Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики

Кафедра мультимедійних систем

Курсова робота

освітній ступінь – бакалавр

на тему «Розробка мобільного застосунку для відео-конференцій»

Виконав: студент 3-го року навчання

Спеціальності 122 комп’ютерні науки

Комонов К. М.

Керівник курсової роботи:

магістр к. н., ас. Калітовський Б. В.

_____

“____” _____ 2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри мультимедійних систем,

доцент, к. ф.-м. н.

_____ Жежерун О.П.

„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Комонову К.М. 3-го курсу факультету інформатики

ТЕМА «Розробка мобільного застосунку для відео-конференцій»

Вихідні дані:

- Мобільний застосунок з функцією ведення відеозв’язку

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

1 Аналіз існуючих рішень

2 Дослідження теоретичних відомостей за темою

3 Розробка власного застосунку

Висновки

Список літератури

Додатки (за необхідністю)

Дата видачі „___” _____ 2021 р.

Керівник _____ Завдання отримав _____

Календарний план виконання курсової роботи

Тема: «Розробка мобільного застосунку для відео-конференцій»

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	Листопад 2020р.	
2.	Огляд технічної літератури за темою роботи.	Листопад – грудень 2020р.	
3	Аналіз існуючих рішеннь	Січень 2021р.	
4	Обрання алгоритмів і технологій	Лютий 2021р.	
5	Розробка серверної частини застосунку	Березень-квітень 2021р.	
6	Розробка клієнтської частини застосунку	Березень-квітень 2021р.	
7	Написання текстової частини	Квітень-Травень 2021р.	
8	Перегляд праці науковим керівником	Травень 2021р.	
9	Підготовка до презентації роботи	Травень 2021р.	
10	Презентація курсової роботи	Травень 2021р.	

Студент __Комонов К.М._____ _____

Керівник _Калітовський Б. В.___ _____

“_____”_____ 2021р.

ЗМІСТ

АНОТАЦІЯ

У роботі розглянуто набір технологій необхідних для створення застосунку з підтримкою відеозв’язку. Розглянуто деякі проблеми і ліміти існуючих технологій, а саме: обмеження у можливостях використання багатокористувацького відеозв’язку. Розроблено застосунок, що має функції ведення текстового і відеозв’язку зв’язку. Детально розглянуто організацію з’єднання і протоколи, що використовуються у рішенні WebRTC.

ВСТУП

Впродовж всієї історії цивілізації людство розвивало засоби комунікації. Від усної передачі інформації вісниками до сучасних месенджерів, що дозволяють підтримувати розмову у реальному часі з людиною на іншій стороні планети. Цей розвиток комунікаційних технологій демонструє важливість спілкування для людства. Особливо велике значення технології спілкування у режимі реального часу стали мати у 2020 році – під час пандемії, що змушує людство залишатись вдома заради суспільного добробуту. Саме завдяки розвитку комунікаційних технологій людству вдалось зберегти чимало життів людей, впровадивши дистанційне навчання і роботу. Одним із найзручніших способів комунікації є відеозв’язок.

У даній роботі було описано процес створення мобільного додатку – відеочату. Було розглянуто способи реалізації багатокористувацького відеозв’язку і шляхи їх оптимізації.

Мета: розглянути способи створення застосунків для відеозв’язку

Об’єкт: Створення застосунку для ведення відеозв’язку

Предмет: Існуючі додатки, технології

Методи: Ознайомлення з існуючими технологіями і дослідженнями по темі

Розділ 1 Аналіз існуючих рішень

1.1 Мета використання подібних систем

Використання відеоконференцій є зручним у дистанційному навчанні і роботі. Перевага відеозв'язку над аудіозв'язком полягає у тому, що він дозволяє передавати невербальну інформацію. Особливо велике значення відеоконференції отримали на початку карантину у 2020 році. Перед мільйонами людей постало завдання – перейти на роботу, чи навчання з дому. Цей перехід був би неможливим без сучасних технологій комунікації, однією з найпопулярнішою з яких є технологія відеоконференцій.

Більшість існуючих застосунків для відеозв'язку мають такі функції:

а) Використання одного облікового запису для роботи однієї людини на декількох пристроях;

б) Система контактів;

в) Використання відеоконференцій – можливість бачити і чути своїх співрозмовників;

г) Обмін тестовими, файловими повідомленнями, зображеннями під час відеовиклику;

ґ) Трансляція екрану – передача зображення з свого екрану до співрозмовників

д) Розмиття фонового зображення з камери.

е) Запис екрану – запис відеоконференції з можливістю подальшого перегляду

Всі застосунки, що будуть розглянуті нижче доступні користувачам безкоштовно з певними обмеженнями функціоналу.

1.2 Аналіз існуючих рішень

1.2.1 Skype

Обмеження і ліміти безкоштовної версії:

- Максимальна кількість користувачів у одній відеоконференції – 50; [1]

- Максимальна тривалість дзвінку – 4 години; [1]

- Сума часу проведена у відеоконференціях не повинна перевищувати 10 годин на добу; [1]

- Сума часу проведена у відеоконференціях не повинна перевищувати 100 годин на місяць. [1]

1.2.2 Teams

Обмеження і ліміти:

- Максимальна тривалість відеоконференції – 16 години; [2]

- Максимальна кількість користувачів – 1000; [2]

- Максимальна кількість слухачів (користувачів, чия участь може бути лише пасивною) –100000. [2]

1.2.3 Zoom

Обмеження і ліміти:

- Максимальна тривалість відеоконференції – 30 годин (доступно лише у придбаній версії); [3]

- Максимальна кількість користувачів – 1000. [3]

1.3 Доцільність розробки нового рішення

Сфера розробки застосунків для відеоконференцій наповнена успішними продуктами з майже ідентичними функціями. Не зважаючи на це, всі наведені вище застосунки мають низку обмежень, а саме – максимальна кількість користувачів і максимальна тривалість дзвінка. Для більшості потреб користувачів вистачає наданих функцій не зважаючи на ліміти.

Розробка нового рішення матиме успіх лише якщо матиме унікальні функції або збільшить ліміти існуючих функцій.

1.4 Постановка задачі

Дослідити етапи розробки застосунку для відеоконференцій. Дослідити причини наявності лімітів на кількість користувачів і тривалість дзвінка.

1.5 Висновки до розділу

Визначено основні характеристики притаманні існуючим рішенням організації відеодзвінків. Проаналізувавши функціонал застосунків конкурентів виявлено, що всі вони мають практично ідентичний функціонал і відрізняються лише обмеженнями і ціновими пропозиціями.

2 Теоретичні відомості

2.1 Аналіз особливостей розробки

2.1.1 Необхідні заходи безпеки

Особисті дані користувачів, такі як паролі не повинні зберігатися у відкритому вигляді. Дані медіапотоків повинні передаватися лише у зашифрованому вигляді.

2.1.2 Особливості зберігання даних мобільних додатків

Застосунок для відеоконференцій потребує інтернет з'єднання для того щоб працювати. Об'єм пам'яті доступний для збереження на мобільному пристрої є обмеженим. Також необхідно мінімізувати кількість даних, що отримуватимуться з сервера. Для того що зменшити навантаження на мережу, частину інформації слід зберігати на пристрої а не отримувати від сервера кожен раз [4, с. 37].

Компромісом до цих обмежень є впровадження політики оновлення даних. Для даного застосунку найбільше підходить модель оновлення даних за часовими відмітками. При впровадженні цієї моделі, застосунок робитиме запит про отримання даних до сервера з інформацією про те коли відбулось останнє оновлення відоме застосунку [4, с. 37]. Отримавши подібний запит сервер надішле лише ті зміни, що сталися після часової відмітки.

2.1.3 Вимоги до інтерфейсу

Інтерфейс має бути інтуїтивно зрозумілим користувачеві. Для того щоб користувач не розгубився необхідно використовувати елементи стандартного інтерфейсу притаманного операційній системі. Для Android такими стандартними елементами керування є :

- Кнопка «Назад» (знаходиться внизу ліворуч) (рисунок 1) – повертає користувача на попередній екран всередині додатку. Якщо це стартовий екран, закриває застосунок. Розробник може модифікувати дію цієї кнопки, але у більшості випадків краще не робити цього, це може призвести до невдоволення чи розгубленості користувача.

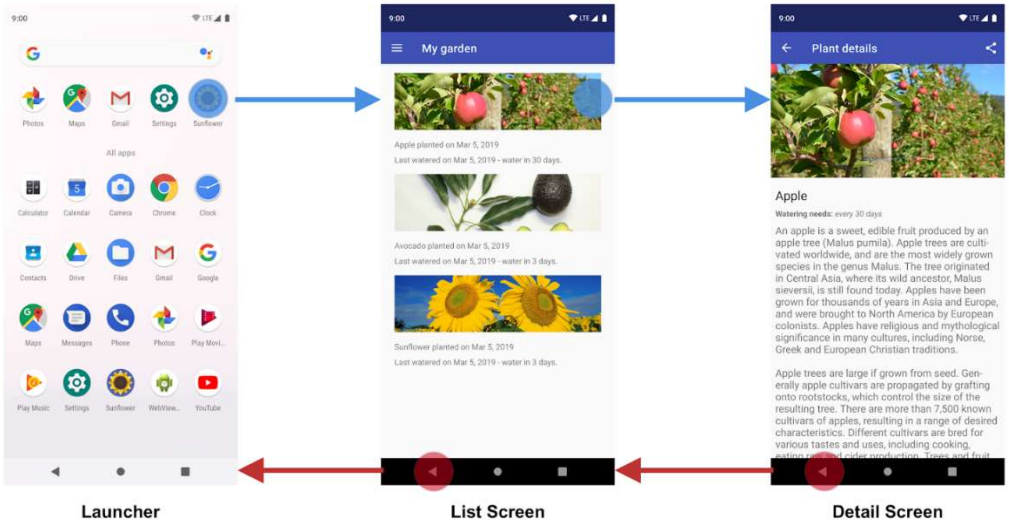

Рисунок 1 - Демонстрація дії кнопки «Назад», демонстрація принципу «Фіксований початок» в операційній системі Android

- Кнопка «Головний екран» (знаходиться внизу у центрі) - повертає користувача на головний екран пристрою (тобто закриває застосунок). Розробник не може змінювати дію цієї кнопки.

Кнопка «Огляд» (знаходиться внизу праворуч) – переводить користувача у режим огляду всіх запущених застосунків. Розробник не може змінювати дію цієї кнопки.

Кнопка «Вгору» (знаходиться вверху ліворуч) – повертає користувача на «батьківський» екран.

При проектуванні застосунку використовується поняття «Навігаційний граф». Навігаційний граф – спосіб представлення екранів програми і зв’язків між ними. Зв’язком між двома екранами називають можливість перейти від одного екрану до іншого. Екрани на навігаційному графі зображують прямокутниками з макетом елементів що знаходяться на даному екрані. Зв’язки між екранами зображують з’єднаннями на графі (зазвичай стрілочками).

Кожен застосунок має фіксований стартовий екран з якого і починається будь яка взаємодія користувача з застосунком (додаток А). Стартовий екран

пов’язаний зв’язками з низкою інших екранів і зазвичай містить в собі кнопки-посилання на різні частини застосунку [5].

Якщо знаходячись на екрані «А» ми натиснули на кнопку і опинились на екрані «Б», то назвемо екран «А» батьківським по відношенню до екрану «Б».

При натисканні кнопки «Вгору» користувач перейде на батьківський екран. Якщо користувач знаходиться на стартовому екрані, йому недоступна кнопка «Вгору» [6].

При натисканні кнопки «Назад» користувач перейде на попередній екран, навіть якщо цей екран не є батьківським [6].

2.1.4 Портативність

Ринок мобільних пристроїв диктує розробникам операційні системи на яких застосунок має працювати. Наразі більшість мобільних пристроїв працюють на операційних системах Android і IOS [7]. Відмова від підтримки хоча б однієї з них є небезпечним з фінансової точки зору і відбувається рідко.

Одним з популярних рішень є використання кросплатформених методів розробки.

2.1.5 Вимоги до роботи з відеопотоком

2.1.5.1 Передача відеопотоку мережею

Використання застосунку для спілкування у режимі реального часу вимагає від застосунку отримувати і передавати дані мережею з найменшою можливою затримкою. Також об’єм відеоданих є великим порівняно з іншими типами інформації (текст, фото, аудіо). Це означає, що передача відеоданих по мережі повинна відбуватись, по найкоротшому можливому шляху.

З вищезгаданого можна зробити висновок, що клієнт-серверна архітектура не дуже підходить для передачі відеопотоку, бо вона вимагає того, щоб весь потік даних передавався через сервер. Наявність проміжного серверу збільшує затримку пересилання даних і є дорогим для власника сервера.

Для вирішення цієї проблеми слід використовувати мережу Peer-to-peer (P2P) (Рівний до рівного) типу. У P2P мережі всі учасники комунікації мають рівні права і можуть звертатись один до одного. Таким чином, зв'язуючись напряму без використання проміжного сервера буде досягнуто максимальну ефективність передачі і отримання потоків даних

Слід зауважити, що даний підхід підходить для зв'язку «один до одного» (тобто між двома користувачами). У роботі буде також розглянуто шляхи отримання зв'язку «багато до багатьох».

2.2 Принципи роботи застосунку для відеоконференцій

2.2.1 Огляд засобів безпеки

Протокол WebRTC передає дані відеопотоку лише зашифрованим шляхом. Конкретніше про це у розділі 2.2.2.2.2 SRTP

2.2.2 Принципи відеозв'язку

Задачу забезпечення роботи відеоконференції можна розбити на наступні підзадачі:

- Організація виклику – домовленість між всіма учасниками про спосіб зв'язку;

- Отримання інформації (у даному випадку дані з мікрофону, дані з камери) з пристрою кожного з учасників, що беруть активну участь у відеоконференції (тобто тих учасників, що передають свої дані іншим учасникам відеоконференції);

- Пересилання відео і аудіо даних між користувачами;

- Відображення відео і аудіо потоків на пристроях всі користувачів.

Виконання більшості з підзадач буде виконано за допомогою набору технологій

Web Real-Time Communication (WebRTC) (Комунікація у режимі реального часу в мережі Інтернет).

2.2.2.1 Організація виклику

2.2.2.1.1 Організація виклику на рівні застосунку

Вище у роботі було запропоновано використовувати P2P технологію для передачі потоків даних між двома учасниками. Використання P2P зв’язку може бути корисним у окремих частинах застосунку, але це не означає, що можна обійтись без централізованої серверної платформи.

Для того, щоб організувати відеодзвінок всі його учасники повинні дізнатись про нього і погодитись приєднатись. Тобто за ініціативою одного користувача, має бути створено повідомлення з пропозицією долучитись до дзвінка і надіслано всім потенційним учасникам. Для реалізації цього необхідно використовувати протокол, що дозволяє незатребувану комунікацію від сервера до клієнта. Таким протоколом є веб-сокетне з’єднання [8].

Встановивши сокетне з’єднання клієнтська частина застосунку зможе отримувати дані за запитом, ініціювати дзвінок, отримувати повідомлення про нові події. Подіями можуть нові повідомлення у чаті, інформація про зміну статусу відеоконференції (повідомлення про створення і закінчення). Сервер відповідно, зможе відповідати на різноманітні запити, а при ініціації нової відеоконференції надсилатиме повідомлення про це до всіх потенційних учасників відеодзвінку якщо вони підтримують сокетний зв’язок у даний момент. Потенційний учасник дзвінку отримає інформацію про ініційовану відеоконференцію як тільки підключиться знову і отримає відповідь на запит про оновлення.

2.2.2.1.2 Організація передачі відео даних засобами WebRTC

2.2.2.1.2.1 Обмін SDP

Процес ініціалізації P2P з’єднання (в даному випадку мається на увазі з’єднання між двома пристроями напряму) називається «Сигналізація».

Учасників P2P з’єднання називають «Пір».

В процесі сигналізації два учасники з’єднання (піри) мають обмінятись наступною інформацією:

Інформація про сесію;

Мережні адреси пірів (IP адреса і порт);

Інформація про типи медіа і кодеки наявні у пірів.

Ця інформація зберігається у форматі Session Description Protocol (SDP) [10].

Крім обміну SDP піри повинні домовитись про те які медіа типи використовувати у даній сесії, так щоб обидва застосунки могли демонструвати надіслану інформацію. Для досягнення цього використовується наступне:

1. Один з пірів генерує SDP бажаної сесії;
2. Застосунок передає цю SDP-пропозицію до іншого піра;
3. Інший пір отримує пропозицію і на основі неї і своїх можливостей генерує SDP-відповідь;
4. Застосунок передає SDP-відповідь до піра ініціатора.

На даному етапі P2P з’єднання ще не встановлено, тому застосунок має взяти на себе обов’язок з передачі SDP. Для цього буде використано сокетне з’єднання , що було описано вище.

2.2.2.1.2.2 Проблеми з NAT

Технологія Network Address Translation (NAT) (Трансляція Адреси Мережі) була розроблена для того щоб боротись з проблемою розподілення IPv4 адрес.

Завдяки цій технології проблему з закінченням вільних IPv4 адрес вдалось відкласти. Кожній локальній мережі з NAT було виділено одну чи декілька «зовнішніх» IP адрес. Всередині ж мережі використовувалися локальні адреси. При проходженні пакетів крізь роутер, локальні адреси замінювались на зовнішню IP адресу, унікальну для всіх пристроїв в мережі.

Крім того, роутер з NAT обмежує трафік зовні (не з локальної мережі) якщо «на нього не чекали». Це робить використання P2P мереж складним а іноді і неможливим [9].

Також проблеми виникають з неналаштованими файрволами.

2.2.2.1.2.3 ICE протокол

Для вирішення проблем з NAT і файрволами було розроблено Interactive Connectivity Establishment (ICE) (Інтерактивний Налагоджувач З'єднання).

Існують деякі способи «обходу» NAT, але для кожної конкретної мережної топології найкращими є різні варіанти. ICE працює наступним чином:

- На кожному з пірів генерується множина всіх можливих підключень доступних для цього піра;
- Піри надсилають один одному по черзі ICE Candidates (кандидатів у найкраще рішення) допоки не буде знайдено найкраще рішення.

В процесі підбору кандидатів піри звертаються до серверів з програмним забезпеченням Session Traversal Utilities for NAT (STUN) (Утиліти Обходу Сесій для Трансляції Адрес Мережі). Ці сервери виконують просту функцію – надіслати піру його зовнішню адресу.

У випадку якщо не вдалось налагодити з'єднання існуючими кандидатами використовується система Traversal Using Relay NAT (TURN) (Обхід Трансляції Адрес Мережі використовуючи Ретрансляцію). Задачею TURN сервера є стати проміжковою ланкою між пірами і ретранслювати потоки даних через себе.

Використання TURN сервера зазвичай є дорогим, бо потребує чимало ресурсів для ретрансляції відео і аудіо даних. Згідно джерела [11] близько 18 відсотків користувачів не можуть встановити з'єднання без використання TURN сервера.

2.2.2.2 Передача даних мережею

2.2.2.2.1 RTP

Для передачі медіа даних у режимі реального часу було розроблено RTP

(Real-time Transport Protocol) (Транспортний протокол реального часу).

Зазвичай RTP працює на основі UDP (User Datagram Protocol) (Протокол датаграм користувача), але можливе і використання TCP (Transmission Control Protocol) (Протокол керування передачею). UDP використовується там де є важливою передача даних у режимі реального часу, а втрата декількох фреймів

не є критичною проблемою. Завдяки гарантії доставки всіх пакетів притаманній протоколу TCP, у випадку пошкодження пакету, його буде надіслано знову, що сповільнить передачу даних. Тому, для передачі відеоданих у режимі реального часу частіше використовують протокол UDP.

RTP містить інформацію про час відправлення, порядковий номер пакету, корисне навантаження (в якому і містяться такі дані як відео чи аудіо фрейми).

RTP працює разом з RTCP (RTP control protocol) (Протокол контролю Транспортного протоколу реального часу) для забезпечення надсилання звітів про якість з'єднання і передачі певної інформації про стан сесії. Наприклад, при відключенні одного з учасників від конференції, по RTCP буде надіслано спеціальний пакет з інформацією про відключення. [12]

2.2.2.2.2 SRTP

Передача даних мережею виконується протоколом The Secure Real-time Transport Protocol (SRTP) (Безпечним Транспортним Протоколом реального часу) який є розширенням RTP. SRTP було створено як фреймворк для шифрування RTP. Метою впровадження протокол є

- Забезпечити конфіденційність корисного навантаження пакетів RTP
- Забезпечити цілісність пакетів і захистити мережу від атаки повторного відтворення

SRTP виконує шифрування лише корисного навантаження.

2.2.2.3 Стискання даних

Відео і аудіо дані, що отримуються мікрофоном і відеокамерою мають занадто великий розмір для того щоб передаватися мережею у режимі реального часу. Для того щоб розв'язати цю проблему, використовується стискання даних.

Стисканням (або компресією) даних називають процес перетворення даних таким чином щоб дані займали менший об'єм але залишали б початкову інформацію. Стиснення повинно бути таким щоб, стисненні дані можна було відновити у початкові дані (або достатньо схожі початковим дані, про що буде вказано нижче). Процес відновлення стиснених даних називається декомпресія.

Алгоритми стискання поділяють на «стиснення із втратами» і «стиснення без втрат».

Стиснення без втрат – алгоритми, які мають властивість повного відтворення даних після процесів компресії і декомпресії.

Стиснення з втратами – алгоритми, що дозволяють деяку відмінність між оригінальними даними і відновленими даними.

Стиснення без втрат використовуються там де важливо відтворити дані повністю, наприклад при стисненні тестових даних.

Стиснення з втратами є компромісом між зменшенням розміру стиснених даних і збереженням якості (близькості даних до оригінальних). Більшість алгоритмів стиснення з втратами використовують особливості того як людина сприймає інформацію. Таким чином стає можливим зберігати частину інформації таким чином, щоб не було помітно різниці [13, с. 4].

Одним з методів стискання відеоданих є запис різниці між кадрами.

2.2.2.4 Огляд способів організації багатокористувацьких викликів

Організація відеозв'язку двох учасників була описана вище. Для того щоб організувати груповий виклик (відеоконференцію) необхідно обрати один з можливих способів топології мережі.

2.2.2.4.1 З'єднання типу Peer to Peer

Ідея з'єднати всіх учасників виклику з усіма. Таким чином кожен учасник мережі передаватиме своєї медіадані кожному з інших учасників мережі.

Проблема з цим підходом у тому, що пристрій повинен підтримувати з'єднання з усіма іншими пристроями, отримувати об'ємні дані з багатьох пристроїв і декодувати декілька аудіо і відеопотоків.

Так, для конференції з N учасників (для прикладу вважатимемо, що всі учасники передають відеодані), кожен учасник повинен підтримувати N – 1 мережних зв'язків і декодувати N – 1 відеопотоків, що є складною обчислювальною задачею для мобільних пристроїв. Так, у операційної системи

Android існують ліміти на те скільки відео одночасно може декодуватися, максимальний бітрейт (швидкість проходження інформації, виміряється у бітах на секунду) і інші обмеження (додаток А). У додатку слід звернути увагу на поле «concurrent-instances» яке визначає максимальну кількість декодерів, що може працювати одночасно і верхню межу поля «bitrate».

2.2.2.4.2 З'єднання використовуючи SFU сервер

Використання Selective Forwarding Unit (SFU) (Блок вибіркового пересилання) – один з варіантів організації відеоконференції. В цьому рішенні використовується сервер (чи більш складна серверна архітектура) з SFU.

Сервер, який отримує медіадані учасників часто називають медіасервером. Але, слід зауважити, що медіасервери можуть бути не лише SFU типу.

Кожен учасник відеозустрічі надсилає свої медіа-дані до сервера і отримує від сервера медіадані всіх інших учасників відеоконференції. Тобто, сервер збирає медіапотоки з усіх користувачів і надсилає їх разом кожному з учасників (рисунок 2).

Цей підхід дозволяє зменшити кількість мережних зв'язків які необхідно підтримувати до 1 (не враховуючи головного сервера застосунку).

Не дивлячись на це покращення, пристрої користувачів все ще повинні декодувати всі відеопотоки.

Враховуючи те, що медіа сервер ніяк не взаємодіє з даними крім пересилання, теоретично можна шифрувати медіадані користувачів при надсиланні на сервер, а розшифровувати при отриманні з сервера. Ця міра додаткового захисту може бути використана, якщо розробник застосунку використовує SFU сервера які обслуговує інша компанія.

Використання SFU сервера не обмежує інтерфейс застосунку з точки зору розміщення відеопотоків на екрані у зручному для себе вигляді.

Також за користувачем залишається можливість відключитись від перегляду відео чи прослуховування звуку якогось конкретного користувача.

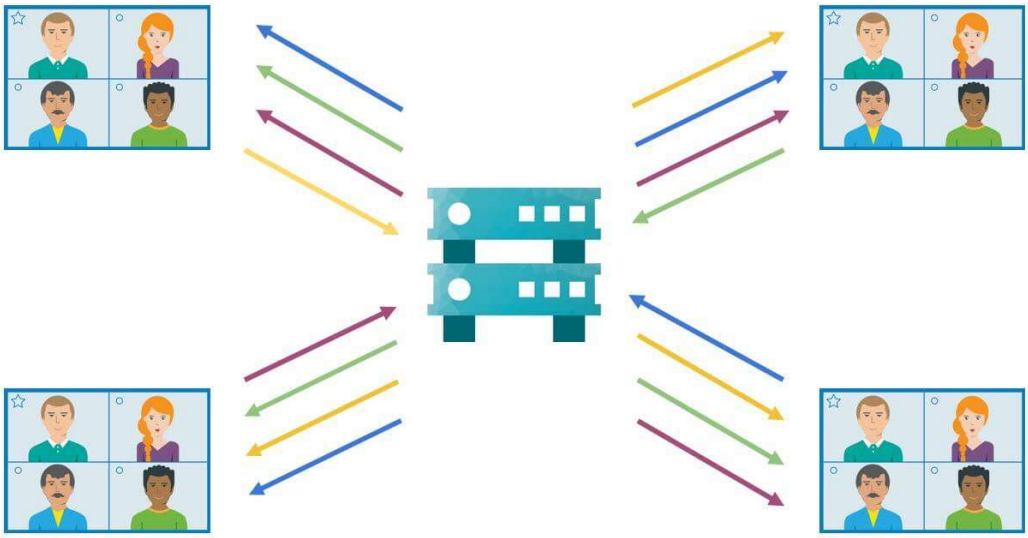

Рисунок 2 - Демонстрація роботи SFU сервера з розподілення потоків даних між учасниками.*[1]

2.2.2.4.3 З’єднання використовуючи MCU сервер

Іншим типом медіасервера є Multipoint Control Unit (MCU) (Багатоточковий Блок Керування).

На відміну від SFU, MCU сервер компонує отримані медіадані в один медіапотік і надсилає його всім учасникам дзвінка. Таким чином, пристрої користувачів отримують один відеопотік і один аудіопотік. (Рисунок 3)

Пристрої користувачів підтримують 1 мережний зв’язок і 1 медіапотік, що знижує навантаження на пристрій. Також більше не потрібно декодувати чимало медіапотоків на кожному пристрої.

Центральний медіасервер може записувати відео зустрічі і зберігати його на сервері чи в хмарному сховищі. Також медіасервер може транслювати відео сторонніми мережами.

Негативними сторонами підходу використання MCU серверів є:

[1] (Зображення розміщене на сайті TRUECONF.COM захищено авторським правом, але згідно правил користування сайтом, може бути використано у навчальних цілях. Витяг з правил користування ресурсом: «The information, artwork, text, video, audio, or pictures (collectively, "Materials") contained on the TRUECONF.COM Web site are protected by copyright laws. You may only access and use the Materials for personal or educational purposes.»)

Велика ціна обслуговування медіасервера (який повинен декодувати кожен з відеопотоків);

Відсутність можливості налаштовувати позицію вікон з відеопотоками;

Медіасервер має доступ до медіапотоку який неможливо зашифрувати

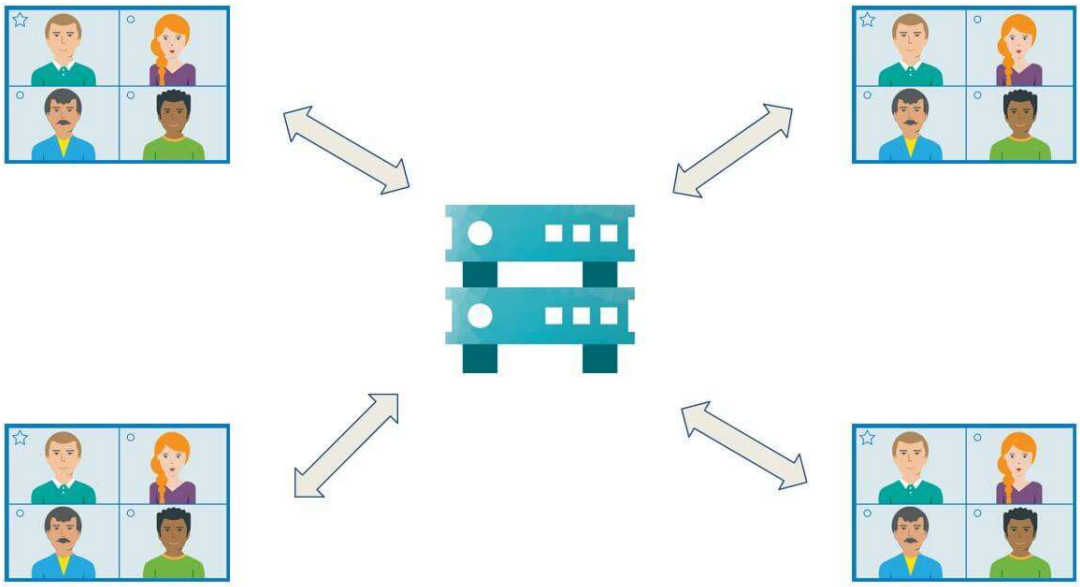

Рисунок 3 - демонстрація роботи MCU сервера. (у з'єднання між застосунком клієнта і сервером передається один медіапотік)[2]

2.2.2.4.4 Порівняння

Отже, ми переглянули 3 підходи до організації відеоконференції.

- P2P - найпростіший спосіб організації відеоконференції, який практично не масштабується і не містить можливостей медійного серверу. Може бути використаний для розмови невеликої кількості учасників;

[2] (Зображення розміщене на сайті TRUECONF.COM захищено авторським правом, але згідно правил користування сайтом, може бути використано у навчальних цілях. Витяг з правил користування ресурсом: «The information, artwork, text, video, audio, or pictures (collectively, "Materials") contained on the TRUECONF.COM Web site are protected by copyright laws. You may only access and use the Materials for personal or educational purposes.»)

- SFU – недороге серверне рішення, залишає гнучкість інтерфейсу застосункам користувачів, але не вирішує проблеми необхідності декодувати декілька медіапотоків. Залишає теоретичну можливість безпечно (з використанням шифрування) використовувати сторонній SFU сервер без довіри до нього;
- MCU – дороге серверне рішення, вирішує проблему з декодуванням багатьох медіапотоків і багатьма мережними з'єднаннями. Найкраще підходить для пристроїв з невеликими системними характеристиками. Не дозволяє зробити гнучкий інтерфейс застосунку. Надає медіасерверу повний доступ до медіа, тому вимагає того, що розробник і користувачі застосунку довіряли стороні що обслуговує медіасервер.

2.3 Огляд використаних технологій

Створення мобільного застосунку складається з:

Створення серверного рішення;

Створення клієнтського рішення (мобільного застосунку).

Також можна виділити етап інтеграції сторонніх сервісів.

2.3.1 Технології використані для створення серверної частини

Задачі, що повинні вирішуватися серверною частиною застосунку такі:

- реалізувати роботу з базою даних (бажано, зручним для розробника шляхом);

- реалізовувати логіку застосунку шляхом створення Application Programming Interface (API) (Прикладного програмного інтерфейсу).

Вимогами і побажаннями до технології є:

- Наявність колекції потрібних бібліотек коду і зручність їх підключення;
- Наявність гарної документації для платформи і бібліотек;

- Підтримка суспільства розробників (підтримка суспільством розробників допомагає покращувати технологію, знаходити проблеми і вирішувати їх, також популярність платформи є запорукою простого вирішення можливих проблем під час розробки, бо існує велика ймовірність того, що хтось хто вже мав подібну проблему ініціював запит про допомогу на спеціальних платформах в інтернеті і отримав відповідь);
- Відсутність проблем з ліцензіями;
- Розвинене середовище розробки.

Вже маючи досвід з платформою NodeJS і мовою програмування Javascript а також переглянувши вимоги і побажання наведені я обрав NodeJS як платформу для розробки серверної частини застосунку.

2.3.1.1 Платформа NodeJS

NodeJS тісно пов’язаний з Node Package Manager (npm) (Пакетний менеджер для NodeJS). Через екосистему npm доступні тисячі бібліотек, чимало з яких пристойної якості. Інтенсивність розробки бібліотек для npm стала своєрідним жартом. Використання npm є зручним і простим.

NodeJS і великі бібліотеки мають гарні документації, чого на жаль не можна сказати про бібліотеки невеликі.

Суспільство розробників створило чимало матеріалу про те як розв’язувати найрізноманітніші проблеми під час розробки. Одним з найпопулярнішим ресурсом з даною тематикою є «stackoverflow.com». Статистика щодо кількості запитів по різним мовам програмування доступна на (рисунку 4). На графіку вісь абсцис займає рік, а вісь ординат – відсоток питанньв на ресурсі, що стосувалися певної мови програмування. Перелік мов програмування розміщено праворуч. Їх відсортовано за значенням на вісі ординат. Як видно на (рисунку 4), Javascript займає друге місце за кількістю запитань у 2021 році. Нажаль, цей рисунок не може слугувати джерелом інформації про популярність NodeJS, але може слугувати джерелом інформації про популярність мови програмування Javascript.

Більшість з бібліотек доступних у npm дозволяють використання себе навіть в комерційних цілях.

Для розробки на платформі NodeJS доступно декілька програм, особисто я використовував Visual Studio Code.

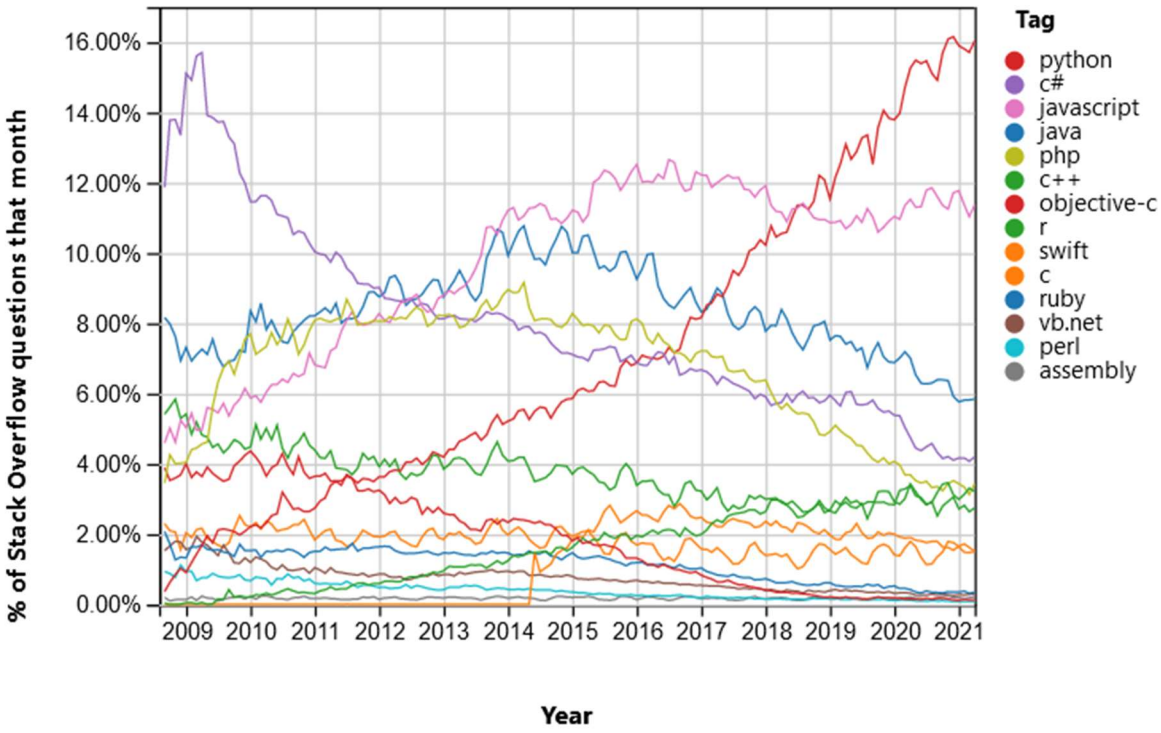

Рисунок 4 – Графік популярності запитань щодо мов програмування на сайті stackoverflow.com у залежності від року

2.3.1.2 Фреймворк Express

Фреймворк Express – простий фреймворк для платформи NodeJS створений для написання серверної частини для веб-застосунків і мобільних застосунків. Головними особливостями фреймворку є ідея проміжних етапів у обробці запиту (Middleware) . Фреймворк є найпопулярнішим серед своїх «конкурентів» і через це співіснує і інтегрується з великою кількістю доступних надбудов.

2.3.2 Технології використані для створення мобільного додатку

Android і IOS ділять між собою практично весь ринок операційних систем для мобільних пристроїв [7]. Застосунок, що якимсь чином працює на різних операційних системах називають «кросплатформеним».

Зазвичай кросплатформеність досягається реалізацією універсальних програмних інтерфейсів які при збірці проекту замінюються на свої точні аналоги реалізовані засобами операційної системи.

Мною було вирішено використовувати кросплатформену технологію для розробки мобільних додатків. Якщо не використовувати подібні технології, то довелось би писати ідентичний код для двох операційних систем.

2.3.2.1 Фреймворк React Native

Одним з лідерів на ринку кросплатформених фреймворків для мобільної розробки є React Native.

React Native оснований на фреймворку для веб-розробки React. Особливостями фреймворку є:

Зручне проектування інтерфейсу (Інтерфейс оновлюється якщо змінюється якась зі змінних компонента);

В основному одностороння передача стану (Мається на увазі, що при оновленні змінних компонента, всі нащадки, що використовують цю змінну оновляться);

React Native спирається на рідне (Native) API операційних систем, а не на WebView (Програмний компонент, що дозволяє запускати веб-програми у собі), що підвищує його продуктивність.

2.3.3 Технології забезпечення зв'язку

2.3.3.1 Сокетне з'єднання

Для використання сокетного з'єднання було використано бібліотеку socket.io.client для мобільного застосунку. Це реалізація веб-сокетів для javascript. Веб-сокети – на відміну від TCP-сокетів працюють на рівні застосунка. Я використав надбудову над TCP-сокетами, а не них через те, що бібліотека socket.io.client містить інтерфейс більш високого рівня, що дозволило мені спростити процес розробки. (А саме інтерфейс взаємодії з «кімнатами» з'єднань, який спростив надсилання одного запиту всім учасникам чату про нове повідомлення).

2.3.3.2 Веб з’єднання

Частина запитів на сервер було виконано за допомогою http запитів. Для цього було використано Fetch API надане React Native.

2.3.3.3 Технологія WebRTC

WebRTC став популярним і потужним механізмом саме завдяки тому, що дав можливість веб-розробникам використовувати можливості комунікації у режимі реального часу у браузері, що до цього було неможливим без використання проміжних серверів.

Існує 2 рівні API для WebRTC:

WebRTC Native API – реалізація на C++

WebRTC API – реалізація на Javascript

Так на сайті з документацією до WebRTC Native API написано:

«The WebRTC Native Code package is meant for browser developers who want to integrate WebRTC. Application developers are encouraged to use the WebRTC API instead.»[14]. (Пакет нативного коду WebRTC розміщений для розробників браузерів які хочуть інтегрувати WebRTC. Розробникам застосунків пропонується використовувати WebRTC API).

Тобто, розробники проекту WebRTC пропонують залишити використання нативної реалізації протоколу розробникам браузерів, а всім іншим розробникам використовувати Web версію API.

Слід зазначити, що існує декілька комерційних сервісів, що пропонують надбудови над WebRTC Native API, але в межах даної роботи було вирішено використовувати Web версію протоколу. Основною причиною є те, що веб версія API є набагато популярнішою, а з чого слідує, що:

- в мережі інтернет можна знайти більше якісних матеріалів для вивчення технології;

- існує більше підтримуваних суспільством розробників бібліотек;

- існує більша ймовірність знайти вирішення проблеми при розробці в інтернеті.

Також як з’ясувалось пізніше, документація WebRTC Native API є менш повною ніж документація WebRTC Web API.

Зазвичай, підтримка спільнотою певної технології покращує її. Чим більше людей буде цікавитись технологією, тим більше проблем (наприклад багів) буде вирішено і тим більше нових можливостей буде впроваджено.

Враховуючи наведене вище, мною було обрано бібліотеку react-native-webrtc, що містить інтерфейс практично ідентичний тому, що надається браузерами (WebRTC WEB API).

2.4 Вибір СКБД

Системи Керування Базами Даних (СКБД) можна поділити на два типи:

реляційні;

нереляційні.

Через те, що Structured query language (SQL) є де-факто стандартом мови запитів для реляційних баз даних, для опису розділення всіх СКБД на реляційні і нереляційні, часто використовують терміни “SQL база даних” і “NoSQL база даних”. Деякі відмінності між реляційними і нереляційними базами даних:

- реляційні бази даних основані на реляційній моделі даних, і відповідно оперують даними у «табличному» вигляді. На відміну від реляційних баз даних, нереляційні використовують декілька різних моделей даних. Серед яких такі: документна модель, модель ключ значення, графова модель;

- Мова запитів реляційних СКБД, як було зазначено вище – SQL. У кожної реляційної СКБД свій діалект SQL, але всі вони підтримують стандарт. (Остання версія ISO/IEC 9075:2016). Кожна нереляційна СКБД створює свою мову запитів.

- NoSQL СКБД більш пристосовані до горизонтального масштабування, що стало одним з ключових аргументів у їх популяризації.

Так, у джерелі [15, с. 7] зазначено: «One of the main characteristics of the NoSQL and NewSQL data stores is their ability to scale horizontally and effectively by adding more servers into the resourcepool. Even though there have been attempts to scale relational databases horizontally, on the contrary, RDBs are designed to scale vertically by means of adding more power to a single existing server» (переклад: «Одна з головних властивостей NoSQL і NewSQL сховищ даних – це їхня здатність масштабуватись горизонтально шляхом додавання серверів до ресурсного пулу. Не зважаючи на те, що були спроби масштабувати реляційні бази даних горизонтально, Реляційні Бази Даних (РБД) спроектовані для вертикального масштабування, шляхом додавання потужності до одного сервера.»).

Для розробки застосунку було обрано NoSQL СКБД з назвою «MongoDB».

Ця СКБД є документо-орієнтованою, тобто використовує як модель даних документи. В даному випадку, під документом мається на увазі модель, яка подібна до об’єкту у ООП (тобто, має поля в яких знаходяться примітивні типи даних, або інші документи).

Ця СКБД не містить схеми даних, що дозволяє швидко змінювати формат документів які зберігаються. Така свобода є небезпечною, тому було прийнято рішення використовувати Object-relational Mapping (ORM), яка дозволяє встановлювати схему даних і дотримуватись її на рівні застосунку (Про що докладніше написано у розділі 3).

MongoDB була обрана через зручність використання документної моделі даних у застосунку, що передає дані мережею у форматі Javascript Object Notation (JSON).

2.5 Висновки до розділу

Створення застосунку для відеоконференцій вимагає використання низки технологій на декількох рівнях розробки. Серед таких рівнів є:

- розробка клієнтського частини (мобільного застосунку);
- розробка серверної частини;

- розробка взаємодії з базою даних (може розглядатись як частина розробки серверної частини).

Крім цього, необхідно розуміти принципи роботи бібліотек і протоколів, які використовуються для того щоб ефективно їх використовувати. Вибір засобів розробки впливає на процес розробки і результат. Тож, не дивлячись на те, що реалізувати застосунок можливо на різних технологіях, слід завжди обирати їх вдумливо і намагаючись спрогнозувати всі етапи розробки.

3 Проектування застосунку

3.1 Технічне завдання

Вимоги до застосунку:

Надавати користувачам інтерфейс керування кімнатами спілкування з іншими користувачами (створення, додавання користувачів, доступ у формі списку);

Надавати користувачам інтерфейс доступу до списку користувачів з якими хотілося б поспілкуватись, або з якими вже спілкувався (список контактів) і його керування;

Надавати користувачам інтерфейс пошуку інших користувачів;

Надавати можливість спілкування у режимі реального часу передачею відео, аудіо даних між двома і більше користувачами.

Побажання до функціоналу застосунку:

- Надавати можливість спілкування у режимі реального часу текстовим шляхом.

3.2 Архітектура додатку

3.2.1 Архітектура серверної частини

Серверна частина застосунку містить два набори програмованих інтерфейсів:

Веб-інтерфейс, що відповідає на http запити. Особливістю є те, що не може ініціювати зв’язок, а може лише відповідати на запити;

Сокетний інтерфейс, що підтримує сокетний зв’язок з застосунками користувачів.

Через веб-інтерфейс відбувається процес аутентифікації, керування контактами:

- додавання користувачів до списку контактів;

- видалення користувачів зі списку контактів;

- пошук контактів.

Сокетний інтерфейс оброблює такі події, реакцію на які потрібно надіслати не тільки відправнику а і іншим користувачам. Наприклад, при надсиланні повідомлення в чат, або при початку дзвінка в чаті, сервер надсилає інформацію про це всім учасника чату які в цей конкретний момент підтримують зв’язок з сервером (рисунок 5).

Ті користувачі, що не були в мережі під час події, отримують оновлення як тільки зайдуть в застосунок.

```
const initCall = async (socket, chat_id) => {
    if(!mongoose.isValidObjectId(chat_id)) return;
    await Chat.findOneAndUpdate({_id:chat_id}, {$set: {isCallActive:true}})
    socket.to(`chat_${chat_id}`).emit('call_started', chat_id)
}
```

Рисунок 5 – Використання механізму кімнат (наданого бібліотекою socket.io) для надсилання всім учасникам чату повідомлення про якісь події в чаті.

3.2.1.1 Аутентифікація

Процес аутентифікації відбувається наступним чином:

Застосунок вмикається і перевіряє в пам’яті наявність спеціального токена.

Якщо токен знайдено, застосунок надсилає його на сервер. Серверна частина програми повинна перевірити, чи дійсний цей токен, тобто чи був він створений сервером у минулому, чи це спроба отримати несанкціонований доступ до застосунку. Рисунки 6, 7.

Функція getUserByToken використовується не тільки при відповіді на запит «/check_token». Також вона використовується для ідентифікації користувача, що надсилає запит на сервер. (userResponse.token.user_id містить об’єкт користувача з бази даних).

```
router.post('/check_token', async (req, res, next) => {
  const { token } = req.body
  const userResponse = await getUserByToken(token)
  return res.send(userResponse.status)
})
```

Рисунок 6 – Процес перевірки токена частина 1

Рисунок 7 – Процес перевірки токена частина 2

Якщо токен не було знайдено, або він виявився неправильним, користувач може увійти в існуючий обліковий запит або створити новий.

Рисунок 8 – Код, що оброблює спробу увійти в обліковий запис

Хешування – це процес незворотного одностороннього перетворення даних. Результат такого перетворення називають хешем, а алгоритм – функцією хешування.

При спробі увійти в обліковий запис, введений користувачем пароль хешується і порівнюється з хешем (рисунок 8), що знаходиться у базі даних. Якщо вони однакові, значить ймовірність того що паролі (той, що був введений користувачем при реєстрації і той що введений при спробі входу в обліковий запис) рівні наближається до 1.

При реєстрації пароль користувача хешується з використанням «солі» - так називають випадкові дані, що додаються до введеного паролю перед хешуванням. При хешуванні пароля з «сіллю» зберігати потрібно не тільки хеш, а і «сіль», для того щоб операцію додавання «солі» можна було повторити з паролем, який вводить користувач при спробі увійти у обліковий запис.

При реєстрації, використовуються валідатори для логіна і пароля. Більше інформації про валідатори знаходиться в розділі (3.2.3.1 Роль ORM у застосунку).

Отримані при валідуванні помилки перетворюються у повідомлення, що показуються на екрані при спробі ввести неправильні дані (рисунок 9).

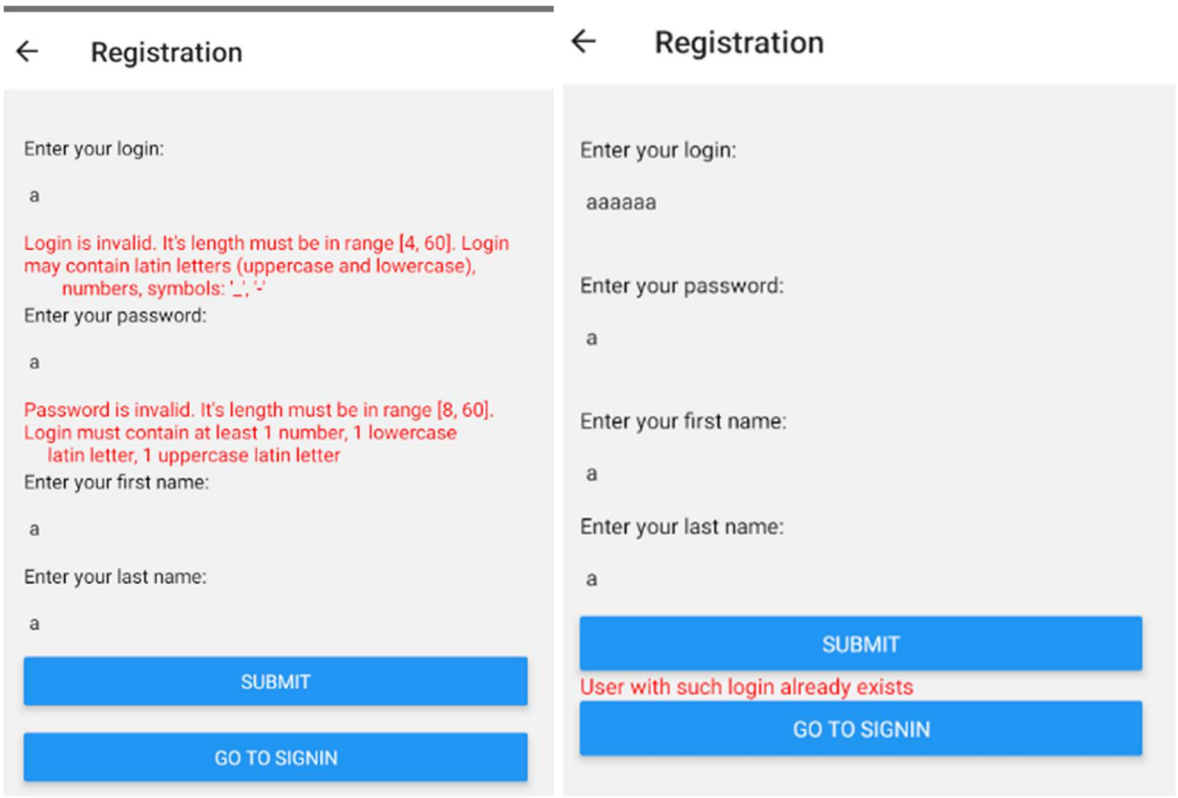

Рисунок 9 – Види помилок при спробі зареєструватися

Для аутентифікації при сокетному з’єднанні використовується трохи інший підхід. Замість токена використовується ідентифікатор сокетного з’єднання, що генерується автоматично бібліотекою для роботи з сокетами. При першому з’єднанні ідентифікатор сокета записується в базу даних. Тож, аналогічно функції «getUserByToken» викликається функція «getUserBySocketId». Ці дві функції практично ідентичні.

3.2.2 Архітектура клієнтської частини

Клієнтська частина застосунку розроблена на фреймворку React Native.

React Native застосунок можна представити як дерево компонентів, що містять стани. При зміні станів, відбувається перемальовка компонентів, що від них залежать.

Для роботи асинхроних функцій і не тільки використовується механізм «хуків».

Хуками називають спеціальні функції, зберігають стан між промальовкою компонентів. Коли необхідно виконати якусь асинхронну дію, слід створити хук, який по завершенню своєї роботи змінить певну змінну стану. Після цього дані на екрані оновляться.

У застосунку використовується бібліотека для навігації, що дозволяє поділити застосунок на авторизовану і неавторизовану частини (рисунок 10).

```
<NavigationContainer>
  {isVerified ? (
    <Main token={token} socket={socket} />
  ) : (
    <Stack.Navigator initialRouteName="SignIn">
      <Stack.Screen name="SignIn">
        {props => <LoginPage {...props} setToken={setAndStoreToken} />}
      </Stack.Screen>
      <Stack.Screen name="Registration">
        {props => <SignupPage {...props} setToken={setAndStoreToken} />}
      </Stack.Screen>
    </Stack.Navigator>
  )}
</NavigationContainer>
```

Рисунок 10 – Розподіл застосунку на авторизовану і неавторизовану частину

У додатку Б надано кореневий компонент авторизованої частини застосунку. У цьому компоненті викликається хук «useApp» який повертає головні змінні застосунку (список чатів, функції для виконання дій, що потребують з’єднання з сервером).

При отриманні оновлень з сервера (наприклад при отриманні повідомлення) викликаються функції слухачі (listeners) і оновлюють відповідні змінні. Код, що оновлює змінні внаслідок повідомлень з сервера надано у додатку В.

3.2.3 Структура бази даних

3.2.3.1 Роль ORM у застосунку

Як було написано вище, мною було використано MongoDB базу даних, яка не містить схем даних, що є небезпечним рівнем свободи.

Для того щоб вбезпечити себе від проблем з некоректним форматом документів в базі даних було використано ORM бібліотеку Mongoose.

Метою створення ORM бібліотек є надання можливості роботи з даними як з об’єктами. Mongoose вимагає визначення схем даних для кожної колекції (аналог таблиць в SQL, але містить документи а не рядки).

Схема дає визначити структуру даних, вказавши:

- назви і типи полів;

- зв’язки між полями – Дозволяє визначити зв’язки аналогом у SQL яких зв’язок Foreign Key (FK) (Зовнішній Ключ) з Primary Key (PK) (Основний Ключ). Дана інформація не зберігається в базі даних MongoDB, а є надбудовою ORM бібліотеки, що дозволяє звертатись до полів документів за якими містяться посилання (FK) на інші документи (рисунок 11). Схема «User» зображена на рисунку 14;

- валідатори для полів, тобто функції, що перевірятимуть коректність даних перед збереженням. Під коректністю маються на увазі як коректність з точки зору БД, так і коректність з точки зору застосунку, так на рисунку 12 зображено валідатор для пароля, що вводиться користувачем. Схема «User» зображена на рисунку 14. Слід зазначити, що перевірка поля написаним нами валідатором відбувається після перевірки поля на відповідність типу зазначеному в схемі.

```
const contacts = await User.findById(req.user._id).populate('contacts')
    .then(found_user => {
        return found_user.contacts.map(u => {
            return fromUserToContact(u, req.user)
        })
    }, err => {
```

Рисунок 11 – Приклад використання ORM бібліотеки для звернення до полів, що посилаються на інші документи. (поле «contacts» схеми «User»)

```
User.path('login').validate(
    (login)=>{
        if(!(login.length >= 4 && login.length <= 60)) return false
        return login.match('[a-zA-Z0-9-_]{4,60}') !== null
    },
    `Login is invalid. It's length must be in range [4, 60]. Login may contain latin letters (uppercase and lowercase),
        numbers, symbols: '_', '-'`,
    'Invalid login'
)
```

Рисунок 12 – Приклад валідатора

функції, що виконуватимуться перед чи після певних подій з циклу життя документа. Їх називають хуками. За допомогою них, можна вказувати певну логіку збереження даних. Рідше – певну логіку застосунку. На рисунку 13 зображено використання хуку перед збереженням документу користувача, що хешує (Спосіб виконання незворотньої операції зі змінення даних, що використовується для збереження паролів користувачів таємницею навіть, при витоку даних) пароль введений користувачем.

```
User.pre('save', async function(next){
    return encryptPassword(this).then(() => next())
})

const encryptPassword = async (user) => {
    return await bcrypt.hash(user.get('password'), salt_rounds).then((hash) => {
        user.set('password', hash)
    })
}
```

Рисунок 13 – Демонстрація використання хуків в ORM

3.2.3.2 ORM схеми у застосунку

У всіх схем є два поля за замовчуванням які не вказуються в схемі:

- _id – унікальний ідентифікатор документа;

_v – версія схеми – для коректної обробки ситуації, коли розробник оновив схему, але документи в базі даних залишились у форматі старої схеми.

3.2.3.2.1 Схема User

Дана схема (рисунок 14) зберігає логін і пароль для аутентифікацією (при першому вході в програму), персональні дані для відображення іншим користувачам, список контактів користувача.

```
const User = new Schema({
    login:{
        type: String
    },
    password: String,
    first_name: String,
    last_name: String,
    contacts: [{
        type: mongoose.Types.ObjectId,
        ref: "users"
    }]
})
```

Рисунок 14 – Схема «User» (Користувач)

3.2.3.2.2 Схема «Token»

Дана схема (рисунок 15) зберігає дані про сесію підключення одного з користувачів.

```
const Token = new Schema({
    user_id: {
        type: mongoose.Types.ObjectId,
        ref: "users"
    },
    token: String,
    online: Boolean,
    socket_id: String
})
```

Рисунок 15 – Схема «Token» (Токен)

3.2.3.2.3 Схема «Chat»

Дана схема (рисунок 16) зберігає дані про користувачів чату і статус дзвінку.

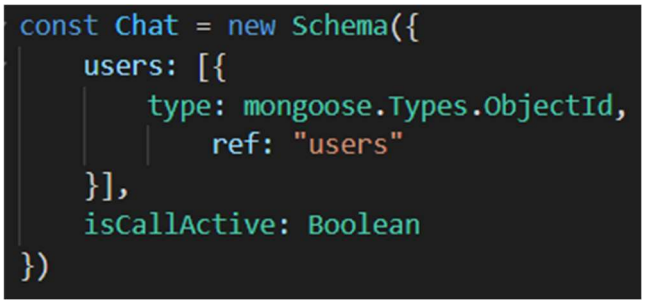

Рисунок 16 – Схема «Chat» (Чат)

3.2.3.2.4 Схема «Message»

Дана схема (рисунок 17) зберігає дані про повідомлення в чаті.

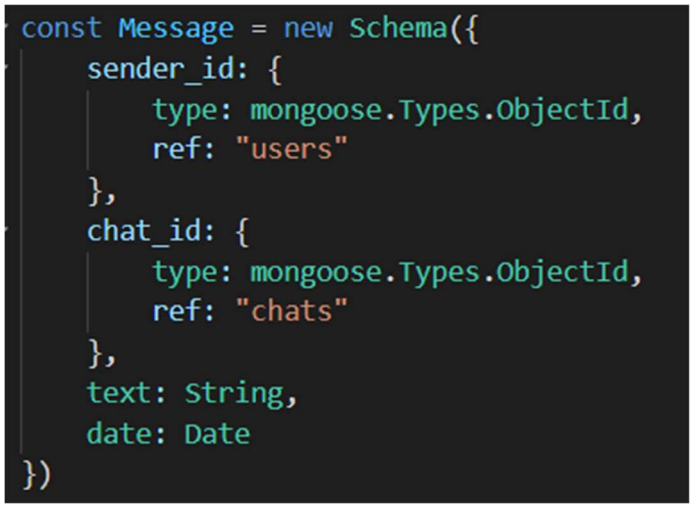

Рисунок 17 – Схема «Message» (Повідомлення)

3.2.3.3 ER модель

Зв’язки між сутностями предметної області зображено на рисунку 18. Дане зображення згенеровано програмою mongoose-erd-generator на основі схем даних наведених вище. Те, що така операція можлива, свідчить про здатність схем Mongoose описувати моделі даних.

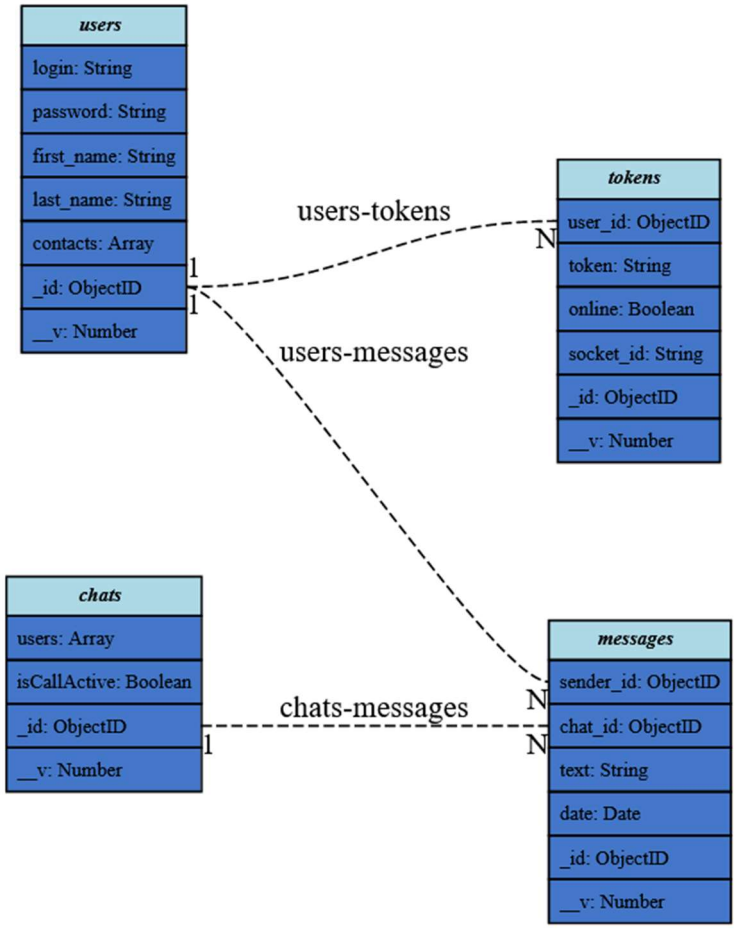

Рисунок 18 – Зв’язок між схемами даних визначених у ORM схемах

3.3 Демонстрація вирішення деяких підзадач

3.3.1 Пошук користувачів

Користувач може шукати інших користувачів за частиною їх імені чи логіну. Пошук відбувається використовуючи обробник регулярних виразів в базі даних MongoDB (Рисунок 19).

```
router.post('/search_user', async function (req, res, next) {
    if (!req.body.query || !isSearchRegexValid(req.body.query)) {
        return res.send({ status: false, message: "Invalid query" })
    }
    const regex = RegExp(req.body.query)
    let users = await User.find({
        $or: [
            { 'login': { $regex: regex } },
            { 'first_name': { $regex: regex } },
            { 'last_name': { $regex: regex } }
        ]
    }, (err) => {
        if (err) throw err;
    }).then(users => {
        return users.map(u => {
            return fromUserToContact(u, req.user)
        })
    })
        .catch(err => {
            console.error(err)
            return res.send({ status: false, message: "unknown error" })
        })

    res.send({
        status: true,
        users: users
    });
});
```

Рисунок 19 – Пошук користувачів

3.5 Висновки до розділу

Було описано декілька механізмів, що використовуються при розробці мобільних застосунків і серверних програм.

Було створено застосунок з можливостями ведення спілкування текстовим шляхом без обмежень на кількість з’єднань і шляхом відеозв’язку між двома учасниками. На рисунку 20 зображено інтерфейс програми під час виклику. На рисунку 21 – інтерфейс чату.

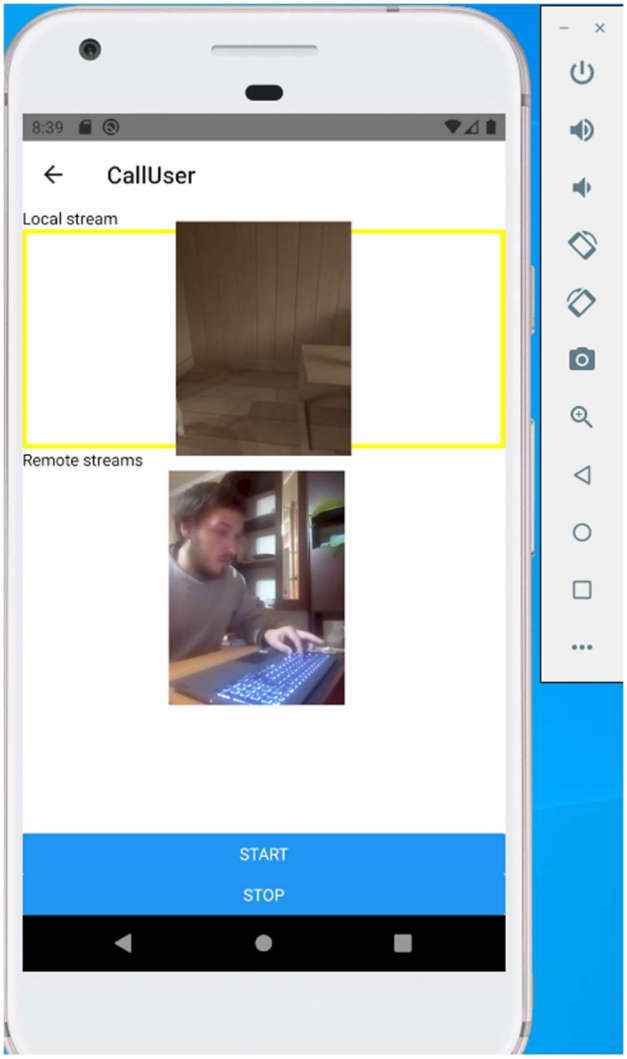

Рисунок 20 – інтерфейс застосунку під час виклику

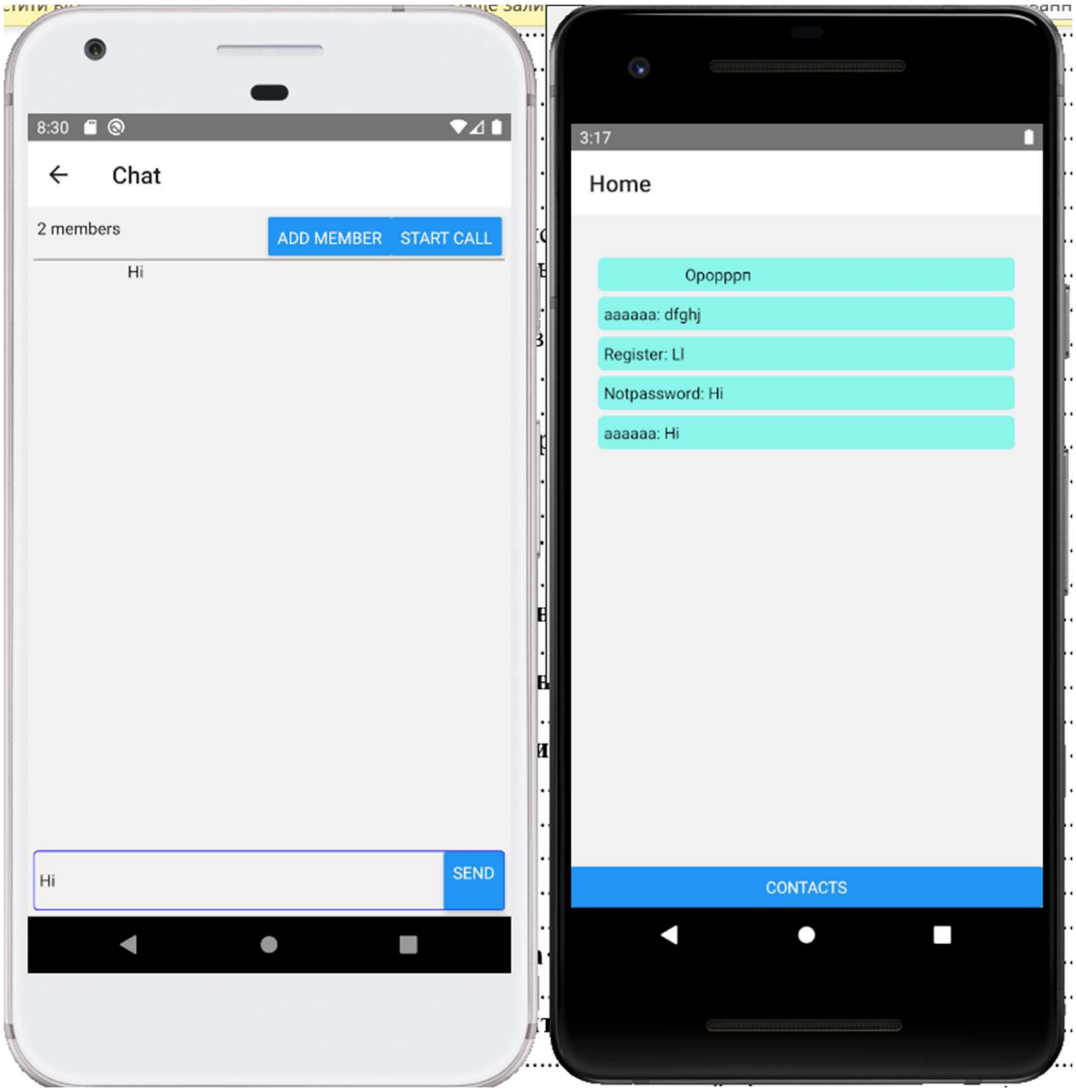

Рисунок 21 – Чат застосунку

Висновки по роботі

У роботі було розглянуто існуючі технології для досягнення відеозв’язку в режимі реального часу. Було детально описано протоколи, що використовуються для організації прямого зв’язку між користувацькими пристроями без необхідності пропускати медіа-дані через сервер у тих випадках коли це можливо. Було розглянуто проблему створення багатокористувацького застосунку для відеозв’язку. Подальший огляд цієї проблеми і реалізація варіантів вирішення є предметом наукового інтересу автора.

Література

1. Правила добросовісного використання. *skype.com*. URL: https://www.skype.com/uk/legal/fair-usage/ (дата звернення: 14.05.2021).
2. Limits and specifications for Microsoft Teams. *docs.microsoft.com*. URL: https://docs.microsoft.com/en-us/microsoftteams/limits-specifications-teams#meetings-and-calls.
3. Pricing. *zoom.us*. URL: https://zoom.us/pricing (дата звернення: 14.05.2021).
4. Nickerson R. C., Mourato-Dussault F. B. Selecting a stored data approach for mobile apps. *Journal of theoretical and applied electronic commerce research*. 2016. Т. 11, № 3. С. 35–50. URL: https://doi.org/10.4067/s0718-18762016000300004 (дата звернення: 15.05.2021). http://doi.org/10.5281/zenodo.3544141
5. Principles of navigation // Android Developers: [Веб-сайт]. URL: https://developer.android.com/guide/navigation/navigation-principles (дата звернення: 14.05.2021).
6. Navigation with Back and Up // Android Developers: [Веб-сайт]. URL: https://stuff.mit.edu/afs/sipb/project/android/docs/design/patterns/navigation.html (дата звернення: 14.05.2021).
7. Mobile Operating System Market Share Worldwide // StatCounter Global Stats: [Веб-сайт]. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (дата звернення: 14.05.2021).
8. Alexey Melnikov, Ian Fette The WebSocket Protocol. 2011. Request for Comments (6455).
9. Ari Keränen, Christer Holmberg, Jonathan Rosenberg Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal 2018. Request for Comments (8445).
10. Henning Schulzrinne, Jonathan Rosenberg An Offer/Answer Model with Session Description Protocol (SDP) 2002. Request for Comments (3264).
11. Medium // What kind of TURN server is being used? | by Philipp Hancke | The Whereby Blog : [Веб-сайт]. URL: https://medium.com/the-making-of-

whereby/what-kind-of-turn-server-is-being-used-d67dbfc2ff5d (дата звернення: 14.05.2021).
12. Henning Schulzrinne, Stephen L. Casner and Ron Frederick, Van Jacobson RTP: A Transport Protocol for Real-Time Applications. 2003. Request for Comments (3450).
13. Marak, Laszlo. On image compression
14. WebRTC // WebRTC Native Code : [Веб-сайт]. URL: https://webrtc.github.io/webrtc-org/native-code/ (дата звернення: 14.05.2021).
15. Grolinger, K.; Higashino, W. A.; Tiwari, A.; Capretz, M. A. M. Data management in cloud environments: NoSQL and NewSQL data stores. (2013).

Додатки

Додаток А (інформативний)

```
−<Decoders>
   <!-- Video Hardware  -->
  −<MediaCodec name="OMX.qcom.video.decoder.avc" type="video/avc">
      <Quirk name="requires-allocate-on-input-ports"/>
      <Quirk name="requires-allocate-on-output-ports"/>
      <Limit name="size" min="64x64" max="4096x2160"/>
      <Limit name="alignment" value="2x2"/>
      <Limit name="block-size" value="16x16"/>
      <Limit name="blocks-per-second" min="1" max="972000"/>
      <Limit name="bitrate" range="1-100000000"/>
      <Feature name="adaptive-playback"/>
      <Limit name="concurrent-instances" max="13"/>
    </MediaCodec>
  −<MediaCodec name="OMX.qcom.video.decoder.avc.secure" type="video/avc">
      <Quirk name="requires-allocate-on-input-ports"/>
      <Quirk name="requires-allocate-on-output-ports"/>
      <Limit name="size" min="64x64" max="1920x1088"/>
      <Limit name="alignment" value="2x2"/>
      <Limit name="block-size" value="16x16"/>
      <Limit name="blocks-per-second" min="1" max="489600"/>
      <Limit name="bitrate" range="1-60000000"/>
      <Feature name="adaptive-playback"/>
      <Feature name="secure-playback" required="true"/>
      <Limit name="concurrent-instances" max="6"/>
    </MediaCodec>
```

Частина файлу /etc/media_codecs.xml що описує існуючі декодери і їх ліміти на мобільному пристрої з операційною системою Android

Додаток Б (інформативний)

```
export default Main = ({token})=>{
    const {chats, newChatUsers, setNewChatUsers, setShouldCreateChat, sendMessage, user_id, addContact} = useApp()
    const {contacts, isLoading, error} = useContacts(token)

    return (
        <NavigationContainer independent >
            <Stack.Navigator initialRouteName="Home">
              <Stack.Screen name="Home">
                {props => <HomePage {...props} chats={chats} token={token} />}
              </Stack.Screen>
              <Stack.Screen name="Contacts">
                {props => <Contacts {...props} contacts={contacts} isLoading={isLoading} error={error} />}
              </Stack.Screen>
              <Stack.Screen name="Search Users">
                {props => <SearchContacts {...props} token={token} />}
              </Stack.Screen>
              <Stack.Screen name="OtherUser">
                {props => <OtherUser {...props} chats={chats} newChatUsers={newChatUsers} addNewUsers={setNewChatUsers}
                createChat={setShouldCreateChat} token={token} />}
              </Stack.Screen>
              <Stack.Screen name="Chat">
                {props => <Chat {...props} send={sendMessage} chats={chats} token={token} />}
              </Stack.Screen>
              <Stack.Screen name="CallUser">
                {props => <CallUser {...props} user_id={user_id}  />}
              </Stack.Screen>
              <Stack.Screen name="NewMemberPicker">
                {props => <NewMemberPicker {...props} chats={chats} addContact={addContact} contacts={contacts} />}
              </Stack.Screen>
            </Stack.Navigator>

        </NavigationContainer>
      )
}
```

Кореневий компонент авторизованої частини застосунку

Додаток В (інформативний)

```
useEffect(() => {
    socket.on('sent_message', async sent_message => {
        console.log(`sent message ${JSON.stringify(sent_message)}`)
        await storeMessage(sent_message)
        setChats(await getFullChats())
    })
    socket.on('new_message', async sent_message => {
        console.log(`new message ${JSON.stringify(sent_message)}`)
        await storeMessage(sent_message)
        setChats(await getFullChats())
    })
    socket.on('created_chat', async chat => {
        // console.log(`Created chat ${JSON.stringify(chat)}`)
        await storeNewChat(chat)
        setChats(await getFullChats())
    })
    socket.on('call_started', async chat_id => {
        console.log(`Started call at chat ${chat_id}`)
        await updateCallStatus(chat_id, true)
        setChats(await getFullChats())
    })
    socket.on('call_ended', async chat_id => {
        console.log(`Ended call at chat ${chat_id}`)
        await updateCallStatus(chat_id, false)
        setChats(await getFullChats())
    })
    socket.on('new_chat_member', async (chat_id, user) => {
        console.log(`New chat member ${chat_id} ${JSON.stringify(user)}`)
        const chats = await getStoredChats()
        const chat_index = chats.findIndex(c => c._id == chat_id)
        if (chat_index == -1) return console.error("Couldn't find chat which has new member")
        chats[chat_index].users_ids.push(user._id)
        await storeUser(user)
        await storeChats(chats)
        setChats(await getFullChats())
    })
}, [])
```

Код, що оновлює змінні застосунку при отриманні оновлень з сервера