УДК 004.02
DOI https://doi.org/10.32782/IT/2024-2-7

Володимир КИСЕЛЕВИЧ
студент фізико-математичного факультету, Житомирський державний університет імені Івана Франка, вул. В. Бердичівська, 40, м. Житомир, Україна, 10008

Олена УСАТА
кандидат педагогічних наук, доцент, завідувач кафедри комп'ютерних наук та інформаційних технологій, Житомирський державний університет імені Івана Франка, вул. В. Бердичівська, 40, м. Житомир, Україна, 10008
***ORCID:** 0000-0002-0610-7007*
***Scopus Author ID:** 57224620743*

Ярослава СІКОРА
кандидат педагогічних наук, доцент, доцент кафедри комп'ютерних наук та інформаційних технологій, Житомирський державний університет імені Івана Франка, вул. В. Бердичівська, 40, м. Житомир, Україна, 10008
***ORCID:** 0000-0003-2621-6638*
***Scopus Author ID:** 57224617763*

Дмитрій ВЕРБІВСЬКИЙ
кандидат педагогічних наук, доцент, доцент кафедри комп'ютерних наук та інформаційних технологій, Житомирський державний університет імені Івана Франка, вул. В. Бердичівська, 40, м. Житомир, Україна, 10008
***ORCID:** 0000-0002-5238-1189*
***Scopus Author ID:** 57224618415*

Дмитро ІВАНОВ
доктор технічних наук, доцент, професор кафедри комп'ютерних наук та інформаційних технологій, Житомирський державний університет імені Івана Франка, вул. В. Бердичівська, 40, м. Житомир, Україна, 10008
***ORCID:** 0000-0001-9956-6589*
***Scopus Author ID:** 57208379523*

Бібліографічний опис статті: Киселевич, В., Усата, О., Сікора, Я., Вербівський, Д., Іванов, Д. (2024). Мікросервісна архітектура: переваги та недоліки її практичного застосування. *Information Technology: Computer Science, Software Engineering and Cyber Security,* 2, 50–59, doi: https://doi.org/10.32782/IT/2024-2-7

МІКРОСЕРВІСНА АРХІТЕКТУРА: ПЕРЕВАГИ ТА НЕДОЛІКИ ЇЇ ПРАКТИЧНОГО ЗАСТОСУВАННЯ

Невпинний розвиток технологій та інновацій є рушієм удосконалення та появи підходів проєктування архітектури програмного забезпечення. Нагальність вибору правильної архітектури для програмних систем зумовлена зростанням складності та масштабів сучасних систем, в яких архітектура відіграє одну з вирішальних ролей у визначенні майбутнього успіху проєкту. ***Метою дослідження*** *є аналіз та визначення переваг і недоліків мікросервісної архітектури шляхом розгляду конкретних випадків практичного застосування її на системах різних масштабів.*

Методологія *полягає у аналізі прикладів застосування мікросервісної архітектури у випадках систем середньої та високої складності, визначенні переваг та ризиків з метою прийняття оптимального рішення щодо обрання архітектури системи, що відповідатиме її специфічним потребам та умовам.*

Наукова новизна. *У дослідженні детально розглядається та аналізується приклад нераціонального застосування мікросервісної архітектури в контексті системи середньої складності, а також приклад застосування монолітної архітектури в контексті системи високої складності, їх ризики та потенційні проблеми кожного з підходів, що в свою чергу, дозволяє осягнути межі ключових факторів, що впливають на доцільність вибору архітектурного підходу проєктування системи.*

Висновки. *Результати дослідження демонструють, що мікросервісна архітектура може стати як ефективним рішенням, так і джерелом значних проблем. Переваги та недоліки мікросервісної архітектури не можуть бути оцінені однозначно поза контекстом конкретної системи, для якої вона застосовується. Перш ніж приймати рішення про імплементацію мікросервісної архітектури, необхідно провести ґрунтовний аналіз вимог та можливостей, оскільки безпідставне та необґрунтоване використання даного архітектурного підходу може викликати технічні та фінансові проблеми у проєкту, котрі потенційно можуть призвести до його краху. Важливо враховувати усі аспекти, включаючи вимоги до масштабованості, швидкості розробки, складності управління та витрат на обслуговування. Таким чином, мікросервісна архітектура не є універсальним рішенням, і її успішне застосування залежить від багатьох факторів, які потрібно ретельно оцінювати у контексті кожної системи окремо.*

Ключові слова: *архітектура програмного забезпечення, монолітна архітектура, мікросервісна архітектура.*

Volodymyr KYSELEVYCH
Student of the Faculty of Physics and Mathematics, Zhytomyr Ivan Franko State University, 40, V. Berdychivska Str., Zhytomyr, Ukraine, 10008, kyselevych.v@gmail.com

Olena USATA
Ph.D. in Pedagogical Sciences, Associate Professor, Head of the Department of Computer Science and Information Technology, Zhytomyr Ivan Franko State University, 40, V. Berdychivska Str., Zhytomyr, Ukraine, 10008, o.j.usata@zu.edu.ua
ORCID: *0000-0002-0610-7007*
Scopus Author ID: *57224620743*

Yaroslava SIKORA
Ph.D. in Pedagogical Sciences, Associate Professor, Associate Professor of the Department of Computer Science and Information Technology, Zhytomyr Ivan Franko State University, 40, V. Berdychivska Str., Zhytomyr, Ukraine, 10008, sikora@zu.edu.ua
ORCID: *0000-0003-2621-6638*
Scopus Author ID: *57224617763*

Dmytrii VERBIVSKYI
Ph.D. in Pedagogical Sciences, Associate Professor, Associate Professor of the Department of Computer Science and Information Technology, Zhytomyr Ivan Franko State University, 40, V. Berdychivska Str., Zhytomyr, Ukraine, 10008, d_verbovskiy@ukr.net
ORCID: *0000-0002-5238-1189*
Scopus Author ID: *57224618415*

Dmytro IVANOV
Doctor of Technical Sciences, Associate Professor, Professor of the Department of Computer Science and Information Technology, Zhytomyr Ivan Franko State University, 40, V. Berdychivska Str., Zhytomyr, Ukraine, 10008, ivanovdmitro18@gmail.com
ORCID: *0000-0001-9956-6589*
Scopus Author ID: *57208379523*

To cite this article: Kyselevych, V., Usata, O., Sikora, Y., Verbivskyi, D., Ivanov, D. (2024). Mikroservisna arhitektura: perevagy ta nedoliky yi praktychnogo zastosuvannya [Microservice architecture: advantages and disadvantages of its practical application]. *Information Technology: Computer Science, Software Engineering and Cyber Security*, 2, 50–59, doi: https://doi.org/10.32782/IT/2024-2-7

MICROSERVICE ARCHITECTURE: ADVANTAGES AND DISADVANTAGES OF ITS PRACTICAL APPLICATION

The relentless advancement of technology and innovation drives the improvement and emergence of software architecture design approaches. The urgency of choosing the right architecture for software systems is driven by the increasing complexity and scale of modern systems, where architecture plays a crucial role in determining the future success of a project. ***The purpose of this study is*** *to analyze and identify the advantages and disadvantages of microservice architecture by examining specific cases of its practical application in systems of various scales.*

The methodology *involves analyzing examples of microservice architecture application in cases of medium and high complexity systems, identifying advantages and risks to make an optimal decision regarding the choice of system architecture that meets its specific needs and conditions.*

Scientific novelty. *The study thoroughly examines and analyzes an example of the irrational application of microservice architecture in the context of a medium-complexity system, as well as an example of the application of monolithic architecture in the context of a high-complexity system. It explores the risks and potential problems of each approach, which in turn allows for a comprehensive understanding of the key factors influencing the appropriateness of choosing a particular architectural design approach for a system.*

Conclusions. *The study's findings demonstrate that microservice architecture can be both an effective solution and a source of significant challenges. The advantages and disadvantages of microservice architecture cannot be assessed unequivocally outside the context of the specific system for which it is applied. Before deciding to implement microservice architecture, it is crucial to conduct a thorough analysis of the requirements and capabilities, as unfounded and unjustified use of this architectural approach can lead to technical and financial issues for the project, potentially resulting in its failure. It is important to consider all aspects, including scalability requirements, development speed, management complexity, and maintenance costs. Therefore, microservice architecture is not a universal solution, and its successful application depends on numerous factors that shall be carefully evaluated in the context of each individual system.*

Key words: *software architecture, monolithic architecture, microservice architecture.*

Актуальність проблеми. У світі інформаційних технологій, де кожне натискання клавіші збурює баланс великої мережі, архітектура програмного забезпечення є беззаперечною опорою. Вона є фундаментом, на якому ґрунтуються інновації та високотехнологічні досягнення. Подібно до кращих інженерів, які майстерно викладають кожну цеглину, архітектори програмного забезпечення ретельно конструюють структури, що витримують випробування часом та об'ємом вимог користувачів.

Архітектура програмного забезпечення – це не просто структура коду. Архітектура – це концептуальний фундамент, який визначає, як система буде працювати, взаємодіяти з іншими складовими, масштабуватися та змінюватися з часом. Це план, який визначає розробникам дорогу до створення чогось великого, довговічного та функціонального.

Правильно спроєктована архітектура програмного забезпечення вирішує численні проблеми, починаючи від ефективності та масштабованості до забезпечення безпеки та зручності для кінцевого користувача. Вона є важливим елементом в розробці будь-якої серйозної програми або системи, забезпечуючи стабільність, надійність та можливість росту.

Але найважливіше – архітектура програмного забезпечення формує майбутнє. Вона визначає можливості та обмеження того, що ми можемо створити. І в той же час вона відкриває двері для невпинного розвитку та інновацій, викликаючи нас завжди шукати інноваційні способи проєктування систем.

Нагальність вибору правильної архітектури для програмних систем неможливо переоцінити. Зі зростанням складності та масштабів сучасних програмних застосунків. Вибір архітектури може суттєво вплинути на продуктивність, масштабованість та обслуговування системи. Неправильний вибір архітектури може дуже дорого коштувати, призводячи до значних витрат, зниження ефективності роботи системи та збільшення зусиль та ресурсів на її підтримку. Важливо розуміти, на чому базувати вибір архітектури, враховуючи вимоги до системи, очікувані навантаження, можливість масштабування, технічні обмеження та довгострокову стратегію розвитку. Обґрунтований підхід до вибору архітектури допоможе забезпечити стійкість, гнучкість та економічну ефективність програмного забезпечення.

Аналіз останніх досліджень і публікацій. Останні дослідження та публікації заглиблюються у переваги та недоліки впровадження мікросервісної архітектури, що прямо свідчить про небайдужість до цієї проблематики.

Науковці відзначають, що мікросервісна архітектура дозволяє розбити програму на незалежні компоненти, що полегшує розробку, тестування та розгортання. Це значно відрізняється від монолітної архітектури, де всі компоненти працюють в одному блоці (Koschel, 2017).

Дослідники підкреслюють, що перехід до мікросервісів виправданий для великих і складних систем, які потребують частих оновлень і швидкого масштабування (Shabani, 2021). Вони також виділяють технічні та нетехнічні проблеми, з якими стикаються організації під час впровадження мікросервісної архітектури, такі як інтеграція, управління і підтримка системи.

Порівняння між монолітною та мікросервісною архітектурами показують, що мікросервіси забезпечують більшу надійність, масштабованість та відмовостійкість. Однак, вони також вимагають складнішого управління та нових підходів до забезпечення продуктивності та узгодженості даних (Lv, 2020).

Метою дослідження є аналіз та визначення переваг і недоліків мікросервісної архітектури шляхом розгляду конкретних випадків практичного застосування її на системах різних масштабів.

Виклад основного матеріалу. В області програмного забезпечення існує велика кількість підходів до проєктування архітектури. Незважаючи на цю різноманітність, багато з них базуються на одній спільній концепції – монолітній архітектурі. Монолітна архітектура являє собою традиційний підхід до розробки програмного забезпечення, в якому весь код розташований в одному монолітному блоку, що розглядається як єдине ціле (Mazlami, 2017).

Монолітна архітектура має глибокі коріння, що сягають появи комп'ютерної індустрії. Зі самого початку ери програмного забезпечення цей підхід був широко використовуваний, оскільки він простий у розумінні і реалізації. Однією з основних переваг монолітної архітектури є її простота. Оскільки увесь необхідний функціонал додатка розміщений в одному монолітному додатку, процеси розробки, тестування і розгортання стають менш складними. Крім того, монолітні додатки зазвичай характеризуються низькими витратами на підтримку, оскільки вони не вимагають складних механізмів взаємодії між компонентами.

Однак, враховуючи стрімкий розвиток технологій і зростання вимог до програмного забезпечення, імплементація монолітної архітектури може мати й суттєві обмеження. Підтримка великих монолітних додатків може стати складним завданням через їх розмір і складність. Крім того, модифікація окремих компонентів може бути складною через їх тісну взаємодію.

У безмежній сфері архітектури програмного забезпечення де інновація визначає прогрес, а адаптивність панує над усім, можемо спостерігати, як з'являються все нові революційні рішення, серед яких зарекомендувала себе мікросервісна архітектура. На відміну від своїх монолітних попередників, мікросервіси втілюють парадигмальний здвиг до модульності, масштабованості та гнучкості у розробці програмного забезпечення.

Термін мікросервісна архітектура став більш широко відомим і використовуваним завдяки ключовими експертам в сфері розробки програмного забезпечення, архітектури систем, які сприяли розвитку цієї концепції, таким як: Martin Fowler, James Lewis та інші.

У самому ядрі мікросервісної архітектури закладено принцип декомпозиції, розбиваючи складні програми на мережу менших, незалежних сервісів (Fowler, 2014). Кожен сервіс інкапсулює та виконує певну бізнес-функцію, працюючи автономно та комунікуючи з іншими сервісами через чітко визначений API (Рис. 1). Такий децентралізований підхід сприяє гнучкості, дозволяючи командам розробляти, впроваджувати та оновлювати сервіси незалежно один від одного, уникаючи обмежень монолітного коду.

Проте справжня сила мікросервісів полягає у їх здатності масштабуватися без особливих зусиль. Розподіляючи робоче навантаження між кількома сервісами, організації можуть оптимізувати використання ресурсів та динамічно реагувати на зміну потреб системи (Khaleq, 2023). Більше того, ізоляція сервісів забезпечує стійкість до відмов, гарантуючи, що помилка в одному компоненті не призведе до падіння усієї системи.

Однак із великою силою приходить велика складність. Управління мікросервісною архітектурою вимагає ретельного оркестрування сервісів та міцної інфраструктури. Інструменти, такі як контейнеризація та платформи для оркестрування, наприклад Kubernetes, Docker swarm тощо, виявилися необхідними супроводжуючими компонентами, забезпечуючи основу для впровадження та управління розподіленими системами.

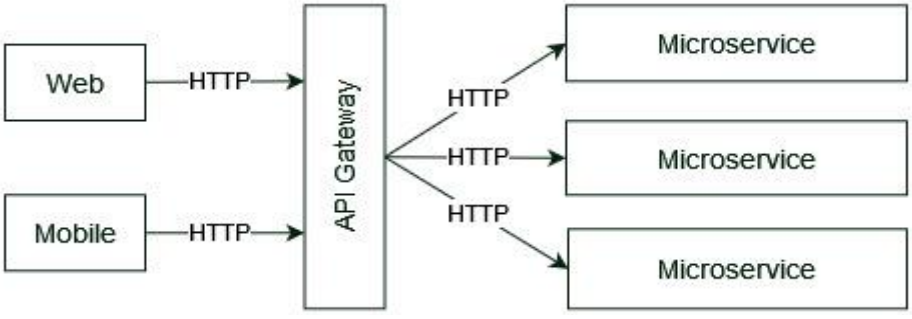

Рис. 1. Схема мікросервісної архітектури

Незважаючи на виклики, переваги мікросервісів все ще мають суттєву вагу. Від підвищення продуктивності розробників до покращеної стійкості системи, мікросервісна архітектура стала кутовим каменем сучасної інженерії програмного забезпечення, приводячи до активного розвитку інновацій.

На перший погляд, розглянувши обидва підходити до проєктування архітектури, можна було б зробити висновок, що імплементація мікросервісної архітектури це вирішення усіх недоліків монолітної архітектури. Проте, це зовсім не так. Ба більше, безпідставне та необґрунтоване використання мікросервісної архітектури може викликати технічні та фінансові проблеми у проєкту, котрі потенційно можуть призвести до його краху. Перед прийняттям рішення про використання мікросервісної архітектури або монолітної архітектури, необхідно враховувати всі фактори, включаючи розмір та складність системи, вимоги до масштабування та модернізації, вартість розробки та підтримки, а також вимоги до безпеки та надійності.

Для осягнення переваг та недоліків мікросервісної архітектури, а також наслідків недоречного використання або ігнорування даного підходу, розглянемо приклади імплементації різних підходів проєктування архітектури.

Спочатку пропонуємо до розгляду приклад імплементації мікросервісної архітектури для сервісу продажів квитків. У даному випадку завдання полягає у розробці проєкту (середньої складності) для сервісу продажів квитків розважальних заходів. Сервіс матиме наступні ключові складові функціоналу (Рис. 2):

- Автентифікація: реєстрація, авторизація в системі та керування обліковим записом.
- Система заходів: управління інформацією про доступні заходи, включаючи дати, час, місця проведення, вартість квитків тощо.
- Система квитків: процес продажу квитків, включаючи резервацію, оплату та відправлення квитків користувачам.
- Система оплати: оплата квитків онлайн, інтеграція з різними платіжними системами.

Проаналізувавши поведінку користувачів сервісу можна зробити висновок, що розподілення навантаження між компонентами є приблизно рівномірним між усіма ключовими компонентами системи. Це означає, що в перспективі та на момент імплементації системи немає необхідності збільшувати ресурси окремо для одного з компонентів системи, оскільки кожен з них буде використовуватися з приблизно однакових навантаженням.

Отже, розглянемо які проблеми можуть виникнути у разі застосування у зазначеному проєкті мікросервісної архітектури.

Фінансові витрати. Утримання відносно простого проєкту на мікросервісній архітектурі може обійтися значно дорожче, ніж монолітної архітектури (Singleton, 2016). Кожен мікросервіс потребує власну інфраструктуру: сервери, пам’ять, мережу та дисковий простір. Це збільшує вартість хостингу та підтримки, оскільки необхідно забезпечувати функціонування багатьох окремих сервісів. Крім того, для підтримки такої системи потрібні спеціалісти високої кваліфікації, які розуміють архітектуру мікросервісів, вміють налаштовувати та обслуговувати їх, забезпечувати безпеку і моніторинг. Наймання таких спеціалістів, їх навчання та підтримка їхньої кваліфікації також додають до загальних витрат на проєкт.

Складність управління. Мікросервісна архітектура додає значної складності у керуванні проєктом. Кожен сервіс має свою власну конфігурацію, яка потребує уважного управління, особливо у розподілених системах. Це включає налаштування мережевих взаємодій

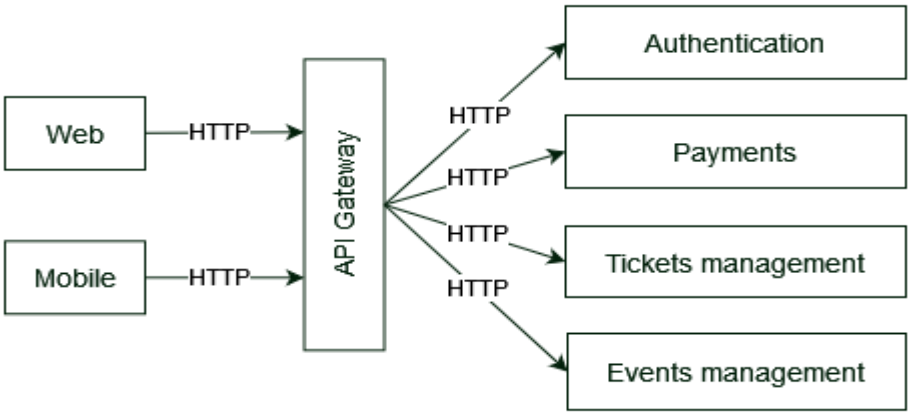

Рис. 2. Схема мікросервісної архітектури системи продажів квитків

між сервісами, розробку механізмів для забезпечення безперебійного функціонування і взаємодії сервісів, а також налаштування моніторингу та логування для виявлення проблем та їх усунення (Feng, 2020).

Моніторинг та логування є ключовими для виявлення проблем та відлагодження, але їх ефективне впровадження може бути викликом у разі великої кількості сервісів (Apolinario, 2021). У випадку несправності або відсутності даної інфраструктури, виявлення проблемного сервісу може бути складним завданням.

Розгортання та сумісність. Розгортання нових версій кожного сервісу та управління сумісністю між ними вимагає уважного планування та керування (Zeng, 2023). Кожен сервіс може оновлюватися незалежно, що ускладнює забезпечення сумісності між різними версіями сервісів. Наприклад, якщо нова версія сервісу автентифікації не сумісна зі старими версіями інших сервісів, це може призвести до збоїв у роботі всієї системи.

Безпека. Забезпечення безпеки кожного сервісу від зовнішніх атак і внутрішніх загроз також є важливим завданням, яке потребує постійної уваги. Кожен мікросервіс повинен бути належним чином захищений, що ускладнює загальне управління безпекою системи. Необхідність в управлінні доступом, аутентифікацією та шифруванням для кожного окремого сервісу збільшує загальну складність архітектури (Mateus-Coelho, 2021).

Інтеграція з існуючими системами. Інтеграція з існуючими системами може становити додаткові складнощі через різні технології та протоколи. Наприклад, якщо один з мікросервісів потребує інтеграції з зовнішньою системою, яка використовує інший протокол або формат даних, це може вимагати додаткових зусиль для розробки адаптерів або конвертерів (Smid, 2019).

Низька продуктивність. Мікросервісна архітектура може мати низьку продуктивність через необхідність взаємодії між сервісами через мережу, що в свою чергу призведе до затримок у комунікації (Ramu, 2023). Кожен окремий мікросервіс може мати свої власні проблеми з продуктивністю, що також може впливати на всю систему. Наприклад, якщо сервіс оплати має високу затримку у відповіді, це може вплинути на весь процес замовлення квитків.

Координація. У мікросервісній архітектурі важливо забезпечити координацію між різними сервісами. Це може бути складно, особливо коли сервіси розробляються та підтримуються різними командами розробників. Без належної координації можуть виникати проблеми з узгодженістю даних та сумісністю між сервісами (Iasio, 2020).

Висновок до прикладу. У випадку відносно невеликих проєктів, до прикладу таких як онлайн сервіс продажів квитків, мікросервісна архітектура може виявитися вкрай неефективною та нераціональною з багатьох причин. Утримання мікросервісної інфраструктури, управління численними конфігураціями, забезпечення безпеки та координація між сервісами створюють значні труднощі, які можуть не виправдатися перевагами архітектури.

В даному випадку, можна розглянути монолітну архітектуру, яка забезпечить простоту управління, менші витрати та більш ефективне використання ресурсів. Монолітна архітектура дозволяє зосередити всі компоненти системи в одному додатку, що спрощує процеси розробки, тестування, розгортання та моніторингу. Це також дозволяє легко масштабувати систему в межах одного серверу або кластера серверів без необхідності розподіляти ресурси між численними мікросервісами.

Ще одним підходом, який можна взяти до уваги, є модульна монолітна архітектура. Вона поєднує в собі переваги монолітного та мікросервісного підходів, дозволяючи структурно організувати додаток на модулі, які семантично розділені, але працюють в межах одного монолітного додатку (Li, 2024). Це забезпечує кращу ізоляцію коду, спрощує управління залежностями між модулями та дозволяє уникнути проблем з масштабуванням та координацією, властивих мікросервісам.

Перейдемо до розгляду прикладу імплементації монолітної архітектури для банківської системи. У даному випадку завдання полягає у розробці проєкту (високої складності) для високонавантаженої банківської системи.

У сучасному світі банківські системи вимагають надзвичайно високого рівня надійності, безпеки та масштабованості. Банківська система складається з десятків компонентів різного призначення та складності, проте для простоти розуміння розглянемо в прикладі наступні ключові складові (Рис. 3):

- Автентифікація: реєстрація, авторизація в системі та керування обліковим записом.
- Система фінансових транзакцій: обробка фінансових транзакцій, включаючи перекази, оплату та інші операції.
- Система управління рахунками: управління рахунками користувачів.
- Аналітика та звітність: збір та аналіз даних про фінансові транзакції та інші дії користувачів.

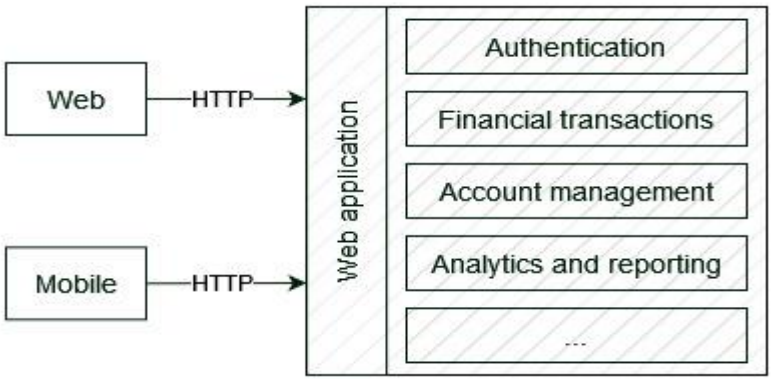

Рис. 3. Схема монолітної архітектури банківської системи

Банківська система складається з різноманітних компонентів, кожен з яких виконує специфічні функції та має різний ступінь навантаження. Наприклад, такі ключові компоненти, як обробка транзакцій та управління рахунками, зазвичай зазнають значного та постійного навантаження, яке зростає пропорційно зі збільшенням кількості користувачів. Це пов'язано з тим, що кожна нова транзакція чи операція на рахунку створює додаткове навантаження на систему, вимагаючи більшої кількості обчислювальних ресурсів для їх обробки.

З іншого боку, компоненти, такі як аналітика та звітність, мають більш стабільний характер навантаження. Вони, як правило, виконують періодичні завдання і не залежать безпосередньо від кількості користувачів системи. Навантаження на ці компоненти може залишатися стабільним або навіть зменшуватися з часом завдяки оптимізації алгоритмів та ефективному використанню ресурсів.

У зв'язку з цим, в перспективі окремим компонентам банківської системи, таким як: обробка транзакцій, управління рахунками тощо, необхідно буде збільшувати ресурси для задоволення зростаючих потреб користувачів, тоді як інші компоненти, такі як аналітика та звітність, можуть залишатися стабільними та не потребувати додаткових ресурсів.

Отже, розглянемо які проблеми можуть виникнути у разі застосування у зазначеному проєкті монолітної архітектури.

Масштабування. У випадку монолітної архітектури всі ці компоненти інтегровані в єдину систему, яка встановлюється на одному сервері або кластері серверів. Це ускладнює горизонтальне масштабування. Уявімо, що з часом навантаження на фунціонал фінансових транзакцій збільшилося на 200%. У монолітній архітектурі єдиним вирішенням даної проблеми буде розширення ресурсів для всієї системи. Це означає, що всі компоненти, навіть ті, що не зазнали збільшення навантаження, отримають додаткові ресурси, що є вкрай неефективним та призведе до зайвих фінансових витрат (Blinowski, 2022). Ба більше, розширення одного великого серверу для обробки всіх запитів може бути значно дорожчим, ніж створення додаткового екземпляру мікросервісу з необхідним функціоналом на окремому сервері, забезпечуючи тим самим ефективне використання ресурсів.

Залежності та зв'язність. Усі компоненти тісно пов'язані між собою, що ускладнює розробку та підтримку системи (Aljaloud, 2023). Наприклад, зміна в модулі аналітики може непередбачувано вплинути на систему авторизації. Це призводить до необхідності ретельного регресивного тестування всієї системи після кожної зміни, що в контексті даної системи може бути дуже ресурсомістким завданням.

Проблеми з оновленнями. Кожне оновлення вимагатиме перезапуску всієї системи, що може призвести до простоїв та порушення роботи усієї системи (Saidi, 2023).

Технічний борг. Використання різних технологій для окремих компонентів у монолітній архітектурі неможливе. Наприклад, якщо компонент аналітики та звітності потребує оптимізації за допомогою іншої мови програмування, це буде неможливо реалізувати в монолітній архітектури, оскільки одне середовище виконання коду підтримує лише одну мову програмування (Fawler, 2015).

Обмежена гнучкість розвитку. Монолітні системи зазвичай мають складну структуру, що робить їх менш гнучкими до змін і впровадження нових функцій (Mosleh, 2018). Це може сповільнювати інновації та адаптацію до нових ринкових вимог.

Тривалий цикл розгортання. Оновлення монолітної системи вимагає ретельного планування та координації, що може збільшити час між випусками нових версій (Chouhan,

2023). Це може бути проблемою в динамічних середовищах, де важлива швидкість впровадження змін.

Вразливість до збоїв. Монолітні системи більш вразливі до збоїв, оскільки проблема в одному компоненті може вплинути на роботу всієї системи, що є вкрай критичним для банківської системи, де безперервність обслуговування є надзвичайно важливою (Tapia, 2020). У мікросервісній архітектурі збої обмежуються окремими сервісами, що зменшує ризик повного виходу системи з ладу.

Висновок до прикладу. Доцільний перехід до мікросервісної архітектури дозволяє уникнути багатьох проблем, властивих монолітним системам. Кожен компонент системи може бути розроблений, розгорнутий і масштабований незалежно від інших, що забезпечує більш гнучке та ефективне управління ресурсами. Це також підвищує стійкість системи до збоїв, спрощує процеси тестування та відлагодження, а також дозволяє використовувати найкращі технології для кожного окремого компонента.

За потреби, варто розглядати гібридний підхід, який може забезпечити баланс між перевагами мікросервісної та монолітної архітектур. Гібридний підхід дозволяє виносити лише необхідні компоненти системи в мікросервіси, залишаючи інші компоненти в межах монолітного додатку. Це дозволяє поступово адаптуватися до мікросервісної архітектури, знижуючи ризики та витрати, пов’язані з повним переходом.

Наприклад, у великій банківській системі компоненти, які часто змінюються або потребують високої масштабованості, такі як система фінансових транзакцій або система аналітики, можуть бути винесені в окремі мікросервіси. Інші компоненти, які менш критичні до змін або мають стабільні навантаження, можуть залишатися в монолітній архітектурі. Це дозволяє зосередити ресурси на тих частинах системи, які найбільше потребують гнучкості та масштабованості, зберігаючи при цьому простоту управління іншими компонентами.

Висновки і перспективи подальших досліджень. Мікросервісна архітектура має численні переваги, такі як можливість незалежного розгортання окремих компонентів, підвищена гнучкість та масштабованість, а також полегшене управління складними системами. Однак, її використання супроводжується також певними недоліками, включаючи складність у налаштуванні взаємодії між сервісами, збільшені вимоги до інфраструктури та потенційні проблеми з відлагодженням та забезпеченням безпеки. Проте, найголовнішим висновком даного дослідження є те, що переваги та недоліки мікросервісної архітектури не можуть бути оцінені однозначно поза контекстом конкретного проєкту та системи, для якої вона імплементується.

Перш ніж приймати рішення про імплементацію мікросервісної архітектури, необхідно провести ґрунтовний аналіз проєкту, його потреб та можливостей. Успішне застосування залежить від глибокого розуміння контексту, у якому вона впроваджується, компетенцій команди розробників, прорахованого навантаження, вимог до безпеки та надійності, вимог до масштабування та модернізації, вимог до швидкості та гнучкості розробки, а також наявних ресурсів тощо. Тільки з урахуванням всіх цих факторів можна прийняти обґрунтоване рішення про раціональність та ефективність використання мікросервісної архітектури в контексті конкретної системи. Безпідставне та необґрунтоване використання мікросервісної архітектури може викликати технічні та фінансові проблеми у проєкту, котрі потенційно можуть призвести до його краху.

Таким чином, мікросервісна архітектура може стати як ефективним рішенням, так і джерелом значних проблем. Тільки через детальний та зважений аналіз можна прийняти оптимальне рішення щодо обрання архітектури системи, що відповідатиме специфічним потребам та умовам конкретного проєкту.

У процесі аналізу досліджень та вивчення різних аспектів мікросервісної архітектури можна виокремити наступні перспективи подальших досліджень: вивчення гібридних підходів, які поєднують елементи монолітних і мікросервісних архітектур, що може допомогти знайти оптимальний баланс між гнучкістю та складністю; дослідження нових інструментів і практик DevOps, які можуть спростити управління складністю мікросервісних архітектур; аналіз методів ефективного горизонтального та вертикального масштабування як для монолітних, так і для мікросервісних систем; вивчення підходів до динамічного розподілу ресурсів і оптимізації витрат на інфраструктуру.

Ці напрямки подальших досліджень можуть допомогти розробникам і архітекторам краще розуміти та нівелювати переваги та недоліки мікросервісної архітектури й ефективніше застосовувати її в залежності від технічних особливостей проєктів.

ЛІТЕРАТУРА:

1. Koschel A., Astrova I., Dotterl J. Making the move to microservice architecture. *International Conference on Information Society (i-Society)*. Dublin, Ireland, 2017. pp. 74–79. DOI: 10.23919/i-Society.2017.8354675.

2. Lv G., Huang B., Liang Z., Qin M., Zou H., Li Z. Microservices: architecture, container, and challenges. *IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. Macau, China, 2020. pp. 629–635. DOI: 10.1109/QRS-C51114.2020.00107.

3. Shabani I., Mëziu E., Berisha B., Biba, T. Design of Modern Distributed Systems based on Microservices Architecture. *International Journal of Advanced Computer Science and Applications* (IJACSA), 12(2), 2021. DOI: 10.14569/IJACSA.2021.0120220

4. Mazlami G., Cito J., Leitner P. Extraction of Microservices from Monolithic Software Architectures. *2017 IEEE International Conference on Web Services (ICWS)*. Honolulu, HI, USA, 2017. pp. 524–531. DOI: 10.1109/icws.2017.61

5. Fowler M. Microservices, a definition of this new architectural term. URL: https://martinfowler.com/articles/microservices.html

6. Khaleq A. A., Ra, I. Intelligent Microservices Autoscaling Module Using Reinforcement Learning. *Springer Science Business Media*. 2023. Vol. 26, Issue 5. pp. 2789–2800. DOI: 10.1007/s10586-023-03999-8

7. Singleton, A. The Economics of Microservices, *Cloud Computing*. 2016. Vol. 3, № 5, pp. 16–20. DOI: 10.1109/MCC.2016.109.

8. Feng Z., Xu Y., Xue X., Chen S. Review on the Development of Microservice Architecture. Journal of Computer Research and Development. 2020, Vol. 57(5). Pp. 1103–1122. DOI: 10.7544/issn1000-1239.2020.20190460

9. Daniel R. F. Apolinário, Breno B. N. de França. A method for monitoring the coupling evolution of microservice-based architectures. Journal of the Brazilian Computer Society. 2021. Vol. 27(1). DOI: 10.1186/s13173-021-00120-y

10. Zeng R., Niu Y., Qiao L. A Novel Construction Technology of Microservice Multi-Instance System Automatic Deployment and Upgrade,» *2023 International Conference on Networking and Network Applications (NaNA)*, Qingdao. China, 2023. pp. 674–679, DOI: 10.1109/NaNA60121.2023.00116.

11. Mateus-Coelho N., Cruz-Cunha M., Ferreira L. Security in Microservices Architectures. Procedia Computer Science. 2021. Vol.181(6). Pp. 1225–1236. DOI: 10.1016/j.procs.2021.01.320

12. Smid A., Wang R., Cerny T. Case study on data communication in microservice architecture. *Proceedings of the Conference on Research in Adaptive and Convergent Systems*. 2019. DOI: 10.1145/3338840.3355659

13. Ramu V. B. Performance Impact of Microservices Architecture. *The Review of Contemporary Scientific and Academic Studies*. Vol. 3(6). DOI: 10.55454/rcsas.3.06.2023.010

14. Iasio A. D., Zimeo E. A framework for microservices synchronization. *Software: Practice and Experience.* Vol. 51(1). Pp. 25–45. DOI: https://doi.org/10.1002/spe.2877.

15. Su R., Li X. Modular Monolith: Is This the Trend in Software Architecture?. Cornell University. 2024. DOI: 10.48550/arxiv.2401.11867

16. Blinowski G., Ojdowska A., Przybyłek A. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation, *IEEE Access.* 2022. Vol. 10. pp. 20357–20374. DOI: 10.1109/ACCESS.2022.3152803.

17. Aljaloud A., Razzaq A. An Innovative Metric-based Clustering Approach for Increased Scalability and Dependency Elimination in Monolithic Legacy Systems. *Engineering, Technology & Applied Science Research*. 2023. Vol. 13(4). Pp. 11375–113876. DOI: 10.48084/etasr.6048

18. Saidi M., Tissaoui A., Faiz S. From a Monolith to a Microservices Architecture Based Dependencies. *Intelligent Systems Design and Applications.* 2023. DOI: 10.1007/978-3-031-35501-1_4.

19. Fowler M. Microservice Architecture: Technology Diversity. URL: https://martinfowler.com/articles/microservice-trade-offs.html

20. Mosleh M., Dalili K., Heydari B. Distributed or Monolithic? A Computational Architecture Decision Framework. *Institute of Electrical and Electronics Engineers.* 2018. Vol. 12(1). Pp. 125–136. DOI: 10.1109/jsyst.2016.2594290.

21. Chouhan U., Tiwari V., Kumar H. P. M. *2023 3rd Asian Conference on Innovation in Technology (ASIANCON)*. Ravet IN, India, 2023. pp. 1–7. DOI: 10.1109/asiancon58793.2023.10270721

22. Tapia F., Mora M. A., Fuerte W., Aules H., Flores E., Toulkeridis T. From Monolithic Systems to Microservices: A Comparative Study of Performance. *Multidisciplinary Digital Publishing Institute*. 2020. Vol. 10(17), 5797. DOI: 10.3390/app10175797

REFERENCES:

1. Koschel, A., Astrova, I., & Dotterl, J. (2017, July 1). Making The Move To Microservice Architecture. DOI: 10.23919/i-society.2017.8354675.

2. Lv, G., Huang, B., Liang, Z., Qin, M., Zou, H. & Li, Z. (2020, December) Microservices: Architecture, Container, And Challenges. DOI: 10.1109/qrs-c51114.2020.00107.

3. Shabani, I., Mëziu, E., Berisha, B. & Biba, T. (2021, January 1). Design of Modern Distributed Systems based on Microservices Architecture. Science and Information Organization, 12(2). DOI: 10.14569/ijacsa.2021.0120220.

4. Mazlami, G., Cito, J. & Leitner, P. (2017, June 1). Extraction of Microservices from Monolithic Software Architectures. DOI: 10.1109/icws.2017.61.

5. Fowler, M. (2014, March 25). Microservice Architecture. https://martinfowler.com/articles/microservices.html.

6. Khaleq, A. A. & Ra, I. (2023, April 28). Intelligent microservices autoscaling module using reinforcement learning. Springer Science Business Media, 26(5), 2789–2800. DOI: 10.1007/s10586-023-03999-8.

7. Singleton, A. (2016, September 1). The Economics of Microservices. Institute of Electrical and Electronics Engineers, 3(5), 16–20. DOI: 10.1109/MCC.2016.109.

8. Feng, Z., Yan-wei, X., Xue, X. & Chen, S. (2020, May 1). Review on the Development of Microservice Architecture. Science Press, 57(5). DOI: 10.7544/issn1000-1239.2020.20190460

9. Apolinario, D. & Franca, B B N D. (2021, December 1). A method for monitoring the coupling evolution of microservice-based architectures. Springer Science Business Media, 27(1). DOI: 10.1186/s13173-021-00120-y

10. Zeng, R., Niu, Y. & Qiao, L. (2023, August 1). A Novel Construction Technology of Microservice Multi-Instance System Automatic Deployment and Upgrade. DOI: 10.1109/nana60121.2023.00116

11. Mateus-Coelho, N., Cruz-Cunha, M. & Ferreira, L. (2021, January 1). Security in Microservices Architectures. Elsevier BV, 181, 1225–1236. DOI: 10.1016/j.procs.2021.01.320

12. Smid, A., Wang, R. & Cerny, T. (2019, September 24). Case study on data communication in microservice architecture. DOI: 10.1145/3338840.3355659

13. Ramu, V. B. (2023, June 1). Performance Impact of Microservices Architecture. The Review of Contemporary Scientific and Academic Studies, 3(6). DOI: 10.55454/rcsas.3.06.2023.010

14. Iasio, A. D. & Zimeo, E. (2020, August 13). A framework for microservices synchronization. Wiley-Blackwell, 51(1), 25-45. DOI: 10.1002/spe.2877

15. Li, C. & Li, X. (2024, January 1). Modular Monolith: Is This the Trend in Software Architecture?. Cornell University. DOI: 10.48550/arxiv.2401.11867

16. Blinowski, G., Ojdowska, A. & Przybyłek, A. (2022, January 1). Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. Institute of Electrical and Electronics Engineers, 10, 20357–20374. DOI: 10.1109/access.2022.3152803

17. Aljaloud, A. & Razzaq, A. (2023, August 9). An Innovative Metric-based Clustering Approach for Increased Scalability and Dependency Elimination in Monolithic Legacy Systems. Engineering, Technology & Applied Science Research, 13(4), 11375–113876. DOI: 10.48084/etasr.6048

18. Saidi, M., Tissaoui, A. & Faiz, S. (2023, January 1). From a Monolith to a Microservices Architecture Based Dependencies. Springer International Publishing, 34–44. DOI: 10.1007/978-3-031-35501-1_4

19. Fowler, M. (2015, July 1). Microservice Architecture: Technology Diversity. https://martinfowler.com/articles/microservice-trade-offs.html

20. Mosleh, M., Dalili, K. & Heydari, B. (2018, March 1). Distributed or Monolithic? A Computational Architecture Decision Framework. Institute of Electrical and Electronics Engineers, 12(1), 125-136. DOI: 10.1109/jsyst.2016.2594290

21. Chouhan, U., Tiwari, V. & Kumar, H P M. (2023, August 25). Comparing Microservices and Monolithic Applications in a DevOps Context. DOI: 10.1109/asiancon58793.2023.10270721

22. Tapia, F., Mora, M A C., Fuertes, W., Aules, H., Flores, E O C. & Toulkeridis, T. (2020, August 21). From Monolithic Systems to Microservices: A Comparative Study of Performance. Multidisciplinary Digital Publishing Institute, 10(17), 5797–5797. DOI: 10.3390/app10175797