КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 2 (34), 2026
ISSN 2663 – 4023
DOI 10.28925/2663-4023.2026.34.1327
УДК 004.415.2:004.65

Трофименко Олена Григорівна
кандидат технічних наук, доцент, доцент кафедри програмної інженерії,
Національний університет "Одеська юридична академія", Одеса, Україна,
ORCID:0000-0001-7626-0886
trofymenko@onua.edu.ua

Лобода Юлія Геннадіївна
кандидат педагогічних наук, доцент, доцент кафедри програмної інженерії,
Національний університет «Одеська юридична академія», Одеса, Україна,
ORCID:0000-0001-7083-552X
loboda@onua.edu.ua

Логінова Наталія Іванівна
кандидат педагогічних наук, доцент, завідувачка кафедри програмної інженерії,
Національний університет «Одеська юридична академія», Одеса, Україна,
ORCID:0000-0002-9475-6188
loginova@onua.edu.ua

Ніколаєнко Сергій Валентинович
кандидат технічних наук, старший інженер з розробки програмного забезпечення,
Одеса, Україна,
ORCID:0009-0004-7276-1818
serezhanik@gmail.com

Карагуц Олександр Сергійович
студент факультету кібербезпеки та інформаційних технологій,
Національний університет «Одеська юридична академія», Одеса, Україна,
ORCID:0009-0008-6593-5938
sashaKaraguts@gmail.com

КЛАСИФІКАЦІЯ ПРОБЛЕМ ПРОЄКТУВАННЯ БАЗ ДАНИХ У МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ ТА РЕКОМЕНДАЦІЇ ЩОДО ЇХ ВИРІШЕННЯ

Анотація. Статтю присвячено дослідженню проблем проєктування систем зберігання даних у мікросервісній архітектурі сучасних інформаційних систем. Актуальність роботи обумовлена широким використанням мікросервісного підходу та концепції Polyglot Persistence, що забезпечують гнучкість і масштабованість програмних систем, проте водночас істотно ускладнюють організацію зберігання, інтеграції та узгодженості даних. Особливо гостро ці проблеми проявляються в інформаційних системах сфери HoReCa, для яких характерні інтенсивні транзакційні навантаження, висока частота оновлення даних, інтеграція із зовнішніми сервісами та необхідність підтримки роботи в режимі реального часу. Метою роботи є комплексний аналіз, систематизація та багаторівнева класифікація проблем проєктування систем зберігання даних у мікросервісній архітектурі, а також формування практичних рекомендацій щодо їх вирішення на основі сучасних архітектурних підходів і патернів. У дослідженні використано методи системного аналізу, порівняльного аналізу сучасних архітектурних рішень, узагальнення наукових публікацій та практик проєктування мікросервісних інформаційних систем. Запропоновано авторську класифікацію проблем, яка охоплює три взаємопов'язані рівні: концептуальний, архітектурний та операційний. Для кожного рівня визначено характерні проблеми, проаналізовано причини їх виникнення, наведено приклади з інформаційних систем сфери HoReCa, електронної комерції та банківських сервісів, а також сформовано комплекс рекомендацій щодо їх практичного вирішення. У роботі показано, що ефективність організації систем зберігання даних визначається не лише правильним вибором технологій баз даних, а й обґрунтованим визначенням меж бізнес-контекстів, принципів Data Ownership, моделей узгодженості даних, способів інтеграції різнорідних сховищ та організації експлуатації мікросервісної системи. Окрему увагу приділено використанню

архітектурних патернів Database per Service, Saga, CQRS, Event Sourcing, Transactional Outbox, Transactional Inbox, API Composition та Domain Events як практичних засобів вирішення виявлених проблем. Наукова новизна дослідження полягає у систематизації проблем керування даними в мікросервісній архітектурі шляхом їх розподілу за концептуальним, архітектурним та операційним рівнями, визначення причинно-наслідкових зв'язків між рівнями та встановлення відповідності між виявленими проблемами, архітектурними патернами, умовами їх застосування й супровідними обмеженнями. Практичне значення отриманих результатів полягає у можливості використання сформованих рекомендацій під час розроблення масштабованих інформаційних систем, побудованих на основі мікросервісної архітектури.

Ключові слова: база даних; СКБД; SQL; NoSQL; Database per Service; поліглотна персистентність; мікросервіси; мікросервісна архітектура; HoReCa; архітектурні патерни.

ВСТУП

Постановка проблеми. Активна цифрова трансформація бізнес-процесів зумовлює стрімке зростання попиту на масштабовані веб- та мобільні інформаційні системи, здатні одночасно забезпечувати високу продуктивність, надійність і підтримку широкого спектра функціональних можливостей. Особливо це стосується систем електронної комерції та сфери HoReCa, які поєднують оброблення фінансових транзакцій, керування каталогами товарів і послуг, персоналізацію контенту, аналітичну обробку даних, інтеграцію із зовнішніми сервісами та підтримку роботи в режимі реального часу.

Традиційний підхід до використання монолітної архітектури зі спільною реляційною базою даних дедалі частіше не відповідає вимогам сучасних інформаційних систем. Різні функціональні компоненти мають неоднакові вимоги до структури даних, продуктивності, масштабованості, транзакційності та швидкодії. Зокрема, реляційні системи керування базами даних (СКБД) забезпечують високий рівень цілісності та ACID-гарантії, необхідні для реалізації фінансових операцій, проте є менш ефективними під час роботи з напівструктурованими або динамічними даними. Натомість документоорієнтовані NoSQL-сховища забезпечують гнучкість моделі даних і горизонтальне масштабування, однак не завжди здатні повною мірою задовольнити вимоги до транзакційної узгодженості [1].

Одним із провідних підходів до побудови сучасних інформаційних систем є мікросервісна архітектура, яка передбачає декомпозицію програмної системи на незалежні функціональні сервіси. На відміну від монолітних застосунків, де використовується спільна база даних, у мікросервісній архітектурі кожен сервіс, як правило, має власне сховище даних відповідно до принципу Database per Service [2]. Такий підхід забезпечує автономність сервісів, незалежне масштабування та спрощує їх подальший розвиток, проте водночас суттєво ускладнює проєктування архітектури зберігання даних.

У цих умовах широкого застосування набув підхід Polyglot Persistence, за якого окремі мікросервіси використовують різні типи СКБД залежно від власних функціональних і нефункціональних вимог [3]. Це дозволяє поєднувати переваги реляційних, документоорієнтованих, графових, колонкових та інших спеціалізованих сховищ даних у межах єдиної інформаційної системи [4]. Водночас використання різнорідних моделей зберігання породжує низку складних архітектурних проблем, пов'язаних із визначенням меж мікросервісів, розподілом даних між ними, вибором типу сховища для окремих сервісів, забезпеченням узгодженості даних, синхронізацією змін і підтримкою міжсервісної взаємодії.

Попри значну кількість досліджень, присвячених мікросервісній архітектурі, Polyglot Persistence та окремим технологіям зберігання даних, питання комплексного проєктування архітектури зберігання даних залишаються недостатньо систематизованими. Більшість наукових праць зосереджена на описі окремих типів сховищ, механізмів забезпечення узгодженості даних або архітектурних патернів, однак не пропонує цілісної класифікації проблем, що виникають на різних етапах проєктування та експлуатації мікросервісних систем. Зокрема, недостатньо дослідженим залишається взаємозв'язок між концептуальними рішеннями щодо декомпозиції предметної області, архітектурними рішеннями щодо вибору технологій зберігання даних та операційними аспектами забезпечення узгодженості, продуктивності й супроводу системи. Саме це зумовлює актуальність проведеного дослідження, спрямованого на формування багаторівневої класифікації проблем організації систем зберігання даних у мікросервісній архітектурі та розроблення практичних рекомендацій щодо їх вирішення.

Аналіз останніх досліджень і публікацій. Розвиток мікросервісної архітектури та поширення підходу Database per Service зумовили значне зростання кількості досліджень, присвячених організації зберігання даних у розподілених інформаційних системах. Особлива увага сучасних науковців

приділяється питанням вибору типу баз даних для окремих мікросервісів, застосуванню концепції Polyglot Persistence, забезпеченню узгодженості даних, масштабованості та підтримці міжсервісної взаємодії. Фундаментальні засади використання різних моделей зберігання даних у межах однієї інформаційної системи були закладені М. Фаулером, який сформулював концепцію Polyglot Persistence як підхід до використання оптимального типу сховища для кожного класу даних [5]. Подальший розвиток цієї ідеї знайшов відображення у сучасних дослідженнях, присвячених застосуванню принципу Database per Service, який розглядається як один із базових механізмів забезпечення автономності та незалежного розвитку мікросервісів [6, 7]. Окремий напрям сучасних досліджень присвячений вибору типу сховища даних для мікросервісів. У роботах [2, 3] проведено порівняльний аналіз реляційних, документоорієнтованих та інших типів баз даних, показано їх вплив на масштабованість, продуктивність, операційну складність та вартість супроводу інформаційних систем. Автори роботи [8] запропонували критерії адаптивного вибору між PostgreSQL, Oracle, MySQL та MongoDB залежно від характеристик предметної області й структури даних. Інший напрям досліджень пов'язаний із проблемами інтеграції різнорідних сховищ даних. Зокрема, у роботі [9] запропоновано метод міграції монолітної бази даних до мультимодельної архітектури на основі Polyglot Persistence, який забезпечує підвищення зрозумілості архітектури, покращення переносимості та підтримуваності програмної системи. Водночас у низці сучасних праць [3, 10] досліджуються проблеми забезпечення узгодженості даних, підтримки розподілених транзакцій, масштабування та взаємодії між сервісами в умовах використання різних типів сховищ даних. Вагомий внесок у теоретичне обґрунтування сучасних підходів до організації даних у розподілених системах зробили дослідження, присвячені CAP-теоремі, еволюції архітектур транзакційних систем та принципам побудови масштабованих розподілених платформ [11, 12]. Отримані результати підтверджують, що значна частина проблем керування даними в мікросервісній архітектурі має системний характер і не може бути повністю усунута лише засобами програмної інженерії.

Узагальнення сучасних досліджень свідчить, що наявні роботи переважно зосереджені на окремих аспектах проєктування архітектури зберігання даних: виборі типу бази даних, застосуванні Polyglot Persistence, реалізації розподілених транзакцій, забезпеченні узгодженості даних або використанні окремих архітектурних патернів. Проте зазначені проблеми переважно розглядаються ізольовано одна від одної без їх систематизації та визначення взаємозв'язків між ними. Крім того, більшість сучасних досліджень орієнтована на універсальні програмні платформи або високонавантажені інформаційні системи загального призначення (Netflix, Uber, Amazon, Shopify тощо) і практично не враховує специфіку інформаційних систем сфери HoReCa, для яких характерні пікові навантаження у визначені часові інтервали, тайм-аутна логіка оброблення замовлень, значна варіативність структури меню, одночасне поєднання транзакційних та документоорієнтованих даних, а також підвищені вимоги до оперативності оновлення інформації та роботи в режимі реального часу.

Наявні праці переважно розглядають окремі аспекти керування даними в мікросервісних системах, тоді як взаємозв'язок концептуальних, архітектурних та операційних проблем залишається недостатньо систематизованим.

Метою роботи є комплексний аналіз, систематизація та багаторівнева класифікація проблем проєктування систем зберігання даних у мікросервісній архітектурі, що охоплює концептуальний, архітектурний та операційний рівні, а також розроблення практичних рекомендацій щодо їх вирішення на основі сучасних архітектурних підходів і патернів.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Сфера громадського харчування є одним із найбільш динамічних напрямів цифрової трансформації. За даними аналітичних досліджень [13], світовий ринок онлайн-замовлення їжі у 2024 році досяг 288,84 млрд дол. США, а до 2030 року прогнозується його зростання із середньорічним темпом 9,4 %. Це свідчить про постійне збільшення навантаження на цифрові платформи та зростання вимог до їх продуктивності, доступності й масштабованості.

Інформаційні системи сфери HoReCa мають низку специфічних особливостей, які безпосередньо впливають на організацію зберігання даних [14]. Такі системи повинні одночасно підтримувати транзакційні операції (оформлення та оплата замовлень), роботу з напівструктурованими даними (меню, фотографії, модифікатори страв, акційний контент), інтеграцію з платіжними системами, службами доставки, картографічними сервісами та механізмами сповіщень у режимі реального часу. Крім того, для HoReCa характерні виражені пікові навантаження у певні часові інтервали, висока частота оновлення контенту та підвищені вимоги до оперативності оброблення подій.

Для дослідження проблем організації зберігання даних у мікросервісній архітектурі розглянемо типову інформаційну систему сфери HoReCa, побудовану за мікросервісною архітектурою. У такій

системі окремі мікросервіси відповідають за автентифікацію користувачів (Auth), керування профілями користувачів (Users), роботу із закладами харчування (Restaurants), меню та стравами (Menu), персоналом закладів (Spot Personnel), оформлення й супровід замовлень (Orders), а також надсилання повідомлень (Notifications). Кожен сервіс реалізує власну бізнес-логіку, має незалежний життєвий цикл та може бути реалізований із використанням різних мов програмування і програмних платформ.

Як видно з рис. 1, окремі функціональні модулі системи використовують різні моделі зберігання даних. Сервіси авторизації, персоналу та замовлень працюють переважно зі структурованими даними, для яких критичними є транзакційна цілісність і узгодженість. Натомість сервіси меню та закладів оперують значними обсягами напівструктурованої інформації, що часто змінюється та потребує горизонтального масштабування. Додатково система використовує об'єктне сховище для медіаконтенту, брокер повідомлень для асинхронної взаємодії та кешувальні механізми для підвищення продуктивності.

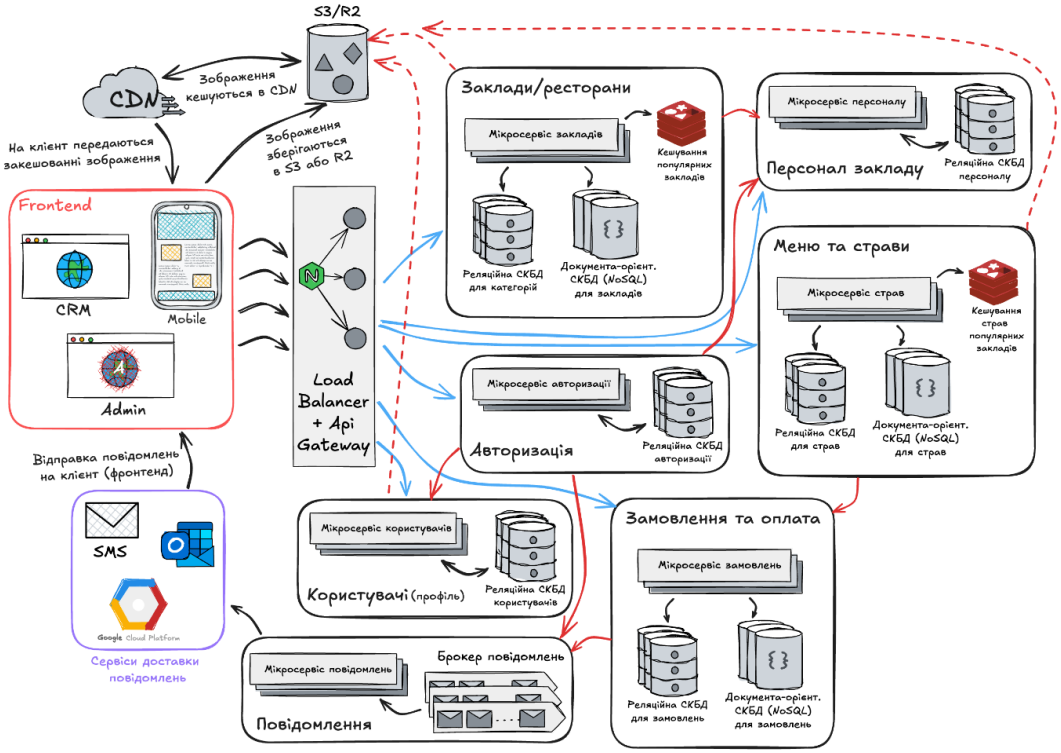

Рис. 1. Приклад загальної архітектури мобільного застосунку сфери HoReCa на основі клієнт-серверного підходу з використанням мікросервісів

Тобто вже на етапі проєктування мікросервісної системи виникає потреба в ухваленні низки архітектурних рішень щодо визначення меж сервісів, розподілу даних між ними, вибору типу сховища для кожного сервісу, забезпечення узгодженості даних, інтеграції різнорідних сховищ і підтримки масштабованості системи. Саме різнорідність функціональних вимог окремих мікросервісів, проілюстрована на рис. 1, є основною причиною виникнення комплексу проблем проєктування системи зберігання даних, які доцільно класифікувати за рівнями їх виникнення.

Водночас декомпозиція системи на мікросервіси принципово змінює традиційні підходи до організації зберігання даних. Якщо в монолітних застосунках переважно використовується єдина централізована база даних, то в мікросервісній архітектурі кожен сервіс, відповідно до принципу Database per Service, зазвичай має власне сховище даних [1, 4]. Такий підхід забезпечує автономність сервісів, можливість їх незалежного розгортання, масштабування та супроводу, проте одночасно ускладнює процес проєктування системи зберігання даних.

Різні мікросервіси працюють із даними, що суттєво відрізняються за структурою, обсягами, інтенсивністю оновлення, вимогами до транзакційності, швидкодії та масштабування [15]. Наприклад, сервіси автентифікації, керування персоналом і замовленнями оперують структурованими даними, для яких критично важливими є цілісність, надійність та підтримка ACID-властивостей. Натомість сервіси меню, каталогу закладів, рекомендацій чи повідомлень переважно працюють із напівструктурованими або слабкоструктурованими даними, що часто змінюються та потребують горизонтального

масштабування. Унаслідок цього використання єдиної моделі зберігання для всієї системи втрачає ефективність.

Саме тому під час проєктування сучасних мікросервісних систем дедалі ширше застосовується концепція Polyglot Persistence, відповідно до якої в межах однієї інформаційної системи використовуються різні типи сховищ даних, найбільш придатні для конкретного мікросервісу [5, 16]. Такий підхід дозволяє поєднати переваги різних типів баз даних, однак водночас суттєво збільшує кількість архітектурних рішень, які необхідно прийняти під час проєктування системи.

Унаслідок цього проблеми проєктування системи зберігання даних перестають обмежуватися лише вибором між SQL- та NoSQL-платформами. Значно важливішими стають питання визначення меж мікросервісів, раціонального розподілу даних між ними, забезпечення узгодженості інформації, організації міжсервісної взаємодії, вибору архітектурних патернів та підтримки масштабованості системи. При цьому зазначені проблеми не є незалежними: рішення, прийняті на ранніх етапах проєктування, безпосередньо впливають на складність реалізації та подальшої експлуатації інформаційної системи.

Аналіз сучасних наукових досліджень і практики проєктування мікросервісних систем свідчить, що проблеми організації зберігання даних доцільно розглядати не ізольовано, а відповідно до етапів їх виникнення під час проєктування архітектури. З цією метою у роботі запропоновано багаторівневу класифікацію проблем проєктування системи зберігання даних у мікросервісній архітектурі (рис. 2).

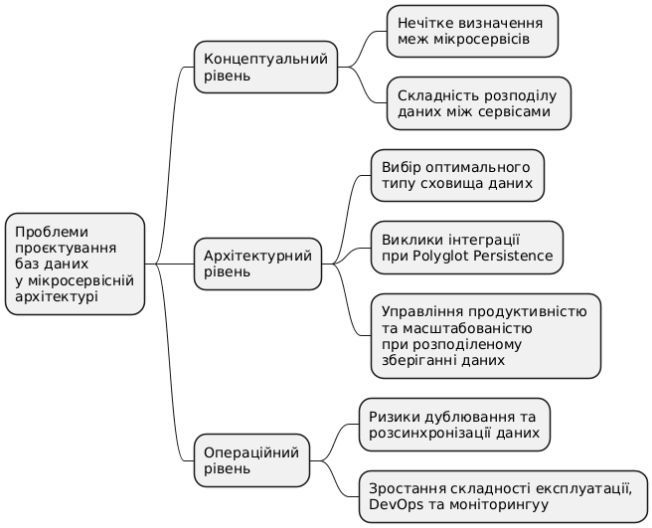

Рис. 2. Багаторівнева класифікація проблем проєктування баз даних у мікросервісній архітектурі

В основу запропонованої класифікації покладено три взаємопов'язані рівні: концептуальний, на якому визначаються межі бізнес-контекстів і принципи розподілу даних; архітектурний, що охоплює вибір моделей зберігання, технологій та механізмів інтеграції; і операційний, який характеризує проблеми, пов'язані з експлуатацією, масштабуванням, моніторингом та супроводом системи. На відміну від традиційного переліку окремих проблем, запропонована класифікація відображає їх причинно-наслідкові зв'язки та послідовність виникнення в процесі проєктування мікросервісної архітектури.

1. Проблеми концептуального рівня. Концептуальний рівень охоплює архітектурні рішення, які приймаються ще до вибору конкретних технологій реалізації інформаційної системи. Саме на цьому етапі визначаються межі бізнес-контекстів, декомпозиція предметної області, відповідальність окремих мікросервісів та принципи розподілу даних між ними. Помилки, допущені під час концептуального проєктування, мають найбільший вплив на подальшу архітектуру системи, оскільки їх усунення після реалізації програмного забезпечення потребує значних витрат часу та ресурсів.

1.1. Нечітке визначення меж мікросервісів.

Однією з фундаментальних проблем концептуального проєктування мікросервісної архітектури є визначення меж бізнес-контекстів (Bounded Context), від правильності яких залежить автономність сервісів, ефективність їх взаємодії та можливість подальшого розвитку інформаційної системи. Саме на цьому етапі ухвалюють рішення, які бізнес-функції повинні реалізовуватися окремими мікросервісами, а які належать до одного функціонального контексту.

Проблема виникає тоді, коли декомпозиція системи здійснюється не відповідно до бізнес-процесів і предметної області, а за технічними, організаційними або випадковими критеріями, наприклад, відповідно до структури бази даних, програмних модулів чи складу команди розроблення. У результаті один мікросервіс може поєднувати декілька незалежних бізнес-доменів або, навпаки, одна бізнес-сутність розподіляється між багатьма сервісами. Це призводить до надлишкових міжсервісних залежностей, дублювання функціональності, циклічних викликів і значного збільшення кількості мережевих взаємодій між сервісами [17]. Наслідком є зниження гнучкості системи, ускладнення її супроводу та обмеження можливостей незалежного масштабування окремих компонентів.

Одним із найхарактерніших проявів некоректної декомпозиції є виникнення архітектурного антипатерну Shared Database, за якого кілька незалежних мікросервісів використовують спільну базу даних [18]. У такому випадку порушується фундаментальний принцип Database per Service, відповідно до якого кожен сервіс повинен бути єдиним власником своїх даних [19]. Спільне використання бази даних формує сильну зв'язаність між сервісами, ускладнює їх незалежне розгортання, тестування, модифікацію та масштабування, а також підвищує ризик поширення помилок між окремими компонентами системи.

У системах сфери HoReCa подібна ситуація може виникнути, якщо мікросервіси Orders, Menu, Restaurants, Delivery або Loyalty використовують спільні таблиці чи одну базу даних для роботи із замовленнями, меню або бонусними програмами. У такому випадку навіть незначна зміна структури даних одного сервісу потребуватиме модифікації інших сервісів, що суттєво ускладнює розвиток усієї інформаційної системи. Аналогічні проблеми спостерігаються й в інших предметних областях. Наприклад, у банківських інформаційних системах використання спільної бази даних сервісами облікових записів, транзакцій, кредитування та звітності призводить до втрати автономності компонентів, ускладнення забезпечення узгодженості даних і зниження ефективності масштабування [20].

З'ясоване дозволяє стверджувати, що правильне визначення меж бізнес-контекстів є першочерговою умовою побудови ефективної мікросервісної архітектури. Проте навіть за умови коректної декомпозиції системи виникає наступне концептуальне завдання – визначення відповідальності кожного сервісу за власні дані та принципів їх розподілу між мікросервісами.

1.2. Проблема розподілу даних між мікросервісами.

Після визначення меж бізнес-контекстів наступним етапом концептуального проєктування є визначення відповідальності кожного мікросервісу за власні дані (Data Ownership). Відповідно до принципів мікросервісної архітектури кожен сервіс повинен бути єдиним власником своїх даних, що забезпечує його автономність, незалежність від інших компонентів системи та можливість їх самостійного розвитку [3]. Проте реальні бізнес-процеси зазвичай охоплюють кілька мікросервісів, кожен із яких володіє лише частиною інформації, необхідної для виконання спільної бізнес-операції.

Наприклад, під час оформлення замовлення в інформаційній системі HoReCa одночасно взаємодіють сервіси Orders, Menu, Payments, Delivery та Users. Кожен із них відповідає лише за власний набір даних: сервіс замовлень – за склад замовлення, сервіс меню – за інформацію про страви, сервіс оплати – за фінансові транзакції, сервіс доставки – за логістичні операції, а сервіс користувачів – за профілі клієнтів. Водночас для кінцевого користувача всі ці взаємодії становлять єдиний бізнес-процес, який має виконуватися цілісно, коректно та без помилок.

Проблема виникає тоді, коли межі відповідальності за дані визначені нечітко або порушуються під час проєктування системи. Надмірне дублювання інформації між сервісами або її невдалий розподіл призводять до порушення принципу Data Ownership, унаслідок чого суттєво ускладнюється підтримання транзакційної цілісності, синхронізації змін і забезпечення узгодженості даних у розподіленому середовищі [21]. Наприклад, після успішного підтвердження онлайн-оплати система повинна не лише зафіксувати факт виконання платежу, а й оновити статус замовлення, зарезервувати відповідні товарні позиції, скоригувати залишки інгредієнтів або складських запасів і передати інформацію сервісу доставки. Збій хоча б одного із задіяних сервісів може призвести до виникнення суперечливих станів, коли платіж уже виконано, але замовлення не підтверджено, необхідні ресурси не зарезервовано або клієнт отримує некоректну інформацію про статус свого замовлення.

Додатково проблему ускладнює використання концепції Polyglot Persistence, коли окремі мікросервіси застосовують різні типи сховищ і моделі подання даних. За таких умов міжсервісна взаємодія потребує трансформації форматів обміну, узгодження моделей даних і підтримання їх логічної цілісності, що суттєво збільшує складність інтеграції та реалізації бізнес-процесів [3].

Одним із ключових концептуальних рішень під час проєктування мікросервісної архітектури є визначення вимог до узгодженості даних для кожного бізнес-процесу. Для критичних операцій, пов'язаних із виконанням фінансових транзакцій, оформленням замовлень або керуванням товарними залишками, доцільно передбачати сильну узгодженість даних (Strong Consistency). Натомість для

допоміжних сервісів, зокрема аналітики, рекомендацій чи персоналізації, достатньою може бути модель остаточної узгодженості (Eventual Consistency), яка забезпечує вищу масштабованість системи за умови допустимої тимчасової розбіжності між копіями даних.

Фундаментальним обмеженням розподілених систем є теорема CAP, відповідно до якої неможливо одночасно забезпечити консистентність (Consistency), доступність (Availability) та стійкість до мережевих розділень (Partition Tolerance) [11]. Оскільки мікросервісна архітектура є розподіленою за своєю природою, універсального способу організації зберігання даних не існує. Уже на етапі концептуального проєктування архітектор змушений визначати компроміс між вимогами до узгодженості, доступності, продуктивності та масштабованості системи [21]. Використання сильної узгодженості гарантує коректність виконання бізнес-процесів, однак негативно впливає на продуктивність і доступність системи. Натомість модель остаточної узгодженості підвищує масштабованість, але потребує впровадження додаткових механізмів синхронізації, компенсації та контролю можливих розбіжностей між сервісами.

Отже, проблема Data Ownership є одним із ключових викликів концептуального рівня проєктування мікросервісної архітектури. Саме на цьому етапі визначаються принципи розподілу даних між сервісами, від яких безпосередньо залежать їх автономність, узгодженість, масштабованість і стійкість до змін. Подальша реалізація цих принципів потребує застосування відповідних архітектурних механізмів інтеграції, синхронізації та підтримання узгодженості даних, які розглядаються на наступному рівні класифікації.

Рекомендації для вирішення проблем концептуального рівня. Проблеми концептуального рівня виникають ще на етапі аналізу предметної області та проєктування архітектури інформаційної системи. Саме на цьому рівні визначаються межі бізнес-контекстів, відповідальність окремих мікросервісів і принципи розподілу даних між ними. Оскільки помилки концептуального проєктування практично неможливо повністю компенсувати подальшими архітектурними рішеннями, доцільно дотримуватися комплексу рекомендацій, спрямованих на забезпечення автономності сервісів, правильного розподілу відповідальності за дані та мінімізації міжсервісних залежностей.

1) Чітко визначати межі бізнес-контекстів. Декомпозицію системи доцільно виконувати відповідно до бізнес-процесів і принципів Domain-Driven Design (DDD) [22]. Кожен мікросервіс повинен реалізовувати одну завершену бізнес-функцію та не містити логіки інших доменів. Це дозволяє уникнути дублювання функціональності, циклічних залежностей і виникнення архітектурного антипатерну Shared Database.

2) Призначати єдиного власника даних (Source of Truth). Для кожної бізнес-сутності (замовлення, клієнт, меню, платіж, доставка тощо) необхідно визначити один сервіс-власник, який відповідає за створення, зміну та зберігання відповідних даних. Інші сервіси можуть використовувати лише локальні копії або підмножини цих даних, синхронізовані через API чи події. Такий підхід реалізує принцип Database per Service та мінімізує ризик порушення узгодженості даних.

3) Визначати модель узгодженості відповідно до вимог бізнес-процесів. Ще на етапі концептуального проєктування для кожного бізнес-процесу необхідно визначити вимоги до узгодженості даних. Для критично важливих операцій (оформлення замовлень, платежі, облік товарних залишків) доцільно передбачати сильну узгодженість даних. Для допоміжних сервісів (аналітика, рекомендації, персоналізація) може використовуватися модель Eventual Consistency, яка забезпечує вищу масштабованість за умови допустимої тимчасової розбіжності між копіями даних.

4) Мінімізувати дублювання даних між сервісами. Кожен мікросервіс повинен зберігати лише ті дані, які необхідні для виконання його функцій. Якщо сервісу потрібна інформація з іншого бізнес-контексту, доцільно використовувати лише мінімально необхідну локальну копію даних або її представлення, що підтримується в актуальному стані за допомогою подій. Це зменшує кількість залежностей між сервісами, спрощує супровід системи та мінімізує ризик виникнення суперечностей між даними.

5) Передбачати подієво-орієнтовану взаємодію між сервісами. Для синхронізації даних між мікросервісами доцільно використовувати подієво-орієнтований підхід, за якого сервіси обмінюються повідомленнями про зміни власного стану, а не звертаються безпосередньо до баз даних один одного. Такий підхід забезпечує слабке зв’язування сервісів, підвищує гнучкість архітектури та спрощує її подальший розвиток.

6) Планувати механізми координації багатосервісних бізнес-процесів. Якщо бізнес-процес охоплює декілька мікросервісів, ще під час його проєктування необхідно передбачити механізми координації розподілених операцій. Для цього можуть застосовуватися архітектурні патерни Saga, Compensating Transaction, Scheduler Agent Supervisor та інші, що забезпечують узгоджене виконання бізнес-процесів без використання глобальних транзакцій.

7) Періодично переглядати декомпозицію системи. Якщо між окремими сервісами постійно виникає велика кількість міжсервісних викликів, дублювання даних або взаємних залежностей, це може свідчити про некоректно визначені межі бізнес-контекстів. У таких випадках доцільно переглянути початкову декомпозицію системи та, за потреби, виконати рефакторинг архітектури.

Реалізація наведених рекомендацій дає змогу мінімізувати ризик виникнення архітектурних антипатернів ще на концептуальному етапі проєктування, забезпечити чіткий розподіл відповідальності між мікросервісами та створити передумови для побудови масштабованої, автономної й узгодженої системи зберігання даних.

2. Проблеми архітектурного рівня. На архітектурному рівні основні виклики пов'язані з технічними рішеннями щодо зберігання та інтеграції даних. Вибір оптимального типу сховища визначає продуктивність, масштабованість і стійкість системи, тоді як використання різних технологій зберігання (Polyglot Persistence) створює складнощі узгодженості, інтеграції та підтримки транзакцій між сервісами.

2.1. Проблема вибору оптимального типу сховища даних.

Після визначення меж мікросервісів і розподілу відповідальності за дані наступним архітектурним завданням є вибір технології їх зберігання. У мікросервісній архітектурі кожен сервіс може використовувати власну СКБД, тому архітектор має визначити, яка модель зберігання найкраще відповідає функціональним вимогам конкретного сервісу. Саме на цьому етапі реалізується принцип Polyglot Persistence, який допускає використання різних типів сховищ у межах однієї інформаційної системи [5].

Основна складність полягає в тому, що жоден тип баз даних не є універсальним. Реляційні СКБД забезпечують високу узгодженість даних, підтримку ACID-транзакцій та ефективну роботу зі складними зв'язками між сутностями, однак менш придатні для зберігання напівструктурованих даних і горизонтального масштабування. Документоорієнтовані бази даних забезпечують гнучку схему даних, високу швидкодію та просте масштабування, проте часто поступаються реляційним системам щодо підтримки складних транзакцій і забезпечення цілісності даних [1]. Інші типи сховищ, зокрема графові, колонкові чи часові бази даних, також оптимізовані лише для окремих класів задач [9].

Унаслідок цього вибір невідповідної технології зберігання може суттєво погіршити характеристики програмної системи. Використання реляційної бази даних для сервісу, який працює переважно з напівструктурованими документами, призводить до частих змін схеми, збільшення кількості допоміжних таблиць і погіршення продуктивності. Натомість застосування документоорієнтованої бази даних у сервісах, що реалізують фінансові операції або інші критичні бізнес-процеси, може ускладнити забезпечення транзакційної цілісності та узгодженості даних [4].

Особливо помітною ця проблема є в інформаційних системах сфери HoReCa, де окремі мікросервіси суттєво відрізняються за характером навантаження та структурою даних. Наприклад, сервіси Orders, Payments або Users працюють із критично важливими транзакційними даними й потребують суворого забезпечення цілісності, тому для них доцільним є використання реляційних СКБД. Водночас сервіси Menu або Restaurants зберігають велику кількість напівструктурованої інформації, яка часто змінюється (описи страв, модифікатори, категорії, варіанти комплектації, мультимедійний контент), що робить документоорієнтовані бази даних більш ефективним рішенням. Для сервісів доставки додатковою вимогою є підтримка геопросторових запитів, які доцільно реалізовувати із застосуванням спеціалізованих геопросторових розширень або відповідних моделей зберігання.

Загалом проблема вибору типу сховища полягає не лише у виборі між реляційними та документоорієнтованими базами даних, а насамперед у встановленні відповідності між функціональними характеристиками мікросервісу та властивостями конкретної технології зберігання. Саме від правильності цього рішення залежать продуктивність, масштабованість, ефективність використання ресурсів і можливість подальшого розвитку всієї програмної системи.

2.2. Проблема інтеграції різнорідних сховищ даних (Polyglot Persistence).

Логічним продовженням вибору оптимального типу сховища для окремих мікросервісів є використання концепції Polyglot Persistence, відповідно до якої різні сервіси можуть застосовувати різні технології зберігання даних залежно від власних функціональних потреб [5, 16]. Такий підхід дозволяє максимально ефективно використовувати переваги кожного типу СКБД, однак водночас суттєво ускладнює архітектуру інформаційної системи.

Основна проблема полягає в тому, що різні сховища використовують відмінні моделі даних, механізми забезпечення узгодженості, засоби індексування, способи виконання запитів і формати зберігання інформації. Унаслідок цього міжсервісна взаємодія вже не обмежується лише передаванням даних, а потребує їх трансформації, узгодження моделей подання інформації, синхронізації змін та підтримання логічної цілісності між різнорідними сховищами [17]. Зі збільшенням кількості

використовуваних технологій складність інтеграції зростає майже пропорційно кількості взаємозв'язків між сервісами.

Особливо гостро ця проблема проявляється під час реалізації бізнес-процесів, які охоплюють кілька мікросервісів. У таких випадках окремі етапи однієї бізнес-операції можуть виконуватися над різними типами баз даних, кожна з яких має власні механізми керування транзакціями та забезпечення цілісності. Унаслідок цього ускладнюється підтримання узгодженості даних, підвищується ризик виникнення суперечливих станів системи та зростає складність супроводу програмного забезпечення.

Для інформаційних систем сфери HoReCa подібна ситуація є досить типовою. Наприклад, сервіс Orders може використовувати реляційну базу даних для забезпечення транзакційної цілісності замовлень, сервіс Menu – документоорієнтовану базу даних для гнучкого зберігання структури меню, а сервіс Loyalty – графову базу для аналізу взаємозв'язків між клієнтами, закладами та програмами лояльності. У разі скасування замовлення або зміни його статусу виникає необхідність синхронізувати зміни між усіма зазначеними сервісами, попри використання різних моделей зберігання даних.

Подібні труднощі характерні й для інших предметних областей. Наприклад, у банківських інформаційних системах транзакційні сервіси часто використовують реляційні СКБД, тоді як аналітичні модулі працюють із колонковими сховищами даних. Це ускладнює побудову актуальної звітності, проведення аудиту та підтримання синхронності інформації між оперативними й аналітичними системами. Аналогічні проблеми виникають і в системах електронної комерції, де сервіси рекомендацій, каталогу товарів та платіжні модулі можуть використовувати різні технології зберігання даних.

Крім інтеграційних труднощів, використання Polyglot Persistence призводить до ускладнення експлуатації інформаційної системи. Зростає кількість технологій, які необхідно адмініструвати, збільшуються вимоги до кваліфікації команди розробників і DevOps-фахівців, ускладнюються резервне копіювання, моніторинг, оновлення та підтримка безпеки різнорідної інфраструктури. Це підвищує загальну вартість володіння системою та збільшує ризик виникнення помилок під час її супроводу.

Отже, застосування концепції Polyglot Persistence забезпечує високу гнучкість під час вибору технологій зберігання даних, однак водночас породжує новий клас архітектурних проблем, пов'язаних з інтеграцією різнорідних сховищ, підтриманням узгодженості даних і супроводом багатотехнологічної інфраструктури. Саме тому ефективне використання різних типів баз даних потребує спеціалізованих архітектурних рішень, які розглянуто в рекомендаціях цього рівня.

2.3. Проблема забезпечення продуктивності та масштабованості розподіленого зберігання даних.

Після вибору оптимальних типів сховищ і організації їх взаємодії виникає наступний архітектурний виклик – забезпечення продуктивності та масштабованості всієї системи. Навіть якщо для кожного мікросервісу використано найбільш придатну технологію зберігання, загальна ефективність інформаційної системи визначається не лише швидкодією окремих баз даних, а й особливостями їх спільної роботи у розподіленому середовищі.

Основна проблема полягає в тому, що зі збільшенням кількості мікросервісів і обсягів даних різко зростає кількість міжсервісних запитів, операцій синхронізації та мережевих взаємодій. Крім безпосереднього виконання запитів до баз даних, система повинна підтримувати реплікацію даних, балансування навантаження, горизонтальне масштабування, кешування, шардінг і забезпечувати відмовостійкість окремих компонентів. У результаті вузьким місцем може стати не окрема база даних, а вся інфраструктура розподіленого зберігання.

Особливо гостро ця проблема проявляється під час пікових навантажень. У системах сфери HoReCa найбільша кількість звернень до сервісів припадає на години обіду та вечірній час, коли одночасно виконуються оформлення замовлень, резервування столиків, оновлення меню, надсилання повідомлень і взаємодія зі службами доставки. Якщо для сервісу Orders не реалізовано ефективного горизонтального масштабування або шардінгу бази даних, він швидко стає вузьким місцем усієї системи, що призводить до затримок під час підтвердження замовлень і погіршення користувацького досвіду.

Подібні проблеми характерні й для інших галузей. У банківських інформаційних системах пікові навантаження під час масових фінансових операцій можуть призводити до перевантаження транзакційних баз даних, якщо архітектура не підтримує ефективного масштабування. У системах електронної комерції затримки реплікації між вузлами можуть спричиняти тимчасову неузгодженість інформації про залишки товарів, що негативно впливає на коректність оформлення замовлень і довіру користувачів до платформи.

Додаткову складність створює необхідність одночасного забезпечення високої продуктивності, відмовостійкості та економного використання ресурсів. Надмірне дублювання даних, неефективний розподіл навантаження або невдало налаштовані механізми реплікації здатні суттєво збільшити час відповіді сервісів, обсяг мережевого трафіку та витрати на підтримку інфраструктури. Тому проблема

масштабованості виходить далеко за межі окремої бази даних і стосується всієї архітектури розподіленої інформаційної системи.

Отже, забезпечення продуктивності та масштабованості є завершальним викликом архітектурного рівня, який поєднує результати попередніх проєктних рішень щодо вибору типів сховищ і організації їх взаємодії. Саме від ефективності цих рішень залежить здатність інформаційної системи підтримувати стабільну роботу під час зростання навантаження та подальшого розвитку програмного продукту.

Рекомендації для вирішення проблем архітектурного рівня. Проблеми архітектурного рівня пов'язані з технічними рішеннями щодо організації зберігання, інтеграції та оброблення даних після визначення меж мікросервісів і розподілу відповідальності між ними. На цьому етапі формується технологічна архітектура системи, від якої залежать її продуктивність, масштабованість, відмовостійкість та можливість подальшого розвитку. Для мінімізації відповідних ризиків доцільно дотримуватися таких рекомендацій.

1) *Вибирати тип сховища відповідно до функціональних вимог сервісу*. Вибір технології зберігання має визначатися характером даних і вимогами бізнес-процесу, а не прагненням використовувати єдину технологію для всієї системи. Для транзакційних сервісів доцільно застосовувати реляційні бази даних із підтримкою ACID, для роботи з напівструктурованими даними – документоорієнтовані сховища, для аналітичної обробки – колонкові бази даних, а для задач, пов'язаних із складними зв'язками між сутностями або маршрутами, – графові чи геопросторові бази даних.

2) *Використовувати Polyglot Persistence лише за наявності обґрунтованої потреби.* Застосування кількох моделей даних має бути виправданим особливостями окремих сервісів. Кожне додаткове сховище збільшує складність інтеграції, адміністрування та супроводу системи, тому різнорідні технології доцільно впроваджувати лише тоді, коли їх використання забезпечує відчутні переваги щодо продуктивності або функціональних можливостей.

3) *Передбачати механізми інтеграції різнорідних сховищ даних*. У системах із Polyglot Persistence слід заздалегідь визначати способи синхронізації даних між сервісами. Для цього доцільно використовувати асинхронний обмін повідомленнями, подієву взаємодію, CDC (Change Data Capture), ETL/ELT-процеси або інші інтеграційні механізми, які забезпечують актуальність інформації без створення жорстких залежностей між сервісами.

4) *Використовувати архітектурні патерни для підтримання узгодженості між різними сховищами.* Оскільки класичні ACID-транзакції працюють лише в межах однієї бази даних, для координації багатосервісних операцій доцільно застосовувати патерни Saga, Compensating Transaction, Event Sourcing або Transactional Outbox. Це дозволяє підтримувати логічну узгодженість даних без використання розподілених транзакцій.

5) *Проєктувати архітектуру з урахуванням горизонтального масштабування*. Ще на етапі проєктування варто передбачати можливість шардінгу, реплікації, балансування навантаження та автоматичного масштабування сервісів. Це дозволяє уникнути появи вузьких місць при зростанні кількості користувачів або інтенсивності оброблення запитів.

6) *Використовувати механізми оптимізації продуктивності.* Для зменшення навантаження на бази даних доцільно застосовувати кешування (Redis, Memcached), асинхронне виконання тривалих операцій, черги повідомлень, оптимізацію запитів та розподіл навантаження між репліками читання. Комплексне використання таких механізмів дозволяє підвищити швидкодію системи без зміни її бізнес-логіки.

7) *Регулярно проводити тестування продуктивності та масштабованості.* Архітектурні рішення необхідно перевіряти під час навантажувального, стресового та масштабного тестування. Такі випробування дозволяють своєчасно виявляти вузькі місця, оцінювати ефективність обраних моделей зберігання даних і перевіряти стійкість системи до пікових навантажень.

8) *Забезпечувати моніторинг стану розподіленої інфраструктури.* Після впровадження архітектурних рішень треба постійно контролювати продуктивність баз даних, реплікацію, затримки синхронізації, використання ресурсів і швидкодію сервісів за допомогою систем моніторингу (Prometheus, Grafana, OpenTelemetry тощо). Це дозволяє оперативно виявляти деградацію продуктивності та своєчасно оптимізувати архітектуру.

Отже, рекомендації архітектурного рівня спрямовані на обґрунтований вибір моделей зберігання даних, ефективну інтеграцію різнорідних сховищ, підтримання узгодженості між сервісами та забезпечення необхідного рівня продуктивності й масштабованості. Їх дотримання дозволяє мінімізувати технологічні ризики, що виникають під час реалізації мікросервісної архітектури, та створює основу для її стабільної експлуатації.

3. Проблеми операційного рівня. Проблеми операційного рівня виникають уже під час експлуатації мікросервісної системи після завершення її проєктування та реалізації. Якщо

концептуальний рівень визначає структуру системи, а архітектурний – способи організації та інтеграції даних, то операційний рівень охоплює забезпечення стабільної роботи системи у виробничому середовищі. Основні виклики пов'язані з підтриманням узгодженості даних між сервісами, автоматизацією процесів розгортання, централізованим моніторингом, тестуванням продуктивності та забезпеченням безперервної доступності сервісів. Саме на цьому рівні оцінюється практична ефективність архітектурних рішень, прийнятих на попередніх етапах.

3.1. Ризики дублювання та розсинхронізації даних.

Після введення системи в експлуатацію одним із найпоширеніших викликів стає підтримання актуальності даних, що дублюються між мікросервісами. Попри принцип Database per Service, окремі сервіси нерідко зберігають локальні копії або проєкції даних інших сервісів для підвищення продуктивності, автономності роботи чи зменшення кількості міжсервісних викликів. Такий підхід підвищує ефективність роботи системи, проте водночас створює ризик виникнення розбіжностей між первинними даними та їх копіями.

Основною причиною розсинхронізації є асинхронний характер взаємодії між мікросервісами. У разі затримки доставки повідомлень, втрати подій, помилок брокера повідомлень або тимчасової недоступності окремих сервісів локальні копії можуть певний час не відповідати актуальному стану даних. Унаслідок цього різні сервіси формують різне уявлення про одну й ту саму бізнес-сутність, що негативно впливає на коректність виконання бізнес-процесів.

Особливо гостро ця проблема проявляється в системах, що активно еволюціонують. Із появою нових сервісів, зміною бізнес-процесів, модифікацією схем даних або міграцією інформації між компонентами кількість локальних копій даних поступово збільшується, а підтримання їх узгодженості стає дедалі складнішим. Якщо механізми синхронізації реалізовано неналежним чином, ризик виникнення розсинхронізації, втрати актуальності даних або появи суперечливих копій суттєво зростає.

Подібні ситуації виникають у різних прикладних областях. У інформаційній системі HoReCa сервіс Orders виступає джерелом істини для замовлень, тоді як сервіс Delivery використовує локальну копію інформації для планування доставки. Якщо повідомлення про скасування або зміну замовлення надходить із затримкою, служба доставки може розпочати виконання вже неактуального замовлення. Аналогічно у банківських інформаційних системах несвоєчасна синхронізація даних між сервісами здатна призвести до відображення некоректних залишків коштів, а в системах електронної комерції – до продажу товарів, яких фактично вже немає на складі.

Тим самим проблема дублювання та розсинхронізації даних є характерною саме для операційного рівня, оскільки проявляється під час реальної експлуатації розподіленої системи. Її вирішення потребує впровадження механізмів контролю доставки повідомлень, моніторингу синхронізації та відновлення узгодженості даних між сервісами.

3.2. Проблеми експлуатації, DevOps та моніторингу.

У міру розвитку мікросервісної архітектури та збільшення кількості сервісів суттєво ускладнюється їх експлуатація. Якщо в монолітних системах розгортання, оновлення та контроль працездатності здійснюються централізовано, то в мікросервісному середовищі кожен сервіс має власний життєвий цикл, конфігурацію, процес розгортання та механізми масштабування. Унаслідок цього забезпечення стабільної роботи всієї системи потребує використання спеціалізованих DevOps-практик, автоматизації процесів розгортання та централізованого моніторингу.

Основні труднощі пов'язані з підтриманням узгодженості конфігурацій між сервісами, керуванням контейнеризованими середовищами, автоматичним масштабуванням, безперервним розгортанням нових версій і своєчасним виявленням відмов окремих компонентів. У розподіленій системі навіть незначний збій одного мікросервісу здатний спричинити каскадні відмови інших сервісів, що суттєво ускладнює локалізацію причин несправностей і відновлення працездатності системи.

Додатковою особливістю операційного рівня є те, що складність експлуатації мікросервісної архітектури зростає разом із розвитком системи. Кожен новий сервіс збільшує кількість конфігурацій, сценаріїв розгортання, точок моніторингу та взаємозв'язків між компонентами. Тому підтримання працездатності системи поступово перетворюється на окреме архітектурне завдання, яке потребує високого рівня автоматизації, централізованого моніторингу та стандартизації процесів DevOps [23].

Подібні проблеми характерні для різних прикладних областей. У HoReCa-платформах під час пікових навантажень у години найбільшої кількості замовлень виникає необхідність автоматичного масштабування сервісу Orders, інакше затримки його роботи можуть негативно вплинути на функціонування інших сервісів системи. У банківських інформаційних системах особливо важливо забезпечити неперервність виконання фінансових операцій навіть під час оновлення окремих мікросервісів, що потребує використання безперервного розгортання (CI/CD) та стратегій оновлення без простоїв. У системах електронної комерції відсутність централізованого моніторингу може призвести до

того, що відмова сервісу Inventory залишиться непоміченою, унаслідок чого користувачі отримуватимуть недостовірну інформацію про наявність товарів.

Отже, забезпечення ефективної експлуатації мікросервісної архітектури є одним із ключових завдань операційного рівня. Його вирішення потребує впровадження засобів автоматизації DevOps-процесів, централізованого моніторингу, журналювання, трасування запитів і безперервного контролю стану розподіленої інфраструктури.

Рекомендації для вирішення проблем операційного рівня.

Проблеми операційного рівня виникають під час експлуатації мікросервісної системи у виробничому середовищі. На цьому етапі головними завданнями стають підтримання узгодженості даних між сервісами, забезпечення безперервної роботи системи, автоматизація процесів розгортання та своєчасне виявлення відмов. Для мінімізації відповідних ризиків доцільно дотримуватися таких рекомендацій.

1) Забезпечувати надійну синхронізацію даних між сервісами. Для підтримання актуальності локальних копій даних доцільно використовувати механізми Change Data Capture (CDC), подієву синхронізацію або інші засоби автоматичного поширення змін між сервісами [24]. Це дозволяє мінімізувати ризик виникнення розбіжностей між джерелом істини та його локальними представленнями.

2) Використовувати централізований моніторинг і журналювання. Експлуатація розподіленої системи потребує постійного контролю її стану. Для цього доцільно впроваджувати єдину систему збору метрик, логів і трасування запитів із використанням інструментів Prometheus, Grafana, ELK Stack, OpenTelemetry або аналогічних платформ. Це забезпечує оперативне виявлення відмов, аналіз причин інцидентів та своєчасне реагування на деградацію продуктивності.

3) Автоматизувати процеси розгортання та супроводу системи. Неперервна інтеграція й безперервне розгортання (CI/CD) дозволяють автоматизувати тестування, складання та впровадження нових версій сервісів, зменшуючи кількість помилок, пов’язаних із ручними операціями. Для керування контейнеризованими застосунками доцільно використовувати платформи оркестрації, зокрема Kubernetes, які забезпечують автоматичне масштабування, відновлення після відмов і централізоване керування конфігураціями [25].

4) Використовувати безпечні стратегії оновлення сервісів. Під час впровадження нових версій мікросервісів рекомендується застосовувати стратегії Blue-Green Deployment, Canary Release або Rolling Update [26]. Такі підходи дають змогу мінімізувати ризик відмов, поступово вводити зміни в експлуатацію та швидко повернутися до стабільної версії у разі виникнення помилок.

5) Регулярно виконувати навантажувальне та відмовостійке тестування. Працездатність системи необхідно перевіряти не лише за штатних умов, а й під час моделювання пікових навантажень, відмов окремих сервісів і мережевих збоїв [27]. Це дозволяє своєчасно виявляти вузькі місця, оцінювати ефективність механізмів масштабування та перевіряти готовність системи до експлуатації в реальних умовах [28].

6) Планувати еволюцію архітектури та керувати змінами даних. У процесі розвитку інформаційної системи необхідно забезпечувати контрольоване версіонування схем даних, керовану міграцію інформації між сервісами та підтримання сумісності між різними версіями API й повідомлень. Такий підхід дозволяє поступово розвивати систему без порушення її працездатності та без втрати узгодженості даних.

Надані рекомендації операційного рівня спрямовані на забезпечення стабільної експлуатації мікросервісної архітектури, підтримання узгодженості даних, автоматизацію процесів розгортання, оперативне виявлення відмов і керовану еволюцію програмної системи. Їх застосування дозволяє підтримувати високу доступність сервісів, зменшувати ризики виникнення критичних інцидентів і забезпечувати безперервний розвиток інформаційної системи.

Еволюція мікросервісної архітектури неминуче супроводжується зміною бізнес-процесів, появою нових сервісів, модифікацією схем даних та міграцією інформації між компонентами. У процесі розвитку системи ускладнюються питання розподілу відповідальності за дані, забезпечення їх узгодженості, підтримки міжсервісної взаємодії та продуктивності. Саме тому ефективне проєктування системи зберігання даних не може обмежуватися лише вибором технологій або окремих патернів. Воно потребує комплексного підходу, який охоплює концептуальний, архітектурний та операційний рівні, забезпечуючи можливість безпечної еволюції системи без втрати її масштабованості, надійності та керованості.

Архітектурні патерни вирішення. В аналізі проблем проєктування системи зберігання даних у мікросервісній архітектурі значну роль відіграють архітектурні патерни, які забезпечують практичну реалізацію сформульованих рекомендацій. На відміну від окремих технологій або програмних засобів,

архітектурні патерни описують перевірені підходи до організації взаємодії між сервісами, керування даними та забезпечення їх узгодженості. Саме вони дозволяють комплексно вирішувати проблеми, які виникають на концептуальному, архітектурному та операційному рівнях. У попередніх розділах окремі архітектурні патерни розглядалися як рекомендації щодо вирішення конкретних проблем різних рівнів. Однак кожен із них має ширшу сферу застосування й не обмежується окремим архітектурним рішенням. Тому доцільно узагальнити їх як єдину систему перевірених архітектурних практик, що забезпечують комплексне вирішення проблем проєктування системи зберігання даних у мікросервісній архітектурі.

Для забезпечення узгодженості розподілених бізнес-процесів широко застосовується патерн Saga, який координує виконання довготривалих операцій через послідовність локальних транзакцій і компенсувальних дій, усуваючи потребу у глобальних транзакціях.

Для розділення навантаження між операціями запису та читання використовується CQRS **(**Command Query Responsibility Segregation), що дозволяє незалежно оптимізувати моделі запису і читання, підвищуючи продуктивність та масштабованість системи.

Патерн Event Sourcing забезпечує зберігання історії змін у вигляді послідовності подій, що дає змогу відновлювати будь-який попередній стан системи, реалізовувати аудит та аналізувати еволюцію бізнес-процесів.

Для гарантування надійної передачі повідомлень між сервісами застосовуються патерни Transactional Outbox та Transactional Inbox, які синхронізують операції запису даних із публікацією подій та мінімізують ризик втрати повідомлень.

У випадках, коли необхідно сформувати відповідь на основі інформації з кількох мікросервісів, доцільно використовувати патерн API Composition, який забезпечує агрегування даних без порушення принципу автономності сервісів.

Одним із фундаментальних принципів мікросервісної архітектури залишається патерн Database per Service, відповідно до якого кожен сервіс є єдиним власником власної бази даних. Саме він забезпечує незалежність сервісів, мінімізує зв’язаність між ними та запобігає виникненню антипатерну Shared Database.

Основою подієво-орієнтованої взаємодії між сервісами є Domain Events, які дозволяють повідомляти інші компоненти системи про зміни бізнес-стану без прямого доступу до їхніх даних. Це забезпечує слабке зв’язування сервісів і підтримує їх незалежний розвиток.

Наведені архітектурні патерни є практичним інструментарієм реалізації рекомендацій, сформульованих для концептуального, архітектурного та операційного рівнів. Їх комплексне застосування забезпечує узгодженість даних, автономність мікросервісів, масштабованість системи та її здатність до подальшої еволюції без суттєвого ускладнення архітектури.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

У статті досліджено проблеми проєктування та експлуатації систем зберігання даних у мікросервісній архітектурі в умовах використання концепції Polyglot Persistence. Проведений аналіз сучасних наукових публікацій і практичних підходів показав, що труднощі виникають не лише під час вибору технологій зберігання даних, а й на всіх етапах життєвого циклу інформаційної системи: від концептуального проєктування до експлуатації у виробничому середовищі.

Основним результатом роботи є запропонована трирівнева класифікація проблем систем зберігання даних у мікросервісній архітектурі, яка охоплює концептуальний, архітектурний та операційний рівні. На концептуальному рівні визначено проблеми формування меж бізнес-контекстів і розподілу відповідальності за дані між сервісами. На архітектурному рівні систематизовано виклики, пов’язані з вибором типів сховищ даних, використанням Polyglot Persistence та забезпеченням продуктивності й масштабованості розподілених систем. На операційному рівні розглянуто проблеми підтримання узгодженості копій даних, автоматизації DevOps-процесів, моніторингу та забезпечення стабільності функціонування мікросервісної інфраструктури.

Для кожної групи проблем сформовано практичні рекомендації, спрямовані на запобігання виникненню архітектурних антипатернів, забезпечення автономності сервісів, узгодженості даних, масштабованості та надійності інформаційних систем. Окремо узагальнено сучасні архітектурні патерни (Database per Service, Saga, CQRS, Event Sourcing, Transactional Outbox, Transactional Inbox, Domain Events, API Composition), які можуть використовуватися як практичні інструменти реалізації запропонованих рекомендацій.

Запропонований підхід дозволяє розглядати проблеми систем зберігання даних не ізольовано, а як взаємопов’язану систему рішень, що охоплює всі етапи проєктування, побудови та експлуатації

мікросервісної архітектури. Це створює методичну основу для прийняття обґрунтованих архітектурних рішень під час розроблення сучасних інформаційних систем.

Перспективи подальших досліджень полягають у розробленні формалізованої методики підтримки прийняття рішень щодо вибору типів сховищ даних для окремих мікросервісів, побудові моделей оцінювання ефективності застосування Polyglot Persistence, а також створенні програмних засобів автоматизованого аналізу архітектури мікросервісних систем для виявлення потенційних проблем розподілу даних, узгодженості та міжсервісної взаємодії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Soni, M., & Jyotinagar, V. (2023). SQL vs NoSQL databases for the microservices: A comparative survey. In *Proceedings of the 2nd International Conference on Edge Computing and Applications* (pp. 17-22). IEEE. https://doi.org/10.1109/icecaa58104.2023.10212190
2. Kavuluri, H. V., & Avula, S. B. (2024). Database engineering for microservices: PostgreSQL as the foundation. *International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5*(4). https://doi.org/10.63282/3050-9262.ijaidsml-v5i4p103
3. Halili, F., Nuhiji, A., & Mustafai Veliu, D. (2025). *Polyglot persistence in microservices: Managing data diversity in distributed systems*. arXiv. https://doi.org/10.48550/arXiv.2509.08014
4. André, M., Raglianti, M., Serbout, S., Cleve, A., & Lanza, M. (2025). *An empirical study on database usage in microservices*. arXiv. https://doi.org/10.48550/arXiv.2510.20582
5. Fowler, M. (n.d.). *Polyglot persistence*. https://martinfowler.com/bliki/PolyglotPersistence.html
6. Ali, A. M. (2024). Exploring database design patterns of microservices. *Journal of Artificial Intelligence, Machine Learning and Data Science*. https://doi.org/10.51219/jaimld/azra-jabeen-mohamed-ali/376
7. Ataei, P., & Staegemann, D. (2023). Application of microservices patterns to big data systems. *Journal of Big Data, 10*, 1-49. https://doi.org/10.1186/s40537-023-00733-4
8. Bandaru, S. (2021). Polyglot persistence in web engineering: Adaptive database selection across PostgreSQL, Oracle, MySQL, and MongoDB. *International Journal of Information and Electronics Engineering*. https://doi.org/10.48047/ijiee.2021.11.4.11
9. Kazanavičius, J., Mažeika, D., & Kalibatienė, D. (2022). An approach to migrate a monolith database into multi-model polyglot persistence based on microservice architecture: A case study for mainframe database. *Applied Sciences, 12*, 6189. https://doi.org/10.3390/app12126189
10. Laigner, R., Zhou, Y., Salles, M., Liu, Y., & Kalinowski, M. (2021). *Data management in microservices: State of the practice, challenges, and research directions*. arXiv. https://doi.org/10.48550/arXiv.2103.00170
11. Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News, 33*(2), 51-59. https://doi.org/10.1145/564585.564601
12. Ragothaman, K. (2025). Modernizing transactional systems: Deep dive from monolithic to microservices architectures. *CLEI Electronic Journal, 28*. https://doi.org/10.19153/cleiej.28.3.11
13. *Online food delivery market (2025-2030).* (2025). Grand View Research. https://www.grandviewresearch.com/industry-analysis/online-food-delivery-market-report
14. Lere, A., Pati, G., & Ege, E. (2023). Web-based restaurant food ordering application information system using agile development method. *Journal of Technology and Health, 1*, 104–110. https://doi.org/10.61677/jth.vi.93
15. Trofymenko, O. H., Manakov, S. Yu., Chykunov, P. O., Loboda, Yu. V., & Hurin, Ye. P. (2025). Routing in microfrontend applications: Architectural approaches, challenges, and solutions. *Cybersecurity: Education, Science, Technique, 3*(31), 635-651. https://doi.org/10.28925/2663-4023.2025.31.1057
16. Kaur, K., Bharany, S., Badotra, S., Aggarwal, K., Nayyar, A., & Sharma, S. (2022). Energy-efficient polyglot persistence database live migration among heterogeneous clouds. *The Journal of Supercomputing, 79*, 1-30. https://doi.org/10.1007/s11227-022-04662-6
17. Genfer, P., & Zdun, U. (2026). Using guided community detection to improve existing microservice designs. In *Service-oriented computing (ICSOC 2025)* (Lecture Notes in Computer Science, Vol. 16320). Springer. https://doi.org/10.1007/978-981-95-5012-8_15
18. Braz, L., França, B., & Cafeo, B. (2025). Dynamic analysis for detecting microservices antipatterns. In *Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software* (pp. 67-78). https://doi.org/10.5753/sbcars.2025.14576
19. Newman, S. (2021). *Building microservices* (2nd ed.). O'Reilly Media.

20. Randa, A. A., & Adi, A. M. (2026). Analyzing the evolution of microservice architecture: Challenges, solutions, and proposed design. *AIP Conference Proceedings, 3406*(1), Article 020023. https://doi.org/10.1063/5.0318956
21. Cho'ponov, O. (2026). Comparative analysis of microservices and monolithic architectures: Evaluating performance, economic sustainability, and human factors in the 2025–2026 ecosystem. *Advances in Science and Sustainability, 2*, 9-12. https://doi.org/10.70728/susta.v02.i05.002
22. Zhang, Y. (2025). *Exploration of enterprise big data microservice architecture based on domain-driven design (DDD)*. arXiv. https://doi.org/10.48550/arXiv.2511.05880
23. Tangudu, R. (2026). A unified architectural framework for microservices integrating DevOps, CI/CD, and runtime orchestration. *World Journal of Advanced Engineering Technology and Sciences, 18*, 98-107. https://doi.org/10.30574/wjaets.2026.18.3.0128
24. Hao, L., Jiang, T., Lin, Y., & Lu, Y. (2023). Methods for solving the change data capture problem. In *Advances in natural computation, fuzzy systems and knowledge discovery* (Vol. 153). Springer. https://doi.org/10.1007/978-3-031-20738-9_87
25. Redžepagić, Ja., Malešević, N., & Dakic, V. (2026). Empirical performance and operational analysis of monolithic and distributed database architectures in Kubernetes environments. *Computers, 15*(5), 282. https://doi.org/10.3390/computers15050282
26. Amgothu, S. (2024). Innovative CI/CD pipeline optimization through canary and blue-green deployment. *International Journal of Computer Applications, 186*(50), 1-5. https://doi.org/10.5120/ijca2024924141
27. Trofymenko, O. H., & Dyka, A. I. (2022). Problems of testing e-commerce websites. In *Proceedings of the International Scientific Conference "Information Technologies and Management in Higher Education and Sciences"* (Part 3, pp. 195-199). Baltija Publishing. https://doi.org/10.30525/978-9934-26-277-7-230
28. Trofymenko, O. H., Pasternak, Yu. Yu., Manakov, S. Yu., & Loboda, Yu. H. (2021). Automation of e-commerce website testing. *Modern Special Technique, 2*(65), 46-59. https://doi.org/10.36486/mst2411–3816.2021.2(65).5

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 2 (34), 2026
ISSN 2663 – 4023

Olena Trofymenko
PhD, Associate Professor, Associate Professor at the Department of Software Engineering,
National University "Odesa Law Academy", Odesa, Ukraine,
ORCID:0000-0001-7626-0886
trofymenko@onua.edu.ua

Yuliia Loboda
PhD, Associate Professor, Associate Professor at the Department of Software Engineering
of the National University "Odesa Law Academy", Odesa, Ukraine,
ORCID:0000-0001-7083-552X
loboda@onua.edu.ua

Nataliia Loginova
PhD, Associate Professor, Head of the Department of Software Engineering
of the National University "Odesa Law Academy", Odesa, Ukraine,
ORCID:0000-0002-9475-6188
loginova@onua.edu.ua

Sergii Nikolaienko
PhD, Senior Software Engineer,
Odesa, Ukraine
ORCID:0009-0004-7276-1818
serezhanik@gmail.com

Oleksandr Karahuts
Student of the Faculty of Cybersecurity and Informaion Technologies,
National University "Odesa Academy of Law", Odesa, Ukraine,
ORCID:0009-0008-6593-5938
sashaKaraguts@gmail.com

CLASSIFICATION OF DATABASE DESIGN PROBLEMS IN MICROSERVICE ARCHITECTURE AND RECOMMENDATIONS FOR THEIR RESOLUTION

Abstract. The paper investigates the problems of designing data storage systems in modern microservice architectures. The relevance of the study is driven by the widespread adoption of microservices and the Polyglot Persistence approach, which improve scalability and flexibility but significantly complicate data organization, integration, and consistency management. These challenges are particularly evident in HoReCa information systems, where intensive transaction processing, continuous data updates, integration with external services, and real-time operations require reliable and efficient data storage solutions. The aim of the study is to perform comprehensive analysis, systematization, and multi-level classification of data storage system design problems in microservice architecture and to develop practical recommendations for addressing them using modern architectural approaches and design patterns. The research is based on system analysis, comparative analysis of contemporary architectural solutions, and generalization of scientific publications and industrial practices related to microservice-based information systems. A novel three-level classification of data storage design problems is proposed, covering conceptual, architectural, and operational levels. For each level, the main challenges, their causes, representative examples from HoReCa, e-commerce, and banking information systems, and corresponding practical recommendations are presented. The study demonstrates that the effectiveness of data storage organization depends not only on selecting appropriate database technologies but also on defining bounded contexts, establishing clear Data Ownership principles, choosing suitable consistency models, integrating heterogeneous storage technologies, and organizing operational support for distributed systems. Attention is paid to the application of architectural patterns such as Database per Service, Saga, CQRS, Event Sourcing, Transactional Outbox, Transactional Inbox, API Composition, and Domain Events as practical mechanisms for solving the identified problems. The scientific novelty of the study lies in the systematization of data management problems in microservice architecture by dividing them into conceptual, architectural, and operational levels, determining cause-and-effect relationships

between levels, and establishing correspondence between the identified problems, architectural patterns, conditions for their application, and accompanying constraints. The proposed approach can be applied in the development of scalable microservice-based information systems requiring reliable, consistent, and efficient data management.

Keywords: database; DBMS; SQL; NoSQL; Database per Service; polyglot persistence; microservices; microservice architecture; HoReCa; architectural patterns.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Soni, M., & Jyotinagar, V. (2023). SQL vs NoSQL databases for the microservices: A comparative survey. In *Proceedings of the 2nd International Conference on Edge Computing and Applications* (pp. 17-22). IEEE. https://doi.org/10.1109/icecaa58104.2023.10212190
2. Kavuluri, H. V., & Avula, S. B. (2024). Database engineering for microservices: PostgreSQL as the foundation. *International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5*(4). https://doi.org/10.63282/3050-9262.ijaidsml-v5i4p103
3. Halili, F., Nuhiji, A., & Mustafai Veliu, D. (2025). *Polyglot persistence in microservices: Managing data diversity in distributed systems*. arXiv. https://doi.org/10.48550/arXiv.2509.08014
4. André, M., Raglianti, M., Serbout, S., Cleve, A., & Lanza, M. (2025). *An empirical study on database usage in microservices*. arXiv. https://doi.org/10.48550/arXiv.2510.20582
5. Fowler, M. (n.d.). *Polyglot persistence*. https://martinfowler.com/bliki/PolyglotPersistence.html
6. Ali, A. M. (2024). Exploring database design patterns of microservices. *Journal of Artificial Intelligence, Machine Learning and Data Science*. https://doi.org/10.51219/jaimld/azra-jabeen-mohamed-ali/376
7. Ataei, P., & Staegemann, D. (2023). Application of microservices patterns to big data systems. *Journal of Big Data, 10*, 1-49. https://doi.org/10.1186/s40537-023-00733-4
8. Bandaru, S. (2021). Polyglot persistence in web engineering: Adaptive database selection across PostgreSQL, Oracle, MySQL, and MongoDB. *International Journal of Information and Electronics Engineering*. https://doi.org/10.48047/ijiee.2021.11.4.11
9. Kazanavičius, J., Mažeika, D., & Kalibatienė, D. (2022). An approach to migrate a monolith database into multi-model polyglot persistence based on microservice architecture: A case study for mainframe database. *Applied Sciences, 12*, 6189. https://doi.org/10.3390/app12126189
10. Laigner, R., Zhou, Y., Salles, M., Liu, Y., & Kalinowski, M. (2021). *Data management in microservices: State of the practice, challenges, and research directions*. arXiv. https://doi.org/10.48550/arXiv.2103.00170
11. Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News, 33*(2), 51-59. https://doi.org/10.1145/564585.564601
12. Ragothaman, K. (2025). Modernizing transactional systems: Deep dive from monolithic to microservices architectures. *CLEI Electronic Journal, 28*. https://doi.org/10.19153/cleiej.28.3.11
13. *Online food delivery market (2025-2030).* (2025). Grand View Research. https://www.grandviewresearch.com/industry-analysis/online-food-delivery-market-report
14. Lere, A., Pati, G., & Ege, E. (2023). Web-based restaurant food ordering application information system using agile development method. *Journal of Technology and Health, 1*, 104–110. https://doi.org/10.61677/jth.vi.93
15. Trofymenko, O. H., Manakov, S. Yu., Chykunov, P. O., Loboda, Yu. V., & Hurin, Ye. P. (2025). Routing in microfrontend applications: Architectural approaches, challenges, and solutions. *Cybersecurity: Education, Science, Technique, 3*(31), 635-651. https://doi.org/10.28925/2663-4023.2025.31.1057
16. Kaur, K., Bharany, S., Badotra, S., Aggarwal, K., Nayyar, A., & Sharma, S. (2022). Energy-efficient polyglot persistence database live migration among heterogeneous clouds. *The Journal of Supercomputing, 79*, 1-30. https://doi.org/10.1007/s11227-022-04662-6
17. Genfer, P., & Zdun, U. (2026). Using guided community detection to improve existing microservice designs. In *Service-oriented computing (ICSOC 2025)* (Lecture Notes in Computer Science, Vol. 16320). Springer. https://doi.org/10.1007/978-981-95-5012-8_15
18. Braz, L., França, B., & Cafeo, B. (2025). Dynamic analysis for detecting microservices antipatterns. In *Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software* (pp. 67-78). https://doi.org/10.5753/sbcars.2025.14576
19. Newman, S. (2021). *Building microservices* (2nd ed.). O'Reilly Media.
20. Randa, A. A., & Adi, A. M. (2026). Analyzing the evolution of microservice architecture: Challenges, solutions, and proposed design. *AIP Conference Proceedings, 3406*(1), Article 020023. https://doi.org/10.1063/5.0318956

21. Cho'ponov, O. (2026). Comparative analysis of microservices and monolithic architectures: Evaluating performance, economic sustainability, and human factors in the 2025–2026 ecosystem. *Advances in Science and Sustainability, 2*, 9-12. https://doi.org/10.70728/susta.v02.i05.002
22. Zhang, Y. (2025). *Exploration of enterprise big data microservice architecture based on domain-driven design (DDD)*. arXiv. https://doi.org/10.48550/arXiv.2511.05880
23. Tangudu, R. (2026). A unified architectural framework for microservices integrating DevOps, CI/CD, and runtime orchestration. *World Journal of Advanced Engineering Technology and Sciences, 18*, 98-107. https://doi.org/10.30574/wjaets.2026.18.3.0128
24. Hao, L., Jiang, T., Lin, Y., & Lu, Y. (2023). Methods for solving the change data capture problem. In *Advances in natural computation, fuzzy systems and knowledge discovery* (Vol. 153). Springer. https://doi.org/10.1007/978-3-031-20738-9_87
25. Redžepagić, Ja., Malešević, N., & Dakic, V. (2026). Empirical performance and operational analysis of monolithic and distributed database architectures in Kubernetes environments. *Computers, 15*(5), 282. https://doi.org/10.3390/computers15050282
26. Amgothu, S. (2024). Innovative CI/CD pipeline optimization through canary and blue-green deployment. *International Journal of Computer Applications, 186*(50), 1-5. https://doi.org/10.5120/ijca2024924141
27. Trofymenko, O. H., & Dyka, A. I. (2022). Problems of testing e-commerce websites. In *Proceedings of the International Scientific Conference "Information Technologies and Management in Higher Education and Sciences"* (Part 3, pp. 195-199). Baltija Publishing. https://doi.org/10.30525/978-9934-26-277-7-230
28. Trofymenko, O. H., Pasternak, Yu. Yu., Manakov, S. Yu., & Loboda, Yu. H. (2021). Automation of e-commerce website testing. *Modern Special Technique, 2*(65), 46-59. https://doi.org/10.36486/mst2411–3816.2021.2(65).5

Отримано редакцією журналу / Received: 03.02.26
Прорецензовано / Revised: 16.02.26
Схвалено до друку / Accepted: 24.09.26