DOI https://doi.org/10.32782/2078-0877-2026-26-1-6
УДК 004.75:004.771

С. С. Грушко, канд. техн. наук[1,2] ORCID: 0000-0002-0064-408X
І. І. Афанасьєв[1], студент ORCID: 0009-0008-5712-9047
А. В. Тіменко[1,2], стар. викл. ORCID: 0000-0002-7871-4543
Н. А. Куликовська[1], стар. викл. ORCID: 0000-0003-4691-5102

[1] *Національний університет «Запорізька політехніка»*

[2] *Таврійський державний агротехнологічний університет імені Дмитра Моторного*
e-mail: grushko@zp.edu.ua

АРХІТЕКТУРА ІНТЕГРОВАНОЇ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ МОНІТОРИНГУ З ВИКОРИСТАННЯМ WEBSOCKET ДЛЯ ОБМІНУ ДАНИМИ В РЕАЛЬНОМУ ЧАСІ

Анотація. У статті розглянуто архітектуру інтегрованої клієнт-серверної системи моніторингу системних параметрів комп'ютерів для сервісного центру. Досліджено особливості застосування протоколу WebSocket у поєднанні з STOMP для реалізації двонаправленого обміну повідомленнями в реальному часі між клієнтами та спеціалістами. Проаналізовано переваги WebSocket порівняно з традиційним HTTP-опитуванням. Описано архітектурні рішення щодо інтеграції компонентів системи на базі Spring Boot, JavaFX та PostgreSQL. Представлено механізми забезпечення безпеки комунікації через JWT-авторизацію на рівні WebSocket-з'єднань.

Ключові слова: WebSocket, STOMP, клієнт-серверна архітектура, інтегрована система, моніторинг, Spring Boot, JavaFX, реальний час.

Постановка проблеми. Сучасні інформаційні системи для технічної підтримки та сервісного обслуговування комп'ютерної техніки стикаються з дедалі більшими вимогами до ефективності обробки звернень користувачів та швидкої діагностики проблем. В умовах зростання кількості звернень до сервісних центрів, а також ускладнення апаратного й програмного забезпечення комп'ютерів виникає потреба в автоматизованих інструментах, що забезпечують збирання, аналіз і передачу технічної інформації про стан систем користувачів.

Традиційний підхід на основі HTTP-запитів із періодичним опитуванням сервера створює надмірне навантаження на інфраструктуру й не забезпечує миттєвої доставки повідомлень. Актуальною є розробка інтегрованих систем, що поєднують збір діагностичних даних, формування звітів та інтерактивну комунікацію в реальному часі.

Аналіз останніх досліджень. Питання побудови розподілених систем реального часу активно досліджували вітчизняні та зарубіжні науковці. Протокол WebSocket, стандартизований у RFC 6455 у 2011 році, визнано ефективним рішенням для двонаправленої комунікації через єдине TCP-з'єднання [1]. Spring Framework забезпечує зручну інтеграцію WebSocket із підтримкою STOMP-протоколу, що спрощує реалізацію обміну повідомленнями в застосунках [2].

Дослідження Pimentel і Nickerson показують, що WebSocket зменшує затримку передачі даних на 50–70 % порівняно з HTTP-опитуванням, а також знижує обсяг трафіку за рахунок відсутності повторних HTTP-заголовків [3]. Роботи Walls присвячені практичному застосуванню Spring Framework для побудови веб-застосунків із підтримкою WebSocket [4]. Проте недостатньо вивченими залишаються питання інтеграції WebSocket-комунікації із системами моніторингу апаратних параметрів комп'ютерів.

Формулювання цілей статті (постановка завдання). Метою статті є дослідження архітектурних рішень для побудови інтегрованої клієнт-серверної системи моніторингу з використанням

WebSocket для забезпечення обміну даними в реальному часі. Завдання дослідження: проаналізувати особливості протоколу WebSocket і STOMP; обґрунтувати вибір архітектури системи; описати механізми інтеграції компонентів; дослідити питання безпеки WebSocket-комунікації.

Основна частина. Розроблена система моніторингу побудована на основі клієнт-серверної архітектури та складається з трьох основних компонентів: клієнтської частини на JavaFX із використанням Spring Framework для керування залежностями, серверної частини на Spring Boot і реляційної бази даних PostgreSQL. Така архітектура забезпечує чітке розмежування відповідальності між компонентами та сприяє масштабованості системи (рис. 1).

Рис. 1. Діаграма компонентів архітектури системи моніторингу

Клієнтська частина забезпечує взаємодію користувача із системою, включаючи збір системних параметрів комп'ютера за допомогою кросплатформеної бібліотеки OSHI-core, формування та надсилання звернень до сервісного центру, перегляду й редагування профілю, а також обміну повідомленнями зі спеціалістами через чат у реальному часі. Для побудови графічного інтерфейсу використано JavaFX з описом компонентів у FXML-файлах, що забезпечує гнучкість і зручність підтримки.

Серверна частина обробляє REST-запити від клієнтів, реалізує основну бізнес-логіку, генерує PDF-звіти на основі бібліотеки Apache PDFBox і забезпечує маршрутизацію повідомлень через WebSocket-брокер. Взаємодія з базою даних здійснюється через Spring Data JPA, що реалізує об'єктно-реляційне відображення даних без необхідності написання SQL-запитів.

Для реалізації чату в реальному часі між клієнтами та спеціалістами обрано протокол WebSocket із використанням STOMP (Simple Text Oriented Messaging Protocol). WebSocket – це протокол, стандартизований у RFC 6455, який забезпечує повнодуплексний канал зв'язку через єдине TCP-з'єднання. На відміну від HTTP, WebSocket підтримує постійне двостороннє з'єднання, що дає змогу миттєво обмінюватися повідомленнями без необхідності встановлювати нове з'єднання для кожного запиту [1].

STOMP (Simple Text Oriented Messaging Protocol) – це текстовий протокол обміну повідомленнями, який працює поверх WebSocket і надає семантику публікації/підписки. STOMP визначає формат фреймів для команд CONNECT, SUBSCRIBE, SEND, MESSAGE [6]. Використання STOMP дає змогу абстрагуватися від низькорівневих деталей протоколу та зосередитися на бізнес-лозіці застосунку.

Основні переваги WebSocket включають зменшення затримки передачі даних завдяки постійному з'єднанню; зниження навантаження на сервер через відсутність повторних HTTP-заголовків; можливість надсилання даних сервером клієнту без попереднього запиту.

Конфігурація WebSocket-сервера реалізована в класі WebSocketConfig і визначає кінцеву точку /ws, через яку клієнти встановлюють з'єднання. Для маршрутизації повідомлень використовується брокер повідомлень, налаштований із префіксами /queue для особистих повідомлень, /topic для групових каналів та /user для адресації конкретним користувачам. Такі префікси дають змогу реалізувати приватні та групові канали обміну повідомленнями [2].

Для кожного звернення (issue) у системі створюється окремий топік /topic/chat/{issueId}, на який підписуються учасники діалогу. Додатково створюється канал /topic/chat-status/{issueId} для сповіщення про зміну статусу звернення (OPEN, IN_PROGRESS, CLOSED).

Процес установлення WebSocket-з'єднання складається з кількох етапів. Спочатку клієнт виконує стандартний WebSocket handshake через HTTP-запит на адресу /ws із заголовком Upgrade: websocket. Після встановлення базового з'єднання клієнт ініціює STOMP-сесію, надсилаючи фрейм STOMP CONNECT з токеном автентифікації у заголовку. Сервер виконує перевірку JWT-токена через Spring Security й у разі успіху підтверджує з'єднання фреймом STOMP CONNECTED [5].

Обробка вхідних повідомлень здійснюється в контролері ChatController за адресою /app/chat/{issueId}/send. Кожне повідомлення проходить валідацію та зберігається в базі даних через сервіс ChatService, де також перевіряється право користувача надсилати повідомлення в конкретний чат. Після збереження повідомлення миттєво транслюється всім підписникам каналу через Spring-брокер.

Безпека WebSocket-з'єднань забезпечується конфігураційним класом WebSocketSecurityConfig, який реалізує авторизацію на рівні повідомлень. Система контролює доступ користувачів до конкретних тем (topic) та особистих каналів (user) згідно з їх ролями та правами доступу. Механізм перевірки базується на JWT-токенах (JSON Web Token), які клієнт передає при встановленні сесії та які містять інформацію про ідентифікатор користувача та його роль [7].

JWT-токен складається із заголовка із зазначенням алгоритму підпису, корисного навантаження з даними про користувача й терміном дії, і підпису для перевірки цілісності. На серверній стороні застосовується Spring Security зі спеціалізованим JWT-фільтром, який перехоплює всі вхідні HTTP-запити й перевіряє валідність токена [8].

Для WebSocket-комунікації механізм перевірки прав реалізований на рівні підключення й підписки до каналів через ChannelInterceptor. Конфігурація розмежування прав доступу реалізована в класі SecurityConfiguration, де визначено дозволи для різних кінцевих точок системи відповідно до ролей: CLIENT, SPECIALIST, ADMIN [9].

Модуль комунікації підтримує ініціацію звернення через REST-запит із можливістю прикріпити звіт; обмін повідомленнями в реальному часі через WebSocket; зміну статусу чату з автоматичним інформуванням; контроль доступу на основі JWT-авторизації.

Для перевірки стійкості й продуктивності системи проведено навантажувальне тестування за допомогою фреймворку Gatling. Сценарій тестування передбачав емулювання одночасної роботи 50 унікальних користувачів, для кожного з яких виконувався ланцюжок операцій: вхід до системи та масова генерація 20 звітів.

Під час тестування виконано 1050 запитів (50 операцій входу та 1000 створень звітів) без помилок. Середній час відповіді для автентифікації становив 9162 мс, максимальний – 11911 мс, що є очікуваним для тестового сервера AWS t2.micro.

Операції генерації звітів виконувалися значно швидше. Після першого запиту (середній час 2628 мс) подальші створення звітів оброблялися із середнім часом відповіді 245–468 мс. Результати навантажувального тестування представлено на рис. 2.

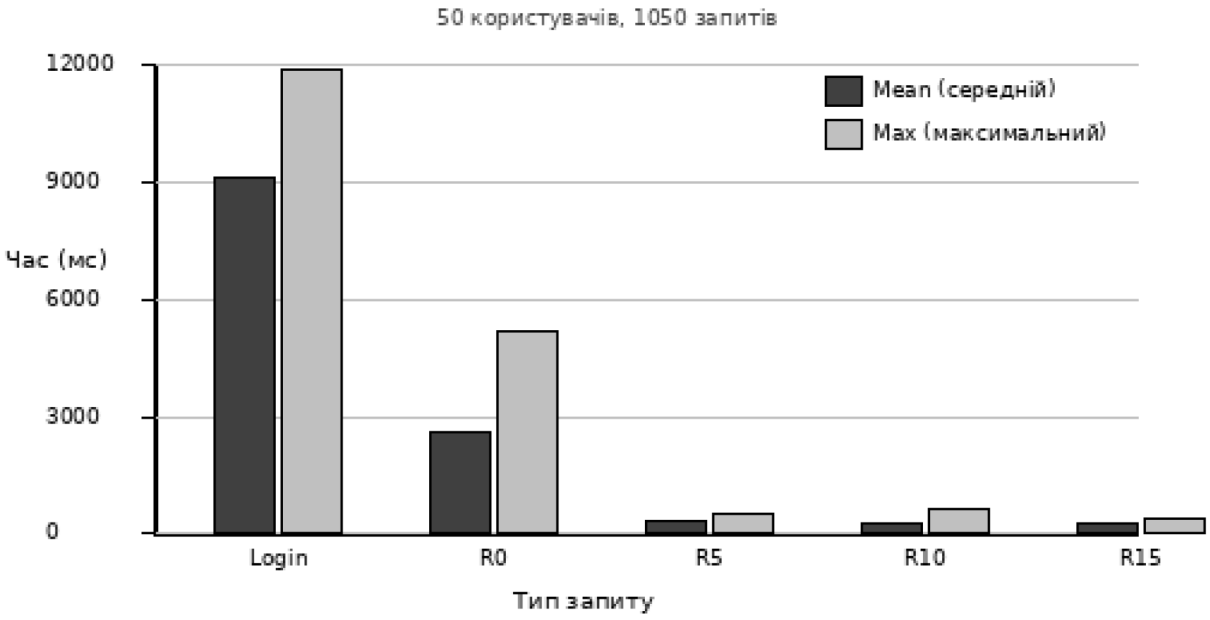

Рис. 2. Час відповіді сервера при навантажувальному тестуванні

Аналіз результатів засвідчив стійкість системи до паралельних навантажень і здатність обробляти запити від 50 одночасних користувачів. Час відповіді для операцій з даними не перевищує 1 секунди, що відповідає нефункціональним вимогам до продуктивності.

Висновки. У статті досліджено архітектуру інтегрованої клієнт-серверної системи моніторингу з використанням WebSocket для обміну даними в реальному часі. Установлено, що застосування WebSocket у поєднанні зі STOMP-протоколом забезпечує ефективну двонаправлену комунікацію між клієнтами та спеціалістами сервісного центру із затримкою, значно меншою за традиційне HTTP-опитування.

Обґрунтовано вибір архітектури на основі Spring Boot для серверної частини, JavaFX із Spring Framework для клієнтської частини та PostgreSQL для зберігання даних. Така архітектура забезпечує чітке розмежування відповідальності між компонентами, модульність та сприяє масштабованості системи.

Описано механізми інтеграції компонентів системи через REST API для основних операцій і WebSocket для обміну повідомленнями в реальному часі. Досліджено питання безпеки WebSocket-комунікації та реалізовано авторизацію на рівні повідомлень з використанням JWT-токенів, що забезпечує конфіденційність даних і захист від несанкціонованого доступу.

Перспективами подальших досліджень є оптимізація масштабування системи для роботи з більшою кількістю одночасних користувачів, упровадження механізмів кластеризації WebSocket-з'єднань із використанням брокерів повідомлень (RabbitMQ, Redis) і дослідження можливостей горизонтального масштабування серверної інфраструктури.

Список використаних джерел

1. Fette I., Melnikov A. The WebSocket Protocol. RFC 6455. IETF, 2011. URL: https://tools.ietf.org/html/rfc6455
2. Spring Framework Documentation. WebSocket Support. URL: https://docs.spring.io/spring-framework/reference/web/websocket.html
3. Pimentel V., Nickerson B.G. Communicating and Displaying Real-Time Data with WebSocket. *IEEE Internet Computing*. 2012. Vol. 16, № 4. P. 45–53. DOI: 10.1109/MIC.2012.64
4. Walls C. Spring in Action. 6th ed. Shelter Island : Manning Publications, 2022. 520 p.
5. Афанасьєв І. І. Розробка програмного комплексу моніторингу системних параметрів та обліку клієнтських звернень для сервісного центру : дипломна робота бакалавра. Запоріжжя : НУ «Запорізька політехніка», 2025. 126 с.
6. STOMP Protocol Specification. Version 1.2. URL: https://stomp.github.io/stomp-specification-1.2.html
7. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. IETF, 2015. URL: https://tools.ietf.org/html/rfc7519
8. Spring Security Reference. URL: https://docs.spring.io/spring-security/reference/index.html
9. Spilca L. Spring Security in Action. Shelter Island : Manning Publications, 2020. 560 p.
10. Gatling Documentation. Load Testing Tool. URL: https://gatling.io/docs/

Дата першого надходження статті до видання: 10.02.2026
Дата прийняття статті до друку після рецензування: 05.03.2026
Дата публікації (оприлюднення) статті: 28.04.2026

S. Hrushko[1,2], I. Afanasiev[1], A. Timenko[1,2], N. Kulykovska[1]

[1] National University Zaporizhzhia Polytechnic
[2] National University of Life and Environmental Sciences of Ukraine

ARCHITECTURE OF AN INTEGRATED CLIENT-SERVER MONITORING SYSTEM USING WEBSOCKET FOR REAL-TIME DATA EXCHANGE

Summary

The article examines the architecture of an integrated client-server system for monitoring computer system parameters designed for service centers. The research investigates the features of implementing the WebSocket protocol in conjunction with STOMP (Simple Text Oriented Messaging Protocol) for bidirectional real-time messaging between clients and technical support specialists. The advantages of WebSocket technology compared to traditional HTTP polling approaches are analyzed, including reduced latency, decreased server load, and efficient network resource utilization through a persistent full-duplex connection.

The paper describes architectural solutions for integrating system components based on modern technologies: Spring Boot framework for the server-side implementation, JavaFX with Spring Framework for the desktop client application, and PostgreSQL relational database for data persistence. The system architecture ensures clear separation of responsibilities between components, modularity, and scalability for future enhancements.

Particular attention is paid to communication security mechanisms through JWT (JSON Web Token) authorization at the WebSocket connection level. The implementation of message-level authorization using Spring Security and ChannelInterceptor for controlling access to specific topics and channels according to user roles is described. The article presents load testing results using the Gatling framework, which confirmed the system's ability to handle 50 concurrent users. All 1050 requests were processed successfully with response times ranging from 245 ms for report generation to 9162 ms for authentication operations.

Keywords: WebSocket, STOMP, client-server architecture, integrated system, monitoring, Spring Boot, JavaFX, real-time.