ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

StrikePlagiarism.com

Дата звіту 6/6/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_M_ПІ_ІПЗм-23-4_Черних_І_А_скорочений
Автор
Черних Ігор Андрійович
Науковий керівник / Експерт
Олена Олійник
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

7.29% 7.29% КП 1

0.18% 0.18% КЦ

25	11956	96870
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		5
Інтервали		0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)	a	32

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://openarchive.nure.ua/server/api/core/bitstreams/26b43333-068d-401b-94c8-f748f9f29dbd/content	61 0.51 %
2	https://openarchive.nure.ua/server/api/core/bitstreams/26b43333-068d-401b-94c8-f748f9f29dbd/content	48 0.40 %
3	https://openarchive.nure.ua/server/api/core/bitstreams/26b43333-068d-401b-94c8-f748f9f29dbd/content	47 0.39 %
4	https://openarchive.nure.ua/server/api/core/bitstreams/26b43333-068d-401b-94c8-f748f9f29dbd/content	39 0.33 %
5	https://openarchive.nure.ua/server/api/core/bitstreams/26b43333-068d-401b-94c8-f748f9f29dbd/content	39 0.33 %

6	https://openarchive.nure.ua/server/api/core/bitstreams/26b43333-068d-401b-94c8-f748f9f29dbd/content	38 0.32 %
7	https://openarchive.nure.ua/server/api/core/bitstreams/26b43333-068d-401b-94c8-f748f9f29dbd/content	37 0.31 %
8	https://openarchive.nure.ua/server/api/core/bitstreams/26b43333-068d-401b-94c8-f748f9f29dbd/content	35 0.29 %
9	https://openarchive.nure.ua/server/api/core/bitstreams/26b43333-068d-401b-94c8-f748f9f29dbd/content	35 0.29 %
10	https://openarchive.nure.ua/server/api/core/bitstreams/26b43333-068d-401b-94c8-f748f9f29dbd/content	35 0.29 %

з бази даних RefBooks (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з домашньої бази даних (0.06 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	2025_М_ІНФ_ІНФм-23-1_Маренич_В_В_перевірка на плагіат [redacted] 12/27/2024 Kharkiv National University of Radio Electronics (каф. ІНФ)	7 (1) 0.06 %

з програми обміну базами даних (0.31 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	МКР Антонишин_без економіки [redacted] 12/17/2024 National University "Lviv Politechnika" (NULP2)	12 (1) 0.10 %
2	Диплом__Ватуля_АП.docx [redacted] 6/19/2023 Ukrainian State University of Railway Transport (Кафедра "Експлуатація та ремонт рухомого складу")	10 (1) 0.08 %
3	Розробка бізнес-плану виходу підприємства на зовнішній ринок [redacted] 3/16/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)	10 (1) 0.08 %
4	Дослідження хмарних архітектур для високо масштабованих веб-рішень на основі AWS [redacted] 11/25/2024 Kharkiv National University of Economics named after S.Kuznets (KNUE) (KNUE)	5 (1) 0.04 %

з Інтернету (6.93 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**https://openarchive.nure.ua/server/api/core/bitstreams/26b43333-068d-401b-94c8-f748f9f29dbd/content**	**779 (38) 6.52 %**
2	https://dspace.znu.edu.ua/jspui/bitstream/12345/20093/1/%D0%94%D1%80%D0%BE%D0%B1%D0%BD%D0%B8%D0%B9_%D0%B4%D0%B8%D0%BF%D0%BB%D0%BE%D0%BC.pdf	38 (5) 0.32 %
3	https://openarchive.nure.ua/bitstreams/ee74f55c-3644-4fe3-8639-345cdfe1b01b/download	11 (1) 0.09 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

ДОДАТОК Б

Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

NURE ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Аналіз ефективності організації обміну повідомленнями для Front-end додатків

Черних Ігор Андрійович, ІПЗм-23-4
Науковий керівник: доц. Лановий Олексій Феліксович

software engineering

16 червня 2025

Актуальність дослідження

- Зростаюча потреба в обробці даних у реальному часі в SPA-додатках
- Традиційна модель HTTP Request-Response не відповідає сучасним вимогам до швидкості оновлення
- Необхідність вибору оптимальної технології для різних сценаріїв використання
- Особлива важливість для чатів, систем сповіщень, онлайн-ігор та платформ колаборації
- Потреба в формуванні чітких критеріїв та рекомендацій щодо вибору технології

software engineering

Мета та завдання дослідження

Проведення порівняльного аналізу ефективності технологій WebSocket та Long Polling для обробки сповіщень на стороні Front-end додатків та розробка гібридного підходу.

Основні завдання:

- Аналіз архітектурних особливостей WebSocket та Long Polling
- Розробка методології порівняльного аналізу та тестових додатків
- Проведення експериментального дослідження продуктивності
- Створення математичної моделі оптимального перемикання
- Проектування архітектури адаптивної системи обміну повідомленнями

Об'єкт і предмет дослідження

Об'єкт дослідження

Системи обміну повідомленнями у реальному часі у Front-end додатках

Предмет дослідження

Методи оптимізації продуктивності систем реального часу через адаптивне використання технологій WebSocket та Long Polling

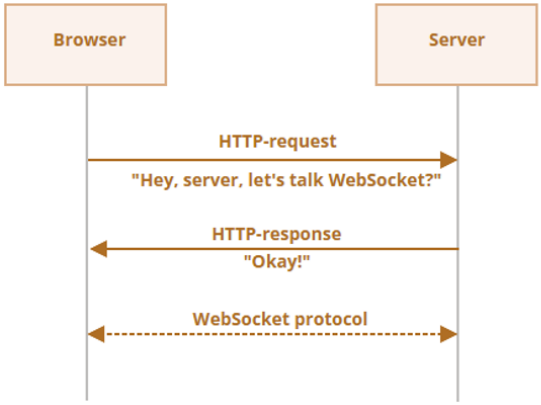
Критерії оцінки ефективності:

- Латентність комунікаційного каналу
- Мережева ефективність та накладні витрати
- Пропускна здатність системи
- Стабільність з'єднання

Аналіз технології WebSocket

Ключові характеристики:

- Двонаправлений повнодуплексний канал зв'язку через TCP-з'єднання
- Початкове HTTP рукостискання з подальшим переходом у режим WebSocket
- Вбудовані механізми контролю стану через ping/pong фрейми
- Постійне з'єднання між клієнтом та сервером
- Події: onopen, onmessage, onclose, onerror



Переваги та недоліки WebSocket

✅ Переваги:

- Низька латентність
- Менші накладні витрати після встановлення з'єднання
- Ефективне використання серверних ресурсів
- Вбудовані механізми виявлення та обробки помилок

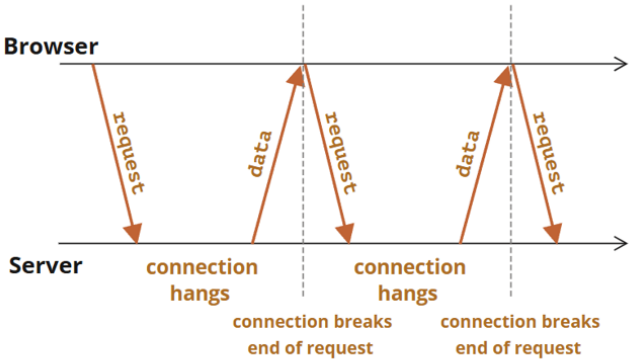
❌ Недоліки:

- Складність масштабування
- Проблеми з проксі-серверами та брандмауерами
- Потреба в спеціальній підтримці інфраструктури
- Вища складність реалізації

Аналіз технології Long Polling

Ключові характеристики:

- HTTP-запити з утриманням до появи нових даних або тайм-ауту
- Імітація push-повідомлень через стандартний HTTP-протокол
- Клієнт одразу формує новий запит після отримання відповіді
- Використання стандартних HTTP-кодів для обробки помилок



Переваги та недоліки Long Polling

✅ Переваги:

- Простота реалізації та налагодження
- Відмінна сумісність з HTTP-інфраструктурою
- Стандартні механізми балансування навантаження
- Працює з проксі-серверами без додаткових налаштувань
- Універсальна підтримка браузерів

❌ Недоліки:

- Вища затримка
- Більше накладних витрат на HTTP-заголовки
- Вище навантаження на сервер
- Потреба в ручній реалізації логіки відновлення
- Менша ефективність при частому обміні

Порівняльна таблиця технологій

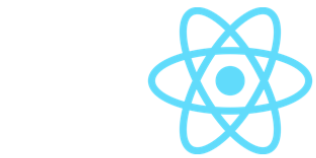
Характеристика	WebSocket	Long Polling
Затримка передачі даних	50-100 мс	200-300 мс
Серверне навантаження	Нижче через єдине з'єднання	Вище через множинні HTTP-з'єднання
Складність реалізації	Вища, потребує спеціальної обробки	Нижча, використовує стандартний HTTP
Масштабованість	Потребує спеціальних рішень	Легше масштабується
Мережеві ресурси	Ефективніше при частому обміні	Більше накладних витрат

9

Архітектура тестових систем

Створено дві ідентичні реалізації для об'єктивного порівняння:

- Технологічний стек: React, TypeScript, SCSS, Recharts
- Модульна архітектура: чіткий розподіл відповідальності між компонентами
- Уніфікований інтерфейс: BaseRealTimeClient та RealTimeCommunicationClient
- Система моніторингу: збір метрик у реальному часі



10

Механізми обробки помилок

WebSocket

- Вбудовані механізми протоколу
- Події onClose з кодами помилок
- Heartbeat ping/pong для моніторингу
- Автоматичне виявлення розривів
- Експоненціальна затримка при відновленні

Long Polling

- Стандартні HTTP-коди помилок
- Ручна реалізація логіки відновлення
- Тайм-аути для виявлення проблем
- Повторні спроби з обмеженням
- Синхронізація стану після відновлення

Результати експериментального дослідження

WebSocket

28 мс

Середня латентність

Long Polling

137 мс

Середня латентність

Ключові висновки:

- Латентність: WebSocket в 5 разів швидший
- Мережева ефективність: WebSocket - 24.3 КБ, Long Polling - 62.7 КБ
- Пропускна здатність: WebSocket - 293 повідомлення за 30 секунд, Long Polling - 134 повідомлення за 30 секунд
- Стабільність: WebSocket показує менші коливання параметрів

Гібридний підхід

Теоретичне обґрунтування:

- Обмеженість використання будь-якої окремої технології
- Динамічність параметрів мережевого середовища
- Потреба в адаптивній системі для різних умов експлуатації

Ключові можливості:

- Моніторинг параметрів середовища
- Безшовне перемикання без втрати даних
- Оцінка ефективності технологій

Математична модель

Параметри контексту C(t):

- $c_1(t)$ - стабільність мережевого з'єднання (%)
- $c_2(t)$ - латентність мережі (мс)
- $c_3(t)$ - інтенсивність обміну повідомленнями (повідомлень/сек)
- $c_4(t)$ - показник серверного навантаження (мс)

Функція ефективності:

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

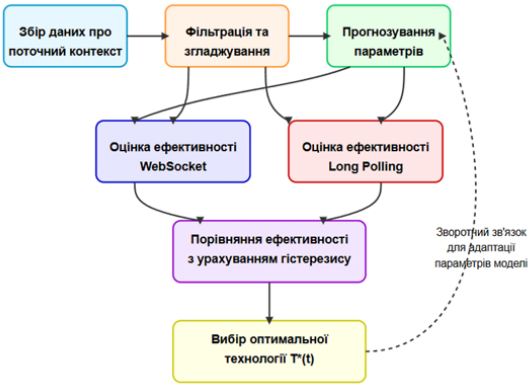
Правило вибору технології:

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

Алгоритм динамічного перемикання

Компоненти алгоритму:

- Збір даних: моніторинг параметрів середовища
- Фільтрація: експоненціальне згладжування для стабільних оцінок
- Прогнозування: лінійна екстраполяція майбутніх значень
- Оцінка ефективності: розрахунок для кожної технології
- Прийняття рішення: з урахуванням гістерезису



Псевдокод алгоритму перемикання технологій

Ключові компоненти алгоритму:

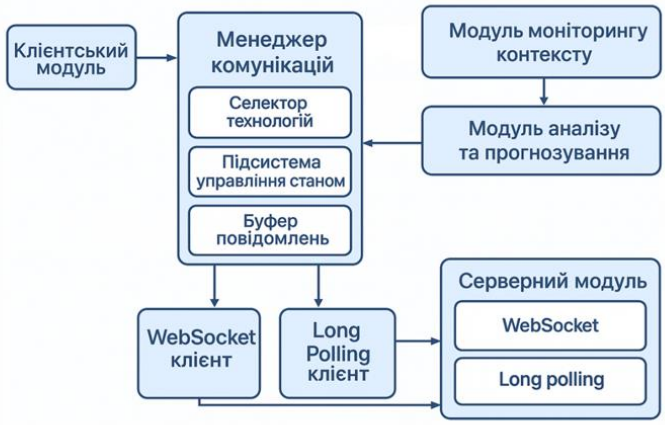
- Моніторинг контексту: збір параметрів c_1-$c_4(t)$ кожні 1-5 секунд
- Експоненціальне згладжування: усунення короткочасних коливань
- Механізм гістерезису: поріг H запобігає частим перемиканням
- Безшовне перемикання: збереження стану сесії при зміні технології

```
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
```

Архітектура гібридної системи

Основні компоненти:

- Менеджер комунікацій: координація роботи різних клієнтів
- Модуль моніторингу: збір параметрів контексту
- Модуль аналізу: оцінка ефективності та прогнозування
- Комунікаційні клієнти: WebSocket та Long Polling
- Модуль управління станом: безшовне перемикання



Апробація результатів дослідження

Наукова публікація

Черних І.А., Лановий О.Ф. Особливості обміну повідомленнями в режимі реального часу

Конференція XXIX Міжнародний молодіжний форум "Радіоелектроніка та молодь у XXI столітті" 2025

Публікація Том 6: Інформаційні інтелектуальні системи Сторінки: 349-351 ХНУРЕ

Основні результати публікації:

- Порівняльний аналіз WebSocket та Long Polling технологій
- Експериментальні дані про ефективність різних підходів
- Методологія оцінки систем реального часу для Front-end
- Практичні рекомендації для розробників веб-додатків

Висновки та рекомендації

Рекомендації по використанню:

WebSocket

- Високочастотний обмін даними
- Системи реального часу
- Чати та месенджери
- Онлайн-ігри

Long Polling

- Обмеження інфраструктури
- Нерегулярні оновлення
- Системи сповіщень
- Простота реалізації

Гібридний підхід

Оптимальне рішення для складних систем з мінливими умовами експлуатації, що забезпечує адаптивність та максимальну ефективність

Дякую за увагу!

Запитання?

ДОДАТОК В

Апробація результатів роботи

УДК 004.031.43

ОСОБЛИВОСТІ ОБМІНУ ПОВІДОМЛЕННЯМИ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ

Черних І.А., Лановий О.Ф.
e-mail: ihor.chernykh@nure.ua, oleksiy.lanovyy@nure.ua
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна

This research provides a comparative analysis of WebSocket and Long Polling technologies for real-time message exchange in front-end applications. The study investigates architectural features, performance characteristics, error handling mechanisms, and reliability of both approaches. Research findings indicate that WebSocket provides better performance for systems with high-frequency updates, while Long Polling offers advantages in compatibility and implementation simplicity for less demanding scenarios.

Виклики, що пов'язані з організацією ефективної обробки даних в реальному часі, стають критично важливими в умовах стрімкого розвитку подій. З розвитком односторінкових додатків (SPA) та зростанням вимог до інтерактивності традиційна модель запит-відповідь (HTTP Request-Response) часто не відповідає сучасним вимогам до швидкості оновлення даних. Це спонукає розробників звертатися до альтернативних технологій, таких як WebSocket та Long Polling, які пропонують різні підходи до організації обміну повідомленнями між клієнтом та сервером.

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.

WebSocket забезпечує двонаправлений повнодуплексний канал зв'язку через TCP-з'єднання, дозволяючи серверу активно надсилати дані клієнту без додаткових запитів. Long Polling базується на стандартному HTTP-протоколі та імітує push-повідомлення через механізм тривалих запитів, що робить його більш сумісним з існуючою інфраструктурою. Вибір між цими технологіями залежить від багатьох факторів: вимог до швидкодії, масштабованості системи, специфіки даних, що передаються, та обмежень інфраструктури.

Метою роботи є проведення порівняльного аналізу ефективності використання технологій WebSocket та Long Polling для обробки сповіщень на стороні Front-end додатків, визначення їх переваг та недоліків у різних сценаріях використання.

Аналіз архітектурних особливостей показав, що WebSocket працює за принципом встановлення початкового HTTP рукостискання з подальшим переходом з'єднання у режим постійного повнодуплексного каналу

зв'язку. Це забезпечує можливість обміну даними в обох напрямках без створення нових запитів та включає вбудовані механізми контролю стану з'єднання через ping/pong фрейми.

Long Polling, в свою чергу, базується на утриманні HTTP-запиту відкритим до появи нових даних або спрацювання тайм-ауту. При появі даних сервер відповідає клієнту, після чого клієнт одразу формує новий запит, забезпечуючи постійний канал комунікації. Такий підхід імітує двосторонню комунікацію, хоча він базується на стандартному HTTP-запиті.

З метою проведення детального аналізу обох підходів при організації одночасної обробки значної кількості повідомлень, в роботі було виконано порівняльний аналіз продуктивності, який довів, що 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. Ця різниця особливо помітна в додатках, що вимагають частого обміну даними.

WebSocket демонструє нижчі накладні витрати після встановлення з'єднання, оскільки не потребує багаторазового створення нових HTTP-запитів. 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.

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.

Особливу увагу в дослідженні приділено механізмам обробки помилок. 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. Long Polling використовує стандартні HTTP-коди відповіді та часові обмеження запитів, що вимагає від розробника самостійної реалізації логіки відновлення з'єднання.

Проведені дослідження виявили важливу відмінність, пов'язану з підходом 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.

В результаті проведеного дослідження було розроблено методологію порівняльного аналізу, яка включає оцінку продуктивності, надійності та масштабованості технологій. Основними критеріями оцінки є латентність, мережеві накладні витрати, пропускна здатність системи, ефективність використання серверних ресурсів, надійність при мережевих збоях, складність реалізації та сумісність з різними платформами.

Для проведення тестування спроектовано архітектуру тестових додатків з використанням React та TypeScript. Архітектура базується на модульній структурі, де мережевий шар реалізується по-різному для WebSocket та Long Polling версій, при цьому зберігаючи однаковий інтерфейс взаємодії з іншими компонентами системи. Система тестування включає генератор навантаження, емулятор мережевих умов та систему моніторингу для збору метрик у реальному часі.

Для забезпечення об'єктивності результатів всі тести проводяться в ідентичних умовах для обох технологій. Система тестування розгортається в ізольованому середовищі, що мінімізує вплив зовнішніх факторів на результати вимірювань. Це дозволяє отримати надійні дані для порівняльного аналізу ефективності WebSocket та Long Polling у різних сценаріях використання.

Результати проведених досліджень показують, що вибір між WebSocket та Long Polling повинен базуватися на конкретних вимогах проекту. WebSocket є більш ефективним для систем реального часу з високою частотою обміну даними, онлайн-ігор, чатів та інтерактивних додатків, де критичною є мінімальна затримка. Long Polling більш підходить для систем сповіщень з нерегулярними оновленнями, додатків з обмеженнями на використання WebSocket та середовищ, де важлива сумісність з широким спектром клієнтських платформ.

Проведені дослідження можуть служити основою для подальшої практичної реалізації та експериментального порівняння обох технологій у різних умовах використання. Розроблені методологія та архітектура тестових додатків дозволять отримати об'єктивні дані про ефективність WebSocket та Long Polling у конкретних сценаріях застосування та сформувати обґрунтовані рекомендації щодо їх використання.

Список використаних джерел:

1. Wang V., Salim F., Moskovits P. The Definitive Guide to HTML5 WebSocket. Apress, 2021. 164 p.

2. Pimentel V., Nickerson B. G. Web-Based Real-Time Communications: Past, Present and Future. Information Systems Research, 2023. Vol. 34(1). P. 289-308.

3. Lombardi A. WebSocket: Lightweight Client-Server Communications. O'Reilly Media, 2015. 143 p.

ДОДАТОК Г

Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008:2015

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-23-4 (група)

Черних Ігор Андрійович
(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
	7.3 Нумерація сторінок звіту	
	7.4 Нумерація розділів, підрозділів, пунктів, підпунктів	
	7.5 Рисунки	
	7.6 Таблиці	
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні познаки	
	7.15 Додатки	

зауважень немає

Експерт _____ (підпис) Олена ОЛІЙНИК (прізвище, ініціали)

07.06.2025

ДОДАТОК Д

Частина коду реалізації з використанням WebSocket

```
1. export class WebSocketClient extends BaseRealTimeClient {
2. private ws: WebSocket | null = null;
3. private reconnectAttempts: number = 0;
4. private maxReconnectAttempts: number = 5;
5. private reconnectDelayMs: number = 1000;
6. private reconnectTimeoutId: number | null = null;
7. private autoReconnect: boolean = true;
8. constructor(url: string, options?: {
9. autoReconnect?: boolean,
10.    maxReconnectAttempts?: number
11.    }) {
12.    super(url);
13.    if (options) {
14.    this.autoReconnect = options.autoReconnect !== undefined ?
   options.autoReconnect : true;
15.    this.maxReconnectAttempts = options.maxReconnectAttempts || 5;
16.    }
17.    }
18.    public async connect(): Promise<void> {
19.    if (this.ws && (this.ws.readyState === WebSocket.CONNECTING ||
   this.ws.readyState === WebSocket.OPEN)) {
20.    return;
21.    }
22.    return new Promise((resolve, reject) => {
23.    try {
24.    this.ws = new WebSocket(this.url);
25.    this.ws.onopen = () => {
26.    this.reconnectAttempts = 0;
27.    this.handleConnect();
28.    resolve();
29.    };
30.    this.ws.onmessage = (event) => {
31.    try {
       a. const message = JSON.parse(event.data) as Message;
       b. this.handleMessage(message);
32.    } catch (error) {
       a. console.error('Error parsing WebSocket message:', error);
       b. this.handleError(new Error(`Failed to parse message:
          ${error.message}`));
```

```
33.    }
34.    };
35.    this.ws.onclose = (event) => {
36.    const reason = event.reason || `Code: ${event.code}`;
37.    this.handleDisconnect(reason);
38.    if (this.autoReconnect && this.reconnectAttempts <
   this.maxReconnectAttempts) {
      a. this.scheduleReconnect();
39.    }
40.    };
41.    this.ws.onerror = (event) => {
42.    const error = new Error(`WebSocket error`);
43.    this.handleError(error);
44.    reject(error);
45.    };
46.    } catch (error) {
47.    this.handleError(error);
48.    reject(error);
49.    }
50.    });
51.    }
52.    public disconnect(): void {
53.    this.autoReconnect = false;
54.    if (this.reconnectTimeoutId !== null) {
55.    clearTimeout(this.reconnectTimeoutId);
56.    this.reconnectTimeoutId = null;
57.    }
58.    if (this.ws && (this.ws.readyState === WebSocket.CONNECTING ||
   this.ws.readyState === WebSocket.OPEN)) {
59.    this.ws.close(1000, 'Normal closure');
60.    }
61.    }
62.    public async send(message: any): Promise<boolean> {
63.    if (!this.ws || this.ws.readyState !== WebSocket.OPEN) {
64.    throw new Error('WebSocket is not connected');
65.    }
66.    try {
67.    const messageToSend = typeof message === 'object' ? message : { data:
   message };
68.    if (!messageToSend.timestamp) {
69.    messageToSend.timestamp = Date.now();
70.    }
71.    if (!messageToSend.id) {
```

```
72.    messageToSend.id = `msg-${Date.now()}-
   ${Math.random().toString(36).substring(2, 9)}`;
73.    }
74.    if (!messageToSend.type) {
75.    messageToSend.type = 'message';
76.    }
77.    const messageString = JSON.stringify(messageToSend);
78.    this.ws.send(messageString);
79.    this.stats.messagesSent++;
80.    this.stats.bytesSent += messageString.length;
81.    return true;
82.    } catch (error) {
83.    this.handleError(error);
84.    return false;
85.    }
86.    }
87.    public isConnected(): boolean {
88.    return this.ws !== null && this.ws.readyState === WebSocket.OPEN;
89.    }
90.    private scheduleReconnect(): void {
91.    this.reconnectAttempts++;
92.    this.stats.reconnectAttempts++;
93.    this.stats.connectionTimeline.push({
94.    event: 'reconnect',
95.    timestamp: Date.now(),
96.    reason: `Attempt ${this.reconnectAttempts} of
   ${this.maxReconnectAttempts}`
97.    });
98.    const delay = this.reconnectDelayMs * Math.pow(1.5,
   this.reconnectAttempts - 1);
99.    this.reconnectTimeoutId = window.setTimeout(() => {
100.   this.reconnectTimeoutId = null;
101.   this.connect().catch(error => {
102.   console.error('Reconnect failed:', error);
103.   });
104.   }, delay);
105.   }
106.   }
```

ДОДАТОК Е

Частина коду реалізації з використанням Long Polling

```
1. export class LongPollingClient extends BaseRealTimeClient {
2. private pollUrl: string;
3. private sendUrl: string;
4. private abortController: AbortController | null = null;
5. private pollTimeout: number = 30000;
6. private retryTimeout: number = 1000;
7. private maxRetries: number = 5;
8. private retryCount: number = 0;
9. private retryTimeoutId: number | null = null;
10.    public polling: boolean = false;
11.    public sessionId: string | null = null;
12.    constructor(baseUrl: string, options?: {
13.    pollTimeout?: number,
14.    maxRetries?: number
15.    }) {
16.    super(baseUrl);
17.    this.pollUrl = `${baseUrl}/poll`;
18.    this.sendUrl = `${baseUrl}/send`;
19.    if (options) {
20.    this.pollTimeout = options.pollTimeout || 30000;
21.    this.maxRetries = options.maxRetries || 5;
22.    }
23.    }
24.    public async connect(): Promise<void> {
25.    if (this.polling) {
26.    return;
27.    }
28.    this.sessionId = `session-${Date.now()}-
   ${Math.random().toString(36).substring(2, 9)}`;
29.    this.resetStats();
30.    this.polling = true;
31.    this.connected = true;
32.    this.handleConnect();
33.    this.poll();
34.    }
35.    public disconnect(): void {
36.    this.polling = false;
37.    if (this.abortController) {
38.    this.abortController.abort();
```

```
39.    this.abortController = null;
40.    }
41.    if (this.retryTimeoutId !== null) {
42.    clearTimeout(this.retryTimeoutId);
43.    this.retryTimeoutId = null;
44.    }
45.    this.connected = false;
46.    this.handleDisconnect('Client disconnected');
47.    }
48.    public async send(message: any): Promise<boolean> {
49.    if (!this.connected) {
50.    throw new Error('Long Polling client is not connected');
51.    }
52.    try {
53.    const messageToSend = typeof message === 'object' ? { ...message } : {
   data: message };
54.    if (!messageToSend.timestamp) {
55.    messageToSend.timestamp = Date.now();
56.    }
57.    if (!messageToSend.id) {
58.    messageToSend.id = `msg-${Date.now()}-
   ${Math.random().toString(36).substring(2, 9)}`;
59.    }
60.    if (!messageToSend.type) {
61.    messageToSend.type = 'message';
62.    }
63.    messageToSend.sessionId = this.sessionId;
64.    const messageString = JSON.stringify(messageToSend);
65.    const response = await fetch(this.sendUrl, {
66.    method: 'POST',
67.    headers: {
68.    'Content-Type': 'application/json'
69.    },
70.    body: messageString
71.    });
72.    if (!response.ok) {
73.    throw new Error(`HTTP error: ${response.status} ${response.statusText}`);
74.    }
75.    this.stats.messagesSent++;
76.    this.stats.bytesSent += messageString.length;
77.    return true;
78.    } catch (error) {
79.    this.handleError(error);
```

```
80.   return false;
81.   }
82.   }
83.   public isConnected(): boolean {
84.   return this.connected && this.polling;
85.   }
86.   private async poll(): Promise<void> {
87.   if (!this.polling) {
88.   return;
89.   }
90.   this.abortController = new AbortController();
91.   try {
92.   const url = new URL(this.pollUrl);
93.   url.searchParams.append('sessionId', this.sessionId ?? '');
94.   url.searchParams.append('timestamp', Date.now().toString());
95.   const response = await fetch(url.toString(), {
96.   method: 'GET',
97.   headers: {
98.   'Accept': 'application/json'
99.   },
100.  signal: this.abortController.signal,
101.  cache: 'no-store'
102.  });
103.  this.retryCount = 0;
104.  if (response.status === 204) {
105.  this.poll();
106.  return;
107.  }
108.  if (!response.ok) {
109.  throw new Error(`HTTP error: ${response.status} ${response.statusText}`);
110.  }
111.  const data = await response.json();
112.  if (Array.isArray(data)) {
113.  data.forEach(message => {
114.  this.handleMessage(message as Message);
115.  });
116.  } else {
117.  this.handleMessage(data as Message);
118.  }
119.  this.poll();
120.  } catch (error) {
121.  if (error.name === 'AbortError') {
122.  return;
```

```
123.   }

124.   this.handleError(error);
125.   if (this.polling && this.retryCount < this.maxRetries) {
126.   this.retryCount++;
127.   this.stats.reconnectAttempts++;
128.   this.stats.connectionTimeline.push({
129.   event: 'reconnect',
130.   timestamp: Date.now(),
131.   reason: `Attempt ${this.retryCount} of ${this.maxRetries}`
132.   });
133.   const delay = this.retryTimeout * Math.pow(1.5, this.retryCount - 1);
134.   this.retryTimeoutId = window.setTimeout(() => {
135.   this.retryTimeoutId = null;
136.   this.poll();
137.   }, delay);
138.   } else if (this.polling) {
139.   this.connected = false;
140.   this.polling = false;
141.   this.handleDisconnect('Max retry attempts reached');
142.   }
143.   }
144.   }
145.   }
```