Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: «Інформаційно-комунікаційна система для проведення відеоконференцій із застосуванням WebRTC»

Виконав студент 6 курсу, групи КН(м)-62
спеціальності 122 „Комп'ютерні науки"
(шифр і назва напрямку підготовки спеціальності)

Варенюк О.Ю.
(прізвище, ініціали)

Керівник: *Флуд Л.О.*
(прізвище, ініціали)

Рецензент: Мокрицька О.В.
(прізвище, ініціали)

Львів-2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій

Кафедра комп'ютерних наук

Рівень вищої освіти другий (магістерський)

Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри КН

_____ Борецька І.Б.

"10" грудня 2025 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

Варенюку Оресту Юрійовичу
(прізвище, ім'я, по батькові)

1.Тема магістерської роботи: Інформаційно-комунікаційна система для проведення відеоконференцій із застосуванням WebRTC

керівник роботи: *Флуд Л.О., кандидат технічних наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "29" квітня 2025р, № С-288

2.Термін подання студентом роботи 10 грудня 2025р

3. Вихідні дані до роботи Розробити програмний застосунок по передачі потокових даних використовуючи мови програмування C++ та JavaScript.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Розділ 1. Стан проблемної області
Розділ 2. Інформаційне забезпечення
Розділ 3. Математичне забезпечення
Розділ 4. Програмне забезпечення
Розділ 5. Розроблення стартапу- проєкту
Висновки. Список використаних джерел. Додатк

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Підготовка матеріалу до доповіді

6. Дата видачі завдання 1 травня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№, з/п	Етапи бакалаврської роботи	Термін виконання етапів роботи	Примітка
1.	Вивчення літературних джерел та збір необхідних матеріалів.	1.05.25-1.06.25	Виконано
2.	Аналіз досліджуваної теми та вибір відповідних варіантів її розробки	2.06.25-1.07.25	Виконано
3.	Постановка задачі та її формалізація	2.07.25-20.07.25	Виконано
4.	Вибір та обґрунтування методів і засобів проведення дослідження	21.07.25-22.18.25	Виконано
5.	Розроблення концептуальної схеми реалізації завдання	23.08.25-25.09.25	Виконано
6.	Програмна реалізація завдання	26.09.25-01.11.25	Виконано
7.	Тестування програмного продукту та отриманих результатів	01.11.25-20.11.25	Виконано
8.	Розробка пояснювальної записки магістерської роботи	21.11.25-03.12.25	Виконано
9.	Корегування пояснювальної записки згідно вимог, розроблення презентації	03.12.25-10.12.25	Виконано

Студент _____ Варенюк О.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Флуд Л.О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 86 сторінки пояснювальної записки, 31 рисунків, 1 додаток, 19 джерел.

У дипломній роботі досліджено можливості побудови системи аудіо- та відеокомунікації на основі технології WebRTC та реалізовано програмний застосунок, що забезпечує проведення відеодзвінків безпосередньо у веббраузері. Окрім індивідуальних сеансів зв’язку, розроблено вебсервіс для організації багатокористувацьких відеоконференцій. Створене програмне забезпечення орієнтоване на використання у малих організаціях, офісах, а також для забезпечення внутрішнього зв’язку між підрозділами великих підприємств. Робота демонструє практичні переваги WebRTC як відкритої технології реального часу, яка не потребує встановлення додаткових модулів чи плагінів.

Ключові слова: JavaScript, WebRTC, browser, plug-in, HTML5

ABSTRACT

The thesis consists of 86 pages of explanatory text, 31 figures, 1 appendix, and 19 references.

This thesis explores the development of an audio and video communication application based on WebRTC technology. The work presents the implementation of a web-based video calling solution that operates directly in a browser environment, without requiring additional plug-ins or external software. In addition to one-to-one communication, a multiuser conferencing service was designed, enabling real-time collaboration among several participants. The proposed software is intended for use in small offices as well as for facilitating communication between departments within large organizations. The project demonstrates the practical benefits of WebRTC as an open, flexible, and cost-effective real-time communication technology.

Keywords: JavaScript, WebRTC, browser, plug-in, HTML5

ТЕХНІЧНЕ ЗАВДАННЯ

Метою проєкту є розробка інформаційно-комунікаційної системи, що забезпечує аудіо-, відео- та конференц-зв’язок у реальному часі. Система має функціонувати на основі технології WebRTC, що дозволить реалізувати прямий обмін мультимедійними потоками між користувачами без використання сторонніх плагінів. Для створення клієнтської та серверної частини передбачається застосування мови програмування JavaScript.

Програмне рішення повинно включати серверний компонент для керування сигналізацією, авторизацією та маршрутизацією з’єднань, а також завантажуваний клієнтський модуль, що забезпечує встановлення WebRTC-сеансів, обробку аудіо- та відеоданих і відображення інтерфейсу користувача. Система має підтримувати обмін відео, аудіо й текстовими повідомленнями, створення конференцій із кількома учасниками та можливість масштабування.

Розроблювана система повинна гарантувати стабільність з’єднання, мінімальну затримку передачі, коректну роботу в вебсередовищі та кросбраузерну сумісність. Передбачається реалізація базових засобів безпеки, включаючи шифрування трафіку та захист сигналізаційних повідомлень.

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

BSD-3 — це ліцензійна угода, що регламентує використання програмного забезпечення з відкритим кодом. На відміну від інших вільних ліцензій, BSD вирізняється мінімальною кількістю обмежень, що наближає її до умов публічного надбання. Ліцензія BSD-3 надає користувачеві право не лише використовувати програмний код, а й вносити до нього зміни та застосовувати модифіковані версії у власних проєктах.

GIPS (Global IP Solutions) — компанія, яка спеціалізувалася на розробленні програмного забезпечення для обробки та передавання аудіо- і відеоданих через IP-мережі. У 2010 році компанію було придбано корпорацією Google, після чого вона стала її структурним підрозділом. Нині діяльність GIPS продовжується у межах проєкту WebRTC Project, спрямованого на вдосконалення технологій комунікації в реальному часі.

FCS MX (Flash Communication Server) — серверне середовище, створене компанією Adobe для реалізації потокової передачі медіаданих і комунікації в реальному часі. Цей сервер став попередником сучасного продукту Adobe Media Server, що забезпечує розширені можливості передавання мультимедійного контенту.

RTMP (Real-Time Messaging Protocol) — власницький протокол потокової передачі мультимедійної інформації, який застосовується переважно для трансляції аудіо- та відеоконтенту через мережу Інтернет. RTMP забезпечує стабільну передачу даних із низькою затримкою та використовується в більшості систем потокового мовлення.

SDP (Session Description Protocol) — це протокол прикладного рівня та формат повідомлень, призначений для опису параметрів сесій потокової передачі даних. Він використовується для узгодження характеристик медіасесії між учасниками зв’язку, зокрема типу медіа, формату кодування, адреси передавання та інших технічних параметрів.

API (Application Programming Interface) — це прикладний програмний інтерфейс, який являє собою сукупність готових класів, процедур, функцій, структур і констант, призначених для використання у сторонніх програмних продуктах. API виконує роль абстракційного шару, що спрощує доступ до функціональних можливостей програмного забезпечення або бібліотеки. Завдяки цьому забезпечується ефективне створення нових програмних рішень на основі існуючих компонентів.

VoIP (Voice over IP) — це технологія передавання голосових і відеосигналів за допомогою IP-протоколу. Вона реалізує функції традиційного телефонного зв'язку, такі як набір номера, здійснення дзвінків і передавання голосу в реальному часі. Завдяки використанню IP-мереж забезпечується висока гнучкість, масштабованість і зниження витрат на комунікацію.

STUN (Session Traversal Utilities for NAT) — це мережевий протокол, який дозволяє клієнту, що перебуває за пристроєм трансляції мережевих адрес (NAT), визначити власну зовнішню IP-адресу, а також спосіб, у який NAT здійснює трансляцію портів і адрес. Отримані дані використовуються для встановлення прямого з'єднання типу UDP між двома клієнтами, які розташовані за NAT-маршрутизаторами.

JRE (Java Runtime Environment) — це середовище виконання програмного коду, написаного мовою Java. Воно містить необхідні бібліотеки, класи та віртуальну машину Java (JVM), які забезпечують коректне функціонування програмного забезпечення, створеного з використанням цієї мови програмування.

RTP (Real-Time Transport Protocol) — відкритий протокол, призначений для потокової передачі мультимедійних даних у реальному часі в межах IP-мереж. Протокол функціонує на прикладному рівні моделі OSI та використовується під час передавання аудіо- і відеотрафіку. Його було розроблено та вперше опубліковано у 1996 році робочою групою Audio-Video Transport Working Group.

NAT (Network Address Translation) — це механізм, який застосовується в IP-мережах для перетворення мережевих адрес у пакетах даних, що передаються між

внутрішніми та зовнішніми мережами. NAT дозволяє кільком пристроям у локальній мережі використовувати одну публічну IP-адресу, забезпечуючи економію адресного простору та додатковий рівень безпеки.

TURN (Traversal Using Relays around NAT) — це протокол, що забезпечує можливість отримання вхідних даних вузлом, розташованим за NAT або міжмережевим екраном (фаєрволом), через TCP або UDP-з'єднання. TURN використовується у випадках, коли пряме з'єднання між двома клієнтами неможливе, виконуючи роль ретранслятора мережевого трафіку.

Libjingle — це набір бібліотек на мові програмування C++ з відкритим вихідним кодом, призначений для створення підключень у реальному часі через мережу Інтернет. Дана бібліотека була розроблена компанією Google та стала основою для подальшої реалізації технології WebRTC, яка забезпечує аудіо-, відео- та дата-комунікацію між клієнтами.

W3C (World Wide Web Consortium) — це міжнародна організація, що займається розробленням і впровадженням стандартів для Всесвітньої мережі. Основною метою діяльності консорціуму є забезпечення сумісності та сталого розвитку вебтехнологій. Керівником W3C є сер Тімоті Джон Бернерс-Лі — автор основних концепцій та технологій, які стали фундаментом сучасного Інтернету.

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології є ключовим чинником розвитку суспільства, економіки та освіти. Вони забезпечують ефективний обмін знаннями, оперативну передачу даних та інтерактивну взаємодію між користувачами незалежно від їхнього місцезнаходження. Одним із напрямів у галузі інформаційно-комунікаційних технологій, що найбільш динамічно розвивається є системи відео конференц зв'язку, які дозволяють організовувати спілкування в режимі реального часу із застосуванням аудіо-, відео- та текстових каналів.

Швидкий розвиток глобальної мережі Інтернет, підвищення пропускної здатності каналів зв'язку та поширення потужних персональних пристроїв створили передумови для переходу від традиційних засобів комунікації до інтерактивних мультимедійних систем. Відеоконференцзв'язок став невід'ємною складовою сучасних бізнес-процесів, державного управління, дистанційного навчання, телемедицини, технічної підтримки та багатьох інших сфер діяльності.

Попри широкий спектр існуючих програмних рішень, актуальною залишається проблема створення універсальних, безпечних і гнучких систем відеокомунікацій, які б забезпечували високу якість передавання даних, мінімальну затримку та зручність використання без потреби встановлення спеціалізованого програмного забезпечення. Саме ці вимоги задовольняє технологія WebRTC (Web Real-Time Communication) — відкрите стандартне рішення, що забезпечує обмін аудіо-, відео- та іншими потоками даних безпосередньо між браузерами чи мобільними клієнтами без використання сторонніх плагінів [2].

Застосування WebRTC відкриває нові можливості для розробників і користувачів, оскільки ця технологія поєднує високу продуктивність, безпеку та сумісність із сучасними вебстандартами. WebRTC підтримує наскрізне шифрування трафіку, що забезпечує захищеність комунікацій, і дозволяє створювати масштабовані архітектури з використанням серверів-сигналізаторів, ретрансляторів та інших компонентів. Водночас відкрита природа стандарту

сприяє інтеграції з існуючими системами керування, навчальними платформами, CRM- і ERP-системами.

Актуальність дослідження полягає в необхідності створення сучасної інформаційно-комунікаційної системи, що поєднує функціональні можливості відеоконференцій із доступністю вебтехнологій і високим рівнем безпеки. Така система повинна забезпечувати якісний обмін медіаданими між користувачами в реальному часі, підтримувати масштабування та бути сумісною з різними типами пристроїв і операційних систем.

Об'єктом дослідження є процес організації комунікацій у реальному часі в інформаційно-комунікаційних системах.

Предметом дослідження є технологія WebRTC та методи її застосування під час створення систем відеоконференцій.

Метою роботи є розроблення інформаційно-комунікаційної системи для проведення відеоконференцій із використанням технології WebRTC, що забезпечує ефективну взаємодію користувачів у реальному часі з високою якістю передавання аудіо- та відеоданих.

Для досягнення поставленої мети необхідно виконати такі завдання:

- провести аналіз сучасних технологій і протоколів, що застосовуються у системах відеоконференцзв'язку;
- дослідити архітектуру та принципи функціонування WebRTC;
- розробити концептуальну модель та архітектуру інформаційно-комунікаційної системи;
- реалізувати сервісну частину та клієнтський модуль системи з використанням JavaScript;
- забезпечити шифрування медіапотоків і механізми автентифікації користувачів;
- провести тестування та оцінити ефективність розробленої системи.

Методи дослідження базуються на системному аналізі, методах об'єктно-орієнтованого програмування, теорії мережевих протоколів, моделюванні

процесів передавання мультимедійних даних, а також на використанні стандартів W3C і IETF, що регламентують технологію WebRTC.

Наукова новизна роботи полягає у розробленні архітектурного рішення для побудови інформаційно-комунікаційної системи відеоконференцій, що забезпечує підвищену ефективність і безпеку комунікації завдяки використанню технологій прямого з'єднання (peer-to-peer) і оптимізованої маршрутизації медіапотоків.

Практичне значення отриманих результатів полягає у створенні функціонального прототипу програмного забезпечення для організації відеоконференцій, який може бути інтегрований у корпоративні або освітні платформи, а також адаптований для внутрішніх комунікацій підприємств.

Структура магістерської роботи. Робота складається зі вступу, п'яти розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Порівняння з аналогічними технологіями

Розвиток інформаційно-комунікаційних технологій зумовив появу численних програмних рішень для організації аудіо-, відео- та конференц-зв'язку в реальному часі. Серед найвідоміших технологій і протоколів можна виділити H.323, SIP, RTMP, Java Media Framework (JMF), а також сучасні комерційні платформи — Zoom, Microsoft Teams, Google Meet, Cisco WebEx тощо. Кожне з рішень має власну архітектуру, специфічний набір протоколів і різний ступінь відкритості.

Традиційні технології відеозв'язку

Перші системи відеоконференцій створювалися на базі стандарту H.323, розробленого ITU-T для передавання мультимедійних потоків через IP-мережі. Незважаючи на високу якість з'єднання, H.323 мав складну структуру, потребував централізованого керування та значних обчислювальних ресурсів.

Іншим важливим етапом розвитку стала поява SIP (Session Initiation Protocol) — протоколу ініціації сеансів, який забезпечував створення, зміну та завершення мультимедійних з'єднань. Хоча SIP став основою IP-телефонії (VoIP), його використання вимагає додаткових серверних компонентів (Proxy, Registrar, Gateway), що ускладнює інтеграцію в браузерні застосунки [1].

У період активного використання Adobe Flash широкого поширення набув власницький протокол RTMP (Real-Time Messaging Protocol), який тривалий час залишався основним засобом потокового відео в Інтернеті. Однак із припиненням підтримки Flash-технологій у 2020 році цей підхід остаточно втратив актуальність.

Java-технології у системах мультимедійного зв'язку

Окреме місце серед аналогічних рішень займають технології, побудовані на основі мови програмування Java. Платформа Java традиційно використовується для створення багатоплатформних клієнт-серверних систем, зокрема для обробки мультимедійних даних. На початку 2000-х років компанія Sun Microsystems

запропонувала Java Media Framework (JMF) — набір бібліотек для роботи з аудіо- та відеопотоками, а також для реалізації функцій захоплення, передавання й відтворення медіаданих.

Попри універсальність, JMF має низку суттєвих обмежень:

- відсутність активної підтримки сучасних кодеків (H.264, VP8, Opus);
- відсутність інтеграції з браузером без додаткових плагінів або аплетів;
- потреба у значних обчислювальних ресурсах та складна інсталяція на клієнтських пристроях.

Таким чином, використання Java-засобів для побудови систем відеоконференцій на сучасному етапі є малоефективним, особливо в умовах переходу до веб-орієнтованих архітектур [3].

Сучасні вебтехнології

На зміну традиційним і Java-орієнтованим рішенням прийшли відкриті вебстандарти. Найбільш перспективною серед них є WebRTC (Web Real-Time Communication) — технологія, створена консорціумом W3C у співпраці з IETF, що забезпечує обмін аудіо-, відео- та довільними даними без додаткових розширень у браузері.

WebRTC реалізує peer-to-peer (P2P) з'єднання, що мінімізує затримку передачі медіапотоків і зменшує навантаження на сервери. Крім того, технологія підтримує шифрування даних за допомогою протоколів DTLS і SRTP, забезпечуючи високий рівень безпеки.

1.2 Основні протоколи та архітектура WebRTC

Технологія WebRTC (Web Real-Time Communication) — це відкритий стандарт, розроблений консорціумами W3C (World Wide Web Consortium) та IETF (Internet Engineering Task Force), який забезпечує передавання аудіо, відео та даних у режимі реального часу між веббраузерами та іншими клієнтами без необхідності встановлення додаткових плагінів або зовнішніх застосунків.

Архітектура WebRTC базується на принципах peer-to-peer (P2P) взаємодії, що дозволяє користувачам встановлювати пряме з'єднання між собою,

мінімізуючи затримку передавання даних і знижуючи навантаження на серверну інфраструктуру. Водночас для ініціалізації з'єднання, узгодження параметрів передачі та обходу мережевих обмежень використовуються допоміжні серверні механізми та стандартизовані протоколи.

1.2.1 Архітектурна схема WebRTC

Система WebRTC складається з трьох основних компонентів:

Клієнтський модуль (Peer) — безпосередньо реалізує відправлення та приймання медіаданих через браузер або мобільний застосунок. Він використовує API WebRTC для доступу до камери, мікрофона, модуля обробки аудіо та відео, а також для створення P2P-з'єднання.

Сервер сигналізації (Signaling Server) — відповідає за обмін службовими повідомленнями між клієнтами під час встановлення з'єднання. Через цей сервер передаються параметри сеансу, коди форматів (codecs), мережеві адреси та інша службова інформація. Сигналізація не регламентується самим стандартом WebRTC, тому може реалізовуватися за допомогою протоколів WebSocket, HTTP, SIP або інших механізмів.

Сервери обходу NAT (STUN / TURN) — допоміжні елементи, необхідні для забезпечення зв'язку між клієнтами, що перебувають за фаєрволами або системами трансляції адрес. Вони визначають зовнішні IP-адреси клієнтів і, за потреби, ретранслюють трафік.

Завдяки цій архітектурі WebRTC забезпечує стабільну роботу навіть у складних мережевих умовах, що є критично важливим для систем відеоконференцзв'язку.

1.2.2 Основні протоколи WebRTC

Для реалізації обміну даними в реальному часі WebRTC використовує комплекс взаємопов'язаних протоколів, кожен із яких виконує окрему функцію у процесі встановлення та підтримання з'єднання.

1. SDP (Session Description Protocol)

Це протокол прикладного рівня, який використовується для опису

параметрів мультимедійної сесії. SDP містить інформацію про типи медіапотоків, кодеки, порти, IP-адреси, а також умови з’єднання. Він використовується під час етапу сигналізації для узгодження характеристик між клієнтами, забезпечуючи сумісність різних пристроїв та програмних реалізацій.

2. STUN (Session Traversal Utilities for NAT)

Протокол STUN дозволяє клієнтам, які знаходяться за мережевими маршрутизаторами або фаєрволами, визначати власну публічну IP-адресу і порт, що використовуються у зовнішній мережі. Це дає змогу встановити пряме UDP-з’єднання між двома клієнтами. У більшості випадків STUN є першим етапом побудови каналу зв’язку між браузерами.

3. TURN (Traversal Using Relays around NAT)

Якщо встановити пряме з’єднання через STUN неможливо (наприклад, через сувору політику NAT або фаєрволу), використовується протокол TURN. У цьому випадку дані передаються через проміжний сервер-ретранслятор, який забезпечує обхід обмежень і стабільність зв’язку, хоча це й збільшує затримку.

4. ICE (Interactive Connectivity Establishment)

Це механізм, що об’єднує STUN і TURN для визначення найоптимальнішого маршруту з’єднання між клієнтами. ICE перевіряє всі можливі варіанти комунікації — прямі, через STUN, або через TURN — і вибирає той, що забезпечує найменшу затримку та найвищу стабільність.

5. RTP / RTCP (Real-Time Transport Protocol / Real-Time Control Protocol)

Протокол RTP відповідає за безпосередню передачу мультимедійних даних (аудіо та відео) у реальному часі. Він визначає структуру пакетів, часові мітки та синхронізацію потоків.

Паралельно з ним працює RTCP, який використовується для моніторингу якості з’єднання та обміну службовою інформацією (наприклад, про затримку, втрату пакетів чи пропускну здатність).

6. SRTP (Secure Real-Time Transport Protocol)

SRTP є розширенням RTP, яке забезпечує шифрування та автентифікацію

мультимедійних потоків. Це дозволяє гарантувати цілісність даних і захищати користувачів від перехоплення чи модифікації трафіку.

7. DTLS (Datagram Transport Layer Security)

DTLS забезпечує захищений обмін ключами для SRTP і відповідає за безпечне встановлення з'єднання між клієнтами. Це еквівалент протоколу TLS, але адаптований до датаграмного формату передавання (UDP).

1.2.3 Особливості безпеки та взаємодії компонентів

WebRTC має високий рівень безпеки завдяки наскрізному шифруванню медіапотоків і обов'язковій автентифікації учасників. Усі з'єднання встановлюються лише після явного дозволу користувача на доступ до мікрофона і камери, що виключає можливість несанкціонованого знімання даних.

У процесі роботи WebRTC клієнт спочатку звертається до сервера сигналізації, через який обмінюється SDP-описами. Після узгодження параметрів з'єднання відбувається пошук можливих маршрутів за допомогою ICE. Якщо прямий канал зв'язку знайдено, дані передаються безпосередньо між клієнтами, і лише в разі обмежень у мережі — через TURN-сервер.

Функціонування технології WebRTC ґрунтується на комплексі протоколів і технологічних рішень, що забезпечують стабільну та безпечну передачу мультимедійних потоків у режимі реального часу. Вона поєднує мережеві, транспортні та прикладні протоколи, які спільно формують цілісну архітектуру для організації відео- та аудіокомунікації без використання додаткових плагінів чи зовнішніх застосунків.

Базовим елементом транспортного рівня є протокол UDP (User Datagram Protocol), який забезпечує мінімальну затримку під час передавання даних. На відміну від TCP, UDP не передбачає перевірки доставки кожного пакета, що є критично важливим для забезпечення плавності відтворення аудіо- та відеопотоків. У випадках, коли необхідна підвищена надійність передавання службових повідомлень або сигналізаційних даних, WebRTC може використовувати TCP (Transmission Control Protocol). Для забезпечення

стабільності з’єднання у змінних мережевих умовах застосовується механізм ICE (Interactive Connectivity Establishment), який об’єднує можливості протоколів STUN та TURN і вибирає найоптимальніший маршрут між клієнтами.

Ключовою складовою системи WebRTC є протоколи встановлення з’єднання. Серед них важливу роль відіграє SDP (Session Description Protocol), що визначає параметри мультимедійної сесії — типи медіапотоків, кодеки, адреси та порти, через які здійснюється обмін даними. Для забезпечення зв’язку між клієнтами, які перебувають за мережевими екранами чи системами трансляції адрес, використовуються STUN (Session Traversal Utilities for NAT) та TURN (Traversal Using Relays around NAT). Перший із них дозволяє клієнту виявити власну зовнішню IP-адресу, тоді як другий забезпечує передачу даних через проміжний сервер у випадку, коли пряме з’єднання неможливе. Безпека з’єднання гарантується за рахунок DTLS (Datagram Transport Layer Security), який реалізує обмін ключами та автентифікацію учасників сесії [7].

Передавання мультимедійних даних здійснюється за допомогою RTP (Real-Time Transport Protocol), який визначає структуру та часову синхронізацію аудіо- і відеопотоків. Паралельно з ним функціонує RTCP (Real-Time Control Protocol), що забезпечує контроль якості з’єднання та збір статистичних даних про стан мережі. Для захисту переданих потоків використовується SRTP (Secure Real-Time Transport Protocol), який гарантує цілісність і конфіденційність медіаданих.

Особливу увагу в WebRTC приділено підтримці сучасних кодеків, які забезпечують високу якість сигналу за мінімальної пропускної здатності каналу. Для відео найпоширенішими є VP8, VP9 та H.264, тоді як для аудіо — Opus, G.711 і G.722. Кодек Opus є базовим у стандарті WebRTC завдяки своїй здатності адаптуватися до змінних умов мережі, динамічно змінюючи бітрейт і частоту дискретизації.

Технологія WebRTC тісно інтегрована з іншими вебстандартами, що забезпечують взаємодію з користувачем. Використання HTML5 дозволяє реалізовувати мультимедійні інтерфейси без сторонніх розширень, а JavaScript

API WebRTC забезпечує розробників інструментами для доступу до камер, мікрофонів, створення однорангових з’єднань і передачі даних. Крім того, стандарт W3C Media Capture and Streams регламентує роботу з мультимедійними потоками в межах браузера, надаючи користувачеві контроль над дозволами на використання пристроїв.

Завдяки поєднанню зазначених протоколів і технологій WebRTC забезпечує повний цикл обробки медіаданих — від встановлення з’єднання до шифрування, передавання та контролю якості потоків. Комплексна взаємодія транспортних, сигнальних і прикладних компонентів створює гнучку, масштабовану й безпечну архітектуру, яка відповідає сучасним вимогам до інформаційно-комунікаційних систем. Саме це робить WebRTC універсальною платформою для створення систем відеоконференцзв’язку, які поєднують високу продуктивність, кросплатформеність і відкритість стандартів.

1.2.4 Переваги та недоліки технології

Технологія WebRTC посідає провідне місце серед сучасних засобів реалізації комунікацій у реальному часі завдяки своїй відкритості, гнучкості та здатності забезпечувати стабільне передавання мультимедійних даних без використання додаткових розширень або плагінів. Її переваги визначаються не лише зручністю застосування для кінцевого користувача, а й технічними можливостями, що надають розробникам широкі інструменти для створення ефективних і безпечних систем відеоконференцзв’язку.

До основних переваг WebRTC слід віднести відкритість стандарту та кросплатформеність. Технологія підтримується всіма провідними браузерами — Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, Opera — що робить її універсальним рішенням для різних операційних систем і пристроїв. Завдяки цьому користувач може долучитися до конференції без попереднього встановлення програмного забезпечення, що значно спрощує процес комунікації.

Важливою перевагою WebRTC є висока якість аудіо- та відеопередачі. Технологія використовує сучасні кодеки (Opus, VP8, VP9, H.264), які забезпечують ефективне стиснення даних без помітної втрати якості, навіть за обмеженої пропускної здатності мережі. Крім того, реалізовано динамічне регулювання бітрейту, що дозволяє автоматично адаптуватися до змін у стані мережевого з'єднання, забезпечуючи безперервність сеансу [6].

Особливу увагу в архітектурі WebRTC приділено безпеці комунікацій. Усі дані шифруються із застосуванням протоколів DTLS та SRTP, що гарантує захист від несанкціонованого доступу або перехоплення трафіку. Крім того, доступ до мультимедійних пристроїв користувача (камери, мікрофона) здійснюється виключно за його дозволом, що відповідає вимогам конфіденційності та етичних норм роботи з персональними даними.

Ще однією перевагою WebRTC є підтримка архітектури peer-to-peer (P2P), яка дозволяє організовувати прямий обмін даними між учасниками без використання проміжних серверів. Це забезпечує низьку затримку передавання інформації та зменшує навантаження на мережеву інфраструктуру. За потреби технологія може масштабуватися за допомогою додаткових серверів типу SFU або MCU, що робить її придатною як для невеликих конференцій, так і для багатокористувацьких систем.

Попри численні переваги, технологія WebRTC має і певні недоліки, які обумовлюють складність її використання в окремих сценаріях. Насамперед, це проблеми з обходом NAT і фаєрволів, які можуть унеможливлювати встановлення прямого з'єднання між користувачами. У таких випадках доводиться застосовувати TURN-сервери для ретрансляції трафіку, що збільшує затримку та вимагає додаткових ресурсів.

Серед інших обмежень можна виокремити відсутність єдиного стандарту сигналізації, через що розробники змушені самостійно реалізовувати протоколи обміну службовими повідомленнями (WebSocket, SIP тощо). Це підвищує гнучкість технології, проте ускладнює проєктування та тестування системи.

Також певні труднощі виникають при масштабуванні групових конференцій, оскільки велика кількість однорангових з’єднань призводить до збільшення навантаження на мережу клієнтів і сервери.

Ще одним аспектом, який потребує уваги, є варіативність реалізацій у різних браузерах. Незважаючи на стандартизацію API, певні відмінності у підтримці функцій WebRTC можуть призводити до несумісності або різниці у якості відтворення мультимедійного контенту. Крім того, робота з відео високої роздільності створює значне навантаження на клієнтські пристрої, що особливо відчутно на мобільних або малопотужних системах.

Таким чином, WebRTC можна охарактеризувати як універсальну, відкриту та перспективну технологію, яка поєднує високу якість мультимедійного зв’язку, безпеку та зручність використання. Незважаючи на окремі технічні обмеження, вона залишається найбільш ефективним рішенням для створення сучасних інформаційно-комунікаційних систем відеоконференцзв’язку. Її переваги суттєво переважають потенційні недоліки, особливо за умови правильного проєктування архітектури системи, оптимізації мережевих процесів і використання допоміжних технологій, що підвищують стабільність і масштабованість.

Висновки до розділу

У першому розділі розглянуто сучасні технології організації відеоконференцзв’язку та проведено порівняльний аналіз їхніх можливостей. Визначено, що серед існуючих рішень технологія WebRTC є найбільш перспективною завдяки відкритому стандарту, кросплатформеності та високому рівню безпеки.

Проаналізовано архітектуру WebRTC, її основні протоколи та підтримувані технології, які забезпечують передавання аудіо- та відеоданих у реальному часі. Встановлено, що поєднання протоколів SDP, STUN, TURN, RTP і DTLS створює гнучку й надійну інфраструктуру для стабільної комунікації між клієнтами.

Виявлено, що попри окремі технічні недоліки — складність масштабування, варіативність реалізацій у браузерах і проблеми з обходом NAT

— технологія WebRTC залишається оптимальним інструментом для створення сучасних систем відеоконференцзв’язку, які поєднують ефективність, безпеку та зручність використання.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Середовище розробки програм

Для реалізації інформаційно-комунікаційної системи відеоконференцзв'язку було обрано сучасне середовище розробки, яке забезпечує ефективну роботу з вебтехнологіями, зокрема з мовою програмування JavaScript, що є основною для технології WebRTC.

У процесі створення програмного забезпечення використовувалося середовище Visual Studio Code (VS Code), яке є одним із найпоширеніших інструментів для веброзробки. Воно поєднує високу продуктивність, зручність інтерфейсу та широкі можливості розширення завдяки системі плагінів. Visual Studio Code підтримує синтаксичне підсвічування, автоматичне доповнення коду (IntelliSense), інтеграцію з системами контролю версій Git, а також засоби для налагодження JavaScript-програм безпосередньо у браузері.

Розроблення клієнтської частини системи здійснювалося із застосуванням HTML5, CSS3 та JavaScript, що дозволяє реалізувати інтерфейс користувача без встановлення додаткових компонентів. Технологія HTML5 забезпечує роботу з мультимедійними елементами, тоді як CSS3 використовується для оформлення інтерфейсу та створення адаптивного дизайну.

Для тестування взаємодії компонентів WebRTC застосовувалися сучасні браузери — Google Chrome і Mozilla Firefox, які мають повну підтримку стандартів W3C та API WebRTC. Це дало змогу перевірити працездатність системи на різних платформах і в різних мережевих умовах [9].

Серверна частина системи реалізована з використанням Node.js, що дозволяє створювати високопродуктивні вебсервери на основі подієвої архітектури. Node.js забезпечує обробку великої кількості одночасних підключень і підтримує бібліотеки, необхідні для роботи з протоколами WebSocket, STUN і TURN. Додатково використовувалася система керування пакетами npm, яка спрощує інсталяцію зовнішніх модулів і керування залежностями проєкту.

Для контролю версій і командної роботи було застосовано систему Git з розміщенням репозиторію на платформі GitHub. Це дало змогу забезпечити збереження історії змін, віддалену співпрацю та автоматизацію процесів розгортання.

Таким чином, середовище розробки програмного забезпечення для інформаційно-комунікаційної системи базується на сучасних інструментах вебпрограмування. Їх використання забезпечує гнучкість, надійність і масштабованість розробленого рішення, а також повну сумісність із технологією WebRTC, що є ключовою для реалізації системи відеоконференцій.

2.2 Вибір технологій і структур даних

Розроблення інформаційно-комунікаційної системи для проведення відеоконференцій потребує використання сучасних вебтехнологій, які забезпечують стабільну роботу в реальному часі, масштабованість і безпеку. Вибір технологічного стеку ґрунтувався на сумісності з архітектурою WebRTC, можливостях інтеграції з браузерами та ефективності роботи з мережевими з’єднаннями.

Основою серверної частини обрано платформу Node.js, яка забезпечує неблокуючу, подієво орієнтовану модель обробки запитів. Це дозволяє одночасно обслуговувати велику кількість клієнтів і гарантує високу продуктивність системи при мінімальних затратах ресурсів. Додатково застосовується фреймворк Express.js, який спрощує розроблення серверних маршрутів і обробку запитів HTTP, забезпечуючи зручну структуру коду та підтримку REST-архітектури [4].

Для організації двостороннього обміну даними між клієнтами та сервером використовується протокол WebSocket, який забезпечує постійне з’єднання без повторного встановлення сеансів. Це дозволяє передавати повідомлення сигналізації WebRTC (обмін SDP-параметрами, кандидатами ICE тощо) з мінімальною затримкою, що є критичним для систем реального часу.

В межах обробки внутрішніх даних використовується формат JSON (JavaScript Object Notation), який є універсальним способом зберігання та передачі

структурованої інформації. JSON забезпечує простоту інтеграції між клієнтською та серверною частинами, оскільки є рідним форматом для мови JavaScript і легко парсується у браузері.

Для зберігання інформації про користувачів, історію сесій та налаштування системи доцільним є використання баз даних NoSQL типу MongoDB, яка підтримує гнучку схему даних і високу швидкодію при роботі з великими обсягами інформації. Її документно-орієнтована структура ідеально поєднується з JSON-форматом, що спрощує обмін даними між рівнями системи.

Клієнтська частина розроблена на мові JavaScript із використанням стандартних вебтехнологій HTML5 та CSS3. Такий підхід забезпечує незалежність від операційної системи та браузера, а також дозволяє використовувати стандартні API WebRTC — getUserMedia(), RTCPeerConnection, RTCDataChannel — для створення з’єднань, обробки потоків і передачі даних.

Для перевірки стабільності роботи системи в реальному часі застосовуються інструменти налагодження браузерів (Chrome DevTools, Firefox Developer Tools), а також бібліотеки Socket.IO для реалізації WebSocket-з’єднань і спрощення обміну повідомленнями між клієнтами [6].

Вибраний набір технологій забезпечує:

- високу продуктивність при одночасному обслуговуванні багатьох користувачів;
- низьку затримку під час передавання аудіо- та відеоданих;
- простоту масштабування системи;
- сумісність з основними стандартами WebRTC і відкритими вебтехнологіями.

Таким чином, використання Node.js, Express.js, WebSocket і MongoDB у поєднанні з JSON-обміном даними та клієнтським JavaScript забезпечує створення надійної та гнучкої архітектури, придатної для реалізації сучасної системи відеоконференцій у браузерному середовищі.

2.3 Структура та функціональні можливості системи

Інформаційно-комунікаційна система для проведення відеоконференцій побудована за архітектурою клієнт–сервер із використанням принципу peer-to-peer для передачі медіаданих. Такий підхід забезпечує мінімальну затримку сигналу, зниження навантаження на сервер і високу якість зв’язку між користувачами.

Клієнтська частина реалізована за допомогою JavaScript, HTML5 та CSS3, що дозволяє користувачам отримувати доступ до системи без встановлення додаткового програмного забезпечення. Вона забезпечує обробку локальних аудіо- та відеопотоків, їх передавання через WebRTC і взаємодію з сервером сигналізації.

Серверна частина розроблена на основі Node.js із використанням Socket.IO, що забезпечує обмін службовими повідомленнями між клієнтами (SDP, ICE-кандидати) та керує створенням з’єднання. Для зберігання службової інформації використовується MongoDB, яка містить дані про користувачів і параметри конференцій. Система підтримує основні функції: створення та приєднання до відеоконференцій, обмін аудіо- й відеопотоками, текстове спілкування, керування мікрофоном і камерою, перегляд активних учасників та завершення сесії.

Завдяки поєднанню вебтехнологій і протоколів WebRTC система є універсальною, безпечною та зручною у використанні. Її структура забезпечує надійну взаємодію користувачів у реальному часі й дає змогу масштабувати рішення залежно від потреб.

Висновки до розділу

У другому розділі розглянуто інформаційне забезпечення розроблення системи відеоконференцзв’язку на основі технології WebRTC. Визначено програмне середовище, інструменти та технології, що забезпечують ефективну реалізацію клієнтської і серверної частин. Показано, що використання Visual Studio Code як основного середовища розробки, у поєднанні з платформою Node.js та фреймворком Express.js, забезпечує гнучкість, зручність налагодження

і високу продуктивність. Застосування WebSocket дає змогу реалізувати двосторонню комунікацію в реальному часі, а MongoDB — зберігати дані користувачів і параметри конференцій [12].

Проаналізовано структуру системи, що поєднує архітектуру «клієнт–сервер» із принципом peer-to-peer, що дозволяє передавати мультимедійні потоки напряму між клієнтами. Такий підхід забезпечує мінімальну затримку зв’язку, стабільність роботи та масштабованість рішення.

Отже, обране технологічне середовище та архітектура повністю відповідають вимогам до сучасних інформаційно-комунікаційних систем, забезпечуючи ефективне, безпечне та зручне проведення відеоконференцій у вебсередовищі.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

Згладжування загалом передбачає отримання максимально регуляризованого та безперервного сигналу. У випадку оброблення зображень під згладженим сигналом часто розуміють шматково-постійну (piecewise-constant) модель, в межах якої зображення представлено як сукупність окремих областей (сегментів), кожна з яких характеризується майже сталим значенням інтенсивності.

Представлення зображення у вигляді кусково-постійної моделі фактично зводиться до задачі сегментації, що є однією з ключових цілей аналітичного опрацювання зображень та слугує основою для формування їх структурного опису. Такий підхід застосовується не лише до зображень із чітко вираженими об’єктами, але й до зображень з текстурними характеристиками, де аналіз зосереджується на властивостях текстури, а не на глобальних інтенсивностях відеосигналу [5].

Крім того, при реалізації процедури згладжування необхідно визначити, які компоненти сигналу повинні бути пригнічені. Зазвичай до таких компонентів належать високочастотні флуктуації, що інтерпретуються як шум. Їх усунення дає змогу наблизити сигнал до бажаної згладженої моделі.

Надалі розглянемо принципи роботи алгоритму двовимірного згладжування, що використовується для формування регуляризованого представлення зображення.

3.1 Двовимірне згладжування зображень

Розвиток обчислювальної техніки та інформаційних технологій призводить до необхідності модифікації відомих алгоритмів обробки зображень. Розглянемо один із таких методів - двовимірне згладжування зображень.

Згладжування зображень - зменшення спотворень, викликаних дією шумів системи відтворення зображень (фото, відео тощо), є практично важливим завданням.

Можливість згладжування обумовлена відмінністю властивостей зображення та шуму. При статистичному підході до згладжування ефективність алгоритму залежить від повноти використовуваного статистичного опису зображення і шуму. Зазвичай шуми відрізняються простою статистичною структурою, та їх властивості або відомі, або можуть бути отримані в результаті нескладних вимірів. Для зображень вимір досить повних статистичних характеристик є складним завданням, яке можна полегшити, використовуючи конструктивну модель зображення.

У задачі згладжування одновимірних сигналів часто використовують гауссівську модель сигналів, яка призводить до вінеровського алгоритму лінійного згладжування, що забезпечує мінімальне середньоквадратичне відхилення (СКВ) згладженого сигналу від вихідного. Таке згладжування здійснюється фільтром із частотною характеристикою.

$$K(\omega) = \frac{S(\omega)}{(S(\omega)+N(\omega))} \tag{3.1}$$

де $S(\omega), N(\omega)$ - енергетичні спектри сигналу і шуму вінера, ω - частота.

Концепцію вінерівського згладжування можна узагальнити на багатовимірні (n-вимірні) сигнали, при цьому оптимальна частотна характеристика фільтра зберігає загальну форму, але замість одномірної частоти в ній присутня n-вимірна частотна змінна.

Однак застосування лінійного вінерівського згладжування до двовимірних зображень не завжди приводить до покращення візуально сприйманої якості. Причина полягає в тому, що шум зазвичай містить суттєві енергетичні компоненти на високих просторових частотах, тоді як корисна інформація зображення зосереджена переважно в області низьких частот. Спектральна щільність зображення S(ω)S(\omega)S(ω) різко зменшується зі зростанням частоти, тому згідно з формулою (1) фільтр буде сильно пригнічувати високочастотні складові. Унаслідок цього зникають важливі деталі — зокрема,

інформація про контури та дрібні структури об’єктів, що робить результат згладжування візуально «розмитим».

Теоретичні засади вінерівського згладжування можуть бути розширені на випадок n-вимірних сигналів, що дозволяє застосовувати метод до двовимірних і тривимірних даних. У цьому разі оптимальна частотна характеристика фільтра зберігає аналогічний аналітичний вигляд, але містить n-вимірну частотну змінну.

Попри математичну обґрунтованість такого підходу, застосування лінійного вінерівського фільтра до зображень не завжди забезпечує покращення візуальної якості. Це пов’язано з тим, що шумові компоненти мають значну енергію на високих просторових частотах, тоді як спектральна щільність зображення S(ω) різко спадає зі збільшенням частоти. Унаслідок цього фільтр пригнічує високочастотні складові, які містять інформацію про краї та дрібні деталі [12].

Як зазначав Д. Габор, спектральний опис є недостатнім для врахування локально-анізотропної структури зображення. Для її опису застосовують структурно-компонентні моделі, що дозволяють отримати оцінку інтенсивності пікселів з мінімальним середньоквадратичним відхиленням.

Формула оптимального вінерівського фільтра:

$$H(\omega) = S_x(\omega) / (S_x(\omega) + S_n(\omega)) \quad (3.2)$$

де $S_x(\omega)$ — спектральна щільність потужності сигналу, $S_n(\omega)$ — спектральна щільність шуму.

3.2 "Складена" модель фрагмента зображення

Структурно-компонентне подання локального фрагмента зображення є одним із ключових концептуальних підходів у сучасних методах цифрової обробки зображень. Воно сформувалося як відповідь на обмежені можливості класичних спектральних підходів, які часто недостатньо точно описують локальну неоднорідність, напрямну структуру та складну просторову організацію реальних зображень. На відміну від глобальних моделей, що трактують зображення як частотно однорідний сигнал, структурно-компонентний підхід

передбачає розкладання фрагмента на декілька змістових елементів, кожен із яких має власні локальні характеристики та виконує специфічну роль у формуванні загального візуального контенту [13].

У практичних умовах природні зображення доцільно інтерпретувати як сумарний результат взаємодії кількох типових складових: областей з майже постійною інтенсивністю, різких переходів (границь), характерних структурних утворень (кутових точок), періодичних або квазірегулярних текстур та шумових домішок. Таким чином, локальний фрагмент зображення розглядається як композиція цих компонентів, кожен з яких відображає певний тип локальної інформації та визначає поведінку сигналу в окремих частинах області аналізу:

$$f(x, y) = f_s(x, y) + f_e(x, y) + f_t(x, y) + n(x, y), \quad (3.3)$$

де:

- $f_s(x, y)$ — згладжена (smooth) складова;
- $f_e(x, y)$ — компонента границь (edges);
- $f_t(x, y)$ — текстурна складова;
- $n(x, y)$ — шум.

Такий підхід дозволяє розглядати фрагмент зображення як систему структурно неоднорідних елементів, які істотно відрізняються за своїми частотними характеристиками.

Д. Габор запропонував представляти локальний сигнал через набір елементарних фільтрів, подібних до частотно-часових вікон. Функція Габора задається виразом:

$$g(x, y) = \exp(-(x^2 + y^2) / (2\sigma^2)) * \exp(j(\omega_x x + \omega_y y)), \quad (3.4)$$

де σ — ширина вікна, ω_x та ω_y — просторові частоти, j — уявна одиниця.

Такі фільтри добре моделюють локальні структури, оскільки забезпечують одночасне частотне та просторове локалізування. Використання банку фільтрів Габора дозволяє розкласти фрагмент зображення на компоненти різних орієнтацій і масштабів, що формує основу складеної моделі.

Локальна модель фрагмента зображення будується з урахуванням орієнтаційних властивостей. Наприклад, границя описується як функція, що має високу енергію в певному напрямку. Формально границю можна наблизити градієнтною моделлю:

$$|\nabla f(x, y)| = \text{sqrt}((\partial f/\partial x)^2 + (\partial f/\partial y)^2). \quad (3.5)$$

Текстура ж описується ансамблем періодичних або квазівипадкових коливань. Для її моделювання застосовують просторові частотні функції:

$$t(x, y) \approx A \cos(\omega\_t\, x + \varphi), \quad (3.6)$$

де A — амплітуда, ω_t — частота текстури, φ — фаза.

Одним із ключових застосувань структурно-компонентної моделі є побудова алгоритмів згладжування, які зберігають контури та важливі локальні структури. Класичний вінерівський фільтр має вигляд:

$$H(\omega) = S\_x(\omega) / (S\_x(\omega) + S\_n(\omega)), \quad (3.7)$$

але він пригнічує не лише шум, а й корисні високочастотні компоненти зображення.

Використання складеної моделі дозволяє будувати адаптивні фільтри, які виконують згладжування окремо для кожної компоненти. Наприклад, однорідні області згладжують сильніше, тоді як компоненти границь обробляють за принципом мінімізації втрати контрасту:

$\min \| f(x, y) - \hat{f}(x, y) \|^2$, за умови збереження орієнтаційних характеристик $\nabla f(x, y)$.

Складена модель дозволяє:

- уникати надмірного розмиття країв;
- зберігати дрібні структурні елементи;
- відтворювати складні текстурні особливості;
- забезпечувати менше середньоквадратичне відхилення при реконструкції.

Структурно-компонентна (складена) модель фрагмента зображення є ефективним математичним апаратом, що дозволяє точно описувати локальні властивості зображення. Вона забезпечує підґрунтя для створення адаптивних

алгоритмів згладжування, здатних зберігати критично важливі деталі. Таким чином, модель має велике практичне значення для систем комп'ютерного бачення, медичної візуалізації та інженерної графіки.

3.3. Алгоритм згладжування

Алгоритми згладжування зображень є важливою складовою цифрової обробки сигналів. Їхнє завдання - пригнічення шуму без суттєвої втрати інформативних структур, таких як краї, контури, локальні деталі. У цьому розділі представлено приклади обробки реального зображення за допомогою різних методів згладжування.

Оригінальне зображення

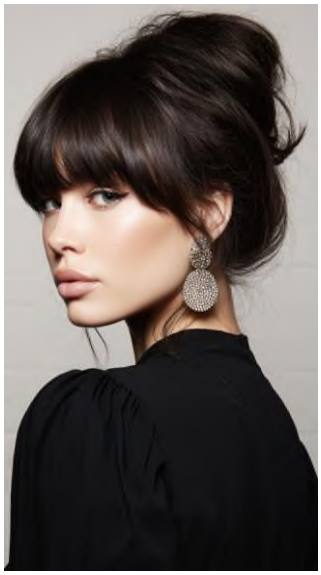

Рисунок 3.1 - Оригінальне зображення

Наведене зображення містить добре виражені структурні елементи: контури, тональні переходи та текстурні деталі, що дозволяє застосувати його як приклад для аналізу алгоритмів згладжування.

Математична модель

Будь-яке реальне цифрове зображення можна подати так: $f(x, y) = s(x, y) + n(x, y)$, де $s(x, y)$ — корисний сигнал (справжнє зображення), $n(x, y)$ — адитивний шум. Мета згладжування — отримати оцінку $\hat{s}(x, y)$, яка мінімізує квадратичну похибку:

$$E = \iint (f(x, y) - \hat{s}(x, y))^2 \, dx \, dy. \quad (3.8)$$

Лінійне згладжування виконується через операцію згортки:

$$\hat{s}(x, y) = \iint f(\xi, \eta)\, h(x - \xi, y - \eta)\, d\xi\, d\eta, \qquad (3.9)$$

де h(x, y) — згладжувальне ядро.

Рисунок 3.2 - Додавання шуму

Для демонстрації ефективності алгоритмів було штучно додано адитивний шум. Таке зображення дозволяє оцінити, наскільки фільтри пригнічують високочастотні компоненти.

Гаусівське згладжування

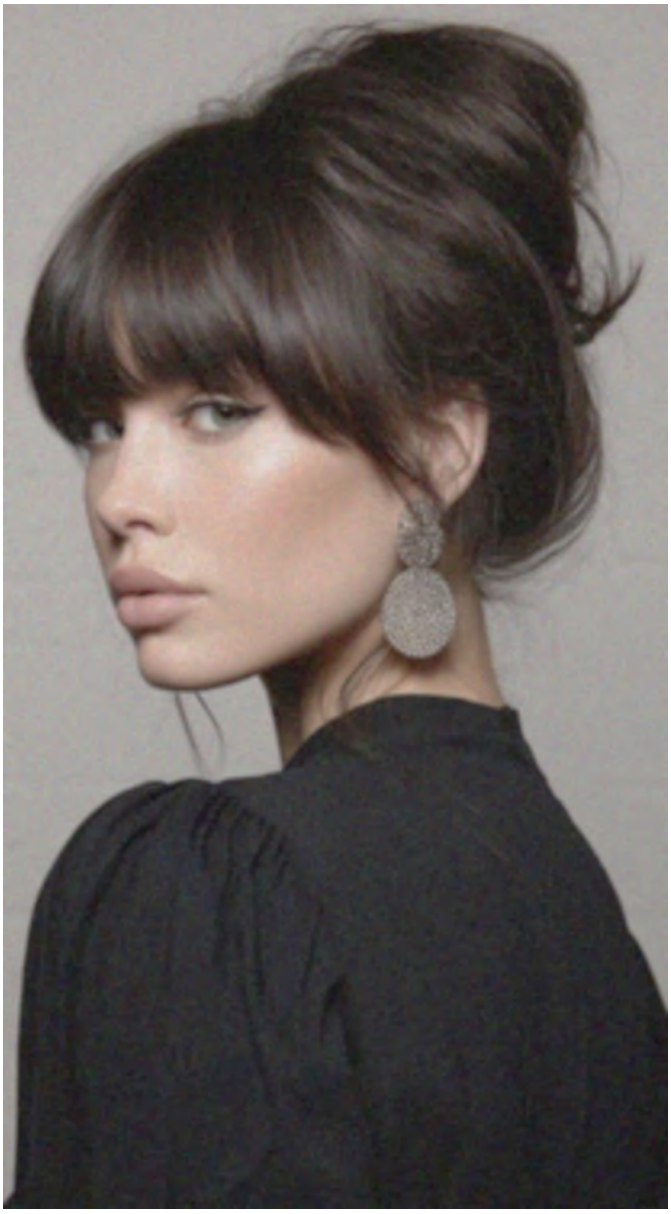

Рисунок 3.3 - Гаусівське згладжування

Гаусівський фільтр виконує згладжування за формулою:

$$h(x, y) = 1/(2\pi\sigma^2) * \exp(-(x^2 + y^2) / (2\sigma^2)) \quad (3.10)$$

Цей метод ефективно зменшує випадковий шум, але має суттєвий недолік - розмивання країв.

Медіанне згладжування

Рисунок 3.4 - Медіанне згладжування

Медіанний фільтр є нелінійним алгоритмом. Значення кожного пікселя замінюється медіаною вікна:

$$\hat{s}(x, y) = \text{median}\{ f(i, j) \} \quad (3.11)$$

Цей метод добре зберігає краї та ефективно пригнічує імпульсний шум. Для об’єктивної оцінки ефективності згладжування використовують статистичні метрики:

$$MSE = 1/(MN) \Sigma\Sigma (f - \hat{s})^2$$

$$PSNR = 10 \log10(255^2 / MSE) \quad (3.12)$$

У даному випадку медіанний фільтр краще зберігає структуру обличчя: краї, тонкі елементи зачіски, текстуру шкіри. Гаусівське згладжування забезпечує більш плавний тон, але втрачає різкість.

На основі порівняння можна зробити висновок, що вибір алгоритму згладжування залежить від типу шуму та завдання обробки. Для збереження деталей доцільно використовувати медіанні або адаптивні фільтри, тоді як гаусівське згладжування підходить для загального шумопригнічення [10].

Висновок до розділу

У цьому розділі було розглянуто теоретичні засади, математичні моделі та практичні аспекти застосування алгоритмів згладжування двовимірних зображень. Проведений аналіз дозволив встановити низку важливих положень.

По-перше, згладжування є базовою операцією цифрової обробки зображень, спрямованою на пригнічення високочастотних шумових компонентів при максимально можливому збереженні структурної інформації. Воно ґрунтується на моделі адитивного шуму та оптимізації середньоквадратичної похибки між вихідним і відновленим сигналами.

По-друге, результати експериментів підтвердили, що різні фільтри по-різному впливають на відновлення зображення. Гаусівський фільтр забезпечує ефективне загальне згладжування, проте розмиває контури та локальні деталі, що є суттєвим недоліком у випадках, коли важливими є точні переходи яскравості. Натомість медіанний фільтр продемонстрував кращу здатність до збереження країв завдяки своїй нелінійній природі й стійкості до імпульсних перешкод.

По-третє, практична частина розділу — обробка реального зображення — підтвердила теоретичні висновки: результати фільтрації суттєво залежать від локальної структури зони обробки, типу шуму та вибору параметрів фільтра. Додавання контрольованого шуму та подальша фільтрація продемонстрували відмінності в роботі фільтрів на текстурах, контурах та рівних поверхнях.

Таким чином, проведене дослідження дозволяє зробити узагальнений висновок, що універсального методу згладжування не існує: оптимальний вибір алгоритму визначається поставленим завданням, характеристиками шуму та вимогами до збереження деталей. Надалі доцільним є застосування адаптивних або комбінованих методів згладжування, які враховують локальні особливості оброблюваного зображення.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Внутрішня архітектура WebRTC

WebRTC (Web Real-Time Communication) — це сучасна технологія, яка забезпечує обмін аудіо-, відео- та іншими видами даних у режимі реального часу без необхідності встановлення додаткових плагінів або зовнішніх модулів. Вона розроблена як відкритий стандарт, що дозволяє інтегрувати мультимедійні можливості безпосередньо у веб-додатки й мобільні застосунки. Поява WebRTC сприяла швидкому розвитку сервісів відеозв'язку, онлайн-конференцій, систем дистанційного навчання та різноманітних комунікаційних платформ.

WebRTC побудовано на концепції прямого обміну даними між клієнтами (peer-to-peer), що зменшує затримки й підвищує швидкість передавання мультимедійного контенту. Однак створення прямого каналу часто є складним через мережеві обмеження, такі як NAT або брандмауери. Для їх подолання в WebRTC використовується сукупність серверних і мережевих компонентів. До архітектури входять: сигнальний сервер, STUN і TURN сервери, криптографічні механізми DTLS та SRTP, а також медійні сервери для групових конференцій.

Сигнальний сервер виконує роль координатора початкової взаємодії між клієтами. Він передає службову інформацію, необхідну для узгодження параметрів з'єднання, зокрема:

- SDP-описи (Session Description Protocol),
- ICE-кандидати з можливими мережевими маршрутами,
- конфігурації аудіо- та відеопотоків,
- інформацію про початок і завершення комунікаційної сесії.

Важливо підкреслити, що сигнальний сервер не бере участі в передачі мультимедійних даних. Він лише допомагає встановити P2P-з'єднання. У ролі сигналізаційних протоколів можуть використовуватися WebSocket, Socket.io, SIP, XMPP або інші мережеві технології, оскільки стандарт WebRTC не встановлює жорстких вимог до їхнього вибору.

Встановлення прямого зв’язку між клієнтами здійснюється за допомогою протоколу ICE (Interactive Connectivity Establishment), який збирає й аналізує можливі мережеві маршрути.

- STUN-сервер надає можливість дізнатися зовнішню IP-адресу клієнта, формуючи так звані server-reflexive кандидати.
- TURN-сервер використовується у ситуаціях, коли побудувати прямий канал неможливо — тоді медіапотоки передаються через ретрансляцію.

Процес ICE передбачає збір локальних (host), відбитих (srflx) і ретрансляційних (relay) маршрутів, а також тестування кожного з них для вибору найстабільнішого шляху з’єднання.

WebRTC передбачає обов’язкове шифрування всіх даних, які передаються між клієнтами. У технології застосовуються:

- DTLS (Datagram Transport Layer Security) — для захисту службових даних і встановлення безпечного каналу,
- SRTP (Secure Real-Time Transport Protocol) — для шифрування мультимедійних потоків.

Комбінація DTLS-SRTP забезпечує повну конфіденційність та цілісність переданих даних. Без активної системи шифрування WebRTC не дозволяє встановити медіасесію, що забезпечує високий рівень безпеки.

Архітектуру WebRTC можна описати як узгоджену взаємодію кількох ключових елементів, які забезпечують повний цикл обробки, передавання та захисту мультимедійних даних. До складу цієї архітектури входять клієнтські веб- та мобільні застосунки, що ініціюють і приймають мультимедійні потоки.

Важливою складовою є сигнальний сервер, який використовується для обміну описами сесій (SDP) та мережевою інформацією у форматі ICE, необхідною для встановлення з’єднання.

Функціонування P2P-комунікацій стає можливим завдяки STUN- і TURN-серверам, що забезпечують визначення зовнішніх адрес клієнтів та організацію ретрансляційних каналів у випадках, коли прямий обмін даними неможливий. У

сценаріях багатосторонніх конференцій застосовуються медійні сервери типу SFU або MCU, які відповідають за маршрутизацію або мікшування потоків між великою кількістю учасників.

Захист переданих даних гарантується за допомогою криптографічних механізмів DTLS-SRTP, які забезпечують шифрування сигналів та конфіденційність комунікації. Доповнює систему медійний стек, що виконує обробку аудіо- та відеосигналів на стороні клієнта. Скоординована робота всіх перелічених компонентів створює гнучку та надійну інфраструктуру, здатну забезпечувати стабільні, безпечні та високоякісні мультимедійні з'єднання в реальному часі.

Загальна архітектура WebRTC (рис. 4.1).

Рисунок 4.1 - Загальна архітектура WebRTC

Основу програмного стека WebRTC становить **C++ API**, який включає набір низькорівневих бібліотек та інтерфейсів, реалізованих мовою C++. Модель передбачає, що розробники браузерів та клієнтських застосунків інтегрують ці C++-компоненти у свої системи, а користувачам надається доступ до їхнього

функціоналу через високорівневий JavaScript-інтерфейс Web API. Саме цей набір бібліотек формує ключову функціональність WebRTC і забезпечує роботу всіх комунікаційних модулів.

Транспортні модулі відповідають за встановлення, підтримку та управління комунікаційними сесіями між клієнтами. Вони є розвитком та розширенням компонентів Libjingle. Цей рівень забезпечує:

- механізми обходу NAT, проксі та брандмауерів за допомогою STUN/TURN-серверів;
- узгодження параметрів з'єднання перед передаванням даних;
- передавання медіапотоків через RTP або SRTP поверх TCP/UDP;
- шифрування даних у SRTP;
- контроль пропускної здатності та адаптацію до мережевих умов.

Підсистема обробки аудіо.

Аудіомодуль відповідає за приймання й передавання аудіосигналу між мережею та аудіопристроями користувача. До його складу входять:

- аудіокодеки **iSAC** та **Opus**;
- системи шумозаглушення та подавлення ехо;
- механізми автоматичного визначення параметрів захоплення аудіо.

Кодек **iSAC** є широкосмуговим рішенням для VoIP та потокової передавання аудіо, працює на частотах 16/32 кГц із динамічною зміною бітрейту в межах 12–52 Кбіт/с.

Кодек **Opus** підтримує як фіксований, так і змінний бітрейт у діапазоні 6–510 Кбіт/с та частоти дискретизації 8–48 кГц, що забезпечує високу якість навіть у нестійких мережевих умовах.

Відеомодуль керує формуванням, кодуванням та оптимізацією відеопотоків. Він включає:

- відеокодек **VP8**;
- адаптивний буфер **JitterBuffer**, який коригує стрибки затримки;
- механізми корекції та покращення якості зображення.

Кодек VP8 відзначається високою стійкістю до втрати пакетів, наявністю алгоритмів фільтрації артефактів залежно від змін між кадрами, можливістю масштабованого декодування та спеціальними профілями для роботи у режимі реального часу, що робить його оптимальним для відеоконференцій.

Основні інтерфейси C++ API

C++ API WebRTC базується на двох ключових інтерфейсах:

1. Stream API

Забезпечує:

- доступ до аудіо- та відеопотоків з апаратури клієнта (мікрофон, вебкамера);
- перетворення локальних даних у медіапотоки;
- можливість трансляції заздалегідь підготовлених файлів;
- обробку аудіо та відео перед передачею.

2. PeerConnection API

Відповідає за:

- ініціалізацію та узгодження параметрів P2P-з’єднання;
- вибір маршрутів обходу NAT, брандмауерів чи проксі;
- організацію та підтримку медіапередавання між клієнтами.

PeerConnection фактично є ядром WebRTC-логіки, оскільки виконує повний цикл встановлення та підтримки мережевої сесії.

Загальна схема роботи(рис 4.2).

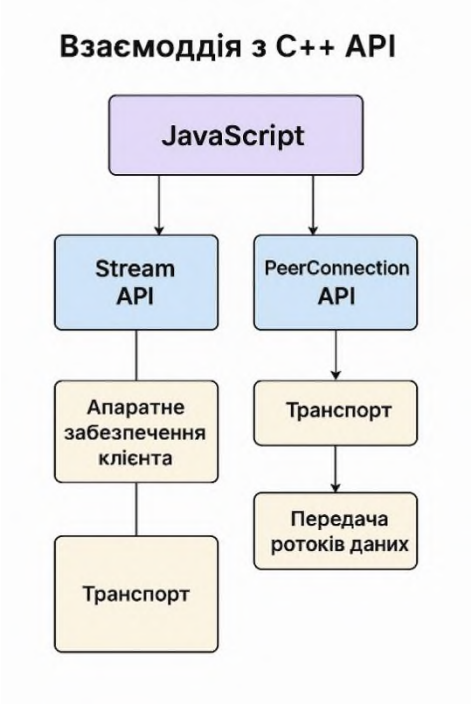

Рисунок 4.2 - Схема взаємодії з C++ API

Розробники можуть інтегрувати C++ API WebRTC безпосередньо у свої програмні продукти, відкриваючи доступ до його можливостей через високорівневий JavaScript-інтерфейс Web API. Останній визначається стандартом W3C, який у поточний час представлений стабільною специфікацією версії 1.0. Завдяки цьому кінцевому користувачеві не потрібно заглиблюватися у внутрішню структуру WebRTC: для взаємодії з можливостями технології достатньо працювати з Web API, зосереджуючи увагу виключно на логіці й архітектурі власного додатка [17].

У межах цієї роботи розглядаються саме методи Web API, оскільки практична частина ґрунтуватиметься на механізмах, що реалізовані через цей інтерфейс. Подібно до C++ API, Web API включає два ключові інтерфейси — MediaStream та RTCPeerConnection.

Інтерфейс MediaStream забезпечує отримання та передачу мультимедійних потоків, формуючи єдиний потік з даних, отриманих від різних пристроїв введення (мікрофонів, камер тощо). Такий підхід дозволяє оперувати кількома

незалежними потоками одночасно, що є важливим для мобільних пристроїв, які можуть мати декілька камер або аудіоджерел.

Інтерфейс RTCPeerConnection відповідає за встановлення прямого каналу зв'язку між клієнтами та управління процесом передавання потоків. Він асоціює вхідні та вихідні потоки з конкретними з'єднаннями: вихідні дані передаються іншим учасникам сесії через RTCPeerConnection, тоді як отримані потоки передаються клієнту для подальшої обробки та відтворення.

4.2 Застосування Web API

4.2.1 Створення підключення

У межах технології WebRTC Web API виконує роль високорівневого інтерфейсу, який надає розробникам інструменти для створення та керування мультимедійними з'єднаннями у реальному часі. На відміну від внутрішніх компонентів, реалізованих мовою C++, Web API пропонує уніфікований набір абстракцій — таких як RTCPeerConnection, MediaStream, MediaStreamTrack та RTCDataChannel — що дозволяють працювати з аудіо-, відео- та дата-потоками без необхідності взаємодії з низькорівневими мережевими механізмами. Через ці інтерфейси здійснюється ініціалізація з'єднання, узгодження параметрів сесії, додавання мультимедійних доріжок, а також обмін службовими повідомленнями та довільними даними між клієнтами [15].

Створення WebRTC-підключення є багатокроковим процесом, у якому беруть участь системи сигналізації, механізм ICE, інфраструктура STUN/TURN-серверів, методи шифрування DTLS-SRTP та алгоритми обробки медіапотоків. Хоча у Web API цей процес виглядає як послідовність методів, за кожним кроком приховано складну взаємодію протоколів та мережевих процедур. Головною метою є встановлення захищеного peer-to-peer каналу між браузерами або іншими клієнтами, через який можна передавати аудіо-, відео- та дата-потоки з мінімальною затримкою.

Центральним елементом у процесі встановлення з'єднання є RTCPeerConnection. Саме він відповідає за формування та обробку SDP-описів,

збір і тестування ICE-кандидатів, встановлення DTLS-каналу та передавання SRTP-пакетів. Додатково до нього використовуються інтерфейси MediaStream та MediaStreamTrack, які забезпечують доступ до камер, мікрофонів та інших мультимедійних пристроїв користувача, формуючи структуровані потоки для передавання мережею. Процес створення WebRTC-з'єднання можна поділити на кілька логічних етапів. Першим є ініціалізація RTCPeerConnection, для чого викликається відповідний конструктор, який приймає об'єкт конфігурації з переліком STUN- і TURN-серверів. Саме ці сервери забезпечують механізми визначення зовнішньої адреси користувача та маршрутизації трафіку у випадках, коли пряме підключення заблоковане NAT або фаєрволом. Правильне налаштування iceServers є критично важливим, оскільки від нього залежить можливість встановлення стабільного та оптимального маршруту між клієнтами [16].

Наступним етапом є отримання локальних медіапотоків за допомогою методу getUserMedia(), який повертає об'єкт MediaStream. Потік може містити кілька доріжок — аудіо, відео або екранний контент. Додані доріжки інтегруються до RTCPeerConnection через методи addTrack або addStream. Браузер перед початком роботи обов'язково запитує дозвіл користувача, що є ключовим аспектом конфіденційності.

```
v=0
o=- 1758966045304012812 2 IN IP4 127.0.0.1
s=-
t=0 0
a=group:BUNDLE audio video
a=msid-semantic: WMS mcjw5hlyaawO3UAD5eCoRMbuVjJJPAKuspWv
m=audio 1 RTP/SAVPF 111 103 104 0 8 106 105 13 126
c=IN IP4 0.0.0.0
a=rtcp:1 IN IP4 0.0.0.0
a=ice-ufrag:rQ8nRK34upAOZ5Mq
a=ice-pwd:wpxbDUvRlBXHSxGhGzeoDiez
a=ice-options:google-ice
a=fingerprint:sha-256 24:35:C4:8A:13:51:B9:C6:EB:C4:2F:01:A1:BA:91:7F:90:DB:DD:64:7C:49:F9:66:0A:2C:62:B1:93:6B:05:F3
a=setup:actpass
a=mid:audio
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=extmap:3 http://www.webrtc.org/experiments/rtp-hdrext/abs-send-time
a=sendrecv
a=rtcp-mux
a=rtpmap:111 opus/48000/2
a=fmtp:111 minptime=10
a=rtpmap:103 ISAC/16000
a=rtpmap:104 ISAC/32000
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
```

Рисунок 4.3 - Приклад SDP при координації WebRTC-з'єднання.

Після підготовки локальних потоків ініціатор з’єднання формує SDP-опис (offer) через метод createOffer(). Він містить інформацію про параметри медіасесії: перелік доступних кодеків, типи носіїв, параметри шифрування, транспортні протоколи та інші метадані. Згенерований offer встановлюється як локальний опис і передається іншому клієнту через сигнальний сервер. Другий клієнт, отримавши пропозицію, застосовує її за допомогою методу setRemoteDescription() і формує SDP-відповідь (answer) через createAnswer(). Після цього відповідь встановлюється як локальний опис і відправляється назад ініціатору. Обидві сторони мають взаємні SDP-описи, що забезпечує узгодження параметрів сесії.

Критично важливим етапом є збір і обмін ICE-кандидатами. Механізм ICE намагається знайти найоптимальніший шлях для підключення, послідовно перевіряючи локальні адреси, server-reflexive адреси, отримані зі STUN, а за потреби — relay-адреси через TURN. Якщо пряме з’єднання неможливе, трафік передається через TURN-сервер. Після успішного узгодження маршрутів встановлюється захищений DTLS-канал, на основі якого конфігурується SRTP. Усі мультимедійні потоки шифруються, що гарантує конфіденційність та захист від атак. Після цього RTCPeerConnection переходить у активний стан, у якому можлива передача аудіо-, відео- та дата-потоків. Вхідні потоки обробляються через подію ontrack, а додаткові дані можуть передаватися через RTCDataChannel. У випадку помилок або змін стану мережі застосунок може реагувати через події oniceconnectionstatechange та onconnectionstatechange, забезпечуючи стабільність роботи [17].

Для цього необхідно задати об'єкту RTCPeerConnection список STUN та TURN серверів(рис.4.4).

```
var ICEServers = {
    iceServers: [
        { url: 'stun:stun4.lgoogle.com:19302' },
        { url: 'turn:turn.anyfirewall.com:3478?transport=udp',
            username: 'webrtckamtest',
            credential: 'kfSDjHJ' },
        { url: 'turn:turn.anyfirewall.com:443?transport=tcp',
            username: 'webrtckamtest',
            credential : 'kfSDjHJ' }
    ]
};
var Peer = new RTCPeerConnection(ICEServers, optional)
```

Рисунок 4.4 - Опис STUN і TURN серверів.

Отже, процес створення WebRTC-підключення — це послідовність чітко організованих кроків, які приховують значний обсяг внутрішньої логіки. Web API дозволяє розробникам працювати на високому рівні абстракції, не занурюючись у складні мережеві механізми, що робить WebRTC універсальним інструментом для створення сучасних систем комунікації реального часу.

У межах сучасних вебкомунікацій WebRTC є фундаментальною технологією, що забезпечує передачу аудіо-, відео- та дата-потоків у режимі реального часу без необхідності встановлення додаткових плагінів. Основним інструментом для взаємодії з WebRTC у браузері є Web API, який спрощує роботу розробника, інкапсулюючи складні мережеві механізми у високорівневі інтерфейси.

Процес створення WebRTC-підключення складається з кількох ключових етапів: отримання локальних медіапотоків, ініціалізація RTCPeerConnection, обмін SDP-описами, збір і тестування ICE-кандидатів, встановлення захищеного каналу DTLS-SRTP та запуск активної передачі мультимедійного контенту. Інтерфейс RTCPeerConnection є центральною компонентою, що керує логікою мережевого підключення. Його завданням є узгодження параметрів сесії, маршрутизація пакетів, встановлення зашифрованих каналів та інтеграція аудіо- й відеодоріжок.

Налаштування підключення виконується через параметр iceServers, що містить STUN- і TURN-сервери, потрібні для обходу NAT та мережевих обмежень.

Особливо важливим є механізм ICE (Interactive Connectivity Establishment), який виконує пошук оптимального маршруту між клієнтами. ICE послідовно перевіряє локальні адреси пристрою, public-адреси зі STUN-сервера та relay-адреси через TURN. Якщо пряме з'єднання недоступне (наприклад, через суворий NAT), WebRTC автоматично переходить до використання TURN-сервера, забезпечуючи гарантоване, хоч і повільніше, з'єднання.

Медіапотоки отримуються за допомогою функції getUserMedia(), яка надає доступ до камер, мікрофонів або екрану користувача. Після цього отримані доріжки додаються до RTCPeerConnection, що дозволяє передавати їх між клієнтами. На етапі сигналізації клієнти обмінюються SDP-описами - текстовими документами, що містять інформацію про кодеки, типи потоків, транспортні протоколи й параметри синхронізації.

Після взаємного обміну SDP-файлами браузери запускають збір ICE-кандидатів. Кожен новий кандидат передається через сигнальний сервер іншому клієнту, після чого обидві сторони здійснюють тестування на доступність маршруту. Успішне тестування призводить до встановлення медіаканалу [18].

Для шифрування даних використовується поєднання DTLS (для ключів та службових даних) і SRTP (для аудіо- та відеопакетів). Це гарантує захист від підміни, перехоплення або модифікації потоку. Нижче наведено приклади коду, які демонструють базові етапи встановлення WebRTC-з'єднання.

```js
async function startCall() {
  // 1. Ініціалізація локального медіа та додавання доріжок
  await initLocalMedia();

  // 2. Створення SDP-пропозиції (offer)
  const offer = await pc.createOffer();

  // 3. Встановлення локального опису
  await pc.setLocalDescription(offer);

  // 4. Передача SDP offer іншому клієнту через сигнальний сервер
  sendSignal({
    type: 'offer',
    sdp: offer.sdp
  });
}

// Обробка отриманої answer від іншого клієнта
async function handleAnswer(sdp) {
  const answerDesc = new RTCSessionDescription({
    type: 'answer',
    sdp
  });

  await pc.setRemoteDescription(answerDesc);
```

Рисунок 4.5 - Сторона ініціатора: створення offer, встановлення localDescription, відправка через signaling

```js
async function handleOffer(sdp) {
  // 1. Підготувати локальні медіапотоки
  await initLocalMedia();

  // 2. Встановити віддалений опис як offer
  const offerDesc = new RTCSessionDescription({
    type: 'offer',
    sdp
  });
  await pc.setRemoteDescription(offerDesc);

  // 3. Створити SDP-відповідь (answer)
  const answer = await pc.createAnswer();

  // 4. Встановити локальний опис
  await pc.setLocalDescription(answer);

  // 5. Відправити answer через сигнальний канал
  sendSignal({
    type: 'answer',
    sdp: answer.sdp
  });
}
```

Рисунок 4.6 - Сторона відповідача: обробка offer, створення answer, встановлення описів

4.2.2 Надсилання та отримання потоків

Для передавання локального медіапотоку, отриманого в результаті виклику getUserMedia(), до віддаленого клієнта необхідно додати цей потік до екземпляра RTCPeerConnection. Це здійснюється за допомогою методу addStream, якому передається відповідний об’єкт потоку.

```js
var Peer = new RTCPeerConnection(ICEServers, optional);

var constraints = { audio: true, video: true };

function success(stream) {
    // Peer.addStream(stream);
}

function error() {
    // ...
}

navigator.webkitGetUserMedia(constraints, success, error);
```

Рисунок 4.7 - Надсилання потоку в RTCPeerConnection

Після завершення процедури сигналізації та успішного створення екземпляра RTCPeerConnection, віддалена сторона отримує медіапотік через активацію події onaddstream. У межах цього обробника браузер передає об’єкт, що містить отриманий від іншого клієнта потік разом із супровідною інформацією про встановлене з’єднання.

```js
Peer.onaddstream = function(event) {
    var stream = event.stream;
    var streamId = stream.id;

    // Відображення отриманого відеопотоку
    video.src = webkitURL.createObjectURL(stream);

    stream.onended = function() {
        // Обробка завершення потоку
    };
};
```

Рисунок 4.8 - Обробка для відлову вхідного потоку

Таким чином, застосування базових механізмів та інтерфейсів Web API, що входять до складу технології WebRTC, дає змогу організувати повноцінний сеанс потокової передачі мультимедійних даних між клієнтами. Використовуючи стандартні засоби доступу до медіапристроїв, формування та обміну SDP-описами, а також процедури встановлення однорангового з’єднання, розробник може реалізувати простий, але функціонально завершений канал передавання аудіо-, відео- або інших даних у режимі реального часу. Такий підхід забезпечує мінімальні затримки, високий рівень сумісності між різними браузерами та достатній рівень безпеки завдяки застосуванню вбудованих криптографічних механізмів.

4.3 Програмна реалізація на основі Web API

4.3.1 Web-додаток, який інтегрується на сторонній ресурс

WebRTC надає можливість встановлювати відеозв'язок, використовуючи JavaScript та HTML5, що означає, що клієнтський додаток може бути інтегрований безпосередньо на веб-сайт або веб-сторінку. Першою практичною реалізацією цієї технології була спроба створити переносний веб-додаток, який може бути вбудований на сторонній веб-сайт для специфічних клієнтів. Це стає корисним, якщо потрібно передавати дані між клієнтами конкретного ресурсу, але немає доступу до вихідного коду цього ресурсу і немає можливості його

змінювати. Наприклад, це може бути корисно, коли функціонал розробляється стороннім розробником або клієнтами ресурсу. Для вирішення цього завдання була обрана соціальна мережа. З метою створення інтеграції програми на сайті було обрано вбудувати її в браузер як розширення, яке виконує код при відвідуванні сайту, а основний виконуваний код WebRTC-програми розміщується на сторонньому сховищі і завантажується в iframe. Загальна структура додатка має вигляд, зображений рисунку 4.9.

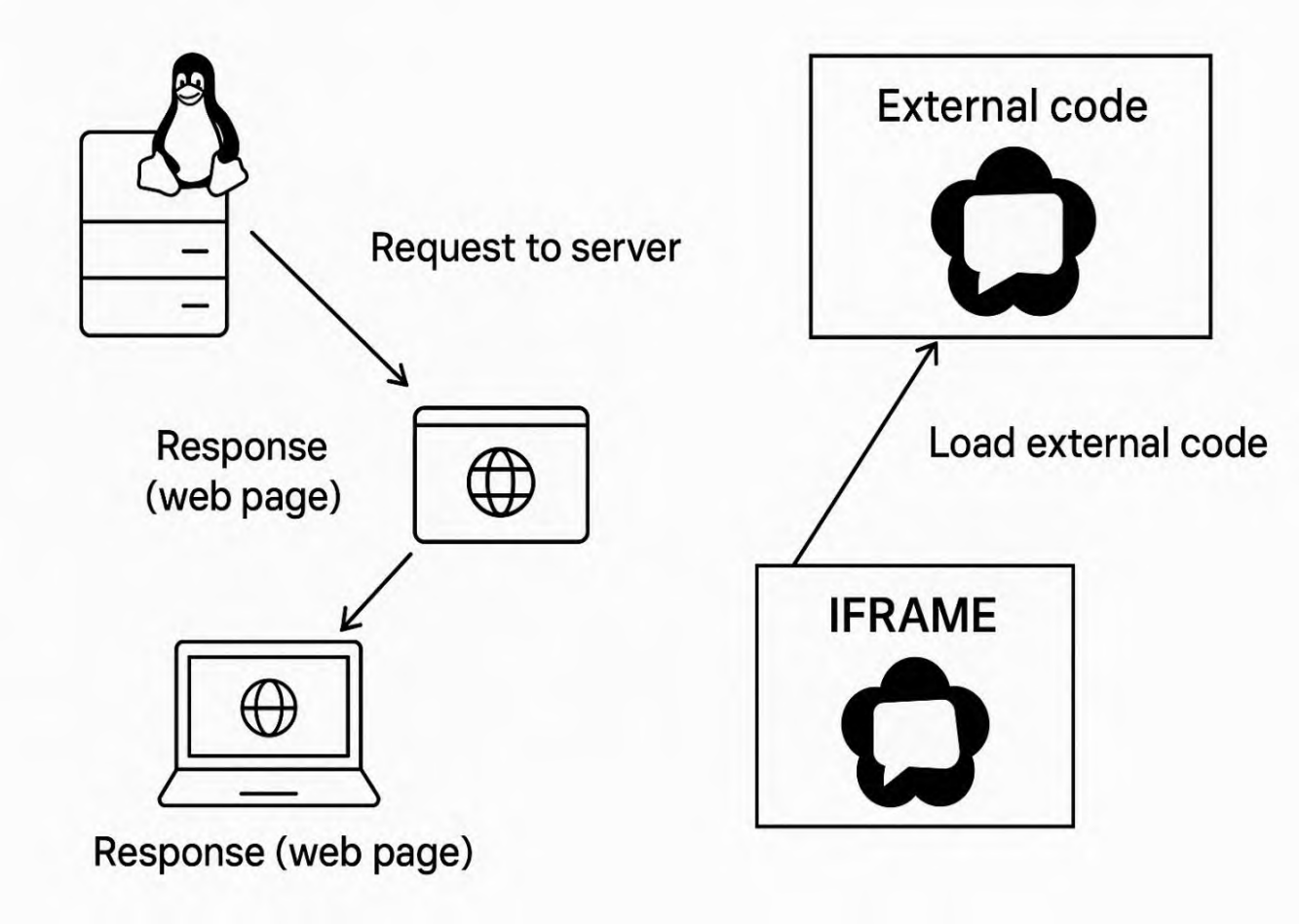

Рисунок 4.9 - Структура браузерного додатку

Для організації обміну службовими SDP-повідомленнями в межах WebRTC було використано хмарну базу даних Firebase, яка виконує функцію сигнального каналу. Такий підхід дозволяє реалізувати механізм сигналізації без окремого WebSocket-сервера, оскільки Firebase забезпечує миттєву синхронізацію даних між клієнтами. Розміщення JavaScript-коду на Google Drive

дало змогу забезпечити його доступність через захищений протокол HTTPS, що є обов’язковою умовою для роботи більшості компонентів WebRTC.

Розроблена система включає WebRTC-додаток, який завантажується з хмарного сховища та виконується у контексті браузера. На рівні клієнтської частини реалізовано інтеграцію з браузерним розширенням, що дозволяє взаємодіяти зі сторінками вебсайту, отримувати необхідні дані та передавати їх до зовнішнього WebRTC-модуля. HTML-структура цільових сторінок аналізувалася для коректного вилучення потрібних елементів та автоматизації взаємодії.

Для забезпечення багатостороннього потокового зв’язку була створена спеціальна JavaScript-бібліотека RTCPeerBroadcasting. Вона містить набір функцій, що полегшують встановлення WebRTC-з’єднань у режимі конференції та організацію передачі аудіо- і відеопотоків. Центральним елементом бібліотеки є функція-обгортка над RTCPeerConnection, яка створює та налаштовує підключення між клієнтами. Параметри цієї функції можуть містити дані про STUN/TURN-сервери, обмін SDP-описами та інші характеристики сесії [15].

Створений об’єкт підключення використовується для обміну медіаданими, узгодження параметрів сесії та підтримання стабільної WebRTC-комунікації між учасниками. Завдяки цьому забезпечується можливість реалізації потокового відеозв’язку та передачі інших мультимедійних даних у режимі реального часу.

```js
var peerConfig = {
    attachStream: config.stream,

    onICE: function(candidate) {
        //
    },

    onRemoteStream: function(stream) {
        //
    },

    onRemoteStreamEnded: function(stream) {
        //
    },

    onOfferSDP: sendSDP
};
```

Рисунок 4.10 - Об'єкт параметрів функції RTCPeerConnection

Метод onICE виконує роль обробника подій, що реагує на появу нового ICE-кандидата та забезпечує його передавання іншому клієнту під час встановлення WebRTC-з’єднання. Окрім цього, реалізовано низку додаткових функцій-обробників: onRemoteStream, який активується при отриманні віддаленого медіапотоку; onRemoteStreamEnded, що спрацьовує після завершення потоку; а також onOfferSDP і onAnswerSDP, відповідальні за обробку службових SDP-повідомлень.

Під час виклику функції RTCPeerConnection створюється новий об’єкт підключення WebRTC, ініціюється процес встановлення з’єднання з іншим клієнтом та, за необхідності, передається локальний медіапотік. Після того як віддалений клієнт підтвердить прийом з’єднання й поверне SDP-відповідь, викликається функція addAnswerSDP, яка зберігає отримане SDP-повідомлення та завершує процедуру встановлення з’єднання.

```js
addAnswerSDP: function(SDP) {
    var session = new SessionDescription(SDP);
    Peer.setRemoteDescription(session, onSuccess, onError);
}
```

Рисунок. 4.11 - Збереження SDP

Функції onOfferSDP та onAnswerSDP задаються не всередині самої реалізації PeerConnection, а передаються під час її виклику. Такий підхід забезпечує вищий рівень гнучкості та переносимості бібліотеки, оскільки механізм обміну SDP-повідомленнями не нав’язується розробнику. Через те, що стандарт WebRTC не містить вбудованих засобів сигналізації, спосіб передавання SDP може визначатися самостійно — за допомогою вебсокетів, хмарної бази даних, HTTP-запитів або окремого сигнального сервера. Таким чином, бібліотека залишається незалежною від конкретної інфраструктури та може бути інтегрована в різні архітектурні рішення.

Функція Broadcasting виступає центральним елементом класу RTCPeerBroadcasting. Вона виконує роль обгортки над RTCPeerConnection, забезпечуючи додатковий рівень абстракції та спрощуючи процес створення багатостороннього підключення. Завдяки цьому розробник може організувати конференц-зв’язок, у якому бере участь більше ніж два клієнти, без необхідності вручну керувати множиною однорангових з’єднань. На вхід функція отримує структурований об’єкт параметрів, загальну форму якого продемонстровано на відповідному рисунку 4.12.

```js
config = {
    openSocket: function(nConfig) {
        //
    },

    onRemoteStream: function(media) {
        //
    },

    onRemoteStreamEnded: function(media) {
        //
    },

    connectToRoom: function(roomName) {
        //
    }
};
```

Рисунок 4.12 - Об'єкт параметрів Broadcasting

У наведеному конфігураційному об’єкті функція openSocket відповідає за ініціалізацію каналу зв’язку, через який здійснюється обмін SDP-повідомленнями між клієнтами. Методи onRemoteStream та onRemoteStreamEnded виступають обробниками відповідно початку та завершення отриманого медіапотоку. Функція connectToRoom викликається у випадку підключення користувача до вже створеної конференції.

Під час запуску функції Broadcasting ініціюється надсилання SDP-опису через сокет та виконується очікування на відповідь від іншого учасника. Якщо відповідь отримано, формується новий екземпляр RTCPeerConnection, який забезпечує встановлення прямого одноранговогo з’єднання з новим клієнтом. Результатом роботи функції є об’єкт, що містить набір службових методів: createRoom, joinRoom та leaveRoom [14].

Метод createRoom створює нову конференцію з визначеною назвою та відправляє на сокет службове повідомлення про її ініціалізацію. Метод joinRoom використовується для підключення до вже існуючої конференції; у процесі виклику створюється новий екземпляр RTCPeerConnection, який забезпечує

взаємодію з іншими учасниками. Метод leaveRoom призначений для завершення участі в конференції: він проходить циклом по всіх активних об'єктах RTCPeerConnection та викликає метод close, який коректно закриває кожне з'єднання.

Таким чином, дана бібліотека надає простий та інтуїтивно зрозумілий інтерфейс для створення та обслуговування багатосторонніх WebRTC-сесій. Вона виступає базовим компонентом WebRTC-додатка, що інтегрується у вигляді браузерного розширення на сторінки вебресурсу. У процесі розроблення застосунку необхідно також сформувати користувацький інтерфейс та додати логіку здійснення та приймання відеодзвінків.

Оскільки WebRTC-модуль завантажується у межах iframe, який розміщено на сторінці з іншим доменом, важливо забезпечити коректний обмін даними між батьківським вікном та вмістом iframe. Для цього використано механізм postMessage, що дозволяє передавати рядкові повідомлення між різними доменами. Перед надсиланням масиви даних кодуються у формат JSON, а після отримання — декодуються на стороні приймача. Такий підхід забезпечує безпечну та коректну взаємодію між компонентами системи в межах браузера.

```javascript
window.onmessage = function(ev) {
    var arr = JSON.parse(ev.data);
    userId = arr["id"];
    userName = arr["name"];
    userPic = arr["pic_link"];
    /*
    ...
    */
};
```

Рисунок 4.13 - Вилов POST-повідомлення

Після надходження POST-повідомлення, яке містить службову інформацію-зокрема ідентифікатори абонента та користувача, якому телефонують, посилання на зображення профілю та ім'я, - скрипт ініціює нову сесію зв'язку. Для цього він

викликає метод createRoom бібліотеки RTCPeerBroadcasting. Назва створюваної конференції встановлюється відповідно до ідентифікатора користувача, що має прийняти виклик. Одразу після створення конференції відомості про запит дзвінка фіксуються у базі даних, яка слугує центральним елементом сигналізації.

Якщо адресат перебуває в мережі, він миттєво отримує повідомлення про вхідний виклик через підписку на відповідну гілку бази даних у режимі реального часу. У разі відхилення дзвінка система формує відповідне службове повідомлення та зберігає його, щоб ініціатор міг побачити результат. Якщо ж користувач приймає дзвінок, браузерне розширення завантажує у iframe сторінку зі вбудованою WebRTC-програмою та надсилає повідомлення про підтвердження. Після цього запускається метод joinRoom бібліотеки RTCPeerBroadcasting, який підключає користувача до створеної кімнати та ініціює встановлення прямого peer-to-peer з’єднання.

Система також керує статусами присутності. Коли користувач відкриває сторінку сайту, інформація про його онлайн-стан записується в базу даних, а при закритті вкладки або завершенні сесії ці дані автоматично видаляються. Для отримання додаткових даних профілю - таких як ім’я, унікальний ідентифікатор та посилання на аватар — виконується асинхронний HTTP-запит за адресою виду 192.168.1.1/id0. Після цього система автоматично перенаправляє браузер на потрібну сторінку користувача. Аналіз HTML-структури сторінки здійснюється за допомогою вибірки через CSS-селектори, що дозволяє витягнути всі необхідні атрибути та використати їх у WebRTC-інтерфейсі.

```
httpReq("https://192.168.1.1/id0", "GET", null, "document", function(res) {
    var newDiv = res.body;

    if (newDiv.querySelector(".page_name.fl_l.ta_l")) {
        var User_name = newDiv.querySelector(".page_name.fl_l.ta_l").innerHTML;
        store("user_name", User_name);
    }

    try {
        var img = newDiv.querySelector("a[id='profile_photo_link']");
        var User_pic_link = img.querySelector("img").src;
    } catch (e) {
        User_pic_link = img_noavatar;
    }

    store("user_pic_link", User_pic_link);

    /*
    ...
    */
});
```

Рисунок 4.14 - Парсинг отриманої сторінки

Після успішного отримання основних даних про користувача в базі даних активується спеціальний обробник, який відстежує появу нових записів, що сигналізують про вхідний відеодзвінок. У момент надходження такого повідомлення інтерфейс розширення формує та відображає діалогове вікно із кнопками для прийняття або відхилення виклику.

Паралельно виконується аналіз URL поточної вебсторінки. Це дає змогу визначити, чи перебуває користувач на профілі іншої особи. Якщо так, зі сторінки автоматично зчитуються дані про цього користувача, зокрема ідентифікатор та допоміжні параметри, необхідні для перевірки наявності встановленого розширення WebRTC.

У випадку, коли відвідуваний користувач також має активоване розширення, інтерфейс доповнюється спеціальною кнопкою, що дозволяє ініціювати відеодзвінок. Після натискання цієї кнопки в базу даних надсилається службове повідомлення про початок виклику. Одночасно на сторінці динамічно

створюється та завантажується iframe, у якому розгортається WebRTC-програма. До неї додатково передається POST-повідомлення, що містить параметри, необхідні для встановлення з'єднання.

```javascript
var win = myFrame.contentWindow;

var dataArr = {
    "id":  store("user_id"),
    "name": store("user_name"),
    "pic_link": store("user_pic_link"),
    "st": status
};

win.postMessage(JSON.stringify(dataArr), "*");
```

Рисунок 4.15 - Надсилання POST-повідомлення

Для коректного використання методу postMessage() необхідно вказати другий параметр — домен, на який дозволено надсилати повідомлення. У даному випадку використовується символ "*", що означає дозвіл на передачу даних у межах будь-якого домену. Такий підхід обумовлений тим, що iframe у даному застосунку завантажується з іншого сервера, а отже, має іншу політику походження (origin), що унеможливлює обмін даними без явного дозволу [12].

Коли користувач приєднується до конференції, відповідний статус відображається на вебсторінці. Для інших учасників системи традиційна кнопка ініціювання дзвінка замінюється на елемент інтерфейсу, який дозволяє під'єднатися до вже створеної конференції. Під час такого підключення, аналогічно до процедури прийняття виклику, у сторінку динамічно завантажується iframe, після чого здійснюється спроба з'єднання із сигнальним каналом через метод joinRoom бібліотеки RTCPeerBroadcasting.

Окрім візуального повідомлення про вхідний виклик, було необхідно передбачити й звукове сповіщення, яке користувач може налаштовувати індивідуально. Для цього використано можливості Audio API, який забезпечує

відтворення звукових файлів як за прямим посиланням з Інтернету, так і у вигляді локальних аудіоданих (наприклад, файлів MP3).

Для реалізації персоналізації рінґтону була впроваджена логіка автоматичного виявлення аудіозаписів на сторінці за допомогою селектора audio. Під час знаходження аудіоелементів до них динамічно додаються спеціальні кнопки, що дозволяють зберігати обрану композицію у якості звуку виклику. Таким чином, користувач отримує можливість самостійно вибирати мелодію сповіщення, що підвищує зручність і персоналізованість роботи з додатком.

```javascript
var audios = document.querySelectorAll(".audio");

for (var i = 0; i < audios.length; i++) {
    if (!audios[i].querySelector(".vd_call_set_as_call_sound_but")) {
        //
        var actDiv = audios[i].querySelector("div[class='actions']");
        //
        var sndPath = audios[i].querySelector("input[type='hidden']");
        sndPath = sndPath.value.split(",")[0];
        /*
        ...
        */
    }
}
```

Рисунок 4.16 - Пошук аудіо-елементів

Після натискання відповідної кнопки вибране аудіо автоматично зберігається як сигнал вхідного виклику. До браузерного розширення було додано спеціальну сторінку налаштувань, на якій користувач може обрати потрібний рінґтон із доступного списку аудіозаписів.

Після завершення реалізації програмної логіки та проведення комплексного тестування було розроблено повноцінний вебінтерфейс, що охоплює всі компоненти застосунку. Вигляд основних елементів фінальної версії програми наведено на рисунках 4.17-4.19.

Рисунок 4.17 - Кнопка відеовиклику.

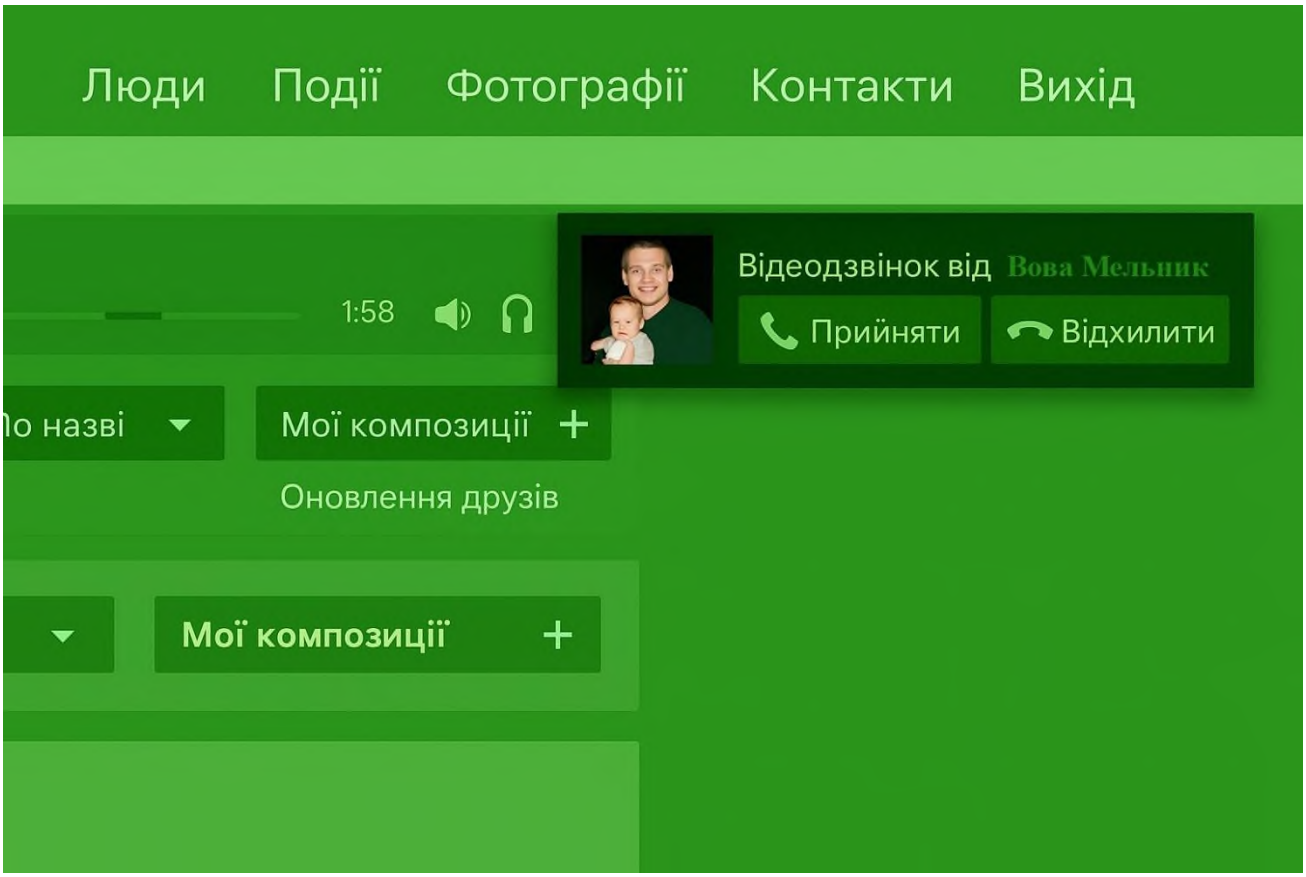

Рисунок 4.18 - Вхідний відеодзвінок.

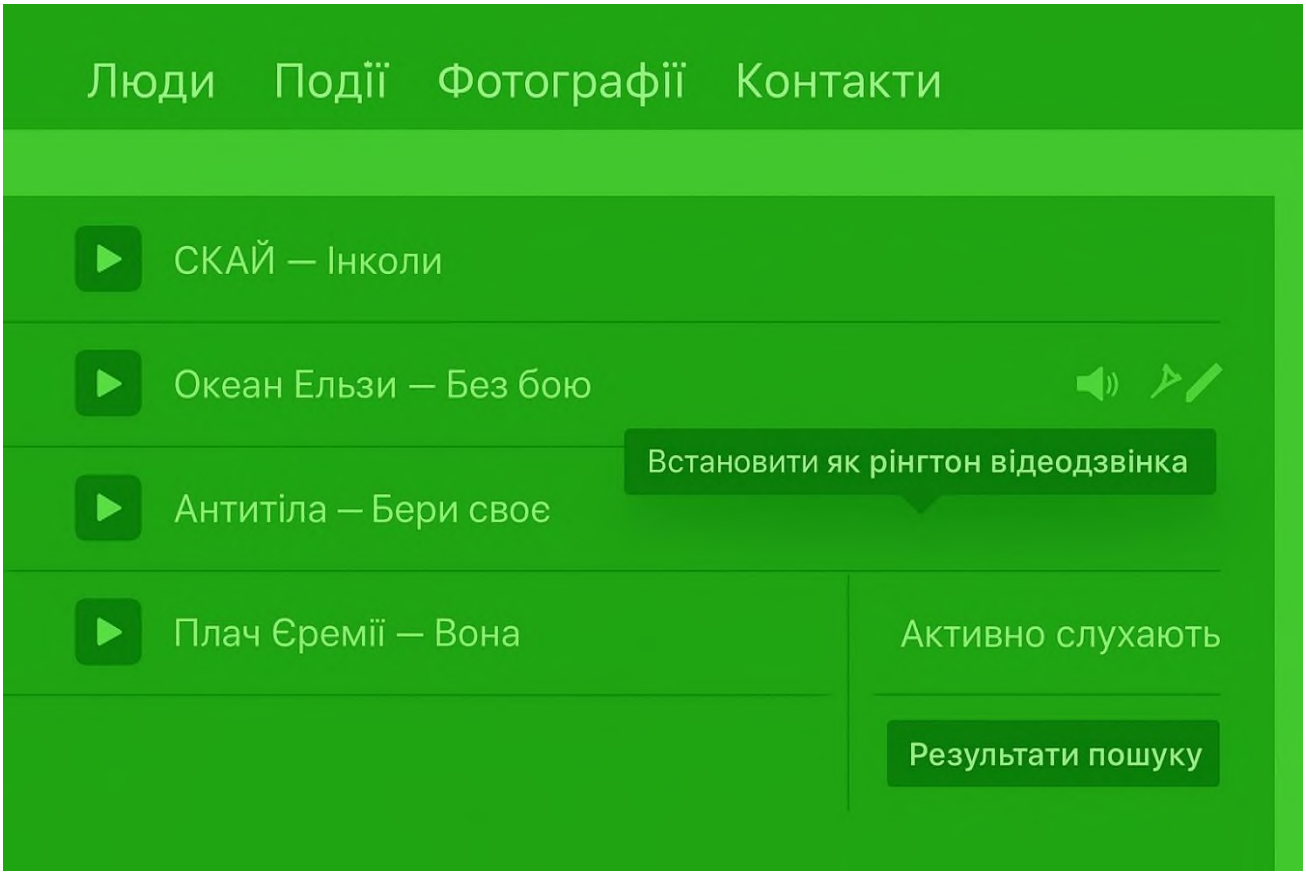

Рисунок 4.19 - Встановлення аудіо в якості рінґтону відеодзвінка

У межах практичної частини дипломної роботи було здійснено спробу з’ясувати, чи можлива на сучасному етапі реалізація повноцінного портативного WebRTC-рішення, яке можна інтегрувати в сторонній вебресурс через клієнтське браузерне розширення, навіть попри те, що стандарт WebRTC усе ще перебуває в активному розвитку. Проведена розробка підтвердила, що така інтеграція є цілком здійсненною. Оскільки запуск застосунку передбачалося виконувати через розширення для Google Chrome, потреби у забезпеченні підтримки кількох браузерів не виникало, що помітно спростило процес створення та тестування програмного продукту.

4.3.2 WebRTC-сервіс

Другою частиною практичної реалізації став вебсервіс, який може працювати у більшості сучасних браузерів, що мають підтримку WebRTC. Основою цього сервісу виступає бібліотека RTCPeerBroadcasting. Щоб забезпечити коректне функціонування у різних браузерних середовищах,

необхідно адаптувати низку функцій, які взаємодіють із Web API WebRTC, а також додати відповідні префіксні версії методів. Це стосується таких компонентів, як window. URL, метод getUserMedia, конструктори RTCPeerConnection і RTCSessionDescription.

```javascript
window.URL = window.webkitURL || window.mozURL || window.URL;

navigator.getUserMedia = navigator.webkitGetUserMedia ||
                         navigator.mozGetUserMedia ||
                         navigator.getUserMedia;
```

Рисунок 4.20 - Перебір префіксів

Для того щоб браузер автоматично обирав коректну реалізацію потрібного методу, використовується логічна конструкція, побудована на послідовних операціях «АБО». Такий підхід дає можливість послідовно перевірити наявність методів із різними вендорними префіксами та підставити той варіант, який доступний у конкретному середовищі виконання. Завдяки цьому система самостійно адаптується до відмінностей між браузерами, що суттєво підвищує універсальність роботи сервісу та усуває потребу вручну визначати тип браузера користувача.

Аналогічний підхід був застосований і до інших ключових компонентів WebRTC, зокрема до об’єктів RTCPeerConnection та RTCSessionDescription, які також мають різні префіксовані версії залежно від браузера [11].

Варто зазначити, що навіть операція прив’язування медіапотоку до HTML-елемента <video> у різних рушіях реалізована по-різному. Зокрема, у браузерах на основі Gecko (Mozilla Firefox) прив’язка відбувається через властивість mozSrcObject, тоді як у WebKit/Chromium-браузерах (Google Chrome, Opera) використовується стандартний атрибут src.

Щоб уніфікувати поведінку додатка та забезпечити правильну роботу на різних платформах, на початку бібліотеки було додано перевірку типу браузера.

Саме від її результатів залежить вибір механізму додавання медіапотоку, що дозволяє реалізувати єдиний універсальний метод відображення відео незалежно від технологічних відмінностей між браузерами.

```javascript
window.moz = !!navigator.mozGetUserMedia;
```

Рисунок 4.21 - Перевірка User-Agent

```javascript
var video = options.video;

if (moz) {
    video.mozSrcObject = stream;
} else {
    video.src = window.URL.createObjectURL(stream) || stream;
}

video.autoplay = "true";
video.play();
```

Рисунок 4.22 - Додавання потоку до відео-елементу

У процесі забезпечення кросбраузерної сумісності було виявлено низку розбіжностей у роботі методу onaddstream об’єкта RTCPeerConnection у браузерах, що використовують різні рушії. Зокрема, у браузерах на основі Webkit (наприклад, Google Chrome) об’єкт stream, переданий до цього обробника події, містить набір ідентифікаційних властивостей — зокрема id та label, які дозволяють однозначно пов’язати отриманий медіапотік із відповідним клієнтом.

У браузерах із рушієм Gecko (Mozilla Firefox) зазначені поля відсутні, що не відображено в офіційній документації WebRTC та спричинило труднощі під час тестування і налагодження бібліотеки. Для усунення цієї невідповідності було застосовано механізм перевизначення об’єкта потоку зі штучним додаванням унікального ключа, який виконував роль ідентифікатора. Такий підхід дозволив

забезпечити стабільну та коректну взаємодію клієнтів у різних браузерах, незалежно від їхньої внутрішньої реалізації обробки медіапотоків.

```javascript
function streaming(stream) {
    stream.userToken = window.selfUserToken || generateRandomToken();
}
```

Рисунок 4.23 - Додавання ключа до потоку

Упроваджені модифікації забезпечили коректну роботу програми в основних сучасних браузерах, зокрема Mozilla Firefox, Google Chrome та Opera. Наступним етапом практичної реалізації стало вдосконалення механізму створення відеоконференцій та розширення функціональних можливостей системи. Серед додаткових можливостей, які необхідно було реалізувати, — блокування небажаних користувачів, контроль увімкнення та вимкнення відеокамер і мікрофонів, а також збереження супровідних даних про учасників конференції.

Для досягнення цих цілей був розроблений спеціалізований клас Conference, який виконує функції проміжного шару абстракції над бібліотекою RTCPeerBroadcasting та істотно спрощує роботу з її методами. Конструктор класу приймає два параметри: адресу бази даних Firebase, що використовується як сигнальне середовище, та назву конференції, яка відіграє роль унікального ідентифікатора сеансу зв’язку [9].

Після створення екземпляра класу викликається метод startConnection, який відповідає за підключення до наявної конференції або ініціацію нової. У параметрах цього методу передаються ім’я користувача та набір callback-функцій, що забезпечують обробку подій під час встановлення та підтримки зв’язку. Додатково у методі виконується валідація імені користувача за допомогою функції checkLoginAlreadyInUse, яка перевіряє наявність такого ідентифікатора в активній конференції та запобігає конфліктам.

```javascript
if (This.checkLogin(userName)) {
    var ref = This.FB_room.child("user_list").child(userName);
    if (ref) {
        ref.once("value", function(data) {
            var ln = data.val();
            if (ln != undefined) {
                errorOrInUse(This.LOGIN_ALREADY_IN_USE);
            } else {
                loginFree();
            }
        });
    }
    else errorOrInUse(This.FIREBASE_REF_ERROR);
}
else errorOrInUse(This.NAME_INCORRECT);
```

Рисунок 4.24 - Метод checkLoginAlreadyInUse

Після підтвердження того, що обране ім’я користувача є валідним та не використовується іншими учасниками, система звертається до бази даних для визначення поточного складу конференції. На основі отриманої інформації визначається подальший сценарій роботи: якщо в записі зберігається інформація про наявних учасників, відбувається підключення до вже активної сесії; у разі відсутності зареєстрованих користувачів ініціюється створення нової конференції. Таким чином, рішення про підключення або створення нової сесії приймається на основі аналізу актуального стану відповідного елементу бази даних.

```javascript
var ref = This.FB.child("_user_num").child(roomName);
ref.once("value", function(data) {
    if (!data.val() || (typeof(data.val()) != "number")) {
        create();
    }
    else connect();
});
```

Рисунок 4.25 - Перевірка кількості користувачів

Механізм завершення участі в конференції реалізовано через метод disconnect, який ініціює процедуру розірвання з'єднання, використовуючи функцію leaveRoom із бібліотеки RTCPeerBroadcasting. Під час виконання цього методу відбувається також видалення відповідної користувацької директорії з бази даних, що забезпечує коректне звільнення ресурсів. Зазначений метод може застосовуватися як для добровільного виходу користувача з конференції, так і у випадках примусового відключення, коли ініціатором розриву з'єднання виступає інший учасник або адміністратор.

Для реалізації механізму примусового видалення окремих учасників передбачено метод kickUser, який отримує ім'я користувача як вхідний параметр. При його виконанні до відповідного запису в базі даних вноситься службове значення "kicked". На стороні клієнта під час підключення до конференції встановлюється обробник події, який відстежує появу цього маркера в користувацькій директорії. За його виявлення автоматично викликається метод disconnect, що забезпечує негайне припинення участі користувача в конференції [13].

Для керування мультимедійними потоками реалізовано методи switchAudio та switchVideo, які приймають логічний параметр (true/false) і відповідають за активацію або вимкнення відповідних компонентів медіапотоку. Маніпуляції з аудіо- та відеоданими здійснюються через механізми Stream API: кожному медіатреку призначається значення властивості enabled, що визначає його участь у передаванні даних.

```javascript
if (This.myStream) {
    var tracks = This.myStream.getVideoTracks();

    for (var i = 0; i < tracks.length; i++) {
        tracks[i].enabled = state;
    }

    This.videoState = state;
    callback();
}
else callback(This.NOT_CONNECTED_TO_CONFERENCE);
```

Рисунок 4.26 - Реалізація методу switchVideo

Механізм отримання медіапотоків від інших учасників конференції реалізовано за допомогою подієвої моделі. Після створення екземпляра класу Conference розробник має визначити обробник події onUserConnected, який активується щоразу, коли до сеансу приєднується новий користувач. У момент спрацювання події викликається відповідна функція-обробник, якій передаються ключові параметри: ім’я підключеного учасника, унікальний ідентифікатор його медіапотоку та HTML-елемент відео, призначений для відтворення цього потоку. Приклад відповідної реалізації наведено у фрагменті коду (рис. 30).

```javascript
onRemoteStream: function(media) {
    if (media.video) {
        var video = media.video;
        var strid = media.stream.userToken || generateRandomToken();
        video.id = strid;

        var userName = This.userList[strid] || "Anonymous";
        This.onUserConnected(userName, strid, video);
    }
    else console.warn("onRemoteStream", "video is undefined");
},
```

Рисунок 4.27 - Подія onRemoteStream

Клас Conference передбачає наявність додаткових подій, серед яких onUserDisconnected, onKicked та onUserCount, що розширюють функціональність базової моделі відеоконференції. Окрім основних механізмів організації багатокористувацького сеансу зв'язку, цей клас виконує роль абстрактного шару над бібліотекою RTCPeerBroadcasting, істотно спрощуючи процес взаємодії з нею.

Реалізований абстракційний механізм автоматизує роботу із сигнальним каналом, забезпечує коректне зчитування та запис службових параметрів, а також здійснює перевірку валідності переданих даних. Завдяки цьому встановлення peer-to-peer з’єднання між учасниками — навіть якщо вони працюють на різних

вебсторінках чи у сторонніх застосунках — зводиться до мінімальної кількості дій з боку розробника.

Для ініціалізації відеоконференції достатньо підключити три необхідні бібліотеки (Firebase, RTCPeerBroadcasting, Conference) та викликати три ключові методи класу Conference, як це продемонстровано на рис. 30.

```javascript
var room = new Conference("https://js-test.firebaseio.com/", "my-room");

room.onUserConnected = function(name, id, video) {
    video.width = 200;
    document.body.appendChild(video);
};

room.startConnection("user-name");
```

Рисунок 4.28 - Виклик трьох основних методів підключення

Іншими словами, функціональність WebRTC може бути значно спрощена шляхом створення власних високорівневих інтерфейсів, які інкапсулюють виклики до методів Web API. Після завершення розроблення класу Conference було сформовано окремий інтерфейс WebRTC-сервісу, логіка якого безпосередньо пов’язана з методами цього класу.

До базової сторінки проведення конференцій було додано додаткові елементи: стартову сторінку сервісу, модуль текстового чату, механізм додавання тегів для визначення рівня доступності (публічна або приватна конференція), а також окремий розділ зі списком усіх доступних публічних кімнат.

У результаті розроблений сервіс був сформований як повноцінна багатофункціональна система з кількома взаємопов’язаними інтерфейсними сторінками. Його завершений вигляд представлено на рисунку 4.29, де показано приклад роботи застосунку під час активної відеоконференції за участю чотирьох користувачів.

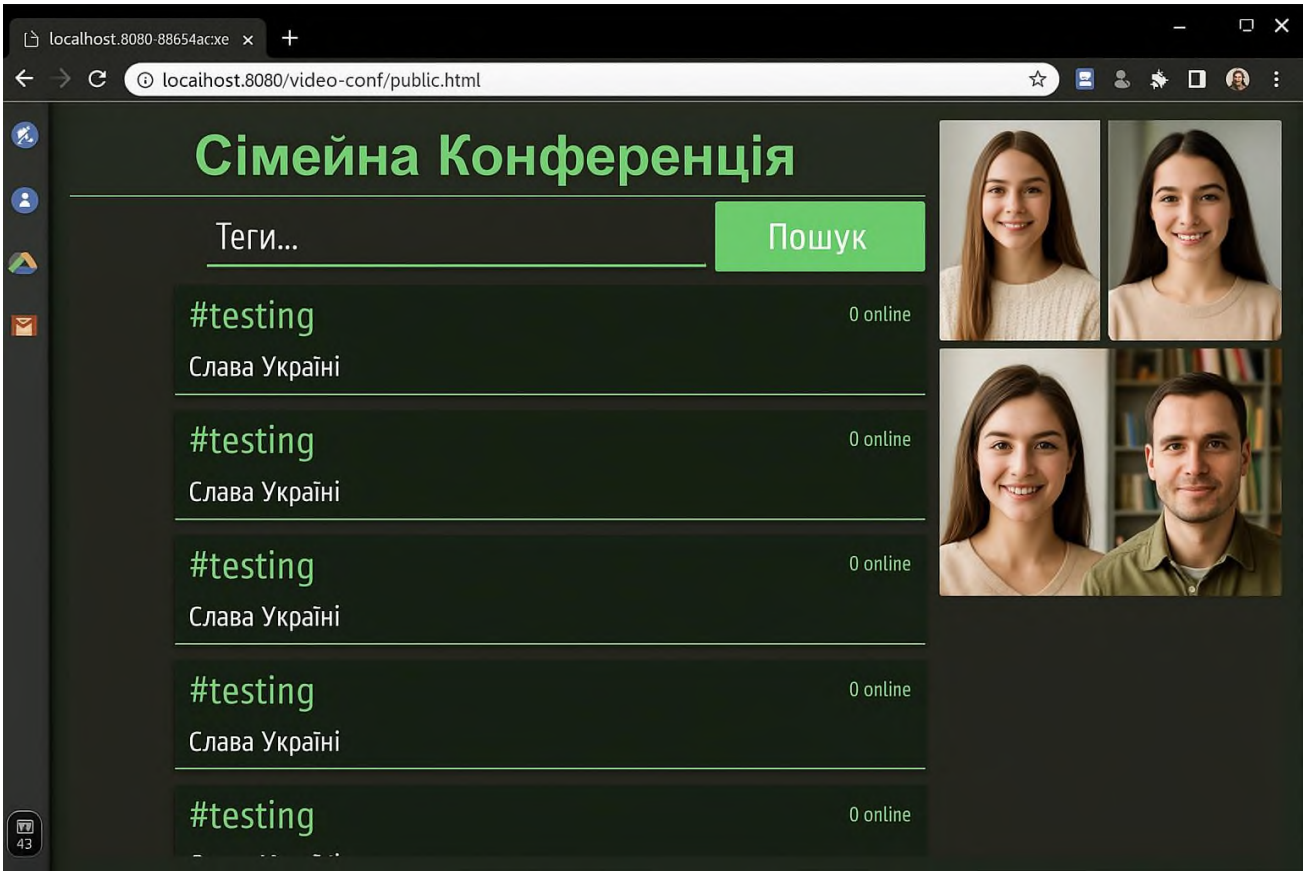

Рисунок 4.29 - Список доступних публічних конференцій

Висновки до розділу

Розв’язання задач, пов’язаних із передаванням даних, набуває особливої актуальності в умовах зростання кількості хакерських атак на ключові комунікаційні сервіси. Використання подібних технологічних рішень дає можливість забезпечити стабільний та безпечний зв’язок із близькими, а також здійснювати обмін повідомленнями без суттєвих ризиків і перешкод.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Сучасний цифровий простір демонструє стрімке зростання попиту на інструменти комунікації реального часу. Глобальна діджиталізація та розширення можливостей віддаленої роботи спричинили значне збільшення потреби у платформах відеоконференцій, онлайн-лекцій, корпоративного обміну даними та телемедицини. На цьому фоні технологія WebRTC стає ключовим рушієм нових рішень у сфері потокової передачі даних. Її відкритість, безкоштовність і сумісність з усіма сучасними браузерами створюють можливість розроблення застосунків нового покоління. Цей стартап-проєкт описує платформу, яка поєднує відеозв'язок, аудіокомунікацію, обмін даними та конференції у межах одного WebRTC-рішення. Сучасний цифровий простір демонструє стрімке зростання попиту на інструменти комунікації реального часу.

Основна концепція проєкту полягає в створенні масштабованої WebRTC-платформи, яка забезпечує потокову передачу даних у режимі реального часу без встановлення додаткового програмного забезпечення. Проєкт орієнтований на організації, що потребують внутрішньої комунікації, та користувачів, яким потрібен гнучкий, безпечний інструмент із можливістю кастомізації.

Зміст ідеї включає розроблення таких модулів: відеоконференції, приватні дзвінки, групові кімнати, чат, обмін файлами, а також розширені можливості для корпоративних клієнтів — інтеграцію з CRM, можливість запису дзвінків, автоматизацію підтримки тощо.
Напрямки застосування:

- корпоративні комунікації;
- онлайн-навчання, дистанційні курси;
- надання консультацій (юридичних, медичних, технічних);
- організація онлайн-заходів;
- сервіс підтримки клієнтів.

Відмінність від аналогів полягає у відкритому коді, можливості локального розгортання всередині підприємства та персоналізації функціоналу. Основна концепція проєкту полягає в створенні масштабованої WebRTC-платформи, яка забезпечує потокову передачу даних у режимі реального часу без встановлення додаткового програмного забезпечення. Проєкт орієнтований на організації, що потребують внутрішньої комунікації, та користувачів, яким потрібен гнучкий, безпечний інструмент із можливістю кастомізації [19]. Основні вигоди користувачів: низькі витрати, простота використання, можливість працювати в будь-якому браузері, захищеність даних через DTLS/SRTP.

Аналіз технологічних можливостей

Технологія WebRTC надає широкий набір можливостей для реалізації комунікаційних сервісів. У рамках проєкту застосовуються такі основні компоненти:

- RTCPeerConnection встановлення P2P-з'єднання;
- MediaStream API доступ до відео- та аудіопристроїв;
- RTCDataChannel - передавання довільних даних;
- STUN/TURN сервери - забезпечення стабільних з’єднань через NAT;
- WebSocket/Firebase - сигнальний канал.

Технологічна здійсненність ідеї підтверджується тим, що всі необхідні інструменти є відкритими, доступними та підтримуються сучасними браузерами. Не потрібно розробляти власний медіатранспорт або алгоритми шифрування - WebRTC інтегрує DTLS, ICE, SRTP та інші стандарти.

За результатами аналізу технологій визначено, що реалізація проєкту можлива з використанням відкритих бібліотек та серверних рішень, а масштабування може бути забезпечене за рахунок контейнеризації (Docker), мікросервісної архітектури та хмарних сервісів (AWS, Google Cloud). Технологія WebRTC надає широкий набір можливостей для реалізації комунікаційних сервісів.

Технологічна здійсненність ідеї підтверджується тим, що всі необхідні

інструменти є відкритими, доступними та підтримуються сучасними браузерами. Не потрібно розробляти власний медіатранспорт або алгоритми шифрування — WebRTC інтегрує DTLS, ICE, SRTP та інші стандарти.

Аналіз ринкових можливостей

Ринок відеокомунікацій оцінюється більш ніж у 30 млрд доларів і щороку зростає на 12–18%. Зростання обумовлене переходом підприємств на віддалений формат роботи та розвитком дистанційних сервісів. Основними конкурентами є Zoom, Microsoft Teams, Google Meet та Webex. Проте вони мають низку недоліків: високу ціну підписок, закриту архітектуру, складності інтеграції у внутрішні системи компаній.

Наш стартап пропонує такі переваги:

1. низька вартість розгортання;
2. можливість встановлення на внутрішні сервери підприємства;
3. відкритість коду та персоналізація;
4. простота інтеграції з будь-якими вебсистемами.

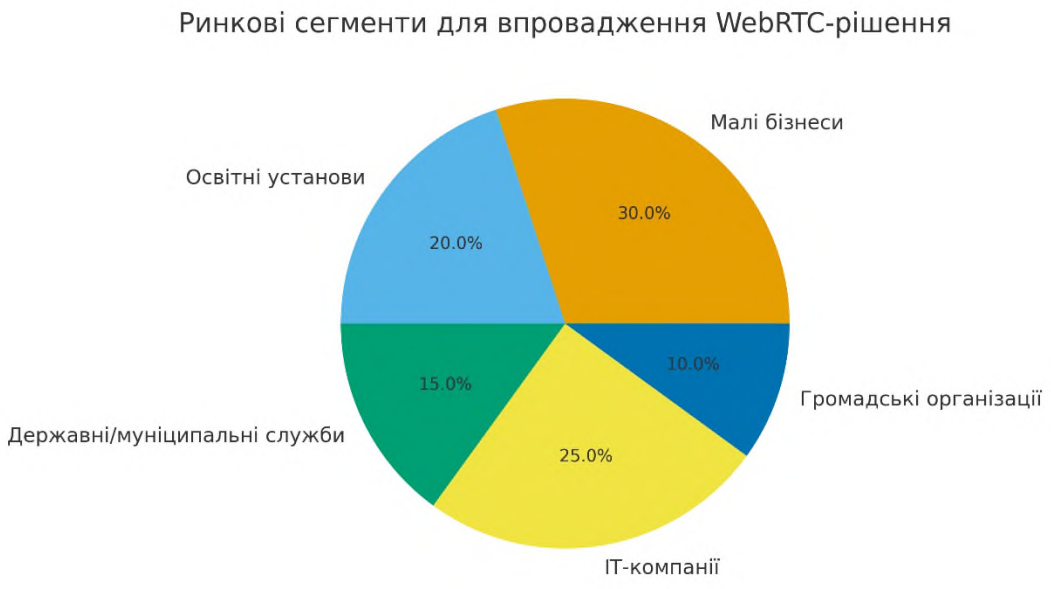

Рисунок 5.1 – Ринкові сегменти для впровадження WebRTC-рішення

Діаграма відображає потенційний розподіл ринку для застосування WebRTC-рішення:

- **30% Малі бізнеси.** Найвищий попит завдяки потребі у доступних засобах комунікації без складних інфраструктур.
- **20% Освітні установи.** Використання для онлайн-лекцій, консультацій, дистанційних занять.
- **15% Державні/муніципальні служби.** Потреба у безпечних каналах відеозв'язку для внутрішньої комунікації.
- **25% IT-компанії.** Активно інтегрують інструменти реального часу в свої продукти.
- **10% Громадські організації.** Проведення онлайн-зустрічей, координаційних зборів, вебінарів.

Потенційними сегментами споживачів є: бізнеси (SMB), школи, університети, державні органи, медичні установи, громадські організації. За результатами опитування користувачів, понад 70% шукали б дешевшу альтернативу Zoom та Teams у разі збереження якості зв'язку.Ринок відеокомунікацій оцінюється більш ніж у 30 млрд доларів і щороку зростає на 12–18%. Зростання обумовлене переходом підприємств на віддалений формат роботи та розвитком дистанційних сервісів. Основними конкурентами є Zoom, Microsoft Teams, Google Meet та Webex. Проте вони мають низку недоліків: високу ціну підписок, закриту архітектуру, складності інтеграції у внутрішні системи компаній.

Стратегія ринкового впровадження

Впровадження проєкту передбачає декілька етапів:

1. створення MVP з базовим функціоналом і тестування в навчальних закладах та малих офісах;
2. запуск бета-версії з розширеними функціями — DataChannel, запис конференцій, шифроване сховище файлів;
3. вихід на B2B-сегмент шляхом партнерств з IT-компаніями;

маркетингова стратегія: SEO, SMM, спільноти, конференції; масштабування через хмарні сервіси та впровадження мобільного застосунку.

Паралельно розробляється бізнес-модель: freemium, корпоративні пакети, white-label рішення, індивідуальні кастомізації.

Висновки до розділу

Стартап-проєкт WebRTC-платформи демонструє високий потенціал розвитку та масштабування завдяки технологічній доступності, низькій вартості та затребуваності ринку. Рішення може застосовуватися в різних сферах та забезпечувати безпечний і ефективний зв'язок у реальному часі. Розроблена платформа має всі передумови для успішної комерціалізації та подальшої експансії на ринок цифрових комунікацій.

Розроблений концепт програмного рішення відповідає сучасним запитам цифрового середовища, де ключову роль відіграють безпечні, швидкі та доступні засоби комунікації в режимі реального часу. Використання відкритих веб-стандартів, відсутність потреби у спеціалізованому клієнтському ПЗ та можливість інтеграції з існуючими веб-ресурсами створюють конкурентні переваги для виходу на ринок.

Проведений аналіз технічної здійсненності показав, що WebRTC забезпечує необхідну інфраструктуру для реалізації аудіо- та відео-комунікацій, включаючи наскрізне шифрування, P2P-взаємодію та можливість створення багатокористувацьких конференцій. Це дозволяє успішно впроваджувати продукт у корпоративне середовище, у громади, освітні та медичні установи, а також у невеликі підприємства, які потребують недорогого та надійного інструмента зв'язку.

Маркетинговий аналіз довів наявність зростаючого попиту на платформи відеокомунікації, особливо з урахуванням віддаленої роботи, дистанційного навчання та цифровізації бізнес-процесів. Запропонований продукт має перспективу зайняти свою частку ринку завдяки поєднанню низької собівартості, високої продуктивності та можливості адаптації під конкретні потреби організацій.

Стратегія ринкового впровадження передбачає орієнтацію на сегмент малих і середніх підприємств, громади та організації, які потребують безпечних внутрішніх засобів комунікації. Подальше розширення функціоналу, інтеграція з CRM/ERP-системами та розроблення мобільної версії можуть забезпечити зростання конкурентоспроможності та привабливості продукту.

Таким чином, проведене дослідження підтвердило, що стартап-проєкт є технологічно здійсненним, економічно доцільним і ринково перспективним. Результати проєкту можуть бути успішно використані для створення комерційного продукту, здатного задовольнити сучасні потреби у високоякісних засобах комунікації в реальному часі.

ВИСНОВКИ

У процесі виконання дипломного дослідження було здійснено комплексний аналіз архітектури та практичних аспектів використання технології WebRTC, що є однією з ключових інновацій у сфері веб-комунікацій. Особлива увага приділялася архітектурним засадам, механізмам передавання аудіо- та відеопотоків у режимі реального часу, а також можливостям застосування WebRTC для побудови сучасних інтерактивних систем зв'язку.

Однією з визначальних передумов розвитку WebRTC є концепція відкритості та вільного доступу до технологій, що історично сприяло стрімкому поширенню Інтернету та супутніх сервісів. На відміну від попередніх рішень, які вимагали використання пропрієтарних протоколів або ліцензійних модулів, WebRTC запроваджує універсальний механізм мультимедійного обміну, інтегрований безпосередньо у браузерне середовище та доступний для розробників без додаткових фінансових та технічних бар'єрів. Завдяки цьому технологія сформувала новий підхід до організації реального часу в мережі, поєднуючи безплатність, стандартизованість і високу якість передавання даних.

У межах дипломного проєкту було детально розглянуто Web API WebRTC та створено дві самостійні програмні реалізації, які продемонстрували практичну придатність технології. Результати розробки підтверджують, що WebRTC характеризується достатньою стабільністю та технологічною зрілістю для впровадження у реальні системи, попри те, що процес її стандартизації все ще триває. На сучасному етапі не існує відкритих аналогів, здатних забезпечити такий самий рівень інтерактивності та якості мультимедійного обміну в браузері без сторонніх плагінів.

Важливо підкреслити, що завдяки відкритому вихідному коду та широкій підтримці з боку світових технологічних компаній WebRTC демонструє високу гнучкість. Технологія може бути інтегрована у широкий спектр проєктів - від корпоративних комунікаційних платформ до освітніх сервісів, телемедицини, систем дистанційного навчання та відеоконференцій. Відкритість та модульність

протоколів WebRTC дозволяють адаптувати функціональність під потреби конкретних користувачів, а також розширювати її шляхом розроблення додаткових інструментів, модулів чи бібліотек.

Загалом отримані результати свідчать про те, що WebRTC є перспективною, технологічно зрілою та стратегічно важливою платформою для подальшого розвитку засобів комунікації у веб-середовищі. Виконана робота підтверджує, що технологія вже сьогодні може застосовуватися у повнофункціональних програмних продуктах, забезпечуючи високу якість зв’язку, гнучкість у налаштуванні та доступність для широкого кола розробників і кінцевих користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jennings C., Rescorla E., Romanov A. *WebRTC 1.0: Real-Time Communication Between Browsers*. W3C Working Draft. 2022. [Електронний ресурс]: https://www.w3.org/TR/webrtc/ (дата звернення: 20.11.2025р.).

2. Rescorla E. *WebRTC Security Architecture*. RFC 8827, IETF. 2021. [Електронний ресурс]: https://www.rfc-editor.org/rfc/rfc8827.html (дата звернення: 20.11.2025р.).

3. Obsolete: getUserMedia API — *Media Capture and Streams*. W3C Recommendation. [Електронний ресурс]: https://www.w3.org/TR/mediacapture-streams/ (дата звернення: 20.11.2025р.).

4. Google Developers. WebRTC Architecture Documentation. [Електронний ресурс]: https://webrtc.org/architecture/ (дата звернення: 20.11.2025р.).

5. Google Developers. WebRTC Code Samples and Tutorials. [Електронний ресурс]: https://webrtc.github.io/samples/ (дата звернення: 20.11.2025р.).

6. Mozilla Developer Network (MDN). WebRTC API Reference. [Електронний ресурс]: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API (дата звернення: 20.11.2025р.).

7. Davis D. *Cross-browser camera capture with getUserMedia*. Mozilla Hacks, 2013. [Електронний ресурс]:https://hacks.mozilla.org/2013/02/cross-browser-camera-capture-with-getusermediawebrtc/ (дата звернення: 20.11.2025р.).

8. Cohen K. *Basics of WebRTC Peer Connections*. Medium, 2018. [Електронний ресурс]:https://medium.com/@kanecohen/basics-of-webrtc-peer-connections-c1ed743de2f6 (дата звернення: 20.11.2025р.).

9. Singh V., Ott J., de Costa R. *Performance analysis of WebRTC-based video conferencing*. IEEE, 2013.

10. Gonzalez R., Woods R. *Digital Image Processing*. 4th ed. Pearson, 2018, 1022p.

11. Rao K. R., Yip P. *The Transform and Data Compression Handbook*. CRC Press, 2000, 41p.

12. Kurose J., Ross K. *Computer Networking: A Top-Down Approach*. 8th ed. Pearson, 2021, 775p.

13. Schulzrinne H., Casner S., Frederick R., Jacobson V. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550, 2003. [Електронний ресурс]: https://www.rfc-editor.org/rfc/rfc3550.html (дата звернення: 20.11.2025р.).

14. Baugher M., McGrew D., Naslund M., Carrara E., Norrman K. *The Secure Real-Time Transport Protocol (SRTP)*. RFC 3711, 2004. [Електронний ресурс]: https://www.rfc-editor.org/rfc/rfc3711.html (дата звернення: 20.11.2025р.).

17. Jansen A., McGovern A. *Real-Time Communications with WebRTC*. O'Reilly Media, 2021, 164p.

18. Loreto S., Romano S. P. *Real-Time Communication with WebRTC: Peer-to-Peer in the Browser*. O'Reilly Media, 2014, 164p.

19. Cáceres R., Narendula R. *Evaluating WebRTC Peer-to-Peer Communication Performance*. ACM, 2018, 214p.

ДОДАТКИ

ДОДАТОК А

```
import java.util.Comparator;
import lombok.Data;
import java.util.Arrays;
public class BasicGA {
   public static void main(String[] args) {
      // Create GA object
      GeneticAlgorithmHelper ga = new GeneticAlgorithmHelper(100, 0.01,
            0.95, 3); // We'll add a lot more here...
      // Initialize population
      Population population = ga.initPopulation(40);
      // The following is the new code you should be adding:
      ga.evaluatePopulation(population);
      int generation = 1;
      while (!ga.isShouldStop(population)) {
         // Print fittest individual from population
         System.out.println("Найкраще рішення: " + population.
               getBest(0).toString());
         // Apply crossover
         population = ga.crossoverPopulation(population);
         // Apply mutation
         population = ga.mutatePopulation(population);
         // Evaluate population
         ga.evaluatePopulation(population);
         // Increment the current generation
         generation++;
      }
      System.out.println("Рішення знайдене за " + generation + " поколінь");
      System.out.println("Найкраще рішення: " + population.getBest(0).toString());
   }}
/**
 * Lots of comments in the source that are omitted here!
```

```
 */
public class GeneticAlgorithmHelper {
    private final int populationSize;
    private final double mutationRate;
    private final double crossoverRate;
    private final int elitismCount;
    public GeneticAlgorithmHelper(int populationSize, double mutationRate, double
            crossoverRate, int elitismCount) {
        this.populationSize = populationSize;
        this.mutationRate = mutationRate;
        this.crossoverRate = crossoverRate;
        this.elitismCount = elitismCount;
    }
    /**
     * Many more methods implemented later...
     */
    public Population initPopulation(int chromosomeLength) {
        return new Population(this.populationSize, chromosomeLength);
    }
    public double calculateFitnessFunction(Person person) {
        // Track number of correct genes
        int correctGenes = 0;
        // Loop over individual's genes
        for (int geneIndex = 0; geneIndex < person.getChromosomeLength(); geneIndex++) {
            // Add one fitness point for each "1" found
            if (person.getGene(geneIndex) == 1) {
                correctGenes += 1;
            }
        }
        // Calculate fitness
        double fitness = (double) correctGenes / person.getChromosomeLength();
        // Store fitness
        person.setFitnessFunction(fitness);
       return fitness;
```

```
    }
    public void evaluatePopulation(Population population) {
        double populationFitness = 0;
        for (Person person : population.getPopulation()) {
            populationFitness += calculateFitnessFunction(person);
        }
        population.setPopulationFitness(populationFitness);
    }
    public boolean isShouldStop(Population population) {
        for (Person person : population.getPopulation()) {
            if (person.getFitnessFunction() == 1) {
                return true;
            }       }
        return false;    }
@Data
    public Person chooseParent(Population population) {
        // Get individuals
        Person[] people = population.getPopulation();
        // Spin roulette wheel
        double populationFitness = population.getPopulationFitness();
        double rouletteWheelPosition = Math.random() * populationFitness;
        // Find parent
        double spinWheel = 0;
        for (Person person : people) {
            spinWheel += person.getFitnessFunction();
            if (spinWheel >= rouletteWheelPosition) {
                return person;
            }       }
        return people[population.size() - 1];
    }
    public Population crossoverPopulation(Population population) {
        // Create new population
        Population newPopulation = new Population(population.size());
        // Loop over current population by fitness
```

```
    for (int i = 0; i < population.size();
        i++) {
       Person parent1 = population.getBest(i);
       // Apply crossover to this individual?
       if (this.crossoverRate > Math.random() && i >
             this.elitismCount) {
          // Initialize offspring
          Person offspring = new Person(parent1.
                getChromosomeLength());
          // Find second parent
          Person parent2 = chooseParent(population);
          // Loop over genome
          for (int geneIndex = 0; geneIndex < parent1.
                getChromosomeLength(); geneIndex++) {
             // Use half of parent1's genes and half of parent2 's genes
             if (0.5 > Math.random()) {
                offspring.setGene(geneIndex,
                      parent1.getGene(geneIndex));
             } else {
                offspring.setGene(geneIndex,
                      parent2.getGene(geneIndex));
             }               }
          // Add offspring to new population
          newPopulation.setPerson(i,
                offspring);
       } else {
          // Add individual to new population without applying crossover
          newPopulation.setPerson
                (i, parent1);
       }       }
    return newPopulation;    }
public Population mutatePopulation(Population population) {
    // Initialize new population
    Population newPopulation = new Population(this.populationSize);
```

```
        // Loop over current population by fitness
        for (int populationIndex = 0; populationIndex < population.size();
             populationIndex++) {
            Person person = population.
                    getBest(populationIndex);
            // Loop over individual's genes
            for (int geneIndex = 0; geneIndex < person.
                    getChromosomeLength(); geneIndex++) {
                // Skip mutation if this is an elite individual
                if (populationIndex >= this.elitismCount) {
                    // Does this gene need mutation?
                    if (this.mutationRate > Math.random()) {
                        // Get new gene
                        int newGene = 1;
                        if (person.getGene(geneIndex) == 1) {
                            newGene = 0;
                        }
                        // Mutate gene
                        person.setGene(geneIndex, newGene);
                    }               }           }
            // Add individual to population
            newPopulation.setPerson(populationIndex, person);
        }
        // Return mutated population
        return newPopulation;
    }}
public class Person {
    private final int[] chromosome;
    private double fitnessFunction = -1;
    public Person(int[] chromosome) {
        // Create individual chromosome
        this.chromosome = chromosome;    }
    public Person(int chromosomeLength) {
        this.chromosome = new int[chromosomeLength];
```

```
        for (int gene = 0; gene < chromosomeLength; gene++) {
            if (0.5 < Math.random()) {
                this.setGene(gene, 1);
            } else {
                this.setGene(gene, 0);          }       }    }
    public int getChromosomeLength() {
        return this.chromosome.length;    }
    public void setGene(int offset, int gene) {
        this.chromosome[offset] = gene;    }
    public int getGene(int offset) {
        return this.chromosome[offset];    }
    public String toString() {
        StringBuilder output = new StringBuilder();
        for (int i : this.chromosome) {
            output.append(i);       }
        return output.toString();    }}
public class PersonComparator implements Comparator<Person> {
    @Override
    public int compare(Person person1, Person person2) {
        if (person1.getFitnessFunction() > person2.getFitnessFunction()) {
            return -1;
        } else if (person1.getFitnessFunction() < person2.getFitnessFunction()) {
            return 1;       }
        return 0;    }}
@Data
public class Population {
    private final Person[] population;
    private double populationFitness = -1;
    public Population(int populationSize) {
        this.population = new Person[populationSize];    }
    public Population(int populationSize, int chromosomeLength) {
        this.population = new Person[populationSize];
        for (int individualCount = 0; individualCount <
                populationSize; individualCount++) {
```

```
        Person person = new
                Person(chromosomeLength);
        this.population[individualCount] = person;
    }    }
public Person getBest(int offset) {
    Arrays.sort(this.population, new PersonComparator());
    return this.population[offset];    }
public int size() {
    return this.population.length;    }
public void setPerson(int offset, Person person) {
    population[offset] = person;    }            }
```