УДК 004.415

М. Д. Такташев, Г. Й. Михальчук

Дніпровський національний університет імені Олеся Гончара

РОЗРОБКА СЕРВЕРНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

Проведено порівняльний аналіз існуючих методів проектування і реалізації архітектури серверного програмного забезпечення. Спроектовано і реалізовано систему мікросервісів, що забезпечує функціонування Telegram-бота. Реалізовано можливість оновлення функціоналу програми без необхідності призупиняти роботу сервера на час оновлення.

Ключові слова: *серверне програмне забезпечення, мікросервісна архітектура, Telegram-бот, черга повідомлень, брокер повідомлень, взаємодія між процесами, модульність.*

Проведён сравнительный анализ существующих методов проектирования и реализации серверного программного обеспечения. Спроектирована и реализована система микросервисов, обеспечивающая работу Telegram-бота. Реализована возможность обновления функционала программы без необходимости приостанавливать работу сервера на время обновления.

Ключевые слова: *серверное программное обеспечение, микросервисная архитектура, Telegram-бот, очередь сообщений, брокер сообщений, взаимодействие между процессами, модульность.*

In the world of modern commercial development, the creation of a product does not complete after release. Instead, the business seeks to expand the functionality to meet the growing demands of users and to attract new customers. Making changes to poorly structured and highly coupled code is time-consuming and potentially insecure. This can be clear from the fact that along with the complexity of the code the risk of new errors increases. And that, in turn, leads to additional time and money expenses and slow down the speed of delivery of new functionality to users, which reduces competitiveness and can cause losses for business or destruct it in a competitive market. That is why it is so essential to choose an appropriate architecture of the system and its components according to the project specification and business needs. Nowadays, most of the created server software can be divided into two groups by its high-level architecture: monolithic applications and microservice-

based applications. A monolithic application is an application that could be considered as a single process which implements all the features of the system. A microservice-based application is a system of loosely coupled applications (microservices), each of which implements a separate business or functional opportunity and can interact with other components of the system using their public interface. The purpose of the work is to investigate existing approaches in server-side architecture creation and to implement loosely coupled system parts of which can be reused in other projects with minimum changes. In the process of work the investigation of existing approaches of the server-side architecture creation and its comparison were made. Furthermore, for the purpose of personalized content delivery the software was created using microservice architecture. The project is implemented using .NET Core 3.1 platform, C# programming language, and Visual Studio 2019 as a development environment. For data storage, SQLite is used. The project consists of 7 separate server-side applications. Each of them can be run on Windows 10, Linux, or macOS operation systems.

Keywords: *server software, design, Telegram bot, microservice architecture, message queue, message broker, interaction between processes, modularity.*

Вступ. Архітектура програмного забезпечення – це спосіб структурування програмної системи, абстракція елементів системи на певній фазі її роботи. Система може складатись з кількох рівнів абстракції і мати багато фаз роботи, кожна з яких може мати окрему архітектуру [1]. Це спосіб організації програмного коду та взаємодії окремих програмних компонентів як між собою, так і із сторонніми ресурсами.

У реаліях сучасного світу комерційної розробки створення продукту часто не завершується після його релізу (тобто його публікації, відкриття доступу для користувачів). Натомість після випуску у світ бізнес прагне розширювати функціонал, який буде задовольняти зростаючим вимогам користувачів і зможе привернути увагу нових клієнтів. Яскравим прикладом може стати компанія Facebook, яка у 2004 [2] вийшла на ринок із своєю соціальною мережею, розробленою за декілька місяців, і на сьогоднішній день кардинально змінилася, намагаючись задовольнити якомога більше потреб користувачів. Тому так важливо закласти правильну архітектуру програмного коду та компонентів системи, адже вносити зміни у погано структурований і сильно зв’язаний код важко та небезпечно, адже разом із складністю коду підвищується і ризик виникнення нових програмних помилок, що призводить до додаткових витрат часу і бюджету, а також зменшує швидкість доставки нового функціоналу користувачам, що призводить до зниження

конкурентоспроможності і може завдати шкоду бізнесу або навіть вбити його на конкурентному ринку.

Аналіз останніх розробок та досліджень. На сьогоднішній день більшість створеного серверного програмного забезпечення за його високорівневою архітектурою можна поділити на дві групи: монолітні додатки і системи мікросервісів. Монолітний додаток – це додаток, що являє собою єдиний процес і реалізує усі можливості системи. Система мікросервісів – це сукупність слабопозв'язаних між собою додатків (мікросервісів), кожен з яких реалізує окрему бізнес- або функціональну можливість і може взаємодіяти з іншими компонентами системи. Кожен із підходів має свої переваги і недоліки. Обрати один з них можна на основі вимог, висунутих до створюваного програмного продукту. У рамках статті акцент зроблено на мікросервісну архітектуру.

Термін «мікросервіс» вперше було запропоновано використовувати до шаблонів архітектури програмного забезпечення у 2011 році на семінарі у Венеції [3]. Концепція мікросервісної архітектури отримала значну популярність після публікації Мартіна Фаулера та Джеймса Льюіса [4]. Докладний аналіз причин, що привели до розповсюдження мікросервісів, та перспективи їх застосування наведено в [3]. Компанія Netflix стала однією з перших великих компаній, що почала публічно говорити про застосування мікросервісної архітектури у власних програмних продуктах. Завдяки її використанню, вдалося значно підвищити надійність і здатність сервісу до масштабування – впровадження нового функціоналу і збільшення пропускної здатності серверів [5]. На сьогоднішній день мікросервісна архітектура є розповсюдженим явищем серед міжнародних програмних продуктів, таких як Amazon, Twitter, Facebook.

Постановка задачі. Дослідити особливості мікросервісної архітектури серверного програмного забезпечення та застосувати її під час проектування і розробки програмної системи, що забезпечує роботу Telegram-бота, який має надавати користувачеві персоналізований контент за налаштованим розкладом.

Основний матеріал. Серед сильних сторін мікросервісної архітектури можна назвати такі:

- легкість підтримки окремих компонентів системи. Оскільки кожен сервіс реалізує лише одну бізнес-можливість, його програмний код є значно компактнішим, аніж будь-який монолітний додаток. Через це його легше «зрозуміти», «дослідити» та, у разі необхідності, виправити некоректно працюючий код;

- незалежність від технологій. Оскільки між собою сервіси взаємодіють через загальновживані протоколи обміну даними, кожен з них може буди реалізований на найбільш вдалій для його конкретних задач технології. Вони можуть використовувати різні мови програмування, різні сховища даних тощо – тобто ті технології, що найкращим чином підходять для вирішення конкретної задачі;
- стійкість до критичних помилок. При належному рівні ізоляції відмова окремого сервісу не стане причиною призупинення роботи системи та навіть збоїв у роботі інших функцій системи;
- потенційно високий рівень повторного використання коду. Наприклад, написавши один раз сервіс для розсилки електронної пошти, його можна використовувати і на інших проектах без будь-яких змін у його кодовій базі.

Серед слабких сторін мікросервісної архітектури можна назвати такі:

- складність первинного налаштування. Первинне налаштування базової структури проекту може потребувати досить багато часу;
- відносна складність тестування. Сервіси є складовими частинами системи. Під час тестування створеного сервісу необхідно не лише перевірити правильність реалізованого у ньому функціоналу, а й коректність його взаємодії із іншими компонентами системи;
- операційні витрати. Виклик функції у межах одного процесу відбувається значно швидше, ніж виклик віддаленої функції із іншого сервісу, що призводить до втрат у швидкодії;
- інфраструктурні витрати. Необхідно забезпечити комунікацію між сервісами, що потребує написання додаткового інфраструктурного коду.

Для практичного дослідження мікросервісної архітектури було спроектовано і реалізовано систему, що забезпечує роботу Telegram-бота, за допомогою якого користувач може отримувати персоналізований контент за налаштованим розкладом. Головною вимогою до системи стала можливість керувати постачальниками даних, із яких користувач може обирати ті, в яких він зацікавлений, без необхідності призупиняти роботи сервера на час оновлення. У ролі постачальника даних може виступати будь-який сервіс, що надає інформацію за запитом, наприклад, сервіс, що надає прогноз погоди, останні новини або статті за інтересами користувача. Слід зазначити, що таке рішення є неможливим у рамках монолітного додатка, адже за

такого підходу увесь функціонал системи сконцентровано у єдиному процесі, і, у разі інтеграції нових можливостей, перезавантаження сервера – це вимушена необхідність, що зробить сервер неспроможним обробляти користувацькі запити протягом часу оновлення, що може стати критичним у деяких системах, таких як онлайн-банкінг. Модульний підхід мікросервісної архітектури, у свою чергу, дозволяє оновлювати окремі частини системи, не втручаючись у роботу решти сервісів. Таким чином, функціонал програмного продукту може бути оновлений без погіршення користувацького досвіду і ризику збоїв у роботі системи.

Для того щоб спроектувати серверне програмне забезпечення на базі мікросервісної архітектури, необхідно визначити область дії для кожного компонента системи. Занадто велика кількість обов'язків, покладених на сервіс, призведе до складнощів його супроводження. Тоді як замалий розмір сервісів є причиною їх несамостійності і високого рівня зв'язаності між компонентами системи. Невірно визначені області дії сервісів можуть звести до мінімуму переваги мікросервісної архітектури [6].

За результатом аналізу бізнес- і технічних можливостей системи [6] було побудовано діаграму компонентів системи та їх взаємодії (рис. 1).

Розглянемо призначення кожного компонента спроектованої системи:

- Services.TelegramBot реалізує інтеграцію із Telegram Bot API
  - може розпізнавати команди, що були відправлені боту користувачем, і реагувати на них;
  - надає публічний інтерфейс, що дозволяє відправляти текстові повідомлення користувачеві;
- Services.ApiGateway є вхідною точкою системи
  - надає доступ до веб-сторінки для налаштування профілю користувача;
  - надає публічний інтерфейс, що дозволяє адміністратору авторизуватися і керувати доступними постачальниками контенту через веб-сторінку, а також публічний інтерфейс для відправлення встановлених користувачем налаштувань сервісу Services.Account;
  - має ізольовану від інших сервісів базу даних для збереження даних адміністратора;
- Services.Account призначений для зберігання даних користувачів

- надає публічний інтерфейс для оновлення даних користувача;
- публікує повідомлення у черзі повідомлень RabbitMQ, коли налаштування користувача було оновлено;
- має ізольовану від інших сервісів базу даних для збереження даних користувачів;

- Services.Scheduler забезпечує своєчасну доставку контенту згідно із заданим розкладом
  - підписується на чергу повідомлень RabbitMQ та очікує публікацію повідомлень про оновлення налаштувань профілю користувача; у разі отримання повідомлення зберігає у власне сховище даних;
  - публікує повідомлення у черзі повідомлень RabbitMQ, коли прийшов час відправляти контент користувачеві;
  - має ізольовану від інших сервісів базу даних для збереження запланованого часу відправлення контенту користувача;
- Services.ContentBroker забезпечує постачання контенту користувачеві
  - підписується на чергу повідомлень RabbitMQ та очікує публікацію повідомлень про настання запланованого часу доставки контенту; у разі отримання повідомлення, одержує дані із необхідних джерел, форматує повідомлення для користувача і відправляє його;
  - має ізольовану від інших сервісів базу даних для збереження списку доступних постачальників контенту;
- Services.ContentProviders.Weather та Services.ContentProviders. Quote реалізують інтеграції із сторонніми API і отримують дані про погоду та випадкові цитати відомої людини.

Усі компоненти системи було створено на платформі .NET Core 3.1. Вони можуть бути запущені у середовищі Windows 10, Linux або MacOS, за умови попереднього встановлення .NET Core Runtime і .NET Core SDK.

Деякі компоненти системи потребують сховища даних. Такі бази даних є ізольованими від впливу ззовні, що забезпечує безпеку і цілісність даних.

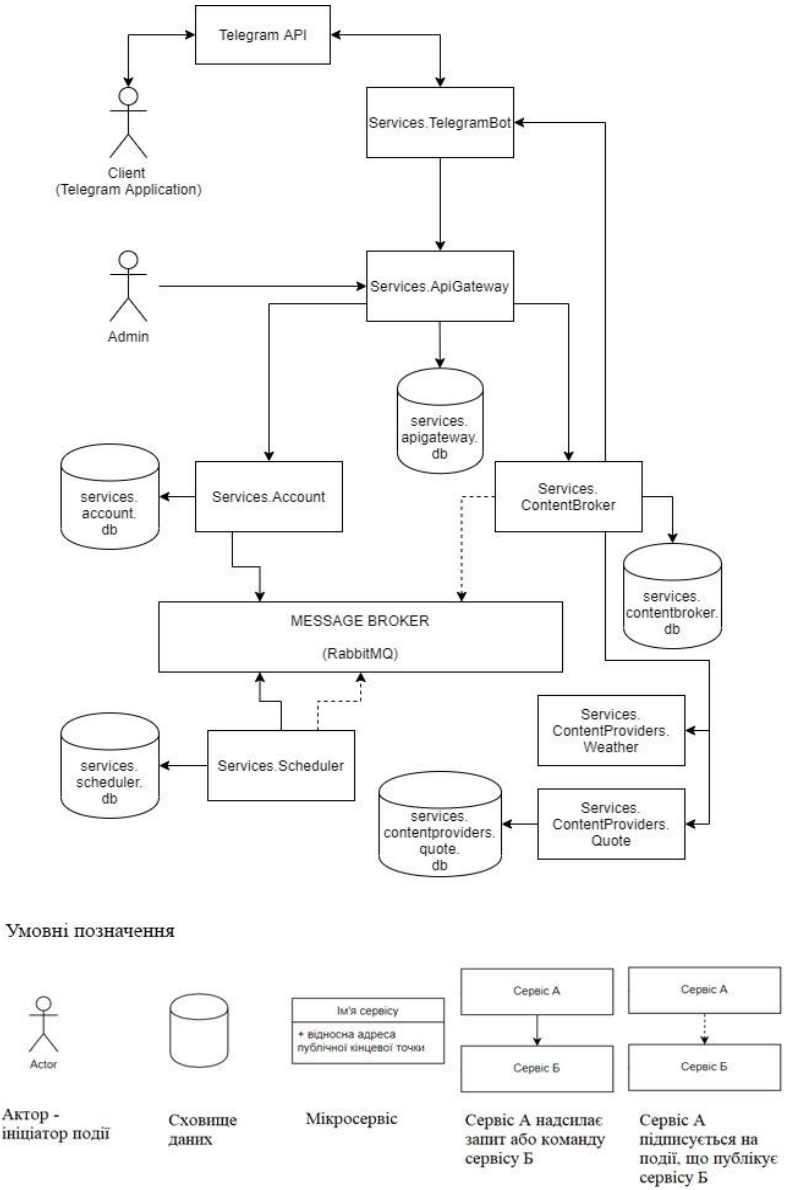

Рисунок 1 – Діаграма компонентів системи та їх взаємодії

З-поміж існуючих брокерів повідомлень було обрано RabbitMQ. Це сервер, що реалізує чергу повідомлень для асинхронної комунікації між компонентами розподіленої системи. Обмін повідомленнями відбувається за протоколом Advanced Message Queuing Protocol, що забезпечує маршрутизацію повідомлень і високий рівень ізоляції компонентів системи. Замість того щоб відправити запит до іншого сервісу, компонент системи публікує повідомлення у чергу. У той же час сервіси можуть підписуватися на оновлення черги повідомлень і обробляти опубліковану інформацію. Таким чином, сервіси прямо не залежать один від одного. Кожен із них може бути замінено іншою реалізацією, при цьому повідомлення не будуть втрачені і будуть оброблені як тільки новий сервіс підключиться до черги повідомлень.

У спроектованій системі сервіс Services.ContentBroker відповідає за постачання контенту користувачеві. Він має власну базу даних, що зберігає список постачальників контенту, і надає інтерфейс, що дозволяє керувати цим списком. Реалізовано можливість керування списком доступних постачальників контенту через веб-сторінку за умови наявності у користувача прав адміністратора.

У системі реалізовано веб-сторінку для налаштування профілю користувача. Отримати посилання на сторінку можна за допомогою команди «/settings» у Telegram-боті. Після переходу за посиланням користувач має змогу обрати контент, який хоче отримувати, і час, коли саме він хоче його отримувати. Посилання містить токен – спеціальний параметр, що ідентифікує користувача. Після п’ятнадцяти хвилин з моменту отримання посилання воно стає недійсним, що захищає налаштування користувача від впливу зловмисників у разі викрадення токену. На сторінці відображається лише актуальний список постачальників контенту. Він генерується автоматично під час створення сторінки на стороні сервера. Таким чином, у разі появи нового постачальника даних він буде відображуватися у списку і користувач зможе обрати його.

Висновки. Розглянуто особливості мікросервісної архітектури. Спроектовано і розроблено систему мікросервісів, що демонструє переваги мікросервісної архітектури над монолітною у ряді задач. Окремі компоненти створеного програмного продукту можуть бути повторно використані в інших проектах або під час адаптації даної концепції для роботи із іншими клієнтами, наприклад із мобільним додатком або веб-сайтом замість Telegram-бота.

Бібліографічні посилання

1. Eeles P. What is a software architecture? URL: https://www.ibm.com/developerworks/rational/library/feb06/eeles/index.html (дата звернення: 02.11.2020)

2. Phillips S. A brief history of Facebook. URL: https://www.theguardian.com/technology/2007/jul/25/media.newmedia (дата звернення: 03.11.2020)

3. Dragoni N., Giallorenzo S., Lafuente A. L., Mazzara M., Montesi F., Mustafin R., Safina L. Microservices: yesterday, today, and tomorrow. *Present and Ulterior Software Engineering* / ed. Mazzara M., Meyer B. 2017. P. 195–216.

4. Fowler M., Lewis J. Microservices. URL: https://martinfowler.com/articles/microservices.html (дата звернення: 02.11.2020)

5. Why You Can't Talk About Microservices Without Mentioning Netflix. URL: https://smartbear.com/blog/develop/why-you-cant-talk-about-microservices-without-ment/ (дата звернення: 05.11.2020)

6. Хорсдал К. Микросервисы на платформе .NET. СПб.: Питер, 2018. 352 с.

Надійшла до редколегії 24.11.2020.