

УДК 004.43

Предко В.Ю.¹, Киричек Г.Г.²

¹ студ. гр. КНТ-516 НУ «Запорізька політехніка»

² канд. техн. наук, доц. НУ «Запорізька політехніка»

МІКРОСЕРВІСИ ПРОТИ МОНОЛІТІВ

Монолітний підхід є найстарішою моделлю проектування програмного забезпечення (ПЗ). В рамках даної архітектури веб-додатки не мають складної структури: сервер зберігає бізнес-логіку, а база даних (БД) – необхідні для роботи сервера дані. Ця архітектура формує додаток у вигляді пов'язаного модуля, де всі компоненти працюють разом, розділяючи спільну пам'ять і ресурси (рис.1). Подібні програми не є складними в розробці і мають низку вартість на ранніх етапах розробки. Однак, при допущенні помилок, маємо проблеми які складно виправити після розгортання продукту. При цьому таких помилок може накопичитися дуже багато, тому підтримувати систему монолітної архітектури в довгостроковій перспективі складно і дорого, іноді простіше зробити «hotfix», ніж розібратися як працює частина програми. Тому проблеми з підтримкою монолітних додатків можна пояснити тим, що в майбутньому весь список функціоналу системи буде потребувати модернізації та внесення змін до вже існуючого коду. Відсутність масштабованості в монолітній архітектурі, коли всі складові системи розташовані в одній точці, веде до того, що потужність цієї точки повинна бути придатною для підтримки в робочому стані сервера і бази даних [1].

Мета роботи – дослідження можливостей масштабування та впровадження автономності функціональних команд і модулів програмного забезпечення та створення архітектури високого ступеня деталізації.

Об'єкт дослідження – процес реалізації мікросервісної архітектури.

Мікросервісна архітектура – підхід до створення додатка, що виключає використання єдиної та монолітної структури. Тобто замість того, щоб виконувати всі контексти додатка на сервері за допомогою внутрішніх процесів взаємодії, використовується декілька додатків, кожен з яких відповідає своєму обмеженому контексту [2].

Сервіси є окремими цілком самодостатніми модулями, які мають власну апаратну базу та розташовані на окремому сервері. Також вони використовують власну базу даних, можуть розміщуватися на окремих апаратних пристроях та, навіть, якщо і віртуально, взаємодія між сервісами здійснюється асинхронно, що надає як певні переваги так і недоліки (рис.1) [1].

Недолік монолітної архітектури у тому, що її складно швидко змінювати і адаптувати в ній великі сервіси. Нескінченно великий додаток з нескінченною функціональністю монолітним бути не може. Для великих додатків корпоративного масштабу використання мікросервісної архітектури є оптимальним рішенням, яке дозволяє швидко реагувати на зміни, що відбуваються на ринку [3].

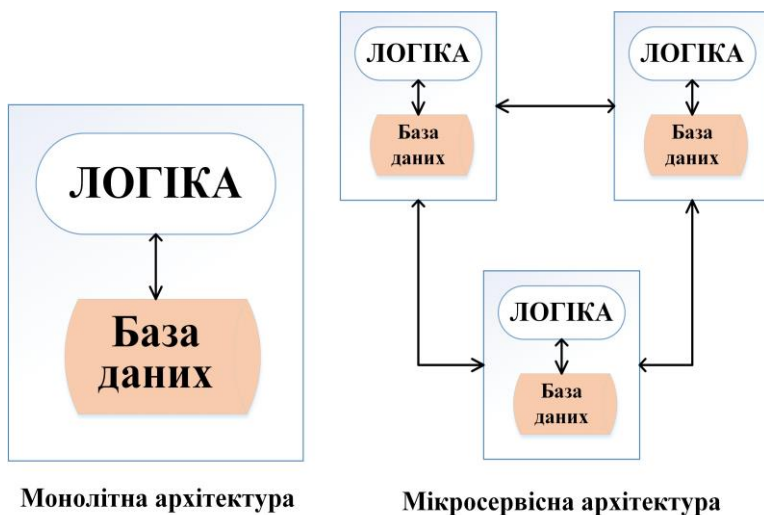


Рисунок 1 – Схема монолітної та мікросервісної архітектури

В процесі реалізації окремі функціональні модулі програм запускаються на різних серверах або різними процесами. Тому, в процесі розробки програмного забезпечення, необхідно змінювати стратегію самої розробки та розділяти додатки на окремі модулі, які реалізують конкретний функціонал або вирішують окремі завдання. Такими модулями є сервіси, які відповідають за конкретний функціонал. Наприклад, завантаження файлів виконує окремий сервіс, а інші виконують свої дії нічого не знаючи про завантаження файлів.

Мікросервіси є окремими монолітними програмами, функції яких

обмежені власними завданнями, що не впливають на стан інших завдань. Програмний продукт з платформою мікросервісної архітектури може включати будь-яку кількість нових мікросервісів, розширюючи свою функціональність [3]. Простота та швидкість впровадження мікросервісів зумовлена тим, що необхідні функції знаходяться в межах одного модуля, тому при виникненні помилок їх відразу ж відслідковують при тестуванні та ліквідують. При внесенні змін до одного сервісу, інші функціональні модулі системи залишаються незмінними: досить просто виключити непотрібний модуль без впливу на весь додаток або додати новий. Це спрощує реалізацію систем в цілому та проведення їх тестування.

Практичне застосування мікросервісної архітектури, більш ніж моноліт забезпечує надійний захист корпоративних даних та даних користувачів. Це пов'язано з тим, що додатки не мають єдиної бази даних, яка б зберігала всю важливу інформацію, бо для кожної частини програми використовується власний простір інформаційних даних.

Зазначимо, що використання одразу декількох БД в одному модулі є допустимим і широко використовується в сучасних розробках. Наприклад, для зберігання конфіденційної інформації користувача, використовується не менше двох БД – одна для публічних даних (ім'я, рік народження, адреса електронної пошти, тощо), а друга для приватних даних (логін, хеш паролю, секретне питання, тощо). При цьому до них може підключитися будь-яка система управління базами даних.

Одна з головних переваг мікросервісної архітектури полягає в тому, що кожен модуль є окремою структурою і може, для реалізації своєї функціональності, використовувати будь-яку програмну платформу. Це означає, що його можна реалізувати будь-якою мовою програмування, тією якою на даний час є найбільш ефективною для реалізації необхідного функціоналу. Однак, на практиці, проблемою є нестача фахівців, які можуть підтримувати весь спектр технологій, що використовуються.

Беручи до уваги всі переваги та недоліки даних архітектур, перебудова або перенесення сервісу з моноліту на мікросервісну архітектуру не завжди є доречним. Якщо програмне забезпечення не передбачає подальшого розширення або воно вирішує незначне коло завдань, немає сенсу вносити в ПЗ будь-які зміни. Але, якщо додаток з великою кількістю об'ємних та різноманітних завдань вже працює з підтримкою монолітної архітектури, то маючи всі необхідні ресурси для капітального оновлення системи, потрібно не доопрацьовувати новий функціонал у межах старого, а повністю його оновити змінюючи підхід та впроваджуючи мікросервісну архітектуру.

В роботі розглянуто дві моделі проектування програмного забезпечення та реалізації функціональних команд і модулів – мікросервісну та монолітну. Визначені головні переваги мікросервісної архітектури перед монолітною та

зроблені висновки у яких випадках доречно переходити на архітектуру з високим ступенем деталізації, а у яких ні.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Богданенко Д. А. Подходы к архитектурному проектированию веб-приложений / Д. А. Богданенко [Электронный ресурс]. – Режим доступа: <https://moluch.ru/archive/195/48609/>.

2. Осипов Д.Б. Проектирование программного обеспечения с помощью микросервисной архитектуры / Д.Б. Осипов [Электронный ресурс]. – Режим доступа: <https://cyberleninka.ru/article/n/proekti-rovanie-programmnogo-obespecheniya-s-pomoschyu-mikroservisnoy-arhitektury>.

3. Ньюмен С. Создание микросервисов. / С. Ньюмен. – СПб: Издательский дом "Питер", 2016. – 307с.