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

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Розробка програмного забезпечення»

Група: 4РП-06

Дипломний проект

здобувача освіти денної форми навчання
РП.06.19.000.ДП

САВІНОВА ВЛАДИСЛАВА СЕРГІЙОВИЧА

м. Одеса
2023 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

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

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

Група: **4РП-06**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи) на тему:

Проектування мікросервісної архітектури для веб-сайту ВСП «ОТФК ОНТУ»

Проектний матеріал складається з пояснювальної записки на __69__ сторінках та графічного (презентаційного) матеріалу на __10__ аркушах (слайдах).

Дипломник _____ (Савінов В.С.)

Керівник _____ (Шувалова І.О.)

Консультанти:

з економічної частини _____ (Копайгородська Т.Г.)

з охорони праці _____ (Чорновол Н.І.)

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

старший консультант _____ (Кунуп Т.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Скорнякова О.В.)

Захист «23» 06 2023 р. Протокол ДКК № 2

Оцінка ДДК 5 (відмінно)

Секретар ДДК _____

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

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

Спеціальність **121 «Інженерія програмного забезпечення»**

Освітня програма **«Розробка програмного забезпечення»**

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР _____

Беркань І.В.

"_____" _____ 2023 р.

ЗАВДАННЯ

на дипломний проект (роботу)

Здобувачеві (здобувачці) освіти Савінова Владислава Сергійовича

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Проектування мікросервісної архітектури для веб-сайту коледжа

затверджена наказом по коледжу від "17" жовтня 2023 р. № 235-А2-ОД

2. Термін здачі закінченого проекту (роботи) 09.06.2023

3. Вихідні данні до проекту (роботи)

1 Проектування мікросервісної архітектури

2.Використання Figma для побудови візуальних схем архітектури.

3.Проектування механізму аутентифікації та авторизації в мікросервісній архітектурі веб-сайту коледжа з використанням OAuth

4.Використання Docker для контейнеризації мікросервісів веб-сайту коледжа.

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

Опис предметної області. Огляд існуючих рішень, або аналогів. Загальний технічний опис вирішення задачі дипломного проекту. Огляд інструментів та програмних рішень для виконання дипломного проекту. Проектування мікросервісної частини сайту коледжу.

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

Титульний ліст, Огляд проєкту, Плюси обраної архітектури, Написання клієнтскої частини на React.js + Next.js, Головна сторінка платформи, Написання мікросервісів на .NET та Python, Основні функціональні можливості, Переваги веб-додатку для коледжу, Ризики та виклики, Підсумок

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Технологічний розділ	Шувалова І.О.		
Економічна частина	Копайгородська Т.Г.		
Охорона праці	Чорновол В.І.		
Нормоконтроль	Петрашова В.І.		

7. Дата видачі завдання 01.05.2023

Керівник Шувалова І.О. _____
(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.	*Вступ*	*15.05.2023*	Виконав
2.	*Опис предметної області*	*16.05.2023*	Виконав
3.	*Огляд існуючих рішень, або аналогів*	*17.05.2023*	Виконав
4.	*Загальний технічний опис вирішення задачі*	*23.05.2023*	Виконав
	дипломного проекту		
5.	*Огляд інструментів та програмних рішень*	*25.05.2023*	Виконав
	для виконання дипломного проекту		
6.	*Проектування мікросервісної частини*	*26.05.2023*	Виконав
	сайту коледжу.		
7.	*Робота над Економічною частиною*	*04.06.2023*	Виконав
8.	*Робота над охороною праці*	*06.06.2023*	Виконав
9.	*Підготовка матеріалів для захисту*	*07.06.2023*	Виконав

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл	Примітка
				Документація		
			РП.06. 19 000.00 ДП	*Дипломний проект*		
А4			*РП.06. 19 000. 00 ДП ПЗ*	*Пояснювальна записка*	*1*	

Ізм.	Арк.	№ докум.	Підпис	Дата	*РП 06.19.000.00 ДП ПЗ*		
Розробив		*Савінов В.С*			*Проектування мікросервісної архітектури для веб-сайту ВСП «ОТФК ОНТУ»*	Літ.	Аркуш 4 / Аркушів 69
Перевірив		*Шувалова І.О.*				*Н Д П*	
Н. Контр.		*Петрашова В. І.*				*ВСП «ОТФК ОНТУ» гр. 4РП-06*	
Затверд.		*Кривченко Ю. В.*					

ЗМІСТ

ВСТУП

У сучасному світі, де швидкість та ефективність є ключовими факторами, мікросервісна архітектура стає все більш популярною у розробці веб-додатків. Однією з важливих галузей, де ця архітектура може бути застосована з вигодою, є освітній сектор, зокрема сайти коледжів.

Проектування мікросервісної архітектури для сайту коледжу відкриває безліч можливостей для поліпшення якості освітнього процесу та забезпечення зручності для студентів, викладачів та адміністраторів. Замість монолітної системи, де всі функції об'єднані в одному моноліті, мікросервісна архітектура дозволяє розбити функціональність на невеликі, незалежні компоненти - мікросервіси.

Це підхід має кілька переваг. По-перше, розбиття системи на мікросервіси спрощує розробку, підтримку та масштабування сайту коледжу. Кожен мікросервіс може бути розроблений та підтримуваний окремо, що дозволяє розвивати його незалежно від інших компонентів системи. По-друге, мікросервіси можуть бути розгорнуті на різних серверах або контейнерах, що дозволяє гнучко розподіляти навантаження та забезпечувати стабільну роботу сайту. Крім того, мікросервісна архітектура дозволяє швидко впроваджувати нові функції та зміни без необхідності змінювати весь код сайту. Це забезпечує швидкий та ефективний розвиток сайту коледжу, а також дає змогу змінювати функції окремих компонентів без впливу на решту системи.

У підсумку, проектування мікросервісної архітектури для сайту коледжу є важливим кроком у напрямку поліпшення освітнього процесу та забезпечення зручності для всіх учасників. Цей підхід дозволяє забезпечити гнучкість, масштабованість та швидкість розвитку системи, що стає невід'ємною частиною сучасного веб-розвитку.

1 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

1.1 Опис предметної області.

Мікросервісна архітектура - це сучасний підхід до розробки програмного забезпечення, який структурує додаток як набір слабко пов'язаних, незалежних і масштабованих сервісів. Кожен сервіс, відомий як мікросервіс, відповідає за певну бізнес-можливість і взаємодіє з іншими мікросервісами через чітко визначені API. Цей опис надає огляд мікросервісної архітектури та підкреслює її переваги над традиційною монолітною архітектурою.

Мікросервісна архітектура розбиває складні додатки на менші незалежні сервіси, які можна розробляти, розгортати і масштабувати незалежно. Кожен мікросервіс є самодостатньою сутністю з власною кодовою базою, базою даних та механізмом розгортання. Архітектура сприяє децентралізованому та модульному підходу до розробки програмного забезпечення, дозволяючи командам автономно працювати над окремими сервісами.

Переваги мікросервісної архітектури:

Мікросервісна архітектура забезпечує горизонтальне масштабування, коли окремі мікросервіси можуть бути масштабовані незалежно в залежності від попиту. Така гнучкість дозволяє організаціям ефективно розподіляти ресурси, уникаючи необхідності масштабування всього додатку. Це підвищує продуктивність, оскільки додаткові ресурси потрібні лише для конкретних сервісів, які відчувають підвищений трафік або навантаження.

Розбиваючи додаток на менші, керовані сервіси, мікросервісна архітектура покращує модульність. Кожен мікросервіс можна розробляти, тестувати та розгортати незалежно, що дозволяє пришвидшити цикл розробки. Така модульна структура спрощує обслуговування та оновлення, оскільки зміни в одному мікросервісі не впливають на інші, зменшуючи ризик збоїв у всій системі.

У монолітній архітектурі збій в одному компоненті може вивести з ладу всю систему. На противагу цьому, мікросервісна архітектура сприяє ізоляції

несправностей. Якщо мікросервіс виходить з ладу, решта програми може продовжувати функціонувати незалежно. Така ізоляція підвищує загальну стійкість системи, оскільки збої локалізуються і не поширюються на інші сервіси.

Архітектура мікросервісів підтримує використання різних технологій і фреймворків для кожного мікросервісу. Така гнучкість дозволяє командам обирати найбільш підходящі інструменти для своїх конкретних сервісів, використовуючи переваги найкращих доступних технологій. Це усуває потребу в єдиному технологічному стеку, що сприяє інноваціям та впровадженню нових технологій, не впливаючи на весь додаток.

Архітектура мікросервісів добре узгоджується з принципами методологій DevOps та Agile. Оскільки кожен мікросервіс має власну команду, відповідальну за його розробку та підтримку, це сприяє автономії та надає командам можливість працювати незалежно. Це розпаралелює зусилля з розробки, забезпечуючи швидшу доставку функцій та масштабованість у великих організаціях-розробниках.

Мікросервісна архітектура полегшує безперервне розгортання та тестування. З меншими сервісами стає простіше тестувати і розгортати зміни, не впливаючи на весь додаток. Для кожного мікросервісу можна налаштувати автоматизовані конвеєри тестування і розгортання, що забезпечує швидку ітерацію і скорочує час, необхідний для випуску нових функцій.

Docker став трансформаційною технологією у сфері розробки та розгортання програмного забезпечення, особливо в контексті архітектури мікросервісів. Цей вступний розділ містить огляд Docker, його призначення і того, як він покращує процес розгортання додатків на основі мікросервісів. Простими словами, Docker можна порівняти з віртуальним транспортним контейнером, який інкапсулює додаток і всі його необхідні компоненти, роблячи його портативним, надійним і простим у розгортанні.

Docker - це платформа з відкритим вихідним кодом, яка дозволяє розробникам упаковувати додатки та їх залежності в ізольовані та легкі

контейнери. Ці контейнери забезпечують узгоджене та відтворюване середовище виконання, що гарантує безперебійну роботу додатків на різних системах та платформах. Docker використовує технологію контейнеризації, яка дозволяє ізолювати додатки від базової інфраструктури, що забезпечує безперешкодне розгортання та масштабування.

Архітектура мікросервісів - це підхід до розробки програмного забезпечення, який декомпозує додатки на менші, незалежно розгорнуті сервіси. Docker відіграє ключову роль у полегшенні розгортання та управління додатками на основі мікросервісів. Кожен мікросервіс може бути інкапсульований у власний контейнер Docker, забезпечуючи модульне та ізольоване середовище для розробки, тестування та розгортання. Ці контейнери можуть взаємодіяти один з одним за допомогою чітко визначених API, забезпечуючи безперешкодну інтеграцію та співпрацю між мікросервісами.

Використання Docker в архітектурі мікросервісів має кілька ключових переваг:

Спрощене розгортання: Docker спрощує процес розгортання, забезпечуючи узгоджене середовище для різних систем. Розробники можуть упакувати додаток та його залежності в контейнер, гарантуючи, що він буде працювати стабільно незалежно від базової інфраструктури.

Масштабованість та гнучкість: Docker дозволяє легко масштабувати мікросервіси. Кожен мікросервіс можна індивідуально масштабувати, створюючи або закриваючи контейнери за потреби. Така гнучкість дозволяє ефективно використовувати ресурси, забезпечуючи оптимальну продуктивність і швидкість реагування.

Ізоляція та безпека: Контейнери Docker забезпечують високий рівень ізоляції, запобігаючи втручанню додатків один в одного. Така ізоляція підвищує безпеку, мінімізуючи вплив потенційних вразливостей і зменшуючи поверхню атаки.

Відтворюваність: Docker забезпечує відтворюваність середовищ додатків, гарантуючи узгоджену поведінку на етапах розробки, тестування та

виробництва. Розробники можуть визначити точні залежності та конфігурації, необхідні для програми, зменшуючи проблеми сумісності та покращуючи співпрацю.

Безперервна інтеграція та розгортання (CI/CD): Docker легко інтегрується з конвеєрами CI/CD, полегшуючи автоматизоване тестування та процеси розгортання. З Docker розробники можуть легко пакувати та розповсюджувати додатки, забезпечуючи швидку та надійну доставку нових функцій та оновлень. Як працює віртуалізаця у Docker можно побачити на рисунку 1.1

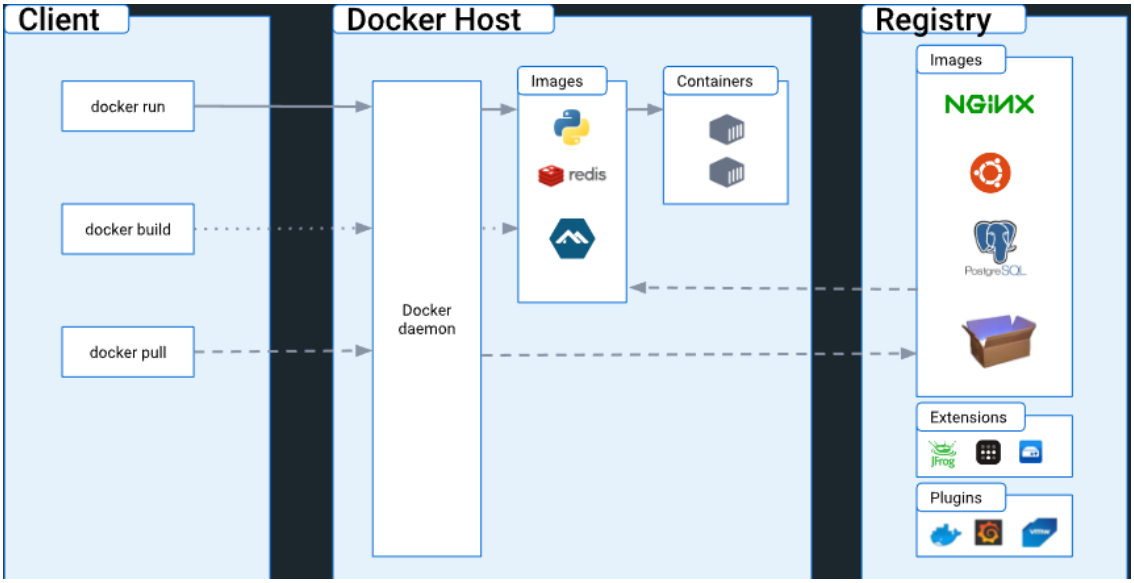


Рисунок 1.1 Віртуалізація у Docker

У сфері розробки програмного забезпечення мікросервісна архітектура набула величезної популярності завдяки своїй здатності розділяти складні додатки на менші, слабко пов'язані між собою сервіси. Кожен сервіс працює незалежно, взаємодіючи з іншими через чітко визначені API. Хоча такий підхід забезпечує гнучкість і масштабованість, він створює нові виклики в управлінні та координації численних контейнерів, які інкапсулюють ці сервіси. Kubernetes покликаний вирішити ці проблеми та спростити розгортання та керування контейнерами.

Контейнери Docker надають легке та ізольоване середовище для пакування та запуску додатків з їхніми залежностями. Вони дозволяють

розробникам створювати узгоджені середовища, які можна легко відтворити на різних системах.

Kubernetes є потужним інструментом оркестрування, що дозволяє автоматизувати розгортання, масштабування та керування контейнерними програмами. Він забезпечує масштабовану та відмовостійку інфраструктуру для запуску та координації контейнерів на кластері машин.

Docker спрощує пакування та розповсюдження додатків, інкапсулюючи їх у контейнери. Kubernetes використовує контейнери Docker як основну одиницю розгортання, забезпечуючи узгодженість та переносимість у різних середовищах.

Kubernetes організовує розгортання та керування контейнерами Docker, виконуючи такі завдання, як планування контейнерів, масштабування, балансування навантаження та моніторинг стану. Він підтримує бажаний стан додатків і автоматично налаштовує ресурси відповідно до попиту.

Kubernetes дозволяє легко масштабувати мікросервіси, динамічно виділяючи ресурси та автоматично розподіляючи робоче навантаження між контейнерами. Це гарантує, що додатки можуть обробляти зростаючий трафік без шкоди для продуктивності.

Постійно відстежуючи стан контейнерів та автоматично перезапускаючи несправні екземпляри, Kubernetes підвищує надійність мікросервісів. Вона може відновлюватися після збоїв, забезпечуючи мінімальний час простою та підвищуючи загальну стійкість системи.

Kubernetes надає вбудовані механізми виявлення сервісів, які дозволяють мікросервісам легко знаходити та спілкуватися один з одним. Це полегшує створення мережевих правил і балансувальників навантаження, забезпечуючи ефективну маршрутизацію трафіку та оптимізовану комунікацію.

З Kubernetes розробники можуть визначити бажаний стан системи за допомогою декларативних конфігураційних файлів. Потім Kubernetes автоматично керує інфраструктурою відповідно до вказаного стану, зменшуючи ручне втручання та забезпечуючи ефективну автоматизацію.

Підсумовуючи, Kubernetes є життєво важливим компонентом у сфері архітектури мікросервісів. Завдяки плавній інтеграції з контейнерами Docker, вона спрощує оркестровку контейнерів, покращує масштабованість, забезпечує високу доступність та автоматизує багато завдань управління. Організації, які впроваджують Kubernetes, можуть використовувати його можливості для створення стійких і масштабованих додатків на основі мікросервісів, сприяючи інноваціям та ефективності в технологічному ландшафті, що постійно змінюється. Приклад розгорнутої архітектури з використанням kubernetes як оркестратора можно побачити на рисунку 1.2.

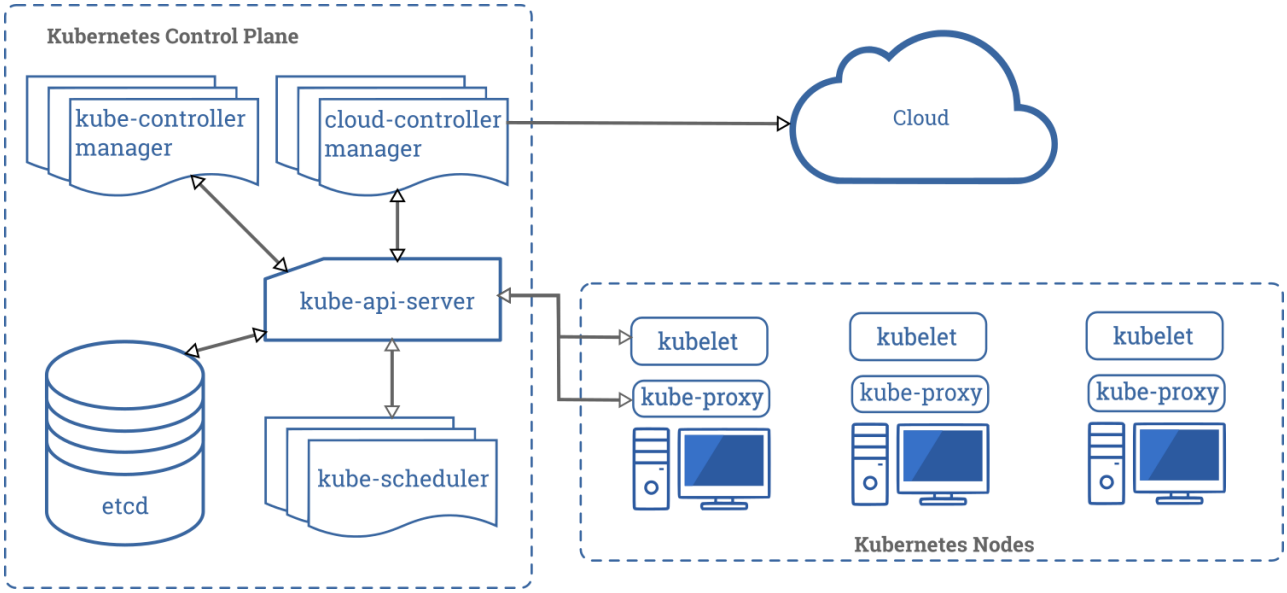


Рисунок 1.2 Розгорнута архітектура з використанням kubernetes як оркестратора

Nginx, що є потужним веб-сервером і зворотним проксі-сервером, який відіграє вирішальну роль в архітектурі мікросервісів. Його можна було б описати як фундаментальний компонент, який забезпечує безперебійну комунікацію між різними мікросервісами в системі.

Простими словами, мікросервісна архітектура - це підхід до побудови програмних систем шляхом розбиття їх на менші, незалежні сервіси, які можна розробляти, розгортати і масштабувати індивідуально. Кожен мікросервіс виконує певне завдання і взаємодіє з іншими мікросервісами для забезпечення повної функціональності системи.

Nginx виступає в ролі посередника, обробляючи вхідні запити від клієнтів і направляючи їх до відповідного мікросервісу на основі заздалегідь

визначених правил. Уявіть собі дорожнього поліцейського на жвавому перехресті, який стежить за тим, щоб кожен транспортний засіб (запит) був направлений до правильного пункту призначення (мікросервісу). Цей процес відомий як зворотне проксіювання.

Однією з основних переваг використання Nginx в мікросервісній архітектурі є його здатність ефективно розподіляти вхідні запити між декількома мікросервісами. Це допомагає збалансувати навантаження і запобігти перевантаженню будь-якого окремого мікросервісу трафіком. Розумно розподіляючи запити, Nginx забезпечує оптимальну продуктивність і доступність всієї системи.

Ще однією перевагою Nginx є його надійність і масштабованість. Він призначений для обробки великої кількості одночасних з'єднань без шкоди для продуктивності. Це особливо важливо в мікросервісних архітектурах, де численні сервіси повинні взаємодіяти одночасно. Легка архітектура Nginx і підхід, заснований на подіях, роблять його високоефективним, дозволяючи йому обробляти тисячі одночасних з'єднань з мінімальним використанням ресурсів.

Крім того, Nginx надає розширені можливості, такі як кешування, завершення SSL/TLS і маршрутизація запитів на основі різних критеріїв, таких як шаблони URL-адрес або значення заголовків. Ці функції підвищують безпеку, продуктивність і гнучкість архітектури мікросервісів. Наприклад, Nginx може виступати в ролі термінатора SSL/TLS, обробляючи шифрування і дешифрування захищених з'єднань, знімаючи навантаження з мікросервісів.

Таким чином, Nginx слугує важливим компонентом в архітектурі мікросервісів, забезпечуючи ефективний зв'язок між мікросервісами, балансуючи навантаження та підвищуючи продуктивність і масштабованість системи. Виконуючи роль зворотного проксі-сервера, Nginx оптимізує потік запитів і забезпечує безперебійну роботу мікросервісів, сприяючи загальному успіху архітектури. Побачити використання NGINX в якості контролера контейнерів Docker можно побачити на рисунку 1.3.

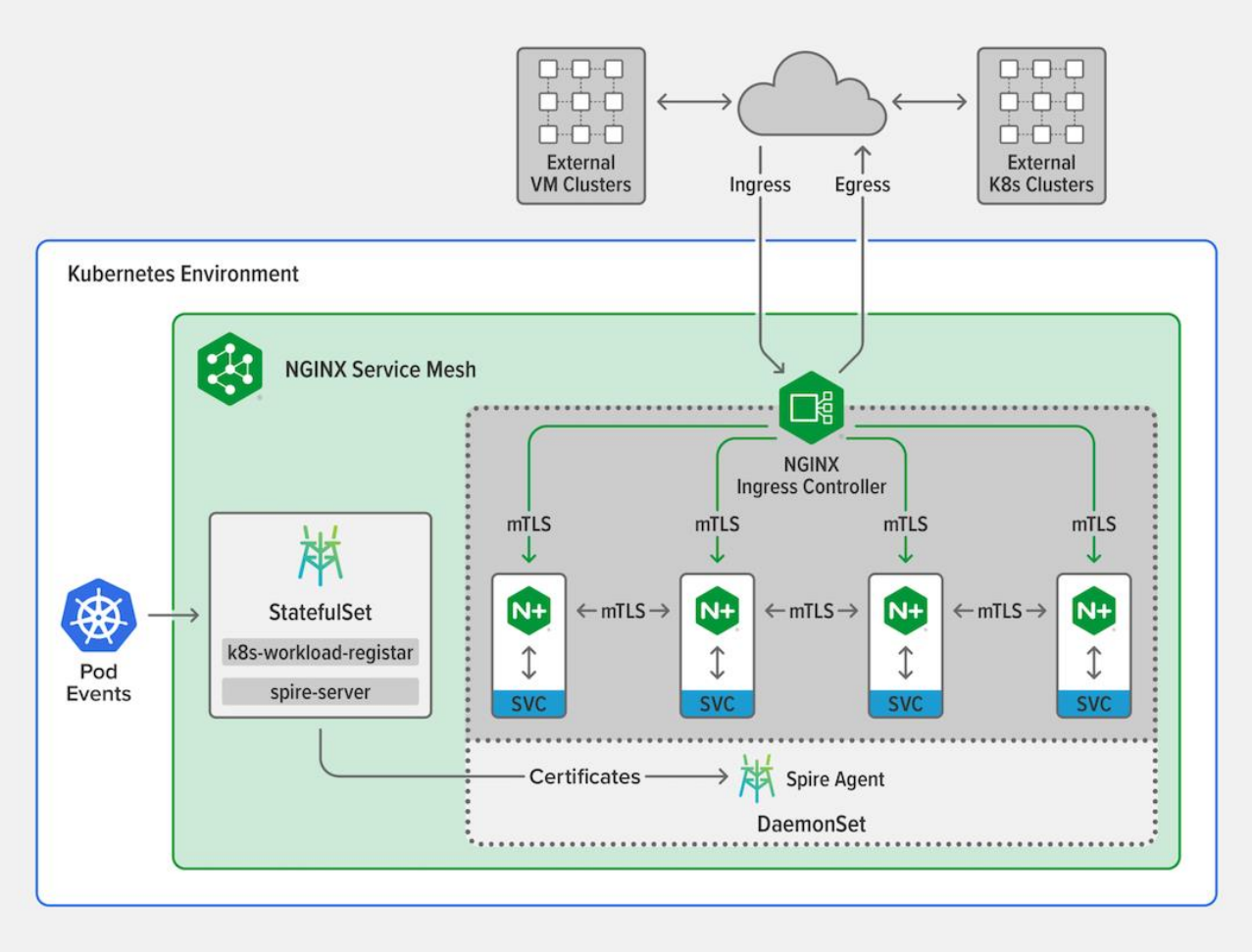


Рисунок 1.3 Використання NGINX в якості контролера контейнерів Docker

Таким чином, проанізувавши дані технології, можна зробити висновок, що цей проект може бути використаний будь-яким учбовим закладом. Це пов’язано з тим, що при його розробці було застосовано мікросервісну архітектуру, один із важливійших плюсів якої є висока гнучкість відносно до різноманітних задач.

1.2 Огляд існуючих рішень для проектування мікросервісної архітектури для вебсайту

Існує велика кількість готових патернів для веб-сайтів, що використовують мікросервісну архітектуру. Під час розроблення дипломного проєкту вибір складався з 9 патернів:

1. Паттерн реєстру служб

Цей паттерн забезпечує центральне сховище для виявлення мікросервісів за іменами. Це шаблон архітектури мікросервісів, який дозволяє сервісам знаходити інші мікросервіси та взаємодіяти один з одним.

У моделі реєстру сервісів використовується центральний реєстр або каталог сервісів для ведення обліку доступних сервісів та їх відповідного розташування. Мікросервіси мають можливість реєструватися в реєстрі, що дозволяє іншим мікросервісам знаходити їх і підключатися до них.

Наприклад, розглянемо великий веб-сайт електронної комерції, що містить різні мікросервіси, такі як управління замовленнями, обробка платежів, доставка та підтримка клієнтів. Кожен з цих сервісів має власний REST API для зв'язку.

Щоб спростити пошук сервісів, можна використати шаблон реєстру сервісів. Реєстр сервісів, таких як Consul або Eureka (надається Spring Cloud), створюється для зберігання повного списку доступних сервісів та їх кінцевих точок.

Коли сервіс запускається, він може зареєструватися в реєстрі, вказавши своє ім'я та кінцеву точку. Наприклад, сервіс замовлення зареєструється як "order-service" з кінцевою адресою "http://order-service:8080". Інші сервіси, які потребують взаємодії зі службою замовлень, можуть шукати в реєстрі її ім'я, щоб отримати відповідну кінцеву адресу.

Наприклад, платіжна служба може зробити запит до реєстру, щоб отримати кінцеву точку "order-service" для надсилання платіжної інформації. Аналогічно, служба доставки може отримати кінцеву точку "замовлення-послуга" з реєстру, щоб отримати доступ до деталей доставки замовлення.

Такий підхід дозволяє розробляти і розгортати кожну службу незалежно без жорсткого кодування кінцевих точок інших служб. Паттерн Service Registry Pattern забезпечує динамічне виявлення сервісів, підвищуючи гнучкість системи та її стійкість до змін. На рисунку 1.4 можна побачити приклад схематичної реалізації цього паттерну

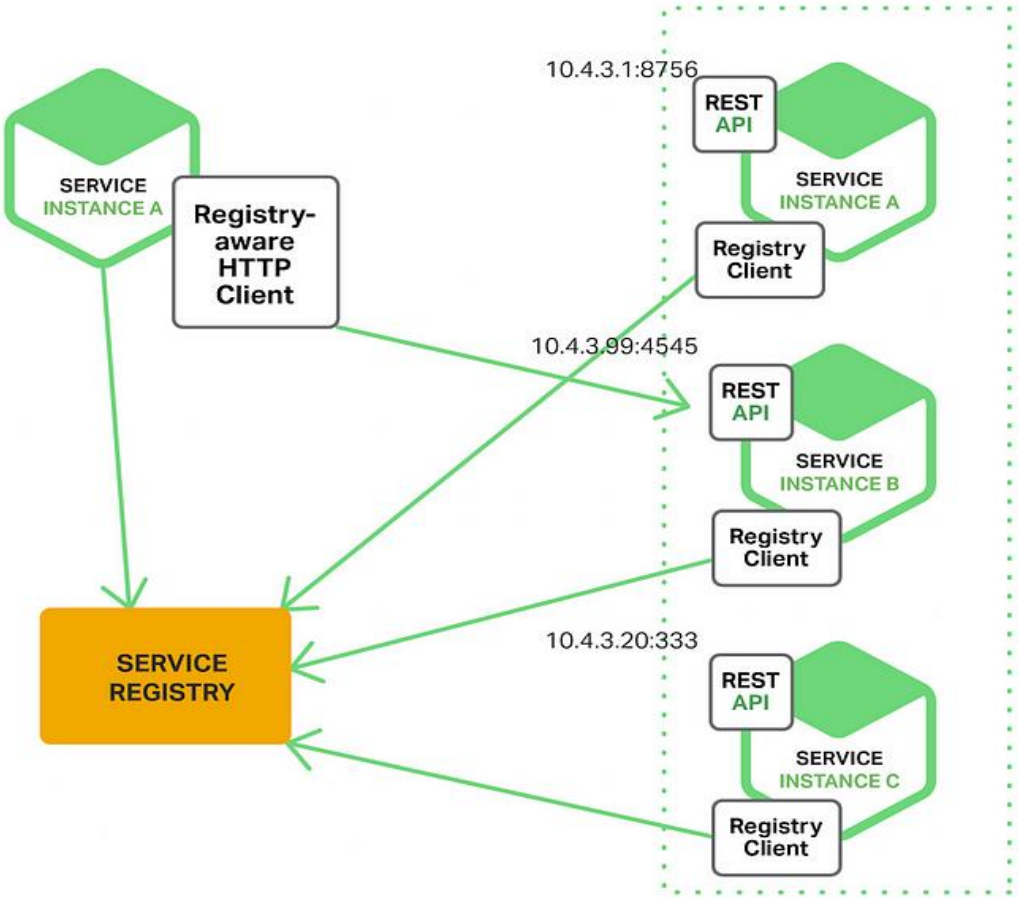


Рисунок 1.4 Схема паттерну реєстру служб

2. Паттерн автоматичного вимикача

Шаблон "Автоматичний вимикач" призначений для запобігання поширенню збоїв і забезпечення безперервності роботи додатків у разі одного або декількох збоїв в обслуговуванні. Вона особливо корисна в архітектурі мікросервісів для ефективної обробки несправностей.

Шаблон обертається навколо компонента автоматичного вимикача, який виступає в якості захисту між клієнтом і сервісом. Його основна функція - захистити клієнта від потенційних збоїв у роботі сервісу, з яким він взаємодіє. Відстежуючи стан сервісу, автоматичний вимикач може вжити заходів, якщо виявить постійні збої. Для цього він розмикає ланцюг, припиняючи надсилання будь-яких подальших запитів до сервісу, доки він не відновиться.

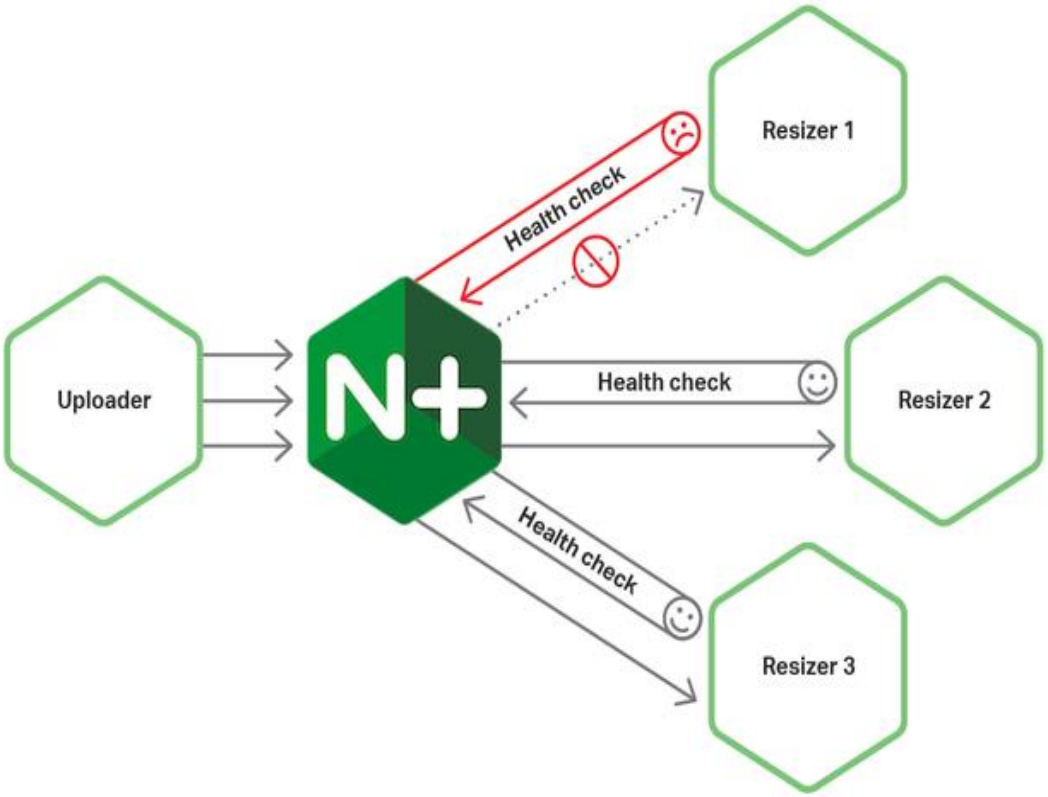


Рисунок 1.5 Схема мікросервісного додатку

На рисунку 1.5 розглянено мікросервісний додаток, що покладається на зовнішній сервіс, який, як відомо, є ненадійним. Незважаючи на непередбачуваність зовнішнього сервісу, додаток повинен продовжувати свою роботу, не будучи повністю залежним від нього.

У цьому сценарії патерн "Автоматичний вимикач" виявляється корисним, оскільки він може визначити, коли зовнішній сервіс стає недоступним. Потім він може швидко переключитися на альтернативний або резервний сервіс, забезпечуючи безперервну функціональність, поки зовнішній сервіс не буде відновлено.

Впровадження патерну Circuit Breaker в архітектуру мікросервісів можна здійснити за допомогою спеціалізованих інструментів, таких як Hystrix від Netflix або Spring Cloud Circuit Breaker. Ці інструменти полегшують управління поведінкою "вимикача" і надають додатку засоби для контрольованого реагування на збої в роботі сервісу.

3. Шаблон API-шлюзу

Шаблон шлюзу API - це широко використовуваний патерн проектування в архітектурі мікросервісів, який передбачає використання шлюзу API як точки входу для вхідних запитів API. Він служить централізованим інтерфейсом для всіх мікросервісів і діє як посередник між клієнтами та мікросервісами, ефективно перенаправляючи запити до відповідного сервісу.

Основна мета шлюзу API - відокремити клієнтів від мікросервісів шляхом надання спрощеного та узгодженого API, таким чином захищаючи клієнтів від складності основної системи. Це усуває необхідність для клієнтів безпосередньо взаємодіяти з окремими мікросервісами, забезпечуючи більш раціональний підхід. Крім того, це позбавляє клієнтів необхідності запам'ятовувати адреси численних мікросервісів REST API та керувати ними.

Крім того, шлюз API пропонує додатковий рівень безпеки та управління. Організації можуть контролювати доступ до сервісів, керувати і контролювати продуктивність системи, а також впроваджувати політики для всіх сервісів.

Щоб проілюструвати роботу шаблону API-шлюзу, розглянемо просту систему електронної комерції з декількома мікросервісами, що виконують різні функції, такі як управління замовленнями, каталог товарів і автентифікація користувачів. Кожен мікросервіс має власну кінцеву точку API для обробки запитів. Однак клієнти, якими можуть бути веб або мобільні додатки, потребують єдиної точки входу для доступу до всіх цих мікросервісів.

Тут API-шлюз відіграє вирішальну роль. Він працює як зворотний проксі-сервер, приймаючи вхідні запити від клієнтів і згодом спрямовуючи кожен запит до відповідного мікросервісу на основі запитуваної кінцевої точки.

Наприклад, якщо запит зроблено до кінцевої точки /orders, API-шлюз перенаправить його до мікросервісу управління замовленнями. Аналогічно, запити до кінцевої точки /products будуть перенаправлені до мікросервісу каталогу продуктів. Приклад проектування можна побачити на рисунку 1.6

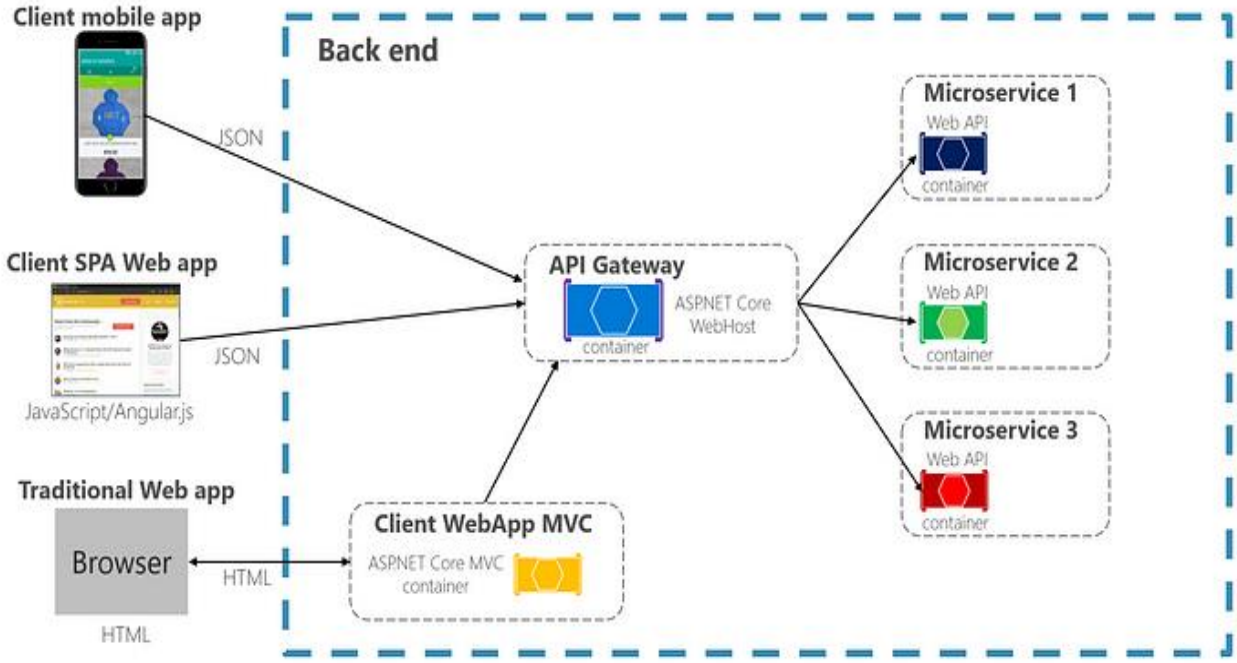


Рисунок 1.6 Схема API-шлюзу

Крім того, шаблон API шлюзу охоплює різні додаткові функції, включаючи перетворення запитів і відповідей, обмеження швидкості, автентифікацію та авторизацію, а також кешування.

Крім того, він полегшує створення уніфікованого API, який приховує тонкощі базових мікросервісів і пропонує клієнтам спрощений і послідовний інтерфейс.

Таким чином, патерн API Gateway забезпечує масштабований, адаптований і безпечний підхід до управління мікросервісами в рамках складної системи. Він спрощує розробку, розгортання та обслуговування додатків на основі мікросервісів.

4. Паттерн "Сага"

Паттерн "Сага" використовується для обробки транзакцій за участю декількох мікросервісів, забезпечуючи успішне завершення серії транзакцій між цими сервісами. У випадку невдачі шаблон дозволяє відкотити або скасувати всі зміни, зроблені до цього моменту.

Паттерн складається з послідовності локальних транзакцій, кожна з яких оновлює стан окремого сервісу, разом з відповідними компенсуючими

транзакціями, що використовуються для скасування наслідків початкових транзакцій у разі збою.

Щоб проілюструвати патерн "Сага" в дії, розглянемо додаток для електронної комерції, заснований на мікросервісах. Припустимо, що в ньому задіяні два мікросервіси: один відповідає за обробку замовлень, а інший - за доставку замовлень.

Коли розміщується нове замовлення, сервіс обробки замовлень підтверджує замовлення і перевіряє наявність товару, тоді як сервіс доставки займається пакуванням і доставкою клієнту. Під час цього процесу вступає в дію шаблон "Сага".

Після перевірки дійсності замовлення та наявності товару служба обробки замовлень надсилає повідомлення службі доставки, щоб розпочати процес доставки.

Після цього служба доставки ініціює транзакцію з пакування та відправлення замовлення. Якщо транзакція проходить успішно, замовлення позначається як відправлене.

Однак, якщо транзакція не вдається (можливо, через проблеми з постачальником послуг доставки), служба доставки запускає компенсаційну транзакцію, щоб скасувати наслідки початкової транзакції. Це може включати такі дії, як скасування відправлення та поповнення запасів товарів для підтримання узгодженості.

Використовуючи патерн "Сага", додаток може ефективно керувати потоком транзакцій між декількома мікросервісами, забезпечуючи цілісність даних і надаючи механізм для скасування або компенсації невдалих транзакцій. Приклад візуалізації цього патерну можна побачити на малюнку 1.7

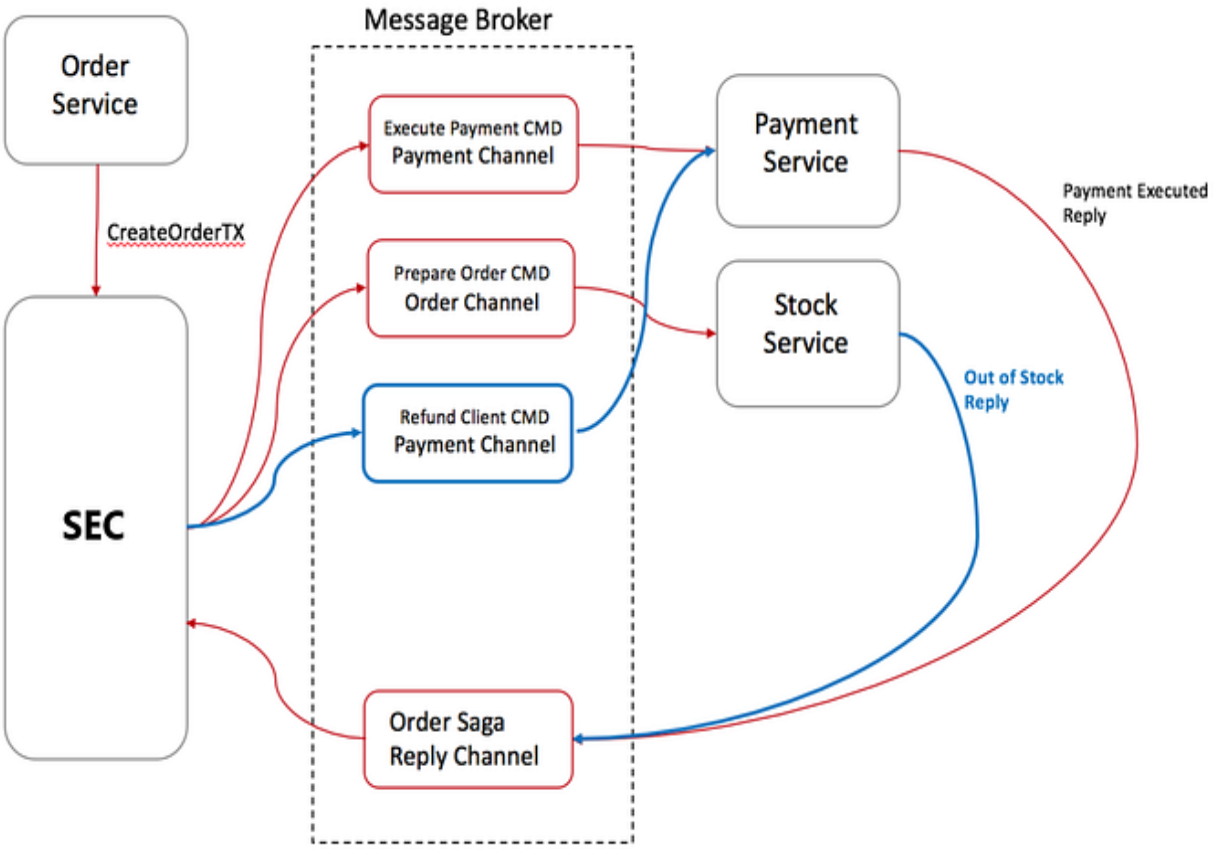


Рисунок 1.7 Схема "Сага" паттерну

Окрім служби доставки, служба обробки замовлень також використовує патерн Saga для обробки власних транзакцій в рамках мікросервісної архітектури. Після того, як служба доставки підтверджує успішне відправлення замовлення, сервіс обробки замовлень оновлює статус замовлення як завершеного.

Однак, якщо служба доставки повідомляє про збій, служба обробки замовлень запускає компенсуючу транзакцію, щоб скасувати замовлення та ініціювати відшкодування будь-яких здійснених платежів.

Таким чином, патерн "Сага" є важливим механізмом для управління складними транзакціями між декількома мікросервісами, забезпечуючи узгодженість і надійність протягом усього процесу. З огляду на його величезну корисність в мікросервісних додатках, розуміння і вивчення патерну "Сага" може виявитися дуже корисним.

5. Шаблон пошуку постачальників подій

Event Sourcing - це мікросервісний патерн, який використовується для збереження даних і запитів у додатку. На відміну від традиційних підходів, які зберігають поточний стан об'єкта, Event Sourcing записує і зберігає всі події, які відбуваються в додатку. Це дозволяє реконструювати стан об'єкта в будь-який момент часу.

У цьому патерні кожна модифікація в додатку фіксується як подія і зберігається у вигляді журналу подій. Повний стан програми можна відновити, відтворивши ці події. В результаті, Event Sourcing забезпечує журнал аудиту всіх змін, зроблених в додатку.

Для ілюстрації розглянемо на рисунку 1.8 додаток для електронної комерції. Щоразу, коли користувач розміщує замовлення, генерується подія OrderPlaced, яка зберігається в журналі подій. Аналогічно, коли замовлення відправляється, створюється і зберігається в журналі подія ShipmentMade.

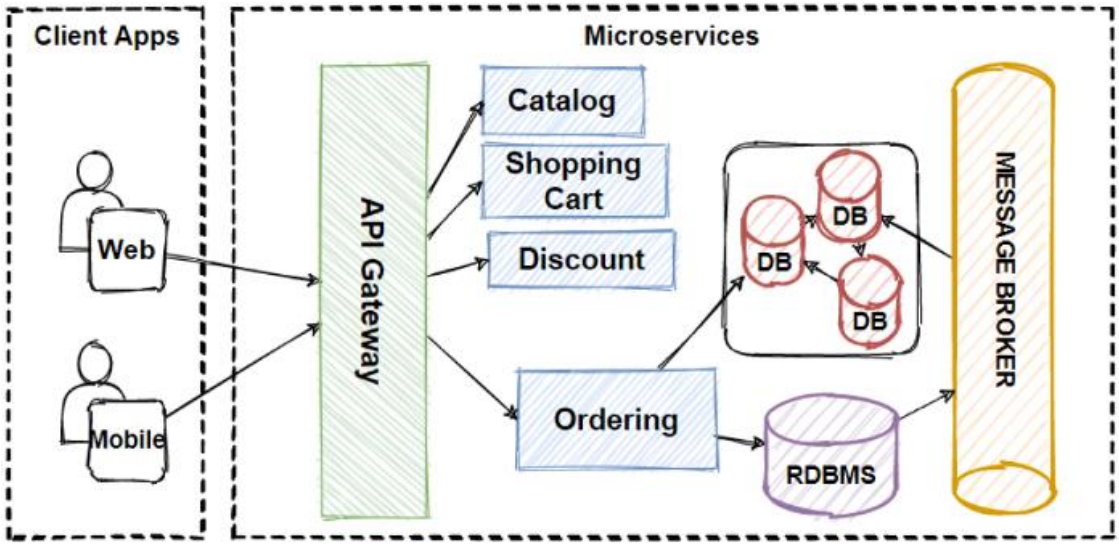


Рисунок 1.8 Схема додатку для електронної комерції

Якщо замовлення згодом скасовується, генерується подія OrderCanceled, яка також зберігається в журналі. Відтворюючи ці події, можна визначити поточний стан замовлення.

Використовуючи Event Sourcing, додаток зберігає історичний запис подій, що дозволяє реконструювати стани об'єктів і забезпечує аудиторський слід усіх змін. Цей патерн виявляється особливо корисним у сценаріях, де здатність аналізувати і розуміти минулі події має вирішальне значення.

Event Sourcing дійсно пропонує кілька переваг в рамках архітектури мікросервісів:

1) Можливість аудиту: Можливість відстежувати та перевіряти всі зміни, внесені в систему, забезпечує прозорість та підзвітність.

2) Масштабованість: Обробка подій може бути розпаралелена, що дозволяє покращити масштабованість і здатність обробляти високі навантаження.

3) Гнучкість: Оскільки події слугують джерелом істини, стає легше змінювати спосіб запиту та збереження даних, не змінюючи самі дані, що лежать в їх основі. Це забезпечує гнучкість в адаптації до мінливих бізнес-вимог.

4) Відмовостійкість: Незмінність подій забезпечує цілісність і відмовостійкість даних. Після того, як подія записана, її неможливо змінити, що знижує ризик пошкодження даних.

Однак важливо враховувати складність і потенційні проблеми, пов'язані з впровадженням Event Sourcing:

1) Складність: Джерела подій вносять додаткову складність в архітектуру, оскільки події потрібно фіксувати, зберігати та відтворювати для відновлення станів об'єктів. Це вимагає ретельного планування та реалізації.

2) Продуктивність запитів: Отримання даних через відтворення подій може бути повільнішим порівняно з традиційними методами запитів. Він передбачає відтворення всіх подій для відновлення потрібного стану, що може вплинути на продуктивність. Як наслідок, слід ретельно зважити, чи переважують переваги пошуку за подіями потенційний вплив на продуктивність.

3) Альтернативні рішення: Подієвий сорсинг не завжди є найкращим рішенням для кожного випадку використання. Залежно від вимог і обмежень системи, можуть існувати альтернативні рішення, які надають аналогічні переваги, але мають простіші механізми реалізації та запитів.

Дуже важливо оцінити компроміси і врахувати конкретні потреби програми, перш ніж прийняти рішення про використання Event Sourcing. Хоча

він пропонує значні переваги, він не може бути оптимальним вибором для кожного сценарію.

6. Шаблон розподілу відповідальності за командні запити (CQRS)

Шаблон розподілу відповідальності за команди та запити (Command Query Responsibility Segregation, CQRS) - це широко розповсюджений шаблон проектування мікросервісів, який розділяє команди (операції запису) та запити (операції читання) на окремі моделі, кожна з яких має власну виділену базу даних.

Фундаментальна концепція CQRS полягає в тому, що моделі, які використовуються для запису даних, відрізняються від моделей, які використовуються для читання даних.

У цій моделі командна модель отримує команди від клієнтів і виконує операції запису до бази даних. З іншого боку, модель запитів відповідає за читання даних з бази даних і відправлення їх назад клієнтам. Розділивши операції запису і читання, кожну модель можна оптимізувати незалежно, щоб підвищити продуктивність і масштабованість системи.

Для ілюстрації розглянемо додаток для електронної комерції, який спочатку використовує традиційний підхід на основі CRUD для управління інформацією про товари. Цей підхід використовує єдину модель і базу даних як для читання, так і для запису даних про товари. Однак у міру зростання додатку модель стає дедалі складнішою, а база даних може стати вузьким місцем у продуктивності.

Прийнявши CQRS, додаток запровадить окремі моделі для запису та зчитування інформації про продукт. Командна модель буде зосереджена на ефективних операціях запису, призначених для обробки великих обсягів модифікацій даних. І навпаки, модель запитів надаватиме пріоритет швидким і оптимізованим операціям зчитування, пристосованим для швидкого надання результатів запиту. Приклад такого проектування з реальним продуктом на рисунку 1.9

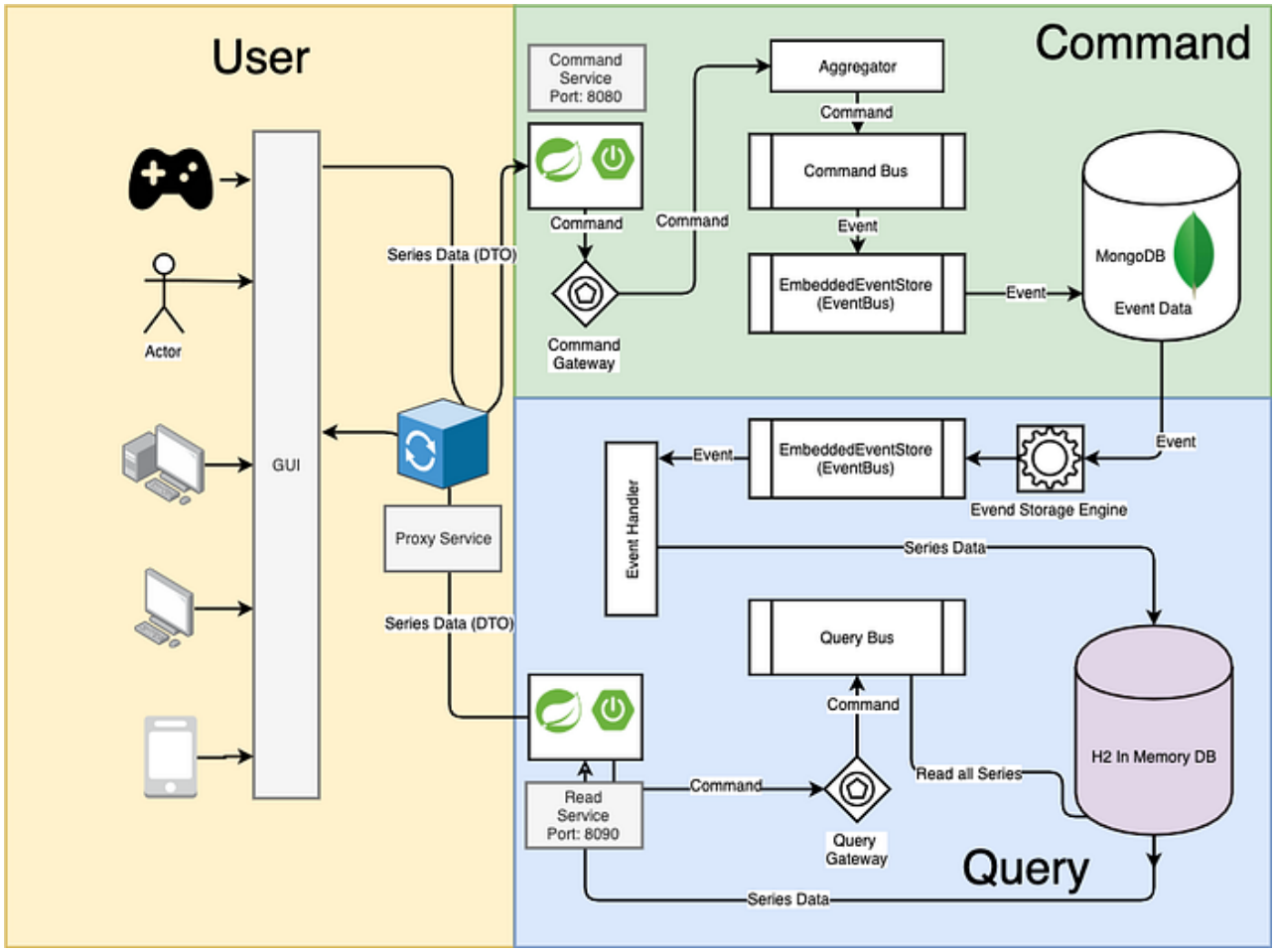


Рисунок 1.9 Схема CQRS паттерну

Командна модель використовує базу даних, оптимізовану для запису, в той час як модель запитів використовує базу даних, оптимізовану для читання. Зв'язок між цими двома моделями здійснюватиметься за допомогою шини подій або черги повідомлень, що забезпечить синхронізацію і узгодженість даних.

Впроваджуючи CQRS, додаток може досягти кращої продуктивності, масштабованості та ремонтопридатності, використовуючи окремі моделі, пристосовані для конкретних завдань, тим самим долаючи обмеження і вузькі місця, пов'язані з уніфікованим підходом, заснованим на CRUD.

Загалом, CQRS може покращити масштабованість та продуктивність системи, а також спростити кодову базу, розділивши проблеми. Однак вона також може додати складності та вимагати більше зусиль для розробки, оскільки потрібні окремі моделі та бази даних.

7. Шаблон перегородки

Шаблон дизайну перегородки дійсно схожий на шаблон автоматичного вимикача у своїй меті - запобігти каскадним збоям у системі. Однак, в той час як патерн "вимикач" фокусується на розриві ланцюга і захисті клієнта від збоїв в конкретному сервісі, патерн "перегородка" робить акцент на ізоляції різних частин системи, щоб запобігти поширенню збоїв по всій системі.

В мікросервісній архітектурі модель перегородки може бути реалізована шляхом ізоляції різних мікросервісів один від одного, гарантуючи, що збій в одному мікросервісі не матиме згубного впливу на інші мікросервіси. Такої ізоляції можна досягти різними способами, наприклад, використовуючи окремі пули з'єднань, пули потоків або навіть окремі фізичні ресурси для кожного мікросервісу.

Наприклад, розглянемо сценарій з трьома мікросервісами: Сервіс А, Сервіс В і Сервіс С. Кожен мікросервіс має власний виділений пул з'єднань, який не використовується спільно з іншими. Це означає, що якщо в сервісі А виникне збій, який призведе до вичерпання його пулу з'єднань, це не вплине на пули з'єднань сервісів В і С. Така ізоляція запобігає каскадному поширенню збою і впливу на доступність і продуктивність системи в цілому.

Завдяки використанню перегородки система стає більш стійкою та відмовостійкою. Збої в одній частині системи локалізуються, дозволяючи іншим частинам продовжувати функціонувати незалежно. Така ізоляція допомагає підтримувати стабільність і надійність системи в цілому, навіть за наявності збоїв в окремих мікросервісах. Схема паттерна на рисунку 1.10.

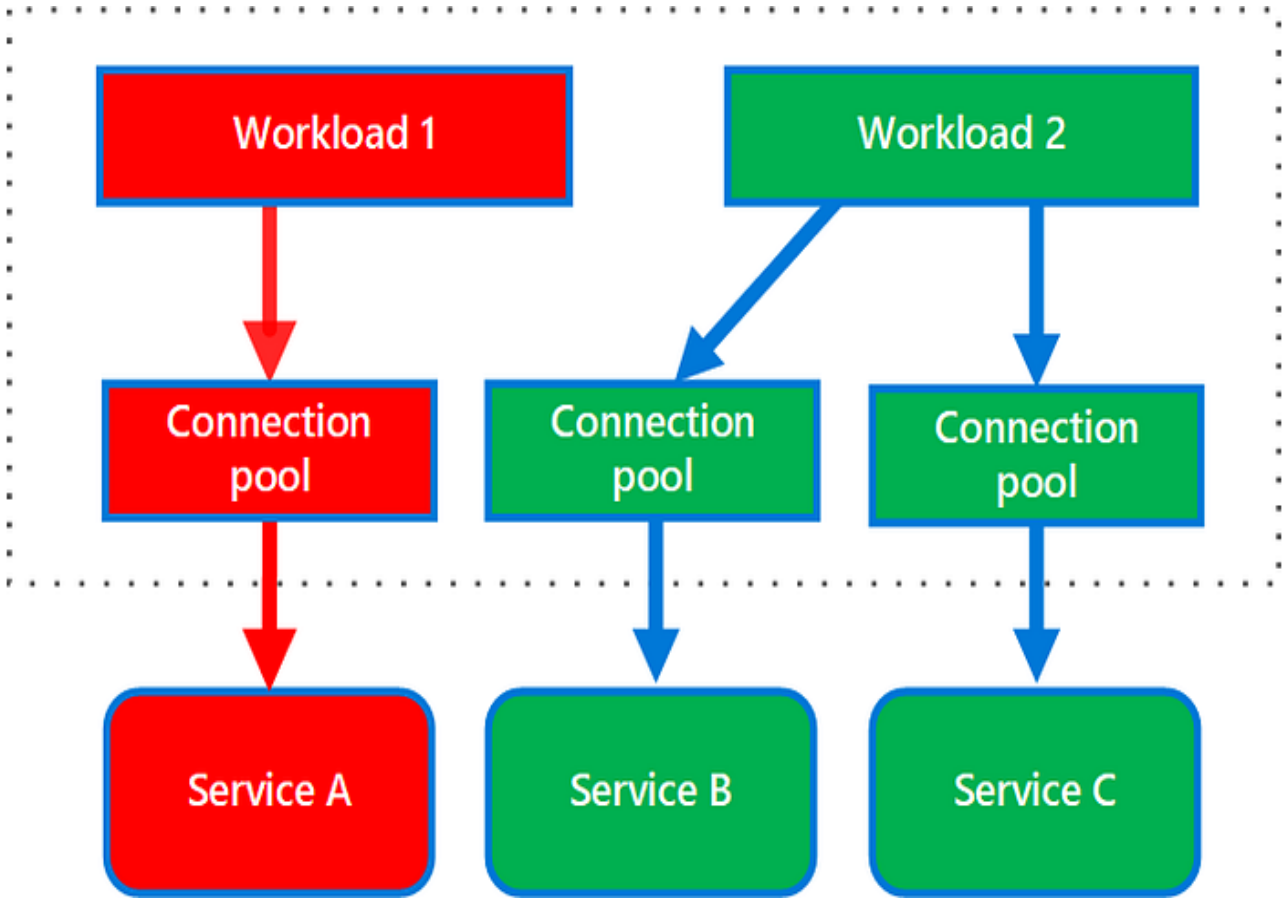


Рисунок 1.10 Схема паттерну Перегородки

8. Паттерн Бекенди для фронтенду (BFF)

Патерн проектування Backends for Frontends (BFF) зазвичай використовується в мікросервісній архітектурі для вирішення складнощів клієнт-серверної комунікації за наявності декількох користувацьких інтерфейсів. Він пропонує створення спеціальних бекенд-сервісів для кожного фронтенду, щоб задовольнити специфічні вимоги цього інтерфейсу.

Впроваджуючи окремі бекенд-сервіси для кожного фронтенду, розробники можуть оптимізувати потік даних, стратегії кешування та механізми автентифікації відповідно до унікальних потреб кожного фронтенду. Такий підхід сприяє модульності та відокремленню бекенд-сервісів, дозволяючи їм розвиватися незалежно.

Щоб проілюструвати це, давайте розглянемо сценарій, коли у вас є і веб-додаток, і мобільний додаток, яким потрібен доступ до одного і того ж набору сервісів. За допомогою патерну BFF ви створите окремі бекенд-сервіси,

адаптовані для кожного додатка, з урахуванням особливостей відповідних платформ.

Для веб-додатків бекенд-сервіс можна оптимізувати для ефективної обробки великих обсягів даних, що дозволить швидше завантажувати і відтворювати контент. З іншого боку, бекенд-сервіс для мобільного додатку може надавати пріоритет меншим затримкам і використанню мережі, оптимізуючи передачу даних для забезпечення безперебійної роботи користувачів на мобільних пристроях.

Використовуючи патерн BFF, ви гарантуєте, що бекенд-сервіси тісно пов'язані з конкретними вимогами та обмеженнями кожного фронтенд-додатку. Такий підхід забезпечує більшу гнучкість і оптимізацію продуктивності, а також можливість розвивати і вдосконалювати кожен фронтенд незалежно, не впливаючи на інші фронтенд-додатки.

Загалом, патерн BFF забезпечує засоби для оптимізації клієнт-серверної взаємодії в мікросервісній архітектурі, дозволяючи розробникам адаптувати бекенд-сервіси до унікальних потреб кожного фронтенду, що призводить до покращення продуктивності, масштабованості та зручності для користувачів.

Цей патерн дозволяє командам оптимізувати користувацький досвід для кожного інтерфейсу, використовуючи окремі бекенд-сервіси для кожного з них. Це також дозволяє уникнути використання одного бекенд-сервісу, який має обслуговувати багато різних інтерфейсів з різними потребами, що може стати дедалі складнішим і важчим у підтримці. Схему роботи цього паттерну можна побачити на рисунку 1.11

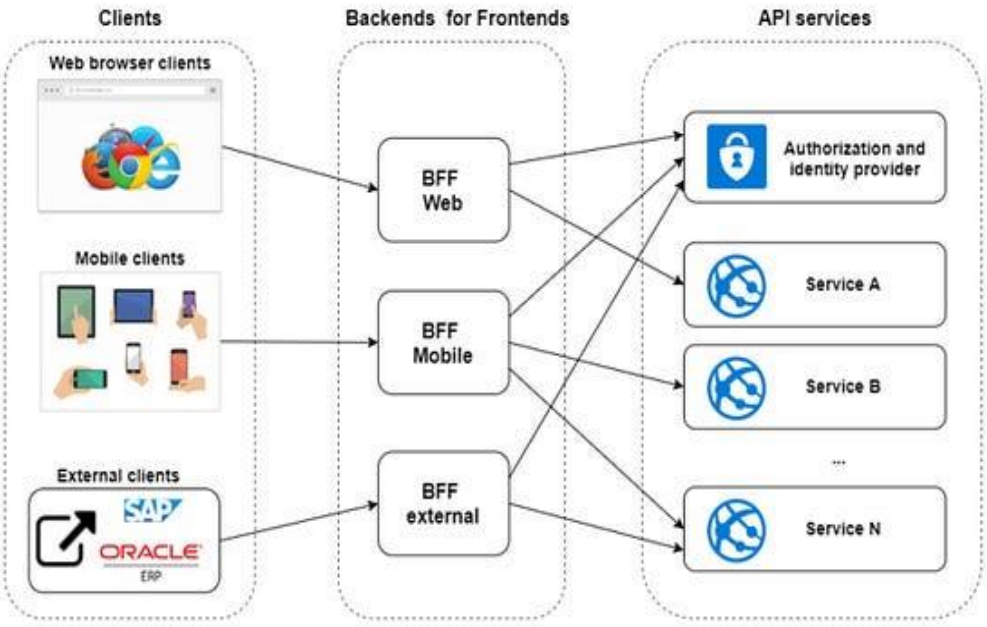


Рисунок 1.11 Схема BFF паттерну

9. Посольський шаблон

Для забезпечення відмовостійкості хмарних додатків певні функції, такі як розрив ланцюга, маршрутизація, вимірювання та моніторинг, а також оновлення мережевої конфігурації, мають вирішальне значення. Однак оновлення застарілих додатків або існуючих бібліотек коду для включення цих функцій може бути складним або навіть неможливим, особливо якщо код більше не підтримується або не може бути легко модифікований командою розробників.

Крім того, налаштування мережевих викликів для з'єднання, автентифікації та авторизації може бути складним, особливо коли ці виклики повинні використовуватися кількома додатками, створеними на різних мовах і фреймворках. Централізоване управління мережевими функціями та функціями безпеки спеціальною командою в організації може ще більше ускладнити ситуацію, оскільки оновлення коду додатків, з яким вони не знайомі, може становити ризики.

Для вирішення цих проблем рекомендується використовувати проксі-сервер, зовнішній по відношенню до програми, який діє як посередник між програмою та зовнішніми сервісами. Цей проксі може бути розгорнутий на тому

ж хост-середовищі, що і додаток, що дозволяє контролювати маршрутизацію, відмовостійкість і функції безпеки, уникаючи при цьому будь-яких обмежень доступу, пов'язаних з хостом. Шаблон посла можна використовувати для стандартизації та розширення інструментарію.

Вивантаживши клієнтські фреймворки та бібліотеки на зовнішній проксі, ви можете керувати цими функціями незалежно від самої програми. Проксі можна оновлювати та модифікувати, не порушуючи функціональність програми. Крім того, окремі команди, що спеціалізуються на безпеці, роботі з мережею або автентифікації, можуть впроваджувати і підтримувати ці функції в проксі, надаючи вузькоспеціалізовану експертизу.

Послуга амбасадора може бути розгорнута як допоміжний модуль поряд з додатком або службою, що споживає її, забезпечуючи їх спільний життєвий цикл. Крім того, якщо амбасадор використовується декількома окремими процесами на одному хості, його можна розгорнути як демон або службу Windows. У випадку контейнерних сервісів, амбасадор має бути створений як окремий контейнер на тому ж хості, з відповідним налаштуванням каналів зв'язку.

Таким чином, використання зовнішнього проксі, такого як посол, дозволяє керувати критично важливими функціями незалежно від програми. Це полегшує контроль над маршрутизацією, відмовостійкістю і безпекою, дозволяючи спеціалізованим командам займатися конкретними аспектами. Розгортання амбасадора як додаткового модуля або окремого контейнера забезпечує безперешкодну інтеграцію з життєвим циклом програми та надає необхідну гнучкість для оновлень і модифікацій.

Використання зовнішнього проксі, такого як посол, має декілька переваг. По-перше, це дозволяє керувати критично важливими функціями незалежно від програми. Такий підхід спрощує контроль над маршрутизацією, відмовостійкістю і безпекою, оскільки спеціалізовані команди можуть зосередитися на конкретних аспектах.

По-друге, розгортання амбасадора як додаткового модуля або окремого контейнера забезпечує гнучкість та безперешкодну інтеграцію з життєвим циклом програми. Це означає, що його можна легко оновлювати і модифікувати відповідно до змінних потреб. Такий підхід дозволяє ефективно використовувати ресурси та забезпечує масштабованість системи. Загалом, використання зовнішнього проксі, наприклад посла, є ефективним способом забезпечення надійності, безпеки та гнучкості управління критичними функціями програми. Побачити реалізацію цього шаблону можна на рисунку 1.12

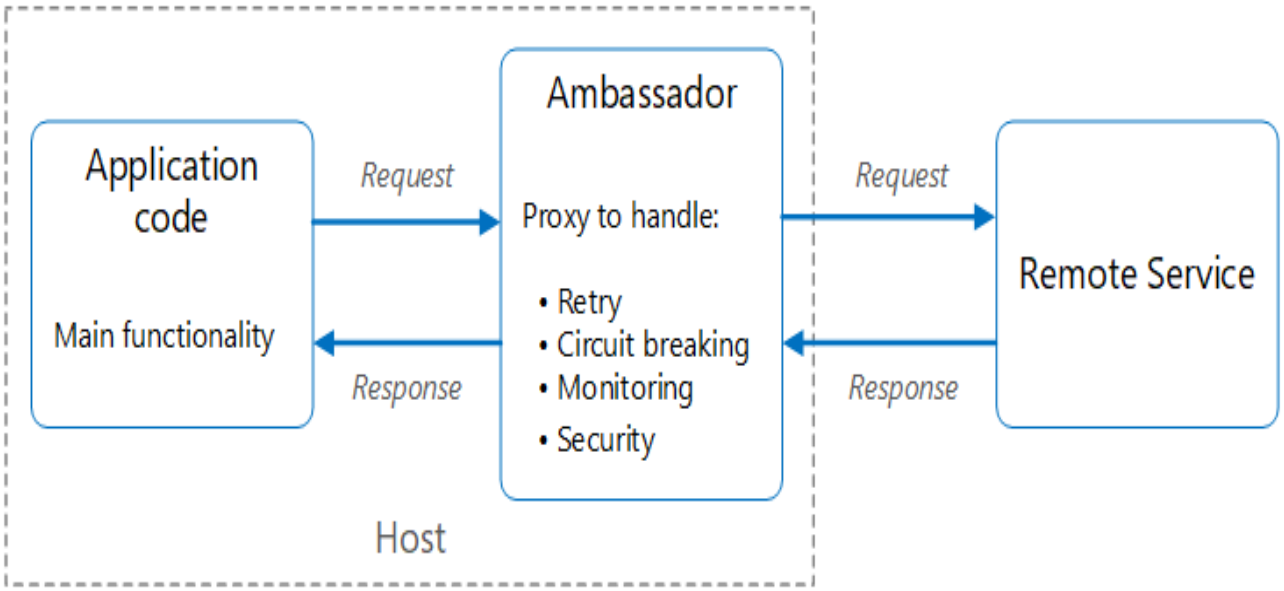


Рисунок 1.12 Схема посольського(Ambassador) шаблону

У цій роботі було прийнято рішення використовувати API-шлюз-шаблон з метою забезпечення простоти та відмовостійкості системи. Вибір цього шаблону базувався на тому, що він є найбільш простим і надійним варіантом для нашого випадку.

Використання API-шлюзу дозволяє забезпечити швидкість та простоту проектування системи. Це дозволяє зручно розробляти та підтримувати систему, а також забезпечувати її стабільну роботу навіть при великому обсязі навантажень. API-шлюз-шаблон забезпечує уніфікований доступ до різних служб і ресурсів, що спрощує розробку, впровадження та масштабування системи.

Цей вибір дозволяє забезпечити ефективну та надійну роботу системи, знижуючи складність розробки і підтримки. API-шлюз-шаблон є надійним рішенням, яке допоможе забезпечити стійкість та стабільність системи навіть у навантажених умовах.

1.3 Загальний технічний опис вирішення задачі дипломного проекту

Ядром будь-якої мікросервісної архітектури є контейнеризація. В результаті для цієї задачі було обрано Docker. Docker - це платформа з відкритим вихідним кодом, яка дозволяє розробникам автоматизувати розгортання та управління додатками в ізольованих контейнерах. Контейнеризація в Docker працює шляхом пакування додатку та його залежностей в стандартизовану одиницю, яка називається контейнер. Ці контейнери інкапсулюють код програми, час виконання, системні інструменти, бібліотеки та конфігурації, необхідні для запуску програми. Кожен контейнер працює як ізольований процес, забезпечуючи узгодженість і відтворюваність середовища на різних машинах.

Процес починається зі створення Docker-файлу - текстового файлу, який містить набір інструкцій для створення образу Docker. Docker-файл визначає базовий образ, додає програмний код і залежності, налаштовує змінні оточення, відкриває мережеві порти і визначає команди для запуску всередині контейнера. Кожна інструкція у докер-файлі являє собою шар образу.

Після створення докер-файлу для створення образу Docker використовується інтерфейс командного рядка Docker CLI (Command Line Interface). Виконується команда docker build, в якій вказується шлях до каталогу, що містить Dockerfile. Потім Docker зчитує інструкції з Dockerfile і виконує їх послідовно, створюючи образ шар за шаром. Кожна інструкція у докер-файлі створює новий проміжний контейнер, а отримані зміни файлової системи фіксуються у вигляді нового шару образу.

Образи Docker будуються за допомогою багаторівневої файлової системи. Кожен шар представляє набір змін файлової системи. Коли інструкція у Docker-файлі змінює файлову систему, новий шар додається поверх попередніх шарів. Шари є незмінними, тобто їх не можна змінити після створення. Цей механізм шарів дозволяє ефективно обмінюватися зображеннями, оскільки різні зображення зі спільними шарами можуть повторно використовувати ці шари, зменшуючи простір для зберігання та пришвидшуючи час збірки.

Після того, як образ Docker створено, його можна перенести до реєстру Docker. Docker Hub є популярним загальнодоступним реєстром, але ви також можете використовувати приватні реєстри. Команда docker push використовується для завантаження образу до реєстру, пов'язуючи його зі сховищем і, за бажанням, позначаючи номером версії або міткою. Перенесення образу до реєстру робить його доступним для завантаження і використання іншими користувачами.

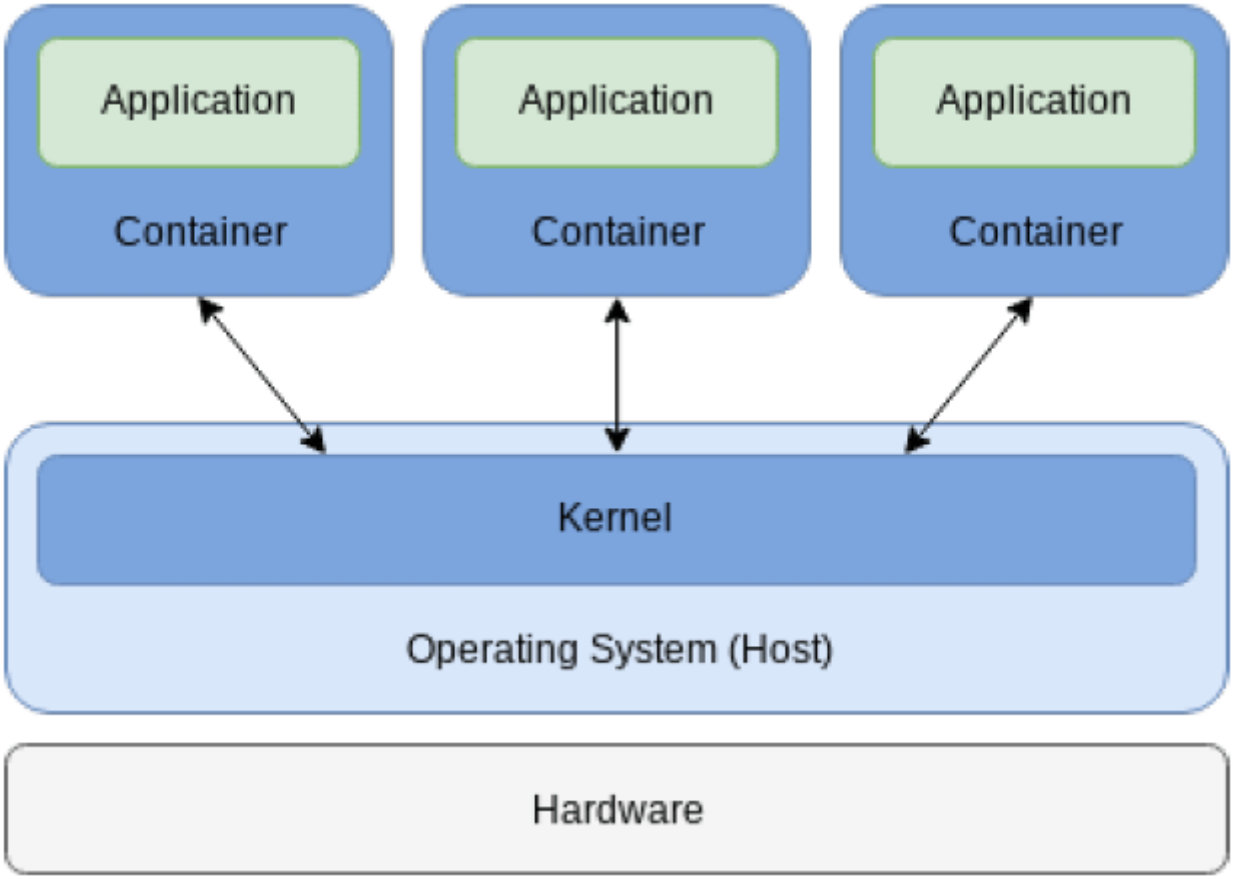


Рисунок 1.13 Схема взаємодії операційної системи та контейнерів

Далі, важливою частиною проектування та реалізації мікросервісної архітектури є правильний розподіл та використання Docker контейнерів в Kubernetes. Цей процес називається Оркестрацією.

Під оркестровкою контейнерів мається на увазі процес автоматизації управління контейнерами, включаючи розгортання, масштабування, мережу та балансування навантаження, серед інших завдань. Kubernetes спрощує оркестровку контейнерів, пропонуючи декларативний підхід до визначення бажаного стану програми. В основі Kubernetes лежить концепція кластера. Кластер - це набір фізичних або віртуальних машин, які називаються вузлами, що працюють разом для запуску контейнерних додатків. Ці вузли організовані у дві основні категорії: вузли площини керування та робочі вузли.

Площина керування складається з декількох компонентів, які керують загальним станом кластера та оркестровкою. Ключові компоненти включають в себе:

1. Сервер API, котрий надає доступ до API Kubernetes, який дозволяє користувачам взаємодіяти з кластером.

2. Планувальник, який призначає робочі навантаження (контейнери) певним робочим вузлам на основі доступності ресурсів та обмежень.

3. Диспетчер контролерів - керує різними контролерами, які відстежують стан кластера і при необхідності вживають коригувальні дії. Це розподілене сховище ключів-значень, яке служить базою даних кластера, зберігаючи дані про конфігурацію та бажаний стан кластера.

Робочі вузли, також відомі як робочі машини або міньйони, відповідають за запуск контейнерів і виконання робочих навантажень додатків. На кожному робочому вузлі запущено декілька важливих компонентів, зокрема:

1. kubelet: Це основний агент вузла, який взаємодіє з площиною керування та керує контейнерами на вузлі.

2. kube-proxy: Займається мережевою маршрутизацією та балансуванням навантаження між контейнерами, запущеними на вузлі.

3. Container runtime: Відповідає за витягування образів контейнерів та запуск контейнерів

Щоб розгорнути програму на Kubernetes, треба визначити її бажаний стан за допомогою файлу маніфесту Kubernetes, написаного у форматі YAML або JSON. У файлі маніфесту вказується бажана кількість контейнерів, їхні вимоги до ресурсів, мережа та інші конфігурації. Потім треба надсилати цей файл на сервер Kubernetes API, який відповідним чином оновлює стан кластера.

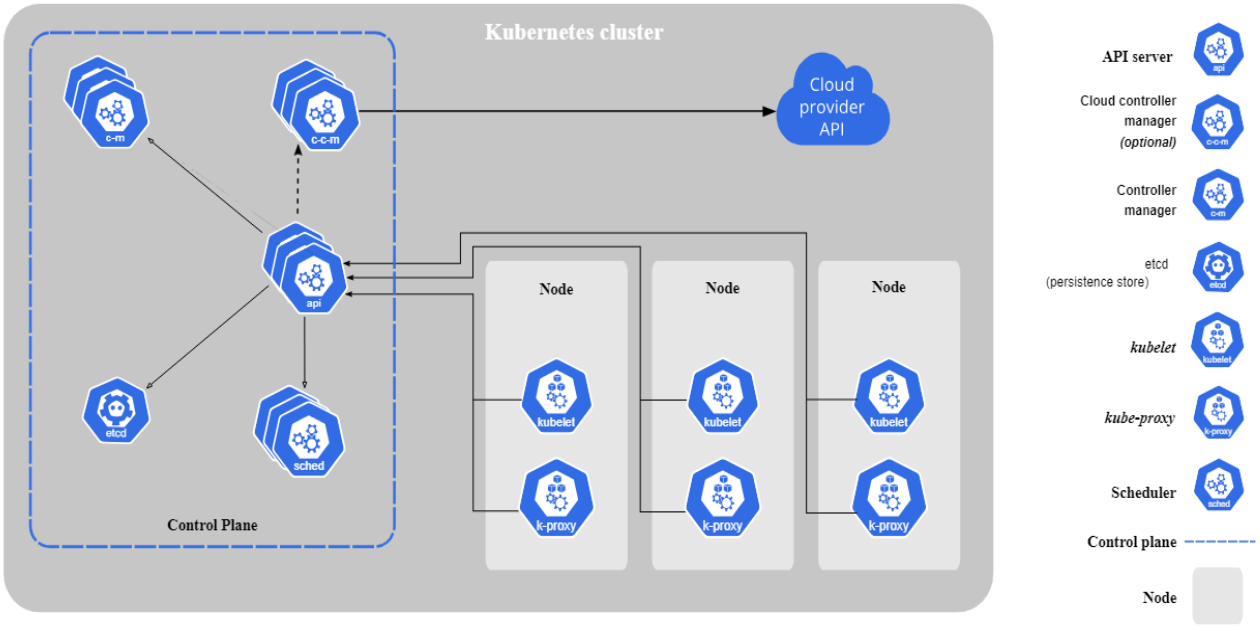


Рисунок 1.14 Схема кластеру Kubernetes

1.4 Огляд інструментів та програмних рішень для виконання дипломного проекту

Для візуального проєктування архітектури здебільшого було використано графічний редактор Figma.

Figma - це хмарний інструмент для проектування та створення прототипів, який широко використовується для створення користувацьких інтерфейсів та візуального дизайну. Він забезпечує середовище для спільної роботи, де дизайнери та зацікавлені сторони можуть працювати разом у режимі реального часу, що робить його придатним для команд, які працюють віддалено або в різних місцях. Figma дозволяє створювати каркаси, макети та інтерактивні

прототипи, що робить його ідеальним вибором для візуального проектування архітектури.

В контексті архітектурного проектування Figma можна використовувати для створення візуальних зображень бажаної архітектури системи. Це можуть бути діаграми, блок-схеми та інші візуальні елементи, які допомагають проілюструвати компоненти, взаємодії та загальну структуру архітектури. Функції Figma, такі як векторне редагування, шарування та бібліотеки компонентів, дозволяють дизайнерам створювати масштабовані та багаторазові проектні ресурси, якими можна легко ділитися та повторювати їх використання.

Docker-CLI (Command-Line Interface - інтерфейс командного рядка) - це інструмент командного рядка, що надається Docker, який дозволяє взаємодіяти з Docker і керувати контейнерами, образами, мережами та іншими ресурсами, пов'язаними з Docker. Він надає набір команд, які дозволяють розробникам створювати, запускати та керувати контейнерами з командного рядка або за допомогою сценаріїв автоматизації.

Використання Docker-CLI для контейнеризації означає, що компоненти архітектури, такі як мікросервіси або модулі, можуть бути упаковані в окремі контейнери. Це забезпечує кращу ізоляцію, масштабованість і гнучкість при розгортанні та управлінні архітектурою. Команди Docker-CLI можна використовувати для створення образів контейнерів, створення контейнерів на основі цих образів та управління життєвим циклом контейнерів під час розробки, тестування та розгортання.

Kubeadm - це інструмент командного рядка, який допомагає налаштувати кластер Kubernetes. Kubernetes - це платформа оркестрування контейнерів з відкритим вихідним кодом, яка автоматизує розгортання, масштабування та управління контейнерними додатками. Вона забезпечує масштабовану та відмовостійку інфраструктуру для запуску розподілених систем.

В контексті локальної емуляції Kubeadm дозволяє створювати кластер Kubernetes на своїй локальній машині або наборі віртуальних машин для тестування та розробки. Це спрощує процес налаштування необхідних

компонентів кластера Kubernetes, таких як площина керування та робочі вузли, шляхом автоматизації кроків конфігурації.

Використовуючи Kubeadm для локальної емуляції, можна створити середовище Kubernetes, яке дуже нагадує виробничий кластер, що дозволяє тестувати програми та конфігурації інфраструктури в контрольованому середовищі. Це дозволяє перевіряти поведінку своєї архітектури та гарантувати, що вона функціонує належним чином у контексті Kubernetes.

Загалом, ці інструменти та технології відіграють важливу роль у проектуванні, контейнеризації та емуляції візуальних архітектур. Вони надають необхідні засоби для створення візуальних представлень, пакування компонентів у контейнери та налаштування локальних середовищ Kubernetes, тим самим сприяючи ефективній розробці та розгортанню.

1.5 Проектування мікросервісної частини сайту коледжу

На першому етапі було узагальнено функції та особливості проекту, щоб визначити, як їх можна ефективно розподілити між різними мікросервісами. Цей крок допомагає визначити межі та обов'язки кожного мікросервісу, гарантуючи, що вони будуть розроблені таким чином, щоб бути узгодженими та слабко пов'язаними між собою.

Для спрощення процесу розробки було прийнято рішення реалізувати клієнтську частину системи як моноліт, а не окремий мікросервіс. В даному контексті під клієнтською частиною мається на увазі користувацький інтерфейс або фронт-енд додатку. Реалізація її як моноліту означає, що клієнтська кодова база розробляється як єдиний, уніфікований додаток. Такий підхід обрано для спрощення процесу розробки, оскільки він дозволяє розробникам зосередитися на створенні користувацького інтерфейсу без додаткової складності в управлінні декількома мікросервісами.

Для клієнтської реалізації було вирішено використовувати React, популярну бібліотеку JavaScript для побудови користувацьких інтерфейсів, та технологію SPA (Single-Page Application). React забезпечує компонентний підхід

до розробки інтерфейсу, що полегшує створення багаторазових і модульних елементів інтерфейсу. Технологія SPA, з іншого боку, забезпечує безперебійну та плавну роботу користувача, завантажуючи весь додаток один раз і динамічно оновлюючи вміст без необхідності перезавантаження сторінок.

Було прийнято рішення впровадити єдину централізовану базу даних для всіх мікросервісів в рамках системи. Цей підхід був обраний для того, щоб спростити процеси обслуговування та розробки. Завдяки впровадженню спільної бази даних управління даними стає простішим, а також забезпечується узгодженість між різними мікросервісами. Усувається дублювання та синхронізація даних у різних базах даних, що зменшує складність та пом'якшує потенційну неузгодженість даних.

PostgreSQL була обрана в якості платформи для баз даних через її придатність для обраних технологій, а саме AsyncPG та Entity Framework. PostgreSQL пропонує ряд переваг, які роблять її ідеальним вибором для цього сценарію. Підтримка асинхронного зв'язку через AsyncPG забезпечує ефективну та швидку взаємодію з базою даних, особливо при обробці паралельних запитів та великих обсягів транзакцій. Крім того, інтеграція з Entity Framework, популярним фреймворком об'єктно-реляційного відображення (ORM), надає розробникам зручний і продуктивний спосіб взаємодії з базою даних, використовуючи знайомі парадигми програмування.

Зручність і швидкість, які пропонує PostgreSQL, сприяють загальній ефективності обраних технологій і полегшують розробку та підтримку системи. Використовуючи можливості PostgreSQL, команда може оптимізувати продуктивність, забезпечити цілісність даних і спростити реалізацію різних функціональних можливостей мікросервісів. Такий централізований підхід до проектування бази даних сприяє узгодженості, спрощує управління даними і, зрештою, підвищує загальну надійність та ефективність системи в цілому.

Для того, щоб розробити архітектуру мікросервісів для системи, на основі наданих вимог можна визначити декілька мікросервісів. Ці мікросервіси будуть відповідати за управління різними аспектами системи, такими як інформація про

студентів, інформація про викладачів, групи, оцінки, курси, облік успішності, зберігання файлів, а також автентифікація та авторизація користувачів.

Студентський сервіс: Цей мікросервіс виконуватиме операції, пов'язані з управлінням студентами. Він надаватиме функції для реєстрації нових студентів, отримання інформації про студентів, оновлення даних про студентів та управління зарахуванням студентів на курси. Студентський сервіс також включатиме систему автентифікації та авторизації користувачів, що дозволить студентам отримати безпечний доступ до власної інформації.

Сервіс для викладачів: Сервіс для викладачів відповідатиме за управління операціями, пов'язаними з викладачами. Він надаватиме функції для реєстрації викладачів, пошуку інформації про викладачів, оновлення інформації про викладачів та призначення викладачів на певні курси чи групи. Як і сервіс для студентів, сервіс для викладачів включатиме систему автентифікації та авторизації користувачів для викладачів.

Сервіс груп: Сервіс груп буде виконувати операції, пов'язані з управлінням групами або класами. Він дозволить створювати нові групи, призначати студентів і викладачів до груп, отримувати інформацію про групи та керувати розкладом груп. Служба груп також включатиме систему автентифікації та авторизації для групового доступу.

Служба оцінювання: Цей мікросервіс буде зосереджений на управлінні оцінками студентів та відстеженні прогресу. Він надаватиме функції для запису та отримання оцінок, обчислення середніх значень, створення звітів та відстеження прогресу студента в часі. Сервіс оцінок буде інтегрований зі Студентським сервісом та Сервісом курсів для забезпечення точного управління оцінками.

Сервіс курсів: Сервіс курсів відповідатиме за управління поточними курсами, що пропонуються в системі. Він виконуватиме такі операції, як створення нових курсів, пошук інформації про курси, оновлення деталей курсів та управління розкладом курсів. Служба курсів буде інтегрована зі службою груп для призначення курсів певним групам.

Служба зберігання файлів: Служба зберігання файлів забезпечить базову систему зберігання файлів для зберігання та пошуку файлів, пов'язаних з роботою системи. Вона оброблятиме завантаження файлів, управління сховищем та безпечний доступ до файлів для інших мікросервісів або користувачів. Цей сервіс відповідатиме за зберігання документів, таких як матеріали курсів або студентські роботи.

Служба автентифікації та авторизації: Цей мікросервіс керуватиме процесами автентифікації, авторизації та ідентифікації системи. Вона оброблятиме реєстрацію користувачів, автентифікацію (вхід/вихід) та авторизацію для різних ролей користувачів, таких як викладач, студент, адміністратор та інші. Цей сервіс гарантуватиме, що лише авторизовані користувачі матимуть доступ до системи та виконуватимуть певні дії відповідно до своїх ролей.

Розподіляючи ці мікросервіси в системі, кожен сервіс може зосередитися на своїй конкретній функціональній області, що призводить до кращого обслуговування, масштабованості та розподілу проблем. Ці мікросервіси можуть взаємодіяти один з одним за допомогою чітко визначених API та протоколів, що дозволяє створити цілісну та інтегровану систему.

Під час подальшого аналізу було виявлено, що дублювання коду стало значною проблемою в архітектурі мікросервісів. Деякі мікросервіси мали схожі блоки коду, що призводило до надмірних зусиль з розробки та збільшення складності обслуговування. Дублювання коду зазвичай вважається небажаним, оскільки воно може призвести до неузгодженостей, розростання коду та більшої ймовірності появи помилок або неузгодженостей при внесенні змін.

Крім того, синхронізація схем даних виявилася проблемою між мікросервісами. Оскільки мікросервіси взаємодіють один з одним за допомогою REST, підтримувати узгодженість схем даних між сервісами стало складно. Зміни, внесені в схему одного мікросервісу, могли вимагати відповідних оновлень в інших мікросервісах для забезпечення сумісності та узгодженості

даних. Відсутність централізованого механізму управління та синхронізації схем загострювала проблему.

С Спочатку було прийнято рішення використання брокера повідомлень, такого як RabbitMQ. Брокери повідомлень полегшують асинхронну комунікацію і можуть бути корисними для розділення сервісів, дозволяючи їм працювати незалежно. Однак, після проведення декількох тестів стало очевидно, що такий брокер повідомлень, як RabbitMQ, не забезпечує необхідного зворотного зв'язку та рівня контролю, який пропонує REST-зв'язок.

REST (Representational State Transfer) - це архітектурний стиль, який використовує протоколи HTTP для зв'язку між системами. Він сприяє бездержавному, масштабованому і простому підходу до обміну даними. REST API широко використовуються, і багато фреймворків для розробки надають надійні інструменти та бібліотеки для створення та споживання REST-сервісів. Простота і легкість інтеграції роблять REST популярним вибором для архітектур мікросервісів.

Рішення використовувати REST як механізм зв'язку між мікросервісами пов'язане з його широким розповсюдженням та усталеними найкращими практиками. REST API, як правило, легше розробляти, підтримувати і використовувати порівняно з більш складними альтернативами, такими як брокери повідомлень. Вони дозволяють здійснювати стандартизовану взаємодію за допомогою методів HTTP (GET, POST, PUT, DELETE) і підтримують бездержавну комунікацію, що добре узгоджується з принципами мікросервісів.

Для розв'язання цих проблем було ухвалено рішення розділити інфраструктуру на 3 частини: Ядро, Другий порядок передачі, і Третій порядок передачі. Схематично, хоч конструювання видачі ендпоїнтів методом "Піраміди", коли вищі сервіси в ієрархії будуються з нижчих і простих сервісів, здавалося розв'язанням проблеми, але за фактом виявилося занадто комплексним і вкрай ненадійним, що перекреслювало всі плюси використання мікросервісів.

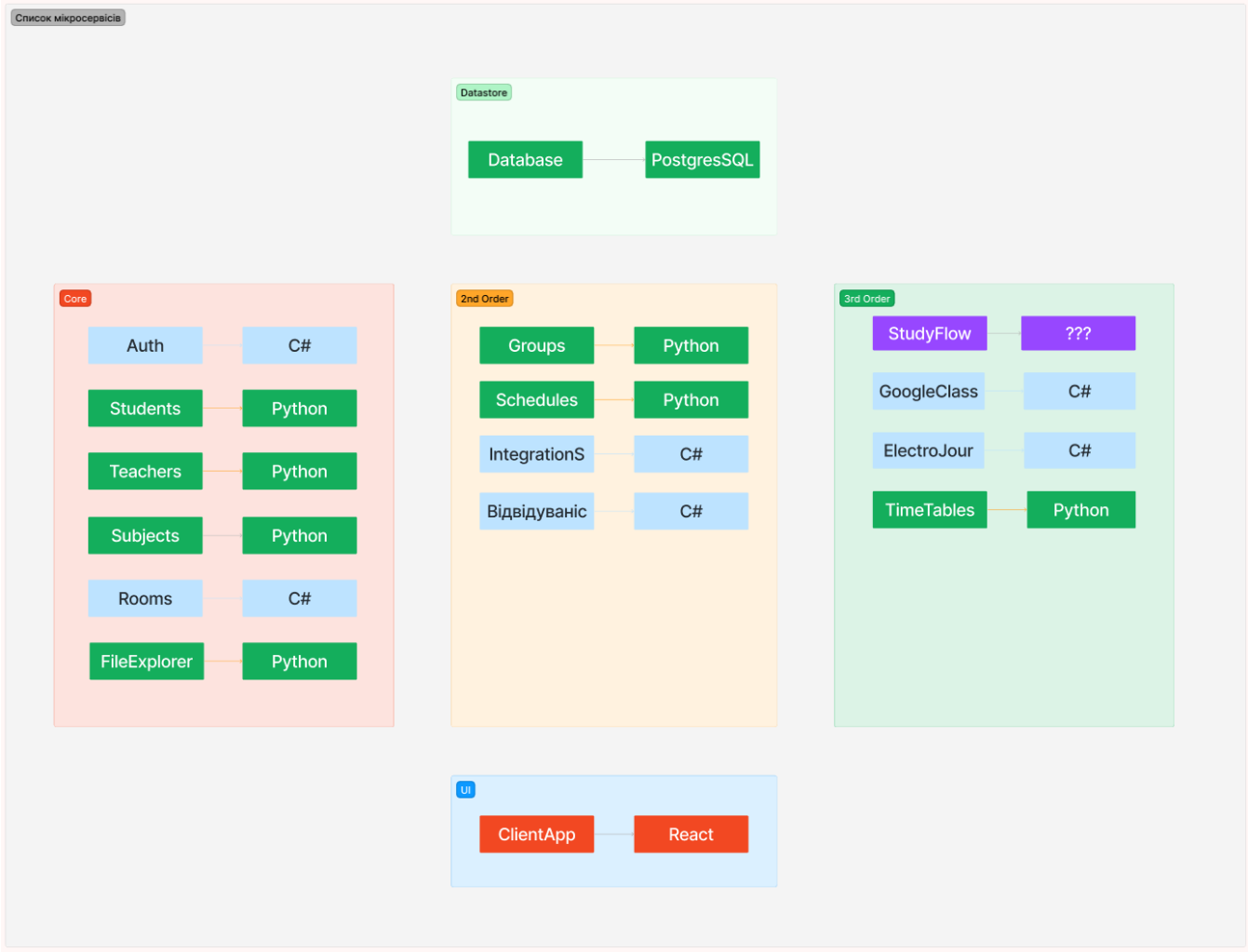


Рисунок 1.15 Перша структурна схема мікросервісів

Після більш поглибленого вивчення проблеми, і складання більш зрозумілого списку потрібних мікросервісів було ухвалено рішення відійти від ідеї дроблення системи, було ухвалено рішення зробити 4 мікросервіси, які матимуть більшу кількість функцій. База даних і Клієнтська частина залишилися недоторканими

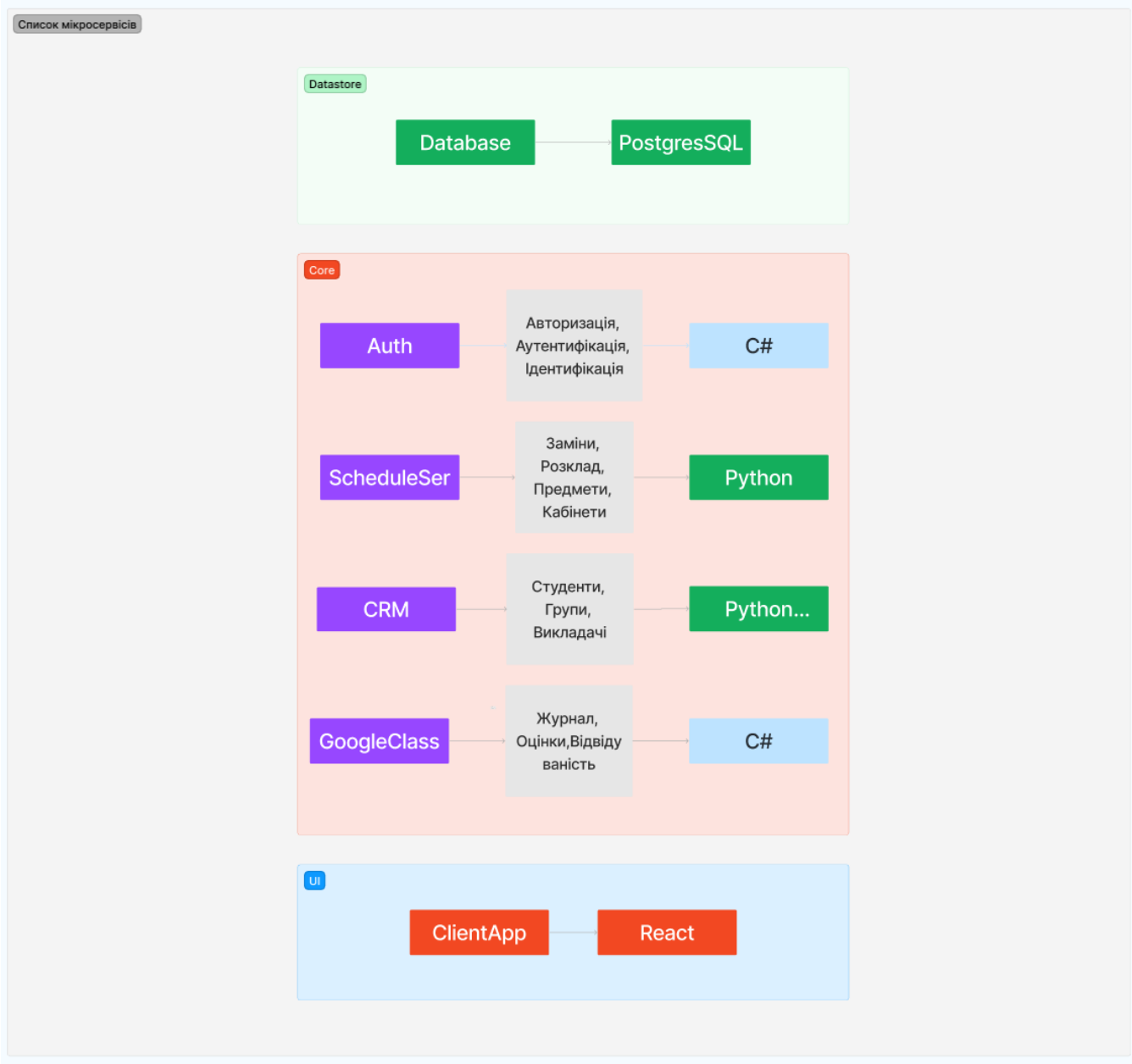


Рисунок 1.16 Остаточний варіант проектування

Служба автентифікації відіграє вирішальну роль в ідентифікації та автентифікації користувачів. Вона слугує ядром ідентифікації слухача і тісно пов'язана з іншими мікросервісами. Під час кожного запиту служба автентифікації розподіляє відповідну роль слухача, забезпечуючи безпечний і санкціонований доступ до системних ресурсів.

Служба розкладу відповідає за управління різними аспектами планування в системі. Вона займається підготовкою замін, розкладів і зберігає інформацію про предмети та аудиторії. Надаючи функціональні можливості для ефективного

складання розкладу, сервіс спрощує процес і гарантує, що заміни та розклади будуть належним чином управлятися.

CRM (Customer Relationship Management) - це об'єднаний мікросервіс, що поєднує інформацію про викладачів, групи та студентів. До об'єднання існували проблеми, пов'язані з доступом декількох мікросервісів до інформації про студентів, що призводило до дублювання коду. Завдяки консолідації відповідної інформації в CRM-сервісі, дублювання коду зведено до мінімуму, а сервіс стає центральною точкою контакту для управління та пошуку інформації про викладачів, групи та студентів.

Сервіс GoogleClass - це мікросервіс, який інтегрує різні функції, пов'язані з електронними журналами, паперовими роботами та оцінюванням учнів. Він діє як комплексна платформа, де вчителі можуть керувати прогресом учнів, отримувати та оцінювати завдання, а також виставляти оцінки. Поєднуючи ці функції у спеціальному мікросервісі, сервіс GoogleClass спрощує процес управління та відстеження успішності учнів.

Шлюз API є ключовим компонентом в архітектурі системи. Він діє як зовнішній шлюз, який отримує запити від зовнішніх систем і клієнтських додатків. Потім API-шлюз перенаправляє ці запити на відповідні внутрішні ресурси системи. Він забезпечує централізовану точку входу для зовнішніх систем для взаємодії з мікросервісами і забезпечує належну маршрутизацію та обробку вхідних запитів.

Додавання цих мікросервісів підвищує функціональність та ефективність системи в цілому. Служба Auth забезпечує безпечну автентифікацію та авторизацію, дозволяючи контролювати доступ до системних ресурсів. Сервіс розкладу спрощує процес складання розкладу, а CRM-сервіс консолідує та управляє інформацією про викладачів, групи та студентів. Сервіс GoogleClass надає комплексну платформу для електронних журналів, паперових робіт та оцінювання. Шлюз API діє як центральна точка входу для зовнішніх систем, полегшуючи зв'язок з внутрішніми мікросервісами.

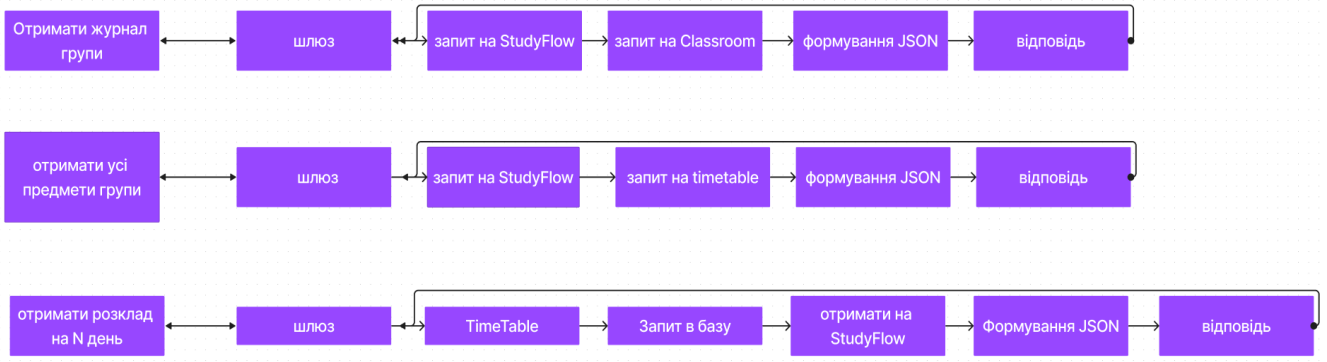


Рис 1.17 Приклад роботи API-шлюзу

Створення списку кінцевих точок API для кожного мікросервісу є важливим кроком у розробці та підтримці проекту. Кінцеві точки API визначають точки входу для зв'язку з кожним мікросервісом, вказуючи доступні операції та формати даних. Цей список допомагає в проектуванні, реалізації та документуванні API, забезпечуючи узгодженість і ясність протягом усього процесу розробки.

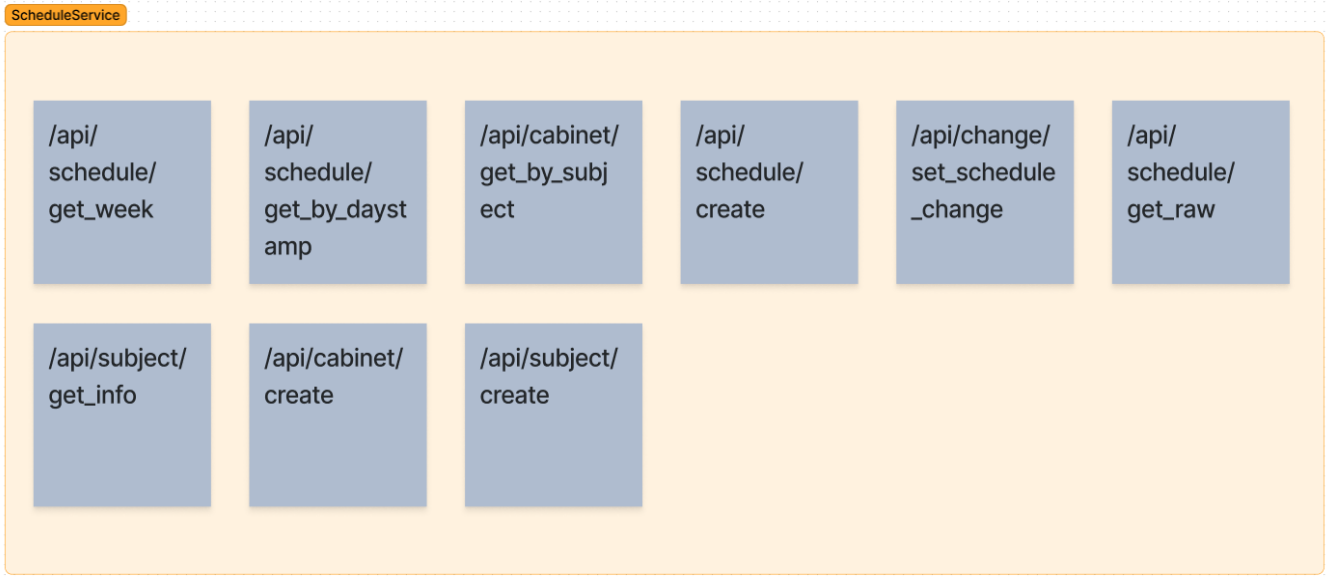


Рисунок 1.18 Кінцеві точки API для Служби розкладу

Кінцеві точки API сервісу розкладу надають функції, пов'язані з керуванням розкладом, замінами, предметами та аудиторіями. Вони можуть включати кінцеві точки для отримання розкладу, додавання або зміни замін, запиту інформації про предмети та управління доступністю класів. Ці кінцеві точки забезпечують ефективну роботу з розкладом у системі (рисунок 1.18).

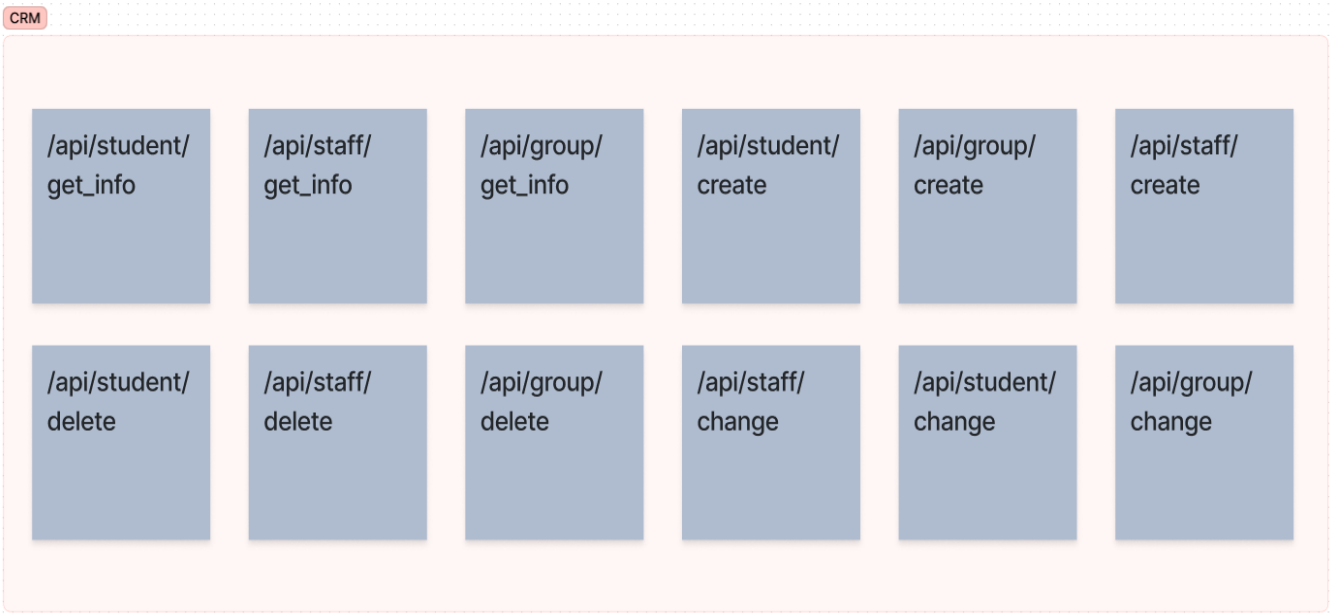


Рисунок 1.19 Кінцеві точки API для CRM системи

CRM Service API виконують операції, пов'язані з управлінням інформацією про викладачів, групи та студентів. Вони включають кінцеві точки для створення, пошуку, оновлення та видалення записів про вчителів, групи та учнів. Крім того, можуть бути включені кінцеві точки для запиту інформації про студентів, призначення викладачів до груп та отримання інформації про групи. Ці кінцеві точки забезпечують безперешкодне керування даними про вчителів, групи та учнів (рисунок 1.19).

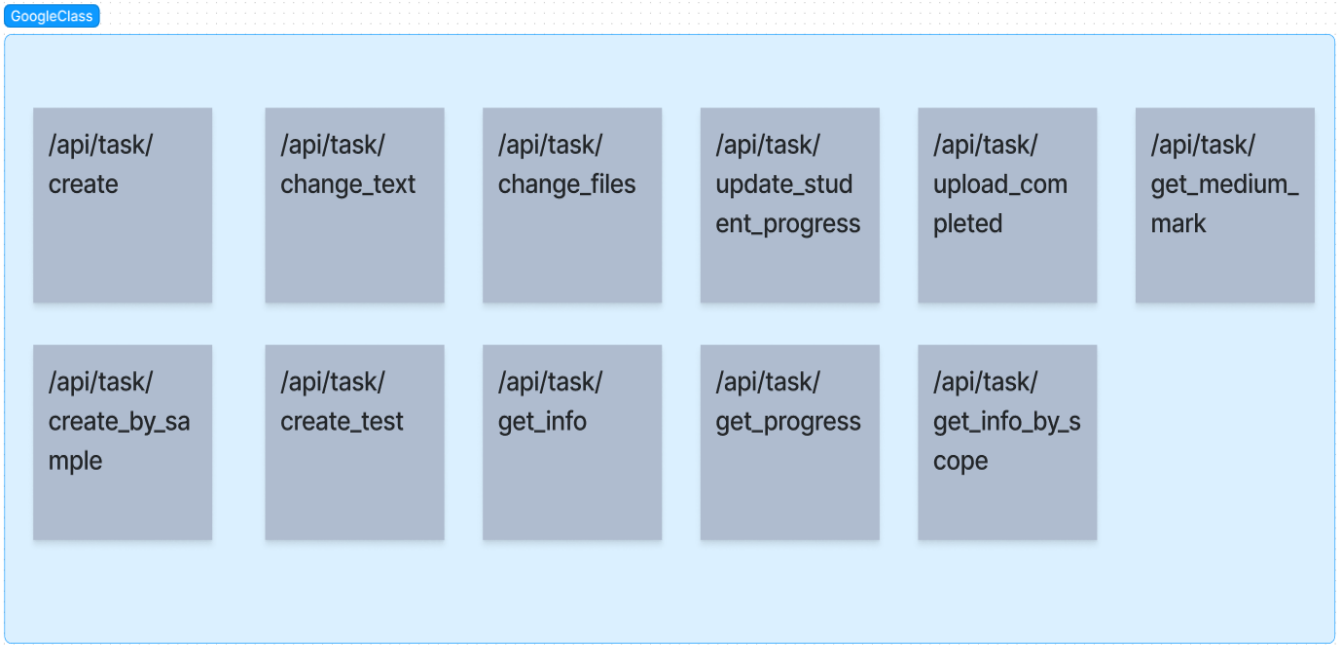


Рисунок 1.20 Кінцеві точки API для GoogleClass мікросервісу

API сервісу GoogleClass надають функціональні можливості для ведення електронних журналів, подання паперових робіт та виставлення оцінок учням. Вони включають кінцеві точки для створення, оновлення та пошуку записів у журналі, надсилання завдань, отримання інформації про завдання та управління інформацією про оцінювання. Ці кінцеві точки дозволяють викладачам ефективно керувати завданнями та оцінками студентів (рисунок 1.20).

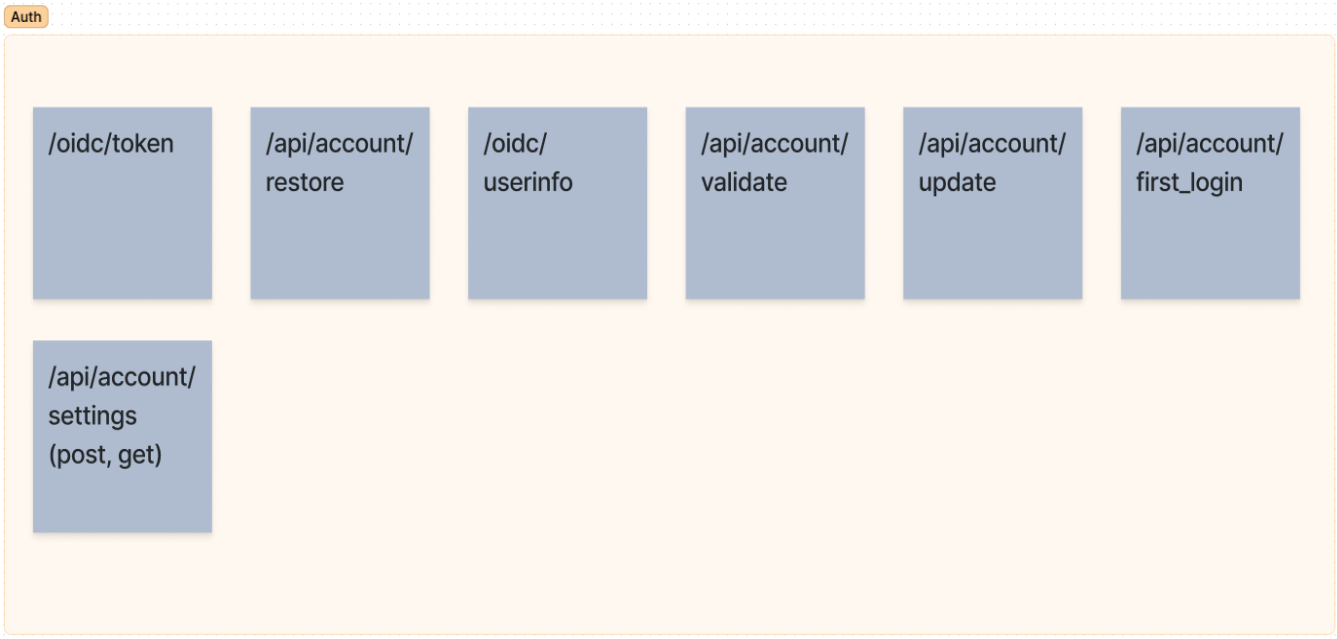


Рисунок 1.21 Кінцеві точки API для мікросервісу Auth

API служби автентифікації обробляють автентифікацію, авторизацію та ідентифікацію користувачів. Вони включають кінцеві точки для реєстрації користувачів, входу, виходу, скидання пароля та управління ролями. Ці кінцеві точки полегшують безпечний доступ до ресурсів системи на основі ролей і дозволів користувачів (рисунок 1.21).

2 ЕКОНОМІЧНА ЧАСТИНА

2.1 Резюме

В умовах розвитку сучасного інформаційного суспільства та економічних відносин, які в ньому встановилися, дуже важливою є роль інформаційних ресурсів, до яких відносять веб-сайти

Вдале використання інформаційного забезпечення та інформаційних ресурсів може значно підвищити ефективність функціонування підприємства.

Основна мета створення пвеб-сайту для коледжу – це забезпечення усіх зацікавлених людей ним зручним користувацьким досвідом

При оцінці ефективності створюваного сайту було виявлено, що сайт має лише функціональну та соціальну ефективність. Відсутність економічної ефективності зумовлена тим, що сайт не має прибутку і несе виключно функціональний та соціальний характер. Людина, що займалася замінами фактично тепер займає посаду адміністратора і має ту ж зарплатню

2.2 Визначення трудомісткості розробки веб-додатку

Загальні витрати ($В_з$) на створення сайту складаються з декількох параметрів:

$$В_з = В_р + В_в + В_е \quad (2.1)$$

де $В_р$ – витрати на розробку сайту;

$В_в$ – витрати на впровадження сайту;

$В_е$ – витрати на експлуатацію сайту;

Витрати на розробку сайту ($В_р$) є одноразовими та залежать від етапів розробки сайту

Для визначення витрат на розробку сайту ($В_р$) розраховуємо оплату праці виконавців, безпосередньо притягнених до її виконання. Для реалізації проекту Web-системи використовуються наступні спеціалісти: дизайнер, архітектор, фронтендер, бекендер, системний адміністратор

Для визначення трудомісткості розробки сайту (B_p) складено план-графік по розробці web-сайту і тривалості виконання робіт. Розподіл робіт по етапах і видах виконавців наведено в таблиці 2.1.

Таблиця 2.1 – План-графік по розробці Web-сайту

№	Назва етапу	Час виконання (годин)	Посада виконавця
1	Проектування	10	Архітектор
2	Створення дизайну	25	Дизайнер
3	Реалізація клієнтської частини	100	Фронтендер
4	Реалізація серверної частини	100	Бекендер
5	Розгортка веб-додатку	5	Сис. адмін
	ВСЬОГО:	240	

Розрахунок трудомісткості здійснений в наступній послідовності:

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

2. По кожному виду робіт визначений кваліфікаційний рівень виконавців. В разі виконання однієї роботи виконавцями різної кваліфікації, робота розподілена на ряд паралельних конкретних робіт для кожної категорії виконавця

Тривалість виконання робіт розраховується на основі вірогідних оцінок робіт, що задаються виконавцями

Розмір заробітної плати розраховано виходячи з чисельності різних категорій виконавців, трудомісткості, що витрачається ними на виконання різних видів робіт, а також їх середньої заробітної плати за годину.

Витрати на заробітну плату приведені в таблиці 2.2.

В умовах відсутності нормативної бази тривалість виконання окремих робіт розраховуємо на основі вірогідних оцінок робіт, що задаються виконавцями.

Таблиця 2.2 Витрати на заробітну плату

№	Персонал	Етапи розробки	Кількість робочих годин	Погодинна ставка, грн.	Заробітна плата, грн.
1	Архітектор	Проектування	10	41	410
2	Дизайнер	Створення дизайну	25	41	1025
3	Фронтендер	Реалізація клієнтської частини	100	41	4100
4	Бекендер	Реалізація серверної частини	100	41	4100
5	Системний адміністратор	Розгортка веб-додатку	5	41	205
				ВСЬОГО:	$В_{зп}$ = 9840

Розмір єдиного соціального внеску складає 22% від заробітної плати, розраховано за наступною формулою:

$$В_{єсв} = В_{зп} \text{ x } 0,22 \quad (2.2)$$

$$В_{єсв} = 9840 \text{ x } 0,22 = 2164.8 \text{ грн.}$$

Загальні витрати ($В_р$) на розробку веб-сайту розраховано як сума витрат на заробітну плату праці персоналу ($В_{зп}$) та єдиного соціального внеску ($В_{єсв}$):

$$В_р = В_{зп} + В_{єсв} \quad (2.3)$$

$$В_р = 9840 + 2164.8 = 12\ 004.8 \text{ грн.}$$

Витрати на впровадження сайту ($В_в$) складаються з двох складових:

- витрати на реєстрацію доменного імені на 1 рік ($В_{в1}$);
- витрати на реєстрацію в пошукових системах ($В_{в2}$),

$$В_в = В_{в1} + В_{в2} \quad (2.4)$$

$$В_в = 432 + 0 = 432 \text{ грн}$$

Витрати на експлуатацію сайту (Ве) включають вартість робіт з підтримки сайту в робочому стані і вартість послуг по продовженню доменного імені на 1 рік

Роботи по підтримці сайту в робочому стані включають в себе:

1. Оновлення даних на сайті;
2. Створення нових розділів на сайті;
3. Видалення застарілої інформації з сайту;
4. Додавання потрібної інформації на сайт;
5. Налаштування параметрів сервера хостингу;
6. Моніторинг роботи сервера хостингу;
7. Забезпечення щомісячного захисту сайту;
8. Створення резервних копій сайту та ін.

Підтримка сайту в робочому стані буде здійснювати сама організація Витрати підраховуються виходячи із заробітної плати співробітникові організації, призначеного для виконання цього виду робіт. Для певних робіт з цього переліку буде використовуватися обслуговуючий персонал (адміністратор web-сайту). У таблиці 2.3 визначаються постійні витрати як сума витрат на впровадження та експлуатацію сайту протягом року

Таблиця 2.3 Постійні витрати

№	Стаття витрат	Вартість за рік, грн.
1	Оренда серверу	2026
2	Доменне ім’я	432
3	Адміністратор	6700*12 = 80 400
	Всього:	$В_{пост}$ = 82 858

Загальні витрати ($В_з$) на розробку, впровадження та експлуатацію веб-сайту розраховуються за наступною формулою:

$$В_з = В_р + В_в + В_е \quad (2.5)$$

$$В_з = 9635 + 205 + 82\,858 = 92\,698 \text{ грн}$$

Функціональна ефективність виявляється у вдосконаленні технологічних процесів, пов'язаних із наданням інформації викладачам та студентам щодо замін у розкладі занять. Це означає, що шляхом використання певного сайту чи системи, коледж забезпечує швидку та точну інформацію про зміни у розкладі, що дозволяє всім зацікавленим сторонам оперативно реагувати та планувати свій час. Така оптимізація технологічних процесів сприяє зменшенню незручностей, пов'язаних зі змінами у розкладі, та підвищує загальну продуктивність навчального процесу.

Соціальна ефективність полягає в тому, що завдяки впровадженню такої системи або сайту коледж стає більш привабливим для потенційних абітурієнтів. Інформованість студентів та викладачів про заміни у розкладі занять позитивно впливає на загальний імідж навчального закладу. Потенційні студенти бачать, що коледж має ефективну систему комунікації та дбає про своїх студентів, що може спонукати їх обрати цей коледж для навчання. Таким чином, соціальна ефективність сприяє збільшенню привабливості навчального закладу для потенційних абітурієнтів та покращенню його репутації у громадськості.

Загалом, функціональна та соціальна ефективність оптимізації технологічних процесів для інформування викладачів та студентів про заміни у розкладі занять допомагають поліпшити якість навчання та створити сприятливі умови для всіх учасників навчального процесу**.**

3 ОХОРОНА ПРАЦІ

Безпека праці є важливою сферою практичної діяльності, що має на меті створення безпечних та нешкідливих умов праці. Забезпечення безпеки та запобігання шкідливим умовам праці вимагає значних фінансових витрат і впровадження науково-дослідних розробок у галузі охорони праці.

Умови праці для програмістів повинні відповідати вимогам безпеки праці, враховуючи нормативно-правові акти, такі як "Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями" (НПАОП 0.00-7.15-18) та "Правила охорони праці під час експлуатації електронно-обчислювальних машин" (НПАОП 0.00-1.28-10), які були затверджені Державним комітетом України з промислової безпеки, охорони праці та гірничого нагляду 26 березня 2010 року наказом № 65.

Тому знання з питань безпеки праці у технічних спеціалістів, зокрема у програмістів, мають вирішальне значення.

3.1 Гігієнічні вимоги до виробничого середовища.

3.1.1 Вимоги до приміщення

– Відповідно до нормативно-правовим актами, площа одного робочого місця повинна бути не меншою за 6.0 квадратних метрии, а оюєм 20,0 куб.м.

– Кліматичні умови: Приміщення повинно бути забезпечено системою опалення для збереження комфортної температури, кондиціонуванням повітря для контроля вологості і припливно-витяжною вентиляцією для забезпечення свіжого повітря.

– Поверхня підлоги: Підлога повинна бути рівною, неслизькою і мати антистатичні властивості, щоб запобігти статичному електрику, який може шкодити обладнанню.

Забезпечення вимог безпеки праці включає в себе не лише правильне організування приміщення, але також розгляд конструкції робочого місця та його оснащення для оптимальної робочої пози та комфорту працівника. З цією метою необхідно враховувати наступні вимоги:

3.1.2 Вимоги до організації робочого місця користувача ПК

Організація робочого місця повинна враховувати не лише вимоги до освітлення, але й інші аспекти ергономіки, такі як правильне розташування обладнання, налаштування стільця та столу для забезпечення оптимальної робочої позиції користувача. Всі ці фактори сприяють збереженню здоров'я, зниженню втоми та підвищенню продуктивності під час роботи з комп'ютером.

– Робочі місця з візуальнимекраном (ВДТ) слід розташовувати таким чином, щоб природне світло падало збоку, переважно зліва. Це допоможе уникнути надмірного навантаження на очі працівника.

– Відстань між робочими місцями: Необхідно дотримувати відстань не менше 1,2 метра між бічними поверхнями ВДТ. Також відстань від тильної поверхні одного ВДТ до екрана іншого ВДТ повинна становити 2,5 метра. Це дозволить запобігти зайвим перешкодам та забезпечити комфортну робочу зону.

– Конструкція робочого столу: Робочий стіл має відповідати сучасним вимогам ергономіки. Він повинен забезпечувати оптимальне розташування обладнання (дисплея, клавіатури, принтера) і документів на робочій поверхні. Крім того, робочий стіл повинен мати достатньо простору для ніг, регульовану висоту, кут і нахил сидіння та спинки, а також зручну форму переднього краю сидіння. Регулювання цих параметрів має бути легким, незалежним і надійним.

– Робочий стілець: Робочий стілець повинен бути підйомно-поворотним, регульованим за висотою, кутом і нахилом сидіння та спинки. Поверхня сидіння має бути плоскою, а передній край заокругленим для забезпечення комфорту працівника. Регулювання кожного параметра має бути легким і надійним. Крім того, шаг регулювання елементів стільця повинен бути в межах 15-20 мм для лінійних розмірів і 2-5 градусів для кутових розмірів.

3.2 Освітлення робочого місця

Згідно з ДСанПІН 3.3.2.007-98, у приміщеннях з робочими місцями, де використовуються ВДТ ЕОМ та ПЕОМ, рекомендується використовувати

систему загального рівномірного освітлення. Це означає, що освітлення повинно бути розподілене рівномірно по всій робочій зоні.

У виробничих та адміністративно-громадських приміщеннях, де переважно проводиться робота з документами, може бути застосована система комбінованого освітлення. Це означає, що крім загального освітлення, встановлюються також світильники місцевого освітлення, які забезпечують додаткове освітлення в окремих зонах, де розміщені документи.

Рекомендована освітленість на поверхні робочого столу в зоні розміщення документів становить 300-500 лк (люкс). Якщо системою загального освітлення досягнути таких значень освітленості не можливо, тоді допускається використовувати місцеве освітлення. Однак, важливо пам'ятати, щоб світильники місцевого освітлення не створювали блисків на поверхні екрана, а освітленість екрана не перевищувала 300 лк.

Зважаючи на ці рекомендації, як джерела світла варто використовувати переважно люмінесцентні лампи типу ЛБ. Однак, у виробничих та адміністративно-громадсь ких приміщеннях також можуть бути застосовані металогалогенні лампи потужністю 250 Вт для влаштування відбитого освітлення. Для світильників місцевого освітлення можуть використовуватись лампи розжарювання.

У разі використання штучного освітлення важливо враховувати, щоб пряме сонячне світло не падало прямо на робоче місце, оскільки це може створювати блиск на екрані і викликати незручності для користувача.

Крім того, лампи типу ЛД (світлодіодні лампи) рекомендується використовувати з розсіювачами та екрануючими сітками. Це допомагає забезпечити рівномірне розподілення світла та уникнути різких тіней чи блисків на робочій поверхні.

3.3 Мікроклімат

Згідно з нормами ДСН 3.3.6.042.-99 виробничі приміщення з робочими місцями, де використовуються ВДТ, мають мати оптимальні параметри

мікроклімату, такі як температура, відносна вологість і рух повітря (згідно з таблицею 3.1).

Таблиця 3.1

Пора року	Категорія робіт	Температура повітря, град.С	Відносна вологість повітря	Швидкість руху повітря м/с
		оптимальна	оптимальна	оптимальна
Холодна	Легка-1 а	22-24	40-60	0.1
	Легка-1 б	21-23	40-60	0.1
Тепла	Легка-1 а	23-25	40-60	0.1
	Легка-1 б	22-24	40-60	0.2

Забезпечення цих параметрів важливо для створення комфортних умов на робочих місцях з ВДТ та забезпечення здоров'я та продуктивності працівників

3.4 Іонізація повітря

Згідно з санітарно-гігієнічними нормами N 2152-80, рівні позитивних і негативних іонів у повітрі приміщень з ВДТ повинні відповідати визначеним стандартам. Таблиця 3.2 містить вказівки щодо цих норм.

Таблиця 3.2

Рівні	Кількість іонів в 1 см куб. повітря	
	N+	N-
Мінімально необхідні	400	600
Оптимальні	1500-3000	3000-5000
Максимально допустимі	50000	50000

3.5 Пожежна безпека

Пожежна безпека визначається як стан об'єкта, в якому з регламентованою ймовірністю виключається можливість виникнення пожежі та її розповсюдження, а також забезпечується захист людей від небезпечних факторів пожежі і збереження матеріальних цінностей.

Можливі причини пожежі в приміщенні включають коротке замикання проводки, неправильне використання побутових електроприладів і порушення протипожежних заходів.

У випадку виникнення пожежі необхідно:

- Негайно відключити живлення електромережі.
- Негайно викликати пожежну команду за телефоном.
- Евакуювати людей з приміщення відповідно до плану евакуації.
- Приступити до ліквідації пожежі за допомогою вогнегасників.
- Якщо пожежа мала, можна використати доступні засоби для припинення доступу повітря до вогнища.

На робочому місці для гасіння пожежі використовують вуглекислотні та порошкові вогнегасники. Кількість і види вогнегасників, а також їхнє розташування відповідають вимогам стандартів ГОСТ 12.4.009-75 та ISO3941-77.

ВИСНОВКИ

Проектування мікросервісної архітектури для сайту коледжу є важливим кроком у забезпеченні ефективного та масштабованого функціонування веб-додатка. Ця архітектура дозволяє розділити веб-додаток на невеликі, незалежні компоненти - мікросервіси, що діють разом для надання більшої функціональності.

Один з головних висновків полягає в тому, що мікросервісна архітектура сприяє підвищенню модульності та гнучкості сайту коледжу. Кожен мікросервіс може бути розроблений та розгорнутий незалежно, що спрощує розвиток та супровід системи. При цьому, різні мікросервіси можуть використовувати різні технології, що дозволяє вибрати оптимальний стек технологій для кожного компонента.

Крім того, мікросервісна архітектура сприяє масштабованості сайту. Кожен мікросервіс може бути масштабований окремо, залежно від потреб системи. Це дозволяє забезпечити високу продуктивність та швидкість роботи навіть при зростанні навантаження.

Також, мікросервісна архітектура покращує можливості впровадження нового функціоналу та оновлення сайту. Завдяки незалежності мікросервісів, їх можна розробляти та випускати в продакшн окремо, без впливу на роботу інших компонентів системи. Це дозволяє більш оперативно відповідати на змінні потреби користувачів та швидко впроваджувати нововведення.

Загалом, проектування мікросервісної архітектури для сайту коледжу є перспективним підходом, що дозволяє досягти високої гнучкості, масштабованості та швидкості роботи системи. Правильно планувати та реалізовувати цю архітектуру допоможе покращити якість навчального процесу та задовольнення потреб користувачів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Building Microservices: Designing Fine-Grained Systems [Книга] / Sam Newman 2015
2. Microservices Patterns: With examples in Java First Edition [Книга] / Chris Richardson 2018.
3. The Tao of Microservices [Книга] / Richard Rodger 2017.
4. Офіційна документація Docker [Електронний ресурс] / Solomon Hykes 2023. URL: https://docs.docker.com/
5. Офіційна документація Kubernetes [Електронний ресурс] / Google 2023. URL: https://kubernetes.io/docs/home/
6. Refactoring a monolith to microservices [Електроний ресурс] / Chris Richardson 2023. URL: https://microservices.io/refactoring/index.html
7. Офіційна документація Nginx [Електронний ресурс] / Igor Sysoev 2023. URL: https://nginx.org/en/docs/
8. What are Microservices? [Електронний ресурс] / Amazon 2023. URL: https://aws.amazon.com/ru/microservices/
9. Design a Microservices-Based Application [Електронний ресурс] / Oracle 2023. URL: https://docs.oracle.com/en/solutions/learn-architect-microservice/design-microservices-based-application1.html
10. Top 10 Microservice Architecture Design Patterns for Developers [Електронний ресурс] / Sasidhar Gadepalli 2023. URL: https://dev.to/sasidhar_gadepalli/top-10-microservice-architecture-design-patterns-for-developers-dgk
11. Application Architecture Guide Microservice architecture style [Електроний ресурс] / Microsoft 2023. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

ДОДАТОК А

Фрагмент коду з CRM мікросервісу

```
from random import randint, choice
from django.http import Http404
from django.shortcuts import get_object_or_404
from rest_framework import viewsets, generics, status
from rest_framework.views import APIView
from rest_framework.response import Response
from .models import Group
from teachers.models import Teacher
from .serializers import GroupListSerializer, CreateGroupSerializer,
ChangeTeacherSerializer
from drf_yasg.utils import swagger_auto_schema


class GroupsViewList(generics.ListAPIView):
    serializer_class = GroupListSerializer

    def get_queryset(self):
        return Group.objects.all()

    @swagger_auto_schema(request_body=CreateGroupSerializer)
    def post(self, request):
        serializer = CreateGroupSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)


class DetailGroupView(APIView):
    def get(self, request, pk):
        group = get_object_or_404(Group, id=pk)
        serializer = GroupListSerializer(group)
        return Response(serializer.data)


class DeleteGroupView(APIView):
    def delete(self, request, pk):
        group = get_object_or_404(Group, id=pk)
        group.delete()
        return Response(status=204)
```

```
class UpdateGroupView(APIView):
    serializer_class = CreateGroupSerializer

    def put(self, request, pk):
        group = get_object_or_404(Group, id=pk)
        group.name = request.data['name']
        group.course = request.data['course']
        group.save()
        return Response(status=200)


class CreateRandomGroupView(APIView):
    def post(self, request, count):
        possible_group_names = [
            'РП', 'КГ', 'КС', 'КБ', 'КВ', 'ИМ', 'ТТ', 'МХ', 'ВВ', 'МК', 'ЕП', 'ОД'
        ]
        for i in range(int(count)):
            group = Group()
            group.name = f"{choice(possible_group_names)}-{str(randint(1, 50))}"
            group.course = randint(1, 4)
            group.save()
        return Response(status=200)


class ChangeGroupTeacher(APIView):
    serializer_class = ChangeTeacherSerializer

    def post(self, request, pk, teacher_id):
        group = get_object_or_404(Group, id=pk)
        group.classroom_teacher = get_object_or_404(Teacher, id=teacher_id)
        group.save()
        return Response(status=200)


from rest_framework import serializers
from .models import Group


# serializer for group creation
class GroupListSerializer(serializers.ModelSerializer):
    class Meta:
        model = Group
        fields = ('id', 'name', 'course', 'classroom_teacher', 'students')
```

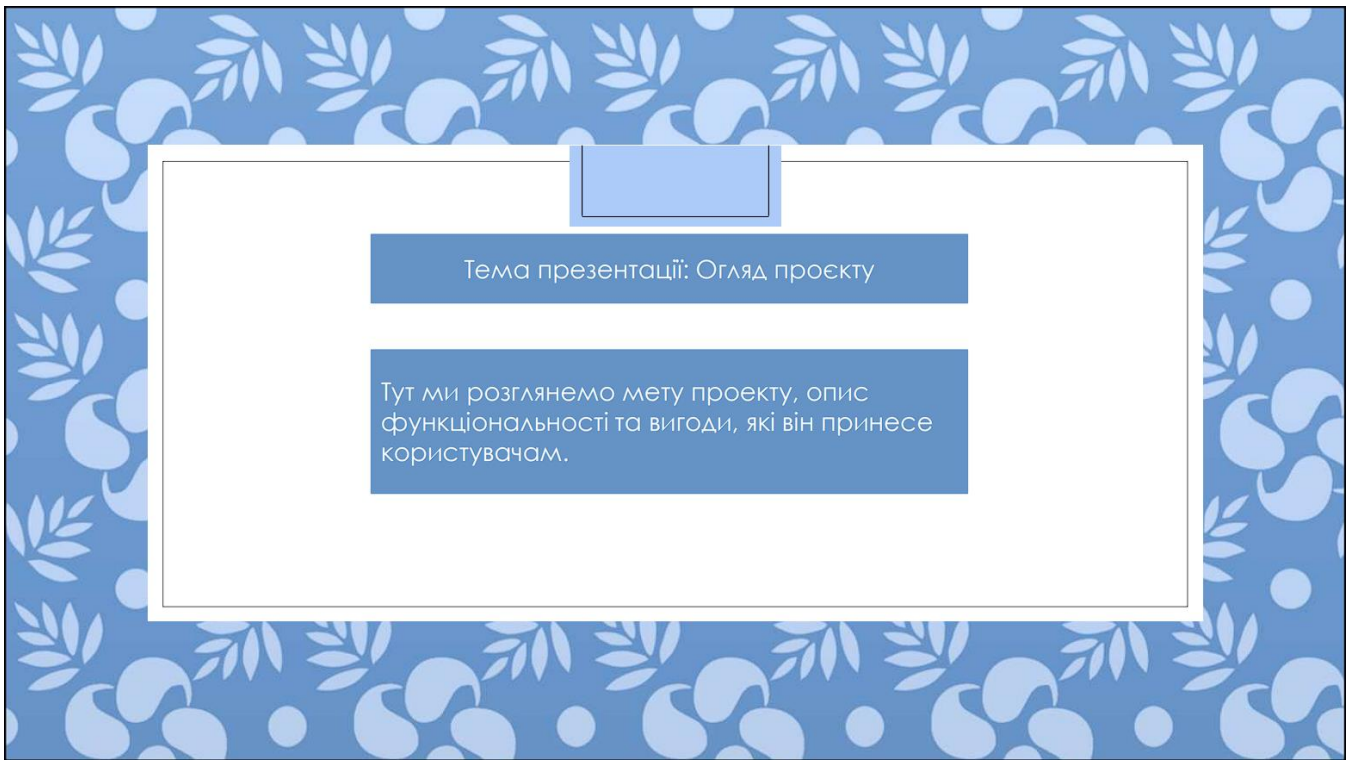
```
class CreateGroupSerializer(serializers.ModelSerializer):
    class Meta:
        model = Group
        fields = ('name', 'course')


class UpdateGroupSerializer(serializers.ModelSerializer):
    class Meta:
        model = Group
        fields = ('id', 'name', 'course')


class ChangeTeacherSerializer(serializers.ModelSerializer):
    class Meta:
        model = Group
        fields = ('id', 'classroom_teacher')

from django.urls import path
from rest_framework import routers
import groups.views as views


urlpatterns = (
    path('group/', views.GroupsViewList.as_view(), name='Groups'),
    path('group/<int:pk>', views.DetailGroupView.as_view(), name='Group'),
    path('group/delete/<int:pk>', views.DeleteGroupView.as_view(),
name='DeleteGroup'),
    path('group/update/<int:pk>', views.UpdateGroupView.as_view(),
name='UpdateGroup'),
    path('group/create_random/<int:count>',
views.CreateRandomGroupView.as_view(), name='CreateRandomGroup'),
    path('group/change_teacher/<int:pk><int:teacher_id>',
views.ChangeGroupTeacher.as_view(), name='ChangeTeacher'),
)


# from django.contrib import admin
from django.urls import path, re_path, include
from rest_framework import permissions
from drf_yasg.views import get_schema_view
from drf_yasg import openapi


schema_view = get_schema_view(
```

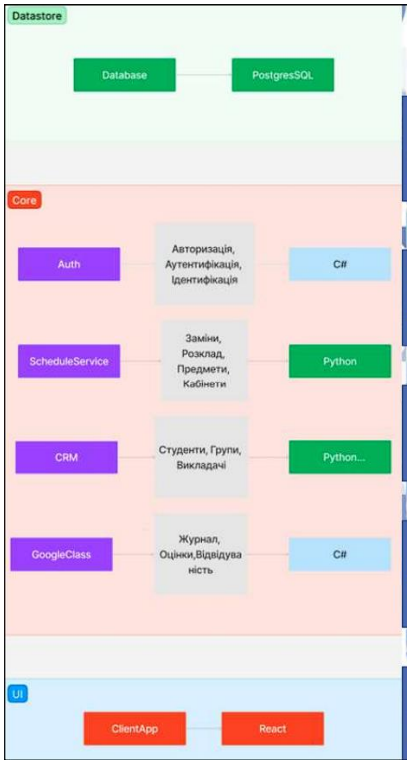
```
  openapi.Info(
    title="STG API",
    default_version='v1',
    description="Students, Teachers, Groups microservice for e.Gu project",
    terms_of_service="https://www.google.com/policies/terms/",
    contact=openapi.Contact(email="example@gmail.com"),
    license=openapi.License(name="BSD License"),
  ),
  #patterns=[path('api/', include('groups.urls')), ],
  public=True,
  permission_classes=[permissions.AllowAny],
)

urlpatterns = [
   re_path(r'^swagger(?P<format>\.json|\.yaml)$',
schema_view.without_ui(cache_timeout=0), name='schema-json'),
   re_path(r'^swagger/$', schema_view.with_ui('swagger', cache_timeout=0),
name='schema-swagger-ui'),
   re_path(r'^redoc/$', schema_view.with_ui('redoc', cache_timeout=0),
name='schema-redoc'),
   path('', include('groups.urls')),
]
from django.db import models
from django.core.validators import MinValueValidator, MaxValueValidator

# Create your models here.


class Teacher(models.Model):
   class Meta:
      db_table = 'college_teachers'
      verbose_name = 'Teacher'
      verbose_name_plural = 'Teachers'

   name = models.CharField(max_length=100)
   surname = models.CharField(max_length=100, null=True)
   patronymic = models.CharField(max_length=100, null=True)
   gender = models.CharField(max_length=1, choices=[('M', 'F')])
   age = models.IntegerField(validators=[MinValueValidator(18),
MaxValueValidator(100)], default=20)
   avatar_url = models.URLField(default="https://www.meme-
arsenal.com/memes/a62ec57bf0e5278aeee2882b19b19b89.jpg", blank=False)

   def __str__(self):
      return f'{self.name} {self.surname}'
```

```
from django.db import models
from django.core.validators import MinValueValidator, MaxValueValidator


# Create your models here.
class Group(models.Model):
    class Meta:
        db_table = 'college_groups'
        verbose_name = 'Group'
        verbose_name_plural = 'Groups'

    name = models.CharField(max_length=20, unique=True)
    course = models.IntegerField(
        validators=[MinValueValidator(0),
                    MaxValueValidator(5)],
        default=1)
    classroom_teacher = models.OneToOneField("teachers.Teacher",
on_delete=models.SET_NULL, null=True)
    #classroom = models.OneToOneField("rooms.Room",
on_delete=models.SET_NULL, null=True)

    def __str__(self):
        return f'{self.course}{self.name}'


class SubGroup(models.Model):
    class Meta:
        db_table = 'college_subgroups'
        verbose_name = 'SubGroup'
        verbose_name_plural = 'SubGroups'

    subject = models.CharField(max_length=200)
    number = models.IntegerField(default=1)
    teacher = models.ForeignKey("teachers.Teacher", on_delete=models.SET_NULL,
null=True, related_name='teachers')
    group = models.ForeignKey(Group, on_delete=models.CASCADE,
related_name='subgroups')

    def __str__(self):
        return f'{self.group.course}{self.group.name} {self.subject} subgroup
{self.number}'
```

ДОДАТОК Б

Слайди мультимедійної презентації





Плюси обраної архітектури

Злиття трафіку: API Gateway дозволяє злити трафік від різних джерел, таких як мобільні додатки, веб-додатки або інші сервіси, і перенаправити його до відповідних мікросервісів. Це спрощує керування трафіком і забезпечує єдину точку входу.

Автентифікація та авторизація: API Gateway забезпечує механізми автентифікації та авторизації, що дозволяють контролювати доступ до ваших мікросервісів. Ви можете встановити правила доступу, перевіряти токени або сертифікати, та налаштовувати політики доступу для різних ролей користувачів.

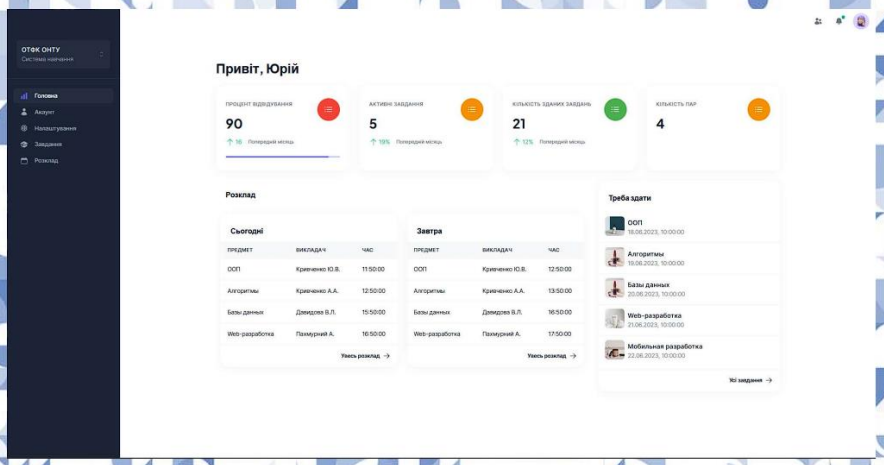
Масштабованість: API Gateway дозволяє масштабувати ваші мікросервіси незалежно один від одного. Ви можете додавати або зменшувати пропускну здатність для кожного сервісу окремо, що дозволяє гнучко реагувати на зміни в навантаженні.

Кешування: API Gateway може кешувати відповіді від мікросервісів, що дозволяє зменшити навантаження на них та покращити швидкодію відповідей. Кешування може бути налаштовано для окремих запитів або груп запитів залежно від їх характеристик.

Моніторинг та аналітика: API Gateway зазвичай надає функції моніторингу та аналітики трафіку. Ви можете відстежувати кількість запитів, час відповіді, помилки та інші метрики, що дозволяє вам виявити проблеми та оптимізувати продуктивність системи.

Етап 2: Написання клієнтської частини на React.js + Next.js

Другий етап передбачає розробку клієнтської частини за допомогою технологій React.js та Next.js. Ми створили інтерфейс користувача, який є зручним та ефективним



Головна сторінка нашої платформи «Електронний коледж»

Етап 3: Написання мікросервісів на .NET та Python

Третій етап включає розробку мікросервісів з використанням технологій .NET та Python. Ми створили різні мікросервіси для різних функціональних модулів та забезпечили їх взаємодію.

Основні функціональні можливості

Наш веб-додаток надає користувачам такі можливості, як доступ до розкладу занять, замін, завданням, статистики та оцінкам у зручному єдиному місці

Переваги веб-додатку для коледжу

Наш веб-додаток забезпечить зручний доступ до інформації для студентів, викладачів та адміністрації, покращить ефективність процесів навчання та оцінювання, а також забезпечить можливість швидкого оновлення та розширення функціональності.

Ризики та виклики

У процесі реалізації проекту ми зіткнулись з технічними проблемами, терміновими завданнями, а також потребою інтеграції з наявною інфраструктурою. Також ми забезпечили надійне шифрування даних та безпечність їх зберігання

Підсумок

У цій презентації ми оглянули проєкт створення веб-додатку для коледжу, звернули увагу на етапи проєктування мікросервісної архітектури, написання клієнтської частини на React.js + Next.js, а також написання мікросервісів на .NET та Python. Розглянули переваги, а також ризики та виклики, з якими ми стикнулись під час реалізації проекту.

UNICHECK

Ім'я користувача:
Наталія Вікторівна Копусь

ID перевірки:
1015554510

Дата перевірки:
12.06.2023 09:58:52 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
12.06.2023 10:04:48 EEST

ID користувача:
100011688

Назва документа: 4РП-06 Савінов В.С

Кількість сторінок: 54 Кількість слів: 9603 Кількість символів: 75561 Розмір файлу: 2.25 MB ID файлу: 1015206419

2.75%
Схожість

Найбільша схожість: 0.91% з Інтернет-джерелом (https://kafedra-aop.at.ua/news/canitarija_ta_gigiena_robochogo_misc...

2.75% Джерела з Інтернету | 406 ..... Сторінка 56

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи | 28

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти

відділення комп'ютерних систем

Савінов Владислав Сергійович

(прізвище, ім`я та по батькові)

Спеціальність 121 " Інженерія програмного забезпечення"

Освітня програма «Розробка програмного забезпечення»

Керівник дипломного проекту (роботи) Шувалова Ірина Олегівна

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) Проектування мікросервісної архітектури для веб-сайту ВСП «ОТФК ОНТУ»

Обсяг розрахунково-пояснювальної записки 70 сторінок

Обсяг графічної (презентаційної) частини 10 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню
Дипломний проект Владислава Савінова відповідає всім вимогам та стандартам проектування мікросервісної архітектури. Робота відображає важливість мікросервісного підходу у створенні сучасних та ефективних веб-сервісів.

б) характеристика виконання кожного розділу дипломного проекту (роботи)
Проект складається з декількох розділів, які детально розкривають теоретичні аспекти мікросервісів, методи проектування архітектури та практичну реалізацію на прикладі веб-сайту ВСП «ОТФК ОНТУ».

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) Розрахунково-пояснювальна записка виділяється глибоким аналізом, а графічна частина ефективно ілюструє проектні рішення і схеми мікросервісів.

г) перелік позитивних якостей дипломного проекту (роботи)

Позитивні якості проекту включають чітке розуміння принципів мікросервісної архітектури, ретельне проектування з врахуванням можливостей масштабування, а також забезпечення високої продуктивності та стабільності системи.

д) основні недоліки дипломного проекту (роботи)

Як недоліки можна відзначити відсутність докладного аналізу альтернативних технологій та підходів, які можна було б використовувати у контексті мікросервісної архітектури.

Оцінка розрахункової частини добре

Оцінка графічної частини добре

Загальна оцінка добре

Прізвище, ім'я, по батькові рецензента Василіу Євген Вікторович

Місце роботи і посада рецензента Державний університет інтелектуальних технологій і зв'язку, д.т.н., проф. кафедри КБ та ТЗІ, декан факультету інформаційних технологій та кібербезпеки

Підпис:

«16» 06 2023 р.

ПІДПИС ПОСВІДЧУЮ
НАЧАЛЬНИК ВІДДІЛУ КАДРІВ ДУІТЗ

ВІДДІЛ КАДРІВ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНАХТ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Савінов Владислав Сергійович

(прізвище, ім`я та по батькові)

Спеціальність: 121 - Інженерія програмного забезпечення

Освітня програма: Розробка програмного забезпечення

Тема дипломного проекту: Проектування мікросервісної архітектури для веб-сайту ВСП «ОТФК ОНТУ»

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки)

Пояснювальна записка була розроблена у відповідності з вимогами ДСТУ. Об'єм та суть роботи в повній мірі відображають завдання, що було поставлено для дипломного проектування. Розрахунково-пояснювальна частина проекту є добре структурованою та грамотно виконаною.

б) самостійність роботи над проектом:

Савінов В.С. успішно опрацював науковий матеріал, здійснив аналіз методів та підходів щодо створення веб-орієнтованих систем, продемонстрував здатність критично оцінювати літературні ресурси та ефективно застосовувати сучасні інформаційні технології у процесі моделювання та проведенні експериментальних досліджень.

в) теоретична підготовка випускника (випускниці):

Студент Савінов В.С. володіє глибокою теоретичною та практичною підготовкою на високому рівні. Він виявив зосередженість, старанність та відповідальний підхід до своєї роботи.

г) вміння розв'язувати виробничі та конструкторські питання _____

Студент Савінов В.С. за період роботи показал Високий рівень теоретичної та практичної підготовки

Оцінка розрахункової частини «Відмінно»

Оцінка графічної частини «Відмінно»

Загальна оцінка «Відмінно»

Прізвище, ім'я, по батькові керівника дипломного проекту _____

Шувалова Ірина Олегівна

Місце роботи і посада керівника дипломного проекту викладач

Підпис _____

«09» 06 2023 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Савінов Владислав Сергійович,
здобувач освіти гр. 4РП-06, та

Шувалова Ірина Олегівна,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи молодшого спеціаліста на тему:

«Проектування мікросервісної архітектури для веб-сайту ВСП «ОТФК ОНТУ»» (автор роботи – Савінов В.С., керівник роботи – Шувалова І.О.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2023 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи, і даємо згоду на обробку персональних даних.

Виконавець _____ / Савінов В.С. /

Керівник _____ / Шувалова І.О. /

«12» 06 2023 р.