Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *Інфокомунікаціїї*
(повна назва)

Кафедра *Інформаційно-мережної інженерії*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти *другий (магістерський)*

Дослідження особливостей та можливостей WebRTC при реалізації функцій зв'язку в мобільних додатках

(тема)

Виконав:

студент 2 курсу, групи *ІМІм-22-1*
Дугінов О.Ю.

Спеціальності *172 Телекомунікації та радіотехніка*
(код і повна назва спеціальності)

Тип програми *Освітньо-професійна*
(освітньо-професійна або освітньо-наукова)

Освітня програма *Інформаційно-мережна інженерія*
(повна назва освітньої програми)

Керівник *доц. Бондар Д. В.*
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри *Безрук В.М.*

(підпис) (прізвище, ініціали)

2024 р.

Не містить відомостей, заборонених до відкритого публікування

Студент _____ *Дугінов О.Ю.*
(підпис) (прізвище та ініціали)

Керівник _____ *Бондар Д. В.*
(підпис) (прізвище та ініціали)

Харківський національний університет радіоелектроніки

Факультет *Інфокомунікацій*
(повна назва)

Кафедра *Інформаційно-мережної інженерії*
(повна назва)

Рівень вищої освіти *другий(магістерський)*

Спеціальність *172 Телекомунікації та радіотехніка*
(код і повна назва)

Тип програми *Освітньо-професійна*
(освітньо-професійна або освітньо-наукова)

Освітня програма *Інформаційно-мережна інженерія*
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри ІМІ_____
(підпис)

—___‖ _____2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Студентові *Дугінову Олександру Юрійовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження особливостей та можливостей WebRTC при реалізації функцій зв'язку в мобільних додатках.*
затверджена наказом університету від 23.10 2024 р. № 1233 Ст.
2. Термін подання студентом роботи до екзаменаційної комісії 9.01 2024 р.
3. Вихідні дані до роботи: *Дослідити та проаналізувати функції зв'язку технології WebRTC: Фунції обміну ICE кандидатами, обробки SDP, втановлення з'єднання між клієнтами.*
4. Перелік питань, що потрібно опрацювати в роботі
Вступ;
1 Огляд технології WebRTC;
2 Особливосі WebRTC;
3 Реалізація функцій зв'язку в мобільних додатках;
4 Аналіз результатів дослідження;
Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри): Слайди у форматі Power Point (назва, зміст, актуальність проблеми, вступ, технологія WebRTC, схема роботи WebRTC, функція ініціалізації дзвінку, функція обробки SDP, функція встановлення з'єднання, взаємодія WebRTC та Dart, порівняння технологій з аналогами, висновки).

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Підбір літератури по тематиці	23.10.2023	Вик.
2.	Аналіз літератури	30.10.2023	Вик.
3.	Огляд технології WebRTC	07.11.2023	Вик.
4.	Особливості WebRTC	14.11.2023	Вик.
5.	Реалізація функцій зв'язку в мобільних додатках	06.12.2023	Вик.
6.	Аналіз отриманих результатів	19.12.2023	Вик.
7.	Висновки	30.12.2023	Вик.
8.	Оформлення пояснювальної записки	05.01.2024	Вик.
9.	Підготовка доповіді	07.01.2024	Вик.

Дата видачі завдання *20.10.2023 р.*

Студент _____ *Дугінов О. Ю.* _____
(підпис) (прізвище та ініціали)

Керівник роботи *Бондар Д. В.*
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка: 72 с. 15 рис., 2 табл., 2 додатки, 7 джерел.

Метою кваліфікаційної роботи є огляд технології WebRTC та аналіз функцій, які дозволяють встановити пряме відео та аудіо з’єднання між двома, або більше користувачами в реальному часі у рамках мобільного додатку використовуючи мову програмування Dart.

Об’єктом дослідження є технологія WebRTC, яка представляє з себе набір методів та функцій, зібраних спеціально для розробки мовою програмування Dart, необхідних для виконання мети.

Результатом дослідження є пакет функцій мобільного додатку для смартфону та серверні функції для обміну даними між клієнтами. Методи взаємодіять з сервером через технологію Socket.io.

WEBRTC, DART, FLUTTER, SOCKET.IO, МОБІЛЬНА РОЗРОБКА, ВІДЕОДЗВІНКИ, ЗВ′ЯЗОК В РЕАЛЬНОМУ ЧАСІ, ОБМІН МЕДІАДАНИМИ

ABSTRACT

Explanatory note: 69 p. 15 figures, 2 tables, 2 appendices, 7 sources.

The purpose of the qualification work is to review the WebRTC technology and analyze the functions that allow establishing a direct video and audio connection between two or more users in real time within a mobile application using the Dart programming language.

The object of research is the WebRTC technology, which is a set of methods and functions collected specifically for development in the Dart programming language, necessary to fulfill the purpose.

The result of the research will be a package of functions of a mobile application for a smartphone and server functions for data exchange between clients. The methods interact with the server through Socket.io technology.

WEBRTC, DART, FLUTTER, SOCKET.IO, MOBILE DEVELOPMENT, VIDEO CALLING, REAL-TIME COMMUNICATION, MEDIA SHARING

ЗМІСТ

ПРЕЛІК СКОРОЧЕНЬ

MDN (Mozilla Developer Network) – Мережа Розробників Mozilla;

W3C (World Wide Web Consortium) – Консорціум всесвітньої павутини;

IETF (Internet Engineering Task Force) – Задачевий фонд Інженерії Інтернету;

NAT (Network Address Translation) – Мережевий Адресний Переклад;

TURN (Traversal Using Relays around NAT) – Трансляція за допомогою ретрансляцій навколо NAT;

STUN (Session Traversal Utilities for NAT) – Сполучення Утиліт для Проходження через NAT;

SDP (Session Description Protocol) – Протокол Опису Сесії;

ICE (Interactive Connectivity Establishment) – Інтерактивне Забезпечення З'єднання;

ВСТУП

Веб-зв’язок у реальному часі (WebRTC) є новим стандартом та індустріальною ініціативою, яка розширює модель веб-перегляду. Вперше браузери можуть безпосередньо обмінюватися медіафайлами в режимі реального часу з іншими браузерами в одноранговому режимі. Спільно World Wide Web Consortium (W3C) та Internet Engineering Task Force (IETF) визначають JavaScript API, стандартні теги HTML5 та базові протоколи зв’язку для створення та керування надійним каналом зв’язку між будь-якою парою веб-браузерів наступного покоління. Основною метою стандартизації є визначення WebRTC API, що дозволяє веб-програмам, запущеним на будь-якому пристрої, безпечно взаємодіяти з вхідними периферійними пристроями (наприклад, веб-камерами та мікрофонами), обмінюватися медіафайлами та даними в режимі реального часу з віддаленою стороною в одноранговій мережі.

Актуальність теми визначається стрімким розвитком інтернет-технологій та зростанням популярності веб-застосунків. З поглибленням цифровізації сучасного суспільства, WebRTC стає важливим інструментом для забезпечення ефективного віртуального спілкування. Високий рівень інтерактивності, доступність та безпека, що надається WebRTC, роблять його актуальним для впровадження у різноманітних сферах, від соціальних мереж до бізнес-застосунків.

Мета дослідження полягає у вивченні особливостей та можливостей використання технології у мобільних додатках для забезпечення функцій зв'язку. У рамках дослідження планується аналіз архітектури та компонентів WebRTC, вивчення технічних вимог до мобільних додатків для ефективної інтеграції WebRTC, реалізація та оцінка функціональних можливостей WebRTC у мобільних додатках, а також порівняння WebRTC з іншими технологіями для реалізації функцій зв'язку.

Об'єктом дослідження є сам WebRTC, тоді як предмет дослідження — особливості та можливості його використання для реалізації функцій зв'язку в мобільних додатках.

У процесі дослідження будуть використані різні методи, зокрема аналіз літератури для розгляду наукових статей та технічної документації, експериментальний метод для розробки та тестування мобільного додатка з використанням WebRTC, а також порівняльний аналіз для визначення переваг та обмежень WebRTC у порівнянні з іншими технологіями для реалізації функцій зв'язку.

Розглянемо кілька прикладів використання технології WebRTC у відомих програмах. Наприклад, WhatsApp, який є однією з найпопулярніших платформ месенджерів, використовує WebRTC для надання можливості користувачам проводити голосові та відеодзвінки в режимі реального часу. Інший відомий месенджер, Facebook Messenger, також використовує технологію WebRTC для реалізації важливих функцій, зокрема відеозв'язку та голосових дзвінків. Це вказує на широкий обсяг використання WebRTC в галузі мобільного зв'язку.

У світі відеоконференцій Google Meet грає значущу роль, адже ця платформа від Google використовує WebRTC для створення надійного та якісного зв'язку під час віртуальних зустрічей та конференцій. Крім того, голосовий та текстовий чат типу, Discord, або вже старий Skype використовує WebRTC для реалізації голосового зв'язку в режимі реального часу. Це вказує на те, що технологія WebRTC виявляється дуже ефективною та важливою для різноманітних областей, включаючи розваги, бізнес та дистанційне навчання.

Всі ці приклади свідчать про широке використання WebRTC в різноманітних популярних програмах, підтверджуючи його значущість у сучасному мобільному зв'язку та віртуальному спілкуванні.

1 ОГЛЯД ТЕХНОЛОГІЇ WEBRTC

1.1 Визначення основних понять

В сучасному інформаційному суспільстві, де важливий акцент робиться на швидкому та ефективному зв'язку, виникає потреба у використанні передових технологій, таких як Веб-зв’язок у реальному часі (WebRTC). Для глибокого розуміння цієї технології необхідно розглянути основні поняття, а також здійснити аналіз архітектурних аспектів та принципів роботи WebRTC.

Веб-зв’язок у реальному часі (WebRTC): Це стандарт та ініціатива, що розширює можливості веб-перегляду, дозволяючи безпосередньо обмінюватися медіафайлами в режимі реального часу між браузерами. WebRTC є результатом спільних зусиль World Wide Web Consortium (W3C) та Internet Engineering Task Force (IETF).

JavaScript API: Інтерфейси прикладного програмування, що визначаються W3C та IETF, які забезпечують можливість взаємодії веб-додатків з WebRTC. API визначає функції для роботи з відео, аудіо, а також іншими функціональностями, які дозволяють забезпечувати зв'язок у реальному часі.

HTML5: Стандартні теги HTML5 використовуються для інтеграції WebRTC у веб-додатки, надаючи можливість вставляти відео- та аудіоелементи безпосередньо в веб-сторінку.

Архітектура та принципи роботи:

- Сесія (Session): Основна одиниця зв'язку у WebRTC. Сесія містить усі необхідні параметри для обміну медіаданими та управління сигналізацією;
- Сигналізація (Signaling): Механізм обміну сигналами між сторонами для встановлення, управління та припинення з'єднання;
- Вона може здійснюватися через різноманітні канали, такі як WebSocket або HTTP;

- ICE (Interactive Connectivity Establishment): Протокол для встановлення з'єднання, який дозволяє обходити обмеження, пов'язані з мережевими конфігураціями, і визначає найбільш ефективний маршрут для обміну даними;
- STUN (Session Traversal Utilities for NAT): Протокол, який дозволяє визначати зовнішній IP-адреси та порти в мережевих конфігураціях, що використовують NAT;
- TURN (Traversal Using Relays around NAT): Протокол, який надає засіб для обходу ситуацій, коли неможливо визначити прямий шлях для обміну даними через NAT.

Завершивши огляд основних понять WebRTC та розглянувши ключові архітектурні елементи та принципи його функціонування, можна зрозуміти, як ця технологія стає основою для надійного та ефективного веб-зв'язку у реальному часі.

1.2 Історія розвитку WebRTC

Веб-зв'язок у реальному часі (WebRTC) представляє собою фундаментальну технологію, яка виникла з постійної потреби в забезпеченні простого та ефективного механізму для реалізації веб-зв'язку. Історія розвитку WebRTC охоплює період від його появи та перших кроків до сучасного стану як ключової технології для веб-додатків у реальному часі.

Початком виникнення і розвинення технології WebRTC характерезують 2011-2013 роки. WebRTC був вперше представлений у травні 2011 року компанією Google. Основною метою було створення відкритого стандарту для реалізації миттєвого обміну відео та аудіо у веб-браузерах без необхідності встановлення додаткових плагінів чи додатків. З цього часу Google активно співпрацював із спільнотою та іншими ключовими гравцями в індустрії для стандартизації та розвитку технології.

У червні 2013 року було досягнуто історичного моменту, коли перша версія WebRTC була включена до браузера Google Chrome. Це дозволило розробникам використовувати API для створення власних веб-застосунків для обміну відео та аудіо.

Стандартизація та Розширення які відбувалися протягом 2014-2016 років WebRTC продовжував залучати увагу індустрії. У травні 2014 року, Internet Engineering Task Force (IETF) та World Wide Web Consortium (W3C) розпочали спільний процес стандартизації WebRTC, забезпечуючи його прийняття як відкритого стандарту для всієї індустрії. Поступово WebRTC отримав підтримку в інших великих браузерах, таких як Mozilla Firefox та Opera, що взяли на озброєння цю технологію та впровадили її у своїх веб-переглядачах.

Протягом періоду 2017-2019 рр. WebRTC не тільки став стандартом для відео- та аудіозв'язку, але й почав розширювати свій функціонал. Додаткові можливості включають роботу з екраном, обмін даними в реальному часі та підтримку конференційного зв'язку з трьома і більше учасниками. У 2018 році WebRTC також отримав статус рекомендованого стандарту від W3C, підкреслюючи його значущість та стабільність у сучасному веб-розвитку.

На сучасному етапі WebRTC став необхідною складовою для веб-додатків, що надають засоби відео- та аудіозв'язку. Зростання популярності віртуальних конференцій, вебінарів та віддаленої роботи підкреслило важливість та актуальність цієї технології.

Зараз WebRTC продовжує розвиватися, вдосконалюючи якість зв'язку, забезпечуючи безпеку та додаткові функції. Перспективи включають розширення можливостей для Інтернету речей (IoT), а також застосування у нових галузях, таких як розширена реальність (AR) та віртуальна реальність (VR).

У підсумку, історія розвитку WebRTC відзначається стрімким ростом та успіхами, зробивши його ключовою технологією для сучасного веб-зв'язку. 1.3 Аналіз сучасного стану розробок у галузі використання WebRTC у мобільних додатках

В сучасному цифровому ландшафті, де мобільні додатки відіграють ключову роль у повсякденному житті, технологія Веб-зв’язку у реальному часі (WebRTC) стає все більш важливою для забезпечення ефективного та інтерактивного зв'язку між користувачами. Аналіз сучасного стану розробок у галузі використання WebRTC у мобільних додатках розкриє основні тенденції, виклики та можливості цієї технології.

На сьогоднішній день спостерігається збільшення кількості мобільних додатків, які використовують WebRTC для забезпечення різноманітних функціональностей, таких як відеодзвінки, голосовий зв'язок та обмін даними в реальному часі. Додатки для особистого спілкування, робочих конференцій та соціальних мереж широко використовують цю технологію.

Однією з основних тенденцій є постійне розширення функціоналу WebRTC у мобільних додатках. Окрім базових функцій відео- та аудіозв'язку, розробники впроваджують нові можливості, такі як екранний обмін, віртуальні фільтри та взаємодія з іншими мультимедійними додатками.

Разом із зростанням популярності WebRTC у мобільних додатках, виникають нові виклики у сфері безпеки та конфіденційності. Оскільки дані передаються в режимі реального часу, забезпечення безпеки інформації та захисту приватності користувачів стає критичним завданням для розробників.

Другою значущою тенденцією є інтеграція WebRTC з іншими передовими технологіями. Розробники поєднують WebRTC зі штучним інтелектом, розпізнаванням обличчя та розширеною реальністю для покращення користувацького досвіду та створення новаторських додатків.

WebRTC не обмежується лише сферою особистого спілкування. Він знаходить застосування у бізнес-секторі для організації віддалених конференцій, у сфері охорони здоров'я для віртуальних консультацій та у навчальному секторі для організації віддалених занять.

Перспективи WebRTC у мобільних додатках обіцяє багатоцільове використання технології. Застосування в інтернеті речей (IoT), розширена реальність (AR) та віртуальна реальність (VR) можуть відкрити нові горизонти

та перетворити спосіб, яким користувачі сприймають та використовують мобільні додатки.

У великому контексті, аналіз сучасного стану розробок у галузі використання WebRTC у мобільних додатках свідчить про його широкий вплив на різні сфери, прискорюючи інновації у сфері комунікацій та забезпечуючи нові можливості для розробників та користувачів.

1.3 Переваги та недоліки використання WebRTC у мобільних додатках

Використання Веб-зв’язку у реальному часі (WebRTC) у мобільних додатках відкриває перед розробниками та користувачами широкий спектр можливостей та функціональностей. Аналіз переваг та недоліків цієї технології у науковому контексті дозволяє краще зрозуміти її вплив на мобільний розвиток.

Переваги використання WebRTC у мобільних додатках:

- Реальний час та безперервність. WebRTC дозволяє передавати відео та аудіо в режимі реального часу, що забезпечує інтерактивність та безперервність спілкування в мобільних додатках. Ця перевага особливо корисна для голосового та відеозв'язку, де важлива митливість та невідкладність;
- Простота впровадження. WebRTC має простий API, який дозволяє легко впроваджувати та інтегрувати його в мобільні додатки. Це робить технологію доступною розробникам, прискорюючи час розробки та зменшуючи складність впровадження;
- Кросплатформеність. WebRTC підтримується різними браузерами та мобільними платформами, що дозволяє побудовувати кросплатформенні додатки. Це робить технологію універсальною та ефективною для великої аудиторії користувачів;
- Низька затрата пропускної здатності. WebRTC оптимізує використання мережевих ресурсів, забезпечуючи високу якість передачі даних при

низькій пропускній здатності. Це важливо для користувачів із слабким Інтернет-з'єднанням, роблячи технологію ефективною в різних умовах.

Недоліки використання WebRTC у мобільних додатках:

- Проблеми з безпекою. Використання WebRTC може призвести до потенційних проблем щодо безпеки, таких як можливість перехоплення даних чи атаки на конфіденційність. Для деяких додатків, особливо у фінансовому або медичному секторах, це може бути значущим викликом;
- Залежність від браузера. WebRTC поки що має певну залежність від підтримки браузерами та мобільними платформами. Це може призвести до обмеженого вибору функцій для деяких користувачів, які використовують застарілі браузери чи платформи;
- Питання приватності. Розробники повинні вдосконалювати практики збору, збереження та обробки особистих даних для відповідності різноманітним регуляторним вимогам.

2 ОСОБЛИВОСТІ WEBRTC

2.1 Архітектура WebRTC

WebRTC розширює семантику клієнт-сервер, запроваджуючи парадигму одноранговогo зв’язку між браузерами. Найзагальніша архітектурна модель WebRTC (див. Малюнок 1-1) черпає натхнення з так званої трапеції SIP (протоколу ініціації сеансу) (RFC3261).

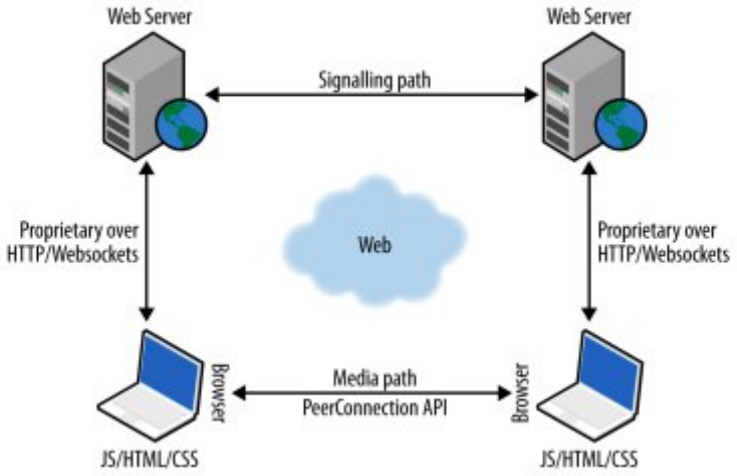

Рисунок 2.1 – Загальна схема WebRTC;

У моделі WebRTC Trapezoid обидва браузери запускають веб-додаток, який завантажується з іншого веб-сервера. Сигнальні повідомлення використовуються для встановлення та завершення зв’язку. Вони передаються за допомогою протоколу HTTP або WebSocket через веб-сервери, які можуть змінювати, перекладати або керувати ними за потреби. Варто зазначити, що сигналізація між браузером і сервером не стандартизована в WebRTC, оскільки вона вважається частиною програми (див. «Сигналізація» на сторінці 5). Що стосується шляху даних, PeerConnection дозволяє медіа-файлам передавати безпосередньо між браузерами без втручання серверів. Два веб-сервери можуть

спілкуватися за допомогою стандартного протоколу сигналізації, такого як SIP або Jingle (XEP-0166). В іншому випадку вони можуть використовувати власну сигналізацію.

Найпоширенішим сценарієм WebRTC, ймовірно, буде той, у якому обидва браузери запускають ту саму веб-програму, завантажену з однієї веб-сторінки. У цьому випадку трапеція стає трикутником (див. рис. 1-2).

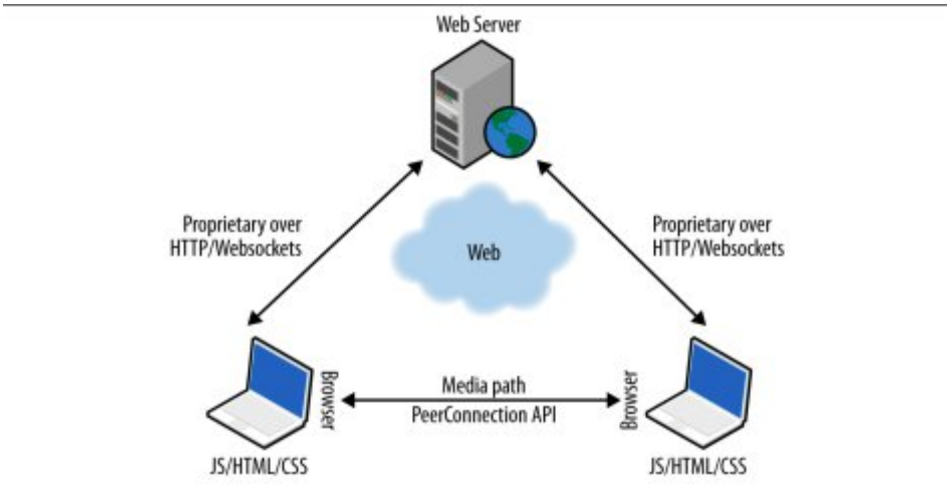

Рисунок 2.2 - Трьохстороння схема WebRTC;

Архітектура WebRTC (Web Real-Time Communication) визначає структуру та взаємодію компонентів, які забезпечують реалізацію зв'язку в режимі реального часу у веб-додатках. Ця архітектура, розроблена для взаємодії з веб-браузерами, є ключовою для впровадження мультимедійного обміну даними, такого як відео та аудіо, безпосередньо в середовище веб-перегляду.

В основі архітектури WebRTC лежить JavaScript API, який визначає інтерфейси для взаємодії з технологією. Ці API дозволяють розробникам взаємодіяти з компонентами WebRTC, встановлювати та керувати з'єднаннями, а також використовувати різноманітні мультимедійні можливості.

Забезпечення комунікації між клієнтами в WebRTC використовує сигналізацію. Це включає передачу метаданих та управління сеансами між

сторонами для належного налаштування з'єднань. Сигналізаційні дані можуть передаватися через різні канали, такі як WebSocket або HTTP.

ICE вирішує проблеми взаємодії з мережевими адресами. Він автоматично визначає доступні мережеві шляхи для забезпечення ефективної передачі даних, навіть при використанні NAT (Network Address Translation) та проксі-серверів.

STUN використовується для визначення реальних IP-адрес та портів клієнта, якщо він перебуває за NAT. Це важливо для правильної взаємодії при побудові підключень в реальному часі. Якщо пристрій не може здійснити прямий обмін даними (наприклад, через строгий NAT), TURN використовується для передачі даних через сервер-посередника, забезпечуючи надійний канал для обміну інформацією.

Медіа-двигуни відповідають за обробку та кодування/декодування аудіо та відео. WebRTC підтримує різноманітні кодеки, що дозволяє оптимізувати якість зображення та звуку під час передачі.

Архітектура WebRTC спирається на принципи peer-to-peer зв'язку, де клієнти взаємодіють без посередництва централізованого сервера. Це забезпечує найкращу продуктивність та знижує затримки, оскільки дані передаються безпосередньо між пристроями.

Переваги архітектури WebRTC:

- Простота інтеграції. Завдяки чітко визначеному API та гнучкості архітектури, WebRTC легко інтегрується в різноманітні веб-додатки;
- Динамічне керування мережевими ресурсами. Використання ICE, STUN та TURN дозволяє ефективно вирішувати проблеми взаємодії з мережевими шляхами та адресами;
- Масштабованість. Peer-to-peer архітектура дозволяє масштабувати систему відповідно до зростання об'єму користувачів без значного впливу на продуктивність.

Недоліки архітектури WebRTC:

- Проблеми з безпекою. Стосовно сигналізації та обміну ключами, можливість атак та перехоплення може піддавати сумніву безпеку системи;
- Залежність від браузеру. Непрозорість у роботі деяких компонентів може створювати залежність від веб-переглядача та може виникнути проблеми сумісності;
- Обмеження в мережах з обмеженим брандмауером. При роботі в корпоративних мережах з обмеженим брандмауером можуть виникати виклики, пов'язані з проксі та NAT, які можуть впливати на надійність підключення. Особливо часто стикаються із цією проблемою користувачі Windows.

Зрештою, архітектура WebRTC визначає ефективний та гнучкий спосіб забезпечення реального часу у веб-додатках, надаючи засоби для забезпечення зручного та якісного взаємозв'язку між користувачами.

2.2. Принципи роботи та особливості передачі даних в WebRTC

WebRTC, як технологічне рішення для реалізації зв'язку в режимі реального часу в мережі Інтернет, базується на конкретних принципах та інноваційних методах передачі даних. Цей розділ детально розглядає основні принципи функціонування WebRTC та аналізує особливості механізмів передачі даних, які забезпечують ефективність та якість комунікацій у режимі реального часу.

Принципи роботи WebRTC заснований на вище перерахованих принципах. Одним із основних принципів WebRTC є використання Peer-to-Peer з'єднань, де клієнти безпосередньо обмінюються даними між собою, минувши централізований сервер. Це зменшує затримки та покращує продуктивність, що є ключовим фактором для забезпечення реального часу взаємодії.

Важливим принципом є використання сигналізації для ініціювання, управління та завершення сесій між користувачами. Цей механізм дозволяє

визначати параметри з'єднань, обмінюватися інформацією про стан та керувати обраною конфігурацією.

Принцип Interactive Connectivity Establishment дозволяє автоматично визначати оптимальний мережевий шлях для передачі даних. Це особливо важливо при використанні NAT та проксі, де може бути кілька доступних шляхів.

Використання STUN дозволяє отримувати реальні IP-адреси та порти користувачів, що перебувають за NAT. Це допомагає визначати маршрутизацію та забезпечує ефективність взаємодії в умовах різних мережевих обмежень. В разі неможливості прямої взаємодії через NAT використовується TURN для передачі даних через сервер-посередника. Це гарантує надійність з'єднання в умовах складних мережевих умов.

Особливості передачі даних в WebRTC:

1) Надійність та низька затрата пропускної здатності. WebRTC використовує оптимізовані механізми передачі даних, які забезпечують високу якість з низькою пропускною здатністю мережі. Це особливо важливо для користувачів із слабким Інтернет-з'єднанням;

2) Медіа-двигуни в WebRTC використовують різноманітні кодеки для оптимізації якості передачі відео та аудіо. Це дозволяє користувачам насолоджуватися високоякісним зв'язком у режимі реального часу;

3) Механізми WebRTC динамічно регулюють використання бандвідту в залежності від мережевих умов. Це дозволяє підтримувати стабільність з'єднання при зміні умов мережі;

4) З урахуванням важливості захисту приватності, WebRTC використовує шифрування для забезпечення конфіденційності переданих даних. Це особливо важливо при обміні особистою інформацією;

5) WebRTC може використовувати різні протоколи передачі даних, такі як UDP (User Datagram Protocol) та TCP (Transmission Control Protocol), що забезпечує гнучкість та адаптабельність в різних сценаріях;

6) Механізми WebRTC розроблені з урахуванням роботи в умовах загальної Інтернет-мережі, де користувачі можуть підключатися через різні пристрої та мережеві середовища;

7) WebRTC підтримує різноманітні відео- та аудіокодеки, такі як VP8, VP9, H.264, Opus, що дозволяє оптимізувати передачу даних для різних вимог та умов.

Загально кажучи, принципи та особливості передачі даних в WebRTC відображаються у високій якості з'єднань, ефективному використанні мережевих ресурсів та забезпеченні конфіденційності та безпеки в режимі реального часу.

2.3. Безпека та конфіденційність у WebRTC

WebRTC, як і будь-яка технологія, що дозволяє обмін даними в режимі реального часу через мережу Інтернет, покладається на важливі принципи безпеки та конфіденційності. Розгляд цих аспектів є критичним для забезпечення надійності та захищеності веб-зв'язку між користувачами.

Однією з ключових аспектів безпеки є використання шифрування даних. WebRTC використовує шифрування на різних рівнях комунікації, починаючи від сигналізації та закінчуючи передачею медіа-даних. У процесі сигналізації, де встановлюються параметри з'єднання та обмінюються метаданими, використовується шифрування для захисту від перехоплення та зміни даних. Це допомагає уникнути атак, таких як «Man-in-the-Middle», де хакер може отримати та модифікувати передавані дані.

Що стосується самої передачі медіаданих, WebRTC використовує шифрування, що ґрунтується на протоколі SRTP (Secure Real-Time Transport Protocol). SRTP забезпечує захист від перехоплення, модифікації та відтворення аудіо- та відеоданих. Це особливо важливо при обміні конфіденційною інформацією, такою як відеоконференції в бізнес-середовищі або особисті розмови.

Однак, треба врахувати, що хоча WebRTC використовує потужні механізми шифрування, важливо також враховувати питання ключового управління та обміну ключами. Шифрування використовується для того, щоб зробити дані нерозбірливими для неповідомлених сторін, але якщо ключі стають доступними несанкціонованим особам, це може призвести до компрометації конфіденційності. Тому безпека ключового обміну є критичною для системи WebRTC.

Іншим аспектом є справедливий контроль доступу. WebRTC намагається мінімізувати ризики, пов'язані із стандартними протоколами веб-зв'язку, такими як HTTP, і використовує протоколи, що сприяють високому рівню безпеки.

Окрім того, WebRTC уникне проблем з витоком інформації через використання UDP (User Datagram Protocol) для передачі даних. В порівнянні з TCP, яке є більш надійним, але менш ефективним у реальному часі, UDP дозволяє більш ефективну передачу даних у режимі реального часу.

Важливим елементом безпеки в WebRTC є ідентифікація та аутентифікація. Протоколи, такі як DTLS (Datagram Transport Layer Security) та SRTP, використовуються для захисту від атак, пов'язаних із зловживанням ідентифікаторів або підробкою відправника.

У разі використання WebRTC в корпоративному або організаційному середовищі, важливо враховувати аспекти, пов'язані з політикою безпеки та зберіганням даних. Це включає в себе визначення та дотримання стандартів щодо зберігання інформації, а також моніторинг заходів безпеки на рівні системи та мережі.

Нарешті, безпека та конфіденційність у WebRTC є багатоаспектним завданням, яке вимагає постійного вдосконалення та адаптації до нових викликів. Розуміння та впровадження цих принципів є важливими для забезпечення довіри користувачів до технології та підтримки стійкості системи у реальних умовах експлуатації.

Щоб забезпечити ефективну захист інформації, WebRTC використовує сучасні криптографічні протоколи. DTLS, наприклад, використовується для

забезпечення конфіденційності та цілісності даних під час передачі. Цей протокол дозволяє встановлювати безпечне з'єднання між вузлами та забезпечує ефективний обмін ключами для шифрування.

Однак, на попри ці заходи, важливо враховувати індивідуальні аспекти безпеки на рівні застосунків, таких як можливості управління доступом, ідентифікації користувачів та контролю за безпекою на рівні додатків. Розробники повинні бути відповідальними за визначення правильних політик безпеки та використання належних методів аутентифікації.

Також важливо враховувати аспекти глобальної безпеки, такі як захист від різноманітних атак, включаючи відмову в обслуговуванні (DDoS) та спроби несанкціонованого доступу. WebRTC, як і будь-яка технологія, може стати об'єктом атак, і тому важливо використовувати інструменти захисту мережі та моніторингу для реагування на потенційні загрози.

Однією з важливих аспектів конфіденційності є обробка та зберігання особистих даних користувачів. Розробники повинні дотримуватися високих стандартів зберігання та обробки особистих даних відповідно до вимог законодавства та політик конфіденційності.

Крім того, WebRTC повинен бути гнучким у реагуванні на швидкі зміни в сфері кібербезпеки. Вдосконалення протоколів шифрування та впровадження нових заходів безпеки є невід'ємною частиною стратегії розвитку технології. Регулярні оновлення та покращення служать запорукою того, що WebRTC залишається відповідним та захищеним від сучасних кіберзагроз.

Враховуючи вище викладене, безпека та конфіденційність у WebRTC є системним завданням, що вимагає комплексного підходу. Здатність ефективно захищати дані та забезпечувати конфіденційність користувачів визначає успіх та прийняття технології серед широкого загалу. Розробники та оператори мереж повинні залишатися завжди на чолі подій у сфері кібербезпеки для забезпечення стабільної та захищеної екосистеми WebRTC.

3 РЕАЛІЗАЦІЯ ФУНКЦІЙ ЗВ'ЯЗКУ В МОБІЛЬНИХ ДОДАТКАХ

3.1 Технічні вимоги до мобільних додатків для використання WebRTC

Зрозуміло, що WebRTC це браузерна технологія, яка має відкритий код, що дозволяє розробникам використовувати її для найрізноманітніших цілей. Для будь якого додатку з використанням WebRTC потрібно виконати чотири основні вимоги: інтерфейс користувача, сервер з базою даних для зберігання важливої інформації (наприклад сесії та дані користувача), та мати підключення до інтернету для взаємодії з API WebRTC.

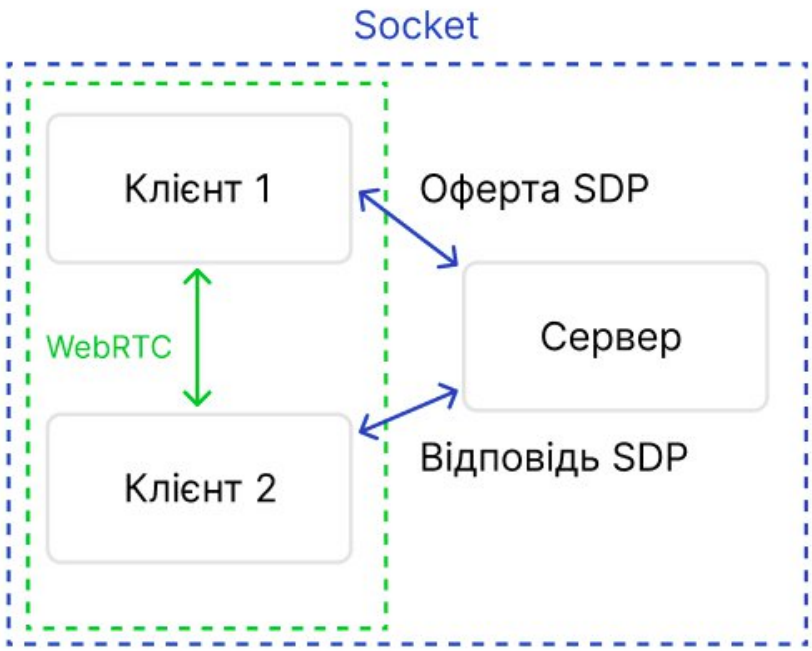

Рисунок 3.1 - Схема роботи додатку

Але вище наведена схема - загальна, а також дуже спрощена відносно реальності. Насамперед, функція звичайного відеодзвінка вже має на увазі використання декількох сервісів технології WebRTC.

MediaStream - згідно з офіційною документацією Mozilla Developer Network представляє собою потік медіа даних, який складається з відео і аудіо треків, які, відповідно, передають однойменну інформацію. За допомогою цього інтерфейсу можна додавати джерела зчитування інформації такі як камера та мікрофон, отримувати доступ до них, та передавати цю інформацію.

RTC Peer Connection - це стороній API, який і буде слугувати комутатором між користувачами з'єднуючи їх за допомогою власних сервісів. Такі сервіси відповідать за адресацію, та передачу інформації отриманої з JavaScript інтерфейсу MediaStream.

RTC Data Channel - це канал, який відповідає саме за передачу між користувачами створюючи WebSocket з використанням UPD-потоку, протоколу SCTP, що значно поліпшує продуктивність та ефектвність з'єднання в порівнянні з TCP.

Рисунок 3.2 - Схема роботи WebRTC у додатку

Обрання платформи розробки є ключовим кроком в розробці мобільного додатку. Гарними та зручними варіантами можуть бути такі мови програмування як JavaScript. JavaScript є основною мовою для розробки мобільних додатків, а React Native та NativeScript дозволяють створювати кросплатформенні додатки з використанням одного коду для обох платформ. Це полегшує розробку та обслуговування, а також забезпечує високу переносимість коду.

Swift (iOS) та Kotlin (Android) надають широкий інструментарій для розробки мобільних додатків з WebRTC для своєї платформи.

Dart із фреймворком Flutter дозволяє створювати кросплатформенні додатки. Ця МП являє собою відносну нову технологію і дає можливість комфортно писати великі, масштабовані додатки, а велика база плагінів допоможе досягти успіху в багатьох сферах. До того ж можливість змінювати інтерфейс та функціонал для конкретної платформи не торкаючись змін в інших розцінюється як суттєва перевага.

Java є традиційним вибором для мобільних додатків на базі Android. Розробник можете використовувати Android SDK для інтеграції різних технологій, у тому числі WebRTC у мобільний додаток.

В даній роботі всі приклади використання коду будуть виконані з МП Dart та фреймворком Flutter. Вибір зумовлений тим, що для розробки мобільного додатку з підтримкою WebRTC на платформі Flutter цей фреймворк підтримує плагіни для інтеграції з низкою низькорівневих API, таких як WebRTC та легко інтегрується. Другим аспектом вибору Flutter є те, що на ньому можна провести низку тестів для аналізу WebRTC у мобільних додатках.

Також варто зазначити, що не менш важливим критерієм для створення додатку є інформаційна база про саму технологію або фреймворк, її функціонал та інструментарій, звичайно, документація або гайд по технології WebRTC у рамках обраної мови програмування та структури програми. За певних обставин або умов завдань вибір мови програмування залежить від замовника, бюджету, строків або знань і умінь команди. Адже кожен додаток переслідує свої цілі та має свої методи їх вирішення. Наприклад JavaScript буде ідеально

працювати з API, який використовує JSON, або Java для реалізації великого додатку, який потребує багато часу і дає впевненість того, що при розробці, потрібні залежності, або функції не застаріють до закінчення розробки.

3.2 Процес інтеграції WebRTC в мобільні додатки

Початком інтеграції WebRTC технології у будь який додаток є саме встановлення однойменного плагіну. У Flutter всі плагіни, включаючи WebRTC встановлюється за допомоги команди “flutter run <name_of_plugin>”. Всі встановлені залежності і їх версії можна побачити в файлі “pubspec.yaml”. Цей файл використовується для опису залежностей проекту, а також для вказівки різноманітних налаштувань, таких як назва проекту, версія, конфігурація ресурсів та інше. В кожній мові програмування є схожий файл, який виконує ті самі функції, але має іншу назву, розширення, та синтаксис.

Наступним кроком є сигналізація. Сигналізація являє собою механізм обміну інформацією між вузлами, який допомагає їм домовитися про параметри з'єднання. Ця інформація включає в себе адреси та порти для передачі медіаданих, кодеки, а також параметри забезпечення якості та інші налаштування.

Щоб встановити з'єднання WebRTC, вузли (наприклад, браузери чи мобільні додатки) повинні обмінюватися даними сигналізації. Ці дані включають інформацію, яка описує сесію (визначає і зберігає інформацію про параметри з'єднання, такі як тип медіа (аудіо, відео, обмін даними), кодеки, роздільна здатність і т. д.) і включає кандидатів ICE.

Кандидати ICE - це інша частина сигналізації, пов'язана з технологією Interactive Connectivity Establishment (ICE). ICE використовує кандидатів для визначення оптимального маршруту для обміну даними між вузлами, особливо в умовах використання NAT (Network Address Translation). Кандидати можуть включати різні IP-адреси та порти, які можуть бути використані для встановлення з'єднання.

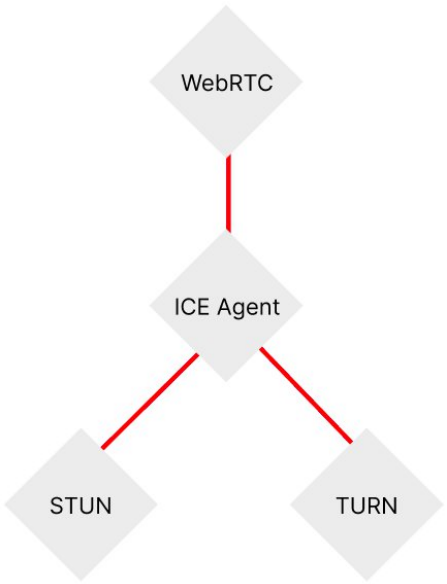

Рисунок 3.3 - кандидати ICE

Але плагін, що найменше, у Flutter не передбачає сервер сигналізації за замовчуванням. Це значить, що він створюється розробником і передбачає індивідуальний функціонал. Натомість можна знайти сервера сигналізації з відкритим кодом, що можуть бути використані розробником. В цій роботі буде використаний власний сервер з використанням Node.js та GraphQL для обміну інформації з сервером.

Щоб налаштувати сервер WebRTC за допомогою Node.js, одним із популярних варіантів є використання фреймворку WebRTC із відкритим кодом під назвою “mediasoup”. Mediasoup розроблено для роботи з Node.js і спрощує процес створення додатків для спілкування в реальному часі.

Після встановлення залежності її можна імпортувати у програму Node.js і налаштувати її відповідно до технічного завдання. Базовими завданнями є налаштування маршрутизатора “mediasoup”, створення споживачів і виробників для обробки медіа-потоків і налаштування зв’язок WebSocket для сигналізації між клієнтами та сервером. Обов’язково потрібно враховувати міркування безпеки, такі як використання захищених з’єднань WebSocket (WSS) і

впровадження належних механізмів автентифікації, наприклад, JWT-токенів. Після налаштування сервера “mediasoup” можна переходити для розробки клієнтської програми, в даному випадку - мобільного додатку.

Важливим етапом розробки є робота з помилками і цей випадок не виняток. Створення серверу та клієнтської сторони завжди задача не з простих, тому відстежити помилки і розуміти їх дуже важливо. На щастя у WebRTC є така можливість. За замовчуванням всі помилки можна дублювати у журналі налагодження, який створюється в коді як окремий екземпляр.

Щоб надати змогу додатку використовувати весь функціонал технології WebRTC у самому додатку потрібно ініціалізувати сервер і вказати посилання на нього. Після чого можливо використовувати методи підключення і відключення від нього в мобільному додатку.

3.3 Функціональні можливості WebRTC для забезпечення зв'язку в мобільних додатках

WebRTC є набором технологій, що надають веб-браузерам та іншим платформам засоби для обміну аудіо, відео та даними в реальному часу, та без необхідності встановлення сторонніх плагінів чи додаткового програмного забезпечення. Функціонал WebRTC включає в себе ряд ключових аспектів.

Однією з ключових функцій WebRTC в мобільних додатках є забезпечення високоякісного голосового та відеозв'язку. За допомогою WebRTC реалізовано роботу з аудіо та відео стрімінгом, оптимізовано обробку широкого спектру кодеків для найкращого співвідношення якості та пропускної здатності. Крім того, WebRTC надає можливості для шифрування медіа-даних, що забезпечує високий рівень конфіденційності та безпеки взаємодії.

Важливою функцію технології є кодування та декодування інформації, яка надсилаються. Тому мова пойде про термін, який зустрічався, але на ньому не акцентувалась увага, але зараз саме той час, коли потрібно про це дізнатись. Кодек - апаратний або програмний процес, який стискає та розпаковує великі

обсяги даних. Кодеки використовуються в програмах для відтворення та створення медіафайлів для користувачів, а також для надсилання медіафайлів через мережу. Кодеки невидимі для кінцевого користувача та вбудовані в програмне, або апаратне забезпечення пристрою. За їхньою допомогою передається та читається весь контент. Технолоція підтримує широкий спектр кодеків для аудіо (наприклад, Opus) та відео (наприклад, VP8, VP9, H.264), що забезпечує високу якість та широку сумісність.

Можливість отримувати доступ та зчитувати дані з пристроїв за допомогою API є ключовою функцію WebRTC що робить цю технологію унікальною не схожу на інші. Назва цього API методу getUserMedia і згідно з MDN це є метод, при виконанні, викликає спливаюче діалог, що запитує дозвіл користувача використання медіапристрою (камера, мікрофон). Результат повертає промис, що містить потік, що складається з треків (доріжок), що містять необхідні типи медіа. Цей потік може включати, наприклад, відеотрек, створений апаратним засобом, або віртуальним відеоджерелом, такими як камера, пристрій відеозапису, сервіс обміну зображеннями і т.д); аудіотрек, створений фізичним або віртуальним аудіоджерелом, наприклад, мікрофоном, аналого-цифровим перетворювачем звуків та можливі інші типи треків.

Він повертає Promise, який, у разі згоди користувача, дозволяється MediaStream об'єктом. Якщо користувач відмовляє в роздільній здатності або медіапристрій не доступний, тоді промис скасовується з об'єктами типу NotAllowedError або NotFoundError відповідно.

Головний компонент WebRTC, що відповідає за встановлення та управління підключенням між двома або більше пристроями є RTCPeerConnection. Інтерфейс RTCPeerConnection представляє з'єднання WebRTC між локальним бенкетом (учасником з'єднання) на локальному комп'ютері та віддаленим бенкетом на віддаленому комп'ютері. Він надає методи для з'єднання з віддаленим учасником з'єднання, обслуговування, моніторингу та закриття з'єднання.

WebRTC автоматично керує вибором транспортних протоколів (UDP, TCP, SCTP) та можливістю тунелювання через NAT, функції, які дозволяють обійти труднощі, пов'язані із розміщенням у мережі.

Network Address Translation (NAT) використовується для отримання пристрою користувача в полі публічних IP адресів. Сам роутер буде мати громадську IP-адресу і всі пристрої підключено до роутеру будуть мати приватну IP-адресу. Запити буде переведено з пристрою з приватним IP до роутеру з публічною IP-адресою та з унікальним портом.

Протокол "Session Traversal Utilities for NAT" (STUN) призначений для визначення громадської IP-адреси користувача та виявлення будь-яких обмежень у маршрутизаторі, які можуть перешкоджати прямому з'єднанню з іншим користувачем.

Клієнт відправляє запит до STUN-сервера в Інтернеті, який відповідає, повідомляючи громадську IP-адресу клієнта та інформацію про те, чи доступний клієнт на маршрутизаторі, який використовує технологію Network Address Translation (NAT).

Якщо приводити приклад, що Клієнт 1 і Клієнт 2 хочуть провести відеодзвінок за допомогою своїх веб-камер. Кожен з них знаходиться за різними маршрутизаторами, які використовують NAT для присвоєння приватних IP-адрес їх пристроям. Приватні IP-адреси ускладнюють пряме підключення між ними через Інтернет. Цей процес описаний алгоритмом наведеним нижче.

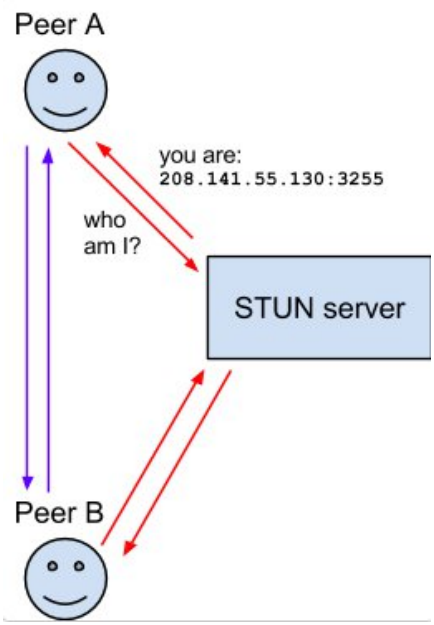

Рисунок 3.4 - Процес роботи STUN серверу

Таким чином, завдяки STUN, клієнти можуть визначити свої громадські IP-адреси та обійти обмеження, які можуть виникнути через використання NAT, забезпечуючи ефективний відеодзвінок через Інтернет.

Деякі роутери, які використовують NAT, застосовують обмеження, відоме як "Симетричний NAT". Це означає, що роутер прийматиме з'єднання лише від тих пір, які ви вже підключалися раніше.

Технологія "Traversal Using Relays around NAT" (TURN) призначена для обходу обмеження симметричного NAT за допомогою відкриття з'єднання із сервером TURN і передачі всієї інформації через цей сервер. Створюється з'єднання із сервером TURN та повідомляється всім пірингам відправляти пакети на сервер, який потім буде пересилати їх вам. Звісно, це супроводжується деякими накладними витратами, тому використовується лише у випадках, коли інших альтернатив немає.

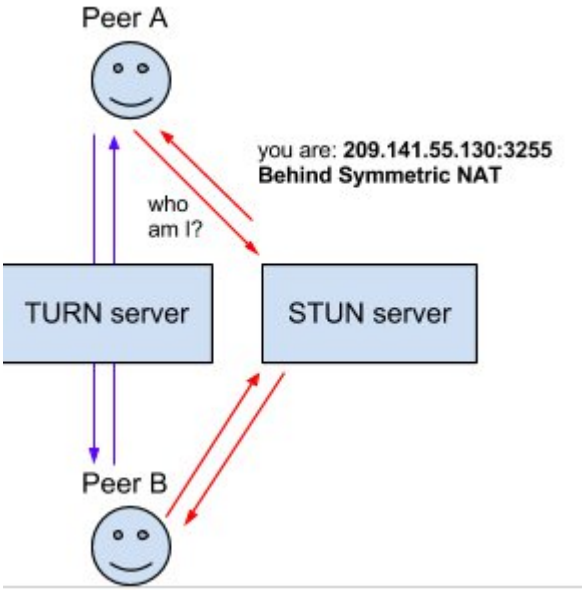

Рисунок 3.5 - Процес роботи TURN серверу

WebRTC має таке поняття як протокол опису сесій (Session Description Protocol, SDP) та є стандартом для опису мультимедійного вмісту з'єднання, такого як роздільність, формати, кодеки, шифрування та інші параметри, з метою того, щоб обидва піринга могли зрозуміти один одного під час передачі даних. З іншого боку, це суттєво метадані, які описують контент, а не сам медіа-вміст.

Згідно з MDN, SDP не є справжнім протоколом, а отже, це формат даних, який використовується для опису з'єднань, що обмінюються медиаконтентом між пристроями (див. рис. 3.6).

Для створення ефективного з’єднання WebRTC необхіден проміжний сервер, який буде виконувати функції канала сигналізації. Він виступає засобом комунікації для обміну інформацією перед встановленням з’єднання. Але вказати це можна навіть вручну, і ця інформація може надаватися іншому клієнту як лист електронної пошти, або навіть листівка написана від руки. Тобто, SDP і є сигналом на встановлення з’єднання, яка містить всю необхідну інформацію про клієнта на стороні якого ця оферта була згенерована.

```
{
  type: 'offer',
  offer: {
    sdp: 'v=0\r\n' +
      'o=- 7894789385469827194 2 IN IP4 127.0.0.1\r\n' +
      's=-\r\n' +
      't=0 0\r\n' +
      'a=extmap-allow-mixed\r\n' +
      'a=msid-semantic: WMS\r\n',
    type: 'offer'
  },
  calleeId: '1'
}
```

Рисунок 3.6 – Приклад зашифрованого запиту SDP, який зберігає іноформацію про користувача

Інформація, яка обмінюється, представляє собою пропозицію та відповідь, які містять лише SDP, що було згадано раніше.

Один з вузлів (в даному випадку вузол А), який ініціює з'єднання, генерує пропозицію. Потім цю пропозицію вони відправляють іншому вузлу (вузол Б) через обраний канал сигналізації. Вузол Б отримує пропозицію з сигнального каналу і створює відповідь. Далі цю відповідь вони надсилають назад вузлу А за допомогою сигнального каналу.

Якщо розмова йде про канали, то варто згадати існуючі, бо кожен з них має свої методи та завдання. У контексті WebRTC термін "канали" вказує на різновиди з'єднань та комунікаційних ліній, які використовуються для взаємодії між пристроями.

Основні типи каналів у WebRTC включають:

- Канал сигналізації - використовується для обміну сигналами між пристроями для встановлення та керування з'єднанням. На цьому каналі передаються сигнали, такі як пропозиції та відповіді SDP;
- Канал передачі медіаданих - використовується для потокової передачі аудіо та відео після встановлення з'єднання;

- Канал управління кандидатами ICE - використовується для обміну інформацією про доступні кандидати для встановлення підключення;
- Канал даних - дозволяє обмінюватися додатковими даними або текстовими повідомленнями між пристроями.

Кожен канал виконує конкретну роль у взаємодії пристроїв, забезпечуючи ефективний обмін інформацією та управління з'єднанням (див. рис. 3.7).

Повертаючись до сигналізації варто наголосити, що обробка сигналізації в WebRTC є ключовим етапом для встановлення та керування з'єднанням між піринговими пристроями. Сигналізація визначає процес обміну інформацією між пристроями, необхідним для правильного встановлення та управління передачею даних.

Рисунок 3.7 - Схема роботи WebRTC

Важливим інструментом є Trickle ICE, підхід до встановлення підключення за допомогою Interactive Connectivity Establishment (ICE), який спрощує процес обміну кандидатами та дозволяє динамічно формувати списки кандидатів під час взаємодії між піринговими пристроями в режимі реального часу. ICE використовується для обходу обмежень, таких як NAT (Network Address Translation), та визначення оптимальних маршрутів для обміну даними між пристроями.

Замість традиційного підходу, де всі кандидати повинні бути зібрані перед відправкою, Trickle ICE дозволяє пристроям відправляти кандидати один за одним, динамічно доповнюючи список під час процесу встановлення з'єднання.

Процес передачі кандидатів починається відразу після початку взаємодії пристроїв, не чекаючи повного збору всієї інформації. Це дозволяє прискорити час встановлення з'єднання, особливо в умовах реального часу.

Trickle ICE допомагає обійти обмеження, пов'язані з використанням NAT, та дозволяє пристроям знаходити оптимальні маршрути для обміну даними. Це особливо важливо в сучасних мережах, де NAT широко застосовується.

Оскільки Trickle ICE дозволяє починати обмін кандидатами миттєво, цей підхід особливо ефективний у випадках реального часу, наприклад, відеодзвінків чи ігрових додатків, де важлива мінімізація затримок.

Trickle ICE спрощує процес встановлення з'єднання, дозволяючи плавно доповнювати список кандидатів та ефективно обходити обмеження, що може виникнути через використання NAT у мережі.

WebRTC має вбудовані механізми безпеки, які забезпечують конфіденційність, цілісність та доступність взаємодії між піринговими пристроями.

По-перше, WebRTC використовує шифрування за замовчуванням, щоб забезпечити конфіденційність передачі даних між пристроями. Всі аудіо- та відеодані, а також сигналізаційні дані, шифруються за допомогою протоколу DTLS (Datagram Transport Layer Security) та SRTP (Secure Real-time Transport Protocol).

По-друге, WebRTC використовує механізми контролю доступу для управління обміном мультимедійними даними та інформацією сигналізації. Управління CORS (Cross-Origin Resource Sharing) та різні політики безпеки браузера допомагають уникнути потенційних загроз, пов'язаних з доступом до мультимедійних ресурсів.

По-третє, WebRTC підтримує функції, такі як "getUserMedia" та відомість користувача про активність камери та мікрофона, що робить можливим контроль користувача над обміном аудіо- та відеоданими, покращуючи приватність та забезпечуючи, що користувачі повідомлені про активність медіаресурсів.

3.4 Спосіб використання WebRTC в проектах

Розглядаючи, мобільний додаток, який був розроблений для цієї роботи, а саме, програма для здійснення дзвінків та комунікації користувачів, яка дозволяє шукати людей, додавати їх у друзі та здійснювати спілкування з ними, як за допомогою текстових повідомлень, так і за допомогою відеодзвінків.

Основною ціллю цього додатка було використання WebRTC, для написання логіки комунікації та передачі відео з найменшою можливою затримкою. Також, не менш важливим завданням, дозволити віддалену взаємодію з веб-камерою та обробку відеоданих в реальному часі.

Архітектура додатку виглядає наступним чином, на цій схемі розкрита передача медіаданих між декількома користувачами за допомогою технології WebRTC. Саме так працюють такі ж програми як Zoom або Google Meet.

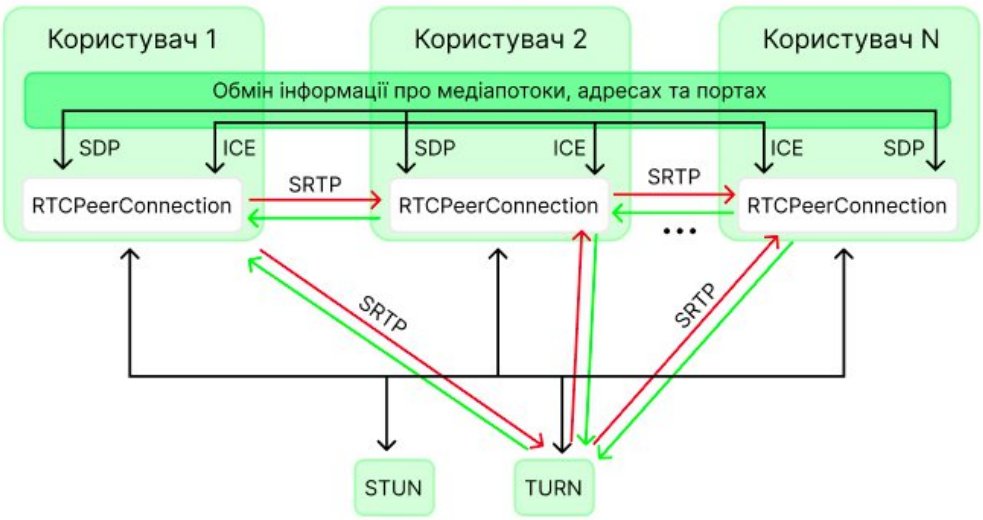

Рисунок 3.8 - Детальна схема роботи додатку з WebRTC

Для інтеграції в Dart додаток був використаний вищезгаданий плагін “Flutter WebRTC”. За допомогою методів та класів цього плагіну розробник може встановлювати з’єднання з іншими учасниками конференції. У коді цей процес інтеграції реалізован таким чином як у лістингу нижче (див. рис. 3.9).

```
final Map<String, dynamic> configuration = {'iceServers': [{'url':
'stun:stun.l.google.com:19302'}]};
final Map<String, dynamic> offer = await createOffer(configuration);

final RTCPeerConnection peerConnection = await
createPeerConnection(configuration, {});

await
peerConnection.setRemoteDescription(RTCSessionDescription(offer['sdp'],
offer['type']));

final Map<String, dynamic> answer = await
createAnswer(peerConnection.localDescription);

await
peerConnection.setLocalDescription(RTCSessionDescription(answer['sdp'],
answer['type']));
```

Рисунок 3.9 – Код інтеграції, та ініціалізації дзвінка

Алгоритм інтеграції виглядає наступним чином:

1) Створення конфігурації серверу, який включає один ICE сервер, який вказує на сервер обробки обміну кандидатами;

2) Далі створюється змінна, яка відповідає за зберігання об’єкту для підключення. Вона створюється за допомогою методу з плагіну;

3) Створюється змінна яка представляє з себе підключення між локальними та віддаленими пірами. Далі він буде використовуватись для встановлення та управлінням з’єднанням;

4) Встановлюється віддалений опис, це об'єкт, що містить SDP. Він слугує для з'єднання, використовуючи отриману пропозицію;

5) Створюється об'єкт, представляючи відповідь за допомогою функції, яка є асинхронною, що в кінцевому результаті генерує відповідь на основі отриманого опису;

6) Встановлюється локальний опис для з'єднання, використовуючи створену відповідь. Це дозволить завершити процес встановлення з'єднання між користувачами.

Важливим для створення додатку з використанням мікрофону та камери є саме надати доступ до них додатку.

Інтегрування у додаток технології вже дозволяє перейти до створення функції дзвінка. Вона декілька відрізняється від функції описаної вище, а саме тим, що має ще кілька полів, які включають у себе ідентифікатор користувача, який передається на сервер, який був також створений для керування всіма дзвінками (див. рис. 3.8).

```dart
Future<void> initCall(User callee) async {
  final Map<String, dynamic> configuration = {
    "iceServers": [
      {"url": "stun:stun.l.google.com:19302"}
    ]
  };

  _peerConnection = await createPeerConnection(configuration, {});

  final offer = await _peerConnection.createOffer({});
  await _peerConnection.setLocalDescription(offer);

  final String calleeId = callee.id.toString();
  final Map<String, dynamic> callData = {
    'type': 'offer',
    'offer': offer.toMap(),
    'calleeId': calleeId,
  };

  print(callData);

  await _sendCallData(callData);
}
```

Рисунок 3.8 – Функція (Dart/Flutter) виклику співрозмовника на сороні клієнта

```javascript
io.on("connection", (socket) => {
  let socketId = socket.id;
  const userId = socket.handshake.query.userId;
  console.log(`New connection with socket ID: ${socketId} bu user ${userId}`);
  socket.on("getRouterRtpCapabilities", () => {···
  });
  socket.on("authenticate", (userId) => {···
  });
  socket.on("RTCPeerConnection", (data) => {
    const userId = data.userId;
    peerConnections.set(userId, data.peerConectionId);
  });
  socket.on("offer", (data) => {
    const callerPeerConnection = findPeerConnectionIdByUserId(data.callerId);
    const calleeSocketId = findSocketIdByUserId(data.calleeId);
    if (calleeSocketId) {
      io.to(calleeSocketId).emit("offer", {
        ...data,
        peerConnectionId: callerPeerConnection,
      });
    } else {
      console.log(`Socket ID not found for user ID: ${data.calleeId}`);
    }
  });
  socket.on("answer", (data) => {
    const callerId = findSocketIdByUserId(data.callerId);
    const peerConnectionId = findPeerConnectionIdByUserId(data.callerId);

    if (peerConnectionId) {
      io.to(callerId).emit("setRemoteDescription", data);
    } else {
      console.log(`Peer connection not found for user ID: ${data.callerId}`);
    }
  });
  socket.on("disconnect", () => {···
  });
```

Рисунок 3.9 – Сигнальний сервер реалізований за допомогою Socket.io

Такий проміжний сервер (див. рис. 3.9) дозволяє створити не тільки створювати конференцію між двома клієнтами, а створити дзвінок саме з обраним користувачем або користувачами. Схематично, згідно з MDN це працює наступним чином (див. рис. 3.10). Де сервер, який налаштований на прийом WebRTC запитів та з налаштованими сокетами виступає як сигнальний канал. І за рахунок постійного підключення до клієнту передасть саме тому клієнту інформацію про виклик в реальному часі, що на клієнті можна налаштувати як спливаюче повідомлення з кнопкою відповісти, або скинути виклик.

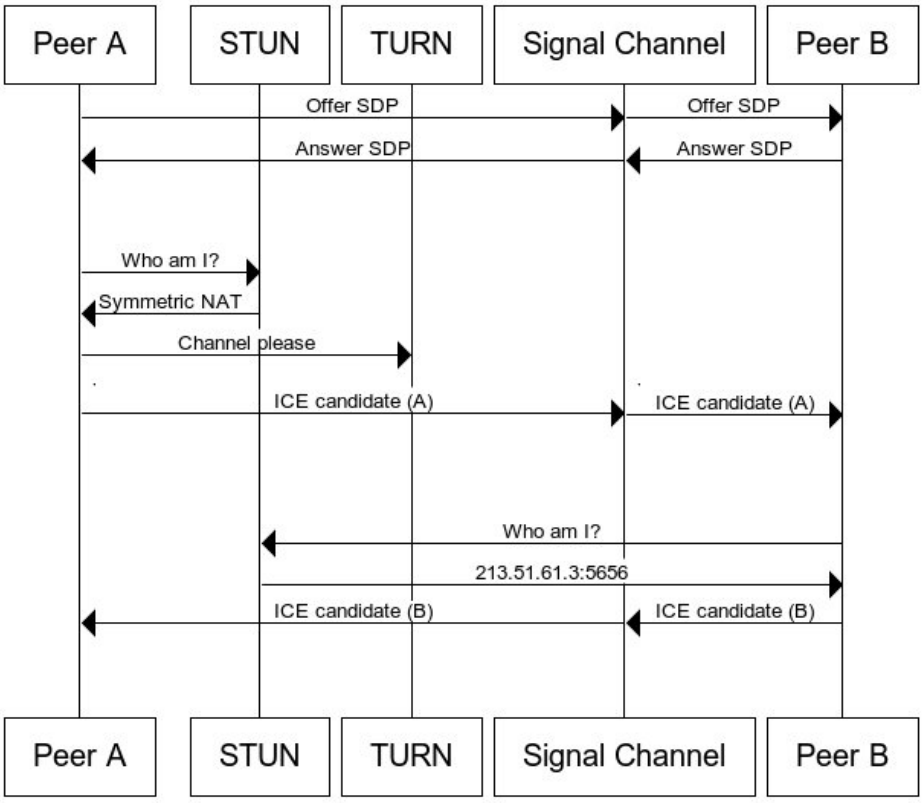

Рисунок 3.10 - Взаємодія компонентів технології

Після того як клієнт приймає дзвінок, створюється його власна оферта, яка включає кандидата ICE, а також налаштування камери і мікрофону у вигляді, або true, або false. Наступним кроком відповідь надсилається на сервер (сигнальний канал) і відповідь про дзвінок повертається адресанту.

```dart
Future<void> handleOffer(Map<String, dynamic> offerData) async {
  final Map<String, dynamic> configuration = {
    "iceServers": [
      {"url": "stun:stun.l.google.com:19302"}
    ]
  };
  final String? peerConnectionId = offerData['peerConnectionId'] as String?;
  _peerConnection = await createPeerConnection(configuration, {});
  peerConnections[peerConnectionId!] = _peerConnection;
  final RTCPeerConnection? peerConnection =
      CallService.peerConnections[peerConnectionId];
  if (peerConnection != null) {
    final Map<String, dynamic>? offerMap =
        offerData['offer'] as Map<String, dynamic>?;
    if (offerMap != null &&
        offerMap['sdp'] != null &&
        offerMap['type'] != null) {
      final RTCSessionDescription offer = RTCSessionDescription(
        offerMap['sdp'],
        offerMap['type'],
      );
      await _peerConnection.setRemoteDescription(offer);
      final RTCSessionDescription answer =
          await _peerConnection.createAnswer({});
      await _peerConnection.setLocalDescription(answer);
      final Map<String, dynamic> answerData = {
        'type': 'answer',
        'answer': answer.toMap(),
        'callerId': offerData['callerId'],
      };
      await _sendAnswerData(answerData);
    } else {
      print("Invalid offer data structure");
    }
  } else {
    print("Peer connection null");
  }
}
```

Рисунок 3.11 – Функція обробки відповіді від співрозмовника

Останім етапом встановлення з’єднання є створення Data Chanel між пірами. Він створиться автоматично, коли об’єкт піру на обох сторонах буде мати SDP і ICE-кандидати один одного. Це дуже важливо, щоб обійти фаєрвол, в іншому випадку є ризик, що не вдасться передавати відео та аудіо сигнал, хоча канал зв’язку буде працювати і з’єднувати клієнтів.

Реалізація відео і аудіозв’язку представляє собою додавання потоків даних отриманих з мікрофону та камери до каналу даних внутрішнім методом технології (див. рис. 3.12).

```
Future<void> establishCall(Map<String, dynamic> offerData) async {
  _peerConnection.onIceCandidate = (RTCIceCandidate candidate) {
    SocketService.iceCandidate(candidate);
  };
  final localVideoTrack = await navigator.mediaDevices.getUserMedia({
    'video': true,
  });
  final localAudioTrack = await navigator.mediaDevices.getUserMedia({
    'audio': true,
  });
  await _peerConnection.addStream(localVideoTrack);
  await _peerConnection.addStream(localAudioTrack);
  _peerConnection.onAddStream = (stream) {
    _remoteVideoRenderer.srcObject = stream;
  };
}
```

Рисунок 3.12 – Функція встановлення медійного з’єднання

4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

4.1 Проблематика технології

Працюючи над додатком з використанням технології WebRTC можна помітити лаконічну логіку побудови додатку. Вона складається, можна сказати, з однієї функції на клієнті і одній на сервері, яка ініціалізує дзвінок і обробляє оферту відповідно, і включаючи різні методи можна коригувати і покращити її результат на базовому рівні, додавши наприклад шифрування каналу, або даних, тим самим воно гарантовано буде працювати. Безумовно, це добре, і дуже корисно для створення додатків з використанням відеозв'язку в реальному часі.

З іншого боку, наприклад, створюючи цей додаток потрібна була функція, яка виводить статус онлайна у користувача в реальному часі. Це було б легше реалізувати за допомогою Socket.io, який, як на мене також конкурентоспроможна технологія, до якої ще прийдеться повернутися.

Якщо брати до уваги, лише думку розробника, то технологія дуже зручна для побудови кроссплатформеного додатку, який буде використовувати відео- та аудіо- зв'язок, що і є великою перевагою цієї технології, але для реалізації наприклад чату, або обміну статичними медіафайлами краще обрати Socket.io.

Проблеми з проходженням через фаєрвол є дуже неприємною стороною, так як викликає ряд помилок, що призводить до підвисання серверу, так наприклад тестування програми на Android з IOS не дало ніяких результатів через відсутність з'єднання використовуючи локальний сервер на базі Linux.

Великою проблемою є масштабування технології в самому додатку, адже функції, які є у технології обмежені вбудовані функції і для отримання бажаного результату потрібно буде реалізовувати їх з нуля. Це, звичайно, дозволяє назвати технологію гнучкою, але все залежить на практиці від умінь розробника і часу на розробку програми.

У WebRTC, поки що, не буде конкурентів у веб проєктах. Однозначно, ця технологія неперевершена для побудови веб додатків які використовують

браузери, але використовуючи її у Flutter на базі Dart для розробки мобільного додатку, знадобився певний час, щоб прочитати багато інформації і технічної документації стосовно цієї технології і плагіну, який використовував на клієнті. Розробнику, у якого мало часу на вивчення технології не варто додавати її у проєкт. Краще скористатися або “easyRTC”, або “Twilio”.

За що розробниками подобається WebRTC, так це за те, що якість відео та аудіо зв'язку максимально якісна, що особливо стане в нагоді, наприклад для сфери онлайн освіти. WebRTC використовує ефективні кодеки, такі як VP8 для відео та Opus для аудіо. Ці кодеки дозволяють високу ступінь стиснення, при цьому забезпечуючи добру якість відео та аудіо під час передачі даних через мережу.

Автоматичне налаштування параметрів в залежності від обмежень мережі та характеристик пристроїв користувачів допомагає уникнути проблем, пов'язаних із збоєм відтворення або втратою якості в умовах обмеженої пропускної здатності мережі. А RTCP-протокол використовується для збору статистики та метаданих в реальному часі, що дозволяє системі адаптувати параметри передачі для оптимальної якості картинки та звуку.

4.2 Порівняння з іншими технологіями для реалізації зв'язку в мобільних додатках

Існує багато технологій, які можуть бути використані для real-time програмування, які використовують інші інструменти, мають іншу реалізацію і логіку, але дають можливість робити подібні додатки (див. табл. 4.1). Це відкриває широкі можливості для розробників, які обирають різні стеки технологій і з різним рівнем знань - робити дивовижні додатки, які будуть відповідати вимогам проєкту. Кожна технологія, в тому числі, WebRTC має свої плюси і мінуси, про які і йде мова в цьому розділі.

Таблиця 4.1 - Порівняння найпопулярніших аналогічних технологій

Технологія	WebRTC	SIP (Session Initiation Protocol)	H.323	WebSocket
Опис	Взаємодія частин додатку в реальному часі за допомогою веб-браузерів і мобільних додатків	Протокол сигналізації для ініціювання, модифікації та завершення інтерактивних сеансів із залученням відео, голосу, обміну повідомленням и та інших комунікаційних програм і служб	Протокол для передачі аудіо, відео та даних у режимі реального часу через пакетні мережі, такі як мережі IP, Ethernet і ATM	Протокол зв'язку, що забезпечує повнодуплексні канали зв'язку через одне, довговічне, надійне з'єднання
Використання	Використовується веб- та мобільні додатки, спілкування в режимі реального часу в браузерах	VoIP, відеоконференції, миттєві повідомлення, мультимедійні сесії	Відеоконференції, передача голосу через IP (VoIP)	Веб-додатки в реальному часі, інтерактивні сервіси
Протоколи	SRTP, RTP, STUN, TURN, ICE, SDP	SIP, SDP, RTP, RTCP	H.225, H.245, RTP, RTCP	WebSocket, HTTP, HTTPS
Безпека	Захищені протоколи, шифрування (SRTP)	Вразливий до різноманітних атак, потребує додаткових заходів безпеки	Питання безпеки залежить від зовнішніх протоколів	Безпека через HTTPS, WSS (WebSocket Secure)
Порти	Зазвичай використовують порти 80 і 443, можна налаштувати	Використовує різні порти, наприклад 5060, 5061	H.323: 1720, RTP: 5004-5006, RTCP: 5007	Використовує порт 80 (WebSocket) і 443 (WebSocket Secure)

Таблиця 4.1 – Продовження таблиці

Технологія	WebRTC	SIP (Session Initiation Protocol)	H.323	WebSocket
Проблеми	Складний у налаштуванні, можуть виникнути проблеми з проходженням NAT	Покладається на зовнішні протоколи безпеки. Також не для новачків.	Не новий протокол, тому може мати проблеми з сумісністю. З технологією Flutter/Dart складний в налаштуванні за відсутністю документації її	Може мати обмеження у обробці широкомасштабного зв’язку в реальному часі
Мобільність	Чудова підтримка як на веб-, так і на мобільних платформах	Деяка підтримка мобільних пристроїв, але переважно використовується в традиційних системах VoIP	Обмежена підтримка мобільних пристроїв, більш поширена в традиційній телефонії	Найкращий вибір для мобільних платформ

У таблиці (див. табл. 4.1), наведеної вище найпопулярніші технології, які виконують схожі функції, разом із технологією WebRTC. Ці результати сформовані на основі дослідження технічної документації та відгуків розробників.

Наприклад технологія WebRTC має низьку плюсів в простій логіці розробки додатку, так як побудована на основі браузерної взаємодії, але поступається тим, що складна в налаштуванні NAT в самому проєкті.

Наступною технологією є сесійний протокол SIP. Він критично потребує логіки захисту даних, та логіки написання того ж додатку відеоконференцій, але

широко використовується для написання програм, де авторизація не грає велику роль, а є лише ініціалізацією користувача. Але така технологія використовується у сучасній телефонії, і підтримує широкий спектр різних функцій.

H.325 також є технологією для традиційних систем телефонії, та не є новою, тому забуваючи цю технологію вона як слід не реалізується та не взаємодіє з новими технологіями. Але все ж може конкурувати з WebRTC так як реалізувати програму з нею все ще можна.

Веб сокети сильно врізались в розробку веб-додатків, вони дозволяють тримати постійне HTTPS з'єднання між сервером та клієнтом. Не дивно, що ця технологія стала популярною і у сфері мобільних додатків. За допомогою сокетів легко реалізувати, наприклад, чат, і це реально не важко зробити просто прочитавши документацію. З іншого боку реалізувати повноцінний додаток з відеоконференціями без втрати якості аудіо та відосигналу буде значно важче.

Це був приклад великих технологій, але не слід забувати про невеликі, які, або працюють на основі WebRTC, або будь якої іншої системи технічних рішень, але великі недоліки та складнощі, притаманні відповіній технології, в реалізації якої були виправлені іншими розробниками і вийшли у світ під іншою назвою.

Таблиця 4.2 - Порівняння найпопулярніших аналогічних технологій на базі вищезгаданих технологій

Технологія	Gevent	Twilio	OpenTok	easyRTC
Переваги	Висока продуктивність через конкурентність та легко інтегрується у Python.	Обширні можливості для реалізації відеодзвінків та текстової комунікації. Хороша документація та підтримка.	Широкі можливості для реалізації відеодзвінків. Простота інтеграції в додатки.	Відкритий код, що дає можливість налаштовувати і розширювати. Простота використання та навчання.

Таблиця 4.2 – Продовження таблиці

Технологія	Gevent	Twilio	OpenTok	easyRTC
Недоліки	Орієнтований на Python, що може бути обмеженням для проектів на інших мовах.	Комерційний продукт, тому вартість велика, особливо для великих обсягів використання.	Комерційний продукт, можуть бути обмеження для безкоштовного використання.	Може бути менше функціональністю порівняно з іншими бібліотеками.

Зшідно з таблицею (табл. 4.2) “Twillio” та “easyRTC” технології на базі WebRTC, але налаштовані вже перед використанням, тому помилок при інтеграції до проекту не виникають. Легке використання технологій дуже велика перевага на ринку, тому на цьому акцентована увага, адже для аутстаф чи аутсорсингових компаній важлива швидка реалізація ключових функцій в найкоротший проміжок часу.

Також не кожний розробник здатний та готовий фізично вчити декілька складних технологій одночасно для ліпшого результату, але за великий проміжок часу.

Gevent є одним із популярних рішень, але обмежується мовою програмування Python, що не завжди є доцільним, коли проєкт або не використовує Python як основну мову програмування, або не має модулей написаних цією мовою програмування. Gevent реалізовано за допомогою "подієвих розеток" під капотом, що забезпечує спрощений інтерфейс програмування та збільшену конкурентну перевагу. Він заснований на концепції "greenlet" і використовує співпрограми для керування асинхронним виконанням. Gevent дозволяє уникнути взаємоблокувань за допомогою системи введення-виведення та інших не блокуючих системних викликів. Це допомагає зручно створювати масштабовані та ефективні веб-програми в середовищі Python.

OpenTok також є технологією відеозв’язку, яка дозволить розробнику інтегрувати функції для роботи з потоками інформації в реальному часі. Вона має SDK для ін’єкцій потрібних методів у проєкт, але так як проєкт комерційний, тому деякі рішення складних питань або статті про технологію можуть бути також не у вільному доступі.

4.3 Проблеми та перспективи використання WebRTC в майбутньому

Технологія WebRTC на зараз підтримується у всіх популярних браузерах, таких як Opera, Mozilla Firefox, Chrome, Safari та Microsoft Edge, а також у ОС Android та IOS, що робить технологію перспективною в майбутньому, а також розгорнута документація MDN робить привабливою для розробників. Кроссплатформеність є сильною стороною цієї технології, бо з’єднання між браузерами є ефективнішим за VoIP, яка використовується переважно в традиційній телефонії і використовує протокол TCP/IP для передачі медіафайлів в реальному часу.

Але як і у будь якій технології у WebRTC є і проблеми, особливо з витоками IP адрес користувачів, що може нести небезпеку для користувача. Технологія має захищене з’єднання, але при переході на сайт, що використовує WebRTC, IP користувача стає зразу відомим і може бути переглянутий будь ким, бо не шифрується належним чином. Це можна вирішити лише використанням VPN, але натомість, адреса користувача може витекти ще до підключення до віртуальної мережі. Ця проблема відноситься до проблем з браузерами, наприклад Safari стійкий до виток, тому таких неприємностей не відбувається.

Масштабування WebRTC для обробки великої кількості одночасних з'єднань ставить перед розробниками кілька проблем, які необхідно вирішити, щоб забезпечити оптимальну продуктивність і зручність для користувачів. Однією з головних проблем є ефективний розподіл та управління ресурсами для підтримки великої кількості одночасних з'єднань. Зі збільшенням кількості користувачів зростає попит на ресурси сервера, такі як пропускна здатність,

процесор і пам'ять. Горизонтальне масштабування, яке розподіляє навантаження між декількома серверами та екземплярами, є важливим для уникнення вузьких місць та підтримки швидкості реагування. Крім того, необхідні ефективні механізми балансування навантаження, щоб рівномірно розподілити трафік і недопустити, щоб один сервер став точкою відмови. Безперешкодне масштабування також вимагає, щоб процеси сигналізації були оптимізовані і могла ефективно справлятися зі зростанням трафіку.

Ще одним важливим елементом масштабування WebRTC є вирішення проблеми обходу NAT і брандмауерів: WebRTC покладається на однорангові з'єднання для зв'язку в реальному часі, і складність обходу мережевої трансляції може бути перешкодою для прямих з'єднань. Реалізація обходу з використанням ретрансляції навколо сервера NAT(TURN) необхідна для полегшення комунікації, коли прямі однорангове з'єднання неможливі. Однак зі збільшенням кількості користувачів підтримка ефективної та надійної мережевої інфраструктури, включаючи інфраструктуру TURN-серверів, стає значною проблемою.

Таким чином, проблеми, пов'язані з масштабуванням WebRTC для обробки великої кількості одночасних з'єднань, включають управління ресурсами, балансування навантаження вирішення мережевих проблем для забезпечення надійної масштабованої системи зв'язку в реальному часі.

Універсальність WebRTC демонструє його потенціал для розширення використання в різних секторах і переосмислення парадигми комунікації та співпраці. У сфері охорони здоров'я WebRTC використовується для телемедицини, забезпечуючи безпечні консультації в режимі реального часу між лікарями та пацієнтами, дистанційну діагностику та спільне прийняття рішень між медичними працівниками. Здатність технології безпечно передавати високоякісний аудіо- та відеосигнал сприяє покращенню догляду за пацієнтами, особливо в ситуаціях, коли необхідна термінова медична консультація або втручання.

В освітньому секторі WebRTC сприяє інтерактивному та захоплюючому онлайн-навчанню. Віртуальні класи, відеоконференції для дистанційних курсів і спільне навчання стають безпроблемними завдяки WebRTC. Можливості комунікації в режимі реального часу дозволяють викладачам спілкуватися зі студентами по всьому світу і створювати більш інклюзивне і доступне навчальне середовище. Крім того, адаптивність технології покращує платформи електронного навчання, дозволяючи інтегрувати мультимедійний контент та інтерактивні елементи, які збагачують навчальний процес.

WebRTC також відіграє важливу роль в електронній комерції, покращуючи взаємодію з клієнтами та підтримку. Інтегруючі функції спілкування в режимі реального часу, такі як відеодопомога в реальному часі і віртуальний досвід покупок, компанії можуть надавати персоналізовану і негайну допомогу онлайн-покупцям. Це призводить до підвищення рівня задоволеності клієнтів, зменшення кількості покинутих кошиків і більш привабливого досвіду онлайн-покупок. На додаток до цих прикладів, WebRTC продовжує знаходити інноваційні застосування в різних сферах, демонструючи свою універсальність і адаптивність до потреб різних галузей.

ВИСНОВКИ

WebRTC представляє сучасну та перспективну технологію для реалізації функцій зв'язку в мобільних додатках. Реалізація мережевого взаємодії в реальному часі безпосередньо в браузері відкриває нові можливості для взаємодії користувачів.

У цілому, WebRTC виступає як потужний інструмент для реалізації зв'язку в мобільних додатках, але успіх його використання визначатиметься не лише технічними можливостями, але й правильним управлінням безпекою та конфіденційністю в цифровому середовищі.

Висновок 1: Дослідження технології показали, що за її допомогою можливо встановити з’єдняння між клієнтами і обмінюватись даними в реальному часі навіть в рамках мобільного додатку, хоча ця технологія є браузерною і використовує його у фоновому режимі значних проблем у продуктивності не виявлено.

Висновок 2: Архітектура і компоненти технології WebRTC, такі як TURN, STUN, Data channel, ICE - дозволяють не витрачати час на додаткову розробки логіки обходу фаєрволів, і алгоритми дій при неможливості з’єднанні. Окремим плюсом є шиврування даних, яке виконується автоматично при створенні або обробки даних, що добре впливає на безпеку з’єднання.

Висновок 3: Кроссплатформеність технології, велика база документації, стандартизація від W3C і IETFI робить цю технологію максимально комфортною для використання, а кількість плагінів і пакетів для роботи у різних мовах програмування робить цю технологію очевидним варіантом вибору її під технічне завдання проєкту.

Висновок 4: Не для будь-яких задач підійде використання WebRTC. Наприклад для створення тестової програми реалізацій функцій зв’язку була використана технологія на бізі сокетів. Хоча у WebRTC є методи передачі тексту, або даних в режимі онлайн для таких задач як створення чату, сигналізації через сервер, або ж вивести стан одного клієнта для інших вибирати дану технологію

не є доцільним, адже такі функції легше можна реалізувати за допомогою інших технологій.

Отже WebRTC сучасна технологія, яка постійно розвивається і доповнюється як функціями, так і документацією, буде мати і багато програмного забезпечення яке використовує її.

В сфері відеодзвінків, онлайн трансляцій, встановлення потоку постійного обміну медіфайлами, конкурентів знайти їй важко. Але як і будь яка технологія вона використовується і буде використовуватись в симбіозі з іншою і тим самим перекривати мінуси одна одної, що забезпечить правильну роботу програми.

ПЕРЕЛІК ПОСИЛАННЬ

1. Олекс Гуаярд. "WebRTC для Вебу та Інтернету речей" // Packt Publishing, 2017 - 426 с.

2. Алан Б. Джонстон, Даніель С. Бернетт. "WebRTC: API та Протоколи RTCWEB для HTML5 в реальному часі" // Digital Codex LLC, 2012 [Текс] - 438 с.

3. Серхіо Гарсія Мурільо, Хосе М. Мартінес Мартінес. "WebRTC Посібник Інтегратора"// Packt Publishing, 2014 [Текс] - 108с.

4. "WebRTC 1.0: Зв'язок в реальному часі між браузерами - Рекомендація W3C". Консорціум всесвітньої павутини (W3C), 2017. [Електронний ресурс] URL: https://www.w3.org/TR/webrtc/

5. Каллен Дженнінгс, Колін Перкінс, Джонатан Розенберг. "Interactive Connectivity Establishment (ICE): Протокол для пройдення маршрутизаторів мережі (NAT)". Задачевий фонд Інженерії Інтернету (IETF), 2015. [Електронний ресурс] URL: https://tools.ietf.org/html/rfc5245

6. Еміль Івов, Джастін Уберті, Каллен Дженнінгс. "Session Traversal Utilities for NAT (STUN)". Задачевий фонд Інженерії Інтернету (IETF), 2020. [Електронний ресурс] URL: https://tools.ietf.org/html/rfc8489

7. Даніель С. Бернетт, Бернард Абоба, Адам Роуч. "WebRTC 1.0: Зв'язок в реальному часі між браузерами - Кандидат в рекомендації W3C". Консорціум всесвітньої павутини (W3C), 2022. [Електронний ресурс] URL: https://www.w3.org/TR/webrtc/