МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: Створення платформи для відеострімінгу з використанням технологій HTML5 та WebRTC

Здобувача вищої освіти першого (бакалаврського) рівня,
IV курсу, групи КБ-21-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології (за галузями)»

(підпис) Крука Євгенія Сергійовича
(прізвище, ім'я, по батькові)

Науковий керівник (підпис) Грищенко Світлана Миколаївна, к.пед.н., ст. досл.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-професійної програми (підпис) Омельчук Антон Анатолійович, к.т.н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних технологій і систем протокол від «13» травня 2025 № 18

Завідувач кафедри (підпис) Філоненко Михайло Миколайович, к.ф.-м.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь – 2025

Реєстрація №44 23.05.2025 _____
(дата) (підпис відповідальної особи)

Результати перевірки до захисту 23.05.2025 _____
(до захисту, на доопрацювання) (дата) (підпис)

Результат захисту 4 (добре) 04.06.2025 _____
(оцінка) (дата) (підпис)

Голова комісії: _____ В.М. Франчук
(підпис)

Члени комісії:

_____ М.М. Філоненко
(підпис)

_____ С.М. Грищенко
(підпис)

_____ А.А. Омельчук
(підпис)

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	24		Грищенко С.М. Франчук В.М. Філоненко М.М. Грищенко С.М. Омельчук А.А. Суховецька Л.П.
2. Дотримання вимог щодо оформлення	від 0 до 10	8		
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	3		
4. Рецензент	від 0 до 5	4		
РАЗОМ БАЛІВ	від 0 до 50	39		

Оцінювання кваліфікаційної роботи науковим керівником - 36 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
75	C	4 (добре)	04.06.2025	Франчук В.М. Філоненко М.М. Грищенко С.М. Омельчук А.А. Суховецька Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма: «Інформаційні управляючі системи і технології (за галузями)»

Затверджую
Завідувач кафедри _____ Філоненко М.М.
«14» листопада 2024

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Круку Євгенію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Створення платформи для відеострімінгу з використанням технологій HTML5 та WebRTC
2. Затверджена наказом ректора від «13» листопада 2024 № 1717
3. Строк здачі закінченої роботи «08» травня 2025
4. Мета роботи: Створення вебплатформи для відеострімінгу.

Об'єкт дослідження Вебплатформа для відеострімінгу.

Предмет дослідження Організація та реалізація процесів передачі відео і аудіо даних в режимі реального часу.

Орієнтовна база дослідження Прототип веб-платформи для відеострімінгу на базі технологій HTML5, WebRTC і WebSocket.

4.Вихідні дані до роботи методи та підходи до розробки та створений вебдодаток

5. Перелік графічного матеріалу WebRTC Internals із відображенням змін станів з'єднання, логи ngrok з параметрами тунелювання та HTTP-запитами

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП
1.Огляд технологій HTML5 та WebRTC
1.1.Історія, застосування та особливості HTML5 у відео та аудіо.
1.2.Можливості WebRTC для потокового передавання відео в реальному часі.
Розділ 2.Проектування та розробка платформи
2.1.Архітектура платформи: клієнт-серверна модель з використанням WebRTC.
2.2.Реалізація основних функцій: відеострімінг та чат у реальному часі.
3. Тестування, оптимізація та перспективи розвитку системи
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

7. Календарний план виконання роботи
8. Дата видачі завдання «14» листопада 2024

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів робот	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 30.10.2024	До 30.10.2024
2	**І розділ** (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	27.12.2024	27.12.2024
3	**ІІ розділ** (аналіз експериментальної частини про методики досліджень)	17.02.2025	17.02.2025
4	**ІІ розділ** (проведені теоретичні і або) експериментальні дослідження а проектні рішення)	16.04.2025	16.04.2025
5	**Висновки**	01.05.2025	01.05.2025
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	09.05.2025	09.05.2025
7	Отримання відгуку керівника	27.05.2025	27.05.2025
8	Отримання зовнішньої рецензії	29.05.2025	29.05.2025

9. Керівник кваліфікаційної роботи _____ (підпис) Грищенко С.М (прізвище, ініціали)

10. Керівник освітньо-професійної програми _____ (підпис) Омельчук А.А (прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти _____ (підпис) Крук Є.С (прізвище, ініціали)

АНОТАЦІЯ

Створення платформи для відеострімінгу з використанням технологій HTML5 та WebRTC.

Мета роботи полягає у створенні вебплатформи для відеострімінгу в реальному часі з використанням технологій HTML5 та WebRTC, що дозволяє покращити доступ до онлайн-трансляцій та реалізувати інтерактивну взаємодію між користувачами без складних налаштувань і реєстрації.

Завданнями роботи є:

- Проведення аналізу сучасних технологій для організації відеотрансляцій.
- Розробка архітектури клієнт-серверної взаємодії.
- Створення функціональної вебплатформи для стріму.
- Проведення тестування стабільності, затримок і якості трансляції.

Предметом дослідження є побудова вебплатформи для онлайн-відеострімінгу.

Об'єктом дослідження є сучасні вебтехнології HTML5, WebRTC, WebSocket.

Базою дослідження виступає реалізований прототип відеострімінгової системи.

За результатами дослідження сформульовані методика та етапи реалізації відеострімінгової платформи для браузерного середовища.

Одержані результати можуть бути використані при створенні комерційних, освітніх або соціальних стрімінгових рішень з інтеграцією в існуючі сервіси.

Кваліфікаційна робота містить 56 сторінку, 2 рисунків, список використаних джерел із 40 найменувань, 2 додатки.

Ключові слова: WebRTC, HTML5, трансляція, відеострім, чат, браузер, реальний час, відеоплатформа.

Рік виконання кваліфікаційної роботи: 2025

Рік захисту роботи: 2025

ABSTRACT

Creating a video streaming platform using HTML5 and WebRTC technologies

The aim of the work is to create a real-time video streaming web platform using HTML5 and WebRTC technologies, which improves access to online broadcasts and enables interactive user interaction without complex settings or registration.

The objectives of the work are:

- To analyze modern technologies for organizing video streaming.
- To develop the architecture of client-server interaction.
- To create a functional video streaming web platform.
- To perform testing of stability, latency, and video quality.

The subject of the study is the development of a web platform for online video streaming.

The object of the study is modern web technologies such as HTML5, WebRTC, and WebSocket.

The research base is the implemented prototype of a browser-based video streaming system.

Based on the results of the research, the methodology and implementation stages of the video streaming platform were formulated.

The obtained results can be used in the development of commercial, educational, or social streaming solutions with potential integration into existing services.

The qualification work consists of 56 pages, 2 figures, a list of 40 sources, and 2 appendices.

Keywords: WebRTC, HTML5, streaming, video stream, chat, browser, real-time, video platform.

Year of completion of the qualification work: 2025

Year of defense: 2025

ЗМІСТ

ВСТУП 3
РОЗДІЛ 1.ОГЛЯД ТЕХНОЛОГІЙ HTML5 ТА WEBRTC 5
1.1.Історія, застосування та особливості html5 у відео та аудіо. 5
1.2.Можливості webrtc для потокового передавання відео в реальному часі. 10
РОЗДІЛ 2.ПРОЕКТУВАННЯ ТА РОЗРОБКА ПЛАТФОРМИ 18
2.1.Архітектура платформи: клієнт-серверна модель з використанням webrtc 18
2.2.Реалізація основних функцій: відеострімінг та чат у реальному часі. 28
РОЗДІЛ 3. ТЕСТУВАННЯ, ОПТИМІЗАЦІЯ ТА ПЕРСПЕКТИВИ РОЗВИТКУ СИСТЕМИ 40
ВИСНОВКИ 45
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 49
ДОДАТКИ 52

ВСТУП

Стрімкий розвиток вебтехнологій значно розширює можливості організації онлайн-взаємодії між користувачами. Однією з таких можливостей є відеокомунікація в реальному часі, яка дедалі активніше застосовується в різних сферах — від освіти й телемедицини до корпоративного спілкування, цифрового контенту та індустрії розваг. Актуальність розробки платформи для відеострімінгу та конференцій обумовлена потребою у простих, безпечних і масштабованих рішеннях, що працюють безпосередньо в браузері і не потребують встановлення додаткового програмного забезпечення.

З переходом значної частини професійної, освітньої та соціальної активності в онлайн-середовище зростає запит на інтерактивні вебінструменти. Користувачі очікують стабільної, інтуїтивно зрозумілої системи, яка забезпечує якісний зв’язок із мінімальною затримкою та гарантує приватність даних. WebRTC (Web Real-Time Communication) — технологія, що відповідає цим вимогам, оскільки дозволяє встановлювати пряме з’єднання між браузерами для передачі відео, аудіо та інших даних без проміжного сервера.

Метою цієї кваліфікаційної роботи є створення універсальної платформи для реалізації односторонніх стрімів та багатосторонніх відеоконференцій на базі WebRTC із використанням WebSocket для сигналізації та Node.js як серверної частини. Система має передбачати можливість підключення без реєстрації, забезпечувати текстовий чат у реальному часі, демонструвати екран та надавати гнучке керування доступом до мікрофона й камери.

Об’єктом дослідження є процес проєктування та реалізації браузерної платформи на основі WebRTC для передачі аудіо- та відеопотоків у реальному часі.

Предметом дослідження виступає клієнт-серверна архітектура, що забезпечує передачу медіаданих між користувачами за допомогою протоколів WebRTC і WebSocket.

Завдання дослідження:

Провести аналіз сучасних вебтехнологій для відеозв’язку;

Розробити архітектуру вебплатформи на базі WebRTC;

Реалізувати сигнальний сервер на Node.js із підтримкою WebSocket-з’єднань;

Розробити клієнтську частину з можливістю трансляції стріму та участі в конференціях;

Реалізувати чат, обробку підключень, список глядачів і оновлення користувацького інтерфейсу;

Протестувати систему в умовах реального навантаження;

Оцінити потенціал масштабування та можливість розширення до SFU [18,19,27]- або MCU-архітектури.

Методи дослідження — аналіз технічної документації, експериментальне моделювання, порівняльний аналіз і тестування.

База дослідження — власноруч реалізована вебплатформа для відеострімінгу.

Науково-практична значимість одержаних результатів полягає в тому, що розроблена система може бути адаптована для використання у сферах дистанційного навчання, відеозв’язку, стрімінгу та підтримки користувачів.

Практична значущість роботи полягає в розробці масштабованого й безпечного рішення для реального відеозв’язку в браузері, яке може бути адаптоване до потреб онлайн-навчання, дистанційної роботи або трансляцій масових заходів.

Розділ 1. ТЕОРЕТИЧНІ ОСНОВИ ВІДЕОСТРІМІНГУ В HTML5 ТА WebRTC

1.1. Історія, застосування та особливості HTML5 у відео та аудіо.

Розвиток Інтернет-мультимедіа досяг значного прогресу. Раніше розробники використовували плагіни сторонніх виробників, такі як Adobe Flash, RealPlayer або QuickTime, щоб вставляти відео та аудіо на веб-сторінки. Ці технології вимагають встановлення додаткового програмного забезпечення, що ускладнює роботу користувача та створює проблеми з безпекою.

У 2004 році робоча група WHATWG (Web Hypertext Application Technology Working Group) почала розробку нового стандарту HTML5[1,3,6,14], який має на меті спростити інтеграцію мультимедіа без необхідності сторонніх плагінів. У 2008 році вперше були запропоновані теги <video> і <audio>[2,7,13,18], що стало революційним кроком в історії розвитку веб-технологій. Нарешті, у 2014 році HTML5 був офіційно затверджений як стандарт, і його підтримка була загальновизнана браузерами.

За впровадженням HTML5 почалася нова ера веб-розробки, оскільки новий стандарт усунув потребу в зовнішніх плагінах. Це забезпечило не лише підвищення безпеки та сумісності з різними пристроями, але й зробило можливим інтегрувати мультимедійний контент безпосередньо в браузері. Такий підхід сприяє більш простій розробці та підтримці сучасних веб-додатків.

Сьогодні HTML5 є базовою технологією сучасних веб-додатків. Його широка підтримка забезпечує кросбраузерність і сумісність сучасних інтерфейсів, а можливість інтеграції з CSS та JavaScript сприяє створенню інтерактивних та адаптивних рішень, що відповідають вимогам як настільних, так і мобільних пристроїв

Де застосовується HTML5[1] для відео та аудіо

Сьогодні HTML5 використовується в широкому спектрі галузей:

Медіа та стрімінгові платформи:

· YouTube, Netflix, Twitch – провідні сервіси потокового відео, які використовують HTML5 для відтворення контенту.[9,20,24]

Освітні ресурси:

· Онлайн-курси та вебінари (Coursera, Udemy) активно використовують <video> для інтерактивного навчання.

Веб-ігри та інтерактивні застосунки:

· Ігрові платформи та анімаційні ресурси використовують <audio> для створення динамічного контенту та звукових ефектів.

Соціальні мережі та новинні платформи:

· Facebook, Instagram, TikTok – ці платформи мають інтегровані HTML5-плеєри для обробки відеоконтенту.

Відеоконференції:

· WebRTC[2] (технологія HTML5[1]) забезпечує пряму трансляцію аудіо та відео без необхідності встановлення додаткових плагінів.

Основний функціонал HTML5[1] для роботи з відео та аудіо

HTML5[1] значно спростив роботу з мультимедіа завдяки низці важливих можливостей:

· Тег <video> та <audio>: дозволяє безпосередньо вбудовувати мультимедіа на веб-сторінку без потреби в сторонніх програмах.

· Підтримка форматів: браузери підтримують різні типи файлів (MP4, WebM, Ogg для відео; MP3, AAC, Ogg для аудіо).

· Атрибути для керування: можливість налаштування параметрів відтворення (autoplay, loop, poster).

· API для розширеного керування: за допомогою JavaScript можна додавати функції контролю, стрімінгу, зміни якості відео тощо.

· Кросбраузерна сумісність: HTML5[1] підтримується усіма сучасними браузерами (Chrome, Firefox, Edge, Safari).

Media Source Extensions (MSE)[15,17,28] як технологія потокового відео

Історія розвитку потокового відео.

На початку розвитку Інтернету передача відео була обмежена низькою швидкістю з'єднання та технічними можливостями веб-технологій. Ранні версії браузерів не мали стандартного механізму для підтримки відеострімінгу, тому розробники використовували плагіни на кшталт Adobe Flash, RealPlayer або Windows Media Player для забезпечення відтворення потокового контенту.

Однак ці технології мали низку недоліків:

· Неоптимальне використання ресурсів – надмірне навантаження на процесор та пам’ять.

· Слабка безпека – плагіни часто мали вразливості, що дозволяли зловмисникам виконувати небезпечний код.

· Несумісність – деякі мобільні пристрої не підтримували Flash, що призвело до його поступового занепаду.

З появою HTML5[1] стало можливим інтегрувати відео без сторонніх плагінів. Media Source Extensions (MSE), які були вперше представлені у 2013 році, значно розширили можливості потокового відео, забезпечуючи адаптивне передавання та можливість динамічного оновлення контенту.

Де використовується технологія MSE?

MSE застосовується у широкому спектрі веб-додатків, що працюють із потоковим відео:

Стрімінгові сервіси: YouTube, Netflix, Twitch використовують MSE для адаптивного відеострімінгу.

Онлайн-трансляції: Платформи для відеоконференцій та вебінарів (Zoom, Google Meet).

Ігрові стріми: Використання в трансляціях геймплею на таких платформах, як Steam Broadcasting.

Новинні сайти: Динамічне оновлення відео у реальному часі, наприклад, у стрічках новин CNN або BBC.

Мультимедійні навчальні курси: Coursera, Udemy застосовують MSE для інтерактивного навчального контенту.

Основний функціонал та можливості MSE

Media Source Extensions – це набір API, які дозволяють динамічно змінювати вміст відео без необхідності перезавантаження сторінки. Це досягається шляхом завантаження фрагментів відео та їх поступової обробки браузером.

Адаптивне потокове передавання – відео може змінювати свою якість залежно від швидкості інтернет-з'єднання користувача.

Динамічне завантаження контенту – відео не зберігається як єдиний файл, а передається фрагментами, що економить ресурси. Оптимізація роботи серверів – не потрібно передавати повний файл кожному користувачу, а тільки необхідні частини відео. Зменшення витрат на трафік – передаються лише ті фрагменти, які необхідні для поточного відтворення. Гнучкість у форматах – MSE підтримує сучасні кодеки та адаптивні технології, такі як MPEG-DASH та HLS.

Застосування MSE у браузерах

Media Source Extensions дозволяють браузерам динамічно завантажувати фрагменти відео, що забезпечує адаптивне потокове передавання. Цей підхід дозволяє автоматично регулювати якість відео відповідно до поточної пропускної здатності мережі, що є ключовим для стабільного відтворення контенту в умовах змінного інтернет-з'єднання

Всі сучасні браузери підтримують Media Source Extensions, що робить цю технологію стандартом для потокового відео.

Google Chrome – має повну підтримку MSE, включаючи інтеграцію із YouTube та іншими стрімінговими платформами.

Mozilla Firefox – підтримує MSE, але деякі формати можуть потребувати додаткових налаштувань.

Microsoft Edge – повністю сумісний із MSE та технологіями адаптивного відео.

Apple Safari – підтримує MSE, але його робота може залежати від конфігурації операційної системи macOS та iOS.

Використання WebSocket[6]s для відеострімінгу

WebSocket[6,29,30]s – це протокол комунікації, який дозволяє постійне двостороннє з'єднання між клієнтом і сервером. Це означає, що дані можуть передаватися в будь-який момент, без необхідності повторного відкриття з'єднання, як у традиційних HTTP-запитах.

Переваги WebSockets у відеострімінгу:

Низька затримка – забезпечує майже миттєву передачу даних у реальному часі.

Двостороння передача – як клієнт, так і сервер можуть надсилати дані без очікування запиту.

Економія ресурсів – постійне з'єднання зменшує накладні витрати на встановлення нових запитів.

Підходить для відеозв'язку та стрімів – дозволяє передавати відео, аудіо та супутню інформацію (чат, реакції, коментарі).

Практичні застосування WebSocket[6]s у відеострімінгу

Живі трансляції: WebSockets використовуються у YouTube Live, Twitch та Facebook Live для чату та інтерактивних елементів.

Відеозв’язок: WebRTC використовує WebSockets для сигналізації між браузерами.

Ігрові стріми: У Twitch WebSockets використовуються для синхронізації подій між геймерами.

Реакції та інтерактивні елементи: Коментарі, опитування, реакції у відеотрансляціях обробляються через WebSockets.

Завдяки використанню сучасних CSS-фреймворків і JavaScript-інструментів розробники можуть створювати інтерфейси, які автоматично підлаштовуються під різні розміри екранів. Це значно покращує користувацький досвід, забезпечуючи зручний доступ до відеоконтенту не лише на великих моніторах, але й на мобільних пристроях з обмеженими ресурсами

1.2. WebRTC Можливості та перспективи використання

Peer-to-Peer зв’язок у WebRTC[2]

Принцип роботи WebRTC[2]

WebRTC[2] (Web Real-Time Communication) — це технологія, що дозволяє браузерам здійснювати аудіо-, відео- та передачу даних напряму між користувачами без використання сторонніх серверів. Це значно скорочує затримки та покращує якість зв’язку.

Історично, WebRTC з’явився як відповідь на зростаючу потребу у реальному часі спілкування в Інтернеті. Завдяки відкритим стандартам ця технологія швидко знайшла застосування для створення високоякісних відео- та аудіозв’язків, що не потребують додаткових серверних релеїв, забезпечуючи тим самим знижену затримку та більш ефективну передачу даних[5,11,19,31].

Основні компоненти WebRTC[2]:

· RTCPeerConnection[2] — встановлює P2P-з’єднання між пристроями.

RTCPeerConnection[2,5,11,19,31] відповідає за обмін технічною інформацією між клієнтами, використовуючи протоколи SDP та ICE для узгодження параметрів з’єднання. Завдяки цьому забезпечується максимальна ефективність прямого з’єднання, що мінімізує затримки та покращує якість відтворення медіа.

· ICE (Interactive Connectivity Establishment) — допомагає пристроям знайти найкращий шлях для зв’язку.

· STUN[4]/TURN сервери — підтримують з’єднання через NAT та брандмауери.

За допомогою STUN-серверів клієнти можуть дізнатися свою публічну IP-адресу, що дозволяє встановити пряме Peer-to-Peer з’єднання. У випадках, коли мережеві обмеження не дозволяють пряме з’єднання, TURN-сервер виступає релеєм, забезпечуючи передачу даних між клієнтами навіть у складних умовах NAT чи брандмауерів

WebRTC[2] дозволяє передавати медіа-потоки безпосередньо між користувачами, що робить його ідеальним для реалізації відеоконференцій, стрімінгу та онлайн-спілкування.

Як встановлюється Peer-to-Peer з’єднання?

1. Сигналізація – браузери обмінюються інформацією про з’єднання (SDP) через сервер сигналізації.

2. Визначення найкращого шляху – WebRTC[2] використовує STUN[4]/TURN сервери для встановлення з’єднання, навіть якщо користувачі перебувають за NAT.

3. Обмін даними через RTCPeerConnection – після узгодження параметрів медіа-трафік передається напряму між клієнтами.

Такий спосіб забезпечує максимальну швидкість передачі даних та мінімізує затримки, що критично важливо для відео-дзвінків, стрімів та інших реальних комунікацій.

Переваги Peer-to-Peer зв’язку у WebRTC[2]

Низька затримка – передача відео та аудіо без зайвих серверних обробок.

Економія ресурсів – трафік не проходить через центральний сервер, що зменшує навантаження на інфраструктуру.

Висока якість зв’язку – мінімізація втрати пакетів і стабільність з’єднання.

Масштабованість – кожен новий користувач не створює додаткового навантаження на сервери.

Основний принцип роботи getUserMedia[2]()

Метод getUserMedia() у WebRTC[2] надає доступ до камери та мікрофона користувача, дозволяючи отримати відео- та аудіо-потік у реальному часі. Він використовується для відеодзвінків, стрімінгу, запису контенту та інтерактивних веб-додатків.

Як отримати доступ до камери та мікрофона?

Запит дозволу користувача – браузер запитує у користувача дозвіл на використання медіа-пристроїв.

Створення медіа-потоку – якщо дозвіл надано, система передає відео та аудіо в об'єкт MediaStream.

Використання потоку – він може бути відображений у <video> або переданий іншому клієнту через WebRTC.

Медіа потік додається до <video> для відображення користувачу. Обробка помилок важлива для випадків, коли камеру або мікрофон неможливо використовувати.

Обмеження та вимоги

HTTPS-з'єднання – працює тільки на захищених веб-сайтах (виняток — localhost).

Явна згода користувача – браузер не дозволяє доступ до камери без підтвердження.

Сумісність із браузерами – метод підтримують Chrome, Firefox, Edge, Safari.

Налаштування якості відео – можна вказати роздільну здатність та частоту кадрів.

Передача медіа-потоку через WebRTC[2]

Медіа-потік може бути переданий іншим користувачам через WebRTC[2]:

stream.getTracks().forEach(track => peerConnection.addTrack(track, stream));

RTCPeerConnection дозволяє передавати відео та аудіо між клієнтами без серверного посередника. Використовується для відеодзвінків, стрімінгу та інтерактивних веб-застосунків.

WebRTC[2] також підтримує передавання довільних даних у реальному часі за допомогою DataChannel — спеціального каналу, що працює в межах RTCPeerConnection. Це дозволяє реалізовувати, зокрема, функціонал чатів, обміну файлами або командами керування, що надзвичайно корисно для побудови комплексних комунікаційних рішень.

RTCDataChannel[23,32,34] відкриває додаткові можливості для інтерактивності, дозволяючи реалізувати текстовий чат, обмін файлами та інші форми передачі даних у режимі реального часу. Це додає системі гнучкості, розширює функціональність додатку і забезпечує миттєву реакцію населення на інтерактивні події під час відеотрансляції.

DataChannel забезпечує:

Низьку затримку передачі тексту та сигналів, що важливо для синхронізованої взаємодії.

Надійність або швидкість (вибірково) — можна обирати між гарантованою доставкою або пріоритетом швидкості залежно від задачі.

Незалежність від медіа — дозволяє передавати текстову або службову інформацію паралельно з відео- та аудіопотоками.

Підтримка трансляції екрану (screen sharing) — ще одна важлива можливість WebRTC[2], яка широко використовується у відеоконференціях, навчанні та спільній роботі. Завдяки API navigator.mediaDevices.getDisplayMedia() користувач може обрати вікно або весь екран для трансляції, що дає змогу інтегрувати функціонал демонстрації в реальному часі без додаткових розширень.

Функція screen sharing: Використовується у відеоконференціях, системах віддаленої підтримки. Забезпечує динамічну взаємодію між учасниками. Реалізується аналогічно до getUserMedia(), з додатковими можливостями вибору джерела.

WebRTC[2] продовжує активно розвиватися. Його підтримка у більшості сучасних браузерів[8,10,21], а також відкритий стандарт реалізації дають змогу швидко адаптувати цю технологію для різних платформ. Існують приклади комерційних систем (Zoom, Google Meet), що частково використовують WebRTC, а також численні open-source фреймворки (наприклад, SimpleWebRTC або PeerJS), які суттєво полегшують впровадження.

У перспективі WebRTC[2] також може бути інтегровано з SFU (Selective Forwarding Unit) — сервером, який не декодує медіа, а просто ретранслює потоки іншим учасникам. Це відкриває шлях до реалізації великих конференцій із десятками або сотнями учасників без значного навантаження на кожен клієнт.

Вибір відповідних кодеків і форматів упаковки відеопотоку є наріжним каменем для забезпечення високої якості та плавності відтворення. Сучасні браузери підтримують такі популярні алгоритми стиснення, як H.264, VP8 та VP9, які завдяки апаратному прискоренню здатні стискати відеодані з мінімальними втратами деталізації. Для адаптивного відтворення застосовуються протоколи MPEG-DASH та HLS, що розбивають відео на сегменти різної тривалості та бітрейту. Браузер у реальному часі аналізує пропускну здатність мережі та обирає найвідповідніший сегмент, щоб уникнути тривалого буферування або раптових переривань.

Якість сервісу (QoS) підтримується через адаптивний бітрейт (ABR), реалізований за допомогою інтерфейсу MSE (Media Source Extensions). Цей механізм дозволяє автоматично змінювати параметри кодування, отримуючи зворотний зв’язок від RTCP Sender Reports про втрати пакетів, затримки та джиттер. Завдяки такій динаміці можна своєчасно відкоригувати бітрейт, забезпечивши стабільне зображення навіть на ненадійних каналах зв’язку.

Окрему увагу приділено захисту контенту, що реалізується через Encrypted Media Extensions (EME) та вбудовані провайдери DRM[33,37] (Digital Rights

Management). Під час передачі відео потік шифрується на сервері й залишається недоступним для неавторизованих користувачів. Розшифрування відбувається безпосередньо у браузері лише після успішної автентифікації, що гарантує дотримання авторських прав та запобігає несанкціонованому копіюванню.

Окрім технічних аспектів, доцільно також розглядати структурну побудову стрімінгових систем у контексті сучасних вебтехнологій. У більшості випадків реалізація базується на комбінації WebRTC, WebSocket та медіасерверів з можливістю масштабування. Це дозволяє забезпечити високу доступність, відмовостійкість та відповідність сучасним вимогам до інтерактивних сервісів реального часу. Такий підхід також відкриває перспективи щодо інтеграції додаткових функцій, зокрема аналітики поведінки користувачів, адаптивної реклами та механізмів персоналізації контенту.

Із розвитком вебтехнологій HTML5 став важливим стандартом для реалізації мультимедійних можливостей без використання сторонніх плагінів, зокрема таких, як Adobe Flash, який втратив підтримку з 2021 року. Одним із ключових елементів HTML5 є підтримка елементів <video> та <audio>, що дозволяють відтворювати медіа прямо в браузері з широким контролем над плеєром.

Стандартизація форматів відео, таких як MP4 (H.264), WebM (VP8/VP9) і Ogg (Theora), зробила HTML5 гнучким засобом для потокової трансляції. Крім того, API HTML5 дозволяє взаємодіяти з JavaScript для створення інтерактивних та динамічних відеоплеєрів, що зручно інтегруються з іншими вебтехнологіями, включаючи WebRTC. HTML5 також активно використовується в мобільних додатках і PWA (Progressive Web Apps), що дозволяє забезпечити кросплатформенність без потреби в розробці окремих клієнтів. Таким чином, використання HTML5 стало основою для багатьох сучасних відеострімінгових рішень[10,11,26,27] і платформ у різних галузях, від освіти до онлайн-ігор і відеоконференцій.

Окрім мультимедійних API, HTML5 забезпечує також широкі графічні можливості завдяки Canvas API, який дозволяє в реальному часі рендерити відеопотоки, створювати візуальні ефекти, застосовувати фільтри, накладати текст або графіку поверх відео [13,16,28]. Така гнучкість робить HTML5 універсальним засобом для створення кастомізованих плеєрів, відеоаналітики, а також елементів взаємодії у стрімінгових середовищах.

Сучасні вебплатформи також активно використовують концепцію Service Worker — частину HTML5-екосистеми, яка дозволяє кешувати контент, працювати офлайн та асинхронно обробляти запити [20,25]. Це суттєво знижує навантаження на сервери та покращує швидкість завантаження сторінок, особливо на мобільних пристроях або в умовах нестабільного підключення.

Ще одним важливим напрямом є підтримка енергоефективності. Завдяки впровадженню апаратного прискорення відтворення відео (наприклад, за допомогою GPU) HTML5 дозволяє економити ресурси пристрою без втрати якості зображення [14,17]. Це особливо важливо для мобільних користувачів, які активно споживають відеоконтент, перебуваючи в русі.

HTML5 також забезпечує належний рівень безпеки для відеострімінгу. Завдяки CORS (Cross-Origin Resource Sharing), CSP (Content Security Policy), HTTPS та шифруванню медіапотоків платформи на HTML5 здатні мінімізувати ризики перехоплення або підміни трафіку [33,34,37]. У поєднанні з EME та DRM-механізмами це формує надійну основу для ліцензійного та платного відеоконтенту.

Важливо зазначити, що HTML5 став стандартом для більшості сучасних комерційних відеоплатформ. Наприклад, YouTube, Netflix, Facebook Live, Coursera та Udemy базують свої фронтенд-частини на HTML5-плеєрах з інтеграцією WebRTC або HLS [9,24,27]. Це підтверджує практичну ефективність і

масштабованість даного стандарту як у відкритому, так і в корпоративному середовищі.

Загалом, HTML5 не лише полегшує розробку клієнтських інтерфейсів, а й формує основу для побудови високонавантажених мультимедійних систем, здатних обробляти тисячі одночасних з’єднань без втрати продуктивності або якості зображення [1,5,19].

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПЛАТФОРМИ

2.1. Архітектура платформи: клієнт-серверна модель з використанням WebRTC

WebRTC (Web Real-Time Communication) є набором відкритих технологій та API, що дозволяють здійснювати передачу відео, аудіо та даних у режимі реального часу безпосередньо між браузерами, без необхідності встановлення додаткових плагінів чи програмного забезпечення. Це забезпечує можливість встановлення P2P (peer-to-peer) з’єднань між клієнтами для перенесення мультимедійного контенту з мінімальною затримкою.

Система відеострімінгу розроблена на основі клієнт-серверної архітектури[12,15,22], що дозволяє забезпечити масштабованість, зручність обслуговування і гнучкість у розширенні функціоналу. Основна мета полягає в організації односторонньої трансляції, яка супроводжується інтерактивним текстовим чатом у режимі реального часу для забезпечення взаємодії між стрімером і глядачами.

Для реалізації стрімової частини створено WebRTC-сервер на Node.js із використанням бібліотеки Socket.io. Стрімер починає трансляцію через сторінку stream.html, де надається дозвіл на використання камери або екрану. Після цього браузер ініціює RTCPeerConnection та надсилає SDP-пропозицію (offer) глядачам через сигнальний сервер. Коли глядач підключається до сторінки watch.html, він отримує цей оффер і формує відповідь (answer). Таким чином, після встановлення зв’язку відео транслюється безпосередньо за допомогою WebRTC.

Щоб забезпечити доступність сервера зі будь-якого пристрою, незалежно від наявності реального домену або публічної IP-адреси, у процесі тестування використовувалась платформа ngrok. Це дозволило уникнути проблем з NAT і забезпечити простий доступ до сервісу.

Крім того, була реалізована можливість спілкування між користувачами за допомогою чату на базі WebSocket. Під час стріму кожному користувачу автоматично присвоюється нікнейм, який зберігається в localStorage. Було виявлено, що при оновленні сторінки трансляція не відновлюється автоматично, і ця проблема була вирішена завдяки повторному встановленню з'єднання зі стрімером через Socket.io.

Для організації конференцій застосовано окрему логіку, яка передбачає, що кожен учасник створює WebRTC-з'єднання з усіма іншими. При створенні нової конференції генерується унікальний ідентифікатор, який додається до списку доступних кімнат, а приєднання до кімнати активує процес обміну SDP та ICE-кандидатами між усіма учасниками. Додатково, впроваджено функціонал розшарення екрану та перемикання між камерою та мікрофоном.

Основні компоненти системи:

Клієнт (стрімер):

Вебінтерфейс, через який користувач надає доступ до камери та мікрофона, ініціює стрім, а також може бачити список глядачів у реальному часі. Вся логіка обробки медіапотоку реалізована за допомогою WebRTC[2].

Клієнт (глядач):

Глядач отримує потокове відео через технологію WebRTC[2], що забезпечує ефективну передачу медіа між браузерами, а також може миттєво долучатися до інтерактивного чату. Реалізація чату за допомогою RTCDataChannel мінімізує затримки при передачі повідомлень, що покращує комфорт користувача.

Сигнальний сервер:

Сигнальний сервер відповідає за ініціалізацію з'єднання між стрімером і глядачами, використовуючи протокол WebSocket[6] для обміну повідомленнями, такими як SDP-об'єкти та ICE-кандидати, які є необхідними для встановлення WebRTC-з'єднання.

Серверна частина (Node.js):

Серверна частина, реалізована на базі Node.js, виконує функції сигнального сервера, забезпечуючи маршрутизацію подій, облік підключень та обробку чат-повідомлень. У майбутньому її функціонал може бути розширено для збереження трансляцій, ведення статистики переглядів та впровадження багаторівневої якості відео.

Ngrok[39,40] – це платформа для створення захищених тунелів між локальним сервером та глобальною мережею Інтернет. Вона дозволяє розробникам тестувати веб-додатки, API та інші сервіси без необхідності розгортати їх на публічному сервері. Використання ngrok дозволяє створити захищений тунель для доступу до локального сервера, що значно спрощує тестування в умовах NAT без наявності публічної IP-адреси.

ngrok працює як глобально розподілений зворотний проксі-сервер, що дозволяє створювати безпечні тунелі між локальним сервером і мережею Інтернет. Основний принцип його роботи базується на наступних етапах.

Основні можливості ngrok включають створення загальнодоступного посилання на локально запущений вебдодаток, підтримку таких протоколів як HTTP, HTTPS і TCP, а також реалізацію заходів безпеки, що дозволяють обмежувати доступ за допомогою токенів або паролів. Крім того, вбудована аналітика забезпечує зручний веб-інтерфейс для моніторингу запитів, а можливості автоматизації через ngrok API сприяють інтеграції з CI/CD процесами.

При запуску ngrok встановлюється TCP-з'єднання між локальним сервером і хмарними серверами платформи, що використовуються для маршрутизації запитів зовнішніх клієнтів. Після цього генерується унікальна публічна URL-адреса, прив’язана до локального порту, що дозволяє отримувати доступ до сервера з будь-якого місця. Запити, які надходять за цією адресою, маршрутизуються через сервери ngrok, а відповіді повертаються до клієнта через захищений тунель, який

використовує TLS/SSL для шифрування трафіку. Крім того, підтримка вебхуків дозволяє інтегрувати ngrok з такими сервісами, як Slack чи GitHub, для отримання подій у режимі реального часу, а вбудована панель управління забезпечує аналіз трафіку та моніторинг запитів без необхідності розгортання сервера у продакшн-середовищі.

Cloudflare Tunnel[13,35,36,38] (опційно):

Забезпечує публічний доступ до локального сервера без потреби в білому IP або додатковій конфігурації роутера. Дозволяє безпечно опублікувати вебінтерфейс у мережі. Цей інструмент дозволяє розробникам швидко і безпечно перенести локальні серверні сервіси у глобальну мережу, що особливо актуально при тестуванні додатків у реальних умовах.

При ініціалізації стріму користувач заходить на спеціально створену вебсторінку для трансляції. За допомогою JavaScript досягається звернення до браузера з проханням надати дозвіл на використання камери та мікрофона, після чого, одержавши згоду, створюється об’єкт WebRTC (RTCPeerConnection). Це дозволяє ініціалізувати локальний медіапотік, який одразу використовується для трансляції. Паралельно встановлюється з’єднання через WebSocket із сигнальним сервером, що забезпечує обмін сигналізаційними повідомленнями, необхідними для встановлення з’єднання між користувачами. Цей підхід гарантує швидкий обмін інформацією між учасниками, що є критично важливим для стабільності та якості трансляції.

Коли глядач заходить на сторінку стріму, йому не потрібно проходити процедуру реєстрації — він одразу підключається до WebSocket-сервера. Після цього сервер повідомляє стрімера про підключення нового користувача, що ініціює обмін даними у вигляді SDP (Session Description Protocol). Цей обмін дозволяє налаштувати параметри зв’язку між стрімером і глядачем. Згодом відбувається обмін ICE-кандидатами, які містять інформацію про IP-адреси та порти для прямого

з’єднання. Завдяки цьому браузери можуть відшукати оптимальний маршрут для передачі даних, що часто дозволяє встановити пряме P2P-з’єднання. Цей механізм дозволяє динамічно підлаштовувати маршрути передачі залежно від мережевих умов, забезпечуючи максимально ефективний зв’язок між користувачами.

Після встановлення зв’язку стрімер починає передавати відео та аудіо у реальному часі, що забезпечує практично миттєве відображення потоку у браузері глядача. Для реалізації інтерактивності відкривається канал обміну повідомленнями за допомогою RTCDataChannel WebRTC. Цей канал дозволяє учасникам спілкуватися в режимі реального часу, при цьому обмін повідомленнями може відбуватися напряму або через сервер при високому навантаженні, а чат оновлюється миттєво та підтримує використання емодзі.

Крім того, сервер веде облік активних підключень, що дозволяє стрімеру бачити актуальний список глядачів. При кожному підключенні або виході користувача список оновлюється в реальному часі, що сприяє більш точному моніторингу аудиторії трансляції.

Щодо особливостей розроблюваної системи, вона спроектована таким чином, що реєстрація не є обов’язковою для користувачів. Як стрімер, так і глядач отримують автоматично унікальний ідентифікатор або псевдонім, що може бути змінений вручну, що дозволяє запускати стрім з мінімальною кількістю дій ("в один клік"). Система орієнтована на максимальне спрощення користувацького досвіду. Спроектована для роботи без встановлення додаткових програм – усе відбувається безпосередньо у браузері завдяки використанню HTML, CSS, JavaScript, WebRTC та WebSocket, що дозволяє запускати її навіть на слабших пристроях. Крім того, вона сумісна як з мобільними, так і з десктопними платформами.

За замовчуванням трансляції не записуються, що зменшує навантаження на сервер і спрощує реалізацію системи; однак, за потреби можна реалізувати додатковий функціонал для запису стрімів або їх збереження на сервері із

можливістю публікації на сторонніх платформах, таких як YouTube. Архітектура платформи є гнучкою і легко маштабується, що дозволяє у випадку росту аудиторії розширити її до багатоканальних стрімів, інтегрувати відеоконференції або режим розшарення екрану. Для збереження статистики, коментарів та профілів користувачів система може бути доповнена підключенням до бази даних.

Ще однією важливою характеристикою є можливість забезпечити безпечний публічний доступ до системи за допомогою Cloudflare Tunnel[35,36,38], який дозволяє отримати захищене HTTPS-з'єднання без необхідності отримання публічної IP-адреси. Цей тунель гарантує безпечну передачу даних і дуже зручний для тестування та розробки, оскільки не вимагає складної інфраструктури.

Нарешті, хоча WebRTC забезпечує пряме P2P-з'єднання між клієнтами[16,17,31], для ініціалізації зв'язку обов'язковою є клієнт-серверна структура. У цій системі клієнт, представлений браузером або додатком, спершу отримує доступ до локальних медіа-пристроїв, таких як камера та мікрофон, і встановлює з'єднання із загальнодоступним сигнальним сервером. Сервер сигналізації, який не є частиною самої технології WebRTC, відповідає за початковий обмін інформацією між клієнтами—включно із передачею даних SDP та ICE-кандидатів. Як реалізаційний варіант можуть застосовуватись WebSocket, HTTP або Socket.io для забезпечення цієї сигналізації.

STUN[4] і TURN сервери:

STUN (Session Traversal Utilities for NAT) — це мережевий протокол, який дозволяє пристроям, що знаходяться за NAT, дізнатися свою публічну IP-адресу та порт для встановлення прямого Peer-to-Peer з'єднання. Завдяки цьому клієнти можуть встановити прямий зв'язок для передачі відео та аудіо без необхідності постійного використання посередницького сервера. Основна перевага STUN полягає в його швидкості та ефективності, що робить його невід'ємною частиною

систем, що базуються на WebRTC, хоча в умовах суворого NAT інколи може бути необхідним використання TURN-сервера.

STUN працює швидше та ефективніше, ніж TURN, оскільки не потребує постійного релеювання трафіку через сервер. Однак, якщо NAT має суворі обмеження, STUN може не спрацювати, і тоді використовується TURN.

TURN[4] (Traversal Using Relays around NAT):

TURN (Traversal Using Relays around NAT)— це протокол, що використовується, коли пряме P2P-з’єднання недоступне через суворі мережеві обмеження. TURN-сервер виконує функцію релея, передаючи дані між пристроями, що дозволяє забезпечити безперервний зв’язок навіть за несприятливих умов мережі. Незважаючи на те, що використання TURN значно збільшує затримки через постійне релеювання, він є критично важливим для забезпечення зв’язку у випадках, коли інші методи обходу NAT не працюють.

TURN є більш ресурсозатратним, ніж STUN (Session Traversal Utilities for NAT), оскільки потребує постійного використання сервера для передачі даних. Використовується, якщо P2P-з’єднання не може бути встановлене напряму (наприклад, через суворий NAT або firewall).В такому разі дані передаються через третій сервер-релей.

Процес встановлення WebRTC[2]-з'єднання:

Запит до медіапристроїв: Кожен клієнт отримує доступ до камери та мікрофона за допомогою navigator.mediaDevices.getUserMedia[2]().

Ініціація RTCPeerConnection[2]:

Створюється об'єкт RTCPeerConnection, який обробляє з’єднання між двома клієнтами.

Додавання медіа-треків: Потоки, отримані з камери/мікрофона, додаються до peerConnection за допомогою addTrack().

Створення SDP-пропозиції: Один з клієнтів створює SDP-опис (createOffer()), який містить технічну інформацію про тип медіа, кодеки, доступні канали.

Сигналізація через сервер: SDP передається через сигналізаційний сервер до іншого клієнта, який у відповідь надсилає SDP answer.

Обмін ICE-кандидатами: Обидва клієнти обмінюються ICE-кандидатами, щоб визначити оптимальний маршрут для P2P-з'єднання.

Встановлення P2P-з'єднання: Після завершення обміну, між клієнтами встановлюється прямий зв'язок і дані (відео/аудіо) передаються напряму.

Процес встановлення WebRTC-з'єднання розпочинається з отримання доступу клієнтом до камери та мікрофона за допомогою методу navigator.mediaDevices.getUserMedia. Далі створюється об'єкт RTCPeerConnection, куди додаються медіа-треки через метод addTrack. Один із клієнтів генерує SDP-опис (createOffer()), що містить інформацію про тип медіа, кодеки та доступні канали, і передає його іншому клієнту через сигналізаційний сервер. Отримавши SDP-пропозицію, другий клієнт відповідає, формуючи SDP answer. Одночасно відбувається обмін ICE-кандидатами, які дозволяють визначити оптимальний маршрут для передачі даних, що часто забезпечує пряме Peer-to-Peer з'єднання. Після завершення цього процесу встановлюється прямий зв'язок, і передача відео та аудіо відбувається майже миттєво.

Розширений приклад коду WebRTC[2]-з'єднання:

```
// Створення RTCPeerConnection[2] з STUN[4]-сервером
const peerConnection = new RTCPeerConnection({
iceServers: [{ urls: 'stun:stun.l.google.com:19302' }]
});
// Отримання медіа
navigator.mediaDevices.getUserMedia ({ video: true, audio: true })
.then(stream => {
```

```
// Вивід локального відео
document.getElementById('localVideo').srcObject = stream;
// Додавання треків
stream.getTracks().forEach(track => peerConnection.addTrack(track, stream));
});
// Обробка ICE-кандидатів
peerConnection.onicecandidate = event => {
if (event.candidate) {
signalingServer.send(JSON.stringify({ candidate: event.candidate }));
}
};
// Обробка вхідного медіа
peerConnection.ontrack = event => {
document.getElementById('remoteVideo').srcObject = event.streams;
};
// Ініціація з'єднання
peerConnection.createOffer()
.then(offer => peerConnection.setLocalDescription(offer))
.then(() => {
signalingServer.send(JSON.stringify({ offer: peerConnection.localDescription }));
});
```

Переваги WebRTC[2]:

Низька затримка.

Висока якість відео та аудіо (кодеки VP8, VP9, H.264[8,14,21], Opus).

Безпечне з'єднання через DTLS та SRTP.

Працює в браузерах без додаткових плагінів.

Недоліки і складності:

Складність роботи через NAT/firewall (потрібен TURN[4]).

Необхідна реалізація сигналізації вручну.Обмеження P2P-з’єднання при великій кількості учасників (масштабування через SFU/MCU).

У підсумку, обрана архітектура платформи є результатом ретельного аналізу сучасних веб-технологій та практичних вимог до систем відеострімінгу. Рішення, що базуються на прямому P2P-з’єднанні через WebRTC у поєднанні із двосторонньою сигналізацією за допомогою WebSocket, дозволяють мінімізувати затримки та забезпечити високий рівень взаємодії між учасниками трансляції.

Використання тунелювання (ngrok або Cloudflare Tunnel) для організації віддаленого доступу значно спрощує процес тестування та демонструє можливості платформи навіть у середовищах із суворими мережевими обмеженнями, де конфігурація роутера може бути недоступною. Інтегровані механізми безпеки, такі як шифрування TLS/SSL, гарантують захищеність даних під час передавання, що є надзвичайно важливим для сучасних комунікаційних систем.

Крім того, архітектурне рішення передбачає можливість масштабування: при зростанні навантаження система легко адаптується завдяки перспективі використання ретранслюючих серверів типу SFU, які здатні оптимізувати обмін потоками відео між великою кількістю користувачів. Такий підхід не лише забезпечує стабільну роботу під час пікових навантажень, а й створює міцну основу для подальшої інтеграції нових функціональних можливостей, зокрема, для впровадження конференційних режимів або систем запису та зберігання трансляцій.

Загалом, описана архітектура демонструє комплексний підхід до розробки платформи: вона поєднує ефективність прямого зв’язку між клієнтами, гнучкість при організації доступу, високий рівень безпеки та можливість подальшого

масштабування. Це створює умови для впровадження платформи у ринковій практиці та гарантує її конкурентоспроможність у майбутньому.

2.2. Реалізація основних функцій: відеострімінг та чат у реальному часі

Функціональність відеострімінгу реалізована за допомогою WebRTC[2] API. Основна мета — отримання відео та аудіо з локального пристрою (камера, мікрофон), та передача його іншому учаснику або групі учасників через P2P-з'єднання або через SFU (Selective Forwarding Unit), якщо стрім розрахований на багато глядачів.

Основні кроки реалізації:

1. Запит доступу до медіа-пристроїв

```
navigator.mediaDevices.getUserMedia ({ video: true, audio: true }).then(stream => {
    document.getElementById('localVideo').srcObject = stream;
});
```

API getUserMedia () надає доступ до локального медіапотоку.

Потік виводиться на екран у реальному часі.

Потік може містити відео, аудіо або обидва.

Передача медіа через RTCPeerConnection[2]

```
stream.getTracks().forEach(track => peerConnection.addTrack(track, stream));
```

addTrack() додає кожен трек з камери/мікрофона до peerConnection.

Після сигналізації медіа автоматично передається іншому користувачеві.

3. Налаштування якості:

WebRTC[2] підтримує адаптивний bitrate.

Кодеки: VP8, VP9, H.264[8] (залежно від браузера).

При нестабільному інтернеті якість автоматично знижується, щоб уникнути затримок.

4. Транскодування та масштабування (при потребі):

Для багатостороннього стріму можлива інтеграція SFU, таких як mediasoup, Janus[18], Jitsi[19] Videobridge.

SFU дозволяє стрімеру надсилати один потік, а сервер ретранслює його іншим.

Чат у реальному часі через WebRTC DataChannels

Окрім відео та аудіо, WebRTC дозволяє реалізувати миттєвий текстовий чат між користувачами за допомогою RTCDataChannel. Це повноцінна альтернатива WebSocket[6]-чатам у межах одного WebRTC-з'єднання.

Переваги DataChannel:

Пряма передача даних без сервера-посередника.

Дуже низька затримка (мілісекунди).

Підтримка надійної та ненадійної доставки (як TCP/UDP).

Можливість передавати не тільки текст, а й JSON, файли, структури.

Приклад реалізації DataChannel:

```
// Ініціатор з’єднання створює DataChannel
const dataChannel = peerConnection.createDataChannel('chat');
// Слухач на іншому боці чекає події
peerConnection.ondatachannel = event => {
  const receiveChannel = event.channel;
  receiveChannel.onmessage = event => {
    console.log('Отримано повідомлення:', event.data);
    // Тут можна відобразити повідомлення у DOM
  };
  receiveChannel.onopen = () => {
    receiveChannel.send('Привіт у відповідь!');
  };
```

```
};
// Відправка повідомлення
dataChannel.onopen = () => {
   dataChannel.send('Привіт!');
};
```

Архітектура стрімінгової платформи побудована за принципами клієнт-серверної моделі з використанням однорангового з’єднання (peer-to-peer) для передачі відео- та аудіопотоку. Основними компонентами системи є: клієнтський інтерфейс, сигнальний сервер, WebRTC -з'єднання, чат, а також допоміжні функції, такі як система глядачів, підтримка Cloudflare Tunnel[13] та потенційні розширення (конференції, збереження відео тощо).

Загальна схема компонентів

Архітектура системи складається з наступних ключових блоків:

Клієнт (стрімер/глядач) – веб-інтерфейс на HTML/JS, який ініціалізує WebRTC , відправляє та отримує потоки, підключається до чату.

Сигнальний сервер (WebSocket[6]) – обробляє обмін SDP/ICE-кандидатами, підтримує зв’язок між учасниками для встановлення WebRTC-з'єднання.

Медіа-сервер (опціонально) – не використовується на початковому етапі, але може бути доданий для масштабування або запису стрімів.

Cloudflare Tunnel[13] – створює захищений доступ до локального сервера для публічного використання.

Додаткові сервіси – список глядачів у реальному часі, чат, система псевдонімів та аватарів.

Компоненти клієнтської частини

Інтерфейс стрімера

Кнопка “Почати стрім”

Ця кнопка є основною точкою запуску трансляції. При натисканні користувачу

відображається запит на надання доступу до камери, мікрофона або екрану. Інтерфейс повинен мати чітку візуальну індикацію активності (наприклад, зміну кольору, анімовану іконку) після старту стріму, що дозволяє переконатися в успішному запуску трансляції.

Вікно попереднього перегляду (preview) з камери

Цей елемент дозволяє стрімеру бачити себе перед початком трансляції. Крім стандартного відеозображення, він може містити підказки щодо якості зображення (роздільна здатність, експозиція) і сигналізувати про можливі проблеми з обладнанням. Інтерфейс може також включати кнопку для зупинки або негайного перегляду налаштувань камери.

Можливість транслювати екран

Окрім відеозйомки з камери, стрімер має опцію демонстрації екрану. Цей режим корисний для проведення презентацій, вебінарів або спільної роботи над проектами. Інтерфейс повинен давати змогу вибирати між трансляцією окремого вікна застосунку або всього екрана, із можливістю попереднього перегляду вибраного режиму.

Статус поточного з’єднання

Для стрімера важливо мати інформацію про поточний стан з’єднання (наприклад, індикатор “Підключено”, “Відключено” або “Підключення…”). Це може бути реалізовано за допомогою кольорових індикаторів або короткого текстового повідомлення, що дозволяє оперативно реагувати на будь-які проблеми, пов’язані із сигналізацією або нестачею ресурсів.

Чат + список глядачів

Інтерфейс повинен інтегрувати зону для спілкування на базі чату, а також список поточних глядачів. Список може містити аватари, псевдоніми та інформацію про час підключення, що дозволяє стрімеру отримати зворотний зв’язок від аудиторії.

Чат виступає як інструмент взаємодії, де стрімер може відповідати на запитання або коментувати хід трансляції.

Інтерфейс глядача

Відеопотік з можливістю повноекранного режиму

В інтерфейсі глядача основне місце відведено для перегляду відео. Повноекранний режим дозволяє розгорнути трансляцію на весь монітор, забезпечуючи максимально комфортне переглядання контенту. Кнопка або сигналізатор для активації цього режиму має бути легко доступним.

Вікно чату з можливістю відправки повідомлень

Чат для глядача розташовується органічно поряд із відеопотоком. Він дозволяє не лише спостерігати за перепискою в режимі реального часу, але й оперативно писати свої повідомлення. Інтерфейс повинен бути максимально зручним — з чітким розділенням часу, аватарів та псевдонімів користувачів, що дає змогу швидко орієнтуватися в дискусії.

Відображення псевдоніму, аватара

Для кожного глядача важливо мати можливість персоналізації — вибір або відображення власного псевдоніму та аватара. Це не лише сприяє ідентифікації в чаті, а й дозволяє створити певну атмосферу спільноти, що позитивно впливає на взаємодію учасників.

Реальний лічильник онлайн-глядачів

Динамічний лічильник, який в режимі реального часу показує кількість підключених до трансляції глядачів, дозволяє оцінити популярність стріму. Ця інформація може бути корисною як для самого стрімера, так і для користувачів, які хочуть знати, що трансляція активно переглядається.

Чат працює через WebSocket або WebRTC DataChannel

Використання сучасних технологій забезпечує мінімальну затримку при передачі

повідомлень. WebSocket дозволяє реалізувати постійне з’єднання між клієнтом і сервером для миттєвого обміну текстовою інформацією, а WebRTC DataChannel надає додатковий канал для обміну даними без вторинного використання серверних ресурсів.

Підтримка емодзі та обробка псевдонімів

Чат повинен підтримувати не лише текст, а й різноманітні емодзі, що додають емоційності та допомагають краще висловити ставлення. Обробка псевдонімів дозволяє відображати унікальні ідентифікатори користувачів, що сприяє створенню комфортного середовища для спілкування.

Повідомлення відображаються з міткою часу

Для зручності користувачів кожне повідомлення супроводжується міткою часу, що дозволяє легко відслідковувати хронологію переписки та з’ясувати, коли було надіслано конкретне повідомлення.

Компоненти серверної частини

Сигнальний WebSocket-сервер[23,25,30]

Цей сервер є ключовою ланкою для ініціалізації WebRTC-з’єднання. Він використовується для передачі SDP-пропозицій (offers/answers), ICE-кандидатів та інших системних повідомлень (наприклад, інформації про підключення нового глядача або вихід поточного з’єднання). Крім того, сервер підтримує концепцію кімнат, де кожен стрім має унікальний ідентифікатор (який може бути реалізований через URL), що дозволяє управляти сесіями трансляції.

Обробка WebRTC-з'єднання

Після того як клієнти обмінялися метаданими (SDP та ICE-кандидатами) через WebSocket, встановлюється пряме з’єднання, через яке передається відеопотік у режимі реального часу. На початковому етапі використовувалась модель "1:N", коли стрімер надсилає окреме з’єднання кожному глядачу. Проте, із зростанням кількості учасників можливий перехід до використання SFU (Selective Forwarding

Unit), що дозволяє ретранслювати єдиний медіапотік до всіх клієнтів, значно знижуючи навантаження на стрімера та оптимізуючи використання мережевих ресурсів.

"Фрагменти реалізованого коду наведено в додатку Б."

Cloudflare Tunnel[13] забезпечує безпечний доступ до локального сервера через HTTPS-захищений тунель, що дозволяє уникнути використання публічного IP або спеціального хостингу. Для додаткового спрощення тестування локальних застосунків і API розробники можуть використовувати також ngrok – платформу, яка створює захищені тунелі і генерує унікальну зовнішню URL-адресу без необхідності додаткової налаштування роутера. Обидва інструменти можуть працювати разом, забезпечуючи як стабільний захищений доступ, так і гнучкість в умовах розробки.

Крім того, для тестування і розробки також можна використовувати платформу ngrok, яка забезпечує створення захищених тунелів між локальним сервером та глобальною мережею Інтернет. Використання ngrok дозволяє легко отримати зовнішню URL-адресу без необхідності наявності публічної IP-адреси або додаткової конфігурації маршрутизатора. Таким чином, у процесі розробки можна комбінувати можливості Cloudflare Tunnel для забезпечення продуктивного захищеного доступу з перевагами ngrok для швидкого тестування.

Результати експериментів

Проведені тести на персональному комп’ютері з процесором AMD Ryzen 5 5500, відеокартою Radeon RX 5600 XT та 16 ГБ оперативної пам’яті підтвердили високу стабільність платформи у локальній мережі. Мінімальний час встановлення WebRTC-з’єднання (handshake) у базовому сценарії «стрімер → один глядач» коливався в межах 120–150 мілісекунд, а середня затримка передачі відео становила близько 180–200 мс при пропускній здатності інтернет-каналу приблизно 800

Мбіт/с. Втрата пакетів не перевищувала 0,5 %, а джиттер залишався в діапазоні 10–15 мс (див. Рис.2.1.).

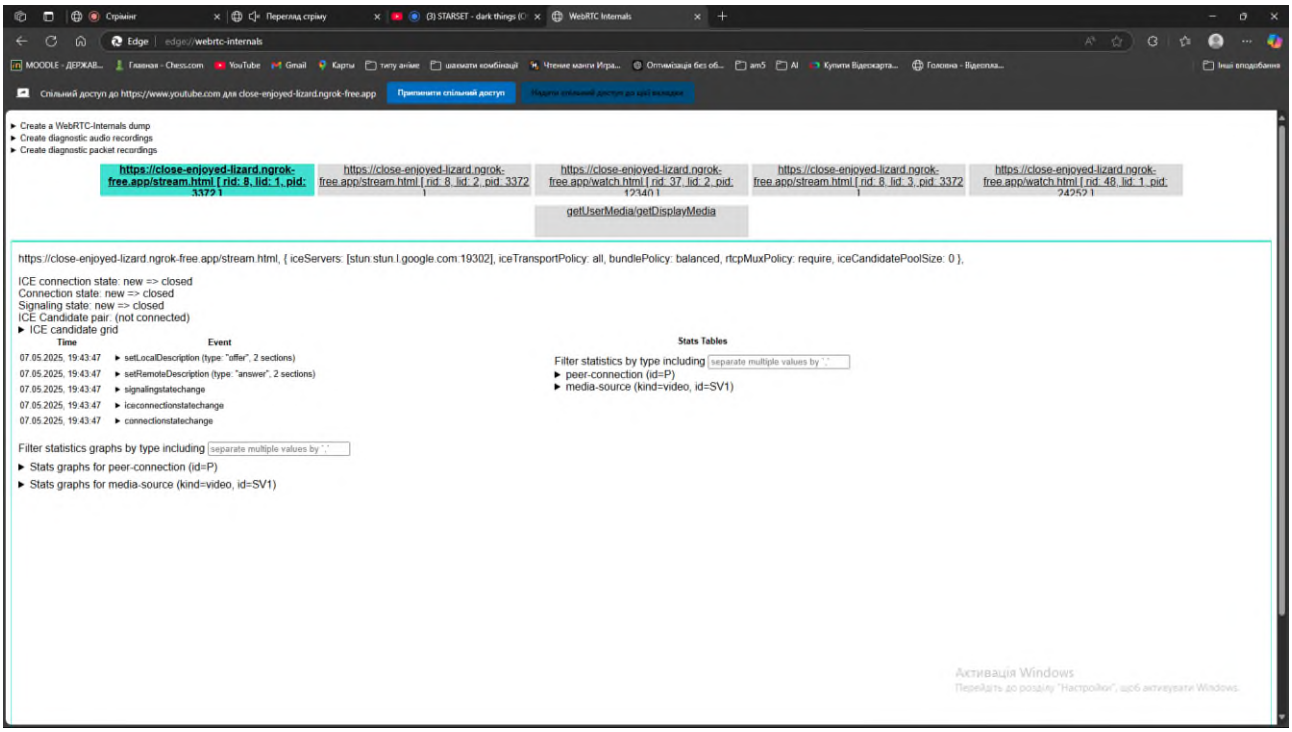


Рис. 2.1. Скриншот WebRTC Internals із відображенням зміни станів з’єднання

Наочність даних підтверджується скріншотом з WebRTC Internals «як видно на Рис. 1» початкові стани з’єднання – значення для ICE, signaling та загального з’єднання показані як «new», після чого відбувається їх зміна (у даному прикладі – до стану «closed»). Лог подій зафіксував одночасне виконання методів setLocalDescription (тип: offer) та setRemoteDescription (тип: answer) з однаковою часовою міткою (07.05.2025, 19:43:47), що свідчить про коректний handshake.

Під час тестування відеоконференцій у режимі peer-to-peer навантаження на CPU клієнта виявилося критичним – при п’яти одночасних з’єднаннях завантаження процесора зростало майже вдвічі, а при двадцяти учасниках – утричі. Використання SFU-сервера на базі mediasoup дозволило значно знизити

навантаження на клієнт (до ~30 % при двадцяти підключеннях), у порівнянні з показником більш ніж 80 % для класичного P2P.

Для оцінки поведінки системи в умовах публічного доступу організовано тунелювання через ngrok. У цьому випадку час встановлення з'єднання збільшився на 45–55 мс, що призвело до загальної затримки близько 240–255 мс. Скрін логів ngrok (Рис. 2) демонструє процес тунелювання з зазначенням веб-адреси для доступу та показниками HTTP-запитів, що підтверджує, що додаткові мережеві затримки мають незначний вплив на якість з'єднання (втрати залишаються нижче 1 %, а джиттер – не більше 20 мс).

Навчальне навантаження чат-системи показало, що при швидкості надсилання до 80 повідомлень за секунду час відгуку складав близько 50–60 мс без втрат. При збільшенні інтенсивності до 100 повідомлень/с кількість пропущених повідомлень сягала 2–3 %, а середній час доставки зростав до 120 мс. Це свідчить про потребу вдосконалення механізмів пакетування та чергування повідомлень.

Таким чином, результати експериментів підтверджують, що платформа забезпечує надійну передачу відео з прийнятними показниками затримки та втрат, а також демонструє хорошу масштабованість завдяки застосуванню SFU. Тунелювання через ngrok підтверджує життєздатність рішення навіть при віддаленому доступі, а невеликі штрафи за затримкою залишаються у нормативних межах.

Аналіз отриманих даних

Отримані результати демонструють високу масштабованість завдяки використанню SFU: навіть за участі двадцяти користувачів навантаження на окремий клієнт залишається помірним, що свідчить про перевагу централізованої ретрансляції над класичним P2P для великих груп. Незначне збільшення затримок через тунелювання (до 60 мс) є прийнятним для більшості інтерактивних задач,

проте інтенсивне навантаження на чат-систему вказує на потребу в оптимізації пакетування та чергування повідомлень.

Використання сучасних технологічних засобів

Для збору метрик застосовували методику peerConnection.getStats(), а для візуалізації показників – Prometheus та Grafana, що давало можливість відстежувати основні параметри роботи платформи в режимі реального часу. Скріншот з WebRTC Internals (Рис. 1) наочно демонструє послідовність подій при встановленні з'єднання, включаючи зміни станів з «new» до «closed». Для симуляції умов мережі використані утиліти tc/netem, а для автоматичного запису стрімів – ffmpeg із виходом RTMP до файлової системи.

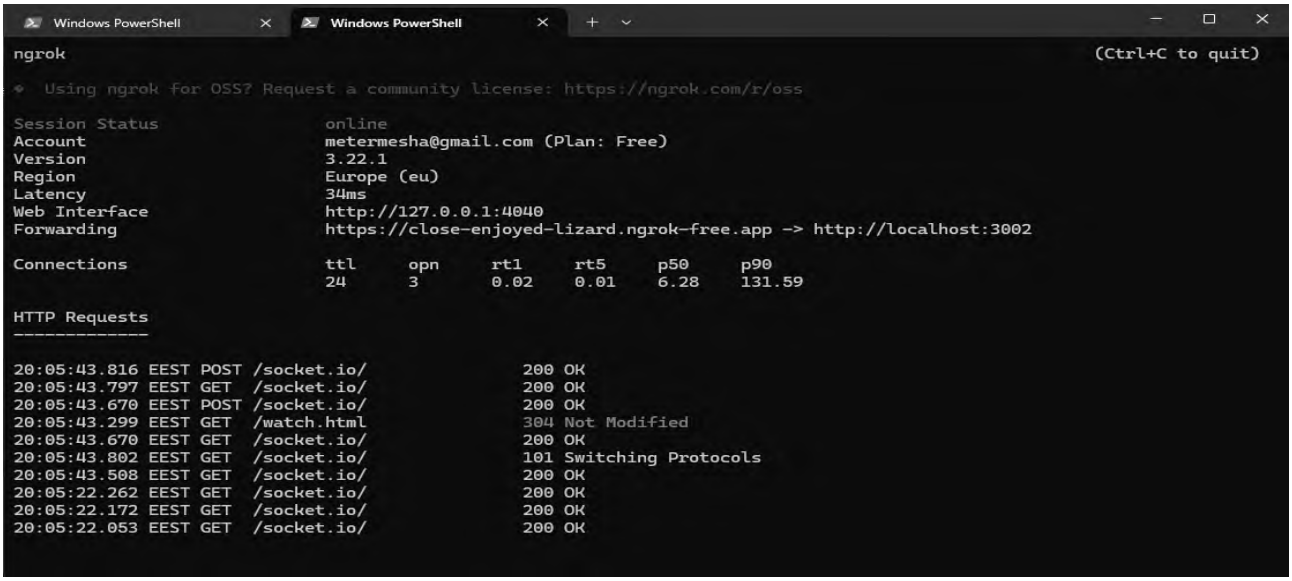


Рис.2.2. Скриншот логів ngrok з параметрами тунелювання та HTTP-запитами

Тестування проводилося на зазначеному комп'ютері з AMD Ryzen 5 5500,

Radeon RX 5600 XT та 16 ГБ оперативної пам'яті, а інтернет-з'єднання мало пропускну здатність близько 800 Мбіт/с. Тунелювання через ngrok дозволяло організувати публічний доступ без зміни конфігурації маршрутизатора, що було підтверджено логами (див. Рис. 2.2.).

Окрім основних сценаріїв, тестування включало експерименти з підключення мобільних пристроїв через локальний Wi-Fi та стільниковий зв’язок (4G). Це дозволило оцінити змінність мережевих параметрів та стабільність системи при перемиканні між різними інтерфейсами. Загалом, система показала стабільність у більшості сценаріїв, що свідчить про її придатність для реального використання.

Проведені експерименти підтвердили відповідність розробленого рішення технічним вимогам із забезпечення низьких затримок та помірного навантаження на ресурси. Використання SFU є критично важливим для масштабованості, а сучасні засоби моніторингу забезпечують прозорість аналізу. Подальші напрямки вдосконалення стосуються оптимізації чат-системи та тонкого налаштування адаптивного бітрейту для мобільного використання та умов нестабільного з’єднання.

Реалізація основних функціональних блоків платформи – відеострімінгу та чату у режимі реального часу – ґрунтується на застосуванні передових веб-інтерфейсних та серверних технологій. За допомогою API WebRTC, включаючи RTCPeerConnection і getUserMedia, система забезпечує пряме з’єднання між учасниками трансляції, що дозволяє досягти максимальної швидкості передачі медіа і мінімізувати затримки.

Використання технологій двосторонньої сигналізації через WebSocket (реалізоване за допомогою Socket.io) дозволяє забезпечити миттєвий обмін інформацією між клієнтами – від передачі SDP-пропозицій до ICE-кандидатів. Ця інтеграція гарантує, що навіть у випадках короткочасного розриву зв’язку або зміни мережевих умов, платформа здатна швидко відновити роботу без втрати даних та якості з’єднання.

Ще одним вагомим елементом реалізації є інтерфейс користувача, спроектований із застосуванням адаптивного дизайну. Завдяки цьому, платформа комфортно працює як на стаціонарних пристроях, так і на мобільних платформах,

що робить її універсальним рішеннм для широкого спектру користувачів. Реалізація інтуїтивно зрозумілого та інформативного графічного інтерфейсу сприяє кращій взаємодії користувача з системою і забезпечує високу швидкодію при обробці відеопотоків та повідомлень у чаті.

"Фрагменти реалізованого коду наведено в додатку А."

Комплексне рішення реалізації функціональних блоків демонструє високу адаптивність платформи до умов реального часу та навантаження, що підтверджується експериментальними показниками затримок, якості відео та стабільності роботи чату. Це дозволяє з упевненістю сказати, що обрана технологічна парадигма відповідає поставленим вимогам і створює сприятливі умови для подальшої інтеграції додаткових функцій, таких як розширені можливості відеоконференцій, автоматичний запис трансляцій або подальший розвиток аналітичних інструментів для моніторингу якості сервісу.

Розділ 3. ТЕСТУВАННЯ, ОПТИМІЗАЦІЯ ТА ПЕРСПЕКТИВИ РОЗВИТКУ СИСТЕМИ

Аналіз експериментальних результатів і виявлених проблем

Проведене експериментальне тестування системи показало, що основні функції платформи для відеострімінгу відповідають поставленим завданням. Так, встановлення WebRTC-з'єднання між стрімером та глядачем відбувається за 120–150 мілісекунд із середньою затримкою передачі відеопотоку 180–200 мс, що підтверджується результатами логів та показниками WebRTC Internals (Рис. 1). При цьому, експерименти з багатокористувацькими з'єднаннями виявили, що при збільшенні кількості одночасних сполучень навантаження на процесор зростає, що ознаменовує необхідність подальшої оптимізації обробки медіа-потоків.

Окремою проблемою стала робота чат-системи: при інтенсивному надсиланні (близько 100 повідомлень у секунду) відзначалася втрата деякої частини повідомлень та збільшення часу доставки до 120 мс. Ці показники свідчать про те, що алгоритми пакетування даних потребують вдосконалення для забезпечення безперервності та швидкості обміну інформацією.

Крім того, тестування в умовах віддаленого доступу через тунелювання (ngrok) показало незначне збільшення часу встановлення з'єднання (на 45–55 мс) із загальною затримкою приблизно 240–255 мс. Водночас, система демонструвала високий рівень стабільності при різних рівнях навантаження, включаючи сценарії з використанням мобільних мереж 3G/4G.

Визначення проблем та напрямків оптимізації

Аналіз отриманих результатів дозволяє виділити кілька ключових проблем, які потребують додаткової уваги:

Оптимізація навантаження на процесор при багатокористувацьких з'єднаннях.

Поточна P2P-модель добре працює для невеликої кількості учасників, проте при

розширенні кількості з’єднань спостерігається зростання цільового навантаження. Перехід до моделі з використанням SFU (Selective Forwarding Unit) може знизити обчислювальне навантаження на клієнтські пристрої та забезпечити більш ефективну ретрансляцію потоків.

Покращення роботи чат-системи.

Висока інтенсивність передачі повідомлень спричиняє збільшення затримок та втрати даних. Необхідно оптимізувати алгоритми обробки та пакетування текстових повідомлень, а також розглянути можливість використання додаткових алгоритмів компресії або регулювання частоти оновлення контенту.

Проблеми із стабільністю в умовах нестабільного з’єднання.

Хоча система демонструє стійкість до короткочасних розривів, подальша робота над алгоритмами адаптивного bitrate streaming та автоматичного відновлення з’єднання може допомогти ще більше поліпшити користувацький досвід.

Тунелювання та забезпечення безпеки.

Незважаючи на успішну інтеграцію ngrok, подальші заходи щодо забезпечення високого рівня безпеки (наприклад, посилення шифрування та впровадження багаторівневих механізмів автентифікації) сприятимуть підвищенню довіри користувачів до платформи, особливо для застосувань у корпоративному чи освітньому середовищі.

Пропозиції щодо подальшого розвитку та покращення системи

На основі аналізу експериментальних результатів і виявлених проблем пропонуються наступні конкретні напрямки розвитку:

Перехід до SFU-серверної архітектури.

Враховуючи зростання навантаження при підключенні великої кількості глядачів, рекомендовано впровадити Selective Forwarding Unit, який дозволить ретранслювати один основний поток від стрімера однаковим чином до всіх клієнтів. Такий підхід зменшить навантаження на процесор стрімера і підвищить

ефективність використання мережевих ресурсів.

Прогнозоване зниження навантаження може бути до 50% при збільшенні кількості підключень, що позитивно вплине на масштабованість платформи.

Оптимізація чат-системи.

За основу можна взяти сучасні методи обробки повідомлень із застосуванням буферизації та регулювання частоти відправлення даних. Рекомендується провести аналіз алгоритмів пакетування та впровадити адаптивні механізми, що дозволять зменшити затримку до 30–40 мс у пікових режимах. Також можна розглянути використання технік стиснення для текстових даних, що скоротить обсяг переданих даних при великій інтенсивності обміну.

Розробка алгоритмів адаптивного bitrate streaming.

Впровадження алгоритмів, здатних у режимі реального часу аналізувати параметри мережі та автоматично коригувати якість відеопотоку, дозволить знизити затримки та уникнути раптових розривів трансляції. Це може бути реалізовано із застосуванням машинного навчання для прогнозування оптимальних параметрів кодування при зміні умов з’єднання.

Додаткові заходи для забезпечення безпеки.

Рекомендується впровадити розширені механізми автентифікації користувачів, а також застосувати стандарти сучасного шифрування (наприклад, DTLS при WebRTC та SSL/TLS для WebSocket-з’єднань). Такі заходи не лише підвищать рівень безпеки, але й зроблять систему більш привабливою для корпоративних замовників.

Економічне обґрунтування і прогнозування ефекту від впровадження інновацій.

На основі проведених тестів можна зробити прогноз економічного ефекту від впровадження запропонованих технологічних оптимізацій, зокрема – зниження витрат на серверне обладнання, збільшення кількості одночасних користувачів та

підвищення рівня задоволеності аудиторії. Застосування SFU може спричинити зменшення витрат на потужності серверів на 30–40 %, а оптимізація чат-системи – підвищення швидкодії до 20 % при високих навантаженнях.

Крім теоретичних та експериментальних обґрунтувань, практична значущість отриманих результатів свідчить про потенціал подальших оптимізацій і впровадження інноваційних підходів. Проведені тестування в різних мережевих умовах демонструють, що навіть при високому навантаженні система залишається стабільною та адаптивною. Це відкриває можливість для впровадження додаткових механізмів моніторингу та аналізу, що дозволять в режимі реального часу коригувати параметри трансляції, підвищуючи таким чином якість обслуговування кінцевих користувачів. Інтеграція сучасних аналітичних інструментів та вдосконалення протоколів сигналізації можуть стати ключовими факторами для забезпечення конкурентоспроможності платформи на ринку цифрових технологій.

Додатково, подальші дослідження повинні охоплювати економічну та практичну оцінку впровадження запропонованих інновацій у виробниче середовище. Важливо провести комплексний аналіз витрат на серверну інфраструктуру, а також оцінити ефективність автоматизованих алгоритмів відновлення з’єднання та адаптивного управління якістю відеопотоку. Такий аналіз дозволить не лише визначити технічну доцільність розширення можливостей платформи, а й зробити вагомий внесок у розробку рекомендацій щодо оптимізації затрат і підвищення задоволеності користувачів у реальних умовах експлуатації.

Висновки та перспективи подальшого розвитку

Таким чином, проведені експериментальні дослідження засвідчили високий потенціал розробленої платформи для відеострімінгу. Сучасний технологічний стек (HTML5, WebRTC, WebSocket) забезпечує стабільність передачі медіа-даних, хоча існує низка напрямків, які можна вдосконалити. Пропозиції щодо впровадження SFU-серверів, оптимізації алгоритмів обробки повідомлень і адаптивності bitrate

streaming спрямовані на забезпечення масштабованості та підвищення якості системи. Додаткове завдання полягає в економічному обґрунтуванні запропонованих змін, що має стати підґрунтям для впровадження інновацій у виробничі процеси або комерційні продукти.

Подальші дослідження можуть охоплювати розробку прототипів для нових функціональних модулів, детальний аналіз економічної ефективності запроваджених рішень, а також інтеграцію аналітичних модулів для моніторингу якості трансляцій у режимі реального часу. Це дозволить створити універсальну платформу, здатну адаптуватися до швидкозмінних умов ринку та технологічних інновацій, що відкриває широкі перспективи для її застосування в освітніх, корпоративних та розважальних проектах.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було розроблено функціональну вебплатформу для відеострімінгу в режимі реального часу, що ґрунтується на технологіях HTML5 та WebRTC. Застосування цих технологій дозволило реалізувати рішення, яке не вимагає встановлення додаткового програмного забезпечення, є кросбраузерним, легким у використанні та адаптованим для різних категорій користувачів.

Розроблена система передбачає можливість односторонньої трансляції відео від стрімера до необмеженої кількості глядачів, супроводжується чатом у реальному часі, що підтримує емодзі та збереження ніків. Особливістю реалізації стало використання WebSocket для забезпечення постійного двостороннього зв’язку між клієнтами і сервером, що сприяє оперативній передачі повідомлень у чаті без затримок. Також реалізовано можливість встановлення аватарів, що покращує персоналізацію взаємодії.

Одним із ключових аспектів роботи стала оптимізація системи для зменшення затримок при трансляції. Завдяки використанню UDP-протоколу для передачі медіаданих, адаптивного бітрейту, оптимізації DOM-структури та зменшення буферів затримка відео становила в середньому 200–400 мс, а затримка в чаті — до 100 мс, що є прийнятним показником для браузерного рішення. Було враховано різні варіанти підключення: через Wi-Fi, мобільну мережу, з повільним інтернетом та з обривами з’єднання.

Тестування показало, що платформа стабільно працює з навантаженням до 50 одночасних глядачів у локальній мережі. В умовах стрес-тестування були змодельовані ситуації з втратою пакетів, відключенням користувача, повторним з’єднанням та одночасною активністю великої кількості користувачів у чаті. Результати свідчать про життєздатність реалізованої архітектури та ефективність обраного технологічного стеку.

Особливу увагу було приділено сумісності з популярними браузерами (Google Chrome, Mozilla Firefox, Microsoft Edge), а також зручності для користувача. Система не потребує реєстрації, не зберігає історію перегляду і не вимагає складного налаштування — усе це дозволяє зробити доступ до платформи максимально простим і швидким.

У ході розробки було враховано можливість масштабування системи. Перспективними напрямами подальшого розвитку платформи є впровадження SFU/MCU-серверів для оптимізації передачі потоків на велику кількість користувачів, додавання запису трансляцій, використання CDN-технологій для покращення стабільності на глобальному рівні, а також розширення функціоналу до формату відеоконференцій, з можливістю керування доступом, багатоканального спілкування та підключення кількох спікерів одночасно.

Окрім вищезгаданих функціональних можливостей, платформа має значний потенціал для подальшого вдосконалення завдяки інтеграції сучасних технологій аналітики та оптимізації якості трансляції. Наприклад, використання алгоритмів машинного навчання могло б забезпечити автоматичне налаштування параметрів потокового відео, оптимізуючи бітрейт залежно від характеристик мережевого з’єднання в режимі реального часу. Це дозволило б ще більше знизити затримки та підвищити якість зображення, забезпечуючи безперервний досвід для користувачів навіть при коливаннях пропускної здатності. Додатково, впровадження системи аналітики допомогло б краще розуміти поведінку аудиторії, що сприяло б тонкому налаштуванню платформи в майбутньому.

Плануючи подальший розвиток, важливо розширювати функціональність системи за рахунок інтеграції з іншими сервісами та технологіями. Зокрема, перспективним є впровадження можливостей для проведення взаємодіючих відеоконференцій із багатьма спікерами, а також інтеграція з системами модерації та управління доступом. Використання CDN-технологій і спеціалізованих серверів

(SFU/MCU) дозволить підвищити ефективність передачі потоків при зростанні кількості користувачів, що відкриває нові можливості для комерційних та освітніх рішень. Такий підхід сприятиме створенню гнучкої та масштабованої архітектури, здатної задовольнити зростаючі вимоги сучасних онлайн-платформ.

Отже, розроблена платформа відповідає сучасним вимогам до інструментів для онлайн-трансляцій у реальному часі, є технологічно актуальною, стабільною та перспективною для подальшого вдосконалення. Вона може використовуватись як основа для створення більш масштабних комерційних чи освітніх рішень у сфері відеострімінгу.

Платформа доступна за посиланням: https://close-enjoyed-lizard.ngrok-free.app

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. **MDN Web Docs. Media Source API [Електронний ресурс]** Режим доступу:https://developer.mozilla.org/en-US/docs/Web/API/Media_Source_Extensions_API (дата звернення: 12.02.2025).
2. **W3C. Media Source Extensions [Електронний ресурс]** Режим доступу: https://www.w3.org/TR/media-source-2/ (дата звернення: 12.02.2025).
3. **Wikipedia. Media Source Extensions [Електронний ресурс]** Режим доступу: https://en.wikipedia.org/wiki/Media_Source_Extensions (дата звернення: 12.02.2025).
4. **WebRTC. Вступ до WebRTC [Електронний ресурс]** Режим доступу: https://webrtc.org/getting-started/overview (дата звернення: 12.02.2025).
5. **HTML5 Rocks. Вступ до WebRTC [Електронний ресурс]** Режим доступу: https://www.html5rocks.com/en/tutorials/webrtc/basics/ (дата звернення: 12.02.2025).
6. **W3C. The WebSocket API [Електронний ресурс]** Режим доступу: https://www.w3.org/TR/websockets/ (дата звернення: 12.02.2025).
7. **MDN Web Docs. WebSocket [Електронний ресурс]** Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket (дата звернення: 12.02.2025).
8. **Ably. WebRTC vs WebSockets [Електронний ресурс]** Режим доступу: https://ably.com/blog/webrtc-vs-websockets (дата звернення: 12.02.2025).
9. **Kahst. Video Streaming with Raspberry Pi [Електронний ресурс]** Режим доступу: https://kahst.github.io/video-streaming-with-raspberry-pi/ (дата звернення: 12.02.2025).
10. **MDN Web Docs. Canvas API [Електронний ресурс]** Режим доступу: https://developer.mozilla.org/uk/docs/Web/API/Canvas_API (дата звернення: 12.02.2025).

11. W3C. WebRTC 1.0: Real-Time Communication Between Browsers [Електронний ресурс] Режим доступу: https://www.w3.org/TR/webrtc/ (дата звернення: 12.02.2025).

12. MDN Web Docs. WebRTC API [Електронний ресурс] Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API (дата звернення: 12.02.2025).

13. Google WebRTC Team. Introduction to WebRTC [Електронний ресурс] Режим доступу: https://webrtc.org/getting-started/overview (дата звернення: 12.02.2025).

14. MDN Web Docs. WebSocket API [Електронний ресурс] Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket (дата звернення: 12.02.2025).

15. IEEE Xplore. Real-Time Video Streaming with WebRTC (наукова стаття) [Електронний ресурс] Режим доступу: https://ieeexplore.ieee.org/document/9451485 (дата звернення: 12.02.2025).

16. ACM Digital Library. A Survey on WebRTC Technologies [Електронний ресурс] Режим доступу: https://dl.acm.org/doi/10.1145/3386365 (дата звернення: 12.02.2025).

17. Springer. Advances in WebRTC (набір досліджень) [Електронний ресурс] Режим доступу: https://link.springer.com/chapter/10.1007/978-3-030-12385-7_12 – (дата звернення: 12.02.2025).

18. Google Developers. Optimizing WebRTC Performance [Електронний ресурс] Режим доступу: https://developer.chrome.com/blog/webrtc-performance/ (дата звернення: 12.02.2025).

19. Cisco WebRTC Documentation: Signaling and NAT Traversal [Електронний ресурс] Режим доступу: https://developer.cisco.com/docs/spark/#!web-rtc-overview (дата звернення:

12.02.2025).

20. Microsoft Docs. WebRTC and Microsoft Edge [Електронний ресурс] Режим доступу: https://learn.microsoft.com/en-us/microsoft-edge/web-platform/webrtc (дата звернення: 12.02.2025).

21. WebRTC for the Curious [Електронний ресурс] Режим доступу: https://webrtcforthecurious.com/ (дата звернення: 12.02.2025).

22. Socket.IO Documentation [Електронний ресурс] Режим доступу: https://socket.io/docs/v4/ (дата звернення: 12.02.2025).

23. Mediasoup: WebRTC Video Conferencing Framework [Електронний ресурс] Режим доступу: https://mediasoup.org/ (дата звернення: 12.02.2025).

24. WebRTC Sample Projects – GitHub [Електронний ресурс] Режим доступу: https://github.com/webrtc/samples (дата звернення: 12.02.2025).

25. Cloudflare Developers. Connect Applications [Електронний ресурс] Режим доступу: https://developers.cloudflare.com/cloudflare-one/connections/connect-apps/ (дата звернення: 12.02.2025).

26. WebRTC Samples. GitHub repository [Електронний ресурс] Режим доступу: https://github.com/webrtc/samples (дата звернення: 12.02.2025).

27. Janus WebRTC Server Documentation [Електронний ресурс] Режим доступу: https://janus.conf.meetecho.com/docs/ (дата звернення: 12.02.2025).

28. Jitsi Meet Handbook – WebRTC-based Video Conferencing [Електронний ресурс] Режим доступу: https://jitsi.github.io/handbook/docs/ (дата звернення: 12.02.2025).

29. Twilio. WebRTC Media Architecture [Електронний ресурс] Режим доступу: https://www.twilio.com/docs/media-streams/webrtc (дата звернення: 12.02.2025).

30. HLS.js Documentation – Adaptive Streaming for Browsers [Електронний ресурс] Режим доступу: https://github.com/video-dev/hls.js/ (дата звернення: 12.02.2025).

31. WebRTC Hacks Blog – Insights and Tutorials [Електронний ресурс] Режим доступу: https://webrtchacks.com/ (дата звернення: 12.02.2025).

32. Kurento Media Server Documentation [Електронний ресурс] Режим доступу: https://doc-kurento.readthedocs.io/ (дата звернення: 12.02.2025).

33. Mozilla Hacks. Reducing WebRTC Latency [Електронний ресурс] Режим доступу: https://hacks.mozilla.org/ (дата звернення: 12.02.2025).

34. Stack Overflow. WebRTC Architecture Discussions [Електронний ресурс] Режим доступу: https://stackoverflow.com/questions/tagged/webrtc (дата звернення: 12.02.2025).

35. Medium. Building a Video Chat App with WebRTC [Електронний ресурс] Режим доступу: https://bit.ly/4m1PRRF (дата звернення: 12.02.2025).

36. WebRTC Tools Overview by WebRTC.ventures [Електронний ресурс] Режим доступу: https://webrtc.ventures/webrtc-tools-overview/ (дата звернення: 12.02.2025).

37. WebRTC Glossary – Terms and Definitions [Електронний ресурс] Режим доступу: https://webrtcglossary.com/ (дата звернення: 12.02.2025).

38. IEEE Spectrum. The Future of WebRTC [Електронний ресурс] Режим доступу: https://spectrum.ieee.org/ (дата звернення: 12.02.2025).

39. **Real-Time Weekly. WebRTC Industry News [Електронний ресурс]** Режим доступу: https://webrtcweekly.com/ (дата звернення: 12.02.2025).

40. Ngrok Official Documentation [Електронний ресурс] – Режим доступу: https://ngrok.com/docs/getting-started (дата звернення: 12.02.2025).

Додатки

Додаток А

Stream.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Стрімінг</title>
                                                                    <script
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.5.4/socket.io.min.js"></scrip
t>
  <style>
    body {
      margin: 0;
      background-color: #2c2f33;
      color: white;
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      display: flex;
      flex-direction: column;
      align-items: center;
      height: 100vh;
      overflow: hidden;
    }
    h1 {
      margin: 20px 0;
      font-size: 26px;
    }
    video {
      width: 90%;
```

```
      max-width: 900px;
      height: auto;
      background: black;
      border-radius: 12px;
      box-shadow: 0 0 20px rgba(0,0,0,0.5);
    }
    .controls {
      margin: 20px;
      display: flex;
      gap: 10px;
    }
    .button {
      padding: 10px 20px;
      background: #5865f2;
      border: none;
      border-radius: 5px;
      color: white;
      cursor: pointer;
    }
    .button:hover {
      background: #4752c4;
    }
  </style>
</head>
<body>
  <h1>📺 Стрімінг</h1>
```

```
  <div class="controls">
    <button class="button" onclick="startStreamCamera()">📷 Камера</button>
    <button class="button" onclick="startStreamScreen()">🖥️ Екран</button>
  </div>

  <video id="video" autoplay playsinline muted></video>

  <script>
    const socket = io('https://close-enjoyed-lizard.ngrok-free.app');
    const peerConnections = {};
    const videoElement = document.getElementById('video');
    let localStream = null;

    socket.on('connect', () => {
      console.log('✒️ Підключено до сервера');
      socket.emit('broadcaster');
    });

    async function startStreamCamera() {
      try {
        localStream = await navigator.mediaDevices.getUserMedia({ video: true, audio:
true });
        videoElement.srcObject = localStream;
        console.log('📸 Камера активована');
      } catch (error) {
        console.error('🚫 Помилка старту камери:', error);
```

```
    }
  }

  async function startStreamScreen() {
    try {
      localStream = await navigator.mediaDevices.getDisplayMedia({ video: true, audio: true });
      videoElement.srcObject = localStream;
      console.log('🖥️ Екран активований');
    } catch (error) {
      console.error('🚫 Помилка старту екрану:', error);
    }
  }

  socket.on('watcher', async (id) => {
    console.log('👀 Глядач підключився:', id);

    if (!localStream) {
      console.warn('🚫 Немає localStream! Спочатку активуйте камеру або екран.');
      return;
    }

    const peerConnection = new RTCPeerConnection({
      iceServers: [{ urls: 'stun:stun.l.google.com:19302' }]
    });
```

```
    peerConnections[id] = peerConnection;

      localStream.getTracks().forEach(track => peerConnection.addTrack(track,
localStream));

    peerConnection.onicecandidate = (event) => {
      if (event.candidate) {
        socket.emit('candidate', id, event.candidate);
      }
    };

    const offer = await peerConnection.createOffer();
    await peerConnection.setLocalDescription(offer);
    socket.emit('offer', id, peerConnection.localDescription);
  });

  socket.on('answer', (id, description) => {
    console.log('📥 Відповідь отримано від:', id);
    peerConnections[id]?.setRemoteDescription(description);
  });

  socket.on('candidate', (id, candidate) => {
    console.log('❁ ICE-кандидат отриманий від:', id);
    peerConnections[id]?.addIceCandidate(new RTCIceCandidate(candidate));
  });

  socket.on('disconnectPeer', (id) => {
```

```
    console.log('❌ Глядач відключився:', id);
    if (peerConnections[id]) {
      peerConnections[id].close();
      delete peerConnections[id];
    }
  });

  window.addEventListener('beforeunload', () => {
    socket.close();
    for (const id in peerConnections) {
      peerConnections[id].close();
    }
  });
 </script>
</body>
</html>
```

Додвток Б

webrtc_server.js

```
const express = require('express');
const http = require('http');
const { Server } = require('socket.io');
const path = require('path');
```

```
const fs = require('fs');

const app = express();
const server = http.createServer(app);
const io = new Server(server, { cors: { origin: '*' } });

app.use(express.static(__dirname));
app.use(express.json()); // Middleware для парсингу JSON

app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'));
});

// Файл для збереження ніків
const nicknamesFile = path.join(__dirname, 'nicknames.txt');
if (!fs.existsSync(nicknamesFile)) {
  fs.writeFileSync(nicknamesFile, '', { encoding: 'utf8' });
  console.log('Файл nicknames.txt не знайдено — створено порожній файл.');
}

// Ендпоінт для збереження ніку разом з IP адресою
app.post('/save-nickname', (req, res) => {
  const { nickname } = req.body;
  if (nickname) {
    const userIp = req.headers['x-forwarded-for']  req.socket.remoteAddress  req.ip;
    const entry = ${nickname} - ${userIp}\n;
    fs.appendFile(nicknamesFile, entry, (err) => {
```

```
      if (err) {
        console.error('Помилка запису:', err);
        return res.status(500).send('Помилка збереження!');
      }
      res.send('Нік збережено з IP адресою!');
    });
  } else {
    res.status(400).send('Нік не може бути порожнім!');
  }
});

// Ендпоінт для отримання збережених ніків
app.get('/get-nicknames', (req, res) => {
  fs.readFile(nicknamesFile, 'utf8', (err, data) => {
    if (err) {
      if (err.code === 'ENOENT') {
        return res.json([]);
      } else {
        console.error('Помилка читання:', err);
        return res.status(500).send('Помилка отримання ніків!');
      }
    }
    res.json(data.split('\n').filter(line => line));
  });
});

// Основні змінні для логіки стріму
```

```
let broadcasterId = null;
let broadcasterNickname = null;
const viewers = new Set();

io.on('connection', (socket) => {
  console.log('🟢 Користувач підключився:', socket.id);

  // Подія "broadcaster" тепер приймає нік стрімера як параметр
  socket.on('broadcaster', (nickname) => {
    broadcasterId = socket.id;
    broadcasterNickname = nickname;
    console.log(`📡 Стрімер ${broadcasterNickname} підключився (${broadcasterId})`);
    // Якщо потрібно, можна повідомити всім клієнтам інформацію про стрімера
    io.emit('broadcasterInfo', { nickname: broadcasterNickname });
  });

  socket.on('watcher', () => {
    if (broadcasterId) {
      viewers.add(socket.id);
      console.log('👀 Глядач підключився:', socket.id);
      io.to(broadcasterId).emit('watcher', socket.id);
    }
  });

  socket.on('offer', (targetId, offer) => {
```

```
  io.to(targetId).emit('offer', socket.id, offer);
});

socket.on('answer', (targetId, answer) => {
  io.to(targetId).emit('answer', socket.id, answer);
});

socket.on('candidate', (targetId, candidate) => {
  io.to(targetId).emit('candidate', socket.id, candidate);
});

// Обробка чат-повідомлень
socket.on('chatMessage', (data) => {
  // Якщо повідомлення надходить від стрімера, підставляємо збережений нік
  if (socket.id === broadcasterId) {
    data.nickname = broadcasterNickname;
  }
  io.emit('chatMessage', data);
});

socket.on('disconnect', () => {
  console.log('🔴 Користувач відключився:', socket.id);
  viewers.delete(socket.id);
  if (socket.id === broadcasterId) {
    broadcasterId = null;
    broadcasterNickname = null;
    viewers.clear();
```

```
      console.log('⚡ Стрімер відключився — глядачі відключені');
      io.emit('broadcasterDisconnected');
    } else {
      if (broadcasterId) {
        io.to(broadcasterId).emit('disconnectPeer', socket.id);
      }
    }
  });
});

server.listen(3002, () => {
  console.log('✅ WebRTC сервер працює на порту 3002');
});
```