https://doi.org/10.31891/2219-9365-2024-80-16
УДК 004.41, 004.75

ВОЛОКИТА Артем
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
https://orcid.org/0000-0001-9069-5544
e-mail: artem.volokita@kpi.ua
ЗОТКІН Кирило
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
https://orcid.org/0009-0001-2320-9021
e-mail: zotkin.kv@gmail.com

ПРОБЛЕМАТИКА ПРОЄКТУВАННЯ ПРОГРАМНИХ СИСТЕМ НА ОСНОВІ ПОДІЙНО-ОРІЄНТОВАНИХ АРХІТЕКТУР (EDA)

У сучасному швидкоплинному світі інформаційні системи повинні обробляти величезні обсяги даних з мінімальною затримкою, що призводить до ускладнення архітектури розподілених систем. Подійно-орієнтована архітектура (ПОА) стала найбільш гнучким та масштабованим рішенням для задоволення зростаючих функціональних вимог, що робить її пріоритетним вибором для проектів з високим навантаженням. Проте впровадження ПОА вимагає ретельного планування на етапі проектування системи, оскільки помилки можуть бути дорогими та призвести до повного збою системи під час великих навантажень.

Ця стаття надає всебічний огляд критичних проблем при проектуванні подійно-орієнтованих архітектур та досліджує ефективні стратегії їх вирішення, забезпечуючи надійну та стабільну роботу системи.

Ключові слова: сервіс-орієнтована архітектура, подійно-орієнтована архітектура, розподілені системи, проблеми побудови архітектури.

VOLOKYTA Artem, ZOTKIN Kyrylo
National Technical University of Ukraine “Ihor Sikorskyi Kyiv Polytechnic Institute"

PROBLEMS OF DESIGNING SOFTWARE SYSTEMS BASED ON EVENT-DRIVEN ARCHITECTURES (EDA)

In today's fast-paced world, information systems must process huge amounts of data with minimal latency, which leads to the complexity of the architecture of distributed systems. Event-driven architecture (EDA) has become the most flexible and scalable solution to meet growing functional requirements, making it a priority choice for projects with high workloads. However, implementing EDA requires careful planning at the system design stage, as errors can be costly and lead to complete system failure during heavy workloads.

Building systems using event-driven architecture requires taking into account numerous limitations and problems that may arise during operation. At the design stage, it is necessary to foresee all possible challenges to ensure the reliability and efficiency of the system. Due to the large number of approaches to building architecture and solving operational problems, there is a need for a comprehensive review of these issues and finding optimal solutions.

The article considered the challenges encountered in designing distributed systems based on event-driven architecture, as well as their solutions. The analysis of the solutions showed that it is impossible to single out a single strategy for building highly loaded and secure systems; in each case, it is necessary to consider the architecture comprehensively and make decisions based on the existing functional and non-functional requirements.

This article provides a comprehensive overview of critical issues in the design of event-driven architectures and explores effective strategies for solving them, ensuring reliable and stable system operation.

Keywords: service-oriented architecture, event-driven architecture, distributed systems, architecture design problems.

ПОСТАНОВКА ПРОБЛЕМИ У ЗАГАЛЬНОМУ ВИГЛЯДІ ТА ЇЇ ЗВ’ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

Однією з основних вимог до сучасних програмних систем є можливість швидкого розширення і масштабування. Ці вимоги зумовили трансформування архітектурних підходів від монолітних систем і Layered Architecture до SOA і як одного з підвидів EDA [1].

Подійно-орієнтована архітектура (Event-Driven Architecture, EDA) [2] - це архітектурний підхід до розробки програмного забезпечення, в якому компоненти системи взаємодіють через обмін подіями. У цій архітектурі компоненти системи, як правило, незалежні один від одного і реагують на події, що відбуваються в системі або надходять ззовні.

Побудова систем із застосуванням подійно-орієнтованої архітектури вимагає врахування численних обмежень та проблем, які можуть виникнути під час експлуатації. На етапі проектування необхідно передбачити всі можливі виклики, щоб забезпечити надійність та ефективність роботи системи. Через велику кількість підходів до побудови архітектури та вирішення експлуатаційних проблем виникає потреба у комплексному огляді цих питань та пошуку оптимальних рішень.

АНАЛІЗ ДОСЛІДЖЕНЬ ТА ПУБЛІКАЦІЙ

Статті [2-4] дають огляд основних принципів EDA. Центральним елементом архітектури є події, які відображають важливі зміни або стан системи. Події можуть виникати внаслідок взаємодії користувача з системою, змін у стані системи або зовнішніх подій, таких як повідомлення від інших систем. Взаємодія компонентів в подійно-орієнтованій архітектурі відбувається асинхронно, що дозволяє системі бути більш гнучкою і масштабованою, а сервісам більш атомарними - кожен компонент системи має власну логіку та може функціонувати незалежно від інших компонентів.

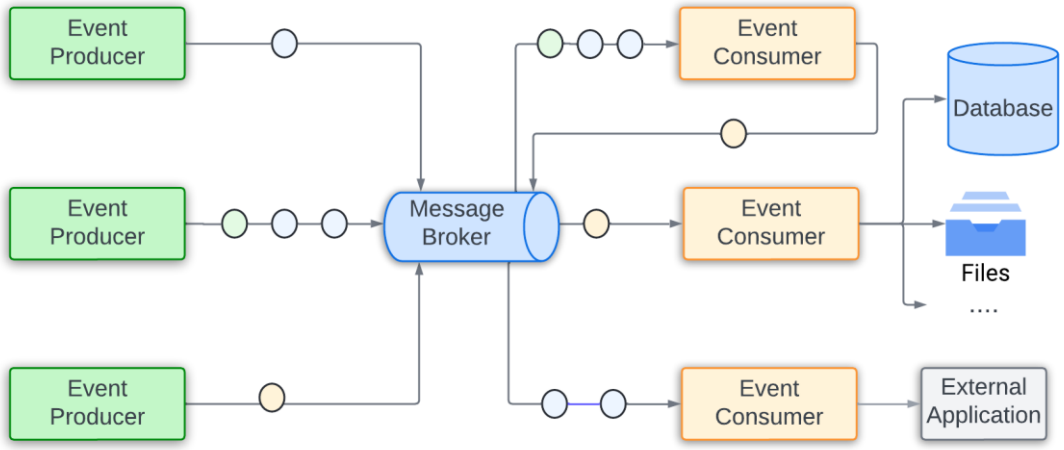

Рис.1. Концептуальна модель EDA архітектури

Джерело: розроблено авторами

Враховуючи наведені принципи, можна зазначити що основною перевагою EDA є гнучкість, розширюваність і масштабованість [2,3,4]. За рахунок низької зв'язаності компонентів, системи побудовані на такій архітектурі дозволяють швидко адаптувати систему під зміни у вимогах і дозволяють будувати складні бізнес-процеси. Асинхронна комунікація і наявність шини даних дозволяють легко інтегрувати зовнішні системи і нові процеси простим додавання нового сервісу обробки без зміни існуючих компонентів.

EDA вирішує низку архітектурних питань і спрощує подальше масштабування і розширення бізнес логіки, проте вона також вносить власні виклики при побудові програмних комплексів, які варто враховувати на етапі проектування і підтримки системи. Основні найпоширеніші проблеми які виникають в процесі експлуатації таких систем наведені нижче:

1. **Порядок подій та консистентність** [5] - Підтримка правильного порядку подій та забезпечення консистентності даних може бути складною задачею в системах EDA, особливо при обробці великих обсягів подій.
2. **Складні потоки подій** - Управління складними потоками подій та бізнес-процесами може стати викликом через потребу в оркестрації, вирішенні конфліктів та забезпеченні коректності потоків даних.
3. **Стійкість до помилок та відновлення** - Забезпечення стійкості до помилок та відновлення роботи системи після збоїв може бути складною задачею через асинхронну природу обробки подій.
4. **Безпека та авторизація** - Забезпечення безпеки та авторизації в системах EDA може бути викликом через використання єдиної шини передачі даних і розподіленості системи.

ФОРМУЛЮВАННЯ ЦІЛЕЙ СТАТТІ

В результаті аналізу доступний стратегій побудови систем із застосуванням EDA виявлено що немає єдиного підходу до побудови таких систем і рішення має залежати від вимог конкретної системи. Є необхідність в дослідженні можливих наявних паттернів в вимогах до систем, які можуть диктувати загальну стратегію побудови архітектури.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

Метою роботи є огляд потенційних проблем і обмежень при побудові систем із застосування подійно-орієнтованої архітектури а також стратегії їх вирішення

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

Розглянемо детально основні **проблеми** при проєктуванні програмного забезпечення на основі подійно-орієнтованої архітектури.

Порядок подій та консистентність. У EDA, де події відправляються асинхронно, порядок, в якому події обробляються, може бути важливим. Наприклад, у системі обробки замовлень, спочатку потрібно обробити подію "створення замовлення", а потім "оплата замовлення". Якщо порядок подій порушується, може виникнути неконсистентність у стані системи. Вірогідність такої проблеми може збільшуватись при використанні горизонтального масштабування сервісів-обновників [7], коли один потік повідомлень можуть обробляти декілька сервісів.

Розглянемо приклад наведений на рисунку 2. Припустимо ми маємо генератор подій, який генерує події у правильній послідовності і також маємо групу з двох підписників на ці події. Шина даних в такому випадку буде використовувати механізм round-robin [8] і відправляти наступне повідомлення сервісу, який перший закінчив обробку.

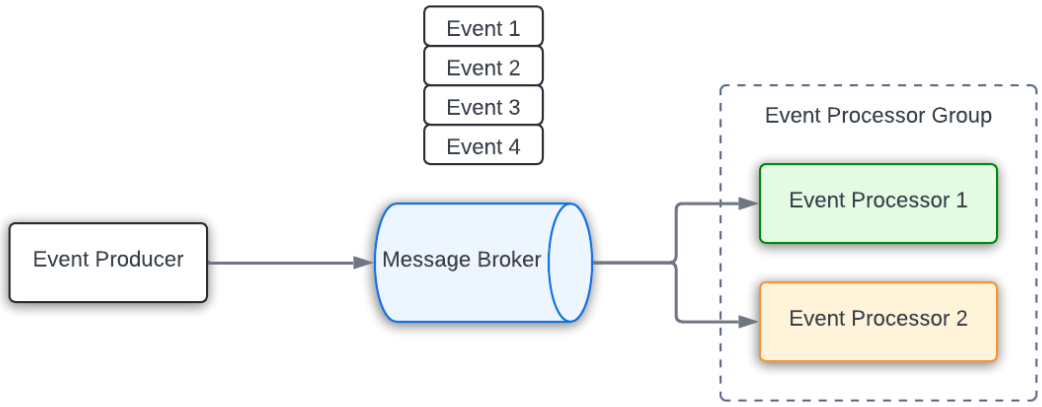

Рис. 2. Ілюстрація послідовності подій

Джерело: розроблено авторами

В такому випадку подія 1 буде оброблятись сервісом 1, подія 2 відповідно сервісом 2 і, якщо сервіс 1 перевантажений чи час обробки події 1 більше ніж події 2, то послідовність обробки буде порушена.

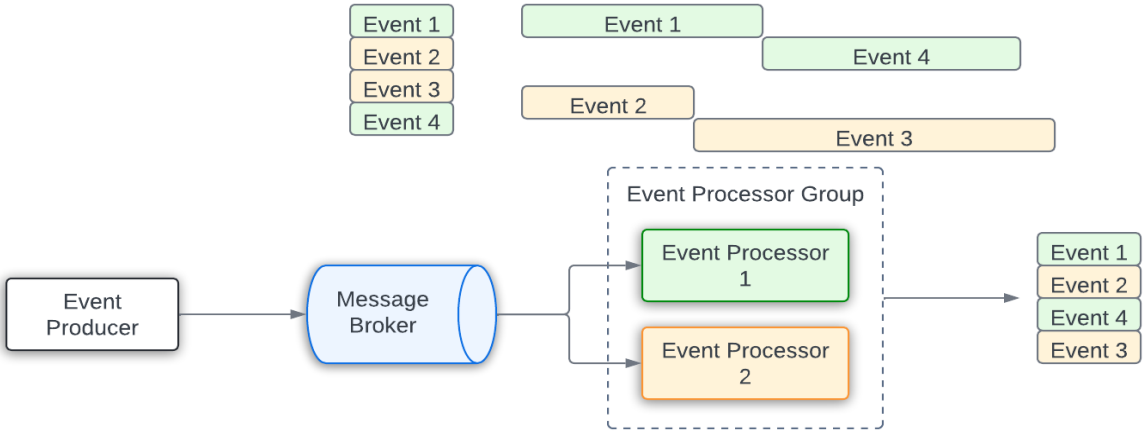

Рис. 3. Приклад черговості обробки

Джерело: розроблено авторами

В таблиці 1 наведені варіанти вирішення проблеми послідовності з урахуванням складності впровадження того чи іншого рішення

Таблиця 1

Варіанти вирішення проблеми послідовності подій в EDA

Рішення	Складність впровадження	Опис
Використання послідовних черг (Ordered Queues) [9, 10]	Низька	Використання черг повідомлень, що підтримують послідовність, таких як Kafka або RabbitMQ з підтримкою ordered queues. Це гарантує, що повідомлення будуть оброблятися у тому порядку, в якому вони надійшли.
Відмітки часу або номери послідовності (Timestamps and Sequence Numbers)	Низька	Додавання відміток часу до подій для забезпечення обробки у правильному порядку. Обробники подій можуть використовувати ці відмітки для сортування подій перед обробкою.
Впровадження патерну SAGA та компенсаційних транзакцій [11, 12]	Висока	Використання патерну SAGA для управління розподіленими транзакціями та збереження консистентності даних, навіть якщо події обробляються асинхронно і можуть приходити у випадковому порядку.
Управління потоками подій (Event Streams) [13]	Висока	Розділення подій на окремі потоки на основі певних критеріїв (наприклад, по користувачам або транзакціям) для забезпечення послідовності в межах кожного потоку або навіть сутності.
Застосування принципу idempotency [14]	Середня	Застосування ідемпотентних операцій для обробки подій, що дозволяє повторно обробляти події без негативних наслідків для системи.

Джерело: розроблено авторами.

Складні потоки подій. У складних бізнес-процесах події можуть генеруватися та оброблятися різними сервісами або компонентами, які працюють незалежно один від одного. Координація цих компонентів для забезпечення коректного виконання бізнес-логіки є серйозним викликом, оскільки такі процеси можуть включати десятки, а інколи й сотні послідовних подій. Наприклад, на рисунку 4 показано BPMN-діаграму [15] процесу оформлення замовлення в інтернет-магазині. Як видно з діаграми, такий бізнес-процес може містити не лише послідовність подій, а й різні умови, підпроцеси та паралельну обробку.

Збереження стану сутності є критично важливим для забезпечення безперервності та коректності складних бізнес-процесів. Це дозволяє системі відстежувати поточний стан кожної сутності і забезпечувати правильну обробку подій навіть у випадку збоїв або відновлення після помилок.

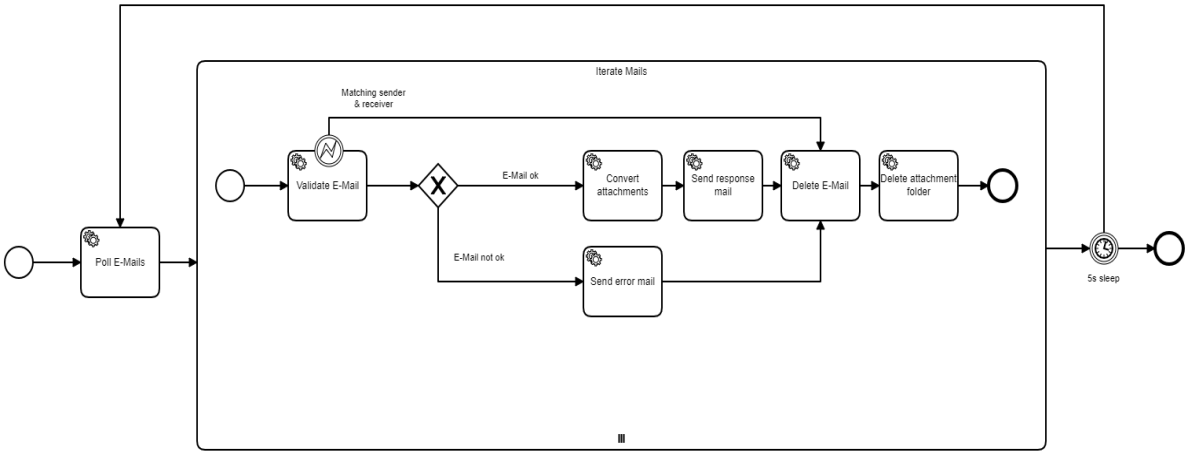

Рис. 4. Приклад BPMN діаграми процесу оформлення замовлення
[https://forum.camunda.io/t/creating-a-loop-modelling-a-service/13930]

В таблиці 2 наведені варіанти побудови систем зі складними бізнес-процесами.

Таблиця 2

Варіанти побудови систем зі складними бізнес-процесами

Рішення	Складність реалізації	Опис
Управління подіями через шаблон проектування «хореографія» [16]	Середня	Хореографія – це патерн взаємодії між сервісами в Event-Driven архітектурі, де кожен сервіс незалежно приймає рішення на основі отриманих подій, без центрального контролера.
Управління подіями через шаблон проектування «оркестрація» [17]	Висока	Оркестрація – це патерн управління взаємодією між сервісами в Event-Driven архітектурі, де один центральний компонент (оркестратор) координує та контролює виконання всіх кроків бізнес-процесу.
Використання BPMN (Business Process Model and Notation) для моделювання та візуалізації складних бізнес-процесів	Середня	BPMN моделі можуть бути використані безпосередньо оркестратором, наприклад Camunda Process Manager [18]

Джерело: розроблено авторами.

Стійкість до помилок та відновлення. З огляду на те, що комунікація в системах з EDA архітектурою відбувається асинхронно, такі системи висувають додаткові вимоги до відмовостійкості.

Ключовою проблемою побудови асинхроних систем є канал передачі даних між сервісами і шиною даних. Будь які затримки в мережі передачі даних або збої в роботі фізичного рівня можуть призвести до втрати підключення до шини даних [6] або втрати пакету підтвердження оброки повідомлення. Ці особливості вимагають включення в архітектуру системи додаткових перевірок на активність підключення, механізму повторної відправки у разі помилки чи втрати пакету а також можливість горизонтального масштабування сервісів [7]. Велике навантаження на сервіс або шину даних також може призвести до втрати пакетів підтвердження обробки, збою в роботі мережі або неправильній послідовності оброки повідомлень.

Другим викликом в правильній обробці помилок може стати не співпадіння контрактів передачі даних між сервісами, що призводить до внутрішніх помилок в сервісі обробки або втраті пакету даних. Для спрощення подальшої підтримки і розробки системи а також мінімізації ризику втрати даних на етапі проектування системи повинен вводитись механізм версіонування контракта і закладатися в сервісі можливість оброки декількох версій цього контракта.

Архітектурний патерн “джерело подій” [23] також може бути використаним для зменшення залежності системи від помилок і спрощення відновлення системи при збої в шині даних. При цьому підході сутність в системі представлена послідовністю подій які трансформуються в модель даних по запиту користувача. В цьому випадку сервіс працює з усім масивом повідомлень що зменшує вірогідність втрати події через проблеми в мережі чи не співпадіння контрактів повідомлень.

У таблиці 3 наведено варіанти вирішення проблем стійкості та відновлення в EDA.

Таблиця 3

Рішення проблеми стійкості і відновлення

Рішення	Складність впровадження	Опис
Використання механізму Dead-letter Queues в шині даних [20]	Низька	Більшість шин даних підтримують механізм збереження повідомлень які були оброблені з помилкою. Такі черги мають регулярно аналізуватись людиною яка забезпечує підтримку системи
Тестування контрактів, кероване споживачем [21, 22]	Середня	Реалізація механізму тестування контракту повідомлення до його обробки
Використання архітектурного патерну «джерело подій» (event sourcing) [23, 24]	Висока	Архітектурний підхід який дозволяє зберігати стан сутності як послідовність подій і відтворювати її з цієї послідовності за потреби.
Використання версіонування контрактів і підтримка зворотної сумісності контрактів [25]	Середня	На етапі проектування системи має бути чітко визначене поняття «версії» контракту і всі сервіси обробки мають підтримувати зворотню сумісність з більш старими версіями.
Горизонтальне масштабування сервісів обробки [7]	Низька	Всі шини даних дозволяють горизонтальне масштабування в рамках одно

Джерело: розроблено авторами.

Безпека та авторизація. Системи побудовані з використанням подійно-орієнтованої архітектури (EDA) покладаються на асинхронну передачу подій між компонентами системи, що дозволяє досягти високої гнучкості, масштабованості та реактивності. Проте, такий підхід також створює унікальні виклики у сфері безпеки, які потребують ретельного опрацювання на етапі проектування системи і моніторингу під час використання [26, 27].

Першочерговим викликом побудови безпеки при використанні шини даних є надійність каналу передачі даних. Події передаються між компонентами через мережу, що робить їх вразливими до перехоплення і маніпуляцій. У випадку, якщо зловмисники отримають доступ до каналу комунікації, вони можуть викрасти або змінити події, що призведе до компрометації системи. Наприклад, подія, яка містить конфіденційні дані клієнтів, може бути викрадена, або подія, що ініціює фінансову транзакцію, може бути змінена.

Другою проблемою побудови захищеної системи підтвердження прав доступу компонентів, є їх аутентифікація і авторизація [28]. Це стає ще більш складним завданням в умовах багато клієнтських (multi-tenant) систем, коли декілька клієнтів (тенантів) використовують спільну інфраструктуру.

У Таблиці 4 наведені варіанти підвищення безпеки систем побудованих на EDA.

Таблиця 4

Рішення для підвищення безпеки системи

Рішення	Складність	Опис
Використання приватної мережі для шини даних і сервісів [29, 30]	Низька	По можливості доступ до шини даних повинен бути обмежений на рівні під-мережі. У випадку необхідності до шини даних за межами приватної мережі, треба використовувати VPN-тунель
Використання шифрування при передачі повідомлень (encryption at transfer)	Низька	Для запобігання "атаки посередника" (MITM) [31] треба використовувати шифрування при передачі повідомлень. Найбільш розповсюдженими протоколами шифрування є TLS [32] та HTTPS [33]
Використання шифрування при збереженні повідомлень (encryption at rest) [34]	Висока	У випадку коли події містять конфіденційну інформацію (наприклад номера рахунків, паролі чи особисту інформацію) важливо мати механізм шифрування даних, які зберігаються в шині даних. Ці дані можуть розшифровуватись як самою шиною даних так і безпосередньо сервісами обробки за допомогою асиметричного ключа
Використання механізму цифрового підпису подій (message signature) [35]	Середня	Якщо виникає необхідність впевнитись в оригінальності повідомлення і відправника, можливо використовувати механізм підпису повідомлень, коли підпис повідомлення розраховується як хеш всіх даних в повідомленні. Найбільш розповсюджений хеш-алгоритм наразі є SHA-512/256 [36]
Авторизація і аутентифікація сервісів в шині даних [37, 38, 39]	Середня	Розмежування прав доступу для сервісів є ключовим механізмом захисту системи. Доступ має регулювати не тільки аутентифікації доступу до шини даних але і перевірку авторизації сервісу, коли перевіряється наявність прав писати або читати з конкретної черги
Безпековий моніторинг [40, 41]	Середня	На етапі проектування системи необхідно закладати підходи і методи до моніторингу безпеки на різних рівнях: аудит доступу до мережі, аудит доступу до шини даних, журнал виконаних операцій і аудит журналу подій на рівні системи

Джерело: розроблено авторами.

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

В статті були розглянуті виклики які зустрічаються при проектуванні розподілених систем на базі подійно-орієнтованої архітектури, а також варіанти їх вирішення. Аналіз варіантів вирішення показав, що неможливо виділити єдину стратегію побудови високо-навантажених і безпечних систем, в кожному випадку треба розглядати архітектуру комплексно і приймати рішення грунтуючись на наявних функціональних і нефункціональних вимогах.

References

1. RICHARDS, Mark. Software architecture patterns. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Incorporated, 2015.
2. Richards M. Software Architecture Patterns [Електронний ресурс] / Mark Richards // O'Reilly Media, Inc.. – 2015. – Режим доступу до ресурсу: https://www.oreilly.com/library/view/software-architecturepatterns/9781491971437/ch02.html.
3. PETRENKO, O. O. Порівняння типів архітектури систем сервісів. System research and information technologies, 2015, 4: 48-62.
4. Event-driven architectures [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/eventarc/docs/event-driven-architectures.
5. Microservices and the Problem of Distributed Data Management [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://www.f5.com/company/blog/nginx/event-driven-data-management-microservices
6. Potential pitfalls in the development of Event-driven architecture - [Електронний ресурс] – 2024. - https://medium.com/@sangam.n.pandey/potential-pitfalls-in-the-development-of-event-driven-architecture-d8ffe224d02c
7. Scaling your app with RabbitMQ [Електронний ресурс] – Режим доступу до ресурсу – 2024. - https://medium.com/poatek/scaling-your-app-with-rabbitmq-eb9cb6c8d9d6
8. RADI, Mohammed. Weighted round robin policy for service brokers in a cloud environment. In: The International Arab Conference on Information Technology (ACIT2014), Nizwa, Oman. 2014. p. 45-49.
9. Message ordering guarantees [Електронний ресурс] – Режим доступу до ресурсу – 2024. - https://www.rabbitmq.com/docs/semantics#ordering
10. How to use Apache Kafka to guarantee message ordering [Електронний ресурс] – Режим доступу до ресурсу – 2024. https://medium.com/latentview-data-services/how-to-use-apache-kafka-to-guarantee-message-ordering-ac2d00da6c22
11. DÜRR, Karolin; LICHTENTHÄLER, Robin; WIRTZ, Guido. An Evaluation of Saga Pattern Implementation Technologies. In: ZEUS. 2021. p. 74-82.
12. HAAPAKOSKI, Johannes. IMPLEMENTING ASYNCHRONOUS SAGAS IN A MICROSERVICE ARCHITECTURE. 2023.
13. SOFFER, Pnina, et al. From event streams to process models and back: Challenges and opportunities. Information Systems, 2019, 81: 181-200.
14. RAMALINGAM, Ganesan; VASWANI, Kapil. Fault tolerance via idempotence. In: Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. 2013. p. 249-262.
15. WHITE, Stephen A. Introduction to BPMN. Ibm Cooperation, 2004, 2.0: 0.
16. 16. Choreography pattern [Електронний ресурс] – 2024 - Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/architecture/patterns/choreography
17. MEGARGEL, Alan; POSKITT, Christopher M.; SHANKARARAMAN, Venky. Microservices orchestration vs. choreography: A decision framework. In: 2021 IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC). IEEE, 2021. p. 134-141.
18. Camunda Process Manager [Електронний ресурс] – 2024 - Режим доступу до ресурсу: https://camunda.com/process-orchestration/

19. GRUENER, Sten; KOZIOLEK, Heiko; RÜCKERT, Julius. Towards resilient IoT messaging: an experience report analyzing MQTT brokers. In: 2021 IEEE 18th International Conference on Software Architecture (ICSA). IEEE, 2021. p. 69-79.
20. REAGAN, Rob; REAGAN, Rob. Message Queues. Web Applications on Azure: Developing for Global Scale, 2018, 343-380.
21. NYMAN, Robin. Consumer-Driven Contract Testing: A Framework and Pilot Implementation. 2022.
22. LEHVÄ, Jyri; MÄKITALO, Niko; MIKKONEN, Tommi. Consumer-driven contract tests for microservices: A case study. In: *Product-Focused Software Process Improvement: 20th International Conference, PROFES 2019, Barcelona, Spain, November 27–29, 2019, Proceedings 20*. Springer International Publishing, 2019. p. 497-512.
23. PACHECO, Vinicius Feitosa. *Microservice Patterns and Best Practices: Explore patterns like CQRS and event sourcing to create scalable, maintainable, and testable microservices*. Packt Publishing Ltd, 2018. p. 109-204
24. OVEREEM, Michiel; SPOOR, Marten; JANSEN, Slinger. The dark side of event sourcing: Managing data conversion. In: 2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER). IEEE, 2017. p. 193-204.
25. Versioning of messages in message brokers pattern [Електронний ресурс] – 2024 - Режим доступу до ресурсу: https://medium.com/@michaelgorishnyi/versioning-of-messages-in-message-brokers-2a1225449562
26. 7 Ways to Protect Your Data in Event-Driven Architectures [Електронний ресурс] – 2024 - Режим доступу до ресурсу: https://medium.com/@bhardwaj-deepak/7-ways-to-protect-your-data-in-event-driven-architectures-8953c0a2eb97
27. Securing Your Event-Driven Architecture: Best Practices and Considerations [Електронний ресурс] – 2024 - Режим доступу до ресурсу: https://www.linkedin.com/pulse/securing-your-event-driven-architecture-best-practices-considerations
28. HEILAND, Randy, et al. Authentication and authorization considerations for a multi-tenant service. In: *Proceedings of the 1st Workshop on The Science of Cyberinfrastructure: Research, Experience, Applications and Models*. 2015. p. 29-35.
29. Коваленко А.А. Метод забезпечення живучості комп'ютерної мережі на основі VPN тунелювання / А.А. Коваленко, Г.А. Кучук, В.М. Ткачов // Системи управління, навігації та зв'язку. Збірник наукових праць. – Полтава: ПНТУ, 2021. – Т. 1 (63). – С. 90-95. – doi:https://doi.org/10.26906/SUNZ.2021.1.090.
30. MARIN, Gerald A. Network security basics. *IEEE security & privacy*, 2005, 3.6: 68-72.
31. MALLIK, Avijit. Man-in-the-middle-attack: Understanding in simple words. *Cyberspace: Jurnal Pendidikan Teknologi Informasi*, 2019, 2.2: 109-134.
32. OPPLIGER, Rolf. *SSL and TLS: Theory and Practice*. Artech House, 2023.
33. NAYLOR, David, et al. The cost of the" s" in https. In: *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 2014. p. 133-140.
34. GIBLIN, Chris, et al. Securing Kafka with encryption-at-rest. In: *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 2021. p. 5378-5387.
35. ZHANG, Kan. Efficient Protocols for Signing Routing Messages. In: *NDSS*. 1998.
36. GILBERT, Henri; HANDSCHUH, Helena. Security analysis of SHA-256 and sisters. In: *International workshop on selected areas in cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 175-193.
37. Authentication, Authorisation, Access Control in RabbitMQ - [Електронний ресурс] – 2024 - Режим доступу до ресурсу: https://www.rabbitmq.com/docs/access-control
38. Securing Kafka: An In-depth Look at Kafka Authentication - [Електронний ресурс] – 2024 - Режим доступу до ресурсу: https://www.scaler.com/topics/kafka-tutorial/kafka-authentication/
39. ĆATOVIĆ, Amar; BUZAĐIJA, Nevzudin; LEMES, Samir. Microservice development using RabbitMQ message broker. *Science, Engineering and Technology*, 2022, 2.1: 30-37.
40. ASIM, Muhammad, et al. Event driven monitoring of composite services. In: *2013 International conference on social computing*. IEEE, 2013. p. 550-557.
41. FUENTES-GARCÍA, Marta; CAMACHO, José; MACIÁ-FERNÁNDEZ, Gabriel. Present and future of network security monitoring. *IEEE Access*, 2021, 9: 112744-112760.