DOI: https://doi.org/10.36910/6775-2524-0560-2024-57-05
УДК 004.72
Горшков Володимир Володимирович, магістрант
Ліщина Наталія Миколаївна, к.т.н., доцент
https://orcid.org/0000-0002-5200-536X
Сичук Віктор Анатолійович, к.т.н., доцент
https://orcid.org/0000-0002-8267-0846
Луцький національний технічний університет, м. Луцьк, Україна

МОДУЛЬНИЙ МОНОЛІТ: АРХІТЕКТУРНИЙ ПІДХІД ДЛЯ ПОБУДОВИ ВИСОКОДОСТУПНОЇ СИСТЕМИ УПРАВЛІННЯ ЗАРЯДНИМИ СТАНЦІЯМИ

Горшков В.В., Ліщина Н.М., Сичук В.А. Модульний моноліт: архітектурний підхід для побудови високодоступної системи управління зарядними станціями. У статті розглядається застосування архітектури модульного моноліту для розробки систем управління зарядними станціями для електромобілів. Висвітлюються основні переваги цієї архітектури, зокрема забезпечення високої доступності, відмовостійкості та ефективного масштабування системи. Наведено приклад побудови програмного комплексу, який забезпечує надійну роботу зарядних станцій в умовах високих навантажень, а також дозволяє розширювати функціонал без порушення стабільності роботи системи. Особлива увага приділена розподілу відповідальності, керуванню даними та методам взаємодії між модулями системи. Розглядаються питання інтеграції з централізованою системою обробки даних та зберігання великих обсягів інформації. У висновках проаналізовано можливості застосування модульного моноліту для побудови сучасних високонадійних систем і перспективи його використання в інших галузях.
Ключові слова: модульний моноліт, зарядні станції, високодоступність, відмовостійкість, масштабування, OCPP, брокер повідомлень, архітектура програмного забезпечення.

Horshkov V., Lishchyna N., Sychuk V. Modular Monolith: An Architectural Approach for Building a High-Availability Electric Vehicle Charging Station Management System. This article explores the application of modular monolith architecture in the development of management systems for electric vehicle charging stations. It outlines the key advantages of the modular monolith architecture, particularly in ensuring high availability, fault tolerance, and efficient scalability of the system. The article provides an example of building a software solution that ensures reliable operation of charging stations under high load conditions while allowing functional expansion without disrupting system stability. Special attention is given to the separation of responsibilities, data management, and methods of interaction between system modules. The integration with a centralized data processing system and large-scale data storage is also discussed. The conclusion analyzes the potential of using the modular monolith for building modern, high-reliability systems and the prospects for implementing such approaches in other industries.
Keywords: modular monolith, charging stations, high availability, fault tolerance, scalability, OCPP, message broker, software architecture.

Постановка наукової проблеми. З розвитком інфраструктури електричного транспорту зростає потреба у створенні високонадійних та масштабованих систем управління зарядними станціями. Такі системи повинні забезпечувати безперебійне функціонування зарядних пристроїв, підтримувати високу доступність навіть за умов інтенсивного навантаження, а також бути гнучкими до змін у вимогах та нових технологічних тенденцій. У сучасних умовах традиційні підходи до побудови систем часто стикаються з проблемами низької відмовостійкості, складнощами масштабування та підтримки, що може призводити до перебоїв у роботі зарядних станцій, втрат даних або уповільнення процесів обслуговування.

Одним із найважливіших аспектів створення таких систем є ретельне планування на етапі вибору архітектури, здатної задовольнити вимоги до продуктивності, доступності та гнучкості. Архітектура повинна забезпечувати можливість швидкої адаптації до змін, розширення функціональності та інтеграції з іншими компонентами системи без шкоди для її стабільності.

Таким чином, постає наукова проблема вибору та адаптації архітектури для розробки високодоступної системи управління зарядними станціями.

Аналіз останніх досліджень і публікацій. У сучасному світі, з розвитком інфраструктури електричного транспорту, активно досліджуються різні архітектурні підходи для створення систем управління зарядними станціями [11]. Однак більшість існуючих реалізацій можна умовно поділити на дві категорії: монолітні та мікросервісні.

Монолітні системи, як правило, є простими на етапі реалізації. Вони забезпечують швидкий запуск проектів, оскільки всі компоненти системи інтегровані в єдину програму. Це дозволяє зосередитися на основних функціональних вимогах без необхідності обробляти складності, пов'язані з комунікацією між модулями. Проте, за рахунок такої простоти, монолітні архітектури часто стикаються з серйозними проблемами під час масштабування. При зростанні навантажень або

необхідності впровадження нових функцій, такі системи можуть виявитися нездатними швидко адаптуватися, що призводить до збільшення часу розробки та ризиків виникнення помилок [5]. Згідно з дослідженнями [1], монолітні архітектури вимагають значних зусиль для модифікації та підтримки, що робить їх менш привабливими для високонадійних систем, які повинні витримувати значні навантаження.

З іншого боку, мікросервісні архітектури пропонують значно вищу гнучкість та масштабованість. Кожен компонент системи реалізується як окремий сервіс, що дозволяє команді працювати над різними частинами проекту незалежно один від одного. Це також спрощує тестування, оскільки кожен мікросервіс можна перевіряти окремо. Однак, незважаючи на свої переваги, мікросервісні системи є дуже складними під час розробки та тестування. Вони потребують належної організації комунікації між сервісами, налаштування безпеки, моніторингу та управління даними, що може значно ускладнити загальний процес розробки [2]. Багато дослідників, вказують на те, що мікросервісні архітектури можуть стати причиною збільшення витрат на розробку та підтримку систем, особливо в умовах недостатньої підготовки команди.

У контексті управління зарядними станціями, зростає інтерес до архітектури модульного моноліту, яка прагне поєднати переваги обох підходів. Модульний моноліт дозволяє реалізувати систему як єдине ціле, що має чітко визначені модулі з дотриманням принципів розділення відповідальності та структур даних. Це забезпечує простоту реалізації та підтримки, а також можливість гнучкого масштабування при потребі. Останні дослідження, зокрема, роботи [3] і [4], показують, що модульний моноліт може бути ефективним рішенням для будь-яких систем, що забезпечує необхідний рівень доступності та відмовостійкості без складнощів, притаманних мікросервісам.

Враховуючи вищевикладене, можна стверджувати, що вибір архітектурного підходу є критично важливим для реалізації системи управління зарядними станціями. Комбінація переваг монолітних і мікросервісних типів архітектури у модульному моноліті може стати оптимальним рішенням для створення високодоступних і надійних систем.

Виділення невирішених раніше частин загальної проблеми. Попри досягнення в розробці систем управління зарядними станціями для електричних транспортних засобів, залишаються невирішені питання щодо вибору оптимальної архітектури, що поєднує простоту впровадження та масштабованість. Монолітні системи, хоча й забезпечують швидке розгортання, мають низьку адаптивність до змін у навантаженнях, тоді як мікросервісні архітектури ускладнюють розробку та інтеграцію через необхідність ретельного управління комунікацією та безпекою. Важливим є також питання інтеграції модульних компонентів та зберігання великих обсягів даних. Таким чином, існує потреба в архітектурних рішеннях, які б усували недоліки традиційних підходів, зокрема через створення модульних монолітів, що забезпечують баланс між простотою, масштабованістю та стабільністю.

Мета дослідження полягає у розробці архітектури модульного моноліту для системи управління зарядними станціями електричних транспортних засобів, яка забезпечить високу доступність, відмовостійкість та ефективне масштабування. Дослідження прагне виявити оптимальні підходи до проектування модульної структури, що дозволить адаптуватися до змінюваних вимог, розширювати функціональність системи без порушення її стабільності та інтегрувати з іншими компонентами екосистеми управління. Основна увага буде приділена аналізу проблем, пов'язаних із традиційними монолітною та мікросервісною архітектурами, та виявленню переваг модульного моноліту як інноваційного рішення для сучасних високонадійних систем.

Основна частина дослідження. Системи управління зарядними станціями електричних транспортних засобів вимагають ретельного підходу до проектування архітектури, що забезпечує ефективну комунікацію, масштабованість і надійність.

В основі взаємодії зарядних станцій та центральної системи управління являється відкритий протокол для комунікації – OCPP (Open Charge Point Protocol). Використання цього протоколу забезпечує стандартизований спосіб обміну даними між найбільш важливими компонентами системи.

Взаємодія між зарядними станціями та центральною системою базується на використанні комунікаційних протоколів, таких як WebSocket або SOAP, які дозволяють забезпечити постійну передачу даних у режимі реального часу та управління процесами зарядки.

WebSocket – це сучасний протокол для двостороннього зв'язку між клієнтом і сервером у реальному часі. Він є одним із найбільш використовуваних механізмів для взаємодії зарядних станцій із центральною системою завдяки своїй ефективності та мінімальним затримкам у передачі даних. Використання WebSocket дозволяє забезпечити постійне з'єднання між зарядною станцією та центральним сервером, що дозволяє станції миттєво надсилати інформацію про стан зарядки, отримувати команди від серверу і оперативно реагувати на зміни.

Як показано на рисунку 1, центральна система виступає як сервер WebSocket, тоді як зарядна станція функціонує як клієнт. Протокол OCPP регламентує типи JSON-повідомлень, які можуть передаватися через WebSocket-з'єднання.

Рис. 1. Взаємодія центральної системи та зарядної станції

У випадку, коли в системі існує клієнтська частина для керування зарядними станціями, для забезпечення передачі інформації у реальному часі, особливу увагу слід звернути на необхідності підтримки стабільного з'єднання зарядних станції, а також на швидкості обробки та керування даними на рівні центральної системи управління. На рисунку 2 зображено взаємодію зарядних станцій та клієнтської частини, яка відбувається в режимі реального часу, шляхом обробки усіх запитів центральною системою управління.

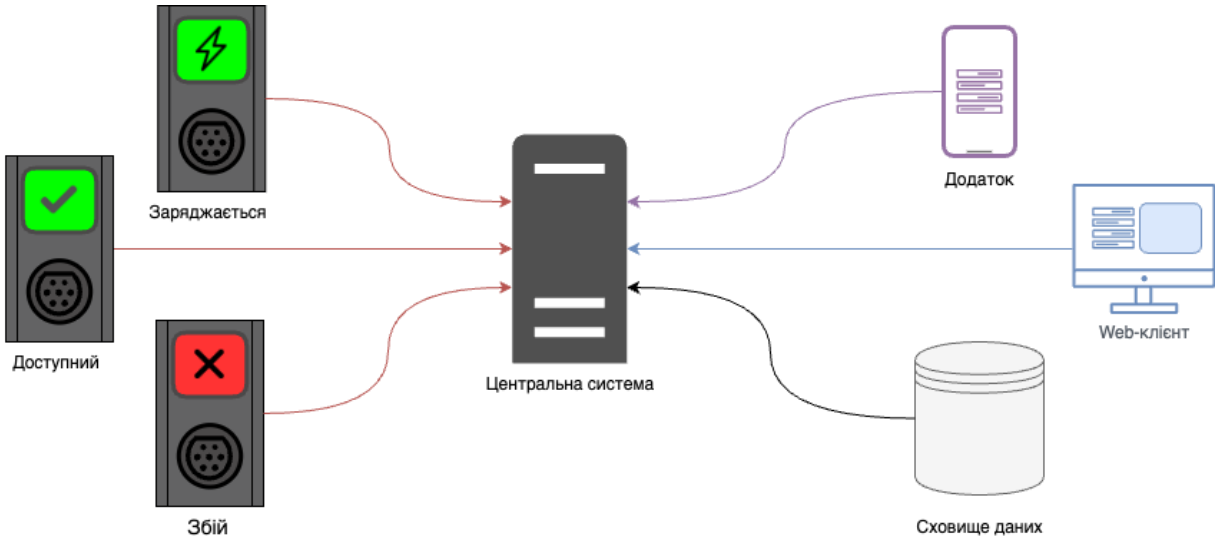

Рис. 2. Взаємодія центральної системи та клієнтської частини

Основні завдання, що потребують управління через програмне забезпечення або додаток:

1. **Авторизація користувача та початок зарядки.** Користувач ідентифікується за допомогою мобільного додатку або RFID-карти. Після авторизації запит на початок зарядки передається на центральний сервер. У відповідь система надсилає команду на зарядну станцію для початку процесу зарядки.
2. **Моніторинг та контроль процесу зарядки**. Програмне забезпечення через WebSocket отримує в реальному часі дані про процес зарядки, такі як поточна потужність, обсяг

споживаної енергії та тривалість зарядки. Це дозволяє користувачам або адміністраторам контролювати стан зарядки та виконувати необхідні дії, наприклад, зупинку зарядки або зміни параметрів.

3. **Оновлення прошивки та конфігураційні зміни**. Дистанційне оновлення програмного забезпечення зарядної станції є критично важливим для підтримки безпеки та функціональності системи. Станція отримує інструкції від центрального серверу для оновлення прошивки, що дозволяє впроваджувати нові функції або виправлення помилок без необхідності фізичного втручання.
4. **Налаштування профілів зарядки**. Користувачі можуть через додаток встановлювати спеціальні профілі зарядки, такі як час початку зарядки або обмеження потужності. Ці параметри передаються на сервер, який потім надсилає команди на зарядну станцію для налаштування необхідних параметрів.
5. **Аналіз даних та звітність**. Дані про стан зарядних станцій і проведені зарядки постійно передаються на центральну систему для зберігання та подальшого аналізу. Адміністратори або оператори можуть через інтерфейс користувача отримувати доступ до статистики, аналізувати споживання енергії, продуктивність станцій та інші показники, що важливі для операційного управління.

Враховуючи вищеописані вимоги, з метою забезпечення високої доступності, швидкості обробки інформації та можливості до швидкого масштабування, слід провести логічний поділ бізнес-логіки системи на наступні складові:

- комунікація із зарядними станціями (communication) – відповідає за підключення зарядних станцій до системи за протоколом OCPP, прийом і відправку повідомлень, керування станом з'єднання і забезпечення зв'язку із зарядними пристроями по WebSocket;
- брокер асинхронних повідомлень – для координації між компонентами використовується Advanced Message Queuing Protocol, реалізований брокером повідомлень, що сприяє обміну даними в реальному часі та знижує навантаження на окремі частини системи за допомогою асинхронної комунікації;
- авторизація (authorization) – відповідає за автентифікацію користувачів або RFID міток, забезпечуючи безпечний доступ до системи і підтримку даних про доступ;
- керування зарядними станціями (charge point management) – відповідає за збереження інформації про зарядні станції, їх параметри та пов'язаних із ними користувачів, надаючи адміністрування та моніторинг обладнання;
- налаштування (configuration) – відповідає за керування налаштуваннями зарядних станцій, оновленням програмного забезпечення зарядної станції, тощо;
- діагностика (diagnostics) – відповідає за збереження статусу та метрик зарядної станції, обробки інформації про їх зміну, тощо;
- статистика (statistics) – зберігає інформацію про тарифи, обсяги спожитої електроенергії, фінансові витрати, що дозволяє аналізувати ефективність системи та її використання;
- керування транзакціями (transaction management) – обробляє та зберігає інформацію про транзакції, забезпечуючи деталізований запис, а також моніторинг усіх операцій на зарядних станціях;
- керування користувачами (user management) – відповідає за збереження та обробку інформації про користувачів, надає можливості управління профілями користувачів клієнтської частини системи;
- комунікація з клієнтською частиною (client communication) – компонент API, який повинен забезпечити взаємодію з користувацькими інтерфейсами, обробляючи HTTP/HTTPS запити з клієнтських додатків і надаючи необхідні дані.

Враховуючи значну кількість компонентів системи, для забезпечення високої доступності, важливо забезпечити щоб кожен елемент системи працював незалежно та взаємодіяв з іншими за допомогою протоколів, таких як HTTP/HTTPS, WebSocket або AMQP. При цьому, кожен компонент на рівні центральної системи управління повинен реалізовувати наскрізний домен у межах певного контексту, мати свою пов'язану модель даних домену та бізнес-логіку. Такі елементи системи повинні бути розроблені автономно та розгорнуті незалежно [6, 8].

На рисунку 3 схематично зображено вищеописану систему, особливості взаємодії між її компонентами, та можливості масштабування.

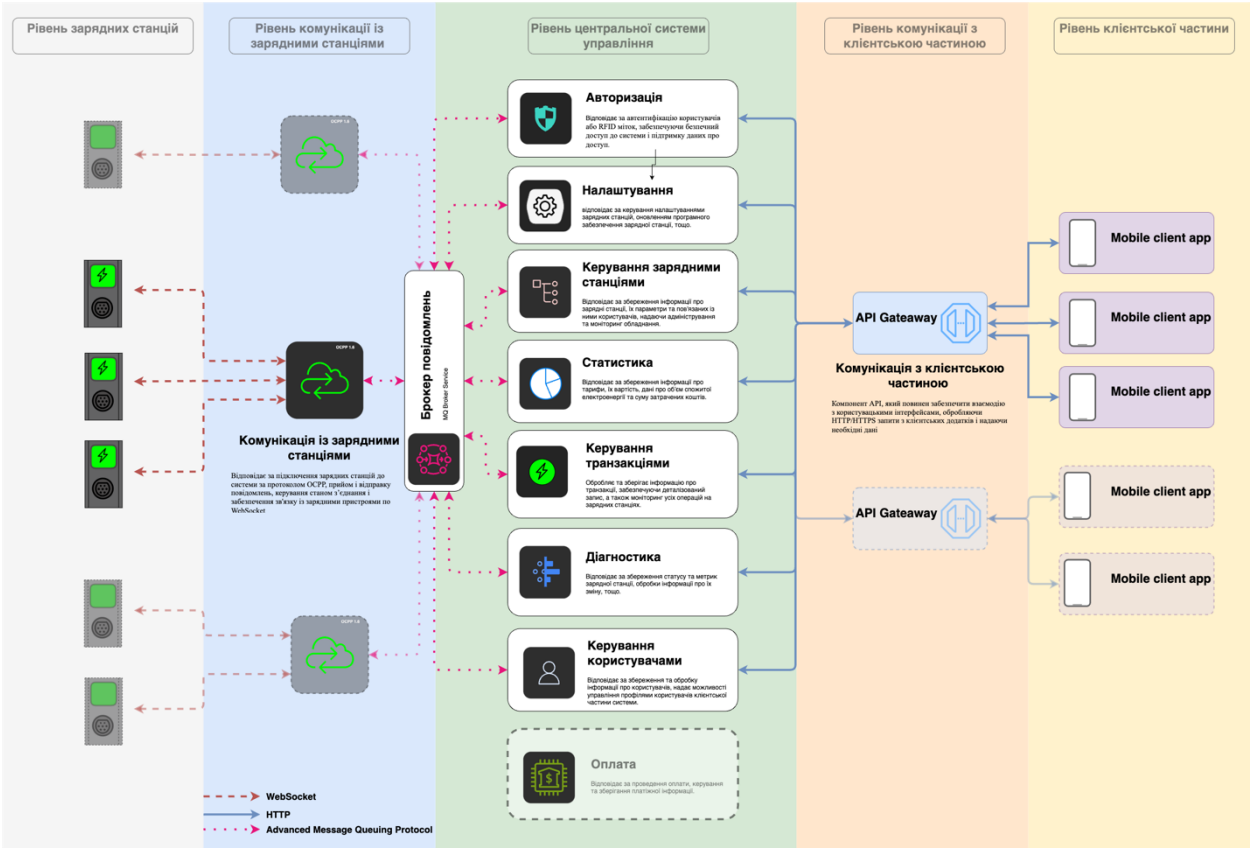

Рис. 3. Схематичне зображення системи центрального управління зарядними станціями

Такий підхід не тільки повинен забезпечити високодоступність та відмовостійкість системи, а також надати можливість для швидкого масштабування. Завдяки цьому система може швидко адаптуватися до змін, зокрема, якщо в подальшому необхідно буде реалізувати додатковий функціонал, наприклад, з оплатою певних послуг, то він буде створений та розгорнутий окремо без необхідності внесення значних змін у вже існуючі компоненти системи, їх моделі даних, тощо. Також, при збільшенні кількості зарядних станцій та, відповідно, навантаження на систему, є можливість додати додаткові сервіси для комунікації без значних змін у логіці інших компонентів системи.

Наступним кроком, після виокремлення компонентів системи, є вибір архітектури. Це без сумніву критично важливий етап розробки системи, оскільки саме він визначає, наскільки ефективно і з якими витратами будуть реалізовані бізнес-вимоги та функціональні можливості. Одним із ключових аспектів є час, потрібний на реалізацію та розробку архітектурного рішення, оскільки він напряму впливає на загальні затрати проекту, включаючи часові та фінансові ресурси. Монолітна і мікросервісна архітектури пропонують різні підходи, кожен із яких має свої переваги та недоліки, що можуть суттєво вплинути на кінцеву продуктивність, масштабованість та витрати на обслуговування системи.

Монолітна архітектура є привабливою з точки зору швидкої розробки і простоти в управлінні на ранніх етапах проекту. Всі компоненти системи інтегровані в один додаток, що дозволяє швидко розгортати оновлення та спрощує процес тестування. Проте, моноліт стає громіздким у довгостроковій перспективі. Кожна зміна потребує повторного тестування всього додатку, і навіть дрібні зміни можуть викликати небажані побічні ефекти в інших частинах системи. Також масштабування монолітної архітектури обмежується вертикальним зростанням (нарощуванням обчислювальних потужностей сервера), що може значно підвищити затрати при необхідності розширення.

Мікросервісна архітектура, з іншого боку, розділяє систему на незалежні сервіси, кожен з яких виконує свою специфічну функцію і може масштабуватися окремо від інших. Це забезпечує високу гнучкість і дозволяє легше підтримувати систему, оскільки кожен сервіс може розроблятись,

тестуватись і розгортатись незалежно. Для прикладу, центральна система управління в контексті мікросервісної архітектури матиме вигляд відображений на схематичному зображенні рисунку 4.

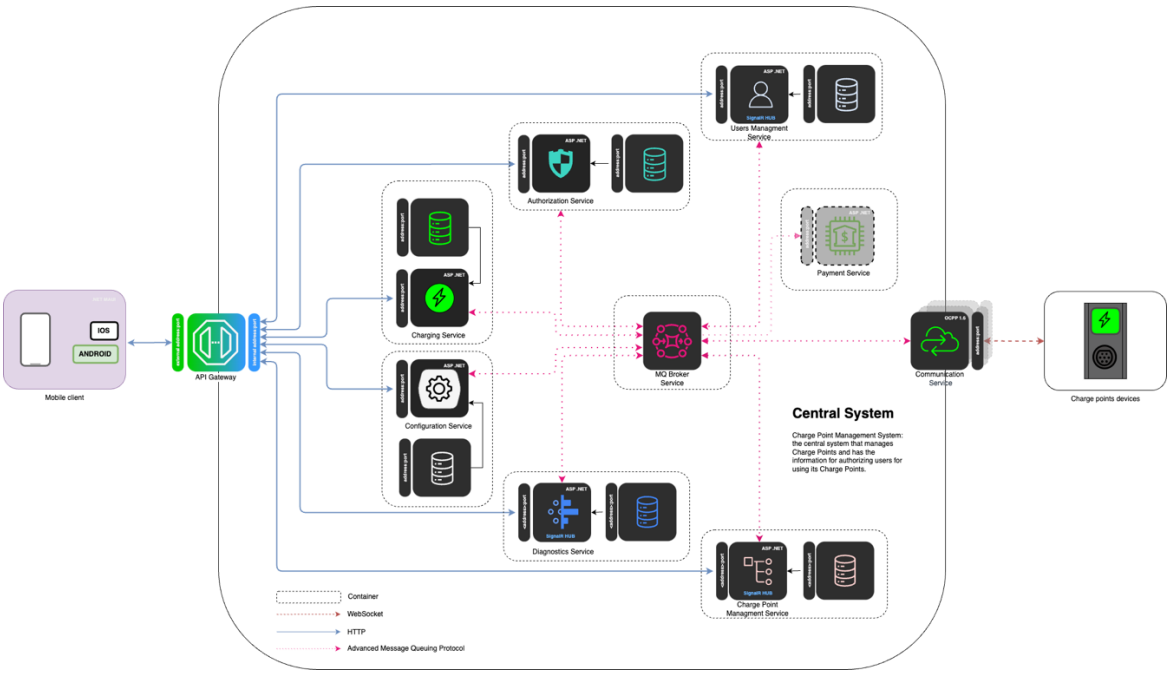

Рис. 4. Схематичне зображення мікросервісної архітектури

Слід зауважити, що поділ системи на окремі компоненти відбувається не тільки на рівні бізнес-логіки, а й на рівні розділення даних. Розділення даних є ключовим принципом побудови мікросервісної архітектури. Як зображено на рисунку 5, такий підхід дозволяє кожному компоненту системи зберігати інформацію в окремих базах даних, організованих відповідно до їх специфіки та потреб.

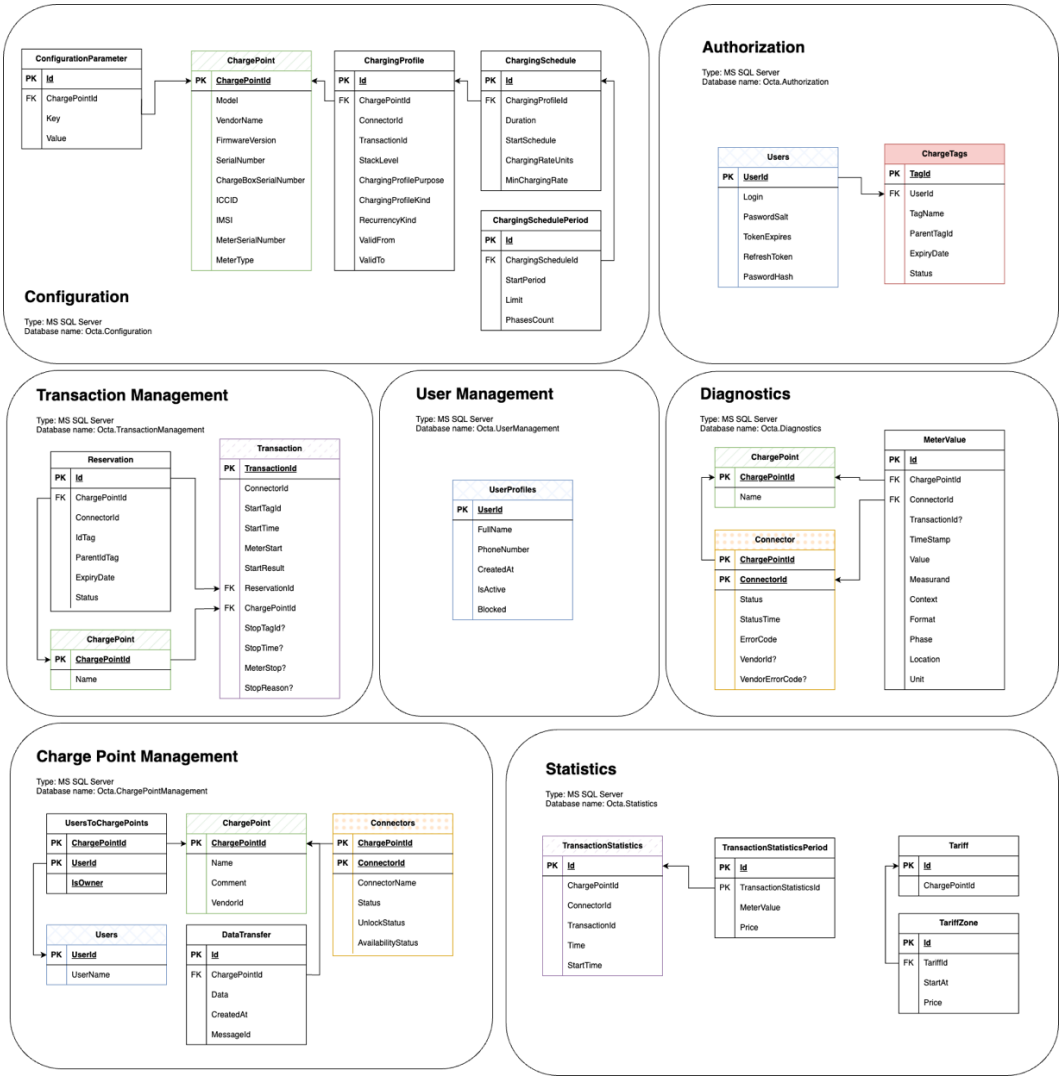

Рис. 5. Схематичне зображення розділення даних

Такий підхід забезпечує високу гнучкість та ефективність при обробці інформації, оскільки кожен модуль має власний «bounded context» – обмежений контекст, в якому визначено його бізнес-логіку, модель даних та операції над ними. Як зображено на рисунку 6, в межах «bounded context» кожен модуль відповідає за свою специфічну функціональність і використовує власну модель даних, яка не залежить від інших модулів. Це допомагає уникати конфліктів при змінах даних та забезпечує чітке розмежування відповідальностей.

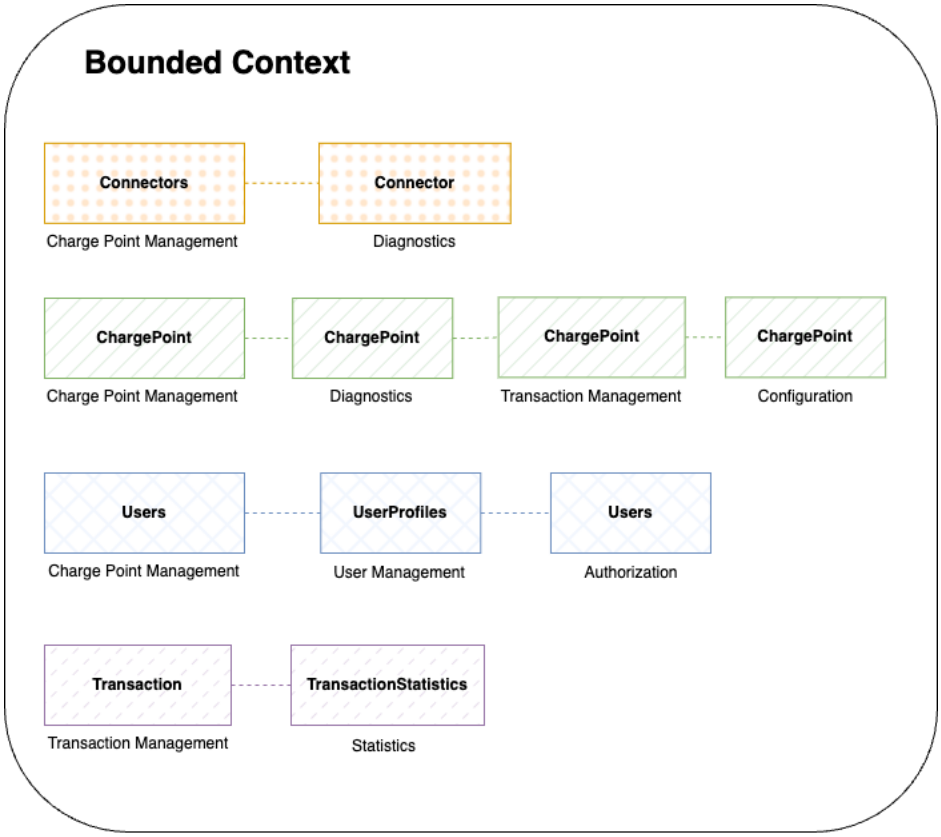

Рис. 6. Обмежений контекст даних системи

В рамках мікросервісної архітектури можна використовувати різні типи баз даних для кожного модуля, щоб оптимізувати їх під специфічні операції. Наприклад, модуль для збереження даних про транзакції може використовувати реляційну базу даних (наприклад, SQL Server) для забезпечення консистентності, тоді як модуль для збору метрик та діагностичних даних може використовувати нереляційне сховище, яке забезпечує швидке збереження великих обсягів інформації. Такий підхід дозволяє обирати найбільш ефективні рішення для кожного модуля, підтримуючи при цьому загальну продуктивність і масштабованість системи.

Однак, архітектурний підхід у вигляді мікросервісів вимагає більш складної інфраструктури, включаючи налаштування міжсервісної комунікації, управління даними та забезпечення безпеки. Це значно збільшує витрати часу на розробку і може вимагати висококваліфікованих фахівців, що в свою чергу підвищує фінансові затрати та збільшує ризики. Крім того, мікросервісна архітектура значно збільшує складність тестування та налагодження комунікацій між сервісами, що додатково може призвести до створення додаткових ризиків, зокрема, затримок у розгортанні нових версій.

Враховуючи вищеописане, більш оптимальним варіантом являється модульний моноліт, що поєднує в собі переваги обох архітектурних підходів та мінімізує їх недоліки. Він дозволяє створювати окремі модулі за аналогією описаних вище сервісів, з чітко визначеними функціями, кожен із яких управляє своїм набором даних та бізнес-логікою, але розгортається як один додаток. Це знижує складність у розгортанні, налаштуванні та підтримці, а також та полегшує тестування, оскільки всі модулі працюють у спільному контексті. Такий підхід зменшує витрати на розробку та дозволяє ефективніше керувати змінами, оскільки можна змінювати функціонал окремих модулів без впливу на інші частини системи.

Важливою перевагою модульного моноліту є можливість масштабування окремих частин системи без значних змін в архітектурі. За потреби масштабування, будь-який модуль можна винести в окремий сервіс, зберігаючи при цьому його автономність та зв'язок з іншими компонентами через стандартизовані протоколи, такі як HTTP/HTTPS, WebSocket або AMQP. Це дозволяє масштабувати систему поступово, адаптуючи її під зростаючі навантаження та нові бізнес-вимоги, не переписуючи всю архітектуру [12].

Одним з найбільш критичних компонентів для забезпечення відмовостійкості та високої доступності є комунікація із зарядними станціями. Оскільки цей компонент відповідає за обробку

великого обсягу вхідних та вихідних повідомлень та підтримує зв'язок у реальному часі, саме його доцільно винести в окремий сервіс. Це дозволить розвантажити основну систему, забезпечуючи стійкість і незалежну масштабованість компоненту комунікації. Окремий комунікаційний сервіс зможе безперебійно обробляти запити від зарядних станцій, підтримувати стабільне підключення через WebSocket і ефективно управляти потоками даних, що підвищить загальну надійність системи та її готовність до подальшого зростання.

Таким чином, архітектуру даної системи слід будувати на основі модульного моноліту, що включає кілька автономних модулів у рамках сервісу керування (Management Service): авторизація, керування зарядними станціями, налаштування, діагностика, статистика, керування транзакціями та користувачами. Кожен модуль, для підвищення продуктивності, має власний «bounded context», дозволяючи їм зберігати інформацію у окремих базах даних, а за необхідності – у різних типах баз даних. Комунікація між модулями та зовнішніми компонентами здійснюється через AMQP-брокер повідомлень, що забезпечує асинхронний обмін інформацією. Для забезпечення надійного зв'язку з зарядними станціями створено окремий сервіс комунікації (Communication Service), який функціонує як WebSocket-сервер та підтримує зв'язок із зарядними станціями через протокол OCPP. Така архітектура забезпечує високу доступність системи та, за необхідності, – дозволяє легко масштабувати окремі модулі [7]. Нижче представлені дві схеми, що ілюструють компоненти системи високого рівня та їхню взаємодію (рисунок 7), а також інфраструктуру сервісів, необхідних для її функціонування (рисунок 8).

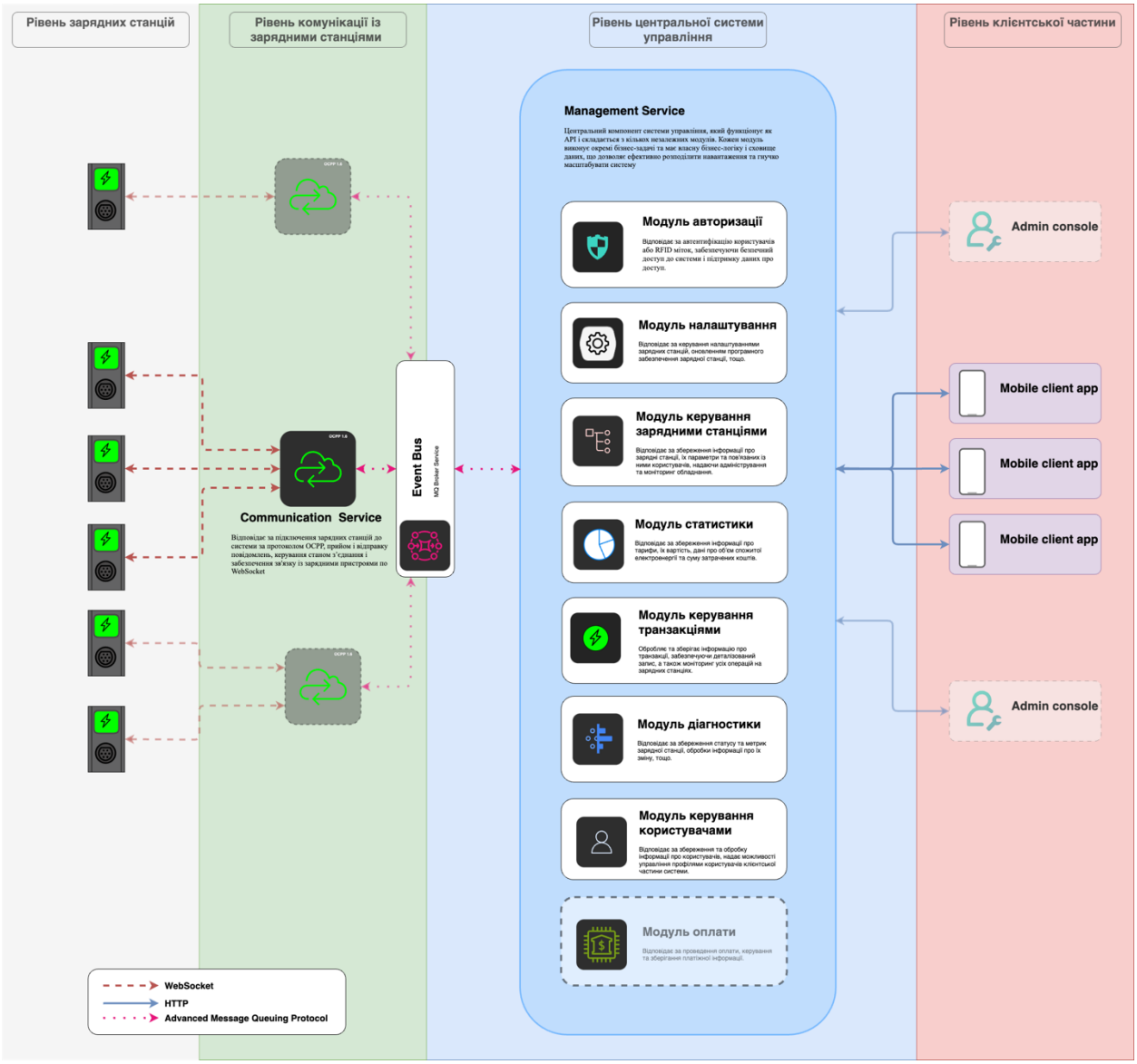

Рис. 7. Схема компонентів системи високого рівня та їхня взаємодія

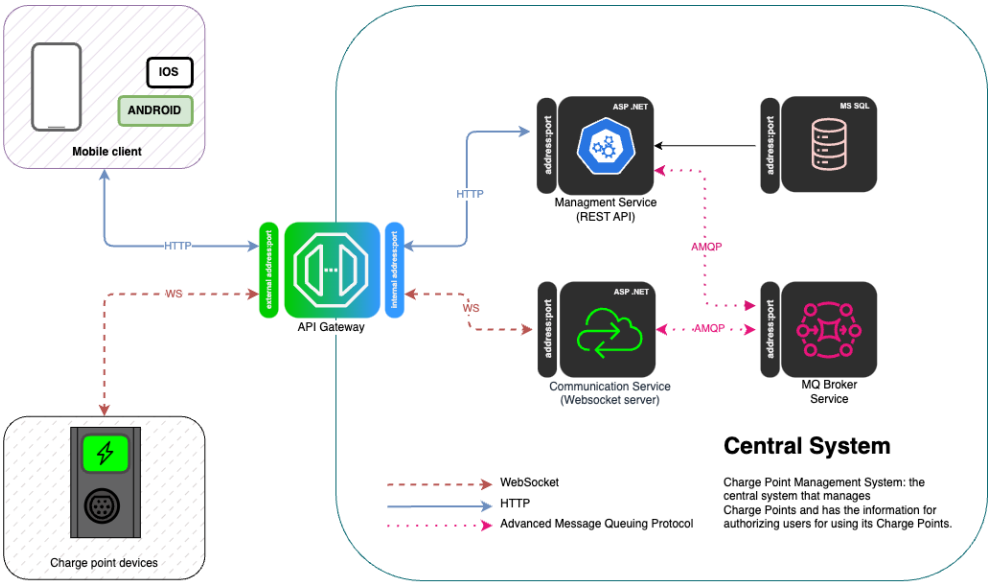

Рис. 8. Схема інфраструктури сервісів

Розгортання інфраструктури системи для управління зарядними станціями виконується за допомогою контейнеризації Docker та оркестрації Kubernetes, що забезпечує гнучкість, масштабованість та стабільність [8]. Docker дозволяє упакувати кожен компонент системи, включно із WebSocket-сервером для комунікації зі станціями, в окремий контейнер з усіма необхідними залежностями. Це забезпечує однакове середовище для розробки, тестування та виробництва, що мінімізує ризики несумісності між середовищами та полегшує управління компонентами. На схематичному зображені рисунку 9 зображено інфраструктуру компонентів центральної системи управління, розгорнутих за допомогою Docker [9].

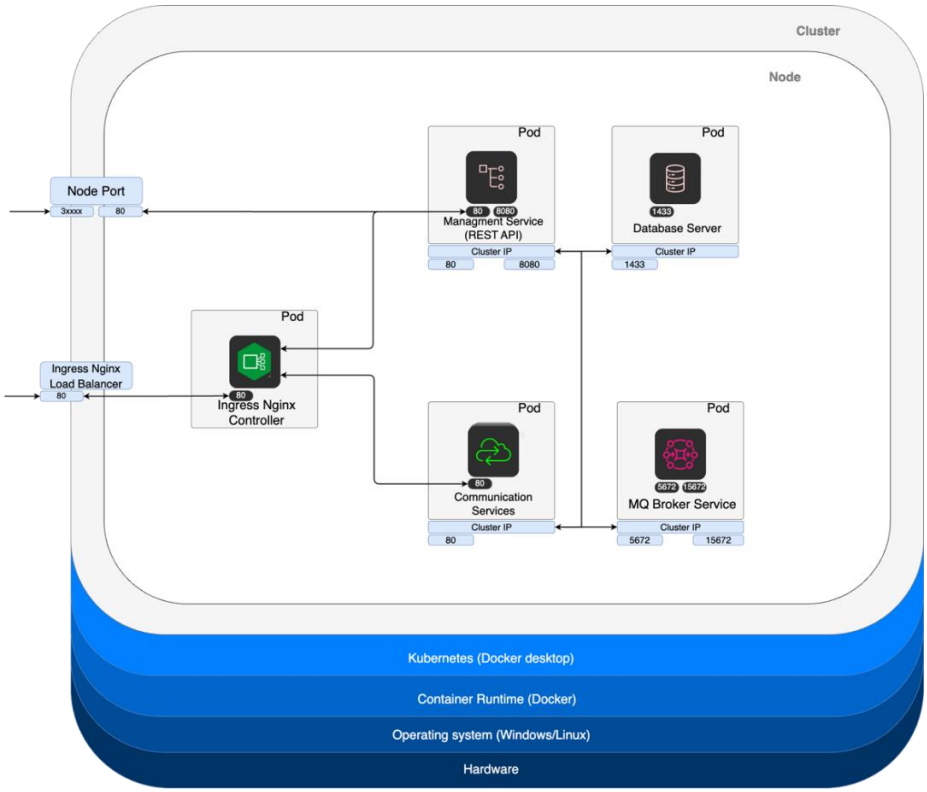

Рис. 9. Схема компонентів системи та їх взаємодія в Docker контейнері

Kubernetes є основним інструментом для оркестрації контейнерів у системі, що дозволяє автоматизувати процеси масштабування, відновлення після збоїв та управління життєвим циклом контейнерів. Kubernetes кластери дозволяють легко масштабувати окремі модулі системи за необхідності, розподіляючи навантаження на основі запитів від зарядних станцій та користувачів. Крім того, оркестрація Kubernetes підтримує балансування навантаження між контейнерами та управління збереженням стану, що важливо для баз даних, прив'язаних до окремих модулів [10].

Для зберігання даних Kubernetes підтримує налаштування обсягів зберігання (Volumes) та інтеграцію з різними типами сховищ, що дозволяє гнучко вибудовувати архітектуру з урахуванням вимог до збереження даних [11-13]. Це особливо корисно у випадках використання баз даних із різними моделями даних, що зберігаються в окремих контейнерах. Завдяки Kubernetes можна встановити політику високої доступності, яка забезпечить автоматичний перезапуск контейнерів у разі збою та розміщення контейнерів на різних вузлах для зменшення ризику відмов.

Використання Docker та Kubernetes значно спрощує процес оновлення системи, оскільки можна розгортати нові версії контейнерів без зупинки роботи всієї системи, оновлюючи лише ті частини, які були змінені. Це забезпечує безперервну роботу системи та мінімізує час простою під час оновлень [14]. Такий підхід, поєднаний із продуманою архітектурою модульного моноліту, дозволяє ефективно управляти інфраструктурою та гнучко масштабувати систему у відповідь на зростання навантаження [15].

Висновки та перспективи подальшого дослідження. Розроблена архітектура системи управління зарядними станціями з використанням модульного моноліту являється досить ефективним рішенням, що дозволяє забезпечити високу доступність, відмовостійкість та можливість для масштабування. Основними перевагами цього архітектурного підходу є його простота в розробці та підтримці порівняно з мікросервісною архітектурою, а також значно більша гнучкість у подальшому масштабуванні системи порівняно зі стандартними монолітними додатками. Зокрема, використання виділеного сервісу для комунікації забезпечує надійність та стабільність роботи зарядних станцій, незалежно від інших модулів системи, а також дозволяє оперативно обробляти запити від станцій.

Архітектура модульного моноліту дозволяє розробляти та підтримувати окремі модулі системи, такі як керування зарядними станціями, авторизація, статистика тощо, як незалежні компоненти, що мають свої контексти та окремі бази даних, за необхідності навіть з різними типами баз даних. Такий підхід дозволяє не лише підвищити продуктивність системи, але й спростити її масштабування в майбутньому, коли будь-який з модулів можна буде винести в окремий сервіс без значних змін у базовій архітектурі.

Перспективи подальших досліджень можуть включати вдосконалення механізмів обміну даними між компонентами системи для підвищення її продуктивності, дослідження нових підходів до обробки великих обсягів даних, наприклад, у модулі статистики, та інтеграцію з платіжними системами. Крім того, можливим напрямом є вивчення варіантів оптимізації комунікаційного сервісу для забезпечення кращої підтримки різних версій протоколу OCPP, що дозволить розширити сумісність з ширшим спектром зарядних станцій.

Таким чином, запропонована архітектура модульного моноліту є оптимальним рішенням для систем управління зарядними станціями, забезпечуючи баланс між гнучкістю, продуктивністю та легкістю підтримки, що робить її перспективною для подальших досліджень та вдосконалення в умовах зростаючої популярності електромобільних технологій.

Список бібліографічного опису

1. Парамуд Я. С., Рак Т. Є., Торський М. В. Принципи моніторингу та керування у мережі зарядних станцій електричних автомобілів, 2020.
2. Плесканка Н., Плесканка М., Слободзян Т., Марко Б. Аналіз ефективності використання мікросервісів при розробці Web додатків, 2024.
3. Микулич О. Архітектура модульного моноліту у Vue-застосунку, 17.05.2024. Retrieved 12.09.2024. URL: https://dou.ua/forums/topic/48721/.
4. M. Jovanović. «What Is a Modular Monolith?». Retrieved 18.10.2024. URL: https://www.milanjovanovic.tech/blog/what-is-a-modular-monolith.
5. C. Richardson. Microservices patterns. With examples in Java. Manning Publications Co., ShelterIsland, NY, USA, 2019.
6. .NET Microservices: Architecture for Containerized .NET Applications, 22.03.2023. Retrieved 10.09.2024. URL: https://learn.microsoft.com/uk-ua/dotnet/architecture/microservices/.

7. Cesar de la Torre, Janine Patrick, Bob Russell. Containerized Docker Applications Lifecycle with Microsoft Platform and Tools. Microsoft Developer Division, Redmond, Washington, USA, 2022.
8. Newman, S. Building microservices. O'Reilly Media, Inc, 2021.
9. Richards, M. Software architecture patterns, 2nd edition. O'Reilly Media, Inc, 2022.
10. Tilkov, S. (2015). Don't start with a monolith when your goal is microservices architecture. Retrieved 05.09.2024. URL: https://www.martinfowler.com/articles/dont-start-monolith.html
11. Google Trends. (n.d.). Explore - Google Trends. Retrieved 05.09.2024. URL: https://trends.google.com/trends/explore?cat=1227&date=2013-01-01%202023-10-15&q=microservices,monolith,modular%20monolith&hl=en
12. Kalske, M., Mäkitalo, N. and Mikkonen, T., 2018. Challenges when moving from monolith to microservice architecture. In Current Trends in Web Engineering: ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, practi-O-web, NLPIT, SoWeMine, Rome, Italy, June 5-8, 2017, Revised Selected Papers 17 (pp. 32-47). Springer International Publishing.DOI: https://doi.org/10.1007/978-3-319-74433-9 3
13. S. Smith. Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure. Microsoft Developer Division, Redmond, Washington, USA, 2023.
14. M. Jovanović. Monolith to Microservices: How a Modular Monolith Helps. Retrieved 18.10.2024. URL: https://www.milanjovanovic.tech/blog/monolith-to-microservices-how-a-modular-monolith-helps .
15. Kamil Grzybek. Modular Monolith: A Primer, 2019. Retrieved 19.10.2024. URL: https://www.kamilgrzybek.com/blog/posts/modular-monolith-primer.

References

1. Paramud, Y. S., Rak, T. E., Torskyi, M. V. Principles of Monitoring and Management in Electric Vehicle Charging Network, 2020.
2. Pleskanka, N., Pleskanka, M., Slobodzyan, T., Marko, B. Analyzing the Efficiency of Microservices in Web Application Development, 2024.
3. Микулич О. Архітектура модульного моноліту у Vue-застосунку, 17.05.2024. Retrieved 12.09.2024. URL: https://dou.ua/forums/topic/48721/.
4. M. Jovanović. «What Is a Modular Monolith?». Retrieved 18.10.2024. URL: https://www.milanjovanovic.tech/blog/what-is-a-modular-monolith.
5. C. Richardson. Microservices patterns. With examples in Java. Manning Publications Co., ShelterIsland, NY, USA, 2019.
6. .NET Microservices: Architecture for Containerized .NET Applications, 22.03.2023. Retrieved 10.09.2024. URL: https://learn.microsoft.com/uk-ua/dotnet/architecture/microservices/.
7. Cesar de la Torre, Janine Patrick, Bob Russell. Containerized Docker Applications Lifecycle with Microsoft Platform and Tools. Microsoft Developer Division, Redmond, Washington, USA, 2022.
8. Newman, S. Building microservices. O'Reilly Media, Inc, 2021.
9. Richards, M. Software architecture patterns, 2nd edition. O'Reilly Media, Inc, 2022.
10. Tilkov, S. (2015). Don't start with a monolith when your goal is microservices architecture. Retrieved 05.09.2024. URL: https://www.martinfowler.com/articles/dont-start-monolith.html
11. Google Trends. (n.d.). Explore - Google Trends. Retrieved 05.09.2024. URL: https://trends.google.com/trends/explore?cat=1227&date=2013-01-01%202023-10-15&q=microservices,monolith,modular%20monolith&hl=en
12. Kalske, M., Mäkitalo, N. and Mikkonen, T., 2018. Challenges when moving from monolith to microservice architecture. In Current Trends in Web Engineering: ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, practi-O-web, NLPIT, SoWeMine, Rome, Italy, June 5-8, 2017, Revised Selected Papers 17 (pp. 32-47). Springer International Publishing.DOI: https://doi.org/10.1007/978-3-319-74433-9 3
13. S. Smith. Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure. Microsoft Developer Division, Redmond, Washington, USA, 2023.
14. M. Jovanović. Monolith to Microservices: How a Modular Monolith Helps. Retrieved 18.10.2024. URL: https://www.milanjovanovic.tech/blog/monolith-to-microservices-how-a-modular-monolith-helps .
15. Kamil Grzybek. Modular Monolith: A Primer, 2019. Retrieved 19.10.2024. URL: https://www.kamilgrzybek.com/blog/posts/modular-monolith-primer.