УДК 004.415

# ПРОЕКТУВАННЯ АРХІТЕКТУРИ ІНФОРМАЦІЙНОЇ СИСТЕМИ У СЕРВІС-ОРІЄНТОВАНІЙ ПАРАДИГМІ

Голуб Д. К.
e-mail: dmytro.holub1@nure.ua
Науковий керівник – к.т.н., доц. Вишняк М. Ю.
Харківський національний університет радіоелектроніки, каф. КМІТ
м. Харків, Україна

This research discusses the process of designing a service-oriented architecture for a social network application, the problems and their solutions that arose during design and development, technologies and features of their application in a specific case. The designed service-oriented architecture includes a number of services that exchange messages with each other using message brokers using the fluffy-core library, an API Gateway service that converts messages from the user interface received using WebSockets technology and the websocket-rush-api library, allows horizontal scaling and performs the function of a security layer for business logic, a separate service for storing assets, requests to which the client part performs using the HTTP REST API.

У сучасній розробці програмного забезпечення вже добре закріпилося поняття мікросервісної архітектури [1] як модульного, ефективного та зручного методу проектування й імплементації застосунків. Суть сервіс-орієнтованого підходу полягає в поділі одного великого програмного рішення на множину невеликих сервісів, згрупованих за предметною областю, де кожен сервіс має власну базу даних, може бути реалізований різними мовами програмування та використовувати різні технології й підходи до реалізації. Мікросервісна архітектура надає розробникам низку переваг, однак потребує значної уваги під час проектування. Розглядається приклад побудови мікросервісної архітектури для соціальної мережі, що демонструє кращі практики та методологію і може виступати як "типове рішення", яке згодом можна адаптувати до інших проектів.

Запропонована архітектура застосунку визначає низку сутностей. Користувацький інтерфейс представлений React-застосунком, який безпосередньо взаємодіє з кінцевим користувачем. Застосунок за допомогою бібліотеки websocket-rush-api встановлює з’єднання з сервісом API Gateway [2].

За потреби клієнтський застосунок також виконує HTTP REST-запити до окремого сервісу picture-service, що відповідає за зберігання асетів, зокрема зображень, завантажених користувачами на платформу.

API Gateway у реальному часі трансформує користувацькі запити, отримані через публічне WebSocket-з'єднання[3], та передає їх до внутрішньої захищеної мережі системи. Обмін повідомленнями між внутрішніми сервісами здійснюється через брокер повідомлень (у даному випадку – NATS) з використанням бібліотеки fluffy-core.

Отримане повідомлення маршрутизується до одного з трьох сервісів бізнес-логіки: user-service, content-service або ads-service. У зворотному напрямку відповідь передається через fluffy-core до API Gateway, після чого за допомогою WebSocket-з'єднання повертається клієнтові.

Кожен сервіс бізнес-логіки (user-service, content-service, ads-service, picture-service) підключений до власної бази даних (або окремої бази для групи сервісів, залежно від конфігурації). Структурну схему розробленої архітектури наведено на рисунку 1.

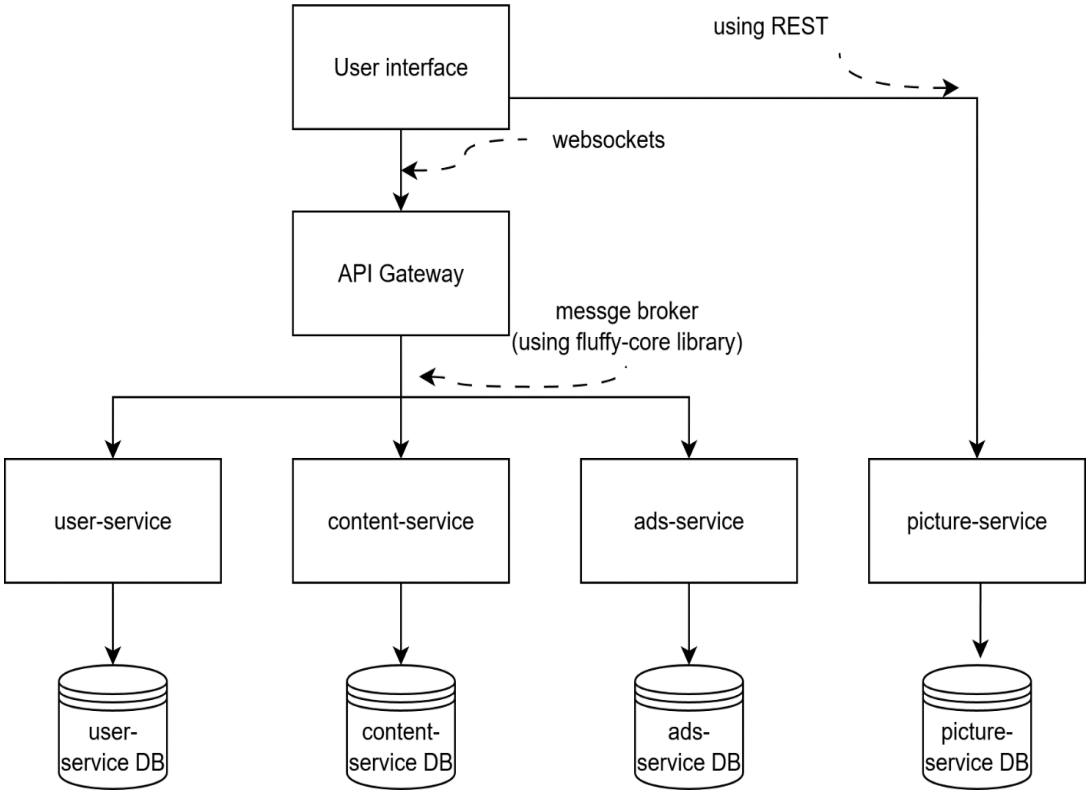


Рисунок 1. – Структурна схема розробленої архітектури

Надзвичайно важливою компонентою цієї архітектури є сервіс API Gateway. Цей сервіс виконує одразу декілька завдань. По-перше, він перевіряє, чи має користувач право виконувати конкретний запит, використовуючи чітко визначений whitelist (список дозволених операцій), що унеможливлює несанкціонований доступ до внутрішніх методів системи.

По-друге, API Gateway трансформує запити користувача та відповіді на них між різними протоколами обміну повідомленнями (WebSocket і fluffy-core).

По-третє, сервіс бере участь у розподілі навантаження між екземплярами сервісів однієї групи. Наприклад, якщо для підвищення пропускної здатності розгорнуто декілька екземплярів одного сервісу, повідомлення, передане через fluffy-core, буде опрацьоване лише одним із них. Вибір конкретного екземпляра здійснюється механізмом брокера повідомлень (у разі використання відповідної моделі підписки), що забезпечує балансування навантаження. Такий підхід дозволяє ефективно реалізовувати горизонтальне масштабування з метою підвищення продуктивності та відмовостійкості системи.

Окремої уваги заслуговує сервіс picture-service. Його основне завдання полягає у зберіганні та безпечному наданні користувачам зображень, завантажених на платформу. Користувачі здійснюють запити до цього сервісу безпосередньо через REST API, минаючи API Gateway. Такий підхід дозволяє зменшити навантаження на API Gateway, оскільки брокери повідомлень зазвичай не призначені для передавання об’єктів великого розміру. Крім того, обробка медіафайлів у межах контуру, побудованого навколо брокера повідомлень, є архітектурно недоцільною. У даній реалізації picture-service представлений як окремий мікросервіс. Проте в більш масштабних проектах його функціональність можуть виконувати спеціалізовані хмарні рішення, наприклад Amazon S3 для зберігання об’єктів та Amazon CloudFront для кешування і масштабованої доставки контенту.

Запропонований варіант реалізації сервіс-орієнтованої архітектури вирішує значну частину складнощів, пов’язаних із підтримкою мікросервісного проєкту, та може бути легко адаптований до інших предметних галузей. Такий підхід забезпечує масштабованість, підвищений рівень безпеки та модульність, що є очікуваними характеристиками мікросервісної архітектури.

***Список використаних джерел:***

1. Newman S. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. O’Reilly, 2020. 274 p.

2. Richards M., Ford N. Fundamentals of Software Architecture. O’Reilly, 2020. 383 p.

3. The WebSocket Protocol. RFC-6455. URL: https://datatracker.ietf.org/doc/html/rfc6455 (дата звернення 20.02.2026).