**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**
**Факультет кібербезпеки, комп'ютерної та програмної інженерії**
**Кафедра комп'ютеризованих систем управління**

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О.Є.

"_____"_____2022 р.

# ДИПЛОМНИЙ ПРОЄКТ
## (ПОЯСНЮВАЛЬНА ЗАПИСКА)

**ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ**
**"БАКАЛАВР"**

**Тема: «Веб-додаток для проведення відеоконференцій»**

**Виконавець: студент, СП-435, Кармазін Кирило Віталійович**

**Керівник: старший викладач Сябрук Ігор Миколайович**

**Нормоконтролер: Тупота Є.В.**

**Київ 2022**

**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Освітнього ступеня бакалавр

Напрям (спеціальність) 6.050102 "Комп'ютерна інженерія"
(шифр, найменування)

ЗАТВЕРДЖУЮ

Завідувач кафедри

Литвиненко О.Є.

"_____"_____2022 р.

**ЗАВДАННЯ**
**на виконання дипломної роботи (проекту)**
Кармазіна Кирила Віталійовича
(прізвище, ім'я, по батькові)

**1.Тема роботи:** «Веб-додаток для проведення відеоконференцій»
затверджена наказом ректора від "15" лютого 2022 року № 251 /ст.
**2. Термін виконання роботи:** з 16.05.2022 до 19.06.2022
**3.Вихідні дані до проекту (роботи):** веб-додаток для проведення відеоконференцій
**4.Зміст пояснювальної записки (перелік питань, що підлягають розробці):**
1. Аналіз аналогічних додатків.
2. Опис використаних технологій.
3. Розробка веб-додатку.

**5.Перелік обов'язкового графічного матеріалу:**
1. Граф залежностей.
2. Схема взаємодії модулів веб-додатку.
3. Блок-схема алгоритму роботи веб-додатку.
4. Діаграма послідовності.
5. Сторінки веб-додатку для проведення відеоконференцій.

**6. Календарний план**

| № п/п | Етапи виконання дипломного проекту | Термін виконання етапів | Примітка |
|---|---|---|---|
| 1 | Аналіз аналогічних додатків для проведення відеоконференцій | 16.05.22-17.05.22 | |
| 2 | Написання першого розділу | 18.05.22-19.05.22 | |
| 3 | Проведення огляду літературних джерел | 20.05.22-21.05.22 | |
| 4 | Написання другого розділу | 22.05.22-23.05.22 | |
| 5 | Розробка веб-додатку для проведення відеоконференцій | 24.05.22-30.05.22 | |
| 6 | Написання третього розділу | 31.05.22-01.06.22 | |
| 7 | Оформлення пояснювальної записки | 02.06.22-03.06.22 | |
| 8 | Підготовка графічних матеріалів | 04.06.22-05.06.22 | |

**7. Дата видачі завдання** « 16 » травня 2022 р.

Керівник дипломного проекту _____ Сябрук І. М.
(підпис)

Завдання прийняв до виконання _____ Кармазін К.В.
(підпис)

# РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Веб-додаток для проведення відеоконференцій»: 42 сторінки, 14 рисунків, 12 літературних джерел, 1 додаток.

Ключові слова: ВЕБ-ДОДАТОК, ВІДЕОКОНФЕРЕНЦІЯ, HTML, CSS, JAVASCRIPT, NODE.JS, REACT, WEBRTC, SOCKET.IO.

Об’єкт дослідження – веб-додаток для проведення відеоконференцій.

Предмет дослідження – розробка веб-додатку для проведення відеоконференцій.

Мета дослідження – створити веб-додаток для проведення відеоконференцій та навчитися працювати з технологією WEBRTC. Веб-додаток матиме такі функції: попередній перегляд та прослуховування веб-камери, створення нової кімнати, приєднання до вже існуючої кімнати, перегляд та прослуховування веб-камер учасників відеоконференції, вивід унікального ідентифікатора кімнати та поточного часу, можливість використання URL-адреси для доступу до кімнати.

Важливість даної розробки полягає у створенні безкоштовного веб-додатку для проведення відеоконференцій, який буде зручним і інтуїтивно зрозумілим для будь-яких користувачів, незалежно від особистого досвіду, освіти, віку та сфери діяльності.

**ЗМІСТ**

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

*HTML (HyperText Markup Language)* – стандартизована мова розмітки для перегляду веб-сторінок у браузері.

*CSS (Cascading Style Sheets)* – формальна мова опису зовнішнього вигляду веб-сторінок.

*JavaScript* – об'єктно-орієнтована мова програмування, яка застосовується для створення сценаріїв веб-сторінок, щоб надати користувачу можливість взаємодіяти з веб-браузером.

*Node.js* – платформа для створення веб-серверів на мові *JavaScript*.

*React* – *JavaScript*-бібліотека для розробки інтерфейсів користувача.

*WebRTC* – технологія, яка призначена для організації передачі потокових даних між веб-браузерами у режимі реального часу.

*Socket.IO* – *JavaScript*-бібліотека для обміну даними між клієнтом і сервером у режимі реального часу.

*WebSocket* – протокол зв’язку, який забезпечує можливість обміну даними між браузером і сервером в обох напрямках у вигляді «пакетів», без розриву з'єднання і додаткових HTTP-запитів.

# ВСТУП

За останні роки в Україні й в усьому світі різко виросла потреба у спілкуванні на дистанції. Через поширення коронавірусної інфекції COVID-19 було запроваджено режим самоізоляції, який вимагає уникнення фізичних контактів задля запобігання поширенню інфекції. Також під час російського вторгнення в Україну більшість людей була змушена змінити своє місце проживання. Все це призвело до переходу всіх підприємств та навчальних закладів на дистанційну роботу.

По-перше, для цього потрібно було розв’язати проблему групових комунікацій із залученням відео-трансляцій з використанням користувацьких веб-камер, аби якомога ближче наблизити дистанційне спілкування до живого.

По-друге, через появу великої кількості нових користувачів, інтерфейс додатків має бути побудований таким чином, щоб кожному було інтуїтивно зрозуміло як вони працюють. В цій сфері широкої популярності здобули такі веб-додатки, як: *Zoom*, *Google Meet* та *Discord*, що були спрямовані саме на недосвідчених у технічному плані користувачів, які своїм простим та зручним інтерфейсом дозволили досить швидко адаптуватися до дистанційного режиму роботи.

Для досягнення поставленої мети необхідно:

1. Дослідити поняття відеоконференції;
2. Розглянути та проаналізувати аналогічні веб-додатки для проведення відеоконференцій;
3. Дослідити основні технології розробки веб-додатків за допомогою платформ *React* і *Node.js* на мові *JavaScript*;
4. Дослідити технології для реалізації потокової передачі даних;
5. Розробити веб-додаток для проведення відеоконференцій, який дозволятиме користувачам спілкуватися та бачити одне одного незалежно від місця розташування.

# РОЗДІЛ 1

# АНАЛІЗ АНАЛОГІЧНИХ ВЕБ-ДОДАТКІВ

## 1.1. Поняття відеоконференції

Відеоконференція – це тип онлайн-зустрічі, коли двоє або більше людей беруть участь у живому аудіо-візуальному дзвінку (рис. 1.1). Завдяки гарному інтернет-з’єднанню учасники можуть бачити, чути та розмовляти один з одним у режимі реального часу, незалежно від того, де вони знаходяться.

Основна перевага відеоконференцій перед телефонними конференц-дзвінками полягає в тому, що користувачі можуть бачити один одного, що в свою чергу дозволяє їм краще розуміти одне одного.

Рис. 1.1. Система для проведення відеоконференцій

Існує безліч способів проведення відеоконференцій. Користувачі можуть використовувати веб-камери, підключені до ноутбуків, планшетів або настільних комп’ютерів або вбудовані в них. Смартфони та інші підключені мобільні пристрої, які оснащені камерами, також можна використовувати для підключення до відеоконференцій. У таких випадках зазвичай використовується програмна платформа для передачі зв’язку через Інтернет-протоколи.

Відеоконференції також можна використовувати в корпоративній сфері як засіб для проведення навчальних тренінгів для отримання працівниками знань, необхідних для кращого виконання своєї роботи. В академічній сфері також можна використовувати відеоконференції, щоб зв’язати викладача зі студентами, які знаходяться на значній відстані від навчального закладу.

### 1.2. Переваги відеоконференцій

Економія часу та грошей. Відео-зв’язок дозволяє проводити зустрічі з віддаленими співробітниками, партнерами та клієнтами, не залишаючи офіс. Жодних витрат за часом на дорогу до місця зустрічі.

Простота використання. Телефонувати та брати участь у конференціях можна з робочого місця або переговорної кімнати, використовуючи спеціальне обладнання, комп'ютер або смартфон.

Масштабованість. Кількість учасників обмежена лише можливостями певних застосунків та потребами користувача. Можна спілкуватися з одним абонентом, групою учасників або створити вебінар, який побачать тисячі людей.

Реалістичність. Відео-зв'язок може передати набагато більше емоцій, ніж телефонна розмова – це важливо, щоб налагодити контакт із співрозмовником та вирішити ділові питання. Крім того, спілкування у відео-форматі не дозволяє відволікатися та фокусує увагу лише на спілкуванні, як за особистої зустрічі.

Безпека. Сучасні застосунки для відео-конференцій дозволяють працювати в закритих мережах без підключення до інтернету, тому забезпечують максимальну безпеку передачі даних. Під час підключення до відкритих мереж

використовуються протоколи *TLS*, які дозволяють надійно шифрувати інформацію.

### 1.3. Недоліки відеоконференцій

Нестабільне з'єднання. Одним з головних недоліків є технічні труднощі, пов'язані з нестабільністю передачі даних, які можуть виникнути через збій програмного забезпечення, апаратного забезпечення або мережі. Іноді, з'єднанню перешкоджає зміна навколишнього середовища. У деяких випадках відсутність персоналу технічної підтримки створює труднощі для учасників, які не знайомі з технологічними концепціями відеоконференцій.

Технічні проблеми. Відеоконференції передбачають, що користувач повинен мати певні технічні знання, апаратне забезпечення та підключення до мережі.

Відсутність особистого контакту. При спробі розвинути ділові відносини або навіть при співбесіді з кимось на роботу, може бути важливим побачити, як людина справляється з собою у професійних ситуаціях. Тверде рукостискання та доброзичливий зоровий контакт є одними з найкращих способів визначити це, але відеоконференція не дозволяє це зробити. Оскільки технологія продовжує вдосконалюватися, це ймовірно стане меншою проблемою, але не буде заміною для зустрічі з кимось особисто.

### 1.4. Google Meet

*Google Meet* – це додаток для відеоконференцій від компанії *Google* (рис. 1.2). Спочатку він був доступний лише для корпоративних клієнтів, але тепер кожен може використовувати *Google Meet* безкоштовно. *Google Meet* доступний в Інтернеті, а також на телефонах і планшетах для *Android* та *iOS*.

*Google Meet* в першу чергу призначений для проведення відеозустрічей. Однак користувач може ввімкнути камеру та мікрофон окремо, тож він може використовувати їх просто для аудіо-дзвінків.

Окрім звичайних дзвінків також можна створювати тимчасові дзвінки та запрошувати своїх друзів або родину. Вони можуть приєднатися, ввівши код зустрічі, або перейшовши за отриманим посиланням.

Однією з найкращих переваг *Google Meet* є те, що не потрібно встановлювати програмне забезпечення на комп'ютері. Усі учасники розмови (організатор та учасники) повинні просто використовувати сучасний веб-браузер.


Рис. 1.2. Вигляд браузерного додатка *Google Meet* під час зустрічі

В цьому додатку є можливість створювати будь-яку кількість зустрічей і приєднуватися до них, тому ніщо не завадить будь-кому провести другу зустріч, якщо буде досягнено ліміт годин. У безкоштовному плані *Google Meet* підтримує зустрічі тривалістю до однієї години.

На одній зустрічі може бути до 100 учасників. Організатор зустрічей може вимкнути звук інших людей, що буде дуже корисним, якщо зустрічі досягнуть таких великих розмірів розміру.

*Google Meet* робить усе можливе, щоб відфільтрувати будь-який фоновий шум, який не є мовленням. Він також має функцію прямих субтитрів, яка

автоматично підписує те, що говорять люди – це досить корисно з точки зору доступності для людей з вадами слуху.

Також можна ділитися своїм екраном для всіх учасників дзвінка. Це може бути конкретне вікно або весь робочий стіл, і це також працює на мобільному пристрої.

*Google Meet* має високий рівень захисту даних. Компанія *Google* стверджує, що відеоконференції шифруються під час передачі і їх набір заходів безпеки постійно оновлюється для додаткового захисту. Бізнес-користувачі отримують безпеку корпоративного рівня, але для звичайного користувача підходять й прості налаштування конфіденційності.

*Google Meet* безкоштовний для всіх, щоб створювати та приєднуватися до дзвінків. Все, що потрібно це мати обліковий запис *Google* (який є у всіх хто використовуєте електрону пошту *Gmail*), але це також безкоштовно.

Однак деякі функції *Google Meet* доступні лише через платну підписку *G Suite* (рис. 1.3). Це коштує 8 доларів США за одного активного користувача на місяць. *G Suite* орієнтований на корпоративних користувачів і являє собою набір програм від *Google* для співпраці та підвищення продуктивності на роботі.

| | Google Meet<br>Завжди безкоштовно<br>Почати | Google Workspace Essentials<br>8 дол. США за активного користувача на місяць*<br>Спробувати безкоштовно | Google Workspace Enterprise<br>Зв'язатися з відділом продажів |
|---|---|---|---|
| ШИРОКІ МОЖЛИВОСТІ ДЛЯ ЗУСТРІЧЕЙ | | | |
| Тривалість зустрічі 1:1 (максимальна) | 24 години | 24 години | 24 години |
| Тривалість групової зустрічі на 3 і більше учасників (максимальна) | 1 година | 24 години | 24 години |
| Кількість учасників (максимальна) | 100 | 150 | 500 |
| Кількість зустрічей | Необмежена | Необмежена | Необмежена |

Рис. 1.3. Тарифні плани та ціни додатку *Google Meet*

Основні переваги:

1) Зручний інтерфейс, без зайвих функцій.
2) Гарна якість звуку, відео та стабільність з’єднання.

3) Можливість побачити декілька відео-трансляцій у форматі сітки 4 на 4.

4) Високий рівень конфіденційності даних користувача.

5) Інтеграція додатками від *Google* (*Gmail*, *Google Calendar*) та *Microsoft Office*.

Основні недоліки:

1) У безкоштовній версії можна додати лише до 100 учасників до дзвінка.

2) Проблеми з відображенням презентацій *Keynote* або *Microsoft PowerPoint*.

3) Вимогливий до ресурсів комп'ютеру.

4) Може одночасно відображати на екрані не більше 16 учасників.

5) Відносно обмежені можливості в порівнянні з іншими аналогічними додатками.

### 1.5. Zoom

*Zoom* — це платформа для відеоконференцій, яку можна використовувати через настільний комп'ютер або мобільний додаток, і дозволяє користувачам підключатися онлайн для проведення відеоконференцій, вебінарів або чатів (рис. 1.4).

*Zoom* дозволяє користувачам створювати віртуальні конференц-зали та приєднуватися до них, де вони можуть спілкуватися один з одним за допомогою відео та аудіо. Додаткові функції можуть надати учасникам можливість ділитися своїм екраном, ділитися файлами та використовувати текстовий чат у групі наради або приватно з іншими учасниками зустрічі.

Щоб приєднатися до зустрічі *Zoom*, учаснику потрібно мати програму *Zoom* і посилання на зустріч, або ідентифікатор зустрічі та пароль. Хоча обліковий запис *Zoom* не потрібен для доступу до зустрічі, користувачам все одно потрібно буде налаштувати тимчасовий обліковий запис, щоб проводити зустріч.

Організатор може обрати миттєву зустріч або запланувати її на певний час. Це згенерує ідентифікатор і пароль зустрічі, а також URL-адресу зустрічі, якою можна поділитися з людьми, які захочуть взяти участь у конференції.

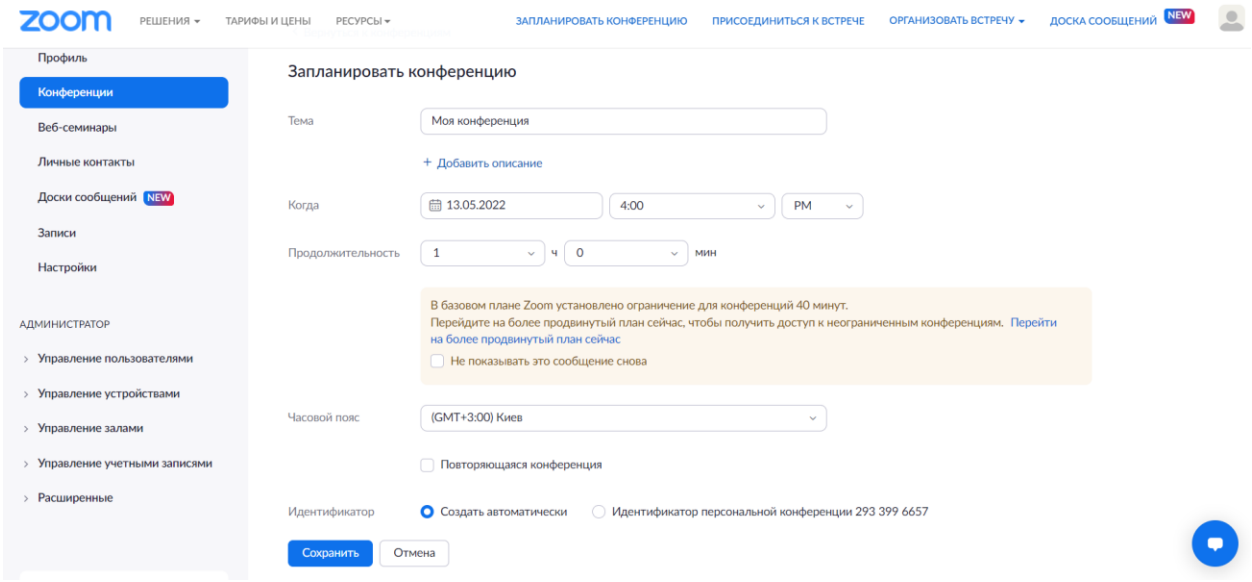


Рис. 1.4. Вікно планування конференції

*Zoom* пропонує чотири тарифи:

1) Безкоштовний: Є можливість проводити необмежену кількість зустрічей. Тривалість групових зустрічей з кількома учасниками обмежена 40 хвилинами і зустрічі не можна записувати.

2) *Zoom Pro*: цей рівень коштує 15 доларів США на місяць. Він дозволяє організаторам створювати особисті ідентифікатори зустрічей для повторюваних зустрічей *Zoom*, а також дозволяє записувати зустріч у хмарі або пристрої, але обмежує тривалість групових зустрічей до 24 години.

3) *Zoom Business*: коштує 20 доларів США на місяць. Він дає змогу брендувати зустрічі *Zoom* за допомогою індивідуальних URL-адрес, а також пропонує стенограми зустрічей *Zoom*, записані в хмарі.

4) *Zoom Enterprise*: коштує 20 доларів США на місяць. Призначений для компаній із понад 1000 співробітників. Він пропонує необмежене хмарне сховище для записів.

Переваги:

1) Є масштабування для малого, середнього та великого бізнесу.

2) Стабільне з’єднання навіть при великій кількості користувачів.

3) Організатор зустрічі має широкий спектр інструментів для керування відео-конференцією.

4) Має вбудовану інтерактивну дошку, якою можуть користуватися всі учасники зустрічі.

Недоліки:

1) Розширені можливості *Zoom* потребують придбання місячної платної підписки.

2) Проблеми з безпекою. До конференції можуть вторгнутися сторонні особи.

3) Складний інтерфейс. Багато користувачів скаржилися, що перед роботою з *Zoom* потрібно витратити певний час на знайомство з додатком.

### 1.6. Discord

*Discord* – це безкоштовний додаток для голосового, відео та текстового спілкування, яким користуються десятки мільйонів людей віком від 13 років, щоб спілкуватися з друзями та учасниками спільнот.

Незважаючи на те, що спочатку *Discord* створювався для ігрового співтовариства, його аудиторія на сьогоднішній день набагато різноманітніша. Щодня люди використовують *Discord* для різних цілей: від обговорення художніх проектів та сімейних поїздок до перевірки домашніх завдань та організації психологічних консультацій. *Discord* відкритий для спільнот різних розмірів, але найбільш широко використовується невеликими активними групами людей для щоденного спілкування.

Основними можливостями даного додатка є:

1) Аудіо та відео-спілкування.

2) Трансляція екрану монітору комп’ютера чи екрану телефона.

3) Тестові чати з різноманітними можливостями (передача файлів, зображень, *emoji*, посилань на інтернет ресурси).

4) Створення серверу. Сервери – це громадські простори *Discord*, створені для певних спільнот і груп (рис. 1.5). Переважна більшість серверів – невеликі і

потрапити до них можна лише за запрошення. Але є й великі сервери, потрапити на які може кожен охочий. Будь-який користувач може безкоштовно створити сервер та запросити на нього своїх друзів.

5) Створення каналів. Кожен сервер в *Discord* має як текстові, так і голосові канали, вони можуть бути присвячені різним темам і можуть мати певні правила.

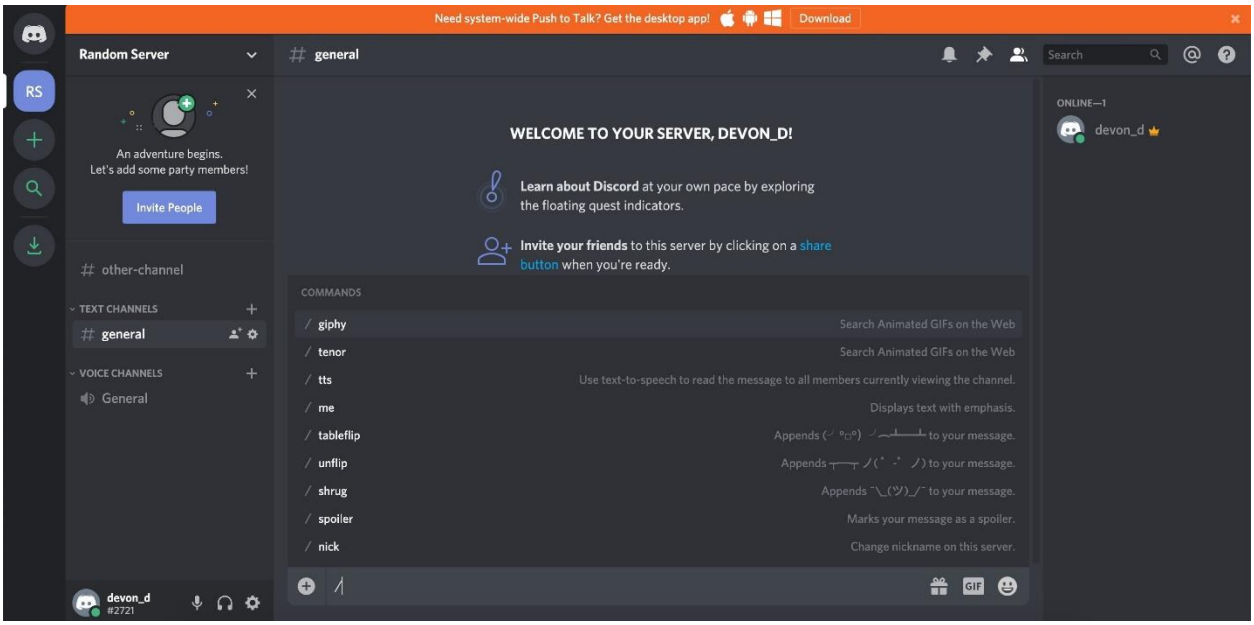


Рис. 1.5. Вікно сервера *Discord*

У текстових каналах користувачі можуть публікувати повідомлення, завантажувати файли та обмінюватись зображеннями з іншими учасниками.

У голосових каналах користувачі можуть спілкуватися за допомогою голосових або відео дзвінків у режимі реального часу, або ділитися своїм екраном.

У користувача також є можливість перейти на платну підписку *Discord Nitro*, яка включає такі функції, як: вища якість відео (1080p і більше), вищий ліміт завантаження файлів, покращена якість прямого ефіру, потокова передача за 10 доларів США на місяць.

Основні переваги:

1) Має сучасний, зручний і простий інтерфейс, який дозволяє користувачу перемикатися між каналами, додавати згадки та хештеги, вставляти майже всі типи медіа безпосередньо в чат.

2) Сервер має велику кількість налаштувань, проте інтерфейс залишається зрозумілим.

3) Безкоштовна версія додатку має більше можливостей порівняно з аналогічними.

4) Текстові чати мають технологію *Drag-and-Drop*, яка надає можливість передачі файлів звичайним перетягуванням мишкою необхідного файлу у вікно чату.

Основні недоліки:

1) Для використання необхідно мати обліковий запис.

2) Демонстрація екрану монітора на слабких пристроях може визвати зависання.

3) В безкоштовній версії максимальна кількість учасників зустрічі дорівнює 10.

4) Відсутня функція запису трансляції навіть у платній версії.

### 1.7. Skype

*Skype* – це постачальник послуг IP-телефонії, який пропонує безкоштовні дзвінки між абонентами та дешеві дзвінки людям, які не користуються послугою *VoIP* (рис. 1.6). Окрім стандартних телефонних дзвінків, *Skype* дозволяє передавати файли, надсилати текстові повідомлення, здійснювати відеочати та проводити відеоконференції. Додаток доступний для настільних комп'ютерів, ноутбуків, планшетів та інших мобільних пристроїв, включаючи мобільні телефони. Ряд компаній, у тому числі *Skype*, виробляють спеціальні телефони для *Skype*.

Окрім того, що користувач може здійснювати безкоштовні дзвінки між комп'ютерами під керуванням *Skype*, за певну плату він також може використовувати *Skype* для підключення до людей через мобільний або стаціонарний телефон. Skype стягує лише кілька доларів на місяць за необмежені дзвінки в США та Канаду і менше ніж 20 доларів за необмежені дзвінки в багато країн світу. Якщо користувач хоче додати Китай до списку країн у які можна

телефонувати, *Skype* стягуватиме з це понад 20 доларів на місяць. У *Skype* також є можливість похвилинно купувати телефонний час.

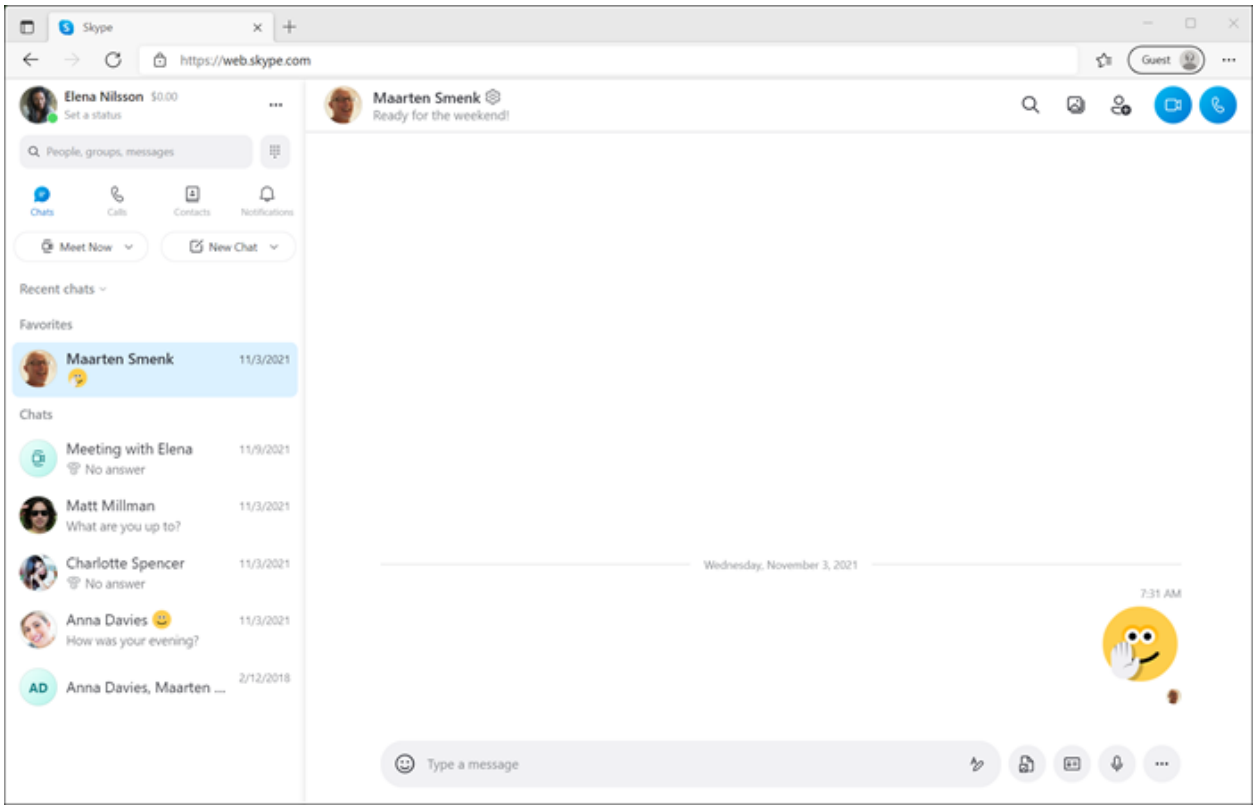


Рис. 1.6. Вікно чату *Skype* в веб-браузері.

*Skype* використовує власну мережу Інтернет-телефонії (*VoIP*), яка називається протоколом *Skype*. Етапи та принципи створення телефонних дзвінків *VoIP* подібні до традиційної цифрової телефонії та включають сигналізацію, налаштування каналу, оцифровування аналогових голосових сигналів та кодування. Замість того, щоб передаватися по мережі з комутацією каналів, цифрова інформація пакетується і передача відбувається у вигляді IP-пакетів по мережі з комутацією пакетів. Вони транспортують медіа-потоки за допомогою спеціальних протоколів доставки медіа, які кодують аудіо та відео за допомогою аудіокодеків та відеокодеків. Існують різні кодеки, які оптимізують медіа-потік на основі вимог програми та пропускної здатності мережі. Деякі реалізації покладаються на вузькосмугову і стиснене мовлення, тоді як інші підтримують високоякісні стереокодеки.

Основні переваги:

1) *Skype* вже інтегрований в операційну систему *Windows 10*.

2) Багато додаткових можливостей, таких як опитування, живі субтитри, створення фотографій, функція запису зустрічі.

3) Комплексна онлайн-довідка, користувач зможе використовувати додаток в повному обсязі за лічені хвилини.

Основні недоліки:

1) Легко вловлює фонові шуми, що в свою чергу погіршує якість дзвінка.

2) Має малий спектр можливостей для керування відеоконференціями.

3) Має проблеми з безпекою, існує багато випадків прослуховування телефонних розмов користувачів.

## 1.8. Висновки до розділу

У першому розділі було досліджено поняття відео-конференції, а також було розглянуто найпопулярніші сучасні додатки для вирішення зазначеної проблеми.

Ці додатки для відео-зустрічей є надійними платформами для телеконференцій, які ідеально підходять для організацій будь-якого розміру. Для команд, які швидко адаптувалися до способу життя віддаленої роботи через COVID-19, ці додатки дозволять без збоїв зв’язуватися зі своїми товаришами по команді. Кожна з цих платформ пропонує безкоштовні відеоконференції, а також ряд інструментів та інтеграцій, щоб користувачі отримували максимальну віддачу від своїх відео-чатів. Деякі з них є галузевими стандартами, та можуть мати високу вартість. Інші набирають популярність і можуть навіть пропонуватися безкоштовно або за низькою ціною. Проте, у кожного було знайдено як переваги, так і недоліки.

# РОЗДІЛ 2

# ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

## 2.1. Технології створення веб-додатку

Архітектура клієнт-сервер – це архітектура веб-додатку, в якій певна кількість клієнтів надсилають *HTTP* запити і отримують відповіді з централізованого серверу, які містять деякий *HTML* код або дані (рис. 2.1). В ідеалі сервер надає клієнтам стандартизований інтерфейс, щоб клієнтам не потрібно було знати про особливості системи, тобто апаратного чи програмного забезпечення, яка надає послугу. Клієнти часто перебувають на робочих станціях або на персональних комп'ютерах, тоді як сервери розташовані в інших місцях мережі, як правило на більш потужних машинах. Ця обчислювальна модель особливо ефективна, коли клієнти і сервер мають різні завдання, які вони регулярно виконують.

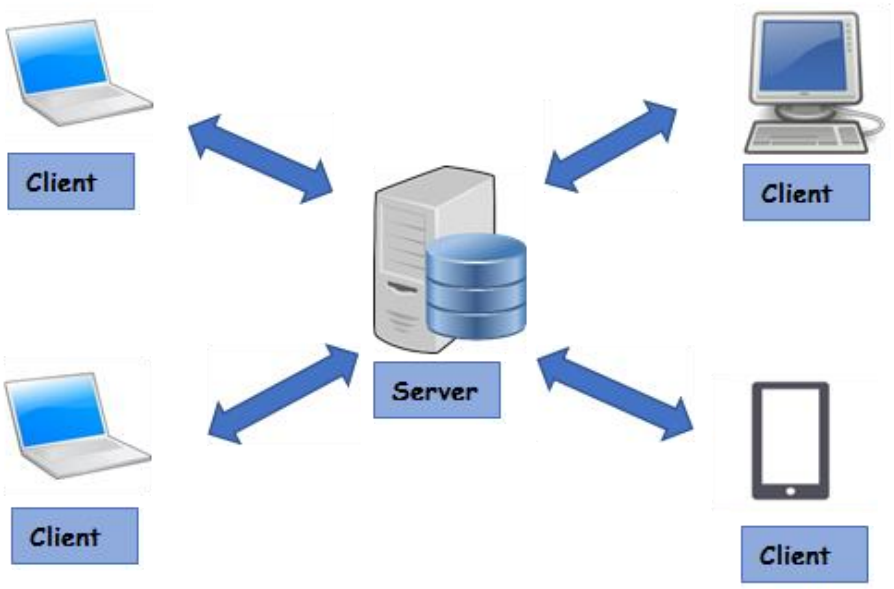


Рис. 2.1. Клієнт-серверна архітектура

Протокол *HTTP*, на якому будуються всі сучасні веб-додатки, заснований на моделі клієнт-сервер. Кожен запит *HTTP* надсилається на сервер, який обробляє запит, а потім надає відповідь (рис. 2.2). Запит може запитувати ресурс, наприклад статичне зображення або сторінку, або запит може запитувати динамічно згенерований вміст, який сервер потім створить на основі запиту.

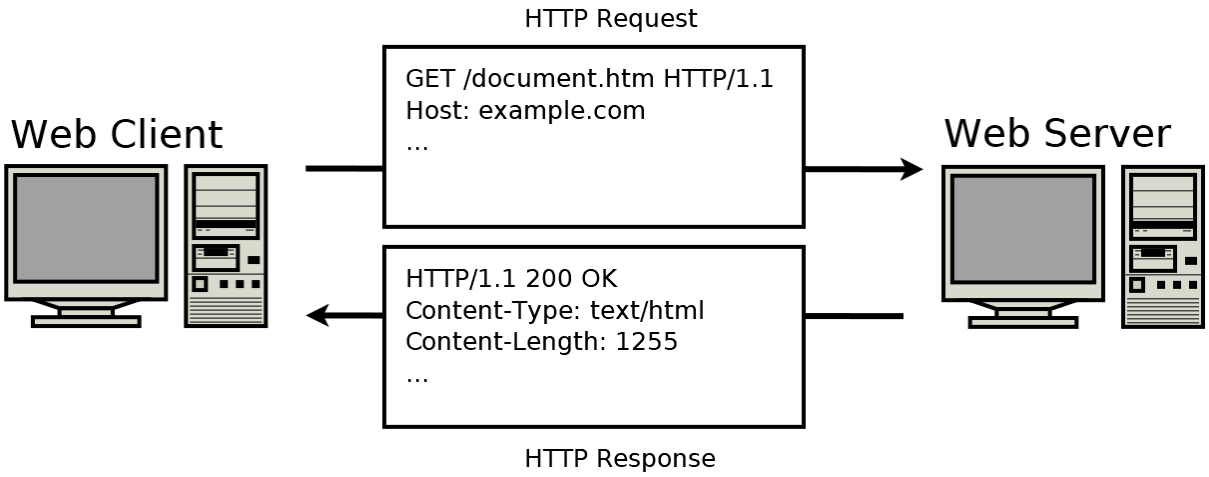


Рис. 2.2. Структура *HTTP* запитів

Оскільки архітектура клієнт-сервер використовує централізований сервер, вона страждає від тих же проблем з надійністю, що й традиційна централізована архітектура: якщо сервер виходить з ладу, доступ до даних припиняється. Однак, оскільки «терміналами» є ПК, будь-які дані завантажені на ПК можна обробляти без доступу до сервера.

## 2.2. React.js

*React.js* – це бібліотека *JavaScript* з відкритим кодом, яка використовується для створення інтерфейсів користувача спеціально для односторінкових програм. Він використовується для обробки шару перегляду для веб- та мобільних додатків. *React* також дозволяє нам створювати багаторазові компоненти інтерфейсу користувача.

*React* дозволяє розробникам створювати великі веб-додатки, які можуть змінювати дані, не перезавантажуючи сторінку. Основна мета *React* — бути швидким, масштабованим і простим. Він працює лише на користувацьких

інтерфейсах програми. Це відповідає поданню в шаблоні *MVC* (Модель-представлення-контролер). Його можна використовувати з комбінацією інших бібліотек або фреймворків *JavaScript*, таких як *Angular JS*.

Властивості *React.js*:

1) Декларативність. *React* значно спрощує створення клієнтських інтерфейсів. Для цього потрібно лише описати, як різні компоненти інтерфейсу будуть виглядати у кожному стані розроблювального додатку і *React* зможе оновити та відрендерити лише необхідні компоненти, коли певні дані змінюються.

2) Компонентно орієнований. *React* дозволяє створювати інкапсульовані компоненти, які керують власним станом. Оскільки логічна частина компонентів створена на основі *JavaScript*, замість шаблонів можна передавати дані в додатку і при цьому зберігати стан окремо від *DOM* (об'єктна модель документа).

3) Продуктивність. *React* використовує віртуальний *DOM* і оновлює лише частини, які змінилися. Отже, це змушує *DOM* працювати швидше. *DOM* виконується в пам'яті, тому можна створювати окремі компоненти, що прискорює роботу *DOM*.

## 2.3. Node.js

*Node.js* – це надзвичайно потужна та найпопулярніша платформа на основі *JavaScript*, яка використовується для розробки онлайн-додатків з чатами, сайтів з потоковим відео, односторінкових додатків (*SPA*) та багатьох інших веб-програм, які часто використовують введення та вивід. Побудований на движку *JavaScript V8 Google Chrome* (рис. 2.3).

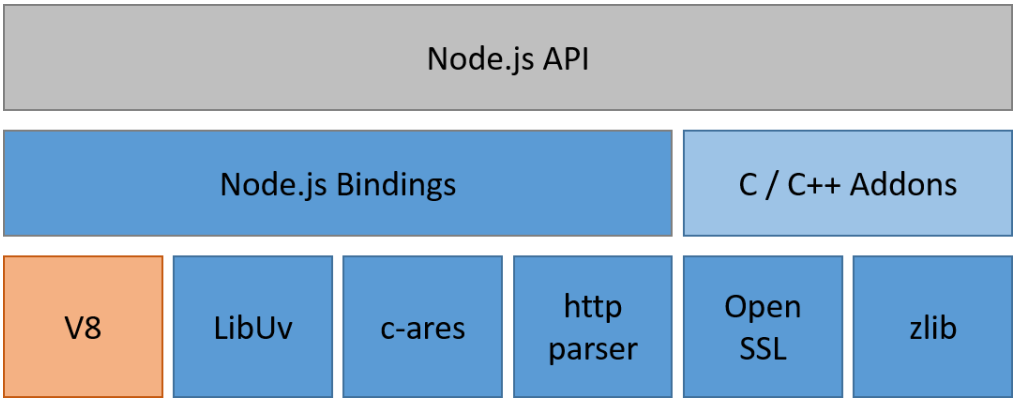

Рис. 2.3. Архітектура ядра *Node.js*

*Node.js* використовує асинхронне програмування. Тобто, усі *API* бібліотеки *Node.js* є асинхронними (не блокуючими), тому сервер на основі *Node.js* не чекає, поки *API* поверне дані. Сервер викликає *API*, і якщо дані не повертаються, сервер переходить до наступного *API*, модуль подій *Node.js* допомагає серверу отримати відповідь від попереднього виклику *API*. Це також допомагає зі пришвидшити роботу *Node.js*.

*Node.js* використовує однопотокову модель із зациклюванням подій. В результаті він може обслуговувати набагато більшу кількість запитів, ніж традиційні сервери, такі як *Apache HTTP Server*.

*Node.js* значно скорочує час обробки під час завантаження аудіо- та відео-файлів. Додатки створені на основі *Node.js* не буферизують дані, а просто виводять ці дані фрагментами.

## 2.4. WebRTC

*WebRTC* (англ. *Web Real-Time Communications*) – це технологія, яка надає можливість голосового, текстового та відео-зв’язку в режимі реальному часі між веб-браузерами та пристроями (рис. 2.4). *WebRTC* надає розробникам програмного забезпечення інтерфейси прикладного програмування (*API*), написані на *JavaScript*. Розробники можуть використовувати ці *API* для створення однорангового зв’язку (*peer-to-peer*) між веб-браузерами та мобільними додатками, не турбуючись про сумісність і підтримку аудіо-, відео- чи текстового вмісту.

*WebRTC* використовує *JavaScript*, *API* та мову розмітки гіпертексту *HTML* для вбудовування комунікаційних технологій у веб-браузери. Він розроблений для того, щоб зробити передачу аудіо, відео та даних між браузерами зручною та легкою у реалізації. *WebRTC* працює з більшістю основних веб-браузерів.

У більшості випадків *WebRTC* з’єднує користувачів, передаючи аудіо, відео та дані в режимі реального часу з пристрою на пристрій за допомогою *P2P*-зв’язку. У ситуаціях, коли користувачі знаходяться в різних мережах Інтернет-

протоколу (*IP*), які мають брандмауери трансляції мережевих адрес (*NAT*), *WebRTC* можна використовувати разом із утилітами обходу сеансів для серверів *NAT* (*STUN*). Це дозволяє перевести дану *IP*-адресу в загальнодоступну Інтернет-адресу, щоб можна було встановити однорангові з'єднання.

Основні переваги *WebRTC*:

1) Підтримується більшістю основних веб-браузерів, включаючи *Google Chrome, Mozilla Firefox* і *Safari*, як для настільних комп'ютерів, так і для *Android*;

2) Використовує сучасні відео- та аудіо-кодеки такі як: *VP8, H.264, Opus* та *G.711*;

3) Має три специфікації шифрування: безпечного протоколу реального часу (*SRTP*), безпечного обміну ключами шифрування (*DTLS*) та безпечної сигналізації. Ці протоколи шифрують дані, які були надіслані через *WebRTC*, захищають ключі шифрування та підключення до веб-сервера.

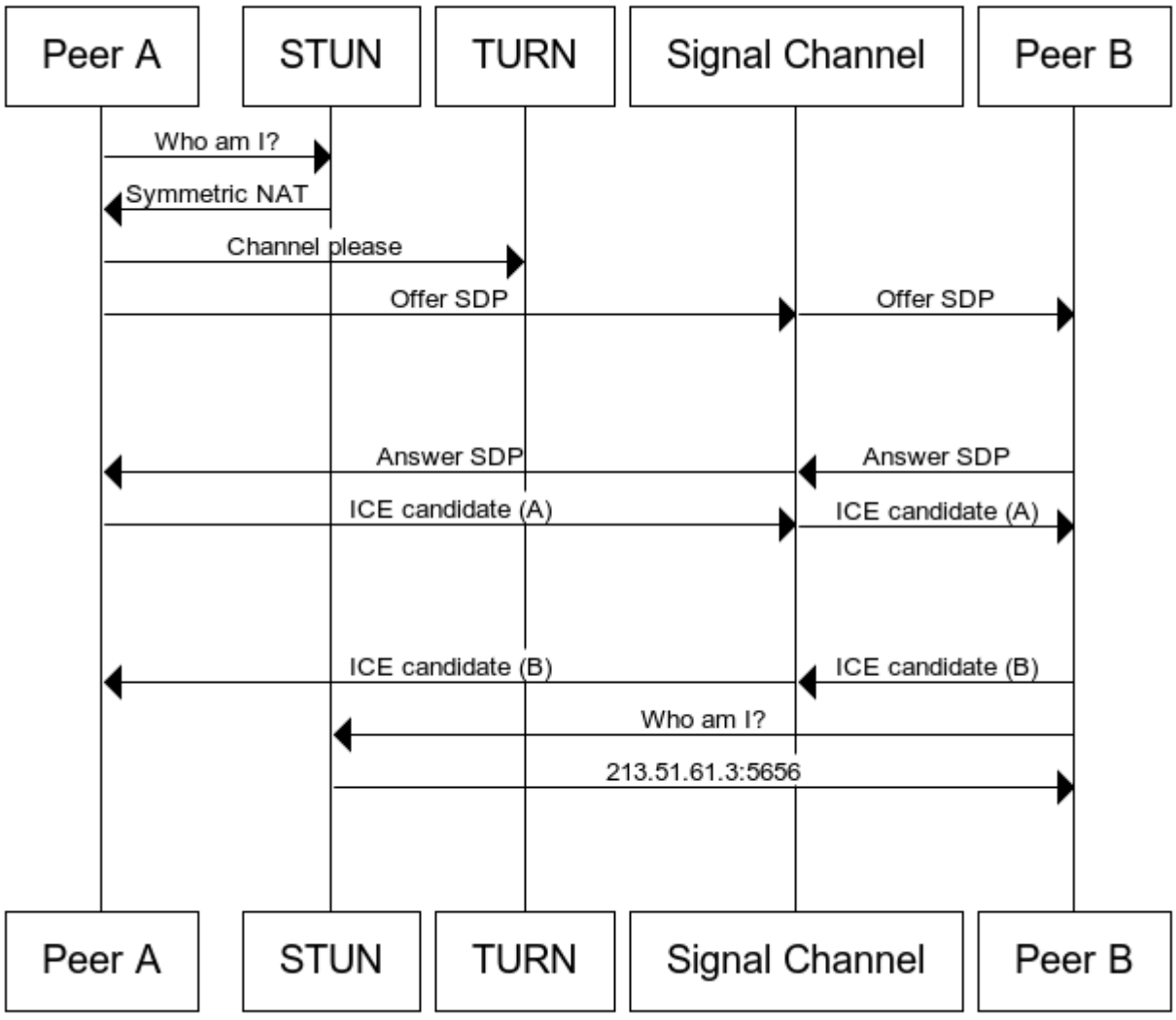

Рис. 2.4. Послідовність дій в технології *WebRTC*

Процес сигналізації працює наступним чином:

- Один з вузлів ініціює виклик;
- Далі перший вузол створює пропозицію за допомогою протоколу опису сеансу (*SDP*) і надсилає її іншому;
- Інший вузол відповідає на пропозицію повідомленням відповіді, яке також містить опис *SDP*.
- Після того, як обидва однорангові вузли встановили свої локальні та віддалені описи сеансів, які включають таку інформацію, як кодеки та метадані браузера, тобто вони знають медіа-можливості, які використовуються для виклику. Однак вони ще не зможуть підключатися та обмінюватися своїми медіа-даними, оскільки *SDP* не знають про такі речі, як зовнішні *NAT* (перетворювач мережевих адрес), *IP*-адреси та способи обробки будь-яких портів. Це досягається за допомогою *Interactive Connectivity Establishment* (*ICE*) кандидатів.

## 2.5. Socket.IO

*Socket.IO* – це бібліотека, яка забезпечує двонаправлений зв’язок між клієнтом і сервером із низькими затримками. Для встановлення з’єднання та обміну даними між клієнтом і сервером *Socket.IO* використовує *Engine.IO*. Це реалізація нижнього рівня, яка використовується під капотом. *Engine.IO* використовується для реалізації сервера, а *Engine.IO-client* — для клієнта.

*Socket.IO* заснований на подіях, тобто клієнт і сервер спілкуються за допомогою подій. І клієнт, і сервер можуть видавати та прослуховувати події, і кожна подія містить тіло, яке в більшості випадків є об’єктом *JSON*.

Для транспортування даних *Socket.IO* використовує протокол *WebSocket*. *Websocket* – це протокол зв'язку, який забезпечує повнодуплексний зв'язок між клієнтом і сервером через одне *TCP*-з'єднання (рис. 2.5). Протокол *Websocket*, який розташований на сьомому рівні моделі *OSI*, забезпечує взаємодію між веб-браузером або іншими клієнтськими додатками та веб-сервером із меншими

витратами, ніж в напівдуплексних протоколах зв’язку, таких як опитування *HTTP*, що сприяє передачі даних у реальному часі між ними.

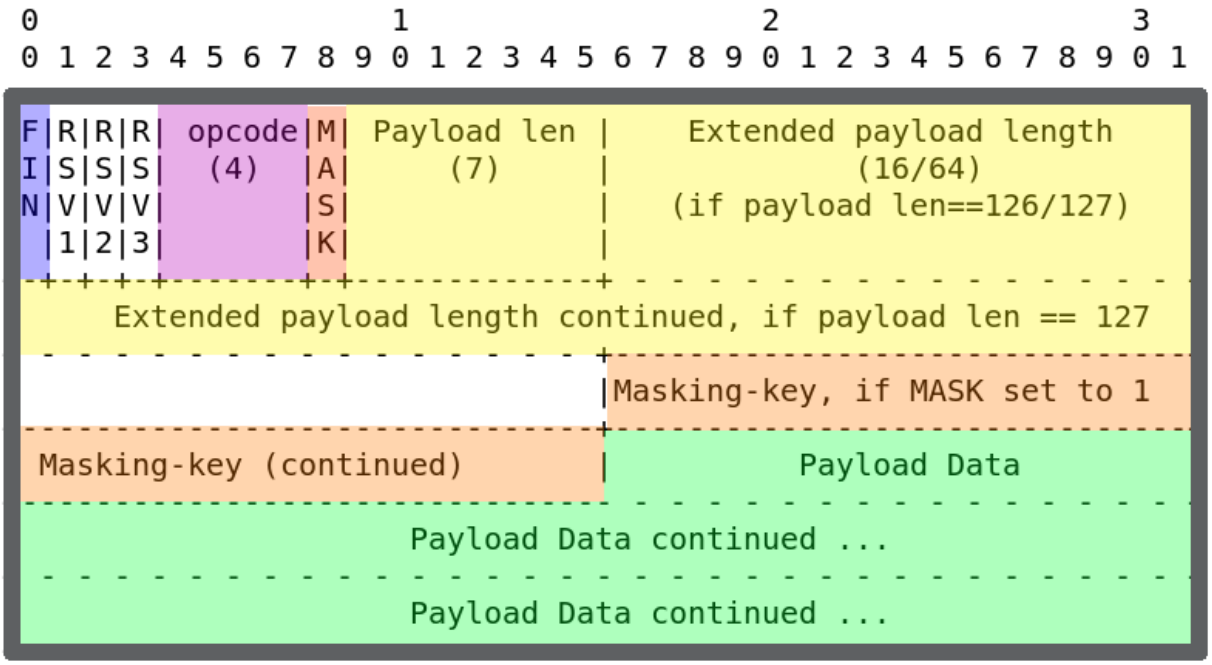


Рис. 2.5. Формат кадру даних *WebSocket*

У протоколі *WebSocket* дані передаються через серію кадрів даних. Щоб уникнути посередництва в мережі, або проблем із безпекою, клієнт повинен замаскувати всі кадри, які він надсилає на сервер. Після отримання кадру даних без маски сервер повинен негайно закрити з'єднання.

## 2.6. Висновки до розділу

У другому розділі було розглянуто всі необхідні технології та бібліотеки для вирішення зазначеної проблеми.

По-перше, було детально досліджено архітектуру клієнт-сервер, яка буде лежати в основі майбутнього веб-додатку. Для реалізації клієнтської частини було обрано *JavaScript* бібліотеку *React JS*, а для серверної частини – *Node.js*.

По-друге, для встановлення зв’язку між клієнтом і сервером використовуватиметься бібліотека *Socket.IO*. Вона дозволить спростити роботу зі звичайними веб-сокетами.

По-третє, реалізовувати функцію передачі аудіо- і відео-даних буде технологія *WebRTC*. Дана технологія базується на архітектурі *Peer-to-Peer*, що дозволяє здійснювати потокову передачу даних. Також вона підтримує аудіо- і відео-кодеків, які використовуються у сучасних веб-браузерах.

# РОЗДІЛ 3

# СТВОРЕННЯ ВЕБ-ДОДАТКУ

## 3.1. Структура веб-додатку

Для розробки веб-додатку було використано наступні технології:

– редактор коду *Visual Studio Code*;
– веб-браузер *Google Chrome*;
– мова розмітки веб-сторінок *HTML*;
– мова опису зовнішнього вигляду веб-сторінок *CSS*;
– мова програмування *JavaScript*;
– бібліотека *React.js*;
– платформа *Node.js*;
– технологія *WebRTC*;
– бібліотеки *Socket.IO* і *Socket.IO-client*.
– бібліотека *Simple-peer*.

Структура клієнтської частини веб-додатку:

– папка *node_modules*;
– папка *public*;
– *Button.jsx*;
– *Camera.jsx*;
– *Clock.jsx*;
– *Input.jsx*;
– *CreateRoom.js*;
– *Room.js*;
– *main.module.css*;
– *conv.module.css*;

– *App.js*;

– *index.js*.

Папка *node_modules* містить всі бібліотеки необхідні для роботи веб-додатку. Папка *public* містить зображення іконки сайту, файл *index.html* в якому вказуються метадані, файл *manifest.json* містить інформацію про іконки, колірну тему, орієнтацію екрана, початковий *URL* тощо.

Файли *Button.jsx, Camera.jsx, Clock.jsx* і *Input.jsx* відповідають за зовнішній вигляд та функцію наступних компонентів: кнопка, камера користувача, поточний час, поле вводу інформації.

*CreateRoom.js* і *Room.js* – містять скрипти (сценарії) веб-сторінок, а також їхня розмітка написана мовою *HTML*.

Файли *main.module.css,* conv.module.css містять набір параметрів, що керують зовнішнім виглядом та положенням елементів веб-сторінок *CreateRoom* та *Room*.

Файли *App.js* і *index.js* містять налаштування маршрутизації веб-додатку.

Структура серверної частини веб-додатку:

– папка *node_modules*;

– *server.js*;

– *package.json*.

Файл *server.js* – файл налаштування та сценаріїв веб-сервера. В даному файлі задається *IP*-адреса, порт та спеціальні функції обробки вхідних *HTTP*-запитів від клієнта.

*Package.json* є файлом маніфестом всього веб-додатку. Він містить таку інформацію як назву, версію, залежності, скрипти запуску додатку тощо.

### 3.2. Реалізація з’єднання між клієнтом і сервером

Щоб реалізувати з’єднання між клієнтом і сервером, спершу необхідно створити *HTTP*-сервер, що прийматиме запити та створюватиме відповіді на них.

*require('dotenv').config()*

```
const express = require('express')
const http = require('http')
const app = express()
const server = http.createServer(app)
const socket = require('socket.io')
const io = socket(server)
```

Для створення даного сервера було використано веб-фреймворк *Express.js*, який використовує вже вбудований в *Node.js* модуль *HTTP*. Також поверх отриманого *HTTP*-сервера, створюється інший сервер за допомогою *Socket.IO*. Цей сервер буде відповідати за з'єднання з клієнтом по протоколу зв'язку WebSocket.

Щоб отримати інформацію про *IP*-адресу та порт, імпортуємо модуль *dotenv*.

Після цього використовуємо метод *server.listen*, котрий змушує сервер прослуховувати заданий порт. Після запуску даного методу сервер буде готовий приймати та відповідати на запити.

```
server.listen(process.env.PORT || 8000, () =>
  console.log('Server is running on port 8000')
)
```

Наступним кроком клієнт приєднується до веб-серверу. Для цього використовується функція *io.connect*, в якій вказується *IP*-адреса і порт веб-сервера.

```
const socketRef = useRef()
socketRef.current = io.connect('/')
```

Після з'єднання з веб-сервером, клієнт підключає медіа пристрої (камера, мікрофон). За допомогою функції, браузерного *API*, *navigator.mediaDevices.getUserMedia* отримуємо потік аудіо- і відео-даних користувача. Далі надсилаємо до веб-сервера подію *join room*, яка містить змінну *roomID* (ідентифікатор кімнати).

```
navigator.mediaDevices
```

```
.getUserMedia({ video: true, audio: true })
.then((stream) => {
  userVideo.current.srcObject = stream
  socketRef.current.emit('join room', roomID)
```

В свою чергу веб-сервер отримує ідентифікатор кімнати і перевіряє чи є вже в ньому інші користувачі, якщо ні, то сервер додає до масиву users новий об’єкт з ідентифікатор користувача *socket.id* і ідентифікатором кімнати в якості ключа. Якщо в кімнаті вже є користувачі, то сервер виконує пошук в масиві *users* об’єкта з отриманим ідентифікатором кімнати і додає до цього об’єкту новий ідентифікатор користувача.

Після цього сервер повертає клієнту подію *all users* з масивом *usersInThisRoom*, який містить всіх поточних користувачів в даній кімнаті.

```
const users = {}
const socketToRoom = {}
socket.on('join room', (roomID) => {
   if (users[roomID]) {
    const length = users[roomID].length
    if (length === 9) {
     socket.emit('room full')
     return
    }
    users[roomID].push(socket.id)
   } else {
    users[roomID] = [socket.id]
   }
   socketToRoom[socket.id] = roomID
   const usersInThisRoom = users[roomID].filter((id) => id !== socket.id)

   socket.emit('all users', usersInThisRoom)
  })
```

Клієнт отримує масив з поточними користувачами в кімнаті, і для кожного користувача створює нове однорангове з’єднання за технологією *WebRTC*. Для цього викликається функція *createPeer*, яка почергово приймає ідентифікатори користувачів, ідентифікатор поточного користувача, медіа потік, і повертає змінну *peer* з даними про нове однорангове з’єднання. Далі дана змінна додається в масив поточних з’єднань *peers* разом з ідентифікатором користувача.

```
socketRef.current.on('all users', (users) => {
        const peers = []
        users.forEach((userID) => {
          const peer = createPeer(userID, socketRef.current.id, stream)
          peersRef.current.push({
            peerID: userID,
            peer,
          })
          peers.push(peer)
        })
        setPeers(peers)
      })
```

Функція *createPeer* створює нове однорангове з’єднання за допомогою метода *new Peer*. Функція приймає три параметри: *initiator* (ініціатор з’єднання), *trickle* (асинхронна обробка *ICE*), *stream*(медіа потік). Далі функція надсилає на сервер подію *sending signal* з медіа потоком, ідентифікаторами отримувача та відправника.

```
function createPeer(userToSignal, callerID, stream) {
    const peer = new Peer({
      initiator: true,
      trickle: false,
      stream,
    })
```

```
    peer.on('signal', (signal) => {
      socketRef.current.emit('sending signal', {
        userToSignal,
        callerID,
        signal,
      })
    })
    return peer
  }
```

Сервер отримує ці дані і надсилає їх кожному користувачу в кімнаті, як подію *user joined*.

```
socket.on('sending signal', (payload) => {
    io.to(payload.userToSignal).emit('user joined', {
      signal: payload.signal,
      callerID: payload.callerID,
    })
  })
```

Клієнт отримує медіа потік і ідентифікатор відправника. Далі викликається функція *addPeer*, яка створює та надсилає відповідь відправнику. Також дана відповідь додається до масиву *peers*.

```
socketRef.current.on('user joined', (payload) => {
  const peer = addPeer(payload.signal, payload.callerID, stream)
  peersRef.current.push({
    peerID: payload.callerID,
    peer,
  })

  setPeers((users) => [...users, peer])
})
```

Сервер перехоплює відповідь і надсилає її користувачу, який хоче приєднатися до кімнати, як подію *returning signal*.

```
socket.on('returning signal', (payload) => {
  io.to(payload.callerID).emit('receiving returned signal', {
    signal: payload.signal,
    id: socket.id,
  })
})
```

Наприкінці клієнт отримує відповідь від сервера і нарешті встановлює зв’язок з іншими користувачами в кімнаті.

```
socketRef.current.on('receiving returned signal', (payload) => {
  const item = peersRef.current.find((p) => p.peerID === payload.id)
  item.peer.signal(payload.signal)
})
```

### 3.3. Функціональні компоненти

Однією з особливостей бібліотеки *React.js* є можливість створення багаторазових функціональних компонентів. Даний компонент є частиною інтерфейсу користувача, який можна використовувати в різних частинах веб-додатку для створення більш ніж одного екземпляра інтерфейсу. Також дана можливість значно зменшує розмір коду та допомагає логічно структурувати його.

Наприклад, у веб-додатку може бути компонент *Button*, який відображає різні тексти на різних сторінках.

Кожен компонент може отримувати *Props*. *Props* – це вхідні дані *React*-компонентів, що передаються від батьківського компонента дочірньому. Ці дані є доступними лише для читання. Це дуже схоже на аргументи функції. Тобто, дані передаються компоненту так само, як і аргументи функції.

Веб-додаток для проведення відеоконференцій має чотири основні функціональні компоненти: *Button.jsx, Camera.jsx, Clock.jsx,* та *Input.jsx*. Всі дані компоненти знаходяться в папці components в основній директорії веб-додатку.

Компонент *Button* може отримувати наступні *Props*: *children, onClick, className*. Параметром *children* є будь яка текстова змінна, або зображення. *onClick* – функція, яку виконує кнопка після натискання на неї. *className* – набір стилів написаних мовою *CSS*, які змінюють зовнішній вигляд кнопки.

```
export function Button(Props) {
 const { children = '', onClick = () => {}, className = styles.btn } = Props
 return (
  <button className={className} onClick={onClick}>
   {children}
  </button>
 )
}
```

Компонент *Camera* отримує наступні *Props*: *stream, muted*. *Sream* – потік медіа даних. *Muted* – булеве значення, яке в залежності від отриманого значення вимикає, або вмикає вивід аудіо-потоку. Даний компонент повертає тег *video,* який додає, відтворює та керує налаштуваннями відео на веб-сторінці.

```
export function Camera({ stream, muted }) {
 const videoRef = useRef()
 const getVideo = (stream) => {
  let video = videoRef.current
  video.srcObject = stream
  video.play()
 }
 useEffect(() => {
  if (stream) {
   getVideo(stream)
```

```
    }
  })
  return (
    <div className={styles.app__cam}>
      <div style={styles.camera_frame}>
        <video
          style={styles.video}
          autoPlay
          muted={muted}
          ref={videoRef}
        ></video>
      </div>
    </div>
  )
}
```

Компонент *Clock* не отримує жодного Props. Але повертає числове значення поточного часу в форматі *HH:MM:SS* (години, хвилини, секунди). Для визначення поточного часу використовується функція *JavaScript new Date*, яка повертає об'єкт, що містить число мілісекунд, що пройшли з 1 січня 1970р. Для зміни даного об’єкту на формат *HH:MM:SS* застосовується функція *toTimeString().split(' ')[0]*.

```
export function Clock() {
  const [value, setValue] = useState(new Date())
  useEffect(() => {
    const interval = setInterval(() => setValue(new Date()), 1000)
    return () => {
      clearInterval(interval)
    }
  }, [])
```

```
  return <div className={styles.btn}>{value.toTimeString().split(' ')[0]}</div>
}
```

Компонент *Input* отримує наступні *Props*: *name*, *handleInput*. *Name* – змінна символьного типу, котра буде відображатися біля поля вводу. *handleInput* – функція *callback*, яка дозволяє батьківському компоненту отримати введене значення в поле вводу. Оскільки в бібліотеці *React* дані можуть передаватися тільки зверху вниз, тобто від батьківського елементу до дочірнього, то для вирішення цієї проблеми використовується функція *callback*. Функція *callback* є функцією зворотного виклику, яка передається як аргумент іншій функції, щоб її можна було виконати в цій іншій функції.

```
export function Input({ name, handleInput }) {
  return (
    <div className={styles.input}>
      <label>{name}</label>
      <input
        type="text"
        onChange={(event) => {
          handleInput(event.target.value)
        }}
      ></input>
    </div>
  )
}
```

### 3.4. Робота веб-додатку для проведення відеоконференцій

Для запуску веб-додатку необхідно відкрити будь-який веб-браузер, наприклад *Google Chrome* і в адресний рядок ввести наступну *URL*-адресу: http://localhost:3000.

Після запуску веб-додатку необхідно дати дозвіл веб-браузеру використовувати камеру та мікрофон, після чого відобразиться головна сторінка (рис. 3.1.).

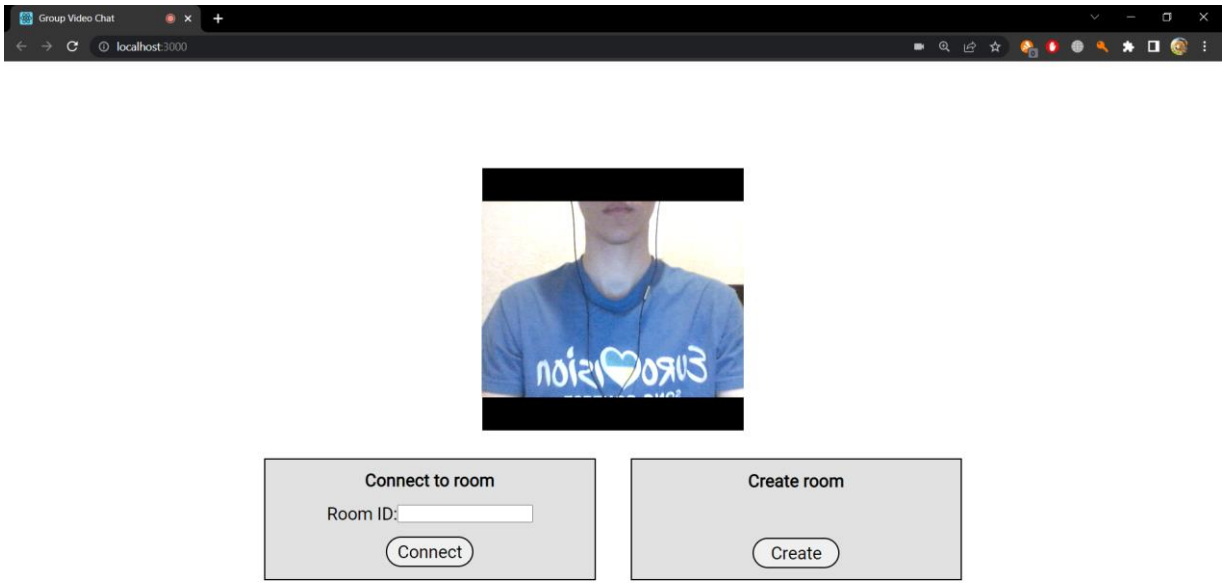


Рис. 3.1. Головна сторінка веб-додатку

На даній сторінці користувач може перевірити роботу своєї камери та мікрофона. Далі він може натиснути на кнопку *Create* для створення нової кімнати для проведення відеоконференції, або в рядку *Room ID* ввести ідентифікатор вже існуючої кімнати.

Після натискання на одну з цих кнопок відкривається друга сторінка *Room* (рис. 3.2.).

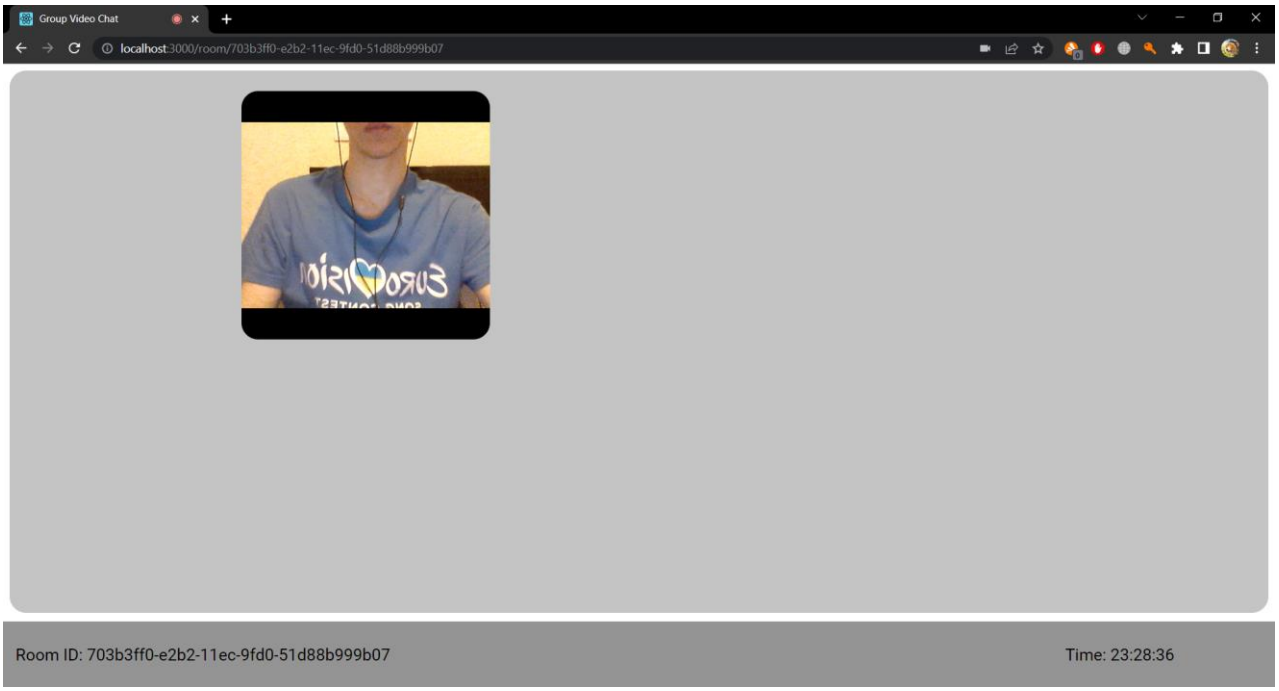


Рис. 3.2. Сторінка *Room* після створення кімнати

На цій сторінці відображаються камери всіх учасників відеоконференції. В лівому нижньому кутку користувач може побачити унікальний ідентифікатор поточної кімнати. В правому нижньому кутку відображається поточний час.

Для того, щоб до кімнати могли приєднатися інші учасники, організатор може скопіювати ідентифікатор кімнати, або *URL*-адресу з адресного рядка і відправити її іншій людині.

Після того, як інша людина перейде за посиланням, або введе ідентифікатор кімнати у відповідний рядок, вона зможе побачити та почути інших учасників відеоконференції (рис. 3.3).

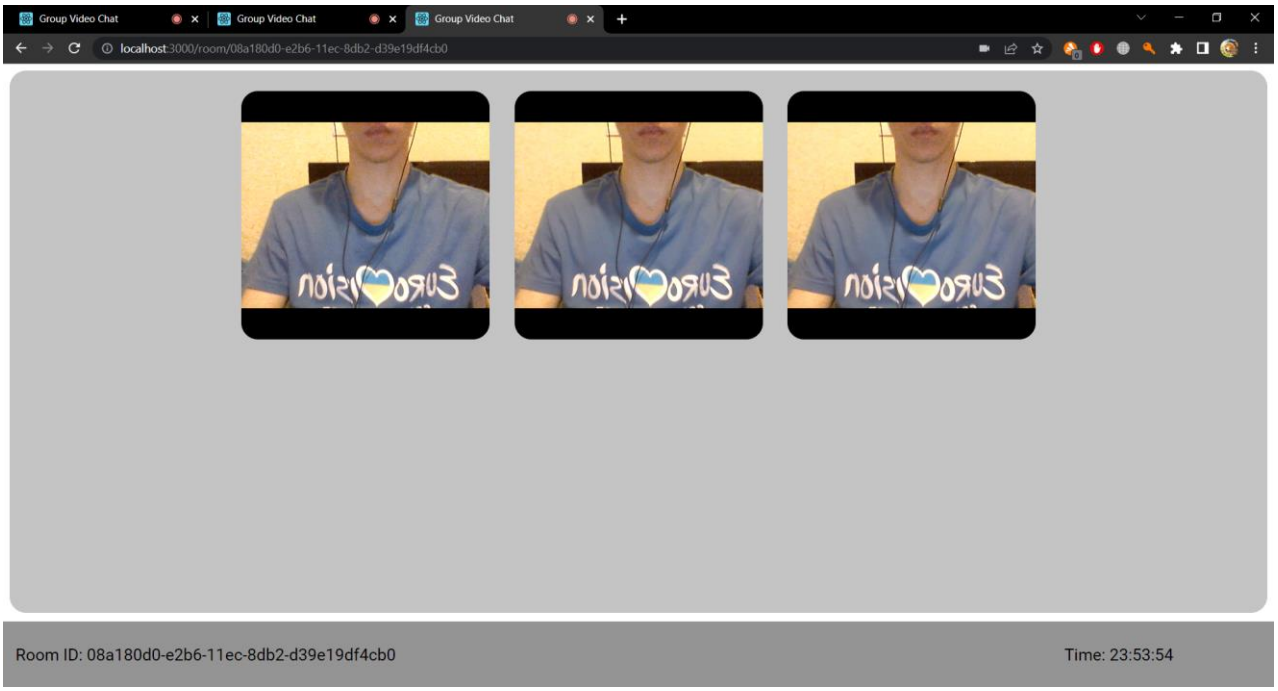


Рис. 3.3. Сторінка *Room* після приєднання кількох учасників

Для виходу з кімнати, користувачу необхідно закрити вкладку або вікно веб-браузера.

### 3.5. Висновки до розділу

У третьому розділі було детально описано алгоритм роботи веб-додатку для проведення відеоконференцій, як саме здійснюється створення зв’язку між клієнтами і сервером за технологією *WebRTC*. Також було продемонстровано роботу веб-додатку та наведено інструкцію користувача.

# ВИСНОВКИ

У дипломному проекту було розроблено веб-додаток для проведення відеоконференцій, що використовую технологію потокової передачі даних *WebRTC* та надає користувачам можливість спілкуватися з будь-якого місця лише маючи доступ до веб-браузеру та Інтернету.

По-перше, було проаналізовано предметну область «відеоконференції». Також було розглянуто та порівняно аналогічні веб-додатки, а також знайдено їх переваги та недоліки. Було розглянуто наступні веб-додатки: *Google Meet*, *Zoom*, *Discord*, *Skype*. Для кожного веб-додатку було надано опис їх функціональності.

По-друге, було розібрано технології створення веб-додатку. Було визначено функції і роль технологій потокової передачі даних в реальному часі, *WebRTC* і *Socket.IO*. Технологію *WebRTC* було обрано, бо вона вже є вбудованою в усі сучасні веб-браузери, а також надає розробникам зручні та корисні функції для створення веб-додатків заснованих на ній. Також не менш важливою метою було визначення бібліотеки та платформи для побудови клієнтської та серверної частини веб-додатку. Для досягнення цієї мети було обрано *JavaScript*-бібліотеку *React.js* і платформу для створення веб-серверів *Node.js*. Дані платформи є найбільш популярними та ефективними серед інших аналогічних в сфері розробки веб-додатків.

По-третє, було досліджено основний алгоритм веб-додатку для встановлення стабільного та постійного *Peer-to-Peer* з'єднання. Для цього було обрано бібліотеку *Simple-peer*, яка значно спрощує роботу з технологією *WebRTC*.

Окрім того, було створено інструкцію користувача, що містить простий та зрозумілий опис дій та знімки екрану.

В результаті було розроблено веб-додаток, що виконує наступні функції:

– створення нової кімнати для зустрічі;
– підключення до вже існуючої кімнати;
– попередній перегляд та прослуховування веб-камери;

– потокова передача аудіо- та відео-інформації;
– отримання поточного часу;
– отримання ідентифікатора кімнати;
– можливість підключення до існуючої кімнати за допомогою *URL*-адреси.

## СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Salvatore Loreto, Simon Pietro Romano, Real-Time Communication with WebRTC: Peer-to-Peer in the Browser, 1st edition. – Publisher: O'Reilly Media, 2014. – 164 pages.
2. Алекс Бенкс, Єва Порселло. React: сучасні шаблони для розробки веб-додатків, 2 видання. – «O'Reilly Media», 2022. – 320с.
3. Алекс Янг, Бредлі Мек, Майк Кантелон, Node.js в дії, 1 видання. – «Manning», 2014. – 548с.
4. Azat Mardan, Pro Express.js: Master Express.js: The Node.js Framework For Your Web Development, 1st edition. – Publisher: Apress, 2014. – 394 pages.
5. Andrew Lombardi, WebSocket: Lightweight Client-Server Communications, 1st edition. – Publisher: O'Reilly Media, 2015. – 144 pages.
6. Офіційна документація Mozilla [Електронний ресурс] 2022 URL: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API
7. Офіційна документація Socket.IO [Електронний ресурс] 2022 URL: https://socket.io/docs/v4
8. Офіційна документація Node.js [Електронний ресурс] 2022 URL: https://nodejs.org/uk/docs
9. Офіційна документація React JS [Електронний ресурс] 2022 URL: https://uk.reactjs.org/docs/getting-started.html
10. Marijn Haverbeke, Eloquent JavaScript: A Modern Introduction to Programming, 3rd edition. – Publisher: No Starch Press, 2018. – 472 pages.
11. David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, Sailu Reddy, HTTP: The Definitive Guide, 1st edition. – Publisher: O'Reilly Media, 2002. – 656 pages.
12. Jon Duckett, HTML and CSS: Design and Build Websites, 1st edition. – Publisher: John Wiley & Sons, 2011. – 490 pages.

## ДОДАТОК А

### Лістинг файлу *server.js*

```
require('dotenv').config()
const express = require('express')
const http = require('http')
const app = express()
const server = http.createServer(app)
const socket = require('socket.io')
const io = socket(server)

const users = {}

const socketToRoom = {}

io.on('connection', (socket) => {
  socket.on('join room', (roomID) => {
    if (users[roomID]) {
      const length = users[roomID].length
      if (length === 9) {
        socket.emit('room full')
        return
      }
      users[roomID].push(socket.id)
    } else {
      users[roomID] = [socket.id]
    }
    socketToRoom[socket.id] = roomID
    const usersInThisRoom = users[roomID].filter((id) => id !== socket.id)
```

```
    socket.emit('all users', usersInThisRoom)
  })

  socket.on('sending signal', (payload) => {
    io.to(payload.userToSignal).emit('user joined', {
      signal: payload.signal,
      callerID: payload.callerID,
    })
  })

  socket.on('returning signal', (payload) => {
    io.to(payload.callerID).emit('receiving returned signal', {
      signal: payload.signal,
      id: socket.id,
    })
  })

  socket.on('disconnect', () => {
    const roomID = socketToRoom[socket.id]
    let room = users[roomID]
    if (room) {
      room = room.filter((id) => id !== socket.id)
      users[roomID] = room
    }
    const usersInThisRoom = users[roomID].filter((id) => id !== socket.id)
    socket.emit('all users', usersInThisRoom)
  })
})

server.listen(process.env.PORT || 8000, () =>
```

```
  console.log('server is running on port 8000')
)
```

**Лістинг файлу *App.js***

```
import React from 'react'
import { BrowserRouter, Route, Switch } from 'react-router-dom'
import CreateRoom from './routes/CreateRoom'
import Room from './routes/Room'

function App() {
  return (
    <BrowserRouter>
      <Switch>
        <Route path="/" exact component={CreateRoom} />
        <Route path="/room/:roomID" component={Room} />
      </Switch>
    </BrowserRouter>
  )
}

export default App
```

**Лістинг файлу *index.js***

```
import React from 'react'
import ReactDOM from 'react-dom'
import App from './App'
import * as serviceWorker from './serviceWorker'
```

```
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
)

serviceWorker.unregister()
```

**Лістинг файлу *CreateRoom.js***

```
import React from 'react'
import { v1 as uuid } from 'uuid'
import { Button, Input, Camera } from '../components'
import styles from '../styles/main.module.css'
import { useEffect, useState } from 'react'

const CreateRoom = (props) => {
  const [localStream, setLocalStream] = useState()
  const [roomID, setRoomID] = useState('')

  const handleClickCreate = () => {
    const id = uuid()
    props.history.push(`/room/${id}`)
  }

  const handleClickConnect = () => {
    if (roomID.length === 0) {
      alert('Room ID entered incorrectly')
    } else {
```

```
    props.history.push(`/room/${roomID}`)
  }
}

const handleInputRoomID = (data) => {
  setRoomID(data)
}

useEffect(() => {
  const localVideo = {
    video: true,
    audio: true,
  }
  async function enableStream() {
    try {
      const stream = await navigator.mediaDevices.getUserMedia(localVideo)
      setLocalStream(stream)
    } catch (error) {
      console.warn(error)
    }
  }
  if (!localStream) {
    enableStream()
  } else {
    return function cleanup() {
      localStream.getTracks().forEach((track: any) => {
        track.stop()
      })
    }
  }
```

```
  }, [localStream])

  return (
    <div className={styles.app}>
      <main className={styles.app__profileData}>
        <Camera stream={localStream} />
        <article className={styles.app__panels}>
          <section className={styles.panels__connect}>
            <div className={styles.connect__form}>
              <p>Connect to room</p>
              <Input name="Room ID:" handleInput={handleInputRoomID} />
              <Button onClick={handleClickConnect}>Connect</Button>
            </div>
          </section>
          <section className={styles.panels__create}>
            <div className={styles.create__form}>
              <p>Create room</p>
              <Button onClick={handleClickCreate}>Create</Button>
            </div>
          </section>
        </article>
      </main>
    </div>
  )
}

export default CreateRoom
```

**Лістинг файлу Room.js**

```
import React, { useEffect, useRef, useState } from 'react'
import io from 'socket.io-client'
import Peer from 'simple-peer'
import styles from '../styles/conv.module.css'
import { Clock } from '../components'

const PeerCamera = (props) => {
  const ref = useRef()

  useEffect(() => {
    props.peer.on('stream', (stream) => {
      ref.current.srcObject = stream
    })
  }, [props.peer])

  return (
    <div className={styles.app__cam}>
      <div className={styles.camera__frame}>
        <video className={styles.camera} ref={ref} autoPlay></video>
      </div>
    </div>
  )
}

const Room = (props) => {
  const [peers, setPeers] = useState([])
  const socketRef = useRef()
  const userVideo = useRef()
```

```
const peersRef = useRef([])
const roomID = props.match.params.roomID

useEffect(() => {
  socketRef.current = io.connect('/')

  navigator.mediaDevices
    .getUserMedia({ video: true, audio: true })
    .then((stream) => {
      userVideo.current.srcObject = stream
      socketRef.current.emit('join room', roomID)
      socketRef.current.on('all users', (users) => {
        const peers = []
        users.forEach((userID) => {
          const peer = createPeer(userID, socketRef.current.id, stream)
          peersRef.current.push({
            peerID: userID,
            peer,
          })
          peers.push(peer)
        })
        setPeers(peers)
      })

      socketRef.current.on('user joined', (payload) => {
        const peer = addPeer(payload.signal, payload.callerID, stream)
        peersRef.current.push({
          peerID: payload.callerID,
          peer,
        })
```

```
        setPeers((users) => [...users, peer])
      })

      socketRef.current.on('receiving returned signal', (payload) => {
        const item = peersRef.current.find((p) => p.peerID === payload.id)
        item.peer.signal(payload.signal)
      })
    })
}, [roomID])

function createPeer(userToSignal, callerID, stream) {
  const peer = new Peer({
    initiator: true,
    trickle: false,
    stream,
  })

  peer.on('signal', (signal) => {
    socketRef.current.emit('sending signal', {
      userToSignal,
      callerID,
      signal,
    })
  })

  return peer
}

function addPeer(incomingSignal, callerID, stream) {
```

```
  const peer = new Peer({
    initiator: false,
    trickle: false,
    stream,
  })

  peer.on('signal', (signal) => {
    socketRef.current.emit('returning signal', { signal, callerID })
  })

  peer.signal(incomingSignal)

  return peer
}

return (
  <div className={styles.app}>
    <div className={styles.app__wrapper}>
      <main className={styles.wrapper__main}>
        <article className={styles.main__cameras}>
          <div className={styles.cameras__list}>
            <div className={styles.app__cam}>
              <div className={styles.camera__frame}>
                <video
                  className={styles.camera}
                  autoPlay
                  muted
                  ref={userVideo}
                ></video>
              </div>
```

```
            </div>
            {peers.map((peer, index) => {
              return <PeerCamera key={index} peer={peer} />
            })}
          </div>
        </article>
      </main>
      <footer className={styles.wrapper__footer}>
        <section className={styles.footer__roomID}>
          <p>Room ID: {roomID}</p>
        </section>
        <section className={styles.footer__time}>
          <p>Time:</p>
           
          {<Clock />}
        </section>
      </footer>
    </div>
  </div>
 )
}

export default Room
```

**Лістинг файлу *main.module.css***

```
@import
url('https://fonts.googleapis.com/css2?family=Roboto&display=swap');

*{
```

```
  padding: 0;
  margin: 0;
  font-family: 'Roboto', sans-serif;
}

.app {
  display: flex;
  height: 100vh;
  width: 100vw;
  align-items: center;
  justify-content: center;
}

.app__profileData {
  display: flex;
  flex-direction: column;
  width: 800px;
}

.app__profileData > * {
  margin-top: 4%;
}

.app__buttons {
  display: flex;
  justify-content: center;
}

.app__inputName {
  display: flex;
```

```
  font-size: 20px;
  justify-content: center;
}

.app__panels {
  display: flex;
  justify-content: center;
  height: 140px;
}

.panels__connect {
  display: flex;
  flex-direction: column;
  background-color: #e1e1e1;
  width: 50%;
  border: 2px solid black;
  margin-right: 5%;
  align-items: center;
  justify-content: space-around;
}

.panels__connect p {
  font-weight: bold;
  font-size: 20px;
}

.connect__form{
  display: flex;
  flex-direction: column;
  justify-content: space-around;
```

```
  align-items: center;
}

.connect__form > * {
  margin-top: 6%;
}

.connect__form button {
  font-size: 20px;
  width: 100px;
  height: 35px;
  border-radius: 20px;
  margin-bottom: 7%;
}

.connect__form button:hover {
background: #bebebe;
}

.panels__create {
  display: flex;
  flex-direction: column;
  background-color: #e1e1e1;
  width: 50%;
  border: 2px solid black;
  align-items: center;
  justify-content: space-around;
}

.create__form{
```

```
  display: flex;
  flex-direction: column;
  justify-content: space-around;
  align-items: center;
 }

 .create__form button {
  font-size: 20px;
  width: 100px;
  height: 35px;
  border-radius: 20px;
  margin-top: 48%;
 }

 .create__form button:hover {
 background: rgb(190, 190, 190);
 }

 .panels__create p {
  font-weight: bold;
  font-size: 20px;
 }
```

**Лістинг файлу *conv.module.css***

```
 @import
url('https://fonts.googleapis.com/css2?family=Roboto&display=swap');

 *{
  padding: 0;
```

```
  margin: 0;
}

.app ::-webkit-scrollbar {
  width: 12px;
}

.app ::-webkit-scrollbar-thumb {
  border-radius: 10px;
  background-color: #dfdfdf;
}

.app ::-webkit-scrollbar-track {
  box-shadow: inset 0 0 6px #00000033;
  border-radius: 10px;
  background-color: #343434;
}

.app {
  height: 100vh;
  width: 100vw;
  font-size: 20px;
}

.app__wrapper {
  display: flex;
  flex-direction: column;
  min-height: 100%;
}
```

```
.wrapper__main {
  display: flex;
  flex: 1;
  flex-direction: row;
  gap: 8px;
  margin: 8px;
}

.main__cameras {
  display: flex;
  flex: 3;
  border-radius: 20px;
  justify-content: center;
  align-items: center;
  background-color: #C4C4C4;
  height: 655px;
}

.cameras__list {
  display: grid;
  padding-top: 25px;
  padding-bottom: 25px;
  grid-template-columns: 1fr 1fr 1fr;
  grid-column-gap: 30px;
  grid-row-gap: 15px;
  overflow-x: hidden;
  overflow: auto;
  height: 605px;
}
```

```
.cameras__list > * {
  display: flex;
  height: 1;
  border-radius: 20px;
}

.wrapper__footer {
  display: flex;
  flex: 0 0 auto;
  box-sizing: border-box;
  height: 80px;
  justify-content: space-between;
  align-items: center;
  background-color: rgb(148, 148, 148);
}

.footer__roomID {
  display: flex;
  text-align: center;
  width: 33%;
}

.footer__roomID p {
  margin-left: 3%;
}

.footer__buttons {
  display: flex;
  justify-content: center;
  width: 33%;
```

```
}

.footer__time {
  display: flex;
  width: 33%;
}

.footer__time p {
  margin-left: 50%;
}

.app__cam {
  margin-left: auto;
  margin-right: auto;
  position: relative;
  width: 300px;
  height: 300px;
  background-color: black;
}

.camera__frame {
  display: flex;
  position: absolute;
  justify-content: center;
  align-items: center;
  width: 100%;
  height: 100%;
}

.camera {
```

```
  width: 100%;
  height: 100%;
  transform: scale(-1, 1);
}
```

**Лістинг файлу *Button.jsx***

```
import styles from './Button.module.css'
import React from 'react'

export function Button(Props) {
  const { children = '', onClick = () => {}, className = styles.btn } = Props
  return (
    <button className={className} onClick={onClick}>
      {children}
    </button>
  )
}

export default Button
```

**Лістинг файлу *Camera.jsx***

```
import styles from './Camera.module.css'
import { useEffect, useRef } from 'react'
import React from 'react'

export function Camera({ stream, muted, ref }) {
  const videoRef = useRef()
```

```
const getVideo = (stream) => {
  let video = videoRef.current
  video.srcObject = stream
  video.play()
}

useEffect(() => {
  if (stream) {
    getVideo(stream)
  }
})

return (
  <div className={styles.app__cam}>
    <div
      style={{
        display: 'flex',
        position: 'absolute',
        justifyContent: 'center',
        alignItems: 'center',
        width: '100%',
        height: '100%',
      }}
    >
      <video
        style={{ width: '100%', height: '100%', transform: 'scale(-1, 1)' }}
        autoPlay
        muted={muted}
        ref={videoRef}
      ></video>
```

```
      </div>
    </div>
  )
}

export default Camera
```

**Лістинг файлу *Clock.jsx***

```
import styles from './Clock.module.css'
import { useEffect, useState } from 'react'
import React from 'react'

export function Clock() {
  const [value, setValue] = useState(new Date())

  useEffect(() => {
    const interval = setInterval(() => setValue(new Date()), 1000)

    return () => {
      clearInterval(interval)
    }
  }, [])

  return <div className={styles.btn}>{value.toTimeString().split(' ')[0]}</div>
}

export default Clock
```

**Лістинг файлу *Input.jsx***

```
import styles from './Input.module.css'
import React from 'react'

export function Input({ name, handleInput }) {
  return (
    <div className={styles.input}>
      <label>{name}</label>
      <input
        type="text"
        onChange={(event) => {
          handleInput(event.target.value)
        }}
      ></input>
    </div>
  )
}

export default Input
```