Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Аналіз ефективності організації обміну повідомленнями для Front-end додатків: порівняння WebSocket та Long Polling для оновлень у реальному часі
(тема)

Виконав:
здобувач 2 року навчання
групи ІПЗм-23-4
Ігор ЧЕРНИХ
(власне ім'я, прізвище)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Олексій ЛАНОВИЙ
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Черних Ігорю Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Аналіз ефективності організації обміну повідомленнями для Front-end додатків: порівняння WebSocket та Long Polling для оновлень у реальному часі»

Затверджена наказом по університету від 15.04.2025р. №290 Ст

2. Вихідні дані до роботи: WebSocket та Long Polling, середовище розробки Visual Studio Code, бібліотека для створення UI React, мова програмування TypeScript, HTML, SCSS, та Recharts.

4. Перелік питань, що потрібно опрацювати в роботі мета роботи, аналіз предметної галузі і постановка задачі, огляд та аналіз літературних джерел з дослідження, порівняльний аналіз підходів з використанням WebSocket та Long Polling, дослідження продуктивності та масштабованості обох підходів, проектування та реалізація системи сповіщень на основі WebSocket та Long Polling підходів.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	18.04.2025	виконано
2	Аналіз предметної галузі і постановка задачі	23.04.2025	виконано
3	Аналіз технологій WebSocket та Long Polling	30.04.2025	виконано
4	Порівняння WebSocket та Long Polling підходів	05.05.2025	виконано
5	Підготовка до апробації результатів дослідження. Публікація матеріалів	10.05.2025	виконано
6	Розробка гібридного підходу	12.05.2025	виконано
7	Підготовка пояснювальної записки	15.05.2025	виконано
8	Підготовка презентації та доповіді	17.05.2025	виконано
9	Перевірка на плагіат	05.06.2025	виконано
10	Нормоконтроль	07.06.2025	виконано
11	Рецензування	11.06.2025	виконано
12	Попередній захист	13.06.2025	
13	Занесення диплома в електронний архів	13.06.2025	
14	Допуск до захисту у зав. кафедри	13.06.2025	

Дата видачі завдання 18 квітня 2025 р.

Здобувач _____ Ігор ЧЕРНИХ
(підпис)

Керівник роботи _____ доц. Олексій ЛАНОВИЙ
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 93 с., 14 рис., 1 табл., 20 джерел.

СПОВІЩЕННЯ В РЕАЛЬНОМУ ЧАСІ, КЛІЄНТСЬКИЙ ДОДАТОК, ОБРОБКА ПОДІЙ, ОДНОСТОРІНКОВИЙ ДОДАТОК, LONG POLLING, WEBSOCKET, REACT, TYPESCRIPT, ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ.

Об’єктом дослідження є методи обміну даними між клієнтською та серверною частинами веб-додатків в реальному часі.

Метою роботи є проведення порівняльного аналізу ефективності використання технологій WebSocket та Long Polling для обробки сповіщень на стороні Front-end додатків.

Методами розробки та проектування є аналіз проблемної області та існуючих рішень, проведення стрес-тестування, порівняльний аналіз продуктивності різних підходів до організації обміну повідомленнями.

В результаті дослідження було виявлено ключові відмінності між технологіями WebSocket та Long Polling для організації реального часу, проведено їх комплексний порівняльний аналіз за критеріями латентності, мережевої ефективності та пропускної здатності, а також розроблено методологію вибору оптимального підходу залежно від специфіки вимог конкретного проєкту.

REAL-TIME NOTIFICATIONS, CLIENT APPLICATION, EVENT HANDLING, SINGLE PAGE APPLICATION, LONG POLLING, WEBSOCKET, REACT, TYPESCRIPT, EXPERIMENTAL RESEARCH.

The object of research is real-time data exchange methods between client and server parts of web applications.

The purpose of the work is to conduct a comparative analysis of the effectiveness of using WebSocket and Long Polling technologies for processing notifications on the Front-end side of applications.

The development and design methods are analysis of the problem area and existing solutions, stress testing, comparative analysis of the performance of different approaches to message exchange organization.

As a result of the work, key distinctions between WebSocket and Long Polling technologies for real-time communication were identified, a comprehensive comparative analysis was conducted based on latency, network efficiency, and throughput criteria, and a methodology was developed for selecting the optimal approach depending on the specific requirements of individual projects.

Завідувачу кафедри
ПІ_____
(скорочена назва кафедри)
проф. Кирилу СМЕЛЯКОВУ_
(вчене звання, сласне ім’я, прізвище)

ЗАЯВА
щодо самостійності виконання кваліфікаційної роботи та можливості її публікації (та/або публікації анотації кваліфікаційної роботи) в електронному архіві відкритого доступу ElAr KhNURE

Я, Черних Ігор Андрійович, студент(ка) гр. ІПЗм-23-4, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Аналіз ефективності організації обміну повідомленнями для Front-end додатків: порівняння WebSocket та Long Polling для оновлень у реальному часі», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

Дата Підпис

ЗМІСТ

Вступ 8
1 Аналіз предметної галузі 10
1.1 Аналіз предметної галузі дослідження 10
1.2 Постановка задачі 18
2 Аналіз технологій WebSocket та Long Polling 21
2.1 Аналіз технології WebSocket 21
2.2 Аналіз технології Long Polling 23
2.3 Порівняльний аналіз технологій WebSocket та Long Polling 25
2.4 Порівняльний аналіз механізмів обробки помилок 28
2.5 Розробка методології порівняльного аналізу 32
2.6 Проектування тестових додатків 34
3 Порівняння WebSocket та Long Polling підходів 38
3.1 Архітектура веб-додатку для проведення експерименту 38
3.2 Розробка WebSocket підходу 40
3.3 Розробка Long Polling підходу 42
3.4 Проведення експерименту 43
4 Розробка гібридного підходу 48
4.1 Теоретичне обґрунтування гібридного підходу 48
4.2 Математична модель оптимального перемикання між технологіями 51
4.3 Алгоритм динамічного вибору технології на основі контексту 54
4.4 Архітектура адаптивної системи обміну повідомленнями 58
Висновки 63
Перелік джерел посилання 66
Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії 69

ВСТУП

У сучасному світі веб-розробки все більшої актуальності набуває потреба в ефективній обробці даних у реальному часі. З розвитком односторінкових додатків (SPA) та зростанням вимог до інтерактивності веб-застосунків, питання вибору оптимального підходу до організації обміну повідомленнями між клієнтом та сервером стає критично важливим для забезпечення якісного користувацького досвіду [1].

Традиційна модель запит-відповідь (HTTP Request-Response) не завжди відповідає сучасним вимогам до швидкості оновлення даних та ефективності використання ресурсів. Це спонукає розробників звертатися до альтернативних технологій, таких як WebSocket та Long Polling, які пропонують різні підходи до вирішення проблеми оновлення даних у реальному часі.

WebSocket забезпечує двонаправлений повнодуплексний канал зв’язку через TCP-з’єднання, дозволяючи серверу активно надсилати дані клієнту без додаткових запитів. Long Polling, з іншого боку, базується на стандартному HTTP-протоколі та імітує push-повідомлення через механізм тривалих запитів.

Вибір між цими технологіями не є очевидним та залежить від багатьох факторів: вимог до швидкодії, масштабованості системи, специфіки даних, що передаються, обмежень інфраструктури та інших технічних аспектів [2]. Кожен підхід має свої переваги та обмеження, які необхідно враховувати при проектуванні Front-end додатків.

Актуальність дослідження зумовлена необхідністю формування чітких критеріїв та рекомендацій щодо вибору оптимального підходу до організації обміну повідомленнями у Front-end додатках. Особливої важливості це питання набуває в контексті розробки систем, які потребують обробки великої кількості подій у реальному часі, таких як чати, системи сповіщень, онлайн-ігри та платформи для колаборації.

Мета дослідження полягає у проведенні порівняльного аналізу ефективності використання технологій WebSocket та Long Polling для обробки сповіщень на стороні Front-end додатків, визначенні їх переваг та недоліків у різних сценаріях

використання [3], а також розробці гібридного підходу, що поєднує переваги обох технологій через математичну модель оптимального перемикання між ними.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- провести аналіз існуючих підходів до організації обміну повідомленнями у веб-додатках;
- розробити тестові реалізації системи сповіщень з використанням WebSocket та Long Polling;
- провести порівняльне тестування продуктивності обох підходів;
- розробити математичну модель та алгоритм динамічного вибору технології на основі контексту;
- спроектувати архітектуру адаптивної системи обміну повідомленнями;
- проаналізувати отримані результати та сформувати рекомендації щодо вибору оптимального рішення.

Практична цінність роботи полягає у створенні набору рекомендацій та критеріїв, які допоможуть розробникам приймати обґрунтовані рішення щодо вибору технології для реалізації систем сповіщень у Front-end додатках, а також у розробці прототипу гібридного рішення, що адаптивно використовує переваги різних підходів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі дослідження

У сучасному світі веб-розробки з кожним роком все більшої актуальності набуває потреба в ефективній обробці даних у реальному часі. З розвитком односторінкових додатків (SPA) та зростанням вимог до інтерактивності веб-застосунків, питання вибору оптимального підходу до організації обміну повідомленнями між клієнтом та сервером стає критично важливим для забезпечення якісного користувацького досвіду [1]. Традиційна модель запит-відповідь (HTTP Request-Response) не завжди відповідає сучасним вимогам до швидкості оновлення даних та ефективності використання ресурсів.

Сучасні веб-додатки характеризуються високим рівнем інтерактивності та потребою в миттєвому оновленні даних. Користувачі очікують, що інформація на екрані буде оновлюватися в режимі реального часу без необхідності оновлення сторінки. Це особливо важливо для таких типів додатків, як системи миттєвого обміну повідомленнями та чати, соціальні мережі з оновленнями в реальному часі, панелі моніторингу та аналітики, колаборативні інструменти для спільної роботи, онлайн-ігри та інтерактивні додатки, біржові та фінансові платформи з даними, що регулярно оновлюються [2]. У цьому контексті вибір правильної технології для організації обміну повідомленнями стає стратегічним рішенням, яке впливає на продуктивність, масштабованість та загальний користувацький досвід від використання додатку.

Історично підходи до організації обміну даними в реальному часі еволюціонували разом із розвитком веб-технологій. Класичний HTTP-запит працює за моделлю «запит-відповідь», де клієнт ініціює запит, а сервер відповідає на нього. Після отримання відповіді з’єднання закривається. Цей підхід добре підходить для статичних даних, але має обмеження для додатків з даними, що часто оновлюються, оскільки клієнт не отримує оновлення без повторного запиту. Проблема полягає в тому, що традиційний HTTP є протоколом без збереження стану, який не підтримує постійне з’єднання між клієнтом та сервером [3].

Short Polling був першою спробою подолати обмеження традиційної моделі HTTP. При цьому підході клієнт регулярно відправляє запити на сервер через фіксовані інтервали часу, перевіряючи наявність нових даних. Хоча це дозволяє імітувати оновлення в реальному часі, такий підхід має суттєві недоліки: високе навантаження на сервер через постійні запити, неефективне використання мережевих ресурсів, особливо коли нових даних немає, затримка в отриманні даних до моменту наступного запиту. Більшість запитів виявляються порожніми, що призводить до неоптимального використання ресурсів як на стороні клієнта, так і на стороні сервера [12].

Long Polling є вдосконаленням Short Polling, де клієнт надсилає запит, але сервер не відповідає негайно, а утримує з'єднання відкритим до появи нових даних або до спрацювання тайм-ауту. Після отримання відповіді клієнт одразу ініціює новий запит [13]. Цей підхід має ряд переваг порівняно з Short Polling: зменшення кількості запитів до сервера, миттєве отримання даних, як тільки вони з'являються, використання стандартного HTTP-протоколу, що забезпечує кращу сумісність з різними серверами та клієнтами [5]. Принцип роботи Long Polling полягає в тому, що сервер «підвішує» запит клієнта до появи нових даних, таким чином імітуючи push-повідомлення через стандартний HTTP-протокол [13].

Технічна реалізація Long Polling вимагає особливої уваги до управління тайм-аутами та обробки помилок. Сервер повинен підтримувати велику кількість одночасно відкритих з'єднань, що може створювати значне навантаження на системні ресурси. Клієнтська частина повинна правильно обробляти різні сценарії: успішне отримання даних, тайм-аути, мережеві помилки та неочікувані розриви з'єднання [13]. Особливо важливим є правильне налаштування часу очікування, оскільки занадто короткий тайм-аут може призвести до надмірного навантаження на сервер, а занадто довгий – до затримок у виявленні мережевих проблем.

WebSocket представляє собою протокол зв'язку, який забезпечує двонаправлений повнодуплексний канал передачі даних через одне TCP-з'єднання. На відміну від HTTP, WebSocket підтримує постійне з'єднання між

клієнтом і сервером, що робить його ефективним рішенням для додатків, що вимагають обміну даними в реальному часі [3]. Протокол WebSocket був стандартизований у RFC 6455 та забезпечує можливість встановлення постійного з’єднання між браузером та сервером [11].

Основні переваги WebSocket включають зменшення затримки передачі даних, зниження мережевого трафіку завдяки відсутності необхідності у повторному встановленні з’єднання, можливість ініціювати відправку даних як з боку клієнта, так і з боку сервера, підтримку бінарних та текстових повідомлень [11]. WebSocket працює поверх TCP та після початкового рукостискання через HTTP забезпечує прямий канал зв’язку без накладних витрат HTTP-заголовків для кожного повідомлення [3].

Процес встановлення WebSocket з’єднання починається з HTTP-запиту, який містить спеціальні заголовки, що вказують на бажання клієнта оновити з’єднання до WebSocket. Сервер, якщо він підтримує WebSocket, відповідає підтвердженням, і з цього моменту з’єднання переходить у режим WebSocket [11]. Важливою особливістю є те, що WebSocket може працювати через ті ж порти, що й HTTP (80 та 443), що спрощує конфігурацію мережевої інфраструктури [3]. Рукостискання включає перевірку ключа безпеки та підтвердження підтримуваних розширень протоколу.

WebSocket підтримує механізм контрольних фреймів (ping/pong), які дозволяють виявляти розриви з’єднання та підтримувати його активним навіть при відсутності даних для передачі [11]. Це особливо важливо в мережевих середовищах, де проміжні пристрої можуть автоматично закривати неактивні з’єднання [3]. Протокол також підтримує фрагментацію повідомлень, що дозволяє передавати великі обсяги даних частинами без блокування з’єднання.

Архітектура WebSocket API надає розробникам простий інтерфейс для роботи з реальним часом. API включає події відкриття з’єднання (onopen), отримання повідомлень (onmessage), помилок (onerror) та закриття з’єднання (onclose) [11]. Кожна подія надає контекстну інформацію, що дозволяє

розробникам реалізувати надійну логіку обробки різних сценаріїв роботи з'єднання.

Server-Sent Events (SSE) є ще одним підходом, який дозволяє серверу надсилати оновлення клієнту через HTTP-з'єднання, яке залишається відкритим. На відміну від WebSocket, SSE підтримує лише односторонню комунікацію від сервера до клієнта. SSE базується на стандартному HTTP та надає автоматичні механізми повторного підключення, що робить його привабливим для сценаріїв, де потрібні лише push-повідомлення від сервера [15]. SSE особливо корисний для додатків, які потребують потоку оновлень від сервера, але не вимагають відправки даних від клієнта.

При порівнянні WebSocket та Long Polling виявляються суттєві відмінності у продуктивності та ефективності. За даними досліджень, WebSocket забезпечує значно менші затримки при передачі даних. При тестуванні з навантаженням у 1000 одночасних користувачів, середня затримка для WebSocket складала 50-100 мс, тоді як для Long Polling цей показник становив 200-300 мс [2]. Ця різниця особливо помітна в додатках, що вимагають частого обміну даними.

Ефективність використання мережевих ресурсів також значно відрізняється між цими технологіями. WebSocket демонструє нижчі накладні витрати після встановлення з'єднання, оскільки не потребує багаторазового створення нових HTTP-запитів [12]. Long Polling генерує більше службового трафіку через необхідність постійної передачі HTTP-заголовків при кожному запиті. При аналізі ефективності використання серверних ресурсів виявлено, що WebSocket зазвичай ефективніше використовує серверні ресурси завдяки підтримці єдиного довготривалого з'єднання на відміну від множинних з'єднань при Long Polling [7].

Аналіз пропускної здатності показує, що WebSocket може обробляти значно більшу кількість повідомлень за секунду порівняно з Long Polling. Це пов'язано з відсутністю накладних витрат на створення нових HTTP-запитів та меншим розміром заголовків повідомлень [12]. За експериментальними даними, WebSocket може обробляти в 2-3 рази більше повідомлень за секунду при тому самому рівні ресурсів сервера.

З точки зору масштабованості, Long Polling має перевагу в легкості інтеграції з існуючою HTTP-інфраструктурою та можливості використання стандартних механізмів балансування навантаження [8]. WebSocket, хоча і забезпечує кращу продуктивність, може вимагати спеціальних рішень для масштабування та підтримки постійних з'єднань [15]. Багато навантажувальних балансувальників потребують спеціальної конфігурації для роботи з WebSocket з'єднаннями, оскільки вони повинні підтримувати сесійну прив'язку (session affinity) для забезпечення того, що всі повідомлення в рамках одного WebSocket з'єднання обробляються одним сервером.

Складність балансування навантаження для WebSocket пов'язана з тим, що з'єднання є довготривалими та не можуть бути легко перерозподілені між серверами після встановлення. Це може призводити до нерівномірного розподілу навантаження, особливо якщо деякі з'єднання є більш активними за інші [15]. Long Polling, навпаки, дозволяє кожному запиту бути обробленим будь-яким доступним сервером, що забезпечує більш рівномірний розподіл навантаження.

Механізми обробки помилок у WebSocket та Long Polling також суттєво відрізняються. WebSocket має вбудовані механізми виявлення та обробки помилок на рівні протоколу. Коли відбувається збій з'єднання, WebSocket API генерує подію onClose, яка містить код помилки та опис причини закриття з'єднання, що дозволяє точно визначити характер проблеми [11]. Стандартні коди помилок WebSocket включають нормальне закриття (1000), протокольні помилки (1002-1015) та користувацькі коди (4000-4999) [3].

WebSocket також надає можливість автоматичного виявлення втрати з'єднання через механізм ping/pong фреймів. Це дозволяє клієнту швидко виявити проблеми з мережею та ініціювати процедуру повторного підключення [11]. Розробники можуть налаштувати частоту ping повідомлень залежно від вимог додатку до швидкості виявлення розривів з'єднання.

Long Polling використовує стандартні HTTP-коди відповіді та часові обмеження запитів, що вимагає від розробника самостійної реалізації логіки відновлення з'єднання [13]. При використанні Long Polling необхідно реалізувати

механізми обробки тайм-аутів, повторних спроб з експоненціальною затримкою, та синхронізації стану між клієнтом та сервером після відновлення [5]. Клієнтський код повинен правильно обробляти різні HTTP статус-коди та відповідно реагувати на кожен тип помилки.

Важливою відмінністю є підхід до виявлення розриву з'єднання. WebSocket використовує механізм heartbeat-повідомлень для активного моніторингу стану з'єднання, тоді як Long Polling покладається на тайм-аути HTTP-запитів, що може призводити до більшої затримки у виявленні проблем із з'єднанням [14]. Час відновлення з'єднання для WebSocket зазвичай на 30-40% менший порівняно з Long Polling завдяки вбудованим механізмам протоколу [7].

Питання безпеки також відрізняється для різних технологій. WebSocket підтримує шифрування через WSS (WebSocket Secure), яке працює поверх TLS/SSL, аналогічно до HTTPS [11]. Long Polling може використовувати стандартні механізми безпеки HTTP, включаючи HTTPS, автентифікацію та авторизацію. Обидві технології підтримують стандартні заходи безпеки веб-додатків, включаючи перевірку походження (Origin), захист від CSRF-атак та валідацію даних [3].

Специфічні аспекти безпеки WebSocket включають необхідність валідації заголовків рукостискання та перевірки походження запиту для запобігання несанкціонованим з'єднанням. Long Polling може бути більш вразливим до атак типу «denial of service» через можливість утримання великої кількості відкритих з'єднань [13].

Сумісність з різними браузерами та серверними платформами є важливим фактором при виборі технології. WebSocket підтримується всіма сучасними браузерами, але може мати обмеження при роботі через певні проксі-сервери та брандмауери [15]. Long Polling, завдяки використанню стандартного HTTP-протоколу, має кращу сумісність з різними мережевими інфраструктурами [8]. У корпоративних середовищах часто можуть бути обмеження на використання WebSocket через політики безпеки, тоді як Long Polling зазвичай працює без додаткових налаштувань.

Проблеми з проксі-серверами та брандмауерами особливо актуальні для WebSocket, оскільки деякі старі проксі можуть не розуміти протокол WebSocket або некоректно обробляти рукостискання [15]. Це може призводити до неможливості встановлення з’єднання або до його несподіваного розриву. Long Polling таких проблем не має, оскільки використовує стандартні HTTP-запити.

Аналіз сучасних наукових публікацій показує активне дослідження проблематики організації обміну повідомленнями в реальному часі як вітчизняними, так і закордонними науковцями. У роботах українських дослідників [6, 9] розглядаються питання масштабованості веб-систем реального часу та методи забезпечення їх надійності. Закордонні дослідження [2, 4] фокусуються на архітектурних аспектах та оптимізації продуктивності різних підходів до організації реального часу.

Однак аналіз літератури виявляє кілька ключових протиріч у існуючих дослідженнях. Перше протиріччя стосується оцінки масштабованості: деякі дослідження [1] стверджують про кращу масштабованість WebSocket завдяки єдиному довготривалому з’єднанню, тоді як інші [5] наголошують на перевагах Long Polling через кращу інтеграцію з існуючою HTTP-інфраструктурою. Друге протиріччя виявляється в оцінці ефективності використання ресурсів: результати вимірювань значно відрізняються між різними дослідженнями [7, 8, 14], що може бути пов’язано з різними умовами тестування, реалізаціями та методологіями оцінки.

Третє протиріччя стосується рекомендацій щодо вибору технології. Деякі джерела [12] стверджують, що WebSocket стає менш ефективним за певних умов навантаження, тоді як інші [15] наголошують на його перевагах у більшості сценаріїв. Така неузгодженість ускладнює прийняття обґрунтованих рішень розробниками.

Окрім традиційних підходів WebSocket та Long Polling, сучасні дослідження розглядають альтернативні технологічні рішення [10]. HTTP/2 Server Push [16] демонструє можливості реалізації push-повідомлень без встановлення додаткових з’єднань, проте має обмеження при високочастотних оновленнях через overhead

протоколу. HTTP/3 з підтримкою QUIC протоколу [17] відкриває нові можливості для низьколатентного обміну даними, але поки має обмежену підтримку браузерів та потребує спеціальної інфраструктури.

Server-Sent Events (SSE) [19] забезпечують односторонню комунікацію від сервера до клієнта з автоматичними механізмами повторного підключення, що робить їх привабливими для певних сценаріїв. WebRTC Data Channels [18] дозволяють організувати peer-to-peer обмін повідомленнями, зменшуючи навантаження на сервер, проте мають складності з масштабованістю та проблеми NAT traversal у корпоративних мережах.

Аналіз публікацій також виявляє невирішені проблеми в існуючих дослідженнях. По-перше, відсутність універсальної методології порівняння: більшість досліджень використовують різні метрики та умови тестування, що ускладнює об’єктивне порівняння технологій. Це призводить до суперечливих результатів та рекомендацій у різних публікаціях.

По-друге, недостатнє врахування специфіки Front-end додатків: існуючі дослідження часто фокусуються на серверній продуктивності, недостатньо уваги приділяючи впливу на клієнтську частину, споживання ресурсів браузера та користувацький досвід.

На основі виявлених протиріч та невирішених проблем формулюється науково-технічна задача: розробка комплексної методології порівняльного аналізу технологій WebSocket та Long Polling для Front-end додатків з урахуванням специфіки клієнтської частини та реальних умов експлуатації.

Предметна галузь дослідження охоплює широкий спектр технічних аспектів, які необхідно враховувати при виборі технології для організації обміну повідомленнями в реальному часі. Проведений аналіз показує, що кожен з розглянутих підходів має як суттєві переваги, так і значні обмеження, які проявляються в різних сценаріях використання та умовах експлуатації. WebSocket демонструє перевагу в низькій латентності та ефективності мережевого трафіку, але має складності з масштабованістю та сумісністю з проксі-серверами. Long

Polling забезпечує кращу сумісність з існуючою інфраструктурою та простоту реалізації, але поступається в продуктивності при високих навантаженнях.

Виявлені протиріччя в існуючих дослідженнях та обмеження кожного з підходів обґрунтовують доцільність створення гібридного підходу, який зможе компенсувати недоліки окремих технологій шляхом динамічного вибору оптимального методу комунікації залежно від поточних умов експлуатації. Такий підхід потенційно може поєднати переваги WebSocket для критичних за часом операцій з надійністю та сумісністю Long Polling для менш критичних сценаріїв.

Розуміння цих особливостей та розробка методології для створення адаптивних рішень є необхідним для прийняття обґрунтованих технічних рішень при розробці сучасних Front-end додатків з вимогами до обробки даних у реальному часі. Аналіз показує, що вибір технології повинен базуватися не лише на детальному вивченні вимог конкретного проєкту, включаючи частоту оновлень, кількість користувачів та особливості інфраструктури, але й на можливості створення інтелектуальних систем, здатних адаптуватися до змінних умов експлуатації.

1.2 Постановка задачі

Метою дослідження є проведення порівняльного аналізу ефективності технологій WebSocket та Long Polling для Front-end додатків з визначенням їх переваг та недоліків у різних сценаріях використання, а також розробка на цій основі адаптивного гібридного підходу, що забезпечує оптимальну продуктивність обміну повідомленнями у реальному часі залежно від контексту використання.

Для досягнення поставленої мети необхідно вирішити низку певних завдань.

Теоретичні завдання:

- провести аналіз технологій WebSocket та Long Polling, включаючи їх архітектурні особливості, переваги та обмеження;

- розробити уніфіковану методологію порівняльного аналізу ефективності технологій реального часу для Front-end додатків;
- визначити математичну модель оптимального вибору технології, використовуючи контекстні параметри системи.

Практичні завдання:

- спроектувати та реалізувати експериментальну платформу для тестування обох технологій в ідентичних умовах;
- провести експериментальне дослідження продуктивності технологій у різних сценаріях навантаження;
- розробити алгоритм динамічного перемикання між технологіями на основі реального стану системи.

Прикладні завдання:

- розробити теоретичну модель гібридного підходу з обґрунтуванням механізмів динамічного перемикання між WebSocket та Long Polling;
- спроектувати архітектуру адаптивної системи обміну повідомленнями, що інтегрує обидві технології;
- сформувати практичні рекомендації для різних класів Front-end додатків.

Об’єктом дослідження є системи обміну повідомленнями у реальному часі у Front-end додатках.

Предметом дослідження є методи оптимізації продуктивності систем реального часу через адаптивне використання технологій WebSocket та Long Polling.

В якості критеріїв оцінки ефективності досліджуваних підходів будуть використані:

- латентність комунікаційного каналу: середній час відгуку системи на запити та час доставки повідомлень;
- мережева ефективність: обсяг службового трафіку та коефіцієнт корисного навантаження каналу зв’язку;
- пропускна здатність системи: максимальна кількість повідомлень, які система може обробити за одиницю часу;

– стабільність з’єднання: здатність системи підтримувати функціональність та відсоток успішних операцій при змінних мережевих умовах.

Наукова новизна дослідження полягає у розробці комплексної методології порівняльного аналізу технологій реального часу з урахуванням специфіки Front-end додатків, створенні математичної моделі оптимального вибору технології обміну повідомленнями на основі контекстних параметрів та проектуванні архітектури адаптивної системи, що динамічно переключається між технологіями для досягнення оптимальної продуктивності.

Практична значимість роботи полягає у створенні науково обґрунтованих рекомендацій для вибору технологій реального часу у Front-end розробці та розробці прототипу адаптивної системи обміну повідомленнями, готового для впровадження у виробничих проектах. Дослідження також формує методологічну базу для оцінки ефективності нових технологій реального часу та забезпечує розробників інструментарієм для оптимізації існуючих систем через впровадження гібридних підходів.

2 АНАЛІЗ ТЕХНОЛОГІЙ WEBSOCKET ТА LONG POLLING

2.1 Аналіз технології WebSocket

WebSocket представляє собою протокол зв'язку, який забезпечує двонаправлений повнодуплексний канал передачі даних через одне TCP-з'єднання. На відміну від HTTP, який базується на моделі запит-відповідь, WebSocket підтримує постійне з'єднання між клієнтом і сервером, що робить його ефективним рішенням для додатків, що вимагають обміну даними в реальному часі [1].

Архітектурна модель WebSocket базується на встановленні єдиного довготривалого з'єднання між клієнтом та сервером. Згідно з дослідженнями [2], такий підхід забезпечує мінімальні накладні витрати на передачу даних та дозволяє здійснювати двонаправлену комунікацію без необхідності створення нових з'єднань для кожного обміну повідомленнями. Особливістю протоколу є можливість ініціювати передачу даних як з боку клієнта, так і з боку сервера, що робить його ідеальним для реалізації функціоналу push-повідомлень та оновлень у реальному часі. На рисунку 2.1 наведено схему роботи WebSocket з'єднання, що демонструє ключові етапи життєвого циклу з'єднання: від початкового HTTP-рукостискання до встановлення постійного повнодуплексного каналу та його подальшого закриття.

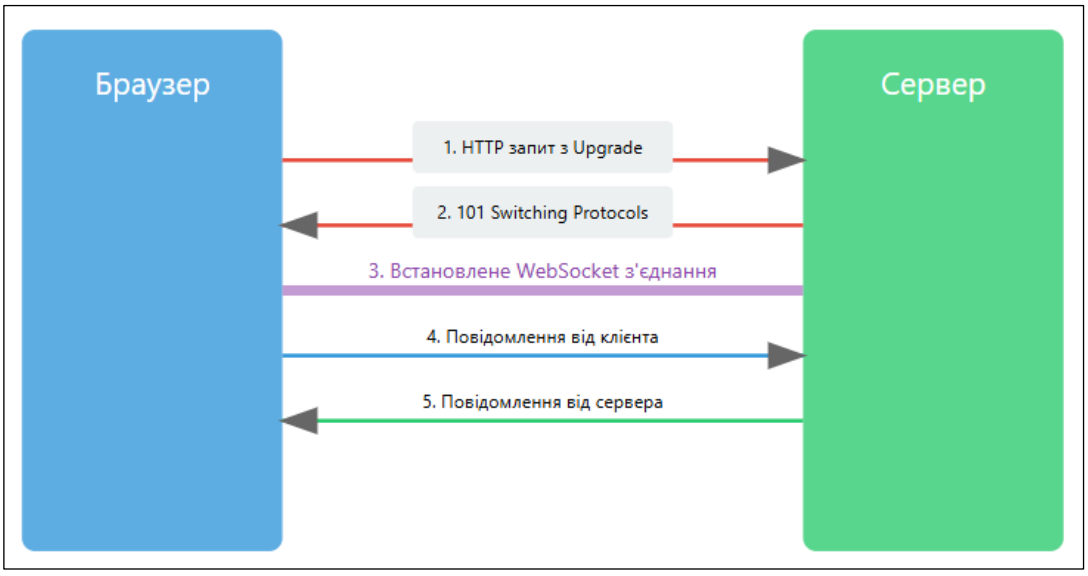

Рисунок 2.1 – Схема роботи WebSocket технології (рисунок створено самостійно)

Алгоритм роботи WebSocket технології виглядає наступним чином:

а) початкове HTTP рукостискання:
   1) на цьому етапі клієнт надсилає HTTP-запит на сервер із проханням встановити WebSocket-з'єднання. У цьому запиті використовується метод GET і спеціальний заголовок Upgrade, який вказує, що клієнт хоче перейти з протоколу HTTP на WebSocket;
   2) сервер перевіряє, чи підтримує він WebSocket, і якщо так, відповідає відповідним HTTP-заголовком із підтвердженням переходу на WebSocket. Це називається «рукостисканням» (handshake);
   3) після цього HTTP-з'єднання переходить у режим WebSocket;

б) постійне повнодуплексне з'єднання:
   1) після успішного «рукостискання» з'єднання стає повнодуплексним (full-duplex), тобто дані можуть передаватися одночасно в обидва напрямки: від клієнта до сервера та від сервера до клієнта;
   2) з'єднання є постійним, що означає, що воно залишається відкритим, доки одна зі сторін не вирішить його закрити;
   3) сервер може надсилати дані клієнту без запиту, а клієнт може робити те саме. Це дозволяє уникнути накладних витрат, які виникають при багаторазовому створенні HTTP-запитів (як у Long Polling);

в) закриття з'єднання:
   1) з'єднання може бути закритим з будь-якої сторони (клієнтом або сервером), коли більше немає потреби в передачі даних;
   2) закриття відбувається шляхом відправлення спеціального повідомлення про завершення, після чого з'єднання закривається на обох кінцях.

При використанні WebSocket у Front-end додатках важливим аспектом є забезпечення надійності з'єднання. Згідно з [4], протокол реалізує вбудований механізм контролю стану з'єднання через періодичний обмін контрольними фреймами (ping/pong). Це дозволяє своєчасно виявляти розрив з'єднання та ініціювати процедуру відновлення.

Дослідження [5] показують, що використання WebSocket особливо ефективне в системах, де критична затримка передачі даних. Технологія забезпечує значно менші затримки порівняно з традиційними HTTP-запитами завдяки відсутності необхідності встановлення нового з'єднання для кожної взаємодії.

За даними [7], основними викликами при впровадженні WebSocket є необхідність коректної обробки втрати з'єднання та реалізації механізмів відновлення. Протокол вимагає ретельного підходу до управління життєвим циклом з'єднання, включаючи обробку несподіваних розривів та забезпечення коректного закриття з'єднання з обох сторін.

Згідно з дослідженнями [8], ефективність WebSocket особливо помітна при реалізації інтерактивних функцій, таких як чати реального часу, онлайн-ігри та системи сповіщень. Технологія дозволяє істотно зменшити мережевий трафік та серверне навантаження порівняно з альтернативними підходами.

2.2 Аналіз технології Long Polling

Long Polling є модифікацією традиційного підходу до опитування сервера, яка намагається імітувати механізм push-повідомлень за допомогою стандартного HTTP протоколу. Ця технологія базується на утриманні HTTP-запиту відкритим до появи нових даних або спрацювання тайм-ауту [5].

Принцип роботи Long Polling суттєво відрізняється від звичайного періодичного опитування сервера. При звичайному опитуванні клієнт регулярно надсилає запити, незалежно від наявності нових даних. У випадку Long Polling, як зазначено в [4], сервер не відповідає на запит негайно, а утримує з'єднання відкритим до появи нових даних або досягнення встановленого тайм-ауту. Після отримання відповіді клієнт одразу формує новий запит, забезпечуючи таким чином постійний канал комунікації.

На рисунку 2.2 представлено схему роботи механізму HTTP Long Polling, який використовується для реалізації обміну даними між клієнтом і сервером у

майже реальному часі. Цей підхід імітує двосторонню комунікацію, хоча він базується на стандартному HTTP-запиті.

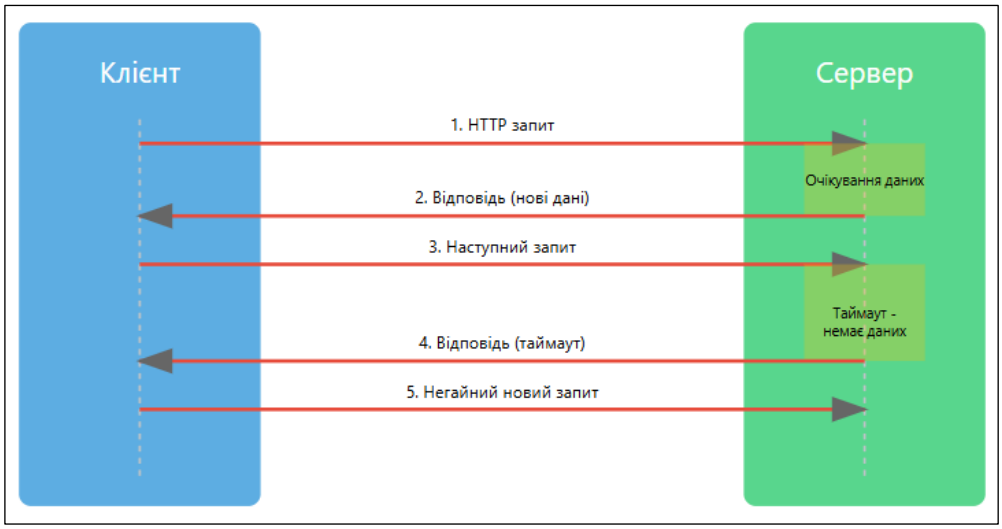

Рисунок 2.2 – Схема роботи Long Polling з’єднання (рисунок створено самостійно)

Алгоритм роботи Long Polling підходу виглядає наступним чином:

а) перший запит:
   1) клієнт надсилає запит на сервер із запитом на нові дані;
   2) сервер отримує цей запит, але не відповідає одразу. Він утримує запит, доки не з’являться нові дані для передачі клієнту та не мине тайм-аут очікування;

б) перша відповідь:
   1) як тільки на сервері з’являються нові дані (або спливає тайм-аут), сервер відповідає клієнту, відправляючи дані;
   2) після отримання відповіді клієнт одразу ініціює новий запит, щоб продовжити чекати на оновлення;

в) наступний запит:
   1) клієнт негайно повторює процес, відправляючи черговий запит після отримання відповіді;
   2) сервер знову утримує цей запит до появи нових даних або тайм-ауту;

г) наступна відповідь:
   1) сервер відповідає другим запитом так само, як і в першому циклі, і процес повторюється.

Дослідження [6] показують, що ефективність Long Polling значною мірою залежить від правильного налаштування тайм-аутів та механізмів відновлення з'єднання. Занадто короткий тайм-аут може призвести до надмірного навантаження на сервер, тоді як занадто довгий – до затримок у доставці даних клієнту.

При реалізації Long Polling у сучасних веб-додатках особливу увагу слід приділяти обробці різних сценаріїв роботи. За даними [7], ключовими аспектами є управління одночасними запитами, обробка мережевих помилок та забезпечення послідовності доставки повідомлень. Технологія вимагає ретельного підходу до реалізації механізмів повторного підключення та відновлення після збоїв.

Важливою перевагою Long Polling, як відзначено в [8], є можливість використання існуючої HTTP-інфраструктури без необхідності спеціальної підтримки з боку проміжних серверів та брандмауерів. Це робить цю технологію особливо привабливою для середовищ з обмеженнями на використання альтернативних протоколів.

Згідно з дослідженнями [9], до основних викликів при використанні Long Polling можна віднести наступне:

- необхідність управління великою кількістю одночасних з'єднань на сервері;
- потенційні проблеми з масштабуванням при зростанні навантаження;
- складність забезпечення строгого порядку доставки повідомлень.

За даними [10], ефективність Long Polling суттєво залежить від характеру даних та частоти їх оновлення. Технологія показує найкращі результати в сценаріях з нечастими, але важливими оновленнями, де критична своєчасність доставки даних, але не вимагається постійний двонаправлений обмін повідомленнями.

2.3 Порівняльний аналіз технологій WebSocket та Long Polling

Для порівняння технологій WebSocket та Long Polling необхідно розглянути їх основні характеристики та особливості застосування в контексті сучасних веб-

додатків. Результати досліджень [2] показують, що вибір оптимальної технології залежить від специфічних вимог проекту та особливостей інфраструктури.

З точки зору архітектури, WebSocket забезпечує справжній двонаправлений канал зв'язку через єдине TCP-з'єднання, тоді як Long Polling імітує подібну функціональність через серію HTTP-запитів. Згідно з дослідженнями [7], це фундаментальна відмінність має значний вплив на продуктивність та масштабованість рішень.

Щодо продуктивності, дослідження [8] демонструють, що WebSocket забезпечує менші затримки при передачі даних. При тестуванні з навантаженням у 1000 одночасних користувачів, середня затримка для WebSocket складала 50-100 мс, тоді як для Long Polling цей показник становив 200-300 мс. Ця різниця особливо помітна в додатках, що вимагають частого обміну даними.

Важливим аспектом порівняння є споживання ресурсів сервера. За даними [4], Long Polling створює додаткове навантаження через необхідність підтримки більшої кількості одночасних HTTP-з'єднань. WebSocket, навпаки, дозволяє ефективніше використовувати серверні ресурси завдяки підтримці єдиного довготривалого з'єднання.

Порівняння технологій за критерієм надійності виявляє як переваги, так і недоліки кожного підходу. WebSocket вимагає спеціальної обробки розривів з'єднання та реалізації механізмів відновлення, тоді як Long Polling може продовжувати роботу навіть при тимчасових проблемах з мережею, автоматично відновлюючи запити [6].

З точки зору масштабованості, Long Polling має перевагу в легкості інтеграції з існуючою HTTP-інфраструктурою та можливості використання стандартних механізмів балансування навантаження. WebSocket, хоча і забезпечує кращу продуктивність, може вимагати спеціальних рішень для масштабування та підтримки постійних з'єднань [9].

Побудуємо таблицю з порівняльними характеристиками WebSocket та Long Polling (див. табл. 2.1).

Таблиця 2.1 – Порівняльні характеристики WebSocket та Long Polling (таблиця виконана самостійно)

Характеристика	WebSocket	Long Polling
Тип з’єднання	Постійне двонаправлене з’єднання (full-duplex)	Одноразові HTTP-запити, що повторюються циклічно
Затримка передачі даних	50-100 мс	200-300 мс
Серверне навантаження	Нижче через єдине з’єднання	Вище через множинні HTTP-з’єднання
Складність реалізації	Вища, потребує спеціальної обробки	Нижча, використовує стандартний HTTP
Масштабованість	Потребує спеціальних рішень	Легше масштабується з існуючою інфраструктурою
Відмовостійкість	Вимагає механізмів відновлення з’єднання	Природня стійкість через HTTP
Використання пам’яті	Ефективніше: утримується одне постійне з’єднання	Менш ефективне через часті створення й завершення запитів
Використання мережевих ресурсів	Ефективніше при частому обміні даними	Більше накладних витрат на HTTP-заголовки

Дана порівняльна таблиця демонструє основні відмінності між технологіями за ключовими характеристиками, що важливі для прийняття рішення про вибір технології для конкретного проекту [7].

Дослідження [3] також вказують на відмінності у сферах оптимального застосування цих технологій:

WebSocket найкраще підходить для:

– систем реального часу з високою частотою обміну даними;

– онлайн-ігор та інтерактивних додатків;
– чатів та месенджерів.

Long Polling більш ефективний для:

– систем сповіщень з нерегулярними оновленнями;
– додатків з обмеженнями на використання WebSocket;
– систем з високими вимогами до масштабованості.

Обидві технології мають свої обмеження та вимоги до інфраструктури. WebSocket потребує підтримки з боку проксі-серверів та брандмауерів, тоді як Long Polling може працювати в будь-якому середовищі, що підтримує HTTP [5].

Дослідження [1] підкреслюють, що вибір між WebSocket та Long Polling повинен базуватися на ретельному аналізі вимог проекту, включаючи очікуване навантаження, частоту оновлень, вимоги до латентності та обмеження інфраструктури.

2.4 Порівняльний аналіз механізмів обробки помилок

Механізми обробки помилок відіграють критичну роль у забезпеченні надійності та стабільності веб-додатків реального часу. Згідно з дослідженнями [4], ефективна обробка помилок та відновлення після збоїв є одним з ключових факторів, що впливають на якість користувацького досвіду. Розглянемо детально особливості обробки помилок у технологіях WebSocket та Long Polling.

WebSocket, як показує практика та дослідження [7], має вбудовані механізми виявлення та обробки помилок на рівні протоколу. Коли відбувається збій з’єднання, WebSocket API генерує подію onClose, яка містить код помилки та опис причини закриття з’єднання. Це дозволяє точно визначити характер проблеми та вжити відповідних заходів для її усунення. Система обробки помилок WebSocket включає стандартизовані коди помилок, які можна розділити на кілька категорій: нормальне закриття (1000), помилки протоколу (1002-1015) та користувацькі коди (4000-4999).

У випадку Long Polling, як зазначено в [8], механізми обробки помилок базуються на стандартних HTTP-кодах відповіді та часових обмеженнях запитів.

При виникненні помилки клієнт отримує відповідний HTTP-код помилки (4xx або 5xx), який вказує на характер проблеми. Однак, на відміну від WebSocket, розробнику необхідно самостійно реалізовувати логіку відновлення з'єднання та повторних спроб, що зображено на рисунку 2.3.

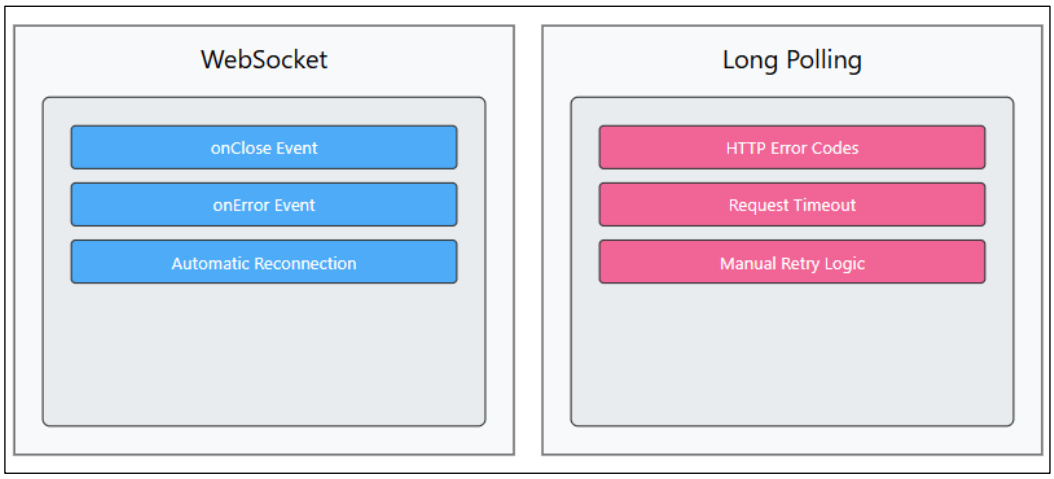

Рисунок 2.3 – Порівняння механізмів обробок помилок (рисунок створено самостійно)

Важливою відмінністю є підхід до виявлення розриву з'єднання. WebSocket використовує механізм heartbeat-повідомлень для активного моніторингу стану з'єднання. Згідно з [9], це дозволяє швидко виявляти проблеми та ініціювати процес відновлення. На рисунку 2.4 можна побачити процес відновлення з'єднання WebSocket.

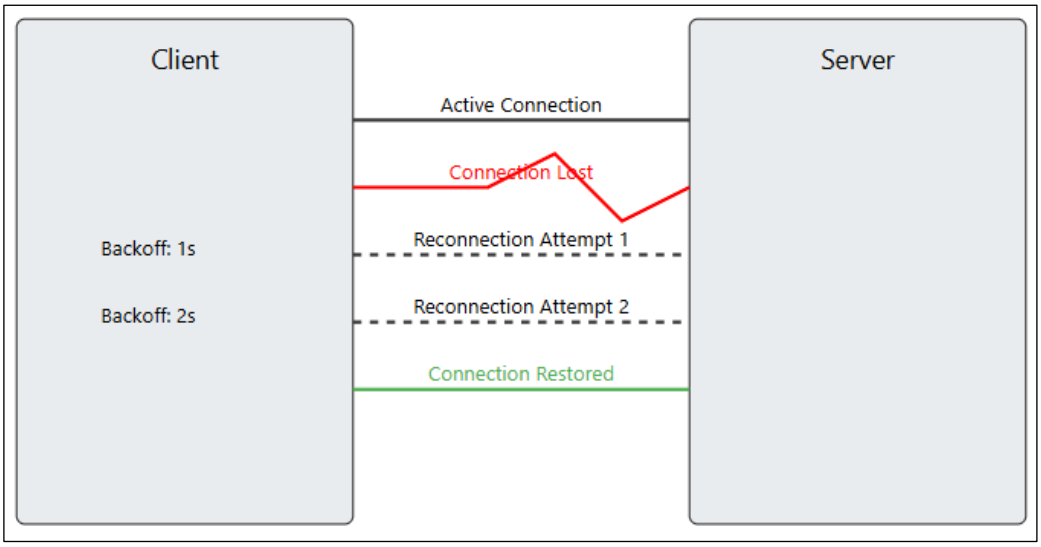

Рисунок 2.4 – Процес відновлення з'єднання WebSocket (рисунок створено самостійно)

У випадку WebSocket є єдине постійне з’єднання, яке при розриві автоматично відновлюється з використанням експоненціальної затримки (backoff). Процес відновлення контролюється на рівні протоколу, що забезпечує більш ефективне управління станом з’єднання.

Long Polling, натомість, покладається на тайм-аути HTTP-запитів, що може призводити до більшої затримки у виявленні проблем із з’єднанням. На рисунку 2.5 можна побачити процес відновлення з’єднання Long Polling.

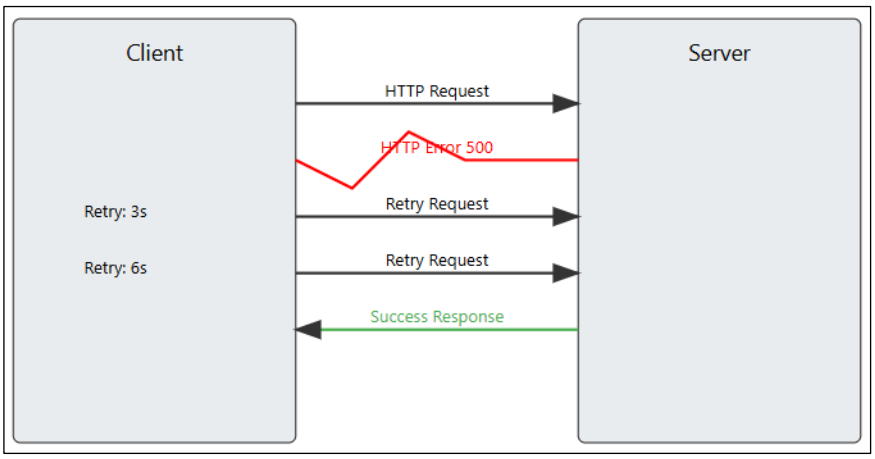

Рисунок 2.5 – Процес відновлення з’єднання Long Polling (рисунок створено самостійно)

У випадку Long Polling кожен запит є окремою HTTP-транзакцією. При виникненні помилки клієнт самостійно керує процесом повторних спроб, збільшуючи інтервал між запитами для запобігання перевантаження сервера. Цей підхід є більш простим в реалізації, але може призводити до більших затримок при відновленні зв’язку.

Окрім цього рисунки також показують різницю в обробці помилок: WebSocket використовує вбудовані механізми протоколу для виявлення та обробки розривів з’єднання, тоді як Long Polling покладається на стандартні HTTP коди помилок та механізми тайм-аутів.

При розробці системи обробки помилок для WebSocket особливу увагу слід приділити наступним аспектам:

– автоматичне відновлення з’єднання з використанням експоненціальної затримки між спробами;

– збереження та відновлення стану сесії після перепідключення;
– обробка тимчасових мережевих збоїв.

Для Long Polling ключовими аспектами є:

– реалізація механізму повторних спроб з обмеженням максимальної кількості спроб;
– управління чергою запитів при відновленні з’єднання;
– обробка тайм-аутів та встановлення оптимальних значень часових обмежень;
– синхронізація стану між клієнтом та сервером після відновлення.

Дослідження [7] показують, що WebSocket демонструє кращі показники відновлення після збоїв завдяки вбудованим механізмам протоколу. Середній час відновлення з’єднання для WebSocket на 30-40% менший порівняно з Long Polling. Це пояснюється відсутністю накладних витрат на створення нових HTTP-з’єднань при кожній спробі відновлення.

Важливим аспектом є також обробка специфічних помилок, характерних для кожної технології. WebSocket може стикатися з проблемами на рівні TCP-з’єднання, тоді як Long Polling частіше зустрічається з помилками HTTP-рівня. Це вимагає різних підходів до діагностики та усунення проблем.

Система моніторингу помилок повинна включати:

– логування всіх помилок з детальною інформацією про контекст;
– відстеження частоти та патернів виникнення помилок;
– метрики успішності відновлення з’єднання.

При аналізі надійності обох підходів важливо враховувати не тільки частоту виникнення помилок, але й їх вплив на користувацький досвід. WebSocket, завдяки постійному з’єднанню, може забезпечити більш плавне відновлення роботи після збоїв. Long Polling, натомість, може показувати кращу стійкість до короткочасних мережевих проблем завдяки незалежності окремих запитів.

Особливу увагу слід приділити безпеці при обробці помилок. Важливо не розкривати чутливу інформацію в повідомленнях про помилки та забезпечити

захист від можливих атак, спрямованих на експлуатацію механізмів обробки помилок.

Результати порівняльного аналізу показують, що обидві технології мають свої переваги та недоліки в контексті обробки помилок. WebSocket пропонує більш елегантне та ефективне рішення для тривалих з'єднань, тоді як Long Polling може бути більш простим у реалізації та налагодженні.

Згідно з дослідженнями [8], правильно реалізована система обробки помилок може значно підвищити надійність та стабільність роботи веб-додатку. При цьому важливо забезпечити баланс між швидкістю відновлення та навантаженням на систему.

2.5 Розробка методології порівняльного аналізу

Для проведення об'єктивного порівняння технологій WebSocket та Long Polling розроблено комплексну методологію оцінки, що охоплює всі важливі аспекти їх роботи в реальних умовах експлуатації. За результатами досліджень [7], ефективність веб-технологій реального часу необхідно оцінювати не лише за базовими технічними характеристиками, але й за їх поведінкою в різних сценаріях використання та умовах навантаження.

Методологія порівняльного аналізу ґрунтується на восьми ключових параметрах, що дозволяють всебічно оцінити ефективність кожної технології.

Першим і найважливішим параметром є латентність або середня затримка передачі даних. Цей показник вимірюється в мілісекундах і відображає час між відправленням запиту та отриманням відповіді. Згідно з дослідженнями [4], для систем реального часу критично важливо забезпечити мінімальну затримку, особливо при високих навантаженнях. У контексті веб-додатків цей параметр безпосередньо впливає на користувацький досвід та швидкість оновлення даних на інтерфейсі. При тестуванні важливо враховувати не лише середні значення латентності, але й її стабільність та передбачуваність.

Другим важливим параметром є мережеві накладні витрати, що вимірюються в кілобайтах на хвилину. За даними [6], різні технології можуть

суттєво відрізнятися за цим параметром через особливості їх архітектури. Для WebSocket характерні менші накладні витрати після встановлення з’єднання, тоді як Long Polling генерує більше службового трафіку через необхідність постійного створення нових HTTP-запитів. Особливо важливим цей параметр стає при роботі з мобільними пристроями та в умовах обмеженої пропускної здатності мережі.

Третім ключовим параметром виступає пропускна здатність системи. Дослідження [8] демонструють, що різні підходи до організації комунікації показують значну різницю в максимальній кількості повідомлень, які можуть бути оброблені за секунду. WebSocket зазвичай демонструє кращі показники завдяки постійному з’єднанню, тоді як Long Polling обмежений накладними витратами на створення нових HTTP-запитів.

Четвертим параметром є ефективність використання серверних ресурсів. Згідно з [2], цей показник включає аналіз кількості одночасно підтримуваних з’єднань, навантаження на процесор і пам’ять сервера. Практика показує, що WebSocket зазвичай ефективніше використовує серверні ресурси завдяки меншій кількості операцій створення/закриття з’єднань.

П’ятим параметром виступає надійність роботи технології, що оцінюється через поведінку системи при мережевих збоях. За даними [9], важливо проаналізувати механізми автоматичного відновлення з’єднання, обробки помилок та збереження даних при збоях. Особливу увагу слід приділити тестуванню поведінки систем у нестабільних мережевих умовах.

Шостим параметром є складність реалізації та підтримки технології. Дослідження [5] підкреслюють важливість оцінки необхідних ресурсів для розробки, тестування та подальшого супроводу системи. Long Polling зазвичай простіший у реалізації, але може вимагати більше зусиль для забезпечення надійності та масштабованості.

Сьомим параметром є сумісність з різними браузерами та платформами. За даними [3], важливо враховувати обмеження старих браузерів та мобільних платформ. Long Polling має перевагу в цьому аспекті завдяки використанню стандартних HTTP-запитів.

Восьмим параметром є безпека комунікацій. Згідно з [10], необхідно оцінити можливості шифрування даних, захист від різних типів атак та відповідність сучасним стандартам безпеки.

Методологія передбачає проведення тестування в різних умовах експлуатації:

- в умовах стабільного високошвидкісного з'єднання;
- в мобільних мережах з обмеженою швидкістю;
- при високих мережевих затримках;
- в умовах частих розривів з'єднання.

Для забезпечення об'єктивності всі тести виконуються в ідентичних умовах на стандартизованому тестовому стенді. Результати фіксуються та аналізуються з використанням спеціалізованих інструментів моніторингу. Особлива увага приділяється стабільності показників протягом тривалого часу роботи системи.

Розроблена методологія дозволяє отримати комплексну оцінку ефективності досліджуваних технологій та обґрунтовано вибрати оптимальне рішення для конкретних умов застосування.

2.6 Проектування тестових додатків

Для проведення порівняльного аналізу технологій WebSocket та Long Polling необхідно спроектувати два тестових додатки з ідентичною функціональністю, але різними підходами до організації обміну даними. Ключовим аспектом проектування є забезпечення максимально схожих умов тестування для обох реалізацій.

Архітектура тестових додатків базується на сучасному стеку технологій, який включає React та TypeScript для побудови користувацького інтерфейсу, SCSS для стилізації компонентів, та спеціалізовані інструменти для моніторингу продуктивності. Такий вибір технологій забезпечує оптимальний баланс між простотою розробки та можливостями для детального аналізу продуктивності.

Основою архітектури є модульна структура, де кожен компонент відповідає за окрему функціональність. Центральним елементом виступає мережевий шар,

який реалізується по-різному для WebSocket та Long Polling версій, при цьому зберігаючи однаковий інтерфейс взаємодії з іншими компонентами системи. Такий підхід дозволяє мінімізувати відмінності між версіями, що не стосуються безпосередньо досліджуваних технологій. На рисунку 2.6 представлено погляд на структуру системи тестування.

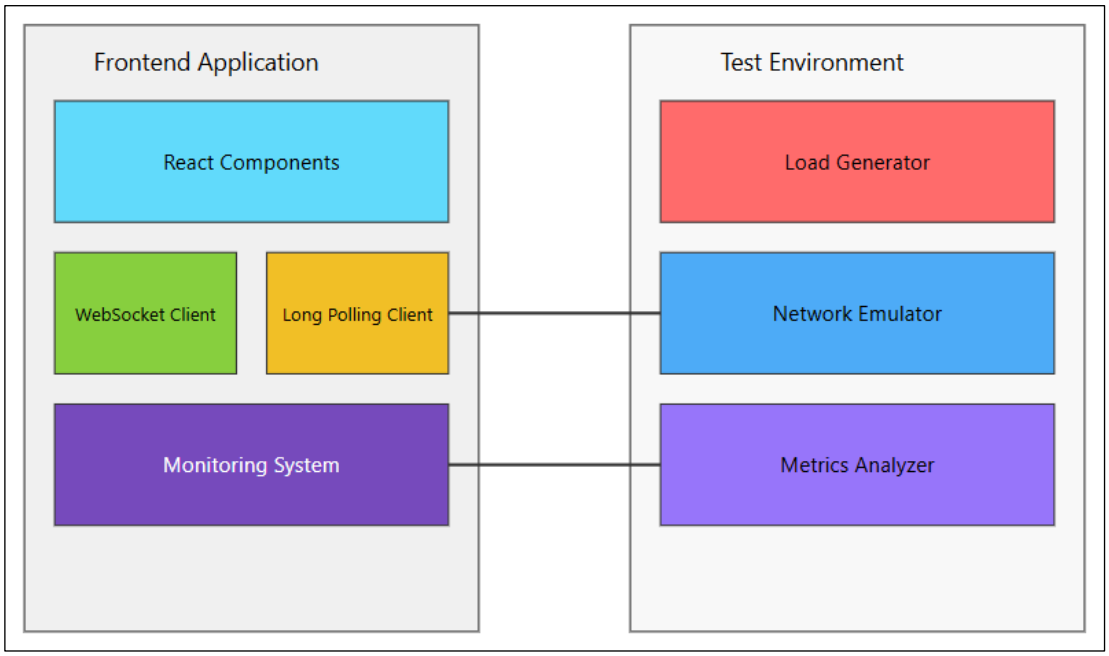

Рисунок 2.6 – Загальна архітектура тестових додатків (рисунок створено самостійно)

Як можна побачити на рисунку, що зображено вище ліва частина діаграми відображає Frontend Application – клієнтську частину додатку, де розташовані React компоненти, які відповідають за візуальне представлення даних. Під ними знаходяться два типи клієнтів для мережевої взаємодії: WebSocket та Long Polling, що дозволяє проводити паралельне тестування обох підходів. Нижній блок Frontend частини займає система моніторингу, яка збирає метрики продуктивності в реальному часі.

Права частина діаграми представляє Test Environment – тестове оточення, що складається з трьох ключових компонентів: генератора навантаження для симуляції різної кількості користувачів, емулятора мережі для відтворення різних мережевих умов, та аналізатора метрик для обробки зібраних даних. З'єднувальні лінії між частинами демонструють взаємодію компонентів системи.

Система моніторингу продуктивності інтегрується в обидві версії додатку та забезпечує збір метрик у реальному часі. Відстежуються такі параметри як час відгуку, затримка мережі, кількість успішних та невдалих операцій. Особлива увага приділяється вимірюванню часу відновлення після збоїв та стабільності роботи при тривалому навантаженні.

Тестове оточення включає генератор навантаження, який дозволяє симулювати різні сценарії використання системи. Важливим компонентом є емулятор мережевих умов, що дозволяє відтворювати реалістичні ситуації з нестабільним підключенням, затримками та втратами пакетів. Це дає можливість оцінити поведінку обох технологій у складних мережевих умовах. На рисунку 2.7 показано внутрішню структуру WebSocket імплементації.

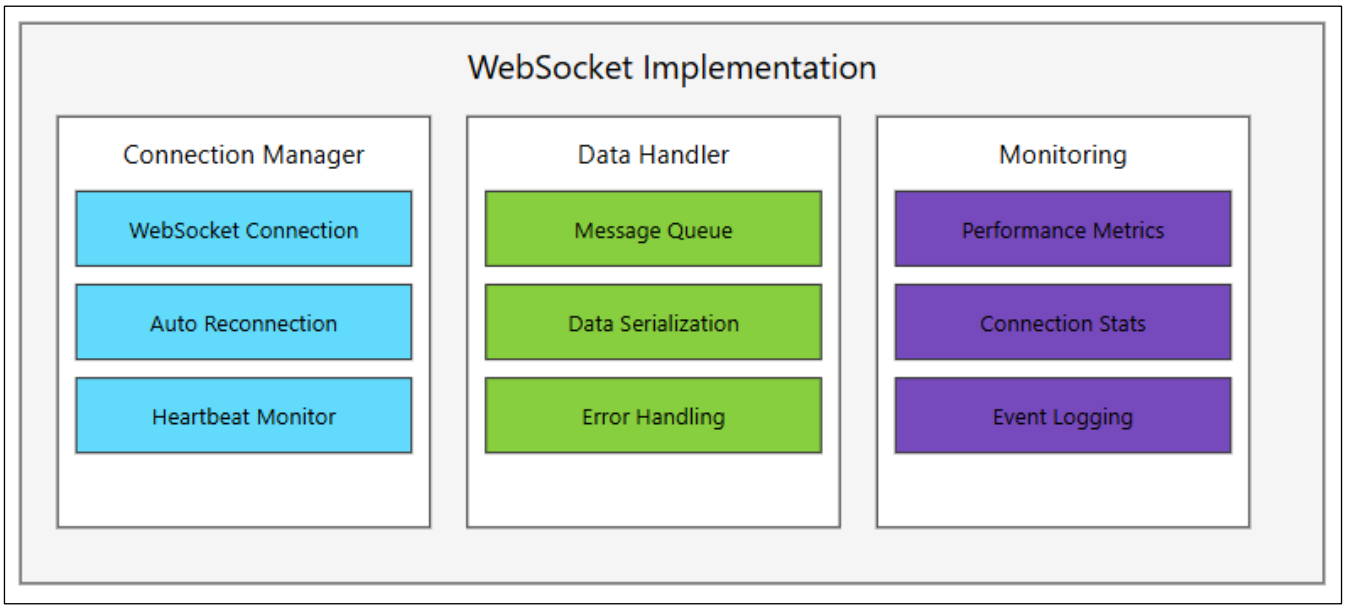

Рисунок 2.7 – Деталі реалізації WebSocket версії (рисунок створено самостійно)

Діаграма розділена на три основні блоки. Connection Manager відповідає за управління WebSocket з’єднанням, включаючи автоматичне відновлення зв’язку та моніторинг активності через heartbeat-повідомлення. Data Handler забезпечує обробку повідомлень, їх серіалізацію та управління чергою повідомлень. Monitoring блок відстежує продуктивність системи, збирає статистику з’єднань та веде журнал подій.

Особлива увага при проектуванні приділена масштабованості тестових додатків. Архітектура дозволяє легко змінювати параметри навантаження, додавати нові сценарії тестування та розширювати набір метрик, що збираються. Це забезпечує можливість проведення різноманітних експериментів без необхідності значних змін у кодовій базі. На рисунку 2.8 показано внутрішню структуру Long Polling імплементації.

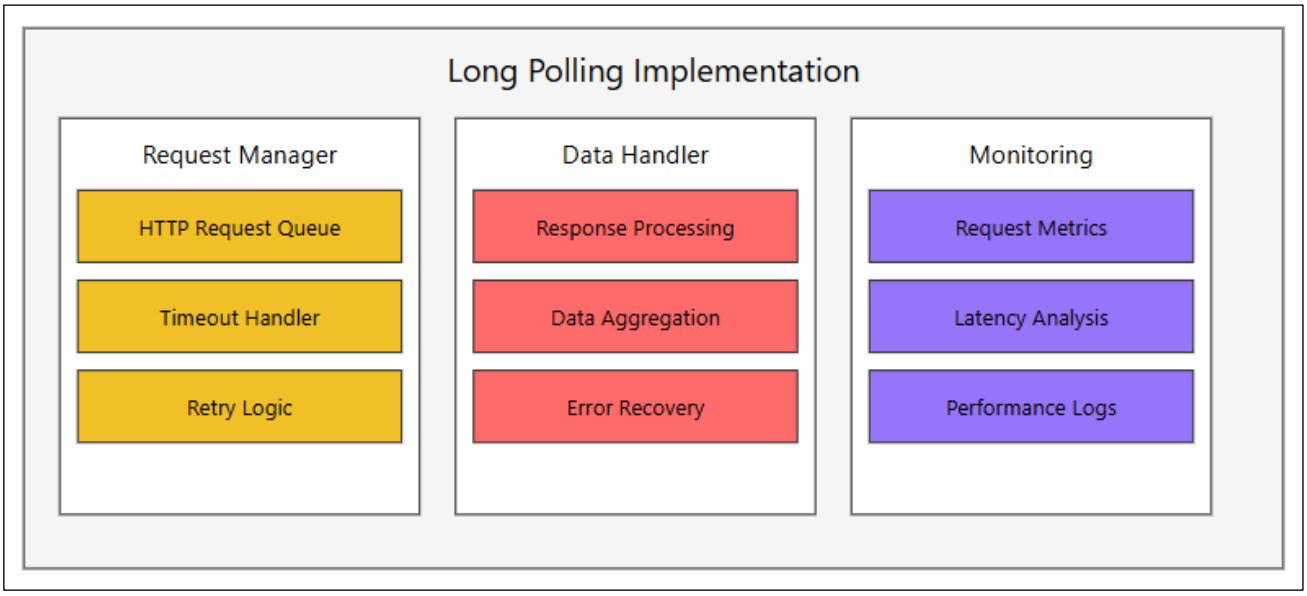

Рисунок 2.8 – Деталі реалізації Long Polling версії (рисунок створено самостійно)

На рисунку вище відображено особливості реалізації Long Polling підходу. Request Manager керує чергою HTTP-запитів, обробляє тайм-аути та реалізує логіку повторних спроб. Data Handler відповідає за обробку відповідей сервера, агрегацію даних та відновлення після помилок. Блок Monitoring, аналогічно до WebSocket версії, забезпечує збір метрик, але з акцентом на специфічні для Long Polling параметри, такі як латентність запитів та аналіз затримок.

Розроблена архітектура тестових додатків забезпечує необхідну базу для проведення комплексного порівняльного аналізу технологій WebSocket та Long Polling, дозволяючи отримати об’єктивні дані про їх ефективність у різних умовах експлуатації.

3 ПОРІВНЯННЯ WEBSOCKET ТА LONG POLLING ПІДХОДІВ

3.1 Архітектура веб-додатку для проведення експерименту

Для практичного порівняння технологій WebSocket та Long Polling було розроблено спеціалізований веб-додаток з модульною архітектурою, яка дозволяє проводити паралельне тестування обох підходів в ідентичних умовах.

Клієнтська частина розроблена з використанням сучасного стеку технологій, який включає React, TypeScript та SCSS. React використовується як основна бібліотека для побудови користувацького інтерфейсу обумовлений його компонентним підходом, що дозволяє створювати модульну структуру додатку з чітким розподілом відповідальності між компонентами. TypeScript вибрано для забезпечення статичної типізації, що значно підвищує якість коду та спрощує розробку складних систем з великою кількістю взаємодіючих компонентів. Використання SCSS дозволяє створювати структуровані та підтримувані стилі для користувацького інтерфейсу. Використовується не просто SCSS, який надає певні функціональні можливості, які відсутні у CSS, а модульний SCSS, який дозволяє уникнути проблеми з глобальними стилями.

Архітектура веб-додатку базується на таких принципах як модульність, розширюваність та тестованість. Модульність забезпечується поділом системи на незалежні компоненти з чітко визначеними інтерфейсами. Розширюваність досягається завдяки використанню абстрактних базових класів та інтерфейсів, які дозволяють легко додавати нові функціональні можливості. Тестованість забезпечується чітким розподілом бізнес-логіки та представлення, що дозволяє ефективно тестувати окремі компоненти системи.

До основних компонентів архітектури веб-додатку можна віднести наступне:

– React-компоненти інтерфейсу користувача: відповідають за відображення даних, включаючи результати тестування, графіки порівняння продуктивності та панель керування тестами. Для побудови

графіків використовується бібліотека Recharts, що базується на D3.js та надає широкі можливості для візуалізації даних у React-додатках;

- модуль тестування продуктивності: реалізований через класи PerformanceTester та TestSuiteRunner, які надають API для проведення тестів з різними параметрами мережевих умов та профілями навантаження. Модуль забезпечує автоматизоване виконання набору тестів та збір статистичних даних;
- комунікаційний шар: включає реалізації клієнтів для обох досліджуваних технологій (WebSocket та Long Polling) з уніфікованим інтерфейсом, що забезпечує справедливе порівняння. Для цього використовується базовий абстрактний клас BaseRealTimeClient та інтерфейс RealTimeCommunicationClient;
- модуль збору та аналізу метрик: відповідає за збір, обробку та візуалізацію даних про продуктивність обох технологій. Реалізований через структуру ConnectionStats та відповідні методи в клієнтських реалізаціях.

Взаємодія між компонентами системи організована за принципом подійно-орієнтованої архітектури. Комунікаційні клієнти генерують події про отримання повідомлень, зміну стану з’єднання та помилки, на які можуть підписуватися інші компоненти системи. Такий підхід забезпечує слабку зв’язаність між компонентами та підвищує гнучкість системи.

Для управління станом додатку використовується комбінація локального стану React-компонентів та централізованого стану тестування. Локальний стан використовується для управління UI-елементами, тоді як централізований стан містить результати тестування та поточний прогрес виконання тестів.

Взаємодія з користувачем реалізована через інтуїтивно зрозумілий інтерфейс, який надає можливість налаштовувати параметри тестування, запускати тести, переглядати результати в реальному часі та аналізувати зібрані метрики після завершення тестування.

Технічна реалізація комунікаційних клієнтів має такі особливості:

- WebSocket клієнт: використовує стандартний Web API для WebSocket з додатковими обгортками для обробки помилок, автоматичного перепідключення та збору метрик. Реалізований механізм heartbeat для визначення стану з'єднання та вимірювання латентності;
- Long Polling клієнт: базується на Fetch API для виконання HTTP-запитів з таймаутами та можливістю переривання довготривалих запитів. Реалізовано механізм послідовних запитів з обробкою різних типів відповідей та помилок.

Для збору метрик використовується комплексний підхід, який включає вимірювання базових показників (кількість повідомлень, розмір даних, час реакції) та розрахунок похідних метрик (пропускна здатність, відсоток втрати пакетів, стабільність з'єднання). Метрики оновлюються в реальному часі та доступні для аналізу як під час виконання тестів, так і після їх завершення.

3.2 Розробка WebSocket підходу

При розробці WebSocket підходу було створено клієнтську реалізацію, що базується на стандартному Web API для WebSocket, але з розширеними можливостями для моніторингу продуктивності, збору статистики та обробки мережевих проблем. WebSocket є сучасною технологією, яка забезпечує двонаправлене повнодуплексне з'єднання через єдиний TCP канал, що суттєво відрізняє її від традиційної моделі HTTP.

Архітектура розробленого WebSocket клієнта включає три основні компоненти: Connection Manager, Data Handler та Monitoring System. Connection Manager відповідає за повний життєвий цикл WebSocket з'єднання – від встановлення початкового з'єднання до обробки розривів та забезпечення автоматичного повторного підключення. Важливою функціональністю цього компонента є реалізація механізму експоненціальної затримки між спробами повторного підключення, що дозволяє ефективно відновлювати з'єднання при тимчасових мережевих проблемах без створення надмірного навантаження на сервер.

Data Handler забезпечує обробку даних, що передаються через з’єднання. Цей компонент виконує серіалізацію та десеріалізацію повідомлень, управління чергою повідомлень, генерацію унікальних ідентифікаторів та типізацію повідомлень. Реалізація підтримує різні типи повідомлень, включаючи звичайні дані, ping/pong повідомлення для вимірювання латентності та системні повідомлення для контролю стану з’єднання.

Monitoring System виконує збір та аналіз метрик продуктивності, що є критично важливим для об’єктивного порівняння технологій. Система вимірює латентність з’єднання, відстежує кількість відправлених та отриманих повідомлень, збирає статистику про помилки та перепідключення, а також розраховує комплексні показники стабільності роботи.

Розроблена реалізація WebSocket клієнта наслідує базовий клас BaseRealTimeClient та реалізує інтерфейс RealTimeCommunicationClient, що забезпечує уніфікований API для обох типів клієнтів та спрощує їх порівняння. Даний підхід дозволяє абстрагуватися від конкретних деталей реалізації та проводити об’єктивне порівняння технологій.

Ключовою особливістю реалізації є підхід до підключення, який забезпечує коректну обробку різних станів з’єднання та можливих помилок. WebSocket з’єднання встановлюється через стандартний конструктор WebSocket API, а потім налаштовуються обробники основних подій з’єднання: onopen, onmessage, onclose та onerror. Для забезпечення надійності та можливості відновлення після збоїв реалізовано механізм автоматичного повторного підключення з експоненціальною затримкою між спробами.

При розробці WebSocket підходу особлива увага була приділена забезпеченню надійності та стійкості до мережевих проблем. Реалізований механізм автоматичного відновлення з’єднання, обробки помилок та збору статистики дозволяє ефективно використовувати WebSocket навіть в умовах нестабільного з’єднання, що є важливим фактором для багатьох реальних сценаріїв використання.

3.3 Розробка Long Polling підходу

Реалізація Long Polling підходу суттєво відрізняється від WebSocket за архітектурою, оскільки використовує стандартні HTTP-запити для імітації двонаправленого каналу зв'язку. Long Polling є модифікацією традиційного підходу з опитуванням сервера, але замість регулярних коротких запитів клієнт відправляє запит, який сервер утримує відкритим до появи нових даних або до спрацювання таймауту. Після отримання відповіді клієнт одразу ініціює новий запит, забезпечуючи постійний канал для отримання оновлень.

Архітектура розробленого Long Polling клієнта також включає три основні компоненти: Request Manager, Data Handler та Monitoring System. Request Manager керує HTTP-запитами та відповідями, що є ключовим аспектом технології Long Polling. Цей компонент відповідає за формування та відправку запитів для отримання даних (polling), формування та відправку запитів для надсилання даних на сервер, керування таймаутами та обробку помилок.

Data Handler, як і в випадку з WebSocket, відповідає за обробку даних. Компонент виконує серіалізацію та десеріалізацію повідомлень, обробку отриманих даних, управління чергою повідомлень для відправки та забезпечення коректного форматування. Важливою відмінністю від WebSocket є необхідність обробки HTTP-відповідей, які можуть містити як одиночні повідомлення, так і масиви повідомлень, залежно від реалізації серверної частини.

Monitoring System також виконує збір та аналіз метрик продуктивності, але з урахуванням специфіки Long Polling. Система вимірює латентність HTTP-запитів, відстежує кількість запитів та відповідей, збирає статистику про помилки та повторні спроби, а також розраховує показники стабільності роботи, що дозволяє об'єктивно порівнювати ефективність з WebSocket.

Реалізація Long Polling клієнта, як і WebSocket клієнта, наслідує базовий клас BaseRealTimeClient та реалізує інтерфейс RealTimeCommunicationClient, що забезпечує уніфікований API та спрощує порівняння. Підключення та організація циклу опитування в Long Polling включає генерацію унікального ідентифікатора

сесії, скидання статистики, оновлення статусу підключення та запуск циклу опитування.

Ключовим елементом реалізації Long Polling є механізм опитування, реалізований у методі poll. Цей метод створює новий HTTP-запит з використанням Fetch API, додає необхідні параметри (ідентифікатор сесії, часова мітка) та відправляє запит на сервер. Для можливості переривання довготривалих запитів використовується AbortController. При отриманні відповіді виконується обробка різних сценаріїв: успішне отримання даних, відсутність нових даних (статус 204), помилки HTTP-запиту. Після обробки відповіді або в разі помилки метод рекурсивно викликає себе для підтримки постійного циклу опитування.

Особливу увагу при розробці Long Polling клієнта було приділено обробці помилок та забезпеченню надійності роботи. Реалізовано механізм обробки тайм-аутів запитів, автоматичне повторення запитів у випадку помилок з експоненціальною затримкою між спробами, можливість переривання запитів та відстеження стану сесії для забезпечення коректної роботи з сервером.

Збір статистики продуктивності Long Polling з'єднання включає ті ж метрики, що й для WebSocket: латентність, пропускну здатність, мережеві накладні витрати та стабільність. Однак специфіка Long Polling впливає на процес збору та інтерпретацію цих метрик. Зокрема, латентність включає не лише затримку мережі, але й час очікування нових даних на сервері, а пропускна здатність обмежена накладними витратами на створення нових HTTP-запитів.

3.4 Проведення експерименту

Для порівняння технологій WebSocket та Long Polling було проведено серію експериментів з використанням спеціально розробленого веб-додатку, який дозволяє візуалізувати та порівнювати ключові метрики продуктивності в реальному часі. Інтерфейс додатку представлено на рисунку 3.1.

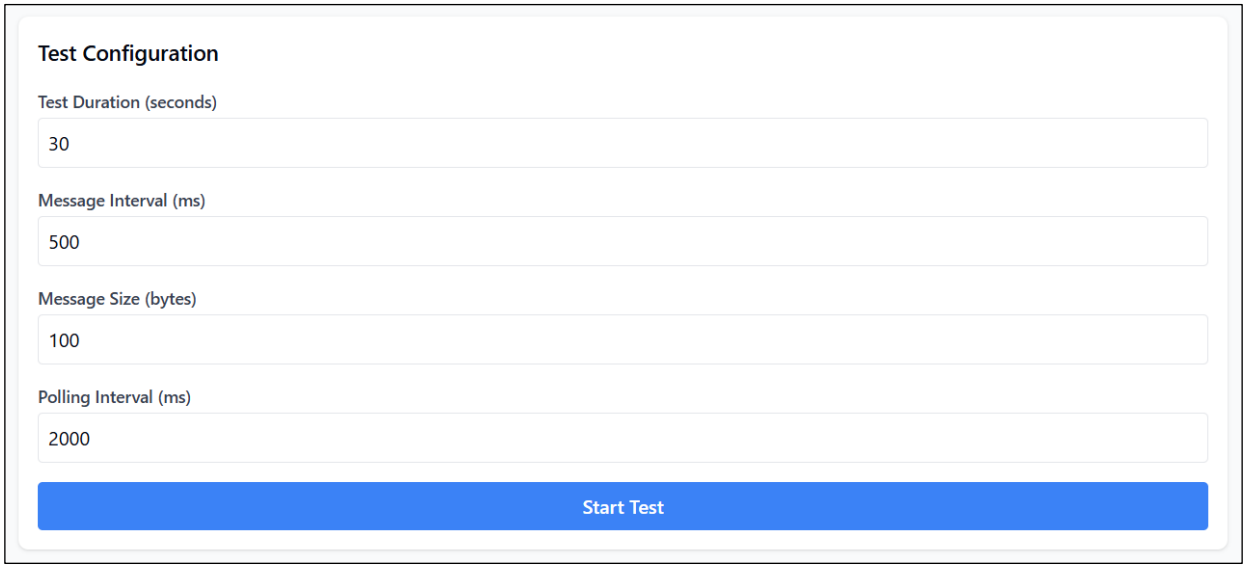

Рисунок 3.1 – Інтерфейс веб-додатку (рисунок створено самостійно)

Розроблений додаток забезпечує уніфіковану методологію тестування, дозволяючи гнучко налаштовувати параметри експерименту та збирати об'єктивні метрики продуктивності. Загальна методологія включала налаштування тестових параметрів, виконання тестів з різними профілями навантаження та збір комплексної статистики.

Інтерфейс додатку дозволяє налаштувати наступні параметри тестування:

– тривалість тесту: визначає загальний час виконання тесту в секундах, який для більшості експериментів встановлювався в діапазоні від 30 до 300 секунд;
– інтервал між повідомленнями: задає частоту відправки повідомлень через WebSocket у мілісекундах. Для різних сценаріїв навантаження використовувалися значення від 100 мс (висока частота) до 5000 мс (низька частота);
– розмір повідомлення: встановлює розмір корисного навантаження повідомлення у байтах. Тестування проводилося з різними розмірами від 10 байт (малі повідомлення) до 10000 байт (великі повідомлення);
– інтервал опитування: визначає частоту HTTP-запитів для технології Long Polling у мілісекундах. Використовувалися значення від 500 мс до 10000 мс для моделювання різної інтенсивності опитування.

Для збору метрик використовується комплексний підхід, який відстежує в реальному часі:

– латентність: час між відправкою запиту та отриманням відповіді, вимірюється в мілісекундах. Додаток відстежує як поточні значення, так і середнє значення латентності;
– обсяг переданих даних: кількість байт, відправлених на сервер;
– обсяг отриманих даних: кількість байт, отриманих від сервера;
– кількість повідомлень: загальна кількість успішно переданих повідомлень;
– статус з'єднання: поточний стан з'єднання для кожної з технологій.

Візуалізація метрик реалізована за допомогою бібліотеки Recharts, яка забезпечує інтерактивні графіки для аналізу латентності в реальному часі та порівняльні діаграми для ключових метрик продуктивності.

Результати тестування за основними метриками для WebSocket представлено на рисунку 3.2.

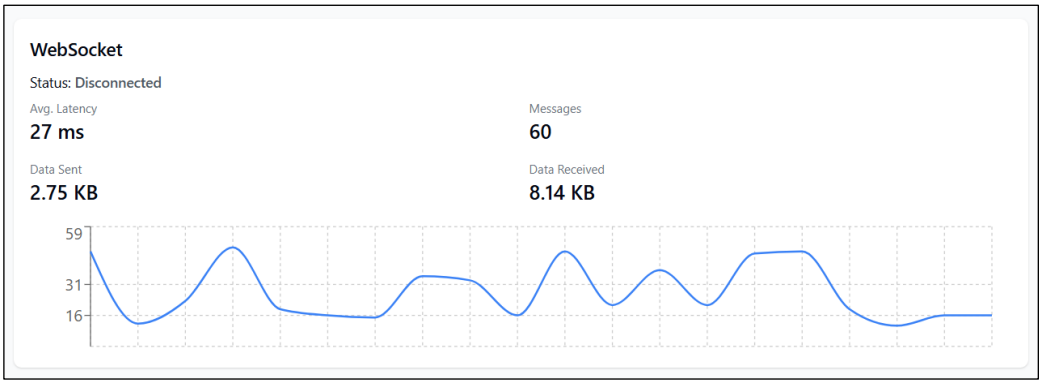

Рисунок 3.2 – Результати тестування для WebSocket (рисунок створено самостійно)

Результати тестування за основними метриками для Long Polling представлено на рисунку 3.3.

Перше, що буде розглянуто – латентність. Експерименти показали значну різницю в латентності між двома технологіями. WebSocket стабільно демонстрував нижчі показники затримки порівняно з Long Polling у всіх сценаріях тестування. При стандартному тестуванні з інтервалом повідомлень 500 мс і

розміром повідомлення 100 байт середня латентність склала: для WebSocket – 28 мс, для Long Polling – 137 мс.

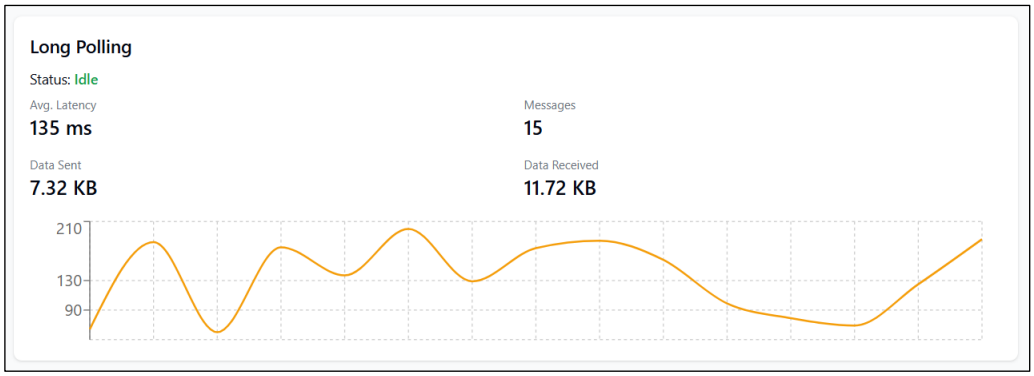

Рисунок 3.3 – Результати тестування для Long Polling (рисунок створено самостійно)

Важливим спостереженням є стабільність латентності. WebSocket демонстрував більш передбачувані значення з меншими коливаннями, тоді як Long Polling мав суттєві варіації латентності, особливо при збільшенні навантаження. Графіки латентності в реальному часі показали, що в періоди пікового навантаження різниця в латентності стає ще більш вираженою.

Наступний критерій – ефективність використання мережі. Порівняння мережевої ефективності показало значну перевагу WebSocket над Long Polling. При передачі однакової кількості корисних даних загальний мережевий трафік для кожної технології склав: для WebSocket 24.3 КБ, для Long Polling 62.7 КБ.

Така значна різниця (майже втричі) пояснюється низькими накладними витратами протоколу WebSocket після встановлення з’єднання, тоді як Long Polling генерує значний обсяг службового трафіку через постійну передачу HTTP-заголовків з кожним запитом і відповіддю.

При збільшенні розміру повідомлення до 1000 байт різниця у відносних накладних витратах зменшувалася, але WebSocket все одно залишався більш ефективним: для WebSocket 107.8 КБ (з них 9.2% службові дані), для Long Polling 164.5 КБ (з них 35.1% службові дані).

Після цього розглянемо – пропускну здатність. Тестування показало суттєву різницю в пропускній здатності (кількості повідомлень, які можуть бути успішно

передані за одиницю часу). За 30-секундний тест з максимальною частотою повідомлень (інтервал 100 мс) було передано: для WebSocket 293 повідомлення (≈ 9.77 повідомлень/секунду), для Long Polling 134 повідомлення (≈ 4.47 повідомлень/ секунду).

При збільшенні інтервалу опитування до 2000 мс для Long Polling кількість повідомлень знижувалася ще більше, до 15 повідомлень за 30-секундний тест (≈ 0.5 повідомлень/секунду), тоді як WebSocket зберігав стабільну продуктивність.

На основі проведених експериментів можна зробити висновок, що WebSocket демонструє кращі показники продуктивності за більшістю ключових метрик у порівнянні з Long Polling. Однак важливо відзначити, що вибір технології повинен базуватися на конкретних вимогах проекту і враховувати не лише продуктивність, але й такі фактори як сумісність з інфраструктурою та особливості сценарію використання.

До основних переваг WebSocket можна віднести наступне:

- суттєво нижча латентність;
- менші накладні витрати на мережевий трафік;
- менше навантаження на сервер;
- вища пропускна здатність.

До основних переваг Long Polling можна віднести наступне:

- краща сумісність з проксі-серверами та брандмауерами;
- працює з будь-яким сервером, що підтримує HTTP;
- не вимагає специфічної підтримки з боку інфраструктури;
- простота реалізації та обслуговування.

Проведене дослідження також підтверджує важливість врахування специфіки конкретного додатку при виборі технології. Для систем з високими вимогами до швидкості оновлення даних та ефективності використання ресурсів WebSocket є кращим вибором, тоді як Long Polling залишається цінною альтернативою для сценаріїв з обмеженнями на використання WebSocket або з низькою частотою оновлень.

4 РОЗРОБКА ГІБРИДНОГО ПІДХОДУ

4.1 Теоретичне обґрунтування гібридного підходу

Результати проведеного експериментального дослідження виявили принципову обмеженість використання будь-якої окремої технології для всіх можливих сценаріїв експлуатації Front-end додатків. WebSocket демонструє оптимальну продуктивність при стабільному з'єднанні та високій інтенсивності обміну даними, тоді як Long Polling виявляє кращу стійкість до мережевих проблем та універсальну сумісність з інфраструктурою. Аналіз реальних умов експлуатації веб-додатків показує, що параметри мережевого середовища є динамічними та непередбачуваними. Користувачі можуть переміщуватися між різними типами з'єднань, якість мережі може змінюватися протягом сесії, а характеристики обміну даними залежать від поведінки користувача та бізнес-логіки додатку.

Це обґрунтовує необхідність створення адаптивної системи, здатної використовувати оптимальну технологію для кожної конкретної ситуації, забезпечуючи максимальну ефективність у всьому спектрі умов експлуатації [20]. Гібридний підхід у контексті даного дослідження визначається як інтелектуальна система обміну повідомленнями, яка динамічно вибирає та застосовує найбільш ефективну технологію комунікації на основі аналізу контекстних параметрів середовища експлуатації в режимі реального часу.

Теоретичною основою для розробки гібридного підходу є концепція адаптивних обчислювальних систем, які можуть змінювати свою поведінку відповідно до змін середовища. У випадку систем обміну повідомленнями адаптація полягає у виборі технології, яка забезпечує найкращу продуктивність для поточних умов. Ключовими характеристиками такої системи є здатність реагувати на зміни умов експлуатації в режимі реального часу, забезпечення перемикання між технологіями без впливу на користувацький досвід, вибір технології на основі формалізованих критеріїв ефективності та забезпечення цілісності сесії при перемиканні.

Для реалізації гібридного підходу необхідно вирішити кілька ключових завдань. По-перше, потрібно розробити механізм моніторингу параметрів середовища, які впливають на ефективність різних технологій. Ці параметри включають якість мережевого з'єднання, що вимірюється через стабільність та відсоток пакетних втрат, латентність мережі з урахуванням часу кругового проходження та варіацій затримки, інтенсивність та характер обміну даними, включаючи частоту та розмір повідомлень.

По-друге, необхідно створити систему оцінки ефективності різних технологій в конкретних умовах. Ця система повинна враховувати всі ключові аспекти продуктивності, включаючи латентність доставки повідомлень, надійність комунікаційного каналу, пропускну здатність системи, ефективність використання мережевих та клієнтських ресурсів, а також масштабованість рішення. На основі цих оцінок система зможе обрати оптимальну технологію для поточних умов.

По-третє, потрібно розробити механізм безшовного перемикання між технологіями без втрати даних або переривання сервісу. Це завдання особливо важливе для систем реального часу, де критична безперервність обміну повідомленнями. Механізм повинен включати паралельну підтримку декількох каналів комунікації протягом перехідного періоду, синхронізацію стану між різними комунікаційними каналами, систему буферизації та відновлення повідомлень у випадку збоїв, а також протоколи підтвердження успішного перемикання між технологіями.

Гібридний підхід має значні переваги в різних сценаріях використання. У мобільних додатках система може використовувати WebSocket при хорошому з'єднанні та автоматично перемикатися на Long Polling при погіршенні якості мережі. У системах з великою кількістю клієнтів можливе динамічне балансування навантаження через вибір технології для кожного клієнта окремо. У системах з нерівномірною інтенсивністю обміну даними гібридний підхід дозволяє використовувати WebSocket в періоди активної взаємодії та переходити на Long Polling у періоди низької активності для економії ресурсів. Для

гетерогенних клієнтських платформ система може адаптуватися до специфіки кожного браузера та пристрою.

Однак реалізація гібридного підходу пов’язана з певними викликами. Збільшення складності системи може призвести до додаткових накладних витрат на управління процесом перемикання між технологіями. Синхронізація стану між різними каналами комунікації також потребує ретельного проектування. Процеси моніторингу параметрів та прийняття рішень споживають додаткові обчислювальні ресурси, що може вплинути на загальну продуктивність системи.

Подолання цих викликів вимагає розробки ефективної математичної моделі та алгоритму перемикання між технологіями. Теоретичний аналіз показує, що гібридний підхід повинен базуватися на формалізованій моделі прийняття рішень, яка враховує як поточний стан системи, так і його динаміку. Ключовим компонентом цієї моделі є функція оцінки ефективності, яка для кожної технології та набору параметрів середовища визначає очікувану продуктивність.

Для побудови такої функції необхідно формалізувати залежність між параметрами середовища та показниками ефективності технологій. Наприклад, латентність WebSocket можна виразити як функцію від базової латентності мережі, стабільності з’єднання та інтенсивності обміну даними. Аналогічно, для Long Polling латентність залежить від базової латентності мережі та частоти оновлення даних.

Функція оцінки ефективності повинна також враховувати пріоритети конкретної системи. Для деяких додатків критичною є мінімальна затримка, для інших – надійність доставки повідомлень або ефективне використання ресурсів. Ця функція може бути реалізована як зважена сума окремих показників ефективності з коефіцієнтами, що відображають їх важливість.

Для врахування динаміки зміни параметрів середовища необхідний механізм фільтрації та прогнозування. Фільтрація дозволяє згладжувати короткочасні коливання параметрів та виділяти суттєві тренди. Прогнозування дає можливість системі діяти проактивно, обираючи технологію з огляду на очікувані зміни умов.

Важливим аспектом є також механізм стабілізації, який запобігає частому перемиканню між технологіями при незначних змінах параметрів. Такий механізм може бути реалізований через використання гістерезису – технологія змінюється тільки якщо різниця в ефективності перевищує певний поріг. Теоретичне обґрунтування гібридного підходу вказує на необхідність розробки математичної моделі, яка формалізує процес вибору технології. Ця модель повинна враховувати всі ключові параметри, що впливають на ефективність різних технологій, а також динаміку їх зміни в часі.

4.2 Математична модель оптимального перемикання між технологіями

Для забезпечення ефективного функціонування гібридного підходу необхідно створити математичну модель, яка формалізує процес прийняття рішення щодо вибору оптимальної технології обміну повідомленнями в залежності від поточного контексту. Розроблена модель базується на результатах експериментального дослідження, що були отриманні у розділі 3.4 та аналізі параметрів середовища, які виявили найбільший вплив на продуктивність різних технологій згідно з дослідженнями [2, 7, 8].

У рамках даної моделі визначимо набір доступних технологій $T = \{WebSocket,\ LongPolling\}$. Контекст функціонування системи представимо як набір параметрів $C(t) = (c_1(t), c_2(t), c_3(t), c_4(t))$, де кожен параметр $c_i(t)$ характеризує певний аспект середовища в момент часу t:

- $c_1(t)$ – стабільність мережевого з’єднання, вимірюється як відсоток успішних exchange/ping запитів за останній період (наприклад, останні 20 запитів);
- $c_2(t)$ – латентність мережі, вимірюється в мілісекундах як середній час round-trip запитів;
- $c_3(t)$ – інтенсивність обміну повідомленнями, вимірюється як кількість повідомлень за секунду, розрахована на основі активності за попередній період;

– $c_4(t)$ – показник серверного навантаження, вимірюється через середній час обробки запитів на сервері (в мс), який включається в метадані відповіді: для Long Polling – як HTTP-заголовок, для WebSocket – як частина структури повідомлення.

Для кожної технології T_i та контексту $C(t)$ визначимо функцію ефективності $E(T_i, C(t))$ (формула 1), яка оцінює очікувану продуктивність системи при використанні технології T_i в контексті $C(t)$:

$$E(T_i, C(t)) = w_1 \cdot P_1(T_i, C(t)) + w_2 \cdot P_2(T_i, C(t)) + w_3 \cdot P_3(T_i, C(t)) + w_4 \cdot P_4(T_i, C(t)), \quad (1)$$

де $P_1(T_i, C(t))$ – очікувана латентність (негативний показник, менше значення краще),

$P_2(T_i, C(t))$ – очікувана надійність (ймовірність успішної доставки повідомлення),

$P_3(T_i, C(t))$ – очікувана пропускна здатність,

$P_4(T_i, C(t))$ – ефективність використання мережевих ресурсів,

w_1, w_2, w_3, w_4 – коефіцієнти, що відображають відносну важливість кожного показника.

Оскільки безпосереднє вимірювання параметрів контексту може бути неточним, в моделі використовується метод експоненціального згладжування для отримання більш стабільних оцінок (формула 2):

$$\hat{c}_i(t) = \alpha \cdot c_i(t) + (1 - \alpha) \cdot \hat{c}_i(t - 1), \quad (2)$$

де $\hat{c}_i(t)$ – згладжена оцінка параметра c_i в момент часу t,

α – коефіцієнт згладжування ($0 < \alpha < 1$).

Для прогнозування майбутніх значень параметрів використовується лінійна екстраполяція (формула 3):

$$\tilde{c}_i(t + \Delta t) = \hat{c}_i(t) + (\hat{c}_i(t) - \hat{c}_i(t - \Delta t)) \cdot (\Delta t/\tau), \quad (3)$$

де $\tilde{c}_i(t + \Delta t)$ – прогнозоване значення параметра c_i в момент часу $t + \Delta t$,

τ – часовий інтервал між вимірюваннями.

Використання прогнозованих значень дозволяє системі діяти проактивно, перемикаючись на більш ефективну технологію до того, як поточна технологія почне демонструвати низьку продуктивність.

Використовуючи функцію ефективності (1), можна сформулювати правило вибору оптимальної технології в момент часу t (формула 4):

$$T^*(t) = argmax\_\{T_i \in T\}\, E(T_i, C(t)), \quad (4)$$

Тобто, обирається технологія, яка максимізує очікувану ефективність для поточного контексту.

Для зменшення частоти перемикань між технологіями при незначних змінах параметрів контексту вводиться механізм гістерезису, який можна представити наступним чином (формула 5):

$$T^*(t) = \{argmax\_\{T_\text{i} \in T\}\, E(T_\text{i}, C(t)),\ \text{якщо}\ |E(argmax\_\{T_i \in T\}\, E(T_i, C(t)), C(t)) - E(T^*(t-1), C(t))| > H, T^*(t-1),$$
$$\text{в іншому випадку}\} \quad (5)$$

де H – поріг гістерезису, який визначає мінімальну різницю в ефективності, необхідну для зміни технології.

Важливою особливістю розробленої моделі є її здатність до самоадаптації. Параметри моделі не є статичними і можуть змінюватися з часом на основі даних про реальну продуктивність системи. Це дозволяє моделі враховувати особливості конкретного додатку та адаптуватися до змін у середовищі функціонування.

Розроблена математична модель є базою для створення алгоритму динамічного вибору технології на основі контексту, який буде реалізовано в прототипі гібридної системи обміну повідомленнями. Вона забезпечує формальне обґрунтування процесу прийняття рішень та дозволяє прогнозувати ефективність гібридного підходу в різних сценаріях використання.

Модель також може бути розширена для включення додаткових технологій обміну повідомленнями, таких як Server-Sent Events або HTTP/2 Server Push. Для цього необхідно розробити відповідні формули для оцінки показників ефективності цих технологій та включити їх у функцію ефективності.

4.3 Алгоритм динамічного вибору технології на основі контексту

На основі розробленої математичної моделі оптимального перемикання між технологіями створено алгоритм динамічного вибору, який забезпечує адаптацію системи до змін умов експлуатації в режимі реального часу. Алгоритм реалізує механізм моніторингу параметрів середовища, оцінку ефективності кожної технології та прийняття рішення щодо вибору оптимальної технології з урахуванням гістерезису.

Основні компоненти алгоритму включають систему збору даних про поточний контекст, модуль фільтрації та згладжування параметрів, модуль прогнозування, модуль оцінки ефективності та модуль прийняття рішень. Кожен з цих компонентів виконує специфічні функції, які в сукупності забезпечують ефективну адаптацію системи до змін умов експлуатації.

Система збору даних про поточний контекст відповідає за моніторинг ключових параметрів середовища, які впливають на ефективність технологій обміну повідомленнями. Ці параметри включають якість мережевого з’єднання, інтенсивність обміну даними, доступні ресурси клієнта та сервера. Для вимірювання якості мережевого з’єднання використовуються механізми активного зондування, які періодично відправляють невеликі тестові повідомлення та вимірюють час їх доставки.

Модуль фільтрації та згладжування параметрів (2) забезпечує стабільність оцінок параметрів контексту шляхом усунення короткочасних коливань та виділення суттєвих трендів. Для цього використовується метод експоненціального згладжування, який надає більшу вагу недавнім спостереженням та поступово зменшує вплив старих даних.

Модуль прогнозування відповідає за передбачення майбутніх значень параметрів контексту на основі їх поточних значень та тенденцій зміни. Використання прогнозованих значень дозволяє системі діяти проактивно, перемикаючись на більш ефективну технологію до того, як поточна технологія почне демонструвати низьку продуктивність. Для прогнозування використовується метод лінійної екстраполяції (3).

Модуль оцінки ефективності обчислює очікувану ефективність кожної технології для поточного та прогнозованого контексту з використанням функції ефективності (1).

Модуль прийняття рішень аналізує оцінки ефективності різних технологій та обирає оптимальну технологію з урахуванням механізму гістерезису, який запобігає частому перемиканню між технологіями (5).

На рисунку 4.1 можна побачити діаграму, що ілюструє алгоритм динамічного вибору технології на основі контексту.

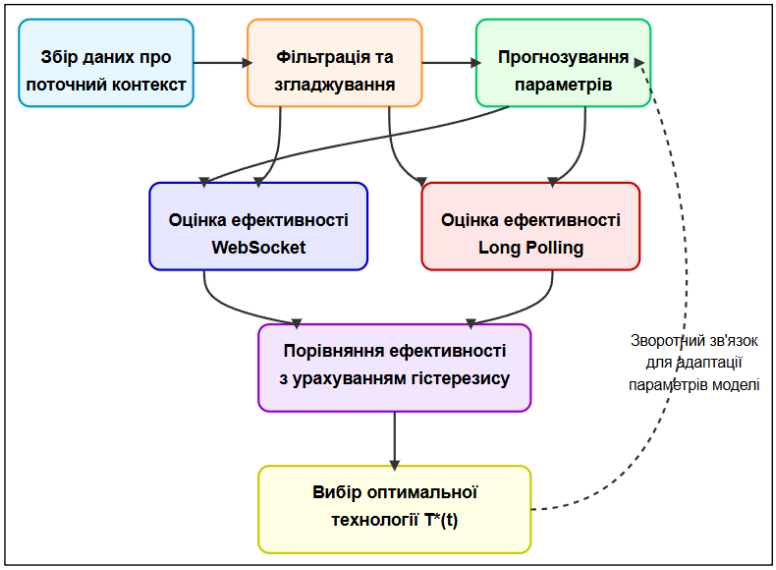

Рисунок 4.1 – Алгоритм динамічного вибору технології на основі контексту (рисунок створено самостійно)

Алгоритм складається з наступних основних компонентів:

- збір даних про поточний контекст: якість мережевого з’єднання (стабільність), вимірюється як відсоток успішних обмінів даними;
- фільтрація та згладжування: обробка зібраних даних для усунення короткочасних коливань та виділення суттєвих трендів. Використовується метод експоненціального згладжування, який надає більшу вагу недавнім спостереженням;
- прогнозування параметрів: передбачення майбутніх значень параметрів контексту на основі їх поточних значень та тенденцій зміни. Це дозволяє системі діяти проактивно, перемикаючись на більш ефективну технологію до того, як поточна технологія почне демонструвати низьку продуктивність;
- оцінка ефективності WebSocket та Long Polling: обчислення очікуваної ефективності кожної технології для поточного та прогнозованого контексту з використанням функції ефективності, яка враховує такі показники як латентність, надійність, пропускна здатність та використання ресурсів;
- порівняння ефективності з урахуванням гістерезису: аналіз оцінок ефективності різних технологій та визначення, чи перевищує різниця в ефективності поріг гістерезису. Механізм гістерезису запобігає частому перемиканню між технологіями при незначних змінах параметрів контексту;
- вибір оптимальної технології: прийняття рішення щодо використання конкретної технології (WebSocket або Long Polling) на основі результатів порівняння ефективності та з урахуванням обмежень гістерезису. Система реалізує безшовне перемикання між технологіями, забезпечуючи збереження стану сесії та безперервність обміну даними.

Важливим компонентом алгоритму є зворотний зв’язок, який забезпечує адаптацію параметрів моделі на основі реальної продуктивності системи. Це

дозволяє системі покращувати точність оцінок ефективності з часом та адаптуватися до специфіки конкретного додатку.

Псевдокод алгоритму для зміни технології представлений на рисунку 4.2, де формалізовано процес прийняття рішень щодо вибору оптимальної технології з урахуванням усіх описаних компонентів. Алгоритм реалізує циклічний процес моніторингу, аналізу, прогнозування та адаптації, забезпечуючи оптимальну продуктивність системи в динамічному середовищі експлуатації.

```
ініціалізувати параметри моделі та порогове значення гістерезису H
встановити початкову технологію T*(0)
цикл для кожного моменту часу t:
    зібрати дані про поточний контекст C_raw(t):
        c_1(t) - стабільність мережевого з'єднання (% успішних обмінів)
        c_2(t) - латентність мережі (мс)
        c_3(t) - інтенсивність обміну повідомленнями (повідомлень/с)
        c_4(t) - показник серверного навантаження (мс)

    для кожного параметра i від 1 до 4:
        застосувати фільтрацію:
        ĉ_i(t) = α·c_i(t) + (1-α)·ĉ_i(t-1)

        застосувати прогнозування:
        c̃_i(t+Δt) = ĉ_i(t) + (ĉ_i(t) - ĉ_i(t-Δt))·(Δt/τ)

    сформувати вектор згладжених параметрів Ĉ(t) = (ĉ_1(t), ĉ_2(t), ĉ_3(t), ĉ_4(t))
    сформувати вектор прогнозованих параметрів C̃(t+Δt) = (c̃_1(t+Δt), c̃_2(t+Δt), c̃_3(t+Δt), c̃_4(t+Δt))

    обчислити оцінку ефективності для WebSocket:
    E(WebSocket, Ĉ(t)) = w₁·P₁(WebSocket, Ĉ(t)) + w₂·P₂(WebSocket, Ĉ(t)) +
                         w₃·P₃(WebSocket, Ĉ(t)) + w₄·P₄(WebSocket, Ĉ(t))

    обчислити оцінку ефективності для Long Polling:
    E(LongPolling, Ĉ(t)) = w₁·P₁(LongPolling, Ĉ(t)) + w₂·P₂(LongPolling, Ĉ(t)) +
                           w₃·P₃(LongPolling, Ĉ(t)) + w₄·P₄(LongPolling, Ĉ(t))

    визначити технологію з максимальною ефективністю:
    T_max = argmax_{T_i ∈ T} E(T_i, Ĉ(t))

    якщо |E(T_max, Ĉ(t)) - E(T*(t-1), Ĉ(t))| > H:
        T*(t) = T_max
    інакше:
        T*(t) = T*(t-1)

    застосувати технологію T*(t)

    зібрати дані про реальну продуктивність
    оновити параметри моделі на основі зібраних даних
кінець циклу
```

Рисунок 4.2 – Псевдокод алгоритму для зміни технології (рисунок створено самостійно)

Для забезпечення безшовного перемикання між технологіями без втрати даних або переривання сервісу, алгоритм реалізує механізм паралельної підтримки обох каналів комунікації протягом певного перехідного періоду. При

перемиканні з WebSocket на Long Polling система продовжує прослуховувати WebSocket-з’єднання до отримання підтвердження успішного встановлення Long Polling-з’єднання. Аналогічно, при перемиканні з Long Polling на WebSocket система продовжує виконувати Long Polling-запити до успішного встановлення WebSocket-з’єднання.

Важливою особливістю алгоритму є його здатність враховувати специфіку різних типів даних. Наприклад, для критично важливих повідомлень з низькою частотою оновлення система може віддавати перевагу надійності доставки, тоді як для некритичних повідомлень з високою частотою оновлення більш важливим фактором може бути латентність та ефективність використання ресурсів.

Алгоритм передбачає можливість налаштування вагових коефіцієнтів w_1, w_2, w_3, w_4, w_5 для різних типів додатків. Наприклад, для онлайн-ігор критичними параметрами є латентність та пропускна здатність, тоді як для систем моніторингу більш важливими є надійність доставки та ефективність використання ресурсів.

Для підвищення точності оцінок ефективності алгоритм використовує техніки машинного навчання для ідентифікації патернів у зміні параметрів контексту та їх впливу на продуктивність системи. Це дозволяє системі з часом покращувати якість прогнозування та приймати більш обґрунтовані рішення.

4.4 Архітектура адаптивної системи обміну повідомленнями

На основі розробленої математичної моделі та алгоритму динамічного вибору технології створено архітектуру адаптивної системи обміну повідомленнями. Ця архітектура забезпечує гнучкість, масштабованість та ефективну адаптацію до змін контексту під час обміну даними між клієнтом та сервером, що є критично важливим для сучасних розподілених додатків з високими вимогами до інтерактивності та надійності передачі даних.

Адаптивна система базується на модульній архітектурі з чітким розподілом відповідальності між компонентами, що дозволяє легко розширювати та модифікувати систему. Такий підхід дозволяє не лише ізолювати зміни в окремих

компонентах, але й підтримувати паралельну розробку та тестування різних частин системи.

Центральним елементом архітектури є менеджер комунікацій, який керує процесом вибору та перемикання між різними технологіями обміну повідомленнями, але й координує взаємодію всіх компонентів. На рисунку 4.3 представлено загальну архітектуру адаптивної системи обміну повідомленнями.

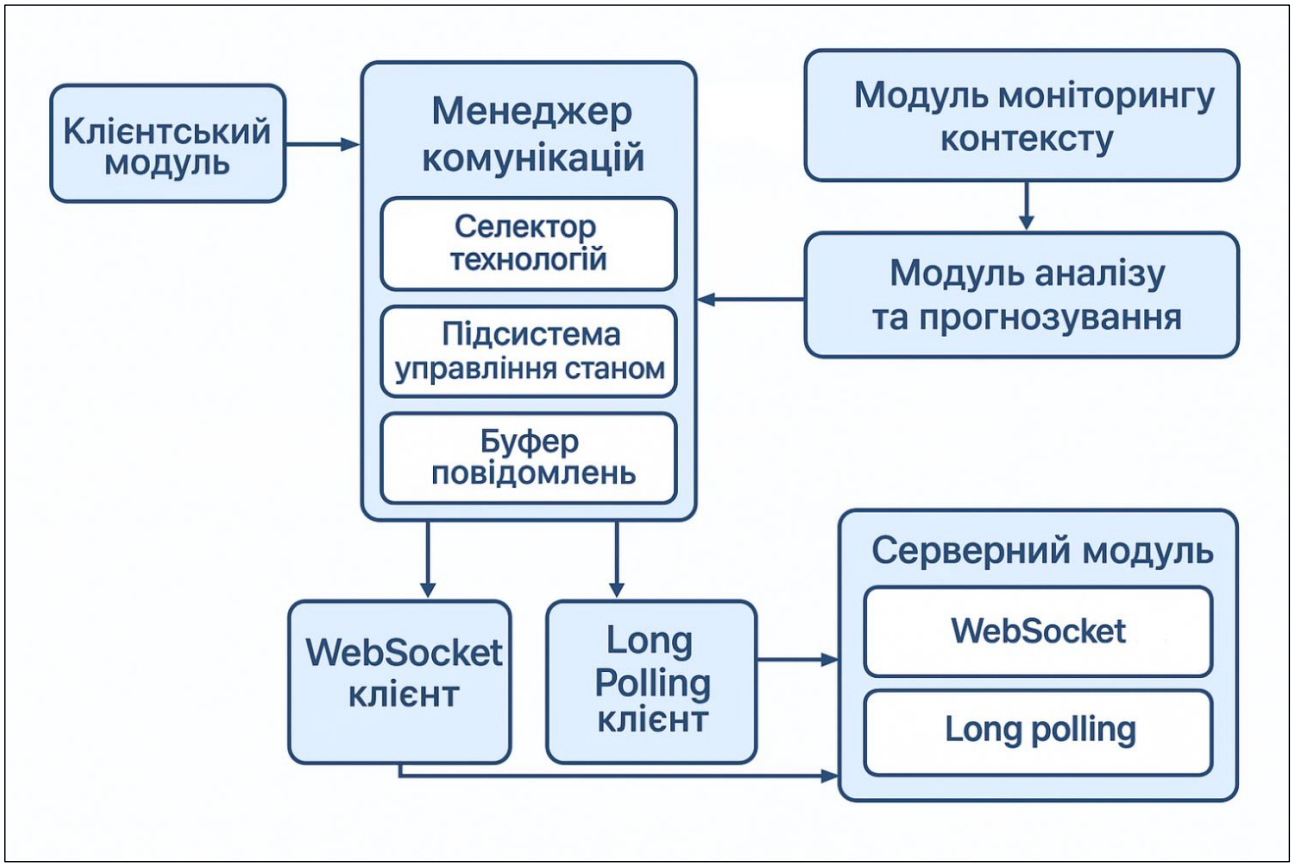

Рисунок 4.3 – Архітектура адаптивної системи обміну повідомленнями (рисунок створено самостійно)

Основні компоненти архітектури адаптивної системи включають клієнтський модуль, який забезпечує взаємодію з користувачем та відображення даних. Цей модуль абстрагований від конкретної технології обміну повідомленнями та працює з єдиним програмним інтерфейсом незалежно від того, яка технологія використовується в даний момент.

Менеджер комунікацій відповідає за координацію роботи різних комунікаційних клієнтів, вибір оптимальної технології на основі даних

моніторингу та забезпечення безшовного перемикання між технологіями. Цей компонент реалізує алгоритм динамічного вибору технології та координує процес передачі стану між різними комунікаційними каналами. Менеджер складається з трьох підкомпонентів: селектор технологій відповідає за прийняття рішень про вибір оптимальної технології, підсистема управління станом забезпечує збереження та відновлення стану сесії, буфер повідомлень гарантує безшовне перемикання без втрати даних.

Комунікаційні клієнти представлені окремими модулями для кожної підтримуваної технології, які відповідають за безпосередню взаємодію з сервером через відповідні протоколи. Кожен клієнт інкапсулює специфіку роботи з конкретною технологією та надає уніфікований інтерфейс для менеджера комунікацій. WebSocket клієнт включає управління з’єднанням, обробку повідомлень та моніторинг стану. Long Polling клієнт містить управління запитами, обробку відповідей та управління таймаутами.

Модуль моніторингу контексту збирає дані про поточні параметри середовища, такі як якість мережевого з’єднання, інтенсивність обміну даними, доступні ресурси клієнта та сервера. Цей модуль також відповідає за фільтрацію та згладжування параметрів для забезпечення стабільних оцінок. Компонент включає збір метрик мережі, моніторинг ресурсів та фільтрацію даних.

Модуль аналізу та прогнозування на основі даних моніторингу оцінює ефективність різних технологій та прогнозує їх майбутню продуктивність. Цей модуль реалізує математичну модель оптимального перемикання між технологіями та надає рекомендації менеджеру комунікацій. Підсистеми включають оцінку ефективності, прогнозування параметрів та рекомендації перемикання.

Модуль управління станом забезпечує збереження та відновлення стану комунікаційної сесії при перемиканні між технологіями. Цей компонент критично важливий для забезпечення безшовного перемикання без втрати даних або порушення порядку їх доставки.

Серверний модуль підтримує обидві технології обміну повідомленнями та забезпечує єдиний програмний інтерфейс для клієнтських додатків, незалежно від технології, що використовується. Включає обробники для WebSocket та HTTP, а також спільну бізнес-логіку.

Взаємодія між компонентами організована за принципом подійно-орієнтованої архітектури. Процес роботи системи включає етапи ініціалізації, нормальної роботи, перемикання технологій та обробки помилок. При ініціалізації менеджер комунікацій реєструє обробники подій, клієнти ініціалізуються та модуль моніторингу починає збір метрик.

Під час нормальної роботи модуль моніторингу періодично збирає параметри контексту, модуль аналізу обчислює ефективність технологій, а при перевищенні порога гістерезису ініціюється перемикання. Процес перемикання включає активацію нового клієнта, паралельну роботу каналів, синхронізацію стану та деактивацію старого каналу.

При обробці помилок система генерує відповідні події, аналізує тип помилки та приймає рішення про перемикання або повторну спробу. Критичні помилки призводять до екстреного перемикання на резервну технологію.

Архітектура забезпечує можливість легкого розширення для підтримки додаткових технологій через реалізацію нових комунікаційних клієнтів та розширення модуля аналізу. Система включає механізми конфігурації для різних типів додатків, дозволяючи налаштовувати вагові коефіцієнти, пороги гістерезису та інтервали моніторингу.

Важливими аспектами архітектури є забезпечення надійності через автоматичне відновлення, резервування компонентів, журналювання операцій та моніторинг стану системи. Вимоги до продуктивності враховуються через асинхронне виконання операцій, ефективні структури даних та кешування.

Розроблена архітектура адаптивної системи обміну повідомленнями представляє собою комплексне рішення, яке успішно поєднує переваги технологій WebSocket та Long Polling через інтелектуальний механізм динамічного перемикання. Модульна структура системи забезпечує чітке

розділення відповідальності між компонентами: менеджер комунікацій координує роботу різних технологій, модулі моніторингу та аналізу забезпечують прийняття обґрунтованих рішень на основі реальних параметрів середовища, а спеціалізовані клієнти інкапсулюють особливості роботи з конкретними протоколами.

Ключовими перевагами запропонованої архітектури є адаптивність до змін умов експлуатації без втручання користувача, масштабованість через можливість легкого додавання нових технологій комунікації, надійність завдяки механізмам резервування та автоматичного відновлення, а також ефективність використання ресурсів через оптимальний вибір технології для кожної конкретної ситуації. Подійно-орієнтована архітектура забезпечує слабку зв’язаність компонентів та високу гнучкість системи.

Практичне значення розробленої архітектури полягає у можливості її застосування в широкому спектрі Front-end додатків – від систем реального часу з критичними вимогами до латентності до корпоративних рішень з високими вимогами до сумісності з існуючою інфраструктурою. Архітектура забезпечує оптимальний баланс між гнучкістю, продуктивністю та надійністю, дозволяючи розробникам створювати системи, які автоматично адаптуються до специфічних вимог різних типів додатків та динамічно змінюваних умов експлуатації.

ВИСНОВКИ

Проведене дослідження було спрямоване на порівняльний аналіз ефективності технологій WebSocket та Long Polling для організації обміну повідомленнями у Front-end додатках, а також на розробку гібридного підходу, який би поєднував переваги обох технологій.

В результаті аналізу предметної області виявлено ключові особливості досліджуваних технологій. WebSocket забезпечує двонаправлений повнодуплексний канал зв'язку через TCP-з'єднання з низькою латентністю та ефективним використанням мережевих ресурсів. Long Polling базується на стандартному HTTP-протоколі та імітує push-повідомлення через механізм тривалих запитів, що забезпечує кращу сумісність з існуючою інфраструктурою.

Експериментальне дослідження показало значну різницю в ефективності технологій у різних умовах експлуатації. При стабільному з'єднанні WebSocket демонструє суттєво нижчу латентність у порівнянні з Long Polling та вищу пропускну здатність. При цьому WebSocket генерує значно менше мережевого трафіку завдяки відсутності накладних витрат на передачу HTTP-заголовків.

Розроблено теоретичне обґрунтування гібридного підходу, який дозволяє динамічно перемикатися між технологіями в залежності від контексту використання. Створено математичну модель оптимального перемикання, яка враховує різні параметри середовища: якість мережевого з'єднання, латентність, інтенсивність обміну повідомленнями, обсяг даних та доступність ресурсів.

Запропоновано алгоритм динамічного вибору технології, який забезпечує адаптацію системи до змін умов експлуатації в режимі реального часу. Алгоритм включає збір даних про поточний контекст, фільтрацію та згладжування параметрів, прогнозування майбутніх значень, оцінку ефективності кожної технології та прийняття рішення з урахуванням гістерезису.

Спроектовано архітектуру адаптивної системи обміну повідомленнями, яка включає ключові компоненти: клієнтський модуль, менеджер комунікацій, модуль моніторингу, модуль аналізу, WebSocket клієнт, Long Polling клієнт та серверний модуль. Розроблена архітектура забезпечує чітке розділення обов'язків

та ефективну взаємодію між компонентами для досягнення оптимальної продуктивності системи.

Визначено оптимальну взаємодію між компонентами системи, де всі метрики спочатку надходять до модуля аналізу, який обробляє їх і передає вже готові рекомендації менеджеру комунікацій. Така послідовність забезпечує чистішу архітектуру з єдиною точкою прийняття рішень та зменшує дублювання логіки.

Практичним результатом дослідження є створення набору рекомендацій та критеріїв, які допомагають розробникам приймати обґрунтовані рішення щодо вибору технології для реалізації систем сповіщень у Front-end додатках. WebSocket рекомендується для систем з високою частотою обміну даними, високими вимогами до латентності та стабільного мережевого з’єднання. Long Polling є оптимальним для систем з нерегулярними оновленнями, обмеженнями на використання WebSocket та середовищ з високими вимогами до сумісності.

Результати проведеного дослідження було представлено науковій спільноті у вигляді доповіді на XXIX Міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті» (Харків, 2025), де опубліковано тези «Особливості обміну повідомленнями в режимі реального часу». Публікація містить основні висновки порівняльного аналізу технологій WebSocket та Long Polling, що підтверджує актуальність та наукову цінність проведеного дослідження для міжнародної наукової спільноти у сфері веб-технологій та систем реального часу.

Перспективними напрямками подальших досліджень є розширення гібридного підходу для підтримки додаткових технологій обміну повідомленнями (Server-Sent Events, HTTP/2 Server Push), застосування методів машинного навчання для оптимізації процесу вибору технології та розробка методів динамічного балансування навантаження з використанням різних технологій.

Розроблений гібридний підхід до організації обміну повідомленнями у Front-end додатках дозволяє досягти оптимального балансу між швидкодією, надійністю та ефективністю використання ресурсів у різних умовах експлуатації,

що підвищує якість користувацького досвіду та забезпечує стабільну роботу додатків у динамічному та гетерогенному середовищі сучасного Web.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Wang V., Salim F., Moskovits P. The Definitive Guide to HTML5 WebSocket. Apress, 2016. 200 p. URL: https://pepa.holla.cz/wp-content/uploads/2016/01/The-Definitive-Guide-to-HTML5-WebSocket.pdf (дата звернення: 24.04.2025).

2. Pimentel V., Nickerson B. G. Communicating and Displaying Real-Time Data with WebSocket // IEEE Internet Computing. 2012. Vol. 16(4). P. 45-53. (дата звернення: 26.04.2025).

3. Fette I., Melnikov A. The WebSocket Protocol. Internet Engineering Task Force (IETF), 2011. URL: https://datatracker.ietf.org/doc/html/rfc6455 (дата звернення: 28.04.2025).

4. Lombardi A. WebSocket: Lightweight Client-Server Communications. O'Reilly Media, 2015. 143 p. URL: https://pepa.holla.cz/wp-content/uploads/2016/12/WebSocket.pdf (дата звернення: 01.05.2025).

5. Loreto S., Saint-Andre P., Salsano S. Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP. Internet Engineering Task Force (IETF), 2021. URL: https://datatracker.ietf.org/doc/html/rfc6202 (дата звернення: 03.05.2025).

6. Skvorc D., Horvat M., Srbljic S. Performance evaluation of Websocket protocol for implementation of full-duplex web streams // 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). 2014. P. 1003-1008. DOI: 10.1109/MIPRO.2014.6859715 (дата звернення: 07.05.2025).

7. Long Polling vs WebSockets: What’s best for realtime at scale? Ably, 2023. URL: https://ably.com/blog/websockets-vs-long-polling (дата звернення: 10.05.2025).

8. Long Polling vs WebSockets: A Developer’s Guide to Real-Time Solutions. VideoSDK, 2024. URL: https://www.videosdk.live/developer-hub/websocket/long-polling-vs-websocket (дата звернення: 12.05.2025).

9. Liu Q.G., Sun X.Y. Research of Web Real-Time Communication Based on Web Socket // International Journal of Communications, Network and System Sciences.

2012. Vol. 5. No. 12. P. 797-801. DOI: 10.4236/ijcns.2012.512083 (дата звернення: 14.05.2025).

10. WebSockets vs Server-Sent-Events vs Long-Polling vs WebRTC vs WebTransport. RxDB, 2024. URL: https://rxdb.info/articles/websockets-sse-polling-webrtc-webtransport.html (дата звернення: 16.05.2025).

11. Mozilla Developer Network. WebSocket API // MDN Web Docs. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 18.05.2025).

12. Stack Overflow. At what point are WebSockets less efficient than Polling? 2023. URL: https://stackoverflow.com/questions/44731313/at-what-point-are-websockets-less-efficient-than-polling (дата звернення: 20.05.2025).

13. JavaScript.info. Long polling // The Modern JavaScript Tutorial. 2024. URL: https://javascript.info/long-polling (дата звернення: 22.05.2025).

14. Svix Resources. Long Polling vs Websockets // Svix. 2024. URL: https://www.svix.com/resources/faq/long-polling-vs-websockets/ (дата звернення: 24.05.2025).

15. MaybeWorks. WebSocket: Pros, Cons, and Limitations // MaybeWorks Blog. 2024. URL: https://maybe.works/blogs/websocket-what-it-is-when-to-use (дата звернення: 26.05.2025).

16. HTTP/2 Server Push // MDN Web Docs. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/HTTP2_server_push (дата звернення: 27.05.2025).

17. HTTP/3 explained // CloudFlare Learning Center. 2024. URL: https://www.cloudflare.com/learning/performance/what-is-http3/ (дата звернення: 28.05.2025).

18. WebRTC Data Channels // MDN Web Docs. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Using_data_channels (дата звернення: 30.05.2025).

19. Server-Sent Events // MDN Web Docs. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events (дата

звернення: 31.05.2025).

20. Черних І.А., Лановий О.Ф. Особливості обміну повідомленнями в режимі реального часу. Радіоелектроніка та молодь у XXI столітті: Матеріали XXIX Міжнародного молодіжного форуму (16-19 квітня 2025 р.). Т. 6: Інформаційні інтелектуальні системи. Харків: ХНУРЕ, 2025. С. 349-351. URL: https://nure.ua/konferencii-ta-workshops/mizhnarodnij-molodizhnij-forum-radioelektronika-i-molod-u-hhi-stolitti/xxix-mizhnarodnyj-molodizhnyj-forum-radioelektronika-i-molod-u-khkhi-stolitti (дата звернення: 02.06.2025).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

20. Черних І.А., Лановий О.Ф. Особливості обміну повідомленнями в режимі реального часу. Радіоелектроніка та молодь у XXI столітті: Матеріали XXIX Міжнародного молодіжного форуму (16-19 квітня 2025 р.). Т. 6: Інформаційні інтелектуальні системи. Харків: ХНУРЕ, 2025. С. 349-351. URL: https://nure.ua/konferencii-ta-workshops/mizhnarodnij-molodizhnij-forum-radioelektronika-i-molod-u-hhi-stolitti/xxix-mizhnarodnyj-molodizhnyj-forum-radioelektronika-i-molod-u-khkhi-stolitti (дата звернення: 02.06.2025).