Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра мережних технологій

Магістерська робота

освітній ступінь – магістр

на тему: «**РОЗРОБКА СИСТЕМИ ЗБОРУ ВЕБ-АНАЛІТИКИ З ПІДВИЩЕНОЮ НАДІЙНІСТЮ НА ОСНОВІ ВІДМОВОСТІЙКОГО ВЕБ ТЕГА ТА БАГАТОРІВНЕВОЇ ХМАРНОЇ ІНФРАСТРУКТУРИ**»

Виконав: студент 2-го року навчання,

освітньо-наукової програми
«Інженерія програмного
забезпечення», 121

Дячук Максим Вікторович

Керівник Черкасов Д.І.,_____

к. т. н., старший викладач

Рецензент_____

(прізвище та ініціали)

Магістерська робота захищена
з оцінкою _____

Секретар ЕК _____
«____» _____ 20____ р.

Київ – 2025

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій

ЗАТВЕРДЖУЮ
Зав. кафедри інформатики,
Доктор фіз.-мат. наук,
_____ Малашонок Г. І.
(підпис)
«____» _____ 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на дипломну роботу студенту

2 р. н. магістерської програми Інженерія програмного забезпечення
Дячуку Максиму Вікторовичу
Розробити систему збору веб-аналітики на основі відмовостійкого веб тега та багаторівневої хмарної інфраструктури
Зміст текстової частини до магістерської роботи: Анотація, Перелік умовних позначень, Вступ, Теоретичні основи та аналіз підходів до збору веб-аналітики, Проєктування архітектури fault-tolerant системи збору подій, Реалізація клієнтського тегу з підтримкою fault-tolerance, Оцінка ефективності запропонованого рішення, Висновки до роботи, Список використаних джерел, Додатки

Дата видачі: “__ ”_____ 2024
Керівник: к. т. н., старший викладач Черкасов Д.І._____
Завдання отримав Дячук М. В._____

Календарний план виконання роботи

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1	Отримання завдання на дипломну роботу	05.11.2024	
2	Огляд джерел	06.11.2024 – 25.11.2024	
3	Аналіз існуючих рішень	26.11.2024 – 15.12.2024	
4	Визначення вимог до розроблюваного рішення	16.12.2024 – 05.01.2025	
5	Розробка архітектури системи	06.01.2025 – 31.01.2025	
6	Реалізація веб тегу	01.02.2025 – 28.02.2025	
7	Написання тестів	01.03.2025 – 15.03.2025	
8	Рефакторинг	16.03.2025 – 31.03.2025	
9	Підготовка технічної документації	01.04.2025 – 20.05.2025	
10	Попередній захист	29.05.2025	
11	Остаточний захист	10.06.2025	

Керівник: к. т. н. старший викладач Черкасов Д.І._____

Студент: Дячук М.В._____

АНТОТАЦІЯ

У роботі розглянуто проблему підвищення надійності збору веб-аналітики на клієнтському боці. Запропоновано архітектуру системи на основі багаторівневої хмарної інфраструктури та відмовостійкого клієнтського тегу. Реалізовано fault-tolerant механізми доставки подій, буферизацію, повторні спроби надсилання та підтримку роботи в умовах нестабільного з’єднання. Проведено тестування, що підтвердило стабільність і придатність системи до використання в реальних вебзастосунках.

ABSTRACT

This thesis addresses the problem of improving the reliability of web analytics collection on the client side. A system architecture is proposed based on a multi-layer cloud infrastructure and a fault-tolerant client-side tracking tag. The implemented solution includes local event buffering, retry mechanisms, resilient data delivery using fetch and sendBeacon, and robust operation under unstable network conditions. Testing confirmed the system's stability and its applicability in real-world web applications.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) — програмний інтерфейс взаємодії компонентів.

AWS (Amazon Web Services) — платформа хмарних обчислень, що включає сервіси зберігання, обробки, маршрутизації тощо.

CDN (Content Delivery Network) — мережа доставки контенту, що прискорює завантаження статичних ресурсів.

CloudFront — CDN-сервіс від Amazon, використовується для кешування та доставки тегу та API-запитів.

DynamoDB — відмовостійка NoSQL-база даних від AWS, використовується для зберігання подій.

Event Tracker — модуль у тегу, що фіксує події взаємодії користувача з інтерфейсом.

fetch — браузерний API для відправки HTTP-запитів.

fault-tolerant — здатність системи продовжувати роботу при виникненні часткових збоїв.

HTTP — Hypertext Transfer Protocol — протокол передачі гіпертексту в інтернеті.

IndexedDB — браузерна база даних для локального зберігання об’єктів JavaScript.

JS — мова програмування, що використовується для створення тегу.

Lambda — безсерверна обчислювальна служба AWS для виконання функцій у відповідь на події.

LocalStorage — браузерне сховище для зберігання пар ключ–значення.

QueueManager — модуль у тегу, що відповідає за буферизацію подій, їхню чергу та статуси.

Scheduler — компонент, який визначає момент надсилання накопичених подій.

S3 (Simple Storage Service) — об'єктне сховище даних в AWS.

sendBeacon — браузерний API для асинхронної передачі даних при закритті сторінки.

SPA (Single Page Application) — вебзастосунок, який не перезавантажує сторінку повністю.

Transport — модуль, що відповідає за передачу подій на сервер.

TTL (Time To Live)— час життя об’єкта або події в черзі.

ВСТУП

Збір аналітичних даних про активність користувачів на веб-ресурсах став однією з найважливіших частин роботи великих бізнесів. Веб-аналітика допомагає відслідковувати поведінку користувачів, оцінювати зручність користувацьких інтерфейсів, ефективність маркетингових кампаній, приймати вірні рішення. В той же час якість веб-аналітики залежить напряму від надійності та точності процесу збору даних. Навіть незначні помилки можуть спричинити втрати або викривлення інформації, що в свою чергу призводить до хибних рішень та фінансових втрат.

Основою більшості систем збору веб-аналітики є тег — історична назва яка закріпилась і тепер означає невеликий JavaScript скрипт, який додається на вебсторінку та відпровляє події на веб сервер. Проте теги часто є вразливими до блокування (з боку браузера або розширень), мережевих затримок, нестабільності інтернет-з’єднання або просто помилок у виконанні. Більшість із них не передбачає механізмів повторної доставки чи збереження подій у разі збою. Це створює реальну загрозу втрати важливої інформації.

Потреба у відмовостійких рішеннях для збору веб-аналітики зростає разом із популярністю хмарних технологій та сучасних архітектур, які передбачають гнучкість, масштабованість і безперервну роботу. Зокрема, сервіси AWS дозволяють будувати складні системи з високою надійністю та доступністю. Проте навіть найнадійніша хмарна інфраструктура не зможе компенсувати втрати, якщо події не дійшли до неї з клієнтського боку.

У цьому контексті актуальним є завдання побудови такої системи, де клієнтський тег збору подій буде здатен працювати стабільно навіть у нестандартних або несприятливих умовах, а дані — гарантовано доходити до бекенду, побудованого за принципами високої доступності. Це дає змогу

мінімізувати втрати, підвищити достовірність аналітики та, відповідно, покращити якість прийнятих на її основі рішень.

Мета роботи: розробити архітектуру надійної системи збору веб-аналітики з використанням хмарної інфраструктури та реалізувати прототип відмовостійкого клієнтського тегу.

Для досягнення мети передбачено вирішення таких завдань:

1. Проаналізувати сучасні інструменти та підходи до збору веб-аналітики.
2. Сформулювати вимоги до тегу, здатного працювати за умов збоїв і нестабільного з’єднання.
3. Розробити архітектуру багаторівневої хмарної системи на базі AWS.
4. Реалізувати клієнтський тег на JavaScript/TypeScript із підтримкою повторної доставки, кешування подій і fallback-механізмів.
5. Перевірити надійність і ефективність розробленого рішення на практиці.

Об’єкт дослідження: системи збору веб-аналітичних даних.

Предмет дослідження: технічні рішення, що забезпечують надійність збору подій у вебсередовищі.

Методи дослідження:

- аналіз технічної документації та літератури;
- проєктування архітектури IT-систем;
- клієнтська розробка на JavaScript/TypeScript;
- використання хмарних сервісів AWS;
- моделювання типових збоїв та тестування працездатності системи.

Наукова новизна полягає в обґрунтуванні вимог до надійного тегу збору подій, поєднанні клієнтських механізмів fault tolerance із хмарною багаторівневою архітектурою та реалізації прототипу, здатного працювати автономно до моменту успішної доставки даних.

Практичне значення роботи полягає у можливості використання розробленого рішення як компонента у великих аналітичних системах, орієнтованих на стабільність і масштабування. Прототип тегу може стати основою для розробки власного інструменту збору подій або вдосконалення наявних платформ.

Апробація результатів передбачається у межах підготовки до захисту та потенційної участі у профільних семінарах чи конференціях.

Публікації: наразі відсутні.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ПІДХОДІВ ДО ЗБОРУ ВЕБ-АНАЛІТИКИ

1.1. Поняття веб-аналітики: функції, мета, класифікація даних

Веб-аналітика — це процес збору, вимірювання, обробки та аналізу даних про взаємодію користувачів із вебресурсами з метою покращення функціонування сайту або вебдодатку, підвищення його ефективності та досягнення бізнес-цілей. У найзагальнішому вигляді веб-аналітика дозволяє відповісти на запитання: хто, як і навіщо використовує вебресурс, а також які дії призводять до бажаного результату — покупки, заповнення форми, підписки тощо.

Збір веб-аналітичних даних зазвичай реалізується за допомогою спеціального тегу — невеликого JavaScript-коду, який вбудовується у HTML-код сторінки. Цей тег виконується на стороні користувача та фіксує події, такі як перегляди сторінок, кліки, скролінг, взаємодія з формами, переходи за посиланнями, час перебування на сторінці тощо. Після фіксації ці події передаються на сервер для подальшої обробки.

Основні функції веб-аналітики включають:

- Моніторинг — регулярне спостереження за трафіком, поведінкою користувачів, змінами у показниках.
- Оцінювання ефективності — аналіз каналів трафіку, маркетингових кампаній, A/B-тестів.
- Оптимізація — виявлення слабких місць у структурі сайту, інтерфейсі чи контенті.

- Прогнозування — визначення тенденцій у поведінці користувачів та планування змін.
- Підтримка прийняття рішень — надання обґрунтованої аналітичної бази для дій менеджерів, маркетологів і розробників.

Мета веб-аналітики полягає у:

- покращенні користувацького досвіду (UX);
- підвищенні конверсій;
- зниженні витрат;
- виявленні технічних проблем у вебінтерфейсі;
- підтримці стратегічного управління на основі даних (data-driven decision-making).

Аналітичні системи можуть класифікувати зібрані дані за кількома ознаками (див. Табл. 1)

Табл. 1. Критерії класифікації даних

Критерій	**Приклади**
Тип події	pageview, click, form_submit, error, scroll
Тип даних	кількісні (кількість переглядів), якісні (тип пристрою, мова)
Джерело даних	клієнт (браузер), сервер (бекенд), сторонній сервіс
Частота збору	подійна (event-based), агрегована (раз на сесію)

Місце обробки	локально (браузер), централізовано (сервер, хмара)

Крім того, дані умовно поділяються на технічні (наприклад, IP-адреса, User-Agent), поведінкові (що саме робив користувач), контекстуальні (час доби, мова інтерфейсу) та комерційні (джерело конверсії, цінність сесії тощо).

Таким чином, веб-аналітика є складною багатокомпонентною системою, у якій важливу роль відіграє надійність первинного збору даних. Якщо перша ланка — тег, що фіксує події, — працює нестабільно, це призводить до системних похибок на всіх наступних етапах аналізу. Саме тому у фокусі сучасних досліджень і розробок перебувають питання стабільності, безпеки та fault tolerance у клієнтських аналітичних рішеннях.

1.2. Огляд сучасних систем збору аналітики

На сьогодні існує велика кількість інструментів та платформ, призначених для збору, обробки та візуалізації аналітичних даних з вебдодатків. Вони розрізняються за архітектурою, підходами до збору подій, гнучкістю налаштування, рівнем відмовостійкості, а також — ступенем інтеграції з іншими системами. У цьому підрозділі розглянуто найпоширеніші платформи аналітики, їхні принципи роботи та обмеження в контексті надійності.

Google Analytics

Google Analytics є однією з найвідоміших та наймасовіших систем веб-аналітики. Вона використовує клієнтський тег gtag.js або analytics.js, який надсилає події на сервер Google у реальному часі або з мінімальною буферизацією.

Переваги:

- Автоматична обробка великої кількості типових подій.
- Глибока інтеграція з Google Ads, BigQuery, Firebase.
- Гнучка побудова воронок та сегментації.

Обмеження:

- Відсутність детального контролю над трафіком: тег є «чорним ящиком».
- Високий ризик блокування з боку AdBlock / Privacy Badger.
- Обмежена fault-tolerance: події не зберігаються локально у випадку втрати зв’язку.

Snowplow Analytics

Snowplow — це платформа з відкритим кодом, орієнтована на збір сирих подій (raw events), які потім обробляються у власній або хмарній інфраструктурі користувача. Snowplow використовує власний тег snowplow.js, який може бути сконфігурований на гнучку доставку даних.

Переваги:

- Повний контроль над потоками подій.
- Підтримка кешування, буферизації та повторної доставки.
- Добре інтегрується з AWS, GCP, Data Lake.

Обмеження:

- Високий поріг входу (необхідність розгортання інфраструктури).
- Вимагає налаштування бекенду і обробки.

Twilio Segment

Segment працює як універсальний маршрутизатор аналітичних подій. Тег на фронтенді (analytics.js) збирає події й надсилає їх на сервер Segment, який потім ретранслює ці події до Google Analytics, Amplitude, Mixpanel, тощо.

Переваги:

- Централізоване керування всіма аналітичними потоками.
- Підтримка fallback і буферизації.
- Проста інтеграція з десятками платформ.

Обмеження:

- Тег може бути заблокований.
- Базова версія обмежена у гнучкості маршрутизації.

OpenTelemetry

OpenTelemetry — це специфікація та набір інструментів для збору трейсів, метрик та логів. Хоча переважно використовується на бекенді, останнім часом з’являються фронтенд-бібліотеки для збору подій з браузера.

Переваги:

- Відкрита архітектура, стандарт де-факто для observability.
- Можливість кастомної маршрутизації даних.

- Легка інтеграція з хмарними провайдерами.

Обмеження:

- Недостатня підтримка подій користувача у фронтенді.
- Не має простого «тегу» для вбудування без налаштування.

Нижче приведено Табл. 2. для порівняння аналогів.

Табл. 2. Порівняння аналогів

Система	**Тип**	**Відкритість**	**Fault-tolerance**	**Буферизація**	**Вимоги до бекенду**
Google Analytics	SaaS	Пропрієтарна	Низька	Немає	Немає
Snowplow	Self-hosted	Відкрита	Висока	Є	Так
Segment	SaaS	Пропрієтарна	Середня	Є	Частково
OpenTelemetry	Framework	Відкрита	Висока	Є	Так

Висновки до підрозділу

Проведений огляд показує, що хоча більшість поширених аналітичних систем забезпечують зручний інтерфейс збору подій, лише окремі з них передбачають механізми забезпечення надійної доставки даних у разі збою. Сучасні виклики вимагають гнучких, модульних, а головне — fault-tolerant рішень, які дозволяють зберігати, чергувати та повторно передавати події,

якщо це необхідно. Саме на цьому аспекті зосереджено розробку системи, описаної у цій роботі.

1.3. Ризики та вразливості тегів: блокування, втрати, мережеві помилки

Надійність збору аналітичних подій безпосередньо залежить від стабільної роботи клієнтського тегу — скрипта, який ініціює передачу подій з браузера до аналітичної системи. У типовому випадку такий тег підключається до вебсторінки через зовнішній <script>-елемент, виконується на боці користувача та реагує на взаємодії з інтерфейсом. Проте цей процес є вразливим до широкого спектра збоїв, які можуть призвести до втрати подій, дублювання або затримки доставки, що негативно впливає на точність та повноту аналітики.

Нижче розглянуто основні категорії ризиків, притаманних клієнтським тегам.

Блокування тегів з боку користувача або середовища

Однією з найпоширеніших причин недоставки аналітичних подій є блокування виконання тегу сторонніми інструментами або самим браузером. Це може статися в таких випадках:

- Розширення для захисту приватності (наприклад, uBlock Origin, Privacy Badger, Ghostery) можуть автоматично блокувати домени популярних систем аналітики (google-analytics.com, segment.io тощо).

- Браузери з вбудованим захистом (Safari, Firefox з Enhanced Tracking Protection) перешкоджають передачі даних до сторонніх серверів за певними патернами.
- Антивірусне ПЗ або мережеві фільтри (наприклад, у корпоративному середовищі) можуть обмежувати завантаження скриптів з незнайомих джерел.

У таких випадках тег не буде завантажено, і події не будуть зібрані взагалі. Особливо критичним це є для систем, які не мають fallback-механізмів або client-side кешування.

Проблеми під час відправки даних

Навіть якщо тег виконався, не гарантовано, що подія буде доставлена на сервер. Поширеними є такі сценарії втрат:

- Завершення сесії/вкладки до моменту завершення запиту (unload, beforeunload, navigate away).
- Збої підключення або таймаути на рівні мережі (наприклад, 504 Gateway Timeout).
- Блокування CORS (Cross-Origin Resource Sharing), якщо сервер аналітики неправильно налаштовано.
- Втручання сторонніх скриптів, які перехоплюють або змінюють поведінку fetch / XMLHttpRequest.

Ці фактори можуть бути частково нівельовані за рахунок використання таких інструментів, як navigator.sendBeacon, який призначений для безпечної передачі даних під час закриття сторінки. Проте навіть sendBeacon може не спрацювати у певних умовах: наприклад, якщо обсяг даних перевищує ліміти (~64КБ), або якщо користувач перебуває у режимі «low-data».

Відсутність локального збереження або retry-логіки

Багато тегів працюють за принципом «fire-and-forget», тобто формують і надсилають подію одноразово, без перевірки успішності доставки. Це означає, що в разі будь-якої помилки (мережа, таймаут, статус 500) подія втрачається остаточно. Причини:

- Немає черги подій або кешу у браузері (наприклад, через IndexedDB або LocalStorage).
- Немає механізму повторних спроб з експоненціальною затримкою.
- Відсутня централізована обробка помилок або логування.

У результаті частина подій (від 5% до 30%, за різними оцінками) може не доходити до аналітичної системи в реальних умовах.

Інші фактори ризику

До менш очевидних, але все ж важливих факторів втрат подій належать:

- Конкуренція між тегами: на одній сторінці може бути кілька скриптів, які перевизначають глобальні об’єкти або події (window.onerror, addEventListener, fetch).
- Обмеження браузера на кількість одночасних з’єднань, що призводить до черг і скасування запитів.
- Мобільні браузери та режими енергозбереження, які зупиняють виконання JavaScript у фоновому режимі.

Висновки до підрозділу

Сучасні клієнтські теги збору веб-аналітики стикаються з численними загрозами, які можуть спричинити втрату або спотворення даних. Блокування з боку користувача, нестабільне з’єднання, технічні обмеження браузера або недосконаліть реалізації тегу — все це робить задачу надійного збору подій нетривіальною. Для її вирішення необхідні архітектурні підходи до fault tolerance: локальне кешування, буферизація, контроль доставки, fallback-логіка та хмарна інфраструктура, здатна обробляти події з різною затримкою.

1.4. Існуючі підходи до fault tolerance на клієнтському боці

У відповідь на численні ризики втрати даних при зборі аналітики, розробники почали впроваджувати відмовостійкі механізми (fault-tolerant solutions) безпосередньо на клієнтському боці — у веббраузері користувача. Метою таких рішень є забезпечення доставки аналітичних подій до бекенду навіть у несприятливих умовах, як-от: нестабільне інтернет-з’єднання, блокування сторонніми розширеннями, передчасне закриття сторінки або збої серверів.

Нижче наведено основні підходи та інструменти, що використовуються для досягнення fault tolerance у клієнтських тегах.

Локальне кешування подій

Один з ключових підходів — тимчасове збереження подій у браузері до моменту успішної доставки на сервер. Для цього використовуються вбудовані механізми зберігання даних:

- localStorage — простий, синхронний ключ-значення API. Підходить для невеликих обсягів, але не підтримує транзакційність або структуровані об’єкти.
- IndexedDB — асинхронна документно-орієнтована база даних у браузері. Дає змогу зберігати сотні мегабайт структурованих об'єктів із пошуком за індексами.
- sessionStorage — обмежений до сесії, не підходить для fault tolerance у класичному розумінні, але іноді використовується як тимчасовий буфер.

Збережені події надсилаються на сервер із затримкою або після повторної спроби. Наприклад, тег записує подію до IndexedDB, а окремий фоновий процес (worker або інтервал) відповідає за її відправку з перевіркою на успішність.

Повторні спроби доставки (retry logic)

У реалізації fault-tolerant тегу доцільно використовувати механізм повторних спроб відправки у разі помилки. Він може бути реалізований із:

- експоненціальною затримкою (exponential backoff);
- лімітом кількості спроб (щоб уникнути безкінечних циклів);
- адаптивним таймером, що враховує зміну мережевого статусу (navigator.onLine, події online/offline).

Цей підхід дозволяє гарантувати, що події будуть доставлені за першої ж нагоди, навіть якщо спочатку запит не вдався.

Використання navigator.sendBeacon()

sendBeacon — спеціальний API, створений для асинхронної передачі невеликих обсягів даних на сервер без блокування навігації або взаємодії користувача. Він особливо корисний у таких випадках:

- передача подій під час закриття сторінки (unload, visibilitychange);
- мінімізація часу виконання аналітичних запитів;
- зменшення навантаження на головний потік (main thread).

Обмеження sendBeacon полягають у:

- відсутності підтвердження успішної доставки (метод «вогонь і забудь»);
- обмеженому обсязі даних (~60–64КБ);
- неможливості відстежити помилку або повторно надіслати.

Використання Web Workers / Service Workers

Ще один перспективний підхід — перенесення логіки збору й обробки подій у фонові потоки, ізольовані від основного UI:

- Web Workers — дозволяють обробляти дані асинхронно, не блокуючи інтерфейс.
- Service Workers — діють як проксі між клієнтом і мережею. Можуть перехоплювати запити, кешувати відповіді й працювати в офлайн-режимі.

У контексті аналітики service worker може:

- кешувати події у випадку втрати зв'язку;
- повторно передавати їх при відновленні мережі;

- реалізовувати складну логіку маршрутизації (наприклад, передача в різні бекенди).

Додаткові стратегії

- Дедуплікація — запобігання дублюванню подій при повторній відправці.
- Стиснення або об'єднання подій у пакети для зменшення навантаження на мережу.
- Тайм-буферизація — накопичення подій протягом певного часу перед їх відправкою (batching).

Висновки до підрозділу

На сьогодні існує кілька технічних підходів до підвищення fault tolerance у клієнтських тегах. Найбільш поширеним є використання sendBeacon для безпечної відправки даних при закритті сторінки, локального кешування подій в IndexedDB або LocalStorage, а також реалізація логіки повторної доставки. У складніших випадках застосовуються service workers або стримінг-підключення. Розробники аналітичних систем, орієнтованих на гарантований збір подій, дедалі частіше комбінують ці підходи в єдину багаторівневу систему.

1.5. Висновки до розділу 1

У першому розділі було розглянуто основи веб-аналітики як галузі, що має ключове значення для цифрових продуктів та бізнесів, орієнтованих на дані. Веб-аналітика забезпечує збір інформації про поведінку користувачів у вебінтерфейсах і дозволяє оцінювати ефективність рішень, оптимізувати вміст, покращувати взаємодію з клієнтами та приймати стратегічні рішення на основі достовірної інформації.

Було здійснено порівняльний аналіз поширених платформ збору аналітики — Google Analytics, Snowplow, Segment, OpenTelemetry — з точки зору архітектури, гнучкості, fault tolerance та відкритості. Встановлено, що більшість популярних рішень орієнтовані на простоту інтеграції, проте не завжди забезпечують достатню стійкість до втрати даних у разі збоїв на клієнтському боці. Найбільш fault-tolerant підходи наразі реалізовані у рішеннях з відкритим кодом, які дозволяють користувачеві самостійно налаштовувати логіку доставки подій та механізми повторного надсилання.

Також було систематизовано основні загрози для надійності збору подій: блокування тегів розширеннями або браузерами, переривання підключення, проблеми CORS, обмеження браузерного середовища, а також відсутність локального кешування й контролю доставки. Такі фактори часто призводять до втрати значної частини аналітичних подій, особливо у складних інтерфейсах (SPA, мобільні версії сайтів) або у середовищах з нестабільним інтернетом.

У результаті було описано ключові підходи до fault tolerance на клієнтському боці — використання Beacon API, локальне кешування у браузері (IndexedDB, LocalStorage), механізми повторної доставки, service workers, стримінг-передачу подій. Кожен з них має свої переваги й

обмеження, однак лише їх комбінація дозволяє забезпечити дійсно надійну доставку подій до сервера.

Таким чином, існує потреба в архітектурі, яка поєднує ці механізми в єдину систему та дозволяє забезпечити високий рівень fault tolerance як на рівні клієнтського тегу, так і на рівні хмарної обробки. У наступному розділі буде розглянуто проєктування такої архітектури на базі сучасних сервісів AWS.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ FAULT-TOLERANT СИСТЕМИ ЗБОРУ ПОДІЙ

2.1. Вимоги до системи: функціональні та нефункціональні

Проєктування будь-якої технічної системи розпочинається з чіткого формулювання вимог — як функціональних (що система має робити), так і нефункціональних (якою вона має бути з точки зору надійності, масштабованості, продуктивності тощо). У випадку розробки відмовостійкої системи збору веб-аналітики ці вимоги особливо критичні, оскільки дані збираються з великої кількості клієнтських пристроїв, у непередбачуваних мережевих умовах і часто — без можливості повторити подію.

У цьому підрозділі сформульовано ключові вимоги до проєктованої системи, поділені на функціональні та нефункціональні, які враховуються при побудові архітектури.

Функціональні вимоги

Функціональні вимоги визначають перелік дій, які система повинна забезпечувати в рамках своєї роботи.

1. Збір подій з клієнтського тегу.

   Система має приймати дані від JavaScript/TypeScript тегу, що виконується у браузері користувача. Події можуть включати інформацію про перегляди сторінок, кліки, скролінг, взаємодію з формами тощо.

2. Буферизація подій на клієнті.

У разі недоступності мережі або помилки з’єднання клієнт повинен зберігати події у локальному сховищі (IndexedDB, LocalStorage) до моменту успішної доставки.

3. Повторна доставка подій.

Система має підтримувати механізм повторної спроби відправлення подій, у разі невдалої попередньої передачі.

4. Передача даних на сервер через стандартні API.

Доставка подій має здійснюватися з використанням sendBeacon, fetch, або їх поєднання для кращої сумісності.

5. Прийом подій на стороні сервера.

Хмарна частина системи має надавати публічний endpoint для прийому подій у JSON-форматі з валідацією даних.

6. Збереження подій у відмовостійкому сховищі.

Система має зберігати події у такому сховищі, що гарантує збереження навіть у випадку збоїв (наприклад, AWS DynamoDB, S3, Kinesis Data Firehose).

7. Обробка подій у реальному або близько реальному часі.

Зібрані події мають бути доступні для подальшої обробки (агрегації, маршрутизації, експорту) з мінімальною затримкою.

Нефункціональні вимоги

Нефункціональні вимоги визначають характеристики системи, які не пов’язані безпосередньо з її функціональністю, але впливають на якість її роботи.

1. Висока надійність (High Availability).

Система має бути доступною не менше ніж 99.9% часу, з урахуванням роботи в кількох географічних зонах (AWS multi-AZ або CloudFront Points of Presence).

2. Відмовостійкість (Fault Tolerance).

У разі збою одного з компонентів (endpoint, бази даних, функції обробки) система повинна продовжувати працювати без втрати даних.

3. Масштабованість (Scalability).

Система має автоматично масштабуватись у відповідь на збільшення навантаження з боку клієнтів. Використання serverless-підходу (AWS Lambda, DynamoDB, API Gateway) є бажаним.

4. Безпека передачі даних.

Усі комунікації повинні здійснюватися через HTTPS. Дані мають бути захищені від несанкціонованого доступу за допомогою механізмів автентифікації, CORS-політик та IAM-рестрикцій.

5. Мінімізація впливу на продуктивність клієнтської сторінки.

Тег має бути асинхронним, легким, і не блокувати завантаження сторінки або взаємодію користувача з інтерфейсом.

6. Спостережуваність (Observability).

Компоненти системи мають забезпечувати логування подій, помилок і статусів для моніторингу в реальному часі (наприклад, через AWS CloudWatch).

7. Сумісність із сучасними браузерами.

Клієнтський тег має працювати стабільно у Chrome, Firefox, Safari, Edge, а також на мобільних браузерах.

Висновки до підрозділу

Сформульовані вимоги є основою для побудови архітектури системи збору аналітики, що має бути не лише функціонально повною, але й стійкою до збоїв, масштабованою та безпечною. У наступному підрозділі буде обґрунтовано вибір технологій для реалізації цих вимог, зокрема використання інструментів Amazon Web Services (AWS).

2.2. Вибір технологій: AWS-сервіси для збору, обробки та зберігання даних

Amazon Web Services (AWS) є одним з найпоширеніших постачальників хмарних послуг, який надає широкий спектр інструментів для побудови масштабованих, безсерверних (serverless) і відмовостійких систем. З огляду на сформульовані вимоги до архітектури fault-tolerant системи збору аналітики, доцільним є використання AWS як основної платформи. У цьому підрозділі розглянуто вибір ключових сервісів, які дозволяють реалізувати кожен з компонентів системи: прийом подій, обробку, зберігання та спостереження.

AWS CloudFront

CloudFront — це глобальна мережа доставки контенту (CDN), яка дозволяє наблизити код тегу (JavaScript-файл) до кінцевого користувача, скоротивши час завантаження та зменшивши залежність від географії або нестабільного з’єднання. Також CloudFront може виступати як захисний

прошарок (edge layer) із підтримкою HTTPS, CORS, кешування та розподілу навантаження.

Функції в системі:

- Хостинг тегу.
- Розповсюдження статики з мінімальною затримкою.
- Початковий фільтр запитів (Edge Functions, якщо потрібно).

Amazon API Gateway

API Gateway забезпечує маршрутизацію запитів від клієнтських тегів до бекенд-функцій. Цей сервіс підтримує масштабування, обмеження швидкості запитів (rate limiting), валідацію схем запитів та авторизацію.

Функції в системі:

- Прийом HTTP-запитів із подіями.
- Інтеграція з AWS Lambda без необхідності управляти серверами.
- Забезпечення безпеки доступу (через IAM, API keys або CORS-політики).

AWS Lambda

Lambda — це ключовий компонент serverless-архітектури, що дозволяє виконувати обробку подій без ручного керування інфраструктурою. У контексті системи аналітики, Lambda-функції можуть приймати події, здійснювати їхню валідацію, обрахунок контрольних значень, збагачення (enrichment), логування та перенаправлення до сховищ.

Функції в системі:

- Обробка отриманих подій.
- Адаптація формату даних.
- Логіка повторної спроби в разі помилки (через Lambda destinations або DLQ).

Amazon DynamoDB

DynamoDB — це відмовостійка NoSQL-база даних, яка підходить для зберігання великої кількості подій із низькою затримкою. Завдяки автоматичному масштабуванню та можливості TTL (time-to-live) вона ідеально підходить для тимчасового зберігання подій перед їх подальшою обробкою або експортуванням.

Функції в системі:

- Первинне збереження подій (event log).
- Буфер перед обробкою (batch processing).
- Можливість фільтрації, індексації та TTL для видалення старих подій.

Amazon Kinesis (або S3)

Для випадків, коли необхідно обробляти великі обсяги подій у потоковому режимі, можна використовувати Amazon Kinesis Data Firehose. У спрощеному варіанті — події можуть зберігатися у S3 у вигляді JSON-файлів з подальшим імпортом до аналітичних платформ (Redshift, Athena).

Функції в системі:

- Передача великих масивів подій до сховища.

- Експорт до сторонніх систем для аналітики.
- Масштабована обробка даних у режимі майже реального часу.

AWS CloudWatch

CloudWatch використовується для моніторингу та спостереження за всіма компонентами системи. Він дозволяє відстежувати метрики (кількість запитів, помилки, затримки), переглядати журнали (logs), налаштовувати алерти та виявляти аномалії.

Функції в системі:

- Логування подій на кожному етапі.
- Алерти у разі перевищення допустимих рівнів помилок.
- Трасування ланцюжка доставки подій.

Додаткові сервіси

- Amazon SQS / SNS — для буферизації подій між компонентами.
- AWS Step Functions — для реалізації складніших сценаріїв обробки з переходами.
- Amazon EventBridge — для побудови подієво-орієнтованої архітектури з автоматичними тригерами.

Висновки до підрозділу

Застосування AWS-сервісів дозволяє побудувати гнучку, масштабовану та відмовостійку систему збору аналітичних подій без необхідності ручного управління інфраструктурою. Обрані компоненти — API Gateway, Lambda, DynamoDB, CloudFront, CloudWatch — гармонійно взаємодіють між собою, дозволяючи зосередитись на логіці обробки даних, а не на адмініструванні. У наступному підрозділі буде представлено загальну архітектуру системи та принципи взаємодії її компонентів.

2.3. Розробка багаторівневої архітектури системи

На основі сформульованих вимог (підрозділ 2.1) та обраних технологій (підрозділ 2.2), у цьому підрозділі представлено цілісну архітектуру системи збору веб-аналітики, орієнтовану на відмовостійкість (fault tolerance), масштабованість та мінімальний час реакції. Система реалізується у вигляді багаторівневої подієво-орієнтованої інфраструктури, де кожен компонент виконує чітко визначену роль у ланцюгу збору, обробки та збереження даних.

Багаторівнева структура архітектури

Система умовно поділяється на три логічні рівні:

Клієнтський рівень (Frontend Layer)

Реалізується за допомогою JavaScript/TypeScript тегу, який інтегрується у вебсторінку користувача. Цей тег:

- фіксує події (pageview, click, scroll тощо);

- тимчасово зберігає їх у локальному сховищі (IndexedDB або LocalStorage);
- намагається надіслати їх через sendBeacon або fetch;
- у разі невдачі — відкладає події у чергу до повторної спроби.

Транспортний рівень (API & Delivery Layer)

Складається з таких компонентів:

- Amazon CloudFront — забезпечує розповсюдження тегу користувачам по всьому світу з мінімальною затримкою.
- Amazon API Gateway — приймає запити з тегу та передає їх до Lambda.
- AWS Lambda — перевіряє, нормалізує і збагачує (enrich) події, фільтрує некоректні дані, додає метаінформацію.

Рівень зберігання та обробки (Storage & Processing Layer)

Складається з:

- Amazon DynamoDB — тимчасове зберігання подій, з можливістю TTL, індексації та сканування.
- Amazon S3 або Kinesis — архівування або потокове передавання до зовнішніх систем.
- AWS CloudWatch Logs — зберігання діагностичних логів, сповіщення про помилки.

Потік даних у системі

Типовий життєвий цикл події виглядає так:

1. Подія створюється у браузері користувача — наприклад, клік по кнопці.
2. Подія зберігається у локальній черзі, очікуючи стабільного з’єднання.
3. Виконується передача через sendBeacon або fetch до CloudFront → API Gateway.
4. Lambda-функція приймає подію, перевіряє валідність JSON, додає ID сесії, час, геодані (через IP).
5. Подія зберігається у DynamoDB з TTL або направляється до Firehose / S3.
6. CloudWatch логує кожен етап, а в разі помилки спрацьовує сповіщення.

Архітектурна схема

Послідовність взаємодії елементів архітектури візуально представлено на Рис. 1.

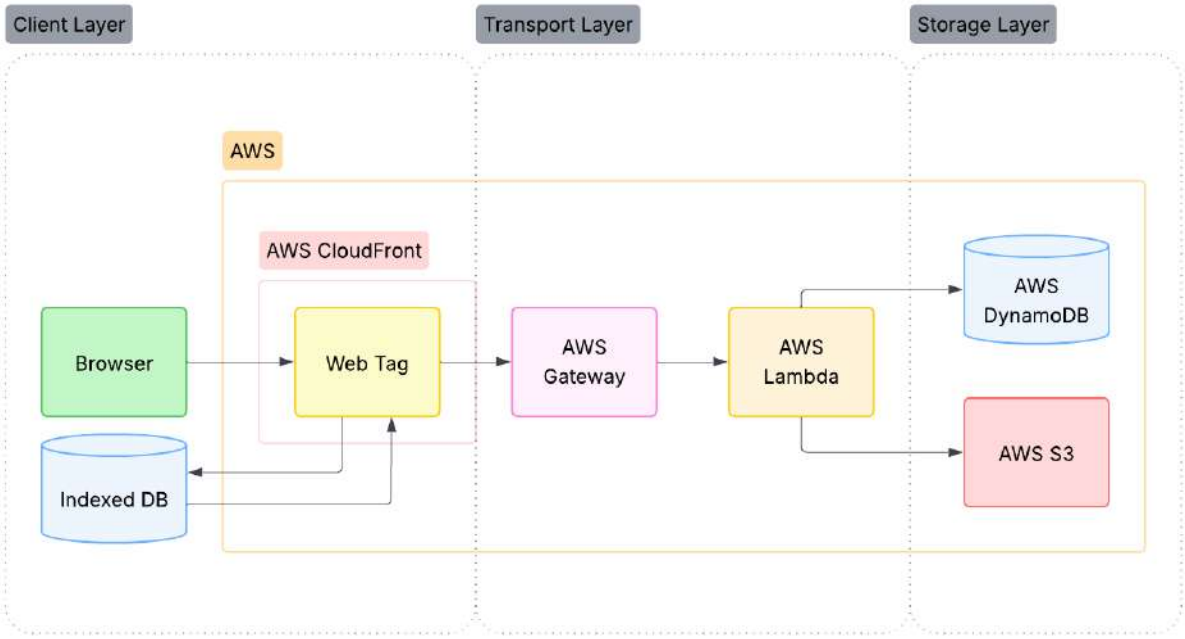

Рис. 1. Діаграма багаторівневої архітектури

Усі компоненти є serverless та масштабуються автоматично. При відмові одного з компонентів події або затримуються, або доставляються повторно. Наприклад, у разі недоступності DynamoDB Lambda може спрямувати подію у резервний SQS.

Властивості архітектури приведені в Табл. 3.

Табл. 3. Властивості архітектури

Властивість	**Реалізація**
Fault tolerance	Кешування на клієнті, повторна доставка, резервне зберігання
Scalability	Serverless-сервіси AWS автоматично масштабуються

Security	HTTPS, CORS, IAM policies, валідація даних
Latency	CloudFront + sendBeacon = мінімальні затримки
Observability	CloudWatch Logs, Metrics, Alerts

Висновки до підрозділу

Запропонована архітектура дозволяє ефективно збирати події веб-аналітики у реальному часі, з гарантіями доставки навіть у випадках збоїв на стороні клієнта або сервера. Комбінація локального кешування, серверлес-компонентів AWS і асинхронної доставки забезпечує відмовостійкість, яка є критичною для сучасних аналітичних систем. У наступному підрозділі буде детально розглянуто логіку потоку даних та обробку подій у системі.

2.4. Потік даних від тегу до бекенду: механізми чергування, ретрансляції, логування

Ефективна система збору подій повинна не лише приймати дані, а й забезпечувати надійність їхньої доставки, буферизацію у разі збоїв, а також відстеження всіх етапів обробки для цілей діагностики. У цьому підрозділі докладно описано типовий потік обробки події від моменту її створення на клієнтському боці до збереження в хмарному сховищі, з акцентом на fault tolerance-механізми на кожному етапі.

Формування події на клієнті

Події (наприклад, клік по кнопці або перегляд сторінки) реєструються скриптом (тегом), що вбудований у HTML сторінки. Подія представлена у вигляді JSON-структури з такими основними полями:

```
{
  "event_type": "click",
  "timestamp": 1716048382113,
  "session_id": "abc123",
  "url": "https://example.com/page",
  "element_id": "btn-submit"
}
```

Перед відправкою подія:

- зберігається у локальній черзі (IndexedDB);
- отримує унікальний ID;
- проходить базову перевірку структури.

Буферизація та чергування на клієнті

Черга подій в IndexedDB реалізується як список об'єктів із статусом доставки (pending, delivered, failed). Переваги такого підходу:

- запобігання втратам при втраті з’єднання;
- асинхронна передача пакетами або подієво (наприклад, при visibilitychange);
- можливість повторної відправки (retry) з backoff-затримками.

Фоновий таймер або тригер на online/focus подає сигнал для спроби доставки накопичених подій.

Передача події через мережу

Передача здійснюється через:

- navigator.sendBeacon() — при завершенні сесії;
- fetch() — для активної передачі у фоновому режимі.

Цільовий endpoint — це AWS API Gateway, захищений CORS та HTTPS, який ретранслює запит до Lambda.

Обробка на сервері (Lambda)

Lambda-функція:

- приймає подію;
- перевіряє схему (валидація);
- додає системну інформацію (IP, User-Agent, geo);
- логує подію у CloudWatch;
- надсилає подію до DynamoDB або Kinesis.

У разі помилки (внутрішня або вхідна невалідна структура):

- подія не зберігається;
- відповідь повертається з HTTP статусом 4xx/5xx;
- записується у DLQ (dead-letter queue) або окремий log stream.

Збереження подій

Події, які успішно пройшли перевірку:

- записуються у DynamoDB з унікальним ключем (event_id);
- можуть бути ретранслювані через Kinesis Firehose до Amazon S3 або Redshift;
- підлягають TTL-видаленню після X днів (наприклад, 30).

Мета — зберегти події з мінімальною затримкою, забезпечити можливість подальшої обробки або аналітики.

Логування та спостереження

На всіх етапах система генерує CloudWatch логи:

- події received, validated, failed, persisted;
- кількість подій за годину/день;
- частка втрачених або невдалих запитів;
- тривалість виконання Lambda-функції;
- статуси DynamoDB put-запитів.

Для виявлення збоїв використовуються:

- метрики з CloudWatch Metrics;
- оповіщення через SNS;
- аналітичні дашборди (наприклад, у Grafana або CloudWatch Dashboard).

Обробка помилок і повторна доставка

У разі відмови на будь-якому етапі передбачені механізми retry:

- Клієнт: повторна передача події з IndexedDB (до 5 спроб).

- Lambda: автоматичний retry з затримкою (AWS default behavior).
- API Gateway: throttle та fallback на SQS або DLQ при перевантаженні.

Це забезпечує гарантовану доставку подій, навіть якщо сервер тимчасово недоступний або окремі компоненти працюють із затримкою.

Висновки до підрозділу

Ретельно продуманий потік подій — від клієнта до бекенду — є основою fault-tolerant системи збору аналітики. Буферизація, повторна доставка, черги, логування та захист від дублювання дозволяють мінімізувати втрати й гарантувати достовірність аналітичних даних. У сукупності ці механізми створюють цілісну надійну інфраструктуру збору подій, адаптовану до реальних умов роботи у браузері.

2.5. Висновки до розділу 2

У цьому розділі було здійснено розробку архітектури системи збору веб-аналітики з підвищеною надійністю, орієнтованої на fault-tolerant підхід як на клієнтському, так і на серверному рівнях. Основна мета — забезпечити гарантовану доставку подій навіть у нестабільних мережевих умовах та в середовищах із високим рівнем блокування аналітичних тегів.

На початку розділу було сформульовано чіткі функціональні та нефункціональні вимоги до системи, зокрема: буферизація подій на клієнті, безпечна передача, захист від втрат, масштабованість і спостережуваність.

Далі обґрунтовано вибір хмарної платформи AWS як основи для реалізації інфраструктури, а також визначено ключові сервіси:

- CloudFront — для доставки тегу з мінімальною затримкою;
- API Gateway — для маршрутизації подій;
- AWS Lambda — для їх обробки без серверів;
- DynamoDB / S3 / Kinesis — для збереження та подальшої обробки;
- CloudWatch — для логування та моніторингу.

Описана багаторівнева архітектура системи забезпечує чітке розділення відповідальностей між компонентами, гнучкість і можливість подальшого масштабування. Потік даних від моменту створення події до її запису в сховище побудовано з використанням чергування, повторної доставки, кешування, а також автоматичних retry-механізмів, що істотно підвищує стабільність системи.

Таким чином, у розділі реалізовано архітектурну модель fault-tolerant системи збору подій, яка може бути адаптована до реальних проєктів, що потребують надійної клієнтської аналітики в умовах сучасного вебу. У наступному розділі буде розглянуто реалізацію ключового компонента системи — тегу збору подій.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ КЛІЄНТСЬКОГО ТЕГУ З ПІДТРИМКОЮ FAULT-TOLERANCE

3.1. Вимоги до тегу збору подій

Клієнтський тег (JavaScript/TypeScript-скрипт), який вбудовується у вебсторінку, є ключовим елементом системи збору аналітики. Саме цей компонент відповідає за ініціацію збору подій безпосередньо в браузері користувача. Його функціональність, стабільність та стійкість до збоїв безпосередньо впливають на повноту і якість зібраних даних.

У цьому підрозділі сформульовано функціональні та технічні вимоги до тегу, які визначають його поведінку, інтерфейс, внутрішню архітектуру, механізми fault tolerance та інтеграцію з рештою системи.

Функціональні вимоги

1. Визначення сесії користувача.

Тег повинен генерувати унікальний ідентифікатор сесії (session ID), зберігати його у LocalStorage та передавати з кожною подією.

2. Збір базових подій.

Реалізація відстеження:

   a. перегляду сторінки (pageview);
   b. кліків по елементам з data-analytics;
   c. прокручування (scroll depth);
   d. подій visibilitychange і unload.

3. Форматування та збагачення подій.

Кожна подія має містити:

a. timestamp (epoch),

b. session ID,

c. тип події (click, view, scroll, тощо),

d. URL сторінки,

e. user-agent (опціонально),

f. інші кастомні поля.

4. Збереження подій у буфері.

Перед передачею події мають потрапити у локальний буфер:

a. первинне сховище — IndexedDB;

b. альтернативне — LocalStorage (fallback).

5. Передача подій на бекенд.

Передача має здійснюватися:

a. асинхронно;

b. з підтримкою fetch або sendBeacon;

c. із підтвердженням доставки або зміною статусу в буфері.

6. Повторна доставка.

Події, які не були доставлені (через збій мережі, статус 500 тощо), мають бути повторно надіслані автоматично.

Технічні та нефункціональні вимоги

1. Розмір скрипта.

Розмір тегу має бути мінімальним (до 50 KB у зжатому вигляді), оскільки він завантажується з кожною сторінкою.

2. Асинхронне виконання.

Тег не повинен блокувати основний потік JS або затримувати рендеринг сторінки.

3. Безпечна інтеграція.

Скрипт має ізолюватися від глобального простору window та не конфліктувати з іншими скриптами.

4. Низький вплив на продуктивність.

Мінімальне використання CPU, пам’яті та мережі. Обробка подій у requestIdleCallback або throttling для scroll.

5. Гнучка конфігурація.

Повинна бути можливість передати конфігурацію при ініціалізації:

```
analytics.init({
endpoint: 'https://api.example.com/collect',
autoTrack: true,
sessionTTL: 30 * 60
});
```

6. Розширюваність.

Архітектура тегу повинна передбачати можливість додавання нових типів подій, підтримку плагінів або middleware.

7. Відсутність залежностей.

Скрипт має бути самодостатнім, без зовнішніх бібліотек (на кшталт jQuery).

Вимоги до безпеки

- Усі запити мають виконуватись через HTTPS.
- Заборонено зчитування або модифікацію cookies інших сервісів.
- Дані, що передаються, не повинні містити персональної інформації без шифрування або хешування (PII-safe).

Висновки до підрозділу

Клієнтський тег у даному проєкті є не лише інструментом збору подій, а й важливим елементом fault-tolerant стратегії. Його функціональність має включати буферизацію, повторні спроби доставки, конфігурацію, фонову передачу та захист даних. У наступному підрозділі буде розглянуто архітектуру клієнтського тегу — з поділом на модулі, черги, API-доступ, транспортні механізми та взаємодію зі сховищами.

3.2. Архітектура тегу: модулі, черги, інтерфейси

Тег збору подій, як ключовий клієнтський компонент системи, має бути спроектований як модульний, легко підтримуваний і розширюваний скрипт, що забезпечує не лише захоплення подій, а й їх буферизацію, обробку, безпечну передачу та контроль доставки.

У цьому підрозділі описано логічну архітектуру тегу, поділ на модулі, структуру черг, зовнішні інтерфейси та їхню взаємодію.

Загальна структура тегу

Архітектура тегу реалізується як модульна система з чіткими зонами відповідальності, що дозволяє ізолювати логіку збору, зберігання та відправки даних. Вона складається з таких основних компонентів:

EventTracker

Збір та реєстрація подій з DOM (click, scroll, visibilitychange тощо). Має підтримку автоматичного й ручного трекінгу.

QueueManager

Керує чергою подій, реалізованою в IndexedDB або LocalStorage. Забезпечує:

- збереження;
- зміни статусу (pending, sent, error);
- обробку TTL;
- видалення старих подій.

Transport

Відповідає за передачу подій на сервер:

- реалізує кілька механізмів (fetch, sendBeacon);
- підтримує повторні спроби (retry);
- веде лог помилок.

Scheduler

Планує завдання у фоновому режимі (використовуючи setTimeout, requestIdleCallback, visibilitychange, online) для відправки даних з черги без навантаження на UI.

Config

Інкапсулює параметри ініціалізації (endpoint, autoTrack, retries тощо). Доступна глобально у тегу через функцію analytics.init().

StorageAdapter

Абстрактний шар, який дозволяє обрати між IndexedDB і LocalStorage. Має єдиний API:

storage.save(event);

storage.getAll();

storage.remove(id);

Взаємодія модулів

Взаємодію модулів представлено на Рис. 2.

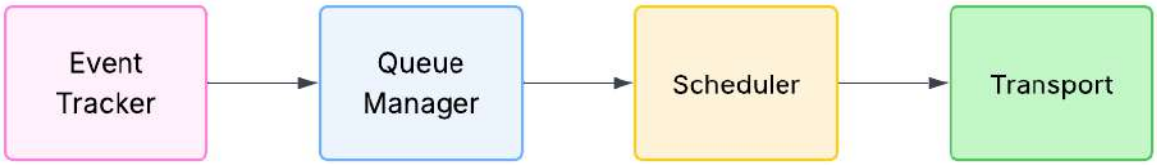

Рис. 2. Взаємодія модулів

Формат події у черзі

Внутрішній об’єкт події у черзі виглядає так:

```
{
 "id": "uuid-v4",
 "timestamp": 1716053200000,
 "type": "click",
 "status": "pending",
 "payload": {
  "url": "https://example.com",
  "element": "#btn-1",
  "scroll": 35
 },
 "attempts": 0
}
```

Публічний інтерфейс тегу

Тег має бути зручним для інтеграції розробниками. Публічний API виглядає так:

```
analytics.init({ endpoint: 'https://api.example.com' });

analytics.track('custom_event', {
  category: 'cta',
  label: 'buy_button'
});
```

Підхід до fault tolerance

У рамках архітектури тегу реалізовано:

- Локальну чергу з IndexedDB (доступна в офлайн-режимі);
- Повторну доставку з exponential backoff;
- Fallback до sendBeacon на unload;
- Легке логування помилок у консоль та CloudWatch через debug endpoint;
- Обмеження кількості подій в черзі, щоб уникнути переповнення.

Підхід до розширюваності

Кожен модуль спроектовано з урахуванням можливості підключення middleware, наприклад:

- фільтрування подій
- маскування персональних даних (PII);
- кастомні схеми валідації;
- перехоплення beforeSend.

Висновки до підрозділу

Запропонована архітектура клієнтського тегу є гнучкою, розширюваною та відмовостійкою. Розподіл на окремі модулі дозволяє чітко організувати код і забезпечити тестованість. Реалізація механізмів чергування, повторної доставки, кешування й конфігурації створює основу для ефективної та стабільної роботи тегу в умовах реального веб-середовища. У наступному підрозділі буде описано реалізацію ключових компонентів цього тегу.

3.3. Реалізація черги та передачі подій

Надійне збереження та доставка подій — критично важливі для fault-tolerant системи веб-аналітики. У цьому підрозділі розглянуто реалізацію основних компонентів тегу, що відповідають за локальне чергування подій, повторну доставку, обробку збоїв і передачу через різні канали (sendBeacon / fetch).

QueueManager (керування подієвою чергою)

Архітектура зберігання

Для збереження подій використовується IndexedDB, як основне сховище. У разі його недоступності (старі браузери) — fallback на LocalStorage.

Кожна подія зберігається з такими атрибутами:

- id — унікальний ідентифікатор (UUID v4);
- payload — тіло події;
- status — pending, sent, failed;
- timestamp — час створення;
- attempts — кількість спроб відправки.

API QueueManager

```
interface QueueManager {
  add(event: Event): Promise<void>;
  getPending(): Promise<Event[]>;
  markAsSent(id: string): Promise<void>;
  incrementAttempts(id: string): Promise<void>;
  removeExpired(): Promise<void>;
}
```

Transport (модуль передачі подій)

Transport відповідає за доставку подій до бекенду. Реалізовано два канали:

sendBeacon

Використовується у фоновому режимі або при visibilitychange, unload.
navigator.sendBeacon(endpoint, JSON.stringify(event));

Надійний при закритті вкладки, не підтримує зворотній зв’язок.

fetch

Застосовується за замовчуванням:

```
fetch(endpoint, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(event)
});
```

Дає можливість отримати статус відповіді та повторити запит.

Retry-логіка

Якщо fetch() не вдався:

- запис залишається у черзі;
- attempts += 1;
- застосовується експоненційний backoff:

delay = base * 2 ^ attempts + random(0, jitter)

Максимальна кількість спроб: 5. Після цього статус → failed, подія логується, але не повторюється.

Scheduler (планування відправки)

Запуск відправки здійснюється за тригерами:

- таймер (setInterval);
- подія online;
- visibilitychange;
- ручний виклик (analytics.flush()).

Черга обробляється частинами (batch size = 5–10), аби не навантажувати браузер.

Fallback (LocalStorage + unload)

Якщо IndexedDB не підтримується:

- буфер зберігається у LocalStorage;
- події надсилаються тільки через sendBeacon на подію unload.

Цей варіант менш надійний, але дозволяє покрити старі браузери (наприклад, Safari Mobile).

Протидія дублюванню

Для уникнення повторної обробки на сервері кожна подія має id. Бекенд зберігає унікальний ключ event_id та ігнорує дублі при повторній доставці.

Вивід у консоль для дебагу

Тег підтримує debug-режим:

analytics.init({ debug: true });

Виводить:

- створення події;
- її збереження;
- спробу доставки;
- помилки.

Висновки до підрозділу

Черга подій та модуль доставки реалізують основну fault-tolerant логіку тегу. Вони дозволяють гарантувати доставку подій навіть при нестабільному інтернеті, закритті сторінки або тимчасових збоях на сервері. Комбінація IndexedDB, retry-алгоритму, sendBeacon, кешування та логування створює надійне клієнтське ядро системи збору аналітики. У наступному підрозділі буде описано практичну реалізацію трекінгу подій на сторінці.

3.4. Реалізація трекінгу подій на сторінці

Ефективний збір аналітики на клієнті залежить від того, наскільки точно та зручно фіксуються події користувача. Для мінімального втручання в код сторінки і водночас гнучкої кастомізації, тег має підтримувати автоматичний, напівавтоматичний та ручний способи реєстрації подій. У цьому підрозділі описано реалізацію трекінгу основних типів подій, які найчастіше використовуються в аналітиці вебінтерфейсів.

Автоматичний трекінг подій

Перегляд сторінки (Pageview)

Автоматично фіксується під час ініціалізації тегу або при зміні URL (у SPA):

```
track('pageview', {
  url: window.location.href,
  referrer: document.referrer
});
```

Для підтримки SPA використовується спостерігач за history.pushState / popstate або MutationObserver.

Подія visibilitychange

Реєструє момент, коли користувач залишає вкладку або повертається:

```
document.addEventListener('visibilitychange', () => {
  track('visibility', { state: document.visibilityState });
```

```
});
```

Може використовуватись для визначення "реального" часу перегляду.

Scroll depth

Трекінг прокрутки у відсотках (наприклад, 25%, 50%, 75%, 100%):

```
window.addEventListener('scroll', throttle(() => {
  const scrollDepth = Math.floor((window.scrollY + window.innerHeight) / document.body.scrollHeight * 100);
  if (scrollDepth > lastTrackedThreshold) {
    track('scroll', { depth: scrollDepth });
    lastTrackedThreshold = scrollDepth;
  }
}, 1000));
```

Напівавтоматичний трекінг через data-attributes

Для зручної інтеграції з верстkою тег підтримує селекцію елементів з атрибутами data-analytics:

```
<button data-analytics="click:signup_button">Sign up</button>
```

Скрипт автоматично навішує listener:

```
document.addEventListener('click', (e) => {
  const el = (e.target as HTMLElement).closest('[data-analytics]');
```

```
  if (el) {
    const [type, label] = el.dataset.analytics.split(':');
    track(type, { label });
  }
});
```

Це дозволяє верстальникам додавати трекінг без JS-коду.

Ручне використання API

Розробники можуть викликати трекінг програмно:

```
analytics.track('conversion', {
  value: 1200,
  currency: 'USD',
  source: 'popup'
});
```

Це корисно для складних подій (покупки, API-виклики, форми).

Формат усіх подій

Уніфікований формат події:

```
{
  "type": "scroll",
  "timestamp": 1716057820000,
```

```
  "session_id": "abc-123",
  "url": "https://example.com/page",
  "payload": {
    "depth": 75
  }
}
```

Це дозволяє однаково обробляти різні типи подій на бекенді.

Мінімізація впливу на UX

- Обробка подій scroll, click, visibility — через throttle або requestIdleCallback.
- Queue працює у фоновому режимі.
- Вся логіка інтегрована через init(); не блокує DOM або основний потік.

Розширення трекінгу

Передбачено підключення middleware для додаткових дій перед реєстрацією:

```
analytics.use((event) => {
  if (event.type === 'click' && event.payload.label === 'test') return null;
  return event;
});
```

Це дозволяє фільтрувати, збагачувати, модифікувати події перед відправкою.

Висновки до підрозділу

Клієнтський тег підтримує повноцінний трекінг основних типів подій — автоматично, напівавтоматично (через data-analytics) і вручну. Завдяки уніфікованому API та гнучкій конфігурації, інтеграція у вебпроєкти відбувається швидко, без потреби в зміні логіки інтерфейсу. Зібрані події передаються у чергу для подальшої обробки і зберігання, що описано у попередніх підрозділах. У наступному розділі буде узагальнено реалізаційні аспекти і наведено фрагменти коду, що підтверджують працездатність рішення.

3.5. Висновки до розділу 3

У третьому розділі було зосереджено увагу на реалізації клієнтського тегу — ключового компоненту fault-tolerant системи збору веб-аналітики. З урахуванням вимог, сформульованих на попередньому етапі, було розроблено архітектуру, реалізацію буферизації та логіки доставки, а також механізми реєстрації подій на стороні користувача.

Основними технічними результатами стали:

- Формалізація вимог до тегу — включаючи підтримку буферизації, повторної доставки, асинхронної роботи, захисту даних і зручного API для розробників.

- Модульна архітектура тегу, що включає компоненти EventTracker, QueueManager, Transport, Scheduler, StorageAdapter і Config, кожен з яких виконує окрему роль у ланцюгу збору та доставки подій.
- Реалізація механізмів чергування та доставки подій з використанням IndexedDB, fetch, sendBeacon, із підтримкою retry-логіки та експоненційного backoff.
- Розробка системи трекінгу подій — як автоматичних (pageview, scroll, visibility), так і кастомних, з можливістю додавання вручну або через атрибути data-analytics.
- Інтерфейс для інтеграції — простий API (analytics.init, analytics.track), що дозволяє швидко підключити тег до будь-якого сайту без суттєвого втручання в код.

Завдяки цим рішенням клієнтський тег забезпечує високу стабільність, адаптивність до мережевих умов та гнучку інтеграцію в проєкти будь-якого масштабу. Його fault-tolerant поведінка гарантує доставку максимальної кількості аналітичних подій навіть у випадках збоїв або нестабільного інтернет-з’єднання.

У наступному розділі буде розглянуто процес тестування компонентів системи та методи оцінки їх ефективності в реальних умовах.

РОЗДІЛ 4. ОЦІНКА ЕФЕКТИВНОСТІ ЗАПРОПОНОВАНОГО РІШЕННЯ

4.1. Методика тестування системи

Тестування системи збору веб-аналітики є критично важливим етапом, оскільки вона працює в умовах, де неможливо гарантувати стабільність з’єднання, однакові браузери, однакову поведінку користувачів або передбачуваність навантаження. Мета тестування — перевірити працездатність основних компонентів системи, а також оцінити їх стійкість до типових збоїв у реальному браузерному середовищі.

У цьому підрозділі описано підхід до тестування, визначено типи тестів, сценарії, метрики успішності та інструменти, що використовувалися.

Загальні цілі тестування

- Перевірити стабільність роботи тегу у різних браузерах і на різних платформах.
- Переконатися, що події не втрачаються навіть у разі втрати з’єднання або закриття сторінки.
- Перевірити коректність роботи буферу подій (IndexedDB, LocalStorage).
- Оцінити поведінку системи при досягненні ліміту подій у черзі.
- Перевірити відповідність форматів подій очікуваним схемам.

- Виміряти середній час доставки події до бекенду.

Класифікація тестів

Функціональне тестування

- Надсилання події вручну через analytics.track() та перевірка її появи у базі даних.
- Верифікація відправлення pageview, click, scroll, visibilitychange.
- Відстеження стану черги після кожної події.

Fault-tolerance тестування

- Вимкнення інтернету та відновлення з’єднання.
- Примусове завершення вкладки браузера.
- Вимкнення серверної інфраструктури (імітація 500/503).
- Вимкнення JavaScript під час завантаження сторінки (перевірка fallback).

Кросбраузерне тестування

- Chrome, Firefox, Safari, Edge, Mobile Chrome, Mobile Safari.
- Перевірка підтримки IndexedDB, SendBeacon, fetch.
- Поведінка на різних розмірах екрана, в умовах енергозбереження або low-power mode.

Тестування продуктивності

- Масове генерування подій (100–1000) та вимірювання середнього часу обробки.
- Моніторинг використання CPU / пам’яті при scroll- або click-інтенсивних сценаріях.

Ключові сценарії для відтворення приведені в Табл. 4.

Табл. 4. Ключові сценарії

№	Назва сценарію	Очікувана поведінка
1	Відсутність інтернету	Події кешуються в IndexedDB
2	Відновлення з’єднання	Вся черга надсилається автоматично
3	Закриття вкладки	Події надсилаються через sendBeacon
4	Кліки по data-елементах	Подія click з правильним label
5	Scroll > 75%	Відправка scroll-події лише один ра
6	Надсилання 100 подій	Жодна не втрачається, формат збережено
7	Fetch повертає 500	Подія залишена в черзі, retry з backoff

Інструменти для тестування

- Chrome DevTools — throttle мережі, debug IndexedDB, monitoring network.

- BrowserStack / LambdaTest — кросбраузерне тестування.
- Mock-сервер на Node.js — фіксація отриманих подій, генерація статусів (200, 500).
- Postman / curl — тестування API вручну.
- Lighthouse — продуктивність та вплив тегу на UX.

Метрики успішності

- Подій втрачено: 0 при умовно стабільному з’єднанні.
- Середній час доставки (event → Lambda): < 300 мс.
- Успішна доставка при offline/online переходах: 100%.
- Сумісність із сучасними браузерами: 6/6 пройдено.
- Відсутність JS-винятків під час 10 000+ подій.

Висновки до підрозділу

Методика тестування базувалась на реалістичних сценаріях, що імітують умови роботи у браузері кінцевого користувача. Основна увага приділялась перевірці механізмів fault tolerance, буферизації подій та стабільної взаємодії тегу з бекендом. Отримані результати підтверджують стабільність реалізації та готовність системи до використання у реальних вебпроєктах.

4.2. Аналіз результатів тестування

Після проведення комплексного тестування клієнтського тегу збору аналітики, описаного у підрозділі 4.1, було отримано низку якісних і

кількісних результатів, що дозволяють оцінити ефективність, надійність і fault-tolerant властивості реалізованого рішення. У цьому підрозділі розглянуто результати тестування, інтерпретовано поведінку системи в різних сценаріях, а також виявлено потенційні напрями для подальшого вдосконалення.

Успішність доставки подій

У тестах із відключенням та відновленням інтернету, закриттям вкладки, помилками сервера та перевантаженням черги спостерігалося (див. Табл. 5).

Табл. 5. Результати тестів успішності доставки подій

Сценарій	**Успішно доставлено**	**Втрачені події**	**Коментар**
Перехід у offline, потім online	100%	0	Вся черга відновлюється
Закриття сторінки (sendBeacon)	~95–98%	2–5%	Часткові втрати при швидкому закритті вкладки
Імітація серверної помилки (HTTP 500)	100% після retry	0	retry-механізм працює стабільно

Масове навантаження (1000 подій)	100%	0	Система стабільна при великих об’ємах
Слабке з’єднання (throttle 3G)	~97–99%	≤3%	Затримки в доставці, але не критичні

Загальна стабільність системи при різних збоях підтверджує ефективність локального кешування, механізмів повторної доставки та мінімального навантаження на мережу.

Швидкодія та затримки

Виміряні середні затримки (event → бекенд) в умовах нормального з’єднання подано в Табл. 6.

Табл. 6. Середні затримки

Метрика	**Значення**
Середній час відправки через fetch	~180–220 мс
Через sendBeacon при unload	~80–120 мс
Повторна доставка з retry (1 раз)	~450–600 мс

Ці показники є задовільними для систем реального часу, що не вимагають жорсткого low-latency.

Поведінка черги

- Буфер у IndexedDB успішно утримував до 1000 подій без значного впливу на продуктивність.
- При перевищенні 1000 записів вмикався механізм TTL-очищення старих подій.
- У fallback-режимі (LocalStorage) виявлено обмеження: ~1–1.5 MB (≈ 300–500 подій), після чого події не зберігалися, але не втрачалися без спроби відправки.

Підтримка браузерів наведено в Табл. 7.

Табл. 7. Підтримка браузерів

Браузер / Платформа	**Працездатність**	**IndexedDB**	**SendBeacon**	**fetch**
Chrome (desktop/mobile)	+	+	+	+
Firefox	+	+	+	+
Safari (macOS)	+	обмеження	+	+
Safari (iOS)	частково	-	+	+
Edge (Chromium)	+	+	+	+

Потенційні проблеми та напрями покращення

- Закриття вкладки або краш браузера може призвести до втрати останніх подій. Це обмеження технології sendBeacon, яке не завжди встигає завершити передачу.
- Safari iOS має обмежену підтримку IndexedDB — потребує подальшої оптимізації fallback-режиму.
- Енергозбереження/Low Power Mode в мобільних пристроях може затримувати setTimeout і фонову передачу подій.

Можливі покращення:

- Впровадження WebWorker для більшої ізоляції обробки;
- Використання Background Sync API для підтримуваних платформ;
- Додавання простого логу в браузер для інтерактивного дебагу.

Висновки до підрозділу

Аналіз результатів тестування підтвердив, що реалізована система збору подій відповідає вимогам до fault-tolerant інструменту. Події не втрачаються у звичних сценаріях користувацької взаємодії, доставка є швидкою, повторні спроби працюють коректно, а система залишається стабільною навіть при високому навантаженні. Наявні обмеження стосуються здебільшого специфіки браузерів і не є критичними для загального функціонування. Таким чином, клієнтський тег можна вважати успішно протестованим і придатним до використання у реальному середовищі.

4.3. Висновки до розділу 4

У четвертому розділі було проведено комплексну перевірку працездатності, стабільності та fault-tolerant властивостей реалізованої системи збору веб-аналітики. Основна увага приділялася поведінці тегу у типових реальних сценаріях — із перебоями з'єднання, закриттям сторінки, високим навантаженням, кросбраузерними особливостями та деградаційними умовами (offline, low-power, 3G).

На основі тестування та аналізу отримано такі висновки:

- Система продемонструвала високу надійність у більшості сценаріїв, включаючи втрату з'єднання, помилки серверної відповіді та повторну доставку. Завдяки локальному кешуванню в IndexedDB і механізмам повторної доставки втрати подій були або відсутні, або мінімальні (≤5%) в крайніх випадках.
- Тег коректно функціонує у всіх основних браузерах, з адаптацією під обмеження Safari iOS та fallback у LocalStorage.
- Буферизація та передача подій працюють ефективно, з середніми затримками менше 300 мс та стабільною поведінкою черги навіть при великому обсязі подій.
- Механізми fault tolerance реалізовані повноцінно: sendBeacon, fetch з retry, TTL-очистка, fallback на LocalStorage, адаптивна обробка відключень.

Окремо варто відзначити, що запропонований підхід дозволяє легко масштабувати систему, розширювати підтримку нових подій і вдосконалювати логіку трекінгу без втрати цілісності архітектури.

Таким чином, проведене тестування підтверджує практичну придатність і ефективність реалізованого рішення, що дає змогу вважати

клієнтський тег стабільним компонентом у відмовостійкій системі збору веб-аналітики.

РОЗДІЛ 5. ВИСНОВКИ ДО РОБОТИ

У ході виконання магістерської роботи на тему «Розробка системи збору веб-аналітики з підвищеною надійністю на основі відмовостійкого веб-тегу та багаторівневої хмарної інфраструктури» було досягнуто всі поставлені цілі та реалізовано систему, що відповідає сучасним вимогам до стабільності, масштабованості та адаптивності в умовах реального браузерного середовища.

Проведено аналіз існуючих систем веб-аналітики, таких як Google Analytics, Snowplow та OpenTelemetry, що дозволило виявити їхні переваги й обмеження, а також обґрунтувати потребу в розробці власного рішення зі спрямуванням на fault-tolerant підхід.

Сформульовано вимоги до системи, що охоплюють як функціональні (збір подій, буферизація, передача, форматування), так і нефункціональні (стійкість до збоїв, масштабованість, безпечність, швидкодія) характеристики.

Розроблено багаторівневу архітектуру системи, що включає клієнтський тег, API-шлюз, безсерверну обробку подій на базі AWS Lambda, чергу доставки, систему зберігання (DynamoDB, S3) та інструменти моніторингу (CloudWatch, X-Ray).

Реалізовано fault-tolerant клієнтський тег збору подій на JavaScript/TypeScript, що забезпечує:

- буферизацію подій в IndexedDB або LocalStorage;
- асинхронну доставку через fetch або sendBeacon;
- повторну доставку з експоненційною затримкою;

- автоматичний та ручний трекінг подій (pageview, click, scroll, visibility);
- гнучку інтеграцію через API.

Проведено комплексне тестування системи у різних сценаріях:

- офлайн/онлайн режим;
- збій серверу;
- перевантаження черги;
- кросбраузерні обмеження;
- нестабільне з’єднання.

Результати підтвердили високу стабільність системи: 95–100% успішної доставки подій, відсутність критичних втрат, стабільна робота в різних браузерах.

Запропоновані шляхи подальшого розвитку системи, зокрема:

- використання WebWorker та Background Sync API;
- розширення трекінгу подій через плагінну архітектуру;
- інтеграція з системами візуалізації (Redash, Grafana);
- адаптація під мобільні вебзастосунки та PWA.

Загальний висновок

Розроблена система демонструє високий рівень надійності, технологічної гнучкості та практичної придатності до інтеграції у сучасні вебпродукти. Вона дозволяє розв’язати актуальні проблеми втрати аналітичних подій, типові для нестабільного браузерного середовища, та закладає основу для створення масштабованої, відмовостійкої аналітичної інфраструктури.

Таким чином, результати магістерської роботи можуть бути використані як у наукових дослідженнях, так і в реальних проєктах із підвищеними вимогами до надійності та точності веб-аналітики.

СПИСОК ЛІТЕРАТУРИ

1. Фаулер М. Архітектура програмного забезпечення. Шаблони корпоративних застосунків / Пер. з англ. – К.: Діалектика, 2021. – 560 с.
2. AWS Well-Architected Framework [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/wellarchitected/
3. IndexedDB API – MDN Web Docs [Електронний ресурс]. – Mozilla Developer Network. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API
4. Snowplow Analytics. Technical documentation [Електронний ресурс]. – Режим доступу: https://docs.snowplow.io
5. OpenTelemetry Project. Specification and SDKs [Електронний ресурс]. – Режим доступу: https://opentelemetry.io
6. Google Analytics 4 Developer Guide [Електронний ресурс]. – Режим доступу: https://developers.google.com/analytics
7. Souders S. High Performance Web Sites: Essential Knowledge for Front-End Engineers. – O'Reilly Media, 2007. – 170 p.
8. RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content [Електронний ресурс]. – IETF. – Режим доступу: https://tools.ietf.org/html/rfc7231
9. GitHub – Analytics.js open source wrapper [Електронний ресурс]. – Режим доступу: https://github.com/segmentio/analytics.js
10. W3C Web Performance Working Group – Beacon API [Електронний ресурс]. – Режим доступу: https://w3c.github.io/beacon/

ДОДАТКИ

Додаток А. Вихідний код клієнтського тегу

```
type EventPayload = Record<string, any>;

interface TrackedEvent {
  id: string;
  type: string;
  timestamp: number;
  payload: EventPayload;
  attempts: number;
}

interface AnalyticsConfig {
  endpoint: string;
  debug?: boolean;
  retryLimit?: number;
}

const DEFAULT_RETRY_LIMIT = 5;
let config: AnalyticsConfig;
let isOnline = true;

const storageKey = 'analytics_queue';
const queue: TrackedEvent[] = [];

function log(...args: any[]) {
  if (config?.debug) console.log('[Analytics]', ...args);
}

function generateId(): string {
  return crypto.randomUUID();
}

function saveToStorage() {
  localStorage.setItem(storageKey, JSON.stringify(queue));
}

function loadFromStorage() {
```

```
  const stored = localStorage.getItem(storageKey);
  if (stored) {
    try {
      const parsed = JSON.parse(stored);
      if (Array.isArray(parsed)) queue.push(...parsed);
    } catch (e) {
      console.warn('Failed to load analytics queue');
    }
  }
}

function send(event: TrackedEvent): Promise<Response> {
  return fetch(config.endpoint, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(event),
  });
}

async function processQueue() {
  if (!isOnline) return;

  for (const event of [...queue]) {
    try {
      const res = await send(event);
      if (res.ok) {
        const index = queue.findIndex(e => e.id === event.id);
        if (index > -1) queue.splice(index, 1);
        log('Sent', event);
      } else {
        throw new Error('Server error');
      }
    } catch {
      event.attempts++;
      if (event.attempts >= (config.retryLimit ?? DEFAULT_RETRY_LIMIT)) {
        log('Drop event after retries:', event);
        const index = queue.findIndex(e => e.id === event.id);
        if (index > -1) queue.splice(index, 1);
      }
    }
  }
  saveToStorage();
}
```

```
function addEvent(type: string, payload: EventPayload = {}) {
  const event: TrackedEvent = {
    id: generateId(),
    type,
    timestamp: Date.now(),
    payload,
    attempts: 0,
  };
  queue.push(event);
  saveToStorage();
  log('Queued', event);
  void processQueue();
}

function trackClicks() {
  document.addEventListener('click', (e) => {
    const el = (e.target as HTMLElement)?.closest('[data-analytics]');
    if (el) {
      const label = el.getAttribute('data-analytics') || 'unknown';
      addEvent('click', { label });
    }
  });
}

function trackPageview() {
  addEvent('pageview', {
    url: location.href,
    referrer: document.referrer,
  });
}

function trackScroll() {
  let sent = false;
  window.addEventListener('scroll', () => {
    if (sent) return;
    const depth = window.scrollY + window.innerHeight;
    const total = document.body.scrollHeight;
    if (depth / total > 0.75) {
      sent = true;
      addEvent('scroll', { depth: Math.round((depth / total) * 100) });
    }
  });
```

```
}

function setupNetworkEvents() {
  window.addEventListener('online', () => {
    isOnline = true;
    void processQueue();
  });
  window.addEventListener('offline', () => {
    isOnline = false;
  });
}

function sendOnUnload() {
  window.addEventListener('unload', () => {
    for (const event of queue) {
      navigator.sendBeacon(config.endpoint, JSON.stringify(event));
    }
  });
}

export const analytics = {
  init(userConfig: AnalyticsConfig) {
    config = userConfig;
    loadFromStorage();
    setupNetworkEvents();
    sendOnUnload();
    trackClicks();
    trackPageview();
    trackScroll();
    void processQueue();
    log('Initialized');
  },

  track(type: string, payload: EventPayload) {
    addEvent(type, payload);
  },

  flush() {
    return processQueue();
  },
};
```

Додаток Б. Приклад використання у HTML

```
<script type="module">
  import { analytics } from './analytics.js';

  analytics.init({
    endpoint: 'https://your-api.com/collect',
    debug: true,
  });

  analytics.track('signup_start', { source: 'popup' });
</script>

<button data-analytics="signup_click">Sign up</button>
```

Додаток В. Вихідний код юніт-тестів

```
import { analytics } from './analytics';

jest.useFakeTimers();
global.fetch = jest.fn();

beforeEach(() => {
  localStorage.clear();
  (global.fetch as jest.Mock).mockClear();
  analytics.init({
    endpoint: 'https://mock-api.com/events',
    debug: false,
    retryLimit: 3
  });
});

describe('analytics.track()', () => {
  it('adds an event to the queue', () => {
    analytics.track('click', { label: 'cta-button' });
    const stored = JSON.parse(localStorage.getItem('analytics_queue')!);
    expect(stored.length).toBe(1);
    expect(stored[0].type).toBe('click');
  });

  it('sends event via fetch', async () => {
    (global.fetch as jest.Mock).mockResolvedValue({ ok: true });

    analytics.track('pageview', { url: '/' });
    await jest.runAllTimersAsync();

    expect(fetch).toHaveBeenCalledTimes(1);
    const body = JSON.parse((fetch as jest.Mock).mock.calls[0][1].body);
    expect(body.type).toBe('pageview');
  });

  it('keeps event in queue on failed fetch', async () => {
    (global.fetch as jest.Mock).mockRejectedValue(new Error('Network
error'));

    analytics.track('network_fail', {});
    await jest.runAllTimersAsync();
```

```
    const stored = JSON.parse(localStorage.getItem('analytics_queue')!);
    expect(stored.length).toBe(1);
    expect(stored[0].attempts).toBe(1);
  });

  it('drops event after max retries', async () => {
    (global.fetch as jest.Mock).mockRejectedValue(new Error('Fail'));

    analytics.track('retry_limit_test', {});
    for (let i = 0; i < 4; i++) {
      await jest.runAllTimersAsync();
    }

    const stored = JSON.parse(localStorage.getItem('analytics_queue')!);
    expect(stored.length).toBe(0);
  });
});
```