Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра обчислювальної техніки
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна система для проведення відеоконференцій»

Виконав: студент 2-го курсу, групи 2КІ-21м
спеціальності 123 — Комп'ютерна інженерія
_____ Бабецький Е.В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. ОТ
_____ Колесник І.С.
(прізвище та ініціали)
«20» 12 2022 р.

Опонент: д.т.н., професор каф. ПЗ
_____ Яремчук Ю.Є.
(прізвище та ініціали)
«22» 12 2022 р.

Допущено до захисту
_____ д.т.н., проф. Азаров О.Д.
(прізвище та ініціали)
«23» 12 2022 р.

Вінниця 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень — магістр
Спеціальність 123 — «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
Азаров О. Д.
«15» 09 2022 р

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бабецький Едуард Владиславович
(прізвище, ім'я, по-батькові)

1 Тема роботи: «Інформаційна система для проведення відеоконференцій» Керівник роботи: к.т.н., доцент кафедри ОТ Колесний Ірина Сергіївна затверджені наказом Вінницького національного технічного університету від 15.09.2022 року № 205-А.

2 Строк подання студентом роботи 10.12.2022.

3 Вихідні дані до роботи: передача відео та аудіо, мінімальна кількість пристроїв – 36, мова програмування – об'єктно-орієнтована.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): аналіз методів виявлення вразливостей та технології обробки відеопотоку за допомогою WebRTC; аналіз та вибір інструментів для розроблення та передачі відео, аудіо між браузерами; розробка програмного компоненту безпечної передачі відео зв'язку між користувачами, тестування програмного засобу, економічна частина.

5 Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): Алгоритм роботи програми для проведення відеоконференцій, UML діаграма класів програми для проведення відеоконференції, робочі вікна програми для проведення відеоконференцій, результати тестування програми для проведення відеоконференцій.

6 Консультанти розділів роботи

Таблиця 6.1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2,3,4	**Колесник І.С., к.т.н., доц. каф. ОТ**		
5	**Небава М.І., к.е.н., професор каф. ЕПВМ**		

7 Дата видачі завдання _____

8 Календарний план

Таблиця 8.1 — Календарний план

№ з/п	Назва етапів виконання бакалаврської дипломної роботи	Строк виконання етапів роботи	Підпис
1	Аналіз вимог завдання, вибір методів і засобів розв'язання поставленої задачі	04.10.2022	вик.
2	Розробка структур окремих підсистем	15.10.2022	вик.
3	Програмна реалізація системи	04.11.2022	вик.
4	Підготовка економічної частини	21.11.2022	вик.
5	Оформлення пояснювальної записки, графічного матеріалу та презентації	30.11.2022	вик.

Студент _____ Бабецький Е. В.
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ Колесник І. С.
(підпис) (прізвище та ініціали)

Консультант з економічної частини _____ Небава М. І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.4

Бабецький Е. В. Інформаційна система для проведення відеоконференцій. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна інженерія, освітня програма — комп'ютерна інженерія. Вінниця: ВНТУ, 2022. 131 с.

Укр. мовою. Бібліогр.: 37 назв; рис. 45; табл.: 6.

Дана магістерська кваліфікаційна робота присвячена розробці програмного забезпечення для проведення відеоконференцій. Були розглянуті та проаналізовані існуючі технології організації передачі аудіо та відео даних між браузерами, як найбільш перспективну, було обрано WebRTC. Удосконалений технологію передачі протоколів між браузерами, що забезпечує безперебійну передачу та підтримку трансляції. Було спроектовано інформаційну систему для проведення відеоконференцій, написано за допомогою React, NodeJS та з використанням середовища WebStorm.

У економічному розділі розраховано суму витрат на розробку та виготовлення нового технічного рішення, яка складає 1607473 гривень, спрогнозовано орієнтовану величину витрат по кожній з статей витрат, розраховано чистий прибуток, термін окупності витрат для виробника 0,67 роки та економічний ефект для споживача при використанні даної розробки.

Ключові слова: відеоконференція, NodeJS, MVC, WebRTC, WebSocket.

ABSTRACT

UDC 004.4

E. V. Babetskyi Information system for conducting video conferences. Master's thesis in specialty 131 — computer engineering, educational program — computer engineering. Vinnytsia: VNTU, 2022.

Ukraine language Bibliography: 37 titles; Fig. 45; tab.: 6.

This master's thesis is devoted to the development of software for conducting video conferences. Existing technologies for organizing audio and video data transfer between browsers were considered and analyzed, and WebRTC was chosen as the most promising. Improved protocol transfer technology between browsers, which ensures uninterrupted transmission and broadcast support. An information system for conducting video conferences was designed, written using React, NodeJS and using the WebStorm environment.

In the economic section, the amount of costs for the development and production of a new technical solution is calculated, which is 1607473 hryvnias, the estimated amount of costs for each of the cost items is predicted, the net profit is calculated, the payback period for the manufacturer is 0.67 years and the economic effect for the consumer when using this developments.

Keywords: video conference, NodeJS, MVC, WebRTC, WebSocket.

ЗМІСТ

ВСТУП

Розвиток швидкісного інтернету розширив межі спілкування між людьми. Відео зв’язок між двома користувачами вже успішно реалізовано в багатьох програмних засобах. Однак в деяких сферах діяльності виникає необхідність забезпечення відео зв’язку між багатьма учасниками. Прикладом може бути забезпечення проведення дистанційних лекційних та практичних занять , проведення засідань в рамках розподілених команд, що часто зустрічається під час розробки програмного забезпечення. Наразі, технічне забезпечення відео зв’язку між багатьма учасниками не є досконалим. В усіх присутніх на ринку програмних засобах є технічні обмеження по кількості учасників (SkypeDesktop – до 25, FaceTime – до 32), і, при збільшенні кількості учасників зменшується якість медіа-потоків (відео та аудіо). Таким чином, виникає необхідність розробки нового програмного продукту, що дозволяє відео-спілкування між багатьма учасниками (більше 35-ти). Існуючі програмні продукти, такі як Skype, FaceTime, працюють за принципом клієнт-серверної архітектури. При великому навантаженні сервера якість зв’язку та швидкість її передачі між клієнтами буде спадати зі зростанням активних користувачів. Таким чином, найбільш гостро відчуваються ці недоліки при забезпеченні відео-зв’язку між багатьма користувачами, такими як відеоконференції. Ще більш складною є реалізація систем, що дозволяють одночасне відображення багатьох відео-потоків, що потребується, наприклад, для відеоконференцій в процесі проведення дистанційного навчання або для забезпечення спілкування віддалених команд розробки програмного забезпечення. Саме тому більшість існуючих систем або зовсім не мають можливості створення відеоконференцій, або накладають суттєві обмеження на кількість відеопотоків. Отже, необхідність зменшення витрат на підтримку серверної частини програмних систем, що надають можливість проведення віддалених відеоконференцій, обумовлює актуальність дослідження існуючих технологій на предмет можливості забезпечення більшої ефективності при передачі

медіа-потоків. Програмний продукт для забезпечення відео конференції повинен підтримувати одночасний зв’язок із багатьма (більше 30-ти) користувачами системи, повинен бути надійним та швидко асинхронно обробляти медіа-потоки відео та аудіо.

Мета та завдання досліджень. Мета магістерської кваліфікаційної роботи полягає в аналізі технологій передачі даних та створення програмного продукту для забезпечення відеоконференцій.

Для досягнення мети роботи потрібно виконати такі задачі:

— проаналізувати існуючі технології передачі потоків відео та аудіо в розрізі підтримки відеоконференцій;

— спроектувати архітектуру програмного продукту для забезпечення відеоконференцій з урахуванням запропонованих методів та алгоритмів;

— реалізувати серверну та клієнтську частини програмного продукту для забезпечення відео-конференцій;

— провести аудит ідеї проекту та методи розроблення маркетингової та ринкової стратегій розвитку програмного продукту.

Об’єкт дослідження — програмне забезпечення технології передачі даних при проведенні відео-конференції.

Предмет дослідження — програмне забезпечення для проведення відеоконференцій з використанням нової технології WebRTC**.**

Методи дослідження. У роботі використані такі методи наукових досліджень:

— методи і шаблони проектування систем за принципами ООП;

— методи розробки з використанням класифікації за топологічною специфікою.

Наукова новизна одержаних результатів. Найбільш суттєвими науковими результатами магістерської дисертації є:

— вперше реалізовано програмний продукт для забезпечення відеоконференцій з використанням технології WebRTC;

— удосконалено спосіб побудови серверної частини програмного продукту за рахунок використання технології WebRTC, що зменшує навантаження на серверну частину програмного продукту для забезпечення відео-конференцій;

— набуло подальшого розвитку використання методів побудови клієнтської частини програмного продукту для проведення відео-конференцій з високим рівнем швидкодії.

Практичне значення одержаних результатів полягає в програмного продукту для забезпечення відеоконференції з кількістю учасників більше 35-ти та високою якістю медіа-даних у таких областях, як менеджмент віддалених команд розробки та у процесі проведення дистанційних лекційних та практичних занять.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ПЕРЕДАЧІ МЕДІА-ПОТОКІВ

Багато галузей сьогодні мають проблеми забезпечення якісного відео зв’язку між багатьма учасниками. Існуючі програмні продукти, такі як Zoom, FaceTime, Meet, Skype працюють за принципом клієнт-серверної архітектури, в якій серверна частина програмного продукту є вузьким місцем. Якщо сервер сильно завантажений, швидкість та якість зв’язку передачі потоків даних між клієнтами будуть погіршуватися зі збільшенням кількості активних користувачів. Найбільш гостро ці недоліки відчуваються при забезпеченні відеозв'язку багатьох користувачів, наприклад відеоконференції. У цьому розділі представлено порівняльний аналіз існуючих технологій передачі даних клієнтськими частинами програмного забезпечення для відеоконференцій, архітектурні відмінності в більшості програмних продуктів що будуються на основі відомих протоколах передачі медіапотоку: HTTP, WebSocket, SSE та WebRTC. Обґрунтовано вибір технології програмного продукту.

1.1 Техніки передачі даних Polling/Long Polling та Server-Sent Events

Протокол Hypertext Transfer Protocol (RFC2616) — це Протокол запиту/відповіді, що визначає такі сутності: клієнт, проксі та сервер. Клієнт встановлює з'єднання з сервером для надсилання запиту HTTP. Сервер приймає клієнтські з'єднання та повертає відповіді для подальшої обробки HTTP-запитів. Проксі — це проміжні об'єкти, які можуть бути задіяні в процесі доставки запитів клієнта на сервер та навпаки[1].

Стандартна модель HTTP не дозволяє відправляти дані на клієнтську сторону обміну без відповідного запиту (не передбачає надсилання данних асинхронно). Тому клієнтська частина веб-додатку, що імітує асинхронне отримання повідомлень (по стандартній моделллі HTTP), має періодично надсилати запити для отримання нових даних. Але постійні запити до сервера мають змогу зменшити пропускну здатність шляхом ініціювання запиту/відповіді в обох напрямках, навіть якщо немає даних, уповільнюючи

здатність до реагування на стороні клієнта. Потрібно більше часу, щоб відповісти на дії користувача, оскільки клієнт очікує відповіді від сервера.

Тому необхідно знайти компроміс між швидкістю отримання нових даних і навантаженням на серверну частину. Порівняно з технікою “short polling”, техніка “long polling” може зменшити навантаження на мереже та постійне опрацювання сервером, підтримуючи з’єднання для запитів. За допомогою техніки “long polling” клієнтська частина відсилає запит на отримання нових даних, а сервер відповідає, якщо у відповіді з’являються нові дані або перевищено ліміт часу з’єднання.

Коли буде отримано відповідь від сервера клієнт знову надсилає запит на отримання данних. Робочий цикл з використанням технології “long polling” можливо описати в послідовності (Рисунок 1.1):

— клієнт надсилає запит та залишається в стані очкування відповіді від сервера.

— сервер підтримує виконання запиту, доки не з’являться нові дані або не закінчиться час очікування.

— після отримання нових даних сервер надсилає відповідь на запит клієнта, що містить нові дані.

— клієнт (за потреби) очікує закінчення “періоду паузи” та надсилає повторний запит на отримання даних.

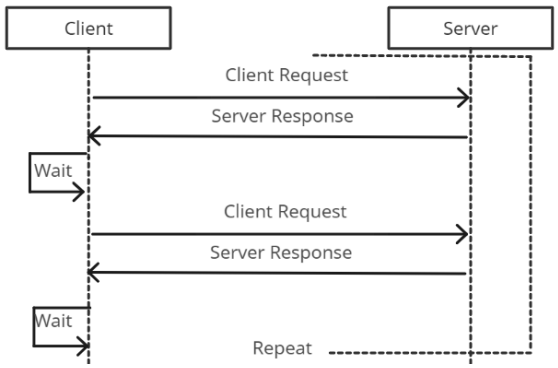

Рисунок 1.1 — Діаграма послідовності обміну даними з використанням технології опитування

Найважливішим недоліком описаної методики є[2]: надлишкові заголовки: згідно з технікою довгого опитування HTTP, кожен запит і відповідь на запит містить повний набір заголовків HTTP в повідомленні. Тому загальний обсяг корисних даних може бути меншим за обсяг заголовків і метаінформації. Максимальна затримка: після надсилання відповіді на довгий запит на опитування серверна частина повинна очікувати наступного запиту від клієнтської частини. Затримка може бути високою, оскільки пакети, надіслані та отримані від клієнта, можуть бути втрачені. Перевагами методів “long polling” і “polling” є можливість використання протоколу HTTP без модифікації чи додаткової конфігурації, кросбраузерна сумісність та незалежність від конкретної реалізації параметрів безпеки обміну даними[3].

Недоліки цих методів включають значні ресурсні витрати мережі, клієнтські та серверні обчислювальні ресурси через постійні запити HTTP, обробку даних на стороні сервера, а також збільшення обробки запитів і відповідей. Крім того, у разі технічного збою серверної частини, клієнтська частина не зможе спілкуватися з іншими клієнтами або серверами[4]. Наступним кроком в еволюції серверної доставки подій є створення технології HttpStreaming. Основна концепція HttpStreaming полягає у встановленні одностороннього з’єднання сервер/клієнт. Реалізації, такі як BOSH і Bayeux, побудовані на концепції змішаного використання потокової передачі HTTP та методів Long Polling/Polling. Потрібна додаткова конфігурація проксі та надлишкової конфігурації розгортки. Він працює на основі концепції утримання з’єднань на вимогу. Наприклад, Bayeux, який підтримує двонаправлений зв’язок, підтримує два мережеві потоки, щоб забезпечити можливості двонаправленого асинхронного обміну повідомленнями. Один потік — клієнт-сервер, а інший — сервер-клієнт. Основна складність реалізації є підтримка і налаштування серверної частини. Тоді стандарт подій, надісланих сервером, визначає концепцію надсилання повідомлень із сервера до клієнта. Відповідно до[5], Server-Sent Events (SSE) — це технологія надсилання повідомлень із серверів до веб-браузерів у формі подій DOM.

Головною перевагою перед опитуванням/довгим опитуванням є обмін миттєвими повідомленнями від сервера до клієнта (Рисунок 1.2).

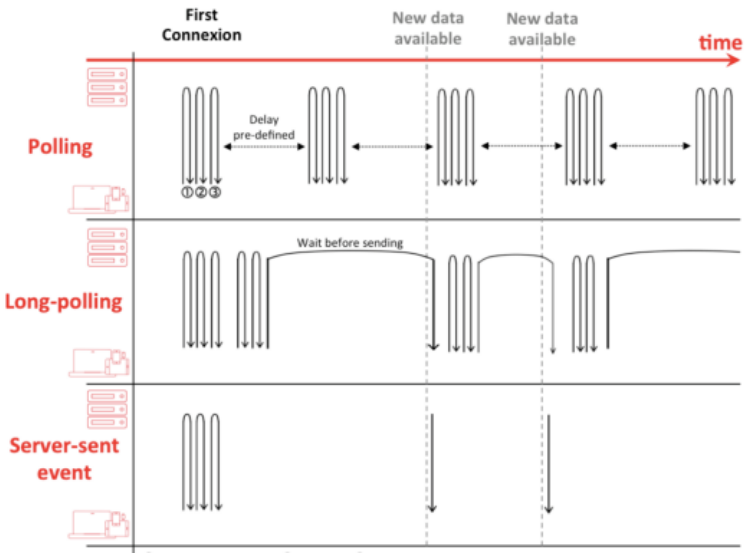

Рисунок 1.2 — Схема порівняння концепцій обміну даними Polling/Long Polling та SSE з часом

Розроблено для оптимізації кросбраузерних трансляцій за допомогою JavaScript API під назвою EventSource. Дані закодовуються у форматі типу text/eventstream і передаються до клієнта за допомогою механізму потокової передачі HTTP. Перевагами впровадження цього стандарту є оптимізоване використання ресурсів сервера, швидкість отримання повідомлень і простота впровадження для серверної та клієнтської частин. Недоліки реалізації механізму потокової передачі HTTP включають постійну підтримку HTTP-з'єднання з сервером і повну залежність від серверної частини.

1.2 Протокол передачі даних WebSocket

Протокол обміну данними WebSocket, регламентований RFC-6455[6], забезпечує двонаправлений повнодуплексний зв'язок із клієнтською частиною (серверна та клієнтська частини надсилають і отримують дані з одного потоку)[7]. Відповідно до RFC 6455, протокол складається з двох частин: рукостискання та передачі даних (рисунок 1.3 та рисунок 1.4). З'єднання

клієнт-сервер за допомогою протоколу WebSocket ініціюється "рукостисканням" запиту GET до сервера WebSocket[6]:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Рисунок 1.3 — Запит на з’єднання по проколу WebSocket

Якщо сервер підтримує протокол, він сформує відповідь клієнту про успішне отримання конфігураційних даних і сигнал про початок обміну даними (рисунок 1.4).

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Рисунок 1.4 — Відповідь сервера на запит з’єднання за протоколом WebSocket

Підключення TCP залишається активним, якщо браузер підтримує протокол. Подальший обмін даними відбувається за допомогою передачі фреймів WebSocket (Рисунок 1.5).

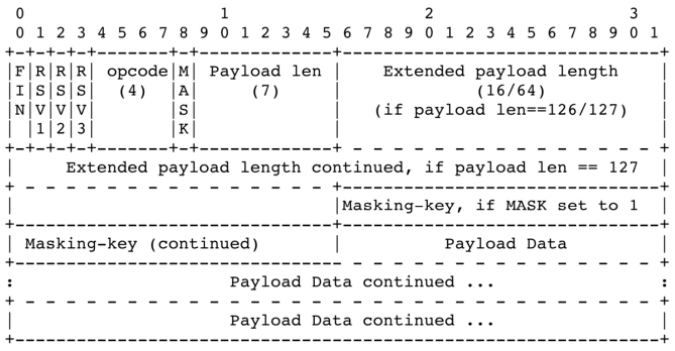

Рисунок 1.5 — Структура WebSocket фрейму за RFC 6455

Заголовок пакета складається з:

— fin (1 біт): маркер того, чи є поточний фрейм останнім. Він відображається, навіть якщо повідомлення складається з одного фрейму.

— rsv1, rsv2, rsv3 (кожен параметр складається з 1 біта): дані поля мають бути встановлені на 0, але лише за відсутності попередньої домовленості про розширення значень цих параметрів. Якщо для кадру встановлено значення 1 без попередньої згоди, одержувач повинен закрити з'єднання.

— opcode (4 біти): кодування вмісту фрейму. Використовуються такі значення: 0x00: поточний фрейм містить іншу частину повідомлення, 0x01: фрейм містить текстові дані (кодування – UTF-8), 0x02: фрейм містить двійкові дані, 0x08: поточний фрейм завершує з'єднання, 0x09: ping-фрейм, 0x0a: pong-фрейм.

— mask (1 біт): вказує на те, що фрейм замасковано. Відповідно до специфікації, усі повідомлення від клієнта до сервера мають бути замасковані. В іншому випадку специфікація рекомендує примусове відключення.

— payload_len (7 біт): містить довжину корисного навантаження повідомлення. Значення від 0 до 125 вказує на довжину корисного навантаження. Значення 126 вказує на те, що наступні 2 байти визначають розмір корисного навантаження. Значення 127 вказує на те, що наступні 8 байтів містять інформацію про розмір корисного навантаження повідомлення.

— masking-key (32-біти): кожен фрейм, надісланий від клієнта до сервера має бути замаскований за допомогою 32-бітового значення, що зберігається у фреймі.

— payload: корисне навантаження повідомлення (його довжина має дорівнювати значенню payload_len)

Використовуючи наведені вище концепції та специфікації фреймів, досягається гнучкість у типології корисних навантажень фреймів та їхніх розмірів.

Повідомлення heartbeat (описані в специфікації RFC 6455) можна використовувати для оптимізації пам'яті. Повідомлення heartbeat базується на концепції повідомлення ping-pong. Сервер надсилає ping-повідомлення на

клієнтську сторону програмного забезпечення. Клієнт потім надсилає pong-відповідь на запит[8]. Таким чином, якщо клієнт відповідає на ping-запит повідомленням pong, підключення вважається активним.

Фрагментація фреймів дозволяє серверній частині програмного продукту використовувати буфер кадрів для заповнення кадрів, надісланих до клієнтської частини програмного продукту. Це можна використовувати для оптимізації використання мережевих ресурсів[9].

Найбільш вразливою частиною будь-якого програмного продукту (основним протоколом обміну даними є WebSocket) є серверна частина (Рисунок 1.6). Клієнтська і серверна сторони повинні відповідати за перетворення даних і підтримку постійного зв'язку з сервером.

Якщо сервер виходить з ладу, усі клієнти, які спілкуються з сервером, не зможуть спілкуватися один з одним. Рішенням цієї проблеми є горизонтальне масштабування серверної частини програмного забезпечення. Це призводить до збільшення як фінансових, так і ресурсних витрат на підтримку сервера.

Перевага WebSockets полягає в тому, що немає обмежень на кількість як активних, так і пасивних з'єднань, за винятком пам'яті самого сервера. Протокол довів свою хорошу позицію за багатьма параметрами, такими як безпека, універсальність і швидкість[10].

Реалізований у сучасних браузерах, його можна використовувати в проектах, заснованих на інтерактивній взаємодії користувача.

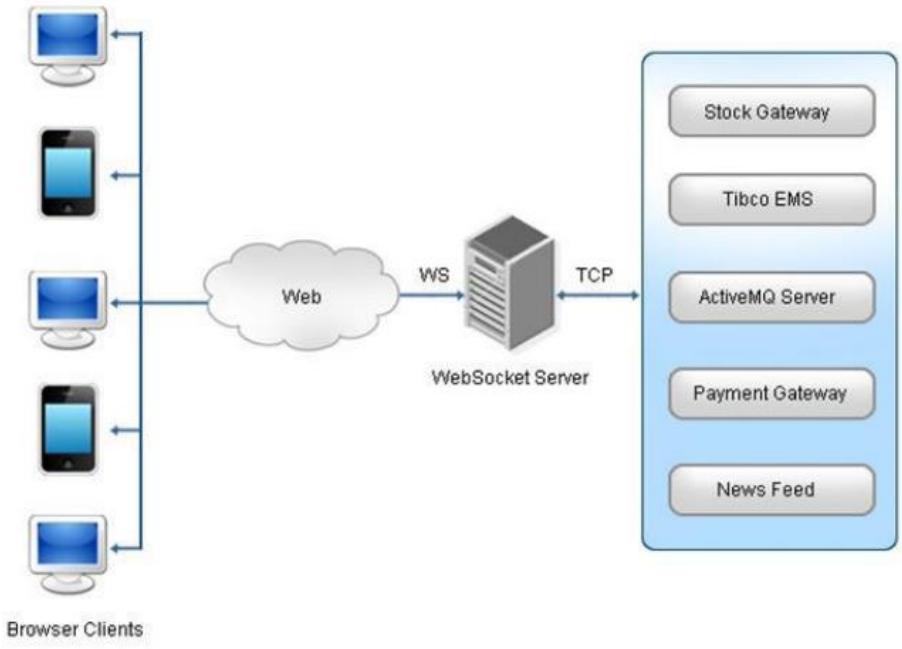

Рисунок 1.6 — Схема відеоконференцій без використання технології WebRTC

Недоліком використання цієї технології є управління потоком на стороні сервера програмного продукту та постійне перетворення даних як на стороні клієнта, так і на стороні сервера програмного продукту.

Серверна частина будь-якого програмного продукту, що реалізує протокол WebSocket, є найбільш вразливим програмним забезпеченням.

Клієнтська частина програмного продукту повинна відповідати за групування повідомлень WebSocket разом, перетворення даних за допомогою паралелізму, перевірку узгодженості та відповідності, а також вибір кодека та перетворення даних.

1.3 Протокол передачі даних WebRTC

Нова технологія WebRTC дає можливість досягти компромісу між вимогами до швидкості та витратами на обслуговування серверного програмного забезпечення.

Технологія WebRTC регулюється RFC-7478[11] і RFC-5245[12]. Вони описують вимоги до розробників браузерів, які надають API для використання

реалізацій протоколу передачі WebRTC для розробників клієнтської частини своїх програмних продуктів.

Основна ідея WebRTC полягає в реалізації конфігурації прямого спілкування користувачів за кросбраузерною концепцією. Тому для передачі медіаданих не потрібна наявність самої серверної частини. Основна роль сервера полягає в налагодженні обміну конфігураційними даними між клієнтськими частинами програмного забезпечення (Рисунок 1.7).

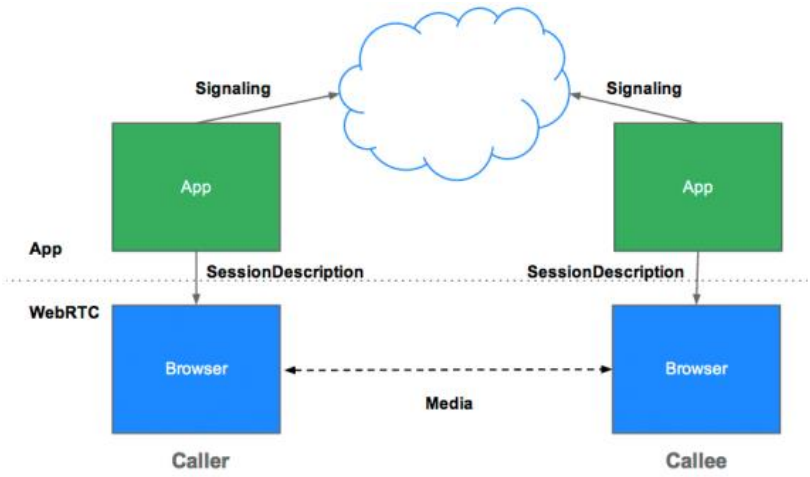

Рисунок 1.7 — Схема взаємодії клієнтів з використанням технології WebRTC

Перед початком обміну медіа-даними клієнтська частина надсилає «запрошення» (offer) і «offer on offer» (пропозиція на оферту) за допомогою протоколу SDP[13]. Коли отримано позитивну відповідь на запит на підключення, клієнт ініціює обмін кандидатами на підключення (ICECandidate) за допомогою протоколу ICE.

Серверна частина виконує лише початкове налаштування зв’язку за такої архітектури. Таким чином, навіть у разі повного виходу з ладу серверної частини програмного забезпечення існуючі з'єднання не будуть зруйновані і будуть передавати інформацію без будь-яких змін якості та швидкості (що не може бути реалізовано з використанням сімейств протоколів Polling та WebSocket).

Таким чином, серверна частина програмного продукту більше не є слабкою ланкою системи, вона виконує лише роль налаштування з’єднання і не потребує управління всім життєвим циклом каналу передачі медіаданих.

Це досягається завдяки інтерфейсу RTCPeerConnection, який дозволяє налаштовувати канали зв'язку між окремими браузерами. Можливість використання цього інтерфейсу для налаштування каналів зв’язку між окремими клієнтами системи, передачі даних безпосередньо між браузерами, розвантаження серверних частин програмних продуктів, а також передачі відео та аудіопотоків між браузерами. Додатковий розвиток логіки перетворення та перекодування.

Далі слід спроектувати та реалізувати сигнальний сервер відповідно до логіки взаємодії з клієнтською частиною програмного продукту.

Основною сферою відповідальності сервера сигналів є передача налаштувань сигналу між клієнтськими частинами програмного продукту, необхідних для налаштування RTCPeerConnection. Відповідно до специфікації, формат передачі даних між клієнтом і сервером може вибрати розробник програмного забезпечення.

Під час обміну кандидатами ICE слід використовувати сервер STUN відповідно до специфікації WebRTC. Без сервера STUN кросбраузерне налагодження можливе, лише якщо обидва браузери фізично знаходяться в одній локальній мережі (Рисунок 1.8).

Рисунок 1.8 — Зв’язок браузерів у локальній мережі

Протокол STUN — це мережевий протокол, який дозволяє користувачам визначати порти у зовнішній мережі, пов’язані з визначенням зовнішніх IP-адрес, методів трансляції адрес і номерів внутрішніх портів.

Ця інформація використовується для встановлення з’єднання UDP (протокол дейтаграм користувача — один із протоколів у стеку TCP/IP), коли два хости знаходяться за маршрутизатором NAT[14].

Протокол визначено в RFC 3489. Трансляція мережевих адрес (NAT) — це механізм зміни мережевої адреси в IP-заголовку дейтаграми під час її проходження через пристрій маршрутизації з метою маршрутизації одного адресного простору до іншого адресного простору[16].

Сервер STUN опублікований в Інтернеті та відкриває публічну частину клієнтського хосту. Процес виявлення дозволяє веб-вузлу WebRTC отримати власну публічну адресу та передати її іншій одноранговій системі через механізм сигналізації для встановлення прямого зв’язку (Рисунок 1.9)[17].

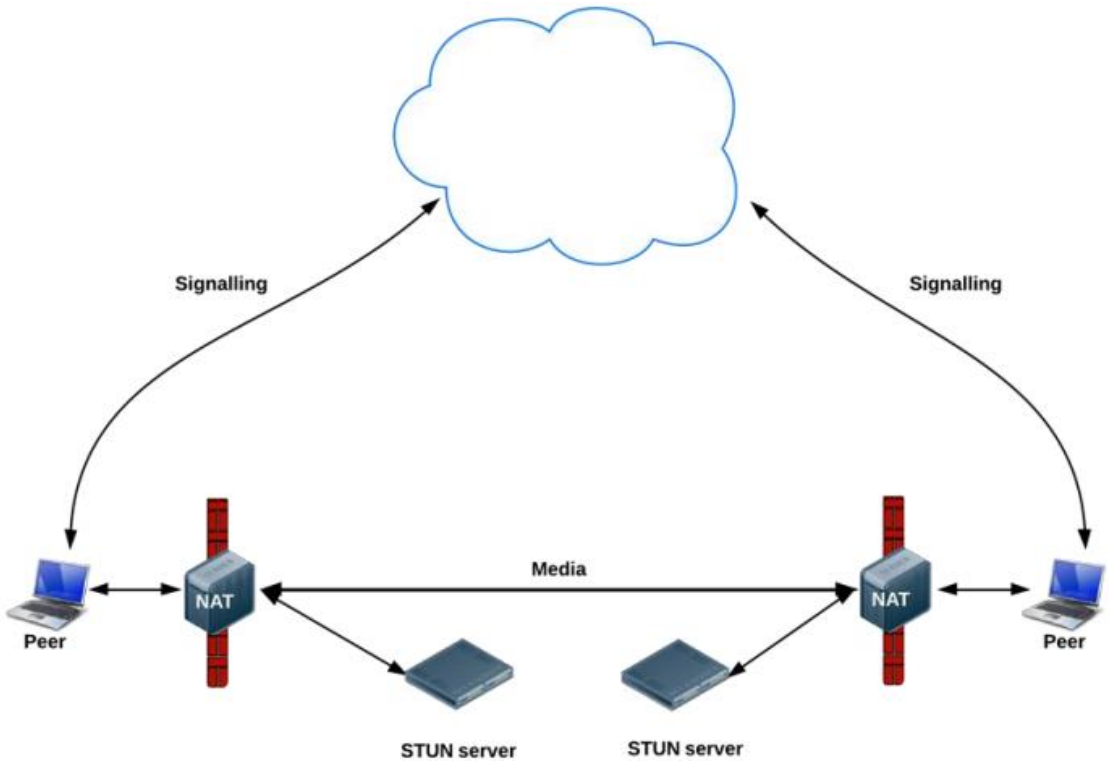

Рисунок 1.9 — Схема взаємодії браузерів з використанням STUN-серверів

Якщо присутній сервер STUN, система може встановити з’єднання, але цього недостатньо для ретрансляції медіа-потоків між браузерами у разі негативних відповідей від сервера STUN.

Ретранслятор NAT TURN дозволяє вузлам за NAT або брандмауером отримувати вхідні дані через з’єднання TCP або UDP. Ця можливість особливо актуальна для вузлів за симетричними NAT або міжмережевими екранами[17].

Сервер TURN має концептуальне завдання ретрансляції адрес для потоку даних між точками доступу (Рисунок 1.10).

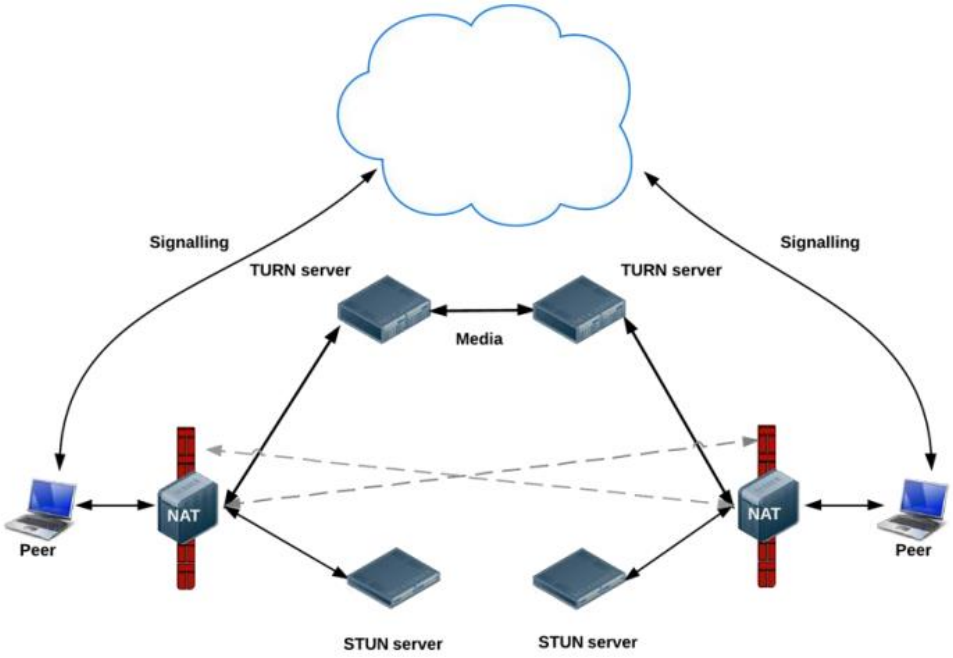

Рисунок 1.10 — Схема взаємодії браузерів з ретрансляцією потоків даних

Рекомендується використовувати протокол SSE для надійного обміну конфігураційними даними для каналів передачі медіапотоку між клієнтськими частинами програмних продуктів. Протокол SSE є найефективнішим (з точки зору швидкості та споживання ресурсів) порівняно з HttpStreaming і WebSockets.

Таким чином, не потрібно підтримувати велику потужність сервера та фінансові витрати для горизонтального масштабування, оскільки сервер використовується лише для передачі конфігурації параметрів медіа-потоку.

1.4 Висновок до розділу 1

Порівнюючи протоколи передачі даних, такі як Polling, Long Polling, потокове передавання HTTP (SSE) і WebRTC, приходимо до висновку, що збій серверної частини в концепції Polling, Long Polling, потокового передавання HTTP (SSE) призводить до повної відсутності доступу робитиінформація для всіх користувачів. Серверна частина стає вузьким місцем, коли справа доходить до надійної передачі медіаданих між багатьма користувачами.

З іншого боку, для технології WebRTC сервер використовується лише для передачі даних конфігурації каналу передачі медіа-потоку між клієнтськими частинами програмного продукту, і немає можливості передавати дані між ними. Використання протоколу WebRTC усуває необхідність серверної частини програмного продукту відповідати за прийом, конвертацію та відправку медіапотоків. Це значно знижує навантаження на серверну частину програмного продукту.

Тому, враховуючи сильні та слабкі сторони існуючих технологій, вибір технології WebRTC для реалізації передачі медіапотоку при розробці програмних продуктів є виправданим.

Вибір протоколу Server-Sent Events виправданий для забезпечення передачі конфігураційних даних, необхідних у процесі підключення за протоколом WebRTC.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМНОГО ПРОДУКТУ ДЛЯ ЗАБЕЗПЕЧЕННЯ ВІДЕОКОНФЕРЕНЦІЙ

Архітектура програмного забезпечення є фундаментальним компонентом на етапі розробки програмного продукту. Відповідно до

ANSI/IEEE 1471-2000, «Архітектура — це базова організація системи, втілена в її компонентах, їхніх зв’язках із середовищем і принципах, які керують проектуванням і розробкою системи». Це набір компонентів, які поєднуються для виконання певної функції або набору компонентів. Архітектура програмного забезпечення передбачає організацію зв’язків між програмними компонентами та зовнішнім середовищем, що містить елементи програмного продукту[18].

Відповідно до технічного аналізу в Розділі 1, використання протоколу WebRTC усуває потребу в серверних частинах програмного продукту для отримання, конвертації та надсилання медіа-потоків. Крім того, підхід до розробки програмних продуктів із використанням технології WebRTC є менш ресурсомістким з точки зору використання порівняно з іншими технологіями, розглянутими в розділі 1.

Архітектура клієнт-сервер є одним із архітектурних шаблонів, який де-факто стає стандартом у галузі розробки програмного забезпечення.

Якщо основною функцією програмного продукту є забезпечення відеозв’язку користувачів за технологією WebRTC, то основним компонентом системи є клієнтська частина.

Далі розробляється аналіз методологій проектування системи та архітектурних шаблонів, блоків архітектури програмного забезпечення та архітектури програмного продукту, що розробляється. Ми розбиваємо поведінку системи на блоки та проектуємо архітектуру системи.

2.1 Шаблони проектування

На сьогоднішній день інтернет у всьому світі зазнав стрімкого розвитку та розширення, що вимагає стандартизованих і в той же час гнучких інструментів для взаємодії з користувачем.

Патерни проектування описують типові способи вирішення типових проблем проектування програмного забезпечення. Патерни проектування вперше були описані Крістофером Олександром у книзі «Мова шаблонів:

міста, будівлі, будівництво». Він запровадив концепцію під назвою шаблони, абстрактні рішення для проблем проектування, які часто виникають, і привернув увагу дослідників в інших областях, особливо архітекторів і розробників програмного забезпечення, у середині-кінці 1980-х років[19].

Дослідження шаблонів проектування програмного забезпечення є найважливішою книгою з об'єктно-орієнтованого проектування, Шаблони дизайну: елементи багаторазового використання об'єктно-орієнтованого програмного забезпечення (Ерік Гамм, Річард Хелм, Ральф Джонсон і Джон Влісайдс (Еріх Гамма, Річард Хелм) авторів) відтоді розвинувся, Ральф Джонсон і Джон Вліссайдс, Шаблони проектування: елементи багаторазового об'єктно-орієнтованого програмного забезпечення, Addison-Wesley)[20].

Найбільша частина Шаблонів проектування – це каталог, який описує 23 шаблони проектування. Інші, новіші версії каталогу шаблонів дизайну лише розширюють цей набір і, що найважливіше, розширюють його сферу застосування до більш спеціалізованих типів завдань. Каталог багаторазових шаблонів проектування, описаних за допомогою UML.

Архітектура програмного забезпечення була побудована з використанням існуючих шаблонів проектування.

Архітектура клієнт-сервер складається з трьох основних компонентів: набір служб, які зберігають і надають інформацію, до якої вони мають доступ, набір клієнтів, які використовують набір служб, що надаються серверами, і мережа, яка має на меті створити мережу відповідно канал взаємодії між клієнтом і сервером \[21].

Сервер і клієнт незалежні один від одного і працюють паралельно. Клієнти і сервери з'єднані між собою за принципом запит-відповідь. Це забезпечує модульну гнучкість і незалежність від глобальної системи World Wide Web.

При аналізі дизайну майбутніх серверних програмних продуктів за основу було обрано концепцію архітектури Model-View-Controller, яка дозволяє розрізнити клієнтську та серверну частини програмного продукту.

Його було обрано на основі широкої сфери впливу на останні розробки програмного забезпечення клієнт-сервер.

Основна ідея принципу MVC полягає в тому, що програмний продукт у першому наближенні можна розділити на два модулі: бізнес-логіка та презентація, спрямовані на створення зв’язку між функціональністю та користувачами[22]. Схема роботи MVC зображено на рисунку 2.1.

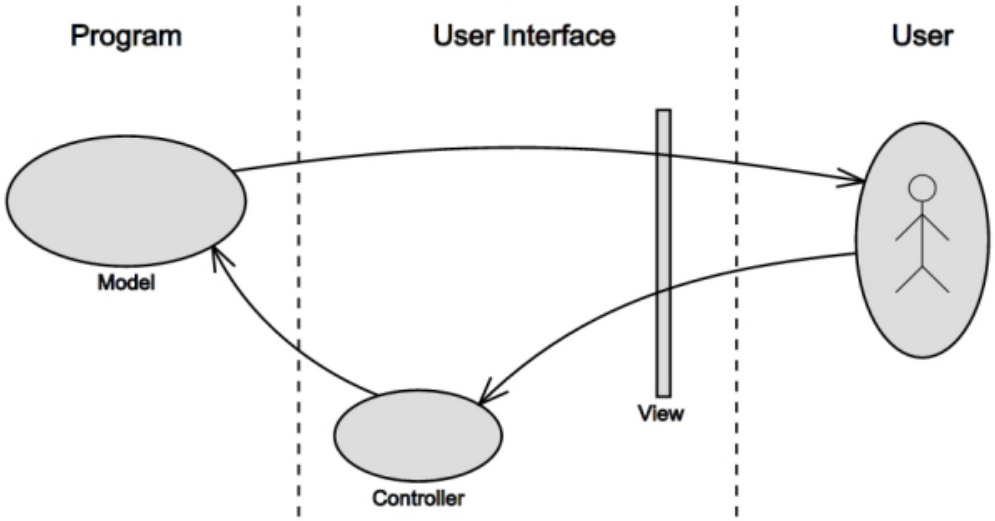

Рисунок 2.1 — Схема роботи MVC

Трирівнева клієнт-серверна архітектура, розроблена в середині 1990-х років, відокремлює рівень програми від управління даними. Окремі рівні програмного забезпечення ізольовані, а логіка застосування додатків централізована[23].

Програми середнього рівня можуть функціонувати під керуванням спеціального серверного додатку, але запуск таких програм може здійснюватися під керуванням звичайного веб-сервера. Даними керує сервер даних.

2.2 Проектування архітектури клієнтської частини програмного продукту з використанням технології WebRTC

Окремі технології WebRTC використовують можливості браузерів для встановлення зв'язку між клієнтами програмного продукту. Тому варто розглянути архітектуру структури клієнтської частини, оскільки саме клієнтська частина висуває вимоги до серверної частини програмного забезпечення.

Щоб забезпечити постійний зв'язок, клієнтська частина програмного продукту повинна завжди зберігати в пам'яті об'єкт RTCPeerConnection, який є інтерфейсом каналу передачі медіаданих між браузерами. Враховуючи це обмеження та необхідність підтримувати інтерактивність клієнтської частини, потрібно створити архітектуру для клієнтської частини, яка не дозволяє перезавантажувати сторінку браузера.

Рішенням цієї проблеми є вибір технології розробки клієнтської частини за принципом SPA (Single Page Application), що використовує одну HTML-сторінку як оболонку для нижніх частин моделі DOM.

Отже, якщо бізнес-модель і модель відображення змінюються, зміниться лише підмодель сторінки[24].

Розробники клієнтських частин програмного забезпечення, які функціонально не статичні, використовують мову програмування JavaScript завдяки повній підтримці браузером мови програмування JavaScript[25].

Стандартом, який визначає мову програмування JavaScript і її версії, є ECMAScript[26].

Перша версія стандарту ES1. Наразі стандартна версія ES5, випущена в 2009 році, має повну підтримку браузера[25].

В даний час існують такі версії стандарту[27]:

1. ECMAScript 1 1997 — перший стандарт який з'явився;

2. ECMAScript 2 1998 — не значні зміни версіїї;

3. ECMAScript 3 1999 — додані підтримка try/catch та Regular Expressions;

4. ECMAScript 4 — дана версія була не випущена;

5. ECMAScript 5 2009 — додано “strict mode”,підтримка JSON;

6. ECMAScript 6 2016 — додані ключові слова let, const;

7. ECMAScript 7 — доданий exponential operator;

8. ECMAScript 8 2017 — додані shared memory оброрбники та async-функції;

9. ECMAScript 9 2018 — додані Promise.finally().

Одним із найпопулярніших фреймворків для побудови клієнтської частини на основі принципів SPA є програмування на JavaScript із підтримкою специфікації ES5 і синтаксичною схожістю з ключовими поняттями в специфікаціях ES6-ES8. Це бібліотека React (версія 5+).

Версія 5+ бібліотеки React використовує мову TypeScript, розширення специфікації JavaScript ES6-ES8. Створюйте динамічні інтерактивні веб-сайти, які підтримують більшість мобільних пристроїв і браузерів.

Основними будівельними блоками фреймворку є модулі React (NgModules), які забезпечують контекст компіляції для компонентів. Модуль NgModules аналізує пов’язаний код і об’єднує його в набір функцій, які складають додаток React. Клієнтське програмне забезпечення, створене на основі бібліотеки, має принаймні одну декларацію NgModule, яка налаштовує розгортання клієнтської програми. Основний модуль об’єднує інші функціональні модулі бібліотеки.

Компоненти визначають компоненти відображення, визначені в мовах розмітки HTML і CSS, щоб відтворювати та змінювати компоненти відображення відповідно до написаної бізнес-логіки та пов’язаних моделей даних.

Компоненти використовують служби (Service) React, які надають спеціальні функції, не пов’язані з представленнями. Сервіси бібліотеки можуть надавати можливість агрегувати сервіси в компоненти за допомогою

концепції впровадження залежностей. Це робить програмний код модульним, ефективним і придатним для повторного використання[28].

Компоненти та служби визначаються класами, які визначають метадані за допомогою декораторів.

Основні блоки бібліотеки React забезпечують гнучкий інструментарій для створення клієнтської частини програмних продуктів. Клієнтська частина програмного забезпечення, створена за допомогою бібліотеки, повністю відповідає вимогам односторінкових програм і забезпечує гнучкий інструментарій для створення клієнтської частини програмних продуктів.

Після того як обрано технології для створення основи клієнтської частини програмного продукту, потрібно визначити вимоги до використання технології WebRTC як клієнтського компонента, що розробляється.

Першим блоком для побудови відеоконференції є побудова двостороннього зв’язку клієнт-клієнт за допомогою технології WebRTC (Рисунок 2.2).

Програма охоплює двох користувачів системи: Alice і Bob намагаються встановити відеозв'язок один з одним.

Клієнт Alice входить у систему, використовуючи своє ім’я для входу та пароль, і підписуються на події сервера за допомогою технології Server-Sent Events. Клієнт Bob також проходить автентифікацію в системі та підписується на події сервера (авторизація може здійснюватися у різний термін).

Клієнт Alice вирішує надіслати пропозицію розмови клієнту Bob та ініціює запит розмови з іншим клієнтом за допомогою веб-інтерфейсу.

Запит надсилається на сервер (частина серверної частини програмного продукту). Сервер аналізує вхідні запити, щоб побачити, чи є одержувач запиту в системі.

Потім клієнт Bob підтверджує запит на розмову з Alice та надсилає нове повідомлення підтвердження на сервер.

На данному етапі ми можемо стверджувати, що обидва клієнти (Bob і Alice) знаходяться онлайн в системі, авторизовані, бачили розмови один

одного і мали можливість спілкуватися один з одним. Після цього кроку зв’язок між браузером Alice та браузером Boba ініціюється за допомогою програмної веб-частини.

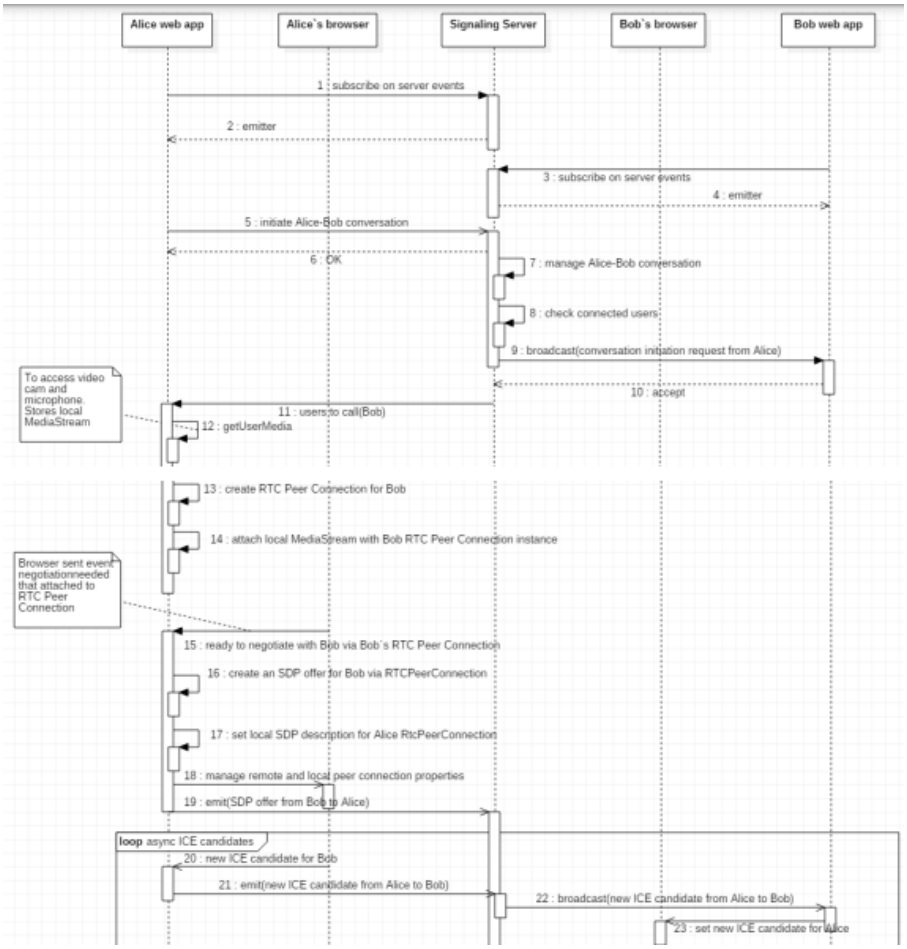

Рисунок 2.2 — Діаграма послідовностей налагодження каналу зв’язку за технологією WebRTC, частина 1

Метаінформація про налаштування обох браузерів і каналу передачі відео та аудіопотоків клієнта повинна бути інкапсульована в об’єкт клієнтської частини програмного продукту. Отримання авторизації в системі та дозволу обох клієнтів на поширення інформації (рисунок 2.3).

Архітектурним принципом SPA є гарантія збереження налаштувань міжбраузерного каналу передачі без необхідності перезавантажувати веб-сторінку та без необхідності повторювати процес налаштування каналу передачі даних під час взаємодії з програмним продуктом.

Наступними кроками для налаштування каналу даних між браузерами є надсилання запитів WebRTC, відповідей і налаштувань кандидатів ICE. Це збирає інформацію про налаштування вашого браузера та налаштування доступу.

З цих етапів можна зробити висновок, що нам потрібно створити окремий веб-компонент для клієнтської частини та налаштувати асинхронну підписку на події сервера.

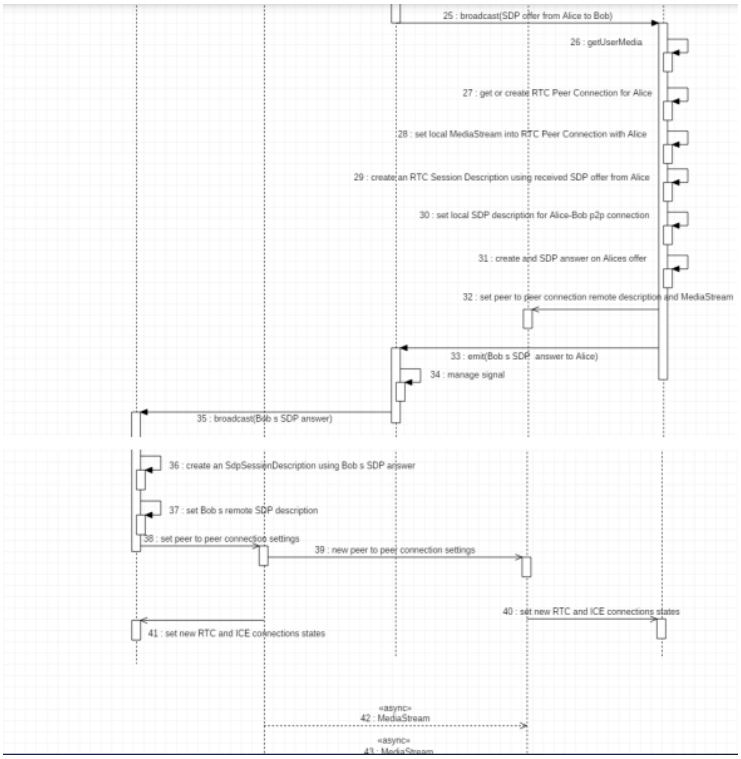

Рисунок 2.3 — Діаграма послідовностей налагодження каналу зв’язку за технологією WebRTC, частина 2

Щоб забезпечити відеоконференцію, описаний вище сценарій слід реалізувати для всіх користувачів системи, які беруть участь відеоконференції. Таким чином, у контексті однієї відеоконференції канал передачі даних налаштовується для кожного учасника розмови за допомогою браузерів кожного з інших учасників розмови.

Інформація встановлює архітектурні вимоги та обмеження для клієнтської частини програмного забезпечення. На рисунку 2.4 показана

ієрархія функціональних блоків, відповідальних за налаштування з'єднання браузер-браузер. Кожен об’єкт WebRTC інкапсулює спосіб створення каналу між браузерами та його життєвий цикл, а також зберігає медіа-потік для відображення блоків відео на рівні відображення.

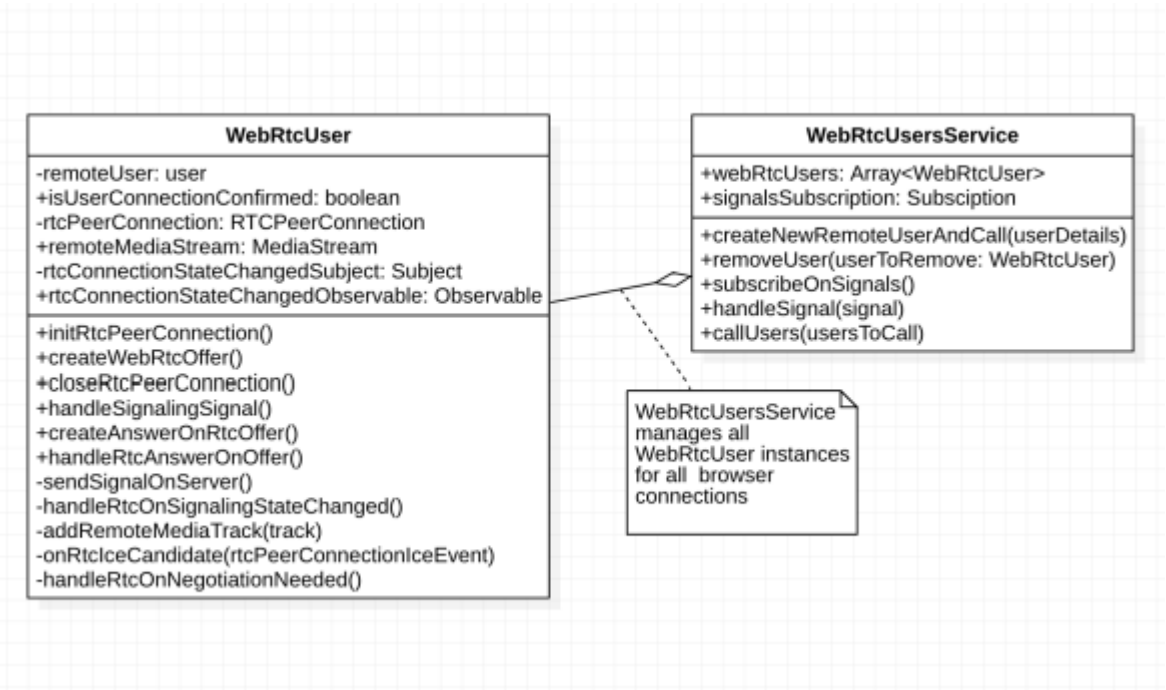

Рисунок 2.4 — Взаємодія класів, що відповідають за налаштування WebRTC з’єднань

Служба WebRtcUsersService — це служба в термінології бібліотеки React, і її об’єкт впроваджується в глобальну область веб-додатку клієнта. Таким чином, уся логіка для налаштування міжбраузерного каналу зв’язку інкапсульована в цих двох класах React.

Основною функцією сервісу є управління всіма користувачами, підключеними до поточного користувача. На рисунку 2.5 схематично показана архітектура клієнтської частини програмного продукту у вигляді пакетної діаграми.

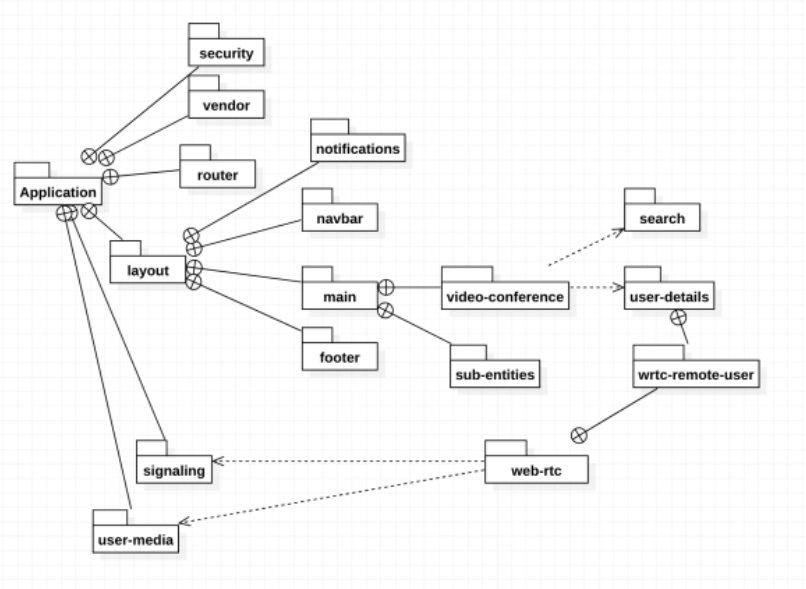

Рисунок 2.5 — Діаграма пакетів клієнтської частини програмного продукту

Кожен пакет містить відповідні модулі, служби, компоненти, файли CSS і HTML (якщо є модель відображення), а також служби маршрутизації сторінок. Структурування за допомогою пакетів дозволяє створювати модульні системи, які можна модифікувати за потреби.

Тому була проведена декомпозиція клієнтської частини програмного продукту для визначення основних пакетів і класів, відповідальних за налагодження каналів зв’язку для забезпечення відеоконференцзв’язку.

2.3 Проектування архітектури серверної частини програмного продукту з використанням технології WebRTC

Сучасні технологічні розробки програмного забезпечення значною мірою вплинули на розвиток методології та основних принципів пошуку інформації в Інтернеті. Це сприяло розвитку середовищ і методів програмування WEB.

За словами М. Фаулера, «Вибір мови програмування визначається виключно смаком читача. У цьому сенсі Java видається найбільш привабливою»[29].

Цю ж думку щодо використання Java як мови програмування підтримує Н. В'язовик. він сказав: Будь-яка платформа без модифікацій (кросплатформенність). Також відомо, що Java орієнтована на Інтернет»[30].

Вибір у роботі базувався на перевагах мови програмування Java

Під час аналізу архітектури майбутніх компонентів серверного програмного забезпечення за основу було обрано концепцію архітектури Model-View-Controller (MVC), що дозволяє розрізнити клієнтську та серверну частини програмного продукту. Його було обрано на основі його широкої сфери впливу на останні розробки програмного забезпечення клієнт-сервер. Приклад взаємодії компонентів MVC між собою наведено на рисунку 2.6.

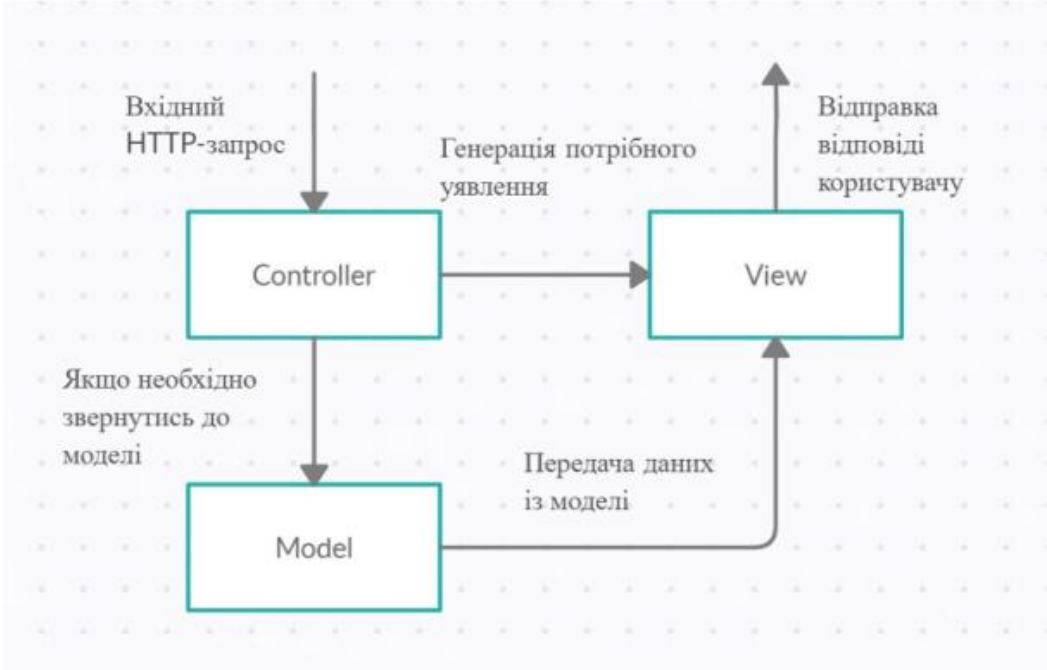

Рисунок 2.6 — Взаємодія компонентів MVC між собою

Основними обов'язками серверної частини програмного продукту є:

— функція сервера сигналів;
— авторизація/Сертифікація;
— керуйте активними відеоконференціями;
— логіка обміну повідомленнями між користувачами системи.

Частина серверної частини програмного продукту, основна роль сервера сигналів полягає в надсиланні та отриманні подій для користувачів системи. Кожен клієнт надсилає на сервер запит на подію HTTP. Сервер аналізує та обробляє сигнали, отримані від користувачів, і при необхідності надсилає відповідні події кожному учаснику в процесі створення відеоконференції.

Технологія WebRTC передбачає побудову каналів зв'язку між браузерами. Сервер сигналізації використовується для передачі конфігураційних даних підключення. У цій роботі рекомендовано використовувати технологію Server-Sent Events для досягнення компромісу між швидкістю системи та навантаженням на сервер і мережу.

За обробку сигналів повинні відповідати існуючі служби управління потоком відеоконференцій SSE і управління станом, а також стан з'єднання пари клієнт-клієнт.

Після визначення понять зв’язку клієнт-сервер і сервер-клієнт нам потрібно розкласти серверну частину програмного продукту.

Для використання функціональних можливостей системи клієнти повинні бути зареєстровані. Для цього вам необхідно буде пройти етап реєстрації та ввести свої дані (ім’я, прізвище, дані поштової скриньки та логін). Після реєстрації на поштову скриньку користувача буде надіслано лист з гіперпосиланням для активації облікового запису.

Тому було запроваджено двофакторну реєстрацію. В системі та через поштові скриньки.

Після підтвердження реєстрації користувач матиме можливість пройти автентифікацію в системі. Після успішної аутентифікації в системі користувач отримує доступ до відповідних модулів функціональності системи в залежності від ролі користувача. Вмикає принципи безпеки для даних користувача, історії повідомлень і відеодзвінків.

Автентифіковані користувачі мають можливість заводити нові «дружні стосунки» з іншими користувачами системи та після подання відповідних запитів брати участь у відеоконференціях з іншими користувачами системи.

Усі дані користувачів, комунікаційних пар, повідомлень і відеоконференц-кімнат зберігаються в базі даних (Рисунок 2.7).

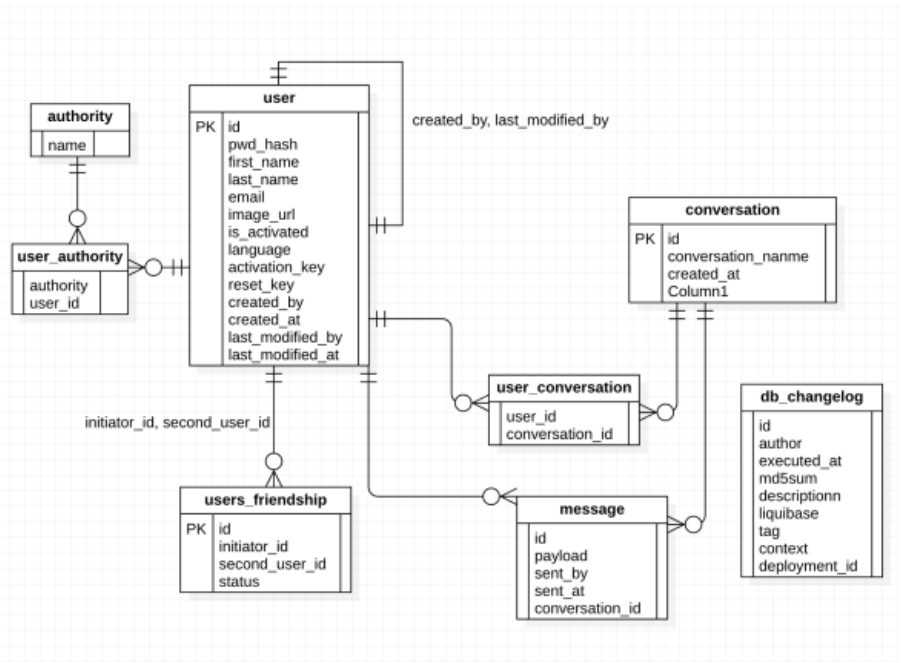

Рисунок 2.7 — Entity Relationship діаграма сутностей БД

Після підтвердження участі в «бесіді» користувач має можливість подати заявку на створення відеоконференції з усіма учасниками «бесіди». На цьому етапі сервер повинен обробити запит, перевірити, чи вибрана розмова вже активна, створити її та зберегти в оперативній пам’яті.

Після цього система повинна асинхронно перевірити активні з’єднання з кожним учасником/парою учасників конференції. Якщо пари не знайдено, система створює пару ініціатор-учасник в RAM і переходить до наступного кроку. Якщо учасник онлайн, система використовує службу сигналізації для надсилання запиту на підтвердження відеоконференції.

Якщо учасник відеоконференції відсутній, система створить для користувача чергу повідомлень (якщо її немає) і збереже повідомлення. Після аутентифікації клієнта в системі клієнтська частина програмного продукту автоматично створює підписку на події сервера. Після успішної підписки на

подію сервер аналізує чергу повідомлень користувача та надсилає відповідне сигнальне повідомлення.

Коли інший користувач приймає запит на відеоконференцію, клієнтська частина програмного продукту повинна автоматично відправити відповідний сигнал на сервер сигналів.

Потім сервер сигналів аналізує, яким користувачам потрібно налаштувати канал підключення, і ініціює обмін сигналами конфігурації кожному користувачеві за допомогою протоколу WebRTC.

Таким чином, сервер сигналів і бізнес-логіка в серверній частині програмного продукту використовуються для ініціювання конфігурації каналу між браузерами для всіх активних учасників відеоконференції.

Після налаштування каналу зв’язку сервер більше не бере участь у передачі медіа-потоків між користувачами відеоконференцзв’язку. Тож навіть у разі технічної несправності сервера встановлене з’єднання не перерветься, а швидкість і якість залишаться на тому ж рівні. Індикатори потоку відео та аудіо.

У документі також пропонується використовувати реляційну базу даних Postgres і структуру сценаріїв, а також версії схеми бази даних Liquibase.

Liquibase — це бібліотека з відкритим вихідним кодом, яка не залежить від бази даних і призначена для відстеження, керування та застосування змін схеми бази даних. Ця бібліотека була створена для полегшення моніторингу та модифікації схем бази даних[31].

Тому була розроблена архітектура основних блоків серверної частини програмного забезпечення.

2.4 Висновок до розділу 2

У цьому розділі обґрунтовано вибір концепцій архітектури програмного комплексу, за основу програмного продукту, що розробляється, обрано шаблон проектування GOF та шаблон архітектури Model-ViewController.

Враховано особливості використання технології WebRTC для побудови клієнтської частини програмного продукту, побудовано схему взаємодії клієнтської частини програмного продукту, інкапсульовано логіку взаємодії за WebRTC та реалізовано програмне забезпечення.

Визначає обов’язки серверної частини програмного продукту, визначає основний напрямок впровадження, вимоги до серверної частини програмного продукту, схему бази даних.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ДЛЯ ЗАБЕЗПЕЧЕННЯ ВІДЕОКОНФЕРЕНЦІЙ

3.1 Програмна реалізація серверної частини програмного забезпечення

Після визначення мови програмування та вибору шаблонів програмування та глобальних архітектурних концепцій необхідно було провести аналіз та вибрати структуру технології програмування.

Після визначення базової основи потрібно розробити рівень для серверної частини програмного продукту. Архітектурно серверна частина вимагає таких рівнів[32]:

— рівень контролера відповідає за похідні інтерфейси (надані інтерфейси в термінології М.Фаулера). Рекомендовано використовувати Spring Core + Spring MVC;

— рівень бізнес-логіки — об’єднує класи обслуговування бізнес-логіки, керування транзакціями та сесіями (Spring Core). Поєднання клієнтської та серверної частин програмного забезпечення є концепцією MVC.

— робочий рівень з базовими даними (persistence layer) — рівень, що відповідає за обмін даними між серверною частиною програмного продукту та засобом зберігання даних (базою даних)[33].

Необхідною частиною технології WebRTC є сигнальний сервер, який відповідає за обмін конфігураційними даними між клієнтськими частинами програмного продукту. Основною властивістю сервера сигналів є швидкість передачі сигналу. Тому рекомендується мати постійний канал передачі даних типу сервер-клієнт, призначений для досягнення швидкості надсилання та отримання сигнальних повідомлень.

Використання протоколу передачі даних WebSocket передбачає побудову єдиного каналу зв'язку між компонентами системи. Це головна вимога протоколу. Ключовою особливістю протоколу WebSocket є використання єдиного каналу зв’язку для обробки повідомлень сервер-клієнт і клієнт-сервер[34].

При побудові сервера сигналів впровадження технології WebSocket передбачає нарощування додаткового навантаження на мережу та витрати на ресурси сервера. Це стає зайвим через кількість сигналів, надісланих від клієнта до сервера. Іншим недоліком використання технології WebSocket як каналу сигналізації є складність налаштування реалізації протоколу як у серверній, так і в клієнтській частинах програмного забезпечення[37].

У цій роботі пропонується використовувати технологію передачі подій, надісланих сервером. Відповідно до цієї концепції, клієнтська частина програмного продукту підписується на події сервера після успішної аутентифікації в системі.

Тому за допомогою технології Server-Sent Events клієнтська частина підписується на повідомлення сервера. Він дешевий з точки зору показників навантаження на мережу і має властивість отримувати серверні події відразу від сервера безпосередньо до клієнта[36].

Події клієнт-сервер надсилаються на сервер сигналів як запити REST HTTP.

Після того, як визначилися з основною технологією, вам необхідно вибрати концепцію реалізації технології Server-Sent Events. Основні концепції впровадження:

— створення власного обробника HttpServletResponse, висхідного коду для обробки потоку.

— використання структури та результату проекту Reactor у формі Flux<ServerSentEvent<?>>.

— використовуйте структуру SseEmitter Spring.

Використання обгортки SSE Project Reactor вимагає перевизначення логіки FluxSink, щоб можна було підтримувати канал зв’язку без повторної підписки передплатників. Кожна оболонка потоку повинна створювати EventListeners у межах серверної частини програмного продукту.

Проблема цього підходу полягає в постійному управлінні EventListeners, час життя яких не обмежений відкритими каналами передачі даних. Таким

чином, навіть якщо сам потік даних уже вважається закритим як на стороні сервера, так і на стороні клієнта програмного забезпечення, в оперативній пам’яті сервера є об’єкт EventListener, який виконує бізнес-процес потоку даних.

Функції фреймворку Spring (клас SseEmitter) дають змогу ефективніше використовувати ресурси оперативної пам’яті та писати чистіший програмний код. Отже, у процесі підписки на стороні клієнта програмного продукту RestController поверне об’єкт SseEmitter для певного потоку.

Завдяки цьому підходу управління об’єктами та життєвий цикл стають відповідальністю рівня бізнес-логіки в серверній частині програмного забезпечення.

Отже, отримавши можливість налаштувати клієнтський зв’язок, є змога налаштувати канал передачі даних між двома клієнтами системи.

Наступний етап — програмна реалізація алгоритмів створення та керування відеоконференціями між багатьма користувачами системи. Відзначимо, що в концепції системи, що розробляється, відеорозмова «один на один» також вважається відеоконференцією між двома користувачами системи.

Подібно до того, як побудова шляху передачі даних за допомогою технології WebRTC розподіляє клієнтів між відправниками та отримувачами, необхідно розробити серверну бізнес-логіку, яка розподіляє учасників відеоконференції між відправниками та отримувачами. Ми пропонуємо споживачам і розподіл цих пар відповідно до їх технічної та логічної складової.

Згідно з компонентом бізнес-логіки, один користувач приймає (або відхиляє) запрошення створити відеоконференцію один раз (для однієї відеоконференції). В іншому випадку, якщо у відеоконференції бере участь велика кількість учасників, клієнту доведеться прийняти або відхилити запрошення створити відеоконференцію для кожного учасника, а також прийняте правило створення інтуїтивно зрозумілого інтерфейсу проти.

Технологія WebRTC, з іншого боку, підтримує чіткий розподіл клієнтів за ініціатором, приймаючи пропозиції щодо створення каналів і відповідаючи власними конфігураційними даними.

Крім того, однією з ключових проблем, яка може виникнути, є початок відеоконференції, коли інші учасники перебувають у режимі офлайн. Таким чином, програмні продукти можуть реагувати непередбачуваним чином, якщо існуючі логічні конфлікти та можливі поведінкові ситуації для користувачів системи не вирішені.

Враховуючи вищезазначене, для вирішення цих проблем пропонуються наступні програмні методи.

Система тимчасово зберігає чергу повідомлень користувача протягом обмеженого часу. Це було б надіслано, якби користувач був онлайн під час створення повідомлення.

Клієнтська частина програмного продукту підписується на події сервера після аутентифікації користувача в системі. При цьому генерується повідомлення про те, що відповідний користувач онлайн.

Після того, як система отримує подію про те, що статус користувача змінився на «онлайн», вона аналізує наявність черги повідомлень користувача та використовує сервер сигналів для надсилання повідомлень у синхронному режимі. Запити на підтвердження відеоконференції також зберігаються в черзі повідомлень.

Якщо користувач системи вже автентифікований у системі та намагається створити відеоконференцію, система перевірить, чи активна відеоконференція для вказаного ідентифікатора.

Якщо вона визначає, що активна відеоконференція за цим ідентифікатором не знайдена, система отримує дані про розмову («розмова» схожа на розмову або відеоконференцію) з бази даних і створює нову активну розмову, у пам’яті та зберігає всіх потенційних учасників розмови.

Крім того, зберігає пари користувачів, які вже встановили з’єднання або встановили шлях передачі без участі в розмові. Це необхідно, щоб система не

намагалася встановити з’єднання для кожної розмови у випадку двох активних відеоконференцій з однією парою користувачів. Тому навіть у описаній вище ситуації клієнтська частина програмного продукту має один канал передачі даних на учасника відеоконференції.

Після створення активного об’єкта розмови (в оперативній пам’яті) система аналізує системного користувача, пов’язаного з поточною розмовою.

У цьому списку користувачів система фільтрує пари користувачів, які вже підключені зі статусом «встановлення підключення» або «підключено».

Відфільтрований список учасників відеоконференції можна розділити на дві групи залежно від їх статусу «онлайн» або «офлайн».

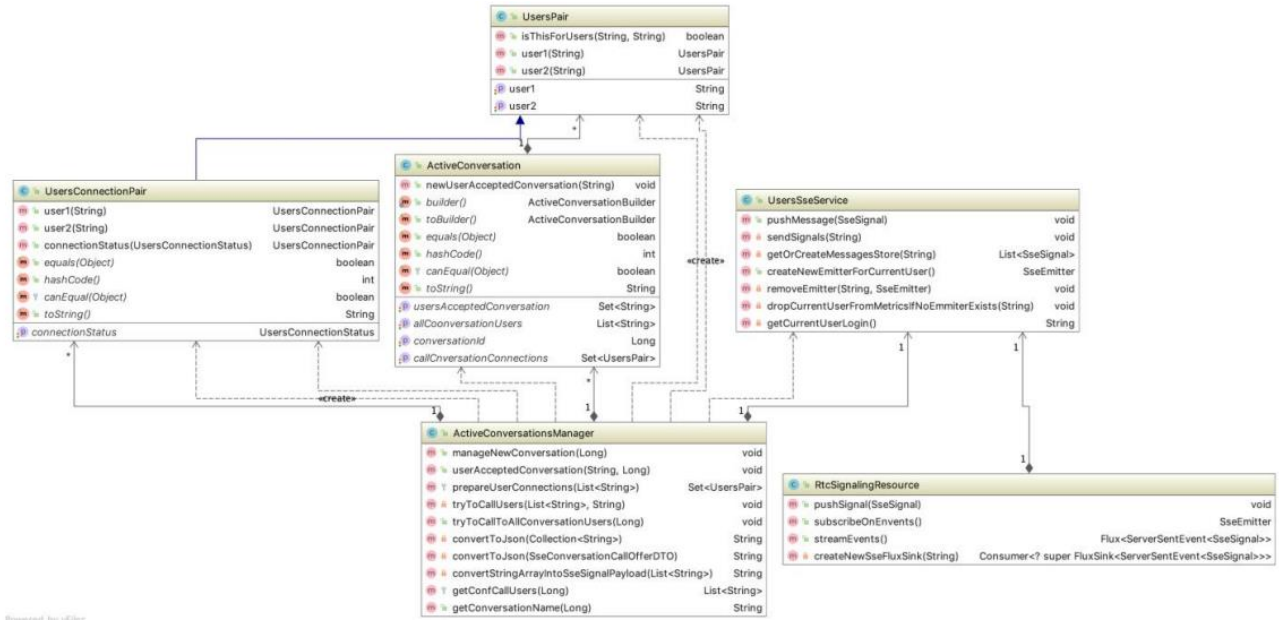

Рисунок 3.1 — Діаграма класів управління відеоконференціями

Для кожного учасника відфільтрованого списку зі станом «онлайн» система використовує розпаралелений метод для надсилання відповідного повідомлення користувачеві з підтвердженням розмови.

На рисунку 3.1 показана діаграма класів серверного компонента, відповідального за створення та керування відеоконференціями, прив’язаних до методів сигналізації.

Після вирішення проблем створення сигнального сервера, створення доменної моделі та реалізації логіки, яка керує налаштуванням та життєвим циклом відеоконференції, можна буде налаштувати канал передачі даних за технологією WebRTC.

Однак використання цього каналу вимагає, щоб клієнтська частина отримувала локальні медіа-потоки (відео та аудіо) за допомогою доступної технології браузера.

Інструментом для цього є Navigator.getUserMedia() для отримання дозволу користувача на отримання даних і їх обробку в клієнтській частині. Однак обмеженнями цього методу є наявність сертифікатів безпеки SSL і префікс визначення HTTP для протоколу https.

Для цього використано команду створення самопідписаного сертифіката (Рисунок 3.2).

```
keytool -genkey -alias <your-application> -storetype PKCS12 -keyalg RSA -keysize 2048 -keystore keystore.p12 -validity 3650
```

Рисунок 3.2 — Команда для генерації самопідписаного SSL сертифікату

Результатом є PKCS#12 із відповідним сховищем ключів. Наступним етапом налаштування є додавання файлу конфігурації YML із конфігурацією, показаною на рисунку 3.3.

```
server:
    port: 8080
    ssl:
        key-store: classpath:config/tls/keystore.p12
        key-store-password: password
        key-store-type: PKCS12
        key-alias: foo
        # The ciphers suite enforce the security by deactivating some old and deprecated SSL cipher, this list was tested against SSL Labs (https://www.ssllabs.com
        ciphers: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

Рисунок 3.3 — Приклад конфігурації TLS налаштувань YML файлу

Після успішного налаштування сховища ключів і відповідних сертифікатів потрібно надати карту перетворення претензій з http на https.

Для цього у фільтрі запиту створено додаткове посилання (Рисунок 2.9).

```
import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class HttpsEnforcer
    implements Filter {

    public static final String X_FORWARDED_PROTO = "x-forwarded-proto";
    private FilterConfig filterConfig;

    public void init(FilterConfig filterConfig) { this.filterConfig = filterConfig; }

    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse,
                         FilterChain filterChain) throws IOException, ServletException {

        HttpServletRequest request = (HttpServletRequest) servletRequest;
        HttpServletResponse response = (HttpServletResponse) servletResponse;

        if (request.getHeader(X_FORWARDED_PROTO) != null) {
            if (request.getHeader(X_FORWARDED_PROTO).indexOf("https") != 0) {
                String pathInfo = (request.getPathInfo() != null) ? request.getPathInfo() : "";
                response.sendRedirect( s: "https://" + request.getServerName() + pathInfo);
                return;
            }
        }

        filterChain.doFilter(request, response);
    }
```

Рисунок 3.4 — Приклад конфігурації редіректу запиту з http на https

Таким чином, реалізовано основні компоненти серверної частини, необхідні для створення відеоконференцій за технологією WebRTC.

Слід запровадити гнучку конфігурацію бази даних для зберігання даних про конфігурацію користувачів і учасників відеоконференцій, а також для забезпечення масштабування, конфігурації, зміни та міграції програмних продуктів.

Фреймворкова бібліотека Liquibase була обрана як засіб динамічного створення схеми та початкових даних бази даних. Усі зміни схеми бази даних зберігаються у файлах XML, YAML та JSON, ідентифікованих комбінацією ідентифікатора та тегів автора, а також назвою самого файлу[37].

Список усіх застосованих змін зберігається в кожній базі даних, якою маніпулюють у кожному оновленні бази даних, щоб визначити, які зміни потрібно застосувати. Ієрархія файлів Liquibase показана на рисунку 3.5 Liquibase.

Після визначення сутностей бази даних вони об’єднуються у файл master.xml і стають виконуваними (подібно до файлів зміни даних).

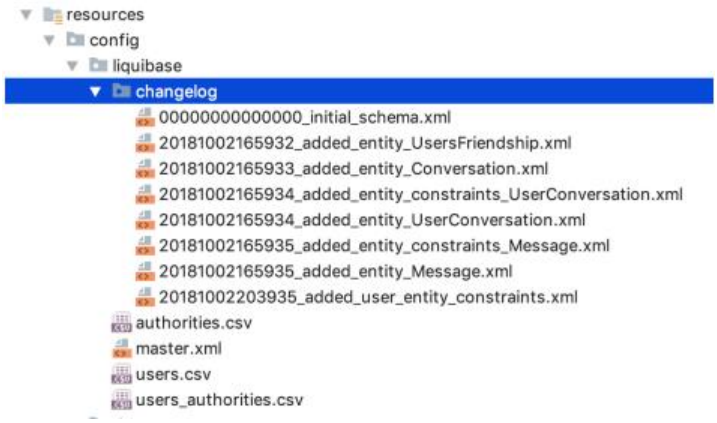

Рисунок 3.5 — Ієрархія файлів змін бази даних Liquibase

Під час першого запуску сервера, якщо існує порожня база даних, Liquibase перевіряє журнали змін проекту асинхронно до основного потоку сервера та виконує їх у порядку, визначеному командою розробників.

Конфігурацію програмного забезпечення запуску Liquibase за допомогою середовища Spring можна реалізувати, як показано на рисунку 3.6.

```
@Configuration
public class LiquibaseConfiguration {

    private final Logger log = LoggerFactory.getLogger(LiquibaseConfiguration.class);

    private final Environment env;

    public LiquibaseConfiguration(Environment env) { this.env = env; }

    @Bean
    public SpringLiquibase liquibase(@Qualifier("taskExecutor") TaskExecutor taskExecutor,
            DataSource dataSource, LiquibaseProperties liquibaseProperties) {

        // Use liquibase.integration.spring.SpringLiquibase if you don't want Liquibase to start asynchronously
        SpringLiquibase liquibase = new AsyncSpringLiquibase(taskExecutor, env);
        liquibase.setDataSource(dataSource);
        liquibase.setChangeLog("classpath:config/liquibase/master.xml");
        liquibase.setContexts(liquibaseProperties.getContexts());
        liquibase.setDefaultSchema(liquibaseProperties.getDefaultSchema());
        liquibase.setDropFirst(liquibaseProperties.isDropFirst());
        liquibase.setChangeLogParameters(liquibaseProperties.getParameters());
        if (env.acceptsProfiles(JHipsterConstants.SPRING_PROFILE_NO_LIQUIBASE)) {
            liquibase.setShouldRun(false);
        } else {
            liquibase.setShouldRun(liquibaseProperties.isEnabled());
            log.debug("Configuring Liquibase");
        }
        return liquibase;
    }
}
```

Рисунок 3.6 — Програмна реалізація запуску на виконання Liquibase обробника

Доступ до фізичної бази даних описано в конфігурації YML з використанням середовища Spring Data (Рисунок 3.7).

```yaml
spring:
    devtools:
        restart:
            enabled: false
        livereload:
            enabled: false
    datasource:
        type: com.zaxxer.hikari.HikariDataSource
        url: jdbc:postgresql://localhost:5432/webrtcHorbenkoProto
        username: postgres
        password: 19069524
        hikari:
            auto-commit: false
    jpa:
        database-platform: io.github.jhipster.domain.util.FixedPostgreSQL82Dialect
        database: POSTGRESQL
        show-sql: false
        properties:
            hibernate.id.new_generator_mappings: true
            hibernate.connection.provider_disables_autocommit: true
            hibernate.cache.use_second_level_cache: false
            hibernate.cache.use_query_cache: false
            hibernate.generate_statistics: false
    liquibase:
        contexts: prod
```

Рисунок 3.7 — Приклад опису конфігурації доступу до БД

Тому розроблено алгоритми створення відеоконференцій, ключові компоненти обробки даних та налаштування безпеки використання технології WebRTC. Було побудовано концепцію створення, адаптації та модифікації моделі схеми бази даних і персистентного рівня серверної частини програмного продукту.

3.2 Програмна реалізація клієнтської частини програмного продукту

З огляду на обрану технологію передачі медіа-потоків між користувачами, було обрано концепт Single Page Application та фреймворк Angular 6.2.2, що задовольняє вимогам технології WebRTC.

Основними відповідальностями клієнтської частини програмного продукту є:

— створення та управління життєвим циклом каналів передачі даних за технологією WebRTC;

— управління медіа-потоками локальними та медіа-потоками учасників відеоконференції;

— відображення медіа-потоків.

Після створення каркасу програмного продукту, основних модулів, що відповідають за верстку, модулів, компонентів та сервісів системи, що

відповідають за налаштування безпеки та роутингу між сторінками, пропонується звернути увагу на створення компонентів, що відповідають за сигналізацію подій сервер-клієнт та клієнт-сервер, оскільки налаштування створення каналів передачі даних за WebRTC буде виконано саме з використанням цих компонентів.

Передача сигналів клієнт-сервер виконується з використанням @angular/common/http/ HttpClient класу, що інтегровано в якості модуля фреймворку Angular. Дані сигнали передаються на сервер за HTTP протоколом з відповідністю концепту REST.

Більш складною є імплементація прийому сигналів із сервера. Було попередньо обрано протокол передачі даних Server-Sent Events у якості каналу передачі подій сервера. Клієнтською частиною програмного продукту створюється підписка на події сервера, що повинна існувати постійно (для одної вкладки браузера). Підписка повинна створюватись після успішної автентифікації системи та знищуватись після операції logout.

Проблема може виникнути у разі збереження даних cookie барузером. У цьому випадку, після створення вкладки браузеру та переходу, наприклад, на сторінку відеоконференцій, ініційовану з історії, повна автентифікація виконана не буде. Для коректності роботи системи, підписка на події сервера повинна відбуватися після перевірки двох тригерів:

— клієнт автентифікувався у системі;

— веб-сторінка розгорнута браузером.

Для обробки автентифікації було створено об’єкт Subject модуля rxjs та створення об’єкта типу Observable у сервісі автентифікації клієнтської частини програмного продукту.

Для обробки другого тригеру (веб-сторінка розгорнута браузером), було проаналізовано програмний компонент на наявність постійних HTML-шаблонів та відповідних до представлення компонентів.

Визначено, що модуль верхнього меню (navbar.component, navbar.module, navbar.html та інші) фізично розгортуються один раз у розрізі

життєвого циклу існування веб-частини програмного продукту в одному вкладенні браузеру.

Бібліотека React передбачає додаткові методи життєвого циклу компонентів — Anngular Lifecycle Hooks. Потребує уваги hook ngOnInit, тіло якого викликається в процесі розгортки компоненту та може бути реалізовано розробниками.

Саме під час розгортки модулю navbar, виконується надсилання події про розгортку сторінки, підписчиком якої є сервіс SseSignalingService, який, у свою чергу, виконує перевірку наявності підписки на події сервера та стану автентифікації користувача.

SseSignalingService є класом та сервісом у термінології Angular. Це гарантує існування одного об’єкту даного класу та можливість ін’єкції цього сервісу в усі необхідні модулі, компоненти та сервіси клієнтської частини програмного продукту.

Завдяки описаній вище реалізації, отримано гарантію створення підписки на події сервера навіть за умови відкриття дублікату вкладки браузеру, фунцкії remember me та збереження cookie браузером.

Саме сервіс SseSignalingService відповідає за сигналізацію клієнт-сервер та сервер-клієнт. Сервіс створює об’єкт типу Subject та об’єкт типу Observable із об’єкта типу Subject (Рисунок 3.8).

```
private newSseSignalingMessageSubject = new Subject<SseMessage>();
newSseSignalingMessageObservable$ = this.newSseSignalingMessageSubject.asObservable();
```

Рисунок 3.8 — Створення об’єктів для можливості реалізації шаблону GOF Observer

Ці два об’єкти реалізують шаблон проектування Observable, завдяки чому компоненти системи можуть провести ін’єкцію сервісу та оформити підписку на отримання подій сервера, прийняті каналом SSE.

Таким чином, було реалізовано логіку сигналізації подій та надано можливість підписки на нові сигнали із сервера та інструмент надсилання подій клієнта на сервер.

Наступним етапом розробки є реалізація логіки створення та управління каналами передачі даних за протоколом WebRTC (Рисунок 3.9).

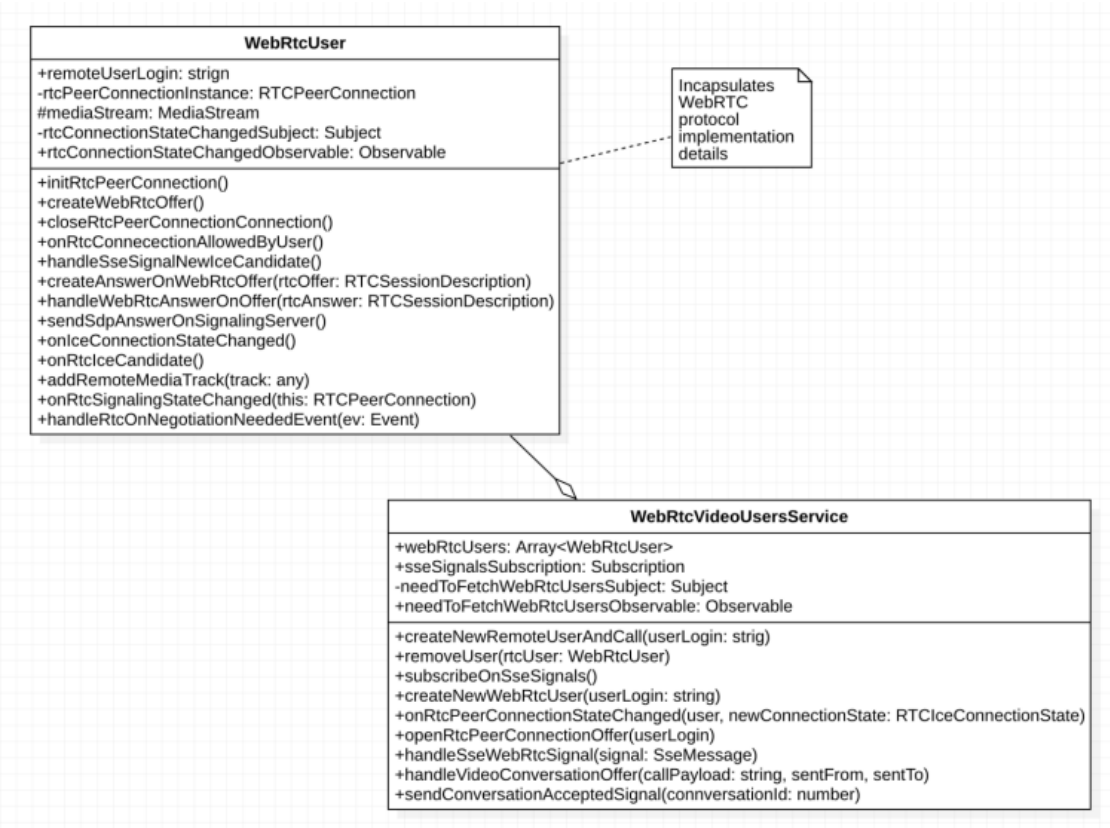

Рисунок 3.9 — Діаграма класів та сервісів реалізації логіки управління каналами передачі даних за WebRTC

Дані два класи інкапсулюють логіку створення та управління учасників розмов, каналів передачі даних за технологією WebRTC (за описом у розділах 1.3, 2.2, 2.3) та збереження медіа-потоків користувачів.

Важливо акцентувати увагу на тому, що WebRtcVideoService зберігає метадані та потоки передачі даних, тому даний клас повинен бути сервісом (Service) у термінології фреймворку Angular.

Після реалізації функціональності створення та управління учасників розмови (відеоконференції), необхідно реалізувати логіку відображення медіа-

потоків учасників розмови на інтерфейсі користувача клієнтської частини програмного продукту.

Одним із основних факторів побудови відображення клієнтської частини програмного продукту є підтримка цілісності інформації, що надається користувачеві. Основною інформацією для відображення програмного продукту є медіа-потоки учасників відеоконференції (блоки відео для кожного учасника розмови) та спливаючі повідомлення про підтвердження запиту на дозвіл користувача в участі у відеоконференції.

Розроблена архітектура передбачає виконання основної логіки налаштування відео зв’язку у «фоновому» режимі без участі клієнта як такого. Також, повідомлення на підтвердження повинні бути відображені користувачу без прив’язки до конкретної сторінки системи, де наразі знаходиться користувач та без необхідності до дій клієнта.

Розглянемо детальніше концепт відображення спливаючих вікон підтвердження. Повідомлення про підтвердження створюються в момент обробки повідомлення-сигналу від сигнального сервера. Після обробки інформації, формується модель відображення, що повинна бути відображена на сторінці користувача.

Було вирішено створити компонент-деталь, що відповідає за події клієнтської частини програмного продукту, що пов’язана з відповідним шаблоном відображення моделей. Даний компонент, у свою чергу, з використанням механізму ін’єкції HTMLшаблонів, агрегується у компоненті-мастері відображення стеку повідомлень.

Шаблон відображення мастер-компоненту, у свою чергу, агрегується компонентом основної сторінки відображення. Цей підхід дозволяє відображати повідомлення на будь-якій активній сторінці веб-частини продукту.

Але оновлення моделей відображення фреймворком Angular стандартно відбувається за зміни моделей, що зв’язані з відображенням конкретної

сторінки. Таким чином, ре-рендеринг компонентів веб-сторінки, за життєвим циклом вебчастини, потребує дій користувача (наприклад, подію кліку).

Але можлива ситуація, за якою користувач не ініціює події браузеру і лиш чекає на повідомлення про підтвердження запиту на створення розмови.

Використання методу ngDoCheck() не дало б бажаного результату, оскільки компонент змінює значення внутрішніх об’єктів масиву. Так як посилання на модель об’єкту масиву не змінюється, Angular Lifecycle Hook ngDoCheck() не гарантує збереження консистентного стану даних-значень.

Вирішенням даної проблеми є ін’єкція об’єкту NgZone та виконання дій всередині області видимості зони перевірки моделі Angular. Приклад використання наведено на рисунку 3.10.

```
this.ngZone.run( fn: () => {
    this.webRtcUsersManager.sendConversationAcceptedSignal(this.alertData.conversationId
    this.alertsManagerService.closeAlert(this.alertData);
});
```

Рисунок 3.10 — Використання NgZone для забезпечення гарантованої перевірки моделей

Таким чином, повідомлення буде відображено користувачеві веб-продукту без необхідності додаткових дій або оновлення сторінки браузеру.

Також, одним із важливих аспектів побудови архітектури відображення є області відображення відео-потоків учасників програмного продукту. Вище згадані компоненти, аналогічно до повідомлень-запитів, побудовано за принципом майстердеталь. Сторінка відеоконференції агрегує в собі компонент відображення учасників відеоконференцій (майстер), що, у свою чергу, агрегує компоненти відображення відео-потоків для кожного користувача (деталі). Важливою особливістю розробки є те, що користувач має можливість до переходу на різні сторінки клієнтської частини програмного продукту.

Кожний перехід ініціює знищення компонентів відображення та деталей фреймів відео-потоків. При зворотному переході на сторінку, компоненти

створюються заново та відбувається рендеринг компонентів сторінки заново. Так, як локальні медіа потоки та медіа-потоки користувачів інкапсюльовано в Angularсервісах, проблем із перетвореннями медіа-потоків бути не повинно. У свою чергу, виникає потреба до відображення відео-потоків користувачів, що тільки-но були успішно з’єднані.

Для цього було використано концепт, схожий на відображення подій-. Рендернинг нових елементів відео відбувається тільки за умови зміни статусу передачі даних на connected, при чому, додання нового об’єкту шаблону відображення медіа-потоків у мастер-компонент відеоконференцій буде виконано з використанням зони дії потоку Angular.

Таким чином, навіть за повної бездіяльності клієнта системи, актуальні елементи відображення медіа-потоків та важливих повідомлень будуть відображені без необхідності додаткових маніпуляцій з боку клієнта програмного продукту.

3.3 Аналіз ефективності різних стандартів стиснення відео

За останні кілька десятиліть методи стиснення цифрового відео стали важливими для створення, передачі та споживання візуальної інформації. Відео складається з багатьох нерухомих зображень, які знімаються за відносно короткий проміжок часу, тому два сусідніх зображення схожі. Відомо, що між сусідніми зображеннями або кадрами існує сильна кореляція. Ця кореляція в часовому напрямку називається міжкадровою кореляцією. Цю кореляцію між кадрами можна використати для ефективного зменшення та досягнення стиснення без втрат.

Якість відео — це показник, який використовується для вимірювання погіршення якості відео порівняно з оригіналом[30]. Якість відео можна оцінити або за допомогою математичної моделі (об’єктивно), або попросивши користувача оцінити його (суб’єктивно).

Бажана модель розробляється на основі наявності інформації про вихідний сигнал.

Повний довідковий метод: метод FR є найточнішим методом і потребує більше обчислювальних зусиль. Цей метод порівнює кожен отриманий піксель із відповідним пікселем у вихідному відео та не знає нічого про процес кодування чи передачі, який мав місце.

Стандартний метод редукції: метод ZS не вимагає наявності повного вихідного відео. Зазвичай ми виділяємо функції з отриманого та оригінального відео та порівнюємо їх, щоб отримати показник якості. Цей метод можна використовувати, коли деякі частини оригінального відео недоступні.

Незареєстрований метод: метод оцінки якості відео без наявності вихідного сигналу. Цей метод характеризується найменшою точністю.

Щоб розпочати стиснення відео, витягується статична версія ffmpeg. Пейтон написав набір кодів для використання кодеків та їхніх параметрів (бітрейт аудіо, відео, розмір кадру тощо). Використовуючи веб-середовище Tomcat, я створив локалізований сервер, який може завантажувати та конвертувати файли. Основна увага полягає в тому, щоб порівняти різні типи кодерів на основі необхідного бітрейту, якості відео та розміру вихідного файлу.

Для дослідження були обрані файли .mp4 довжиною 00:02:08 у трьох різних форматах. Їхні розміри наведені в таблиці 1. Під час стиснення відео розмір файлу насправді не враховується, і система дозволяє індивідуально вибирати параметри кодека за замовчуванням.

У таблиці 2 ми знову зважили налаштування бітрейту 320 кбіт/с для всіх кодеків і зміну розміру вихідного файлу залежно від якості відео. Ми виявили, що для певного кодека тип вихідного файлу залишався досить постійним незалежно від розміру вхідних даних. Таблиця 2 показує, що лише кодек H.264 значно зменшив розмір вихідного файлу, але якість вихідного відео була задовільною.

3.5 Висновок до розділу 3

Було програмно реалізовано серверний програмний компонент та базисну бізнес-логіку створення та управління сигналізації, що виступає у ролі сигнального сервера, необхідного для реалізації технології передачі медіа-потоків WebRTC.

Проведено тестування та перевірку системи на працездатність та реакцію до помилок. Клієнтська частина програмного продукту працює без збоїв, виконує весь необхідний функціонал.

Загалом тестування програмного продукту показало повну відповідність поставленому технічному завданню

Реалізовано логіку створення та управління відеоконференціями, програмно реалізовано прошарки серверної частини програмного продукту: прошарок ресурсів, бізнес-логіки та доменів бази даних з використанням фреймворку Spring. Також реалізовано логіку зміни схеми бази даних та даних з використанням Liquibase.

Визначені ключові місця появи колізій на клієнтській частині програмного продукту та розроблено методи авторизації та автентифікації, написано реалізацію компонентів обміну сигналами із сигнальним сервером, програмно реалізовано створення відеоконференцій за підтвердженням та відповідна модель компонентів системи.

4 ПЕРЕВІРКА ПРАЦЕЗДАТНОСТІ ТА ТЕСТУВАННЯ

4.1 Методики перевірки працездатності та тестування програмних компонентів

Оскільки основна ціль покладена в досягненні максимальної кількості учасників в відео-конференції, тестування має проходити з навантаженням користувачів на сервер та функціонал відео-конференцій.

Навантажувальний тест — це процес тестування, який виконується за допомогою інструменту навантажувального тестування. Apache JMeter для навантажувального тестування є важливим інструментом для визначення того, чи може тестована веб-програма відповідати вимогам високого навантаження.

Це також корисно для аналізу популярних серверів з високим навантаженням. Виявлення продуктивності веб-програми за різних умов за допомогою JMeter. Алгоритм роботи Jmeter показаний на рисунку 4.1.

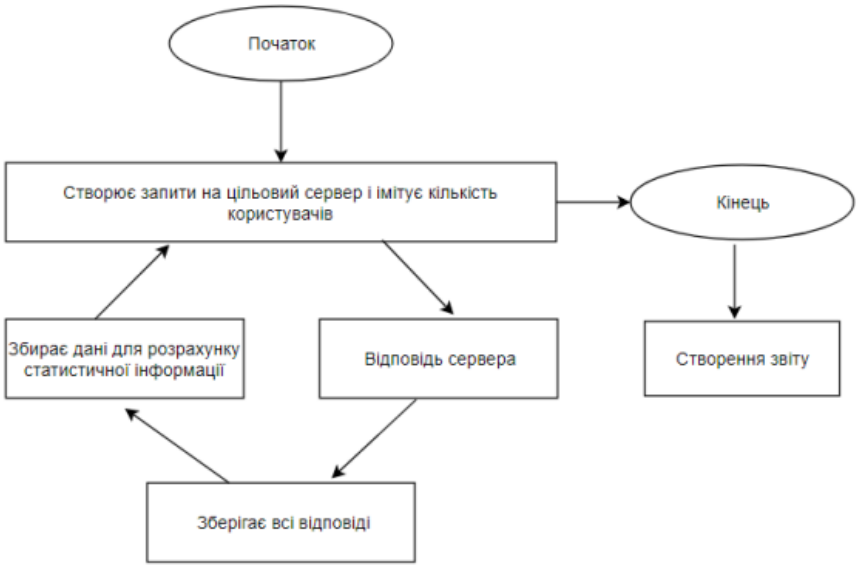

Рисунок 4.1 — Алгоритм роботи Jmeter

На початку алгоритму Jmeter робить HTTP-запит до потрібного веб-сервера. Указуються такі параметри, як кількість одночасних користувачів з попередніми налаштуваннями запиту. Потім веб-програма (сервер) відповідає на HTTP-запит у вигляді цифрового значення:

— інформаційні відповіді (100–199);
— успішні відповіді (200–299);
— переспрямування (300–399);
— помилки клієнта (400–499);
— помилки сервера (500–599).

Усі відповіді, незалежно від того, є вони неправильними чи ні, фіксуються у звіті для подальшого розповсюдження інформації. Наступним кроком в алгоритмі програми є зберігання даних у вигляді статистики. Алгоритм працює, доки не мине заданий проміжок часу або поки всі користувачі не завершать свою роботу. Завершальним етапом є створення звітів на основі збереженої інформації.

У контексті тестування продуктивності передбачає використання програмних інструментів для моделювання продуктивності програми за певних обставин. Кількісне тестування продуктивності розглядає такі показники, як час відгуку, тоді як якісне тестування дивиться на масштабованість, стабільність і сумісність. Коли ми чуємо слово «продуктивність», більшість із нас відразу думають про швидкість. Швидке завантаження та час відповіді сьогодні важливі, але потрібно думати про загальну картину, а не просто перевіряти всі посилання, щоб переконатися, що вони працюють. Те, що все працює ідеально під час тестування, не означає, що це працюватиме, коли система переповнений трафіком. Цей тест виконується за звичайних умов використання веб-програми. Приклад показано на рисунку 4.2.

Тестування продуктивності програми може допоможе виявити проблеми та покращити загальну продуктивність. Тестування продуктивності виявляє багато типових проблем, наприклад вузькі місця. Переривання потоку даних через обмеження ємності називаються вузькими місцями. Вузьке місце може виникнути, наприклад, коли відбувається раптове збільшення трафіку, який сервер не може впоратися.

Рисунок 4.2 — Тестування в звичайних умовах

Функціональне тестування — це тип тестування, коли система повинна робити з точки зору функціональності. Наприклад, введення логіна та пароля, додавання 45, видалення інформації з сервера тощо. Функціональне тестування передбачає здатність виконувати набір завдань, порівнювати однакові результати з очікуваними результатами та повторювати той самий набір завдань кілька разів з різними вхідними даними та з однаковим рівнем точності. Jmeter дозволяє виконувати цю частину тестування за допомогою тестових планів і різних HTTP-запитів.

Тестування на витривалість — це тип тестування веб-програм, який перевіряє програмне забезпечення під великим навантаженням протягом значного часу, щоб оцінити його поведінку під час тривалого використання. Основна мета тестування на довговічність полягає в тому, щоб переконатися, що система добре справляється з тривалими навантаженнями без зниження часу відгуку. Приклад графіка випробувань на витривалість наведено на рисунку 4.3.

Цей тип тестування має різні цілі. Одна з головних цілей тестування на витривалість — перевірити наявність витоків пам’яті та побачити, як система поводиться протягом тривалих періодів використання. Після тривалого

періоду часу переконатись, що час відповіді системи такий самий або кращий, ніж на початку тесту.

Рисунок 4.3 — Тест на витривалість

Для визначення кількості користувачів і транзакцій, які підтримуватиме конкретна система, і досягнення цілей продуктивності.

Управління майбутніми робочими навантаженнями вимагає розуміння додаткових ресурсів (ємності процесора, дискового простору, використання пам’яті, пропускної здатності мережі тощо), необхідних для підтримки майбутнього використання. Тестування довговічності зазвичай виконується шляхом перевантаження системи або зменшення певних системних ресурсів і оцінки результатів.

Перевірка безпеки дуже важлива для веб-програми електронної комерції, які зберігають конфіденційну інформацію про клієнтів, наприклад кредитні картки. Тестування включає:

— не можна допускати несанкціонований доступ до захищених сторінок;

— файли з обмеженим доступом неможливо завантажити без відповідного доступу;

— сеанс перевірки автоматично припиняється після тривалої бездіяльності користувача;

— під час використання сертифіката SSL веб-сайт повинен перенаправляти на сторінку, зашифровану SSL.

Основною метою тестування безпеки в Інтернеті є виявлення та усунення потенційних вразливостей. щось схоже на те:

— сканування мережі;

— сканування вразливостей;

— злом паролів;

— огляди журналів;

— перевірка комплектності.

Ручне тестування — це тестування програмного забезпечення, у якому тестування виконується вручну аналітиком контролю якості. Він здійснюється для виявлення помилок у програмному забезпеченні. При ручному тестуванні тестувальники перевіряють усі основні функції конкретної програми або програмного забезпечення. У цьому процесі тестувальники програмного забезпечення виконують тестові сценарії та генерують звіти про тестування без допомоги інструментів автоматизованого тестування програмного забезпечення. Це класичний метод для всіх видів тестування, який допомагає знаходити помилки у програмних комплексах. Зазвичай, це виконується досвідченими тестувальниками для завершення процесу тестування програмного забезпечення. Коротше кажучи, ручне тестування найкраще підходить для наступних областей чи сценаріїв:

— дослідницький тест: цей тип тесту вимагає знань, досвіду, аналітичного та логічного, творчості та інтуїції тестувальника;

— юзабіліті-тестування: це область, де необхідно виміряти, чи є програмне забезпечення продукту зручним, ефективним або корисним для кінцевих користувачів. Спостереження людиною є найважливішим фактором тут і в основному здійснюється вручну;

— користувальницький тест: для цього сценарію немає спеціального підходу;

Абсолютно незапланований метод тестування, при якому розуміння тестувальника є єдиним важливим фактором.

Переваги ручного тестування:

— отримайте швидкий і точний візуальний зворотний зв'язок;

— це дешевше, тому що вам не потрібно витрачати свій бюджет на автоматизовані інструменти та процеси;

— людське судження та інтуїція завжди піддаються ручним елементам;

— при тестуванні невеликих змін автоматичне тестування може тривати кілька годин, але можливе ручне тестування на льоту.

Недоліки ручного тестування:

— ненадійний метод випробувань, оскільки він виконується людьми. Тому завжди схильний до помилок і помилок;

— не можна записати процес ручного тестування, тому ручне тестування не можна використовувати повторно;

— цей метод тестування може ускладнити виконання деяких завдань вручну і вимагати додаткового часу на етапі тестування програмного забезпечення.

Під час автоматизованого тестування програмного забезпечення тестувальники перевіряють сценарії для автоматизації виконання тестів. Тестувальники розробляють сценарії тестування та перевіряють програмне забезпечення за допомогою відповідних засобів автоматизації. Мета полягає в тому, щоб пройти тест у найкоротші терміни. Автоматизовані тести повністю покладаються на попередні сценарії, які запускаються автоматично для порівняння фактичних та очікуваних результатів. Це допомагає тестувальникам визначити, чи правильно працює програма. Автоматизоване тестування дозволяє виконувати повторювані завдання та регресійне тестування без залучення ручних тестувальників. Усі процеси автоматизовані, але автоматизації потрібна ручна робота зі створення початкових тестових сценаріїв.

Автоматизоване тестування є найкращим варіантом у таких областях або сценаріях:

— регресійне тестування: автоматизоване тестування зі зміною часу підходить, якщо дозволено своєчасне ініціювання регресії;

— тестування навантаження: коли справа доходить до навантажувального тестування, автоматизоване тестування також є найкращим способом тестування;

— ітеративне виконання: тести, які потребують повторного виконання завдань, найкраще автоматизуються;

— тестування продуктивності: аналогічні тести, які мають імітувати тисячі одночасних користувачів, потребують автоматизації.

Переваги автоматизованого тестування:

— автоматизоване тестування допомагає знайти більше помилок у порівнянні з тестувальниками-людьми;

— більшість процесу тестування автоматизована, що робить його швидким і ефективним;

— запис автоматизованих процесів. Це дозволяє повторно використовувати і запускати ті самі тестові операції;

— автоматизоване тестування виконується за допомогою програмного забезпечення, тому воно працює без утоми та втоми, на відміну від ручних тестувальників;

— простота підвищення продуктивності за рахунок швидких та точних результатів випробувань;

— пвтоматизовані випробування підтримують різні програми.

Недоліки автоматизованого тестування: без людського чинника важко зафіксувати візуальні аспекти інтерфейсів, як-от кольори, шрифти, розміри, контрастність і розміри кнопок. Інструменти для запуску автоматичних тестів можуть бути дорогими та можуть збільшити вартість вашого тестового проекту. Інструменти автоматизації тестування, як і раніше, ненадійні. Кожен

інструмент автоматизації має обмеження, які зменшують область автоматизації.

4.2 Структура платформи тестування

Структура тестової платформи складається із сервера та клієнтської системи. Клієнтська частина включає Jmeter і ServerAgent (клієнтська частина), а серверна частина включає XAMPP, ServerAgent (серверна частина) та веб-додаток. Структура тестової платформи показано на рисунку. 4.4.

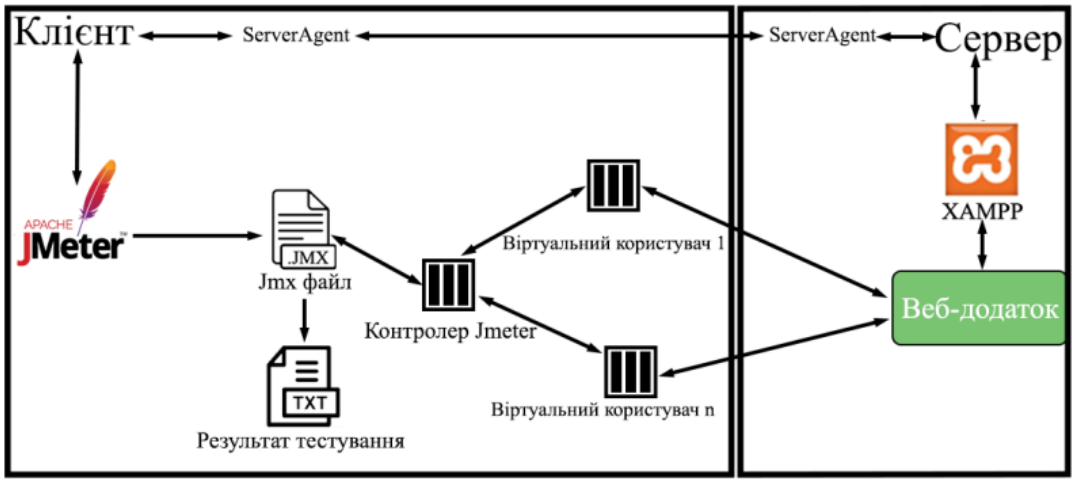

Рисунок 4.4 — Структура платформи для тестування

Як показано на рисунку 4.4, тестовий сценарій створюється за допомогою Jmeter. Він зберігається у файлі jmx та має тестовий контролер. Цей контролер тесту має різні параметри. Різні компоненти контролера використовують для різних типів випробувань. Jmeter імітує поведінку реального користувача у веб-програмі. Результати тесту зберігаються у форматі CSV і можуть бути доступні за допомогою програми MS Excel.

XAMPP діє як програмне забезпечення веб-сервера, на якому розміщуються веб-програми. XAMPP є комбінацією HTTP-сервера Apache, бази даних та мови програмування PHP. Разом ці компоненти дозволяють запускати веб-програми без встановлення додаткового програмного забезпечення. Jmeter взаємодіє з веб-сервером через запити HTTP. HTTP

функціонує як протокол запиту-відповіді між клієнтами та серверами. Коли Jmeter відправляє запит на сервер HTTP, сервер повертає відповідь клієнту. Відповідь містить інформацію про статус запиту та може містити запитаний вміст.

ServerAgent об'єднує серверне та клієнтське обладнання, щоб забезпечити віддалений моніторинг ресурсів веб-сервера з клієнтського комп'ютера під час тестування веб-додатків. Основна структура файлу JMX складається з наступних компонентів:

— план тестування: основний елемент, що описує деякі кроки, які виконує JMeter під час виконання тесту.

— група потоків: це найпростіша існуюча група потоків. Є деякі параметри за замовчуванням, які підходять для більшості сценаріїв тестування навантаження.

— sampler: дозволяє JMeter надсилати різні типи запитів на сервер.

— слухачі: слухачі — це компоненти, які відображають результати тестів.

Test Script Recorder — це один з компонентів, який дозволяє записувати поведінкову модель вашого сценарію за допомогою веб-браузера. Оскільки веб-сервер знаходиться на тому ж комп'ютері, що й тест, я використовую проксі-сервер та локальний хост для доступу до веб-додатку.

Проксі-сервер — це посередник між вашою мережею та зовнішнім Інтернетом. Коли ви відвідуєте веб-адресу у своєму браузері, ваш запит надсилається на проксі-сервер, який завантажує сторінку та надсилає її на ваш комп'ютер. Порт 8080 tcp використовується для доступу до веб-програми через проксі.

4.3 Тестування та аналіз результатів роботи програмного продукту

Метою даного тесту є реалізація постійно зростаючого навантаження та визначення точки насичення (для заданої апаратної та програмної

конфігурації), максимальної кількості паралельних активних потоків, при якому починається відмова в обслуговуванні.

У цьому тесті використовується веб-додаток та елементи попереднього тесту. Перед тестуванням налаштуйте "Групу потоків". Ми встановлюємо наступні параметри: 807 користувачів, час запуску 88 секунд та час виконання 88 секунд. Додайте наступні слухачі, щоб перевірити точку насичення та виконати аналіз даних: Звіт. Після початку тестування слід перевірити кількість активних користувачів, а також поля Devition та Summary Report. Коли середньоквадратичне відхилення починає швидко збільшуватись і в полі помилки з'являються помилки, це означає, що точку насичення конфігурації знайдено.

Як бачимо на рисунку 4.5 час відгуку почав експоненційно збільшуватися, коли кількість користувачів перевищила 1500, і в цей момент стандартне відхилення почало збільшуватися разом з кількістю користувачів.

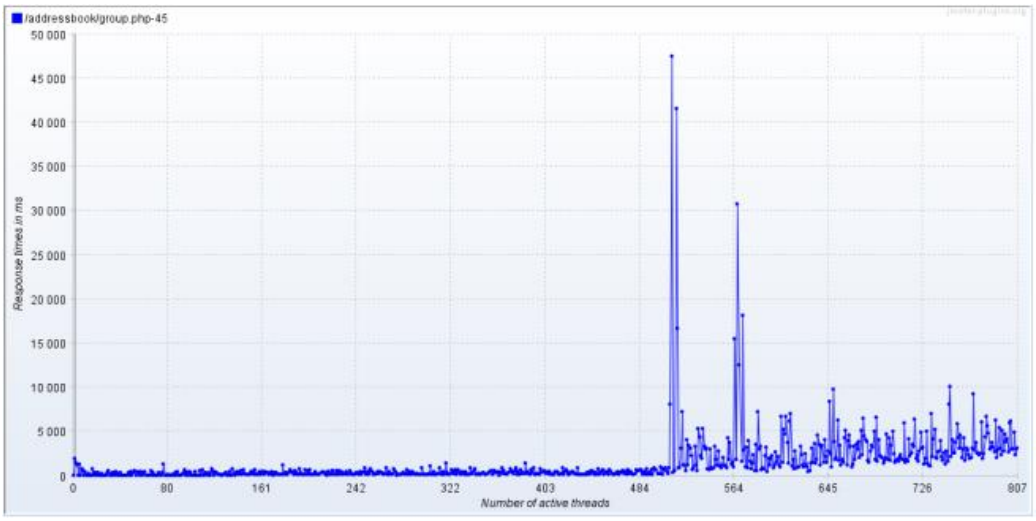

Рисунок 4.5 — Співвідношення часу відгуку та кількісті користувачів

Після того, як кількість користувачів перевищила 1500 (точка насичення), я почав одержувати коди помилок. Це означає, що сервер не встигає обробляти всі запити. На рисунку 4.6 показаний графік успішних та неуспішних відповідей на запити. Частота помилок складає 2,4% від усіх запитів.

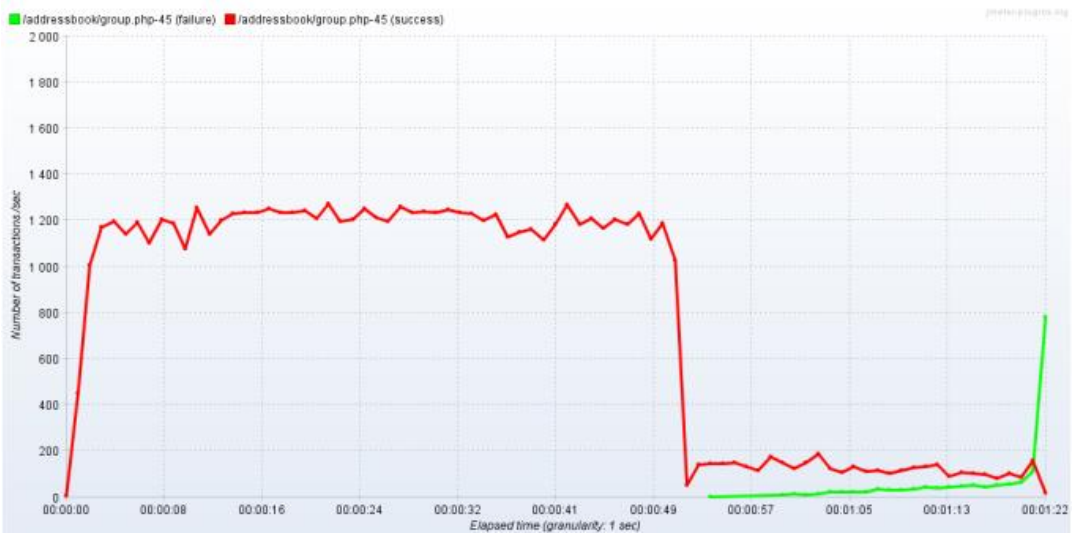

Рисунок 4.6 — Успішні відповіді та помилки

За допомогою тесту було визначено точку насичення 1500-2000 активних користувачів. Ця інформація дає можливість підготуватися до таких навантажень і уникнути зниження продуктивності системи.

Наступна мета тестування є динамічне завантаження та перевірка поведінки веб-програми після пікового навантаження. У цьому тесті використовується агент сервера PerfMon для моніторингу ресурсів сервера. По-перше, вам потрібно завантажити ServerAgent та розархівувати його на своєму сервері. Потім запустіть файл під назвою startAgent.bat. Якщо все зроблено правильно, ви отримаєте інформацію, зображену на рисунку 4.7.

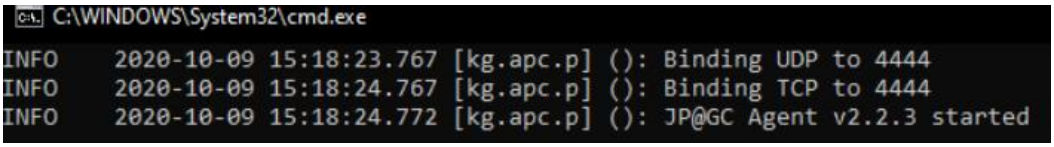

Рисунок 4.7 — Робочий ServerAgent

В попередніх тестуваннях пікової потужності було визначено, що пікова потужність системи складає від 1500 до 2000 активних користувачів. У динамічних тестах використовується елемент Ultimate Thread Group, який дає повний контроль над динамічною кількістю користувачів. Налаштування

базуються на точці насичення 1500 користувачів. На рисунку 4.8 показаний графік переваг та активних користувачів з часом.

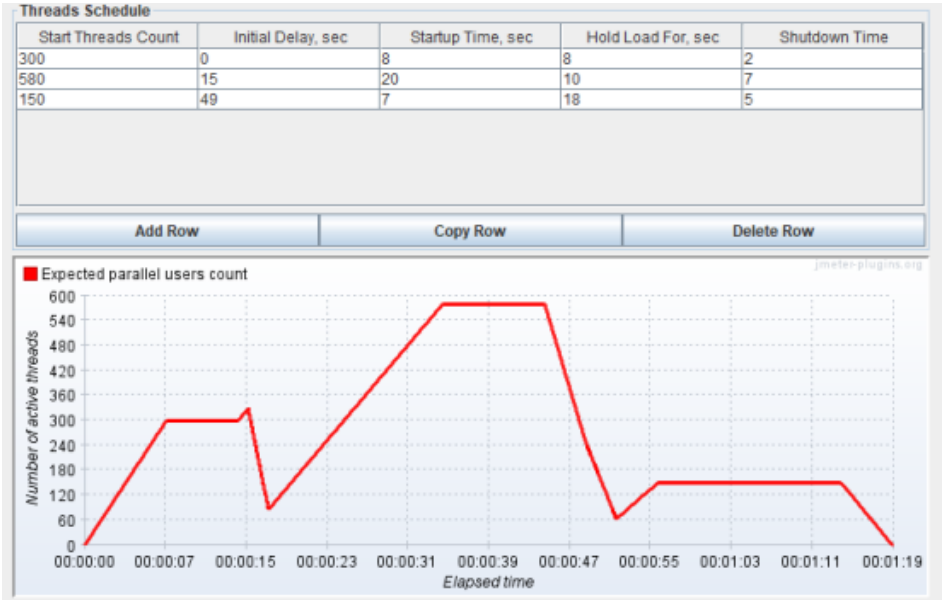

Рисунок 4.8 — Налаштування користувачів

Для наступних відповідей по тестуванні потрібен додатковий сервіс jp@gc — Коди відповідей за секунду. Як і в попередньому тесті, були отримані хибні запити з 1500 активними користувачами, як показано на 31 секунді на рисунку 4.9. Однак після зниження навантаження до 800 користувачів на 55 секунд все ж було отримання хибні запити.

Таблиця 4.1 — Основні метрики Get запиту

Вид запиту	Значення				
	Кількість запитів	Середній час відгуку, с	Максимальний час відгуку, с	Помилкові запити, %	Кількість запитів в секунду
Get	38709	0.37	6.452	8.42	490

Таблиця 4.1 показає основні показники, отримані під час випробувань. Наприклад: відсоток запитів з помилками, середній та максимальний час відповіді, кількість запитів, пропускна спроможність тощо.

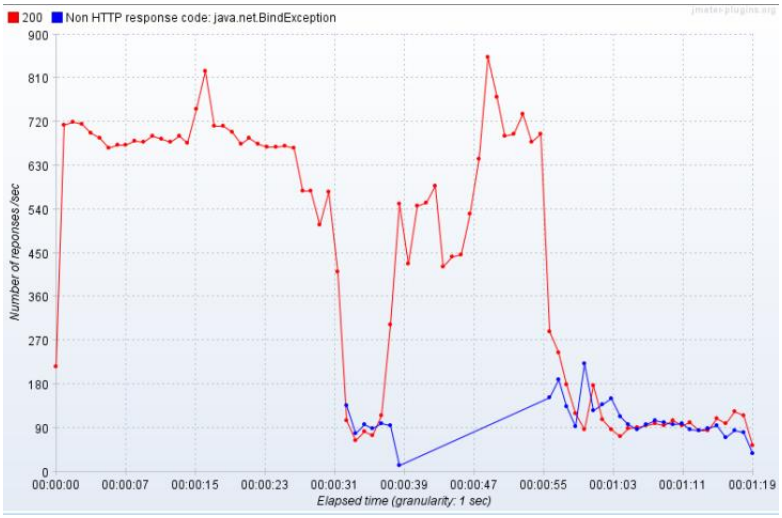

Рисунок 4.9 — Співвідношення помилкових запитів

Як видно з таблиці 4.1, з 38709 запитів 8,42% були помилками. Це говорить про те, що система не витримує навантаження 490 кбіт/с.

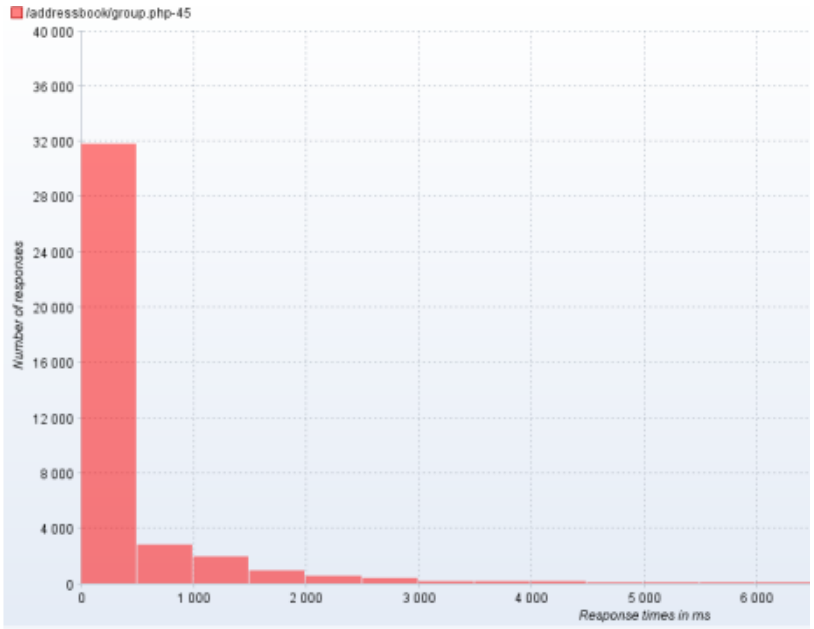

Рисунок 4.10 — Співвідношення помилкових запитів

Час відгуку також є важливим фактором під час тестування. Середній час відгуку становив 300 мс із відгуком 82,7%. Час відгуку становить від 500 до 6400 мс, що становить 17,3% відгуку. Час відгуку до 2 секунд є нормальним результатом.

Як видно з рисунку 4.11 моніторинг показує стан трьох компонентів серверу: завантаження оперативної пам’яті, процесору та запис на диск.

Рисунок 4.11 — Моніторинг ресурсів серверу

Після проведення динамічного тесту навантаження можна зробити наступні висновки: веб-програм витримує до 1500 активних користувачів і після цієї позначки починає отримувати фіктивні запити. Таблиця 4.1 показує, що з 38709 запитів 8,42% або 3259 запитів невірні. Також було виявлено, що навіть після зниження навантаження до 160 активних користувачів веб-програма, як і раніше, відповідав помилковими запитами. Під час тестування збирали метрики сервера. Завантаження центрального процесора складало від 20% до 90%, завантаження пам'яті — 40%, а завантаження диска — від 0,1% до 1%.

Тести на витривалість проходядть для визначення довгострокової стабільності вашого продукту. Перевірка працездатності сервера (використання пам'яті, завантаження диска, завантаження ЦП, пропускну здатність мережі тощо) за допомогою програми ServerAgent.

Встановимо "Групу потоків". Було встановлено параметри: 300 користувачів, період набору потужності 90 секунд, тестування 400 секунд.

На рисунку 4.12 показано час відгуку сторінки входу та експорту. Можливо побачити, що вхід до сайту займає більше часу, ніж перехід на сторінку експорту.

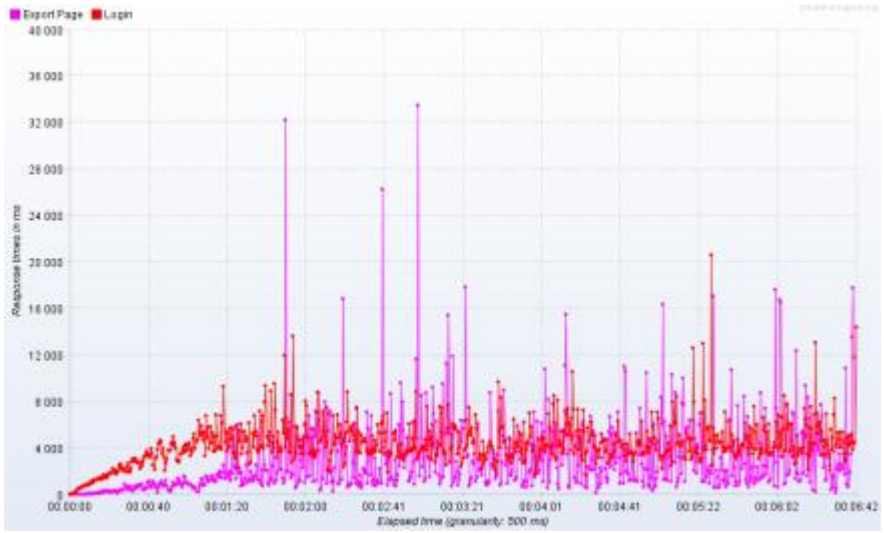

Рисунок 4.12 — Розподіл часу відгуку

На рисунку 4.13 показано стан сервера під час тривалого тестування. Вузьким місцем на сервері є процесор, який видно за 100% завантаження протягом усього тесту.

Рисунок 4.13 — Стан сервера

На рисунку 4.14 показано розподіл часу відгуку. Як результат, 90% значень сторінки входу мають час відгуку від 5 до 4000 мс, а значення сторінки

експорту - від 5 до 2000 мс. А час відгуку для 10% значення становить від 2000 до 36000 мс.

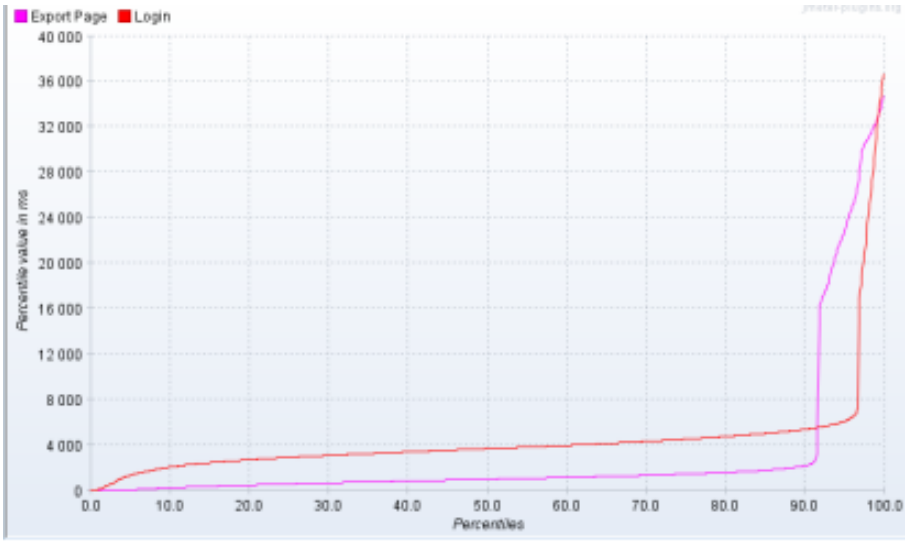

Рисунок 4.14 — Перцентиль часу відгуку

Було проведене випробування на витривалість, щоб побачити, як система поводиться після тривалого використання. За допомогою цього тесту було виявлено, що система може витримати 250 одночасних користувачів, які виконують кроки входу до веб-програми та під час відео-конференції. Тест тривав 400 секунд, де можливо зафіксувати, що час відгуку сторінки входу трохи вищий за комфортний рівень в 2 секунди. Вузьким місцем сервера був процесор, завантажений на 100%.

4.3 Висновок до розділу 4

Тестування продуктивності було виконано для визначення функціональності веб-програми. Це дозволило провести аналіз використовувати доступні можливості системи та забезпечити стабільну та швидку роботу відео-конференцій, тому є відповідь скільки користувачів можуть бути присутніми під час проведення відео-конференції.

Об'ємне тестування призвело до додавання нових даних на сервер із навантаженням 50 одночасних користувачів. Після тестування всі дані були додані до бази та час відгуку не перевищував 2 секунд.

Тестування на витривалість виявила, що 300 користувачів протягом тривалого часу завантажують процесори серверів на 90-100%. Це говорить про те, що вузьким місцем для веб-програми є процесор. Заміна процесора на більш досконалий покращить масштабування та продуктивність.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку нової інформаційної система для проведення відеоконференцій, яка дозволяє створювати конфіденційні конференції з доступом лише перевірених осіб. Сервіс був розроблений на новій системі передачі пакетів даних, за допомогою якого з поганим зв’язком буде хороше з’єднання.

Особливістю розробки є новий метод організації передачі потокових даних між браузерам, універсальний та швидкий метод передачі пакетів даних, які дозволять мати безперебійну та конфіденційну роботу. Новизною в даному випадку є те, що буде створено новий метод поверх відомої технології з відкритим кодом, під назвою WebRTC.

Прикладом аналогу є Zoom ціна на початковому етапі розробки 67000$.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 5.1.

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження таблиці 5.1

3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так із комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження таблиці 5.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 5.2.

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	4	4
Наявність аналогів на ринку	3	3	4
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	4	3	4

Продовження таблиці 5.2

Експлуатаційні витрати	4	4	3
Ринок збуту	4	3	4
Конкурентоспроможність	3	4	3
Фахівці з технічної і комерційної реалізації	4	3	4
Фінансування	4	4	3
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	4	4
Супровідна документація	4	3	4
Сума	44	42	44
Середньоарифметична сума балів	(44+42+44) / 3 = 43,33		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3.

Таблиця 5.3 - Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок того, що розроблена інформаційної система для проведення відеоконференцій дозволяє створювати конфіденційні конференції з доступом лише перевірених осіб. Сервіс був розроблений на новій системі передачі пакетів даних, за допомогою якого з поганим зв'язком буде хороше з'єднання. Особливістю розробки є новий метод організації передачі потокових даних між браузерам, універсальний та швидкий метод передачі пакетів даних, які дозволять мати безперебійну та конфіденційну роботу. Новизною в даному випадку є те, що буде створено новий метод поверх відомої технології з відкритим кодом, під назвою WebRTC.

5.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

5.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{М}{Т_р} \cdot t \qquad (5.1)$$

де М — місячний посадовий оклад конкретного розробника (дослідника), грн.;

Тр — число робочих днів в місяці, 23 днів;

t — число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 5.4.

Таблиця 5.4 — Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	55000	2391,30	45	107608,696
Програміст	53500	2326,09	45	104673,913
Всього				212282,61

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

5.2.2 Додаткова заробітна плата розробників, які брати участь в розробці обладнання/програмного продукту.

Додаткову заробітну плату прийнято розраховувати як 15 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 15\ \% / 100\ \% \qquad (5.2)$$

$$З_д = (212282{,}61 \cdot 15\ \% / 100\ \%) = 31842{,}39 \ \text{(грн.)}$$

4.2.3 Нарахування на заробітну плату розробників.

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \qquad (5.3)$$

$$Н_з = (212282,61 + 31842,39) \cdot 22\ \% / 100\ \% = 53707,50\ (грн.)$$

5.2.4. Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

5.2.5 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді розраховується за формулою:

$$А = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12}\ [грн.]. \qquad (5.4)$$

де Ц — балансова вартість обладнання, грн.;

Т — термін корисного використання обладнання згідно податкового законодавства, років

$t_{вик}$ — термін використання під час розробки, місяців

Розрахуємо, для прикладу, амортизаційні витрати на комп’ютер балансова вартість якого становить 30850 грн., термін його корисного використання згідно податкового законодавства — 2 роки, а термін його фактичного використання — 1,96 міс.

$$А_{обл} = \frac{30800}{2} \times \frac{1,96}{12} = 2514,95грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 5.5.

Таблиця 5.5 —Амортизаційні відрахування на матеріальні та нематеріальні ресурси для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна периферія (NVIDIA GeForce MX130, Процессор Intel Core i7-10700F s-1200 2.9GHz/16MB, SSD Kingston A400 240GB 2.5, 21.5" Samsung F22T350)	30850	2	1,96	2514,946
Офісне обладнання (меблі)	29300	4	1,96	1194,293
Приміщення	1150000	20	1,96	9375,000
Всього				13084,24

Так як вартість ліцензійної ОС, спеціалізованих ліцензійних нематеріальних ресурсів та їх підписка менше 20000 грн.: Mongodb — 2094,55 грн., WebStorm — 5879,43 грн/рік, оренда серверу — 350 грн/місяць, то дані нематеріальні активи не амортизується, а їх вартість включається у вартість розробки повністю, $B_{нем.ак.}$ = 3766 грн.

5.2.6 Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (5.5)$$

де В — вартість 1 кВт-години електроенергії для 1 класу підприємства, В = 6,2 грн./кВт;

П — встановлена потужність обладнання, кВт. П = 0,4 кВт;

Ф — фактична кількість годин роботи обладнання, годин.

$К_п$ — коефіцієнт використання потужності, $К_п$ = 0,9.

$$В_е = 0,9 \cdot 0,4 \cdot 8 \cdot 45 \cdot 6,2 = 803,52 \text{ (грн.)}$$

5.2.7 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.6)$$

де $Н_{ів}$ — норма нарахування за статтею «Інші витрати».

$$І_в = 212282,61 * 95\% / 100\% = 201668,5 \text{ (грн.)}$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$Н_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.7)$$

де $Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$Н_{нзв} = 212282,61 * 135\ \% / 100\ \% = 286582 \text{ (грн.)}$$

5.2.9 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$В_{заг}$ = 212282,61+31842,39+53707,50+13084,24+3766+803,52+201668,5+
+286582 = 803736,26 грн.

5.2.11 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (5.8)$$

де η — коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

ЗВ = 803736,26 / 0,5 = 1607473 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

а) вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

б) зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

в) кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб’єктів (зацікавлених осіб) цього попиту;

г) визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов’язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об’єднати, враховуючи визначені ситуації з відповідними умовами.

5.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.9)$$

де $\pm\Delta Ц_о$ — зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N — кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_о$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_о = Ц_б \pm \Delta Ц_о$;

Цб – вартість програмного продукту у році до впровадження результатів розробки;

ΔN — збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

р — коефіцієнт, який враховує рентабельність продукту;

ϑ — ставка податку на прибуток, у 2022 році ϑ =18%.

Припустимо, що при прогнозованій ціні 53000 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *3000* грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року — на 1000 шт., протягом другого року — на 800 шт., протягом третього року на 500 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$\Delta П_1$ = (0*3000 + (53000 + 3000)*1000)* 0,8333* 0,32) * (1 - 0,18) = 11589332,870 грн.

$\Delta П_2$ = (0*3000 + (53000 + 3000)*(1000+800)* 0,8333* 0,32) * (1 - 0,18) = 22041599,118 грн.

$\Delta П_3$ = (0*3000 + (53000 + 3000)*(1000+800+500)* 0,8333* 0,32) * (1 - 0,18) = 28164265,540 грн.

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 61795197,53 грн.

5.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_1^T \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.10)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т — період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ = 0,05…0,15;

t — період часу (в роках).

Збільшення прибутку ми отримаємо, починаючи з першого року:

$$ПП = (11589332,870/(1+0,1)^1)+(22041599,118/(1+0,1)^2)+(28164265,540/(1+0,1)^3) = 10535757,15 + 18216197,62 + 21160229,56 = 49912184,33$$ грн.

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (5.11)$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}$=2...5, але може бути і більшим;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 1607473 = 3214945,04 \text{ грн.}$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід *(NPV, Net Present Value)* для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП – PV, \quad (5.12)$$

$$Е_{абс} = 49912184,33 – 3214945,04 = 46697239,29 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності *(IRR, Internal Rate of Return)* вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1, \quad (5.13)$$

$T_ж$ — життєвий цикл наукової розробки, роки.

$$\sqrt{E_в} = з\ (1 + 46697239,29/3214945,04\ – 1 = 1,495$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \quad (5.14)$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d = (0,09…0,14);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{min} = 0{,}14 + 0{,}05 = 0{,}19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, \quad (5.15)$$

$$T_{ок} = 1 / 1{,}495 = 0{,}67 \text{ р.}$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,67 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 1607473 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,67 роки.

ВИСНОВКИ

В магістерській кваліфікаційній роботі було створено програмний продукт для проведення віеоконференцій, на основі технології WebRTC.

Проведено аналіз сучасних технологій розробки веб-програм, їх особливостей, в результаті якого визначено основні вимоги до програмного продукту для проведення відеоконференції.

Проведено аналіз інструментів для проведення відеоконференцій, в результаті якого вибрано технологію WebRTC. Також використано сервіс Jmeter для перевірки програмного продукту до нагрузки.

Розроблено клієнт-серверний програмний продукт для проведення відеоконференцій, що включає серверний програмний компонент, який реалізує бізнес-логіку створення та управління конфігураціями відеоконференцій, та клієнтську частину на основі WebRTC, який на відміну від існуючих аналогів дозволяє одночасну передачу медіа-потоків між користувачами кількістю до 120. За рахунок використання технології WebRTC вдалося позбутися необхідності великих потужностей серверів, що дозволяє зменшити грошові витрати.

Розроблений програмний продукт протестовано. В процесі тестування було знайдено помилку, згенеровано звіт, проблемний код не було розгорнуто в середовищі.

В кваліфікаційній роботі проведено розрахунок економічної ефективності розробленого рішення, який показав, що комерційний ефект від реалізації результатів розробки за три роки складає майже 61 млн.грн., а період окупності — близько 0,67 роки.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Loreto S. Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP. [Електронний Ресурс]. / G. Wilkins, S. Salsano, P. Saint-Andre. Режим доступу до ресурсу: https://tools.ietf.org/html/rfc6202#page-16.

2. Russell, Alex. "Comet: Low latency data for the browser." / Alex Russell [Електронний Ресурс] Режим доступу до ресурсу: http://alex.dojotoolkit.org/?p=545, 2006.

3. Горбенко О.Ю. Огляд технології WebRTC для реалізації програмного забезпечення відеоконференцій / Горбенко О.Ю., Третяк В.А. / Сталий розвиток – XXI століття: управління, технології, моделі. Дискусії – 2018: колективна монографія / Міненко М.А. та ін.. НТУУ КПІ ім. Ігоря Сікорського, Національний університет “КиєвоМогилянська академія”; Вища економіко-гуманітарна школа. – Київ, 2018 . – С.467-472.

4. Estep, Eliot. Mobile html5: Eciency and performance of websockets and serversent events. / Estep, Eliot. [Електронний Ресурс] Master thesis, 3.3 Web techniques, June 2013. Режим доступу до ресурсу: http://goo.gl/n0TTHo.

5. Hickson, I. Server-sent events. W3C Working Draft WD-eventsource-20091222, / I. Hickson, [Електронний Ресурс] Режим доступу до ресурсу: http://www. w3. org/TR/eventsource, page 18.

6. RFC-6455. – Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011. Режим доступу до ресурсу: https://tools.ietf.org/html/rfc6455.

7. Furukawa, Y. Web-based control application using WebSocket. / Y. Furukawa, [Електронний Ресурс] European Synchrotron Radiation Facility ESRF, 38 Grenoble (France); 1423 p;ISSN 2226-0358; Worldcat; 2012; p. 695-697. Режим доступу до ресурсу : http://www.iaea.org/INIS/contacts.

8. Liu, Q. Research of web real-time communication based on web socket. / Liu, Q., Sun, X. // International Journal of Communications, Network and System Sciences, 5(2012), p. 797–801.

9. Crowther, R. HTML5 in Action. Manning / Crowther, R., Lennon, J., Blue, A., Wanish, G., Heilmann, C. / Publisher: Manning Publications Release – 2014. — 444 p.

10. Беседа Д.Г. Исследование технологии построения приложений реального времени с использованием протокола websocket / Беседа Д.Г., Семенистый Н.В., Аноприенко А.Я. // Конференция ИУС КМ - 2013: Донецк: Донецкий национальный технический университет, 2013. - С. 276-282.

11. RFC 7478 - Web Real-Time Communication Use Cases and Requirements Режим доступу до ресурсу: https://tools.ietf.org/html/rfc7478.

12. RFC 5245 - Interactive Connectivity Establishment ICE: A Protocol for Network Address Translator NAT Traversal for Offer/Answer Protocols Режим доступу до ресурсу: https://tools.ietf.org/html/rfc5245.

13. RFC-4566. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc4566.

14. Forouzan, B.A. TCP/IP Protocol Suite. / B.A. Forouzan / McGraw-Hill, New York, 2002 – 979p.

15. RFC6263 Marjou, X. and A. Sollaud, "Application Mechanism for Keeping Alive the NAT Mappings Associated with RTP / RTP Control Protocol (RTCP) Flows", RFC 6263, DOI 10.17487/RFC6263, June 2011, – Режим доступу до ресурсу: http://www.rfc-editor.org/info/rfc6263.

16. RFC 3489 Rosenberg J. STUN: Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). / J.Rosenberg, J. Weinberger, C. Huitema, R. Mahy. / RFC 3489, IETF, Mar. 2003.

17. RFC5766 Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

18. Mark W., ANSI/IEEE 1471 and systems engineering. / Mark W., Maier, David Emery, Rich Hilliard. / Systems Engineering 7.3— 2004. — 257-270p.

19. Шевчук Р.П. Підвищення ефективності клієнт-серверних систем середньої складності / Р.П. Шевчук., А.І. Яцинич // Вісник Тернопільського державного технічного університету. — 2010. — Том 15. — № 1. — С. 182—186.

20. Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software, /Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides / Addison-Wesley, Reading, MA, USA: Addisson-Wesley, 1995.

21. Berson, A. Client - server architecture. / Berson, Alex. / New York, NY : McGraw-Hill, 1992. - 452 p.

22. Пат. №. 6,996,800 Lucassen, John M., and Stephane H. Maes. "MVC (modelview-controller) based multi-modal authoring tool and development environment." U.S. Patent №. 6,996,800. 7 Feb. 2006

23. Krasner G. E. A description of the model-view-controller user interface paradigm in the smalltalk-80 system / Krasner, Glenn E., Stephen T. Pope //Journal of object oriented programming. – 1988. – Т. 1. – №. 3. – С. 26-49.

24. Mikowski M. Single page web applications: JavaScript end-to-end. / Mikowski M., Powell J. / – Manning Publications Co., 2013. - 432 p.

25. Flanagan, David. JavaScript: the definitive guide. " O'Reilly Media, Inc.", 2006. - 994 p.

26. Пат. № 7509584 США. Moser M. Dynamic ECMAScript class loading : пат. 7509584 США. – 2009.

27. ECMAScript I. S. O. ISO/IEC 16262: 1998 //ECMAScript Language Specification. Available from ECMA (European Computer Manufacturers Association) at http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM.

28. Schmiedehausen K. Single Page Application Architecture with Angular. /Schmiedehausen Kim. /Technology and Communication. 2018. - 44 p.

29. Фаулер М. Шаблоны корпоративных приложений. /М. Фаулер / Пер. с англ. — М.: ООО «ИД Вильямс», 2010. — 544 с

30. Вязовик Н. А. Программирование на Java / Н.А. Вязовик/ — М.: ИНТУИТ.РУ, 2016. – 604

31. Dziadosz, Radoslaw, et al. Liquibase: Version Control for Database Schema. 2017.

32. Eagle, Mark. Wiring your web application with open source java. 2004-04-07. http://www.onjava.com/pub/a/onjava/2004/04/07/wiringwebapps

33. Пат. № 7,565,443. 21 Rossmanith, Stefan, et al. "Common persistence layer." U.S. Patent № 7,565,443. 21 Jul. 2009

34. Furukawa, Y. Web-based control application using WebSocket. / Y.Furukawa, ICALEPCS, 2011, – 673-675

35. RFC 7395 Stout, L., Moffitt, J., Cestari, E. (2014). An extensible messaging and presence protocol (XMPP) subprotocol for websocket (№. RFC 7395) режим доступу: https://www.rfc-editor.org/rfc/pdfrfc/rfc7395.txt.pdf.

36. Dahl D. A. The W3C multimodal architecture and interfaces standard / D.A. Dahl //Journal on Multimodal User Interfaces. – 2013. – Т. 7. – №. 3. – С. 171-182.

37. Пат. № 15/364,676 STEPHEN, Gregor Leonard; BIRSE, Stuart. Data migration. U.S. Patent Application № 15/364,676, 2018.

ДОДАТОК А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

д.т.н., проф. О. Д. Азаров

"__" _____ 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи на тему:

«Інформаційна система для проведення відеоконференцій»

08-23.МКР.018.00.000 ТЗ

Науковий керівник к.т.н., доц. каф. ОТ

_____ Колесник І. С.

Студента групи 2КІ-21м

_____ Бабецький Е. В.

ВНТУ 2022

1 Підстава для використання МКР

1.1 Актуальність розробки полягає у вдосконаленні методу розгортання веб-додатку шляхом інтеграції тестування безпеки в CI/CD-процес з подальшою контейнеризацією.

1.2 Наказ про затвердження теми кваліфікаційної роботи №205-А від 15.09.2022 року.

2 Мета МКР і призначення розробки

2.1 Мета роботи — аналіз технологій передачі даних та створення програмного продукту для проведення відеоконференції;

2.2 Призначення розробки — покращити способи створення серверної частини програмних продуктів за допомогою технології WebRTC.

3 Вихідні дані для виконання МКР

3.1 Проведення аналізу існуючих принципів та технології WebRTC;

3.2 Розробка структури програмного засобу веб-програми;

3.3 Середовище розробки WebStorm для платформи JavaScript;

4 Вимоги до виконання МКР

4.1 Наявність веб-програми для проведення відеоконференцій;

4.2 Наявність системи передачі відео та аудіо звязку між користувачами за допомогою відеоконференції.

5 Етапи МКР та очікувані результати

Робота виконується за п’ять етапів, таблиця А.1.

Таблиця А.1 — Етапи виконання МКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз сучасного стану досліджень в галузі передачі відео аудіо між браузерами	04.10.2022	15.10.2022	Аналітичний огляд літературних джерел
2	Побудова структурних моделей системи тестування	16.10.2022	04.11.2022	Структурні моделі, 2 розділ
3	Практичне застосування та оцінка ефективності розроблених моделей	05.11.2022	30.11.2022	3 і 4 розділи
4	Підготовка економічної частини	30.11.2022	03.12.2022	5 розділ
5	Оформлення пояснювальної записки, графічного матеріалу і/або презентації	04.12.2022	18.12.2022	пояснювальна записка, графічний матеріал і/або презентація

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відзив наукового керівника, відзив опонента, протоколи проходження перевірки на плагіат, анотації до МКР українською та іноземною мовами, нормоконтроль про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами.

Захист МКР відбувається на засіданні екзаменаційної комісії, затверджено наказом ректора.

8 Вимоги до оформлення МКР

8.1 При оформлювання МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Едина система конструкторської документації. Основні написи»;

— Методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп'ютерна інженерія». Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК Б

Діаграма класів управління відеоконференціями

l

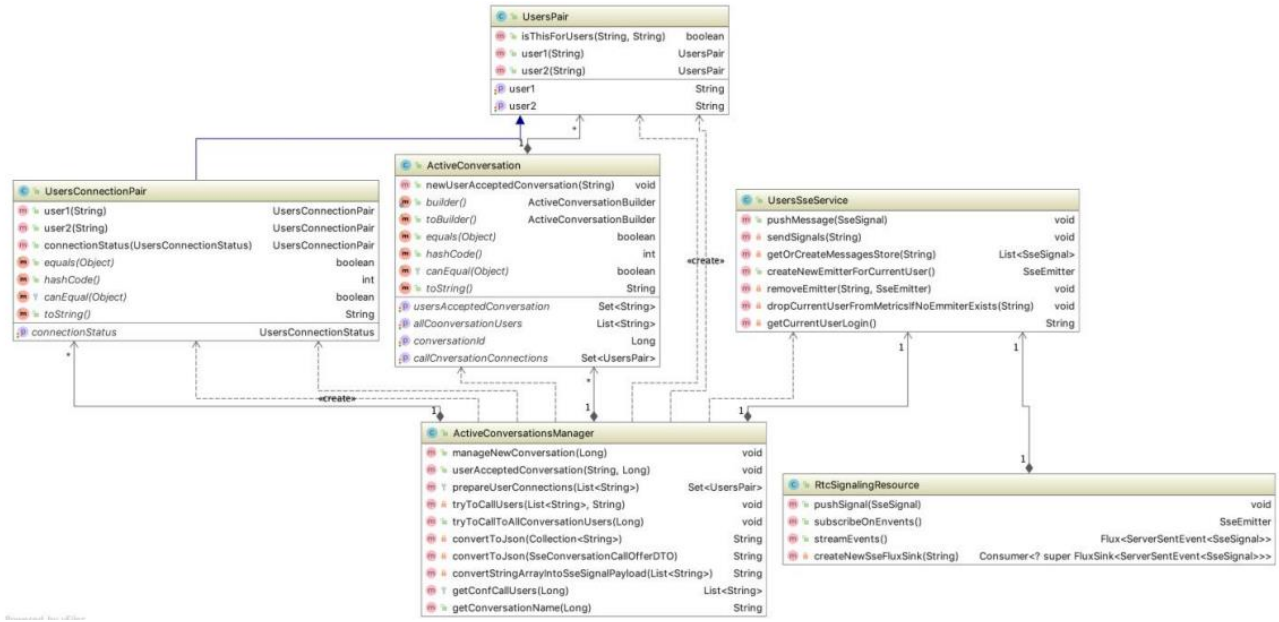

Рисунок Б.1 — Діаграма класів управління відеоконференціями сигнального сервера

ДОДАТОК В

Діаграма послідовностей налагодження каналу зв’язку за технологією WebRTC

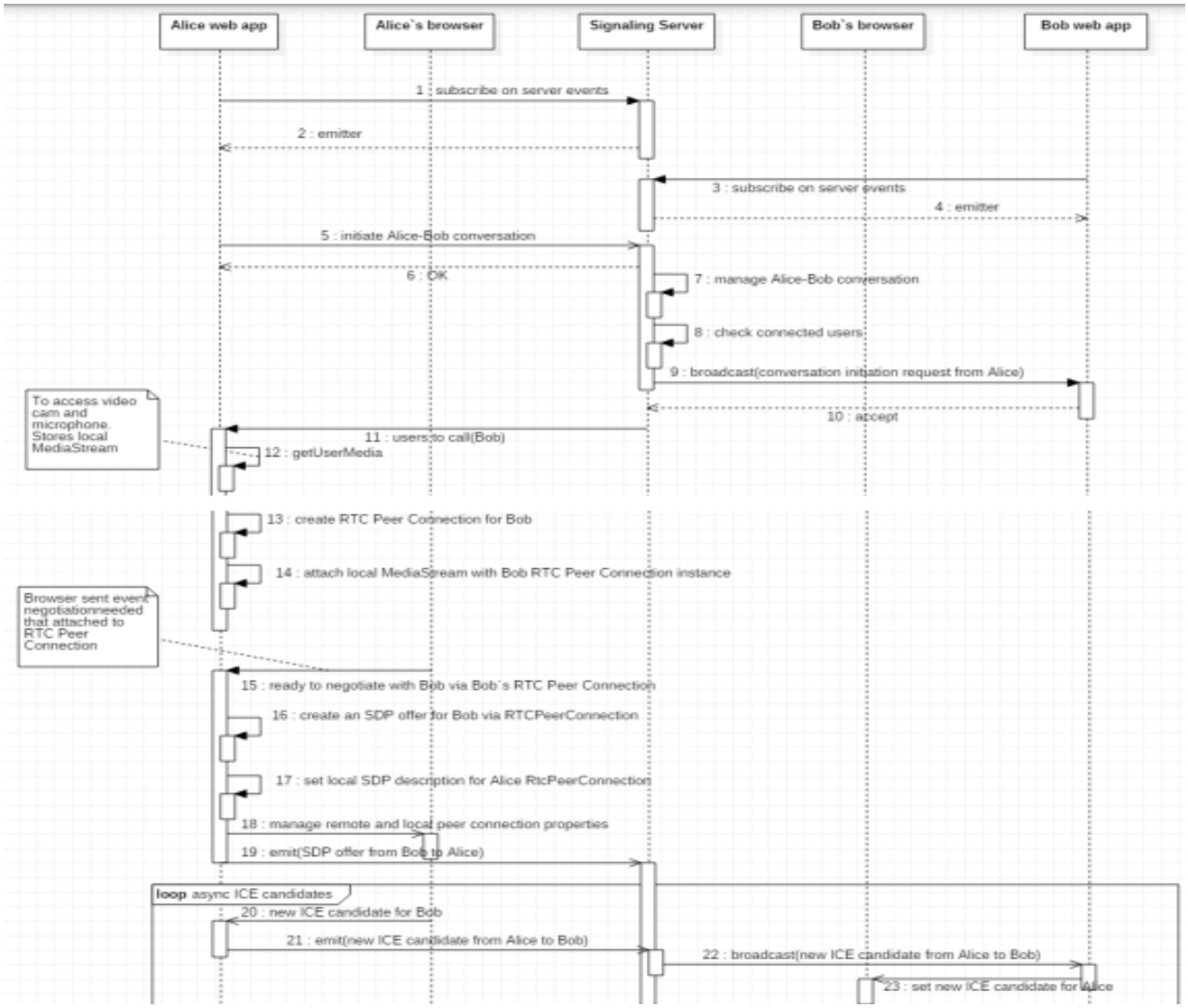

Рисунок В.1 — Діаграма послідовностей налагодження каналу зв’язку за технологією WebRTC

ДОДАТОК Г

Взаємодія класів, що відповідають за налаштування WebRTC з’єднань

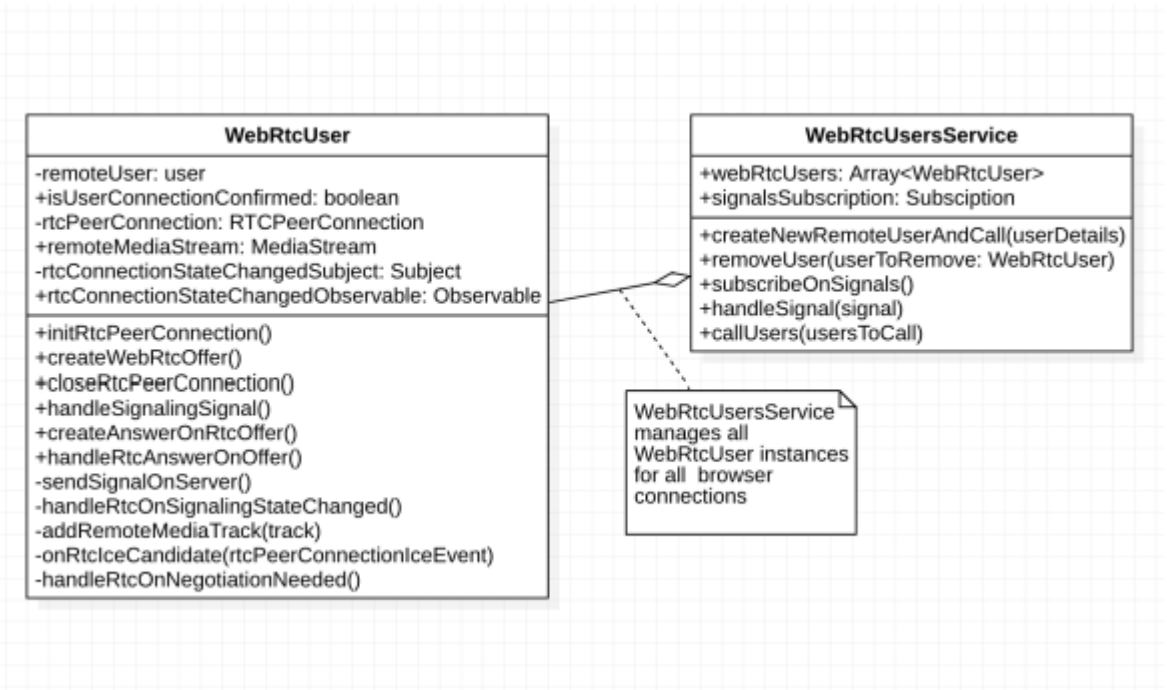

Рисунок Г.1 — Взаємодія класів, що відповідають за налаштування WebRTC з’єднань

ДОДАТОК Д

Entity Relationship діаграма сутностей БД

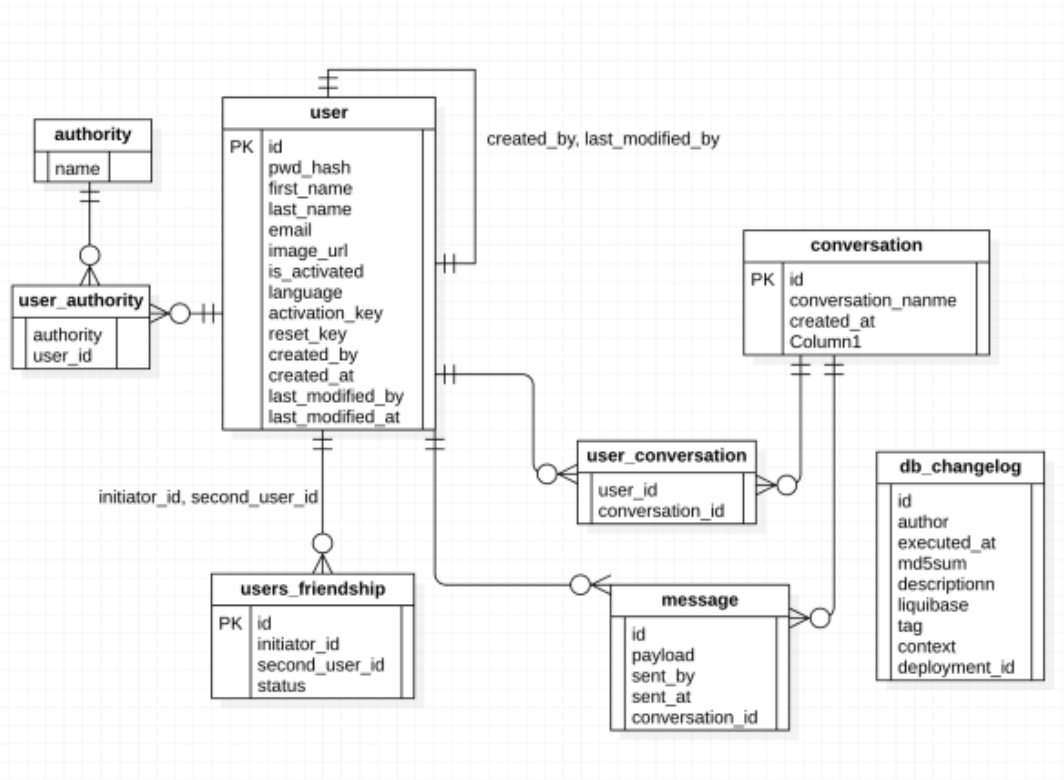

Рисунок Д.1 — Entity Relationship діаграма сутностей БД

ДОДАТОК Е

Лістинг програмного коду для керування функціоналу створення та підтримки відеочату за допомогою WebRTC

```
import {io} from 'socket.io-client';
const options = {
  "force new connection": true,
  reconnectionAttempts: "Infinity", // avoid having user reconnect manually in order to prevent dead clients after a server restart
  timeout : 10000, // before connect_error and connect_timeout are emitted.
  transports : ["websocket"]
}
const socket = io('/', options);
export default socket;
const path = require('path');
const express = require('express');
const app = express();
const server = require('http').createServer(app);
const io = require('socket.io')(server);
const {version, validate} = require('uuid');
const ACTIONS = require('./src/socket/actions');
const PORT = process.env.PORT || 3001;
function getClientRooms() {
  const {rooms} = io.sockets.adapter;
  return Array.from(rooms.keys()).filter(roomID => validate(roomID) && version(roomID) === 4);
}
function shareRoomsInfo() {
```

```
  io.emit(ACTIONS.SHARE_ROOMS, {
    rooms: getClientRooms()
  })
}
io.on('connection', socket => {
  shareRoomsInfo();
  socket.on(ACTIONS.JOIN, config => {
    const {room: roomID} = config;
    const {rooms: joinedRooms} = socket;
    if (Array.from(joinedRooms).includes(roomID)) {
      return console.warn(`Already joined to ${roomID}`);
    }
    const clients = Array.from(io.sockets.adapter.rooms.get(roomID) || []);
    clients.forEach(clientID => {
      io.to(clientID).emit(ACTIONS.ADD_PEER, {
        peerID: socket.id,
        createOffer: false
      });
      socket.emit(ACTIONS.ADD_PEER, {
        peerID: clientID,
        createOffer: true,
      });
    });
    socket.join(roomID);
    shareRoomsInfo();
  });
  function leaveRoom() {
    const {rooms} = socket;
    Array.from(rooms)
      // LEAVE ONLY CLIENT CREATED ROOM
```

```
    .filter(roomID => validate(roomID) && version(roomID) === 4)
    .forEach(roomID => {
      const clients = Array.from(io.sockets.adapter.rooms.get(roomID) || []);
      clients
        .forEach(clientID => {
        io.to(clientID).emit(ACTIONS.REMOVE_PEER, {
          peerID: socket.id,
        });
        socket.emit(ACTIONS.REMOVE_PEER, {
          peerID: clientID,
        });
      });
      socket.leave(roomID);
    });
  shareRoomsInfo();
}
socket.on(ACTIONS.LEAVE, leaveRoom);
socket.on('disconnecting', leaveRoom);
socket.on(ACTIONS.RELAY_SDP, ({peerID, sessionDescription}) => {
  io.to(peerID).emit(ACTIONS.SESSION_DESCRIPTION, {
    peerID: socket.id,
    sessionDescription,
  });
});
socket.on(ACTIONS.RELAY_ICE, ({peerID, iceCandidate}) => {
  io.to(peerID).emit(ACTIONS.ICE_CANDIDATE, {
    peerID: socket.id,
    iceCandidate,
  });
});
```

```
});

const publicPath = path.join(__dirname, 'build');
app.use(express.static(publicPath));
app.get('*', (req, res) => {
  res.sendFile(path.join(publicPath, 'index.html'));
});
server.listen(PORT, () => {
  console.log('Server Started!')
})
import {useState, useCallback, useRef, useEffect} from 'react';
const useStateWithCallback = initialState => {
  const [state, setState] = useState(initialState);
  const cbRef = useRef(null);
  const updateState = useCallback((newState, cb) => {
    cbRef.current = cb;
    setState(prev => typeof newState === 'function' ? newState(prev) : newState);
  }, []);
  useEffect(() => {
    if (cbRef.current) {
      cbRef.current(state);
      cbRef.current = null;
    }
  }, [state]);
  return [state, updateState];
}
export default useStateWithCallback;
import {useEffect, useRef, useCallback} from 'react';
import freeice from 'freeice';
import useStateWithCallback from './useStateWithCallback';
```

```
import socket from '../socket';
import ACTIONS from '../socket/actions';
export const LOCAL_VIDEO = 'LOCAL_VIDEO';
export default function useWebRTC(roomID) {
  const [clients, updateClients] = useStateWithCallback([]);
  const addNewClient = useCallback((newClient, cb) => {
    updateClients(list => {
      if (!list.includes(newClient)) {
        return [...list, newClient]
      }
      return list;
    }, cb);
  }, [clients, updateClients]);

  const peerConnections = useRef({});
  const localMediaStream = useRef(null);
  const peerMediaElements = useRef({
    [LOCAL_VIDEO]: null,
  });
  useEffect(() => {
    async function handleNewPeer({peerID, createOffer}) {
      if (peerID in peerConnections.current) {
        return console.warn(`Already connected to peer ${peerID}`);
      }
      peerConnections.current[peerID] = new RTCPeerConnection({
        iceServers: freeice(),
      });
      peerConnections.current[peerID].onicecandidate = event => {
        if (event.candidate) {
          socket.emit(ACTIONS.RELAY_ICE, {
```

```
      peerID,
      iceCandidate: event.candidate,
    });
  }
}
let tracksNumber = 0;
peerConnections.current[peerID].ontrack = ({streams: [remoteStream]}) => {
  tracksNumber++
  if (tracksNumber === 2) { // video & audio tracks received
    tracksNumber = 0;
    addNewClient(peerID, () => {
      if (peerMediaElements.current[peerID]) {
        peerMediaElements.current[peerID].srcObject = remoteStream;
      } else {
        // FIX LONG RENDER IN CASE OF MANY CLIENTS
        let settled = false;
        const interval = setInterval(() => {
          if (peerMediaElements.current[peerID]) {
            peerMediaElements.current[peerID].srcObject = remoteStream;
            settled = true;
          }
          if (settled) {
            clearInterval(interval);
          }
        }, 1000);
      }
    });
  }
}
localMediaStream.current.getTracks().forEach(track => {
```

```
        peerConnections.current[peerID].addTrack(track, localMediaStream.current);
      });
      if (createOffer) {
        const offer = await peerConnections.current[peerID].createOffer();
        await peerConnections.current[peerID].setLocalDescription(offer);
        socket.emit(ACTIONS.RELAY_SDP, {
          peerID,
          sessionDescription: offer,
        });
      }
    }
    socket.on(ACTIONS.ADD_PEER, handleNewPeer);
    return () => {
      socket.off(ACTIONS.ADD_PEER);
    }
  }, []);
  useEffect(() => {
    async function setRemoteMedia({peerID, sessionDescription:
remoteDescription}) {
      await peerConnections.current[peerID]?.setRemoteDescription(
        new RTCSessionDescription(remoteDescription)
      );
      if (remoteDescription.type === 'offer') {
        const answer = await peerConnections.current[peerID].createAnswer();
        await peerConnections.current[peerID].setLocalDescription(answer);
        socket.emit(ACTIONS.RELAY_SDP, {
          peerID,
          sessionDescription: answer,
        });
      }
```

```
  }
  socket.on(ACTIONS.SESSION_DESCRIPTION, setRemoteMedia)

  return () => {
    socket.off(ACTIONS.SESSION_DESCRIPTION);
  }
}, []);
useEffect(() => {
  socket.on(ACTIONS.ICE_CANDIDATE, ({peerID, iceCandidate}) => {
    peerConnections.current[peerID]?.addIceCandidate(
      new RTCIceCandidate(iceCandidate)
    );
  });
  return () => {
    socket.off(ACTIONS.ICE_CANDIDATE);
  }
}, []);
useEffect(() => {
  const handleRemovePeer = ({peerID}) => {
    if (peerConnections.current[peerID]) {
      peerConnections.current[peerID].close();
    }
    delete peerConnections.current[peerID];
    delete peerMediaElements.current[peerID];
    updateClients(list => list.filter(c => c !== peerID));
  };
  socket.on(ACTIONS.REMOVE_PEER, handleRemovePeer);
  return () => {
    socket.off(ACTIONS.REMOVE_PEER);
  }
```

```
}, []);
useEffect(() => {
  async function startCapture() {
    localMediaStream.current = await navigator.mediaDevices.getUserMedia({
      audio: true,
      video: {
        width: 1280,
        height: 720,
      }
    });
    addNewClient(LOCAL_VIDEO, () => {
      const localVideoElement = peerMediaElements.current[LOCAL_VIDEO];
      if (localVideoElement) {
        localVideoElement.volume = 0;
        localVideoElement.srcObject = localMediaStream.current;
      }
    });
  }
  startCapture()
    .then(() => socket.emit(ACTIONS.JOIN, {room: roomID}))
    .catch(e => console.error('Error getting userMedia:', e));
  return () => {
    localMediaStream.current.getTracks().forEach(track => track.stop());
    socket.emit(ACTIONS.LEAVE);
  };
}, [roomID]);
const provideMediaRef = useCallback((id, node) => {
  peerMediaElements.current[id] = node;
}, []);
return {
```

```
    clients,
    provideMediaRef
  };
}
import {useParams} from 'react-router';
import useWebRTC, {LOCAL_VIDEO} from '../../hooks/useWebRTC';
function layout(clientsNumber = 1) {
  const pairs = Array.from({length: clientsNumber})
    .reduce((acc, next, index, arr) => {
      if (index % 2 === 0) {
        acc.push(arr.slice(index, index + 2));
      }
      return acc;
    }, []);
  const rowsNumber = pairs.length;
  const height = `${100 / rowsNumber}%`;
  return pairs.map((row, index, arr) => {
    if (index === arr.length - 1 && row.length === 1) {
      return [{
        width: '100%',
        height,
      }];
    }
    return row.map(() => ({
      width: '50%',
      height,
    }));
  }).flat();
}
export default function Room() {
```

```
const {id: roomID} = useParams();
const {clients, provideMediaRef} = useWebRTC(roomID);
const videoLayout = layout(clients.length);

return (
  <div style={{
    display: 'flex',
    alignItems: 'center',
    justifyContent: 'center',
    flexWrap: 'wrap',
    height: '100vh',
  }}>
    {clients.map((clientID, index) => {
      return (
        <div key={clientID} style={videoLayout[index]} id={clientID}>
          <video
            width='100%'
            height='100%'
            ref={instance => {
              provideMediaRef(clientID, instance);
            }}
            autoPlay
            playsInline
            muted={clientID === LOCAL_VIDEO}
          />
        </div>
      );
    })}
  </div>
);
```

```
}
import {io} from 'socket.io-client';
const options = {
  "force new connection": true,
  reconnectionAttempts: "Infinity", // avoid having user reconnect manually in order
to prevent dead clients after a server restart
  timeout : 10000, // before connect_error and connect_timeout are emitted.
  transports : ["websocket"]
}
const socket = io('/', options);
export default socket;
import {useEffect, useRef, useCallback} from 'react';
import freeice from 'freeice';
import useStateWithCallback from './useStateWithCallback';
import socket from '../socket';
import ACTIONS from '../socket/actions';
export const LOCAL_VIDEO = 'LOCAL_VIDEO';
export default function useWebRTC(roomID) {
  const [clients, updateClients] = useStateWithCallback([]);
  const addNewClient = useCallback((newClient, cb) => {
    updateClients(list => {
      if (!list.includes(newClient)) {
        return [...list, newClient]
      }
      return list;
    }, cb);
  }, [clients, updateClients]);
  const peerConnections = useRef({});
  const localMediaStream = useRef(null);
  const peerMediaElements = useRef({
```

```
  [LOCAL_VIDEO]: null,
});
useEffect(() => {
  async function handleNewPeer({peerID, createOffer}) {
    if (peerID in peerConnections.current) {
      return console.warn(`Already connected to peer ${peerID}`);
    }
    peerConnections.current[peerID] = new RTCPeerConnection({
      iceServers: freeice(),
    });
    peerConnections.current[peerID].onicecandidate = event => {
      if (event.candidate) {
        socket.emit(ACTIONS.RELAY_ICE, {
          peerID,
          iceCandidate: event.candidate,
        });
      }
    }
    let tracksNumber = 0;
    peerConnections.current[peerID].ontrack = ({streams: [remoteStream]}) => {
      tracksNumber++
      if (tracksNumber === 2) { // video & audio tracks received
        tracksNumber = 0;
        addNewClient(peerID, () => {
          if (peerMediaElements.current[peerID]) {
            peerMediaElements.current[peerID].srcObject = remoteStream;
          } else {
            // FIX LONG RENDER IN CASE OF MANY CLIENTS
            let settled = false;
            const interval = setInterval(() => {
```

```
            if (peerMediaElements.current[peerID]) {
              peerMediaElements.current[peerID].srcObject = remoteStream;
              settled = true
            if (settled) {
              clearInterval(interval);
            }
           }, 1000);
         }
        });
      }
    }
    localMediaStream.current.getTracks().forEach(track => {
      peerConnections.current[peerID].addTrack(track, localMediaStream.current);
    });
    if (createOffer) {
      const offer = await peerConnections.current[peerID].createOffer();
      await peerConnections.current[peerID].setLocalDescription(offer);
      socket.emit(ACTIONS.RELAY_SDP, {
        peerID,
        sessionDescription: offer,
      });
    }
  }
  socket.on(ACTIONS.ADD_PEER, handleNewPeer);
  return () => {
    socket.off(ACTIONS.ADD_PEER);
  }
}, []);
useEffect(() => {
```

```
  async function setRemoteMedia({peerID, sessionDescription: remoteDescription}) {
    await peerConnections.current[peerID]?.setRemoteDescription(
      new RTCSessionDescription(remoteDescription)
    );
    if (remoteDescription.type === 'offer') {
      const answer = await peerConnections.current[peerID].createAnswer();

      await peerConnections.current[peerID].setLocalDescription(answer);

      socket.emit(ACTIONS.RELAY_SDP, {
        peerID,
        sessionDescription: answer,
      });
    }
  }
  socket.on(ACTIONS.SESSION_DESCRIPTION, setRemoteMedia)
  return () => {
    socket.off(ACTIONS.SESSION_DESCRIPTION);
  }
 }, []);
 useEffect(() => {
  socket.on(ACTIONS.ICE_CANDIDATE, ({peerID, iceCandidate}) => {
    peerConnections.current[peerID]?.addIceCandidate(
      new RTCIceCandidate(iceCandidate)
    );
  });
  return () => {
    socket.off(ACTIONS.ICE_CANDIDATE);
  }
```

```
}, []);
useEffect(() => {
  const handleRemovePeer = ({peerID}) => {
    if (peerConnections.current[peerID]) {
      peerConnections.current[peerID].close();
    }
    delete peerConnections.current[peerID];
    delete peerMediaElements.current[peerID];
    updateClients(list => list.filter(c => c !== peerID));
  };
  socket.on(ACTIONS.REMOVE_PEER, handleRemovePeer);
  return () => {
    socket.off(ACTIONS.REMOVE_PEER);
  }
}, []);
useEffect(() => {
  async function startCapture() {
    localMediaStream.current = await navigator.mediaDevices.getUserMedia({
      audio: true,
      video: {
        width: 1280,
        height: 720,
      }
    });
    addNewClient(LOCAL_VIDEO, () => {
      const localVideoElement = peerMediaElements.current[LOCAL_VIDEO];
      if (localVideoElement) {
        localVideoElement.volume = 0;
        localVideoElement.srcObject = localMediaStream.current;
      }
```

```
      });
    }
    startCapture()
      .then(() => socket.emit(ACTIONS.JOIN, {room: roomID}))
      .catch(e => console.error('Error getting userMedia:', e));
    return () => {
      localMediaStream.current.getTracks().forEach(track => track.stop());
      socket.emit(ACTIONS.LEAVE);
    };
  }, [roomID]);
  const provideMediaRef = useCallback((id, node) => {
    peerMediaElements.current[id] = node;
  }, []);
  return {
    clients,
    provideMediaRef
  };
}
export default useStateWithCallback;
import {useEffect, useRef, useCallback} from 'react';
import freeice from 'freeice';
import useStateWithCallback from './useStateWithCallback';
import socket from '../socket';
import ACTIONS from '../socket/actions';
export const LOCAL_VIDEO = 'LOCAL_VIDEO';
export default function useWebRTC(roomID) {
  const [clients, updateClients] = useStateWithCallback([]);
  const addNewClient = useCallback((newClient, cb) => {
    updateClients(list => {
      if (!list.includes(newClient)) {
```

```
      return [...list, newClient]
    }
    return list;
  }, cb);
}, [clients, updateClients]);

const peerConnections = useRef({});
const localMediaStream = useRef(null);
const peerMediaElements = useRef({
  [LOCAL_VIDEO]: null,
});
useEffect(() => {
  async function handleNewPeer({peerID, createOffer}) {
    if (peerID in peerConnections.current) {
      return console.warn(`Already connected to peer ${peerID}`);
    }
    peerConnections.current[peerID] = new RTCPeerConnection({
      iceServers: freeice(),
    });
    peerConnections.current[peerID].onicecandidate = event => {
      if (event.candidate) {
        socket.emit(ACTIONS.RELAY_ICE, {
          peerID,
          iceCandidate: event.candidate,
        });
      }
    }
    let tracksNumber = 0;
    peerConnections.current[peerID].ontrack = ({streams: [remoteStream]}) => {
      tracksNumber++
```

```
  if (tracksNumber === 2) { // video & audio tracks received
    tracksNumber = 0;
    addNewClient(peerID, () => {
      if (peerMediaElements.current[peerID]) {
        peerMediaElements.current[peerID].srcObject = remoteStream;
      } else {
        // FIX LONG RENDER IN CASE OF MANY CLIENTS
        let settled = false;
        const interval = setInterval(() => {
          if (peerMediaElements.current[peerID]) {
            peerMediaElements.current[peerID].srcObject = remoteStream;
            settled = true;
          }
          if (settled) {
            clearInterval(interval);
          }
        }, 1000);
      }
    });
  }
}
localMediaStream.current.getTracks().forEach(track => {
  peerConnections.current[peerID].addTrack(track, localMediaStream.current);
});
if (createOffer) {
  const offer = await peerConnections.current[peerID].createOffer();
  await peerConnections.current[peerID].setLocalDescription(offer);
  socket.emit(ACTIONS.RELAY_SDP, {
    peerID,
    sessionDescription: offer,
```

```
        });
      }
    }
    socket.on(ACTIONS.ADD_PEER, handleNewPeer);
    return () => {
      socket.off(ACTIONS.ADD_PEER);
    }
  }, []);
  useEffect(() => {
    async function setRemoteMedia({peerID, sessionDescription: remoteDescription}) {
      await peerConnections.current[peerID]?.setRemoteDescription(
        new RTCSessionDescription(remoteDescription)
      );
      if (remoteDescription.type === 'offer') {
        const answer = await peerConnections.current[peerID].createAnswer();
        await peerConnections.current[peerID].setLocalDescription(answer);
        socket.emit(ACTIONS.RELAY_SDP, {
          peerID,
          sessionDescription: answer,
        });
      }
    }
    socket.on(ACTIONS.SESSION_DESCRIPTION, setRemoteMedia)

    return () => {
      socket.off(ACTIONS.SESSION_DESCRIPTION);
    }
  }, []);
  useEffect(() => {
```

```
  socket.on(ACTIONS.ICE_CANDIDATE, ({peerID, iceCandidate}) => {
    peerConnections.current[peerID]?.addIceCandidate(
      new RTCIceCandidate(iceCandidate)
    );
  });
  return () => {
    socket.off(ACTIONS.ICE_CANDIDATE);
  }
}, []);
useEffect(() => {
  const handleRemovePeer = ({peerID}) => {
    if (peerConnections.current[peerID]) {
      peerConnections.current[peerID].close();
    }
    delete peerConnections.current[peerID];
    delete peerMediaElements.current[peerID];
    updateClients(list => list.filter(c => c !== peerID));
  };
  socket.on(ACTIONS.REMOVE_PEER, handleRemovePeer);
  return () => {
    socket.off(ACTIONS.REMOVE_PEER);
  }
}, []);
useEffect(() => {
  async function startCapture() {
    localMediaStream.current = await navigator.mediaDevices.getUserMedia({
      audio: true,
      video: {
        width: 1280,
        height: 720,
```

```
        }
      });
      addNewClient(LOCAL_VIDEO, () => {
        const localVideoElement = peerMediaElements.current[LOCAL_VIDEO];
        if (localVideoElement) {
          localVideoElement.volume = 0;
          localVideoElement.srcObject = localMediaStream.current;
        }
      });
    }
    startCapture()
      .then(() => socket.emit(ACTIONS.JOIN, {room: roomID}))
      .catch(e => console.error('Error getting userMedia:', e));
    return () => {
      localMediaStream.current.getTracks().forEach(track => track.stop());
      socket.emit(ACTIONS.LEAVE);
    };
  }, [roomID]);
  const provideMediaRef = useCallback((id, node) => {
    peerMediaElements.current[id] = node;
  }, []);
  return {
    clients,
    provideMediaRef
  };
}
import {useParams} from 'react-router';
import useWebRTC, {LOCAL_VIDEO} from '../../hooks/useWebRTC';
function layout(clientsNumber = 1) {
  const pairs = Array.from({length: clientsNumber})
```

```
    .reduce((acc, next, index, arr) => {
      if (index % 2 === 0) {
        acc.push(arr.slice(index, index + 2));
      }
      return acc;
    }, []);
  const rowsNumber = pairs.length;
  const height = `${100 / rowsNumber}%`;
  return pairs.map((row, index, arr) => {
    if (index === arr.length - 1 && row.length === 1) {
      return [{
        width: '100%',
        height,
      }];
    }
    return row.map(() => ({
      width: '50%',
      height,
    }));
  }).flat();
}
export default function Room() {
  const {id: roomID} = useParams();
  const {clients, provideMediaRef} = useWebRTC(roomID);
  const videoLayout = layout(clients.length);

  return (
    <div style={{
      display: 'flex',
      alignItems: 'center',
```

```
      justifyContent: 'center',
      flexWrap: 'wrap',
      height: '100vh',
    }}>
      {clients.map((clientID, index) => {
        return (
          <div key={clientID} style={videoLayout[index]} id={clientID}>
            <video
              width='100%'
              height='100%'
              ref={instance => {
                provideMediaRef(clientID, instance);
              }}
              autoPlay
              playsInline
              muted={clientID === LOCAL_VIDEO}
            />
          </div>
        );
      })}
    </div>
  );
}
import {io} from 'socket.io-client';
const options = {
  "force new connection": true,
  reconnectionAttempts: "Infinity", // avoid having user reconnect manually in order
to prevent dead clients after a server restart
  timeout : 10000, // before connect_error and connect_timeout are emitted.
  transports : ["websocket"]
```

```
}
const socket = io('/', options);
export default socket;
import {useEffect, useRef, useCallback} from 'react';
import freeice from 'freeice';
import useStateWithCallback from './useStateWithCallback';
import socket from '../socket';
import ACTIONS from '../socket/actions';
export const LOCAL_VIDEO = 'LOCAL_VIDEO';
export default function useWebRTC(roomID) {
  const [clients, updateClients] = useStateWithCallback([]);
  const addNewClient = useCallback((newClient, cb) => {
    updateClients(list => {
      if (!list.includes(newClient)) {
        return [...list, newClient]
      }
      return list;
    }, cb);
  }, [clients, updateClients]);
  const peerConnections = useRef({});
  const localMediaStream = useRef(null);
  const peerMediaElements = useRef({
    [LOCAL_VIDEO]: null,
  });
  useEffect(() => {
    async function handleNewPeer({peerID, createOffer}) {
      if (peerID in peerConnections.current) {
        return console.warn(`Already connected to peer ${peerID}`);
      }
      peerConnections.current[peerID] = new RTCPeerConnection({
```

```
  iceServers: freeice(),
});
peerConnections.current[peerID].onicecandidate = event => {
  if (event.candidate) {
    socket.emit(ACTIONS.RELAY_ICE, {
      peerID,
      iceCandidate: event.candidate,
    });
  }
}
let tracksNumber = 0;
peerConnections.current[peerID].ontrack = ({streams: [remoteStream]}) => {
  tracksNumber++
  if (tracksNumber === 2) { // video & audio tracks received
    tracksNumber = 0;
    addNewClient(peerID, () => {
      if (peerMediaElements.current[peerID]) {
        peerMediaElements.current[peerID].srcObject = remoteStream;
      } else {
        // FIX LONG RENDER IN CASE OF MANY CLIENTS
        let settled = false;
        const interval = setInterval(() => {
          if (peerMediaElements.current[peerID]) {
            peerMediaElements.current[peerID].srcObject = remoteStream;
            settled = true
          if (settled) {
            clearInterval(interval);
          }
        }, 1000);
      }
```

```
        });
      }
    }
    localMediaStream.current.getTracks().forEach(track => {
      peerConnections.current[peerID].addTrack(track, localMediaStream.current);
    });
    if (createOffer) {
      const offer = await peerConnections.current[peerID].createOffer();
      await peerConnections.current[peerID].setLocalDescription(offer);
      socket.emit(ACTIONS.RELAY_SDP, {
        peerID,
        sessionDescription: offer,
      });
    }
  }
  socket.on(ACTIONS.ADD_PEER, handleNewPeer);
  return () => {
    socket.off(ACTIONS.ADD_PEER);
  }
 }, []);
 useEffect(() => {
  async function setRemoteMedia({peerID, sessionDescription:
remoteDescription}) {
    await peerConnections.current[peerID]?.setRemoteDescription(
      new RTCSessionDescription(remoteDescription)
    );
    if (remoteDescription.type === 'offer') {
      const answer = await peerConnections.current[peerID].createAnswer();

      await peerConnections.current[peerID].setLocalDescription(answer);
```

```
      socket.emit(ACTIONS.RELAY_SDP, {
        peerID,
        sessionDescription: answer,
      });
    }
  }
  socket.on(ACTIONS.SESSION_DESCRIPTION, setRemoteMedia)
  return () => {
    socket.off(ACTIONS.SESSION_DESCRIPTION);
  }
}, []);
useEffect(() => {
  socket.on(ACTIONS.ICE_CANDIDATE, ({peerID, iceCandidate}) => {
    peerConnections.current[peerID]?.addIceCandidate(
      new RTCIceCandidate(iceCandidate)
    );
  });
  return () => {
    socket.off(ACTIONS.ICE_CANDIDATE);
  }
}, []);
useEffect(() => {
  const handleRemovePeer = ({peerID}) => {
    if (peerConnections.current[peerID]) {
      peerConnections.current[peerID].close();
    }
    delete peerConnections.current[peerID];
    delete peerMediaElements.current[peerID];
    updateClients(list => list.filter(c => c !== peerID));
```

```
  };
  socket.on(ACTIONS.REMOVE_PEER, handleRemovePeer);
  return () => {
    socket.off(ACTIONS.REMOVE_PEER);
  }
}, []);
useEffect(() => {
  async function startCapture() {
    localMediaStream.current = await navigator.mediaDevices.getUserMedia({
      audio: true,
      video: {
        width: 1280,
        height: 720,
      }
    });
    addNewClient(LOCAL_VIDEO, () => {
      const localVideoElement = peerMediaElements.current[LOCAL_VIDEO];
      if (localVideoElement) {
        localVideoElement.volume = 0;
        localVideoElement.srcObject = localMediaStream.current;
      }
    });
  }
  startCapture()
    .then(() => socket.emit(ACTIONS.JOIN, {room: roomID}))
    .catch(e => console.error('Error getting userMedia:', e));
  return () => {
    localMediaStream.current.getTracks().forEach(track => track.stop());
    socket.emit(ACTIONS.LEAVE);
  };
```

```
  }, [roomID]);
  useEffect(() => {
    const handleRemovePeer = ({peerID}) => {
      if (peerConnections.current[peerID]) {
        peerConnections.current[peerID].close();
      }
      delete peerConnections.current[peerID];
      delete peerMediaElements.current[peerID];
      updateClients(list => list.filter(c => c !== peerID));
    };
  useEffect(() => {
    async function setRemoteMedia({peerID, sessionDescription:
remoteDescription}) {
      await peerConnections.current[peerID]?.setRemoteDescription(
        new RTCSessionDescription(remoteDescription)
      );
      if (remoteDescription.type === 'offer') {
        const answer = await peerConnections.current[peerID].createAnswer();

        await peerConnections.current[peerID].setLocalDescription(answer);

        socket.emit(ACTIONS.RELAY_SDP, {
          peerID,
          sessionDescription: answer,
        });}}
    socket.on(ACTIONS.SESSION_DESCRIPTION, setRemoteMedia)
    return () => {
      socket.off(ACTIONS.SESSION_DESCRIPTION);}}, []);
```

ДОДАТОК Ж

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи:

_____

_____

Тип роботи: магістерська кваліфікаційна робота

(БДР, МКР)

Підрозділ кафедра обчислювальної техніки

_____

(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 82.5% Схожість 17.5%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.

(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____

_____

(підпис) (прізвище, ініціали)

Керівник роботи _____

_____

(підпис) (прізвище, ініціали)