УДК 004.4'22 DOI: 10.31673/2412-4338.2024.014555

Терещенко Олександр Ігорович
Національний університет «Одеська політехніка», Одеса
ORCID 0000-0003-4510-5255

Корецька Вікторія Олександрівна
Державний університет інформаційно-комунікаційних технологій, Київ
ORCID 0000-0003-1570-7669

Трінтіна Наталя Альбертівна
Державний університет інформаційно-комунікаційних технологій, Київ
ORCID 0000-0001-6827-4030

Олєнєва Ксенія Миколаївна
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
ORCID 0000-0002-5576-4601

СТВОРЕННЯ ІНСТРУМЕНТАРІЮ ДЛЯ СПРОЩЕННЯ ПОБУДОВИ МІКРОСЕРВІСНИХ ДОДАТКІВ

Анотація: *Предметом дослідження цієї статі є мікросервісна архітектура та її сучасний стан розвитку. Метою цієї статті є розробка графічного інструментарію проектування структури мікросервісних додатків, який б значно полегшував створення такого програмного забезпечення, минаючи основні проблеми, які породжуються при неправильному застосуванні цієї архітектури.*

Основними задачами було категоризувати існуючі шаблони проектування мікросервісної архітектури, виділити критично важливі групи та розглянути конкретні шаблони з цих груп. Наступним кроком була інтеграція цих шаблонів в створений графічний інструментарій. Кінцевою задачею було тестування створеного інструментарію та перевірка його працездатності на прикладі медичної системи з використанням предметно-орієнтовного підходу до створення програмного забезпечення. В результаті виконання поставлених задач було розроблено графічний інструментарій проектування структури мікросервісних додатків, який частково автоматизує процес створення програмного забезпечення завдяки наданню широкого набору інструментів, які вирішують такі задачі, як забезпечення виявлення, безпеки, комунікації, взаємодії та спостережуваності мікросервісів.

Наукова новизна цієї роботи полягає у тому, що подібного комплексного рішення для побудови мікросервісних додатків на ринку не існує на даний момент. При цьому, актуальність створеного графічного інструментарію висока, так як мікросервісна архітектура знаходить все більше прихильників серед розробників програмного забезпечення завдяки перевагам, які вона надає. Але ці переваги досягаються лише при правильній інтеграції цієї архітектури у додаток, тому існує потреба в інструментах, які беруть на себе частину цієї роботи та реалізують її правильним чином. Створений інструментарій генерує структуру мікросервісного додатку на мові програмування JavaScript та платформі NodeJS.

Ключові слова: *мікросервісна архітектура, шаблони проектування, графічний інструментарій, предметно-орієнтований підхід, виявлення мікросервісів*

Tereshchenko Oleksandr
Odessa Polytechnic National University, Odessa
ORCID 0000-0003-4510-5255

Koretska Viktoriia
State University of Information and Communication Technologies, Kyiv
ORCID 0000-0003-1570-7669

Trintina Natalia
State University of Information and Communication Technologies, Kyiv
ORCID 0000-0001-6827-4030

Olienieva Kseniia
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
ORCID 0000-0002-5576-4601

CREATION OF TOOLKIT TO SIMPLIFY THE CONSTRUCTION OF MICROSERVICE APPS

Abstract: *The subject of this article is a microservice architecture and its current state of development. The purpose of this article is to develop a graphical toolkit for designing the structure of microservice applications, which would greatly facilitate the creation of such software, bypassing the main problems that arise from the misapplication of this architecture.*

The main tasks were to categorize the existing templates for designing microservice architecture, identify critical groups and consider specific templates from these groups. The next step was to integrate these templates into the created graphical toolkit. The ultimate task was to test the created tookit and check its efficiency on the example of the medical system using a subject-oriented approach to software development. As a result of the tasks, a graphical toolkit for designing the structure of microservice applications was developed, which partially automates the software development process by providing a wide range of tools that solve such tasks as detection, security, communication, interaction and monitoring of microservices.

The scientific novelty of this work is that such a comprehensive solution for building microservice applications on the market does not exist at the moment. At the same time, the relevance of the created graphical tools is high, as the microservice architecture is finding more and more supporters among software developers due to the advantages it provides. But these benefits are only achieved with the right integration of this architecture into the application, so there is a need for tools that take on some of this work and implement it properly. The created toolkit generates the structure of the microservice application in the JavaScript programming language and the NodeJS platform.

Keywords: *microservice architecture, design templates, graphic toolkit, subject-oriented approach, detection of microservices*

1. Вступ.

Мікросервісна архітектура набуває все більшої популярності серед розробників і це не випадково. Мікросервіси мають ряд переваг над монолітними додатками, які роблять програмне забезпечення більш простим у розробці та надійним.

Розглянемо лише деякі найголовніші переваги, які надає мікросервісна архітектура:

- легкість масштабування мікросервісів [1];
- можна розгортувати мікросервіси незалежно;
- можна організувати окремі групи розробників, які будуть працювати над конкретними мікросервісами;
- можна експериментувати з технологіями [2];
- висока надійність, так як помилки локалізуються та ізолюються в мікросервісі.

Перехід з монолітної архітектури на мікросервісну не завжди є простим процесом. Побудова мікросервісних додатків потребує від розробника високого рівня знань, інакше можуть виникнути слабкі місця в додатку. Тому виникає потреба в інструментарії, який б полегшував створення мікросервісних додатків, беручи на себе відповідальність за реалізацію певних функцій та дозволяючи розробнику концентруватися на реалізації бізнес-логіці. Крім цього, створення подібного інструментарію оптимізує час, необхідний на розробку програмного забезпечення.

Вищесказане обумовлює актуальність та необхідність проведення досліджень і розробок у цьому напрямі.

2. Аналіз літературних даних і постановка проблеми.

В роботі [1] розглядаються основні мікросервісні шаблони, які отримали широке розповсюдження. Цей аналіз дав змогу зрозуміти, яким функціям мікросервісної архітектури слід приділити найвищу увагу та застосувати їх в створеному інструментарії.

В роботі [2] розглядається процес розвитку мікросервісної архітектури, зокрема приділяється увага її перспективам. Перелік інструментів, наведений в цій роботі, дозволив значно розширити можливості створеного інструментарію.

В роботі [3] розглядаються методи забезпечення моніторингу та тестування мікросервісних додатків. Отриманий аналіз допоміг реалізувати механізми моніторингу в інструментарії.

Робота [4] розглядає мікросервісну архітектуру з аспекту міграції з монолітної архітектури, аналізуються найкращі підходи. Це дозволило краще зрозуміти, які сильні сторони має мати інструментарій, для виконання міграцій додатків.

В роботі [5] розглядається шаблони розбиття додатків на мікросервіси. Найкращим варіантом, як зазначено в статті, є предметно-орієнтований підхід до виділення зв'язних контекстів. Кожен зв'язний контекст ідеально підходить для виділення його в мікросервіс, саме тому інструментарій проектувався з урахування цього підходу.

3. Мета і задачі дослідження.

Метою дослідження є створення графічного інструментарію для полегшення розробки мікросервісних додатків.

Для досягнення поставленої мети вирішено такі завдання:

- розглянуто існуючі шаблони проектування мікросервісних додатків;
- розділено існуючі шаблони в окремі групи відповідно до задач, які вони вирішують;
- обрано найважливіші групи мікросервісних шаблонів;
- розроблено графічний інструментарій, який застосовує реалізації цих шаблонів.

4. Огляд шаблонів мікросервісних додатків.

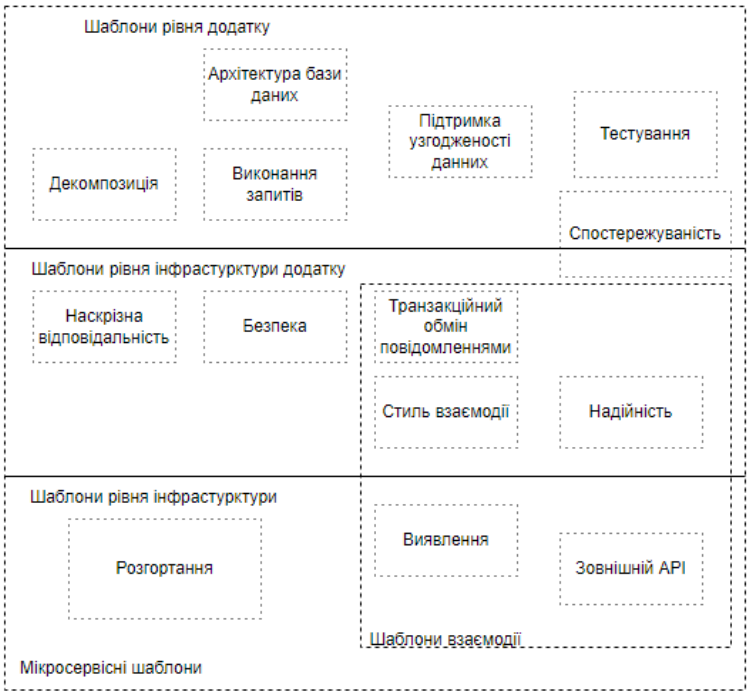

Рис. 1. Групи шаблонів мікросервісних додатків

Для того, щоб проектувати якісні мікросервісні додатки, слід розглянути найбільш необхідні методики проектування мікросервісних додатків, які формалізуються у вигляді шаблонів проектування мікросервісних додатків.

Розпочнемо з категоризації існуючих шаблонів. Всі шаблони мікросервісної архітектури можна розділити на три рівні: шаблони рівня інфраструктури, шаблони рівня інфраструктури додатку та шаблон рівня додатку [3]. До першого рівня відносяться такі групи, як шаблони розгортання, виявлення та зовнішніх API. До шаблонів другого рівня можна віднести такі шаблони: шаблони наскрізної відповідальності, безпеки, транзакційного обміну повідомлень, шаблони стилю взаємодії та надійності. До третьої групи відносяться шаблони декомпозиції, виконання запитів, архітектури баз даних, підтримки узгодженості даних та тестування. Шаблони спостережуваності можуть відноситися як до другого, так і до третього рівнів [4].

Деякі з цих груп шаблонів вирішують критичні проблеми мікросервісної архітектури та повинні застосовуватися в усіх мікросервісних додатках. Інша частина є специфічними та потребують самостійних реалізацій для кожної ситуації в окремості. Створюваний інструментарій буде надавати лише базові шаблони мікросервісної архітектури, зокрема шаблони виявлення, зовнішнього API, надійності, стилів взаємодії, безпеки, спостережуваності і архітектури баз даних [4].

5. Шаблони виявлення.

Шаблони виявлення відносяться до групи шаблонів взаємодії. Мікросервісний додаток може складатися з великої кількості сервісів, тому необхідно створити механізм автоматичного виявлення цих сервісів. Найпростішим варіантом може бути прописування ip-адрес всіх мікросервісів в одному файлі, якщо їх адреси статичні. Для цього найнадійнішим варіантом є використання DNS [5].

DNS дозволяє розробникам зв'язати ім'я з IP-адресою одного або кількох серверів. Наприклад, можна вирішити, що сервіс оплати купівель буде знаходитись за адресою payments.project.com. У спеціальному файлі буде запис, що вказує на IP-адресу хоста, на якому працює цей сервіс, або, можливо, ця адреса буде вказувати на балансувальник навантаження, що розподіляє навантаження між декількома примірниками цього мікросервісу.

Але це не найкращий варіант, бо у випадку використання автоматизованих платформ розгортання контейнерів, адреси можуть змінюватись динамічно, тому потрібен інший підхід. У мікросервісах часто виникає ситуація, коли деякі сервіси вимикаються, а інші, навпаки запускаються, і ці процеси потрібно відстежувати автоматично у режимі реального часу. У цьому випадку може допомогти динамічна реєстрація сервісів. При цьому використовується самостійна реєстрація сервісів із застосуванням центрального реєстру, який займається пошуком сервісів [5].

6. Шаблони зовнішнього API.

Найпоширенішим представником цієї групи шаблонів є шаблон API-шлюзу. В мікросервісній архітектурі кожен мікросервіс реалізує певні бізнес-функції і може знадобитися доступ до них зовні. Звертатися до кожного з сервісів окремо не є гарною практикою, тому слід надати єдину точку входу для клієнтів. Крім цього, цей шаблон надає ще одну перевагу, а саме можливість агрегації результатів, отриманих від декількох сервісів.

По суті цей шаблон є об’єднанням двох класичних шаблонів, а саме шаблону адаптер та шаблону фасад. Так само як адаптер, API-шлюз дозволяє робити запити, навіть якщо інтерфейси несумісні. З точки зору мікросервісів, API-шлюз грає роль оркестратора, з чого випливає його головний недолік, а саме можлива єдина точка відмови [6].

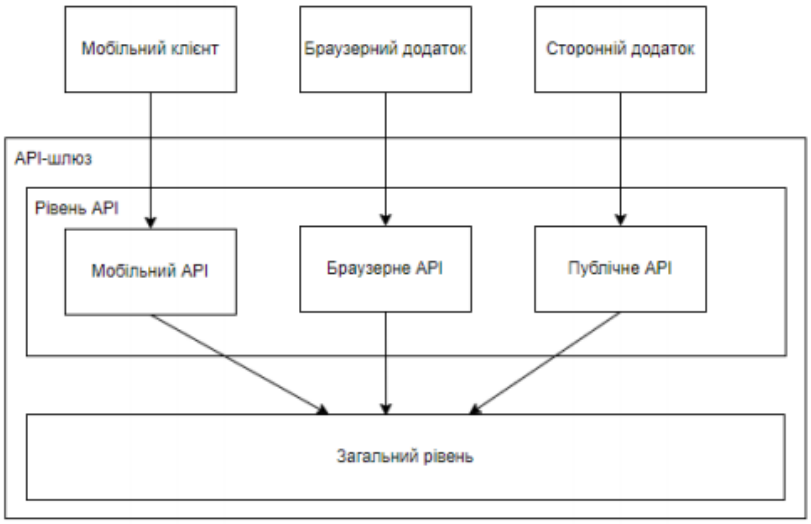

Рис. 2. Архітектура API-шлюзу

7. Шаблони забезпечення безпеки.

Для забезпечення безпеки в мікросервісних додатках, велику увагу слід приділити автентифікації та авторизації, бо зазвичай мікросервіси будуються так, щоб довіряти один одному. Ризик злому одного мікросервісу може поставити під загрозу безпеку даних та роботу інших мікросервісів.

В монолітній архітектурі за автентифікацією та авторизацією відповідає сама програма. Наприклад, Django, веб-середовище мови Python, постачається з вже готовою системою керування користувачами. Що ж до розподілених систем, то тут слід використовувати інші підходи. Не найкращий варіант, щоб усі реєструвалися в різних системах окремо, застосовуючи для кожної системи різні імена користувачів та паролі. Наша мета полягає у створенні єдиного ідентифікатора, що дозволяє проходити автентифікацію лише один раз.

Замість обтяження кожного сервісу вирішенням питань автентифікацією та авторизації з використанням провайдера ідентифікації, можна скористатися шлюзом, працюючим як проксі-сервер і розташованим між мікросервісами і зовнішнім світом. Ідея полягає в тому, що ми можемо централізувати поведінку для перенаправлення користувача та виконання автентифікації та авторизації в одному місці [7].

8. Шаблони забезпечення надійності.

Найбільш популярними шаблонами забезпечення надійності є шаблони «Запобіжник», «Retry Policy» та «Backoff». Шаблони «Retry Policy» та «Backoff» доволі прості. Єдиною відмінністю між ними є те, що шаблон «Backoff» використовує адаптивні інтервали. Довжина інтервалу може змінюватися лінійно або експоненційно.

Шаблон «Запобіжник» націлений на помилки, які можуть виникати в мікросервісах та бути довготривалими. Такими помилками можуть бути проблеми мережі, програмного забезпечення, обладнання. Цей шаблон дозволяє запобігти повторним викликам, які скоріш за все закінчаться помилками, що дозволяє зберегти ресурси, наприклад пам’ять.

У цього шаблону є 3 стани: closed, open, half-open.

1. Стан closed значить, що всі запити надсилаються сервісу. При цьому відбувається підрахунок кількості невдалих запитів проксі-сервісом. Коли ця кількість досягає якогось порогу, цей просі-сервер переводиться в стан open. Через деякий час проксі-сервіс переводиться в стан half-open для чергової спроби надіслати запит.
2. Коли проксі-сервіс знаходиться в стані open, сервісам миттєво повертається помилка.
3. Коли проксі-сервіс знаходиться в стані half-open лише обмежена кількість запитів надсилається. Цей стан існує для того, щоб не було різкого росту запитів до сервісів. Якщо, запити проходять вдало, то проксі сервіс переходить до стану closed, інакше до стану open.

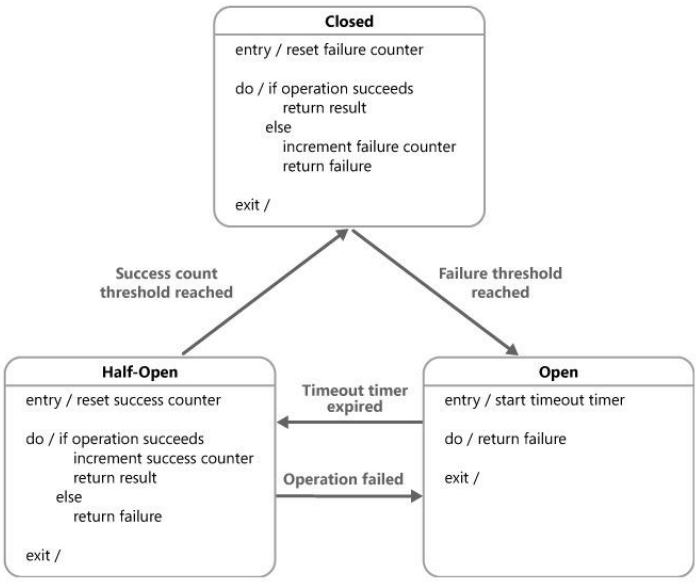

Рис. 3. Алгоритм роботи шаблону «Запобіжник»

9. Шаблони стилів взаємодії.

У випадку стилів взаємодії слід розглянути два найпоширеніші стилі, а саме стиль «запит-відповідь» та асинхронного обміну даними на основі подій.

Представниками першого стилю є технології віддаленого виклику процедури (RPC) та передачі репрезентативного стану (REST).

Віддалений виклик процедури є технологією локального виклику, який виконується на віддаленому сервісі. Технологія RPC має безліч різновидів. Деякі їх засновані на застосуванні певного інтерфейсу (SOAP, Thrift, Protocol Buffers) [8].

Передача репрезентативного стану (REpresentational State Transfer (REST)) є архітектурним стилем, інспірованим Всесвітньою мережею. Найважливішим поняттям у REST є ресурс. У самій REST-технології про застосовувані протоколи не йдеться, хоча найчастіше при її реалізації використовується протокол HTTP. У самому протоколі HTTP визначається ряд дуже корисних можливостей, які дуже добре працюють на реалізацію REST-стилю. Наприклад, дієслова, що фігурують у HTTP-специфікації, такі як GET, POST і PUT, мають цілком зрозумілий сенс, що визначає характер їх роботи з ресурсами.

Тепер розглянемо стиль асинхронного обміну даними на основі подій.

Для задач видачі мікросервісам подій та визначення споживачами моменту наступу події використовуються брокери повідомлень. Брокер обробляє підписки, дозволяючи споживачам отримати інформацію під час генерації тієї чи іншої події. Такі брокери можуть навіть обробляти стан споживачів, наприклад, сприяючи відстеженню того, які повідомлення вони бачили раніше. Прикладом брокеру повідомлень є RabbitMQ [9].

10. Шаблони спостережуваності.

З архітектурної точки зору, мікросервісні додатки є більш складними ніж монолітні, бо вони складаються з великої кількості компонентів. Тому особливу увагу слід приділити механізмам спостережуваності, які дозволять швидко виявляти проблеми в системі.

Найкращим варіантом буде відстеження окремих мікросервісів, а щоб одержати більш загальну картину слід використовувати об'єднання отриманих даних [10].

11. Огляд створеного інструментарію.

Отже, було розглянуто основні шаблони, без яких побудова якісного мікросервісного додатку неможлива. Використовуючи цей аналіз, побудовано інструментарій, який спрощує створення мікросервісних додатків. Графічний інструментарій був побудований за допомогою

мови програмування JavaScript. Згенеровані мікросервісні додатки, використовують JavaScript та платформу NodeJS.

На наступному рисунку наведено вигляд вікна проектування мікросервісного додатку. Воно складається з двох основних компонент: панелі керування та області редагування діаграм.

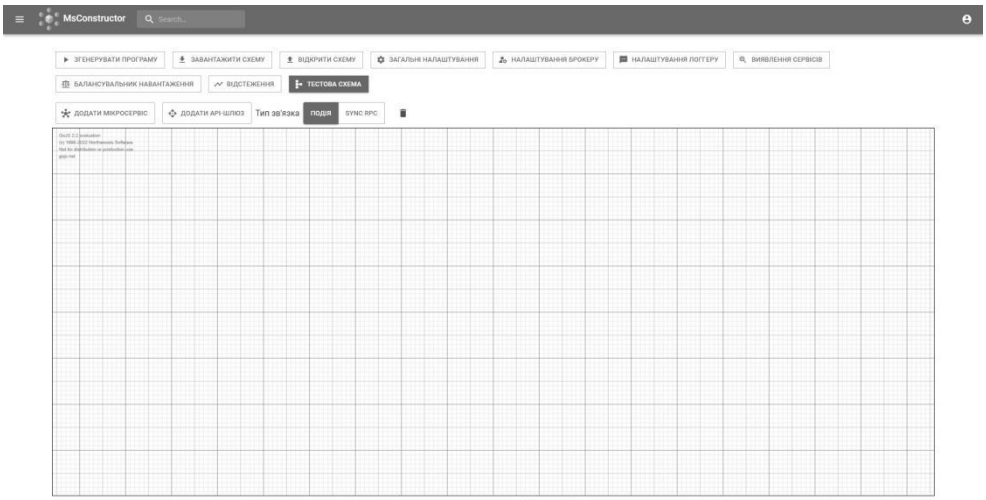

Рис. 4. Вигляд вікна редагування інструментарію

Розглянемо основні можливості інструментарію більш детально.

Розроблений інструментарій підтримує всі стилі взаємодії: RPC, REST та повідомлення. Так, для підтримки повідомлень, розробнику мікросервісного додатку необхідно налаштувати брокери повідомлень, які підтримують шаблон «видавець/підписник».

Перелік наступних доступних брокер повідомлень: TCP, NATS, Redis, MQTT, AMQP (0.9), AMQP (1.0), Kafka, NATS Streaming (STAN).

Перелік підтримуваних серіалізаторів повідомлень: JSON, Avro, MsgPack, Notepack, ProtoBuf, Thrift, CBOR.

Використовуючи налаштування брокеру сервісів, можна крім цього налаштувати шаблони надійності, а саме шаблон «Запобіжник», «Retry Policy» та «Bulkhead».

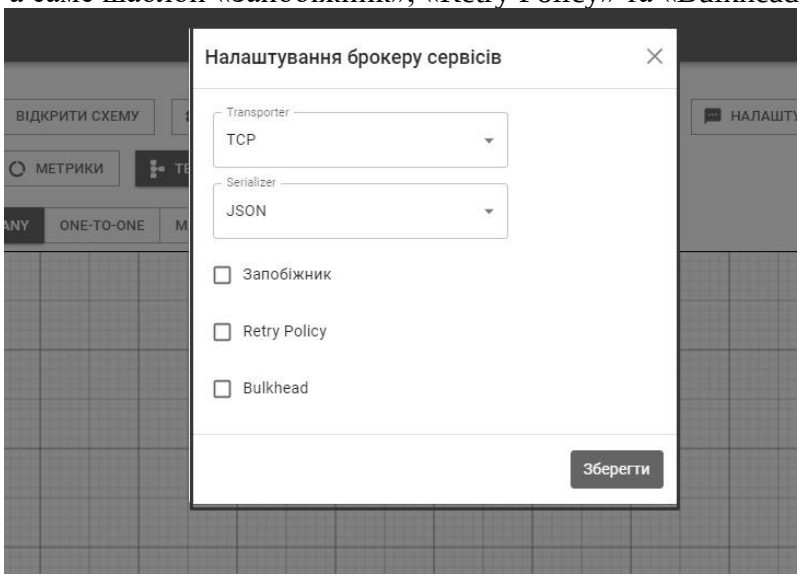

Рис. 5. Налаштування брокеру сервісів інструментарію

На наступному зображені наведені доступні в інструментарії механізми спостережуваності за роботою мікросервісних додатків.

Використовуючи це вікно, користувач може обрати зручний спосіб журналювання інформації про стан мікросервісного додатку. Існує широкий перелік підтримуваних

сторонніх логерів, таких як Winston, Bunyan та інших. Додатково є можливість налаштувати форматування даних та рівень журналювання.

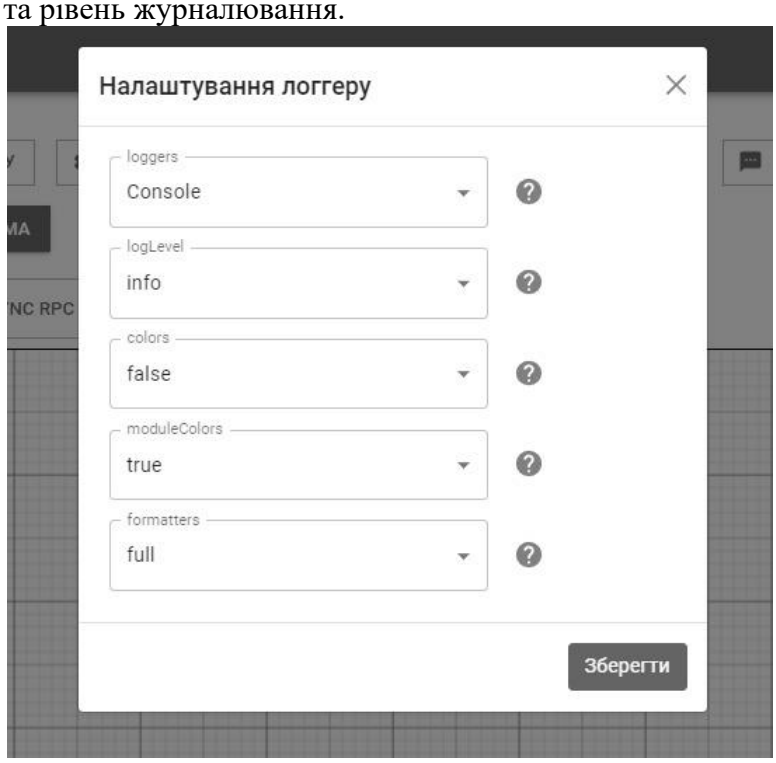

Рис. 6. Налаштування логеру в інструментарії

Для виявлення мікросервісів інструментарій надає декілька варіантів, а саме Local, Redis та Etcd3.

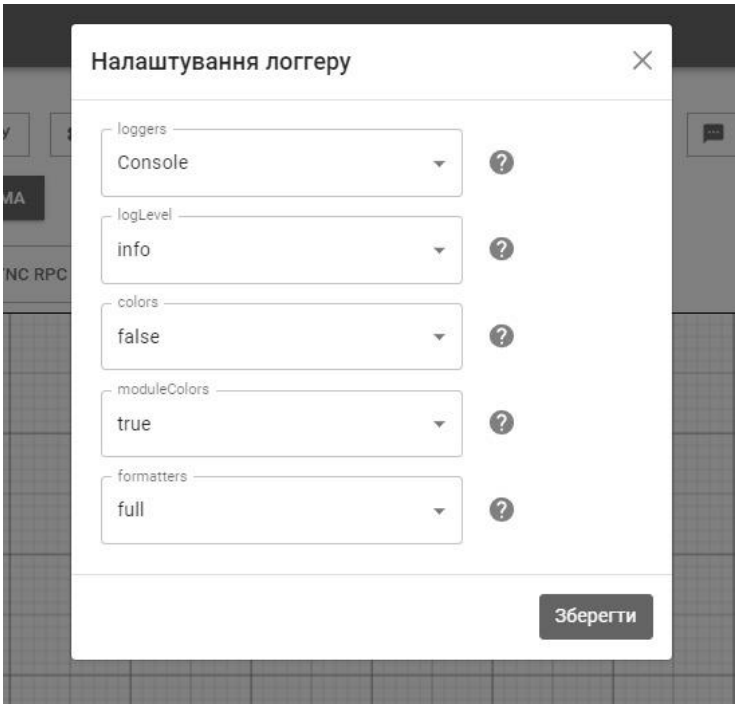

Рис. 7. Налаштування механізму виявлення сервісів

Локальне виявлення полягає в тому, що не потребує додаткових засобів та інформація про нові сервіси передається через транспортер. У випадках Redis та Etcd3 потрібно додатково налаштовувати сервери.

Так як мікросервісна архітектура використовує горизонтальне масштабування, то слід впровадити механізми балансування.

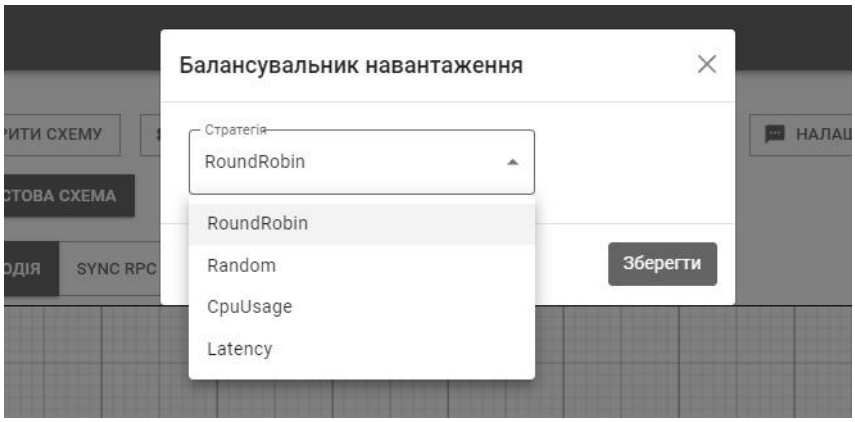

Рис. 8. Налаштування балансувальника навантаження

Інструментарій підтримує 4 найрозповсюдженіших алгоритми, а саме RoundRobin, Random, CpuUsage та Latency.

На наступному зображені наведено процес налаштування API-шлюзу, який буде виконувати роль RESTful API.

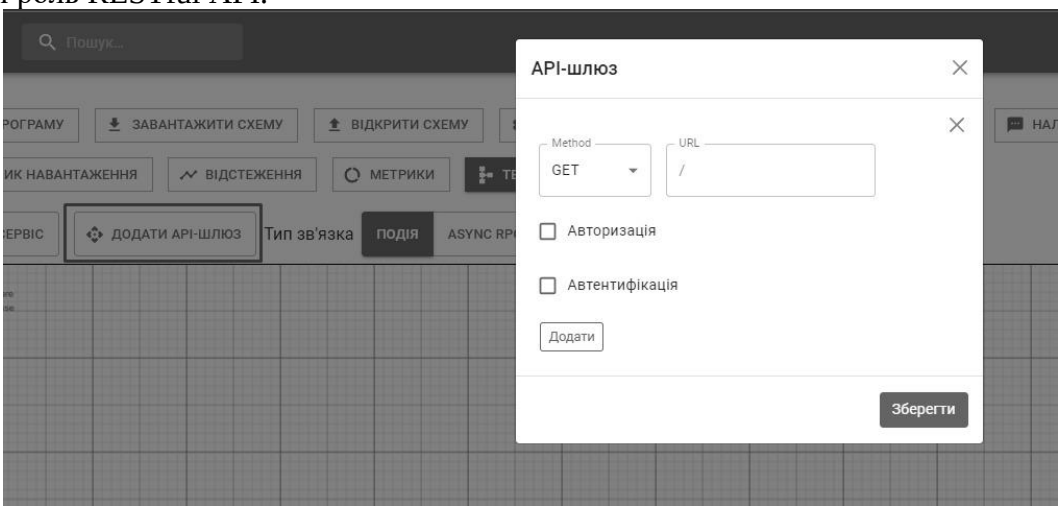

Рис. 9. Налаштування API-шлюзу

Шаблон API-шлюз в інструментарії був реалізований з врахуванням безпеки. Таким чином розробники можуть додати на цьому проксі-сервісі можливість авторизації та автентифікації користувачів.

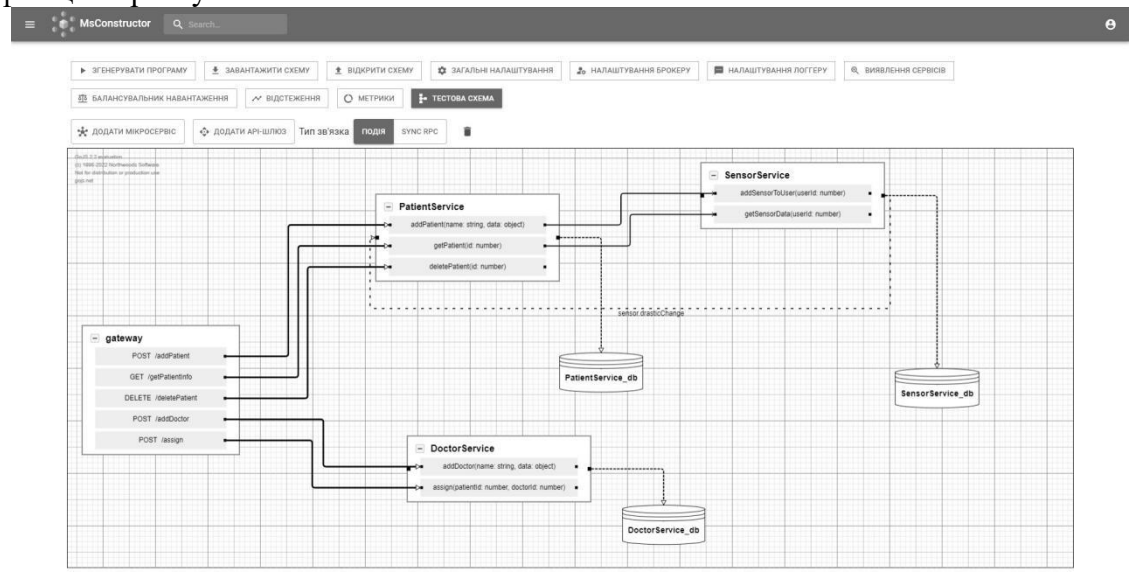

Рис. 10. Вигляд схеми розробленого додатку

12. Тестування розробленого інструментарію.

Використовуючи весь функціонал створеного інструментарію, було створено структуру мікросервісного програмного забезпечення для контролю за пацієнтами. На наступному рисунку наводиться вигляд діаграми мікросервісів та зв'язки між ними. Суцільними лініями показано віддалені виклики процедур, пунктирними лініями – повідомлення.

Згенерований мікросервісний додаток має наступну структуру.

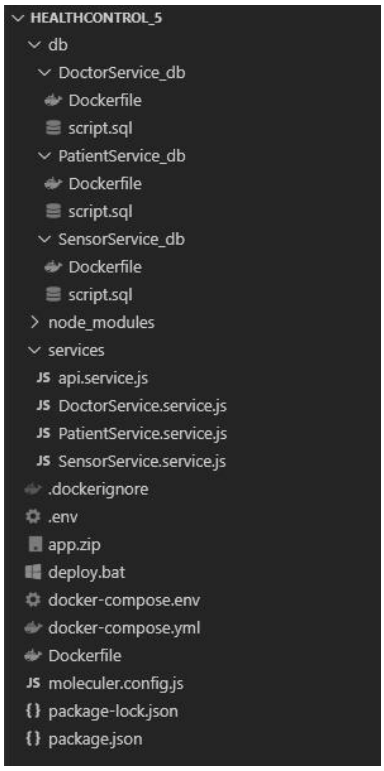

Рис. 11. Згенерований додаток

Після цього розробнику залишається лише запустити проект у контейнерах за допомогою Docker Compose. Для цього необхідно використати команду *npm start*.

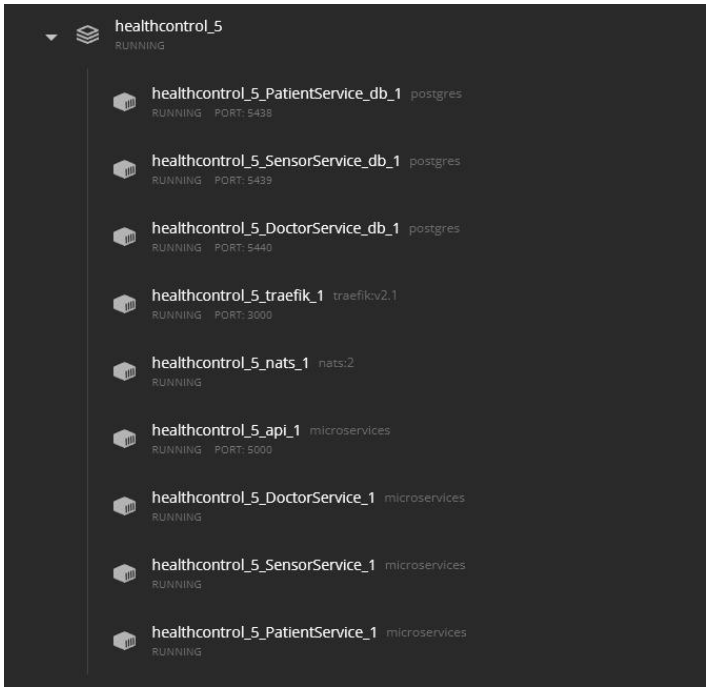

Рис. 12. Запущений додаток

Як бачимо, було створено 9 контейнерів. У трьох контейнерах запущено мікросервіси, ще в трьох – бази даних для них. В одному з контейнерів запускається один сервіс API-шлюзу. Крім цього додатково в контейнерах розташовані два допоміжних сервісів: NATS та Traefik.

13. Висновки.

Створений інструментарій підтвердив свою працездатність на прикладі реалізації системи для контролю за пацієнтами, яка була виконана у вигляді мікросервісного додатку. Тестування показало, що інструментарій може значно пришвидшити створення мікросервісних додатків шляхом автоматизації деяких важливих компонентів мікросервісної архітектури. Так як інструментарій виконаний за модульною схемою, тобто клієнтська частина генерує команди серверній частині у вигляді універсальних інструкцій, інтегрувати інші мови програмування не є великою проблемою (наприклад Java та C++) та це може буде реалізовано у найближчій перспективі.

Список використаної літератури

1. Taibi D., Lenarduzzi V., Pahl K. Architectural Patterns for Microservices: a Systematic Mapping Study, 8th International Conference on Cloud Computing and Services Science. 2018.
2. Jamshidi P., Lewis J., Tilkov S. Microservices: The Journey So Far and Challenges Ahead. IEEE SUSTAINABLE COMPUTING. 2020.
3. Waseem M., Liang P., Shahin M. Design, monitoring, and testing of microservices systems: The practitioners' perspective. Journal of Systems and Software. Vol. 182. 2021.
4. Ponce F., Márquez G. Migrating from monolithic architecture to microservices: A Rapid Review. 2019 38th International Conference of the Chilean Computer Science Society (SCCC). 2019.
5. Tyszberowicz S. Identifying Microservices Using Functional Decomposition. LNPSE, Vol. 10998. 2018.
6. Hannousse A., Yahiouche S. Securing microservices and microservice architectures: A systematic mapping study. Computer Science Review, Vol. 41. 2021.
7. Niu Y. Load Balancing Across Microservices. IEEE INFOCOM 2018 - IEEE Conference on Computer Communications. 2018.
8. Zhongshan R., Wei W. Migrating Web Applications from Monolithic Structure to Microservices Architecture. Internetware '18: Proceedings of the Tenth Asia-Pacific Symposium on Internetware. 2018. P. 1–10.
9. Bucchiarone A., Dragoni N. From Monolithic to Microservices: An Experience Report from the Banking Domain. IEEE Software. Vol. 35. 2018. P. 50–55.
10. Rossi F. Hierarchical Scaling of Microservices in Kubernetes. 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS). 2020.