами через API.

Отже мікросервісна архітектура є дуже актуальною тенденцією в розробці програмного забезпечення сьогодення, адже покликана вирішити проблему великої зв`язанності програмного коду.

1.2 Аналіз існуючих аналогів

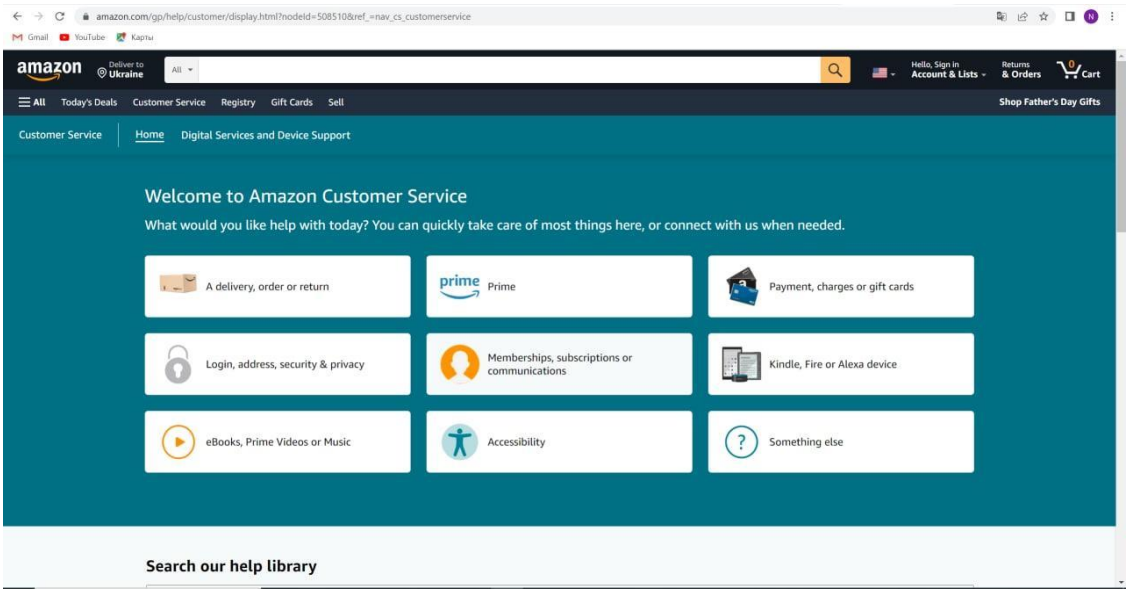
Огляд існуючих веб-сайтів побудованих на мікросервісній архітектурі дозволить сформувати вимоги до побудови власного проекту.

Прикладом відомого великого веб-сервісу побудованого на мікросервісній архітктурі є Amazon. Amazon орієнтований на продаж товарів та послуг в Інтернеті. Функціонал цього сайту налічує велику кількість можливостей для покупців та продавців і постійно піддається змінам та оновлення. Тому Amazon є гарним прикладом, коли мікросервісна архітектура застосовується для веб-застосунків, які динамічно змінюються при цьому зберігаючи свою надійність.



1.2 Шкла змін, що були внесені за весь час існування веб-ресурсу Amazon

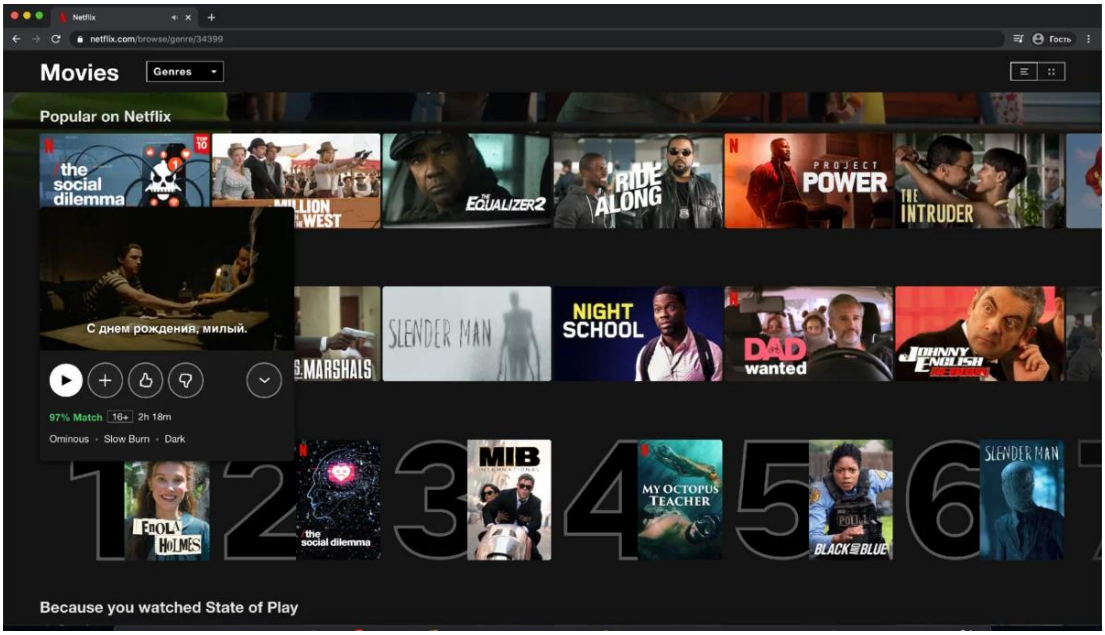
Amazon має інтуїтивно зрозумілий користувацький інтерфейс. Всі позиції сайту розділені на категорії. Продумана фільтрація та сортування товарів забезпечують покупця дуже легким, чітким та зрозумілим пошуком необхідних товарів. Зважаючи на масштаб сайту, можуть виникнути лише деякі проблеми з пошуком необхідної категорії, так як їх налічується дуже багато.



1.3 Інтерфейс сайту Amazon

Іншим відомим прикладом веб-ресурсу, побудованого на мікросервісній архітектурі є Netflix. Компанія Netflix в 2013 році перейшла від монолітної архітектури до мікросервісної архітектури й продовжує розвиватися в цьому напрямку, постійно створюючи нові сервіси. Backend включає сервіси бази даних та сховища, що повністю працюють у хмарі AWS. Backend переважно обробляє все, крім потокового відео.

Інтерфейс Netflix є дуже простим та зрозумілим. Користувач сайту може дуже легко знайт потрібний фільм за допомогою строки пошуку або за категоріями.



1.4 Головна сторінка сайту Netflix

Таблиця 1.1. — Аналіз відомих аналогів та відповідність їх критеріям

Назви веб-ресурсів	Критерії			
	Структура	Дизайн	Масштабність	Швидкість
Amazon	4	5	Великий	5
Netflix	5	5	Середній	5

1.3 Огляд існуючих архітектур

1.3.1 Монолітна архітектура

Монолітна архітектура – це такий спосіб розробки програмного забезпечення, коли всі компоненти програми представляють єдине ціле, тобто є дуже зв’язаними. Програмне забезпечення розгортається на єдиному сервері і працює з єдиною базою даних. Така архітектура добре підходить для маломасштабних проектів, в яких не передбачається додавання великої кількості нового функціоналу. Такий проект дуже легко реалізувати та протестувати в порівнянні з програмним забезпеченням побудованим на основі мікросервісної архітектури.

Моноліти можуть добре масштабуватися та підтримуватися до певного моменту, після чого складність починає зростати дуже стрімко. Також кожне масштабування передбачає перебудову цілого проекту й повторне його тестування. Велика кількість програмного коду, фреймворків, таблиць баз даних – все це негативно впливає на швидкість орієнтування в проекті працівників. Підтримувати такий додаток стає дуже важко та невигідно.

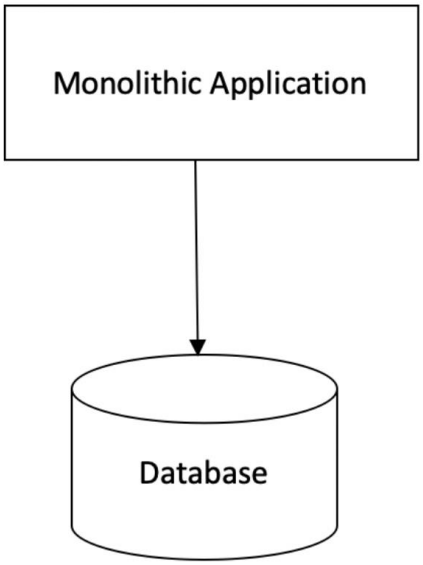


Рисунок 1.5 Представлення монолітної архітектури

Переваги монолітної архітектури :

- Легка реалізація
- Порівняно низька вартість розгортання
- Простий процес тестування

Недоліки монолітної архітектури

- Якщо система планується як розширювана, то з часом її складність буде зростати і підтримувати її буде важче
- Кожна внесена зміна, передбачає перебудову всього проекту
- Виникнення помилок при внесенні змін в проект
- Дуже важко здійснити перехід на інші технології

1.3.2 Мікросервісна архітектура

Мікросервіси як архітектурний стиль є спрощеним варіантом сервіс-орієнтованої архітектури (service-oriented architecture (SOA)), в якій сервіси орієнтовані на якісне виконання однієї дії. Сам мікросервіс як складову одиницю мікросервісної архітектури можна представити як сервіс, що постачається віддаленим API для використання системою. Він наділений однією дуже вузьконаправленою функціональною можливістю.

Застосування мікросервісів дозволяє використовувати різні підходи у проектуванні програмного забезпечення, адже кожен мікросервіс може бути реалізований з власними: набором інструментів розробки, мовою програмування та базою даних. Доступ до потрібної бази даних здійснюється шляхом виклику мікросервісу, що містить необхідну базу даних.

Мікросервіс повинен виконуватись в окремому процесі або окремих процесах, щоб залишатися якомога незалежнішим від інших мікросервісів тієї ж системи та мати можливість окремого розгортання. При внесенні змін в мікросервіс повинна бути можливість ввести цей змінений мікросервіс в експлуатацію, не розгортаючи інші частини системи. Інші мікросервіси в системі повинні виконуватися під час того як змінений мікросервіс розгортається, а також продовжувати виконання після того як нова версія буде розгорнута.

Факт того, що мікросервіси бажано розгортати окремо один від одного впливає на спосіб їх заємодії. Зміни в інтерфейсі повинні бути обернено сумісні, щоб інші мікросервіси могли продовжувати роботу з новою версією так як працювали зі старою.

Спосіб взаємодії мікросервісів повинен бути стійким до виникнення періодичних збоїв інших мікросервісів і при цьому продовжувати працювати в міру можливості. Припинення роботи одного мікросервісу, наприклад, через простої під час розгортання не повинен призводити до збою інших

мікросервісів, а лише до скорочення функціональності або трохи більш тривалий час обробки.

У мікросервіса повинно бути власне сховище зберігання даних, що є наслідком того, що кожен мікросервіс повинен бути функціонально завершеним. Це дозволяє використовувати різні технології баз даних для різних мікросервісів в залежності від їх потреб, що приносить велику користь в плані часу розробки, продуктивності та масштабованості системи. Недоліком є необхідність адміністрування і супровіду великої кількості баз даних.

Грамотно спроектовані та розроблені мікросервіси легко супроводжуються як в плані розгортання так і в плані експлуатації.

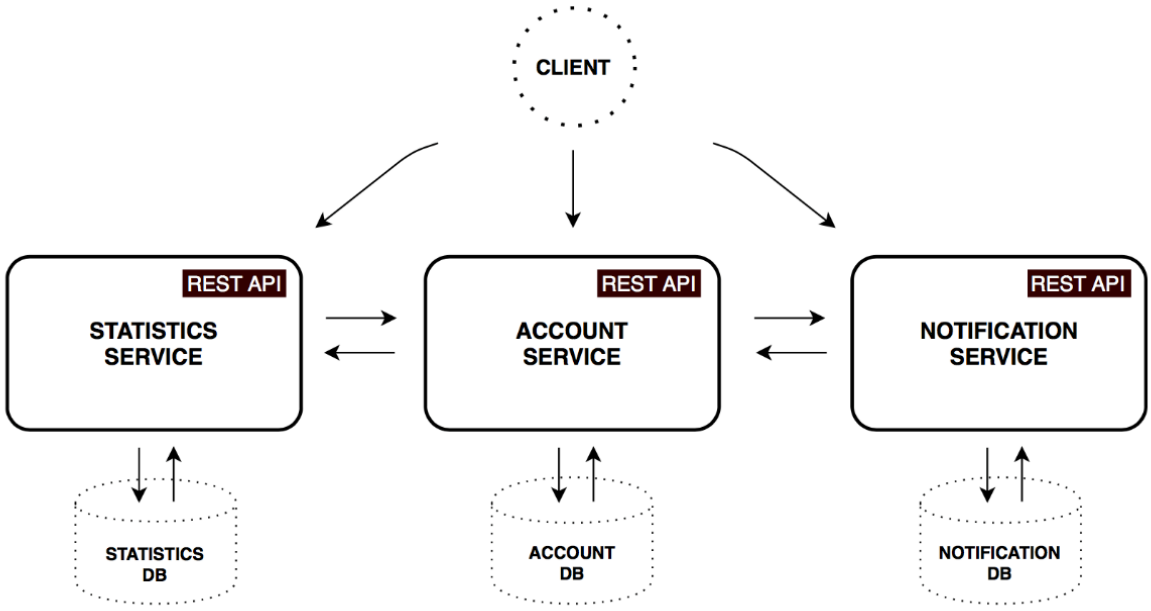


Рисунок 1.6 Мікросервісна архітектура

Характерні ознаки мікросервісів :

- Мікросервіс відповідає за одну функцію.
- Мікросервіси можливо розгортати окремо один від одного.

- Мікросервіс може містити один чи декілька тісно взаємопов’язаних процеси.
- Кожен мікросервіс має власне сховище даних.
- Мікросервіс можно легко замінити.
- Невелика команда розробників може супроводжувати кілька мікросервісів.

Мікросервісний архітектурний стиль швидко знаходить все більшу популярність при створенні і підтримці масштабного програмного забезпечення. Причина цього очевидна: мікросервіси забезпечують безліч потенційних переваг в порівнянні як з більш традиційними сервіс-орієнтованими підходами, так і з монолітною архітектурою.

Переваги мікросервісної архітектури :

- Легкість внесення змін в програмне забезпечення.
- Ізольованість.
- Висока відмовостійкість.
- Простота підтримки.
- Не має прив’язки до конкретних технологій.
- Один сервіс – це відповідальність за одну функціональну можливість.
- Сервіси можливо розгортати незалежно один від одного.
- Вилучення мікросервіса не впливає на роботу додатку.

Недоліки мікросервісної архітектури :

- Процес тестування.
- Складність реалізації правильної відмовостійкої системи координування між мікросервісими.
- Значні фінансові витрати, які ростуть із збільшенням кількості мікросервісів.

1.3.3 Порівняння мікросервісної та сервіс-орієнтованої архітектури

Сервіс-орієнтована архітектура - модульний підхід до розробки програмного забезпечення, заснований на використанні сервісів (служб) зі стандартизованими інтерфейсами. Дана архітектура з'явилася наприкінці 1980-х років.

В основі SOA лежать принципи багаторазового використання функціональних елементів, ліквідації дублювання функціональності у програмному забезпеченні, уніфікації типових операційних процесів, забезпечення переведення операційної моделі компанії на централізовані процеси та функціональну організацію на основі промислової платформи інтеграції.

Сервіс-орієнтована архітектура передбачає повторне використання сервісів та їх зв'язанність, а мікросервісна архітектура навпаки – уникає зв'язанності та дозволяє дублювання необхідної логіки для можливості незалежного існування мікросервісу. Вносити зміни до сервіс-орієнтованої архітектури дуже важко, адже перебудова одного сервісу передбачає внесення змін до всіх зв'язаних сервісів. Також якщо порівнювати процес тестування, то в сервіс-орієнтованій архітектурі він буде набагато складнішим, так як тестування одного сервісу передбачає також тестування всіх залежних від нього сервісів.

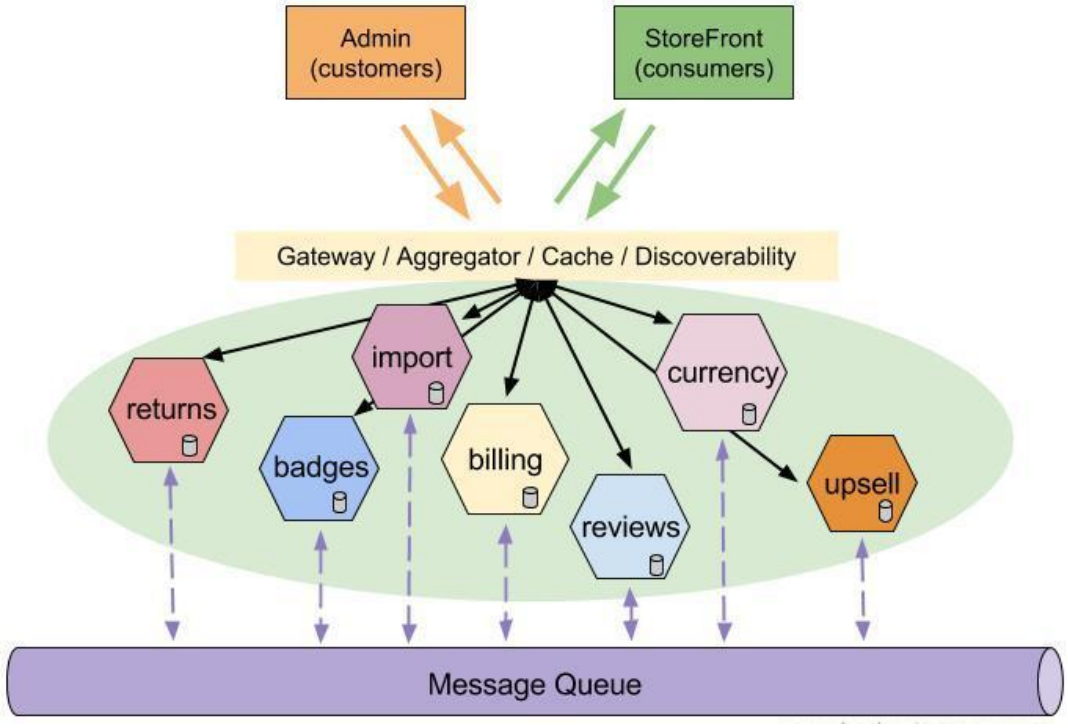


Рисунок 1.7 Схема сервіс-орієнтованої архітектури

Багато проблем з використанням SOA, пов'язані з протоколами обміну даними (наприклад, SOAP), постачальниками сполучних програмних засобів, відсутністю методик, що дозволяють визначити ступінь деталізації сервісів, або невірними методиками вибору місць розподілу системи.

1.3.4 Порівняння мікросервісної та монолітної архітектури

Таблиця 1.2 порівняння монолітної та мікросервісної архітектури

Характеристика	Мікросервісна архітектура	Монолітна архітектура
Складність системи	Складність системи зростає, але орієнтування в системі є досить легким.	Складність системи зростає стрімко та напряму залежить від масштабності системи. Орієнтування в системі стає надскладним завданням.
Легкість масштабування системи	Система масштабується легко. Розгортання нового сервісу не впливає на роботу системи в цілому.	Процес масштабування ускладнюється з ростом складності системи.
Тестування системи	Тестування правильності роботи окремих сервісів є дуже легким. Складно тестувати процес взаємодії сервісів.	Тестування є досить легким, коли складність системи є низькою. Тестування складної системи є дуже важкою задачею, адже в системі наявна велика кількість зв’язаного програмного коду.
Вартість системи	Порівняно велика.	Порівняно низька.
Розгортання системи	Всі сервіси системи можуть розгортатися як одночасно так і по черзі. Розгортання окремого мікросервісу не потребує перебудови всього проекту.	Система розгортається повністю як єдине ціле. Кожна зміна в проекті потребує його повної перебудови.
Зв’язаність елементів системи	Елементи системи є дуже малозв’язаними.	Елементи системи є дуже зв’язаними.
Бази даних	Безліч баз даних.	Одна база даних.

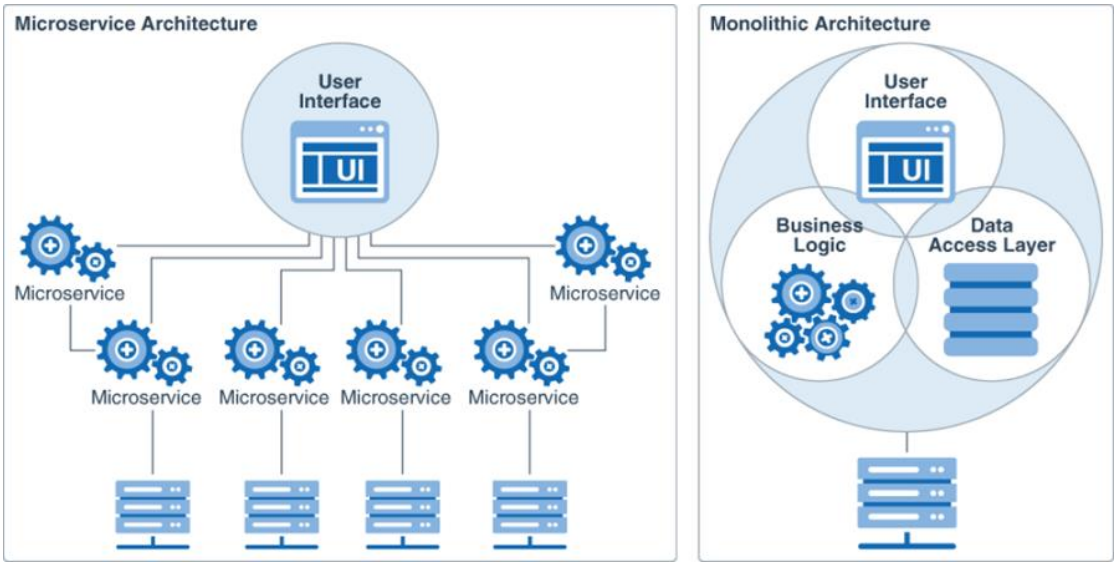


Рисунок 1.8 Порівняння мікросервісної та монолітної архітектури

1.4. Огляд існуючих технологій для створення та підтримки мікросервісів

Мікросервісна архітектура є новітнім напрямком у практиці створення програмного забезпечення, тому налічує невелику кількість перевірених інструментів розробки.

Docker – це платформа з відкритим кодом для розробки, доставки та розгортання контейнерних додатків. Docker дозволяє створювати контейнери, автоматизовувати їх запуск та розгортання. Кожен контейнер містить вихідний код програми із всіма необхідними залежностями для виконання цього коду в будь-якому середовищі. Дана платформа має можливість запуску великої кількості контейнерів на одній хост- машині.

Kubernetes – це платформа з відкритим кодом, що дозволяє управляти контейнерними робочими навантаженнями та сервісами. Kubernetes дозволяє автоматизувати роботу пов’язану з масштабуванням та обробкою помилок в контейнеризованому додатку.

Minikube — це інструмент, який дозволяє легко запустити Kubernetes на локальній машині. Minikube запускає одновузловий кластер Kubernetes всередині віртуальної машини на комп’ютері користувача.

Azure Functions — це хмарний обчислювальний сервіс, що дозволяє виконувати код, ініційований подіями, без попереднього налаштування середовища. Кожна функція містить частину програмного коду, що виконує певну логіку. Окремі функції можуть об’єднуватися в один додаток-функцію та використовувати одні й ті ж ресурси. Спосіб угруповання функцій та програм-функцій може вплинути на продуктивність, масштабування, конфігурацію, розгортання та безпеку загального рішення.

Nancy — полегшений фреймворк на платформі .NET, зручний для початку роботи з мікросервісами. Модулі фреймворку Nancy використовуються для створення та налаштування кінцевих точок API у мікросервісах додатку.

OWIN – стандарт, що визначає інтерфейс між веб-серверами .NET і веб-додатками. Метою інтерфейсу OWIN є відокремити сервер і додаток, заохочувати розробку простих модулів для веб-розробки в .NET.

Conductor – це механізм оркестрації мікросервісів, створений компанією Netflix. Ядром механізму є служба прийняття рішень. Коли відбувається виконання робочого процесу, сторона, що приймає рішення, об'єднує схему робочого процесу з поточним станом робочого процесу, визначає наступний стан і планує відповідні завдання або оновлює стан робочого процесу.

AWS Lambda – це хмарний обчислювальний сервіс, який запускає програмний код у відповідності до певних подій і відповідає за автоматичне виділення необхідних обчислювальних ресурсів. AWS Lambda можна використовувати для розширення можливостей інших сервісів AWS за допомогою спеціальної логіки або для створення власних серверних сервісів із застосуванням можливостей масштабування, ефективності та безпеки AWS.

Хмарні функції Google Cloud Platform – це легка безсерверна платформа, яку легко розгортати та обслуговувати. Її консоль надає

розробникам гарне рішення для створення мікросервісів, що керуються подіями. Контейнери платформи сплачуються залежно від використання.

Postman — це потужний інструмент тестування API. Postman допоможе зімітувати виклик контейнера та перевірити правильність різних ймовірних моделей взаємодії контейнерів додатку.

.NET Core web API — технологія для побудови додатків з API в стилі REST(передача стану представлення).

1.5 Огляд технологій для реалізації проекту

Для реалізації бекенду використовується платформа .NET та мова програмування C#.

Платформа ASP.NET Core – це технологія від компанії Microsoft. Позначення Core вказує на кросплатформеність, тобто з платформою можна працювати на різних пристроях з різною архітектурою та операційною системою. Перша версія платформи була випущена в 2016 році.

Платформа складається з таких основних частин:

.NET Runtime(CLR) — надає систему типів, завантаження виконуваного файлу, збирач сміття, вбудовану взаємодії та інші базові сервіси.

Фундаментальні бібліотеки — набір бібліотек фреймворків, які надають примітивні типи данних.

SDK & Compiler — набір із засобів розробки, утиліт і документації для платформи .NET Core.

Хост ‘dotnet’ — використовується для запуску додатків .NET Core. Він обирає середовище виконання, надає політику завантаження виконуваних файлів і завантажує додатки.

Також ASP.NET Core включає такі засоби як :

MVC – це архітектурний шаблон, що передбачає поділ програми на три частини: модель, представлення та котролер. Модель відповідає за логіку пов’язану з обробкою даних. Представлення відповідає за зовнішній вигляд програми та відображення даних. Контролер формує зв’язок між моделлю даних та представленням.

Web API – прикладний програмний інтерфейс. API визначає методи взаємодії між компонентами програмного забезпечення, а також правила взаємодії однієї програми з іншими програмами.

Razor Pages — це новий набір інструментів у .NET Core для створення динамічних веб-сторінок.

Мова програмування C# — об`єктно-орієнтована мова програмування, що у поєднанні з платформою .NET є одночасно легким та сильним інструментом розробки бекенд частини як веб-додатків так і десктопних програм.

Для реалізації фронтенду використовується фреймворк Angular.

Angular — це фреймворк від компанії Google, що дозвляє розробляти клієнтську частину web-додатків.

Як платформа, Angular включає:

- Компонентний фреймворк для створення масштабованих веб-додатків
- Колекцію добре інтегрованих бібліотек, які охоплюють широкий спектр функцій, включаючи маршрутизацію, керування формами, зв’язок клієнт-сервер тощо.
- Набір інструментів для розробників, які допомогають розробляти, створювати, тестувати та оновлювати код

Розробка на даній платформі ведеться за допомогою мови TypeScript, а також HTML5 та CSS3.

Основними елементами додатку побудованого за допомогою Angular є модулі, компоненти та сервіси.

Якщо програмне забезпечення побудоване в напрямку монолітної архітектури, то взаємодія клієнтскої частини та серверної частини буде виконуватися:

Для клієнтскої частини - за допомогою класу HttpClient, який дозволяє формувати HTTP запит з клієнтскої частини до серверної частини за певним API

Для серверної частини – за допомогою класів контролерів, де буде зазначено API, за яким клієнт має звертатися до серверу.

Якщо програмне забезпечення побудоване в напрямку мікросервісної архітектури, то взаємодія клієнтскої частини та серверної частини буде виконуватися лише з однією відмінністю: клієнт буде звертатися не до одного класу з контролерів серверної частини, а до окремого мікросервісу, що вміщатиме в собі API та всю необхідну логіку для самостійного існування.

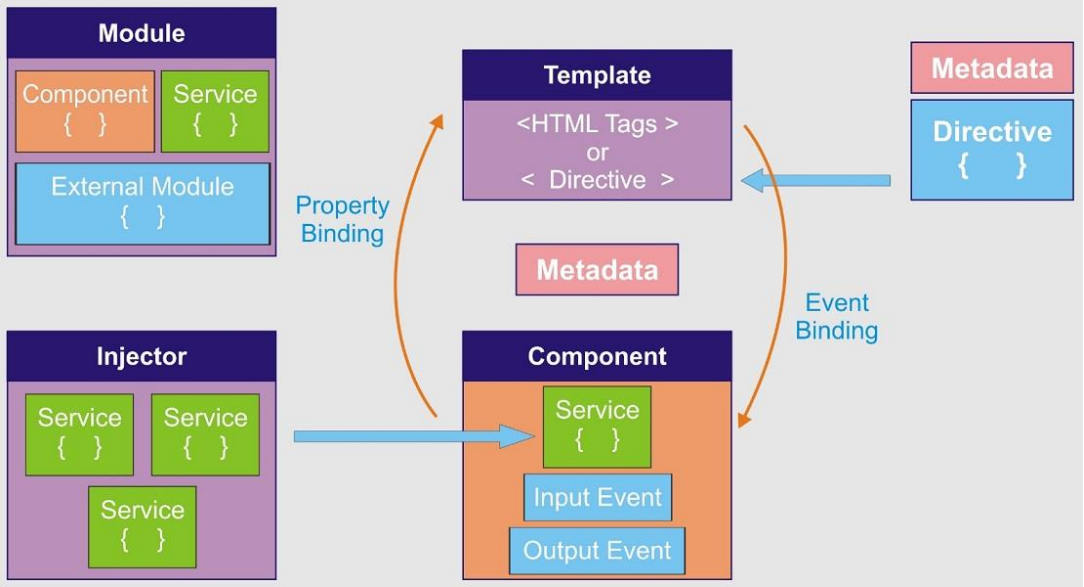


Рисунок 1.9 Взаємодія елементів клієнтської частини

Для того, щоб створити контейнери, використовується платформа Docker.

Docker – це платформа для розробки, доставки та запуску контейнерних програм. Docker дозволяє створювати контейнери, автоматизувати їх запуск та розгортання, а також керує їх життєвим циклом. Він дозволяє запускати безліч контейнерів на одній машині.

Контейнери дозволяють упакувати в єдиний образ програму та всі її залежності: бібліотеки, системні утиліти та файли налаштування.. Цей образ запускається в ізольованому середовищі, що не впливає на основну операційну систему. Контейнери дозволяють відокремити додаток від інфраструктури: розробникам не потрібно замислюватися, в якому оточенні працюватиме їхня програма, чи будуть там потрібні налаштування та залежності. Вони просто створюють програму, упаковують всі залежності та налаштування в єдиний образ. Потім цей образ можна запускати на інших системах, не турбуючись, що програма не запуститься.

1.6 Відмінність понять контейнеризації та віртуалізації

В мікросервісній архітектурі широко використовується поняття контейнеризації, яке перемежовується з поняттям віртуалізації.

Віртуалізація — це процес запуску віртуального екземпляра комп’ютерної системи на рівні, відокремленому від фактичного обладнання. Найчастіше це стосується одночасного запуску кількох операційних систем на комп’ютерній системі. Додаткам, які запущені поверх віртуалізованої машини, може здаватися, що вони знаходяться на власній спеціалізованій машині, де операційна система, бібліотеки та інші програми є унікальними для гостьової віртуалізованої системи і не підключені до хостової операційної системи, яка знаходиться під ним.

Контейнеризація — це підхід до розробки програмного забезпечення, при якому програма або служба, їх залежності та конфігурація (абстрактні файли маніфесту розгортання) упаковуються разом у образ контейнера. Контейнерна програма може тестуватися як єдине ціле та розгортатися як екземпляр образу контейнера в операційній системі.

Контейнери концептуально схожі на віртуальні машини, але функціонують дещо інакше. Хоча як контейнери, так і віртуальні машини дозволяють запускати програми в ізольованому середовищі як ніби це окремі комп'ютери, контейнери не є повними незалежними машинами. Насправді контейнер — це просто ізольований процес, який використовує те саме ядро, що й операційна система хоста, а також бібліотеки та інші файли, необхідні для виконання програми, що працює всередині контейнера. Як правило, контейнери призначені для виконання однієї програми, а не для емуляції повноцінного багатоцільового серверу.

1.7 Побудова архітектури проекту. Розробка інтерфейсу користувача

Важливим этапом проектування проектування програмного забезпечення є визначення його об`єктної моделі. Цілью визначення об`єктної моделі є виявлення та опис усіх об`єктів програми, що є абстрактним представленням об`єктів реального світу в предметній області програми.

Для реалізації сайту була обрана тема магазину гаджетів, то ж об `єктну модель моєї програми складають:

1. покупець
2. адміністратор
3. товар
4. корзина
5. замовлення

За допомогою фреймворку EntityFramework та підходу code first в проектуванні бази даних, представлення даних сутностей у вигляді класів буде перенесено в таблиці бази даних.

Інтернет-магазин на мікросервісах – це програма з сучасною архітектурою програмного забезпечення, яка чітко розроблена і заснована на легких технологіях .NET. Всього програма налічує 3 мікросервіси: мікросервіс основної частини сайту, мікросервіс корзини, мікросервіс панелі адміністратора. Кожен мікросервіс вміщає в собі частину бекенду, побудовану за N-рівневою архітктурою.

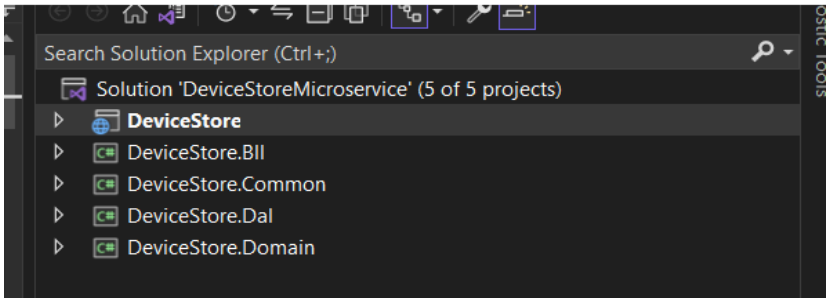


Рисунок 1.10 Структура мікросервісу основної частини сайту

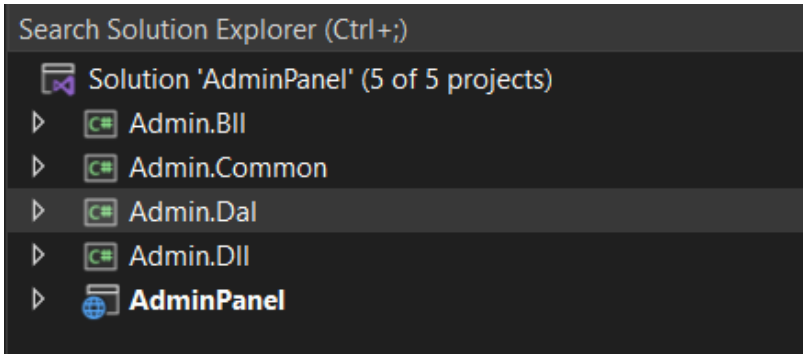


Рисунок 1.11 Структура мікросервісу панелі адміністратора

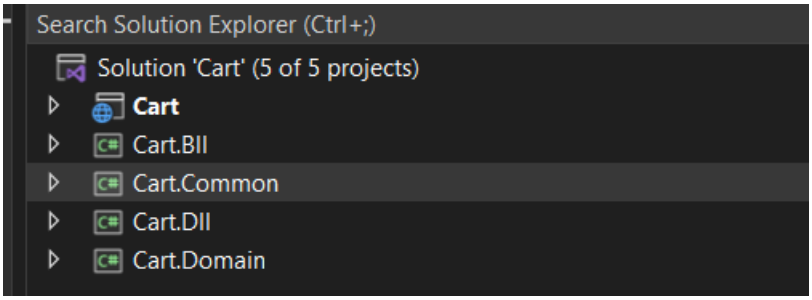


Рисунок 1.12 Структура мікросервісу корзини

Основна частина роботи полягає в створенні такого функціоналу:

1. Інтерфейс користувача
2. Аутентифікація та авторизація
3. Адмін панель
4. Обслуговування кошика
5. Сервіс замовлень

Бекенд-мікросервіси написані на C# та взаємодіють з мікросервісами бази даних MSSQL та фронтендом. Частина фронтенду написана за допомогою фреймворку Angular, мови typescript. Взаємодія мікросервісів між собою здійснюється за допомогою побудови REST API та спільною мережі.

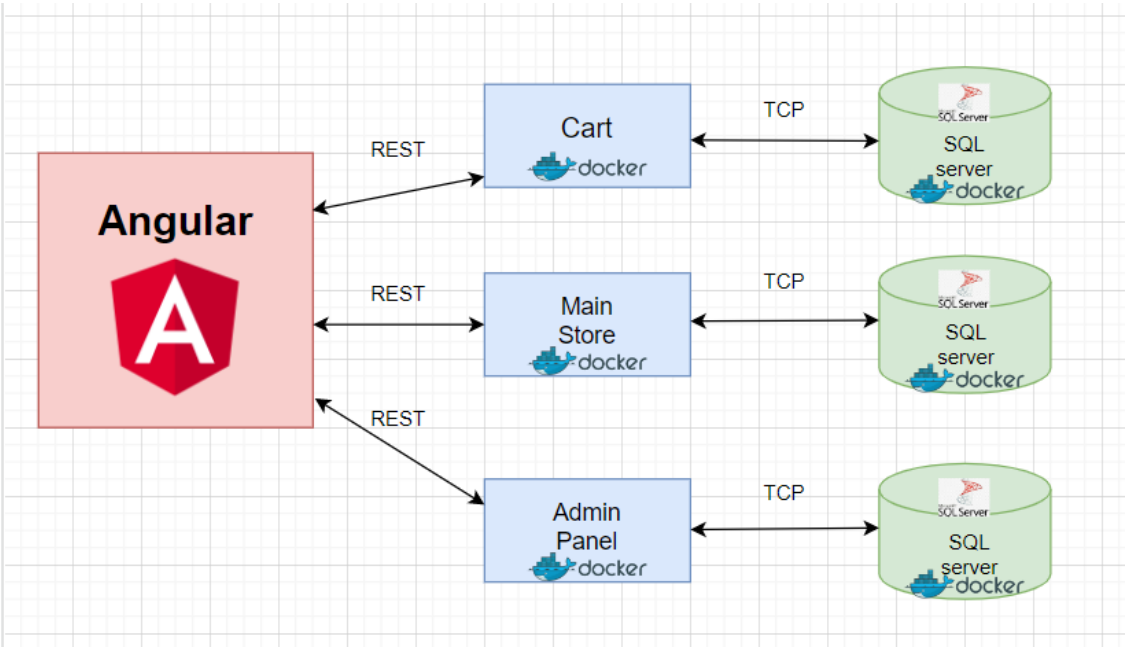


Рисунок 1.13 Схема взаємодії мікросервісів додатку

Дуже важливо під час проектування програмного забезпечення продумати інтерфейс користувача. Потрібно чітко визначити майбутні дії користувача на сайті та після цього створити дизайн. Інтерфейс користувача створюватиметься за допомогою таких засобів як: HTML, CSS3, Angular Material UI. Для створення дизайну використовуються спокійні кольори.

Покупець на моєму сайті може виконувата такі дії: реєструватися, авторизуватися, переглядати каталог товарів, додавати товар до “улюблених”, додавати товар до корзини, переглядати історію замовлень. То ж для втілення цього функціоналу мені потрібні такі компоненти: діалогові вікна, поля вводу, кнопки, форми, слайдери, списки, значки.

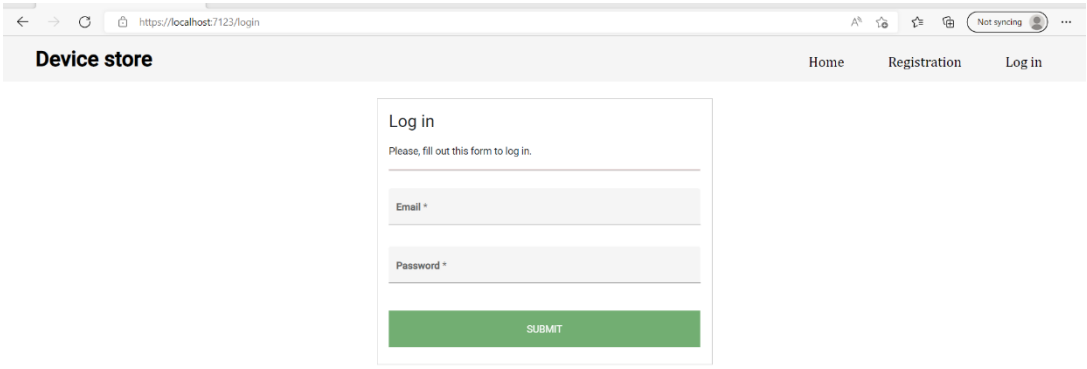


Рисунок 1.14 Сторінка авторизації покупця

Адміністратор на сайті може виконувати такі дії: авторизуватися, додавати нові категорії, змінювати категорії, видаляти категорії, додавати нові продукти, змінювати проодукти, видаляти продукти. Для втілення цього функціоналу потрібні такі компоненти: діалогові вікна, поля вводу, кнопки, таблиці.

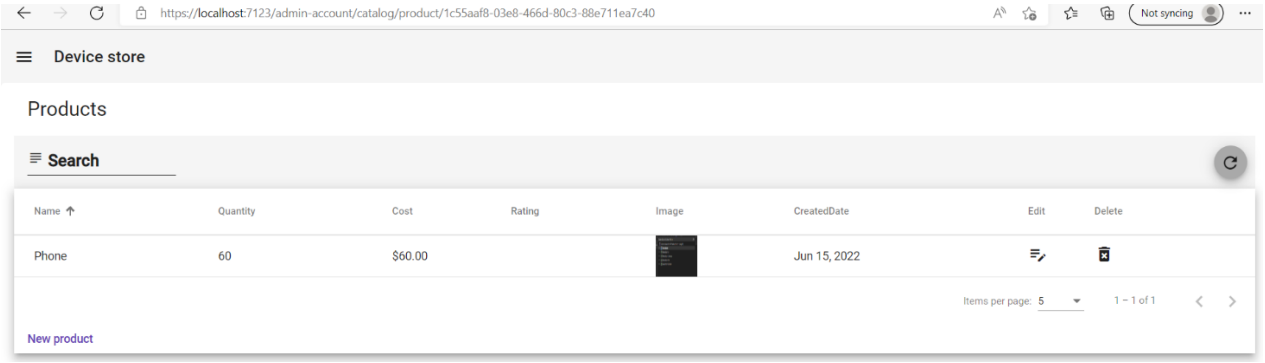
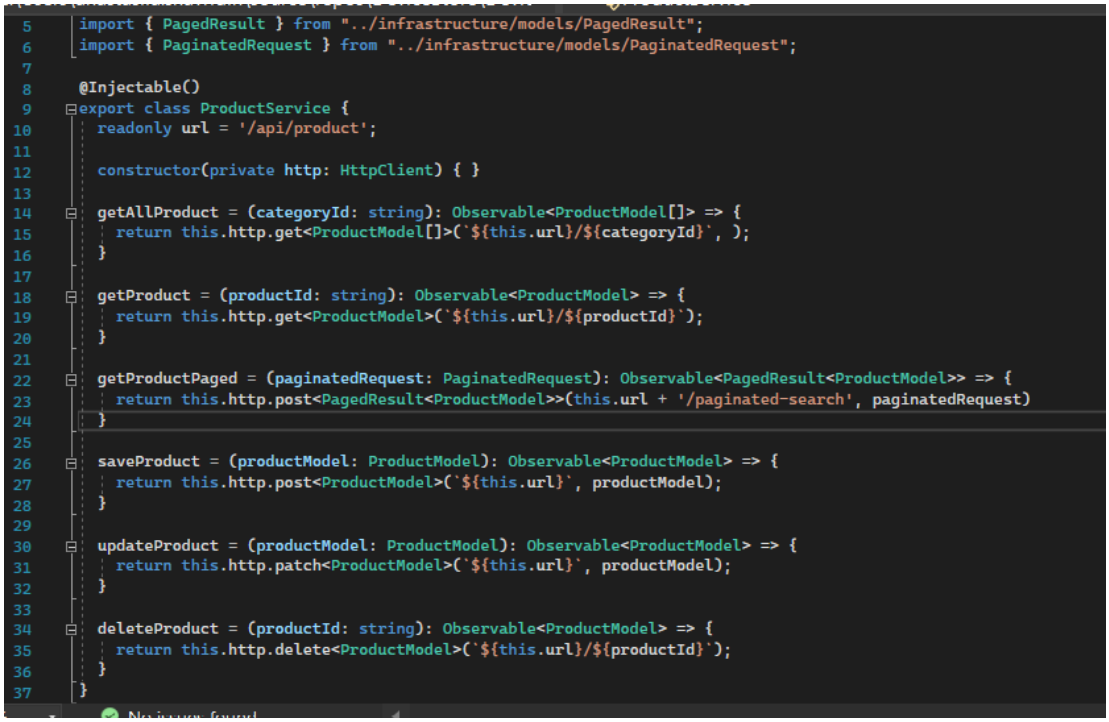


Рисунок 1.15 Сторінка продуктів на панелі адміністратора

1.8. Реалізація частини фронтенду

Для написання частини фронтенду використовувався фреймворк Angular. Основною задачею фронтенду є отримування даних від користувача та відправка їх на бекенд і навпаки. Обмін даними з бекендом відбувається в сервісах. Якщо дані отримаються з бекенду для відображення, то вони передаються в компоненти, в яких відбувається необхідні перетворення над даними та їх вивід в HTML розмітці.

Розглянему реалізацію сторінки товарів на фронтенді. В сервісі є низка CRUD операцій аби отримати, оновити, відправити, видалити продукти. Методи сервісу використовують асинхронний паттерн проектування Observable, що дозволяє представити дані у вигляді потоку подій. Потоки можливо комбінувати, перевикористовувати, фільтрувати. Асинхронність є дуже важливим критерієм для побудови швидкого додатку, адже дозволяє виконувати одночасно декілька операцій без блокування основного потоку.



```
import { PagedResult } from "../infrastructure/models/PagedResult";
import { PaginatedRequest } from "../infrastructure/models/PaginatedRequest";

@Injectable()
export class ProductService {
  readonly url = '/api/product';

  constructor(private http: HttpClient) { }

  getAllProduct = (categoryId: string): Observable<ProductModel[]> => {
    return this.http.get<ProductModel[]>(`${this.url}/${categoryId}`, );
  }

  getProduct = (productId: string): Observable<ProductModel> => {
    return this.http.get<ProductModel>(`${this.url}/${productId}`);
  }

  getProductPaged = (paginatedRequest: PaginatedRequest): Observable<PagedResult<ProductModel>> => {
    return this.http.post<PagedResult<ProductModel>>(this.url + '/paginated-search', paginatedRequest)
  }

  saveProduct = (productModel: ProductModel): Observable<ProductModel> => {
    return this.http.post<ProductModel>(`${this.url}`, productModel);
  }

  updateProduct = (productModel: ProductModel): Observable<ProductModel> => {
    return this.http.patch<ProductModel>(`${this.url}`, productModel);
  }

  deleteProduct = (productId: string): Observable<ProductModel> => {
    return this.http.delete<ProductModel>(`${this.url}/${productId}`);
  }
}
```

Рисунок 1.16 Сервіс з CRUD операціями для опрацювання даних продуктів.

Далі сервіс визивається в компоненті таблиці. Для того, щоб вивести усі продукти з пагінацією в таблиці, додати новий продукт до таблиці, оновити продукт в таблиці та видалити продукт з таблиці використовуються методи з цього сервісу. Для того, щоб Observable метод виконався, потрібно виконати підписку на Observable.

```
confirm()
{
  this.orderService.saveOrder(this.cartData).subscribe()
}

reset(cartId: string)
{
  this.cartService.deleteCart(cartId).subscribe();
}

delete(productId: string, cartId: string )
{
  this.cartService.deleteProductFromCart(productId, cartId).subscribe()
}
}
```

Рисунок 1.17 Методи, що визиваються в компоненті та підписуються на Observable

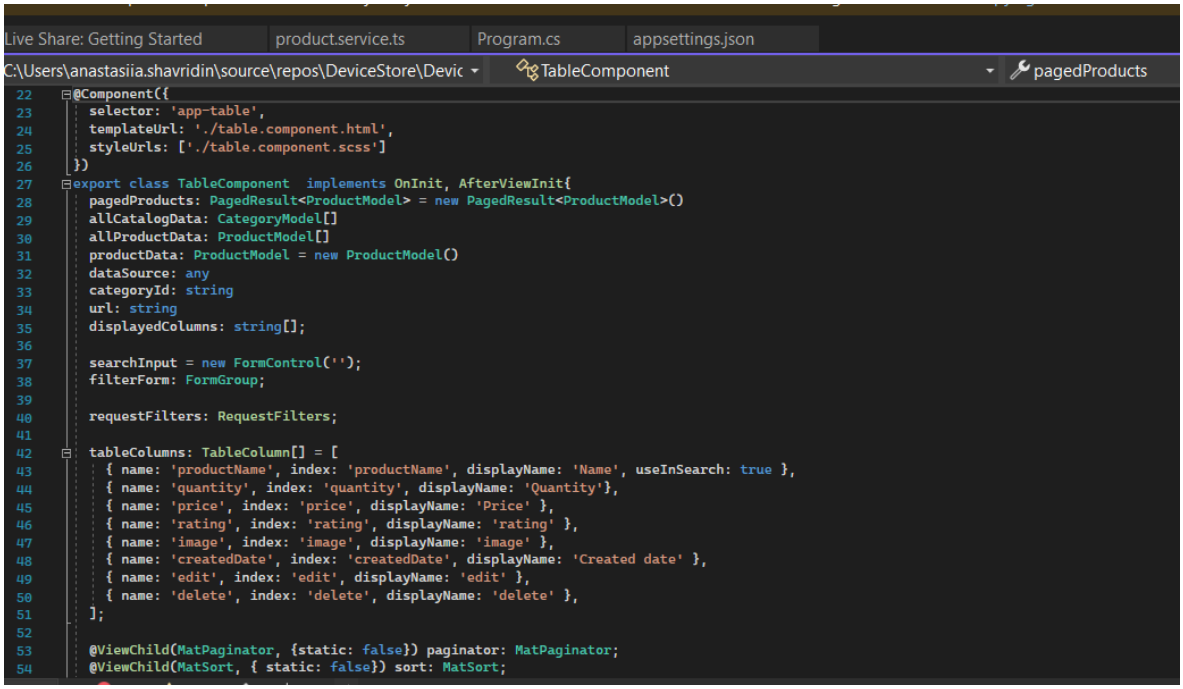


Рисунок 1.17 Компонент таблиці

Після чого отримані дані додаються в дизайн, або навпаки беруться з дизайну.

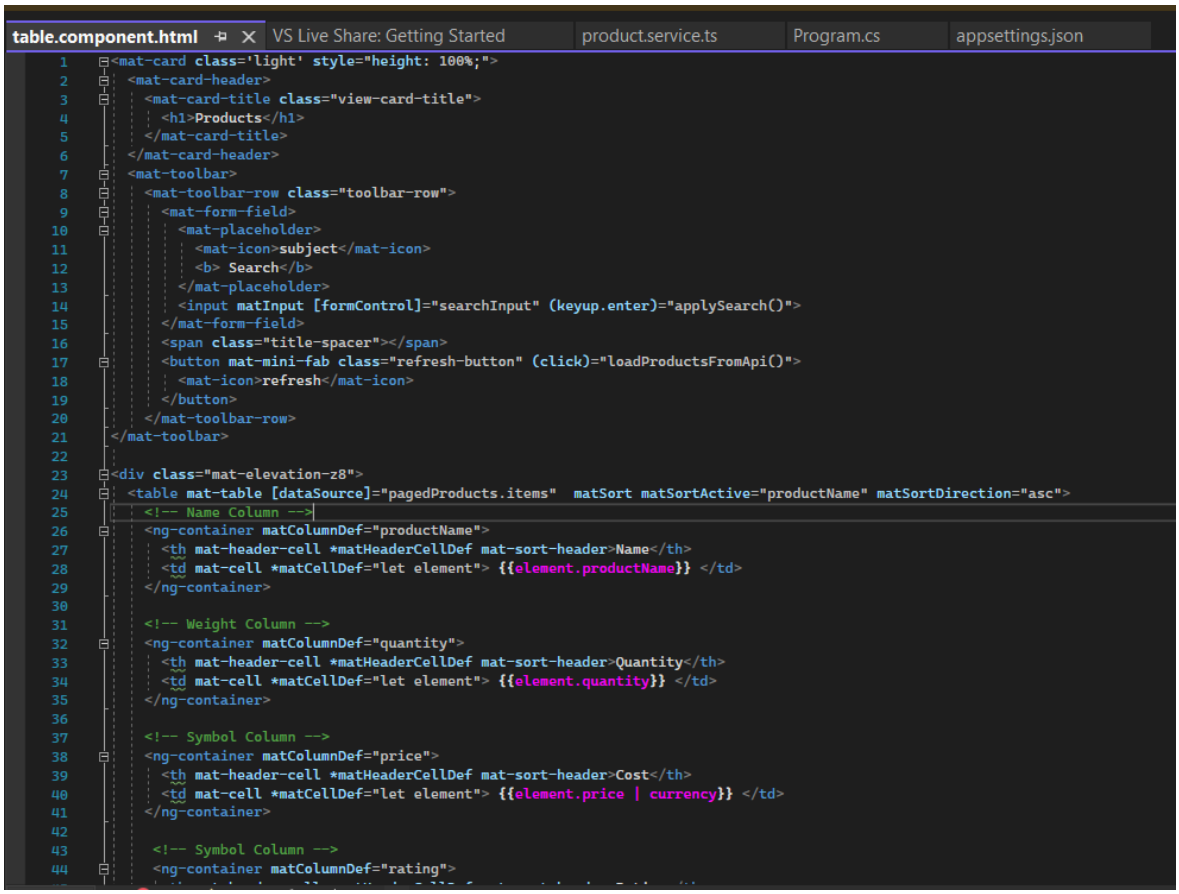


Рисунок 1.18 HTML розмітка сторінки продуктів

1.9 Реалізація мікросервісів на бекенді

Розглянемо реалізацію мікросервісу корзини. Мікросервіс складається з таких частин:

ASP.NET Web API Core – що вміщає в собі API для взаємодії з фронтендом, а також файл program, де відбувається налаштування і запуск проекту. Він має посилання на бібліотеки Dal, Bll та Common.

```
44          [HttpGet]
            0 references
45          public async Task<IActionResult> GetCart()
46          {
47              try
48              {
49                  if (!this.ModelState.IsValid)
50                  {
51                      return this.BadRequest(this.ModelState);
52                  }
53
54                  var result = await this.cartService.GetCart(this.UserId);
55
56                  if (!result.Equals(null))
57                  {
58                      return this.Ok(result);
59                  }
60
61                  return this.BadRequest("Some problem with saving cart");
62              }
63              catch (Exception ex)
64              {
65                  return this.BadRequest($"Category creation failed. Error message: {ex.Message}");
66              }
67          }
```

Рисунок 1.19 Метод отримання корзини в контролері

HttpGet – вказує, що цей метод оброблює запит на отримання корзини клієнтом. Далі отримується Id зареєстрованого користувача, що передається в метод GetCart сервісу корзини для отримання корзини даного користувача.

Cart.Bll – це бібліотека, що вміщає в собі логіку, яка реалізує бізнес можливості проекту. На цьому рівні знаходяться сервіси, які отримують дані у вигляді DTO(об`єкт передачі даних) та моделей. У сервісах виконується перетворення моделей та DTO в сутності бази даних та реалізується уся потрібна логіка обробки даних.

```
105     public async Task<CartModel> GetCart(string userId)
106     {
107         userId = userId ?? throw new ArgumentNullException(nameof(userId));
108
109         var customer = await customerRepository.FindCustomerByUserId(userId);
110
111         List<ProductModel> productsList = new();
112
113         if (!Equals(customer, null))
114         {
115             var cart = await this.cartRepository.FindCartByCustomerId(customer.CustomerId);
116
117             if (!Equals(cart, null))
118             {
119                 var products = cart.Products;
120
121                 foreach (var product in products)
122                 {
123                     productsList.Add(mapper.Map<Product, ProductModel>(product));
124                 }
125
126                 var cartModel = mapper.Map<Cart, CartModel>(cart);
127
128                 cartModel.ProductModels = productsList;
129
130                 return cartModel;
131             }
132
133             return null;
134         }
135         else
136         {
137             throw new Exception("Cart doesn`t exists");
138         }
139     }
```

Рисунок 1.20 Метод отримання корзини в сервісі

Cart.Common – це бібліотека, яка зберігає DTO та моделі.

DTO – це шаблон, який містить окремі поля сутності і створюються для відправки та отримання тільки тих даних, які необхідні згідно вимог бізнес логіки.

Моделі – це шаблон, що повністю повторює структуру сутності, тобто має усі поля, що містить об`єкт бази даних.

```
9 references
public class CartModel : ApplicationModel
{
    [StringLength(36)]
    0 references
    public string CartId { get; set; }

    0 references
    public decimal Sum { get; set; }

    0 references
    public int Count { get; set; }

    2 references
    public List<ProductModel> ProductModels { get; set; }

    0 references
    public UserModel UserModel { get; set; }
}
```

Рисунок 1.21 Модель корзини

Cart.Dal – це бібліотека, яка вміщає в собі логіку взаємодії з базою даних. На цьому рівні знаходяться репозиторії, в яких реалізуються операції з базою даних - додавання, оновлення, отрмання, видаленння сутностей.

```
public class CartRepository: BaseRepository, ICartRepository
{
    0 references
    public CartRepository(DataContext context) : base(context)
    {
    }

    3 references
    public async Task<Cart> FindCartByCustomerId(string customerId)
    {
        return await this.context.Carts.Include(p => p.Products).Where(c => c.CustomerId.Equals(customerId)).FirstOrDefaultAsync();
    }

    2 references
    public async Task<Cart> GetCartWithProductsById(string cartId)
    {
        return await this.context.Carts.Include(p => p.Products).Where(a => a.CartId.Equals(cartId)).FirstOrDefaultAsync();
    }
}
```

1.22 Репозиторій корзини

Даний репозиторій включає в себе 2 методи: отримання корзини за ідентифікаторм користувача, та отримання корзини за ідентифікатором корзини. Для пошуку запису корзини в базі даних використовується бібліотека LINQ – мова інтегрованих запитів.

Cart.Dll – це бібліотека, в якій знаходяться сутності бази даних.

```
6   public class Cart : Application
7   {
8       [Key]
9       [DatabaseGenerated(DatabaseGeneratedOption.None)]
10      [StringLength(36)]
        2 references
11      public string CartId { get; set; }
12
13      [ForeignKey("Customer")]
14      [StringLength(36)]
        2 references
15      public string CustomerId { get; set; }
16
17      [Column(TypeName = "decimal(18,4)")]
        2 references
18      public decimal? Sum { get; set; }
19
        2 references
20      public int? Count { get; set; }
21
        0 references
22      public virtual Customer Customer { get; set; }
23
        6 references
24      public virtual ICollection<Product> Products { get; set; }
25
        1 reference
26      public virtual ICollection<CartProduct> CartProducts { get; set; }
27  }
```

1.23 Сутність корзини

Як зазначалось на стадії проектування, сайт складається з 3 мікросервісів. Усі мікросервіси містяться в docker контейнерах. Для того, щоб запустити мікросервіс, потрібно спочатку створити образ – шаблон для запуску контейнера, що буде містити в собі весь програмний код та середу виконання. Для створення образів та запуску контейнерів я використовую docker-compose файл, що дозволяє запустити одночастно усі контейнери в одній спільній мережі.

```yaml
version: '3.4'

services:
  cart:
    image: ${DOCKER_REGISTRY-}cart
    ports:
      - 8080:80


  cartDb:
    image: mcr.microsoft.com/mssql/server
    environment:
        ACCEPT_EULA: Y
        SA_PASSWORD: myPassword20
    ports:
      - 1430:1433

  admin:
    image: ${DOCKER_REGISTRY-}admin
        ports:
      - 5000:80

  adminDb:
    image: mcr.microsoft.com/mssql/server
    environment:
        ACCEPT_EULA: Y
        SA_PASSWORD: myPassword20
    ports:
      - 1431:1433

  devicestore:
    image: ${DOCKER_REGISTRY-}devicestore
        ports:
      - 3000:80

  deviceStoreDb:
    image: mcr.microsoft.com/mssql/server
    environment:
        ACCEPT_EULA: Y
        SA_PASSWORD: myPassword20
    ports:
      - 1432:1433

    build:
      context: .
```

Рисунок 1.24 Вміст docker-compose файлу

version – дозволяє вказата версію специфікації docker-compose.

services – дозволяє вказати всі контейнери для запуску шляхом вказання назви контейнеру та назви образу, який буде використовуватися для запуску. Також для кожного контейнера можно вказати порт запуску.

image – образ, який вміщає програмний код та середу розробки підготовлені для запуску.

environment – дозволяє задати змінні середи в контейнері. В моємі випадку – це змінні, що є налаштуваннями для запуску бази даних.

ports – порти на яких запускаються, контейнери.

build – запускає процес побудови контейнерів.

1.10 Взаємодія мікросервісів

Мікросервіси є дрібноструктурованими елементами з вузькою областю дії. Для доставки функціональності кінцевому користувачеві мікросервіси повинні взаємодіяти. Мікросервіс відповідає за одну можливість і зазвичай це бізнес-можливість. Призначені для кінцевого користувача функціональні можливості - сценарії використання, часто включають в себе кілька бізнес-можливостей, тому мікросервіси, що реалізують ці можливості, повинні взаємодіяти для надання кінцевому користувачеві необхідної функціональності.

Існує три основні стилі взаємодії двох мікросервісів:

- Команди. Використовуються, коли одному мікросервісу необхідно, щоб інший виконав якусь дію.
- Запити. Застосовуються, якщо одному мікросервісу потрібна інформація від іншого.
- Події. Використовуються, коли мікросервісу необхідно реагувати на події, що відбувається в іншому мікросервісі.

В процесі взаємодії двох мікросервісів можуть використовуватися один, два або всі три стилі взаємодії. Кожен раз, коли потрібно реалізувати взаємодію мікросервісів потрібно вирішити, який стиль використовувати.

При основаній на командах та запитах взаємодії використовуються високорівневі команди та запити. Запити вважаються ввіддаленими, тому що воно проходять по мережі перш ніж дістануться отримувача. Команди та запити обробляються синхронно.

Як команди, так і запити є синхронними формами взаємодії. Вони реалізуються у вигляді HTTP-запитів від одного мікросервісу до іншого. Запити реалізуються за допомогою HTTP-запитів типу GET, а команди - за допомогою HTTP-запитів типу POST або PUT.

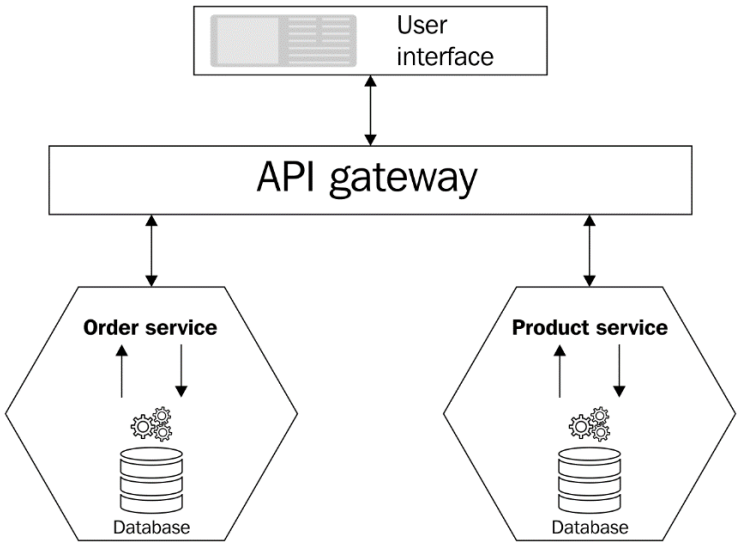


Рисунок 1.25 Схема взаємодії мікросервісів з використанням стилів команд та запитів

Взаємодія побудована на подіях зв’язує мікросервіси меньше в плані часу, ніж команди та запити, адже виконується асинхронно. Це означає, що публікуючий події мікросервіс не виконує звернення до підписаних на ці події мікросервісів. Мікросервіси-отримувачи опитують публікуючий події мікросервіс на предмет нових подій.

Мікросервіси можуть опубліковувати події для інших мікросервісів за допомогою стрічки подій(event feed), що представляє собою кінцеву точку HTTP за адресою /events. Підписка на стрічку подій означає опитування кінцевої точки мікросервісу. Через певний інтервал часу мікросервіс-підписник здіймнює відправку HTTP запиту типу GET до кінцевої точки, щоб перевірити чи не з'явились не оброблені події призначені для нього.

Мої мікросервіси побудовані на стилі запитів, тобто якщо одному мікросервісу потрібно отримати якісь дані, він звертається за потрібним API іншого мікросервісу.

Щоб протестувати можливість окремого мікросервісу відправляти та отрисувати дані можна скористатися інструментом Postman. Так в даному випадку я перевіряю здатність мікросервіса відповідати на запити.

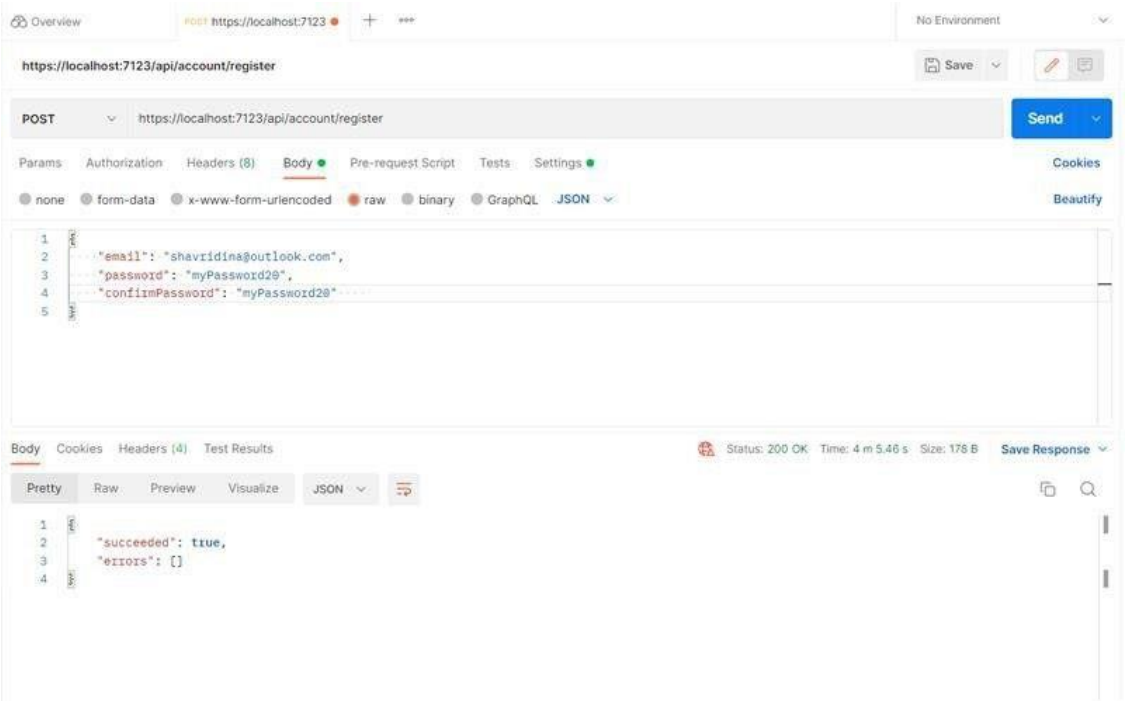


Рисунок 1.26 Тестування працездатності мікросервісу.

Метод POST дозволяє відправити дані у мікросервіс за вказаним портом localhost та шляхом до потрібного API. У Body вказуються дані для відправки у форматі JSON.

2 ЕКОНОМІЧНИЙ РОЗРАХУНОК

Метою даних розрахунків є обчислення вартості виконання науково-дослідної роботи «Дослідження переваг застосування мікросервісної архітектури в порівнянні з монолітним додатком в розробках веб-орієнтованих систем». Мікросервісна архітектура передбачає розробку програм, які розділені на мікросервіси. Кожен мікросервіс містить елемент програми, який характеризується вузькоспрямованою функціональною можливістю: кожен сервіс відповідальний за конкретний процес, згідно поставлених перед ним задач. Даний вид проекту відноситься до науково-дослідницької розробки. Оцінка якості розробленого проекту включає визначення трудомісткості і вартості його створення.

Розрахунок трудомісткості НДР здійснений в наступній послідовності:

1) Складений перелік всіх етапів і видів робіт, які необхідно виконати в ході даної НДР. Після узгодження з керівником проекту допущено виключення, доповнення, об'єднання окремих етапів і видів робіт;

2) По кожному виду робіт визначений кваліфікаційний рівень виконавців. Перелік етапів і робіт, що виконуються при проведенні НДР, приведений в таблиці 3.1.

Таблиця 2.1. - Розподіл робіт по етапах і видах виконавців.

Етап проведення НДР	**Вигляд робіт**	**Посада виконавця**	
Розробка технічного завдання (ТЗ)	1.Складання і затвердження ТЗ для НДР по розробці «Дослідження переваг застосування мікросервісної архітектури в порівнянні з монолітним додатком в	Дипломник, керівник	

	розробках веб-орієнтованих систем»»		
Вибір напряму дослідження	1. Збір і вивчення науково-технічної літератури. 2. Формулювання можливих напрямів вирішення завдань, поставлених в технічному завданні НДР і їх порівняльна оцінка. 3. Вибір напряму проведення досліджень 4. Розробка плану проведення досліджень для подальшої розробки.	Дипломник керівник	
Теоретичні і експериментальні дослідження	1.Огляд мікросервісної та монолітної архітектур 2. Використання мікросервісної архітектури увеб-орієнтованої системі.	Дипломник керівник консультанти	
Узагальнення і оцінка результатів досліджень	1. Узагальнення результатів 2. Оцінка повноти вирішення поставлених завдань. 3.Складання і оформлення звіту. Розгляд результатів проведеною НДР і прийняття результатів в цілому.	Дипломник керівник консультанти	

Оцінка тривалості виконання робіт розраховується на основі вірогідних оцінок робіт, що задаються виконавцями.

Таблиця 2.2.- Очікувана трудомісткість робіт

Вигляд роботи	**Очікуваний час виконання (дні)**
1. Складання і затвердження ТЗ для НДР «Дослідження переваг застосування мікросервісної архітектури в порівнянні з монолітним додатком в розробках веб-орієнтованих систем».	2
2. Збір і вивчення науково – технічної літератури, технічної документації і інших матеріалів.	5
3. Формулювання можливих напрямів вирішення завдань, поставлених в технічному завданні НДР і їх порівняльна оцінка.	3
4. Розробка плану проведення досліджень для подальшої розробки.	2
5. Огляд мікросервісної та монолітної архітектур	5
6. Використання мікросервісної архітектури увеб-орієнтованої системі.	6
Всього:	23

Розрахунок собівартості і ціни виконання НДР. Виходячи з особливостей створення науково – технічної продукції і її залежності від інтелектуальної праці, розрахунок собівартості і ціни виконання НДР включає наступні статті витрат: витрати на матеріали, основна і додаткова заробітна плата, відрахування до єдиного соціального фонду страхування, витрати на роботи, що виконуються сторонніми організаціями, і деякі інші.

1) Витрати на матеріали складають 165 грн.

2) До витрат «Основна заробітна плата» відносяться оплата праці виконавців, безпосередньо притягнених до її виконання. Розмір основної зарплати встановлюється виходячи з чисельності різних категорій виконавців, трудомісткості, що витрачається ними на виконання різних видів робіт, а

також їх середньої заробітної плати (ставки) за один робочий день. Відповідно до статті 8 «Закону про Державний бюджет України на 2021» встановлено мінімальну заробітну плату у місячному розмірі з з 1 січня 2022 року - 6500 гривень; мінімальну погодинну тарифну ставку – 39,26 грн.

Середня зарплата за один робочий день для кожного виконавця визначена по формулі:

Зден= п.т.с. * 8;

де п.т.с – погодинна тарифна ставка, грн..;

8 – тривалість робочого дня,год.

Зден дипломника = 39.26*8 = 314,08 грн.

Зден керівника = 64.00 *8 = 512 грн.

Зден консультантів = 61.00 *8 = 488 грн.

Витрати на основну заробітну плату, НДР, що включаються в собівартість, приведені в таблиці 3.3.

Таблиця 2.3.- Витрати на основну заробітну плату.

Виконавець	**Погодинна тарифна ставка, грн**	**Денна ставка, грн**	**Трудомісткість робочих днів**	**Сума основної зарплати, Грн**
Дипломник	39,26	314,08	22	6909,76
Керівник	64,00	512	1	512
Консультант по економічній частині	61,00	488	0,25	122
Консультант по охороні праці	61,00	488	0,25	122
Нормоконтроль	61,00	488	0,25	122
Всього (Зо)				7787,76

3) Витрати на додаткову заробітну плату визначаються у відсотках від основної. У наукових закладах додаткова заробітна плата складає 10-12% від основної заробітної плати.

$$Зд=10\%Зо;$$

$$Зд= \quad 7787,76*0,10 = 778,77 \text{ грн}$$

4)До складу собівартості НДР включаються податки, збори і інші обов'язкові платежі, встановлені системою оподаткування що діє. Відрахування до єдиного соціального внеску складає:

$$Зєсв=0,22*(Зо+Зд);$$

$$Зєсв=0,22*(7787,76+778,77) = 1884,63 \text{ грн.}$$

5) До накладних витрат відносять витрати на управління і господарське обслуговування, що відноситься до всіх виконуваних НДР.. У наукових закладах накладні витрати складають 40 -120% від основної і додаткової заробітної плати.

$$Рнакл= (З0+Зд)*0,4;$$

$$Рнакл= (7787,76+778,77)* 0,4 = 3426,61 \text{ грн.}$$

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому НДР за формою, приведеною в таблиці 3.4.

Таблиця 2.4. - Калькуляція планової собівартості

Статті витрат	**Сума, грн.**
1. Матеріали	165,00
2. Основна заробітна плата	7787,76
3. Додаткова заробітна плата	778,77
4. Відрахування до єдиного соціального внеску	1884,63
5. Накладні витрати	3426,61
Планова собівартість (Спл)	14042,77

Плановий прибуток визначений по формулі:

Ппл = 0,1*Спл = 0,1*14042,77= 1404,27 грн

Де 0,1 – норматив, який враховує граничний рівень рентабельності, встановлений чинним законодавством для науково-технічної продукції.

Договірна ціна визначається по формулі

Цнір = Спл + Ппл = 14042,77+1404,27 = 15447,04 грн.

Ціну реалізації встановлюємо з урахуванням ПДВ

ПДВ = 0,2*Цнір= 0,2*15447,04= 3089,41 грн.

Звідси ціна реалізації становить:

Цр = Цнір + ПДВ Цр = 15447,04+3089,41 = 18536,45 грн.

3 ОХОРОНА ПРАЦІ

3.1 Аналіз умов праці програміста

Охорона праці є дуже важливим поняттям для кожного працівника й професія програміста не є виключенням. Формування таких умов праці, за яких здоров`ю робітників нічого не буде загрожувати під час виконання свої обов`язків, є найголовнішим завданням будь-якого підприємства чи установи. На перший погляд може здаватись, що в професії програміста майже немає шкідливих факторів, а особливо, якщо порівнювати професію програміста з професіями важкої промисловості. Але, нажаль, програмісти також можуть відчувати на собі вплив багатьох шкідливих факторів та небезпечних факторів, якщо їх робочий процес не підконтрольний нормам охорони праці.

Охорона праці - це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності.

Дотримання норм охорони праці є спільним завданням як роботодавця, так і працівника. У вирішенні питань з охорони праці можна звернутися до законодавства України з охорони праці.

Під час робочого процесу програміст піддіється впливу великої кількості шкідливих та небезпечних факторів, а саме: шуми, вібрації, інфрачервоне випромінювання, електромагнітне випромінювання, електричний струм, емоційне та нервове навантаження, сидяче положення тіла протягом дового часу. Тому дуже важливо забезпечити правильний нормований графік та організувати робочий процес так, щоб мінімізувати вплив усіх перелічених раніше небезпечних та шкідливих факторів.

3.2 Заходи з охорони праці для забезпечення безпечного робочого процесу

Для забезпечення безпечного робочого процесу програміста, потрібно слідкувати за відповідністю нормам наступних параметрів :

- Робоче місце
- Освітлення
- Мікроклімат
- Шум і вібрація
- Пожежна безпека

Робоче місце - це частина простору, в якому програміст здійснює трудову діяльність, і проводить велику частину робочого часу. Правильно організоване робоче місце, повинно забезпечувати програмісту зручне положення при роботі, що, у свою чергу, дозволить збільшити продуктивність праці робітника. При правильній організації робочого місця продуктивність праці інженера зростає з 8 до 20 відсотків. Відповідно до ГОСТ 12.2.032-78 конструкція робочого місця і взаємне розташування всіх його елементів повинне відповідати антропометричним, фізичним і психологічним вимогам.

Головними елементами робочого місця програміста є письмовий стіл і крісло. Основним робочим положенням є положення сидячи. Робоче місце для виконання робіт у положенні сидячи організується відповідно до ГОСТ 12.2.032-78. Робоча поза сидячи викликає мінімальне стомлення програміста. Раціональне планування робочого місця передбачає чіткий порядок і сталість розміщення предметів, засобів праці і документації. Те, що потрібно для виконання робіт частіше, розташоване в зоні легкої досяжності робочого простору.

Стіл повинен мати відповідну для конкретної людини висоту, а його нижня частина повинна бути такої конструкції, щоб не було потрібно підбирати ноги. На поверхні стола не повинні з'являтися відблиски, які завадять нормально бачити інформацію на дисплеї. Оптимальні розміри столу

- це довжина 1300 мм і ширина 650 мм, а також висота 710 мм і глибина мінімум 400 мм.

Освітлення в робочому процесі програміста відіграє велику роль. Адже програмісти отримують велике навантаження на зоровий апарат, тому умови освітлення повинні зменшувати це навантаження максимально, наскільки це можливо.

Рівень освітлення повинен бути нормованим, адже недостатнє чи надмірне освітлення викликають напруження зорового апарату, роздратування та проблеми з концентацією.

Природне освітлення - освітлення приміщень світлом, що виходить від неба (прямим або відбитим), що проникає через світлові прорізи в зовнішніх огороджувальних конструкціях. Нормованої характеристикою такого освітлення є коефіцієнт природної освітленості.

Штучне освітлення – використовується, коли коефіцієнт природньої освітленості є недостатнім та в темний час доби. Для забезпечення приміщення штучним освітленням, застосовуються люмінісцентні лампи типу ЛБ, також допускається використання металогалогенних ламп потужністю 250 ВТ – для загального освітлення; ламп розжарювання у світильниках – для місцевого освітлення.

Значення освітленості на поверхні робочого столу в зоні розміщення документів має становити 300-500 лк. При цьому світильники місцевого освітлення слід встановлювати таким чином, щоб не створювати відблиски на поверхні екрана, а освітленість екрана не має перевищувати 300 лк.

Якщо вдень недостатнє природнє освітлення доповнюється штучним, то таке освітлення називатиметься суміщеним освітленням.

Мікрокліматичні умови виробничих приміщень характеризуються такими показниками:

- температура повітря,
- відносна вологість повітря,
- швидкість руху повітря,
- інтенсивність теплового (інфрачервоного) опромінення,
- температура поверхні.

Параметри мікроклімату нормуються для робочої зони – простору, обмеженого по висоті 2 м від рівня підлоги або майданчика, на якому знаходяться місця постійного або тимчасового перебування працівників.

Допустимі мікрокліматичні умови - поєднання параметрів мікроклімату, які при тривалому та систематичному впливі на людину можуть викликати зміни теплового стану організму, що швидко минають і нормалізуються та супроводжуються напруженням механізмів терморегуляції в межах фізіологічної адаптації. При цьому не виникає ушкоджень або порушень стану здоров'я, але можуть спостерігатися дискомфортні тепловідчуття, погіршення самопочуття та зниження працездатності.

Таблиця 3.1. - Оптимальні величини температури, відносної вологості та швидкості руху повітря в робочій зоні програміста

Пора року	Категорія робіт	Температура повітря. С	Відносна вологість повітря. %	Швидкість руху повітря, м/с
Холодна	Легка 1-а	22-24	60-40	0.1
	Легка 1-б	21-23	60-40	0.1
Тепла	Легка 1-а	23-35	60-40	0.1
	Легка 1-б	22-24	60-40	0.2

Інтенсивність теплового опромінення працюючих від нагрітих поверхонь технологічного устаткування, освітлювальних приладів не повинна перевищувати 35,0 Вт/м 2 - при опроміненні 50% та більше поверхні тіла, 70 Вт/м 2 - при величині опромінюваної поверхні від 25 до 50%, та 100 Вт/м 2 - при опроміненні не більше 25% поверхні тіла працюючого.

Формовані параметри мікроклімату на робочих місцях повинні бути досягнені, в першу чергу, за рахунок раціонального планування виробничих приміщень і оптимального розміщення в них устаткування з тепло-, холодо- та вологовиділеннями.

Для забезпечення оптимальних мікрокліматичних умов в офіснихь приміщеннях можуть застосовуватися такі технологічні засоби як: зволожквач повітря, кондиціонер, природня та примусова вентиляції.

Невід`ємною частиною роботи програміста є комп`ютер, що є джерелом вібрації та шуму. Генераторами шуму в комп`ютері є жорсткий диск; вентилятор блока живлення мережі; вентилятор, розташований на процесорі.

Таблиця 3.2. - Гранично допустимі рівні шуму на робочих місцях

Робочі місця	Рівні звукового тиску(дБ) в октавних смугах з середньогеометричними частотами								Еквіва лентні рівні звуку(дБА)
	63 Гц	125 Гц	250 Гц	500 Гц	1000 Гц	2000 Гц	4000 Гц	8000 Гц	
Приміщення конструкторських бюро,програмістів ЕОМ, лабораторій для теоретичних робіт	71	61	54	49	45	42	40	38	50

Заходи та засоби захисту від шуму поділяються на колективні та індивідуальні. Індивідуальна профілактика: введення в слухові проходи різних заглушок, вати; протишумові вкладиші типу «Беруші»; застосування навушників, шумозаглушувальні шоломи (застосовуються при рівнях звуку більше 130 дБА).

Тобто, засоби індивідуального захисту поділяють на:

- протишумові вкладиші (перекривають слуховий прохід усередені);

- протишумові навушники (закривають вушну раковину зовні);

До індивідуальних заходів відносять й лікувально-профілактичні – попередні та періодичні медичні огляди (аудіометричний контроль), використання раціональних режимів праці та відпочинку для

Організаційно-технічні засоби захисту від шуму передбачають: застосування малошумних технологічних процесів та устаткування, оснащення шумного устаткування засобами дистанційного керування, дотримання правил технічної експлуатації, проведення планово-попереджувальних оглядів та ремонтів. Це й: позначення робочих місць з рівнем звуку більше 80 дБА .

Пожежна безпека входить в комплекс заходів з охорони праці.

Первинні засоби пожежогасіння. Призначення та місцезнаходження первинних засобів пожежогасіння.

Будинки, споруди, приміщення, технологічні установки повинні бути забезпечені первинними засобами пожежогасіння: вогнегасниками, ящиками з піском, покривалами з негорючого теплоізоляційного полотна, грубововняної тканини чи повсті, іншим пожежним інструментом, які використовуються для локалізації і ліквідації пожеж у початковій стадії їхнього розвитку.

Коли від пожежі захищаються приміщення з персональними комп’ютерами, то слід ураховувати специфіку вогнегасних речовин у вогнегасниках, які призводять під час гасіння до псування обладнання. Ці

приміщення рекомендується оснащувати вуглекислотними вогнегасниками з урахуванням граничнодопустимої концентрації вогнегасної речовини. Для зазначення місцезнаходження первинних засобів пожежогасіння слід установлювати відповідні знаки згідно з чинними державними стандартами. Знаки слід розміщувати на видних місцях на висоті 2-2,5 м від рівня підлоги як у середині, так і поза приміщеннями (у разі потреби).

Переносні вогнегасники повинні розміщуватися шляхом:

1) навішування на вертикальні конструкції на висоті не більше 1,5 м від рівня підлоги до нижнього торця вогнегасника і на відстані від дверей, достатній для її повного відчинення;

2) установлення в пожежні шафи пожежних кранів, або у спеціальні тумби;

3) навішування вогнегасників на кронштейни, розміщення їх у тумбах або пожежних шафах повинне забезпечувати можливість прочитання маркувальних написів на корпусі.

Експлуатація та технічне обслуговування вогнегасників повинно здійснюватися відповідно до вимог Правил експлуатації вогнегасників (НАПБ Б.01.008-2004).та ін.

ВИСНОВКИ

Монолітна архітектура краща для розробки дуже маленьких, простих та легких програм. Так як монолітна архітектура вважається традиційним способом розробки додатків, завжди краще мати гарне знання про них. Монолітну архітектуру легше протестувати та вона є набагато дешевшою за мікросервісну як вплані проектування, так і в плані розробки, так як потребує меншого часу та ресурсів. Але з ростом програмного забезпечення стрімко зростає його складність. Тому для вирішення проблеми орієнтування у великомасштабному програмному забезпеченні, яке постійно оновлюється, застосовується мікросервісна архітектура.

Головними факторами при виборі для реалізації проекту саме мікросервісної архітектури є частота оновлень програмного забезпечення та його складність.

Мікросервісна архітектура вимагає від розробника гарного володіння технологіями контейнеризації, вміння розбиття великої системи на добре продумані та здатні до самостійного існування частини.

Мікросервіси можуть підходити не всім. Якщо застаріла монолітна програма працює без нарікань, її руйнація може не коштувати зусиль. Проте архітектура мікросервісів може бути корисною у міру зростання організації та підвищення вимог до додатків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Newman S. Building Microservices: Designing Fine-Grained Systems 1st Edition / Sam Newman. – Newton: O'Reilly Media, 2015. – 262 с.

2. Рідчарсон К. Мікросервіси. Патерни розробки і рефакторінга. / Кріс Рідчарсон. – Санкт-Петербург: Пітер Прес, 2019. – 544 с. – (Бібліотека програміста).

3. Ньюмен С. Від моноліту до мікросервісів /Сем Ньюмен. – Санкт-Петербург: БХВ-Петербург, 2021. – 272 с.

4. Мікрослужби .NET: Архітектура контейнерних програм .NET [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/dotnet/architecture/microservices/

5. Що таке ASP.NET: [Электронный ресурс]. Режим доступу до ресурсу: http://www.internet-technologies.ru/articles/lekciya-1-chto-takoe-asp-net-installyaciya-i-testovyy-proekt.html

6. Порівняння монолітної та мікросервісної архітектури [Електронний ресурс] – Режим доступу до ресурсу: https://proglib.io/p/monolitnaya-vs-mikroservisnaya-arhitektura-2019-09-16.

7. Плюси та мінуси використання мікросервісної архітектури [Електронний ресурс] – Режим доступу до ресурсу: https://simpleone.ru/blog/primenenie-mikroservisnoj-arhitektury-plyusy-minusy-podvodnye-kamni/.

8. Мікросервісна архітектура від А до Я [Електронний ресурс] – Режим доступу до ресурсу: https://axiomjdk.ru/announcements/2021/12/02/microservices-101/.

9. Бойчик І. М. Економіка підприємства : навчальний посібник для студентів економічних спеціальностей вищих навчальних закладів I-IV рівнів акредитації. Третє видання, випр. і доп. / І. М. Бойчик, П. С. Харів., М. І. Холчан, Ю. В. Піча. – К. : Каравела, 2016. – 328 с.

10. Закон України про охорону праці [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12#Text.