ДВНЗ «Донецький національний технічний університет»
Факультет комп'ютерно інформаційних технологій та автоматизації
Кафедра прикладної математики та інформатики

«До захисту допущено»
Завідувачка кафедри ПМІ

_____ Ольга ДМИТРІЄВА
(підпис) (ім'я та прізвище)

«14» червня 2022р.

Випускна кваліфікаційна робота

бакалавр
(освітній ступінь)

на тему «Розробка платформи для організації групових відеодзвінків» із спецчастиною «Розробка клієнтської та серверної частини платформи для проведення групових відеодзвінків з використанням технології WebRTC з топологією SFU, засобами JavaScript, Node.js, MongoDB, Docker, AWS »
Виконала: студентка 4 курсу, групи ІПЗ-18
(шифр групи)
спеціальності 121 «Інженерія програмного забезпечення»

Коврижко Н. І. _____
(прізвище та ініціали) (підпис)

Керівник доц. каф. ПМІ, к.т.н., доц. Ірина НАЗАРОВА _____
(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Рецензент зав. каф. каф. КІ, к.т.н., доц. Сергій КОВАЛЬОВ _____
(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Нормоконтроль:

_____ к.т.н., доц. Ірина НАЗАРОВА
(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Луцьк – 2022

ДВНЗ «Донецький національний технічний університет»
Факультет комп'ютерних наук технологій
Кафедра прикладної математики та інформатики
Освітньо-кваліфікаційний рівень (освітній ступінь) бакалавр
Напрям підготовки (спеціальність) 121 Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Завідувачка кафедри ПМІ

Ольга ДМИТРІЄВА

"6" 04. 2022 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ

Наталія Коврижко
(ім'я, прізвище)

1. Тема проєкту (роботи) Розробка платформи для організації групових відеодзвінків
Спецчастина Розробка клієнтської та серверної частини платформи для проведення групових відеодзвінків з використанням технології WebRTC з топологією SFU, засобами JavaScript, Node.js, MongoDB, Docker, AWS.
Керівник проєкту (роботи) Ірина НАЗАРОВА, доц. каф. ПМІ, к.т.н., доц.
(ім'я, прізвище, науковий ступінь, вчене звання, посада)
затверджені наказом від "6" травня 2022 року № 180
2. Строк подання студентом проєкту (роботи) 18.06.2022 року
3. Вихідні дані до проєкту (роботи) платформа для організації групових відеодзвінків, результати науково-дослідної роботи, матеріали переддипломної практики.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) виконати аналіз існуючих способів організації онлайн дзвінків, визначити протоколи та оптимальну топологію для передачі даних у реальному часі. Розглянути готові рішення для організації дзвінків, стандарт WebRTC. Спроектувати архітектуру серверної частини та веб – додатку. Розглянути способи тестування та доставки програми до робочого сервера.

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Підпис, дата	Підпис, дата

7. Дата видачі завдання 6 квітня 2020

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
	Об'єктний аналіз предметної області і постановка завдання	10.04.2022	
	Огляд та аналіз теоретичних відомостей про організацію дзвінків, протоколів для організації дзвінків та топологій обміну даними.	14.04.2022	
	Проектування бази даних	20.04.2022	
	Проектування архітектури програмної системи	24.04.2022	
	Розробка системи тестування	01.05.2022	
	Розробка модулів програмної системи	10.05.2022	
	Тестування програмної системи та аналіз отриманих результатів	28.05.2022	
	Оформлення пояснювальної записки	01.06.2022	
	Підготовка слайдів презентації	05.06.2022	
.	Підготовка демонстраційної версії розробленого програмного продукту	10.06.2022	
11.	Написання доповіді та підготовка до захисту	20.06.2022	

Студент _____ Наталія КОВРИЖКО
(підпис) (ім'я та прізвище)

Керівник роботи _____ Ірина НАЗАРОВА
(підпис) (ім'я та прізвище)

АНОТАЦІЯ

Наталія КОВРИЖКО. Платформа для організації групових відеодзвінків. Випускна кваліфікаційна робота на здобуття освітньо-кваліфікаційного рівня «бакалавр» за спеціальністю 121 Інженерія програмного забезпечення. – ДВНЗ ДонНТУ, Покровськ, 2022.

Об’єктом дослідження є сучасні інструменти розробки веб платформ та способи передавання медіаданих між користувачами.

Мета роботи полягає у дослідженні технологій для передавання медіаданих між групою користувачів та подальшої розробки веб платформи для спілкування з використанням технології WebRTC з топологією SFU.

Платформа включає в себе серверну частину для проведення дзвінків та веб додаток який має можливість відео та аудіо спілкування, створення власної кімнати для дзвінку, систему реєстрації та авторизації через почтові мережі, функціонал для додавання друзів.

Роботи містить 121 сторінку, 45 рисунків, 1 таблицю, 3 додатки, 28 посилань.

Ключові слова: онлайн дзвінки, обмін даними у реальному часі, React, WebRTC, Node.js.

ЗМІСТ

ВСТУП

На сьогоднішній день є актуальними групові відеодзвінки, як в повсякденному житті, так для праці і навчання. Відеоконференції підвищують продуктивність, економлять час, скорочують витрати та найголовніше не вимагають постійних подорожей для особистого спілкування.

Тому для виконання роботи було обрано тему створення платформи для організації групових відеодзвінків, яка надає можливість спілкуватися з людьми різною мовою та на різні теми, тему та мову можна обрати переглядаючи список кімнат на головній сторінці платформи. Також кожен клієнт матиме можливість створення власної кімнати з вказанням теми, мови та заголовку.

Об’єктом дослідження стали сучасні інструменти розробки вебплатформ та способи передавання медіаданих між користувачами.

Метою роботи являються створення платформи для організації групових відеодзвінків під десктопні та мобільні браузери. Поставлені завдання:

а) ознайомитися з існуючими рішеннями для організації дзвінків;

б) розглянути існуючі топології та протоколи для передачі даних;

в) розглянути методи стиснення медіаданих;

г) визначити всі необхідні компоненти для розробки платформи;

д) виконати проектування, реалізацію та тестування платформи.

1 АНАЛІЗ АРХІТЕКТУРНИХ РІШЕНЬ ПРОГРАМНОЇ СИСТЕМИ

1.1 Протоколи передачі потокових даних

Для передачі онлайн – відео використовують протоколи потокової передачі даних та протоколи на основі HTTP. Загалом протокол зв'язку це правила, які керують передачею даних та визначають такі елементи як – синтаксис заголовку файлових даних, аутентифікацію, обробку помилок. HTTP це протокол передачі гіпертексту, він керує обміном даними між веб-серверами та браузерами і є протоколом, який використовується для поширення всього вмісту веб-сайтів серед віддалених користувачів. При передачі потокових даних, протоколи на основі HTTP покладаються на звичайні веб – сервери для оптимізації перегляду та швидкого масштабування, тоді як протоколи потокової передачі даних, передають відео за допомогою виділених серверів потокової передачі.

Протоколи користувацьких датаграм – інформація, яка передана протоколом через мережу без попередньо встановленого зв’язку, та створення віртуального каналу [1] UDP та протокол керування передачею TCP, є основними компонентами інтернет – протоколів. Основною різницею між UDP та TCP є те, що TCP використовує процесор який називають «триетапним рукостисканням» при передачі даних. Тобто клієнт який є ініціатором, відправляє запит до сервера з проханням встановити з’єднання, сервер надає відповідь, а ініціатор підтверджує відповідь та підтримує зв’язок між двома сторонами. З цієї причини TCP досить надійний і може вирішити проблему втрати та впорядкування пакетів. З іншого боку, UDP запускається без «рукостискання». Він передає дані незалежно від будь – яких обмежень пропускної здатності, що робить його швидшим та ризикованим. Оскільки UDP не підтримує повторну передачу, впорядкування пакетів або перевірку

помилок, збій мережі може призвести до пошкодження даних. На рисунку 1.1 наведено роботу протоколів UDP та TCP.

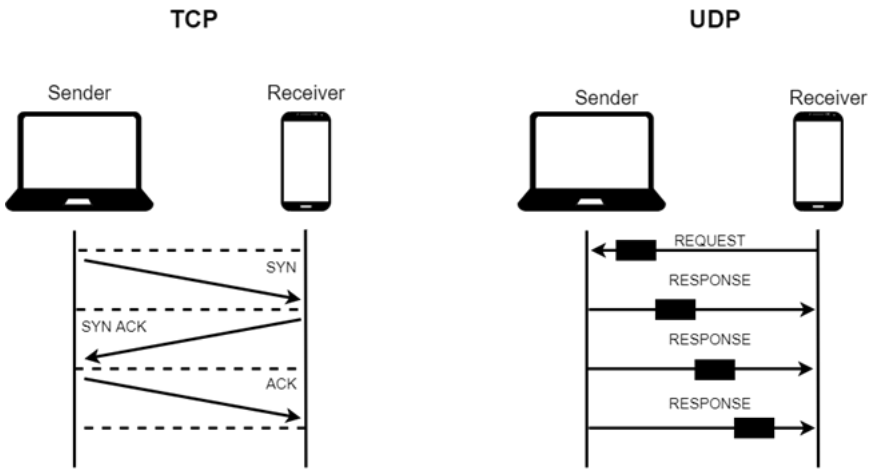


Рисунок 1.1 – Протокол UDP та TCP

1.1.1 Традиційні протоколи потокової передачі медіаданих

Традиційні протоколи потокової передачі, такі як RTSP та RTMP, підтримують потокову передачу з малою затримкою. Але вони з початку не підтримуються в браузерах, мобільних телефонах, комп'ютерах та телевізорах. Сьогодні ці формати потокової передачі краще за все підходять для передачі відео між IP – камерою або кодером та виділеним медіасервіром.

Adobe розробив специфікацію RTMP коли почалась популяризація стримінгу. Протокол міг передавати аудіо та відео дані між виділеним сервером потокової передачі та Adobe Flash Player. Але після закриття Flash, не можна сказати, що RTMP не користується популярністю, його досі використовують багато виробників контенту.

RTSP і RTP часто використовуються як взаємозамінні. Але для ясності: RTSP – це протокол рівня подання, який дозволяє кінцевим користувачам керувати медіасерверами за допомогою можливостей паузи та відтворення, тоді як RTP – це транспортний протокол, який використовується для переміщення зазначених даних.

Пристрої Android та iOS не мають вбудованих програвачів, сумісних з RTSP, що робить цей протокол ще одним, що рідко використовується для відтворення.

1.1.2 Адаптивні протоколи потокової передачі на основі HTTP

Використовуючи потокову передачу з адаптивним бітрейтом, протоколи на основі HTTP забезпечують найкращу якість відео та зручність перегляду, незалежно від підключення програмного забезпечення або пристрою.

Протокол Apple HLS який засновано на протоколі HTTP є провідним у світі. Він підтримує потокову передачу з адаптивним бітрейтом та більш важливе це те, що потік, який буде передаватися через HLS, буде відтворюватися на більшості пристроїв, що забезпечує доступність широкої аудиторії.

HLS з малою затримкою (LL – HLS) – це нова технологія, яка гарантує передачу потоку за менше ніж три секунди по всьому світу. Іншими словами, він призначений для забезпечення тієї ж простоти, масштабності та якості як і HLS, при значному скорочені затримки.

Ще одним протоколом, який засновано на HTTP є MPEG – DASH та CMAF з малою затримкою для DASH, який є альтернативою HLS. Хоча технологія CMAF з малою затримкою для DASH знаходиться в початковому стані, вона обіцяє надавати найшвидке відео в масштабі за рахунок використання більш коротких сегментів даних.

1.1.3 Нові технології протоколів потокової передачі

Новими технологіями є такі протоколи як WebRTC та SRT.

SRT це протокол з відкритим вихідним кодом, який признано технологією, що доставляє потоки незалежно від якості мережі.

WebRTC є самою швидкою та доступною технологією яка забезпечує майже миттєву передачу потокових даних. Вона також може

використовуватися наскрізним чином, через що конкурує з протоколами приймання та доставки. Фреймворк був розроблений виключно для програм на основі чату, але тепер він знаходить застосування і в різноманітних варіантах використання.

1.2 Методи стиснення медіаданих при передачі по мережі

Розмір файлу зазвичай є основною проблемою для тих, хто відображає або дивиться прямі трансляції або просто завантажує файл, оскільки великі файли затримують усі ці процеси. Це пов’язано з тим, що вони потребують багато місця для збереження та споживають більше пропускної здатності, яка може бути обмеженою чи дорожчою на платформі відеохостингу. Для вирішення цих питань використовують різні формати стиснення медіаданих.

Стиснення медіаданих – це процес зменшення бітів, необхідних для представлення даних, без втрат для його візуальної якості. Метою стиснення є зменшення розмірності та полегшення передачі по мережі [2]. Стиснення відео оптимізує потребу в ємності сховища, а також швидкість передачі файлів. Це спрощує завантаження та спільне використання відео, дозволяючи глядачам насолоджуватися плавною потоковою передачею [2].

Стискання відео відбувається за допомогою кодеків та відеокодеків. Кодеки відфільтровують допоміжні дані, такі як кадри схожих відео, згруповуючи їх за категоріями, у результаті чого, вони видаляють деякі кадри даних та кодують дані які залишилися, що зводиться до втрати бітів та зменшенню розміру файлу [3]. Відеокодеками є стандарти стиснення відео, які реалізуються за допомогою програмних або апаратних додатків [3].

Існує два типи стиснення відео: стиснення без втрат та стиснення з втратами.

Стиснення з втратами це найчастіший тип стиснення який використовується для зменшення розмірів даних. Стиснення з втратами відбувається за рахунок видалення частини даних, які вважаються

непотрібними. Ці частини можуть бути будь чим, наприклад звуком який повторюється. Мінусом даного стиснення є те, що він може привести до різкого падіння якості.

Стиснення без втрат найкраще для всього підходить для збереження якості медіаданих під час зменшення їх загального розміру. Мінус в тому, що на виході стиснутий файл має той же розмір, що й початковий.

1.3 Топології

1.3.1 Принцип роботи топології Mesh

В одноранговій мережі клієнти обмінюються даними без додаткового серверу. В даній мережі кожен клієнт відправляє усім іншим дані, та отримує дані від усіх інших користувачів, через що зростає навантаження на мережу та вимоги до пристрою клієнта.

Топологія Mesh загалом використовується в системах, де кількість клієнтів не велика. Якщо казати про аудіо та відео зв’язок, то в середньому це до чотирьох учасників, число може змінюватися в залежності від характеристик пристроїв користувачів системи, потужності інтернету, алгоритмів стиснення даних та інших факторів [4]. Діаграма топології яка розглядається, показано на рисунку 1.2.

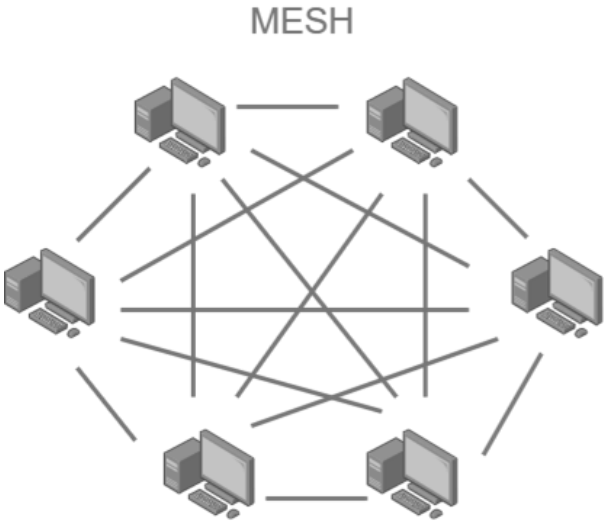


Рисунок 1.2 – Топологія передачі даних Mesh

Завдяки відсутності додаткового сервера між учасниками зменшується затримка при передачі даних, та відсутні затрати на сервер.

1.3.2 Принцип роботи топології SFU

Топологія SFU з’єднує однорангову мережу між сервером та клієнтом, а не між одноранговими мережами клієнтів. При всіх типах з'єднання, клієнту потрібно надсилати на сервер лише свої власні дані без необхідності надсилати їх всім підключеним користувачам. Перевагою цієї топології в залежності від попередньої є те, що навантаження на клієнта знижується, але при великомасштабній структурі, клієнт, як і раніше, обробляє велике навантаження [4]. Діаграма топології яка розглядається, показано на рисунку 1.3.

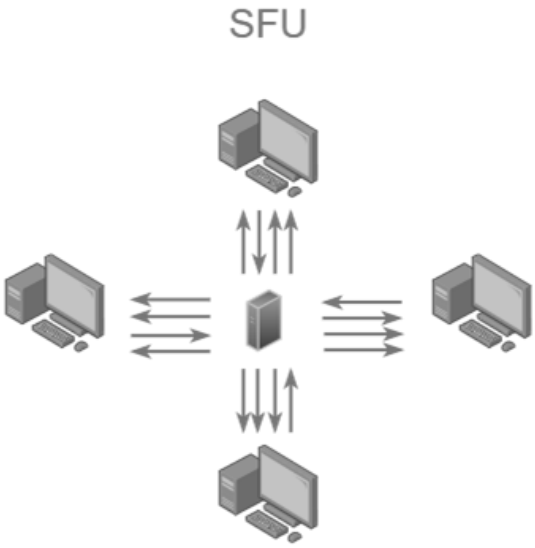


Рисунок 1.3 – Топологія передачі даних SFU

1.3.3 Принципи роботи топології MSU

Топологія MSU це аналог топології SFU, при якому кілька середовищ передачі змішуються або обробляються на центральному сервері і доставляють стороні, що приймає [4]. Наприклад, якщо до мітингу підключаються 5 осіб, відеодані 4 особи, крім однієї з них, редагуються в одні відеодані, а аудіодані також редагуються та надсилаються одній людині. Те ж

саме відноситься і до 4 людей, що залишилися. Топологія з'єднує одноранговий вузол між сервером та клієнтом, а не між одноранговими вузлами клієнта. При всіх типах підключення клієнту необхідно відправити на сервер лише власні відео без необхідності відправки даних всім підключеним користувачам (тобто є 1 висхідний канал) та отримувати дані від сервера тільки одного вузла, незалежно від кількості підключених користувачів. Діаграма топології яка розглядається, показано на рисунку 1.4.

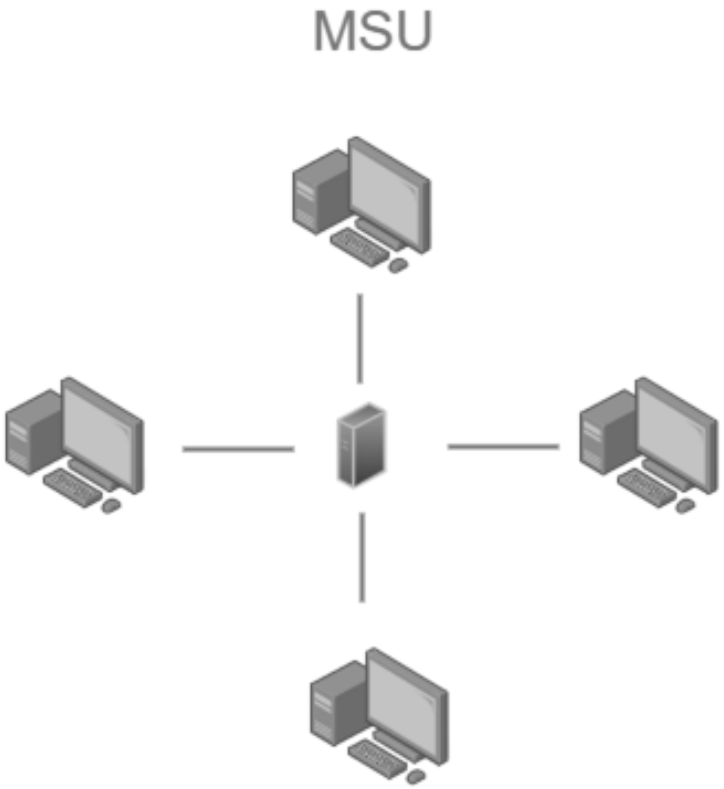


Рисунок 1.4 – Топологія передачі даних MSU

1.4 Огляд готових рішень для організації дзвінків

1.4.1 Jitsi

Jitsi – це програмне забезпечення для відеоконференцій з відкритим вихідним кодом, яке допомагає підприємствам створювати захищені паролем віртуальні кімнати для проведення аудіо та відеодзвінків [5]. Адміністратори можуть створювати налаштовані URL – адреси зборів, відключати звук або

видаляти учасників, надавати спільний доступ до всього екрану або певних програм, а також налаштовувати платформу кількома мовами.

1.4.2 Kurento

Kurento – це медіа – сервер WebRTC і набір клієнтських API, що спрощують розробку передових відеододатків для браузерів і смартфонів. Функції Kurento Media Server включають групове спілкування, перекодування, запис, мікшування, трансляцію та маршрутизацію аудіовізуальних потоків [6].

Як відмінну особливість, Kurento Media Server також надає розширені можливості обробки медіа, включаючи комп'ютерний зір, індексування відео, доповнену реальність та аналіз мовлення. Модульна архітектура Kurento спрощує інтеграцію алгоритмів обробки медіа сторонніх розробників (наприклад, розпізнавання мовлення, аналізу настроїв, розпізнавання облич тощо), які можуть прозоро використовуватися розробниками додатків як інші вбудовані функції Kurento.

1.4.3 Mediasoup

Mediasoup це бібліотека яка розгортає WebRTC для Node.js, який в свою чергу дозволяє додаткам запускати багатосторонні відеоконференції в браузері та мобільному пристрої [6]. Mediasoup є не ще одним автономним медіа-сервером, а бібліотекою/модулем Node.js, який можна легко інтегрувати в існуючі програми. Mediasoup надає API WebRTC, дозволяючи додатку програмувати багатосторонні відеоконференції за допомогою JavaScript. Mediasoup мінімалістичний, він просто фокусується на обробці мультимедіа. Він забезпечує і не наказує конкретний протокол мережевої сигналізації.

1.4.4 Janus

Janus – це сервер WebRTC, розроблений Meetecho, задуманий як сервер загального призначення [6]. Таким чином, він не надає жодної функціональності як такої, крім реалізації засобів для налаштування мультимедійного зв'язку WebRTC з браузером.

1.5 Стандарт WebRTC

WebRTC це технологія, яка дозволяє спілкуватися в режимі реального часу через браузер. Без WebRTC між пристроями повинен бути проміжний сервер, щоб вони могли спілкуватися один з одним. Один пристрій відправляє повідомлення на сервер, а сервер відправляє його іншому пристрою. Це значить, що для роботи зв’язку між пристроями, на них повинен бути встановлений один і той самий модуль або додаток.

Технологія WebRTC побудована на кількох стандартах та протоколах [7]. Вона забезпечує мультимедійний зв'язок у режимі реального часу у браузерах та додатках. WebRTC дозволяє розробникам вбудовувати голосовий та відеозв'язок безпосередньо на веб – сторінки, не вимагаючи від кінцевого користувача встановлення будь – яких плагінів для браузера [7]. Набір WebRTC API також дозволяє розробникам використовувати WebRTC у програмах для зв'язку в реальному часі.

1.5.1 Етапи з'єднання WebRTC :

Пристрій – STUN – сервер – одноранговий канал зв'язку – пристрій одержувача.

Однак на кожному етапі відбуваються процеси. Ось як це працює:

Коли починається аудіо або відеодзвінок WebRTC має встановити з'єднання з усіма іншими пристроями, які будуть підключені до дзвінка.

Перш ніж можна буде встановити з'єднання WebRTC має пройти через брандмауер та NAT. Брандмауери та пристрої NAT працюють, встановлюючи загальнодоступну IP – адресу для комп'ютера, який транслюється у зовнішній світ і маскує приватну IP – адресу.

Комп'ютер знає лише приватну IP – адресу. Таким чином, WebRTC зв'язується з сервером STUN, щоб отримати загальнодоступну IP – адресу. Таким чином WebRTC може направити вхідне з'єднання на правильну IP – адресу. Як тільки загальнодоступна IP – адреса буде отримана з сервера, WebRTC отримає загальнодоступну IP – адресу для інших пристроїв, які

будуть підключатися до дзвінка. Як тільки програма дізнається всі необхідні IP – адреси, вона створює список потенційних конфігурацій підключення, також званих кандидатами ICE, і вибирає найефективнішу конфігурацію. Потім WebRTC використовує цю конфігурацію підключення для відкриття приватного каналу даних, де всі пристрої, що беруть участь у виклику можуть обмінюватися аудіо та відео в режимі реального часу. А оскільки лише пристрої, що беруть участь у дзвінку, знають конфігурацію з'єднання, яке є приватним і не може бути доступним нікому, хто не бере участь у ньому.

Це звичайний метод підключення для зв'язку WebRTC, коли всі пристрої WebRTC передають аудіо і відеопотоки безпосередньо один одному. Однак цей прямий зв'язок іноді не може бути встановлений [7].

У цих випадках WebRTC використовує сервер TURN. Сервер черги легко діє як повторювач. Якщо пряме з'єднання не може бути встановлене між пристроєм під час виклику WebRTC змусить комп'ютери надсилати аудіо та відео на сервер TURN, який передає дані на приймаючий пристрій і навпаки. Однак використання сервера TURN для зв'язку з WebRTC – це крайній захід.

Переваги WebRTC:

а) краща якість звуку;

б) технологія з відкритим вихідним кодом;

в) спрощена розробка;

г) безпека та стабільність;

д) сумісність;

е) не потребує плагінів.

WebRTC пропонує вбудовану підтримку ехоподавлення та шумоподавлення, а також автоматичне регулювання чутливості мікрофона.

Незважаючи на те, що WebRTC заснований на архітектурі C++, WebRTC має вбудований рівень API JavaScript, який розробники можуть використовувати.

WebRTC захищений кількома обов'язковими специфікаціями шифрування. Це забезпечує наскрізне шифрування для будь – яких даних, що надсилаються

Завдяки відкритому вихідному коду WebRTC підтримується всіма основними настільними та мобільними операційними системами.

Для більшості технологій зв'язку в реальному часі потрібний модуль для здійснення дзвінків за допомогою браузера. Ці плагіни повинні бути встановлені кінцевим користувачем, що погіршує його роботу. Більшість браузерів підтримують WebRTC без будь – яких плагінів.

Недоліки WebRTC:

а) WebRTC – технологія, що розвивається;

б) конференц – зв'язок потребує великих ресурсів.

WebRTC це нова технологія, і поточна версія це робочий проект. Це означає, що в майбутньому вихідний код WebRTC може зазнати значних змін.

Веб – браузери не можуть синхронізувати кілька вхідних аудіо та відеопотоків. Це означає, що конференц – дзвінки вимагають, щоб сервер конференц – зв'язку мікшував аудіо та відеопотоки перед їх розподілом у вигляді єдиного потоку даних.

1.5.2 Аналіз безпечної передачі даних WebRTC

Є декілька причин, через які додатки для зв’язку в реальному часі можуть становити загрозу безпеки як для оператора зв’язку, так і для користувачів. Такі ризики можуть бути застосовані до будь – якої програми, яка має справу з передачею даних та мультимедіа в реальному часі. Безпека иа шифрування є не додатковими функціями WebRTC, а забезпечуються влаштованими компонентами за замовчуванням. Крім того, WebRTC пропонує шифрування між вузлами на будь – якому сервері, забезпечуючи безпечний зв'язок у режимі реального часу [8].

WebRTC вимагає від користувача явним чином дозволити доступ до камери та мікрофону. Таким чином, користувач знає про включення камери на його комп'ютері. Коли користувач дозволяє доступ, на вкладці браузера відображається червона точка, що вказує на доступ до медіапристроїв [9].

Для передачі в режимі реального часу WebRTC використовує протокол датаграм безпеки транспортного рівня – DTLS. Цей протокол за замовчуванням вбудований у всі браузери, що підтримують технологію WebRTC. У з'єднанні, зашифрованому за допомогою DTLS, виключається підслуховування та підробка інформації [10].

1.6 Висновки за розділом

В даному розділі було розглянуто протоколи передачі даних, методи стиснення медіаданих при передачі по мережі, розглянуто готові рішення для організації дзвінків такі як: Jitsi, Kurento, Mediasoup та Janus. Було розглянуто стандарт WebRTC, його етапи з’єднання та проаналізовано безпечність передачі даних за допомогою WebRTC.

2 ПРОЕКТУВАННЯ АРХІТЕКТРИ ПРОГРАМНОЇ СИСТЕМИ

2.1 Визначення функціональних, не функціональних та бізнес вимог

Для створюваної платформи було визначено наступні функціональні, не функціональні та бізнес вимоги:

а) підтримка організації дзвінків до 50 учасників;

б) можливість запуску в сучасних десктопних та мобільних браузерах;

в) підтримка мобільної та десктопної веб версії;

г) незалежний додаток для створення дзвінків та користування сайтом;

д) система автоматичної авторизації та реєстрації через сервіс пошти;

е) можливість запуску програми в віртуальному окрузі;

ж) збереження статичних даних в окремих серверах;

з) можливість трансцяції відео через веб камеру;

и) можливість включення та виключення мікрофону;

к) функція відправки запрошення на вступ у дзвінок іншому учаснику;

л) використання автоматичного тестування;

м) використання системи контролю версій ;

н) зберігання інформації у власній базі даних;

о) система логування роботи додатку.

2.2 Вибір топології для передачі даних

У першому розділі роботи було досліджено та описано особливості трьох існуючих топологій які можуть бути використані для обміну інформацією між користувачами в реальному часі.

Розглянута топологія Mesh яка може забезпечити найбільш меншу затримку при передачі даних між користувачами не підходить для обраної області задач через свою архітектуру. В цій топології кожен клієнт повинен відправляти та отримувати дані від усіх інших клієнтів, через що створюється велике навантаження на пристрій клієнта та не підходить для організації дзвінків з більшою кількістю учасників [11]. Завдяки використанню топології MCU навантаження на пристрій і передачу по мережі клієнта стає мінімальною. Дана топологія усі відео та аудіо потоки від різних користувачів збирає на сервері в один, і тільки цей об'єднаними потік передає на сервер. Через що навантаження топології лягає на сервер.

Тому для використання такої топології потрібні потужні обчислювальні сервери, а при використанні малої кількості серверів які не підходять за потужністю може збільшуватися затримка між клієнтами.

Виходячи з особливостей цих топологій, топологія SFU є найбільш оптимальним варіантом для організації дзвінків з великою кількістю учасників та менш обчислювальним навантаженням на сервер.

Так як топологія SFU всі дані кодуються на клієнті, відправляються на сервер, а сервер в свою чергу перенаправляє їх іншим учасникам дзвінка, які їх назад декодують на клієнта. Тому навантаження розподіляється між сервером та клієнтом.

В таблиці 2.1 наведено порівняння навантаження на пристрій користувача та сервера при використанні різних топологій для обміну даними між користувачами.

Таблиця 2.1 – Порівняння навантаження на пристрій користувача та сервера при використанні різних топологій для обміну даними [11]

	Mesh	MCU	SFU
вхідні/вихідні дані	3/3	1/1	3/1

	Mesh	MCU	SFU
вхідний трафік	3 Мбіт/сек	1 Мбіт/сек	3 Мбіт/сек
вихідний трафік	3 Мбіт/сек	1 Мбіт/сек	1 Мбіт/сек
процесор клієнта	3*енкодер + 3*декодер = 60%	1* енкодер + 1* декодер = 20%	1* енкодер + 3* декодер = 40%
процесор сервера	0	100%	10%
затримка	min	max	avg
максимальна кількість учасників	~8	∞	~50

2.3 Архітектура програмного засобу

Архітектуру програмного засобу зображено на рисунку 2.1 за допомогою діаграми розміщення. Діаграма розміщення складається з 5 обчислювальних вузлів. На ПК клієнта розміщено браузер в якому розташовується розробляєма платформа. Сервер на якому зберігається API сервер, WebRTC сервер та база даних. Всі сервери між собою та база даних зв’язується за допомогою локальної мережі, а комп’ютер клієнта з серверами за допомогою Інтернету.

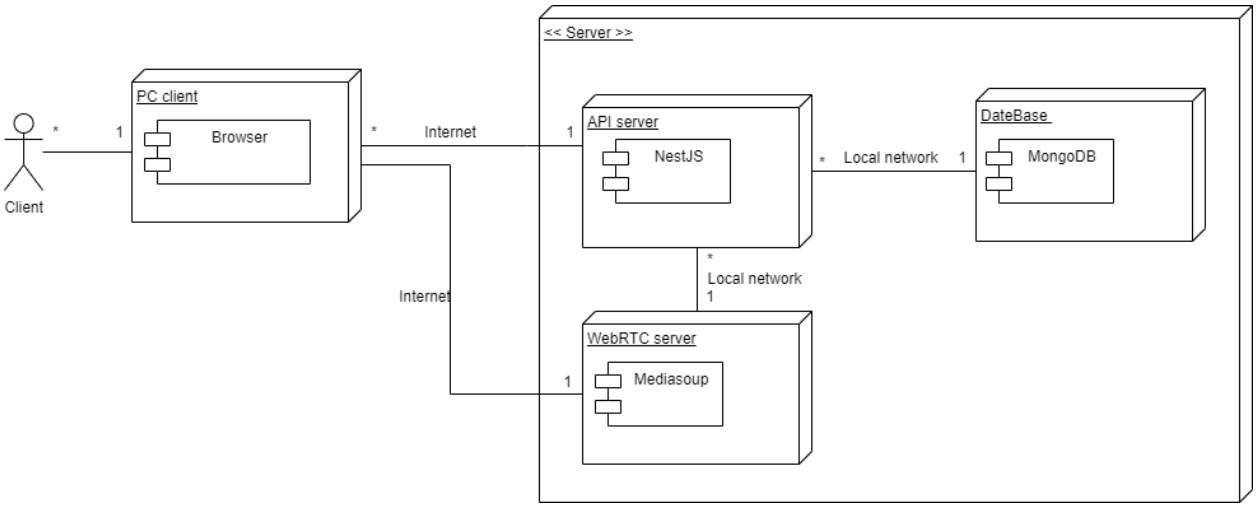


Рисунок 2.1 – Архітектура програмного засобу

Процес взаємодії користувача з сайтом проходить через центральний додаток API Server, який обробляє запити користувача на такі дії як авторизація, створення кімнати, відображення списку друзів, кімнат, ця програма може безпосередньо взаємодіяти з базою даних, і зберігає всю бізнес логіку програми. Процес спілкування відбувається з використанням додаткового додатку WebRTC сервер, який забезпечує обмін даними між користувачами в реальному часі.

2.4 Можливість автоматичного тестування модулів програми

Під час розробки системи основні модулі програми повинні бути покриті автоматичними тестами для більш надійної та стабільної роботи програми. Етап тестування повинен запускатися при релізі додатку, а у разі успішного проходження нова версія програми буде доступна користувачам.

Деякі переваги автоматичного тестування:

а) швидший цикл зворотного зв’язку;

Без автоматизації тестування зворотний зв'язок про недавно розроблені функції може зайняти деякий час. Автоматизація тестування є особливо корисною, оскільки допомагає виявляти проблеми або помилки на ранній стадії розробки

б) високе тестове покриття;

Ручне тестування накладає обмеження кількості тестів, які можна перевірити. Автоматизація дозволяє витрачати час на написання нових тестів та додавання їх до набору автоматизованих тестів. Це збільшує тестове охоплення розробленого продукту, тому належним чином тестується більше функцій, що призводить до більш високої якості програми.

в) повторне використання набору тестів;

Спочатку створення набору автоматизованих тестів – складне завдання. Однак після того, як набір тестів було визначено, дуже просто повторно використовувати їх для інших варіантів використання або навіть для інших проектів.

Існує багато типів автоматичного тестування, для розробляємої платформи було обрано модульне, через те, що модульний тест не повинен стосуватися іншого коду або взаємодіяти з базою даних, конфігураційним файлом або мережею. Кінцева мета – перевірити правильність логіки [12].

Модульне тестування визначається як тип тестування програмного забезпечення, у якому перевіряються окремі підпрограми, підпрограми, класи чи процедури у програмі[12]. Замість того, щоб тестувати всю програму відразу, модульне тестування рекомендує тестувати дрібніші будівельні блоки програми [12].

На рисунку 2.2 зображено етапи автоматичного тестування за допомогою діаграми послідовності.

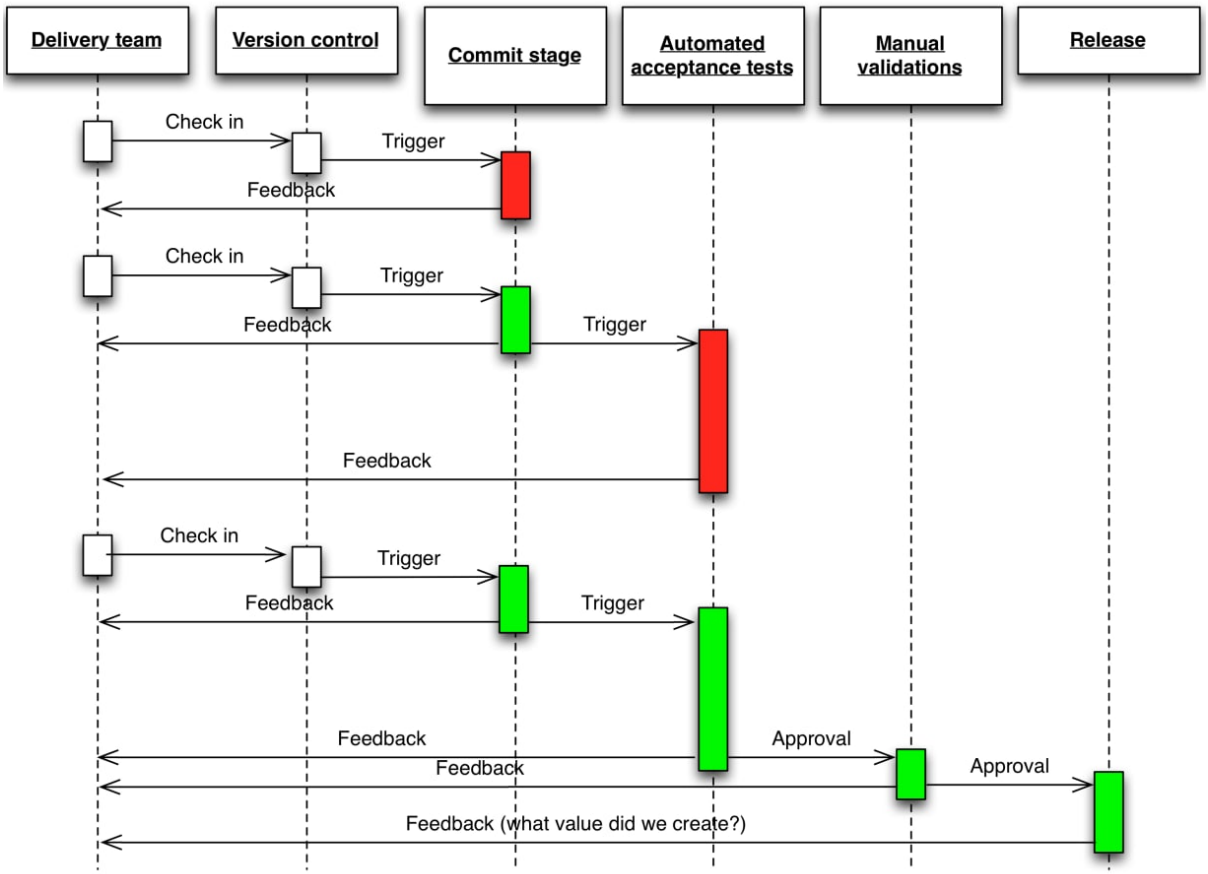


Рисунок 2.2 – Етапи автоматичного тестування [4]

2.5 Визначення апаратних та програмних вимог користувача

Визначення апаратних та програмних вимог користувача:

а) Windows 7 і пізніші операційні системи;

б) 32 – бітній (x86) або 64 – розрядний (x64) процесор;

в) мінімум 2 ГБ оперативної пам'яті;

г) Intel Pentium 4 / Athlon 64 або пізнішої версії з підтримкою SSE2.

д) підтримка наступних десктопних браузерів[3]:

- Microsoft Edge 12+;
- Google Chrome 28+;
- Mozilla Firefox 22+;
- Safari 11+;
- Opera 18+;
- Vivaldi 1.9+;

– Brave;

е) підтримка наступних мобільних браузерів:

– Google Chrome 28+ ;
– Mozilla Firefox 24+;
– Opera Mobile 12+;
– Chrome OS;
– Firefox OS;
– BlackBerry 10;
– MobileSafari/WebKit (iOS 11+).

2.6 Побудова діаграми варіантів використання

Діаграма варіантів використання в UML – це діаграма яка описує функціональні та поведінкові можливості користувача і системи.

На даній діаграмі яку зображено на рисунку 2.3 наведено приклад UML діаграми варіантів використання яка представляє можливості користувача в системі.

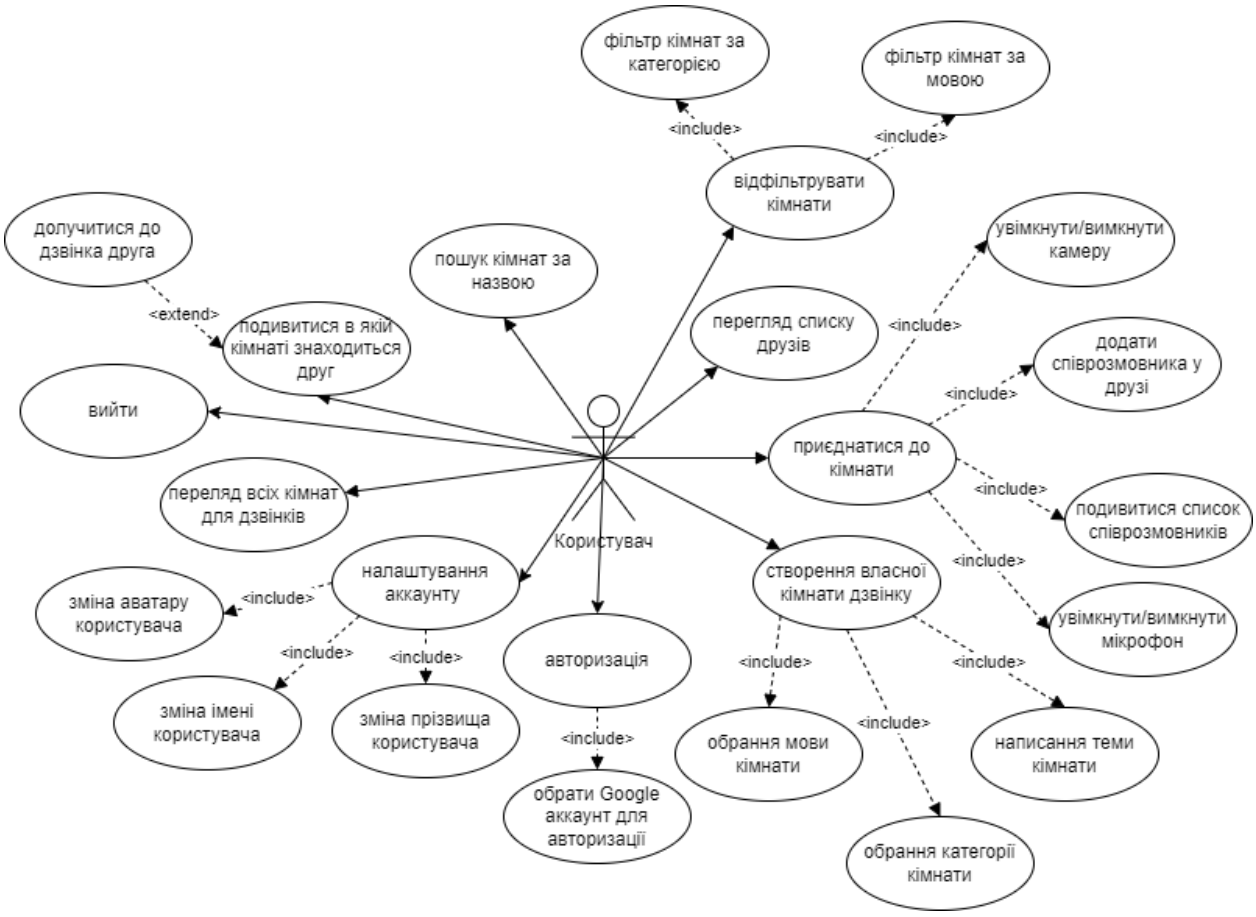


Рисунок 2.3 – Діаграма варіантів використання яка представляє можливості користувача в системі.

Як видно за діаграмою користувачу доступні такі можливості як:

а) авторизація;

б) налаштування аккаунту яке включає в себе:
- зміну аватару користувача;
- зміну прізвища користувача;
- зміну ім’я користувача;

в) перегляд всіх кімнат для дзвінку;

г) пошук кімнат за назвою;

д) фільтрація кімнат що включає:
- фільтр кімнат за мовою;
- фільтр кімнат за категорією.

е) переглянути список друзів;

ж) подивитися в якій кімнаті знаходиться друг, що розширюється приєднанням до кімнати друга;

з) приєднання до кімнати в якій можна:

- увімкнути/вимкнути камеру;
- увімкнути/вимкнути мікрофон;
- подивитися список співрозмовників;
- додати співрозмовника у друзі.

и) створити власну кімнату дзвінку яка включає:

- написання теми кімнати;
- вибір категорії кімнати;
- вибір мови кімнати.

к) вийти з сайту.

2.7 Побудова діаграми діяльності

Діаграма діяльності це одна з діаграма в UML, що описує динамічні аспекти які представляють потік від однієї дії до іншої.

На даній діаграмі діяльності яку зображено на рисунку 2.4 відображено послідовність дії, яку виконує сервер та фронт під час спроби входу користувача до сторінок сайту крім сторінки логіну. Під час спроби користувача увійти до будь – якої сторінки сайту, фронт перевіряє наявність токену входу у браузері користувача, якщо токен є, він відправляється на сервер для спроби прочитати з нього інформацію. Якщо інформацію вдалося зчитати, то перевіряється термін дії токену, при дійсності токена, відбувається пошук юзера за зчитаю інформацією, якщо користувача було знайдено, він отримує доступ до запитаної сторінки. В усіх інших випадках: токену немає, інформація з токену не прочитана, токен не дійсний, юзера не знайдено, користувача буде перенаправлено на сторінку входу.

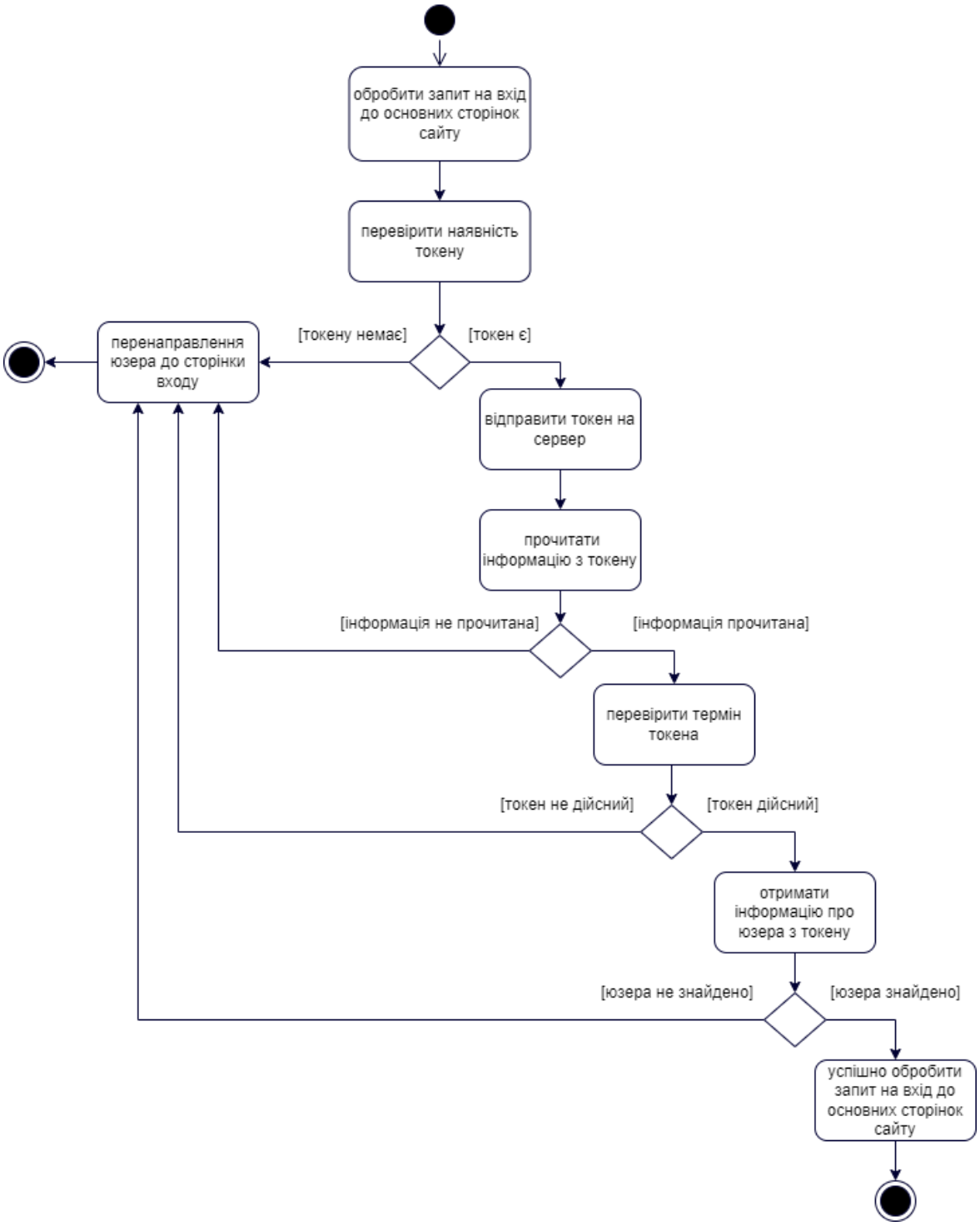


Рисунок 2.4 – Діаграма діяльності під час входу до системи

2.8 Висновки за розділом

В даному розділі було визначено функціональні, не функціональні та бізнес вимоги до програмного продукту. Обрано топологію для передачі даних між клієнтами. Описано та зображено архітектуру програми за допомогою

діаграми розміщення. Обрано та описано метод тестування розробляємої програми, визначено апаратні та програмні вимоги до користувача для використання сервісу. Спроектовано діаграму варіантів використання та діяльності.

3 РОЗРОБКА КЛІЄНТСЬКОЇ ТА СЕРВЕРНОЇ ЧАСТИНИ

3.1 Структура проекту

Платформа складається з трьох основних модулів:

а) клієнтський додаток;

б) серверний додаток;

в) серверний додаток для обробки дзвінків.

Кожен модуль ізольований від іншого, а загальні функції та дані розміщені в окремих бібліотеках, щоб уникнути дублювання в коді. Завдяки відділенню програми для організації дзвінків було отримано можливість вносити зміни в основний серверний додаток і перезапускати його в подальшому без перебоїв сесій спілкування користувачів у платформі.

3.1.1 Розробка клієнтського додатку

Для розробки клієнтської програми було обрано бібліотеку React. React має можливості побудови гнучкої архітектури та швидкий рендеринг модулів, крім цього він має безліч бібліотек та велике комьюніті, які збільшують швидкість розробки продукту. Для реалізації механізму спілкування між користувачами було застосовано бібліотеку mediasoup – client. Для обміну подіями у програмі використовується бібліотека Redux [13].

Структура проекту клієнтського додатка показано рисунку 3.1, та складається з таких основних директорій :

а) public – директорія для зберігання статичних файлів;

б) src – основний вихідний код програми;

в) components – всі компоненти інтерфейсу;

г) constants – незмінні опції програми;

д) core – функції та модулі сайту не прив'язані до React;

е) routes – адреси сторінок доступні для навігації по сайту.

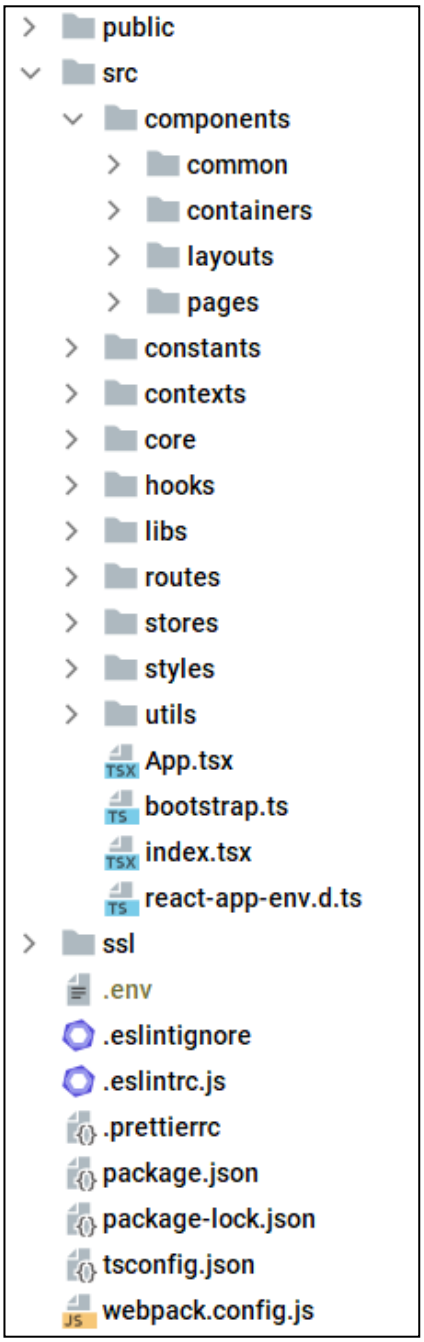


Рисунок 3.1 – Структура проекту клієнтського додатка

Для збірки файлів вихідного коду, статичних файлів, ресурсів клієнтської програми використовується бібліотека Webpack. Webpack має можливості переводу файлів вихідного коду у більш старі версії JS для підтримки старих браузерів. Також Webpack стискає код, який не буде використано на сайті, завдяки чому забезпечує приріст швидкості

завантаження сайту [14]. На рисунку 3.2 проілюстровано схему роботи бібліотеки Webpack.

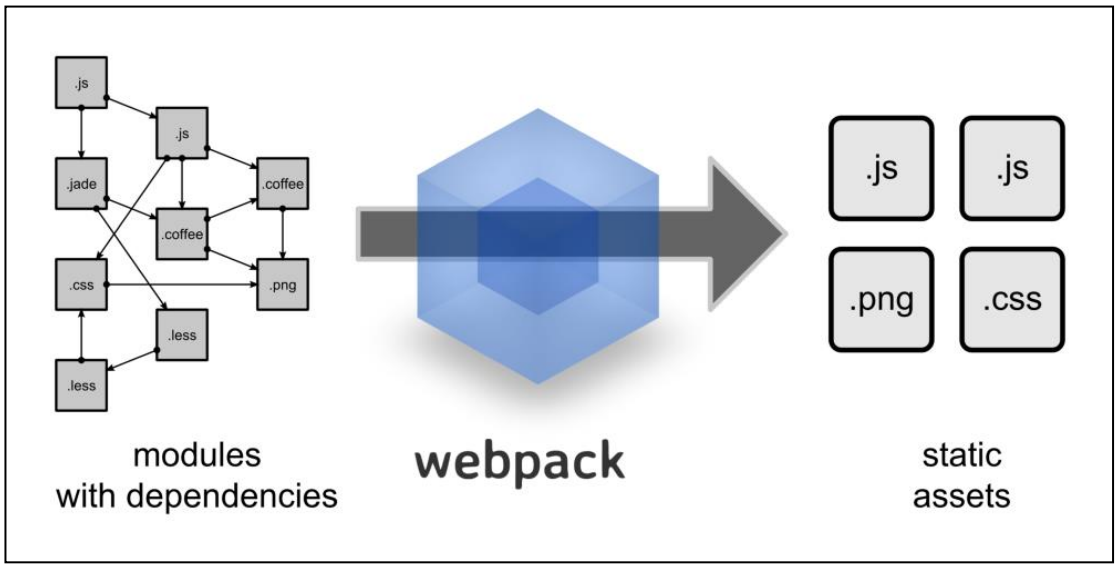


Рисунок 3.2 – Схема роботи бібліотеки Webpack[25]

Приклад зібраного та стисненого клієнтського додатку відображено на рисунку 3.3.

Рисунок 3.3 – Отримані файли після зборки клієнтської програми.

Візуалізацію сховища програми, що зберігається в бібліотеці Redux і згенеровано за допомогою Redux DevTools, відображено на рисунку 3.4.

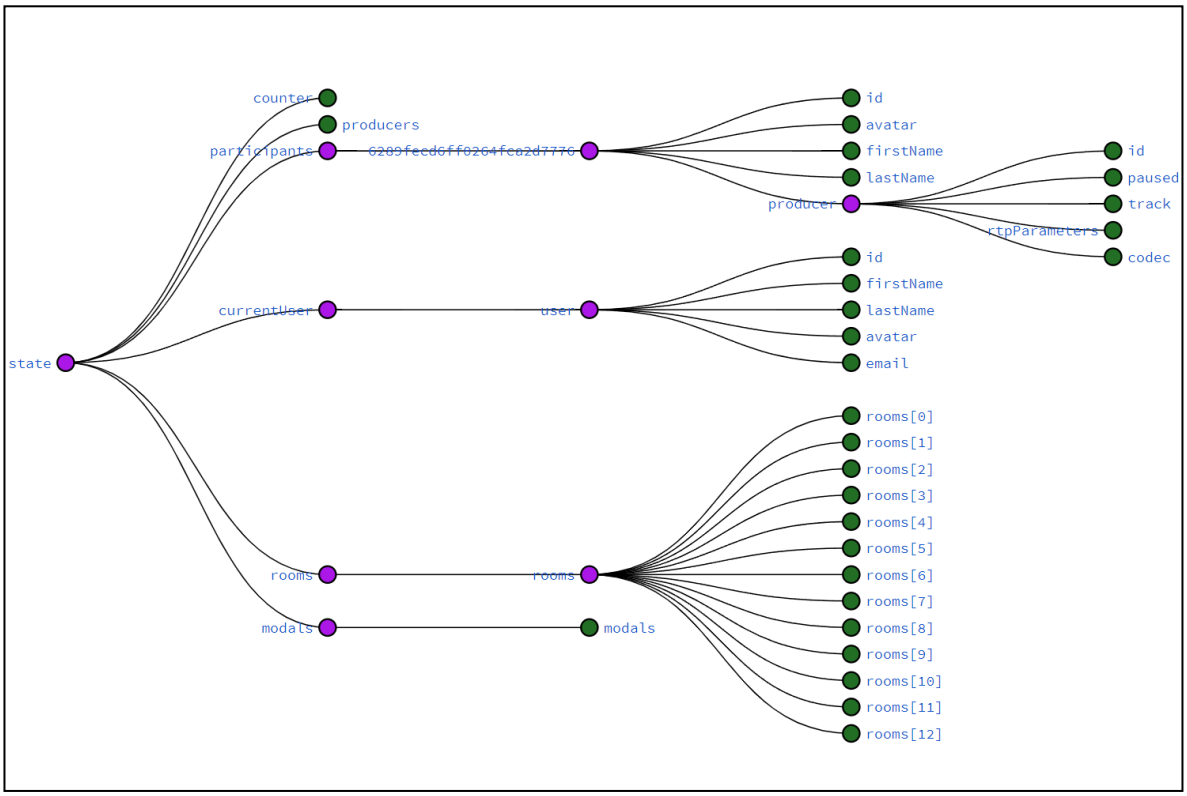


Рисунок 3.4 – Сховище програми

3.1.2 Розробка серверного додатку

Серверний додаток обробляє запити користувача та не містить механізми спілкування та проведення дзвінків. Механізми спілкування та проведення дзвінків було винесено в окремий додаток для того, щоб не переривати сесії користувачів після внесення зміни до коду програми.

Для реалізації програми був обраний фреймворк NestJS з використанням NodeJS, який використовує мову програмування TypeScript [15]. Такий вибір визначений можливістю побудови модульної архітектури та використанню єдиної мови програмування для проекту, яка прискорює розробку так як не доводиться перемикатися між різними мовами і дозволяє виносити спільні функції та дані, що застосовуються у різних додатках.

Приклад отриманої структури проекту для серверного додатка з використанням фреймворку NestJS відображено на рисунку 3.5.

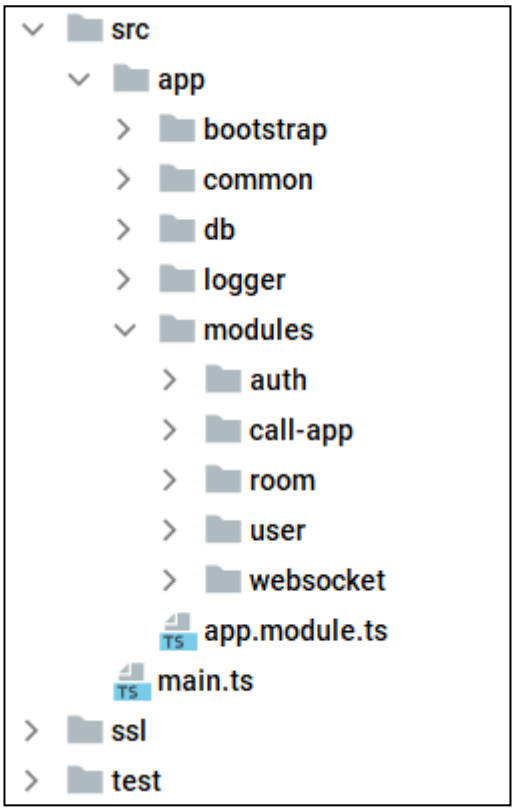


Рисунок 3.5 – Отримана структура проекту для серверного додатку

Оперуючи бізнес процесами що виникають у роботі програми, такі як створення кімнат, отримання кімнат, можна припустити що операції читання і запису в базу даних виникає частіше, і слідуючи наведеним тестам на рисунку 3.6 та рисунку 3.7, MongoDB має кращі показники запису і читання в базу даних по порівняно з реляційною базою даних MySQL.

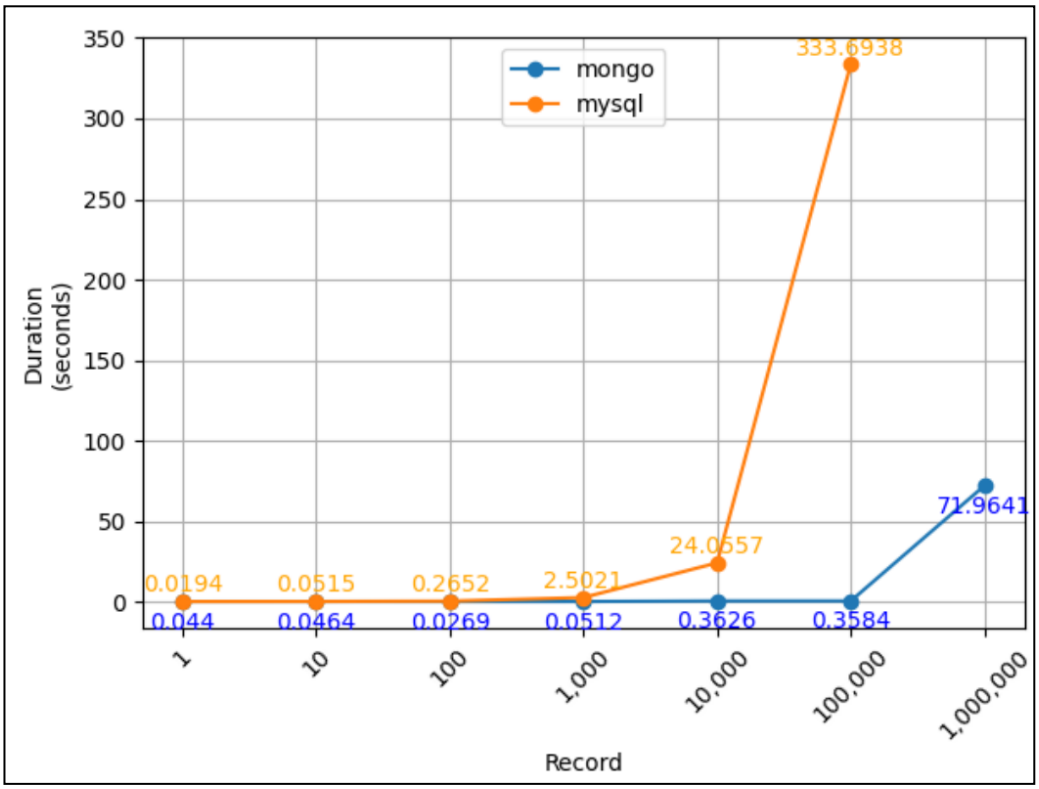


Рисунок 3.6 – Вставка n записів в одну таблицю [28]

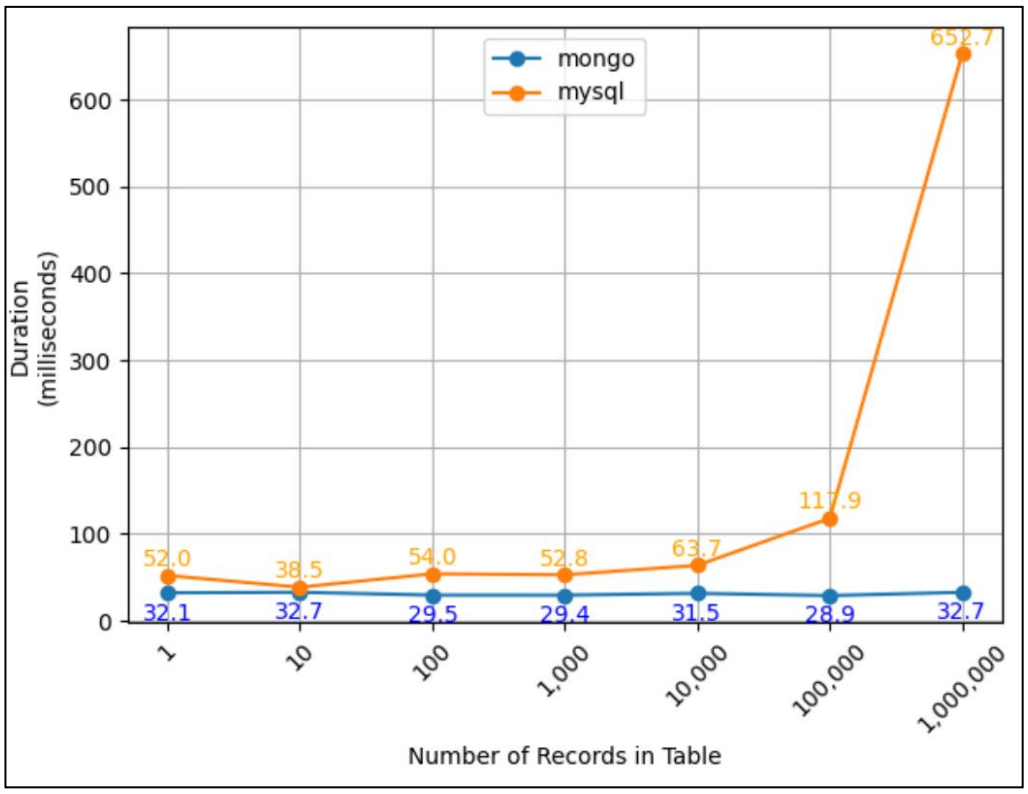


Рисунок 3.7 – Порівняння продуктивності читання записів за одним з їх неіндексованих полів [28]

За результатами тесту було обрано документоорієнтовану базу даних MongoDB. Так як у додатку не використовується безліч зв'язків між сутностями в базі, то застосування документорієнтованої бази даних замість реляційної зможе показати кращу швидкість читання і запису. Крім цього MongoDB дозволяє зберігати неструктуровані дані, що є перевагою при ранньому етапі розробки продукту коли структура даних може часто змінюватися і не доводиться змінювати схему таблиць як у випадку з реляційними базами даних, такими як MySQL або PostgreSQL.

Схема даних, що використовується для додатку представлена на рисунку 3.8.

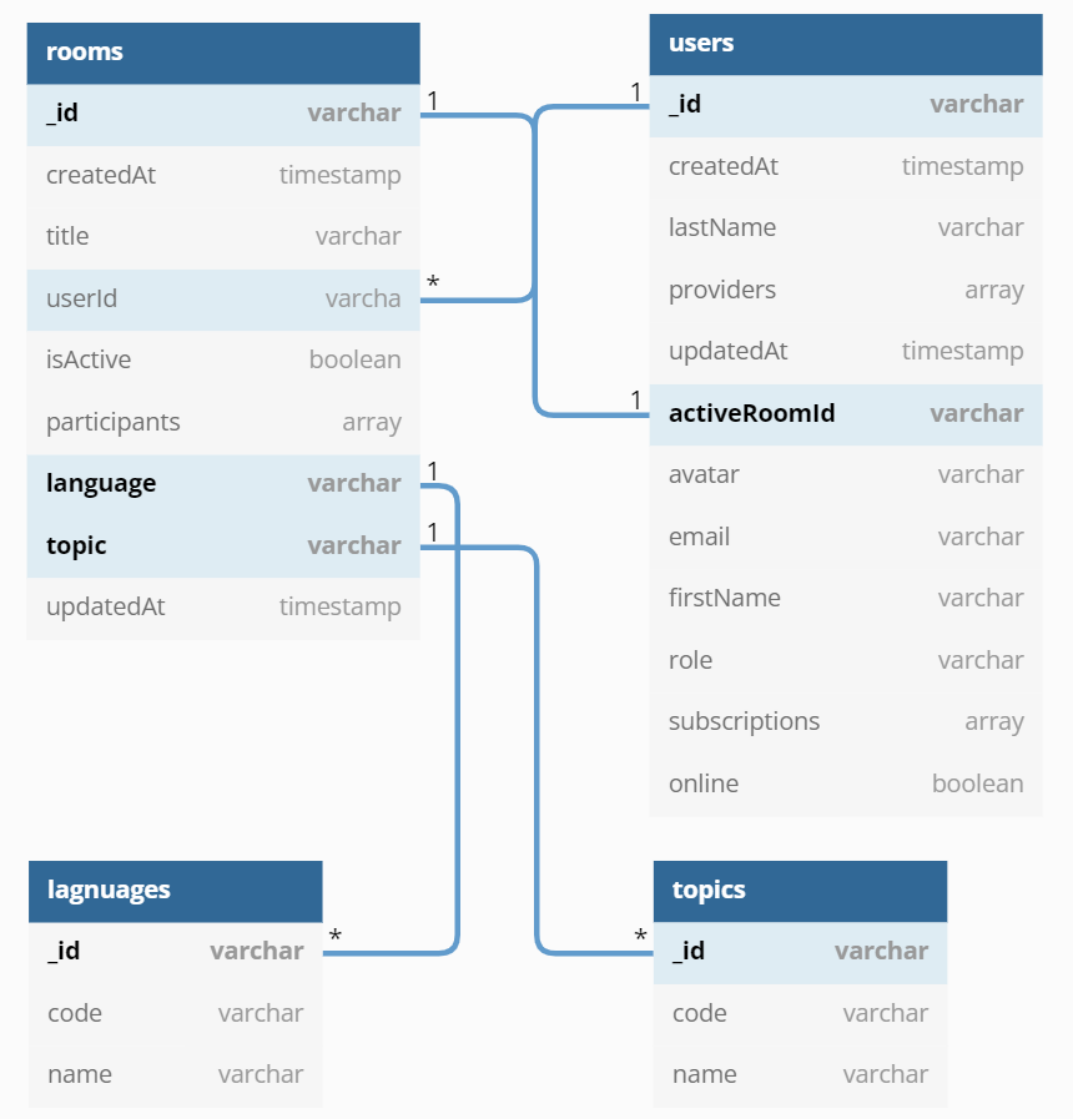


Рисунок 3.8 – Схема даних БД додатку

Для більш зручної та швидкої установки та конфігурації MongoDB на локальному комп'ютері при розробці, та в основному додатку на сервері було прийнято рішення помістити MongoDB додаток у Docker контейнер, тим самим ізолювавши його від зовнішнього оточення системи. Отриманий файл конфігурації наведено рисунку 3.9.

```
services:
  mongo:
    image: mongo
    container_name: talkspace_mongo
    restart: always
    ports:
      - target: 27017
        published: 27018
    environment:
      MONGO_INITDB_ROOT_USERNAME: dev
      MONGO_INITDB_ROOT_PASSWORD: dev
      volumes:
        mongodata:/data/db
```

Рисунок 3.9 – Конфігурація контейнера для MongoDB

Приклад використовуваних ресурсів MongoDB контейнера відображено на рисунку 3.10 за допомогою програми Docker Desktop.

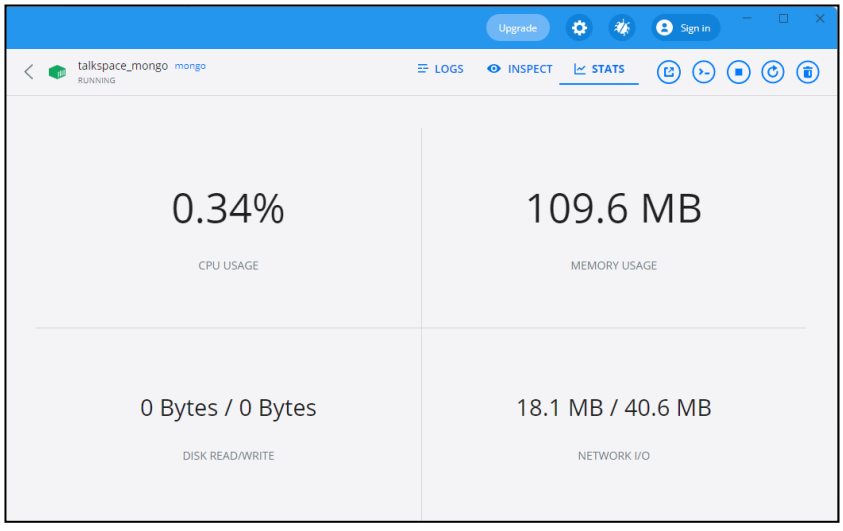
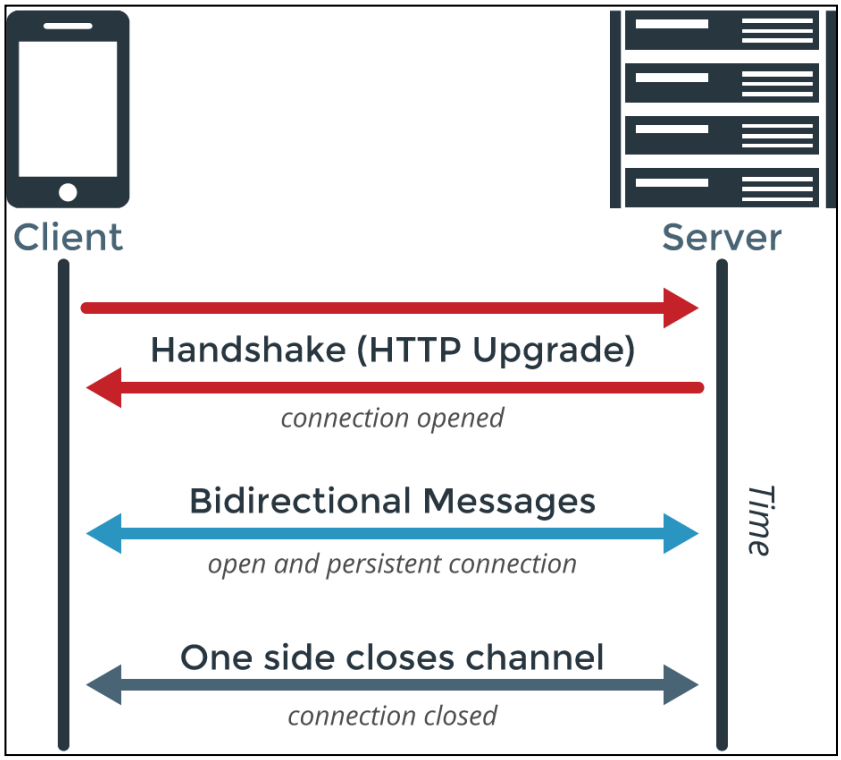


Рисунок 3.10 – Використання ресурсів MongoDB контейнера

Для реалізації двонаправленого зв'язку між клієнтським додатком і сервером використовується протокол WebSockets, який також є протоколом зв'язку з додатком для організації дзвінків, оскільки забезпечує кращу швидкість порівняно з HTTP запитами через відсутність установки з'єднання з хостом для кожного запиту.

Приклад організації спілкування між клієнтом та сервером наведено на рисунку 3.11.



Рисунку 3.11 – Обмін повідомлень між клієнтом та сервером у WebSockets протоколі [24]

3.1.3 Розробка додатку для дзвінка

Програма для організації дзвінків служить WebRTC сервером для обміну повідомленнями між учасниками. Для початкової установки з'єднання між клієнтом і сервером використовується WebSocket протокол, повідомлення якого надходить від клієнта до серверної програми і далі перенаправляються

сервером до програми для організації дзвінків. Слідуючи цьому клієнт повинен надіслати повідомлення тільки в один додаток.

Для забезпечення проведення дзвінків та обміну даними через WebRTC, було обрано бібліотеку Mediasoup [16]. Структура додатку відображена рисунку 3.12.

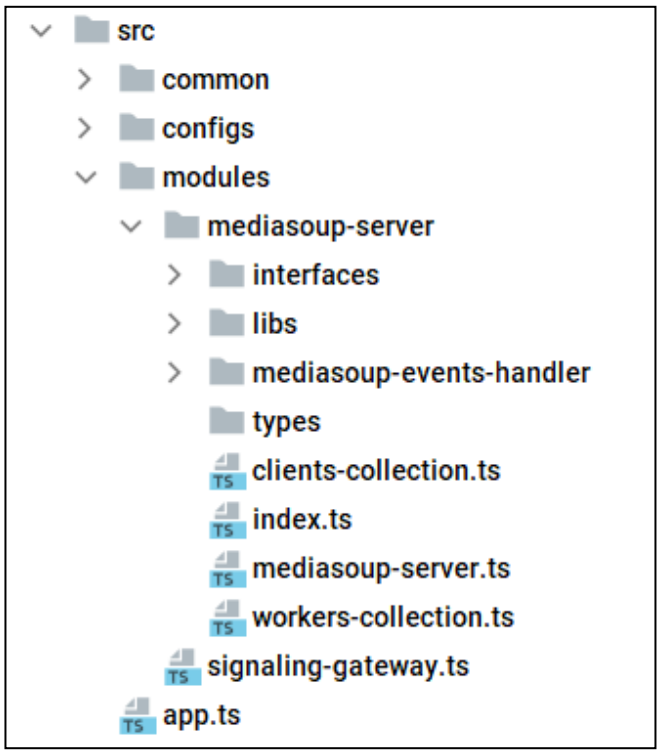


Рисунок 3.12 – Структура додатку для організації дзвінків

Додаток може розподіляти навантаження на різні потоки центрального процесора. Наприклад у поточній реалізації при створенні кімнати дзвінка вибирається потік процесора відносного того, що був обраний раніше, такий чином дозволяючи розподіляти навантаження рівномірно.

3.2 Використані бібліотеки

Для розробки платформи для організації групових відеодзвінків були використані такі основні бібліотеки:

а) react;

б) webpack;

в) react – redux;

г) socket.io – client;

д) axios.

React – це бібліотека JavaScript, яка створює інтерфейс для веб – сайтів та мобільних додатків. Він автоматично поєднується з іншими JavaScript – фреймворками та бібліотеками та включає невеликі фрагменти коду, які називають компонентами. Бібліотеки компонентів React оптимізують процес розробки інтерфейсу користувача, та забезпечує виняткову гнучкість завдяки високій модульності [19].

Webpack – це збирач статичних модулів. Webpack переглядає пакет та створює граф залежностей, який скалатається з різних модулів, які потрібні веб – додатку для правильної роботи. Далі в залежності від графу, він створює новий пакет який складається з мінімально необхідної кількості файлів, які можна легко підключити до html-файлу та використовувати для програми [20].

Redux – це безкоштовна бібліотека керування станом, яка працює на інтерфейсі програм JavaScript, керуючи станом кожного компонента в інтерфейсі користувача [21].

Socket.IO – JavaScript – бібліотека для веб – застосунків та обміну даними в реальному часі [22].

Axios – бібліотека з відкритим вихідним кодом, яка дозволяє виконувати HTTP – запити [23].

3.3 Робота з запитами

Так як основний канал зв’язку для спілкування між додатком клієнта та сервером є HTTP запити, було прийнято рішення використовувати патерн проектування «Фасад». Також були спроектовані класи для відправки запитів на сервер за допомогою власного інтерфейсу шляхом інкапсуляції функцій бібліотеки axios [17]. Це дозволить в майбутньому не залежати від реалізації конкретної бібліотеки та більш легко змінювати на іншу. Завдяки такій

організації розширення класів не потребує змін основного класу відправки запитів. Отриману структуру класів зображено на рисунку 3.13.

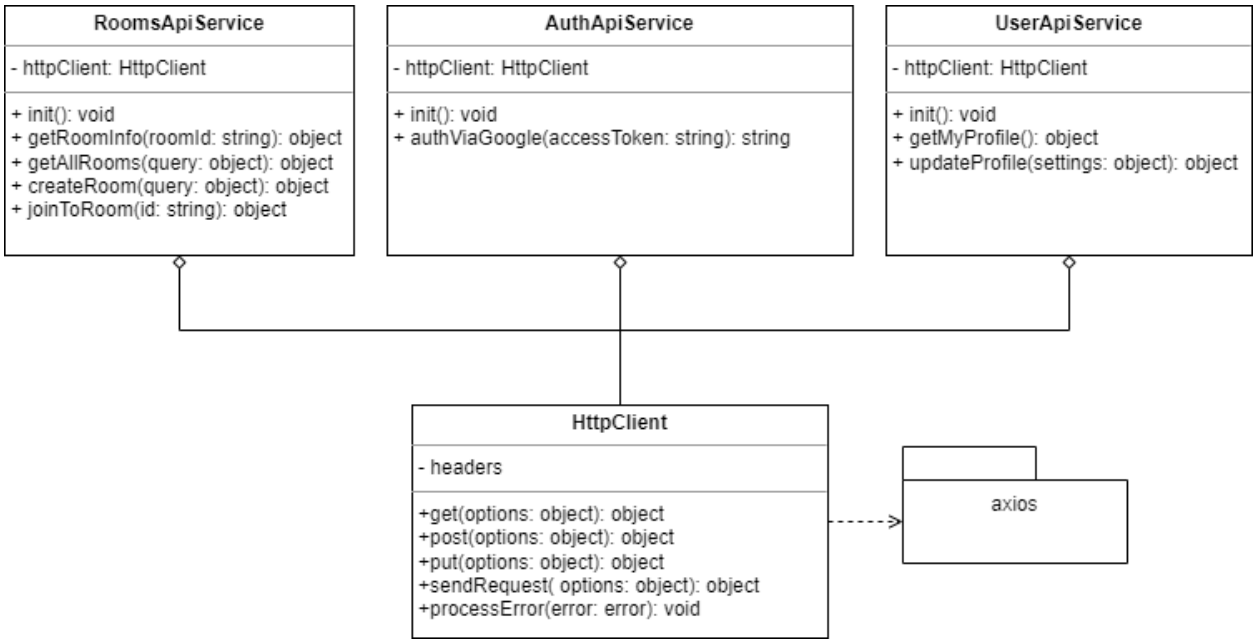


Рисунок 3.13 – Діаграма класів для роботи з http запитами

3.4 Прототипування інтерфейсів сайту

Платформа для спілкування має містить в собі багато вбудованих інструментів та функцій, прототипування розробляємої системи потрібне для більш кращого визначення зв’язку в програмі та зазделегіть вирішити можливі проблеми при побудові системи. Інтерфейс сайту матиме такі основні складові:

а) сторінка входу до сайту;

б) головна сторінка;

в) сторінка онлайн зустрічі;

г) сторінка налаштування профілю.

Сторінка входу на сайт мітитиме в собі модальне вікно для входу за допомогою Google.

На рисунку 3.14 наведено приклад прототипу сторінки входу.

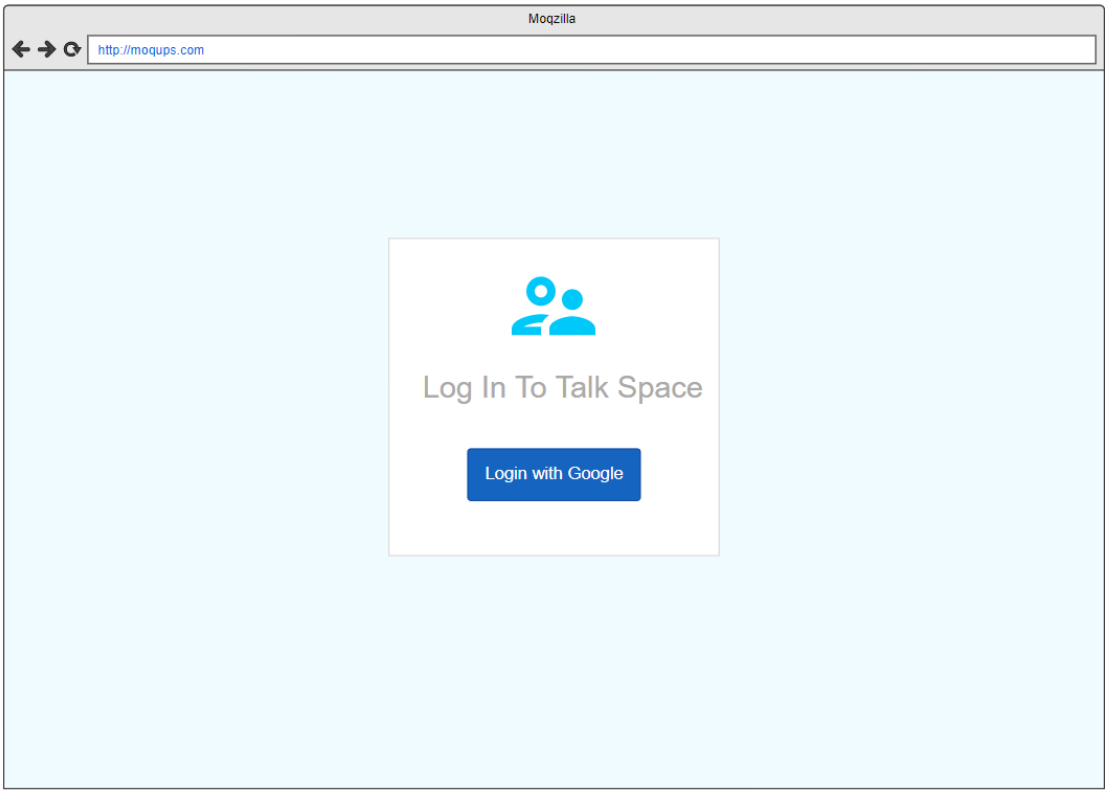


Рисунок 3.14 – Прототип сторінки входу до сайту.

Головна сторінка має містити наступні елементи:

а) профіль користувача (ім’я, прізвище та аватар);

б) список друзів з можливістю перегляду кімнати в якій вони знаходяться;

в) кнопка створення нової кімнати;

г) модальне вікно для створення кімнати;

д) модуль пошуку кімнати за назвою;

е) модуль фільтрування кімнати за мовою та категорією;

ж) список всіх кімнат;

з) кнопка долучитися до кімнати.

На рисунку 3.15 наведено приклад прототипу головної сторінки сайту.

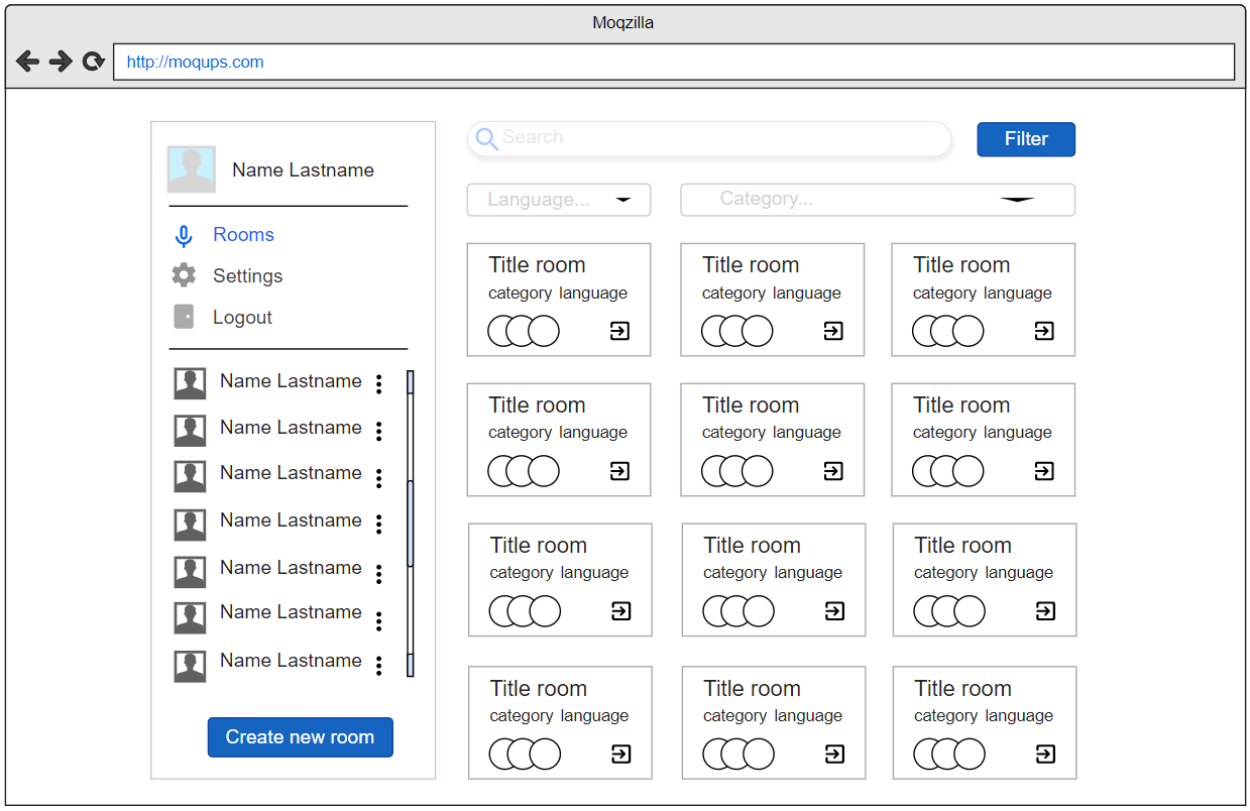


Рисунок 3.15 – Прототип головної сторінки сайту

У лівій частині сайту відображено меню в якому знаходиться, профіль користувача, меню переходу до вікна кімнат, налаштувань та виходу. Далі відображено список друзів та кнопка для створення нової кімнати, при її натискання повинно з’явитися модальне вікно створення кімнати яке зображено на рисунку 3.16. Правіше від меню знаходиться поле пошуку кімнати за назвою, та фільтр за мовою та категорією. Фільтр кімнат повинен з’являтися при натисканні кнопки «Filter».

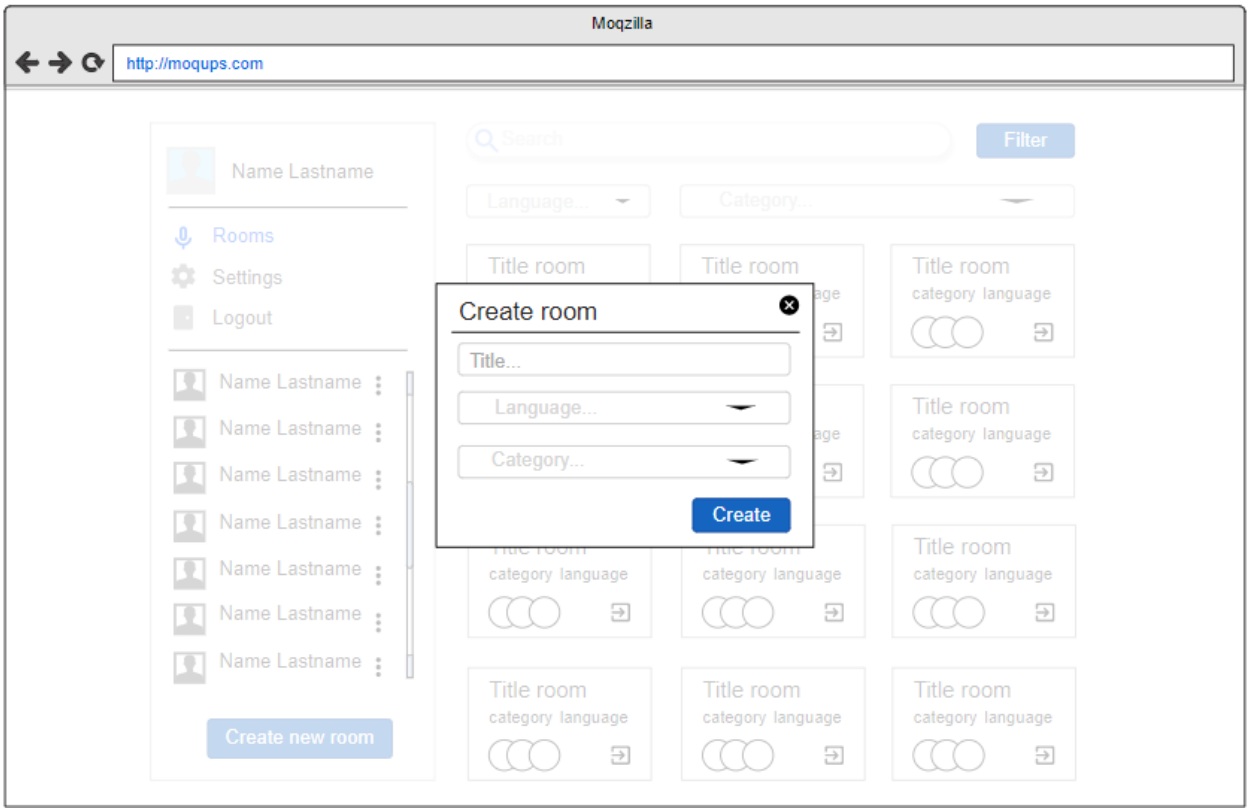


Рисунок 3.16 – Модальне вікно для створення кімнати

Для створення кімнати користувач повинен ввести назву, обрати мову та категорію, далі натиснути кнопку створити. Якщо кімнату створено успішно, користувача буде перенаправлено до сторінки кімнати яку зображено на рисунку 3.17. Всі інші користувачі зможуть побачити створену кімнату на головній сторінці сайту яку зображено на рисунку 3.15 правіше від меню.

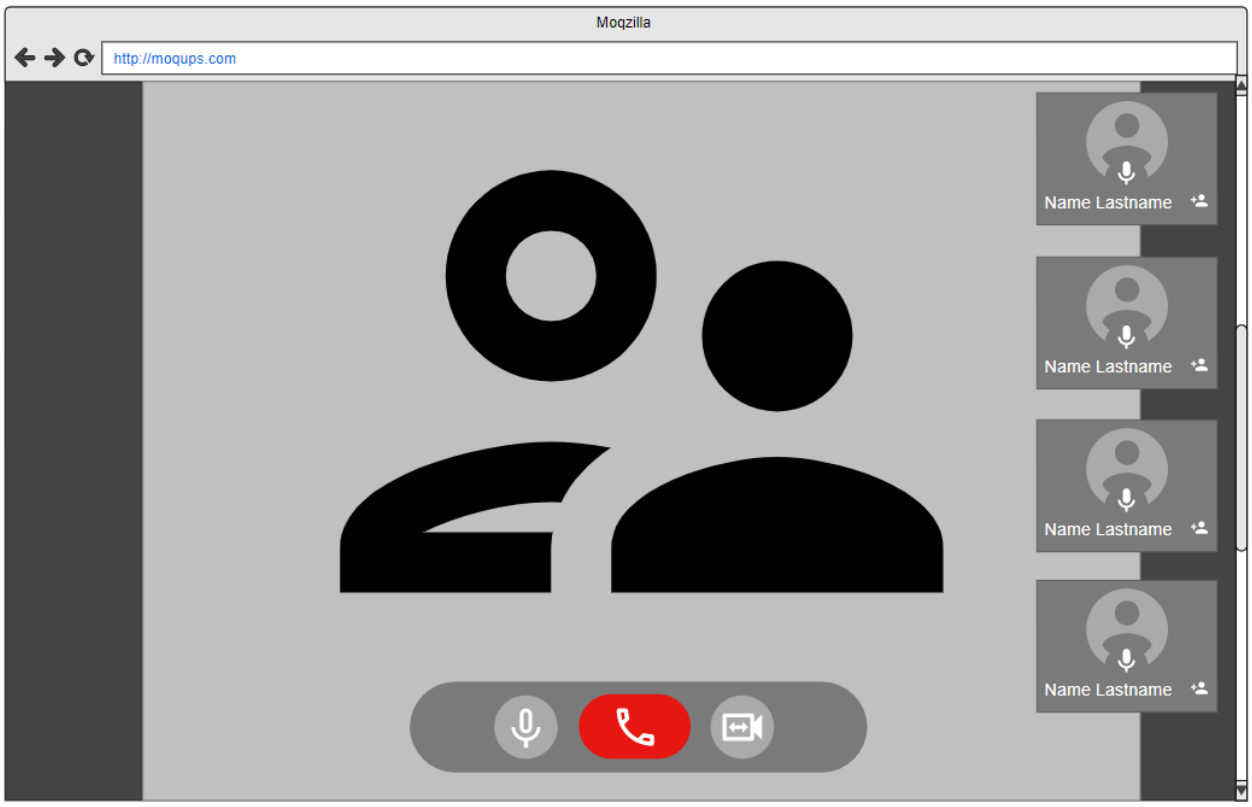


Рисунок 3.17 – Кімната дзвінку

У кімнаті дзвінку користувач має можливість додати людину з якою спілкується у друзі натиснувши на іконку поруч з ім’ям співрозмовника у списку який знаходиться справа.

Після додавання людини в друзі користувач зможе передивлятися в якій кімнаті знаходиться його друг та долучатися до розмови з ним.

Користувач також матиме можливість налаштовувати свій профіль, а саме:

а) змінити ім’я та прізвище;

б) обрати фото профілю.

Прототип сторінки налаштувань наведено на рисунку 3.18.

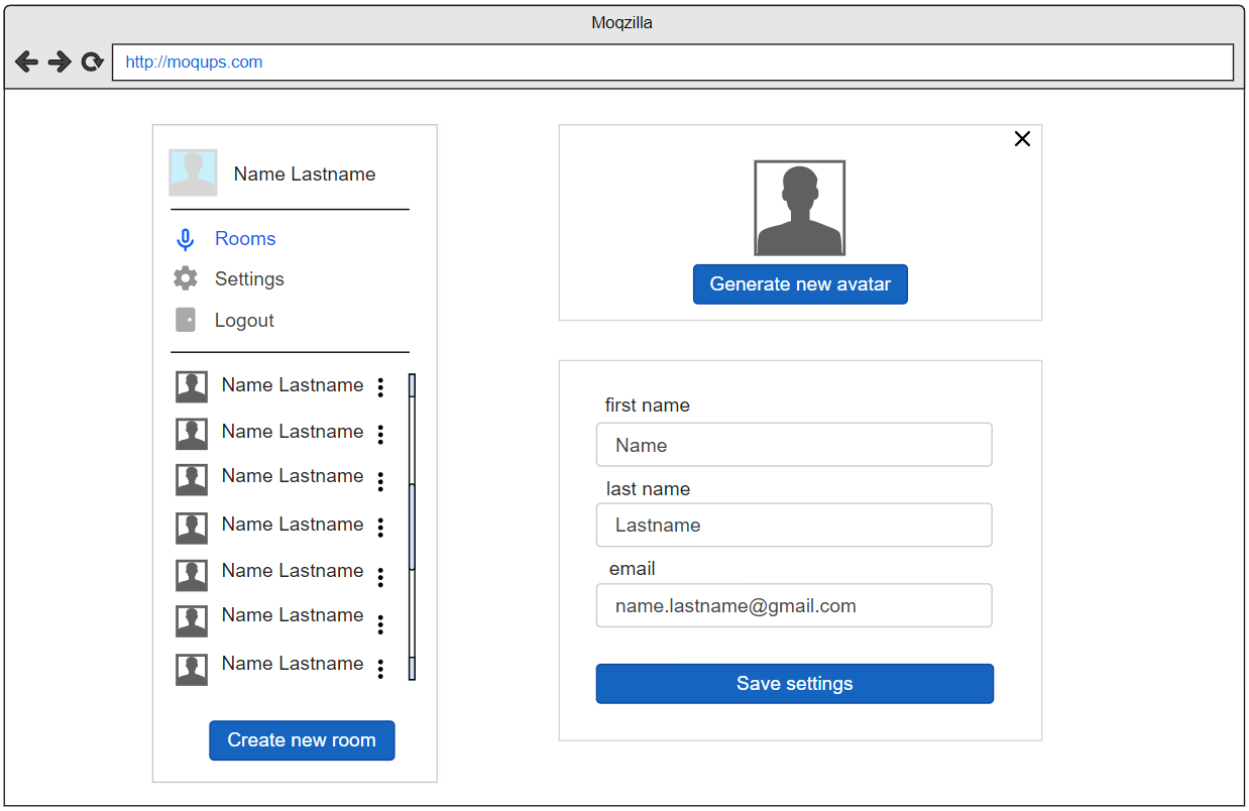


Рисунок 3.18 – Прототип сторінки налаштувань

3.5 Висновки за розділом

В даному розділі було описано структуру проекта та використані бібліотеки під час розробки системи. Далі було описано роботу з запитами та наведено діаграму класів для відправки HTTP запитів. Також в розділі було описано можливості користувача у системі та наведено прототип інтерфейсу кожної сторінки.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

4.1 Безпека з’єднання користувача з системою

Для безпечного з’єднання користувача з системою було використано механізм токенів який побудовано на основі Google API Server. Після успішної авторизації користувача у системі на клієнтська сторона повертає тимчасовий токен який можна використовувати для здійснення запитів на Google. Наприклад для отримання докладної інформації про користувача, включаючи аватарку та електронну пошту. Надалі при авторизації цей токен відправляється серверу розробленої системи для перевірки наявності такого користувача або реєстрації його в системі, після чого сервер повертає власний тимчасовий токен, завдяки якому може здійснювати запити до API [18].

Механізм спілкування з використанням тимчасових токенів гарантує що у разі крадіжки ці токени надалі стануть недійсними.

На рисунку 4.1 наведено процес отримання токена для входу в систему.

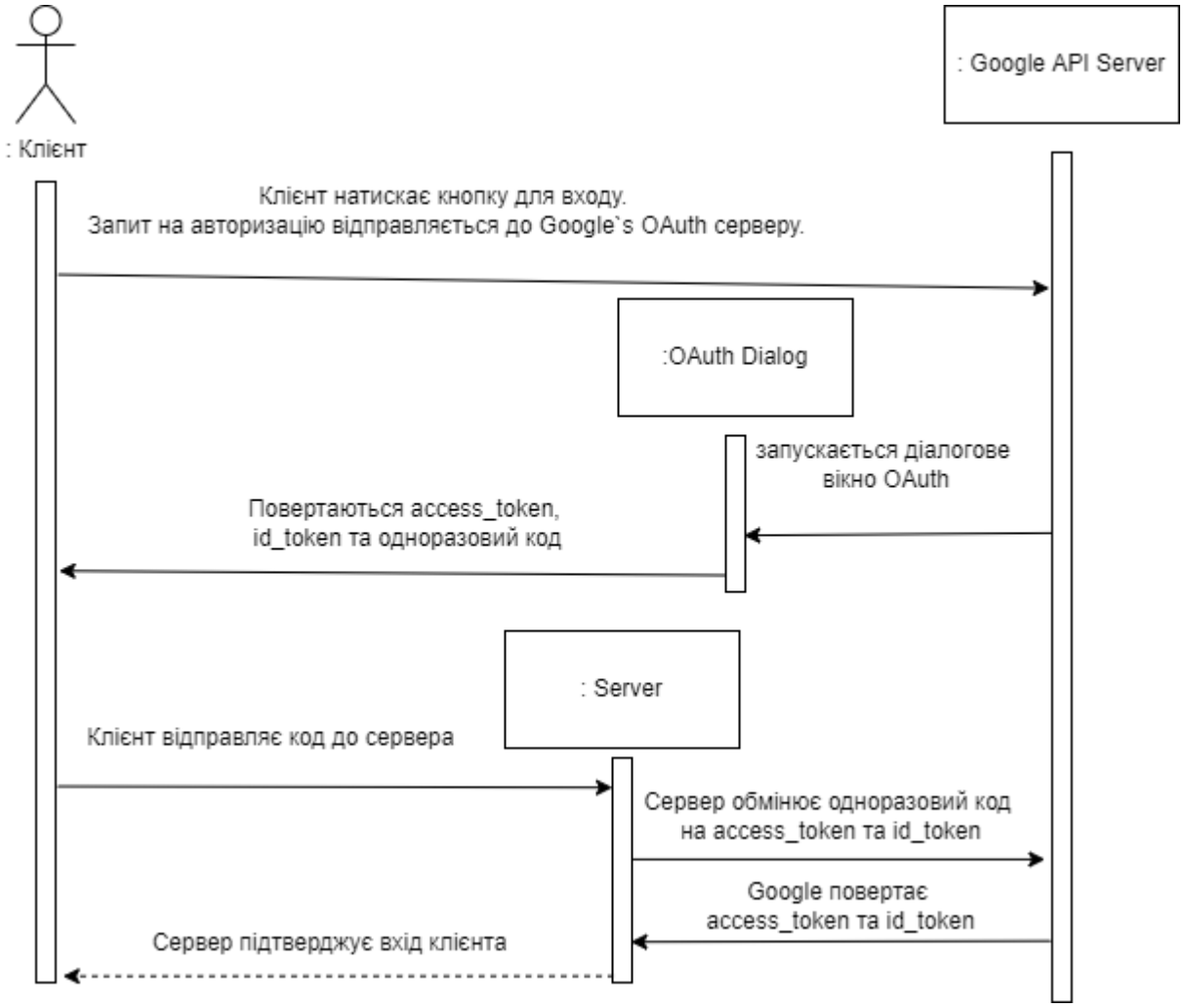


Рисунок 4.1 – Процес отримання токена для входу в систему

4.2 Опис користувацького інтерфейсу

Інтерфейс програми виконано у світлому стилі. Основною палітрою програми є білий та голубий кольори. Користувач має доступ до чотирьох основних сторінок системи, а саме:

а) сторінка авторизації;

б) головна сторінка з кімнатами;

в) сторінка налаштувань;

г) сторінка дзвінку.

На рисунку 4.2 наведено сторінку авторизації в системі.

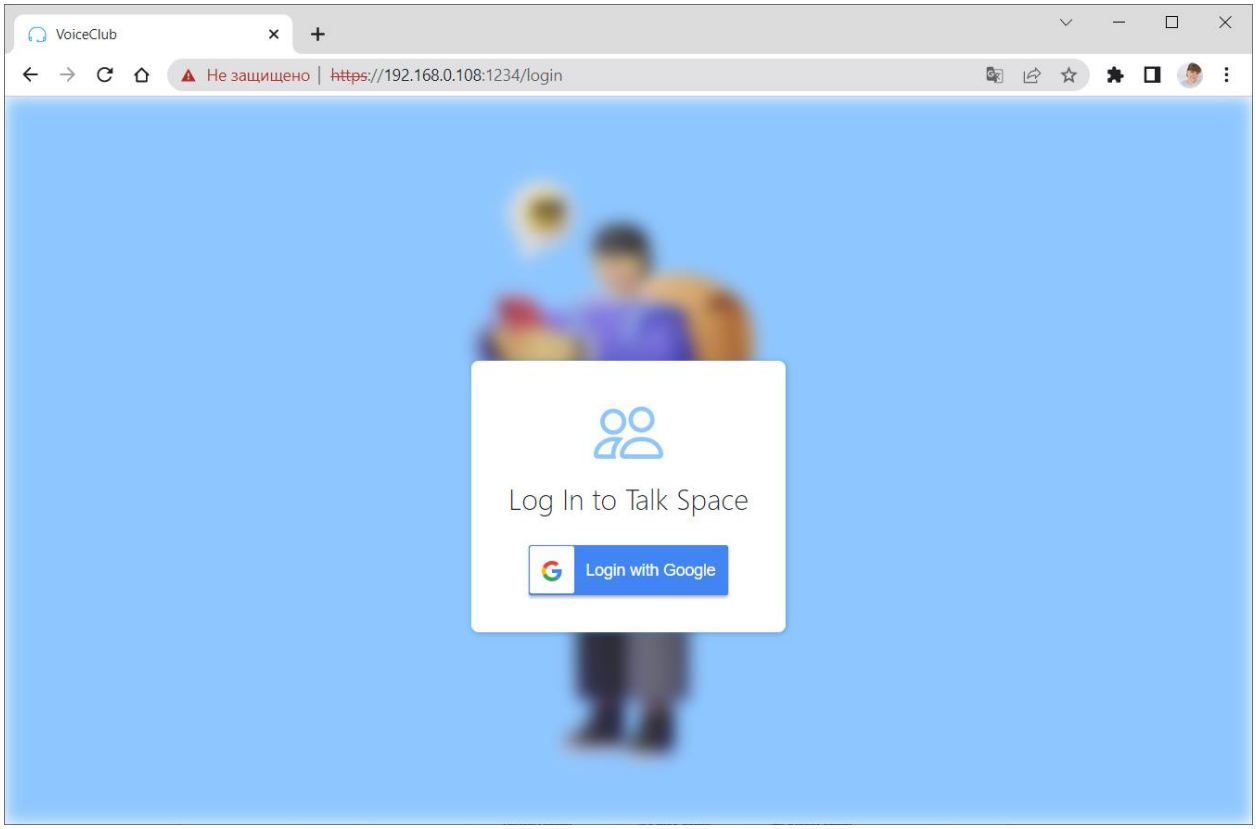


Рисунок 4.2 – Сторінка авторизації в системі

Після успішної авторизації в системі користувач буде направлений на головну сторінку сайту де знаходяться кімнати та меню системи. Головну сторінку сайту представлено на рисунку 4.3.

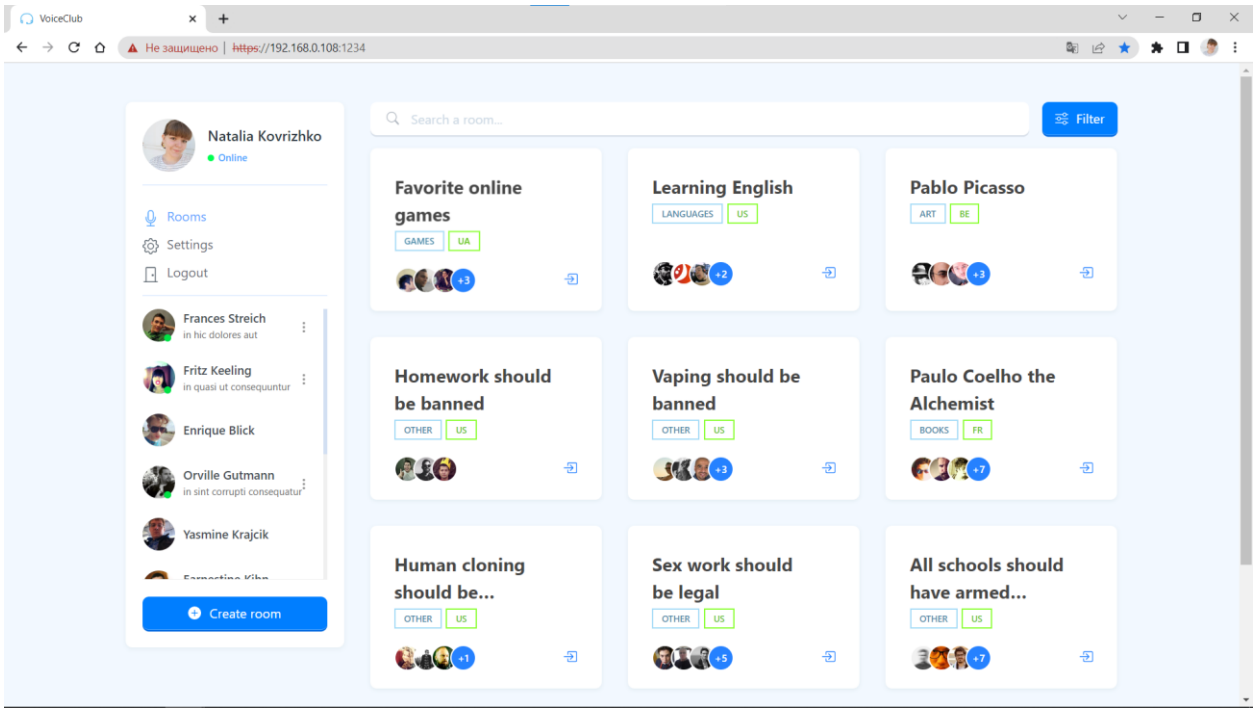


Рисунок 4.3 – Головна сторінка сайту

На головній сторінці користувач має можливість переглянути всі кімнати, знайти цікаву для нього кімнату за допомогою фільтрів, подивитися своїх друзів та долучитися до кімнати в якій знаходиться друг та створити власну кімнату.

Для перегляду кімнати в якій знаходиться друг користувачу необхідно натиснути на три точки поруч з іменем та прізвищем друга у списку друзів. А для переходу до його кімнати натиснути кнопку «Join» Приклад перегляду кімнати друга наведено на рисунку 4.4.

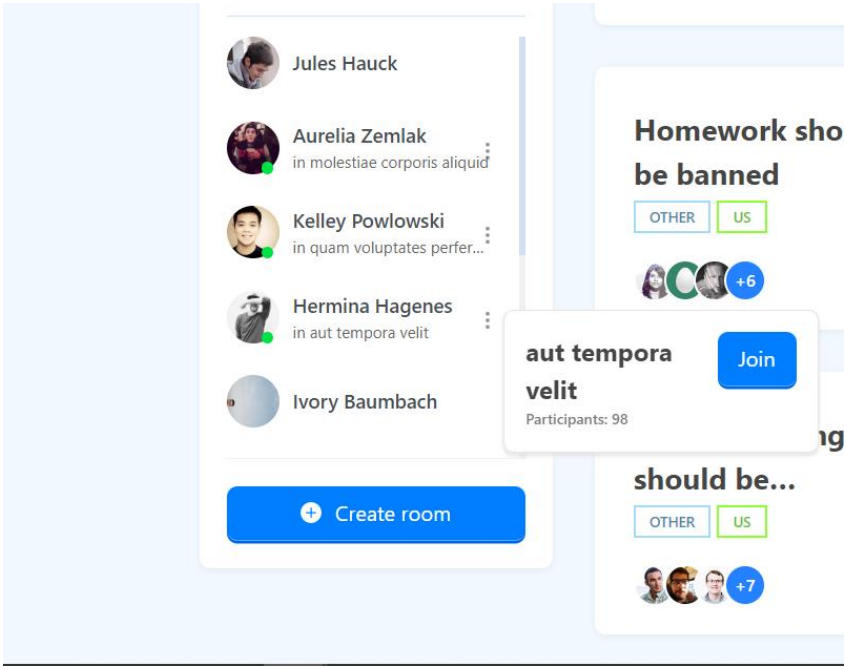


Рисунок 4.4 – Перегляд кімнати друга

На рисунку 4.5 наведено приклад фільтрації кімнати за мовою.

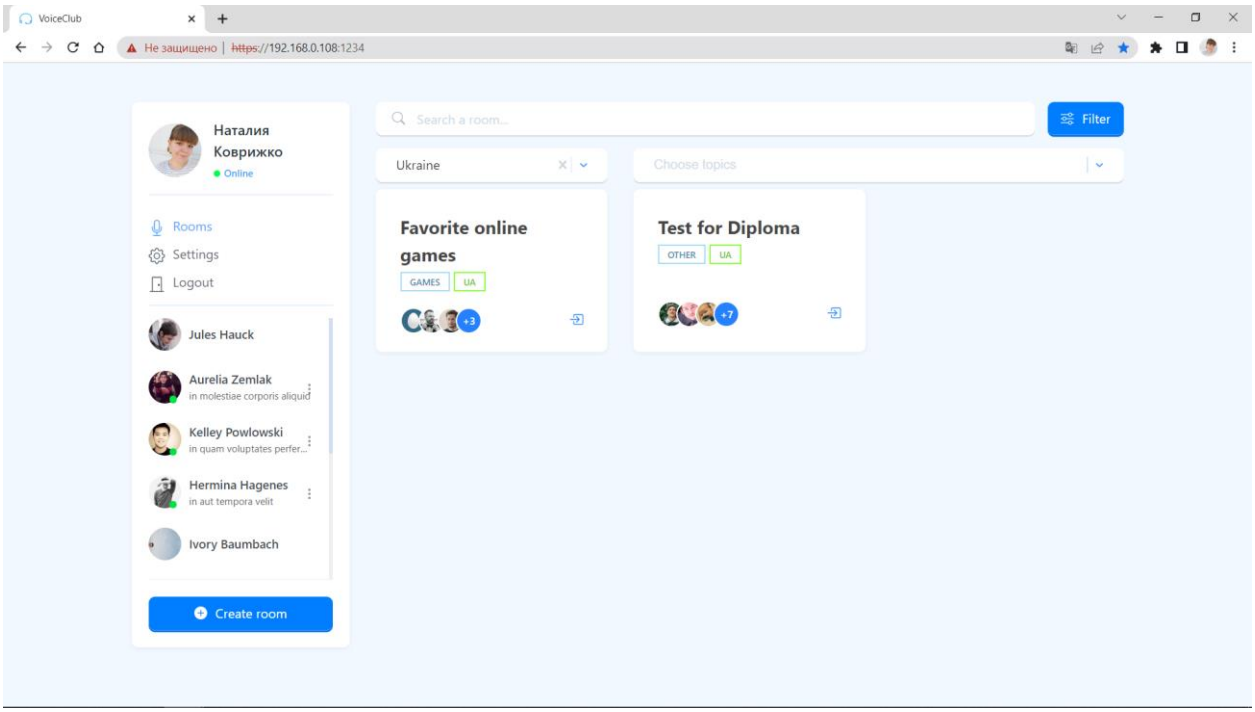


Рисунок 4.5 – Фільтр кімнат за мовою

На рисунку 4.6 наведено приклад пошуку кімнати за назвою.

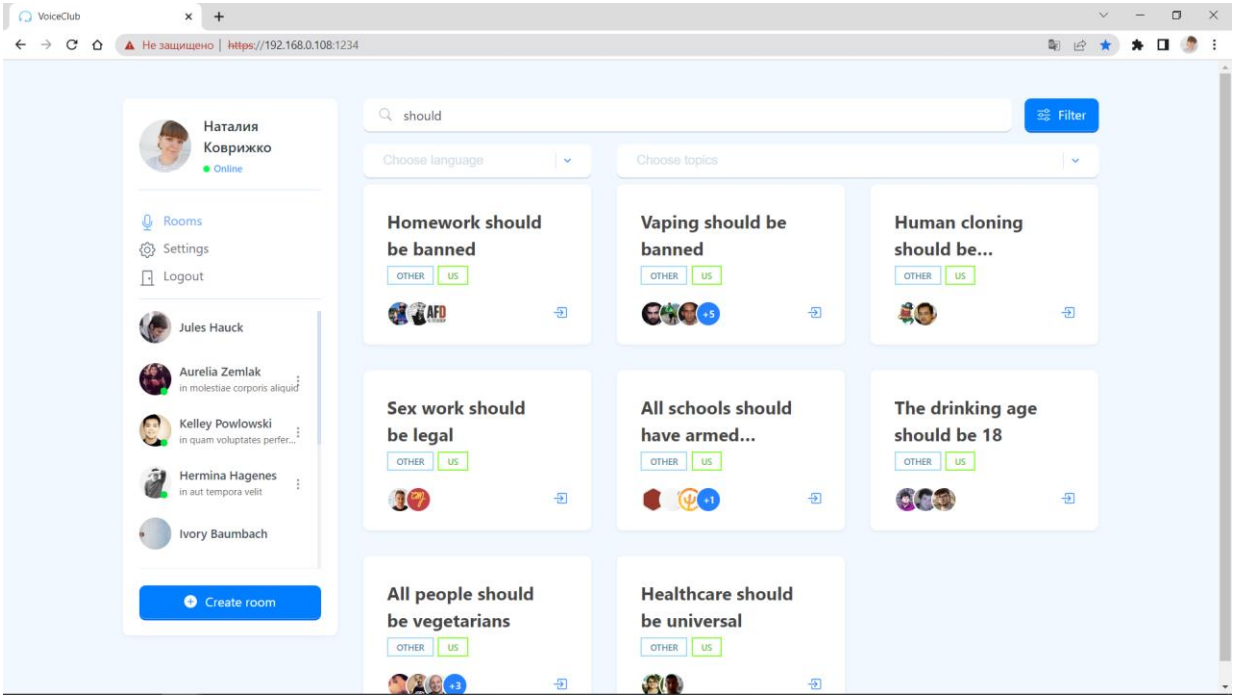


Рисунок 4.6 – Пошук кімнат за ключовим за назвою

Для створення власної кімнати користувачу необхідно натиснути кнопку «Create room» після чого йому буде показано модальне вікно для створення кімнати, де можна обрати мову та тему кімнати, а також вказати назву. На рисунку 4.7 зображено приклад створення кімнати.

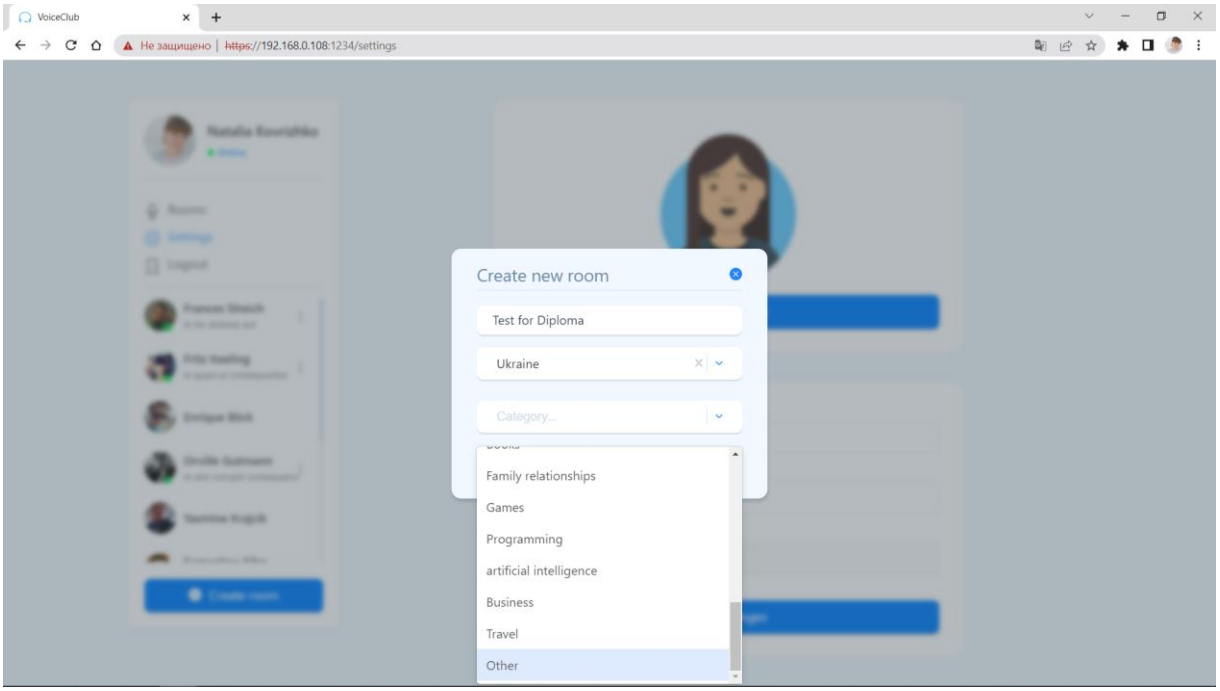


Рисунок 4.7 – Приклад створення кімнати

На рисунку 4.8 наведено результат створення кімнати.

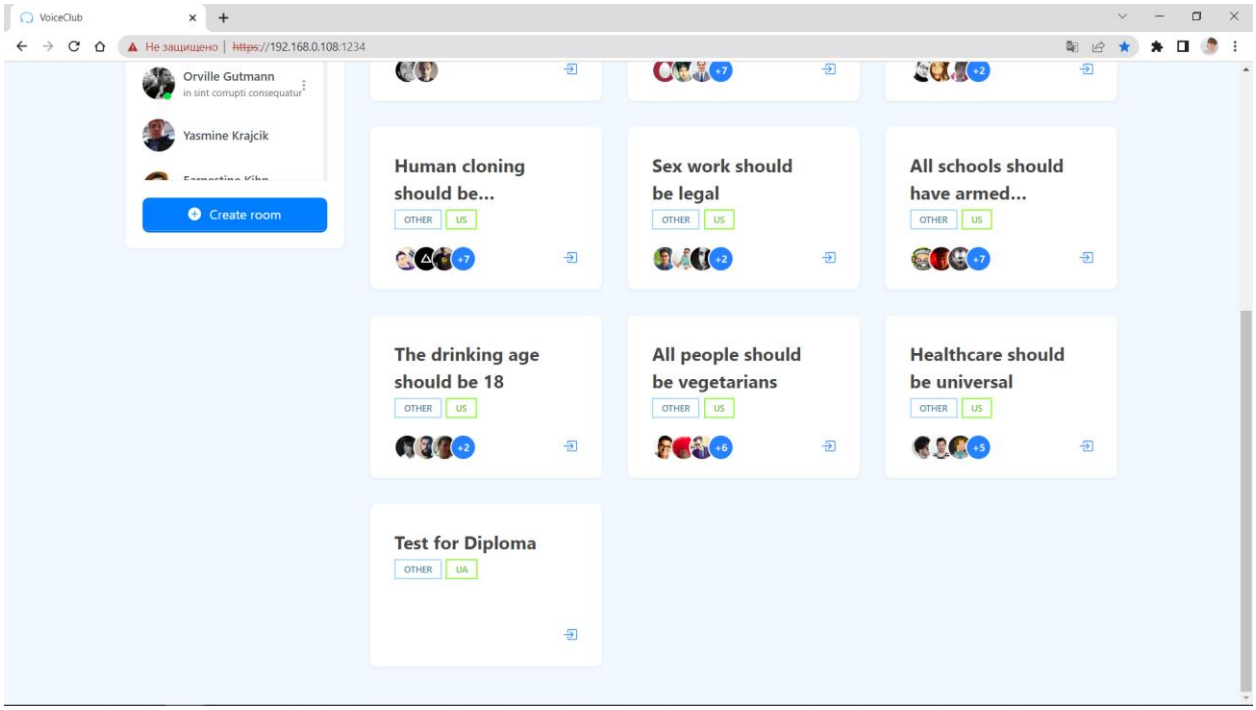


Рисунок 4.8 – Результат створення кімнати

На рисунку 4.9 відображено сторінка підключення до кімнату. Зверху відображено назву кімнати по середині людей які підключені та внизу кнопка щоб долучитися.

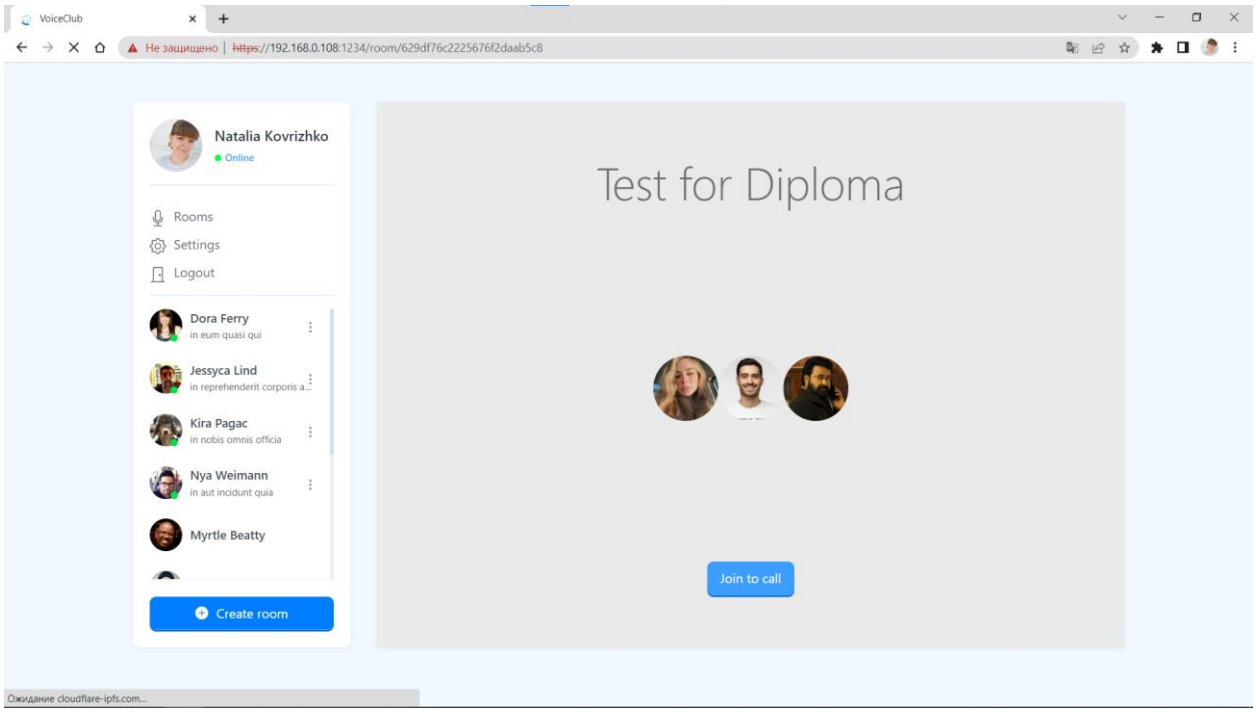


Рисунок 4.9 – Сторінка підключення до кімнати

На рисунку 4.10 зображено сторінку кімнати.

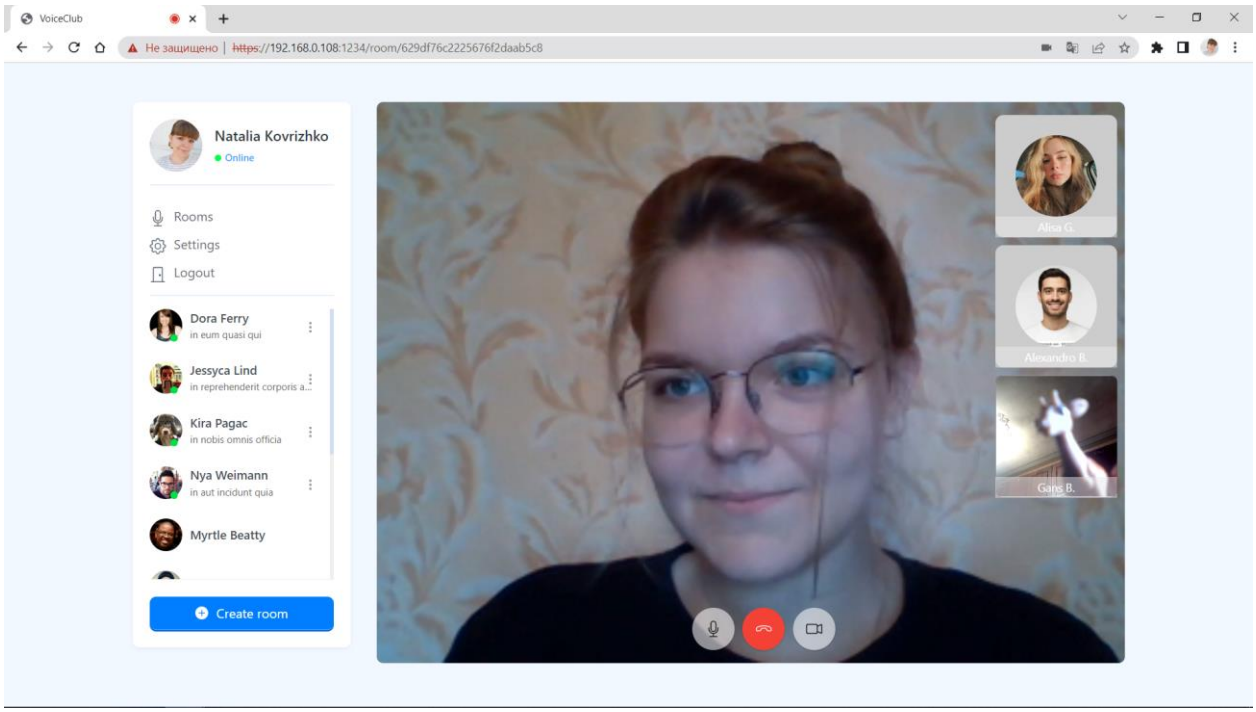


Рисунок 4.10 – Сторінка кімнати

На рисунку 4.11 зображено сторінку кімнати з вимкненою камерою.

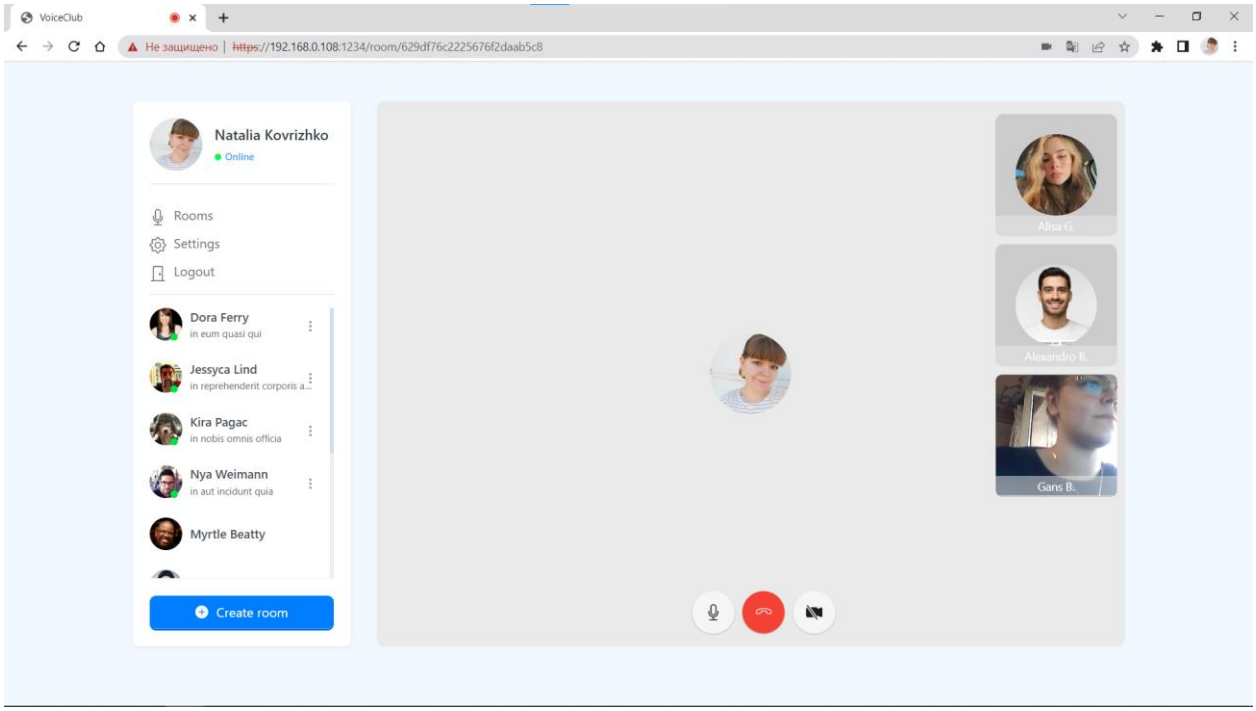


Рисунок 4.11 – Сторінка дзвінку з вимкненою камерою

Також користувач може налаштовувати свій профіль. Залишати головне фото яке синхронізується з акаунту Google під час авторизації або згенерувати рандомне та змінити ім’я та прізвище. На рисунку 4.12 наведено сторінку налаштувань профіля.

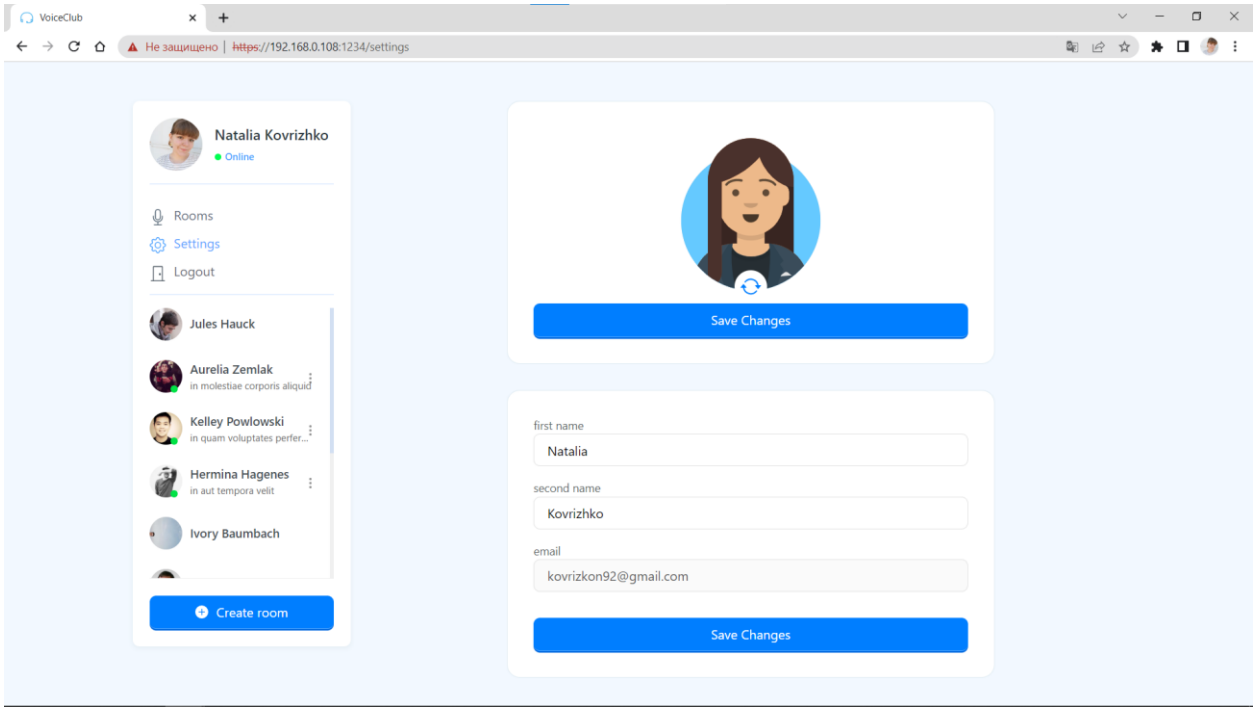


Рисунок 4.12 – Сторінка налаштувань профіля

На рисунку 4.13 наведено результат зміни ім’я та прізвища користувача.

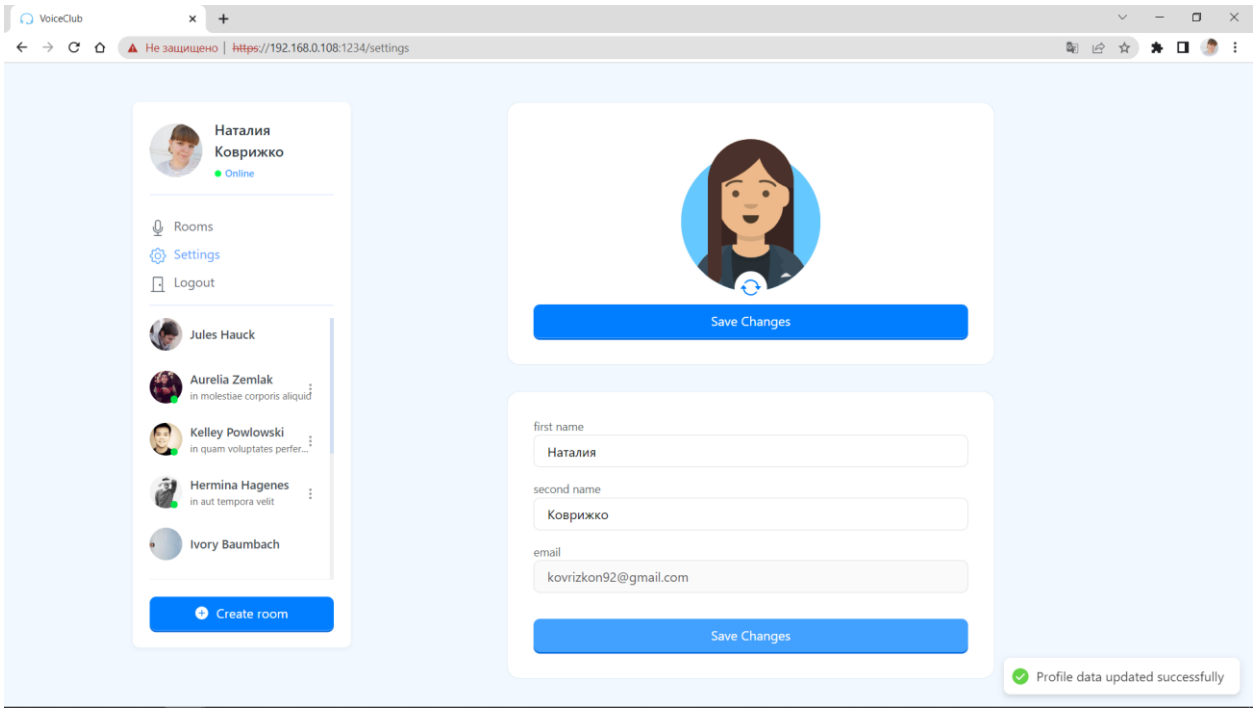


Рисунок 4.13 – Результат зміни ім’я та прізвища користувача

Компоненти сайту були розроблені таким чином, щоб сайт відображався правильно у мобільному, планшетному та широкоекранному режимі. На рисунку 4.14 та 4.15 наведено приклад відображення деяких сторінок сайту з мобільного пристрою.

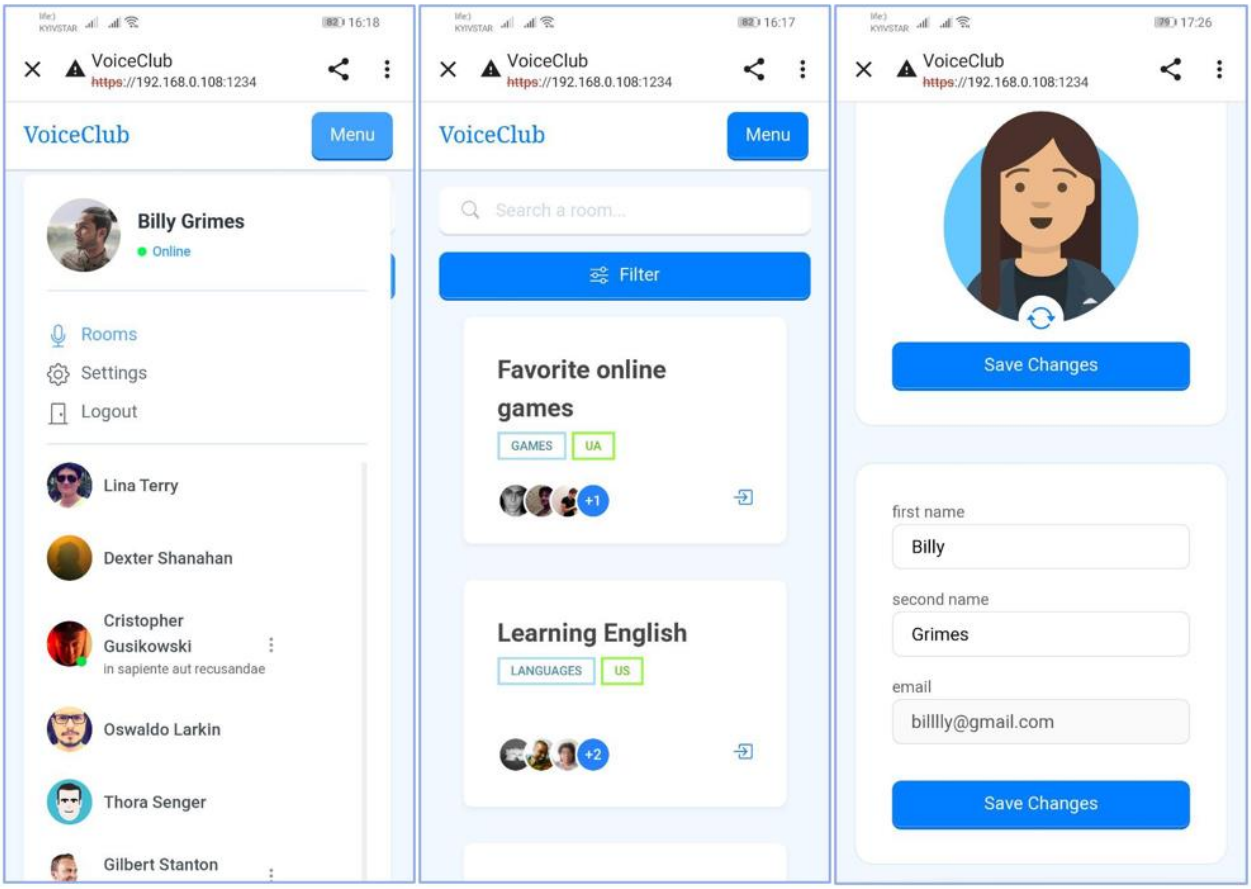


Рисунок 4.14 – Відображення деяких сторінок сайту у мобільному пристрої

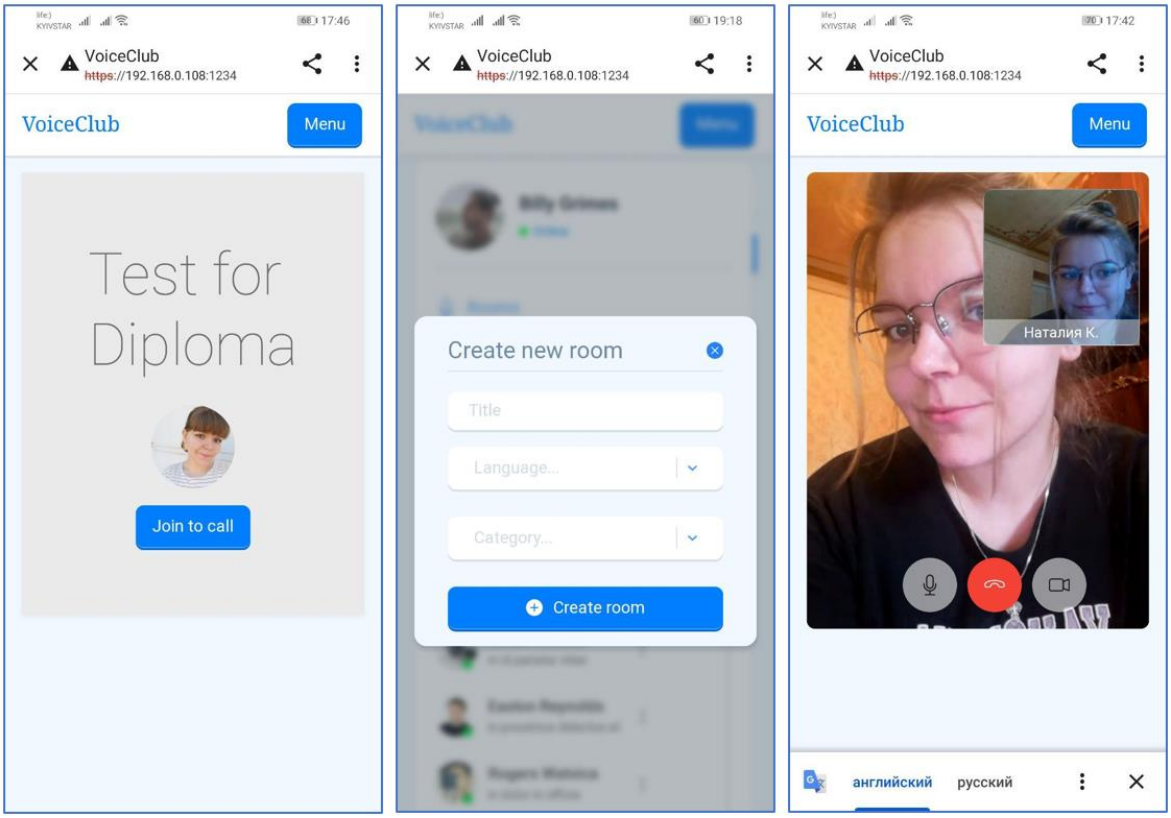


Рисунок 4.15 – Відображення деяких сторінок сайту у мобільному пристрої

На рисунку 4.16 наведено відображення сайту на широкому екрані.

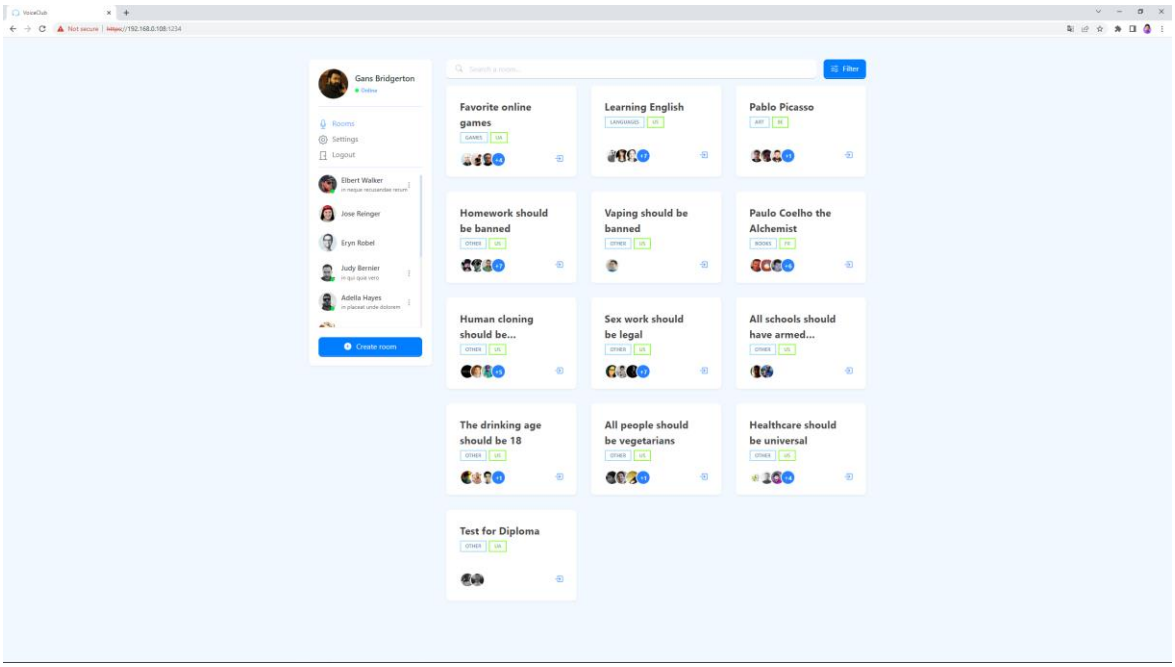


Рисунок 4. 16 – Відображення сайту на широкому екрані

4.3 Висновки за розділом

У даному розділі було описано як проходить процес входу у систему для користувача та чому він є безпечним, та описано і проілюстровано користувацький інтерфейс сайту.

5 ТЕСТУВАННЯ

5.1 Використані методи тестування

Тестування системи дозволяє легко вносити зміни в код, та не боятися зламати залежності в проекті. Завдяки використанню тестів у вихідному коді збільшується надійність систем, але це займає більший час на розробку, тому що крім основного коду треба писати також тести до них. Але в подальшому при збільшенні складності системи тести покращують супроводження проекту.

Таким чином можна виділити кілька переваг тестування програмного забезпечення:

а) тестування програмного забезпечення необхідне для виявлення дефектів та помилок, допущених на етапах розробки;

б) тестування необхідне для того, щоб надати клієнтам такі можливості, як постачання високоякісного продукту;

в) якісний продукт, доставлений клієнтам, допомагає завоювати їхню довіру;

г) тестування необхідне для ефективної роботи програми.

Існують різні методи тестування програмного забезпечення спрямованих на різні критерії, прикладом може бути поведінка системи, тестування на продуктивність, використання ресурсів. На рисунку 5.1 наведено приклад різних рівнів тестування.

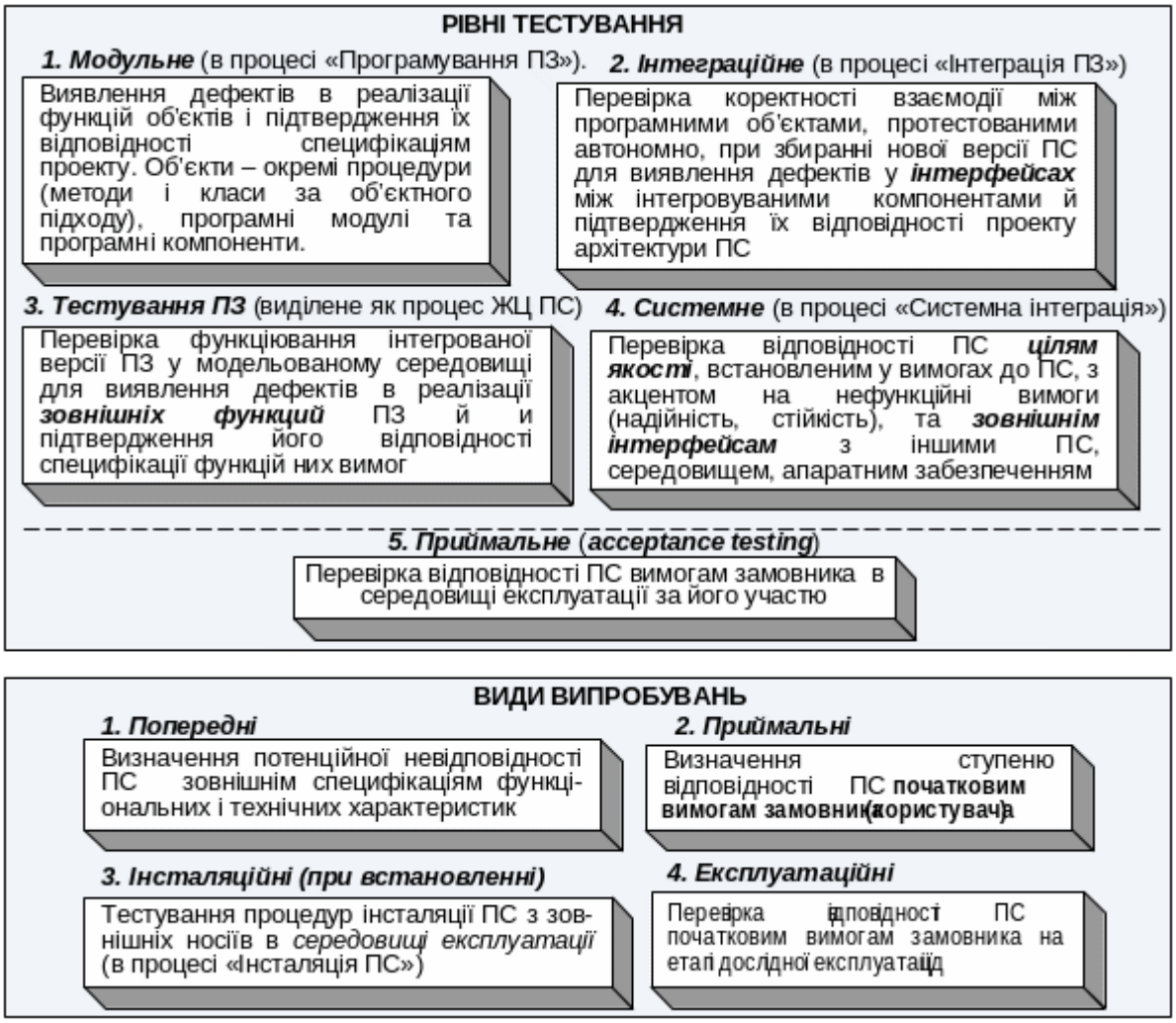


Рисунок 5.1 – Рівні тестування ПЗ [26]

На рисунку 5.2 розташовані види тестування програмного забезпечення від конкретнішого до загального.

Рисунок 5.2 – Види тестування ПЗ за рівнем [27]

Модульне тестування – це метод тестування програмного забезпечення, при якому для кожного окремого модуля в програмі, поведінка перевіряється окремо та ізольовано від інших.

Функціональне тестування проводиться для визначення, наскільки компонент або розроблена система відповідають функціональним вимогам, описаним у специфікаціях.

Тестування інтерфейсу користувача - є точкою взаємодії клієнта і продукту.

5.2 Програмна реалізація тестів

Під час розробки програмної системи розроблялися модульні випробування до основних компонентів системи. Тести розроблялися за допомогою бібліотеки Jest для серверного додатку для організації дзвінків.

Приклад проходження тестів програмного забезпечення показаний рисунку 5.3.

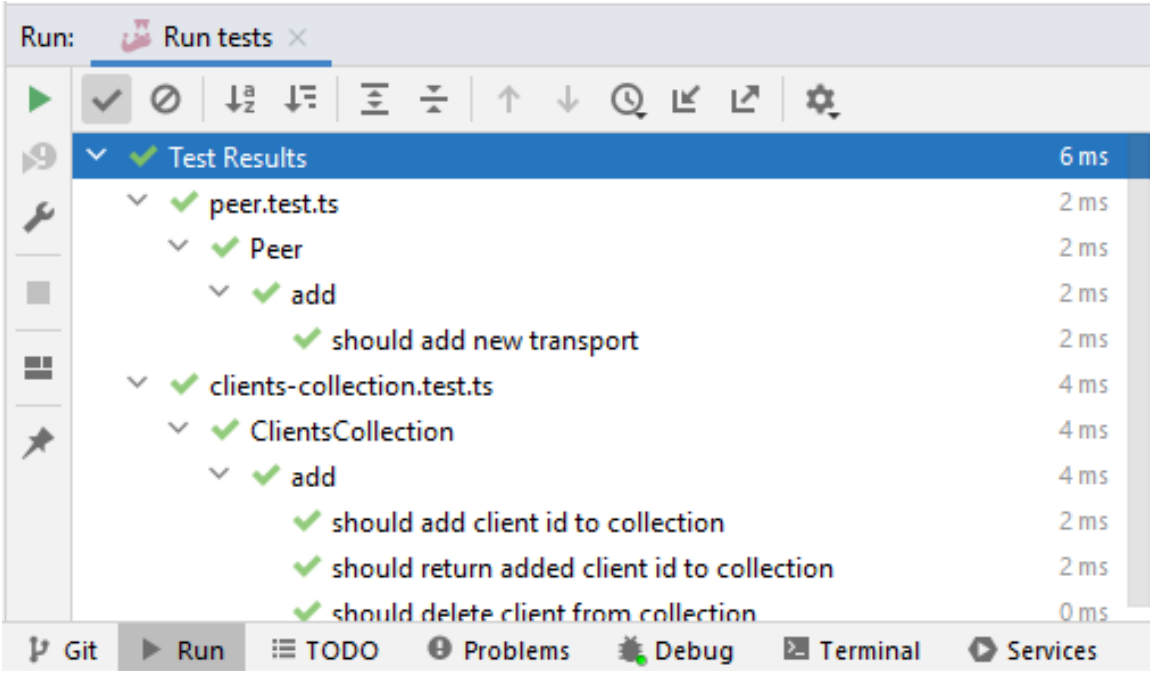


Рисунок 5.3 – Проходження тестів програмного забезпечення

5.3 Висновки за розділом

Завдяки написанню тестів до модулів вдалося створити більш надійний і якісний продукт, завдяки застосуванню модульного тестування вдалося виявити багато помилок на етапі розробки при спробі змінити існуючий модуль у програмі, без модульних тестів це стало б можливо тільки в кінцевому продукті. Оскільки модульне тестування передбачало написання тестів до конкретних модулів або функцій системи, то було приємне рішення розробки тестів тільки для деяких основних модулів, які використовуються для здійснення дзвінків.

ВИСНОВКИ

Під час виконання роботи було проведено детальний аналіз протоколів передачі даних, методів стиснення медіаданих, різних топологій та розглянуто готові рішення для організації групових дзвінків. Було розглянуто стандарт WebRTC, спроектовано архітектуру програмної системи, визначено функціональні, не функціональні та бізнес вимоги, обрано топологію для розробки, визначено апаратні та програмні умови. Спроектовано UML діаграми класів, побудовано діаграму варіантів використання яка показує можливості користувача в системі, побудовано діаграму діяльності входу до системи.

Під час розробки сайту були використані різні бібліотеки для створення інтерфейсу, збірки модулів, роботи з запитами та інші. Розроблено різні прототипи інтерфейсу сайту.

Результатом виконання кваліфікованої роботи є сайт для створення групових відео та аудіо дзвінків з використанням технології WebRTC з топологією SFU, засобами JavaScript, Node.js, MongoDB, Docker, AWS.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Датаграмма [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D1%82%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0

2. What is video compression and what does it do [Електронний ресурс] – Режим доступу: https://marketing.istockphoto.com/blog/what-is-video-compression/

3. What is Video Encoding? Codecs and Compression Techniques [Електронний ресурс] – Режим доступу: https://blog.video.ibm.com/streaming-video-tips/what-is-video-encoding-codecs-compression-techniques/

4. WebRTC implementation method(Mesh, SFU, MCU) [Електронний ресурс] – Режим доступу: https://millo-l.github.io/WebRTC-implementation-method-Mesh-SFU-MCU/

5. Jitsi [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Jitsi

6. Comparative Study of WebRTC Open Source SFUs for Video Conferencing [Електронний ресурс] – Режим доступу: https://mediasoup.org/resources/CoSMo_ComparativeStudyOfWebrtcOpenSourceSfusForVideoConferencing.pdf

7. What is WebRTC and What does it Mean for Customer Service? [Електронний ресурс] – Режим доступу: https://gettalkative.com/what-is-webrtc;

8. Насколько безопасна технология WebRTC 3CX? [Електронний ресурс]. – Режим доступу: https://www.3cx.ru/blog/webrtc-secure/

9. A Study of WebRTC Security [Електронний ресурс] – Режим доступу: https://webrtc-security.github.io

10. Complete Guide to WebRTC and How it Works [Електронний ресурс] – Режим доступу: https://telnyx.com/resources/what-is-webrtc

11. Как это устроено: видеоконференции ВКонтакте на безлимитное число участников [Електронний ресурс] – Режим доступу: https://habr.com/ru/company/vk/blog/575358/

12. О модульном тестировании [Електронний ресурс] – Режим доступу: https://cyberleninka.ru/article/n/o-modulnom-testirovanii-na-c/viewer

13. Redux [Електронний ресурс] – Режим доступу: https://redux.js.org/

14. Webpack [Електронний ресурс] – Режим доступу: https://webpack.js.org/

15. NestJS [Електронний ресурс] – Режим доступу: https://nestjs.com/

16. Mediasoup – Режим доступу: https://mediasoup.org/;

17. Axios [Електронний ресурс] – Режим доступу: https://www.axios.com/

18. Использование OAuth 2.0 для приложений веб-сервера [Електронний ресурс] – Режим доступу: https://developers.google.com/identity/protocols/oauth2/web-server

19. The Top React Component Libraries that are Worth Trying [Електронний ресурс] – Режим доступу: https://technostacks.com/blog/react-component-libraries

20. An intro to Webpack: what it is and how to use it [Електронний ресурс] – Режим доступу: https://www.freecodecamp.org/news/an-intro-to-webpack-what-it-is-and-how-to-use-it-8304ecdc3c60/

21. What Is React Redux and How Do You Use it? [Електронний ресурс] – Режим доступу: https://www.makeuseof.com/how-to-use-redux/

22. Socket.IO [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Socket.IO#:~:text=Socket.IO%20%E2%80%94%20JavaScript

23. Как использовать axios в приложении javascript [Електронний ресурс] – Режим доступу: https://www.8host.com/blog/kak-ispolzovat-axios-v-prilozhenii-javascript

24. WebSocket Application Monitoring [Електронний ресурс] – Режим доступу: https://www.dotcom-monitor.com/blog/2022/03/22/websocket-application-monitoring/;

25. Webpack [Електронний ресурс] – Режим доступу: http://webpack.github.io/;

26. Методи пошуку помилок в програмах [Електронний ресурс] – Режим доступу: https://studfile.net/preview/1850561/;

27. Система автоматизації тестування фреймворків Web порталу [Електронний ресурс] – Режим доступу: https://ela.kpi.ua/bitstream/123456789/31092/1/Ustenko_bakalavr.pdf;

28. Analyzing Performance Differences Between MySQL and MongoDB [Електронний ресурс] – Режим доступу: https://www.researchgate.net/publication/349764039_Analyzing_Performance_Differences_Between_MySQL_and_MongoDB.

Додаток А

Зауваження нормоконтролера

Таблиця А.1 – Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження
С. 22-23		Невірно оформлено продовження таблиці 2.1
ПЗ		Нещільне заповнення сторінок ПЗ, посилання на джерела
С. 24-25		Невірний перелік

Додаток Б

Презентація до захисту кваліфікованої роботи

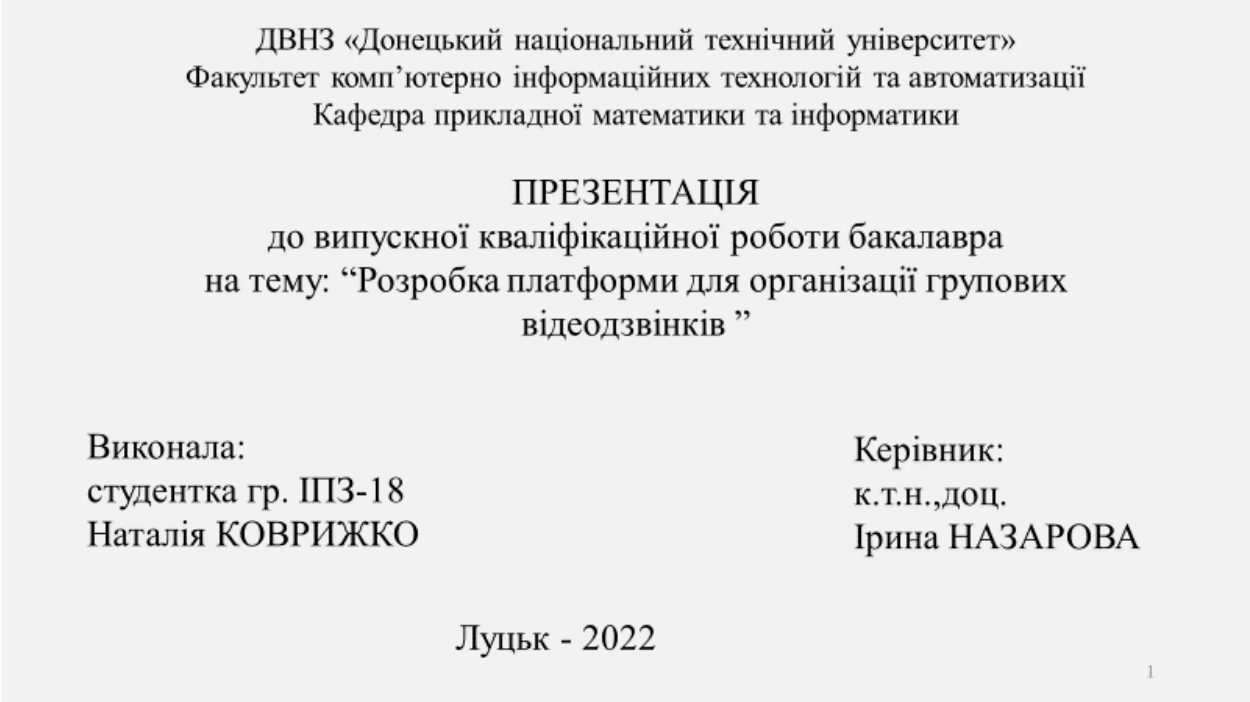


Рисунок Б.1 – Титульний слайд презентації

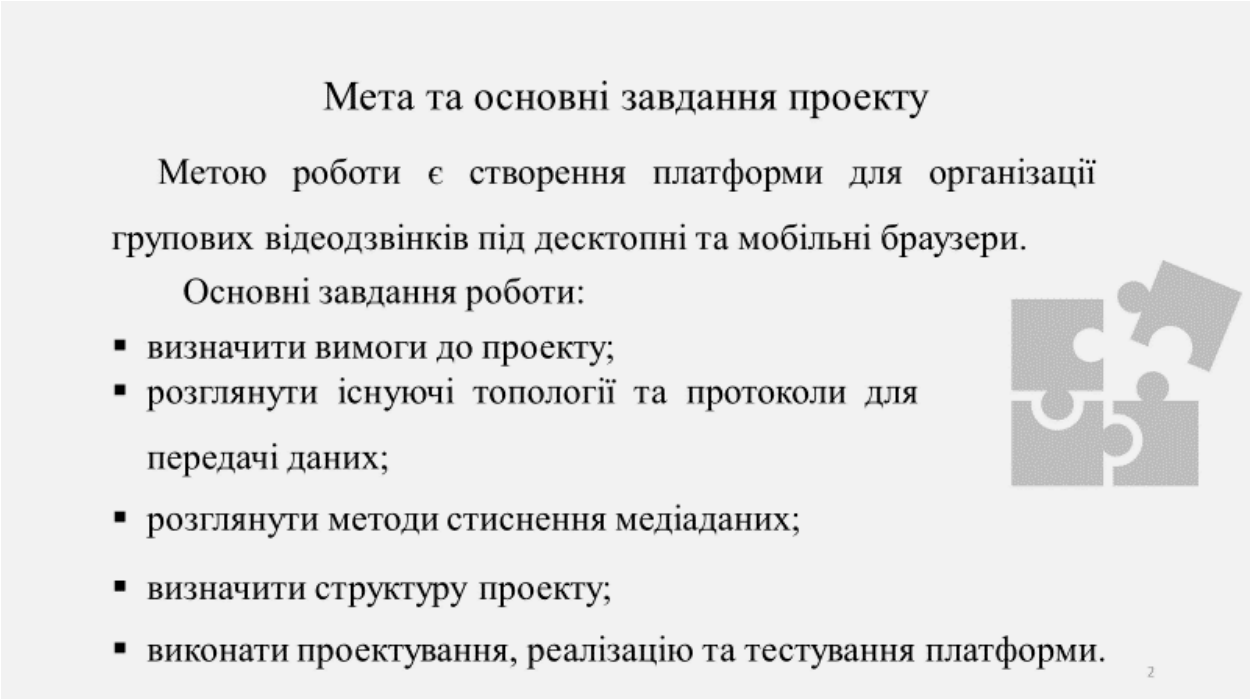


Рисунок Б.2 – Другий слайд презентації

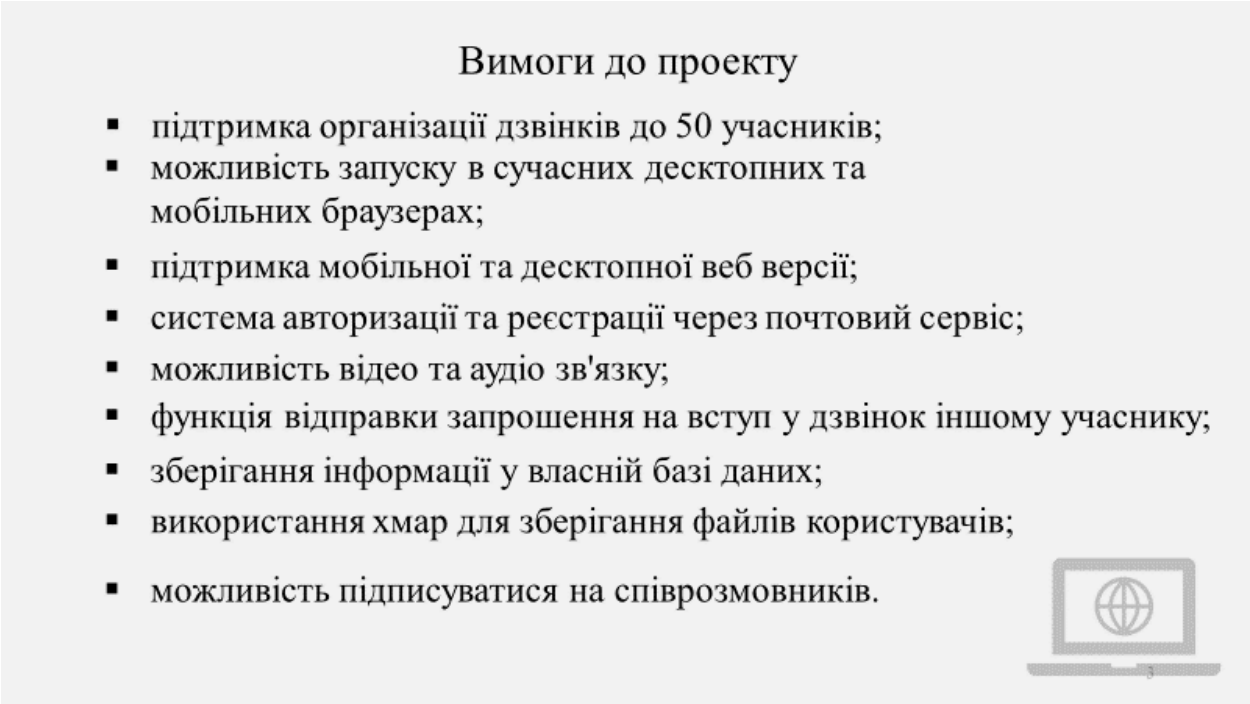


Рисунок Б.3 – Третій слайд презентації

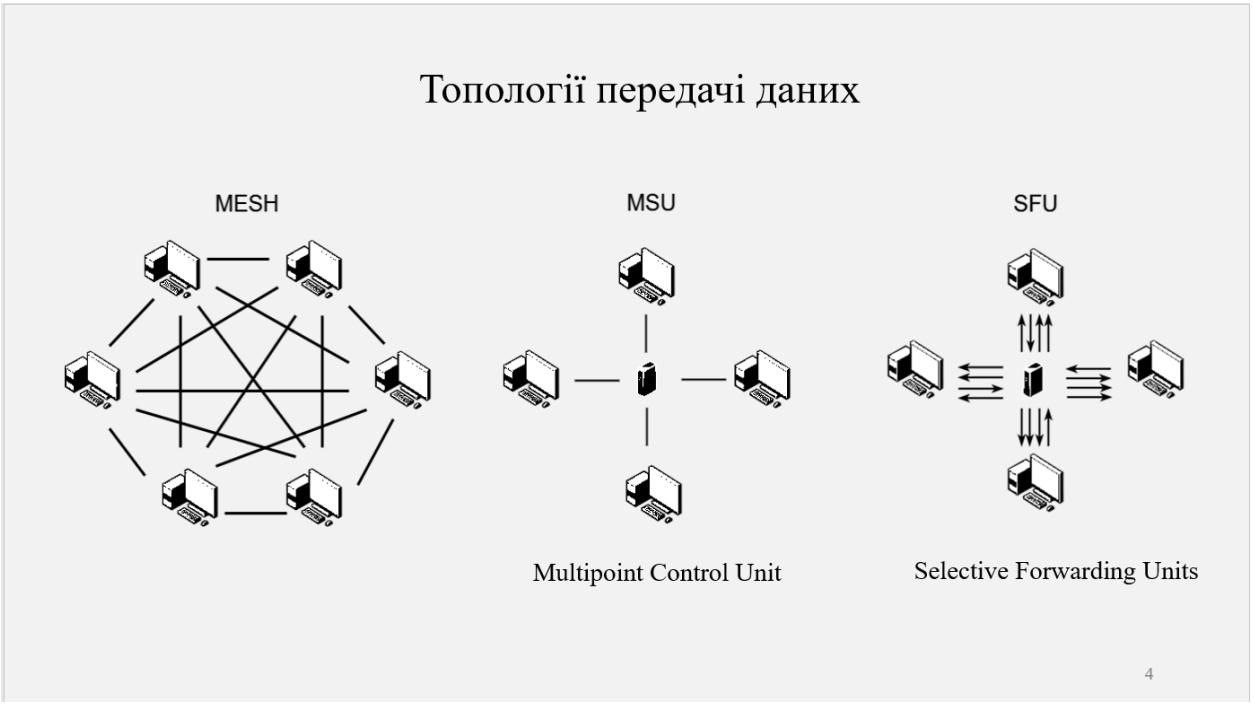


Рисунок Б.4 – Четвертий слайд презентації

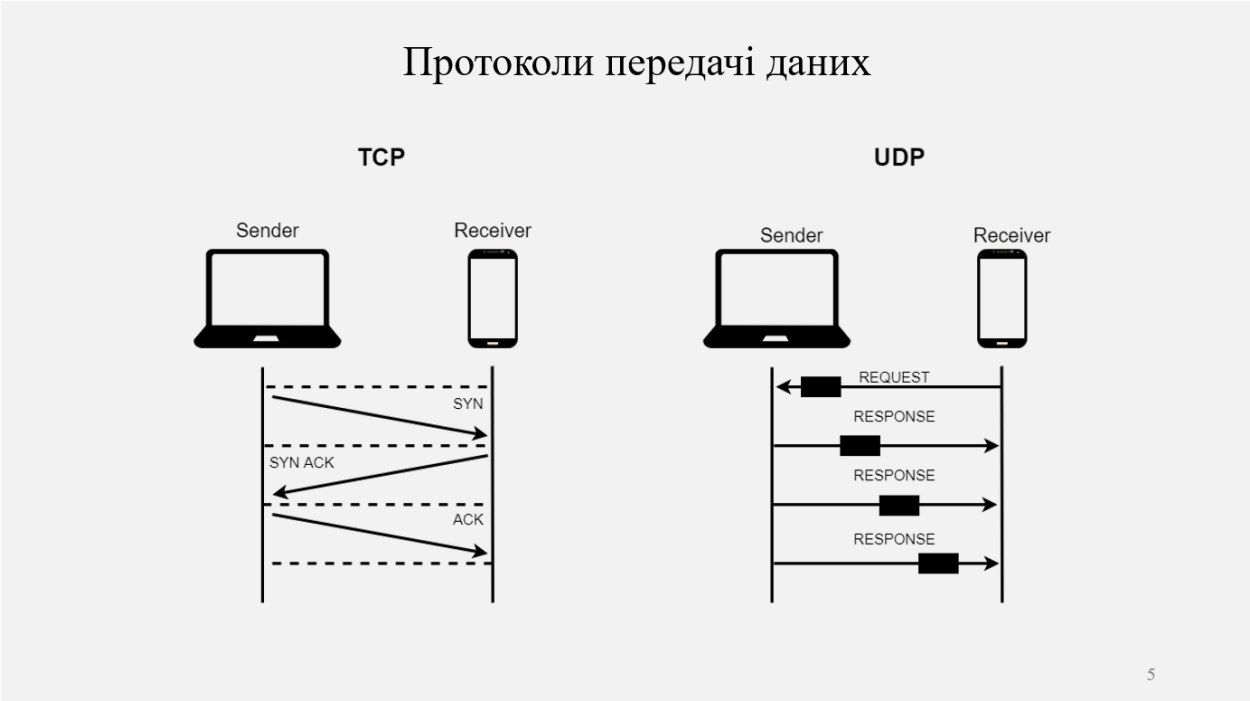


Рисунок Б.5 – П’ятий слайд презентації

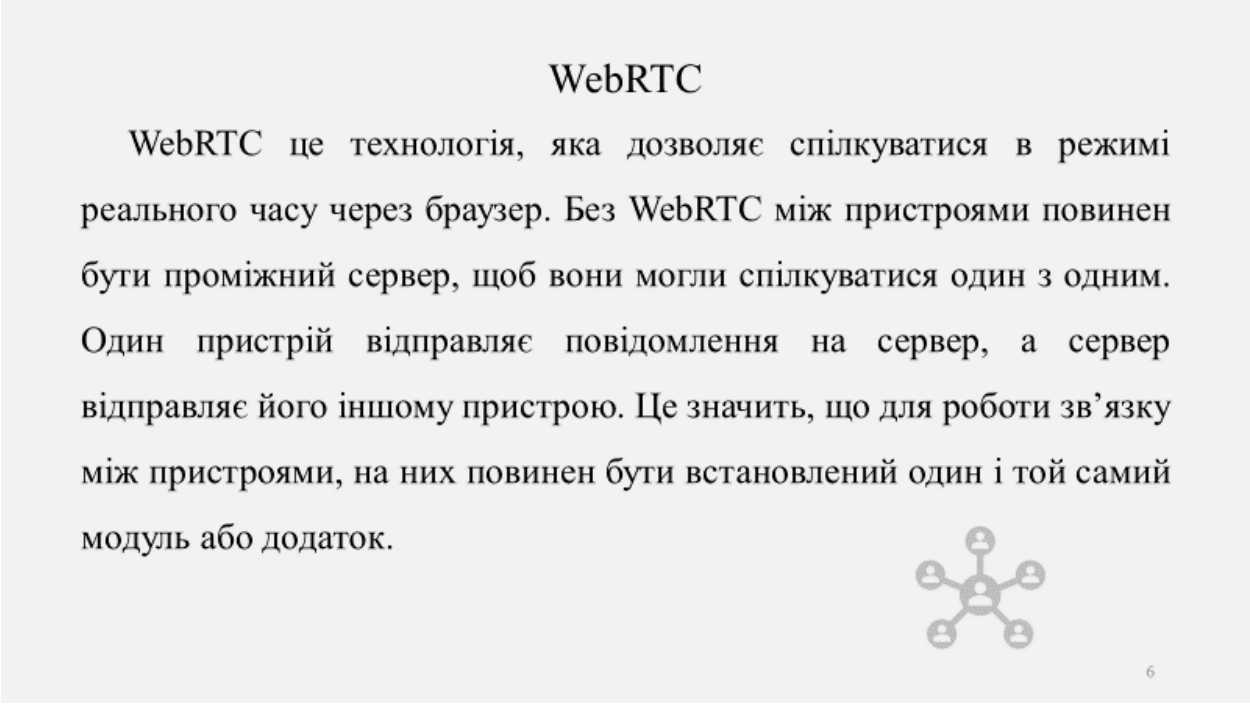
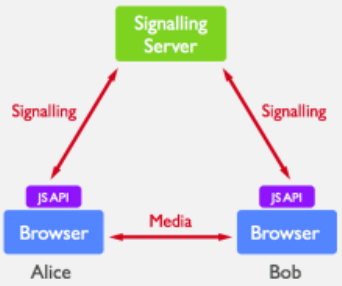


Рисунок Б.6 – Шостий слайд презентації

WebRTC

Включає:

- Кодування даних;
- Розкодування даних;
- Стиснення даних;
- Безпечну передачу по мережі (TLS);
- Потокову передачу даних між пристроями (UDP).



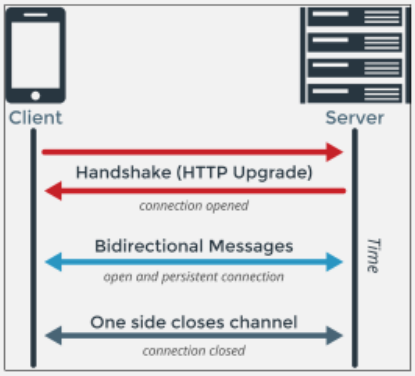
Для встановлення з'єднання між пристроями використовується сигнальний сервер, який потрібен для одноразового встановлення з'єднання. Сигнальним сервером може виступати що завгодно, найчастіше це власні сервери.

7

Рисунок Б.7 – Сьомий слайд презентації

WebSocket

Для реалізації двонаправленого зв'язку між клієнтським додатком і сервером використовується протокол WebSockets, який також є протоколом зв'язку з додатком для організації дзвінків, оскільки забезпечує кращу швидкість порівняно з HTTP запитами через відсутність установки з'єднання з хостом для кожного запиту.



8

Рисунок Б.8 – Восьмий слайд презентації

Структура проекту

Користувач має доступ до чотирьох основних сторінок системи:

- сторінка авторизації;
- головна сторінка з кімнатами;
- сторінка налаштувань;
- сторінка дзвінку.

9

Рисунок Б.9 – Дев’ятий слайд презентації

Архітектура проекту

Платформа складається з трьох основних модулів:

- клієнтський додаток;
- серверний додаток;
- серверний додаток для обробки дзвінків.

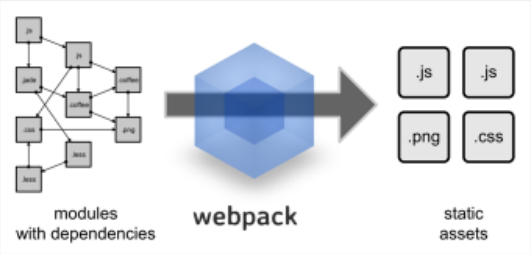
Кожен модуль ізольований від іншого, а загальні функції та дані розміщені в окремих бібліотеках, щоб уникнути дублювання в коді. Завдяки відділенню програми для організації дзвінків було отримано можливість вносити зміни в основний серверний додаток і перезапускати його в подальшому без перебоїв сесій спілкування користувачів у платформі.

10

Рисунок Б.10 – Десятий слайд презентації

Клієнтський додаток

Для розробки клієнтської програми було обрано бібліотеку React. Для реалізації механізму спілкування між користувачами було застосовано бібліотеку mediasoup – client. Для обміну подіями у програмі використовується бібліотека Redux. Для збірки файлів вихідного коду, статичних файлів, ресурсів клієнтської програми використовується бібліотека Webpack.



11

Рисунок Б.11 – Одинадцятий слайд презентації

Серверний додаток

Серверний додаток обробляє запити користувача та не містить механізми спілкування та проведення дзвінків. Механізми спілкування та проведення дзвінків було винесено в окремий додаток для того, щоб не переривати сесії користувачів після внесення зміни до коду програми.

Для реалізації програми був обраний фреймворк NestJS з використанням NodeJS, який використовує мову програмування TypeScript.

12

Рисунок Б.12 – Дванадцятий слайд презентації

Додаток для дзвінку

Програма для організації дзвінків служить WebRTC сервером для обміну повідомленнями між учасниками. Для початкової установки з'єднання між клієнтом і сервером використовується WebSocket протокол, повідомлення якого надходить від клієнта до серверної програми і далі перенаправляються сервером до програми для організації дзвінків. Слідуючи цьому клієнт повинен надіслати повідомлення тільки в один додаток.

Для забезпечення проведення дзвінків та обміну даними через WebRTC, було обрано бібліотеку Mediasoup

13

Рисунок Б.13 – Тринадцятий слайд презентації

База даних системи

Оперуючи бізнес процесами що виникають у роботі програми, такі як створення кімнат, отримання кімнат, можна припустити що операції читання і запису в базу даних виникає частіше і це є пріоритетом. Тому проаналізувавши існуючі база даних і слідуючи тестам, будо обрано нереляційну базу даних MongoDB яка має кращі показники запису і читання по порівняно з реляційною базою даних MySQL.

14

Рисунок Б.14 – Чотирнадцятий слайд презентації

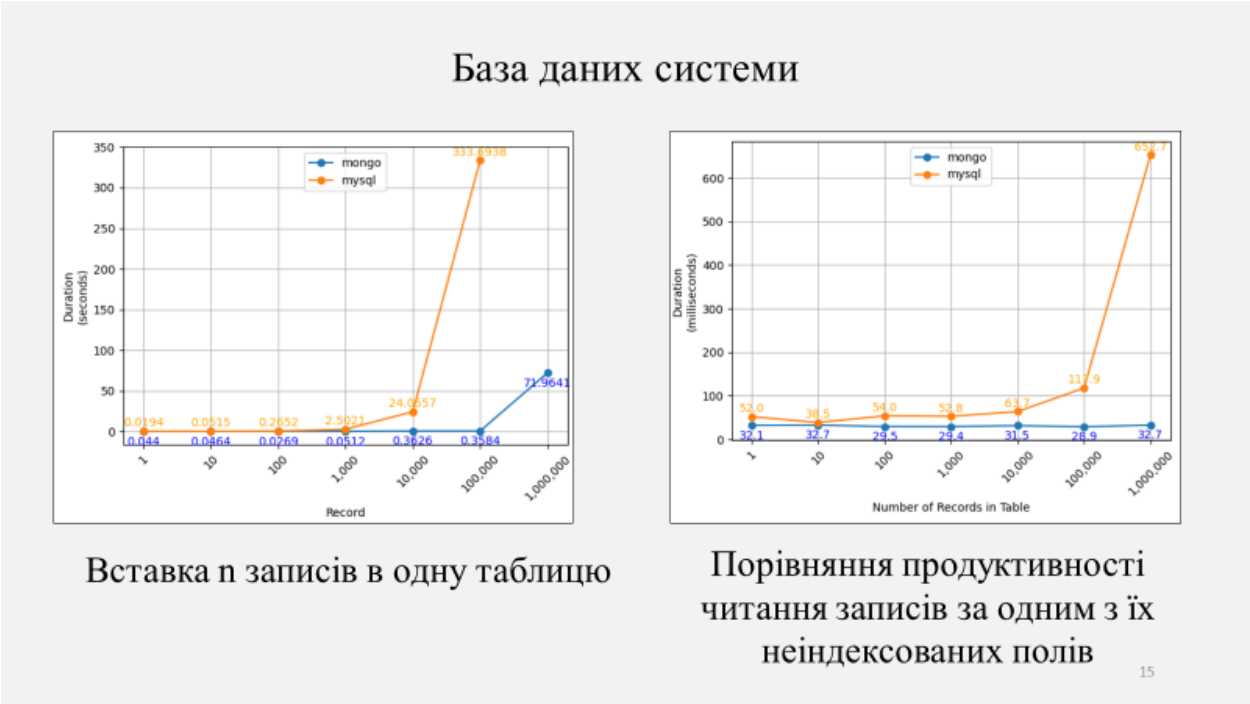


Рисунок Б.15 – П’ятнадцятий слайд презентації

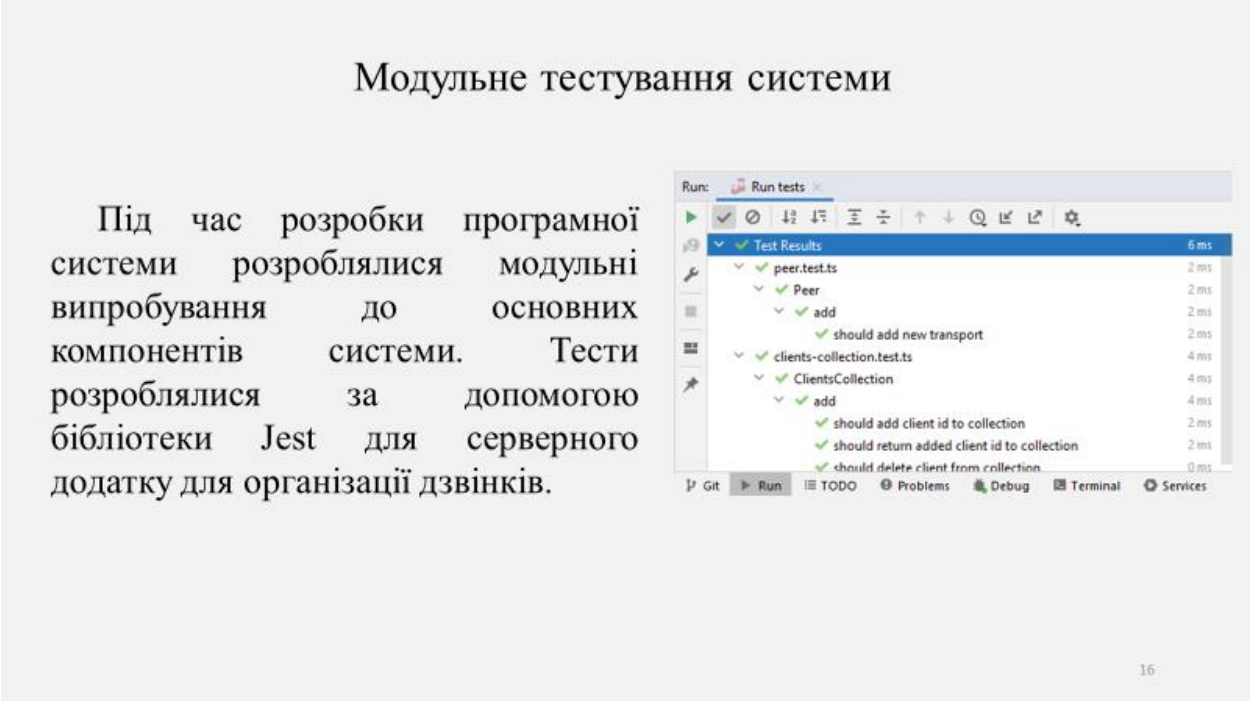


Рисунок Б.16 – Шістнадцятий слайд презентації

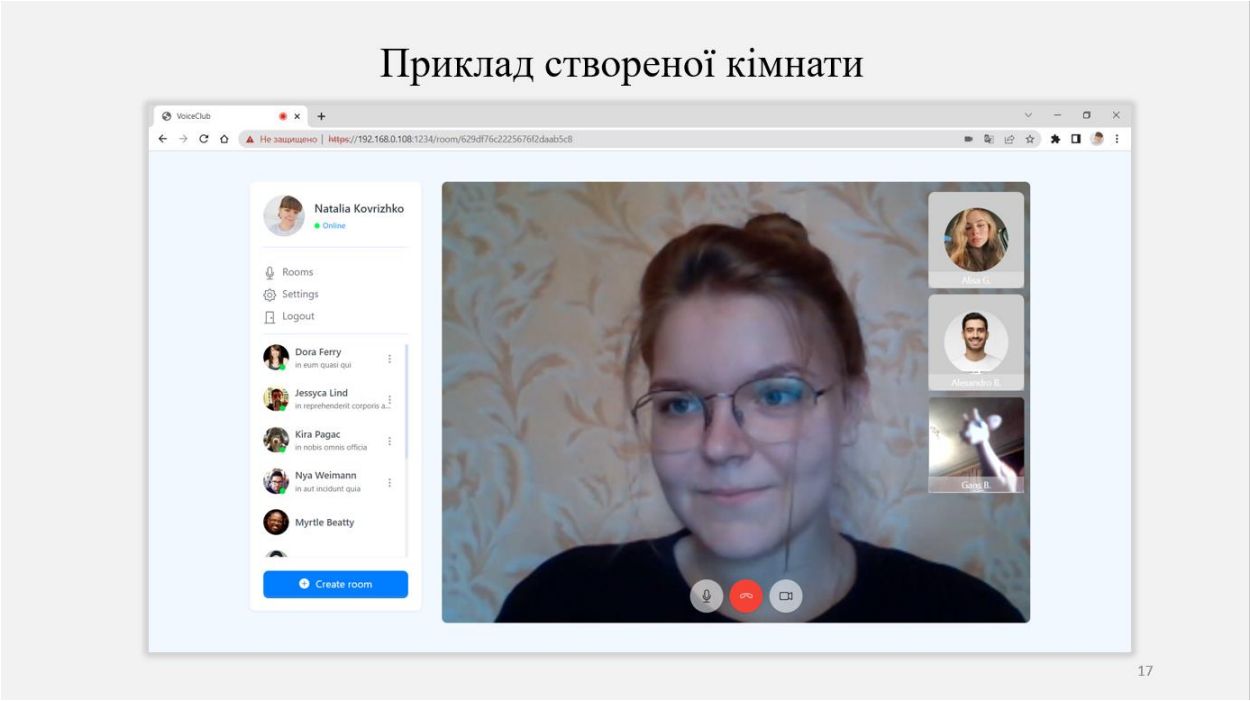


Рисунок Б.17 – Сімнадцятий слайд презентації

Рисунок Б.18 – Вісімнадцятий слайд презентації

Додаток В

Лістинг основних програмних модулів

В.1 Класи для роботи з запитами

```
import axios, { AxiosError, AxiosRequestHeaders, AxiosResponse, Method } from
"axios";
import { InvalidTokenException } from "./errors";
import toast from "react-hot-toast";
type RequestOptions = {
  url: string;
  method: Method;
  data?: object;
  params?: object;
  headers?: AxiosRequestHeaders;
type RequestOptionsWithoutMethod = Omit<RequestOptions, "method">;
export class HttpClient {
  private headers: AxiosRequestHeaders = {};
  useSettings(options: {
    headers?: RequestOptions["headers"]) {
    if (options.headers) {
      this.headers = options.headers;}}
async get<TResponse = any>(options: RequestOptionsWithoutMethod): Promise<{
data: TResponse }> {
    return this.sendRequest({
      ...options,
      method: "GET" });}
async post<TResponse = any>(options: RequestOptionsWithoutMethod): Promise<{
data: TResponse }> {
    return this.sendRequest({
      ...options,
      method: "POST"});}
async put<TResponse = any>(options: RequestOptionsWithoutMethod): Promise<{
data: TResponse }> {
    return this.sendRequest({
      ...options,
      method: "PUT" });}
  async sendRequest(options: RequestOptions): Promise<{ data: any }> {
    try {
      const response: AxiosResponse = await axios({
        ...options,
        headers: options.headers ? options.headers : this.headers});
      return response.data;
    } catch (e) {
      this.processError(e);}}
  processError(error: Error | unknown) {
    if (error instanceof Error) {
      toast.error(error?.message || 'Occurred not recognized error');}
    if (error instanceof AxiosError && error.response?.status === 401) {
      throw new InvalidTokenException();}
    throw error }}
import { apiHost } from "./urls";
import { HttpClient } from "../http-client";
const authUrl = apiHost + "auth/google";
export class AuthApiService {
  private static httpClient: HttpClient;
  static async init() {
```

```
    this.httpClient = new HttpClient();}
  static async authViaGoogle(accessToken: string): Promise<{ accessToken:
string }> {
    const response = await this.httpClient.post<{ accessToken: string }>({
      url: authUrl,
      data: {
        accessToken: accessToken}});
    return response.data;}}
import {HttpClient} from "../../http-client";
import {apiHost} from "../urls";
import {getAuthUserHttpHeaders} from "../auth-user-http-headers";
import { RoomModel } from "../../models/room.model";
import {faker} from '@faker-js/faker'
export class RoomsApiService {
    private static httpClient: HttpClient;
    static async init(){
        this.httpClient = new HttpClient();
        this.httpClient.useSettings({
            headers: await getAuthUserHttpHeaders()});}
    static async getRoomInfo(roomId: string): Promise<RoomModel>{
      await RoomsApiService.init();
      const response = await this.httpClient.get({
        url: apiHost + `room/${roomId}`,});
      return response.data}
    static async getAllRooms(query?: {
      title?: string,
      language?: string
      topics?: string[ }): Promise<{rooms: RoomModel[] }>{
        await RoomsApiService.init();
        const response = await this.httpClient.get({
           url: apiHost + "room/list",
           params: query});
        const rooms = response.data;
        rooms.rooms.forEach((room: any) => {
          const numberParticipants = +faker.datatype.number({min:1, max:
10});
          let participants = new Array(numberParticipants).fill(0);
          participants = participants.map(() => ({
            id: faker.random.numeric(100000),
            avatar: faker.image.avatar(),
            firstName: faker.name.firstName(),
            secondName: faker.name.lastName()}))
          // eslint-disable-next-line no-param-reassign
          room.participants = participants;
          // eslint-disable-next-line no-param-reassign
          room.totalParticipants = numberParticipants;
          console.log(numberParticipants);})
        return rooms}
    static async createRoom(options: {
        title: string,
        language: string,
        topic: string,
    }): Promise<any> {
        const response = await this.httpClient.post({
            url: apiHost + "room",
            data: options});
        return response.data;}
    static async joinToRoom(id: string): Promise<any> {
      const response = await this.httpClient.post({
        url: `${apiHost}room/${id}/join`});
      return response.data;}}
import axios, { AxiosError } from "axios";
import { apiHost } from "../urls";
```

```
import { AccessTokenStorage } from "../../access-token";
import { InvalidTokenException } from "../../errors";
import { HttpClient } from "../../http-client";
import { getAuthUserHttpHeaders } from "../auth-user-http-headers";
import { CurrentUserModel } from "../../models";
import {UserSettingsForUpdate} from "./types";
export class UserApiService {
  private static userHttpClient: HttpClient;
  static async init() {
    this.userHttpClient = new HttpClient();
    this.userHttpClient.useSettings({
      headers: await getAuthUserHttpHeaders()});}
  static async getMyProfile(): Promise<CurrentUserModel> {
    const response = await this.userHttpClient.get<CurrentUserModel>({
      url: apiHost + "user/me"});
    response.data = {
      ...response.data,
      email: 'test@gmail.com'}
    return response.data;}
  static async updateProfile(settings: UserSettingsForUpdate):
Promise<CurrentUserModel> {
    const response = await this.userHttpClient.put({
      url: apiHost + "user/settings",
      data: settings});
    return response.data;}}
```

B.2 Код основних сторінок сайту

```
import React, { useContext, useMemo } from "react";
import BaseLayout from "../../layouts/base-layout/base-layout";
import styles from "./call-room.scss";
import { Room } from "../../../libs/meeting-room/components/room/room";
import SocketContext from "../../../contexts/socketContext";
import useAppSelector from "../../../hooks/useAppSelector";
import { currentUserSelector } from "../../../stores/user/slices/current-
user.selectors";
import { useLocation, useParams } from "react-router";
export default function CallRoom() {
  const socketContext = useContext(SocketContext);
  const currentUser = useAppSelector(currentUserSelector);
  const { roomId } = useParams<{ roomId: string }>();
  const MemoMeetingRoom = useMemo(() => () => {
      if (!currentUser || !roomId) return <></>;
      return <Room roomId={roomId} socket={socketContext}
currentUser={currentUser} />;
    }, [currentUser]);
  return (
    <BaseLayout>
      <div className={styles.callRoom}>
        <MemoMeetingRoom />
      </div>
    </BaseLayout>);}
import React, { useEffect, useState } from "react";
import BaseLayout from "../../layouts/base-layout/base-layout";
import styles from "./home.scss";
import FilterBar from "../../common/filter-bar/filter-bar";
import RoomsContainer from "../../common/rooms-container/rooms-container";
import { RoomsApiService } from "../../../core/api-services/rooms";
import useAppSelector from "../../../hooks/useAppSelector";
import { getRoomsSelector } from
"../../../stores/rooms/slices/rooms.selector";
import appStore from "../../../stores/app-store";
```

```
import { roomsActions } from "../../../stores/rooms/slices/rooms.slice";
export default function Home() {
  const [modalActive, setModalActive] = useState(false);
  const rooms = useAppSelector(getRoomsSelector);
  useEffect(() => {
    RoomsApiService.getAllRooms().then((roomsResponse) => {
      appStore.dispatch(roomsActions.setRooms(roomsResponse.rooms));
    });
  }, []);
  return (
    <div className={styles.mainContent}>
      <BaseLayout>
        <div className={styles.content}>
          <FilterBar orientation="vertical" sizeLanguageBlock={285} />
          <div className={styles.roomsContainer}>
            <RoomsContainer rooms={rooms} />
          </div>
        </div>
      </BaseLayout>
    </div>)}
import React, { useEffect, useLayoutEffect, useRef } from "react";
import anime from 'animejs';
import styles from './styles.scss';
import AuthForm from "../../common/auth-form/auth-form";
import modelIcon from './assets/model.png';
import { People } from "react-bootstrap-icons";
import { AccessTokenStorage } from "../../../core/access-token";
const animationMoveFormToCenter = (formContainer: HTMLDivElement,
pageContainer: HTMLDivElement) => {
  anime({
    targets: `.${styles.loginForm}`,
    easing: 'easeInOutQuad',
    top: [-formContainer.clientHeight, window.innerHeight*0.31,
window.innerHeight*0.3],
    duration: 1500,
    complete: () => animationBlurBackground(pageContainer)});};
const animationBlurBackground = (backgroundContainer: HTMLDivElement) => {
  anime({
    targets: `.${styles.pageBackground}`,
    easing: 'easeInOutQuad',
      duration: 1500,
    update: (anim) => {
      backgroundContainer.style.filter = 'blur(' + 7 * anim.progress / 100 +
'px)'}})}
export default function LoginPage() {
    const loginFormRef = useRef<HTMLDivElement>(null);
    const backgroundContainerRef = useRef<HTMLDivElement>(null);
    useLayoutEffect(() => {
AccessTokenStorage.setAccessToken('eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzd
WIiOiI2MjVkY2RiYTkwYjlmYzM1YjJiYzVlMjIiLCJpYXQiOjE2NTQ1MDA4MDAsImV4cCI6MTY1NT
EwNTYwMH0.28QSGNUu1YXoqT5clWWYe5q9Aj8IFKU_3gKCYWmrEgI')
        if (loginFormRef.current && backgroundContainerRef.current) {
          animationMoveFormToCenter(loginFormRef.current,
backgroundContainerRef.current);}
    }, [loginFormRef])
    return (
        <div className={styles.page}>
          <div className={styles.pageBackground}
ref={backgroundContainerRef}>
          </div>
            <div className={styles.pageContainer}>
                <div className={styles.loginContainer}>
                    <div className={styles.loginForm} ref={loginFormRef}>
```

```
                    <People className={styles.logo}/>
                    <h1 className={styles.header}>Log In to Talk Space</h1>
                      <AuthForm/>
                  </div>
              </div>
          </div>
        </div>);}
import React, {useCallback, useEffect, useState} from "react";
import toast, { Toaster } from 'react-hot-toast';
import styles from "./settings-profile-page.scss";
import SettingsProfile, { SettingsProfileProps } from "../../common/settings-
profile/settings-profile";
import BaseLayout from "../../layouts/base-layout/base-layout";
import useAppSelector from "../../../hooks/useAppSelector";
import { currentUserSelector } from "../../../stores/user/slices/current-
user.selectors";
import {UserApiService} from "../../../core/api-services/user";
import appStore from "../../../stores/app-store";
import {currentUserActions} from "../../../stores/user/slices/current-
user.slice";
export default function SettingsProfilePage() {
  const currentUser = useAppSelector(currentUserSelector);
  const [userSettings, setUserSettings] =
useState<SettingsProfileProps["userSettings"]>({
    firstName: "",
    lastName: "",
    avatar: "",
    email: ""});
  useEffect(() => {
    if (currentUser) {
      setUserSettings({
        firstName: currentUser.firstName,
        lastName: currentUser.lastName,
        avatar: currentUser.avatar,
        email: currentUser.email});}
  }, [currentUser]);
  const onSubmitForm = useCallback((options: { firstName: string, lastName:
string }) => {
    UserApiService.updateProfile(options);
    appStore.dispatch(currentUserActions.setName(options))
    toast.success('Profile data updated successfully')
  }, []);
  return (
      <BaseLayout>
        <SettingsProfile userSettings={userSettings}
onSubmit={onSubmitForm}/>
      </BaseLayout>);}
```

В.3 Клас для забезпечення з’єднання з сервером

```
mport { Device } from 'mediasoup-client';
import { Transport } from 'mediasoup-client/lib/Transport';
import { Consumer } from 'mediasoup-client/lib/Consumer';
import { Producer } from 'mediasoup-client/lib/Producer';
import { RtpParameters } from 'mediasoup-client/lib/RtpParameters';
import { Store } from '@reduxjs/toolkit';
import SignalingGateway from './SignalingGateway';
import { SignalingGatewayListeningEvents } from './signaling-gateway.events';
import { participantsActions } from "./store/slices/participantsSlice";
const PC_PROPRIETARY_CONSTRAINTS =
  {optional: [{ googDscp: true }]};
const WEBCAM_SIMULCAST_ENCODINGS =
```

```
  [{scaleResolutionDownBy: 4,
      maxBitrate: 500000},
    {scaleResolutionDownBy: 2,
      maxBitrate: 1000000},
    {scaleResolutionDownBy: 1,
      maxBitrate: 5000000}];
export type RoomClientOptions = {
  roomId: string;
  producerTransport?: Transport;
  consumerTransport?: Transport;
  signalingGateway: SignalingGateway;}
export default class RoomClient {
  private readonly roomId: string;
  private producerTransport?: Transport;
  private consumerTransport?: Transport;
  private readonly consumers: Map<string, Consumer>;
  private readonly producers: Map<string, Producer>;
  private readonly signalingGateway: SignalingGateway;
  private mediasoupDevice!: Device;
  private webcams: Map<string, MediaDeviceInfo> = new Map();
  private currentWebcam: { device: MediaDeviceInfo | null, resolution: string
} = {
    device: null,
    resolution: 'hd'};
  private webcamProducer?: Producer;
  static store: Store;
  static currentUser: { id: string };
  constructor({
    roomId,
    signalingGateway
  }: RoomClientOptions) {
    this.roomId = roomId;
    this.consumers = new Map<string, Consumer>();
    this.producers = new Map<string, Producer>();
    this.signalingGateway = signalingGateway;}
  static init({ store, currentUser }: {
    store: Store
    currentUser: { id: string }
  }) {
    RoomClient.store = store;
    this.currentUser = currentUser}
  private static dispatch(action: any) {
    RoomClient.store.dispatch(action);}
  async processSignalingServerEvents(event: SignalingGatewayListeningEvents,
payload: any): Promise<void> {
    const listeningMapper: Record<SignalingGatewayListeningEvents, Function>
= {
      [SignalingGatewayListeningEvents.NEW_PRODUCERS]:
this.receivedNewProducers.bind(this),
      [SignalingGatewayListeningEvents.CONSUMER_CLOSED]:
this.consumerClosed.bind(this),
      [SignalingGatewayListeningEvents.NEW_CONSUMER]:
this.receivedNewConsumer.bind(this)};
    if (event && listeningMapper[event]) {
      await listeningMapper[event](payload);
    } else {
      console.warn('Event not registered for listening', {
        event,
        payload});}}
  async createRoom(): Promise<void> {}
  private async receivedNewProducers(producers: { producerId: string }[]):
Promise<void> {
    for (const { producerId } of producers) {
```

```
      // eslint-disable-next-line no-await-in-loop
      await this.consume(producerId);}}
  private async receivedNewConsumer(consumerData: {
    peerId: string,
    producerId: string,
    id: string,
    kind: 'audio' | 'video',
    rtpParameters: RtpParameters,
    type: string,
    producerPaused: boolean,}) {
    const {
      id,
      kind,
      rtpParameters,
      producerId
    } = consumerData;
    const consumer = await this.consumerTransport?.consume({
      id,
      producerId,
      kind,
      rtpParameters,
      appData: {}});
    if (consumer) {
      this.consumers.set(consumer.id, consumer);
      consumer.on('transportclose', () => {
        this.consumers.delete(consumer.id);});
      RoomClient.dispatch(participantsActions.addParticipantProducer({
        userId: consumerData.peerId,
        producer: {
          id: producerId,
          paused: consumer.paused,
          track: consumer.track,
          rtpParameters: consumer.rtpParameters,
          codec: consumer.rtpParameters.codecs[0].mimeType.split('/')[1] }
}));}}
  private async consumerClosed(consumerData: { consumerId: string }):
Promise<void> {
    const consumer = this.consumers.get(consumerData.consumerId);
    if (!consumer) {
      return;}
    consumer.close();
    this.consumers.delete(consumerData.consumerId);
RoomClient.dispatch(participantsActions.removeParticipantProducer(consumer.pr
oducerId));}
  private async consume(producerId: string): Promise<void> {
    this.getConsumeStream(producerId);}
  private async getConsumeStream(producerId: string): Promise<any> {
    const { rtpCapabilities } = this.mediasoupDevice;
    await this.signalingGateway.consume({
      rtpCapabilities,
      consumerTransportId: this.consumerTransport?.id,
      producerId });}
  async join() {
    console.log('Attempt to join');
    // await this.createRoom();
    this.mediasoupDevice = new Device();
    const routerRtpCapabilities = await
this.signalingGateway.getRouterRtpCapabilities();
    // @ts-ignore
    await this.mediasoupDevice.load({ routerRtpCapabilities });
    await this.signalingGateway.joinToRoom({
      roomId: this.roomId,
      rtpCapabilities: this.mediasoupDevice.rtpCapabilities,
```

```
      sctpCapabilities: this.mediasoupDevice.sctpCapabilitie});
    console.log('Mediasoup device loaded');
    await this.createProducerTransport();
    await this.createConsumerTransport();
    await this.enableWebcam();
    await this.signalingGateway.getProducers();}
  private async createProducerTransport() {
    const transportInfo = await this.signalingGateway.createWebRtcTransport({
      forceTcp: false,
      producing: true,
      consuming: false,
      rtpCapabilities: this.mediasoupDevice.rtpCapabilities});
    this.producerTransport = this.mediasoupDevice.createSendTransport({
      id: transportInfo.id,
      iceParameters: transportInfo.iceParameters,
      iceCandidates: transportInfo.iceCandidates,
      dtlsParameters: transportInfo.dtlsParameters,
      sctpParameters: transportInfo.sctpParameters,
      iceServers: [],
      proprietaryConstraints: PC_PROPRIETARY_CONSTRAINTS});
    /**
     * Emitted when the transport is about to establish the ICE+DTLS
connection
     * and needs to exchange information with the associated server side
transport.
     */
    this.producerTransport.on('connect', async ({ dtlsParameters }, callback,
errorCallback) => {
      try {
        const connectWebRtcTransportRes = await
this.signalingGateway.connectWebRtcTransport({
          transportId: this.producerTransport?.id,
          dtlsParameters});
        callback(connectWebRtcTransportRes);
      } catch (e) {
        errorCallback(e);}});
    /**
     * Emitted when the transport needs to transmit information
     * about a new producer to the associated server side transport.
     */
    this.producerTransport.on('produce', async ({
      kind,
      rtpParameters
    }, callback, errorCallback) => {
      try {
        const produceRes = await this.signalingGateway.transportProduce({
          transportId: this.producerTransport?.id,
          kind,
          rtpParameters
        });
        callback({ id: produceRes.producerId });
      } catch (e) {
        errorCallback(e);}});
    /**
     * Emitted when the transport needs to transmit information
     * about a new data producer to the associated server side transport.
     */
    this.producerTransport.on('producedata', async (params, callback,
errorCallback) => {
      try {
        const transportProduceData = await
this.signalingGateway.transportProduceData({
          transportId: this.producerTransport?.id,
```

```
          sctpStreamParameters: params.sctpStreamParameters,
          label: params.label,
          protocol: params.protocol});
        callback({ id: transportProduceData });
      } catch (e) {
        errorCallback(e); } });
    this.producerTransport.on('connectionstatechange', (connectionState:
RTCPeerConnectionState) => {
      console.log('Producer transport\'s connection state changed: ',
connectionState); })}
  private async createConsumerTransport() {
    const transportInfo = await this.signalingGateway.createWebRtcTransport({
      forceTcp: false,
      producing: false,
      consuming: true,
      rtpCapabilities: this.mediasoupDevice.rtpCapabilities });
    this.consumerTransport = this.mediasoupDevice.createRecvTransport({
      id: transportInfo.id,
      iceParameters: transportInfo.iceParameters,
      iceCandidates: transportInfo.iceCandidates,
      dtlsParameters: transportInfo.dtlsParameters,
      sctpParameters: transportInfo.sctpParameters,
      iceServers: [],
      proprietaryConstraints: PC_PROPRIETARY_CONSTRAINTS });
    this.consumerTransport.on('connect', async ({ dtlsParameters }, callback,
errorCallback) => {
      try {
        const connectInfo = await
this.signalingGateway.connectWebRtcTransport({
          transportId: this.consumerTransport?.id,
          dtlsParameters});
        callback(connectInfo);
      } catch (e) {
        errorCallback(e);}});
    this.consumerTransport.on('connectionstatechange', (connectionState:
RTCPeerConnectionState) => {
      console.log('Consumer transport\'s connection state changed: ',
connectionState);})}
  async enableWebcam() {
    if (this.webcamProducer) {
      return;}
    if (!this.mediasoupDevice.canProduce('video')) {
      console.error('Cannot produce video');
      return;}
    await this.updateAvailableWebcams();
    if (!this.currentWebcam.device) {
      console.log('No webcam devices');
      return; }
    const stream = await navigator.mediaDevices.getUserMedia({
      video: {
        deviceId: {
          ideal: this.currentWebcam.device.deviceId,}}});
    const track = stream.getVideoTracks()[0];
    const codecOptions = {
      videoGoogleStartBitrate: 1000
    };
    this.webcamProducer = await this.producerTransport?.produce({
      track,
      encodings: WEBCAM_SIMULCAST_ENCODINGS,
      codecOptions,});
    if (!this.webcamProducer) {
      console.error('Producer for webcam not created');
      return;}
```

```
    await this.webcamProducer.replaceTrack({ track });
    this.webcamProducer?.on('transportclose', () => {
      this.webcamProducer = undefined;});}
  async updateAvailableWebcams(): Promise<MediaDeviceInfo | null> {
    this.webcams.clear();
    const devices = await navigator.mediaDevices.enumerateDevices();
    devices.forEach((device) => {
      if (device.kind === 'videoinput') {
        this.webcams.set(device.deviceId, device);}});
    const currentWebcamId = this.currentWebcam.device &&
this.currentWebcam.device.deviceId;
    if (devices.length === 0) {
      this.currentWebcam.device = null;
    } else if (!currentWebcamId || !this.webcams.has(currentWebcamId)) {
      this.currentWebcam.device = this.webcams.entries()
        .next().value;}
    return this.currentWebcam.device;}
  async disableWebcam(): Promise<void> {
    if (!this.webcamProducer) {
      return;}
    this.webcamProducer.close();
RoomClient.dispatch(participantsActions.removeParticipantProducer(this.webcam
Producer.id));
    this.signalingGateway.closeProducer({
      producerId: this.webcamProducer.id});
    this.webcamProducer = undefined;}
  async endCall(): Promise<void> {
    if (this.webcamProducer) {
      this.webcamProducer.close();
      this.signalingGateway.closeProducer({
        producerId: this.webcamProducer.id});}
    if (this.producerTransport) {
      this.producerTransport.close();}
    if (this.consumerTransport) {
      this.consumerTransport?.close();}
    RoomClient.dispatch(participantsActions.removeAllParticipants());}}
```

В.4 Класи для роботи з редаксом

```
import {CaseReducer, createSlice, PayloadAction} from "@reduxjs/toolkit";
export type RoomItem = {
    id: string;
    title: string,
    language: string,
    topic: string,
    isActive: boolean,
    participants: Array<{
        id: string;
        avatar: string;
        firstName: string;
        lastName: string;}>;
    totalParticipants: number;
    author: {
        id: string;
        avatarUrl?: string;
        firstName: string;
        lastName: string;};
    limitDisplayParticipants?: number;}
export type Rooms = {
    rooms: RoomItem[]}
const setRooms: CaseReducer<Rooms,
    PayloadAction<RoomItem[]>> = (state, payloadAction) => {
```

```
    return {
        rooms: [
            ...payloadAction.payload]}}
const addRoom: CaseReducer<Rooms, PayloadAction<RoomItem>> = (state,
payloadAction) => {
    const existsRoom = state.rooms;
    return {
        rooms: [
            payloadAction.payload,
            ...existsRoom]}}
const reducers = { setRooms, addRoom }
export const roomsSlice = createSlice<
    Rooms,
    typeof reducers,
    'rooms'>({
    name: "rooms",
    reducers,
    initialState: {
        rooms: []}});
export const roomsActions = roomsSlice.actions
export default roomsSlice.reducer
import {AppStore} from "../../app-store";
import {Rooms} from "./rooms.slice";
export const getRoomsSelector = (state: AppStore):Rooms['rooms'] => {
    return state.rooms.rooms;}
import { AppStore } from "../../app-store";
import { CurrentUserState } from "./current-user.slice";
export const currentUserSelector = (state: AppStore):CurrentUserState['user']
=> { return state.currentUser.user;}
import { CaseReducer, createSlice, PayloadAction } from "@reduxjs/toolkit";
import { CurrentUserModel } from "../../../core/models";
export type CurrentUserState = {
  user?: CurrentUserModel};
const setCurrentUser: CaseReducer<CurrentUserState,
  PayloadAction<CurrentUserModel>> = (state, payloadAction) => {
  return {
    user: {
      ...payloadAction.payload
    }};};
 // @ts-ignore
const setName: CaseReducer<CurrentUserState,
     PayloadAction<{firstName: string, lastName: string}>> = (state,
payloadAction) => {
   return {
     user: {
       ...state.user ? state.user : {},
       ...payloadAction.payload
     }}}
const reducers = { setCurrentUser, setName };
export const currentUserSlice = createSlice<
  CurrentUserState,
  typeof reducers,
  'currentUser'
  >({
  name: "currentUser",
  initialState: {},
  reducers,
});
export const currentUserActions = currentUserSlice.actions;
export default currentUserSlice.reducer;
import { configureStore } from "@reduxjs/toolkit";
import meetingRoomStore from "../libs/meeting-room/store/meeting-room.store";
import currentUserReducer from './user/slices/current-user.slice';
```

```
import roomsReducer from './rooms/slices/rooms.slice';
import modalsReducer from './modals/modals.slice';
const appStore = configureStore({
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({
      serializableCheck: false,}),
  reducer: {
    ...meetingRoomStore,
    currentUser: currentUserReducer,
    rooms: roomsReducer,
    modals: modalsReducer,}});
export type AppStore = ReturnType<typeof appStore.getState>;
export type AppDispatch = typeof appStore.dispatch;
export default appStore;
import { CaseReducer, createSlice, PayloadAction } from "@reduxjs/toolkit";
export type ModalState = {
  modals: Record<string, boolean>;};
const showModal: CaseReducer<ModalState,
  PayloadAction<string>> = (state, payloadAction) => {
  return {
    modals: {
      ...state.modals,
      [payloadAction.payload]: true}};};
const hideModal: CaseReducer<ModalState,
  PayloadAction<string>> = (state, payloadAction) => {
  return {
    modals: {
      ...state.modals,
      [payloadAction.payload]: false}};};
const reducers = { showModal, hideModal };
export const modalsSlice = createSlice<
  ModalState,
  typeof reducers,
  'modals'>({
  name: "modals",
  initialState: {
    modals: {}},
  reducers,});
export const modalsActions = modalsSlice.actions;
export default modalsSlice.reducer;
```

В.5 Класи для роботи з БД

```
import BaseOrmEntity from '@db/base-orm-entity';
import { ObjectId } from 'mongodb';
export class RoomOrmEntity extends BaseOrmEntity {
  userId: ObjectId;
  title: string;
  language?: string;
  topic?: string;
  isActive: boolean;
  participants: ObjectId[];}
import { Repository } from '@db/decorators';
import { BaseRepository, OmitInsertFields } from '../../base-repository';
import { RoomOrmEntity } from './room.orm-entity';
import { ObjectId } from 'mongodb';
@Repository('rooms')
export class RoomRepository extends BaseRepository<RoomOrmEntity> {
  async insertRoom(doc: { userId: string } & Omit<RoomOrmEntity,
OmitInsertFields | 'userId'>): Promise<RoomOrmEntity> {
    return super.insertOne({
```

```
      ...doc,
      userId: new ObjectId(doc.userId),});}
  async listActiveRooms(): Promise<RoomOrmEntity[]> {
    return super.findMany({
      isActive: true})}
  async addParticipant(roomId: string, userId: string): Promise<void> {
    await this.updateOne({
      _id: new ObjectId(roomId)}, {
      $addToSet: {
        participants: new ObjectId(userId)}});}
  async removeParticipant(roomId: string, userId: string): Promise<void> {
    await this.updateOne({
      _id: new ObjectId(roomId)}, {
      $pull: {
        participants: new ObjectId(userId) } });}}
import { UserProvider, UserRole } from '../../../modules/user/entities';
import BaseOrmEntity from '@db/base-orm-entity';
import { ObjectId } from 'mongodb';
export class UserOrmEntity extends BaseOrmEntity {
  firstName: string;
  lastName: string;
  providers: UserProvider[];
  role: UserRole;
  avatar?: string;
  email?: string;
  activeRoomId?: ObjectId;}
import { BaseRepository } from '../../base-repository';
import { UserOrmEntity } from '@db/entities';
import { AuthProvider, UserProvider } from '../../../modules/user/entities';
import { Repository } from '@db/decorators';
import { ObjectId, UpdateResult } from 'mongodb';
@Repository('users')
export class UserRepository extends BaseRepository<UserOrmEntity> {
  async findUserWithProvider(
    providerName: AuthProvider,
    providerId: string,
  ): Promise<UserOrmEntity | null> {
    const searchProvider: UserProvider = {
      id: providerId,
      name: providerName,};
    return this.findOne({
      providers: {
        $elemMatch: searchProvider,},});}
  async updateUser(userId: string, setFields: Partial<UserOrmEntity>):
Promise<UpdateResult> {
    return this.updateOne({
      _id: new ObjectId(userId)}, {
      $set: setFields})}
  async findByIds(userIds: string[]): Promise<UserOrmEntity[]> {
    const usersIdsForMongo = userIds.map((id) => new ObjectId(id));
    return this.findMany({
      _id: {
        $in: usersIdsForMongo }})}
  setActiveRoomForUser(userId: string, roomId: string):
Promise<UpdateResult>{
    return this.updateOne({
      _id: new ObjectId(userId),}, {
      $set: {
        activeRoomId: new ObjectId(roomId)}})}
  clearActiveRoom(userId: string) {
    return this.updateOne({
      _id: new ObjectId(userId),}, {
```

```
      $unset: {
        activeRoomId: ''}})}}
import { ObjectId } from 'mongodb';
export default class BaseOrmEntity {
  readonly _id: ObjectId;
  readonly id: string;
  readonly createdAt: Date;
  readonly updatedAt: Date;}
import { Injectable } from '@nestjs/common';
import { MongoConnector } from './mongo-connector';
import {
  Collection,
  Db,
  Filter,
  FindOptions,
  ObjectId,
  OptionalUnlessRequiredId,
  UpdateFilter,
  UpdateResult, WithId,
} from 'mongodb';
export type OmitInsertFields = 'id' | '_id' | 'createdAt' | 'updatedAt';
export type CollectionOptions = {
  collectionName: string;};
export const COLLECTION_SETTINGS_KEY = 'collectionSettings';
@Injectable()
export class BaseRepository<TEntity> {
  private readonly db: Db;
  private collectionOptions: CollectionOptions;
  constructor(private readonly connector: MongoConnector) {
    this.setupCollectionSettings();
    this.db = connector.db;}
  private setupCollectionSettings() {
    this.collectionOptions = <CollectionOptions>Reflect.get(this,
COLLECTION_SETTINGS_KEY);
    if (!this.collectionOptions) {
      throw new Error('Collection settings not found');  }}
  private get collectionName(): string {
    return this.collectionOptions.collectionName;}
  private get collection(): Collection<TEntity> {
    return this.db.collection<TEntity>(this.collectionName);}
  private prepareRawMongoEntity(entity: TEntity): TEntity {
    // @ts-ignore
    entity.id = entity._id.toHexString();
    return entity;}
  async findOne(filter: Filter<TEntity>): Promise<TEntity | null> {
    const entity = await this.collection.findOne(filter);
    if (entity) {
      // @ts-ignore
      return this.prepareRawMongoEntity(entity);}
    return null;}
  async findMany(filter: Filter<TEntity>, findOptions?:
FindOptions<TEntity>): Promise<WithId<TEntity>[]> {
    let entities = await this.collection.find(filter, findOptions).toArray();
    // @ts-ignore
    entities = entities.map((entity) => this.prepareRawMongoEntity(entity));
    return entities;}
  async insertOne(doc: Omit<TEntity, 'id' | '_id' | 'createdAt' |
'updatedAt'>): Promise<TEntity> {
    const insertDoc = {
      ...doc,
      createdAt: new Date(),
      updatedAt: new Date(),
    } as unknown as OptionalUnlessRequiredId<TEntity>;
```

```
    const insertRes = await this.collection.insertOne(insertDoc);
    // @ts-ignore
    return this.findOne({
      _id: insertRes.insertedId,});}
  findById(entityId: string): Promise<TEntity> {
    // @ts-ignore
    return this.findOne({
      _id: new ObjectId(entityId),});}
  updateOne(filter: Filter<TEntity>, update: UpdateFilter<TEntity> |
Partial<TEntity>): Promise<UpdateResult> {
    return this.collection.updateOne(filter, update);  }}
```

В.6 Класи для організації кімнат дзвінку

```
export interface SignalingGateway {
  emit(id: string, eventName: string, args?: string | number | object |
boolean): boolean;}
export enum MediasoupEmittingEventsEnum {
  NEW_PRODUCERS = 'newProducers',
  NEW_CONSUMER = 'newConsumer',
  CONSUMER_CLOSED = 'consumerClosed',}
import { Logger } from '../../../common';
import { Client } from '../libs/interfaces';
import Room from '../libs/room';
import {
  CallServerListeningEventsEnum,
  GuestUserCallServerListeningEvents,
  CallServerSendEvents,
} from 'call-app-utils';
import WorkersCollection from '../workers-collection';
import { SignalingGateway } from '../interfaces';
import { Peer } from '../libs/peer';
import { MediasoupEmittingEventsEnum } from './mediasoup-emitting-
events.enum';
import ClientsCollection from '../clients-collection';
import { MediasoupConfigType } from '../../../configs';
import { DtlsParameters } from 'mediasoup/node/lib/WebRtcTransport';
import { MediaKind, RtpParameters } from 'mediasoup/node/lib/RtpParameters';
type ListeningEventFunction = (client: Client, payload: unknown) => unknown;
export class MediasoupEventsHandler {
  private readonly guestEventsMapper: Record<
    GuestUserCallServerListeningEvents,
    ListeningEventFunction
  > = {
    [GuestUserCallServerListeningEvents.ConnectClient]:
this.connectClient.bind(this),};
  private readonly authUserEventsMapper: Record<
    CallServerListeningEventsEnum,
    ListeningEventFunction
  > = {
    [CallServerListeningEventsEnum.CREATE_ROOM]:
this.createRoomEvent.bind(this),
    [CallServerListeningEventsEnum.JOIN]: this.joinToRoomEvent.bind(this),
    [CallServerListeningEventsEnum.GET_PRODUCERS]:
this.getProducers.bind(this),
    [CallServerListeningEventsEnum.GET_ROUTER_RTP_CAPABILITIES]:
      this.getRouterRtpCapabilitiesEvent.bind(this),
    [CallServerListeningEventsEnum.CREATE_WEBRTC_TRANSPORT]:
      this.createWebRtcTransportEvent.bind(this),
    [CallServerListeningEventsEnum.CONNECT_WEBRTC_TRANSPORT]:
      this.connectWebRtcTransport.bind(this),
    [CallServerListeningEventsEnum.CONNECT_TRANSPORT]:
```

```
this.connectTransport.bind(this),
    [CallServerListeningEventsEnum.PRODUCE]: this.produce.bind(this),
    [CallServerListeningEventsEnum.CONSUME]: this.createConsumer.bind(this),
    [CallServerListeningEventsEnum.RESUME]: this.resume.bind(this),
    [CallServerListeningEventsEnum.GET_MY_ROOM_INFO]:
this.getMyRoomInfo.bind(this),
    [CallServerListeningEventsEnum.EXIT]: this.exitRoom.bind(this),
    [CallServerListeningEventsEnum.PRODUCER_CLOSED]:
this.producerClosed.bind(this),
    [CallServerListeningEventsEnum.EXIT_ROOM]: this.exitRoom.bind(this),
    [CallServerListeningEventsEnum.CLOSE_PRODUCER]:
this.closeProducer.bind(this),
  };
  constructor(
    private readonly rooms: Map<string, Room>,
    private readonly clientsCollection: ClientsCollection,
    private readonly logger: Logger,
    private readonly workersCollection: WorkersCollection,
    private readonly signalingGateway: SignalingGateway,
    private readonly mediasoupConfig: MediasoupConfigType,) {}
  async processClientEvent(
    clientId: string,
    eventName: CallServerListeningEventsEnum,
    payload?: unknown,
  ): Promise<unknown> {
    if (!eventName) {
      this.logger.warn('Processing event is empty', {
        eventName,});
      return;}
    try {
      if (this.guestEventsMapper[eventName]) {
        return this.guestEventsMapper[eventName](clientId, payload);}
      if (!this.authUserEventsMapper[eventName]) {
        this.logger.warn('Specified event for processing not found', {
          clientId,
          eventName,
          payload,});}
      if (!this.clientsCollection.hasClientById(clientId)) {
        this.logger.info('Client has not access to send the event', {
          clientId,
          eventName,});
        return;}
      const client = this.clientsCollection.getById(clientId);
      return this.authUserEventsMapper[eventName](client, payload);
    } catch (e) {
      this.logger.error(
        'Failed to processing an event',{
          clientId,
          eventName,
          payload,},e,);}}
  async connectClient(clientId: string, payload?: { roomId: string }) {
    if (this.clientsCollection.hasClientById(clientId)) {
      this.logger.warn('Client already exists in system by specified ID', {
        clientId,});
    } else {
      this.clientsCollection.add(clientId);}
    this.clientsCollection.getById(clientId).roomId = payload.roomId;
    this.logger.info('Client connected', {
      clientId,});
    return true;  }
  private async createRoomEvent(client: Client, payload?: { roomId: string
}): Promise<string> {
    const { roomId } = payload;
```

```
    if (this.rooms.has(roomId)) {
      return roomId;
    }
    const worker = this.workersCollection.getNextWorker();
    const room = new Room(roomId, worker, this.signalingGateway, this.logger,
this.mediasoupConfig);
    this.rooms.set(roomId, room);
    this.logger.info('Created new room', {
      roomId,
    });
    return roomId;
  }
  private async joinToRoomEvent(client: Client, payload?: { roomId: string
}): Promise<boolean> {
    const { roomId } = payload;
    const room = this.rooms.get(roomId);
    if (!room) {
      this.logger.warn('Room for joining does not exists', {
        clientId: client.id,
        payload,});
      return false;}
    const peer = new Peer(client.id, `#client:${client.id}`, this.logger,
this.signalingGateway);
    room.addPeer(peer);
    client.roomId = room.id;
    client.peer = peer;
    return true;}
  private async getProducers(client: Client, payload?: unknown):
Promise<boolean> {
    const room = this.rooms.get(client.roomId);
    if (!room) {
      this.logger.warn('Room does not exists for getting producers', {
        clientId: client.id,
        payload,});
      return false;}
    const producersIds = room.getProducersForPeer();
    this.signalingGateway.emit(client.id, CallServerSendEvents.NEW_PRODUCERS,
producersIds);
    return true;}
  private async getRouterRtpCapabilitiesEvent(client: Client, payload?:
unknown): Promise<unknown> {
    const room = this.rooms.get(client.roomId);
    if (!room) {
      this.logger.warn('Room does not exists for get RTP capabilities', {
        clientId: client.id,
        payload,});
      return false;}
    return room.getRtpCapabilities();}
  private async createWebRtcTransportEvent(client: Client, payload?:
unknown): Promise<unknown> {
    const room = this.rooms.get(client.roomId);
    if (!room) {
      this.logger.warn('Room does not exists for
createWebRtcTransportEvent',{
        clientId: client.id,
        payload,});
      return false;}
    const { params } = await room.createWebRtcTransport(client.id);
    return params;}
  private async connectWebRtcTransport(
    client: Client,
    payload?: { transportId: string; dtlsParameters: DtlsParameters },
  ): Promise<unknown> {
```

```
    if (!client.peer) {
      this.logger.error('Client has not peer', {
        clientId: client.id,
      });
      return;}
    const { transportId, dtlsParameters } = payload;
    await client.peer.connectTransport(transportId, dtlsParameters);}
  private async connectTransport(
    client: Client,
    payload?: { transportId: string; dtlsParameters: DtlsParameters },
  ): Promise<unknown> {
    const room = this.rooms.get(client.roomId);
    if (!room) {
      this.logger.warn('Room does not exists for
createWebRtcTransportEvent',{
        clientId: client.id,
        payload,
      });
      return false;}
    const { transportId, dtlsParameters } = payload;
    await room.connectPeerTransport(client.id, transportId, dtlsParameters);
    return 'success';}
  private async produce(
    client: Client,
    payload?: {
      transportId: string;
      rtpParameters: RtpParameters;
      kind: MediaKind;},
  ): Promise<{ producerId: string }> {
    const room = this.rooms.get(client.roomId);
    const { kind, rtpParameters, transportId } = payload;
    const producerId = await room.produce(client.id, transportId,
rtpParameters, kind);
    return {
      producerId,};}
  private async closeProducer(client: Client, payload: { producerId: string
}) {
    const room = this.rooms.get(client.roomId);
    room.closeProducer(client.id, payload.producerId);
  }
  private async createConsumer(client: Client, payload?: any):
Promise<boolean> {
    const room = this.rooms.get(client.roomId);
    const { consumerTransportId, producerId, rtpCapabilities } = payload;
    return room.createConsumer(client.id, consumerTransportId, producerId,
rtpCapabilities);}
  private async resume(client: Client, payload?: unknown): Promise<void> {
    this.logger.warn('No implemented resume() method');}
  private async getMyRoomInfo(client: Client, payload?: unknown):
Promise<unknown> {
    const room = this.rooms.get(client.roomId);
    return room.toJson();
  }
  private async producerClosed(client: Client, payload?: { producerId: string
}): Promise<void> {
    const room = this.rooms.get(client.roomId);
    const { producerId } = payload;
    room.closeProducer(client.id, producerId);
  }
  private async exitRoom(client: Client) {
    const room = this.rooms.get(client.roomId);
    if (!room) {
      this.logger.warn('Not currently in a room');
```

```
      return;}
    await room.removePeer(client.id);
    this.clientsCollection.delete(client.id);
    if (!room.getPeers().size) {
      this.rooms.delete(room.id);}}}
import { Logger } from '../../../common';
import { DtlsParameters, WebRtcTransport } from
'mediasoup/node/lib/WebRtcTransport';
import { Consumer } from 'mediasoup/node/lib/Consumer';
import { Producer } from 'mediasoup/node/lib/Producer';
import { MediasoupEmittingEventsEnum } from '../mediasoup-events-handler';
import { SignalingGateway } from '../interfaces';
import { MediaKind, RtpParameters } from 'mediasoup/node/lib/RtpParameters';
export class Peer {
  private readonly transports: Map<string, WebRtcTransport>;
  private readonly consumers: Map<string, Consumer>;
  private readonly _producers: Map<string, Producer>;
  constructor(
    public readonly id: string,
    public readonly name: string,
    private readonly logger: Logger,
    private readonly signalingGateway: SignalingGateway, ) {
    this.transports = new Map<string, WebRtcTransport>();
    this.consumers = new Map<any, any>();
    this._producers = new Map<any, any>();}
  get producers(): Map<string, Producer> {
    return this._producers; }
  addTransport(transport: WebRtcTransport) {
    this.transports.set(transport.id, transport);}
  async connectTransport(transportId: string, dtlsParameters:
DtlsParameters): Promise<void> {
    const transport = this.transports.get(transportId);
    if (!transport) {
      this.logger.warn('Transport for connecting not found', {
        transportId,
      });
      return;}
    await transport.connect({
      dtlsParameters: dtlsParameters,
    });}
  async createProducer(producerTransportId: string, rtpParameters:
RtpParameters, kind: MediaKind) {
    const producerTransport = this.transports.get(producerTransportId);
    if (!producerTransport) {
      this.logger.error('Producer transport not found');
      return;}
    const producer = await producerTransport.produce({
      kind,
      rtpParameters,
    });
    this._producers.set(producer.id, producer);
    producer.on('transportclose', () => {
      this.logger.info('Producer transport close', {
        name: `${this.name}`,
        consumer_id: `${producer.id}`,
      });
      producer.close();
      this._producers.delete(producer.id);
    });
    return producer;}
  async createConsumer(
    consumerTransportId: string,
    producerId: string,
```

```
    rtpCapabilities: any,
  ): Promise<Consumer> {
    const consumerTransport = this.transports.get(consumerTransportId);
    if (!consumerTransport) {
      this.logger.error('Consumer transport not found', {
        consumerTransportId,
      });
      return;}
    let consumer: Consumer;
    try {
      consumer = await consumerTransport.consume({
        producerId,
        rtpCapabilities,
        paused: false,
      });
    } catch (error) {
      console.error('Consume failed', error);
      return;
    }
    this.consumers.set(consumer.id, consumer);
    consumer.on('transportclose', () => {
      this.removeConsumer(consumer.id);
    });
    consumer.on('producerclose', () => {
      console.log('Consumer closed due to producerclose event', {
        name: `${this.name}`,
        consumer_id: `${consumer.id}`,
      });
      this.removeConsumer(consumer.id);
      // tell client consumer is dead
      this.signalingGateway.emit(this.id,
MediasoupEmittingEventsEnum.CONSUMER_CLOSED, {
        consumerId: consumer.id,
      });});
    return consumer;
  }
  closeProducer(producerId: string) {
    try {
      const producer = this._producers.get(producerId);
      if (!producer) {
        this.logger.warn('Producer not found for closing', {
          producerId,
        });
      } else {
        producer.close();
      }
    } catch (e) {
      this.logger.error('Failed to close transport', e);
    }

    this._producers.delete(producerId);
  }
  getProducer(producerId: string) {
    return this._producers.get(producerId);
  }

  close() {
    this.transports.forEach((transport) => transport.close());
  }

  removeConsumer(consumer_id: string) {
    this.consumers.delete(consumer_id);
```

```
  }
}
import { Worker } from 'mediasoup/node/lib/Worker';
import { Router } from 'mediasoup/node/lib/Router';
import { Peer } from './peer';
import { MediaKind, RtpCapabilities, RtpParameters } from
'mediasoup/node/lib/RtpParameters';
import {
  DtlsParameters,
  IceCandidate,
  IceParameters,
  WebRtcTransport,
} from 'mediasoup/node/lib/WebRtcTransport';
import { WebrtcTransportFactory } from './webrtc-transport-factory';
import { SignalingGateway } from '../interfaces';
import { Producer } from 'mediasoup/node/lib/Producer';
import { Logger } from '../../../common';
import { MediasoupConfigType } from '../../../configs';
import { MediasoupEmittingEventsEnum } from '../mediasoup-events-handler';
export default class Room {
  private router!: Router;
  private peers: Map<string, Peer> = new Map();
  constructor(
    public readonly id: string,
    private readonly worker: Worker,
    private readonly signalingGateway: SignalingGateway,
    private readonly logger: Logger,
    private readonly mediasoupConfig: MediasoupConfigType,
  ) {
    this.createRouter();
  }
  private async createRouter(): Promise<void> {
    this.router = await this.worker.createRouter({
      // @ts-ignore
      mediaCodecs: this.mediasoupConfig.mediasoup.router.mediaCodecs,
    });
  }
  /**
   * Add new peer to room.
   */
  addPeer(peer: any) {
    this.peers.set(peer.id, peer);
  }
  getProducersForPeer() {
    const producers: { producerId: string }[] = [];
    this.peers.forEach((peer) => {
      peer.producers.forEach((producer) => {
        producers.push({
          producerId: producer.id,
        });
      });
    });
    return producers;
  }
  getRtpCapabilities(): RtpCapabilities {
    return this.router.rtpCapabilities;
  }
  async createWebRtcTransport(socketId: string): Promise<{
    params: {
      iceParameters: IceParameters;
      iceCandidates: IceCandidate[];
      dtlsParameters: DtlsParameters;
      id: string;
```

```
    };
  }> {
    const webRtcTransport = await WebrtcTransportFactory.build(this.router,
this.mediasoupConfig);
    const peer = this.peers.get(socketId);
    if (!peer) {
      throw new Error('Peer not found');
    }
    peer.addTransport(webRtcTransport);
    return {
      params: {
        id: webRtcTransport.id,
        iceParameters: webRtcTransport.iceParameters,
        iceCandidates: webRtcTransport.iceCandidates,
        dtlsParameters: webRtcTransport.dtlsParameters,
      },};}
  async connectPeerTransport(
    socketId: string,
    transportId: string,
    dtlsParameters: DtlsParameters,
  ): Promise<void> {
    const peer = this.peers.get(socketId);
    if (!peer) {
      this.logger.info('Peer by specified ID not found', {
        socketId,});
      return;}
    await peer.connectTransport(transportId, dtlsParameters);}
  async produce(
    socketId: string,
    transportId: string,
    rtpParameters: RtpParameters,
    kind: MediaKind,
  ): Promise<string> {
    // handle undefined errors
    return new Promise(async (resolve: (id: string) => void, reject: any) =>{
      const peer = this.peers.get(socketId);
      if (!peer) {
        this.logger.error('Peer not found for produce', {
          peerId: socketId,});
        return false;}
      const producer = await peer.createProducer(transportId, rtpParameters,
kind);
      if (!producer) {
        this.logger.error('Failed to create producer', {
          transportId,
          rtpParameters,
          kind,});
        return false;}
      resolve(producer.id);
      this.broadCast(socketId, MediasoupEmittingEventsEnum.NEW_PRODUCERS, [{
          producerId: producer.id,
          producerSocketId: socketId,},]);});}
  async createConsumer(
    socketId: string,
    consumerTransportId: string,
    producerId: string,
    rtpCapabilities: any,
  ): Promise<boolean> {
    // handle nulls
    const routerCanConsume = this.router.canConsume({
      producerId,
      rtpCapabilities,
    });
```

```
    if (!routerCanConsume) {
      this.logger.error('Can not consume', {
        producerId,
        rtpCapabilities,});
      return false;}
    const peer = this.peers.get(socketId);
    if (!peer) {
      this.logger.error('Consume peer not found');
      return false;}
    const consumer = await peer.createConsumer(consumerTransportId,
producerId, rtpCapabilities);
    this.logger.info('Consumer #createConsumer', consumer);
    if (consumer) {
      let producerPeer;
      this.peers.forEach((peer) => {
        if (peer.getProducer(producerId)) {
          producerPeer = peer}});
      this.signalingGateway.emit(socketId,
MediasoupEmittingEventsEnum.NEW_CONSUMER, {
        peerId: producerPeer.id,
        producerId: producerId,
        id: consumer.id,
        kind: consumer.kind,
        rtpParameters: consumer.rtpParameters,
        type: consumer.type,
        producerPaused: consumer.producerPaused,});}}
  async removePeer(socketId: string) {
    this.peers.get(socketId)?.close();
    this.peers.delete(socketId);}
  closeProducer(socketId: string, producerId: string) {
    const peer = this.peers.get(socketId);
    if (peer) {
      peer.closeProducer(producerId);}}
  broadCast(socketId: string, name: string, data: any) {
    for (const otherID of Array.from(this.peers.keys()).filter((id) => id !==
socketId)) {
      this.send(otherID, name, data);}}
  send(socketId: string, name: string, data: any) {
    this.signalingGateway.emit(socketId, name, data);
    this.logger.info(`Sent socket event`, {
      socketId,
      name,});}
  getPeers() {
    return this.peers;}
  toJson() {
    return {
      id: this.id,
      peers: JSON.stringify([...this.peers]),
    };}}
import { Router } from 'mediasoup/node/lib/Router';
import { WebRtcTransport } from 'mediasoup/node/lib/WebRtcTransport';
import { MediasoupConfigType } from '../../../configs';
export class WebrtcTransportFactory {
  public static async build(router: Router, config: MediasoupConfigType):
Promise<WebRtcTransport> {
    const { maxIncomingBitrate, initialAvailableOutgoingBitrate } =
      config.mediasoup.webRtcTransport;
    const transport = await router.createWebRtcTransport({
      listenIps: config.mediasoup.webRtcTransport.listenIps,
      enableUdp: true,
      enableTcp: true,
      preferUdp: true,
      initialAvailableOutgoingBitrate,});
```

```
    if (maxIncomingBitrate) {
      await transport.setMaxIncomingBitrate(maxIncomingBitrate);}
    transport.on('dtlsstatechange', (dtlsState) => {
      if (dtlsState === 'closed') {
        console.info('Transport close');
        transport.close();}});
    return transport;}}
export enum MediasoupEmittingEventsEnum {
  NEW_PRODUCERS = 'newProducers',
  NEW_CONSUMER = 'newConsumer',
  CONSUMER_CLOSED = 'consumerClosed',}
import { Logger } from '../../../common';
import { Client } from '../libs/interfaces';
import Room from '../libs/room';
import {
  CallServerListeningEventsEnum,
  GuestUserCallServerListeningEvents,
  CallServerSendEvents,
} from 'call-app-utils';
import WorkersCollection from '../workers-collection';
import { SignalingGateway } from '../interfaces';
import { Peer } from '../libs/peer';
import { MediasoupEmittingEventsEnum } from './mediasoup-emitting-
events.enum';
import ClientsCollection from '../clients-collection';
import { MediasoupConfigType } from '../../../configs';
import { DtlsParameters } from 'mediasoup/node/lib/WebRtcTransport';
import { MediaKind, RtpParameters } from 'mediasoup/node/lib/RtpParameters';
type ListeningEventFunction = (client: Client, payload: unknown) => unknown;
export class MediasoupEventsHandler {
  private readonly guestEventsMapper: Record<
    GuestUserCallServerListeningEvents,
    ListeningEventFunction
  > = {
    [GuestUserCallServerListeningEvents.ConnectClient]:
this.connectClient.bind(this),
  };
  private readonly authUserEventsMapper: Record<
    CallServerListeningEventsEnum,
    ListeningEventFunction
  > = {
    [CallServerListeningEventsEnum.CREATE_ROOM]:
this.createRoomEvent.bind(this),
    [CallServerListeningEventsEnum.JOIN]: this.joinToRoomEvent.bind(this),
    [CallServerListeningEventsEnum.GET_PRODUCERS]:
this.getProducers.bind(this),
    [CallServerListeningEventsEnum.GET_ROUTER_RTP_CAPABILITIES]:
      this.getRouterRtpCapabilitiesEvent.bind(this),
    [CallServerListeningEventsEnum.CREATE_WEBRTC_TRANSPORT]:
      this.createWebRtcTransportEvent.bind(this),
    [CallServerListeningEventsEnum.CONNECT_WEBRTC_TRANSPORT]:
      this.connectWebRtcTransport.bind(this),
    [CallServerListeningEventsEnum.CONNECT_TRANSPORT]:
this.connectTransport.bind(this),
    [CallServerListeningEventsEnum.PRODUCE]: this.produce.bind(this),
    [CallServerListeningEventsEnum.CONSUME]: this.createConsumer.bind(this),
    [CallServerListeningEventsEnum.RESUME]: this.resume.bind(this),
    [CallServerListeningEventsEnum.GET_MY_ROOM_INFO]:
this.getMyRoomInfo.bind(this),
    [CallServerListeningEventsEnum.EXIT]: this.exitRoom.bind(this),
    [CallServerListeningEventsEnum.PRODUCER_CLOSED]:
this.producerClosed.bind(this),
    [CallServerListeningEventsEnum.EXIT_ROOM]: this.exitRoom.bind(this),
```

```
    [CallServerListeningEventsEnum.CLOSE_PRODUCER]:
this.closeProducer.bind(this),
  };
  constructor(
    private readonly rooms: Map<string, Room>,
    private readonly clientsCollection: ClientsCollection,
    private readonly logger: Logger,
    private readonly workersCollection: WorkersCollection,
    private readonly signalingGateway: SignalingGateway,
    private readonly mediasoupConfig: MediasoupConfigType,
  ) {}
  async processClientEvent(
    clientId: string,
    eventName: CallServerListeningEventsEnum,
    payload?: unknown,
  ): Promise<unknown> {
    if (!eventName) {
      this.logger.warn('Processing event is empty', {
        eventName,});
      return;}
    try {
      if (this.guestEventsMapper[eventName]) {
        return this.guestEventsMapper[eventName](clientId, payload)}
      if (!this.authUserEventsMapper[eventName]) {
        this.logger.warn('Specified event for processing not found', {
          clientId,
          eventName,
          payload,});}
      if (!this.clientsCollection.hasClientById(clientId)) {
        this.logger.info('Client has not access to send the event', {
          clientId,
          eventName,
        });
        return;
      }
      const client = this.clientsCollection.getById(clientId);
      return this.authUserEventsMapper[eventName](client, payload);
    } catch (e) {
      this.logger.error(
        'Failed to processing an event',
        {
          clientId,
          eventName,
          payload,
        },e,);}}
  async connectClient(clientId: string, payload?: { roomId: string }) {
    if (this.clientsCollection.hasClientById(clientId)) {
      this.logger.warn('Client already exists in system by specified ID', {
        clientId,
      });
      // return false;
      // }
    } else {
      this.clientsCollection.add(clientId);
    }
    this.clientsCollection.getById(clientId).roomId = payload.roomId;
    this.logger.info('Client connected', {
      clientId,
    });
    return true;
  }
  private async createRoomEvent(client: Client, payload?: { roomId: string
}): Promise<string> {
```

```
    const { roomId } = payload;
    if (this.rooms.has(roomId)) {
      return roomId;
    }
    const worker = this.workersCollection.getNextWorker();
    const room = new Room(roomId, worker, this.signalingGateway, this.logger,
this.mediasoupConfig);
    this.rooms.set(roomId, room);
    this.logger.info('Created new room', {
      roomId,
    });
    return roomId;
  }
  private async joinToRoomEvent(client: Client, payload?: { roomId: string
}): Promise<boolean> {
    const { roomId } = payload;
    const room = this.rooms.get(roomId);
    if (!room) {
      this.logger.warn('Room for joining does not exists', {
        clientId: client.id,
        payload,
      });
      return false;
    }
    const peer = new Peer(client.id, `#client:${client.id}`, this.logger,
this.signalingGateway);
    room.addPeer(peer);
    client.roomId = room.id;
    client.peer = peer;
    return true;
  }
  private async getProducers(client: Client, payload?: unknown):
Promise<boolean> {
    const room = this.rooms.get(client.roomId);
    if (!room) {
      this.logger.warn('Room does not exists for getting producers', {
        clientId: client.id,
        payload,
      });
      return false;
    }
    const producersIds = room.getProducersForPeer();
    this.signalingGateway.emit(client.id, CallServerSendEvents.NEW_PRODUCERS,
producersIds);
    return true;
  }
  private async getRouterRtpCapabilitiesEvent(client: Client, payload?:
unknown): Promise<unknown> {
    const room = this.rooms.get(client.roomId);
    if (!room) {
      this.logger.warn('Room does not exists for get RTP capabilities', {
        clientId: client.id,
        payload,
      });
      return false;
    }
    return room.getRtpCapabilities();
  }
  private async createWebRtcTransportEvent(client: Client, payload?:
unknown): Promise<unknown> {
    const room = this.rooms.get(client.roomId);
    if (!room) {
      this.logger.warn('Room does not exists for createWebRtcTransportEvent',
```

```
{
        clientId: client.id,
        payload,
      });
      return false;
    }
    const { params } = await room.createWebRtcTransport(client.id);
    return params;
  }
  private async connectWebRtcTransport(
    client: Client,
    payload?: { transportId: string; dtlsParameters: DtlsParameters },
  ): Promise<unknown> {
    if (!client.peer) {
      this.logger.error('Client has not peer', {
        clientId: client.id,
      });
      return;
    }
    const { transportId, dtlsParameters } = payload;
    await client.peer.connectTransport(transportId, dtlsParameters);
  }
  private async connectTransport(
    client: Client,
    payload?: { transportId: string; dtlsParameters: DtlsParameters },
  ): Promise<unknown> {
    const room = this.rooms.get(client.roomId);
    if (!room) {
      this.logger.warn('Room does not exists for createWebRtcTransportEvent',
{
        clientId: client.id,
        payload,
      });
      return false;
    }
    const { transportId, dtlsParameters } = payload;
    await room.connectPeerTransport(client.id, transportId, dtlsParameters);
    return 'success';
  }
  private async produce(
    client: Client,
    payload?: {
      transportId: string;
      rtpParameters: RtpParameters;
      kind: MediaKind;
    },
  ): Promise<{ producerId: string }> {
    const room = this.rooms.get(client.roomId);
    const { kind, rtpParameters, transportId } = payload;
    const producerId = await room.produce(client.id, transportId,
rtpParameters, kind);
    return {
      producerId,
    };
  }
  private async closeProducer(client: Client, payload: { producerId: string
}) {
    const room = this.rooms.get(client.roomId);
    room.closeProducer(client.id, payload.producerId);
  }
  private async createConsumer(client: Client, payload?: any):
Promise<boolean> {
    const room = this.rooms.get(client.roomId);
```

```
    const { consumerTransportId, producerId, rtpCapabilities } = payload;
    return room.createConsumer(client.id, consumerTransportId, producerId,
rtpCapabilities);
  }
  private async resume(client: Client, payload?: unknown): Promise<void> {
    this.logger.warn('No implemented resume() method');
  }
  private async getMyRoomInfo(client: Client, payload?: unknown):
Promise<unknown> {
    const room = this.rooms.get(client.roomId);
    return room.toJson();
  }
  private async producerClosed(client: Client, payload?: { producerId: string
}): Promise<void> {
    const room = this.rooms.get(client.roomId);
    const { producerId } = payload;
    room.closeProducer(client.id, producerId);
  }
  private async exitRoom(client: Client) {
    const room = this.rooms.get(client.roomId);
    if (!room) {
      this.logger.warn('Not currently in a room');
      return;
    }
    await room.removePeer(client.id);
    this.clientsCollection.delete(client.id);
    if (!room.getPeers().size) {
      this.rooms.delete(room.id);
    }
  }
}
import { Client } from './libs/interfaces';
export default class ClientsCollection {
  private readonly clients: Map<string, Client>;
  constructor() {
    this.clients = new Map<string, Client>();
  }
  list(): Readonly<Map<string, Client>> {
    return this.clients;
  }
  add(id: string) {
    this.clients.set(id, {
      id: id,
    });
  }
  getById(id: string): Client | undefined {
    return this.clients.get(id);
  }
  hasClientById(id: string): boolean | undefined {
    return this.clients.has(id);
  }
  delete(id: string) {
    this.clients.delete(id);
  }
}
import { Logger } from '../../common';
const mediasoup = require('mediasoup');
import Room from './libs/room';
import { Worker } from 'mediasoup/node/lib/Worker';
import { Client } from './libs/interfaces';
import { MediasoupEventsHandler } from './mediasoup-events-handler';
import WorkersCollection from './workers-collection';
import { SignalingGateway } from './interfaces/signaling-gateway';
```

```
import ClientsCollection from './clients-collection';
import { CallServerListeningEventsEnum } from 'call-app-utils';

export default class MediasoupServer {
  private readonly rooms: Map<string, Room>;
  private readonly workersCollection: WorkersCollection;
  private readonly clientsCollection: ClientsCollection;
  private readonly mediasoupEventsListener: MediasoupEventsHandler;
  constructor(
    private readonly mediasoupConfig: any,
    private readonly signalingServer: SignalingGateway,
    private readonly logger: Logger,
  ) {
    this.rooms = new Map();
    this.workersCollection = new WorkersCollection();
    this.clientsCollection = new ClientsCollection();
    this.mediasoupEventsListener = new MediasoupEventsHandler(
      this.rooms,
      this.clientsCollection,
      this.logger,
      this.workersCollection,
      this.signalingServer,
      this.mediasoupConfig,
    );
  }
  async start() {
    this.logger.info('Starting mediasoup server');
    await this.createWorkers();
  }
  async createWorkers() {
    const { numWorkers, worker: workerConfig } =
this.mediasoupConfig.mediasoup;
    const { logLevel, logTags, rtcMinPort, rtcMaxPort } = workerConfig;
    for (let i = 0; i < numWorkers; i++) {
      const worker = await mediasoup.createWorker({
        logLevel,
        logTags,
        rtcMinPort,
        rtcMaxPort,
      });
      worker.on('died', () => {
        this.logger.error('Mediasoup worker died, exiting in 2 seconds...
[pid:%d]', worker.pid);
        setTimeout(() => {
          process.exit(1);
        }, 2000);
      });
      this.workersCollection.addWorker(worker);
    }
    this.logger.info(`Created ${this.workersCollection.listWorkers().length}
mediasoup workers`);
  }
  async handleClientEvent(
    clientId: string,
    eventName: CallServerListeningEventsEnum,
    payload?: unknown,
  ): Promise<unknown> {
    return this.mediasoupEventsListener.processClientEvent(clientId,
eventName, payload);
  }
  async exit() {
    this.logger.info('Stopping call server..');
```

```
  }
}
import { Worker } from 'mediasoup/node/lib/Worker';
export default class WorkersCollection {
  private readonly workers: Array<Worker>;
  private nextWorkerIndex = 0;
  constructor() {
    this.workers = [];
  }
  listWorkers(): Readonly<Array<Worker>> {
    return this.workers;
  }
  addWorker(worker: Worker) {
    this.workers.push(worker);
  }
  getNextWorker(): Worker {
    const worker = this.workers[this.nextWorkerIndex];
    if (++this.nextWorkerIndex == this.workers.length) {
      this.nextWorkerIndex = 0;
    }
    return worker;
  }
}
import { Server } from 'socket.io';
import { SignalingGateway as SignalingGatewayInterface } from './mediasoup-
server/interfaces';
import { Logger } from '../common';
export class SignalingGateway implements SignalingGatewayInterface {
  constructor(
    private readonly socketServer: Server,
    private readonly logger: Logger,
  ) {
  }
  emit(id: string, eventName: string, args?: string | number | object |
boolean): boolean {
    this.logger.info('Emitting the event', {
      clientId: id,
      eventName,
      payload: args,
    });
    // TODO: Refactor, send to specified client
    return this.socketServer.local.emit(eventName, {
      clientId: id,
      payload: args,
    });
    // return this.socketServer.to(id).emit(eventName, args);
  }
}
import dotenv from 'dotenv-safe';
import io from 'socket.io';
import MEDIASOUP_CONFIG from './configs/mediasoup-config';
import MediasoupServer from './modules/mediasoup-server/mediasoup-server';
import { Logger } from './common';
import { SignalingGateway } from './modules/signaling-gateway';
import express from 'express';
import fs from 'fs';
import path from 'path';
import https, { ServerOptions } from 'https';
import { nanoid } from 'nanoid';
dotenv.config();
async function run() {
  const logger = new Logger();
  const expressApp = express();
```

```
  expressApp.use(express.static(path.join(__dirname, '..', 'public')));
  expressApp.use(express.static(path.join(__dirname, '..', 'node_modules')));
  const httpsServer = https.createServer(
    {
      key: fs.readFileSync(path.join(__dirname, '..', 'ssl', 'key.pem'),
'utf-8'),
      cert: fs.readFileSync(path.join(__dirname, '..', 'ssl', 'cert.pem'),
'utf-8'),
    },
    expressApp,
  );
  const socketIoServer = new io.Server(httpsServer, {
    cors: {
      origin: '*',
      methods: ['GET', 'POST'],
    },
  });
  const signalingGateway = new SignalingGateway(socketIoServer, logger);
  const mediasoupServer = new MediasoupServer(MEDIASOUP_CONFIG,
signalingGateway, logger);
  socketIoServer.on('connection', (socket) => {
    logger.info(`Socket connected`, {
      id: socket.id,
    });
    // TODO: Listen only required events
    socket.onAny(async (eventName, payload, responseCallback) => {
      const requestId = nanoid();
      logger.info('Processing request of event', {
        requestId,
        clientId: payload.clientId,
        eventName,
        payload: payload.payload,
      });
      // eslint-disable-next-line max-len
      const eventResponse = await mediasoupServer.handleClientEvent(
        payload.clientId,
        eventName,
        payload.payload,
      );
           logger.info('Response for request of event', {
        requestId: requestId,
        response: eventResponse,
      });
      if (typeof responseCallback === 'function') {
        responseCallback(eventResponse);
      }
    });
    socket.on('disconnect', (reason) => {
      logger.info(`Socket disconnected for reason: ${reason}`, {
        id: socket.id,
      });
    });
  });
  logger.info('Starting app');
  await mediasoupServer.start();
  const PORT = 3016;
  const LISTEN_IP = '192.168.0.105';
  httpsServer.listen(PORT, LISTEN_IP, () => {
    logger.info('Listening on https://' + LISTEN_IP + ':' + PORT);
  });
}
run();
```

В.6 Класи модулів користувача, кімнати та вебсокетів

```
import { Body, Controller, Get, Param, Post, Query, UseGuards } from
'@nestjs/common';
import { ApiBearerAuth, ApiExtraModels, ApiOperation, ApiTags } from
'@nestjs/swagger';
import { RoomService } from './room.service';
import { CreateRoomRequestDto, ListRoomsRequestDto, ListRoomsResponseDto,
RoomResponseDto } from './dtos';
import { UserEntity } from '../user/entities';
import { User } from '@common/decorators';
import { ClientRoleGuard } from '../auth/http-auth-role.guard';
import { ApiSuccessResponse, SuccessResponse } from '@common/response-data';
import { ParseObjectIdPipe } from '@common/pipes';
@ApiTags('room')
@ApiExtraModels(RoomResponseDto, ListRoomsResponseDto)
@ApiBearerAuth()
@Controller('room')
export class RoomController {
  constructor(
    private readonly roomService: RoomService,
  ) {
  }
  @UseGuards(ClientRoleGuard)
  @ApiOperation({
    description: 'Get a list of all rooms',
  })
  @ApiSuccessResponse(ListRoomsResponseDto)
  @Get('list')
  async getListRooms(
    @User() user: UserEntity, @Query()  query: ListRoomsRequestDto,
  ): Promise<SuccessResponse<ListRoomsResponseDto>> {
    const listRooms = await this.roomService.getListRooms(user, query);
    return {
      data: listRooms,
    };
  }
  @UseGuards(ClientRoleGuard)
  @ApiOperation({
    description: 'Get a list of all rooms',
  })
  @ApiSuccessResponse(ListRoomsResponseDto)
  @Get(':id')
  async getRoom(
    @User() user: UserEntity, @Param('id')  roomId: string,
  ): Promise<SuccessResponse<RoomResponseDto>> {
    const room = await this.roomService.getRoomInfo(user, roomId);
    return {
      data: room,
    };
  }
  @UseGuards(ClientRoleGuard)
  @ApiOperation({
    description: 'Create a call room',
  })
  @ApiSuccessResponse(RoomResponseDto)
  @Post()
  async createRoom(
    @User() user: UserEntity, @Body() requestBody: CreateRoomRequestDto,
  ): Promise<SuccessResponse<RoomResponseDto>> {
    const createdRoom = await this.roomService.createRoom(user, requestBody);
    return {
```

```
      data: createdRoom,
    };
  }
  @UseGuards(ClientRoleGuard)
  @ApiOperation({
    description: 'Join to room',
  })
  @Post(':id/join')
  async joinToRoom(
    @User() user: UserEntity,
    @Param('id', ParseObjectIdPipe) roomId: string
  ): Promise<SuccessResponse<unknown>> {
    const joinResult = await this.roomService.joinToRoom(user, { roomId });
    return {
      data: joinResult,};}}
import { BadRequestException, Injectable, InternalServerErrorException,
NotFoundException } from '@nestjs/common';
import { RoomRepository, UserRepository } from '@db/entities';
import { CreateRoomRequestDto, ListRoomsRequestDto, ListRoomsResponseDto,
RoomResponseDto } from './dtos';
import { makeUserEntity, UserEntity } from '../user/entities';
import { makeRoomEntity, RoomEntity } from './entities';
import { CallAppClient } from '../call-app/call-app.client';
import { AppLogger } from '../../logger/app-logger';
import { WebsocketService } from '../websocket';
import { RoomConstants } from './constants';
@Injectable()
export class RoomService {
  constructor(
    private readonly roomRepository: RoomRepository,
    private readonly userRepository: UserRepository,
    private readonly callAppClient: CallAppClient,
    private readonly appLogger: AppLogger,
    private readonly websocketService: WebsocketService,
  ) {
  }
  private async makeRoomResponse(room: RoomEntity) {
    const roomCreator = await this.userRepository.findById(room.userId);
    let participants = [];
    if (room.participants && room.participants.length) {
      const subParticipants = room.participants.slice(0,
RoomConstants.MaxDisplayParticipantsInRoomList);
      participants = await this.userRepository.findByIds(subParticipants);
    }
    const participantsResponse = participants.map((participant) => {
      return {
        id: participant.id,
        avatar: participant.avatar,
        firstName: participant.firstName,
        lastName: participant.lastName,
      };
    });
    return {
      ...room,
      totalParticipants: participants.length,
      author: {
        id: roomCreator.id,
        avatar: roomCreator.avatar,
        firstName: roomCreator.firstName,
        lastName: roomCreator.lastName,
      },
      limitDisplayParticipants:
RoomConstants.MaxDisplayParticipantsInRoomList,
```

```
      participants: participantsResponse,
    };
  }
  async getListRooms(user: UserEntity, queryOptions: ListRoomsRequestDto):
Promise<ListRoomsResponseDto> {
    const roomsEntities = await this.searchRoomsByQuery(queryOptions);
    const roomsResponsePromises = roomsEntities.map((room) => {
      return this.makeRoomResponse(room);
    });
    const roomsResponses = await Promise.all(roomsResponsePromises);
    return {
      rooms: roomsResponses,
    };
  }
  private async searchRoomsByQuery(query: {
    title?: string
    topics?: string[],
    language?: string,
  }): Promise<RoomEntity[]> {
    const titleFilter = query.title ? { title: { $regex: `.*${query.title}.*`
} } : {};
    const topicFilter = (query.topics && query.topics.length) ? { topic: {
$in: query.topics } } : {};
    const languageFilter = query.language ? { language: query.language } :
{};
    const roomOrmEntities = await this.roomRepository.findMany({
      isActive: true,
      ...titleFilter,
      ...topicFilter,
      ...languageFilter,
    });
    const roomEntities = roomOrmEntities.map((ormRoom) =>
makeRoomEntity(ormRoom));
    return roomEntities;
  }
  async getRoomInfo(user: UserEntity, roomId: string):
Promise<RoomResponseDto> {
    const roomOrmEntity = await this.roomRepository.findById(roomId);
    if (!roomOrmEntity) {
      throw new NotFoundException();
    }
    const roomEntity = makeRoomEntity(roomOrmEntity);
    return this.makeRoomResponse(roomEntity);
  }
  async createRoom(user: UserEntity, options: CreateRoomRequestDto):
Promise<RoomResponseDto> {
    const roomOrmEntity = await this.roomRepository.insertRoom({
      userId: user.id,
      title: options.title,
      topic: options.topic,
      language: options.language,
      isActive: true,
      participants: [],
    });
    const roomEntity = makeRoomEntity(roomOrmEntity);
    await this.callAppClient.authorizeUser(user.id);
    try {
      // await this.callAppClient.createRoom(roomEntity.id, user.id);
    } catch (e) {
      this.appLogger.error('Failed at creating room in call app', {
        roomId: roomEntity.id,
        userId: user.id,
      }, e);
```

```
      throw new InternalServerErrorException('Error creating room, please try
again later');
    }
    return this.makeRoomResponse(roomEntity);
  }
  async joinToRoom(user: UserEntity, options: { roomId: string }):
Promise<unknown> {
    if (user.activeRoomId) {
      throw new BadRequestException('User already joined to another room');
    }
    const roomOrmEntity = await this.roomRepository.findById(options.roomId);
    if (!roomOrmEntity) {
      throw new BadRequestException('Room not found');}
    await this.roomRepository.addParticipant(roomOrmEntity.id, user.id);
    const roomEntity = makeRoomEntity(roomOrmEntity);
    const socket = await this.websocketService.getSocketByUser(user.id);
    this.websocketService.joinSocketToCallRoom(socket, options.roomId);
    try {
      await this.callAppClient.connectUser(options.roomId, user.id);
      await this.callAppClient.createRoom(options.roomId, user.id);
      // await this.callAppClient.joinToRoom(options.roomId, user.id);
    } catch (e) {
      this.appLogger.error('Failed at joining room in call app', {
        roomId: options.roomId,
        userId: user.id,
      }, e);
      throw new InternalServerErrorException('Failed at joining room in call
app');}
    await this.userRepository.setActiveRoomForUser(user.id, options.roomId);
    this.websocketService.sendToCallRoom(options.roomId,
'call.newParticipant', { user });
    return this.makeRoomResponse(roomEntity);}
  async removeUserFromActiveRoom(userId: string) {
    const userOrmEntity = await this.userRepository.findById(userId);
    if (!userOrmEntity) {
      this.appLogger.info('User not found at removing participant from
room');
      return;}
    const user = makeUserEntity(userOrmEntity);
    if (!user.activeRoomId) {
      this.appLogger.info('The user does not have an active room for
exiting');
      return;}
    await this.roomRepository.removeParticipant(user.activeRoomId, user.id);
    await this.userRepository.clearActiveRoom(user.id);
    await this.callAppClient.removeUser(user.id);
    this.websocketService.sendToCallRoom(user.activeRoomId,
'call.participantExited', { user });
}}
import { Injectable } from '@nestjs/common';
import { MessageBody, SubscribeMessage } from '@nestjs/websockets';
import { OnEvent } from '@nestjs/event-emitter';
import { WebsocketService } from '../websocket';
import { Socket } from 'socket.io-client';
import { AuthorizedSocket } from '../websocket/websocket.server';
import { RoomService } from './room.service';
import { makeUserEntity } from '../user/entities';
import { AppLogger } from '../../logger/app-logger';
import { UserRepository } from '@db/entities';
@Injectable()
export class RoomWsEvents {
  constructor(
    private readonly websocketService: WebsocketService,
```

```
    private readonly roomService: RoomService,
    private readonly appLogger: AppLogger,
    private readonly userRepository: UserRepository ) {}
  @OnEvent('room.getParticipants')
  async getRoomParticipants(event: { payload: { roomId: string }, socket:
AuthorizedSocket }) {
    const userOrmEntity = await
this.userRepository.findById(event.socket.user.id);
    if (!userOrmEntity) {
      this.appLogger.info('User not found at removing participant from
room');
      return;}
    const user = makeUserEntity(userOrmEntity);
    const roomInfo = await this.roomService.getRoomInfo(user,
event.payload.roomId);
    this.websocketService.sendToUser(event.socket.user.id,
'room.getParticipants', roomInfo.participants);}
  @OnEvent('call.participantLeaveCall')
  async participantLeavesCall(event: { socket: AuthorizedSocket }) {
    this.roomService.removeUserFromActiveRoom(event.socket.user.id);}
  @OnEvent('socket.disconnected')
  async socketDisconnected(event: { socket: AuthorizedSocket }) {
    this.roomService.removeUserFromActiveRoom(event.socket.user.id);}}
import { Body, Controller, Get, Put, Req, UseGuards } from '@nestjs/common';
import { ClientRoleGuard } from '../auth/http-auth-role.guard';
import { ApiBearerAuth, ApiExtraModels, ApiOkResponse, ApiOperation, ApiTags
} from '@nestjs/swagger';
import { UserResponseDto } from './dtos';
import { UserService } from './user.service';
import { ApiSuccessResponse, SuccessResponse } from '@common/response-data';
import { UpdateProfileRequestDto } from './dtos/update-profile.request.dto';
@ApiTags('user')
@ApiBearerAuth()
@UseGuards(ClientRoleGuard)
@Controller('user')
@ApiExtraModels(UserResponseDto)
export class UserController {
  constructor(private readonly userService: UserService) {}
  @ApiOperation({
    description: 'Get profile for current authorize user',})
  @ApiSuccessResponse(UserResponseDto)
  @Get('me')
  async profile(@Req() req): Promise<SuccessResponse<UserResponseDto>> {
    const { user } = req;
    const profile = await this.userService.getProfile(user);
    return {
      data: profile,};}
  @ApiOperation({
    description: 'Update user\'s profile',})
  @ApiSuccessResponse(UserResponseDto)
  @Put('settings')
  async updateProfile(@Req() req, @Body() options: UpdateProfileRequestDto):
Promise<SuccessResponse<UserResponseDto>> {
    const { user } = req;
    const profile = await this.userService.updateProfile(user, options);
    return {
      data: profile,};}}
import { Injectable } from '@nestjs/common';
import { UpdateProfileRequestDto, UserResponseDto } from './dtos';
import { makeUserEntity, UserEntity } from './entities';
import { UserRepository } from '@db/entities';
@Injectable()
export class UserService {
```

```
  constructor(
    private readonly userRepository: UserRepository,
  ) {}
  private makeUserResponse(user: UserEntity): UserResponseDto {
    return {
      id: user.id,
      firstName: user.firstName,
      lastName: user.lastName,
      avatar: user.avatar,
      email: user.email,};}
  async getProfile(user: UserEntity): Promise<UserResponseDto> {
    return this.makeUserResponse(user);}
  async updateProfile(user: UserEntity, options: UpdateProfileRequestDto):
Promise<UserResponseDto> {
    await this.userRepository.updateUser(user.id, {
      firstName: options.firstName,
      lastName: options.lastName,});
    const updatedUser = await this.userRepository.findById(user.id);
    return this.makeUserResponse(makeUserEntity(updatedUser));}}
import {
  ConnectedSocket,
  MessageBody,
  OnGatewayConnection,
  OnGatewayInit,
  SubscribeMessage,
  WebSocketGateway,
  WebSocketServer,
} from '@nestjs/websockets';
import { Server, Socket } from 'socket.io';
import { CallAppClient } from '../call-app/call-app.client';
import { AppLogger } from '../../logger/app-logger';
import { EventEmitter } from 'events';
import JwtAuthService from '../auth/jwt/jwt-auth.service';
import { UserRepository } from '@db/entities';
import { makeUserEntity, UserEntity } from '../user/entities';
import { EventEmitter2 } from '@nestjs/event-emitter';
export enum WebsocketServerEvents {
  NewConnection = 'newConnection'}
export type AuthorizedSocket = Socket & {
  user: { id: UserEntity['id'] }}
@WebSocketGateway({
  cors: {
    origin: '*',
  },})
export class WebsocketServer extends EventEmitter implements OnGatewayInit,
OnGatewayConnection {
  @WebSocketServer()
  server: Server;
  constructor(
    private readonly callAppClient: CallAppClient,
    private readonly appLogger: AppLogger,
    private readonly jwtAuthService: JwtAuthService,
    private readonly userRepository: UserRepository,
    private eventEmitter: EventEmitter2,
  ) {super();}
  async afterInit(server: Server): Promise<void> {}
  async getSocketByUserAndRoom(userId: string, roomId: string):
Promise<AuthorizedSocket | null> {
    // const sockets = await this.server.fetchSockets();
    const sockets = await this.server.of(`room:${roomId}`).fetchSockets();
    // @ts-ignore
    return sockets.find((socket) => socket.user?.id === userId);}
  async getSocketByUser(userId: string): Promise<AuthorizedSocket | null> {
```

```
    const sockets = await this.server.fetchSockets();
    // @ts-ignore
    return sockets.find((socket) => socket.user?.id === userId);}
  sendToRoom(room: string, eventName: string, payload: unknown): void {
    this.server.to(room).emit(eventName, payload);}
  async handleConnection(socket: AuthorizedSocket): Promise<any> {
    try {
      const verifyResult = await this.verifyNewConnection(socket);
      socket.user = verifyResult.user;
      socket.join(socket.user.id);
      this.appLogger.info('Socket connected', {
        socketId: socket.id,
        userId: socket.user.id,});
    } catch (e) {
      socket.disconnect();
      this.appLogger.info('Failed to handle new socket connection, reject
connection', {
        details: e.message,
        socketId: socket.id,});
      return;}
    this.emit(WebsocketServerEvents.NewConnection, socket);
    socket.on('disconnect', (reason) => {
      this.appLogger.info(`Socket disconnected ${socket.id} by reason
${reason}`);
      this.eventEmitter.emit('socket.disconnected', {
        socket,}); });}
  async verifyNewConnection(socket: Socket): Promise<{ user?: UserEntity }> {
    // @ts-ignore
    const authorizationHeader = socket.handshake.headers?.authorization;
    let bearerToken = authorizationHeader.split('Bearer ')[1];
    if (!bearerToken) {
      throw new Error('Not found Bearer token for new socket connection');}
    const tokenPayload = this.jwtAuthService.decode(bearerToken);
    if (!tokenPayload) {
      throw new Error('Failed to decode Bearer token for new socket
connection');}
    const user = await this.userRepository.findById(tokenPayload.sub);
    if (!user) {
      throw new Error('Not found user for new socket connection');}
    const userEntity = makeUserEntity(user);
    return { user: userEntity };}
  @SubscribeMessage('room.getParticipants')
  async getRoomParticipants(@MessageBody() payload: unknown,
@ConnectedSocket() socket: Socket) {
    this.eventEmitter.emit('room.getParticipants', {
      socket,
      payload,
    });
  }
  @SubscribeMessage('call.leaveCall')
  async participantLeavesCall(@MessageBody() payload: unknown,
@ConnectedSocket() socket: Socket) {
    this.eventEmitter.emit('call.participantLeaveCall', {
      socket, });}}
import { Injectable } from '@nestjs/common';
import { AuthorizedSocket, WebsocketServer, WebsocketServerEvents } from
'./websocket.server';
import { EventEmitter } from 'events';
@Injectable()
export class WebsocketService extends EventEmitter {
  constructor(
    private readonly websocketServer: WebsocketServer,
  ) {
```

```
    super();
    websocketServer.addListener(WebsocketServerEvents.NewConnection,
(...args) => {
      this.emit(WebsocketServerEvents.NewConnection, ...args);
    });}
  async getSocketByUser(userId: string): Promise<AuthorizedSocket | null> {
    return this.websocketServer.getSocketByUser(userId);}
  async getSocketByUserAndRoom(userId: string, roomId: string):
Promise<AuthorizedSocket | null> {
    return this.websocketServer.getSocketByUserAndRoom(userId, roomId);}
  joinSocketToCallRoom(socket: AuthorizedSocket, roomId: string) {
    // @ts-ignore
    socket.join(`call:${roomId}`);}
  sendToClients(eventName, payload: unknown): void {
    this.websocketServer.sendToRoom('clients', eventName, payload);}
  sendToUser(userId: string, eventName: string, payload: unknown) {
    this.websocketServer.sendToRoom(userId, eventName, payload);}
  sendToCallRoom(roomId: string, eventName: string, payload: unknown) {
    this.websocketServer.sendToRoom(`call:${roomId}`, eventName, payload);}}
```

В.7 Класи авторизації

```
import { Module } from '@nestjs/common';
import { AuthProcessorService } from './auth-processor.service';
@Module({
  providers: [AuthProcessorService],
  exports: [AuthProcessorService],
})
export class AuthProcessorModule {}
import { Injectable } from '@nestjs/common';
import { UserRepository } from '@db/entities';
import { LoginUserDto } from './dtos';
import { makeUserEntity, UserEntity, UserProvider, UserRole } from
'../../user/entities';
@Injectable()
export class AuthProcessorService {
  constructor(private readonly userRepository: UserRepository) {}
  async handleAttemptLogin({
    user,
    provider,
  }: {
    user: LoginUserDto;
    provider: UserProvider;
  }): Promise<UserEntity> {
    const searchUser = await
this.userRepository.findUserWithProvider(provider.name, provider.id);
    if (searchUser) {
      return makeUserEntity(searchUser);
    }
    return await this.registerUser(user, provider);
  }
  async registerUser(user: LoginUserDto, provider: UserProvider):
Promise<UserEntity> {
    const newUser = await this.userRepository.insertOne({
      firstName: user.firstName,
      lastName: user.lastName,
      providers: [provider],
      avatar: user.avatar,
      role: UserRole.CLIENT,
    });
    return makeUserEntity(newUser);}}
```

```
import { Body, Controller, Get, Param, Post, Req, UseGuards } from
'@nestjs/common';
import { GoogleOauthGuard } from '../google/google-oauth.guard';
import JwtAuthService from '../jwt/jwt-auth.service';
import { Request } from 'express';
import { ApiOperation, ApiResponse, ApiTags } from '@nestjs/swagger';
import { TokenVerificationRequestDto, TokenVerificationResponseDto } from
'./dtos';
import { GoogleOauthService } from './google-oauth.service';
import { ParseObjectIdPipe } from '@common/pipes';
@ApiTags('auth')
@Controller('auth/google')
export default class GoogleOauthController {
  constructor(
    private readonly googleOauthService: GoogleOauthService,
    private readonly jwtAuthService: JwtAuthService,
  ) {}
  @Post()
  @ApiResponse({
    type: TokenVerificationResponseDto,
  })
  async authenticate(@Body() tokenVerificationRequestDto:
TokenVerificationRequestDto) {
    const accessToken = await this.googleOauthService.authenticate(
      tokenVerificationRequestDto.accessToken,
    );
    return {
      data: accessToken,
    };}
  @Get('/:userId')
  @ApiOperation({
    summary: 'Authorize a test user',
  })
  async authenticateTestUser(@Param('userId', ParseObjectIdPipe) userId:
string) {
    return this.jwtAuthService.login({id: userId}).accessToken;
  }}
import { BadRequestException, Injectable } from '@nestjs/common';
import { GoogleOauthClient } from './google-oauth-client';
import { AuthProcessorService } from '../auth-processor/auth-
processor.service';
import { AuthProvider } from '../../user/entities';
import JwtAuthService from '../jwt/jwt-auth.service';
import { AppLogger } from '../../../logger/app-logger';
@Injectable()
export class GoogleOauthService {
  constructor(
    private readonly googleOauthClient: GoogleOauthClient,
    private readonly authProcessorService: AuthProcessorService,
    private readonly jwtAuthService: JwtAuthService,
    private readonly appLogger: AppLogger,
  ) {}
  async authenticate(accessToken: string): Promise<{
    accessToken: string;
  }> {
    let googleProfile;
    try {
      googleProfile = await
this.googleOauthClient.getProfileByToken(accessToken);
    } catch (e) {
      this.appLogger.warn('Failed at getting google profile at
authentication', {
        errorMessage: e.message,
```

```
        accessToken,
      });
      throw new BadRequestException('Token is invalid');}
    try {
      const user = await this.authProcessorService.handleAttemptLogin({
        user: {
          firstName: googleProfile.given_name,
          lastName: googleProfile.family_name,
          avatar: googleProfile.picture,
        },
        provider: {
          name: AuthProvider.GOOGLE,
          id: googleProfile.id,
        },
      });
      return this.jwtAuthService.login(user);
    } catch (e) {
      this.appLogger.error('Failed at handle the user attempt login', {
        accessToken,
        provider: AuthProvider.GOOGLE,
      }, e);
    }}}
import { PassportStrategy } from '@nestjs/passport';
import { Profile, Strategy } from 'passport-google-oauth20';
import { Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { UserRepository } from '@db/entities';
import { AuthProcessorService } from '../auth-processor/auth-
processor.service';
import { AppConfig } from '@common/interfaces';
import { AuthProvider } from '../../user/entities';
@Injectable()
export default class GoogleOauthStrategy extends PassportStrategy(Strategy,
'google') {
  constructor(
    private readonly configService: ConfigService<AppConfig>,
    private readonly userRepository: UserRepository,
    private readonly authProcessorService: AuthProcessorService,) {
    super({
      clientID: configService.get<string>('GOOGLE_OAUTH_CLIENT_ID'),
      clientSecret: configService.get<string>('GOOGLE_OAUTH_CLIENT_SECRET'),
      callbackURL: configService.get<string>('GOOGLE_OAUTH_REDIRECT_URL'),
      scope: ['email', 'profile'],});}
 async validate(accessToken: string, refreshToken: string, profile: Profile){
    return this.authProcessorService.handleAttemptLogin({
      user: {
        firstName: profile.name.givenName,
        lastName: profile.name.familyName,
        avatar: profile.photos.length && profile.photos[0].value,},
      provider: {
        name: AuthProvider.GOOGLE,
        id: profile.id,},});}}
import { Auth, google } from 'googleapis';
import { Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { AppConfig } from '@common/interfaces';
import { AppLogger } from '../../../logger/app-logger';
import { TokenPayload } from 'google-auth-
library/build/src/auth/loginticket';
import { GoogleProfileDto } from './dtos';
@Injectable()
export class GoogleOauthClient {
  private authClient: Auth.OAuth2Client;
```

```
  constructor(
    private readonly configService: ConfigService<AppConfig>,
    private readonly appLogger: AppLogger,) {
    this.createClient();}
  private createClient(): void {
    try {
      this.authClient = new Auth.OAuth2Client({
        clientId: this.configService.get('GOOGLE_OAUTH_CLIENT_ID'),
        clientSecret: this.configService.get('GOOGLE_OAUTH_CLIENT_SECRET')
});
    } catch (e) {
      this.appLogger.error('Failed to create Google oauth client', null,
e);}}
  async getProfileByToken(token: string): Promise<GoogleProfileDto> {
    const userInfoClient = google.oauth2('v2').userinfo;
    this.authClient.setCredentials({
      access_token: token,});
    const userInfoResponse = await userInfoClient.get({
      auth: this.authClient,});
    return <
      { id: string;
        given_name: string;
        family_name: string;
        picture?: string;}
      >userInfoResponse.data;}}
import { JwtService } from '@nestjs/jwt';
import { Injectable } from '@nestjs/common';
import { JwtPayload } from './jwt-auth.strategy';
@Injectable()
export default class JwtAuthService {
  constructor(private readonly jwtService: JwtService) {}
  login(user: any) {
    const payload: JwtPayload = {
      sub: user.id,};
    return {
      accessToken: this.jwtService.sign(payload),};}
  decode(token: string): JwtPayload | null {
    return this.jwtService.decode(token) as (JwtPayload | null) }}
import { PassportStrategy } from '@nestjs/passport';
import { ExtractJwt, Strategy } from 'passport-jwt';
import { Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { UserRepository } from '@db/entities';
import { makeUserEntity } from '../../user/entities';
import { AppConfig } from '@common/interfaces';
export type JwtPayload = { sub: string };
@Injectable()
export default class JwtAuthStrategy extends PassportStrategy(Strategy) {
  constructor(
    configService: ConfigService<AppConfig>,
    private readonly userRepository: UserRepository,
  ) {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: configService.get<string>('JWT_SECRET'),
    });}
  async validate(payload: JwtPayload) {
    const user = await this.userRepository.findById(payload.sub);
    return makeUserEntity(user);
  }}
```