DOI: https://doi.org/10.36910/6775-2524-0560-2024-55-22
УДК 004.75
Пригода Андрій Ярославович, аспірант, асистент
https://orcid.org/0000-0003-3774-4583
Державний торговельно-економічний університет Київ, Україна

ОЦІНКА ЕФЕКТИВНОСТІ ПРОЄКТУ РОЗРОБКИ ТА ВПРОВАДЖЕННЯ CRM-СИСТЕМИ НА ОСНОВІ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

Пригода А.Я. Оцінка ефективності проєкту розробки та впровадження crm-системи на основі мікросервісної архітектури. У статті проведено оцінку ефективності проєкту розробки та впровадження CRM-системи на основі мікросервісної архітектури. Описано принципи та сутність CRM-системи. Детально розглянуто види мікросервісів, описано технологію переходу між будівельними блоками архітектури. Наголошено, що застосування мікросервісів у проектуванні та впровадженні систем має багато переваг, таких як автономність, стабільний зв'язок, можливість компонування, масштабованість, стійкість до відмов. Описано принципи деталізації MSA, які сприяють узгодженню функціональності мікросервісу з однією окремою можливістю, яку сервіс надає загальній архітектурі програмного забезпечення. Підкреслено, що мікросервісна архітектура дозволяє уникнути більшості труднощів, які є в монолітній архітектурі пропонуючи концепцію часткового розгортання, коли виробничий сервер втрачає лише частину функціоналу. Зазначається, що додавання нового сервісу також не впливає на працездатність всієї хмари. Окреслено основні переваги мікросервісної архітектури, які включають в себе: простоту, зосередження на бізнес-функціоналі, покращену продуктивність і швидкість, масштабованість і гнучкість, автономні, багатопрофільні команди. Визначено недоліки мікросервісної архітектури такі як, збільшена складність, витрати, більш високий попит ресурсу, безпека, надійність, неузгодженість. Наголошено, що мікросервісна архітектура передбачає розподіл відповідальності між командами розробки за цілком природними межами: за межами самих сервісів. Так, у віданні кількох команд розробки знаходяться непересічні Набори мікросервісів, що забезпечує високу ізоляцію роботи. Додатковою приємною перевагою архітектури мікросервісу є можливість вибору різних технологій у різних мікросервісах. Загалом, підкреслено, що мікросервісна архітектура CRM-системи дозволяє підвищити надійність і стабільність функціонування за рахунок відносної автономності її окремих мікросервісних компонентів. Архітектура мікросервісу CRM-системи має високу масштабованість, у тому числі за рахунок використання хмарних технологій.
Ключові слова. CRM-система, мікросервіс, архітектура, проєкт, оцінка, ефективність, розробка.

Pryhoda A. Evaluation Of The Effectiveness Of The Crm System Development And Implementation Project Based On Microservice Architecture. The article evaluates the effectiveness of the CRM system development and implementation project based on microservice architecture. The principles and essence of the CRM system are described. The types of microservices are considered in detail, the technology of transition between building blocks of architecture is described. It is emphasized that the use of microservices in the design and implementation of systems has many advantages, such as autonomy, stable communication, the possibility of composition, scalability, resistance to failures. MSA's granularity principles are described, which help to align the functionality of a microservice with a single capability that the service provides to the overall software architecture. It is emphasized that the microservice architecture avoids most of the difficulties that exist in the monolithic architecture by offering the concept of partial deployment, when the production server loses only a part of the functionality. It is noted that adding a new service also does not affect the performance of the entire cloud. The main advantages of microservice architecture are outlined, which include: simplicity, focus on business functionality, improved performance and speed, scalability and flexibility, autonomous, multidisciplinary teams. Disadvantages of microservice architecture are identified, such as increased complexity, costs, higher resource demand, security, reliability, inconsistency. It is emphasized that the microservice architecture provides for the distribution of responsibility between development teams along completely natural boundaries: beyond the boundaries of the services themselves. So, several development teams have unique sets of microservices, which ensures high isolation of work. An additional nice benefit of microservice architecture is the ability to choose different technologies in different microservices. In general, it is emphasized that the microservice architecture of the CRM system allows to increase the reliability and stability of functioning due to the relative autonomy of its individual microservice components. The microservice architecture of the CRM system is highly scalable, including due to the use of cloud technologies.
Keywords. CRM system, microservice, architecture, project, evaluation, efficiency, development.

Вступ та постановка проблеми дослідження. Архітектура програмного забезпечення системи зображує організацію або структуру системи та надає пояснення того, як вона веде себе. Система являє собою сукупність компонентів, які виконують певну функцію або набір функцій. Іншими словами, архітектура програмного забезпечення забезпечує міцну основу, на якій можна будувати програмне забезпечення [1]. Серія архітектурних рішень і компромісів впливають на якість, продуктивність, ремонтопридатність і загальний успіх інформаційної системи. Неврахування поширених проблем і довгострокових наслідків може поставити систему під загрозу. Існує безліч моделей і принципів архітектури високого рівня, які зазвичай використовуються в сучасних інформаційних системах. Їх часто називають архітектурними стилями. Архітектура програмної системи рідко обмежується одним архітектурним стилем. Натомість комбінація стилів часто складає повну систему [2].

В наш час конкуренція на ринку досить висока та при ситуації взаємодії з клієнтами кожна компанія замислюється про впровадження CRM систем, а саме використання хмарних CRM, які доступні не тільки великим компаніям та й малим підприємцям. Концепція CRM (Customer Relationship Management – система управління взаємовідносинами з клієнтами) реалізовується шляхом застосування програмного забезпечення для досягнення наступних цілей: – автоматизації стратегій взаємодії із клієнтами; – оптимізації надання послуг; – отримання інформації про клієнтів та історію взаємовідносин з ними; – удосконалення процедур керування процесами спрямованих на надання клієнту повного пакету послуг відповідно до умов контракту; – забезпечення контролю результатів роботи.

Таким чином, система управління взаємовідносинами з клієнтами або CRM-система характеризується як інструмент підвищення ефективності взаємодії підприємства з клієнтами. Сьогодні CRM-системи є невід'ємним інструментом діяльності багатьох світових підприємств, що дозволяє їм набагато ефективніше встановлювати та розвивати відношення з клієнтами.

В умовах сьогодення мікросервісна архітектура відома як успішна та перспективна архітектура для CRM-систем. Застосування мікросервісів у проектуванні та впровадженні систем має багато переваг, таких як автономність, стабільний зв'язок, можливість компонування, масштабованість, стійкість до відмов. Однак складність викликів між мікросервісами призводить до проблем із безпекою, доступністю та керуванням даними під час виконання систем. Щоб вирішити ці проблеми, в останні роки різні дослідники та розробники зосередилися на використанні мікросервісних шаблонів у реалізації систем на основі мікросервісів. Патерни мікросервісів є результатом успішного досвіду розробників у вирішенні типових проблем у системах на основі мікросервісів. Однак досі не було надано жодних вказівок для поглибленого розуміння шаблонів мікросервісів і того, як їх застосувати до реальних CRM-систем.

Аналіз останніх досліджень і публікацій. Наукові діячі сьогодення внесли значний вклад у розробку та впровадження CRM-системи на основі мікросервісної архітектури. Був проведений ряд досліджень для вирішення даної проблеми.

У роботі [3] доведено, що архітектура мікросервісу забезпечує масштабованість, надійність і орієнтований на користувача підхід, який узгоджується з динамічною природою клінічних робочих процесів. Дослідження базується на прикладній методології дослідження, одночасно використовуючи підхід дизайнерського мислення для ітеративної розробки масштабованої базової системи, яка може врахувати особливості суб'єктивних даних пацієнтів і полегшити впровадження систем CBR у CDSS.

Дослідження [4] описує практичне дослідження переходу моноліту до архітектури мікросервісу, де модульна монолітна архітектура використовується як проміжний крок. Вплив на зусилля з міграції та продуктивність вимірюється для обох кроків. Поточний стан техніки аналізує міграцію монолітних систем до архітектури мікросервісів, зазначається, що зусилля з міграції та проблеми з продуктивністю вже значні при переході до модульного моноліту. Таким чином, для кожного з кроків встановлено чітке розмежування, яке може інформувати архітекторів програмного забезпечення про планування міграції монолітних систем. Зокрема, розглянуто компроміси виконання всього процесу міграції або просто переходу на модульний моноліт.

Тим не менш, окрім зазначеної вище документації, варто зазначити праці ще наступних науковців: Міліч Мілош, Ніколіч Драгана [5], Лі Сяочжоу, Калефато Фабіо, Ленардуцці Валентина, Тайбі Давіде [6], Чжен Лін, Вей Бо [7], Хамза Мухаммад [8], Чаплія Олег, Клім Галина [9], Темоор Музаффар [10], Маркес Гастон, Тарамаско Карла, Астудільо Ернан, Зальк Вінсент, Істрате Дан [11], Лі Вей [12], Існаені Рискулла, Утама Нур, Суаканто Сінунг [13], Странд Рубен, Крістенсен Ларс, Петручі Лауре [14], Браубах Ларс, Джандер Кай, Покар Олександр [15] та інших.

Проте, беручи до уваги вище зазначену наукову документацію, питання, пов'язане з розробкою та впровадженням CRM-системи на основі мікросервісної архітектури, все ще залишається недостатньо дослідженим та потребує подальшого опрацювання.

Мета дослідження. Метою даного дослідження є оцінка ефективності проєкту розробки та впровадження CRM-системи на основі мікросервісної архітектури.

Викладення основного матеріалу дослідження. Архітектура мікросервісу (MSA) – це новий підхід до реалізації сервісної архітектури програмного забезпечення (SBSA – Service-based Software Architecture) [16]. Загалом, термін «Мікросервісна архітектура» вперше використаний на конференції Cloud Computing Expo в 2005-му році Пітером Роджерсом, а почав набирати популярність у практиці та дослідженнях у 2014 та 2015 роках відповідно. Архітектура є сучасним

представленням SOA (Service-Oriented Architecture) і являє собою де-факто сучасний стандарт написання сучасних розподілених обчислювальних систем. Починаючи з 2012-го року термін набув широкого поширення, і саме з цього моменту часу багато команд підтримки веб-додатків стали розбивати свої моноліти на мікросервіси [17]. Отже, SOA та MSA мають спільне те, що вони використовують сервіси як функціональні будівельні блоки для реалізації архітектур розподіленого програмного забезпечення. Таким чином, вони стикаються з проблемами, властивими поняттю сервісу та пов'язаними, наприклад, з налаштуванням можливостей, композицією та координацією для виконання завдань і розгортанням середовища виконання. Однак мікросервіс MSA демонструє характеристики, які відрізняють його від сервісу SOA.

Деталізація MSA сприяє узгодженню функціональності мікросервісу з однією окремою можливістю, яку сервіс надає загальній архітектурі програмного забезпечення. SOA явно не передбачає певного ступеня деталізації сервісу. Натомість може реалізовувати детальну функціональність, специфічну для конкретної програми, але також враховувати реалізацію бізнес-процесів для всієї системи.

Крім того, SOA спрямований на максимізацію повторного використання послуг. MSA та покладається на адаптацію сервісів і прагне зберегти кількість мікросервісів. Тобто зменшити зв'язок між мікросервісами та підвищити автономію та ізоляцію мікросервісів з метою сприяння масштабованості та модифікованості архітектури [18]. Спільні бібліотеки – це підхід до зменшення повторного використання мікросервісів шляхом переміщення незалежних від сервісів функціональних можливостей до повторно використовуваних артефактів [19].

Окрім доступних операцій сервісу, інтерфейси також визначають структуру та формати обмінюваних даних. SOA зазвичай використовує Enterprise Service Bus (ESB) як платформу інтеграції послуг [20]. ESB може перетворювати дані, передані однією службою, у структури та формати, які можна інтерпретувати іншими несумісними службами. MSA, з іншого боку, не використовує складні засоби інтеграції для абстрагування від структур даних або форматів, передбачених інтерфейсами. Натомість мікросервіси самі відповідають за структуру даних і перетворення формату за необхідності. Знову ж таки, обґрунтуванням цієї характеристики MSA є зменшення зв'язку та підвищення автономності та ізоляції обслуговування.

Окрім перетворення структури даних і формату, ESB також можуть трансформувати протоколи технічного зв'язку, наприклад, MQTT [21] у SOAP і навпаки. Таким чином, SOA може забезпечити взаємодію, навіть якщо служби використовують різні технічні протоколи зв'язку. Навпаки, MSA делегує відповідальність за такі перетворення мікросервісам. Щоб уникнути або принаймні пом'якшити складність, спричинену перетвореннями протоколів, архітектури мікросервісів зазвичай покладаються щонайбільше на два протоколи – один для взаємодії мікросервісів «один-до-одного» (зазвичай синхронний), а інший – для мікросервісу «один-до-багатьох» (зазвичай асинхронний).

Для взаємодії між сервісами однієї архітектури програмного забезпечення (взаємодія між сервісами) SOA підтримує оркестровку та хореографію як основні моделі взаємодії [22]. У сценаріях оркестровки центральний посередник, наприклад, ESB, визначає взаємодію служб. У сценаріях хореографії служби самостійно приймають рішення про свою взаємодію. MSA зазвичай використовує шаблон хореографії для взаємодії між службами, щоб збільшити автономію мікросервісів.

Для взаємодії із зовнішніми компонентами архітектури (взаємодія поза послугами) SOA зазвичай покладається на ESB для маршрутизації та, можливо, перетворення обмінюваних даних. В архітектурах мікросервісів шлюзи API можуть використовуватися для посередництва між зовнішніми компонентами архітектури та внутрішніми мікросервісами архітектури. Однак, порівняно з ESB, шлюзи API забезпечують досить прості фасади, які агрегують операції інтерфейсу вибраних мікросервісів і надають їх зовнішнім компонентам [23].

Сфера застосування SOA призначена для реалізації корпоративних і міжкорпоративних SBSA, які вимагають різноманітних гетерогенних зовнішніх систем для участі в складних бізнес-процесах. Отже, SOA особливо кваліфікується як засіб інтеграції. MSA, однак, сприяє розробці, наприклад, хмарних додатків (CNA) або SBSA з чітко визначеними потоками обробки. Крім того, MSA часто використовується для декомпозиції додатків зі зниженою масштабованістю та зручністю обслуговування. Крім того, на відміну від SOA, MSA зазвичай зосереджується на розробці або міграції окремих програмних систем.

Порівняно з SOA передбачається, що MSA краще відповідає практичним потребам практиків, які займаються розробкою SBSA, наприклад, архітекторів програмного забезпечення та розробників. Це припущення викликано тим фактом, що, окрім SOA, MSA виникла виключно з практики застосування. Поки академічні кола не почали досліджувати це в 2015 році, MSA керувалося виключно потребами галузі.

Крім того, вважається, що MSA демонструє нижчий рівень складності, ніж SOA, це сприйняття значною мірою навіяно такими характеристиками MSA:

MSA зводить до мінімуму спільне використання функцій між службами. Хоча ця характеристика підвищує масштабованість і придатність до обслуговування, дозволяючи службам розвиватися незалежно, вона також сприяє гнучкості.

MSA опускає поняття ESB. Як результат, архітектури мікросервісів зазвичай не інтегрують комбіновані централізовані засоби для перетворення формату даних, структури та протоколу, а також оркестровки сервісу. Однак мікросервіси мають збільшений набір обов'язків порівняно з сервісами SOA.

MSA зменшує таксономію послуг. У SOA існує декілька типів сервісів, наприклад, бізнес-сервіси, корпоративні сервіси, сервіси додатків та сервіси інтеграції. Вони відрізняються деталізацією, сферою застосування та типом можливостей. З іншого боку, MSA зазвичай складається лише з двох типів послуг, які відрізняються лише типом можливостей. Функціональні служби реалізують бізнес-логіку, а служби інфраструктури забезпечують технічні можливості, наприклад, для виявлення служби, моніторингу або автентифікації.

MSA покладається на «полегшені» технології. Наприклад, SOA часто ототожнюють із застосуванням SOAP, тоді як очікується, що MSA використовуватиме Representational State Transfer (REST) [24] для синхронної взаємодії сервісів. Враховуючи підтримку SOAP різноманітних стандартів, наприклад мови опису веб-сервісів (WSDL) і сімейства стандартів WS-*, він зазвичай сприймається як більш складний ніж REST.

Право власності на послуги в SOA зазвичай залежить від типу служби. Наприклад, розробники несуть відповідальність за реалізацію служб додатків, а архітектори програмного забезпечення компонують послуги додатків у виконувані бізнес-послуги для всього підприємства. Тому SOA зазвичай збирає однорідні команди на основі професійного досвіду їхніх членів.

MSA, з іншого боку, сприяє передачі права власності на мікросервіс виділеній команді, яка несе повну відповідальність за всі проблеми, пов'язані з проектуванням, впровадженням і роботою мікросервісу.

Зосередженість MSA на неоднорідному складі команди та невеликому розмірі команди зрештою сприяє прийняттю DevOps [25]. DevOps – це набір практик, спрямованих на скорочення часу між фіксацією та розгортанням зміни, одночасно забезпечуючи високу якість програмної системи, що розробляється. Практики DevOps включають, наприклад, постійну інтеграцію та розгортання, а також проактивний моніторинг розглянутої системи програмного забезпечення. Хоча SOA може не перешкоджати прийняттю DevOps, вона все ж виходить за межі архітектурного стилю SOA.

Підсумовуючи, мікросервіс – це сервіс SBSA з наступними характеристиками:

– надає окремі можливості для інших компонентів архітектури. Тобто всі функціональні можливості мікросервісу вирішують одну проблему. Ця проблема має функціональний або інфраструктурний характер;

– максимально незалежний від інших компонентів з точки зору його впровадження, управління даними, тестування, розгортання та роботи;

– повністю відповідає за всі аспекти, пов'язані із взаємодією з іншими компонентами. Такі аспекти включають, наприклад, рішення щодо взаємозв'язків взаємодії, визначення протоколу зв'язку, структуру даних і перетворення формату та обробку відмов. Без серйозних технічних причин мікросервіс підтримує щонайбільше два протоколи зв'язку – один для синхронної взаємодії «один-до-одного», а інший – для асинхронної взаємодії «один-до-багатьох»;

– належить лише одній команді. Команда несе повну відповідальність за всі аспекти, пов'язані з розробкою, реалізацією та роботою мікросервісу.

Архітектура мікросервісу – це SBSA, функціональні компоненти якого, тобто компоненти, які реалізують його бізнес-логіку, є мікросервісами.

Мікросервісна архітектура дозволяє уникнути більшості труднощів, які є в монолітній архітектурі пропонуючи концепцію часткового розгортання, коли виробничий сервер втрачає лише частину функціоналу. Додавання нового сервісу також не впливає на працездатність всієї хмари. Ці переваги досягаються за рахунок того, що додаток зберігає більшу частину свого функціоналу, в той час як один мікросервіс знаходиться в стані перезапуску або запуску.

Основними перевагами мікросервісної архітектури є наступне [26].

Простота. Програми стає легше побудувати і підтримати, коли вони розділені на ряд менших, складових фрагментів.

Зосередження на бізнес-функціоналі. Мікросервіси дозволяють будувати продукт замість проєктів.

Покращена продуктивність і швидкість. Мікросервісна архітектура вирішує проблему продуктивності і швидкості, розкладаючи програми на керовані сервіси, які швидко розвиваються.

Масштабованість і гнучкість. Кожний мікросервіс може бути написаним з використанням різних технологій.

Автономні, багатопрофільні команди. Мікросервіси дуже зручні для розподілених команд. Розвиток великої системи моноліту може бути складним, якщо співробітники працюють з підрозділами у всьому світі [27].

Найбільший недолік мікросервісної архітектури – збільшена складність. Складність мікросервісного застосування безпосередньо корелює з кількістю залучених сервісів. У цього типу архітектури є набагато більше рухомих частин, ніж у традиційних програмних систем, що вимагає додаткового зусилля, планування, і автоматизації, щоб контролювати міжсервісне спілкування, моніторинг, тестування та розгортання. Ініціатива створення програмної системи мікросервісного типу вимагатиме структурної зміни в команді. Вона вимагає зрілої та гнучкої культури DevOps. Із заснованою на мікросервісах програмою командам потрібно вміти управляти всім життєвим циклом усіх сервісів. Це часто вимагає мігруючих компетенцій та прийняття різних рішень від менеджерів і архітекторів окремих команд. Ця зміна в ієрархії може бути важкою для компанії [28].

Крім того, зв'язок між людьми і командами стає набагато більш складним, оскільки у команд може не бути видимості усієї картини і того, як окремі послуги повинні працювати одна з одною, щоб створити повноцінне програмне забезпечення. Організація повинна також визначити, чи володіють їх люди навичками і мають необхідний досвід, щоб розробляти засновану на мікросервісах програмну систему. Оскільки команда може бути відповідальною за один сервіс, то розробники повинні бути добре обізнані про розвиток, розгортання, тестування та контроль застосування. Останньою вимогою буде мати хоча б одного DevOps розробника з відповідними навичками на команду.

Серед інших недоліків мікросервісів – витрати. Сервіси повинні будуть спілкуватися один з одним, що призведе до великої кількості запитів. Ці віддалені запити призводять до більш високих цін, пов'язаних з мережевим часом очікування і обробки, порівняно з традиційною архітектурою. Розробники повинні прикласти максимальних зусиль, щоб скоротити кількість запитів [29].

Інший фактор збільшеної вартості – більш високий попит ресурсу, оскільки кожен сервіс вимагатиме свого власного середовища і центрального процесора. Це – вимога, щоб зберегти кожен сервіс ізольованим. Крім того, через кожний сервіс, що використовує власну мову і технології, розробка додатку та неоднорідність архітектури можуть збільшити ресурси, які організація витрачає на управління і обслуговування [30].

Мікросервіси ставлять під сумнів рівень проблеми безпеки. Мікросервіси можуть поставити величезні проблеми безпеки через збільшення кількості комунікації міжсервісного спілкування по мережі. Всі ці взаємодії створюють можливість зовнішнім елементам отримати доступ до системи. Мікросервіси можуть стати причиною низької продуктивності. Наприклад, якщо сервіс викликає п'ять віддалених сервісів, кожен з яких викликає ще п'ять віддалених сервісів, то загальний час затримки може стати надмірно великим. Для вирішення цієї проблеми було придумано кілька способів. Можна робити менше віддалених викликів, збільшивши їх деталізацію, але це ускладнює модель програмування, так як тепер потрібно думати про те, як об'єднувати взаємодії між сервісами. Ще одним варіантом вирішення може буде асинхронність. Якщо зробити кілька асинхронних викликів одночасно, то загальний час очікування буде визначатися самим повільним викликом, а не сумою всіх. Це може дати великий виграш в продуктивності, але тягне за собою іншу проблему – асинхронність складно програмувати і ще важче налагоджувати.

Ще одна проблема при будування програмної системи мікросервісного типу – надійність. Очікується, що внутрішні виклики завжди будуть працювати, але віддалений виклик може відмовити в будь-який момент. З ростом кількості мікросервісів потенційних точок відмови стає більше. Розробники намагаються проєктувати систему з урахуванням відмов. Існують техніки, які потрібні для реалізації асинхронної взаємодії, вони добре підходять для роботи з відмовами, підвищуючи відмовостійкість. Однак, це не повна компенсація, тому що як і раніше залишаються додаткові складності з виявлення наслідків відмови для кожного зовнішнього виклику.

Мікросервіси схильні до проблеми неузгодженості. У монолітної системи можуть бути оновленими великою кількістю об'єктів в одній транзакції. З мікросервісами потрібно кілька ресурсів для оновлення та розподілення транзакції.

Як результат, розробники повинні пам'ятати про можливість неузгодженості і думати про те, як визначати моменти розсинхронізації, перш ніж програмувати щось, що потрібно буде покращувати з часом. Один з недоліків – складний онбордінг (процес адаптації користувача до продукту). Існує точка зору, що оскільки кожен сервіс невеликий, його простіше зрозуміти, але небезпека полягає в тому, що складність не зникає, вона просто зміщується на взаємозв'язку між сервісами. Це може маскувати збільшення складності експлуатації, наприклад, труднощі при налагодженні функцій, які включають кілька сервісів. Якісний вибір меж сервісів зменшує цю проблему, інший – робить її набагато складніше.

У сукупності з підходом безперервної інтеграції, що прийшов зі світу DevOps, мікросервісна архітектура дозволяє вводити у виробничу фазу повністю протестований (пройшов модульні тести, інтеграційні тести і end-to-end тести) функціонал. Написання якісних масштабних тестів для монолітного додатку часто виявляється на порядок більш важким завданням в силу змішування функціоналу в одному великому додатку [31].

Мікросервіси за своєю суттю дотримуються філософії UNIX: «робіть одне, але добре». Кожен сервіс має вузьку зону відповідальності, який якісно вирішує малий обсяг покладених на нього завдань. Нові розробники легше вливаються в робочий процес, оскільки вивчення і підтримка вихідного коду об'ємом в 2–3 тисячі рядків є куди більш простим завданням, ніж підтримка 30–40 тисяч рядків коду. Коли мова заходить про вихідний код, не можна не згадати філософію слабкої зв'язності мікросервісів. Використання повідомлень для взаємодії сервісів між собою дозволяє розробнику зосередитися на вирішенні поточного завдання, нехтуючи випадковим пошкодженням вже існуючий функціонал, яку часто відчувають розробники монолітних додатків.

Мікросервісна архітектура передбачає розподіл відповідальності між командами розробки за цілком природними межами: за межами самих сервісів. Так, у віданні кількох команд розробки знаходяться непересічні Набори мікросервісів, що забезпечує високу ізоляцію роботи. По суті, кожна команда вільна робити в своїх сервісах будь-які бажані зміни, якщо ці зміни не порушують наступних умов [32]:

Сервіс зберігає працездатність і виконує покладені на нього функціональні обов'язки, обумовлені проєктними керівниками.

Дотримано виконання контрактів споживачів. Ці контракти наказують, яку інформацію даний сервіс повинен надавати іншим сервісам за допомогою REST API або асинхронних повідомлень, що передаються в рамках повсюдно впровадженої event-driven архітектури.

З вище сказаного можна зробити висновок, що кожен мікросервіс зобов'язаний бути незалежним компонентом. Так як кожен мікросервіс працює в своєму процесі, він має чітко винести інтерфейс взаємодії з ним (API). Інші структурні елементи системи можуть взаємодіяти з сервісом лише через API, тому мінімізація зв'язків – один а найважливіших процесів в плануванні такої архітектури.

Додатковою приємною перевагою архітектури мікросервісу є можливість вибору різних технологій у різних мікросервісах. Наприклад, один сервіс може використовувати базу даних Neo4j, що працює з графами даних, прекрасно підходить для зберігання соціальних даних, а інший – класика PostgreSQL. І при розумному розбитті розв'язуваного веб-додатком завдання на контексти (області відповідальності, що характеризуються сильним зчепленням за змістом, наприклад, контекст користувачів, контекст електронних листів, контекст замовлень) простота підтримки програми і його продуктивність можуть відчутно зрости. Ця ж перевага дозволяє командам експериментувати з новими технологіями, які будуть впроваджені і в інші сервіси, якщо пропоновані рішення виявляться вдалими.

Висновки та перспективи подальших досліджень. Мікросервісна архітектура CRM-системи дозволяє підвищити надійність і стабільність функціонування за рахунок відносної автономності її окремих мікросервісних компонентів. Архітектура мікросервісу CRM-системи має високу масштабованість, у тому числі за рахунок використання хмарних технологій. Складність таких систем полягає в підтримці стану, синхронізації та узгодженості даних але вона є максимально масштабованою та володіє підвищеною здатністю ефективно виконувати всі функції покладені на неї.

Список бібліографічного опису

1. Аль Равашдех Лейт Ахмед Мустафа. Оцінка похибок динамічних нейронних мереж для вимірювальних систем. *Метрологія та прилади.* 2018. № 3. С. 33–35.
2. Бойчук В. О. Сучасні штучні нейронні мережі та підходи до їх моделювання. *Вимірювальна та обчислювальна техніка в технологічних процесах.* 2014. № 4. С. 216–219.
3. Jaiswal Amar, Meisingset Ingebrigt. A Microservice-Based Architecture for Clinical Decision Support System for Addressing Non-Specific Musculoskeletal Disorders: The SupportPrim Project. 2024. DOI: 10.21203/rs.3.rs-4122773/v1.
4. Faustino Diogo, Gonçalves Nuno, Portela Manuel, Silva António. Stepwise migration of a monolith to a microservice architecture: Performance and migration effort evaluation. *Performance Evaluation*. 2024. № 164. P. 102-411. DOI: 10.1016/j.peva.2024.102411.
5. Milić Miloš, Nikolić Dragana. Development of a Quality-Based Model for Software Architecture Optimization: A Case Study of Monolith and Microservice Architectures. Symmetry. 2022. № 14. 1824 p. DOI: 10.3390/sym14091824.
6. Li Xiaozhou, Calefato Fabio, Lenarduzzi Valentina, Taibi Davide. Toward Collaboration Optimization in Microservice Projects based on Developer Personalities. 2024.
7. Zheng Ling, Wei Bo. Application of microservice architecture in cloud environment project development. *MATEC Web of Conferences*. 2018. 189 p. 03023. DOI: 10.1051/matecconf/201818903023.
8. Hamza Muhammad. Transforming Monolithic Systems to a Microservices Architecture. *ACM SIGSOFT Software Engineering Notes*. 2023. №48. P. 67-69. DOI: 10.1145/3573074.3573091.
9. Chaplia Oleh, Klym Halyna. NODE.JS PROJECT ARCHITECTURE WITH SHARED DEPENDENCIES FOR MICROSERVICES. *Measuring Equipment and Metrology*. 2023. №84. P. 53-58. DOI: 10.23939/istcmtm2023.03.053.
10. Temoor Muzaffar. Architecture for Microservice Based System. *A Report*. 2020. DOI: 10.13140/RG.2.2.17340.16004/1.
11. Márquez Gastón, Taramasco Carla, Astudillo Hernán, Zalc Vincent, Istrate Dan. Involving Stakeholders in the Implementation of Microservice-Based Systems: A Case Study in an Ambient-Assisted Living System. *IEEE Access*. 2021. PP. 1-1. DOI: 10.1109/ACCESS.2021.3049444.
12. Li Wei. Design and Implementation of Flight Inquiry and Booking System Based on Microservice Architecture. *Highlights in Science, Engineering and Technology*. 2023. № 56. P. 697-703. DOI: 10.54097/hset.v56i.10836.
13. Isnaeni Rizqullah, Utama Nur, Suakanto Sinung. Backend Development of a Microservice-Based Website Application for Public Issue Reporting: Case Studyn in People Representative Council. Journal La Multiapp. 2024. № 5. P. 63-77. DOI: 10.37899/journallamultiapp.v5i2.1148.
14. Strand Ruben, Kristensen Lars, Petrucci Laure. Development and Verification of a Microservice Architecture for a Fire Risk Notification System. 2023. DOI: 10.1007/978-3-662-68191-6_2.
15. Braubach Lars, Jander Kai, Pokahr Alexander. (2022). Macro Architecture for Microservices: Improving Internal Quality of Microservice Systems. 2022. DOI: 10.1007/978-3-030-96627-0_10.
16. Jo Yong, Cho Se, Choi Byoungwook. Towards a ROS2-based software architecture for service robots. *Bulletin of Electrical Engineering and Informatics*. 2023. №12. P.3027-3038. DOI: 10.11591/eei.v12i5.5590.
17. Fajar Ahmad, Limonthy Stanley, Handopo Josua, Purnawan Fandy, Kesuma Adidharma. (2023). System Architecture for IT Talent Ecosystem Using Service Oriented Approach. *HighTech and Innovation Journal*. 2023. № 4. P. 739-748. DOI: 10.28991/HIJ-2023-04-04-03.
18. Graves Tom. Basics – Service-Oriented Architecture. 2023. DOI: 10.1007/978-1-4842-9189-4_2.
19. Gottardi Thiago, Braga Rosana. Run-time Adaptable Service Oriented Architecture in the Context of Repository Systems. Anais do Workshop em Modelagem e Simulação de Sistemas Intensivos em Software (MSSiS). 2023. P.21-30. DOI: 10.5753/mssis.2023.235434.
20. Bais Aditya, Mishra Varun. Analysis of Data Functionality in Enterprise Service Bus. 2017. DOI: 10.4018/978-1-5225-2157-0.ch005.
21. Shah Pujan. MQTT Systems: A Survey. International Journal for Research in Applied Science and Engineering Technology. 2024. № 12. P. 1063-1067. DOI: 10.22214/ijraset.2024.59000.
22. Karande Aarti, Karande Milind, Meshram Bhushan. Choreography and Orchestration using Business Process Execution Language for SOA with Web Services. *International Journal of Computer Science Issues*. 2011. № 8.
23. Roshen Waseem. Service oriented architecture enterprise service bus with universal ports. 2014.
24. Arsana I Nyoman, Purnawati Niwayan. Analysis of the implementation of Restful (Representational State Transfer) web services for data integration between information systems. 2023. № 7. DOI: 10.35335/mantik.v7i2.4016.
25. Tatineni Sumanth. Applying DevOps Practices for Quality and Reliability Improvement in Cloud-Based Systems. 2023. DOI: 10.13140/RG.2.2.25688.88326.
26. Kamil Figura. Why you should choose the microservices architecture? Режим доступу до ресурсу: https://pretius.com/blog/benefits-of-microservices/ Дата звернення: 14.05.2024

27. Antipattern-Based Problem Injection for Assessing Performance and Reliability Evaluation Techniques. *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW).* 2016. С. 64–70.
28. Тарарака В. Д. Архітектура комп'ютерних систем: навчальний посібник. Житомир: ЖДТУ, 2018. 383 с.
29. Rajesh RV, Spring Microservices. Packt Publishing, 2016. 436 с.
30. AlQershi N., Mokhtar S.S.M. Innovative CRM and Performance of SMEs: The Moderating Role of Relational Capital. *Journal of Open Innovation*. 2020, № 6, Pp. 155. https://doi.org/10.3390/joitmc6040155.
31. Anna K., Lazaros P. Exploring E-CRM Implementation in Sport Tourism Hotels in Peloponnese. *In: Springer Proceedings in Business and Economics*. 2020. Pp. 611–627. https:// doi.org/10.1007/978-3-030-36342-0_47.
32. Wilson H., Daniel E., McDonald M. Factors for success in customer relationship management (CRM) systems. *Journal of marketing management*, 2002, № 18(1-2), Pp. 193-219.

References:

1. Al Rawashdeh Leit Ahmed Mustafa. Fehlerschätzung dynamischer neuronaler Netze für Messsysteme. Messtechnik und Geräte. 2018. Nr. 3. S. 33
2. Boychuk V. O. Moderne künstliche neuronale Netze und Ansätze zu ihrer Modellierung. Mess- und Rechengeräte in technologischen Prozessen. 2014. Nr. 4. S. 216–219.
3. Jaiswal Amar, Meisingset Ingebrigt. A Microservice-Based Architecture for Clinical Decision Support System for Addressing Non-Specific Musculoskeletal Disorders: The SupportPrim Project. 2024. DOI: 10.21203/rs.3.rs-4122773/v1.
4. Faustino Diogo, Gonçalves Nuno, Portela Manuel, Silva António. Stepwise migration of a monolith to a microservice architecture: Performance and migration effort evaluation. *Performance Evaluation*. 2024. № 164. P. 102-411. DOI: 10.1016/j.peva.2024.102411.
5. Milić Miloš, Nikolić Dragana. Development of a Quality-Based Model for Software Architecture Optimization: A Case Study of Monolith and Microservice Architectures. Symmetry. 2022. № 14. 1824 p. DOI: 10.3390/sym14091824.
6. Li Xiaozhou, Calefato Fabio, Lenarduzzi Valentina, Taibi Davide. Toward Collaboration Optimization in Microservice Projects based on Developer Personalities. 2024.
7. Zheng Ling, Wei Bo. Application of microservice architecture in cloud environment project development. *MATEC Web of Conferences*. 2018. 189 p. 03023. DOI: 10.1051/matecconf/201818903023.
8. Hamza Muhammad. Transforming Monolithic Systems to a Microservices Architecture. *ACM SIGSOFT Software Engineering Notes*. 2023. №48. P. 67-69. DOI: 10.1145/3573074.3573091.
9. Chaplia Oleh, Klym Halyna. NODE.JS PROJECT ARCHITECTURE WITH SHARED DEPENDENCIES FOR MICROSERVICES. *Measuring Equipment and Metrology*. 2023. №84. P. 53-58. DOI: 10.23939/istcmtm2023.03.053.
10. Temoor Muzaffar. Architecture for Microservice Based System. *A Report*. 2020. DOI: 10.13140/RG.2.2.17340.16004/1.
11. Márquez Gastón, Taramasco Carla, Astudillo Hernán, Zalc Vincent, Istrate Dan. Involving Stakeholders in the Implementation of Microservice-Based Systems: A Case Study in an Ambient-Assisted Living System. *IEEE Access*. 2021. PP. 1-1. DOI: 10.1109/ACCESS.2021.3049444.
12. Li Wei. Design and Implementation of Flight Inquiry and Booking System Based on Microservice Architecture. *Highlights in Science, Engineering and Technology*. 2023. № 56. P. 697-703. DOI: 10.54097/hset.v56i.10836.
13. Isnaeni Rizqullah, Utama Nur, Suakanto Sinung. Backend Development of a Microservice-Based Website Application for Public Issue Reporting: Case Studyn in People Representative Council. Journal La Multiapp. 2024. № 5. P. 63-77. DOI: 10.37899/journallamultiapp.v5i2.1148.
14. Strand Ruben, Kristensen Lars, Petrucci Laure. Development and Verification of a Microservice Architecture for a Fire Risk Notification System. 2023. DOI: 10.1007/978-3-662-68191-6_2.
15. Braubach Lars, Jander Kai, Pokahr Alexander. (2022). Macro Architecture for Microservices: Improving Internal Quality of Microservice Systems. 2022. DOI: 10.1007/978-3-030-96627-0_10.
16. Jo Yong, Cho Se, Choi Byoungwook. Towards a ROS2-based software architecture for service robots. *Bulletin of Electrical Engineering and Informatics*. 2023. №12. P.3027-3038. DOI: 10.11591/eei.v12i5.5590.
17. Fajar Ahmad, Limonthy Stanley, Handopo Josua, Purnawan Fandy, Kesuma Adidharma. (2023). System Architecture for IT Talent Ecosystem Using Service Oriented Approach. *HighTech and Innovation Journal*. 2023. № 4. P. 739-748. DOI: 10.28991/HIJ-2023-04-04-03.
18. Graves Tom. Basics – Service-Oriented Architecture. 2023. DOI: 10.1007/978-1-4842-9189-4_2.
19. Gottardi Thiago, Braga Rosana. Run-time Adaptable Service Oriented Architecture in the Context of Repository Systems. Anais do Workshop em Modelagem e Simulação de Sistemas Intensivos em Software (MSSiS). 2023. P.21-30. DOI: 10.5753/mssis.2023.235434.
20. Bais Aditya, Mishra Varun. Analysis of Data Functionality in Enterprise Service Bus. 2017. DOI: 10.4018/978-1-5225-2157-0.ch005.
21. Shah Pujan. MQTT Systems: A Survey. International Journal for Research in Applied Science and Engineering Technology. 2024. № 12. P. 1063-1067. DOI: 10.22214/ijraset.2024.59000.
22. Karande Aarti, Karande Milind, Meshram Bhushan. Choreography and Orchestration using Business Process Execution Language for SOA with Web Services. *International Journal of Computer Science Issues*. 2011. № 8.
23. Roshen Waseem. Service oriented architecture enterprise service bus with universal ports. 2014.
24. Arsana I Nyoman, Purnawati Niwayan. Analysis of the implementation of Restful (Representational State Transfer) web services for data integration between information systems. 2023. № 7. DOI: 10.35335/mantik.v7i2.4016.
25. Tatineni Sumanth. Applying DevOps Practices for Quality and Reliability Improvement in Cloud-Based Systems. 2023. DOI: 10.13140/RG.2.2.25688.88326.

26. Kamil Figura. Why you should choose the microservices architecture? Режим доступу до ресурсу: https://pretius.com/blog/benefits-of-microservices/ Дата звернення: 14.05.2024
27. Antipattern-Based Problem Injection for Assessing Performance and Reliability Evaluation Techniques. *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW).* 2016. С. 64–70.
28. Тарарака В. Д. Архітектура комп'ютерних систем: навчальний посібник. Житомир: ЖДТУ, 2018. 383 с.
29. Rajesh RV, Spring Microservices. Packt Publishing, 2016. 436 с.
30. AlQershi N., Mokhtar S.S.M. Innovative CRM and Performance of SMEs: The Moderating Role of Relational Capital. *Journal of Open Innovation*. 2020, № 6, Pp. 155. https://doi.org/10.3390/joitmc6040155.
31. Anna K., Lazaros P. Exploring E-CRM Implementation in Sport Tourism Hotels in Peloponnese. *In: Springer Proceedings in Business and Economics*. 2020. Pp. 611–627. https:// doi.org/10.1007/978-3-030-36342-0_47.
Wilson H., Daniel E., McDonald M. Factors for success in customer relationship management (CRM) systems. *Journal of marketing management*, 2002, № 18(1-2), Pp. 193-219