НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут телекомунікаційних систем
Кафедра інформаційних технологій в телекомунікаціях

«До захисту допущено»
В.о. завідувача кафедрою
_____ Валерій ПРАВИЛО
«____» _____ 2026 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційно-комунікаційні технології»
спеціальності 172 Телекомунікації та радіотехніка

на тему: Методи та засоби реалізації web-платформи для стримінгу музичного контенту з використанням технологій gRPC та SignalR

Виконав: здобувач вищої освіти 4 курсу, групи ТІ-21
Власенко Роман Сергійович _____
(прізвище, ім’я, по батькові) (підпис)
Науковий керівник: доцент кафедри ІТТ НН ІТС,
кандидат технічних наук, доцент
Суліма Світлана Валеріївна _____
Рецензент: доцент кафедри ТК НН ІТС, кандидат технічних наук, доцент
Міночкін Дмитро Анатолійович _____

(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Здобувач вищої освіти _____

Київ 2026 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут телекомунікаційних систем
Кафедра інформаційних технологій в телекомунікаціях

Рівень вищої освіти – перший (бакалаврський)
Освітньо-професійна програма «Інформаційно-комунікаційні технології»
спеціальності 172 Телекомунікації та радіотехніка

ЗАТВЕРДЖУЮ
В.о. завідувача кафедрою
_____ Валерій ПРАВИЛО
«____» _____ 2025 р.

ЗАВДАННЯ
на дипломну роботу студенту

Власенку Роману Сергійовичу

1. Тема дипломної роботи: «Методи та засоби реалізації web-платформи для стримінгу музичного контенту з використанням технологій gRPC та SignalR», науковий керівник роботи Суліма Світлана Валеріївна, доцент кафедри ІТТ НН ІТС, кандидат технічних наук, доцент, затверджені наказом по університету від 26.05.2026 р. № 1912-с

2. Строк подання студентом дипломної роботи: 01.06.2026 р.

3. Об’єкт дослідження: процеси передачі мультимедійного контенту в розподілених інформаційно-телекомунікаційних системах на основі мікросервісної архітектури.

4. Предмет дослідження: Методи та засоби побудови архітектури вебплатформи стримінгу з використанням технологій gRPC та SignalR для оптимізації міжсервісної комунікації.

5. Вихідні дані до роботи: методи та засоби побудови розподіленої вебплатформи стримінгу аудіоконтенту із застосуванням протоколів gRPC і SignalR; цільові показники — зменшення затримки міжсервісних викликів та обсягу мережевого трафіку порівняно з підходом REST/JSON.

6. Перелік завдань, які потрібно розробити:

1) проаналізувати сучасні системи потокової передачі музичного контенту та архітектурні підходи до їх побудови;
2) дослідити протоколи передачі даних у розподілених телекомунікаційних системах (REST/JSON, gRPC, SignalR/WebSocket) та механізми транспорту HTTP/2;
3) визначити показники й критерії оцінювання ефективності протоколів міжсервісної та real-time комунікації;
4) розробити архітектуру розподіленої вебплатформи стримінгу аудіоконтенту на основі мікросервісів;
5) спроєктувати та реалізувати міжсервісну взаємодію засобами gRPC і real-time підсистему засобами SignalR;
6) розробити методику експериментального дослідження ефективності застосованих протоколів;
7) провести експериментальне дослідження продуктивності та проаналізувати отримані результати.

7. Перелік ілюстративного матеріалу: рисунки, таблиці, презентаційні матеріали до захисту.

8. Дата видачі завдання: 02.10.2025 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів	Примітка
1	Аналіз предметної області, опрацювання методичних і теоретичних джерел	жовтень – листопад 2025	Виконано
2	Робота над першим розділом	грудень 2025	Виконано
3	Робота над другим розділом	грудень 2025 – січень 2026	Виконано
4	Проєктування архітектури вебплатформи	січень – лютий 2026	Виконано
5	Програмна реалізація системи (третій розділ)	лютий – квітень 2026	Виконано
6	Експериментальне дослідження (четвертий розділ)	квітень 2026	Виконано
7	Формулювання висновків	квітень – травень 2026	Виконано
8	Оформлення дипломної роботи	травень 2026	Виконано
9	Підготовка презентації до захисту	травень – червень 2026	Виконано

Здобувач вищої освіти _____ Роман ВЛАСЕНКО

Науковий керівник _____ Світлана СУЛІМА

РЕФЕРАТ

Дипломна робота містить 62 сторінок, 11 рисунки, 11 таблиць та використовує 38 літературне джерело посилань.

Актуальність теми. Стрімке зростання обсягів аудіоконтенту, що передається через інтернет, та перехід індустрії музичного стримінгу до розподілених систем висувають підвищені вимоги до ефективності міжсервісної та real-time комунікації. Традиційний підхід на основі REST/JSON через текстову серіалізацію та обмеження протоколу HTTP/1.1 не завжди забезпечує потрібну продуктивність у високонавантажених мікросервісних системах. Тому дослідження методів і засобів побудови вебплатформи стримінгу на основі сучасних протоколів gRPC та SignalR є актуальним завданням у галузі телекомунікацій.

Мета роботи — Підвищення ефективності міжсервісної комунікації у вебплатформі стримінгу музичного контенту шляхом розробки методів та засобів побудови архітектури з використанням технологій gRPC та SignalR, що забезпечує зменшення обсягу переданих даних та затримок при передачі порівняно з REST/JSON.

Об’єкт дослідження — процеси передачі мультимедійного контенту в розподілених інформаційно-телекомунікаційних системах на основі мікросервісної архітектури.

Предмет дослідження — протоколи міжсервісної комунікації gRPC та SignalR (WebSocket), методи бінарної серіалізації даних, механізми мультиплексування HTTP/2 та їх вплив на продуктивність розподілених систем.

Методи дослідження — системний аналіз науково-технічних джерел, порівняльний аналіз протоколів передачі даних, архітектурне проєктування, програмне моделювання та експериментальне дослідження продуктивності розробленої системи.

Отримані результати. Розроблено архітектуру розподіленої вебплатформи стримінгу музичного контенту «Sonar» з міжсервісною

взаємодією на основі gRPC та real-time підсистемою на основі SignalR. Обґрунтовано вибір протоколів передачі даних, спроєктовано та реалізовано програмні компоненти системи, розроблено методику експериментального дослідження ефективності застосованих протоколів порівняно з підходом REST/JSON.

Ключові слова: СТРИМІНГ АУДІОКОНТЕНТУ, МІКРОСЕРВІСНА АРХІТЕКТУРА, gRPC, SIGNALR, HTTP/2, PROTOCOL BUFFERS, WEBSOCKET, БІНАРНА СЕРІАЛІЗАЦІЯ, REAL-TIME КОМУНІКАЦІЯ.

ABSTRACT

The bachelor’s thesis contains 61 pages, 11 figures, 11 tables and references 38 sources.

Relevance of the topic. The rapid growth of audio content transmitted over the internet and the transition of the music streaming industry towards distributed systems place increased demands on the efficiency of inter-service and real-time communication. The traditional REST/JSON approach, due to text-based serialization and the limitations of the HTTP/1.1 protocol, does not always provide the required performance in high-load microservice systems. Therefore, the study of methods and means of building a streaming web platform based on modern gRPC and SignalR protocols is a relevant task in the field of telecommunications.

The objective of this work is to investigate the efficiency of applying the gRPC (HTTP/2, Protocol Buffers) and SignalR (WebSocket) protocols for inter-service and real-time communication in a distributed audio content streaming system compared to the traditional REST/JSON approach.

The object of the study is the processes of multimedia content transmission in distributed information and telecommunication systems based on a microservice architecture.

The subject of the study is the gRPC and SignalR (WebSocket) inter-service communication protocols, binary data serialization methods, HTTP/2 multiplexing mechanisms and their impact on the performance of distributed systems.

The research methods include systematic analysis of scientific and technical sources, comparative analysis of data transfer protocols, architectural design, software modeling and experimental study of the performance of the developed system.

Results obtained. An architecture of the distributed music content streaming web platform “Sonar” has been developed, with inter-service interaction based on gRPC and a real-time subsystem based on SignalR. The choice of data transfer protocols has been substantiated, the software components of the system have been designed and implemented, and a methodology for the experimental study of the efficiency of the applied protocols compared to the REST/JSON approach has been developed.

Keywords: AUDIO CONTENT STREAMING, MICROSERVICE ARCHITECTURE, gRPC, SIGNALR, HTTP/2, PROTOCOL BUFFERS, WEBSOCKET, BINARY SERIALIZATION, REAL-TIME COMMUNICATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API	Application Programming Interface (інтерфейс прикладного програмування)
CDN	Content Delivery Network (мережа доставки контенту)
CI/CD	Continuous Integration / Continuous Delivery (безперервна інтеграція та доставка)
CRUD	Create, Read, Update, Delete (створення, читання, оновлення, видалення)
DTO	Data Transfer Object (об’єкт передачі даних)
gRPC	gRPC Remote Procedure Calls (фреймворк віддаленого виклику процедур)
HLS	HTTP Live Streaming (потокова передача через HTTP)
HPACK	Header Compression for HTTP/2 (алгоритм стиснення заголовків HTTP/2)
HTTP	HyperText Transfer Protocol (протокол передавання гіпертексту)
IDL	Interface Definition Language (мова опису інтерфейсів)
JSON	JavaScript Object Notation (нотація об’єктів JavaScript)
JWT	JSON Web Token (веб-токен у форматі JSON)
ORM	Object-Relational Mapping (об’єктно-реляційне відображення)
Protobuf	Protocol Buffers (бінарний формат серіалізації даних)
REST	Representational State Transfer (передача репрезентативного стану)
RPC	Remote Procedure Call (віддалений виклик процедур)
RTT	Round-Trip Time (час проходження сигналу в обидва боки)
SPA	Single Page Application (односторінковий застосунок)
TCP	Transmission Control Protocol (протокол керування передаванням)
TLS	Transport Layer Security (протокол захисту транспортного рівня)
UI	User Interface (інтерфейс користувача)
WebSocket	повнодуплексний протокол обміну даними поверх TCP

ВСТУП

Музичний стримінг давно витіснив фізичні носії. Сьогодні такі платформи генерують значну частку мультимедійного трафіку глобальних мереж, а їхня аудиторія обчислюється сотнями мільйонів. При такому масштабі до платформ висуваються жорсткі вимоги: широка пропускна здатність, низька затримка та можливість нарощувати потужність горизонтально.

Щоб виконати ці вимоги, сучасні стримінгові системи будують на мікросервісній архітектурі: система ділиться на незалежні сервіси, які спілкуються мережею. У такій схемі швидкість міжсервісної комунікації прямо впливає на продуктивність усієї платформи. До того ж, сучасні музичні сервіси давно вийшли за межі простого відтворення: спільне прослуховування, чати та сповіщення в реальному часі вимагають повнодуплексного обміну даними.

Традиційний вибір для міжсервісної взаємодії — REST із JSON поверх HTTP/1.1. Цей підхід простий і добре підтримується, але має конкретні недоліки: текстовий JSON важкий порівняно з бінарними форматами, HTTP/1.1 не мультиплексує запити в межах одного з'єднання, а потокова передача та двосторонній обмін реалізуються складно. У системах із тисячами міжсервісних викликів на секунду ці обмеження стають відчутними.

gRPC та SignalR пропонують інший підхід. gRPC працює поверх HTTP/2 із бінарною серіалізацією Protocol Buffers: мультиплексування запитів, стиснення заголовків і компактне подання даних входять у протокол. SignalR через WebSocket дає повнодуплексний канал із низькою затримкою. Наскільки ці технології реально вигідніші за REST у стримінговій платформі — питання, яке потребує вимірювань, а не припущень.

Мета роботи — Підвищення ефективності міжсервісної комунікації у вебплатформі стримінгу музичного контенту шляхом розробки методів та засобів побудови архітектури з використанням технологій gRPC та SignalR, що забезпечує зменшення обсягу переданих даних та затримок при передачі порівняно з REST/JSON.

Для досягнення поставленої мети необхідно вирішити такі задачі:

1) проаналізувати сучасні системи потокової передачі музичного контенту та архітектурні підходи до їх побудови;
2) дослідити протоколи передачі даних у розподілених телекомунікаційних системах (REST/JSON, gRPC, SignalR/WebSocket) та механізми транспорту HTTP/2;
3) визначити показники й критерії оцінювання ефективності протоколів міжсервісної та real-time комунікації;
4) розробити архітектуру розподіленої вебплатформи стримінгу аудіоконтенту на основі мікросервісів;
5) спроєктувати та реалізувати міжсервісну взаємодію засобами gRPC і real-time підсистему засобами SignalR;
6) розробити методику експериментального дослідження ефективності застосованих протоколів та проаналізувати отримані результати.

Об’єкт дослідження — процеси передачі мультимедійного контенту в розподілених інформаційно-телекомунікаційних системах на основі мікросервісної архітектури.

Предмет дослідження — протоколи міжсервісної комунікації gRPC та SignalR (WebSocket), методи бінарної серіалізації даних, механізми мультиплексування HTTP/2 та їх вплив на продуктивність розподілених систем.

Методи дослідження. У роботі використано системний аналіз науково-технічних джерел, порівняльний аналіз протоколів, методи архітектурного проєктування, програмну реалізацію компонентів та експериментальне вимірювання продуктивності.

Практична цінність роботи двояка. По-перше, архітектура та код платформи «Sonar» можуть слугувати основою для подібних розподілених систем. По-друге, результати порівняння протоколів дають конкретні дані для вибору засобів комунікації при проєктуванні високонавантажених телекомунікаційних систем.

Структура роботи. Дипломна робота складається зі вступу, чотирьох розділів, загальних висновків, списку використаних джерел і додатків. У першому розділі проаналізовано предметну область та архітектурні підходи до побудови систем стримінгу. У другому розділі досліджено протоколи передачі даних у розподілених системах. У третьому розділі описано проєктування та програмну реалізацію вебплатформи «Sonar». У четвертому розділі розроблено методику та подано результати експериментального дослідження ефективності протоколів.

РОЗДІЛ 1

АНАЛІЗ СИСТЕМ СТРИМІНГУ МУЗИЧНОГО КОНТЕНТУ ТА ПІДХОДІВ ДО ЇХ ПОБУДОВИ

1.1 Потокова передача музичного контенту як об’єкт телекомунікаційних систем

Потокова передача (streaming) — це безперервна доставка мультимедійного контенту від сервера до клієнта, за якої відтворення починається ще до того, як файл повністю передано. На відміну від завантаження, коли користувач чекає на повний файл перш ніж запустити відтворення, стримінг мінімізує час до першого звуку і не вимагає зберігати копію контенту на пристрої [1].

Музичний стримінг — окремий клас таких систем, орієнтований на аудіоконтент. З технічного погляду він належить до розподілених інформаційно-телекомунікаційних систем: поєднує засоби зберігання даних, мережевого транспортування, кодування та декодування сигналу й управління сеансами між вузлами. Саме тому методи побудови музичних платформ безпосередньо стосуються предметної області телекомунікацій.

Доставка аудіоконтенту в сучасних платформах відбувається поетапно. Спочатку вихідний файл кодується в один або кілька форматів стиснення (AAC, MP3, Opus) і ділиться на невеликі сегменти фіксованої тривалості. Сегменти розміщуються на серверах зберігання або в CDN. Під час відтворення клієнт послідовно запитує сегменти, буферизує їх і передає декодеру. Цей принцип покладено в основу протоколів адаптивного стримінгу — HLS та MPEG-DASH [2].

Адаптивний бітрейт (adaptive bitrate streaming) означає, що один і той самий фрагмент готується у кількох варіантах якості, а клієнт під час відтворення обирає варіант залежно від поточної пропускної здатності каналу.

Це дозволяє зберегти безперервне відтворення навіть при нестабільному з'єднанні — особливо важливо для мобільних користувачів [2].

Основні показники якості систем музичного стримінгу: затримка початку відтворення (startup delay), кількість повторних буферизацій (rebuffering events), пропускна здатність, необхідна для безперебійної доставки, та навантаження на серверну інфраструктуру. Останній показник безпосередньо залежить від архітектури серверної частини платформи, яку розглянуто далі.

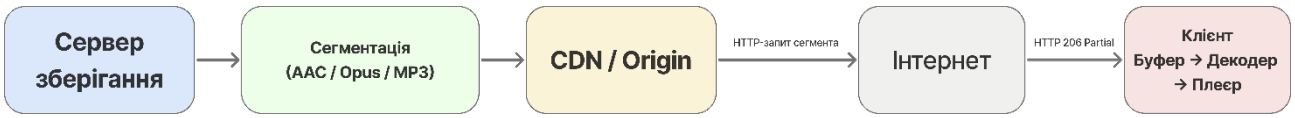

Рисунок 1.1 — Узагальнена схема потокової передачі аудіоконтенту

Окрім доставки контенту, сучасні музичні платформи реалізують пошук і каталогізацію треків, персональні бібліотеки та плейлисти, рекомендації, а також соціальні сценарії. Поєднання потокової передачі великих обсягів із інтерактивними функціями реального часу ускладнює архітектуру таких систем і підвищує вимоги до внутрішньої комунікації між компонентами.

1.2 Архітектурні підходи до побудови вебплатформ стримінгу

Архітектура програмного забезпечення визначає, як система ділиться на частини, як вони взаємодіють і хто за що відповідає. Для платформ стримінгу з високим навантаженням і потребою в постійному розвитку функціональності це питання не теоретичне. Склалося два основні підходи до побудови серверної частини — монолітний та мікросервісний [3].

1.2.1 Монолітна архітектура

Монолітна архітектура — це застосунок у вигляді єдиного процесу, де обробка запитів, бізнес-логіка, доступ до даних та автентифікація зібрані разом і розгортаються як ціле. Компоненти спілкуються через виклики методів у спільній пам'яті, без мережі [3].

Переваги такого підходу найбільш відчутні на старті: простота розробки, єдине середовище для налагодження й тестування, відсутність накладних витрат на мережеву взаємодію. Для невеликих систем з обмеженою функціональністю — це цілком розумний вибір.

Але зі зростанням системи моноліт починає заважати. Будь-яка зміна, навіть дрібна, вимагає перезбірки та повторного розгортання всього застосунку. Масштабувати можна лише систему цілком — виділити ресурси окремому навантаженому компоненту не вийде без дублювання всього іншого. Тісний зв'язок між модулями ускладнює підтримку і підвищує ризик ланцюгових збоїв. Для стримінгової платформи, де каталог, відтворення, рекомендації та соціальні функції мають принципово різний характер навантаження, це вже не незручність, а проблема [3; 4].

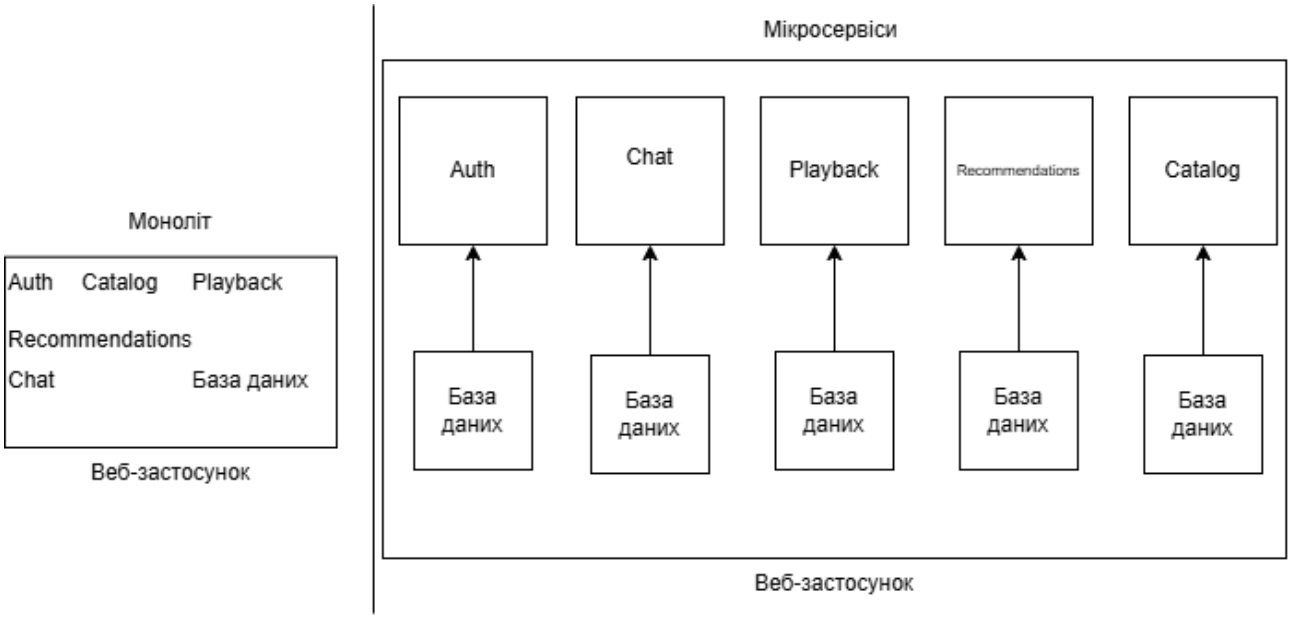

Рисунок 1.2 — Узагальнена структура монолітної та мікросервісної архітектур

1.2.2 Мікросервісна архітектура

Мікросервісна архітектура ділить систему на невеликі автономні сервіси, кожен з яких відповідає за одну функціональну область, має власну базу даних і розгортається незалежно. Сервіси спілкуються тільки через мережу, через чітко визначені API [4].

Головні переваги: кожен сервіс масштабується окремо; команди можуть використовувати різні технології для різних частин системи; збій одного сервісу не валить усе інше; нову версію компонента можна випустити, не зупиняючи платформу. Для стримінгового сервісу це означає, наприклад, що аналітика масштабується окремо від відтворення, а оновлення рекомендацій не потребує зупинки всього [4].

Разом з тим мікросервісна архітектура додає нові складності. Те, що в моноліті було викликом методу, тут стає мережевим запитом із затримкою, ймовірністю відмови й потребою серіалізувати дані. У високонавантаженій системі таких запитів тисячі на секунду — і ефективність протоколу міжсервісної взаємодії прямо визначає продуктивність усієї платформи. Саме тому вибір протоколів передачі даних є центральним предметом цієї роботи.

Таблиця 1.1 — Порівняння монолітної та мікросервісної архітектур

Характеристика	**Монолітна архітектура**	**Мікросервісна архітектура**
Масштабування	Лише цілісне, дублювання всієї системи	Незалежне для кожного сервісу
Розгортання	Повне перескладання застосунку	Незалежне для кожного сервісу
Відмовостійкість	Збій модуля впливає на всю систему	Локалізація відмов у межах сервісу
Технологічний стек	Єдиний для всієї системи	Окремий для кожного сервісу
Міжкомпонентна взаємодія	Виклики методів у спільному процесі	Мережеві виклики через API
Складність	Зростає з розміром	Розподілена між

Характеристика	Монолітна архітектура	Мікросервісна архітектура
супроводу	системи	сервісами

1.3 Огляд протоколів комунікації у розподілених системах

У мікросервісній системі є два принципово різні типи взаємодії. Перший — міжсервісна комунікація (service-to-service): сервіси обмінюються даними у внутрішній мережі для виконання спільних операцій. Другий — комунікація реального часу між сервером і клієнтом: сервер миттєво сповіщає користувача про події. Кожен тип має свої вимоги до протоколів [5].

1.3.1 Архітектурний стиль REST та формат JSON

REST — найпоширеніший стиль побудови вебінтерфейсів. Взаємодія будується на HTTP: ресурси ідентифікуються URL-адресами, дії виражаються стандартними методами (GET, POST, PUT, DELETE), дані передаються у текстовому JSON [5].

Переваги REST/JSON — простота, читабельність, незалежність від мови програмування та широка інструментальна підтримка. Саме тому він залишається стандартом для зовнішніх API.

Але для інтенсивної внутрішньої міжсервісної взаємодії він має конкретні слабкості. JSON надлишковий: назви полів повторюються в кожному повідомленні, числа передаються як рядки символів. HTTP/1.1 не мультиплексує запити — в межах одного з'єднання вони обробляються послідовно, що призводить до блокування початку черги (head-of-line blocking). Потокова передача та двосторонній обмін даними реалізуються складно [6].

1.3.2 Протокол gRPC

gRPC — сучасний фреймворк віддаленого виклику процедур для ефективної міжсервісної комунікації. Він побудований на HTTP/2 і використовує бінарну серіалізацію Protocol Buffers. Замість оперування ресурсами, як у REST, gRPC дозволяє викликати методи віддаленого сервісу так само, як локальні функції [7].

Ключові переваги: компактне бінарне подання, мультиплексування викликів у межах одного з'єднання, стиснення заголовків, підтримка потокових режимів. Інтерфейс сервісу описується мовою IDL, з якої автоматично генерується код клієнта та сервера. Детальний аналіз gRPC наведено в розділі 2.

1.3.3 Технологія SignalR та протокол WebSocket

Для комунікації реального часу між сервером і клієнтом використовують WebSocket — протокол, який, на відміну від HTTP, відкриває повнодуплексний канал: після встановлення з'єднання обидві сторони можуть надсилати дані в будь-який момент без повторних запитів [8].

SignalR — бібліотека для .NET, що спрощує реалізацію функціональності реального часу. Вона абстрагує транспортний рівень і автоматично обирає найкращий доступний механізм зв'язку (насамперед WebSocket). Модель взаємодії будується на «хабах»: сервер може напряму викликати методи на підключених клієнтах. Це зручно для чатів, сповіщень та спільного прослуховування [8]. Принципи роботи SignalR розглянуто в розділі 2.

1.4 Аналіз існуючих платформ музичного стримінгу

Щоб розуміти типові архітектурні рішення та функціональні вимоги до систем стримінгу, варто подивитися на те, що вже побудовано.

Spotify — одна з найбільших світових платформ. Серверна частина побудована на мікросервісній архітектурі з великою кількістю незалежних сервісів. Spotify раніше використовувала власний протокол для внутрішньої комунікації, а потім перейшла на gRPC — що прямо підтверджує доцільність

цього вибору для високонавантажених систем. Платформа реалізує адаптивний бітрейт, рекомендаційні механізми та соціальні функції [9].

Apple Music забезпечує потокову передачу, зокрема у форматах високої роздільної здатності. Сервіс тісно інтегрований з екосистемою пристроїв Apple і використовує власну інфраструктуру доставки контенту.

YouTube Music використовує відеоінфраструктуру материнської компанії. Особливість — поєднання аудіо- та відеоконтенту в межах єдиної платформи.

SoundCloud орієнтована на незалежних виконавців і поєднує стримінг із розвиненою соціальною складовою: коментарями до треків, підписками, взаємодією між користувачами. Це робить її показовим прикладом інтеграції потокової передачі з функціональністю реального часу.

Таблиця 1.2 — Порівняльна характеристика платформ музичного стримінгу

Платформа	**Тип архітектури**	**Адаптивний бітрейт**	**Соціальні функції**
Spotify	Мікросервісна	Так	Розвинені
Apple Music	Мікросервісна	Так	Обмежені
YouTube Music	Мікросервісна	Так	Помірні
SoundCloud	Мікросервісна	Так	Розвинені

Всі провідні платформи будуються на мікросервісній архітектурі, реалізують адаптивний стримінг і дедалі ширше поєднують його з функціями реального часу. Тенденція до gRPC для внутрішньої комунікації підтверджується. Водночас детальні дані про внутрішні протоколи взаємодії цих систем закриті — що і зумовлює потребу в самостійному дослідженні на прикладі власної реалізації.

1.5 Постановка задачі дослідження

Сучасні вебплатформи музичного стримінгу будуються на мікросервісній архітектурі. Ефективність міжсервісної та real-time комунікації безпосередньо визначає продуктивність такої системи. REST/JSON має відомі обмеження при інтенсивній внутрішній взаємодії; gRPC та SignalR теоретично їх усувають. Але «теоретично» — не достатньо. Потрібні вимірювання.

Задача роботи: дослідити ефективність gRPC та SignalR для міжсервісної та real-time комунікації у розподіленій системі стримінгу аудіоконтенту порівняно з REST/JSON шляхом проєктування, реалізації та експериментального дослідження вебплатформи.

Для цього потрібно:

1. дослідити теоретичні засади gRPC та SignalR, механізми HTTP/2 і WebSocket, методи бінарної серіалізації;
2. визначити метрики оцінювання ефективності протоколів;
3. розробити архітектуру розподіленої вебплатформи стримінгу та спроєктувати її компоненти;
4. реалізувати міжсервісну взаємодію через gRPC і real-time підсистему через SignalR;
5. провести порівняльний експеримент і проаналізувати результати.

Висновки за розділом 1

1. Платформи музичного стримінгу належать до розподілених інформаційно-телекомунікаційних систем. Сегментний підхід та адаптивний бітрейт є основою сучасних технологій потокової передачі аудіоконтенту.
2. Мікросервісна архітектура краща за монолітну для високонавантажених стримінгових систем завдяки незалежному масштабуванню, локалізації відмов і технологічній гнучкості. Платою за це є ускладнення міжсервісної комунікації.

3. REST/JSON має конкретні обмеження при інтенсивній внутрішній взаємодії. gRPC та SignalR адресують ці обмеження за рахунок HTTP/2, бінарної серіалізації та WebSocket.
4. Провідні платформи підтверджують цю тенденцію на практиці: всі побудовані на мікросервісах, всі поєднують стримінг з функціями реального часу, Spotify перейшла на gRPC для внутрішньої комунікації.
5. Детальні дані про протоколи реальних платформ закриті. Кількісне порівняння gRPC та SignalR відносно REST/JSON потребує власного дослідження — що і є задачею цієї роботи.

РОЗДІЛ 2

ПРОТОКОЛИ ПЕРЕДАЧІ ДАНИХ У РОЗПОДІЛЕНИХ ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМАХ

2.1 Транспортний протокол HTTP/2 та його механізми

HTTP є основою взаємодії в сучасних вебсистемах. Версія HTTP/1.1 тривалий час залишалася стандартом, але має обмеження, які у високонавантажених розподілених системах стають відчутними. Протокол HTTP/2, стандартизований у RFC 7540, усуває ці обмеження. Саме HTTP/2 є транспортною основою gRPC, тому розуміння його механізмів потрібне для оцінювання ефективності міжсервісної комунікації [10] [14].

Головна проблема HTTP/1.1 проста: у межах одного TCP-з'єднання запити обробляються строго послідовно. Наступний запит не вирушає, поки не прийде відповідь на попередній. Це явище називають блокуванням початку черги (head-of-line blocking): один повільний запит затримує всі інші. Браузери частково обходять це, відкриваючи кілька паралельних з'єднань, але це збільшує навантаження на сервер і мережу. Додатково HTTP/1.1 передає заголовки у текстовому вигляді без стиснення — при великій кількості запитів це суттєвий зайвий трафік [10].

HTTP/2 вирішує обидві проблеми через кілька фундаментальних механізмів.

2.1.1 Бінарне кадрування

HTTP/2 передає дані не текстом, а бінарними кадрами (frames). Кадри бувають різних типів: заголовків (HEADERS), даних (DATA), службових для керування потоком та з'єднанням. Кожен кадр має фіксований заголовок із типом, довжиною та ідентифікатором потоку, якому він належить [10].

Бінарний формат однозначно розбирається, компактний і дозволяє реалізувати мультиплексування. Сукупність кадрів однієї логічної взаємодії «запит — відповідь» утворює потік (stream).

2.1.2 Мультиплексування потоків

Мультиплексування — головна перевага HTTP/2. У межах одного TCP-з'єднання одночасно існує багато незалежних логічних потоків, кадри яких передаються вперемішку. Приймальна сторона збирає повідомлення за ідентифікатором потоку в кожному кадрі [10].

Кілька запитів та відповідей обробляються паралельно через єдине з'єднання — блокування черги на рівні застосунку зникає, і відкривати десятки паралельних з'єднань більше не потрібно. Для мікросервісної системи це означає, що один сервіс підтримує єдине постійне з'єднання з іншим і передає через нього всі виклики, суттєво знижуючи накладні витрати.

Рисунок 2.1 — Мультиплексування потоків у межах одного з’єднання HTTP/2

2.1.3 Стиснення заголовків HPACK

HTTP/2 стискає заголовки за алгоритмом HPACK (RFC 7541). Алгоритм використовує статичну та динамічну таблиці часто вживаних заголовків і кодування Гафмана. Замість повного тексту заголовків у кожному повідомленні передаються посилання на елементи таблиці або лише відмінності від попередніх значень [10][13].

У міжсервісній взаємодії значна частина заголовків (адреса, тип контенту, службові поля) повторюється від запиту до запиту. HPACK зменшує обсяг цього службового трафіку — особливо помітно при великій кількості дрібних викликів.

2.1.4 Пріоритизація та керування потоком

HTTP/2 дозволяє вказувати відносну важливість потоків і залежності між ними. Приймальна сторона використовує цю інформацію для розподілу ресурсів обробки. Механізм керування потоком (flow control) обмежує обсяг даних, які можна надіслати до отримання підтвердження, і таким чином запобігає перевантаженню. Керування діє на рівні окремих потоків і на рівні з'єднання загалом [10].

2.2 Протокол gRPC та формат серіалізації Protocol Buffers

gRPC — відкритий фреймворк віддаленого виклику процедур, розроблений Google і переданий до Cloud Native Computing Foundation. Назва — рекурсивний акронім від «gRPC Remote Procedure Calls». Призначений для комунікації між сервісами у розподілених системах [7].

Модель gRPC проста: клієнт викликає метод віддаленого сервісу так само, як локальну функцію. Серіалізація аргументів, мережева передача, виклик методу на сервері та повернення результату — все це бере на себе фреймворк. Це відрізняється від ресурсно-орієнтованого REST і робить взаємодію між сервісами прямолінійнішою з погляду розробника.

2.2.1 Мова опису інтерфейсів та формат Protocol Buffers

Інтерфейс сервісу gRPC описується у файлі .proto мовою Protocol Buffers IDL. У ньому визначаються структури повідомлень (message) та сервіси (service) із переліком методів, їхніх вхідних і вихідних типів. Кожне поле повідомлення має ім'я, тип та унікальний числовий тег — саме тег використовується для ідентифікації поля у бінарному поданні [11].

З .proto-файлу компілятор автоматично генерує класи повідомлень і заготовки клієнта та сервера для обраної мови програмування. Це дає сувору типізацію взаємодії, узгоджений контракт між сервісами та незалежність від мови реалізації.

Protocol Buffers — бінарний формат серіалізації. JSON передає і назви полів, і значення символами; Protocol Buffers передає лише числовий тег поля та значення у компактному бінарному вигляді. Для цілих чисел використовується кодування змінної довжини (varint): малі значення займають менше байтів. У результаті серіалізоване повідомлення у Protobuf зазвичай у кілька разів компактніше за еквівалентний JSON [11].

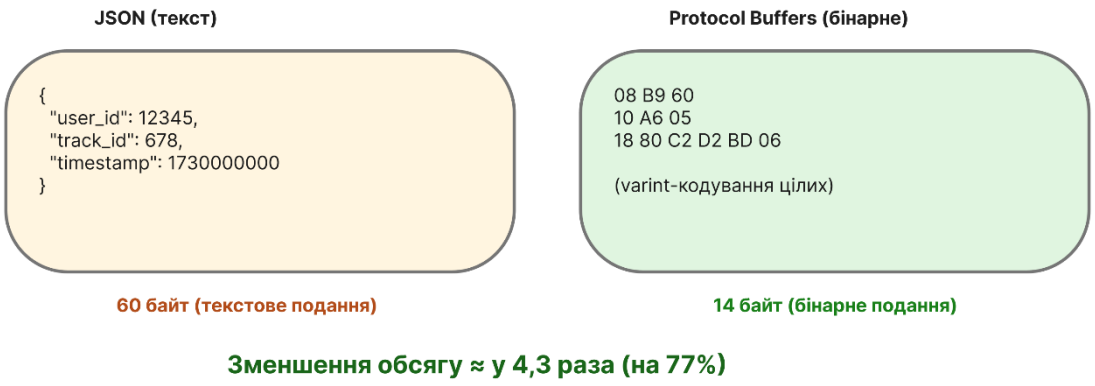

Рисунок 2.2 — Порівняння подання даних у форматах JSON та Protocol Buffers

2.2.2 Режими взаємодії gRPC

gRPC підтримує чотири режими взаємодії::

- **унарний виклик (unary RPC)** — один запит, одна відповідь; класична модель виклику функції;
- **серверний потік (server streaming RPC)** — клієнт надсилає один запит, сервер повертає послідовність повідомлень;
- **клієнтський потік (client streaming RPC)** — клієнт надсилає послідовність повідомлень, сервер повертає одну відповідь;

- **двонапрямлений потік (bidirectional streaming RPC)** — обидві сторони незалежно надсилають послідовності повідомлень в межах одного з'єднання.

Потокові режими реалізуються через мультиплексування HTTP/2 і дозволяють передавати великі обсяги даних або послідовності подій без повторного встановлення з'єднання.

2.2.3 Переваги та обмеження gRPC

Переваги: компактне бінарне подання зменшує трафік; серіалізація та десеріалізація швидкі; мультиплексування HTTP/2; сувора типізація і автоматична генерація коду; потокові режими; незалежність від мови.

Обмеження теж є. Бінарний формат не читається людиною, що ускладнює налагодження. Підтримка gRPC у браузерах обмежена через відсутність прямого доступу до низькорівневих можливостей HTTP/2 — потрібен проміжний шар gRPC-Web. Тому gRPC використовують переважно для внутрішньої міжсервісної взаємодії, а зовнішні інтерфейси часто залишають на REST [7].

2.3 Технологія SignalR та протокол WebSocket

gRPC та REST працюють за моделлю «запит — відповідь». Але сучасні вебплатформи потребують іншого: сервер повинен сам ініціювати передачу даних клієнту без запиту з його боку. Для цього використовують WebSocket і побудовану на ньому технологію SignalR.

2.3.1 Протокол WebSocket

WebSocket (RFC 6455) — повнодуплексний канал поверх єдиного TCP-з'єднання. На відміну від HTTP, де кожна взаємодія починається із запиту клієнта, WebSocket після встановлення з'єднання дозволяє обом сторонам надсилати дані в будь-який момент [12].

З'єднання встановлюється через HTTP-запит з upgrade-заголовками. Сервер підтверджує перехід, і канал переходить у режим WebSocket. Подальший обмін відбувається невеликими кадрами з мінімальними службовими витратами. Відсутність повторного handshake і малий overhead забезпечують низьку затримку доставки повідомлень [12].

2.3.2 Архітектура та принципи роботи SignalR

SignalR — бібліотека для ASP.NET Core для реалізації функціональності реального часу. Вона не замінює WebSocket, а надає над ним абстракцію, яка приховує деталі транспортного рівня [8].

SignalR автоматично обирає транспорт. Пріоритет — WebSocket; якщо він недоступний (через обмеження мережевого обладнання), бібліотека перемикається на Server-Sent Events або long polling. Застосунок продовжує працювати в різних мережевих умовах без змін у коді [8].

Центральне поняття SignalR — хаб (hub). Це серверний компонент, через який сервер і клієнти викликають методи один одного. Виклик можна адресувати конкретному клієнту, групі клієнтів або всім підключеним одразу. Механізм груп дозволяє логічно об'єднувати клієнтів — наприклад, учасників одного чату або сеансу спільного прослуховування [8].

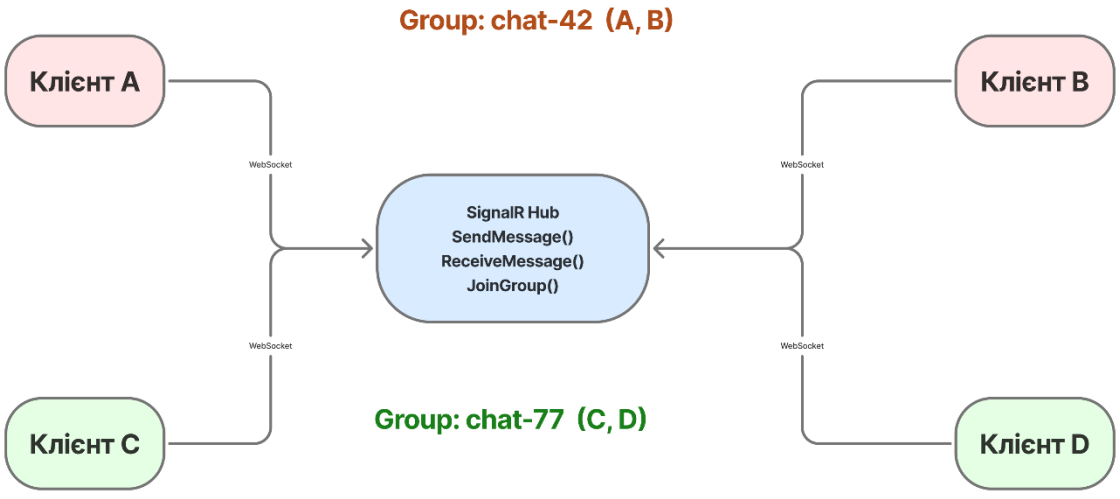

Рисунок 2.3 — Архітектура взаємодії клієнтів із сервером через хаб SignalR

Для подання даних SignalR підтримує JSON та компактніший бінарний формат MessagePack. Це дозволяє додатково зменшити трафік у сценаріях, де обсяг повідомлень має значення.

2.4 Порівняльний аналіз протоколів передачі даних

Три розглянутих протоколи закривають різні задачі. Таблиця 2.1 зводить їхні технічні характеристики разом.

Таблиця 2.1 — Порівняльна характеристика протоколів передачі даних

Ознака	REST/JSON	gRPC	SignalR/WebSocket
Транспортний протокол	HTTP/1.1	HTTP/2	WebSocket поверх TCP
Формат даних	Текстовий (JSON)	Бінарний (Protobuf)	JSON або MessagePack
Модель взаємодії	Запит — відповідь	RPC, потокові режими	Повнодуплексна, реального часу
Мультиплексування	Відсутнє	Підтримується	Не застосовне (один канал)
Стиснення заголовків	Відсутнє	HPACK	Не застосовне
Типізація контракту	Слабка	Сувора (.proto)	Слабка
Зручність налагодження	Висока	Низька (бінарний)	Середня
Типова сфера застосування	Зовнішні API	Міжсервісна взаємодія	Сповіщення, чат

REST/JSON зручний для зовнішніх API через простоту і читабельність. gRPC підходить для інтенсивної внутрішньої міжсервісної взаємодії завдяки бінарній серіалізації та мультиплексуванню. SignalR через WebSocket — правильний вибір для функціональності реального часу. У розробленій вебплатформі всі три використовуються разом, кожен на своєму місці.

2.5 Показники та критерії оцінювання ефективності протоколів

Щоб порівняти протоколи кількісно, потрібно визначити, що саме вимірювати.

Для міжсервісної комунікації:

- **затримка (latency)** — час від надсилання запиту до отримання відповіді, у мілісекундах; аналізуються середнє значення та процентилі;
- **пропускна здатність (throughput)** — кількість успішно оброблених викликів за секунду;
- **обсяг переданих даних** — сумарний розмір корисного навантаження та службових даних на один виклик, у байтах;
- **використання ресурсів** — завантаження CPU та споживання пам'яті під час обміну.

Для real-time комунікації додатково:

- **затримка доставки повідомлення** — час від ініціювання повідомлення сервером до його отримання клієнтом;
- **масштабованість** — здатність підсистеми обслуговувати зростаючу кількість одночасних підключень без суттєвого погіршення інших показників.

Критерій ефективності: один протокол кращий за інший, якщо за однакових умов він дає меншу затримку, менший обсяг переданих даних і більшу пропускну здатність. Ці показники і критерій становлять методичну основу експерименту, детально описаного в розділі 4.

Висновки за розділом 2

1. HTTP/2 усуває обмеження HTTP/1.1 через бінарне кадрування, мультиплексування потоків, стиснення заголовків HPACK та керування потоком. Ці механізми є технічною основою ефективної міжсервісної комунікації через gRPC.
2. gRPC поєднує HTTP/2 з бінарною серіалізацією Protocol Buffers, суворою типізацією через .proto-файли та чотирма режимами взаємодії включно з потоковими. Це робить його підходящим для внутрішньої взаємодії між мікросервісами.
3. WebSocket забезпечує повнодуплексний канал із низькою затримкою. SignalR надає над ним зручну абстракцію з автоматичним вибором транспорту, моделлю хабів і підтримкою груп клієнтів.
4. REST/JSON, gRPC та SignalR оптимальні для різних задач. У розробленій платформі всі три застосовуються разом.
5. Для порівняльного експерименту визначено систему показників: затримка, пропускна здатність, обсяг переданих даних, використання ресурсів, масштабованість.

РОЗДІЛ 3

ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБПЛАТФОРМИ SONAR

3.1 Загальна архітектура вебплатформи

На основі аналізу, проведеного в попередніх розділах, розроблено вебплатформу стримінгу музичного контенту «Sonar». Платформа побудована за мікросервісним підходом, що дозволяє застосувати кожен із досліджуваних протоколів передачі даних там, де він має реальні переваги.

Систему поділено на чотири складові: клієнтський застосунок у вебпереглядачі, основний серверний застосунок із бізнес-логікою, окремий мікросервіс аналітики та реляційне сховище даних. Кожен тип взаємодії між ними реалізований відповідним протоколом.

Звичайні операції між клієнтом і сервером (автентифікація, каталог, бібліотека) — REST/JSON. Real-time функції (чат, сповіщення) — SignalR поверх WebSocket. Внутрішня взаємодія між основним сервером і мікросервісом аналітики — gRPC. Таке розмежування дає змогу порівняти всі три протоколи в межах єдиної робочої системи.

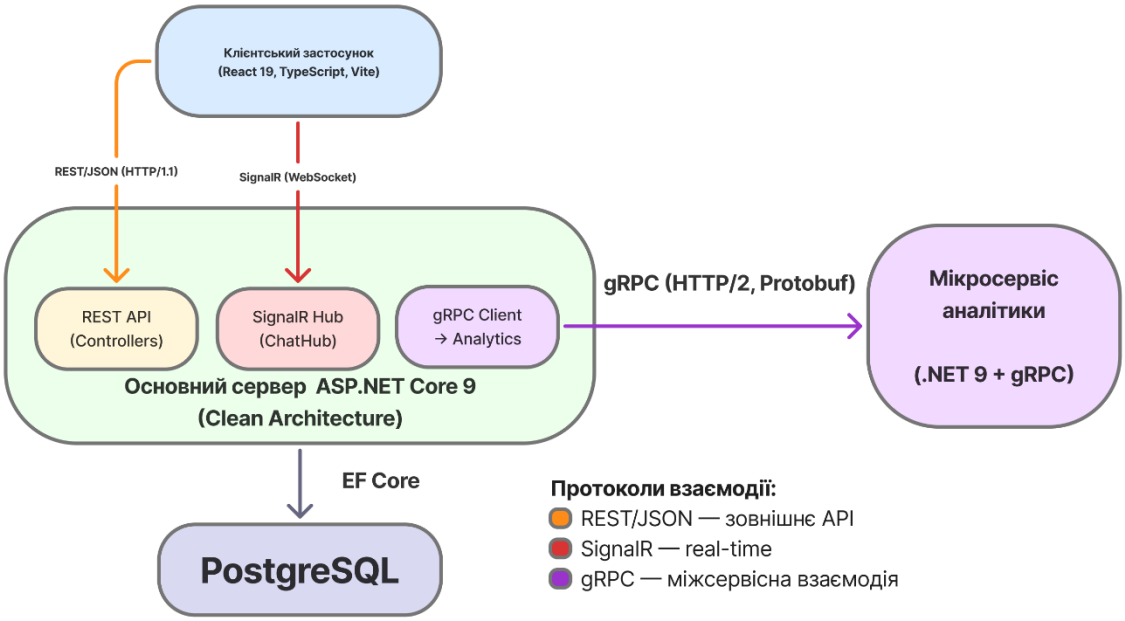

Рисунок 3.1 — Узагальнена архітектура вебплатформи Sonar

3.2 Проєктування серверної частини

Серверну частину реалізовано на платформі .NET мовою C#. Основний сервер побудовано за принципами чистої архітектури (Clean Architecture): система поділена на шари з однонаправленими залежностями — від зовнішніх до внутрішніх.

3.2.1 Шарова організація основного сервера

Основний сервер складається з п'яти шарів:

- **Entities** — доменні моделі (користувач, трек, альбом, плейлист, повідомлення чату); не залежить від жодного зовнішнього шару;
- **Application** — бізнес-логіка, сценарії використання та інтерфейси до зовнішніх ресурсів;
- **Infrastructure** — конкретні реалізації доступу до бази даних і зовнішніх сервісів;

- **Presentation** — контролери REST API, хаби SignalR, обробка вхідних запитів;
- **Logging** — наскрізне протоколювання подій системи.

Внутрішні шари не залежать від зовнішніх; вся взаємодія відбувається через абстрактні інтерфейси. Це дозволяє замінювати конкретні технічні реалізації, не зачіпаючи бізнес-логіку.

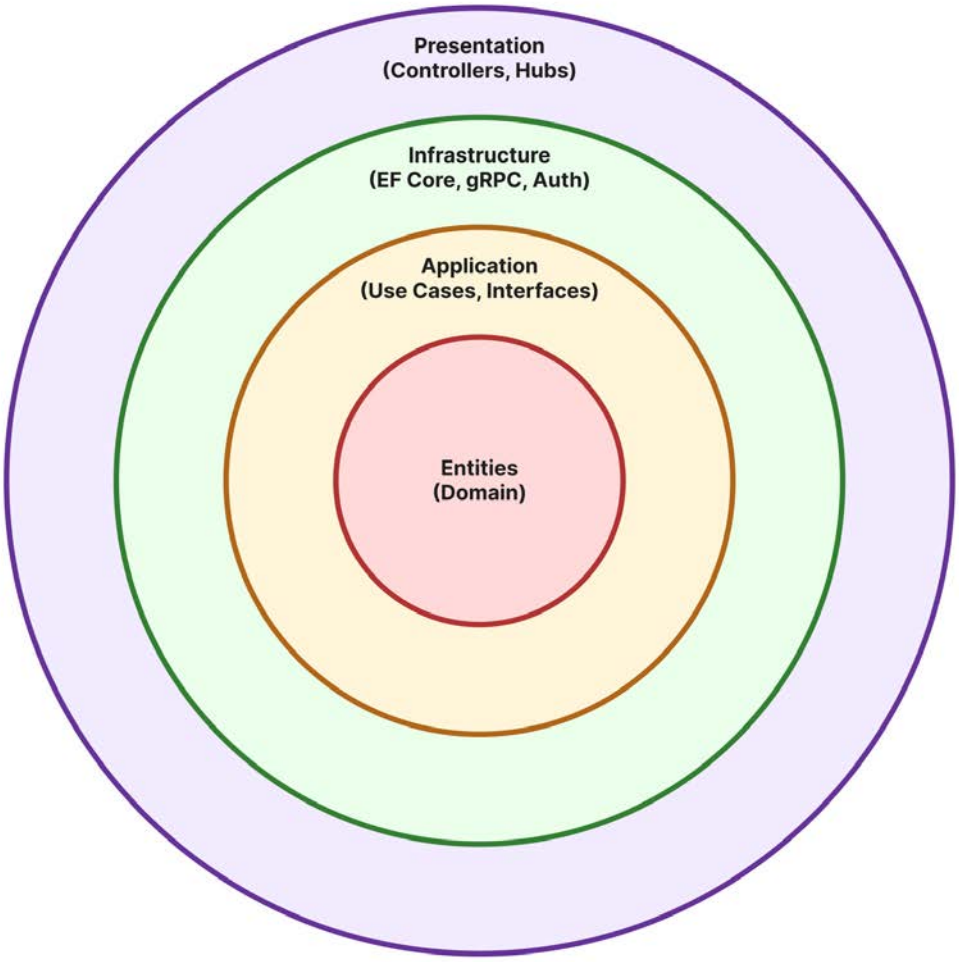

Рисунок 3.2 — Шарова організація основного серверного застосунку за принципами Clean Architecture

3.2.2 Реалізація REST API

Звичайні операції клієнта реалізовано через REST API. Кінцеві точки згруповано за функціональними областями: автентифікація, каталог треків та альбомів, бібліотека користувача, пошук. Контролери шару Presentation делегують виконання сервісам шару Application.

Автентифікацію реалізовано на JWT: після входу сервер видає підписаний токен, який клієнт надсилає в заголовку кожного наступного запиту. Дані між клієнтом і сервером передаються у форматі JSON.

3.2.3 Мікросервіс аналітики

Збирання та обробку статистики (прослуховування треків, популярність контенту, дані виконавців) винесено в окремий мікросервіс. Аналітичне навантаження за характером відрізняється від основних операцій платформи і потребує незалежного масштабування — це і є головне обґрунтування виокремлення.

Мікросервіс реалізовано як окремий .NET-застосунок і взаємодіє з основним сервером виключно через gRPC.

3.3 Реалізація міжсервісної взаємодії засобами gRPC

Взаємодію між основним сервером та мікросервісом аналітики реалізовано через gRPC: бінарна серіалізація та мультиплексування HTTP/2 підходять для інтенсивного внутрішнього обміну між сервісами.

3.3.1 Опис контракту сервісу у форматі Protocol Buffers

Першим кроком є визначення контракту у .proto-файлі. Лістинг 3.1 наводить спрощений фрагмент.

Лістинг 3.1 — Фрагмент опису контракту сервісу аналітики (.proto)

```
syntax = "proto3";


service AnalyticsService {
```

```
  rpc RegisterPlayback (PlaybackEvent) returns (PlaybackAck);
  rpc GetTrackStatistics (TrackRequest) returns (TrackStatistics);
}


message PlaybackEvent {
  int64 user_id   = 1;
  int64 track_id  = 2;
  int64 timestamp = 3;
}


message TrackStatistics {
  int64 track_id     = 1;
  int64 play_count   = 2;
  int64 listeners    = 3;
}
```

Компілятор Protocol Buffers генерує з цього файлу класи повідомлень і заготовки клієнта та сервера для C#. Контракт між сервісами суворо типізований і узгоджений автоматично.

3.3.2 Реалізація серверної та клієнтської частин gRPC

На стороні мікросервісу реалізовано клас, що успадковує згенерований базовий клас сервісу та наповнює методи бізнес-логікою. На стороні основного сервера gRPC-клієнт викликає методи віддаленого сервісу так само, як локальні. Лістинг 3.2 наводить фрагмент такого виклику.

Лістинг 3.2 — Виклик методу мікросервісу аналітики через gRPC-клієнт

```
// реєстрація події прослуховування треку
var request = new PlaybackEvent {
    UserId    = userId,
    TrackId   = trackId,
    Timestamp = DateTimeOffset.UtcNow.ToUnixTimeSeconds()
};


var reply = await _analyticsClient
    .RegisterPlaybackAsync(request);
```

Подія реєструється асинхронно — основний потік обробки запиту не блокується. Постійне HTTP/2-з'єднання з мультиплексуванням дозволяє передавати велику кількість викликів без витрат на повторний handshake.

3.4 Реалізація real-time підсистеми засобами SignalR

Чат між користувачами та сповіщення реалізовано через SignalR: сервер має ініціювати передачу даних клієнту без попереднього запиту з його боку, і WebSocket під цю задачу підходить природно.

3.4.1 Реалізація хаба чату

Центральний компонент — хаб чату, серверний клас, що успадковує базовий клас хаба SignalR. Хаб приймає повідомлення від користувачів, зберігає їх і розсилає отримувачам. Лістинг 3.3 — спрощений фрагмент.

Лістинг 3.3 — Фрагмент реалізації хаба чату на основі SignalR

```
public class ChatHub : Hub
{
   public async Task SendMessage(long chatId, string text)
   {
      var message = await _chatService
         .SaveMessageAsync(chatId, Context.UserIdentifier, text);


      await Clients.Group($"chat-{chatId}")
         .SendAsync("ReceiveMessage", message);
   }
}
```

Учасники чату об'єднані в групу SignalR — повідомлення надходить лише їм. Постійне WebSocket-з'єднання забезпечує мінімальну затримку доставки.

3.4.2 Інтеграція SignalR із клієнтським застосунком

На стороні клієнта взаємодію з хабом реалізовано через клієнтську бібліотеку SignalR. Клієнт встановлює з'єднання, підписується на події та реєструє обробники, які оновлюють інтерфейс у міру надходження даних. Якщо пряме WebSocket-з'єднання недоступне, SignalR автоматично переходить до резервного транспорту.

3.5 Організація потокової передачі аудіоконтенту

Аудіофайли передаються клієнту частинами через механізм HTTP Range Requests. Клієнт запитує не весь файл, а лише потрібний діапазон байтів; сервер відповідає частковим вмістом із відповідним кодом стану. Це дає буферизацію аудіопотоку, перемотування на довільну позицію без завантаження попередніх частин і раціональне використання каналу.

На стороні клієнта відтворенням керує компонент аудіоплеєра: буферизація, черга відтворення, взаємодія з аудіопідсистемою браузера.

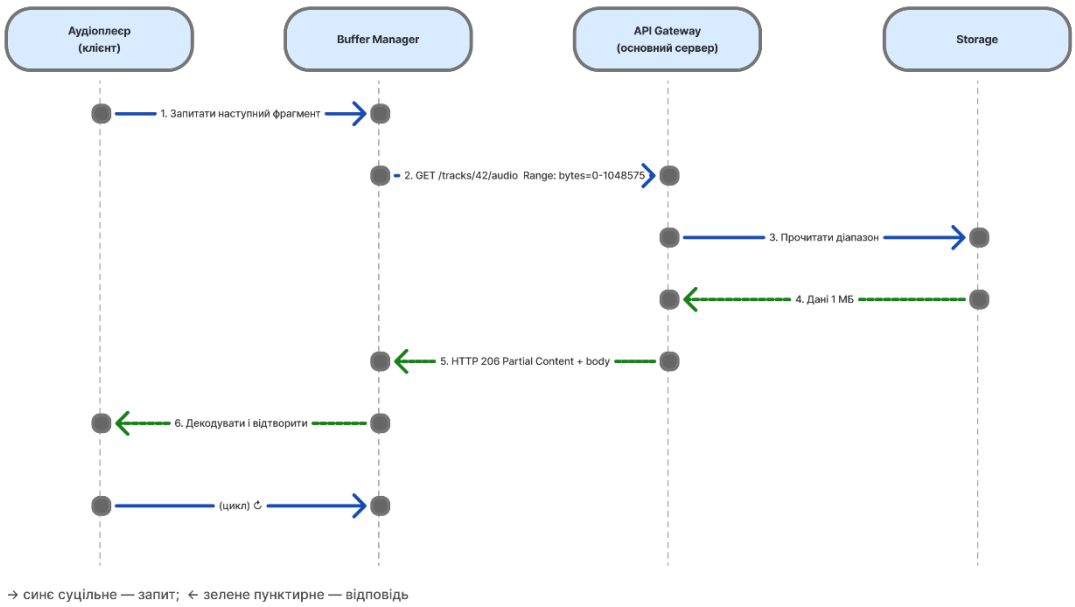

Рисунок 3.3 — Схема організації потокової передачі аудіоконтенту в платформі Sonar

3.6 Реалізація клієнтської частини

Клієнтський застосунок — SPA на React і TypeScript, зібраний інструментом Vite. Структуру кодової бази організовано за методологією Feature-Sliced Design: поділ на шари за рівнем абстракції від спільних засобів до завершених сторінок. Це контролює залежності між частинами застосунку і спрощує його розширення.

Для REST API використовується RTK Query: запити, кешування відповідей, автоматичне оновлення даних. Real-time взаємодія — через клієнтську бібліотеку SignalR. Застосунок покриває основні сценарії: автентифікацію, каталог і пошук, відтворення треків із чергою, особисту бібліотеку та чат.

3.7 Технологічний стек та розгортання системи

Узагальнений перелік технологій, застосованих під час реалізації вебплатформи «Sonar», наведено в таблиці 3.1.

Таблиця 3.1 — Технологічний стек вебплатформи Sonar

Складова системи	Застосовані технології
Основний серверний застосунок	Платформа .NET, мова C#, ASP.NET Core
Мікросервіс аналітики	Платформа .NET, фреймворк gRPC
Доступ до даних	Entity Framework Core (ORM)
Система керування базою даних	PostgreSQL
Real-time взаємодія	SignalR, протокол WebSocket

Складова системи	Застосовані технології
Міжсервісна взаємодія	gRPC, Protocol Buffers, HTTP/2
Клієнтський застосунок	React, TypeScript, Vite
Керування станом клієнта	RTK Query
Розгортання та доставка	Docker, AWS (ECR, EC2), CI/CD

Дані зберігаються в PostgreSQL; доступ реалізований через Entity Framework Core без написання SQL вручну. Розгортання автоматизовано через CI/CD: компоненти пакуються у Docker-контейнери, образи зберігаються в AWS ECR, виконуються на AWS EC2.

Висновки за розділом 3

1. Розроблено архітектуру платформи «Sonar» на основі мікросервісного підходу. Чотири складові системи спілкуються через три різних протоколи залежно від типу взаємодії: REST/JSON, gRPC, SignalR.
2. Основний сервер побудовано за Clean Architecture з п'ятьма шарами; внутрішні шари не залежать від зовнішніх.
3. Міжсервісну взаємодію з мікросервісом аналітики реалізовано через gRPC: контракт описано у .proto-файлі, код клієнта та сервера згенеровано автоматично.
4. Real-time підсистему чату та сповіщень реалізовано через SignalR з механізмом хабів і груп.
5. Потокову передачу аудіо реалізовано через HTTP Range Requests. Клієнтський застосунок побудовано як SPA на React/TypeScript за методологією FSD; розгортання автоматизовано через Docker і AWS CI/CD

РОЗДІЛ 4

ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПРОТОКОЛІВ

У цьому розділі експериментально перевіряється, чи дають gRPC та SignalR реальний виграш у затримці та обсязі переданих даних порівняно з REST/JSON у вебплатформі стримінгу аудіоконтенту. Дослідження спирається на показники та критерій ефективності з розділу 2 і виконане на спеціально розробленому програмному стенді.

4.1 Методика проведення експериментального дослідження

Методика базується на ізоляції чинників: треба відділити вплив транспортного протоколу та формату серіалізації від інших джерел затримки. Для цього розроблено окремий програмний стенд, що відтворює міжсервісну взаємодію та real-time доставку повідомлень без зовнішніх компонентів.

Стенд — автономний застосунок .NET 9, у якому серверна і клієнтська частини виконуються в одному процесі, а обмін даними йде через loopback. Серверну частину побудовано на Kestrel, який одночасно обслуговує gRPC поверх HTTP/2, REST поверх HTTP/1.1 та SignalR-хаб поверх WebSocket. Умови вимірювання для всіх трьох протоколів однакові. Для виключення впливу БД (затримки СУБД однакові для всіх варіантів і не стосуються предмета дослідження) під час замірів використано внутрішнє сховище в пам'яті з фіксованим зерном генератора — це дає побітову відтворюваність вмісту повідомлень між запусками.

Етапи методики:

1. підготувати стенд і реалізувати одну й ту саму операцію у двох варіантах — gRPC з Protocol Buffers і REST з JSON — за ідентичної бізнес-логіки;
2. виміряти обсяг серіалізованого повідомлення для обох варіантів за різних розмірів корисного навантаження;

3. виміряти затримку викликів у локальних умовах (loopback) для оцінки витрат на серіалізацію та обробку протоколу;
4. виміряти затримку в моделі реального мережевого з'єднання для оцінки впливу обсягу даних на затримку;
5. виміряти пропускну здатність та використання процесора під сталим конкурентним навантаженням;
6. виміряти затримку доставки повідомлень і масштабованість real-time підсистеми порівняно з періодичним опитуванням.

Для кожного вимірювання затримки виконується 20 викликів попереднього прогрівання середовища і подальша серія вимірюваних викликів. Зі статистики обчислюються середнє значення, медіана (p50) та 99-й процентиль (p99). Усі часові вимірювання — через єдине високороздільне джерело часу (System.Diagnostics.Stopwatch), що дозволяє зіставляти моменти формування повідомлення сервером і його отримання клієнтом без коригування розходження годинників.

Для кожного вимірювання затримки виконується 20 викликів попереднього прогрівання середовища і подальша серія вимірюваних викликів. Зі статистики обчислюються середнє значення, медіана (p50) та 99-й процентиль (p99). Усі часові вимірювання — через єдине високороздільне джерело часу (System.Diagnostics.Stopwatch), що дозволяє зіставляти моменти формування повідомлення сервером і його отримання клієнтом без коригування розходження годинників (4.1):

$$t = t_{proc} + RTT + payload\_bits / B, \quad (4.1)$$

де t_{proc} — виміряний у локальних умовах час обробки запиту; RTT — час подвійного проходження сигналу мережею; payload_bits — обсяг серіалізованого повідомлення у бітах; B — пропускна здатність каналу. Модель застосовується симетрично до обох порівнюваних варіантів, причому для кожного варіанта береться його власний обсяг повідомлення — компактність формату напряму впливає на сумарну затримку.

4.2 Тестове середовище та інструментарій

Параметри тестового середовища наведено в таблиці 4.1.

Таблиця 4.1 — Параметри тестового середовища

Параметр	Значення
Операційна система	Windows 11
Платформа виконання	.NET 9, конфігурація Release
Веб-сервер	Kestrel (ASP.NET Core 9)
Транспортні протоколи	HTTP/2 (gRPC), HTTP/1.1 (REST), WebSocket (SignalR)
Формати серіалізації	Protocol Buffers (gRPC), JSON (REST, SignalR)
Розміщення компонентів	Внутрішній процес, loopback-з’єднання
Сховище даних під час замірів	Внутрішнє сховище у пам’яті, фіксоване зерно
Джерело часу	System.Diagnostics.Stopwatch (тактовий)

Розміщення сервера й клієнта в одному процесі виключає вплив реальної мережі, контейнеризації та операційної системи на іншому вузлі. Виходить чиста оцінка витрат на серіалізацію та обробку транспортних протоколів — стандартний підхід для синтетичних мікробенчмарків. Вплив реальної мережі додатково оцінено через аналітичну модель з підрозділу 4.1.

Параметри продакшн-середовища (PostgreSQL, Docker-контейнери, AWS) у замірах свідомо не задіяні. Вони додають однакових для всіх варіантів затримок, які не залежать від транспортного рівня і лише знижують відношення сигнал/шум.

4.3 Дослідження ефективності міжсервісної взаємодії

Бенчмарк реалізує одну й ту саму операцію GetTopTracks (отримання списку записів) у двох варіантах: gRPC з Protocol Buffers та REST з JSON. Бізнес-логіка та дані відповіді тотожні — всі вимірювані відмінності зумовлені виключно транспортним протоколом і форматом серіалізації.

4.3.1 Дослідження обсягу переданих даних

Обсяг переданих даних вимірювався як точний розмір серіалізованого тіла повідомлення. Для gRPC — штатний метод обчислення розміру повідомлення Protocol Buffers; для REST — фактичний розмір JSON-тіла відповіді. Замір виконано для чотирьох розмірів: 100, 500, 1 000 та 2 000 записів. Результати — в таблиці 4.2.

Таблиця 4.2 — Обсяг серіалізованого повідомлення залежно від кількості записів

Кількість записів	REST/JSON, байт	gRPC/Protobuf, байт	Зменшення обсягу, %
100	3 475	861	75,2
500	17 369	4 328	75,1
1 000	34 781	8 671	75,1
2 000	69 548	17 339	75,1

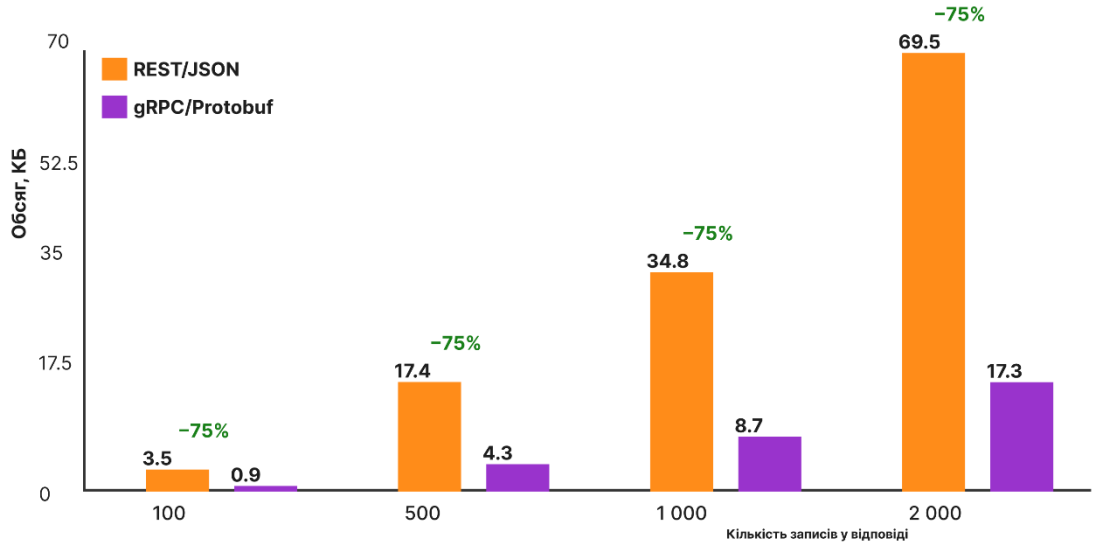

Рисунок 4.1 — Обсяг переданих даних: REST/JSON та gRPC/Protobuf

Protocol Buffers дає приблизно вчетверо менше байтів за JSON на всіх досліджених розмірах. Відношення обсягів стале (~4,0), що означає лінійну залежність розміру від кількості записів для обох форматів і незмінне співвідношення між ними. Зменшення обсягу — близько 75 %.

4.3.2 Дослідження затримки в локальних умовах

Затримку в loopback вимірювали як час від ініціювання виклику клієнтом до отримання відповіді. Для кожного варіанта — 20 викликів прогрівання та 200 вимірюваних; з них рахували медіану (p50) і 99-й процентиль (p99). Результати — в таблиці 4.3.

Таблиця 4.3 — Затримка виконання викликів у локальних умовах (loopback)

Кількість записів	REST p50, мс	REST p99, мс	gRPC p50, мс	gRPC p99, мс
100	0,17	0,30	0,27	0,59
500	0,22	0,77	0,41	0,92
1 000	0,51	1,06	0,72	1,41
2 000	0,83	2,30	0,94	1,62

У loopback час проходження мережею дорівнює нулю — виміряна затримка показує тільки витрати на серіалізацію, фреймування протоколу та обробку запиту в межах процесу. У цих умовах REST/JSON має дещо нижчу медіану на малих обсягах: шлях обробки JSON у System.Text.Json коротший за стек gRPC (десеріалізація Protobuf плюс фреймування HTTP/2). Зі зростанням обсягу розрив зникає, а p99 у gRPC залишається стабільнішим: за 2 000 записів p99 для gRPC становить 1,62 мс проти 2,30 мс для REST. На більших обсягах gRPC дає передбачуваніший час відгуку.

Виміряні в loopback значення не показують переваги компактнішого Protocol Buffers — час передавання мережею тут нульовий. Для оцінки впливу

обсягу в реальних мережевих умовах застосовано аналітичну модель з наступного підрозділу.

4.3.3 Дослідження затримки у моделі мережевого з’єднання

Аналітичну модель з підрозділу 4.1 параметризовано значеннями RTT = 50 мс і пропускної здатності B = 10 Мбіт/с — типові показники віддаленого міжсервісного з'єднання у глобальній мережі. Результати — в таблиці 4.4.

Таблиця 4.4 — Сумарна затримка виконання викликів за моделі мережевого з’єднання (RTT = 50 мс, B = 10 Мбіт/с)

Кількість записів	REST час передавання, мс	gRPC час передавання, мс	REST p50 сумарна, мс	gRPC p50 сумарна, мс	Зменшення, %
100	2,78	0,69	52,97	51,03	3,7
500	13,90	3,46	64,18	53,98	15,9
1 000	27,82	6,94	78,35	57,69	26,4
2 000	55,64	13,87	106,40	64,90	39,0

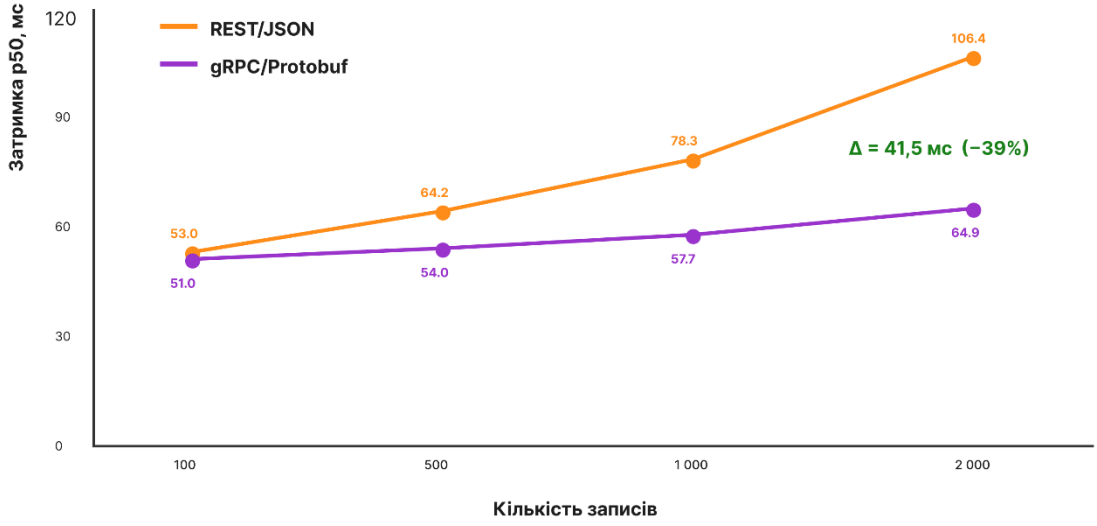

Рисунок 4.2 — Сумарна затримка у моделі мережі (RTT = 50 мс, B = 10 Мбіт/с)

Перевага компактнішого Protocol Buffers, яка в loopback ховається за нульовою затримкою мережі, у мережевих умовах напряму перетворюється на зменшення сумарної затримки. На малих обсягах виграш gRPC незначний — 3,7 % за 100 записів — бо домінує RTT. Зі зростанням обсягу зростає частка часу передавання у сумарній затримці, і виграш швидко росте: 15,9 % за 500 записів, 26,4 % за 1 000, 39,0 % за 2 000. Абсолютний виграш сягає 41,5 мс на виклик за 2 000 записів.

4.3.4 Дослідження пропускної здатності та використання процесорного ресурсу

Пропускну здатність і використання процесора оцінено під сталим конкурентним навантаженням: 16 одночасних клієнтів виконували виклики операції протягом 3 секунд після прогрівання. Рахували тільки успішні виклики. Використання процесора — як відношення сумарного часу процесора процесу до часу спостереження, в перерахунку на один виклик. Результати — в таблиці 4.5.

Таблиця 4.5 — Пропускна здатність та використання процесорного ресурсу під сталим навантаженням

Кількість записів	REST, викл./с	gRPC, викл./с	Зміна, %	REST CPU/виклик, мс	gRPC CPU/виклик, мс
100	36 134	26 137	−27,7	0,452	0,440
500	26 442	26 566	+0,5	0,604	0,323
1 000	18 097	22 488	+24,3	0,863	0,476
2 000	8 183	13 874	+69,5	1,659	0,838

На малих обсягах (100 записів) REST/JSON випереджає за пропускною здатністю: шлях обробки JSON коротший, а даних настільки мало, що компактність Protobuf не покриває накладних витрат gRPC. На 500 записах паритет. Зі зростанням обсягу ситуація інвертується: за 1 000 записів gRPC дає

на 24,3 % більшу пропускну здатність, за 2 000 — на 69,5 %. Вирішальний чинник — використання процесора: на середніх і великих обсягах gRPC споживає у 1,8–2,0 раза менше CPU-часу на один виклик. Причина — компактніше бінарне подання даних і відсутність накладних витрат на синтаксичний розбір текстового JSON.

4.4 Дослідження ефективності real-time підсистеми

Real-time підсистема на SignalR порівнюється з періодичним опитуванням (polling) — типовою альтернативою за відсутності постійного з'єднання. Оцінюються затримка доставки повідомлень і масштабованість зі зростанням кількості одночасних підключень.

4.4.1 Затримка доставки повідомлень

Затримку доставки вимірювали як проміжок часу від моменту формування повідомлення на сервері до моменту його отримання клієнтом. Для SignalR сервер надсилав повідомлення всім підключеним клієнтам через хаб одразу після формування. Для опитування клієнти запитували сервер кожні 2 секунди з випадковим зсувом фази — це дає рівномірний розподіл моменту опитування відносно моменту появи нового повідомлення. У межах одного запуску надсилалося 20 повідомлень. Результати — в таблиці 4.6.

Таблиця 4.6 — Затримка доставки повідомлень засобами SignalR та періодичного опитування

Кількість клієнтів	SignalR p50, мс	SignalR p99, мс	Опитування p50, мс	Опитування p99, мс	Марних запитів, %
1	0,52	17,93	708,5	1 713,7	15,4
10	0,41	0,66	995,7	1 980,8	19,2
50	0,79	1,07	996,7	1 980,9	19,6
100	1,19	1,59	997,5	1 986,7	19,1

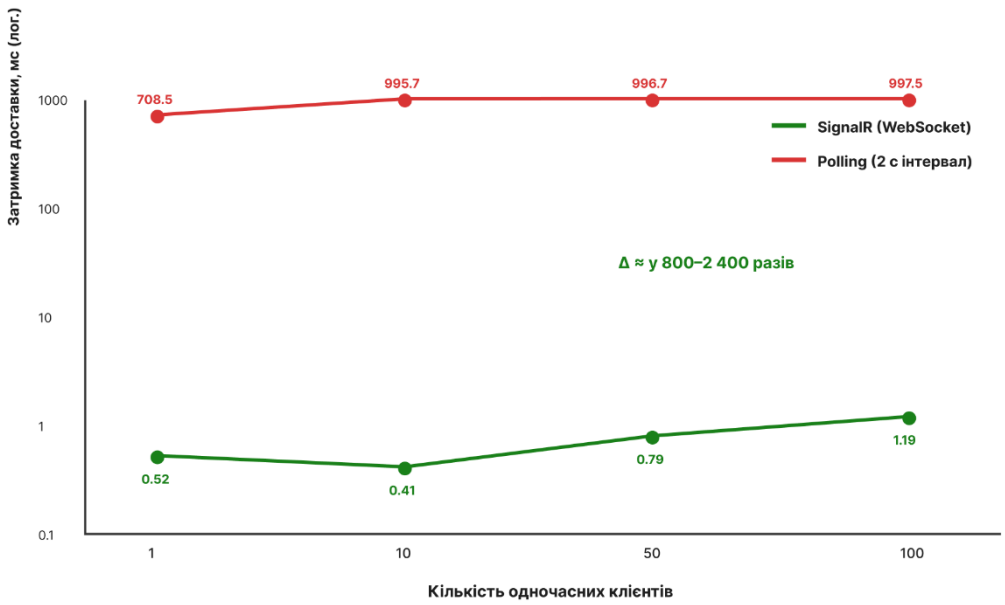

Рисунок 4.3 — Затримка доставки повідомлень: SignalR та періодичне опитування

Цифри підтверджують теоретичні очікування. Медіана затримки SignalR — менше 1,2 мс на всіх рівнях навантаження: повідомлення йде клієнту негайно через постійне WebSocket-з'єднання. Медіана опитування — близько 1 000 мс, тобто половина інтервалу опитування (2 000 мс / 2), а p99 наближається до повного інтервалу. Це збігається з теоретичним розподілом часу очікування рівномірно на інтервалі [0; T) і додатково підтверджує коректність експерименту.

Відношення медіан затримки для 100 клієнтів — 997,5 / 1,19 ≈ 838 разів. Це принципова перевага моделі server push над опитуванням для функціональності реального часу. Окремий показник неефективності опитування — частка марних запитів, які повертають порожню відповідь, бо нових повідомлень за період не було. Виміряна частка стабільно близько 19 %

— майже п'ятий запит даремний. У SignalR таких запитів немає за визначенням: одне підключення на клієнта, одна доставка на повідомлення.

Виміряний для 1 клієнта p99 SignalR (17,93 мс) — поодинокий викид від одноразової ініціалізації каналу під час першого виклику. На серіях з більшою кількістю клієнтів і повідомлень викид зникає (p99 не перевищує 1,6 мс).

4.4.2 Масштабованість real-time підсистеми

Масштабованість оцінено за зміною медіани затримки доставки зі зростанням кількості одночасних клієнтів від 1 до 100. Медіана SignalR росте з 0,4–0,8 мс за 1–50 клієнтів до 1,2 мс за 100 клієнтів. Залежність фактично лінійна, відносне зростання в межах одного порядку — нормальна поведінка масштабованого рішення.

Медіана опитування залишається сталою на рівні близько 997 мс і не залежить від кількості клієнтів — її визначає інтервал опитування. Якщо зменшити інтервал, щоб скоротити затримку, кількість марних запитів зростає обернено пропорційно, що призводить до неприйнятного навантаження на сервер. SignalR такого компромісу не потребує.

4.5 Аналіз та узагальнення результатів дослідження

Узагальнення виконано за критерієм з підрозділу 2.5: протокол ефективніший, якщо за однакових умов дає меншу затримку, менший обсяг переданих даних і більшу пропускну здатність.

Обсяг переданих даних. gRPC з Protocol Buffers дає приблизно вчетверо менше байтів за REST з JSON. Зменшення — близько 75 %, стабільне на всіх досліджених обсягах. Причина — бінарне подання числових значень і відсутність текстових назв полів у Protocol Buffers.

Затримка викликів. Результат двозначний. У loopback, де мережа не задіяна, REST/JSON має дещо нижчу медіану на малих обсягах — шлях обробки JSON у .NET коротший. Але як тільки вводиться модель реальної мережі, компактність Protocol Buffers перетворюється на виграш у сумарній затримці. На каналі RTT = 50 мс, B = 10 Мбіт/с gRPC зменшує сумарну

затримку на 15,9 % за 500 записів, 26,4 % за 1 000 і 39,0 % за 2 000 — і виграш зростає з обсягом.

Пропускна здатність і CPU. Під конкурентним навантаженням gRPC ефективніший на середніх і великих обсягах: за 1 000 записів пропускна здатність вища на 24,3 %, за 2 000 — на 69,5 %; витрати CPU на один виклик у 1,8–2,0 раза менші. На малих обсягах (100 записів) REST/JSON випереджає — накладні витрати gRPC за такого обсягу не покриваються виграшем від компактнішого формату.

Real-time підсистема. SignalR на WebSocket дає медіану затримки доставки менше 1,2 мс на всіх рівнях навантаження від 1 до 100 клієнтів. Опитування з інтервалом 2 секунди — близько 1 000 мс. Різниця — у 800–2 400 разів, що підтверджує перевагу server push. Додатковий показник неефективності опитування — стала частка ~19 % марних запитів, яких у SignalR немає за визначенням.

Узагальнені результати — в таблиці 4.7.

Таблиця 4.7 — Узагальнені результати порівняльного дослідження протоколів

Показник	**Ефективніший варіант**	**Виграш**
Обсяг переданих даних	gRPC	приблизно у 4 рази (зменшення на 75 %)
Затримка виклику у локальних умовах (малі обсяги)	REST/JSON	у межах 0,1–0,2 мс
Затримка виклику у моделі мережі, 1 000–2 000 зап.	gRPC	зменшення на 26–39 %
Пропускна здатність, 1 000–	gRPC	більше на 24–70 %

Показник	Ефективніший варіант	Виграш
2 000 зап.		
Витрати процесора на виклик	gRPC	менші у 1,8–2,0 раза
Затримка доставки real-time	SignalR	менше у 800–2 400 разів
Відсутність марних запитів	SignalR	усунення ≈19 % надлишкового трафіку

Сукупна картина підтверджує доцільність архітектурного рішення з розділу 3 — диференційованого застосування протоколів. gRPC ефективніший для міжсервісної взаємодії за умов реальної мережі та значних обсягів повідомлень — саме того режиму, який характерний для внутрішньої комунікації мікросервісів розподіленої стримінгової системи. SignalR ефективніший для real-time доставки на всіх досліджених рівнях навантаження. REST/JSON залишається придатним для зовнішніх інтерфейсів з малим обсягом окремих повідомлень, де перевага в простоті та коротшому шляху обробки реалізується найповніше.

Висновки за розділом 4

1. Бінарний формат Protocol Buffers дає приблизно у 4 рази менший обсяг повідомлення за JSON. Зменшення близько 75 % стабільне для всіх обсягів відповіді від 100 до 2 000 записів.
2. У моделі реального мережевого з'єднання (RTT = 50 мс, B = 10 Мбіт/с) gRPC зменшує сумарну затримку виклику на 15,9 % за 500 записів, 26,4 % за 1 000 і 39,0 % за 2 000. На малих обсягах і в loopback переваги немає — REST/JSON виграє за рахунок коротшого шляху обробки.

3. Під конкурентним навантаженням gRPC дає на 24,3 % більшу пропускну здатність за 1 000 записів і на 69,5 % — за 2 000, споживаючи у 1,8–2,0 раза менше CPU на виклик. На 100 записах REST/JSON випереджає.
4. SignalR дає медіану затримки доставки повідомлень менше 1,2 мс на навантаженні від 1 до 100 клієнтів — приблизно у 838 разів менше за періодичне опитування з інтервалом 2 секунди (медіана ~997 мс). Опитування додатково генерує близько 19 % марних запитів, яких у SignalR немає за визначенням.
5. Сукупні результати кількісно обґрунтовують диференційований вибір протоколів: gRPC для міжсервісної взаємодії, SignalR для real-time доставки, REST/JSON для зовнішніх інтерфейсів з малим обсягом повідомлень.

ЗАГАЛЬНІ ВИСНОВКИ

У дипломній роботі розв'язано науково-технічну задачу дослідження ефективності протоколів gRPC та SignalR для міжсервісної та real-time комунікації у розподіленій системі стримінгу аудіоконтенту порівняно з REST/JSON. Мету роботи досягнуто, усі поставлені задачі виконано.

1. Проаналізовано предметну область потокової передачі музичного контенту. Сучасні платформи музичного стримінгу — це розподілені інформаційно-телекомунікаційні системи, побудовані переважно на мікросервісній архітектурі. Така архітектура дає незалежне масштабування та локалізацію відмов, але робить ефективність міжсервісної комунікації одним із головних факторів продуктивності.
2. Досліджено протоколи передачі даних у розподілених системах. REST/JSON має обмеження для інтенсивної внутрішньої взаємодії через текстову серіалізацію та відсутність мультиплексування в HTTP/1.1. gRPC адресує ці обмеження через HTTP/2, бінарну серіалізацію Protocol Buffers і потокові режими, а SignalR на основі WebSocket — через повнодуплексний канал з низькою затримкою.
3. Визначено систему показників для оцінювання ефективності протоколів: затримка виконання викликів, пропускна здатність, обсяг переданих даних, затримка доставки повідомлень, масштабованість. Сформульовано критерій ефективності, за яким протокол вважається кращим, якщо за однакових умов дає менші затримку та обсяг переданих даних і більшу пропускну здатність.
4. Розроблено архітектуру розподіленої вебплатформи стримінгу музичного контенту «Sonar» на основі мікросервісного підходу. Система складається з клієнтського застосунку, основного сервера, мікросервісу аналітики та сховища даних. Взаємодія між ними

реалізована трьома різними протоколами — REST/JSON, gRPC і SignalR — кожен на своєму місці.

5. Спроєктовано та програмно реалізовано компоненти вебплатформи: серверну частину за принципами Clean Architecture, міжсервісну взаємодію через gRPC із контрактом у .proto-файлі, real-time підсистему чату та сповіщень через SignalR, потокову передачу аудіо через HTTP Range Requests і клієнтський застосунок як SPA на React і TypeScript.
6. Розроблено методику експериментального дослідження ефективності протоколів. Методика передбачає ідентичне тестове середовище, реалізацію порівнюваних варіантів взаємодії за однакової бізнес-логіки та статистичне опрацювання результатів із аналізом медіани та 99-го процентиля затримки.
7. Кількісно підтверджено перевагу досліджуваних протоколів за визначеним критерієм: gRPC дає приблизно у 4 рази менший обсяг переданих даних, на 26–39 % меншу сумарну затримку в моделі мережі та до 70 % більшу пропускну здатність за середніх і великих обсягів повідомлень; SignalR дає затримку доставки real-time повідомлень у 800–2 400 разів меншу за періодичне опитування і не генерує марних запитів.
8. Архітектуру та програмні рішення вебплатформи «Sonar» можна використати як основу для побудови розподілених систем стримінгу мультимедійного контенту. Методику та результати порівняльного дослідження можна застосовувати для обґрунтованого вибору протоколів при проєктуванні високонавантажених телекомунікаційних систем.

Перспективні напрями подальших досліджень: повномасштабне експериментальне вимірювання продуктивності протоколів за розробленою методикою на реальній мережі, дослідження потокових режимів gRPC для

передавання аудіоконтенту та оцінювання gRPC-Web для прямої взаємодії клієнтського застосунку із сервером.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mueller S. HTTP Live Streaming: Architecture and Implementation. New York : Apress, 2019. 198 p.
2. HTTP Live Streaming : RFC 8216 / Internet Engineering Task Force. 2017. URL: https://www.rfc-editor.org/rfc/rfc8216 (дата звернення: 18.04.2025).
3. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 612 p.
4. Richardson C. Microservices Patterns: With Examples in Java. Shelter Island : Manning Publications, 2019. 520 p.
5. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : Ph.D. dissertation. Irvine : University of California, 2000. 162 p.
6. A Comparative Analysis of Communication Efficiency: REST vs. gRPC in Microservice-Based Ecosystems. Proceedings of the IEEE International Conference on Software Engineering. IEEE, 2024. P. 112–119.
7. Indrasiri K., Kuruppu D. gRPC: Up and Running. Building Cloud Native Applications with Go and Java for Docker and Kubernetes. Sebastopol : O'Reilly Media, 2020. 214 p.
8. Overview of ASP.NET Core SignalR : Microsoft Learn [Електронний ресурс]. URL: https://learn.microsoft.com/aspnet/core/signalr/introduction (дата звернення: 14.04.2025).
9. Performance Evaluation of Microservices Communication with REST, GraphQL, and gRPC. International Journal of Electronics and Telecommunications. 2024. Vol. 70, no. 2. P. 429–436.
10. Hypertext Transfer Protocol Version 2 (HTTP/2) : RFC 7540 / Internet Engineering Task Force. 2015. URL: https://www.rfc-editor.org/rfc/rfc7540 (дата звернення: 10.04.2025).
11. Protocol Buffers Documentation : Overview [Електронний ресурс]. URL: https://protobuf.dev/overview/ (дата звернення: 12.04.2025).

12. The WebSocket Protocol : RFC 6455 / Internet Engineering Task Force. 2011. URL: https://www.rfc-editor.org/rfc/rfc6455 (дата звернення: 10.04.2025).
13. HPACK: Header Compression for HTTP/2 : RFC 7541 / Internet Engineering Task Force. 2015. URL: https://www.rfc-editor.org/rfc/rfc7541 (дата звернення: 10.04.2025).
14. HTTP/2 : RFC 9113 / Internet Engineering Task Force. 2022. URL: https://www.rfc-editor.org/rfc/rfc9113 (дата звернення: 10.04.2025).
15. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing : RFC 7230 / Internet Engineering Task Force. 2014. URL: https://www.rfc-editor.org/rfc/rfc7230 (дата звернення: 10.04.2025).
16. Martin R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
17. Newman S. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. Sebastopol : O’Reilly Media, 2019. 272 p.
18. Elgheriani N. S., Ahmed N. A. S. Microservices vs. Monolithic Architectures: The Differential Structure Between Two Architectures. International Journal of Applied Sciences and Technology. 2022. Vol. 12, no. 1. P. 47–54.
19. Bhatnagar N., Saxena A. Impact of Protocol Selection on Performance and Scalability in Microservices: A Comparison of gRPC, REST, and GraphQL. Journal of Software Engineering Research. 2025. Vol. 13, no. 1. P. 55–70.
20. gRPC Documentation : Introduction to gRPC [Електронний ресурс]. URL: https://grpc.io/docs/what-is-grpc/introduction/ (дата звернення: 12.04.2025).
21. gRPC Core Concepts, Architecture and Lifecycle [Електронний ресурс]. URL: https://grpc.io/docs/what-is-grpc/core-concepts/ (дата звернення: 12.04.2025).
22. gRPC services with ASP.NET Core : Microsoft Learn [Електронний ресурс]. URL: https://learn.microsoft.com/aspnet/core/grpc/aspnetcore (дата звернення: 14.04.2025).
23. Use hubs in ASP.NET Core SignalR : Microsoft Learn [Електронний ресурс]. URL: https://learn.microsoft.com/aspnet/core/signalr/hubs (дата звернення: 14.04.2025).

24. Pimentel V., Nickerson B. G. Communicating and Displaying Real-Time Data with WebSocket. IEEE Internet Computing. 2012. Vol. 16, no. 4. P. 45–53.
25. Łasocha W., Badurowicz M. Comparison of WebSocket and HTTP Protocol Performance. Journal of Computer Sciences Institute. 2021. Vol. 19. P. 67–74.
26. HTTP, WebSocket, and SignalR: A Comparison of Real-Time Online Communication Protocols. Lecture Notes in Networks and Systems. Cham : Springer, 2023. Vol. 730. P. 145–156.
27. Gorton I. Scaling Up REST versus gRPC Benchmark Tests. Journal of Systems and Software Architecture. 2023. Vol. 8, no. 2. P. 33–42.
28. Real-Time Interaction on the Web: How WebSockets Change Application Performance. International Journal of Web Engineering and Technology. 2025. Vol. 20, no. 1. P. 24–39.
29. Стрельніков О. В., Іваненко Т. П. Аналіз протоколів міжсервісної взаємодії у розподілених системах. Вісник Національного технічного університету України «КПІ». Серія: Телекомунікації. 2023. № 24. С. 41–49.
30. Common web application architectures : Microsoft Learn [Електронний ресурс]. URL: https://learn.microsoft.com/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (дата звернення: 15.04.2025).
31. Entity Framework Core Documentation : Overview [Електронний ресурс]. URL: https://learn.microsoft.com/ef/core/ (дата звернення: 15.04.2025).
32. PostgreSQL 16 Documentation [Електронний ресурс]. URL: https://www.postgresql.org/docs/16/ (дата звернення: 15.04.2025).
33. React Documentation : Describing the UI [Електронний ресурс]. URL: https://react.dev/learn (дата звернення: 16.04.2025).
34. TypeScript Documentation : The TypeScript Handbook [Електронний ресурс]. URL: https://www.typescriptlang.org/docs/handbook/ (дата звернення: 16.04.2025).
35. Vite Guide : Why Vite [Електронний ресурс]. URL: https://vite.dev/guide/why (дата звернення: 16.04.2025).

36. Redux Toolkit : RTK Query Overview [Електронний ресурс]. URL: https://redux-toolkit.js.org/rtk-query/overview (дата звернення: 16.04.2025).
37. Feature-Sliced Design : Official Documentation [Електронний ресурс]. URL: https://feature-sliced.design/docs/get-started/overview (дата звернення: 17.04.2025).
38. Docker Documentation : Get Started [Електронний ресурс]. URL: https://docs.docker.com/get-started/ (дата звернення: 18.04.2025).