

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

Сергій СТИРЕНКО

(підпис)

“ _ ” _____ 2022 р.

**Дипломний проєкт
на здобуття ступеня бакалавра
за освітньо-професійною програмою “Інженерія програмного
забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”**

на тему: «Програмний модуль контейнера мікросервісів»

Виконав : студент 4 курсу, групи ІП-84
(шифр групи)

Василиненко Даніїл Денисович

(прізвище, ім’я, по батькові)

(підпис)

Керівник доц., к.т.н., доц. Болдак А.О.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Консультант (нормоконтроль) проф., д. т. н., проф. Сімоненко В. П.

(назва розділу)

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Рецензент _____

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Засвідчую, що у цьому дипломному
проєкті немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____

(підпис)

Київ – 2022 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення
комп'ютерних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТИПЕНКО
(підпис)
“ ” _____ 20__ р.

ЗАВДАННЯ

на дипломний проект студенту

Василиненко Даніїл Денисович

1. Тема проекту «Програмний модуль контейнера мікросервісів»
керівник проекту Болдак А. О., доц., к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 11 травня 2022 року №1139-с
2. Термін здачі студентом закінченого роботи 2022р.
3. Вихідні дані до проекту: технічна документація, теоретичні та статистичні дані.
4. Зміст пояснювальної записки:
Розділ 1. Розподілені системи. Визначення термінології та огляд існуючих рішень
Розділ 2. Система управління. Проектування і вибір технологій.
Розділ 3. Реалізація модуля і тестування

5. Консультант роботи, з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В. П.		

6. Дата видачі завдання

КАЛЕНДАРНИЙ ПЛАН

п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту(роботи)	Примітки
1.	Затвердження теми роботи	10.12.2021-15.12.2021	
2.	Вивчення та аналіз завдання	15.12.2021-15.03.2022	
3.	Розробка архітектури та загальної структури системи	15.03.2022-25.03.2022	
4.	Розробка структур окремих підсистем	25.03.2022-05.04.2022	
5.	Програмна реалізація системи	05.04.2022-15.04.2022	
6.	Оформлення пояснювальної записки	15.04.2022-20.05.2022	
7.	Захист програмного продукту	15.05.2022	
8.	Перед захист	28.05.2022	
9.	Захист	2022	

Студент

Даніїл ВАСИЛИНЕНКО

(підпис)

Керівник

Андрій БОЛДАК

(підпис)

АНОТАЦІЯ

У даній дипломній роботі було розроблено систему програмного модуля контейнера мікросервісів розподіленої інформаційної системи, використовуючи необхідні готові рішення. Дана система дає змогу створювати та перевіряти коректність специфікацій, що призначені для опису розподілених інформаційних систем з мікросервісною архітектурою, передають конфігурації до програмних модулів, у яку буде інтегроване наше архітектурне рішення.

Дана система може бути використана для рішень на основі систем з мікросервісними архітектурами, це дасть змогу значно підвищити повторне використання програмного коду.

Ключові слова: специфікації, мікросервісна архітектура, Node.js, OpenAPI.

ANNOTATION

In this thesis, the system of the software module of the microservice container of the distributed information system was developed, using the necessary ready-made solutions. This system allows you to create and verify the correctness of specifications designed to describe distributed information systems with microservice architecture, transfer configurations to software modules, which will be integrated into our architectural solution.

This system can be used for solutions based on systems with microservice architectures, which will significantly increase the reuse of software code.

Keywords: specifications, microservice architecture, Node.js, OpenAPI.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

[illegible]

					ІАЛЦ.467200.001 ВП						
Зм.	Арк.	№ документна	Підпис	Дата	Програмний модуль контейнера мікросервісів Відомість дипломного проекту	Літ.	Аркуш	Аркушів			
Розробив		Василиненко Д. Д.									
Перевірив		Болдак А.О.					1	1			
Реценз.						НТУУ КПІ ім. Ігоря					
Н. Контр.		Сімоненко В.П.				Сікорського, ФІОТ, ІП-84					
Затверд.											

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Програмний модуль контейнера мікросервісів»

Київ – 2022

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ.....	2
2 ПІДСТАВИ ДЛЯ РОЗРОБКИ.....	2
3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ.....	2
4 ДЖЕРЕЛА РОЗРОБКИ.....	2
5 ТЕХНІЧНІ ВИМОГИ	3
5.1 Вимоги до розробленого продукту	3
5.2 Вимоги до програмного забезпечення.....	3
5.3 Вимоги до апаратної частини	3
6 ЕТАПИ РОЗРОБКИ.....	3

					ІАЛЦ.467200.002 ТЗ						
		№ докум.	Підпис	Дата							
Розробив		Василиненко Д. Д.			Програмний модуль контейнера мікросервісів Технічне завдання			Літ.	Аркуш	Аркушів	
Перевірив		Болдак А. О.								1	3
								НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ПІ-84			
Н. Контр.		Сімоненко В. П.									
Затвердив											

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється інтеграції в мікросервісну архітектуру, розробку системи програмного модулю контейнера мікросервісів, для того щоб мати можливість проводити налаштування мікросервісів за допомогою специфікацій.

Областю застосування цієї системи є налаштування конфігурацій контейнерів для їх координації у системі з мікросервісною архітектурою.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка системи, що дозволить конфігурувати контейнери мікросервісів та перевіряти специфікації на валідність.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки для даного дипломного проєкту є: науково-технічна література, книги, статті, з теорії і практики програмування в мережі Інтернет, офіційна документація.

					ІАЛЦ.467200.003 ПЗ	Арк.
						2
Зм.	Арк.	№ докум.	Підпис	Дата		

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Простий, інтуїтивно-зрозумілий інтерфейс системи.
- Надати користувачам формувати специфікації для контейнерів мікросервісів.
- Надати можливість інтегрувати дане рішення у власну мікросервісну архітектуру.
- Надати коректну інформацію щодо розгортання та запуску проєкту

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- Node packet manager
- Node js 12 або вище

5.3. Вимоги до апаратної частини

- Центральний процесор Intel або AMD.
- 1 ГБ вільного місця на жорсткому диску.
- Не менше 1 ГБ оперативної пам'яті.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.12.2021-15.12.2021
Вивчення та аналіз завдання	15.12.2021-15.03.2022
Розробка архітектури та загальної структури системи	15.03.2022-25.03.2021
Розробка структур окремих частин системи	25.03.2022-5.04.2022
Програмна реалізація системи	5.04.2022-15.04.2022
Виправлення помилок	15.04.2022-20.05.2022
Оформлення пояснювальної записки	25.05.2022

					ІАЛЦ.467200.003 ПЗ	Арк.
						3
Зм.	Арк.	№ докум.	Підпис	Дата		

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: *«Програмний модуль контейнера мікросервісів»*

Київ – 2022

					ІАЛЦ.467200.003 ПЗ	Арк.
						2
Зм.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ВСТУП.....	6
РОЗДІЛ 1. РОЗПОДІЛЕНІ СИСТЕМИ. ВИЗНАЧЕННЯ ТЕРМІНОЛОГІЇ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ	7
1.1 Розподілена система	7
1.1.1 Мікросервіс та мікросервісна архітектура.....	10
1.1.2 SOA архітектура	12
1.1.3 SOA vs Microservices....	Помилка! Закладку не визначено.
1.2 Огляд існуючих рішень.....	18
1.2.1 Amazon Web Services	18
1.2.2 Інструменти моніторингу	19
1.2.3 Prometheus.....	20
ВИСНОВКИ ДО РОЗДІЛУ 1	22
РОЗДІЛ 2. СИСТЕМА УПРАВЛІННЯ. ПРОЕКТУВАННЯ ТА ВИБІР ТЕХНОЛОГІЙ.	23
2.1 Загальний концепція системи	23
2.2 Взаємодія між компонентами	23
2.3 Комунікація у розподілених системах	23
2.3.1 Широкоадресна комунікація	26
2.3.2 Мультиадресна комунікація	27
2.3.3 Одноадресна комунікація	28
2.4 Передача даних між сервісами	29
2.4.1 HTTP Протокол	29
2.4.2 Брокер повідомлень	30
2.4.3 Використання gRPC у мікросервісній комунікації	31
2.5 КОНВЕРТАЦІЯ З ФОРМАТУ YAML У ФОРМАТ JSON	32
2.6 Специфікації опису систем оброблення даних з мікросервісною архітектурою	33
2.6.1 Загальні принципи опису процесу за допомогою специфікацій	33
2.6.2 Особливості використання специфікацій	33

2.7 Конфігурування робочих процесів за допомогою специфікацій	37
2.7.1 Базовий процес.....	37
2.7.2 Прослуховування декількома споживачами однієї черги повідомлень	38
2.7.3 Конфігурування мікросервісів з допомогою специфікацій.....	43
2.8 Процес взаємодії між компонентами системи.....	45
2.9 Вибір засобів проєктування	45
2.9.1 Javascript	45
2.9.2 NodeJS	47
2.9.3 OpenAPI	48
2.9.4 AMPQ.....	49
ВИСНОВКИ ДО РОЗДІЛУ 2	50
РОЗДІЛ 3. РЕАЛІЗАЦІЯ МОДУЛЯ І ТЕСТУВАННЯ	51
3.1 Структура проєкту.....	52
3.2 Огляд файлів у проєкті.....	53
3.3 Огляд основних компонентів системи.....	54
3.3.1 MSAPI	50
3.3.2 METADATA.....	53
3.3.3 COMPONENTS	53
3.3.4 EXCHANGES.....	53
3.3.5 QUEUES.....	54
3.3.6 SCHEMAS	53
3.3.7 CONSUMER.....	53
3.3.8 PRODUCES	53
3.3.9 EXCHANGE	53
3.3.11 QUEUE-OPTIONS.....	56
3.3.12 CONNECTION.....	53
3.3.10 EXCHANGE-OPTIONS.....	54
3.3.13 PRODUCE	53
3.3.14 SERVICE	53

3.3.15 WORKFLOW	60
3.4 Тестування Модуля	61
3.5 Демонстрація роботи програми	61
ВИСНОВОК ДО РОЗДІЛУ 3	67
ВИСНОВКИ	68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	70
ДОДАТОК 1.....	72
ДОДАТОК 2.....	74
ДОДАТОК 3.....	76
ДОДАТОК 4.....	78

ВСТУП

Розподілені системи є важливою розробкою для ІТ та інформатики, оскільки кількість пов'язаних робіт, що зростає, настільки масово і складно, що один комп'ютер не зможе впоратися з ними самотійно. Але розподілені обчислення також пропонують додаткові переваги проти традиційними обчислювальними середовищами. Розподілені системи знижують ризик єдиної точки відмови, підвищуючи надійність та стійкість. Сучасні розподілені системи зазвичай розраховані на масштабування майже як реального часу; Крім того, ви можете використовувати додаткові обчислювальні ресурси на льоту, підвищуючи продуктивність та ще більше скорочуючи час завершення. Історично розподілені обчислення були дорогими, їх складно було налаштувати та керувати важко. Але завдяки таким програмним платформам, як сервіси (SaaS), що пропонують розширені функції, розподілені обчислення стали більш упорядкованими та доступними для великих та малих компаній. В результаті всі види комп'ютерних завдань, від управління базою даних до відеоігор використовують розподілені обчислення. Насправді багато типів програмного забезпечення, такі як системи криптовалюти, наукове моделювання, технології блокчейн і платформи штучного інтелекту, були б неможливі без цих платформ.

					ІАЛЦ.467200.003 ПЗ	Арк.
						6
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1.

РОЗПОДІЛЕНІ СИСТЕМИ. ВИЗНАЧЕННЯ ТЕРМІНОЛОГІЇ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Розподілена система

Більшість проблем, які вирішують використання розподілених програмних комплексів, а також обмеження, з якими постійно стикається таке використання, можна зрозуміти, розглянувши еволюційний розвиток методів розподіленої обробки інформації. Необхідно усвідомити, що, незважаючи на те, що технології програмістів та користувачів їх програм змінилися, проблеми залишаються такими ж, як у минулому. Вивчення різних методів програмного забезпечення для розподілених систем вестиметься в хронологічному порядку від перших систем, заснованих безпосередньо на методі віддаленого виклику процедур до найсучасніших форм підтримки системи - мережевих сервісів.

Розподілена система[1] – це набір незалежних комп'ютерів, що представляються їх користувачам єдиною об'єднаною системою (визначення вільне, але придатне).

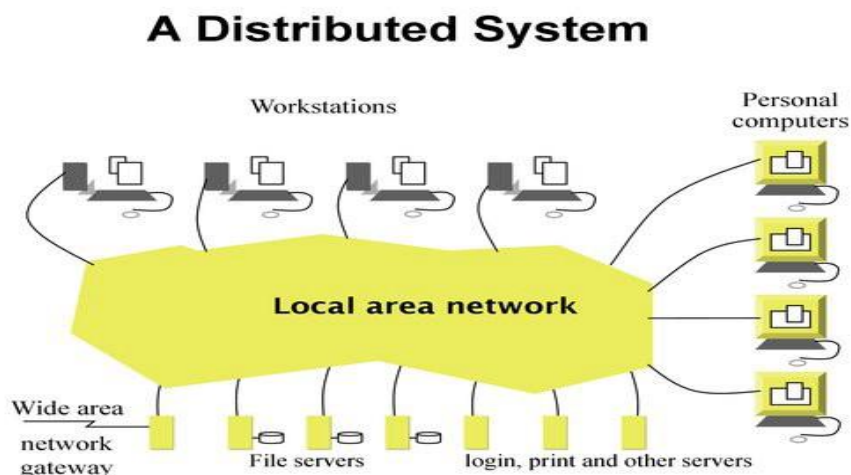


Рис.1.1 Приклад розподіленої системи[2]

Відмінності між комп'ютерами та способами їх приховування від користувачів (навіть той факт, що може бути тільки один комп'ютер,

прихований від користувача), принаймні комп'ютери в системі розповсюдження є автономними. Користувачі та програми працюють разом у розподілених системах, незалежно від того, де і коли вони взаємодіють.

Комп'ютерну систему, що складається з багатьох різних комп'ютерів з різним програмним забезпеченням, можна назвати розподіленою лише в тому випадку, якщо вона виглядає як класична однопроцесорна система з розподілом часу для своїх користувачів. Він часто включає додатковий рівень програмного забезпечення для підтримки представлення різних комп'ютерів і комп'ютерних мереж як єдиної системи організованих систем. Цей рівень називається рівнем системної підтримки (middleware).

До основних переваг розподілених систем відносять:

- Спільне використання ресурсів. Розподілені системи дозволяють спільно використовувати апаратні та програмні ресурси, наприклад, жорсткі диски, принтери, файли, компілятори тощо, об'єднані засобами мережі. Очевидно, що розподіл ресурсів можливий і в розрахованих на багато користувачів системах, але в цьому випадку при наданні ресурсів і їх управлінні повинен керувати центральний процесор. Це можливість розширювати систему шляхом додавання нових ресурсів.
- Розподілені системи – це відкриті системи, до яких підключаються апаратне та програмне забезпечення різних виробників.
- Паралельність. У розподілених системах кілька процесів можуть виконуватись одночасно на різних комп'ютерах у мережі. Ці процеси можуть взаємодіяти один з одним у разі їх виконання.
- Масштабованість. У принципі, всі розподілені системи масштабуються: щоб система відповідала новим вимогам, її можна нарощувати за допомогою додавання нових обчислювальних ресурсів. Але на практиці нарощування може обмежуватися мережею, що об'єднує окремі комп'ютери системи. Якщо підключити багато нових машин, то пропускна здатність мережі може бути недостатньою.

					ІАЛЦ.467200.003 ПЗ	Арк.
						8
Зм.	Арк.	№ докум.	Підпис	Дата		

- Відмовостійкість. Наявність декількох комп'ютерів та можливість дублювання інформації означає, що розподілені системи стійкі до певних апаратних та програмних помилок. Більшість розподілених систем у разі помилки, як правило, можуть підтримувати хоча б частково функціональність. Повний збій у системі відбувається лише при помилках мережі.

До основних недоліків відносять:

- Складність[4]. Розподілені системи складніші за централізовані. Набагато складніше зрозуміти та оцінити властивості розподілених систем, а також тестувати ці системи. Наприклад, продуктивність системи залежить від швидкості роботи одного процесора, а від смуги пропускання мережі та швидкодії різних процесорів. Переміщуючи ресурси з однієї частини мережі до іншої, можемо радикально вплинути на продуктивність системи.
- Безпека. Як правило, доступ до системи можна отримати з різних машин, повідомлення в мережі можна переглядати чи перехоплювати. Тому у розподіленій системі набагато складніше підтримувати безпеку.
- Непередбачуваність. Як відомо всім користувачам Web-мережі, реакція розподілених систем на певні події непередбачувана і залежить від повного завантаження системи, її організації та навантаження. Оскільки всі ці параметри можуть постійно змінюватися, час, витрачений виконання запиту користувача, у той чи інший момент може істотно відрізнятись.

Отже, основне завдання розподілених систем програмного забезпечення – полегшити їх користувачам доступ до віддалених ресурсів, а також контролювати спільне використання цих ресурсів. Вони мають свої переваги і недоліки. Ресурси можуть бути віртуальними, але можуть бути і традиційними – комп'ютерами, принтерами, пристроями зберігання файлів, файлами і даними. В якості основних вимог до розподілених систем пред'являється

					ІАЛЦ.467200.003 ПЗ	Арк.
						9
Зм.	Арк.	№ докум.	Підпис	Дата		

досягнення їх прозорості, відкритості і масштабованості.

1.1.1 Мікросервіс та мікросервісна архітектура

Мікросервіс[4] - сервіс, що має дрібну гранулярність та має легкий (в сенсі мережевого Інтернет навантаження) протокол спілкування. Також характеристиками даного типу сервісів є:

- мікросервіси – це процеси, які взаємодіють через мережу, використовуючи протоколи, такі як HTTP;
- розгортаються незалежно один від одного; проектуються та розробляються відповідно до певних бізнес-вимогами та бізнес-можливостями; можуть бути розроблені, використовуючи різні мови програмування, бази даних і можуть бути розгорнуті в різних середовищах оточення;
- розробляються відносно невеликими командами із 5-12 людей.
- маючи такі характеристики, мікросервіси ідеально підходять під принципи CD та CI, оскільки зміни малої частини кодової бази не призводять до повному перерозгортання програми, а лише його невеликої частини.

Стрімкий розвиток і поширення мережевих хмарних сервісів на початку 2010-х років призвело до розчарування в класичному, так званому монолітному варіанті архітектури додатків. Через складність окремих модулів, які часто представляють собою цілі програмні системи, а також через необхідність забезпечувати сумісність між ними за допомогою стандартних протоколів, внесення будь-яких змін і доповнень стало нетривіальним завданням, що забирає занадто багато часу. В якості відповіді на цей виклик була запропонована архітектура мікросервісів як розподілена система найпростіших і легкозамінних модулів, що виконують по можливості єдину елементарну функцію. При цьому мікросервісна система має симетричну, однорангову, а не ієрархічну організацію, що знімає необхідність у складній організації взаємозв'язків. Сервіси зв'язуються між собою і з клієнтами з

					ІАЛЦ.467200.003 ПЗ	Арк.
						10
Зм.	Арк.	№ докум.	Підпис	Дата		

використанням легких протоколів, наприклад, через HTTP або текстовими повідомленнями. В результаті створюється система, проста в розгортанні і модернізації, з функціями автоматичної розробки та оновлення. До 2021 року мікросервісна архітектура в центрі уваги, причому, не тільки фахівців: про неї пишуть в блогах, в соцмережах, обговорюють в пресі і на різних конференціях.

При цьому у співтоваристві програмістів все частіше звучать голоси скептиків, які не вважають мікросервіси чимось принципово новим. На їхню думку, це просто реалізація сервіс-орієнтованої архітектури (SOA) на більш низькому рівні. Як би там не було, мікросервісна архітектура має очевидні переваги, особливо в сфері Agile-розробки і розгортання складних додатків корпоративного рівня.

З точки зору бізнесу і чисто організаційних завдань, переваги мікросервісів зводяться до трьох основних:

- легкість оновлення коду;
- різні команди можуть використовувати різні стеки для різних модулів;
- компоненти можуть масштабуватися незалежно один від одного, що знижує витрати і вартість масштабування всього додатка в цілому в тих випадках, якщо вузьким місцем виступає лише якась одна функція.

					ІАЛЦ.467200.003 ПЗ	Арк.
						11
Зм.	Арк.	№ докум.	Підпис	Дата		

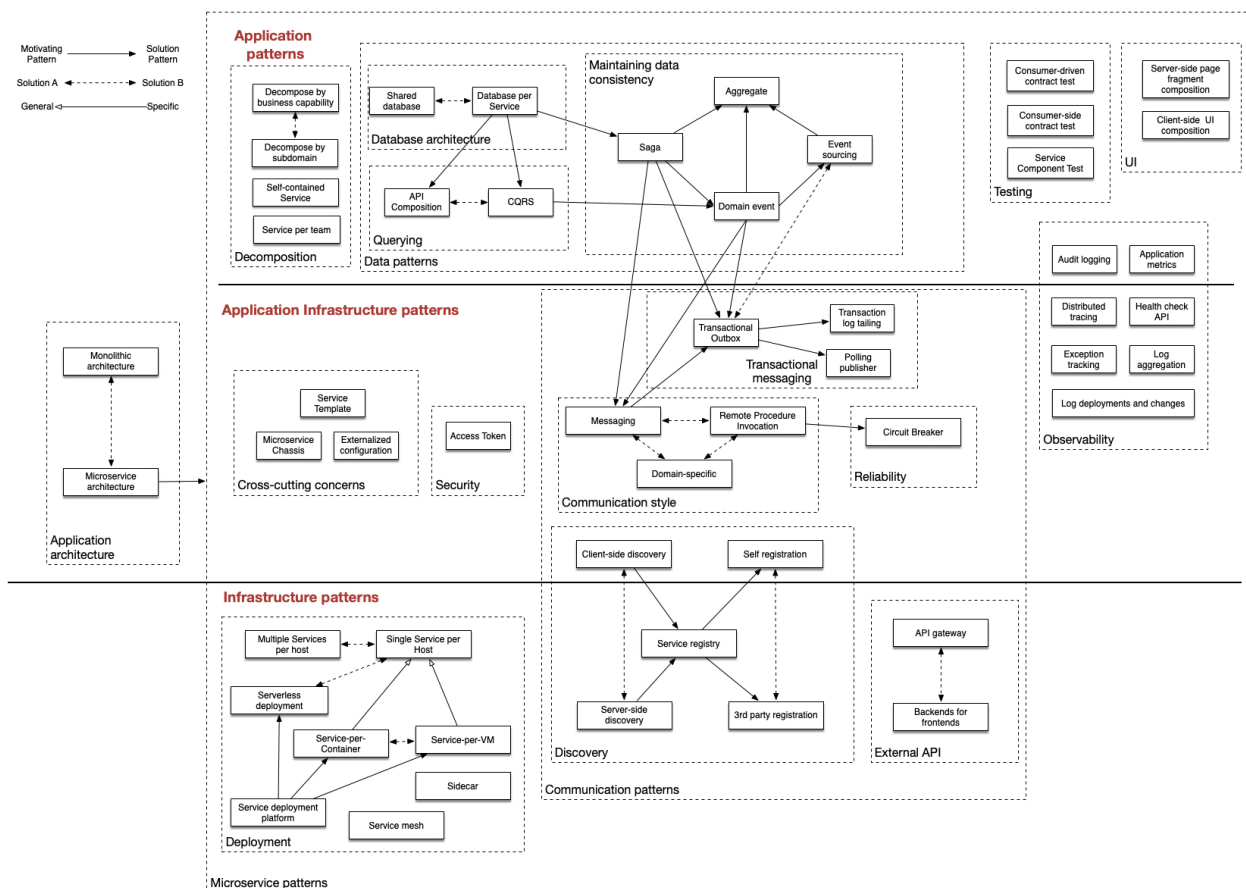


Рис.1.2 – «Множина шаблонів проектування мікросервісної архітектури»

1.1.2 SOA архітектура

SOA[5], або сервісно-орієнтована архітектура, визначає спосіб зробити програмні компоненти повторно використовуваними та сумісними через сервісні інтерфейси. Служби використовують загальні стандарти інтерфейсу та архітектурний шаблон, щоб їх можна було швидко включити в нові програми. Це видаляє завдання від розробника програми, який раніше переробляв або дублював існуючі функції або мав знати, як підключити чи забезпечити взаємодію з наявними функціями.

Кожен сервіс в SOA містить код і дані, необхідні для виконання повної окремої бізнес-функції (наприклад, перевірка кредиту клієнта, розрахунок щомісячного платежу по кредиту або обробка заявки на іпотеку). Інтерфейси служб забезпечують слабке зацеплення, тобто їх можна викликати, не знаючи або зовсім не знаючи, як реалізується сервіс, що зменшує залежності між

Зм.	Арк.	№ докум.	Підпис	Дата

додатками.

Сервіси доступні за допомогою стандартних мережевих протоколів, таких як SOAP (протокол доступу до простого об'єкта)/HTTP або Restful HTTP (JSON/HTTP) — для надсилання запитів на читання або зміну даних. Керування послугами контролює життєвий цикл розробки, і на відповідному етапі служби публікуються в реєстрі, що дозволяє розробникам швидко знаходити їх і повторно використовувати для створення нових програм або бізнес-процесів.

Таким чином, SOA представляє важливий етап в еволюції розробки та інтеграції додатків за останні кілька десятиліть. До того, як наприкінці 1990-х з'явилася SOA, підключення програми до даних або функцій, розміщених в іншій системі, вимагало складної інтеграції «точка-точка» — інтеграції, яку розробники повинні були відтворювати, частково або повністю, для кожного нового проекту розробки. Надання цих функцій через служби SOA дозволило розробнику просто повторно використати наявні можливості та підключитися через архітектуру SOA.

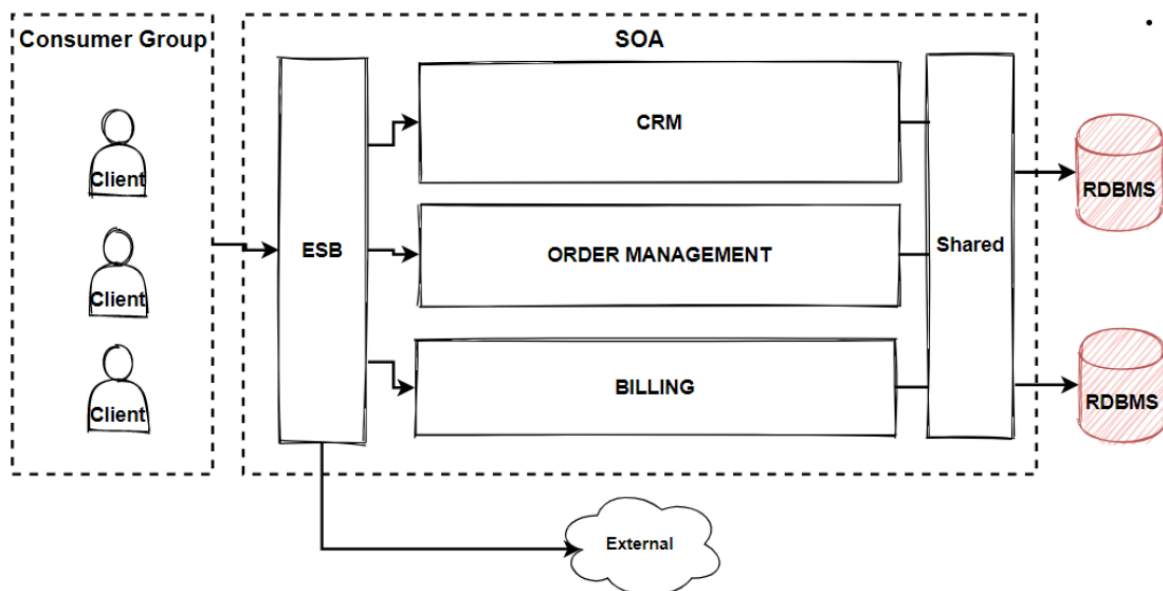


Рис. 1.3 Приклад Service-Oriented Architecture[6]

Основні переваги SOA архітектури[5]:

- Значна гнучкість у розробці;

- Ефективність збирання додатків із сервісів багаторазового використання, тобто будівельних блоків, а не переписування та реінтеграції з кожним новим проектом розробки, дозволяє розробникам створювати програми набагато швидше, реагуючи на нові бізнес-можливості.
- Можливість використовувати застарілі функції на нових ринках: добре розроблений SOA дозволяє розробникам легко використовувати функціональні можливості, «заблоковані» в одній комп'ютерній платформі або середовищі, і поширювати їх на нові середовища та ринки. Наприклад, багато компаній використовували SOA, щоб надати функціональність фінансових систем на базі мейнфрейма новим веб-додаткам, дозволяючи своїм клієнтам обслуговувати процеси та інформацію, до яких раніше був доступ лише через пряму взаємодію з співробітниками компанії або бізнес-партнерами.
- Покращена співпраця між бізнесом та ІТ: у SOA послуги можна визначити в бізнес-термінах (наприклад, «генерувати страхову пропозицію» або «розрахувати рентабельність капітального обладнання»). Це дає змогу бізнес-аналітикам ефективніше працювати з розробниками над важливими знаннями, такими як обсяг бізнес-процесу, визначений за допомогою послуг, або бізнес-наслідки зміни процесу, що може призвести до кращого результату.

Основні недоліки:

- У SOA всі введені дані перевіряються перед відправкою до сервіса. Якщо ви використовуєте кілька сервісів, це перевантажить вашу систему додатковими обчисленнями;
- Оскільки деякі сервіси часто надсилають і отримують повідомлення та інформацію, вони легко досягають мільйона запитів на день. Таким чином, для запуску сервіса використовується високошвидкісний сервер із великою пропускнуою здатністю даних;

					ІАЛЦ.467200.003 ПЗ	Арк.
						14
Зм.	Арк.	№ докум.	Підпис	Дата		

1.1.3 SOA vs Microservices

	Мікросервіси	SOA
Архітектура	Призначена для розміщення служб, які можуть функціонувати незалежно.	Розроблена для спільного використання ресурсів між сервіс
Спільний доступ до сервісів	Зазвичай не передбачає спільне використання сервісів.	Часто передбачає спільне використання сервісів.
Гранулярність	Висока ступінь гранулярності	Великі, більш модульні сервіси
Зберігання даних	Кожна сервіс може мати незалежне сховище даних.	Включає спільне зберігання даних між сервісами.
Розмір і обсяг	Краще для невеликих і веб-додатків	Краще для великомасштабної інтеграції
Передача даних	Використовує REST і JMS	Використовує такі протоколи, як SOAP і AMQP
Розгортання	Швидке та просте розгортання	Менша гнучкість у розгортанні

Таблиця 1.1 Порівняльна характеристика архітектур SOA та Microservices

- Архітектура.

Архітектура мікросервісів заснована на менших, детальних сервісах, які зосереджені на одній цілі і можуть функціонувати незалежно один від одного, але взаємодіяти для підтримки однієї програми. Отже, мікросервіси спроектовані так, щоб поділитися якомога меншою кількістю сервісних ресурсів. Оскільки SOA має більші, більш модульні служби, які не є незалежними одна від одної, вона розроблена для максимального спільного використання ресурсів.

- Спільний доступ до компонентів

Незалежність мікросервісів мінімізує необхідність спільного використання компонентів і робить служби більш стійкими до збоїв. Крім того, відносна відсутність спільного використання компонентів дозволяє розробникам легко розгортати нові версії та масштабувати окремі послуги набагато швидше, ніж із SOA. З іншого боку, спільне використання компонентів набагато частіше зустрічається в SOA. Зокрема, служби спільного доступу до ESB. Таким чином, якщо є проблеми з однією послугою щодо ESB, це може поставити під загрозу ефективність інших підключених служб.

- Гранулярність

Сервіси SOA великі, деякі модульні служби нагадують монолітні програми. Завдяки можливості масштабування кожної служби, SOA зазвичай мають більш широкий діапазон фокусу. Більш детальний характер мікросервісів означає, що окремі служби відмінно виконують одне конкретне завдання. Поєднання цих завдань призводить до створення однієї програми.

- Зберігання даних

З мікросервісами окремі служби зазвичай мають власне сховище даних. З SOA майже всі служби використовують одні й ті самі одиниці зберігання даних. Спільне використання одного сховища даних дозволяє

сервісам SOA повторно використовувати спільні дані. Ця можливість корисна для максимізації цінності даних шляхом розгортання одних і тих самих даних або програм між бізнес-підрозділами. Однак ця можливість також призводить до тісного зв'язку та взаємозалежності між сервісами.

- Розмір і сфера застосування

Розмір і область дії є однією з найбільш помітних відмінностей між мікросервісами та SOA. Тонка природа мікросервісів значно зменшує розмір і масштаби проектів, для яких вони розгорнуті. Його відносно невеликий обсяг послуг добре підходить для розробників. На відміну від цього, більший розмір і обсяг SOA краще для складніших інтеграцій різноманітних сервісів. SOA може підключати служби для міжпідприємницької співпраці та інших масштабних інтеграційних заходів.

- Передача даних

SOA і мікросервіси використовують різні протоколи для доступу до віддалених служб. Основні протоколи віддаленого доступу для SOA включають протокол Simple Object Access Protocol (SOAP) і обмін повідомленнями, такі як Advanced Messaging Queuing Protocol (AMQP) і Microsoft Messaging Queuing (MSMQ).

Найпоширенішими протоколами для мікросервісів є Representational State Transfers (REST) і простий обмін повідомленнями, такі як Java Messaging Service (JMS). Протоколи REST часто використовуються з API. Протоколи для мікросервісів є більш однорідними, ніж протоколи для SOA, які зазвичай використовуються для неоднорідної взаємодії.

- Розгортання

Простота розгортання є ще однією важливою відмінністю між мікросервісами та SOA. Оскільки сервіси в мікросервісах менші та значною мірою незалежні один від одного, вони розгортаються набагато

					ІАЛЦ.467200.003 ПЗ	Арк.
						17
Зм.	Арк.	№ докум.	Підпис	Дата		

швидше та легше, ніж служби SOA. Ці фактори також полегшують створення сервісів у мікросервісах.

Розгортання SOA ускладнюється тим фактом, що додавання служби передбачає відтворення та повторне розгортання всієї програми, оскільки служби поєднані разом.

Є кілька моментів, які слід враховувати, вирішуючи, чи краще використовувати мікросервіси чи SOA для конкретного бізнесу. SOA — це модульний засіб розбиття монолітних додатків на більш дрібні компоненти, тоді як мікросервіси забезпечують менший, більш детальний підхід до досягнення тієї ж мети. Обидві ці архітектури регулярно запускаються в хмарі, що підвищує гнучкість створення та розгортання програм. Зрештою, найкращий підхід залежить від власних унікальних потреб і варіантів використання кожного бізнесу.

1.2 Огляд існуючих рішень

1.2.1 Amazon Web Services

Веб-сервіс Amazon — це платформа, яка забезпечує гнучку, надійну, елегантну, ефективну та зручну платформу хмарних обчислень. AWS має величезну колекцію облачних сховищ, що створює цілу платформу. Це відомо для створення потужних репозиторій, баз даних, аналітиків, мережових та розробок. AWS Cloud є в 16 різних регіонах.

Основні переваги AWS:

- AWS пропонує простий процес розгортання програми
- Веб-сервіс AWS — ідеальний варіант, коли вашому проекту потрібна висока обчислювальна потужність
- Набір автоматизованих функцій, включаючи конфігурацію, масштабування, налаштування та інші
- Використовуючи AWS, ви маєте повний контроль над ресурсами.
- AWS дозволяє організаціям використовувати вже знайомі моделі програмування, операційні системи, бази даних та архітектури.

					ІАЛЦ.467200.003 ПЗ	Арк.
						18
Зм.	Арк.	№ докум.	Підпис	Дата		

- Це економічно ефективна послуга, яка дозволяє платити лише за те, що ви використовуєте, без будь-яких попередніх або довгострокових зобов'язань.
- Вам не потрібно буде витрачати гроші на роботу та обслуговування центрів обробки даних.
- Пропонує швидке розгортання
- Ви можете легко додати або видалити ємність.
- Вам надається швидкий доступ до хмари з необмеженою ємністю.

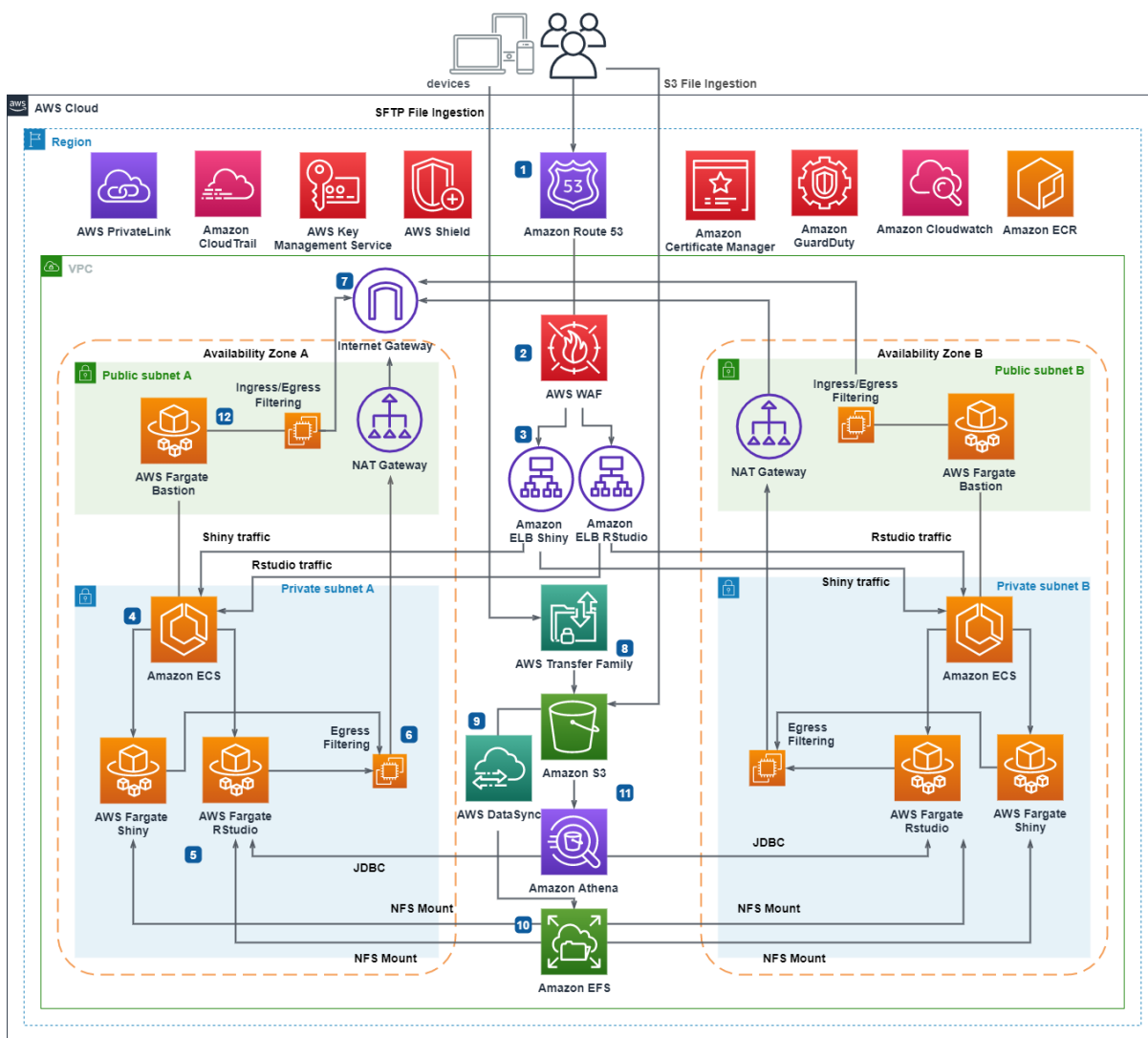


Рис 1.6 Архітектура AWS та елементи його еко-системи[7]

Основні недоліки AWS:

- Процес розгортання AWS непростий і дуже тривалий, який може зайняти від 15 до 20 хвилин для простого веб-додатку

Зм.	Арк.	№ докум.	Підпис	Дата

ІАЛЦ.467200.003 ПЗ

Арк.

19

- AWS не є ідеальним варіантом для новачків, які не володіють технікою
- Вам потрібно розгорнути програму самостійно за допомогою рецептів, Capistrano або вручну.

1.2.2 Інструменти моніторингу

Інструменти моніторингу мають вирішальне значення не лише для спостережливості та моніторингу, а й для відстеження та усунення проблем із здоров'ям системи та вузьких місць у ресурсах. Знайти найкращий варіант означає порівнювати не лише функції, а й те, як організація буде використовувати інструмент.

1.2.3 Prometheus

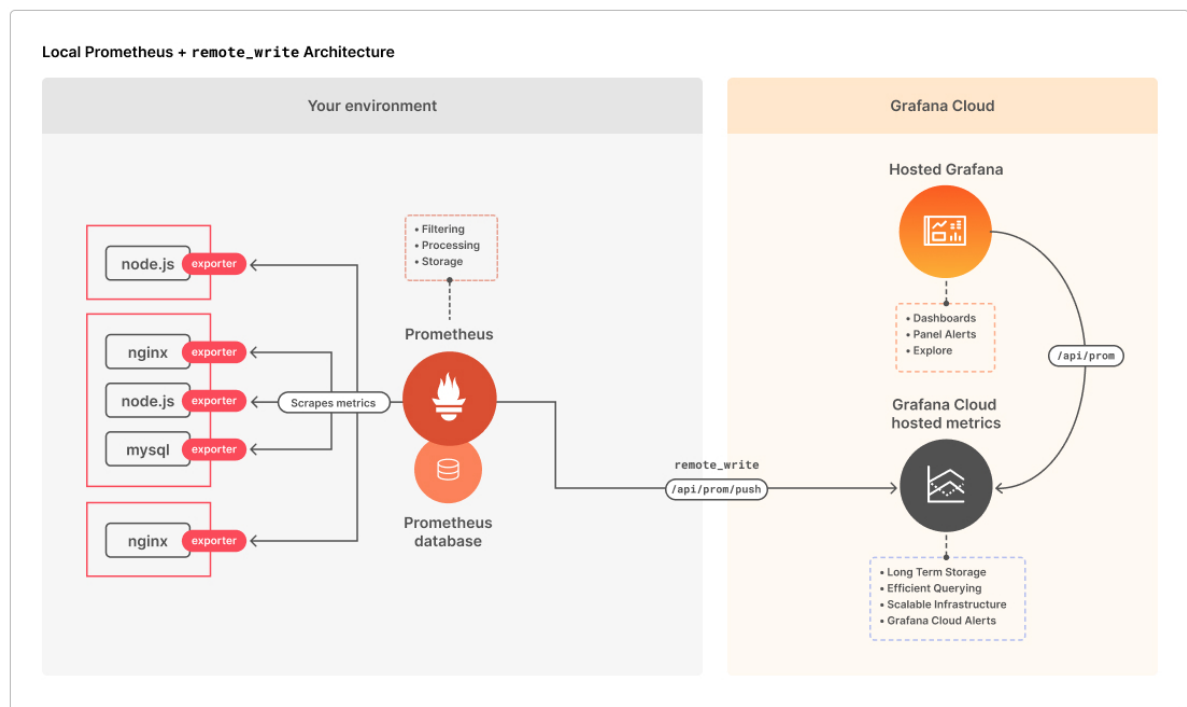


Рис 1.7 Архітектура Prometheus та додаткові елементи що можна включити в його системи[8]

Prometheus записує метрики серії в реальному часі за допомогою методу HTTP pull в базу даних часових рядів. Він також надає багатовимірну модель даних і гнучку мову запитів.

Для моніторингу додатків і служб у Prometheus потрібні додаткові інструменти в коді, використовуючи одну з клієнтських бібліотек Prometheus,

які реалізують різні типи підтримуваних показників. Популярні бібліотеки написані на Go, Java або Scala, Python і Ruby. Використовуючи одну з клієнтських бібліотек, написану рідною мовою програмування, розробники можуть визначати метрики та показувати їх у своєму екземплярі програми через кінцеві точки HTTP.

Плюси Prometheus включають:

- Інтеграція з Grafana. Це забезпечує синергію високоякісної колекції показників і багатофункціональних візуалізацій.
- PromQL. Потужна мова запитів для отримання та аналізу даних метрик Prometheus.

Мінуси Prometheus включають:

- Без тривалого зберігання. Він не має надійного довгострокового зберігання, яке може масштабувати кількість поглинених журналів. Користувачі повинні підтримувати резерв для зберігання журналів Prometheus і керувати його ємністю в міру зростання журналів.

					ІАЛЦ.467200.003 ПЗ	Арк.
						21
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ ДО РОЗДІЛУ 1

В ході написання даного розділу були розглянуті визначення розподіленої системи, існуючі архітектурні рішення. Було визначено, що існує два основних архітектурних підходи для вирішення задачі з побудови розподіленої системи: SOA та мікросервісна архітектура. Було проведено порівняння через основні параметри даних архітектур та складена порівняльна таблиця і характеристика. Для розгортання системи було розглянуто платформу AWS, яка надає можливість також проводити додаткові вимірювання продуктивності роботи системи, хоча рішення, що пропонує ця платформа піднімає вартість обслуговування. Також було розглянуто платформи Prometheus та Grafana, які будуть обслуговувати метрики та відображати показники системи.

					ІАЛЦ.467200.003 ПЗ	Арк.
						22
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2.

СИСТЕМА УПРАВЛІННЯ. ПРОЕКТУВАННЯ ТА ВИБІР ТЕХНОЛОГІЙ

2.1 Загальна концепція системи

Майбутня програмна система має на своїй меті управління контейнерами мікросервісів. Система дозволяє налаштовувати контейнери мікросервісів за допомогою специфікацій для об'єднання останніх у ланцюги, для підвищення їх функціональних можливостей та для підвищення відсотку повторного використання сервісів. Для проектування даної системи у цьому розділі будуть розглянуті принципи та технології, що дозволяють ці принципи реалізувати.

2.2 Взаємодія між компонентами

На даний момент є два основні архітектурні рішення:

Централізована мережа[11] - побудована навколо єдиного центрального сервера, який виконує всі основні функції управління та обробки даних. Інші типи серверів можуть підключатися до цього головного сервера та керувати іншими специфічними функціями, але ці інші сервери не можуть працювати незалежно від центрального сервера.

Клієнтські системи та користувачі не можуть отримати прямий доступ до ресурсів або служб на різних серверах без попереднього проходження через централізований головний сервер. Якщо центральний сервер вийде з ладу, разом з ним вийде вся мережа. Прикладом централізованої мережі може бути малий бізнес, який використовує єдиний контролер домену (DC). Вони можуть мати окремий сервер бази даних або сервер друку, але цей сервер покладається на DC для синхронізації годинника, керування ідентифікацією та аутентифікації, DHCP та інших життєво важливих мережевих служб.

					ІАЛЦ.467200.003 ПЗ	Арк.
						23
Зм.	Арк.	№ докум.	Підпис	Дата		

З основних переваг:

- Єдиний центральний сервер швидко й легко розгорнути, оскільки вам потрібно керувати лише однією конфігурацією без балансування навантаження чи оркестрування.
- Ви можете легко додавати та видаляти клієнтські системи, користувачів та інші сервери, не чекаючи реплікації між децентралізованими або розподіленими серверами.
- Централізовані мережі є відносно недорогими, оскільки ви використовуєте обмежену кількість серверів, що означає купівлю менше обладнання та менше ліцензій.

Але є й недоліки:

- Один головний сервер представляє єдину точку збою в мережі. Якщо головний сервер виходить з ладу через помилку чи атаку, або якщо вам потрібно перезапустити сервер для обслуговування, вся ваша мережа вийде з ладу.
- Усі ваші цінні та конфіденційні дані зберігаються та доступні з одного сервера в централізованій мережі, що становить загрозу безпеці. Якщо хакери проникнуть у ваш єдиний DC, вони зможуть отримати доступ до всього з цього одного місця, замість того, щоб перейти до різних систем і серверів, щоб знайти все, що їм потрібно.
- Централізовані мережі важко масштабувати, оскільки існує обмеження на кількість обчислювальної потужності, яку ви можете додати до одного сервера. Центральний сервер також може створювати вузькі місця, коли ваш мережевий трафік збільшується за межі обмежень одного вузла.

Архітектура децентралізованої[11] мережі використовує декілька серверів замість одного централізованого сервера. Кожен із цих серверів може діяти як незалежний головний сервер з необхідними робочими навантаженнями,

					ІАЛЦ.467200.003 ПЗ	Арк.
						24
Зм.	Арк.	№ докум.	Підпис	Дата		

розподіленими між ними для балансування навантаження. Якщо один сервер виходить з ладу, інший сервер може взяти на себе його навантаження, щоб мінімізувати переривання мережі.

Децентралізоване керування мережею не розподіляє рівномірно або автоматично навантаження мережі між кластером головних серверів, а натомість розподіляє навантаження відповідно до параметрів, визначених адміністратором мережі. Прикладом децентралізованої мережі може бути малий або середній бізнес (SMB), який використовує три контролери домену. Кожен контролер домену надає однакові важливі мережеві послуги, але бізнес розділив свої клієнтські системи та користувачів між трьома контролерами домена, тому кожен обробляє лише приблизно 1/3 загального навантаження. Якщо один контролер домену виходить з ладу, його навантаження буде тимчасово перенаправлено на інший, доки проблему не буде вирішено.

З основних переваг[11]:

- Децентралізовані мережі є більш надійними, ніж централізовані, тому що є кілька точок збою.
- Децентралізоване керування мережею масштабується ефективніше, оскільки дозволяє додавати більше серверів до кластера в міру зростання вашого бізнесу.
- У децентралізованій мережі менше вузьких місць, оскільки ця модель може розподіляти трафік між кількома серверами, і ці сервери також можна розгорнути у філіалах і центрах обробки даних.

Недоліки:

- Розгортання децентралізованих мереж дорожче та вимагає багато часу, оскільки вам потрібно встановити та налаштувати декілька серверів із можливостями балансування навантаження та перемикання збоїв.

					ІАЛЦ.467200.003 ПЗ	Арк.
						25
Зм.	Арк.	№ докум.	Підпис	Дата		

- Децентралізовані мережі вимагають реплікації та координації між кількома серверами, і будь-які помилки або перерви в цьому процесі можуть спричинити проблеми з безпекою та збої в роботі служби.
- Хоча децентралізована мережа є більш безпечною, ніж централізована, оскільки існує більше точок збою, реплікація між головними серверами створює ймовірність, що хакери все ще зможуть отримати доступ до більшості або всієї вашої мережі з одного місця.

2.3 Комунікація у розподілених системах

Зв'язок між двома процесами в розподіленій системі необхідний для обміну різними даними, такими як код або файл, між процесами. Коли один вихідний процес намагається зв'язатися з кількома процесами одночасно, це називається груповим зв'язком. Група – це сукупність взаємопов'язаних процесів з абстракцією. Ця абстракція має приховати передачу повідомлення, щоб спілкування виглядало як звичайний виклик процедури. Групове спілкування також допомагає процесам з різних хостів працювати разом і виконувати операції синхронізовано, отже, підвищує загальну продуктивність системи.

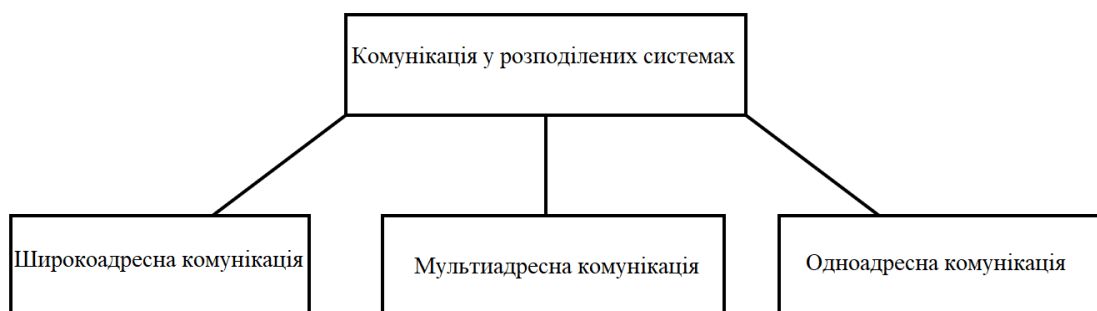


Рис. 2.1 Типи комунікації у розподілених системах

2.3.1 Широкоадресна комунікація:

Коли головний процес намагається спілкуватися з кожним процесом у розподіленій системі одночасно. Трансляційне спілкування[12] стає в нагоді,

коли загальний потік інформації має бути доставлений до кожного процесу найбільш ефективним способом. Оскільки він не потребує будь-якої обробки, спілкування дуже швидке в порівнянні з іншими способами спілкування. Однак він не підтримує велику кількість процесів і не може розглядати конкретний процес окремо.

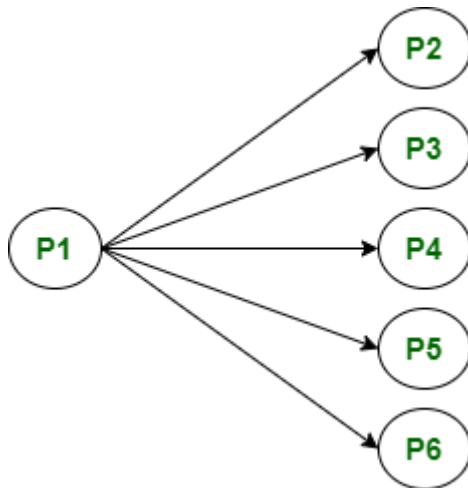


Рис. 2.2 Приклад широкоадресної комунікації[13]

2.3.2 Мультиадресна комунікація:

Коли головний процес намагається одночасно зв'язатися з визначеною групою процесів у розподіленій системі. Цей метод[12] в основному використовується для пошуку способу вирішення проблеми високого навантаження на хост-систему та надлишкової інформації з процесу в системі. Багатозадачність може значно скоротити час на обробку повідомлень.

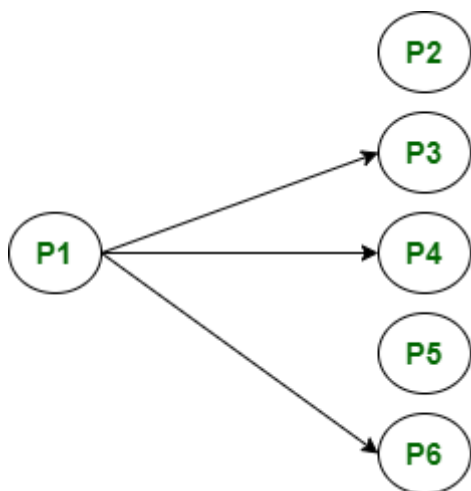


Рис. 2.3 Приклад мультиадресної комунікації[14]

2.3.3 Одноадресна комунікація:

Коли головний процес намагається спілкуватися з одним процесом у розподіленій системі одночасно. Хоча одна і та ж інформація може передаватися кільком процесам[12]. Це найкраще працює для двох процесів, які взаємодіють, оскільки лише він має розглядати лише певний процес. Однак це призводить до накладних витрат, оскільки потрібно знайти точний процес, а потім обмінюватися інформацією/даними.

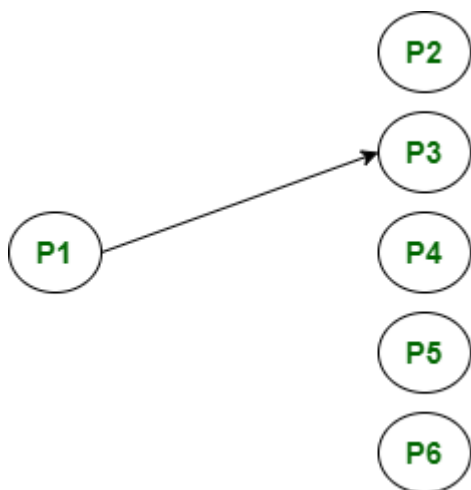


Рис. 2.4 Приклад одноадресної комунікації[15]

2.3 Передача даних між сервісами

2.3.1 HTTP Протокол

Лідером при виборі способу взаємодії служб один з одним є HTTP.

Фактично, ми могли б стверджувати, що всі канали комунікації походять від цього. Але, відкидаючи це в сторону, виклики HTTP між сервісами є життєздатним варіантом для зв'язку між послугами. Це може виглядати приблизно так, якщо в нашій архітектурі є два сервіси. ServiceA може обробити запит і викликати ServiceB, щоб отримати іншу інформацію. Але ми маємо робити асинхронні виклики HTTP між двома службами. Бо інакше дана модель створює зв'язок між двома службами, який нам, ймовірно, не потрібен. Завдяки такому підходу ми тримаємо послуги ізольованими один від одного, а зв'язок слабка. Недоліком є те, що він створює додаткові HTTP-запити до другого сервісу; тепер він буде опитуватися ззовні, доки запит не буде завершено. Це також ускладнює клієнта, оскільки тепер він повинен перевіряти хід виконання запиту. Але асинхронний зв'язок дозволяє службам залишатися слабо пов'язаними один з одним.

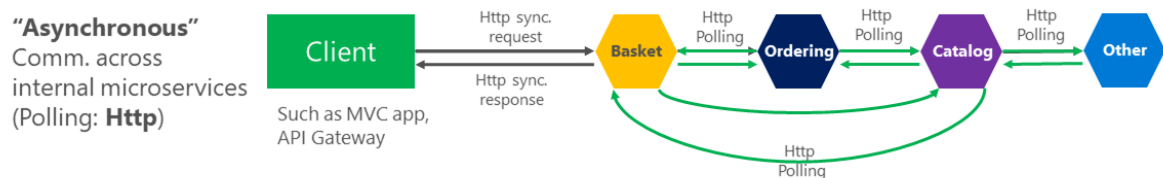


Рис. 2.5 Архітектура асинхронної моделі зв'язку між сервісами через протокол HTTP[16]

2.3.2 Брокер повідомлень

Інший шаблон комунікації, який ми можемо використовувати в архітектурі мікросервісів, — це спілкування на основі повідомлень.

На відміну від зв'язку HTTP, залучені сервіси не спілкуються безпосередньо один з одним. Замість цього сервіси надсилають повідомлення

брокеру повідомлень, на якого підписані інші послуги. Це усуває багато складнощів, пов'язаних із зв'язком HTTP.

Не потрібні сервіси, щоб знати, як розмовляти один з одним; це усуває потребу, щоб сервіси надсилали повідомлення один одному напряму.

Натомість усі сервіси знають про посередника повідомлень, і вони надсилають повідомлення цьому брокеру. Інші сервіси можуть підписатися на повідомлення брокера, який їм цікавий.

Якщо наша програма знаходиться в Amazon Web Services, ми можемо використовувати Simple Notification Service (SNS) як наш посередник повідомлень. Тепер ServiceA може надсилати повідомлення в тему SNS, яку слухає ServiceB.

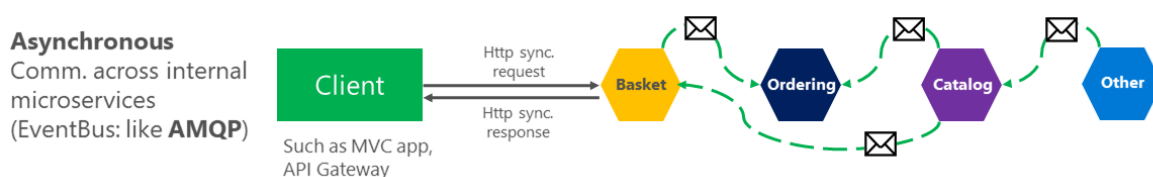


Рис. 2.6 Архітектура асинхронної моделі зв'язку між сервісами через Брокер повідомлень[17]

2.3.3 Використання gRPC у мікросервісній комунікації

gRPC[19] — це фреймворк віддаленого виклику процедур з відкритим кодом, який використовується для високопродуктивного зв'язку між службами. Це ефективний спосіб підключення служб, написаних різними мовами, із підключенням підтримкою балансування навантаження, відстеження, перевірки справності та аутентифікації.

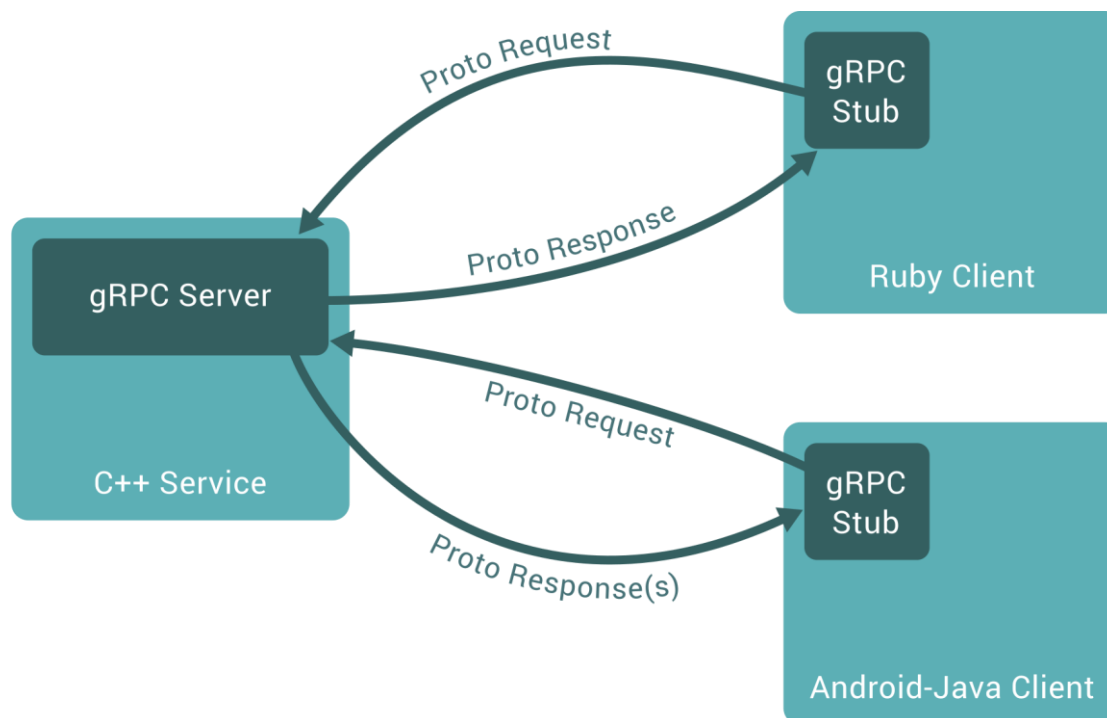


Рис. 2.7 Приклад архітектури з використанням gRPC[18]

За замовчуванням gRPC використовує буфери протоколу для серіалізації структурованих даних. Як правило, gRPC вважається кращою альтернативою протоколу REST для архітектури мікросервісів. RPC означає віддалений виклик процедури. Як випливає з назви, ідея полягає в тому, що ми можемо викликати функцію/метод на віддаленому сервері. Протокол RPC дозволяє отримати результат для задачі в одному форматі, незалежно від того, де вона виконується. Він може бути локальним або на віддаленому сервері з використанням кращих ресурсів. RPC є набагато старішим протоколом, ніж REST. Він використовувався з часів ARPANET[20] в 1970-х роках для виконання мережових операцій. Термін RPC вперше ввів Брюс Джей Нельсон у 1981 році. Але, як ми побачимо, RPC все ще актуальний і реалізований у сучасних програмах на основі API різними способами. Ідея та сама. API створюється шляхом визначення відкритих методів. Потім методи викликаються з аргументами. RPC — це всього лише набір функцій, але в контексті HTTP API він передбачає розміщення методу в URL-адресі, а аргументів — у рядку або в тілі запиту. Найважливіша відмінність полягає в

тому, що gRPC використовує буфери протоколу як мову визначення інтерфейсу для серіалізації та спілкування замість JSON/XML[20]. Буфери протоколу можуть описувати структуру даних, і код може бути згенерований з цього опису для генерування або аналізу потоку байтів, який представляє структуровані дані. Це причина, чому gRPC віддають перевагу для веб-додатків, які є поліглоти (реалізовані за допомогою різних технологій). Формат двійкових даних дозволяє спілкуватися легше. gRPC також можна використовувати з іншими форматами даних, але кращим є буфери протоколу. Щоб дізнатися більше про буфери протоколу, перегляньте цю статтю. Крім того, gRPC побудований на основі HTTP/2[20], який підтримує двонаправлений зв'язок разом із традиційним запитом/відповіддю. gRPC забезпечує слабе з'єднання між сервером і клієнтом. На практиці клієнт відкриває довговічне з'єднання з сервером gRPC, і новий потік HTTP/2 буде відкриватися для кожного виклику RPC.

2.4 Конвертація з формату YAML у формат JSON

YAML — це доступна для читання мова серіалізації даних, яка в основному використовується для файлів конфігурації та в додатках для зберігання або передачі даних. Він широко використовується багатьма мовами програмування, такими як Javascript. Файл YAML створюється з розширенням «.yaml» або «.yml». Ви можете використовувати будь-який формат розширення, оскільки обидва працюють однаково. Мені особисто подобається ".yml", оскільки він виглядає круто, ніж ".yaml" Основна структура Основна структура YAML — це пари ключ і значення. Ключ завжди є рядком, тоді як значенням може бути будь-який тип даних.

JSON – JavaScript Object Notation, зазвичай це текстовий формат, який представляє структуровані дані в синтаксисі об'єкта JavaScript. Він використовується в тих випадках, коли серверу потрібно надіслати деякі дані клієнту, і вони можуть бути відображені на веб-сторінці і навпаки. Незважаючи на те, що він дуже схожий на синтаксис літералу об'єкта

					ІАЛЦ.467200.003 ПЗ	Арк.
						32
Зм.	Арк.	№ докум.	Підпис	Дата		

JavaScript, його можна використовувати незалежно від JavaScript, багато програм можуть читати та створювати документ JSON.

Існують деякі правила, які застосовуються під час роботи з JSON:

1. Дані містяться в парах ім'я/значення.
2. дані виділяються комами.
3. Фігурні дужки утримують об'єкти.
4. Квадратні дужки містять масив об'єктів JSON.

Тож для того щоб конвертувати один формат в інший, нам потрібно проітеруватися по усім вузлам YAML файла, і записати їх ключі та значення у відповідні в JSON форматі.

2.5 Специфікації опису систем оброблення даних з мікросервісною архітектурою

Специфікації призначені для опису розподілених інформаційних систем з мікросервісною архітектурою. Взаємодія програмних модулів (мікросервісів) таких систем здійснюється за допомогою обміну повідомленнями за протоколом Advanced Message Queuing Protocol (AMQP).

2.5.1 Загальні принципи опису процесу за допомогою специфікацій

Опис робочого процесу оброблення даних складається з декларації множини розміщених на обчислювальних вузлах системи налаштованих екземплярів сервісів.

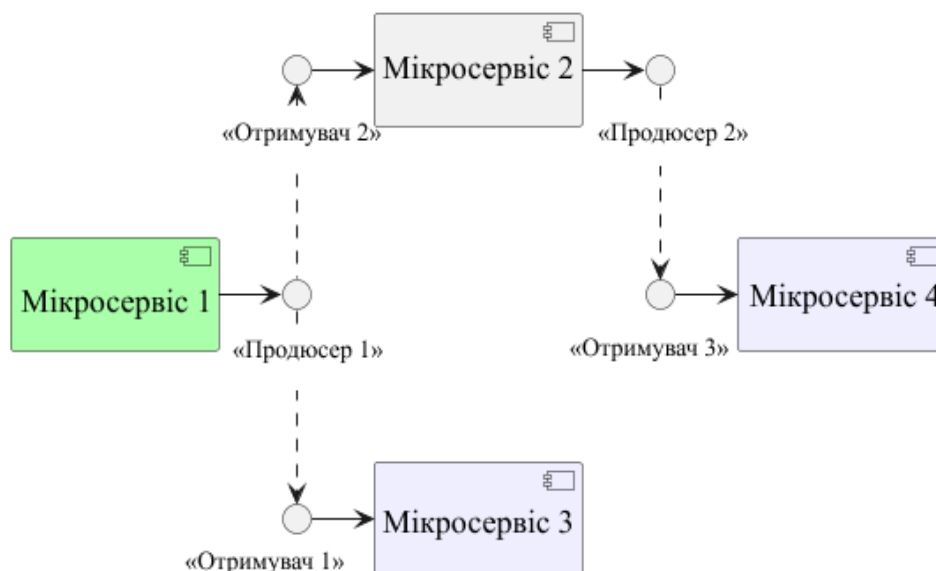


Рис. 2.8 Опис робочого процесу

Тип сервісу декларується у відповідності до концепції "чорної скриньки":

- вхідні налаштування
- прослуховувач повідомлень
- публікувальник повідомлень produce

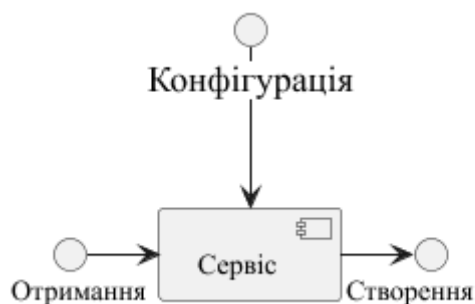


Рис. 2.10 Налаштування сервісу

Тип прослуховувача consume декларує необхідні налаштування обмінника, черги та процесу вибірки повідомлень з використанням брокера повідомлень RabbitMQ:

- amqp - налаштування url для з'єднання з брокером повідомлень
- queue - налаштування черги повідомлень, що буде прослуховуватись

- message - опис типу повідомлень (JSON Schema), які оброблюються сервісом

Тип публікувальника produce декларує необхідні налаштування обмінника в брокері повідомлень RabbitMQ, в якому сервіс буде здійснювати публікування повідомлень:

- amqp - налаштування url для з'єднання з брокером повідомлень
- exchange - налаштування обмінника повідомлень, що буде використовуватись сервісом для публікування повідомлень
- message - опис типу повідомлень (JSON Schema), які надсилаються сервісом

Оскільки сервіси, об'єднані в робочий процес повинні мати спільні налаштування, наприклад, налаштування produce та consume для двох послідовно з'єднаних сервісів, важливим є забезпечення можливості опису компонентів повторюваного використання (components) та способу посилання на такі компоненти. В MSAPI-специфікаціях можна використовувати наступні компоненти повторюваного використання:

- settings - будь-які налаштування
- schemas - описи типів (JSON Schema)
- exchanges - описи налаштувань обмінників повідомлень
- queues - описи налаштувань черг повідомлень
- consumes - описи налаштувань прослуховувачів повідомлень
- produces - описи налаштувань публікувальників повідомлень

Для посилання на компоненти повторюваного використання застосовуються JSON References та JSON Pointers. Можна використовувати як локальні посилання (в межах локальної файлової системи), так і віддалені.

Для збереження та спільного доступу до специфікацій можна використовувати сервіс msapi-registry, що реалізує реєстр MSAPI-специфікацій.

2.5.2 Особливості використання специфікацій

Мінімальний склад специфікації повинен містити:

- версію MSAPI, наприклад:

```
msapi: "1.0.1"
```

- метадані metadata, до яких входять:
 - ідентифікатор – id, що використовується у посиланнях
 - назва – title
 - опис – description, який дозволяє зрозуміти зміст специфікації та що вона описує

Файл 01.example-min-spec.yaml. Базовий приклад демонструє мінімальний склад специфікації MSAPI.

```
msapi: 1.0.1
```

```
metadata:
```

```
  id: 01.example-min-spec
```

```
  title: Приклад специфікації MSAPI
```

```
  description: Демонструє мінімальний склад специфікації MSAPI
```

Секція налаштувань settings містить всі описи налаштувань.

Секція налаштувань може містити будь-які описи, як прості, так і об'єкти. Загальні налаштування url-з'єднання можуть бути подані у вигляді рядка, наприклад

```
amqp_url:
```

```
amqps://xoilebqg:Nx46t4t9cxQ2M0rF2rIyZPS_xbAhmJIG@horner.rmcloudamqp.com/xoilebqg
```

					ІАЛЦ.467200.003 ПЗ	Арк.
						36
Зм.	Арк.	№ докум.	Підпис	Дата		

а налаштування системи моніторингу стану мікросервісів – у вигляді об'єкта, наприклад

```
health_control:
```

```
  exchange_name: ms_health
```

```
  interval: 1000 #ms
```

2.6 Конфігурування робочих процесів за допомогою специфікацій

2.6.1 Базовий процес

Базовий приклад описує два екземпляри сервісів producer та consumer, які створюються на основі базових і користувацьких параметрів підключення та використовують спільну чергу queue для передачі повідомлень.

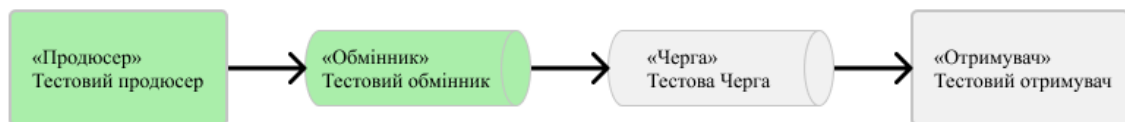


Рис. 2.10 Приклад базового процесу

Його опис виглядає наступним чином:

```
msapi: "1.0.1"
```

```
metadata:
```

```
  id: 01.use-msapi
```

```
  title: Базовий приклад специфікації MSAPI
```

```
  description: >
```

Описує два екземпляри сервісів, які використовують спільну чергу для передачі повідомлень

```
workflow:
```

```
  - instance: # producer instance
```

```
    produce:
```

```
      amqp:
```

```
        url: "amqp://yourRabbitMQconnectionURL"
```

```
      exchange:
```

```
        name: amqp_test_exchange
```

```
        mode: fanout
```

					ІАЛЦ.467200.003 ПЗ	Арк.
						37
Зм.	Арк.	№ докум.	Підпис	Дата		

```

- instance: # consumer instance
  consume:
    amqp:
      url: "amqp://yourRabbitMQconnectionURL"
    queue:
      name: test
      exchange:
        name: amqp_test_exchange
        mode: fanout

```

2.6.2 Прослуховування декількома споживачами однієї черги повідомлень

Для того щоб організувати роботу системи з декількох мікросервісів, які будуть надсилати результати своєї роботи, нам буде потрібна черга завдань. У випадку, коли одні й ті ж самі повідомлення повинні бути оброблені за допомогою різних алгоритмів, можна організувати прослуховування однієї черги повідомлень декількома споживачами. В цьому випадку оброблення повідомлень цими споживачами буде здійснюватися паралельно.

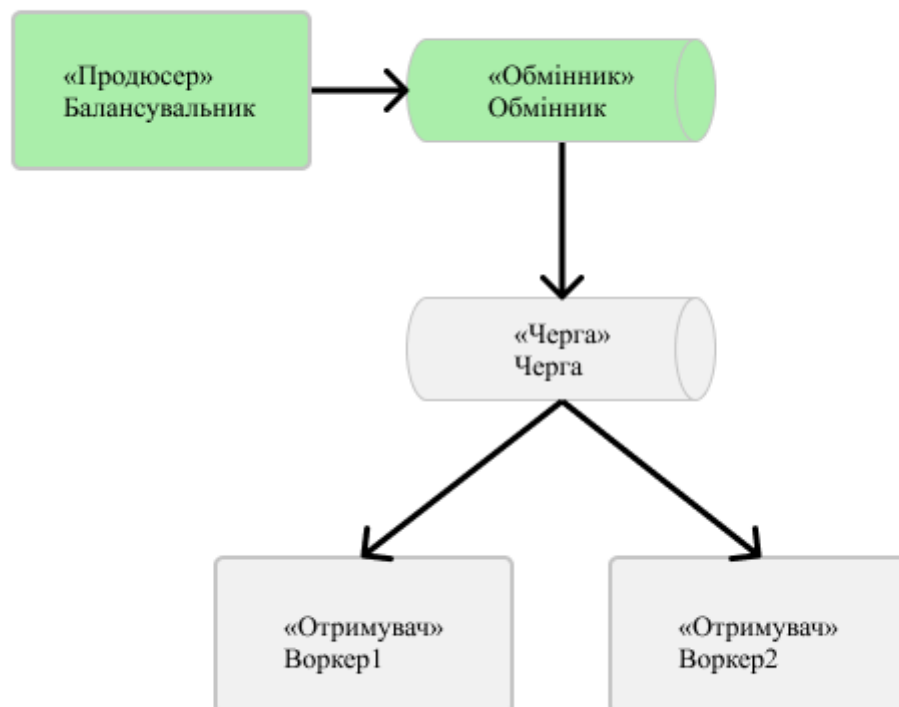


Рис. 2.11 Приклад роботи черги повідомлень

Для цього необхідно скористатись прикладами yaml конфігів, які наведені нижче. З головних деталей, які додалися в порівнянні з попереднім прикладом:

- `queue.options.prefetch` - якщо в обміннику є повідомлення, то `consumer` (в даному випадку `worker` за 1 раз "забере" вказане число повідомлень)
- `queue.options.noAck` - `consumer` може автоматично "давати знати", що повідомлення доставлено, а можна це зробити мануально, викликавши метод `.ack()`, але пр цьому вказавши `noAck: false`.

Ініціалізація одного продюсера і 2 отримувачів, які слухають одну й ту саму чергу як і в попередньому прикладі. Але, різниця полягає в тому, що в `yaml` конфігу задано `noAck: false`. Це означає, що потрібно викликати `.ack()`. Якщо цього не зробити, слухач "зупиниться" на цьому повідомленні, як і реалізовано: за час 4 сек, другий воркер опрацює всі повідомлення в черзі, а перший воркер буде 'висіти' на першому повідомленні.

Приклад конфігурації для такого випадка, налаштування першого прослуховувача:

```
msapi: "1.0.1"

metadata:

  id: 04.use-msapi.listener1

  title: Прослуховувач 1

  description: >
    Описує сервіс прослуховування черги повідомлень. Повідомлення
    типізовані та описані схемою message
    config та configuredBy

service:
  consume:
    amqp:
      url: amqps://RabbitMQconnectionURL

  queue:
    name: listen1
```

					ІАЛЦ.467200.003 ПЗ	Арк.
						39
Зм.	Арк.	№ докум.	Підпис	Дата		

```

exchange:
  name: broadcast
  mode: fanout
  options:
    durable: false
options:
  noAck: true

message:
  type: object
  required:
    - id
    - listener
    - timeout
  properties:
    id:
      type: number
    listener:
      type: number
    timeout:
      type: number

```

Налаштування другого прослуховувача:

```

msapi: "1.0.1"

metadata:

  id: 04.use-msapi.listener2

  title: Прослуховувач 2

  description: >
    Описує сервіс прослуховування черги повідомлень. Повідомлення
    типізовані та описані схемою message
    config та configuredBy

service:
  consume:
    amqp:
      url: amqps://RabbitMQconnectionURL

  queue:
    name: listen2
    exchange:

```

					ІАЛЦ.467200.003 ПЗ	Арк.
						40
Зм.	Арк.	№ докум.	Підпис	Дата		

```

      name: broadcast
      mode: fanout
      options:
        durable: false
    options:
      noAck: true

  message:
    type: object
    required:
      - id
      - listener
      - timeout
    properties:
      id:
        type: number
      listener:
        type: number
      timeout:
        type: number

```

Налаштування публікувальника

```

msapi: "1.0.1"

metadata:

  id: 04.use-msapi.producer

  title: Публікувальник 1

  description: >
    Описує сервіс публікувальника повідомлень. Повідомлення типізовані
та описані схемою message

service:

  produce:
    amqp:
      url:
amqps://xoilebqg:Nx46t4t9cxQ2M0rF2rIyZPS_xbAhmJIG@hornet.rmq.cloudamqp.com
/xoilebqg

    exchange:
      name: broadcast

```

					ІАЛЦ.467200.003 ПЗ	Арк.
						41
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        mode: fanout
        options:
            durable: false

    message:
        type: object
        required:
            - id
            - listener
            - timeout
        properties:
            id:
                type: number
            listener:
                type: number
            timeout:
                type: number

```

Налаштування робочого процесу

```

msapi: "1.0.1"

metadata:

    id: 04.use-msapi.publisher

    title: Публікувальник 1

    description: >
        Описує робочий процес для прослуховування декількома
        споживачами однієї черги повідомлень

workflow:
    - instance:
        $ref: "./producer/#/service"
    - instance:
        $ref: "./listener1/#/service"
    - instance:
        $ref: "./listener2/#/service"

```

2.6.3 Конфігурування мікросервісів з допомогою специфікацій

Приклад використання MSAPI-специфікацій В прикладі визначається робочий процес з двох екземплярів сервісів. Перший - завантажує повідомлення з джерел на основі завдань, що вибирає з черги завдань. Другий

					ІАЛЦ.467200.003 ПЗ	Арк.
						42
Зм.	Арк.	№ докум.	Підпис	Дата		

- виконує нормалізацію повідомлень. В окремих файлах визначаються глобальні налаштування, налаштування за замовченням, типи черг та обмінників, типи мікросервісів та робочий процес. Файл глобальних налаштувань містить налаштування для брокера повідомлень та налаштування підсистеми виявлення та моніторингу працездатності мікросервісів.

```
msapi: "1.0.1"

metadata:

  id: "http://localhost:8081/spec/ms_example_global_settings.yaml"

  title: "Global settings"

  description: Default queue and exchange settings

components:

#####

#####

  Settings:          # available section

    amqp:

      url:

"amqps://xoilebqg:Nx46t4t9cxQ2M0rF2rIyZPS_xbAhmJIG@hornet.rmq.cloudam
qp.com/xoilebqg"

    health:

      amqp:

        $ref: "#/components/Settings/amqp"

    produce:

      exchange:

        name: health_messages
```

					ІАЛЦ.467200.003 ПЗ	Арк.
						43
Зм.	Арк.	№ докум.	Підпис	Дата		

```
time_interval: 30000
```

```
#####  
#####
```

Файл налаштувань за замовчуванням містить налаштування для каналів, обмінників, черг ...

```
msapi: "1.0.1"
```

```
metadata:
```

```
id: "http://localhost:8081/spec/ms_example_defaults.yaml"
```

```
title: "Default queue and exchange settings"
```

```
description: Default queue and exchange settings
```

```
components:
```

```
#####  
#####
```

```
Exchanges:          # available section
```

```
  default:           # an example of exchange definition
```

```
    type: fanout      # or "topic" or "direct" - required type of exchange
```

```
    options:          # optional exchange options
```

```
      durable: true   # (boolean): if true, the exchange will survive broker  
restarts. Defaults to true.
```

```
      autoDelete: false # (boolean): if true, the exchange will be destroyed  
once the number of bindings for which it is the source drop to zero. Defaults to  
false.
```

```
#####  
#####
```

```

Queues:                                # available section

default:                                # an example of queue definition

exchange:

    $ref: "#/components/Exchanges/default"

options:

    exclusive: false                    # if true, scopes the queue to the connection
                                        (defaults to false)

    durable: true                       # if true, the queue will survive broker restarts,
                                        modulo the effects of exclusive and autoDelete; this defaults to true if not supplied,
                                        unlike the others

    autoDelete: false                  # if true, the queue will be deleted when the number
                                        of consumers drops to zero (defaults to false)

#####

Consumes:                              # available section

default:

queue:

    $ref: "#/components/Queues/default"

message:

options:

    noAck: false

    exclusive: false

#####

```

```

Producers:          # available section

default:

  exchange:

    $ref: "#/components/Exchanges/default"

  message:

    options:

      persistent: true

```

2.7 Процес взаємодії між компонентами системи

Теперь нам потрібен модуль, що буде відповідати за спілкування з центральним вузлом (брокером повідомлень), продюсерів, отримувачів, та буде включати в себе розуміння за допомогою чого будуть продукуватися продюсери і отримувачі центральним вузлом. Базова концепція їх взаємодії відображена на діаграмі класів.

					ІАЛЦ.467200.003 ПЗ	Арк.
						46
Зм.	Арк.	№ докум.	Підпис	Дата		

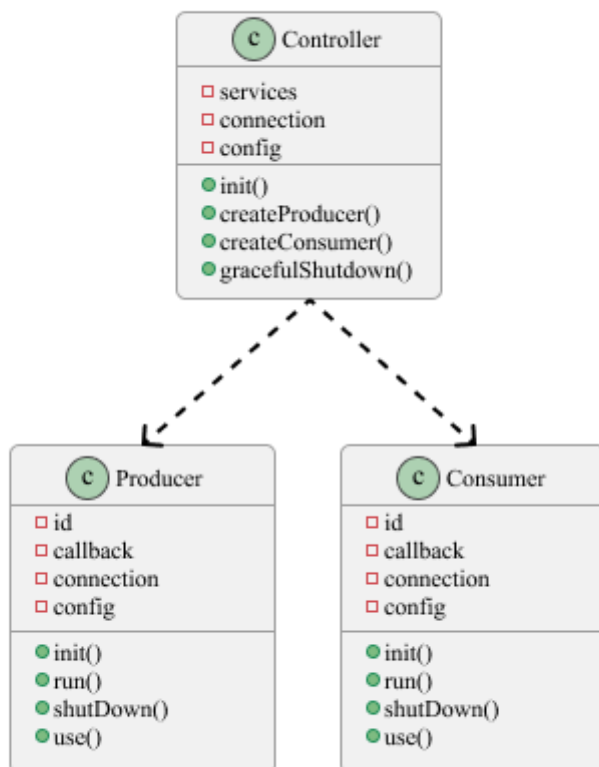


Рис. 2.13 Схематичне зображення системи

2.8 Вибір засобів проєктування

2.8.1 Javascript

Розробка буде вестися мовою Javascript. JavaScript[23] – це мова програмування, яка вперше була створена в 1994 році як спосіб додати функціональності та взаємодії з користувачем веб-сайту. Спочатку JavaScript був розроблений для використання виключно на передньому плані як спосіб для веб-розробників додавати функціональні можливості до своїх веб-сторінок, і на початку він робив саме це. Нещодавно запровадження «движка V8»[23] від Google покращило швидкість та функціональність JS. Це призвело до розробки та випуску нових захоплюючих фреймворків переднього плану JavaScript і, зрештою, Node.js, способу запуску JavaScript на сервері (бек-енд). Ця нова розробка призвела до відродження JavaScript. Зараз JavaScript є однією з найпоширеніших мов програмування у світі. Ми

можемо знайти JavaScript, який використовується на передньому плані, серверній частині, мобільному пристрої, Інтернеті речей (Iot), розробці ігор і насправді скрізь, де буде використовуватися традиційна мова програмування. Нещодавно була випущена найновіша версія мови JavaScript, ES6.

2.8.2 NodeJS

Node.js[23] — це фреймворк для написання серверних програм JavaScript. Він побудований на основі середовища виконання V8 JavaScript і використовує керовану подіями, неблокуючу модель вводу-виводу, що робить його ідеальним для додатків реального часу з інтенсивним використанням даних. Ці програми отримують і надсилають дані через серверну службу, яка називається API. API служить інтерфейсом між різними програмами, щоб вони могли спілкуватися один з одним. Веб-програма та мобільний додаток, наведені нижче, можуть використовувати той самий API для зберігання даних, надсилання електронних листів, push-повідомлень або ініціювання робочих процесів на сервері.

2.8.3 OpenAPI

Специфікація OpenAPI[25] (раніше відома як Swagger) визначає стандартний інтерфейс, що не залежить від мови, для RESTful API. Основна перевага використання стандартного визначення полягає в тому, що всі споживачі сервісу (включаючи інших розробників, клієнтів або навіть інші мікросервіси) можуть розуміти сервіс і взаємодіяти з ним з мінімальною логікою реалізації. Використання специфікації OpenAPI (OAS[25]) має й інші особливі переваги, такі як автоматичне генерування коду для клієнтів і серверів, створення інтерактивної документації API та інструменти тестування. Наразі OAS знаходиться у версії 3.0.2. Версія OAS здійснюється за допомогою Semantic Versioning 2.0.0 (semver).

2.8.4 OpenAPI

AMQP[26] передбачає поведінку видавця повідомлень і споживача повідомлень для безперебійного транспортування повідомлень між різними

					ІАЛЦ.467200.003 ПЗ	Арк.
						48
Зм.	Арк.	№ докум.	Підпис	Дата		

програмами, створеними різними постачальниками на різних мовах програмування. Брокери обміну повідомленнями отримують повідомлення від видавців (програми, які їх публікують, також відомі як виробники) і направляють їх споживачам (програми, які їх обробляють). Оскільки це мережевий протокол, видавці, споживачі та брокер можуть перебувати на різних машинах. Модель AMQP має такий погляд на світ: повідомлення публікуються на обмінниках, які часто порівнюють з поштовими відділеннями чи поштовими скриньками. Потім обмінники розподіляють копії повідомлень у черги за допомогою правил, які називаються прив'язками. Потім брокер або доставляє повідомлення споживачам, які підписалися на черги, або споживачі отримують/витягують повідомлення з черг на вимогу.

					ІАЛЦ.467200.003 ПЗ	Арк.
						49
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ ДО РОЗДІЛУ 2

У цьому розділі ми детально розглянули предметну область завдання, яке ми сформувавши в попередньому розділі, а також, яким саме чином специфікації, призначені для опису розподілених інформаційних систем з мікросервісною архітектурою, передають конфігурації до програмних модулів. Було сформовано, яким чином формати передачі даних конвертуються один в одного. Була також розглянута взаємодія програмних модулів (мікросервісів). Ми розглянули деякі інструменти розробки, такі як мова програмування Javascript, технологічна платформа Node JS, протокол AMQP, а також бібліотеку OpenAPI.

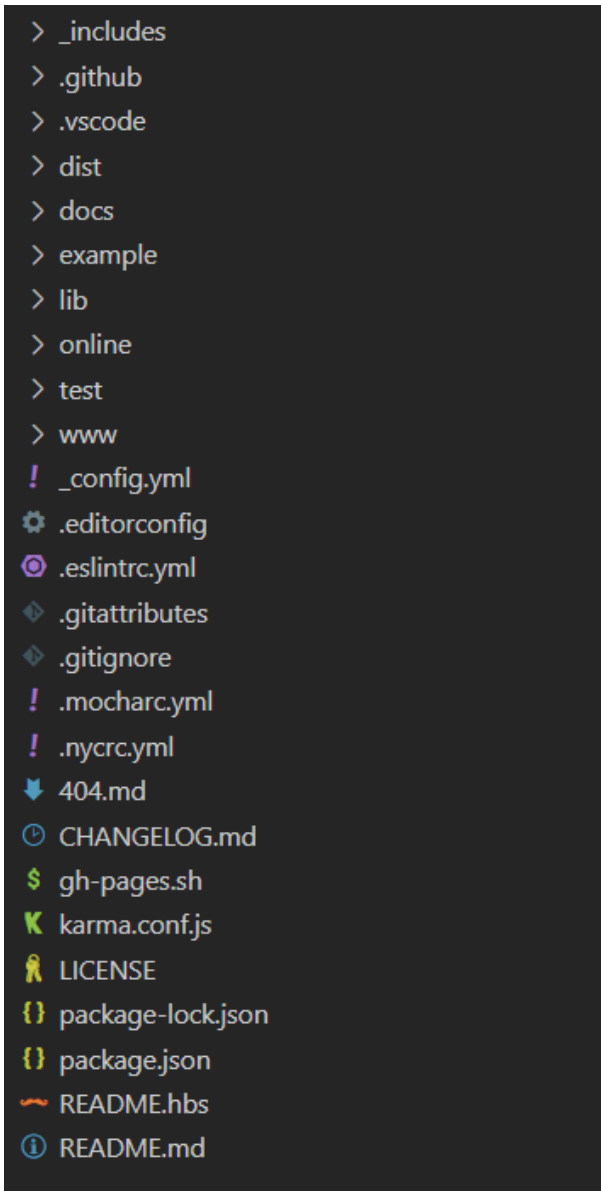
					ІАЛЦ.467200.003 ПЗ	Арк.
						50
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3.

РЕАЛІЗАЦІЯ МОДУЛЯ І ТЕСТУВАННЯ

Після того, як ми поставили задачу, розглянули вимогу, рекомендації до системи, скористуємося діаграмою класів, вказаними інструментами, що потрібні в імплементації нашої системи. У цьому розділі ми розглянемо опис вирішення заданої проблеми, створення системи для цього, викладемо принципи роботи, конфігурації.

3.1 Структура проєкту



```
> _includes
> .github
> .vscode
> dist
> docs
> example
> lib
> online
> test
> www
! _config.yml
⚙ .editorconfig
🌀 .eslintrc.yml
💎 .gitattributes
💎 .gitignore
! .mocharc.yml
! .nycrc.yml
📄 404.md
🕒 CHANGELOG.md
$ gh-pages.sh
📄 karma.conf.js
📄 LICENSE
📄 package-lock.json
📄 package.json
📄 README.hbs
📄 README.md
```

Рис. 3.1 Структура проєкту

3.2 Огляд файлів у проєкті

- `package.json` — містить різні метадані, пов’язані з проєктом. Цей файл використовується для надання інформації npm, що дозволяє йому ідентифікувати проєкт, а також обробити його спеціально. Це також можуть бути інші метадані, такі як опис проєкту, версія проєкту в певному дистрибутиві, інформація про ліцензію, навіть конфігурації - все це може бути дійсним так само, як і для пакетів npm і кінцевого користувача.
- `lib` – містить кодову базу даного рішення
- `examples` – має приклади для використання даної системи
- `test` – містить тест кейси для створеної системи, в основному, `unit`, для тестування функціоналу. Можна переглянути покриття і результати тестів у документації
- `.eslint.json` – конфігурації для літера ESLint.
- `.prettierrc` – конфігурації для бібліотеки форматування коду, Prettier.
- `gh_pages.sh` – скрипт, що додає документацію на додаток Github Pages, для онлайн перегляду.
- `node_modules` – каталог що містить зовнішні залежності проєкта, описані в `package.json`
- `docs` – каталог, що містить документацію, згенеровану за допомогою бібліотеки `vuepress`

3.3 Огляд основних компонентів системи

3.3.1 msapi

Схема перевірки MSAPI-специфікації 1.0.X.

					ІАЛЦ.467200.003 ПЗ	Арк.
						52
Зм.	Арк.	№ докум.	Підпис	Дата		

Properties(5)

Name	Type	Description	Required
msapi	string	Pattern: ^1\,0\,d(-.+)?\$	yes
metadata	object	Підсхема "metadata".Метадані специфікації Subschema: Підсхема "metadata"	yes
components	object	Підсхема "components".Компоненти повторюваного використання Subschema: Підсхема "components"	no
service	object	Підсхема "service".Налаштування типу сервісу Subschema: Підсхема "service"	no
workflow	object[]	Підсхема "workflow".Налаштування робочого процесу. Визначає сукупність екземплярів сервісів та їх налаштування Subschema: Підсхема "workflow"	no

msapi(Версія MSAPI-специфікації)

Description: Наразі підтримується версія 1.0.1

Reference to [#/definitions/Version](#)

metadata (Підсхема "metadata")

Description: Метадані специфікації

Reference [msapi.metadata](#)

components (Підсхема "components")

Description: Компоненти повторюваного використання

Reference [msapi.components](#)

Рис. 3.20 Опис об'єкту msapi

3.3.2 metadatata

Метадані специфікації

Name	Type	Description	Required
id	string	id.Ідентифікатор специфікації. Зазвичай співпадає з посиланням на неї та використовується у <code>\$ref</code>	yes
title	string	title.Назва специфікації	yes
description	string	description.Опис специфікації	no
contact	object	Contact.Контактна інформація	no
license	object	License.Інформацію про ліцензію	yes
repo	string	repo.Посилання на репозиторій, де міститься вихідний код Format: "uri-reference"	no

Рис. 3.20 Опис об'єкту metadatata

3.3.3 components

Компоненти повторюваного використання

Name	Type	Description
<code>^exchanges\$</code>	object	exchanges.Налаштування обмінників повідомлень
<code>^queues\$</code>	object	queues.Налаштування черг повідомлень
<code>^schemas\$</code>	object	schemas.Опис типів повідомлень (JSON Schema)
<code>^consumes\$</code>	object	consumes.Налаштування прослуховувачів повідомлень
<code>^produces\$</code>	object	produces.Налаштування публікувальників повідомлень
<code>^settings\$</code>	object	settings.Налаштування, які можна повторювано використовувати будь-де.

Рис. 3.21 Опис об'єкту components

3.3.4 exchanges

Налаштування обмінників повідомлень

Properties (Pattern)

Name	Type	Description	Required
<code>^[a-zA-Z0-9\.\- _]+\$</code>		Будь-який ідентифікатор.Налаштування обмінника	

`^exchanges$.^[a-zA-Z0-9\.\- _]+$`(Будь-який ідентифікатор)

Description: Налаштування обмінника

- Option 1 (alternative):
Description: Посилання на налаштування обмінника
Reference to [#/definitions/Reference](#)
- Option 2 (alternative):
Description: Налаштування обмінника
Reference [msapi.components.exchange](#)

Рис. 3.22 Опис об'єкту exchanges

3.3.5 queues

Налаштування черг повідомлень

Properties (Pattern)

Name	Type	Description	Required
<code>^[a-zA-Z0-9\.\- _]+\$</code>		Будь-який ідентифікатор.Налаштування черги	

`^queues$.^[a-zA-Z0-9\.\- _]+$`(Будь-який ідентифікатор)

Description: Налаштування черги

- Option 1 (alternative):
Description: Посилання на налаштування черги

Reference to [#/definitions/Reference](#)
- Option 2 (alternative):
Description: Налаштування черги повідомлень

Reference [msapi.components.queue](#)

Рис. 3.23 Опис об'єкту queues

3.3.6 schemas

Опис типів повідомлень (JSON Schema)

Properties (Pattern)

Name	Type	Description	Required
<code>^[a-zA-Z0-9\.\- _]+\$</code>		Будь-який ідентифікатор.Опис типу повідомлень	

`^schemas$.^[a-zA-Z0-9\.\- _]+$`(Будь-який ідентифікатор)

Description: Опис типу повідомлень

- Option 1 (alternative):
Description: Посилання на опис типу повідомлень

Reference to [#/definitions/Reference](#)
- Option 2 (alternative):
Description: Опис типу повідомлень

Reference to [#/definitions/Schema](#)

Рис. 3.24 Опис об'єкту schemas

3.3.7 consumes

Налаштування прослуховувачів повідомлень

					ІАЛЦ.467200.003 ПЗ	Арк.
						55
Зм.	Арк.	№ докум.	Підпис	Дата		

Description: Налаштування прослуховувача повідомлень

- Option 1 (alternative):

Description: Посилання на налаштування прослуховувача повідомлень

Reference to [#/definitions/Reference](#)

- Option 2 (alternative):

Description: Налаштування споживача

Additional Properties: not allowed

Properties (Pattern)

Name	Type	Description	Required
^amqp	object	Підсхема "connection".Налаштування з'єднання з брокером повідомлень Subschema: Підсхема "connection"	yes
^queue\$		queue.Налаштування черги	
^message\$		message.Налаштування повідомлень	

Option 2: ^amqp (Підсхема "connection")

Description: Налаштування з'єднання з брокером повідомлень

Reference [msapi.components.connection](#)

Option 2: ^queue\$(queue)

Description: Налаштування черги

- Option 1 (alternative):

Description: Налаштування черги повідомлень

Reference [msapi.components.queue](#)

- Option 2 (alternative):

Description: Посилання на налаштування черги

Reference to [#/definitions/Reference](#)

Option 2: ^message\$(message)

Description: Налаштування повідомлень

- Option 1 (alternative):

Description: Посилання на налаштування повідомлень

Reference to [#/definitions/Schema](#)

- Option 2 (alternative):

Description: Посилання на налаштування повідомлень

Reference to [#/definitions/Reference](#)

Рис. 3.25 Опис об'єкту consumes

3.3.8 produces

Налаштування публікувальників повідомлень

					ІАЛЦ.467200.003 ПЗ	Арк.
						56
Зм.	Арк.	№ докум.	Підпис	Дата		

Properties (Pattern)

Name	Type	Description	Required
<code>^[a-zA-Z0-9\.\- _]+\$</code>		Будь-який ідентифікатор.Налаштування публікувальника повідомлень	

`^produces$.^[a-zA-Z0-9\.\- _]+$`(Будь-який ідентифікатор)

Description: Налаштування публікувальника повідомлень

- Option 1 (alternative):

Description: Посилання на налаштування публікувальника повідомлень

Reference to [#/definitions/Reference](#)

- Option 2 (alternative):

Description: Налаштування публікувальника

Reference [msapi.components.produce](#)

Рис. 3.26 Опис об'єкту produces

3.3.9 exchange

Параметри обміника

Additional Properties: not allowed

Properties (Pattern)

Name	Type	Description	Required
<code>^name\$</code>	string	name.Ім'я обміника	
<code>^mode\$</code>		mode.Тип обміника Default: "fanout" Enum: "fanout", "direct", "topic"	
<code>^options\$</code>	object	Підсхема "exchange-options".Налаштування обміника Subschema: Підсхема "exchange-options"	

Рис. 3.27 Опис об'єкту exchange

3.3.10 exchange-options

Налаштування обміника

Properties (Pattern)

Name	Type	Description	Required
<code>^durable\$</code>	boolean	durable.Налаштування обмінника. Якщо встановлено значення <code>true</code> обмінник переживе перезапуск брокера Default: <code>true</code>	
<code>^autoDelete\$</code>	boolean	autoDelete.Налаштування обмінника. Якщо встановлено значення <code>true</code> , обмінник буде знищено, коли кількість прив'язок, для яких він є джерелом, знизиться до нуля Default: <code>false</code>	
<code>^persistent\$</code>	boolean	persistent.Налаштування обмінника. Якщо встановлено значення <code>true</code> , повідомлення переживе перезапуск брокера за умови, що воно знаходиться в черзі, яка також витримує перезапуски. Відповідає і перевизначає властивість <code>deliveryMode</code> Default: <code>true</code>	

Рис. 3.28 Опис об'єкту exchange-options

3.3.11 queue-options

Налаштування черги повідомлень

Description:

Налаштування черги

Properties (Pattern)

Name	Type	Description	Required
<code>^noAck\$</code>	boolean	noAck.Налаштування черги. Якщо встановлено значення <code>true</code> , то брокер буде видаляти повідомлення, надіслані цьому споживачеві, з черги, як тільки вони будуть відправлені Default: <code>false</code>	
<code>^exclusive\$</code>	boolean	exclusive.Налаштування черги. Якщо встановлено занчення <code>true</code> , то брокер не дозволить нікому другому споживати з цієї черги. Default: <code>false</code>	
<code>^durable\$</code>	boolean	durable.Налаштування черги. Якщо встановлено значення <code>true</code> , черга переживе перезапущ брокера, модулюючи вплив <code>exclusive</code> та <code>autoDelete</code> . Default: <code>true</code>	
<code>^autoDelete\$</code>	boolean	autoDelete.Налаштування черги. Якщо встановлено значення <code>true</code> , черга буде видалена, коли кількість прив'язок знизиться до нуля Default: <code>false</code>	
<code>^prefetch\$</code>	number	prefetch.Налаштування черги. Визначає максимальну кількість повідомлень, які вибираються з черги одночасно Default: <code>1</code>	

`^noAck$(noAck)`

Description: Налаштування черги. Якщо встановлено значення `true`, то брокер буде видаляти повідомлення, надіслані цьому споживачеві, з черги, як тільки вони будуть відправлені

Type: `boolean`

`^exclusive$(exclusive)`

Description: Налаштування черги. Якщо встановлено занчення `true`, то брокер не дозволить нікому другому споживати з цієї черги.

Type: `boolean`

`^durable$(durable)`

Description: Налаштування черги. Якщо встановлено значення `true`, черга переживе перезапущ брокера, модулюючи вплив `exclusive` та `autoDelete`.

Type: `boolean`

`^autoDelete$(autoDelete)`

Description: Налаштування черги. Якщо встановлено значення `true`, черга буде видалена, коли кількість прив'язок знизиться до нуля

Type: `boolean`

`^prefetch$(prefetch)`

Description: Налаштування черги. Визначає максимальну кількість повідомлень, які вибираються з черги одночасно

Type: `number`

Рис. 3.29 Опис об’єкту queue-options

3.3.12 connection

Налаштування з'єднання з брокером повідомлень

msapi.components.connection(Підсхема "connection")

Description:

Налаштування з'єднання з брокером повідомлень

Properties (Pattern)

Name	Type	Description	Required
^url\$	string	url.url з'єднання з брокером повідомлень Format: "uri-reference"	no

^url\$(url)

Description: url з'єднання з брокером повідомлень

Type: string

Format: "uri-reference"

Рис. 3.30 Опис об'єкту connection

3.3.13 roduce

Налаштування публікувальника

Properties (Pattern)

Name	Type	Description	Required
^amqp	object	Підсхема "connection".Налаштування з'єднання з брокером повідомлень Subschema: Підсхема "connection"	yes
^exchange\$		exchange.Налаштування обмінника	
^message\$		message.Налаштування повідомлення	

^amqp (Підсхема "connection")

Description: Налаштування з'єднання з брокером повідомлень

Reference [msapi.components.connection](#)

^exchange\$(exchange)

Description: Налаштування обмінника

- Option 1 (alternative):
Description: Налаштування обмінника
Reference [msapi.components.exchange](#)
- Option 2 (alternative):
Description: Посилання на налаштування обмінника
Reference to [#/definitions/Reference](#)

^message\$(message)

Description: Налаштування повідомлення

- Option 1 (alternative):
Description: Посилання на налаштування повідомлень
Reference to [#/definitions/Schema](#)
- Option 2 (alternative):
Description: Посилання на налаштування повідомлень
Reference to [#/definitions/Reference](#)

Рис. 3.31 Опис об'єкту produce

3.3.14 service

Налаштування типу сервісу

Additional Properties: not allowed

Properties (Pattern)

Name	Type	Description	Required
^config\$	object	config.Дані для налаштування сервісу	
^consume\$		consume.Налаштування прослуховувача повідомлень	
^produce\$		produce.Налаштування публікувальника повідомлень	

^config\$(config)

Description: Дані для налаштування сервісу

No properties.

^consume\$(consume)

Description: Налаштування прослуховувача повідомлень

- Option 1 (alternative):
Description: Налаштування споживача

Additional Properties: not allowed

Properties (Pattern)

Name	Type	Description	Required
^amqp	object	Підсхема "connection".Налаштування з'єднання з брокером повідомлень Subschema: Підсхема "connection"	yes
^queue\$		queue.Налаштування черги	
^message\$		message.Налаштування повідомлень	

Рис. 3.32 Опис об'єкту service

3.3.15 workflow

Налаштування робочого процесу. Визначає сукупність екземплярів сервісів та їх налаштування

Item Properties (Pattern)

Name	Type	Description	Required
<code>^instance\$</code>		instance.Налаштування екземпляру сервісу	no
<code>^name\$</code>	string	name.Назва екземпляру сервісу	no
<code>^configuredBy\$</code>	object	configuredBy.Налаштування екземпляру сервісу	no

Рис 3.33 Опис об'єкту workflow

3.4 Тестування Модуля

Для запуску набору тестів використовується команда `npm test`. Для розробки тест кейсів було використано фреймворк для тестування Mocha. Принципи написання тестів дуже легкий, а також чудово інтегрується в проєкт. За допомогою цього фреймворка було створено тести для основних функцій, методів класів.

Програмний модуль не має призводити до збою (фатального порушення роботи системи) та повинен задовільняти наступним вимогам:

- повертати помилку валідації для пустої специфікації
- повертати помилку валідації для специфікації з відсутною версією `msapi`
- успішно валідувати специфікацію з коректно заповненими метаданими

```
PASS test/metadata/metadata.test.js
Тести для перевірки підсхеми "metadata"
  ✓ Повинен повернути помилку валідації для пустої специфікації (2 ms)
  ✓ Повинен повернути помилку валідації для специфікації з відсутною версією msapi (15 ms)
  ✓ Повинен успішно валідувати специфікацію з коректно заповненими метаданими (1 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:  0 total
Time:        1.231 s
Ran all test suites.
```

Рис. 3.34 Інформаційний екран звіту по тестуванню

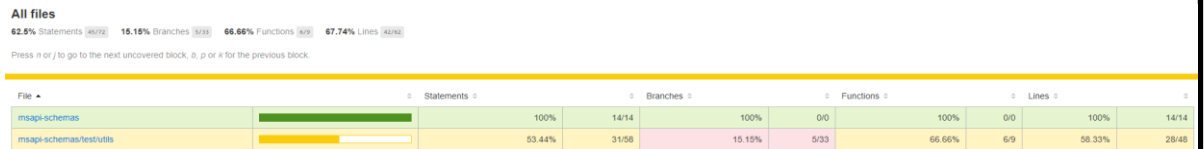


Рис. 3.34 Відсоток покриття тестами системи

3.5 Демонстрація роботи програми

Для того щоб показати роботу програми, ми використаємо приклад некоректної схеми

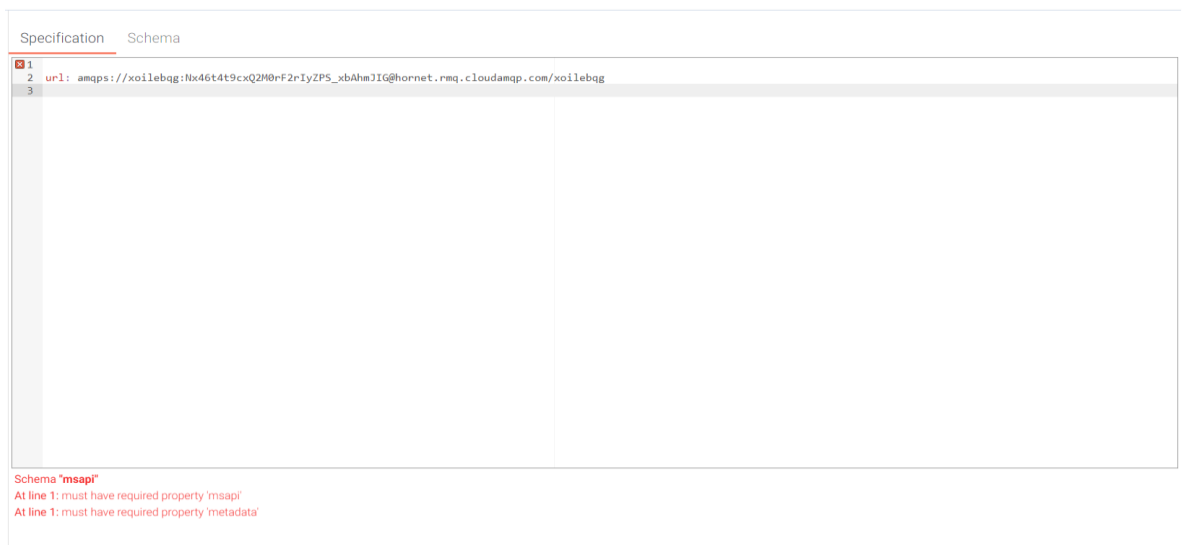


Рис. 3.35 Інтерфейс додатку, некоректна валідація

Як можемо бачити зі знімку екрану, в зв'язку з тим, що мінімальні вимоги до специфікації не були виконані, система, провалідувавши некоректну специфікацію, відреагувала відповідно і повернула помилки, де зазначила необхідні правки.

Тепер, ми використаємо приклад коректної конфігурації

```
msapi: 1.0.1

metadata:
  id: 03.example-components
  title: Приклад специфікації MSAPI для компонентів повторюваного використання
```

```
description: Демонструє опис JSON-схем повторюваного
використання
```

```
# Секція JSON-схем може містити будь-які JSON-схеми
```

```
schemas:
```

```
    TASK_MSG_SCHEMA:
```

```
        type: object
```

```
        required:
```

- id
- options

```
        properties:
```

```
            id:
```

```
                type: string
```

```
            options:
```

```
                type: object
```

```
        additionalProperties: false
```

```
    DATA_MSG_SCHEMA:
```

```
        type: object
```

```
        required:
```

- id
- text
- meta

```
        properties:
```

```
            id:
```

```
                type: string
```

```
            text:
```

```
                type: string
```

					ІАЛЦ.467200.003 ПЗ	Арк.
						64
Зм.	Арк.	№ докум.	Підпис	Дата		

```
meta:

  type: object

  additionalProperties: false
```

Ми додаємо дану специфікацію у визначене поле, і як можемо бачити, система відреагувала і провалідувала дану конфігурацію

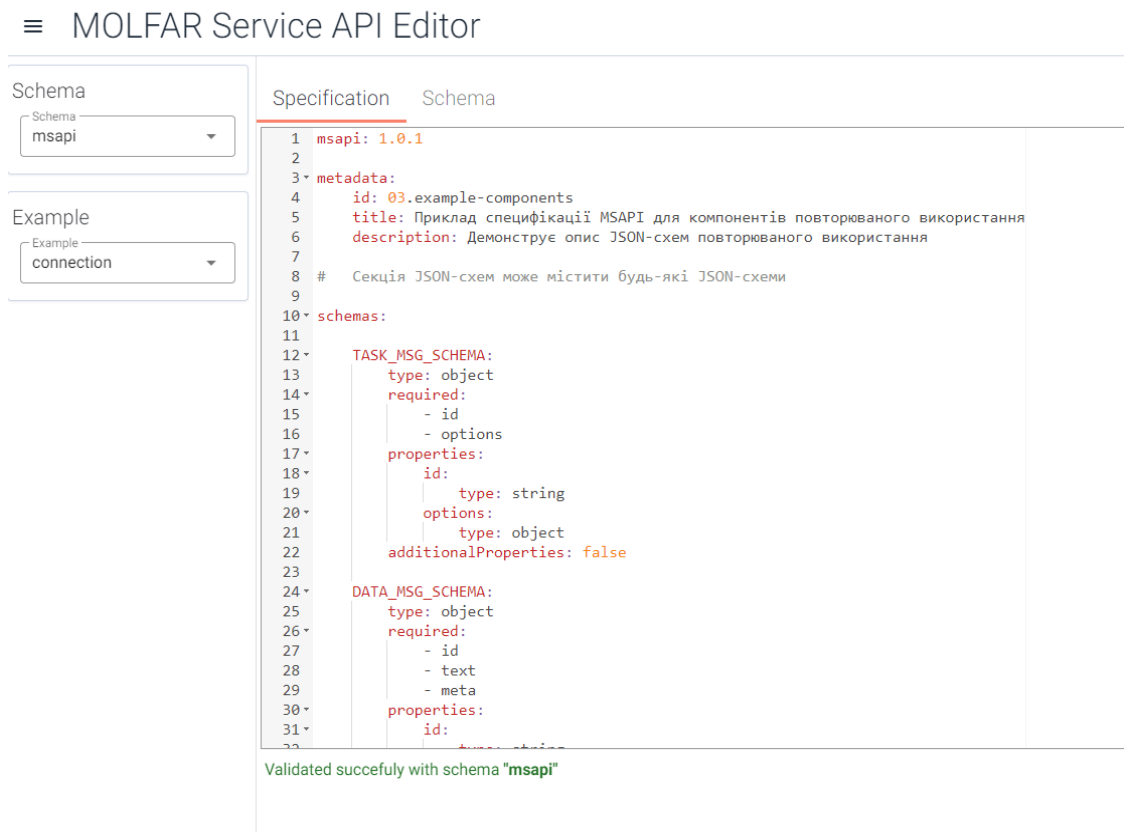


Рис. 3.35 Інтерфейс додатку, успішна валідація

У результаті, ми можемо бачити повністю розгорнуту схему:

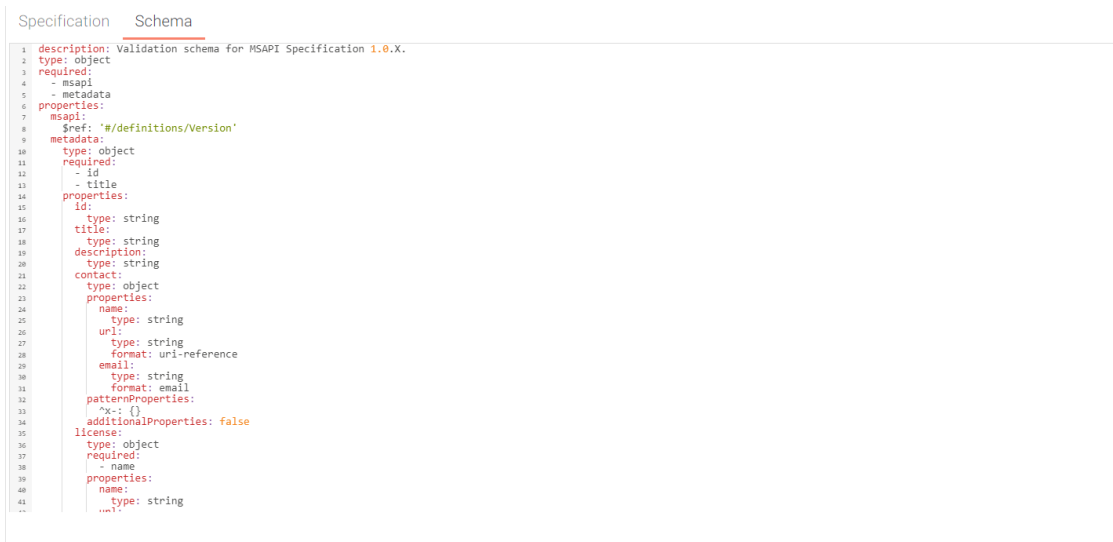


Рис. 3.35 Інтерфейс додатку, розгорнута схема

І тепер ця схема готова для використання у контейнері.

ВИСНОВКИ ДО РОЗДІЛУ 3

У цьому розділі була описана фаза імплементації бажаної системи. Було детально описано структуру Node js проєкта, конфігураційних файлів, залежностях на бібліотеки і інструменти, згенерованій документації основних компонентів. Були детально розглянуті класи, яким чином вони пов'язані і взаємодіють між собою. У результаті роботи, було отримано бібліотеку, що використовує специфікації, призначені для опису розподілених інформаційних систем з мікросервісною архітектурою, а також конвертує їх з YAML у JSON формати.

У кінці розділу була представлена робота додатку, що був розроблений протягом даної роботи. Приклади, представлені, дозволяють зрозуміти, як функціонує система, які параметри слід зазначати і передавати в методи і функції. Були додані результати роботи програми, які дають розуміння, що система налаштована успішно. Було також додано рисунки, де зазначене покриття тестами і результати їх перевірки. Мету було досягнуто.

					ІАЛЦ.467200.003 ПЗ	Арк.
						67
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

В процесі реалізації даної дипломної роботи було створено систему, а саме, програмний модуль контейнера мікросервісів. Дана система допомагає користувачам утворювати ланцюжки мікросервісів, інтегруючи їх один з одним, і отримуючи можливість утворювати з тих самих мікросервісів нові системи, значно підвищуючи рівень повторного використання коду. У першому розділі було розглянуто поняття розподіленої системи, її переваги та недоліки, було розглянуто можливі архітектурні підходи. У ході порівняння мікросервісної архітектури і SOA архітектури за рядом параметрів було виявлено, що на даному етапі мікросервісна архітектура має переваги у масштабованості, відмовостійкості, а також іншими, що були розглянуті у першому розділі. Також, як можлива платформа була розглянута AWS, але у зв'язку з тим, що дана платформа вимагає доволі високої зв'язності із середовищем, вона не підходить. В рамках другого розділу увага була сконцентрована на предметній області. призначені для опису розподілених інформаційних систем з мікросервісною архітектурою, передають конфігурації до програмних модулів. Було сформовано, яким чином формати передачі даних конвертуються один в одного. Була також розглянута взаємодія програмних модулів (мікросервісів). Ми розглянули деякі інструменти розробки, такі як мова програмування Javascript, технологічна платформа Node JS, протокол AMQP, а також бібліотеку OpenAPI. В ході роботи над третім розділом було розглянуто хід імплементації бажаної системи. Спочатку було розглянуто структуру проекту, детально було розглянуто всі компоненти системи. Далі були описані складові класи, принципи їх роботи і взаємодії. У результаті роботи, було отримано бібліотеку, що використовує специфікації, призначені для опису розподілених інформаційних систем з мікросервісною архітектурою, а також конвертує їх з YAML у JSON формати.

					ІАЛЦ.467200.003 ПЗ	Арк.
						68
Зм.	Арк.	№ докум.	Підпис	Дата		

У кінці розділу була представлена робота додатку, що був розроблений протягом даної роботи. Приклади, представлені, дозволяють зрозуміти, як функціонує система, які параметри слід зазначати і передавати в методи і функції. Були додані результати роботи програми, які дають розуміння, що система налаштована успішно. Було також додано рисунки, де зазначене покриття тестами і результати їх перевірки.

					ІАЛЦ.467200.003 ПЗ	Арк.
						69
Зм.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fleischmann, A.: Distributed Systems: Software Design and Implementation. Springer, Heidelberg
2. Distributed System [Рисунок] – Режим доступу до ресурсу: – https://ds.cs.luc.edu/_images/issues-3.jpg – Дата доступу: 07.04.2022
3. Microservices and it's definition [Електронний ресурс] – Режим доступу до ресурсу: – <https://martinfowler.com/articles/microservices.html> – Дата доступу: 05.04.2022
4. "Analytic Computational Complexity" in Computer, vol. 9, no. 05, pp. 39, 1976.
5. Papazoglou, M.P., van den Heuvel, WJ. Service oriented architectures: approaches, technologies and research issues. The VLDB Journal 16, 389–415 (2007).
6. SOA [Рисунок] – Режим доступу до ресурсу: – https://miro.medium.com/max/1400/1*wueZLPaGKr5iMOLHRfoYHQ.png – Дата доступу: 07.04.2022
7. Архітектура AWS та елементи його еко-системи [Рисунок] – Режим доступу до ресурсу: – https://miro.medium.com/max/1400/0*J5T1xunmLdlMB-t8.png – Дата доступу: 11.04.2022
8. Архітектура Prometheus [Рисунок] – Режим доступу до ресурсу: – https://grafana.com/static/img/docs/grafana-cloud/arch_diagrams/localprom.jpg – Дата доступу: 11.04.2022
9. Grafana setup [Електронний ресурс] – Режим доступу до ресурсу: – <https://grafana.com/docs/grafana/latest/basics/exemplars/> – Дата доступу: 30.04.2022
10. Grafana Dashboard [Рисунок] – Режим доступу до ресурсу: – <https://i0.wp.com/dotsandbrackets.com/wp->

- content/uploads/2017/01/grafana-
dashboard.jpg?resize=1172%2C600&ssl=1 – Дата доступу: 30.04.2022
11. Mullender, Sape J.. “Distributed systems.” (1989).
 12. Mehdizadeh, Abbas et al. “Multicast-Unicast Data Delivery Method” Journal of Network and Systems Management 22 (2013)
 13. Приклад широкоадресної комунікації [Рисунок] – Режим доступу до ресурсу: – <https://media.geeksforgeeks.org/wp-content/uploads/20201113210702/Broadcast.png> – Дата доступу: 30.04.2022
 14. Приклад мультиадресної комунікації [Рисунок] – Режим доступу до ресурсу: – <https://media.geeksforgeeks.org/wp-content/uploads/20201113210935/Multicast.png> – Дата доступу: 30.04.2022
 15. Приклад одноадресної комунікації [Рисунок] – Режим доступу до ресурсу: – <https://media.geeksforgeeks.org/wp-content/uploads/20201113211058/unicast.png> – Дата доступу: 30.04.2022
 16. Архітектура асинхронної моделі зв'язку між сервісами через протокол HTTP [Рисунок] – Режим доступу до ресурсу: – <https://docs.microsoft.com/ru-ru/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture> – Дата доступу: 6.05.2022
 17. Архітектура асинхронної моделі зв'язку між сервісами через Брокер повідомлень [Рисунок] – Режим доступу до ресурсу: – <https://docs.microsoft.com/ru-ru/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture> – Дата доступу: 6.05.2022
 18. Приклад архітектури з використанням gRPC [Рисунок] – Режим доступу до ресурсу: – <https://grpc.io/img/landing-2.svg> – Дата доступу: 6.05.2022

19. gRPC setup [Електронний ресурс] – Режим доступу до ресурсу: – <https://grpc.io/docs/what-is-grpc/introduction/> – Дата доступу: 6.05.2022
20. Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617
21. J.E. Allchin, M.S. McKendry. Synchronization and Recovery of Actions. In Proceedings of the Second Annual Symposium on Principles of Distributed Computing, pages 31.44. ACM
22. Zeng, Liangzhao et al. "QoS-aware middleware for Web services composition." IEEE Transactions on Software Engineering 30 (2004): 311-327.
23. Shusterman, Anatoly et al. "Prime+Probe 1, JavaScript 0: Overcoming Browser-based Side-Channel Defenses." USENIX Security Symposium (2021).
24. Merkel, Dirk. "Docker: lightweight Linux containers for consistent development and deployment." Linux Journal 2014 (2014): 2.
25. Grünwald, Elias et al. "TIRA: An OpenAPI Extension and Toolbox for GDPR Transparency in RESTful Architectures." 2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW) (2021): 312-319.
26. Donca, Ionut-Catalin et al. "Autoscaled RabbitMQ Kubernetes Cluster on single-board computers." 2020 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR) (2020): 1-6.

ДОДАТОК 1

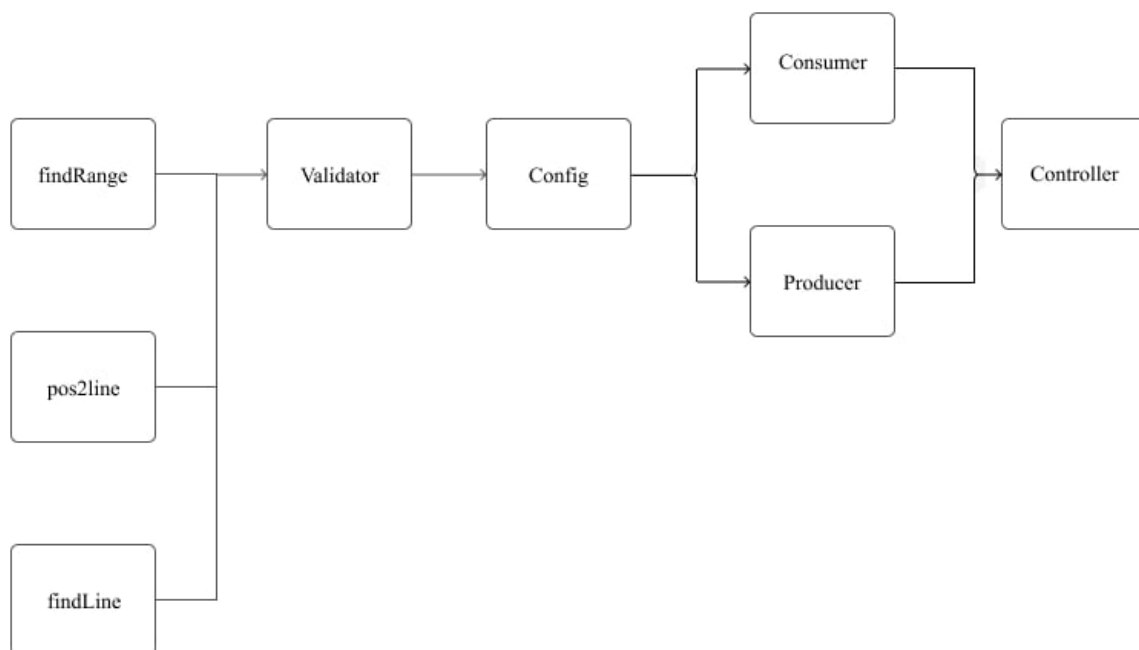
Програмний модуль контейнера мікросервісів

Структурна схема системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.004 Д1						
		№ докум.	Підпис	Дата							
Розробив		Василиненко Д. Д.			Програмний модуль контейнера мікросервісів Структурна схема системи			Літ.	Аркуш	Аркушів	
Перевірив		Болдак А. О.								1	1
								НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-84			
Н. Контр.		Сімоненко В. П.									
Затвердив											

ДОДАТОК 2

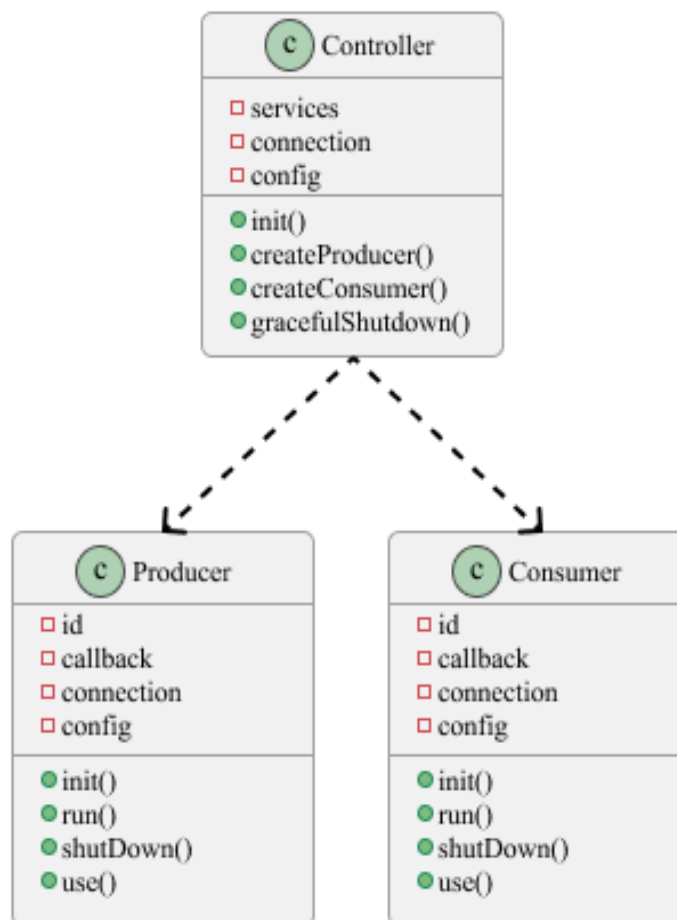
Програмний модуль контейнера мікросервісів

Функціональна схема (діаграма класів)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.005 Д2			
		№ докум.	Підпис	Дата				
Розробив	Василиненко Д. Д.				Програмний модуль контейнера мікросервісів Структурна схема системи	Літ.	Аркуш	Аркушів
Перевірив	Болдак А. О.						1	1
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-84		
Н. Контр.	Сімоненко В. П.							
Затвердив								

ДОДАТОК 3

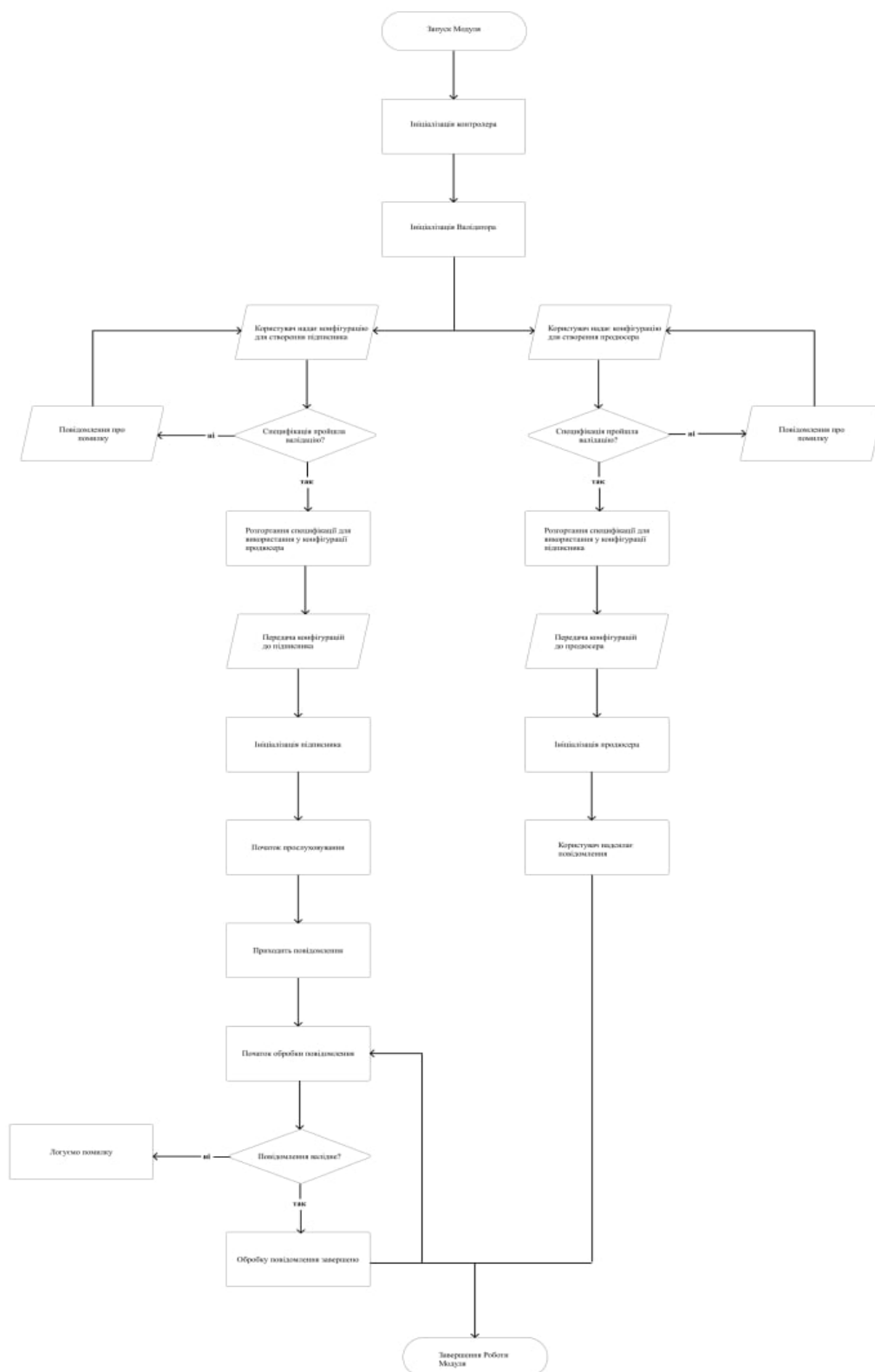
Програмний модуль контейнера мікросервісів

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 ДЗ

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.006 Д2								
		№ докум.	Підпис	Дата									
Розробив		Василиненко Д. Д.			Програмний модуль контейнера мікросервісів Алгоритм дій програмного забезпечення				Літ.	Аркуш	Аркушів		
Перевірив		Болдак А. О.									1	1	
									НТУУ КПІ ім. Ігоря				
Н. Контр.		Сімоненко В. П.							Сікорського, ФІОТ, ІП-84				
Затвердив													

ДОДАТОК 4

Програмний модуль контейнера мікросервісів

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 22

Київ 2022

msapi: 1.0.1

metadata:

id: 01.example-min-spec

title: Приклад специфікації MSAPI

description: Демонструє мінімальний склад специфікації MSAPI

msapi: 1.0.1

metadata:

id: 02.example-components

title: Приклад специфікації MSAPI для компонентів повторюваного використання

description: Демонструє опис налаштувань повторюваного використання

Секція налаштувань може містити будь-які описи (прости і об'єкти)

Доступ до них здійснюється за доповогою посилань \$ref

Наприклад:

- Локальне посилання (в межах поточної специфікації) - \$ref:

"#/settings/amqp_url"

- Зовнішнє посилання (в іншій специфікації) - \$ref: "02.example-components/#!/settings/amqp_url"

settings:

Приклад налаштування - рядка

amqp_url:

amqps://xoilebqg:Nx46t4t9cxQ2M0rF2rIyZPS_xbAhmJIG@hornet.rmq.cloudamqp.com/xoilebqg

Приклад налаштування - об'єкта

health_control:

exchange_name: ms_health

interval: 1000 #ms

msapi: 1.0.1

metadata:

id: 03.example-components

title: Приклад специфікації MSAPI для компонентів повторюваного використання

description: Демонструє опис JSON-схем повторюваного використання

Секція JSON-схем може містити будь-які JSON-схеми

schemas:

TASK_MSG_SCHEMA:

type: object

required:

- id

- options

properties:

id:

type: string

options:

type: object

additionalProperties: false

DATA_MSG_SCHEMA:

type: object

required:

- id

- text

```

        - meta
    properties:
        id:
            type: string
        text:
            type: string
        meta:
            type: object
        additionalProperties: false
msapi: 1.0.1

metadata:
    id: 04.example-components
    title: Приклад специфікації MSAPI для компонентів повторюваного
використання
    description: Демонструє опис обмінників даних повторюваного використання

# Секція обмінників може містити описи обмінників даних

exchanges:

    TASK_EXCHANGE:
        name: task_exch
        mode: fanout
        options:
            durable: true

    SCRAPPED_MSG_EXCHANGE:
        name: scrapper_exch
        mode: fanout

msapi: 1.0.1

metadata:
    id: 03.example-components
    title: Приклад специфікації MSAPI для компонентів повторюваного
використання
    description: Демонструє опис черг повторюваного використання

# Секція черг може містити описи черг повідомлень

queues:

    # Опис черги має містити назву черги, налаштування обмінника,
    # а також черги та процесу вибірки повідомлень.
    TASK_QUEUE:
        # Назва черги
        name: task_queue
        exchange:
            name: task_exch
            mode: fanout
        # Налаштування обмінника
        options:
            durable: true
        options:
            # Налаштування черги та процесу вибірки повідомлень
            noAck: false
            prefetch: 1
            exclusive: true
msapi: 1.0.1

```

```
metadata:
  id: 06.example-use-refs
  title: Приклад специфікації MSAPI для компонентів повторюваного
використання
  description: Демонструє використання посилань
```

```
settings:

  task_exchange_name: task_exch

  task_exchange_options:
    durable: true

  task_queue_name: task_queue

  task_queue_options:
    noAck: false
    prefetch: 1
    exclusive: true
```

```
exchanges:

  TASK_EXCHANGE:
    name:
      $ref: "#/settings/task_exchange_name"
    mode: fanout
    options:
      $ref: "#/settings/task_exchange_options"
```

```
queues:

  TASK_QUEUE:
    name:
      $ref: "#/settings/task_queue_name"
    exchange:
      $ref: "#/exchanges/TASK_EXCHANGE"
    options:
      $ref: "#/settings/task_queue_options"
```

```
msapi: 1.0.1
```

```
metadata:
  id: 07.example-produce
  title: Приклад специфікації MSAPI для компонентів повторюваного
використання
  description: Демонструє опис публікувальника даних
```

```
produces:
  # Опис публікувальника має містити налаштування підключення до брокера
повідомлень,
  # опис обмінника, в якому будуть публікуватися повідомлення,
  # а також опис схеми повідомлень, які будуть публікуватися цим
публікувальником
  TASK_SCHEDULER:
    # налаштування підключення до брокера повідомлень
    amqp:
      url:
        $ref: "02.example-components/#!/settings/amqp_url"
    # опис обмінника, в якому будуть публікуватися повідомлення
    exchange:
```

```

    $ref: "06.example-use-refs/#!/exchanges/TASK_EXCHANGE

    # опис схеми повідомлень
    message:
        $ref: "03.example-components/#!/schemas/TASK_MSG_SCHEMA"
msapi: 1.0.1

metadata:
    id: 08.example-consume
    title: Приклад специфікації MSAPI для компонентів повторюваного
    використання
    description: Демонструє опис споживача даних

consumes:
    # Опис споживача має містити налаштування підключення до брокера
    повідомлень,
    # опис черги повідомлень,
    # а також опис схеми повідомлень
    TASK_CONSUME:
        # налаштування підключення до брокера повідомлень
        amqp:
            url:
                $ref: "02.example-components/#!/settings/amqp_url"
            # опис черги повідомлень
            queue:
                $ref: "06.example-use-refs/#!/queues/TASK_QUEUE"
            # опис схеми повідомлень
            message:
                $ref: "03.example-components/#!/schemas/TASK_MSG_SCHEMA"
msapi: "1.0.1"

metadata:
    id: "ms_01_global_settings"
    title: "Global settings"
    description: Default queue and exchange settings

components:
    settings:
        amqp:
            url:
                "amqps://xoilebqg:Nx46t4t9cxQ2M0rF2rIyZPS_xbAhmJIG@hornet.rmq.cloudamqp.com/xoilebqg"

            health:
                amqp:
                    $ref: "#/components/settings/amqp"
                produce:
                    exchange:
                        name: health_messages
                    time_interval: 30000
msapi: "1.0.1"

metadata:
    id: "ms_02_defaults"
    title: "Default queue and exchange settings"
    description: Default queue and exchange settings

components:

```

```
#####
```

```

        DEFAULT:                                # an example of exchange definition
            mode: fanout                          # or "topic" or "direct" - required
type of exchange
            options:                             # optional exchange options
                durable: true                    # (boolean): if true, the exchange
will survive broker restarts. Defaults to true.
                autoDelete: false              # (boolean): if true, the exchange
will be destroyed once the number of bindings for which it is the source drop
to zero. Defaults to false.

```

```
#####
```

```

    queues:
        DEFAULT:                                # an example of queue definition
            exchange:
                $ref: "#/components/exchanges/DEFAULT"
            options:
                exclusive: false                # if true, scopes the queue to the
connection (defaults to false)
                durable: true                  # if true, the queue will survive
broker restarts, modulo the effects of exclusive and autoDelete; this
defaults to true if not supplied, unlike the others
                autoDelete: false             # if true, the queue will be deleted
when the number of consumers drops to zero (defaults to false)
                noAck: false                  # (boolean): if true, the broker
won't expect an acknowledgement of messages delivered to this consumer; i.e.,
it will dequeue messages as soon as they've been sent down the wire. Defaults
to false (i.e., you will be expected to acknowledge messages).

```

```
#####
```

```

    consumes:
        DEFAULT:
            queue:
                $ref: "#/components/queues/DEFAULT"

```

```
#####
```

```

    produces:
        DEFAULT:
            exchange:
                $ref: "#/components/exchanges/DEFAULT"

```

msapi: "1.0.1"

metadata:

id: "ms_03_components_message_processing"

title: "Shared components for scrappers"

description: >

Defines the components for task messages and for multimedia messages.

Scrapers consume task messages, scrap sources, and create multimedia messages.

Normalizer consumes dirty multimedia messages, handles it, and creates normalized messages.

All parameters, exchange definitions and queue definitions will be set to

default or overwritten by expanding in the parent spec.

components:

#####

schemas:

TASK_MESSAGE_SCHEMA:

type: object

required:

- id
- timestamp
- options

properties:

id:

type: string

timestamp:

type: string

options:

type: object

MEDIA_MESSAGE_SCHEMA:

type: object

required:

- id
- timestamp
- scrapper_instance
- data

properties:

id:

type: string

timestamp:

type: string

scrapper_instance:

type: string

data:

type: object

required:

- text

properties:

meta:

type: object

text:

type: string

#####

consumes:

TASK_QUEUE:

message:

\$ref: "#/components/schemas/TASK_MESSAGE_SCHEMA"

MEDIA_MESSAGES:

message:

\$ref: "#/components/schemas/MEDIA_MESSAGE_SCHEMA"

#####

produces:

```

    MEDIA_MESSAGES:
      message:
        $ref: "#/components/schemas/MEDIA_MESSAGE_SCHEMA"

#####

msapi: "1.0.1"

metadata:

  id: "ms_04_scraper_service"

  title: "Scraper Service type definition"

  description: >
    Defines the type of scraper service.
    Scrapers consume task messages, scrap sources, and create multimedia
    messages.
    Normalizer consumes dirty multimedia messages, handles it, and
    creates
    normalized messages.Normalizer consumes dirty multimedia messages,
    handles it, and creates
    normalized messages.

  repo: "@molifar/scraper"

service:

  config:

    amqp:
      $ref: "ms_01_global_settings/#!/components/settings/amqp"

    consume:
      amqp:
        $ref: "#/service/config/amqp"
      queue:
        exchange:
          name: scrapper_task_messages
    produce:
      amqp:
        $ref: "#/service/config/amqp"
      exchange:
        name:
          schema:
            type: string

    health_check:
      $ref: "ms_01_global_settings/#!/components/settings/health"

  consume:
    def:
      $ref: "ms_02_defaults/#!/components/consumes/default"
    extendedBy:
      - $ref:
"ms_03_components_message_processing/#!/components/consumes/TASK_QUEUE"
      - $ref: "#/service/config/consume"

```

```

    produce:
      def:
        $ref: "ms_02_defaults/#!/components/produces/default"
      extendedBy:
        - $ref:
"ms_03_components_message_processing/#!/components/produces/MEDIA_MESSAGES"
        - $ref: "#!/service/config/produce"

msapi: "1.0.1"

metadata:

  id: "http://localhost:8081/spec/ms_example_normalizer_service.yaml"

  title: "Normalizer Service type definition"

  description: >
    Defines the type of normalizer service.
    Normalizer consumes dirty multimedia messages, handles it, and
creates
    normalized messages.

  repo: "@molnar/normalizer"

service:

  config:

    amqp:
      $ref: "ms_01_global_settings/#!/components/settings/amqp"

    consume:
      amqp:
        $ref: "#!/service/config/amqp"
      queue:
        exchange:
          name:
            schema:
              type: string
        produce:
          amqp:
            $ref: "#!/service/config/amqp"
          exchange:
            name:
              schema:
                type: string

      health_check:
        $ref: "ms_01_global_settings/#!/components/settings/health"

    consume:
      def:
        $ref: "ms_01_defaults/#!/components/consumes/DEFAULT"
      extendedBy:
        - $ref:
"ms_03_components_message_processing/#!/components/consumes/MEDIA_MESSAGES"
        - $ref: "#!/service/config/consume"

```

```

    produce:
      def:
        $ref: "ms_01_defaults/#!/components/Produces/default"
      extendedBy:
        - $ref:
            "/ms_03_components_message_processing/#!/components/produces/MEDIA_MESSAGES"
        - $ref: "#!/service/config/produce"

msapi: "1.0.1"

metadata:

  id: "http://localhost:8081/spec/ms_example_workflow.yaml"

  title: "Define a workflow for handling multimedia messages"

  description: >
    Defines a chain of two services

workflow:

  - instance:
      $ref: "ms_04_scraper_service/#!/service"
      name: SCRAPPER
      configuredBy:
        produce:
          exchange:
            name: dirty_messages

  - instance:
      $ref: "ms_05_normalizer_service/service"
      configuredBy:
        consume:
          queue:
            exchange:
              name: dirty_messages
        produce:
          exchange:
            name: normalized_messages

title: Підсхема "connection"
description: Налаштування з'єднання з брокером повідомлень
subschema: msapi.components.connection
type: object

required:
  - url

patternProperties:
  '^url$':
    title: url
    description: url з'єднання з брокером повідомлень
    type: string
    format: uri-reference

errorMessage:
  type: Налаштування з'єднання повинно бути об'єктом.
  required: url з'єднання є обов'язковим.

```

```

    properties:
      url: url повинен бути коректним RabbitMQ url з'єднання.
      title: Підсхема "consume"
      description: Налаштування споживача
      type: object
      additionalProperties: false
      patternProperties:

        '^amqp':
          title: amqp
          description: Налаштування брокера повідомлень
          $ref: "./connection"

        '^queue$':
          title: queue
          description: Налаштування черги
          oneOf:
            - $ref: "./queue"
              description: Налаштування черг
            - $ref: "#/definitions/Reference"
              description: Посилання на налаштування черги

        '^message$':
          title: message
          description: Налаштування повідомлень
          oneOf:
            - $ref: "#/definitions/Schema"
              description: Посилання на налаштування повідомлень
            - $ref: "#/definitions/Reference"
              description: Посилання на налаштування повідомлень

title: Підсхема "exchange-options"
description: Налаштування обмінника
subschema: msapi.components.exchange.exchange-options

type: object
patternProperties:

  '^durable$':
    title: durable
    description: >
      Налаштування обмінника. Якщо встановлено значення ``true`` обмінник
      переживе перезапуск брокера

    type: boolean
    default: true

  '^autoDelete$':
    title: autoDelete
    description: >
      Налаштування обмінника. Якщо встановлено значення ``true``, обмінник
      буде знищено, коли
      кількість прив'язок, для яких він є джерелом, знизиться до нуля
    type: boolean
    default: false # if true, scopes the queue to the connection
    (defaults to false)

  '^persistent$':
    title: persistent
    description: >

```

```
        Налаштування обмінника. Якщо встановлено значення ``true``,  
повідомлення переживе  
        перезапуск брокера за умови, що воно знаходиться в  
черзі, яка також витримує перезапуски. Відповідає і  
перевизначає властивість ``deliveryMode``  
        type: boolean  
        default: true  
title: Підсхема "exchange"  
description: Налаштування обмінника  
subschemata: msapi.components.exchange
```

```
type: object  
additionalProperties: false
```

```
patternProperties:
```

```
  '^name$':  
    title: name  
    description: Ім'я обмінника  
    type: string  
  
  '^mode$':  
    title: mode  
    description: Тип обмінника  
    enum:  
      - fanout  
      - direct  
      - topic  
    default: "fanout"  
  
  '^options$':  
    title: options  
    description: Налаштування обмінника  
    $ref: "../exchange-options"
```

```
# components schema  
title: Підсхема "components"  
description: Компоненти повторюваного використання  
subschemata: msapi.components  
type: object  
additionalProperties: false
```

```
patternProperties:
```

```
  '^exchanges$':  
    title: exchanges  
    description: Налаштування обмінників повідомлень  
    type: object  
    patternProperties:  
      '^[a-zA-Z0-9\\._-]+$':  
        title: Будь-який ідентифікатор  
        description: Налаштування обмінника  
        oneOf:  
          - $ref: '#/definitions/Reference'  
            description: Посилання на налаштування обмінника  
          - $ref: '../exchange'  
            description: Налаштування обмінника  
  
  '^queues$':  
    title: queues
```

```

description: Налаштування черг повідомлень
type: object
patternProperties:
  '^[a-zA-Z0-9\\.\\-_]+$':
    title: Будь-який ідентифікатор
    description: Налаштування черги
    oneOf:
      - $ref: '#/definitions/Reference'
        description: Посилання на налаштування черги
      - $ref: './queue'
        description: Налаштування черги

'^schemas$':
  title: schemas
  description: Опис типів повідомлень (JSON Schema)
  type: object
  patternProperties:
    '^[a-zA-Z0-9\\.\\-_]+$':
      title: Будь-який ідентифікатор
      description: Опис типу повідомлень
      oneOf:
        - $ref: '#/definitions/Reference'
          description: Посилання на опис типу повідомлень
        - $ref: '#/definitions/Schema'
          description: Опис типу повідомлень

'^consumes$':
  title: consumes
  description: Налаштування прослуховувачів повідомлень
  type: object
  patternProperties:
    '^[a-zA-Z0-9\\.\\-_]+$':
      title: Будь-який ідентифікатор
      description: Налаштування прослуховувача повідомлень
      oneOf:
        - $ref: '#/definitions/Reference'
          description: Посилання на налаштування прослуховувача повідомлень
        - $ref: './consume'
          description: Налаштування прослуховувача повідомлень

'^produces$':
  title: produces
  description: Налаштування публікувальників повідомлень
  type: object
  patternProperties:
    '^[a-zA-Z0-9\\.\\-_]+$':
      title: Будь-який ідентифікатор
      description: Налаштування публікувальника повідомлень
      oneOf:
        - $ref: '#/definitions/Reference'
          description: Посилання на налаштування публікувальника
повідомлень
        - $ref: './produce'
          description: Налаштування публікувальника повідомлень

'^settings$':
  type: object
  title: settings
  description: Налаштування, які можна повторювано використовувати будь-де.

```

```
title: Підсхема "produce"
description: Налаштування публікувальника
type: object
additionalProperties: false
subschema: msapi.components.produce
```

```
patternProperties:
```

```
  '^amqp':
    title: amqp
    description: Налаштування брокера повідомлень
    $ref: "../connection"
    definitions: Налаштування з'єднання

  '^exchange$':
    title: exchange
    description: Налаштування обмінника
    oneOf:
      - $ref: "../exchange"
        description: Налаштування обмінника

      - $ref: "#/definitions/Reference"
        description: Посилання на налаштування обмінника

  '^message$':
    title: message
    description: Налаштування повідомлення
    oneOf:
      - $ref: "#/definitions/Schema"
        description: Посилання на налаштування повідомлень
      - $ref: "#/definitions/Reference"
        description: Посилання на налаштування повідомлень
```

```
const ast = require("yaml-ast-parser")
const YAML = require("js-yaml")
```

```
const Ajv = require("ajv")//.default
const ajv = new Ajv({ allErrors: true })
require("ajv-formats")(ajv)
require("ajv-errors")(ajv)
ajv.addKeyword("subschema")
```

```
const findRange = (node, path) => {
  let p = path.map( d => d )
  let key = (node.key) ? p.splice(0,1)[0] : p[0]
  if(node.key && node.key.value == key && p.length == 0) return [
node.key.startPosition,node.key.endPosition ]
```

```
    if(node.value && node.value.items ) {
      key = p.splice(0,1)[0]
      // console.log(JSON.stringify(key),node.value.items)
      if(p.length == 0) return [
node.value.items[key].startPosition, node.value.items[key].endPosition ]

      let res = findRange(node.value.items[key], p)
```

```

        if (res) return res
    }

    if(node.value && node.value.mappings) {
        for (let n in node.value.mappings){
            let res = findRange(node.value.mappings[n], p)
            if (res) return res
        }
    }

    if(node.mappings) {
        for (let n in node.mappings){
            let res = findRange(node.mappings[n], p)
            if (res) return res
        }
    }
}

const pos2line = (str, pos) => str.substr(0,pos[0]).split("\n").length -1

const findLine = (astDoc, path, doc) => {
    path = path.split("/").filter(d => d).map( d => {
        if(Number.isNaN(Number.parseInt(d))) return d
        return Number.parseInt(d)
    })

    let range = ( path.length > 0 ) ? findRange(astDoc, path) : [0, 0]
    return pos2line(doc, range)
}

const Validator = class {
    constructor (schema){
        this.v = ajv.compile(schema)
    }

    validate(doc){
        doc = doc.replace(/\t/gm," ")
        let astDoc = ast.load(doc)

        let json

        this.errors = null

        try {
            json = YAML.load(doc)
        } catch (e) {
            this.errors = [{
                line: e.mark.line,
                snippet: e.mark.snippet,
                reason: e.reason
            }]
            return false
        }
    }
}

```

```

        if( json ) {
            let result = this.v(json)
            if ( result == false) {
                this.errors = this.v.errors.map( e => ({
                    line:
findLine(astDoc,e.instancePath,doc),
                    snippet: e.instancePath,
                    reason: e.message
                }))
                return false
            } else {
                return true
            }
        }

        return false
    }
}

module.exports = Validator
const Validator = require("../utils/schema-validator")
const msapiSchema = require("../../index").msapi.json
const fs = require("fs")
const path = require("path")

const v = new Validator(msapiSchema)

describe('Тести для перевірки підсхеми "metadata"', () => {

    test('Повинен повернути помилку валідації для пустої специфікації', () => {
        {
            let result =
v.validate(fs.readFileSync(path.resolve(__dirname,
"invalid/empty.yaml")).toString())
            expect(result).toEqual(false)
        });

        test('Повинен повернути помилку валідації для специфікації з відсутною
версією msapi', () => {
            let result =
v.validate(fs.readFileSync(path.resolve(__dirname, "invalid/without-msapi-
version.yaml")).toString())
            expect(result).toEqual(false)
        });

        test('Повинен успішно валідувати специфікацію з коректно заповненими
метаданими', () => {
            let result =
v.validate(fs.readFileSync(path.resolve(__dirname,
"valid/metadata.yaml")).toString())
            expect(result).toEqual(true)
        });

    });

    msapi: "1.0.1"

    metadata:

        id: "http://localhost:8081/spec/ms_example_workflow.yaml"

```

```

    title: "Define a workflow for handling multimedia messages"

    description: >
        Defines a chain of two services
queue:
    name:                                # optional name of queue

    exchange:
        name:                            # required name of exchange

        mode: fanout                      # required type of exchange. Available
values "fanout" or "topic" or "direct"

        options: # !!! options for assertExchange method call
            durable: true                 # (boolean): if true, the exchange will
survive broker restarts. Defaults to true.
            autoDelete: false            # (boolean): if true, the exchange will
be destroyed once the number of bindings for which it is the source drop to
zero. Defaults to false.

        options: # !!! options for assertQueue method call
            noAck: false
            exclusive: false              # if true, scopes the queue to the
connection (defaults to false)
            durable: true                 # if true, the queue will survive broker
restarts, modulo the effects of exclusive and autoDelete; this defaults to
true if not supplied, unlike the others
            autoDelete: false            # if true, the queue will be deleted when
the number of consumers drops to zero (defaults to false)
            prefetch: 1

exchange:
    name:                                # required name of exchange

    mode: fanout                          # required type of exchange. Available values
"fanout" or "topic" or "direct"

    options: # !!! options for assertExchange method call
        durable: true                    # (boolean): if true, the exchange will
survive broker restarts. Defaults to true.
        autoDelete: false                # (boolean): if true, the exchange will be
destroyed once the number of bindings for which it is the source drop to
zero. Defaults to false.
# service schema
title: Підсхема "service"
description: Налаштування типу сервісу
subschema: msapi.service

type: object
additionalProperties: false

patternProperties:
    '^config$':
        title: config
        description: >
            Дані для налаштування сервісу
        type: object

    '^consume$':
        title: consume

```

```

    description: Налаштування прослуховувача повідомлень
    oneOf:
      - $ref: "../components/consume"
        description: Налаштування прослуховувача повідомлень
      - $ref: "#/definitions/Extensible"
        description: Посилання на налаштування прослуховувача повідомлень

    '^produce$':
      title: produce
      description: Налаштування публікувальника повідомлень
      oneOf:
        - $ref: "../components/produce"
          description: Налаштування публікувальника повідомлень
        - $ref: "#/definitions/Extensible"
          description: Посилання на налаштування публікувальника повідомлень

# workflow schema
title: Підсхема "workflow"
description: >
    Налаштування робочого процесу. Визначає сукупність екземплярів сервісів та
    їх
    налаштування
  subschema: msapi.workflow

type: array

items:
  type: object

  required:
    - instance

  additionalProperties: false

  patternProperties:

    '^instance$':
      title: instance
      description: Налаштування екземпляру сервісу
      oneOf:
        - $ref: '../service/index'
          description: Налаштування екземпляру сервісу
        - $ref: '#/definitions/Reference'
          description: Посилання на налаштування екземпляру сервісу

    '^name$':
      title: name
      description: Назва екземпляру сервісу
      type: string

    '^configuredBy$':
      title: configuredBy
      description: Налаштування екземпляру сервісу
      type: object

# service schema
title: Підсхема "service"
description: Налаштування типу сервісу
  subschema: msapi.service

type: object
additionalProperties: false

```

```

patternProperties:
  '^config$':
    title: config
    description: >
      Дані для налаштування сервісу
    type: object

  '^consume$':
    title: consume
    description: Налаштування прослуховувача повідомлень
    oneOf:
      - $ref: "../components/consume"
        description: Налаштування прослуховувача повідомлень
      - $ref: "#/definitions/Extensible"
        description: Посилання на налаштування прослуховувача повідомлень

  '^produce$':
    title: produce
    description: Налаштування публікувальника повідомлень
    oneOf:
      - $ref: "../components/produce"
        description: Налаштування публікувальника повідомлень
      - $ref: "#/definitions/Extensible"
        description: Посилання на налаштування публікувальника повідомлень

type: object
required:
  - url
properties:
  url:
    type: string
errorMessage:
  type: Connection options should be an object.
  required: Connection url is required.
  properties:
    url: Connection url should be a valid RabbitMQ connection url.
title: Consumer options
description: Bla bla bla

type: object
required:
  - amqp
  - queue
  - message
properties:
  amqp:
    type: object
    required:
      - url
    properties:
      url:
        type: string

    errorMessage:
      properties:
        url: url should be a valid AMQP-connection string

  queue:
    type: object

```

```
title: Consumes queue options
description: Options for queue configuration
required:
  - exchange
  - options
properties:
  name:
    anyOf:
      -
        type: string
      -
        type: "null"

exchange:
  type: object
  required:
    - name
    - mode
    - options
  properties:
    name:
      type: string
    mode:
      type: string
    options:
      type: object
      required:
        - durable
        - autoDelete
      properties:
        durable:
          type: boolean
        autoDelete:
          type: boolean
    errorMessage:
      properties:
        name: Exchange name is required. It should be a string.
options:
  type: object
  required:
    - noAck
    - exclusive
    - durable
    - autoDelete
    - prefetch
  properties:
    noAck:
      type: boolean
    exclusive:
      type: boolean
    durable:
      type: boolean
    autoDelete:
      type: boolean
    prefetch:
      type: number
message:
  type: object
  required:
    - options
  properties:
```

```

    options:
      type: object
      required:
        - noAck
        - exclusive
      properties:
        noAck:
          type: boolean
        exclusive:
          type: boolean

type: object
required:
  - exchange
  - message
properties:
  exchange:
    type: object
    required:
      - name
      - mode
      - options
    properties:
      name:
        type: string
      mode:
        type: string
      options:
        type: object
        required:
          - durable
          - autoDelete
        properties:
          durable:
            type: boolean
          autoDelete:
            type: boolean
  errorMessage:
    properties:
      name: Exchange name is required. It should be a string.

message:
  type: object
  required:
    - options
  properties:
    options:
      type: object
      required:
        - persistent
      properties:
        persistent:
          type: boolean

errorMessage:
  properties:
    exchange: Exchange should be an object
    message: Message should be an object

```