Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Дослідження технологій передачі потокового відео для реалізації стрімінгового веб-сервісу

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-23-1м
_____ / Я. Д. Кулінич /

Керівник кваліфікаційної роботи _____ / А. М. Стрюк /

Економіко-організаційна частина _____ / _____ /

Нормоконтроль _____ / А. М. Стрюк /

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг
2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: магістр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри
_____ А. М. Стрюк
«___» _____ 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-19-1 Кулінич Яну Дмитровичу

Тема: Дослідження технологій передачі потокового відео для реалізації стрімінгового веб-сервісу затверджено наказом по КНУ № ___ від «_» _____ 2024 р.

Термін подання студентом закінченої роботи: «_» _____ 2024 р.

Вихідні дані по роботі: розроблена система повинна забезпечувати стабільну передачу потокового відео в реальному часі, мати механізми адаптивного бітрейту, підтримувати інтеграцію з CDN та базові функції стрімінгового веб-сервісу, такі як обробка запитів користувачів і керування потоками відео.

Зміст пояснювальної записки (перелік питань, що їх треба розробити): проаналізувати літературу щодо сучасних технологій передачі потокового відео, провести огляд існуючих рішень подібних веб-сервісів, спроектувати архітектуру веб-сервісу, реалізувати систему передачі потокового відео з адаптивним бітрейтом, підготувати рекомендації щодо подальшого вдосконалення системи.

Перелік ілюстративного матеріалу: сторінки аналогів, функціональна схема, блок-схеми розроблених алгоритмів та роботи програми, зображення інтерфейсу системи, ER-діаграма бази даних, схеми інтеграції з іншими сервісами та системами.

РЕФЕРАТ

NEXTJS, PROSTGRES, PRISMA, WEBRTC, HLS, DASH, MEDIASOUP, WSS, SOCKET.IO, NESTJS, REDUX TOOLKIT, TYPESCRIPT.

Пояснювальна записка: 108 с., 1 табл., 18 рис., 1 дод., 60 джерел.

Метою кваліфікаційної роботи є дослідження та реалізація технологій передачі потокового відео в реальному часі.

У роботі проаналізовано сучасні технології передачі потокового відео, зокрема DASH, HLS та WebRTC та інші. Після порівняльного аналізу вибрано технологію WebRTC із реалізацією SFU за допомогою бібліотеки Mediasoup, яка забезпечує ефективне управління потоками відео та аудіо. Розроблений веб-сервіс поєднує функції конференц-зв'язку та потокової передачі, зосереджуючи увагу на інтерактивності користувачів.

На серверній стороні використано NestJS із Websocket (Socket.IO) для комунікації в реальному часі, Prisma для роботи з базою PostgreSQL, а також Mediasoup для обробки WebRTC-з'єднань.

Клієнтська частина розроблена на Next.js із використанням Redux Toolkit для управління станом, Tailwind CSS для стилізації та клієнтської бібліотеки Mediasoup для інтеграції WebRTC.

Додаток протестовано на продуктивність і стабільність у реальних умовах, звертаючи увагу мінімізацію затримок та забезпечення якості відео за умов нестабільного мережевого з'єднання.

ABSTRACT

NEXTJS, PROSTGRES, PRISMA, WEBRTC, HLS, DASH, MEDIASOUP, WSS, SOCKET.IO, NESTJS, REDUX TOOLKIT, TYPESCRIPT.

Explanatory note: 108 p., 1 table, 18 fig., 1 pp., 60 sources.

The aim of the qualification work is to research and implement real-time streaming video transmission technologies.

The study analyzes modern streaming video transmission technologies, including DASH, HLS, WebRTC, and others. Following a comparative, WebRTC technology was selected and implemented as an SFU using the Mediasoup library, which provides efficient management of video and audio streams. The developed web service combines the functionality of conferencing and streaming, focusing on user interactivity.

The server side uses NestJS with Websocket (Socket.IO) for real-time communication, Prisma for working with a PostgreSQL database, and Mediasoup for handling WebRTC connections.

The client side is built with Next.js, using Redux Toolkit for state management, Tailwind CSS for styling, and the Mediasoup client library for WebRTC integration.

The application has been tested for performance and stability under real-word conditions, with particular attention to minimizing latency and ensuring video quality under unstable network conditions.

ЗМІСТ

ВСТУП 5

1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ 6

1.1. Критичний аналіз літературних джерел за темою 6

1.2. Актуальність теми кваліфікаційної роботи 8

1.3. Цілі та завдання кваліфікаційної роботи 11

2 РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ 13

2.1. Розробка та докладний опис функціональної схеми програми 13

2.2 Розробка та докладний опис алгоритму роботи програми 16

2.3. Розробка інтерфейсу програми 21

3 РОЗРОБКА БАЗИ ДАНИХ І ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 27

3.1 Аналіз обраної середи програмування 27

3.2 Розробка бази даних 29

4 АНАЛІЗ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ІННОВАЦІЇ 34

4.1 Розрахунок собівартості програмної інновації 34

4.2 Розрахунок ефективності впровадження програмної інновації 36

ВИСНОВОК 39

ПЕРЕЛІК ПОСИЛАНЬ 40

ДОДАТОК А - КОД ПРОГРАМИ 45

ВСТУП

У сучасному світі технології передачі потокового відео та аудіо стали важливою частиною життя. Вони використовуються для розваг, освіти, бізнесу та просто для спілкування. Стрімінгові платформи, такі як Twitch та YouTube, забезпечують можливість трансляції контенту для широкої аудиторії, тоді як сервіси на кшталт Google Meet і Zoom фокусуються на інтерактивному відео- та аудіозв’язку між учасниками в реальному часі.

Розвиток цих технологій значно змінив характер взаємодії в інтернеті. Стрімінгові платформи зробили можливим масштабне масштабне поширення контенту, а конференційні сервіси забезпечили умови для спільної роботи та спілкування. Однак створення сучасних рішень для передачі потокового відео в реальному часі залишається складним завданням, що вимагає інтеграції передових технологій, таких як WebRTC, DASH, HLS тощо.

У цій роботі роботі досліджуються сучасні підходи до передачі потокового відео, що є актуальними для створення масштабованих веб-сервісів реального часу.

Актуальність теми зумовлена великим попитом на багатофункціональні веб-сервіси, що забезпечують високоякісний відео- та аудіозв’язок у релаьному часі з мінімальними затримками. Це робить дослідження в цій галузі важливим і перспективним для подальшого вивчення та реалізації.

1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1. Критичний аналіз літературних джерел за темою

WebRTC [1] (Web Real-Time Communications) - це сучасна технологія, яка дозволяє передавати аудіо та відео в реальному часі через браузер без необхідності встановлення додаткового забезпечення. Вона була вперше представлена у 2011 році і з того часу активно розвивається для застосувань у відеоконференціях, потоковому відео, онлайн-іграх та інших сферах, які потребують зв’язку в реальному часі.

У січні 2021 року WebRTC був офіційно прийнятий як стандарт Світовим консорціумом Всесвітньої павутини (W3C) та Робочою групою з інженерних завдань Інтернету (IETF). Ця стандартизація закріпила WebRTC як ключову технологію для забезпечення стабільності, сумісності та надійності в області відео- та аудіозв’язку в реальному часі. Завдяки цьому WebRTC став універсальним інструментом для створення веб-додатків, що підтримують інтерактивність та високу якість передачі даних.

У ряді літературних джерел надається загальний огляд технології WebRTC, включаючи "Learning WebRTC" [3] та "WebRTC Cookbook" [4]. Ці книги докладно описують архітектуру WebRTC, її протоколи (SRTP [5], ICE [6], DTLS [7]) і принципи роботи, зокрема передачу потокового відео. Однак ці джерела більше фокусуються на технічних аспектах роботи WebRTC, а не на конкретних прикладах його використання. Книга "High Performance Browser Networking" [8], на яку я посилався у попередній кваліфікаційній роботі, також містить розділ про WebRTC, акцентуючи увагу на її мережевих можливостях. Проте, через швидкий розвиток технологій, частина інформації у цьому джерелі вже застаріла.

Окрім WebRTC, для передачі потокового відео часто використовуються й інші технології та протоколи. Наприклад, SRT [9] (Secure Reliable Transport)

є відкритим протоколом, розробленим для передачі відео з низькою затримкою через ненадійні мережі. SRT забезпечує надійність за рахунок корекції втрат даних і адаптації до зміни пропускної здатності мережі. Він активно використовується в трансляціях високої якості, зокрема у спортивних заходах та новинних репортажах. Проте SRT вимагає спеціалізованих програмних або апаратних рішень, що обмежує його використання у веб-браузерах.

Інші технології, такі як DASH [10] (Dynamic Adaptive Streaming over HTTP) і HLS [11] (HTTP Live Streaming), широко застосовуються для адаптивної передачі відео. Вони забезпечують високу якість трансляції завдяки розподілу відео на сегменти і адаптації до доступної пропускної здатності мережі.

У ряді літературних джерел надається загальний огляд цих технологій. Наприклад, дослідження "Comparing video streaming protocols: A comprehensive analysis" [12] розглядає переваги HLS, DASH (та інших) для адаптивного стрімінгу. Проте джерела фокусуються на принципах роботи цих протоколів і не враховують їх обмеження у контексті передачі відео в реальному часі.

На практиці WebRTC активно застосовується у таких рішеннях, як Mediasoup [13], Janus [14] і Jitsi [15], які забезпечують масштабованість і продуктивність для потокової передачі. Офіційна документація Mediasoup [16] та інші ресурси детально описують можливості реалізації WebRTC, але не роглядають інтеграцію в багатофункціональні веб-сервіси.

Таким чином, критичний аналіз доступних літературних джерел показує, що для реалізації такого веб-сервісу необхідно врахувати переваги та обмеження різних технологій, таких як WebRTC, SRT, DASH, HLS, з огляду на специфіку передачі даних в реальному часі. Незважаючи на широкий вибір рішень, саме WebRTC забезпечує оптимальний баланс між функціональністю, масштабованістю та доступністю для веб-додатків, що робить цю технологію найкращим вибором для поставленої задачі.

1.2. Актуальність теми кваліфікаційної роботи

Аналіз предметної галузі "технології передачі потокового відео" можна розглядати з різних точок зору. З точки зору технологій, передача відео в реальному часі є критично важливою складовою сучасних веб-сервісів, які забезпечують інтерактивність, масштабованість та високу якість взаємодії користувачів. Технології, такі як WebRTC, HLS, DASH. відіграють ключову роль у створенні платформ для відеоконференцій, стрімінгових сервісів та інших додатків, що забезпечують передачу аудіо та відео в реальному часі.

З точки зору комунікації, сервіси для відеозв’язку та потокового відео значно змінили спосіб взаємодії між людьми, організаціями тощо. Вони створюють можливість для проведення онлайн-зустрічей, навчання, трансляцій подій і навіть вирішення бізнес-задач, таких як віддалена співпраця.

З точки зору дослідження соціальних процесів, сервіси відеозв’язку є важливим інструментом для аналізу поведінки користувачів, впливу відео контенту на аудиторію та вивчення соціальних трендів. Такі платформи стають джерелом великих обсягів даних, які можуть використовуватися в маркетингових дослідженнях, освітніх програмах та інших галузях.

Аналіз ринку платформ для потокового відео демонструє зростання популярності таких сервісів, як Zoom [17], Google Meet [18], Twitch [19] тощо. Ці сервіси стали ключовими для роботи, навчання, розваг та проведення заходів у цифровому середовищі.

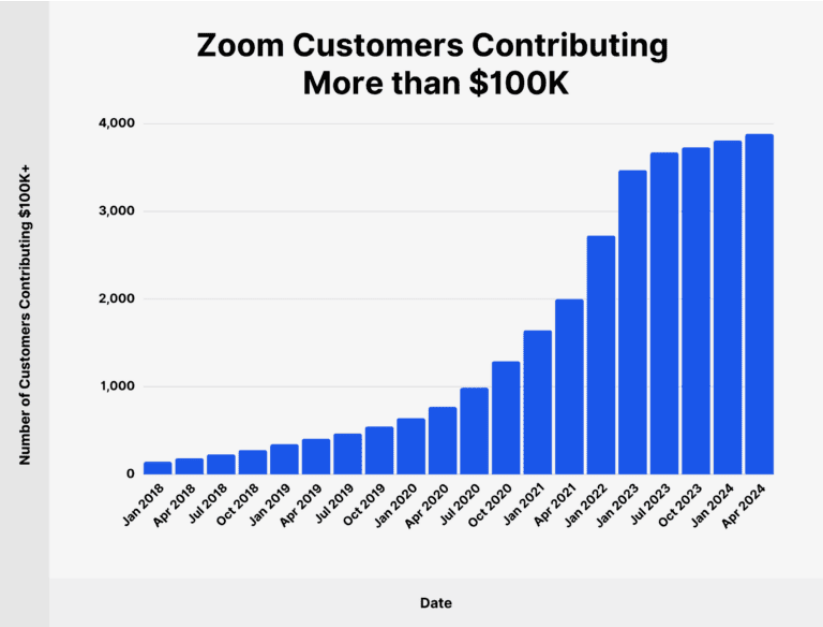

Рисунок 1.1 - Динаміка зростання кількості клієнтів Zoom із внеском біл ніж $100,000

За даними статистики компанії Zoom [20], кількість клієнтів, які щорічно витрачають понад $100,000, зросла з кількох сотень до у 2018 році до понад 4,000 у квітні 2024. Їхній щоквартальний дохід за той же період виріс із менш ніж $100 млн у 2017 році до понад $1,2 млрд у 2024 році. Ця динаміка свідчить про значний попит на сервіси, що надають високу якість відео- та аудіозв’язку в реальному часі.

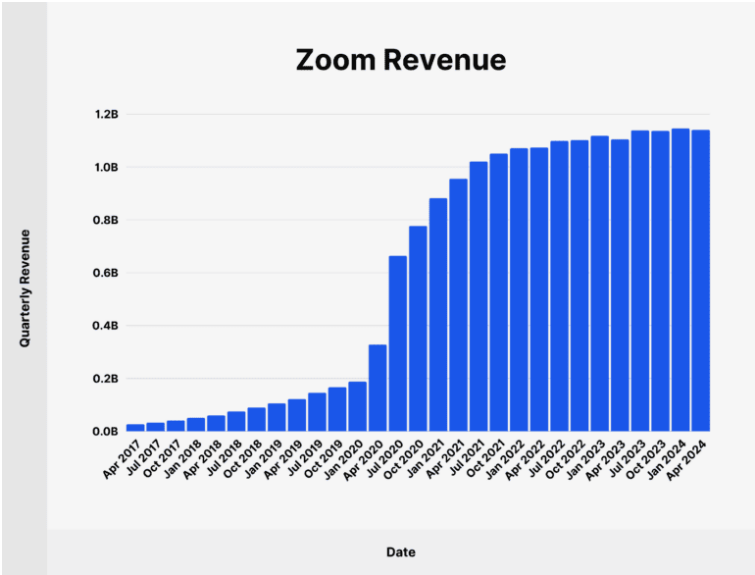

Рисунок 1.2 - Динаміка квартального виторгу Zoom

На ринку стрімінгових сервісів, таких як Twitch, спостерігається не менш вражаюче зростання. Станом на 2023 рік загальна аудиторія Twitch досягла 250 мільйонів зареєстрованих користувачів [21], переважна більшість (приблизно три четверті) віком до 35 років [22]. Це підкреслює інтерес до стрімінгового контенту, а також роль інтерактивності, яку платформа пропонує своїм користувачам.

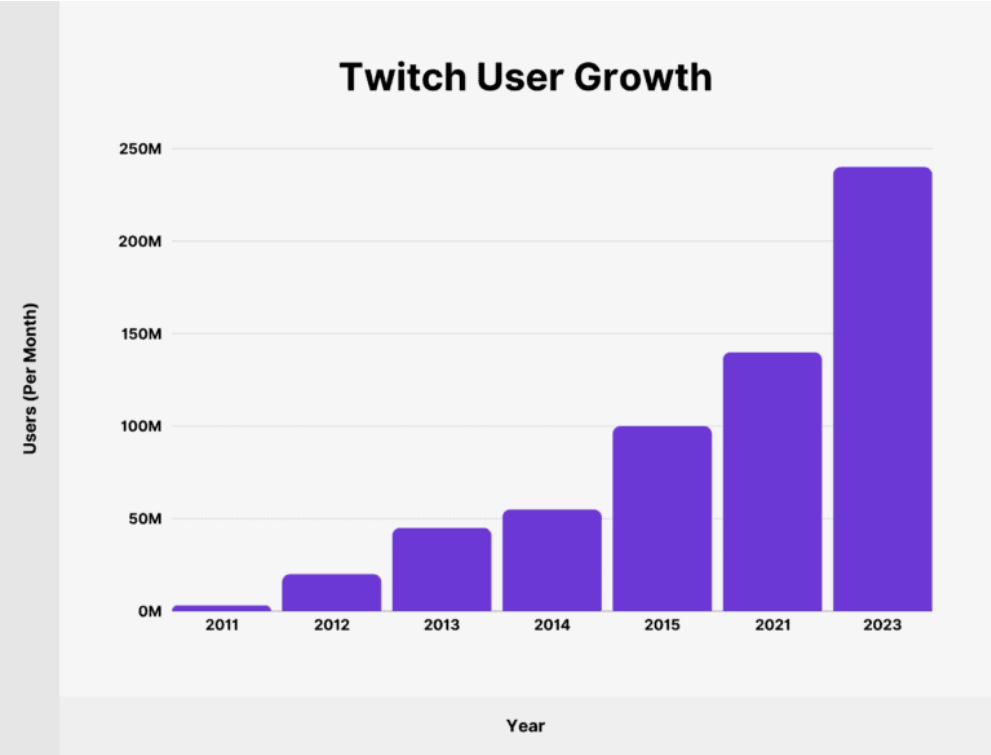

Рисунок 1.3 - Динаміка зростання кількості користувачів Twitch

Актуальність теми дослідження підкреслюється потребою у розробці платформ, які інтегрують найкращі функціональні можливості стрімінгових сервісів і конференційних рішень. Такий підхід відкриває нові можливості для організації інтерактивних онлайн-заходів, навчання та трансляцій подій.

Крім того, розвиток сучасних технологій робить можливим створення платформ, які можуть адаптуватися до умов нестабільної мережі, забезпечуючи низьку затримку та високу якість передачі даних.

Таким чином, актуальність теми кваліфікаційної роботи обумовлена високим попитом на багатофункціональні веб-сервіси, які здатні забезпечувати якісну передачу аудіо та відео, а також інтерактивність у

реальному часі. Це робить дослідження в цій галузі важливим і перспективним для подального вивчення.

1.3. Цілі та завдання кваліфікаційної роботи

Технології передачі потокового відео та відеозв'язку відіграють ключову роль у сучасному цифровому світі. Вони використовуються для організації конференцій, вебінарів, трансляції подій, а також у багатьох інших сферах. Висока якість відео, мінімальні затримки та інтерактивність є ключовими факторами успіху таких платформ.

Метою кваліфікаційної роботи є дослідження сучасних технологій передачі потокового відео, таких як WebRTC, DASH, HLS, і створення веб-сервісу використовуючи ці технології.

Для досягнення цієї мети необхідно виконати наступні завдання:

1. Провести аналіз сучасних технологій передачі потокового відео. Розглянути WebRTC (в реалізаціях MESH, MCU, SFU), DASH, HLS та інші технології з точки зору їх переваг і недоліків для створення веб-сервісів.
2. Визначити функціональні та нефункціональні вимоги до веб-сервісу. Зокрема, вимоги до якості відео, затримки, бітрейту тощо.
3. Спроектувати архітектуру платформи. Розробити архітектуру, яка підтримуватиме, потокову передачу відео, розраховану на велику кількість користувачів і інтерактивні можливості, використовуючи вищезазначені технології передачі потокового відео.
4. Розробити базу даних для зберігання інформації. Створити схему бази даних, яка забезпечить зберігання профілів користувачів, відео- та аудіосесій, чату, статистики та налаштувань.
5. Розробити серверну частину. Використовуючи NestJS [28], як основний фреймворк, Prisma [29] (PostgreSQL [30]). Створити

бекенд для обробки потокового відео та керування взаємодією між користувачами.

6. Розробити клієнтську частину. Створити інтуїтивно зрозумілий інтерфейс використовуючи Next.js [31], Redux Toolkit [32], Tailwind CSS [33], який дозволить користувачам легко організовувати та приєднуватися до трансляцій (або конференцій).
7. Інтегрувати безпекові механізми. Забезпечити захист даних користувачів, захисту від несанкціонованого доступу, а також заходів безпеки для стабільної роботи системи.
8. Тестування системи. Проведення тестів для перевірки працездатності, швидкодії та надійного сервісу. Усунення помилок і оптимізація роботи схеми. Перевірка роботи сервісу в умовах реального навантаження, оцінка якості відео.
9. Оцінка продуктивності та масштабованості. Проведення тестування системи при різних навантаженнях, визначення її здатності обробляти одночасні з’єднання без зниження якості обслуговування.

Ці завдання дозволять розробити веб-сервіс, який відповідає сучасним вимогам до якості, безпеки, забезпечує масштабованість і високу якість передачі даних у реальному часі, а також забезпечує користувачам зручний і ефективний інструмент.

2 РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ

2.1. Розробка та докладний опис функціональної схеми програми

На початковому етапі роботи передбачалося створення веб-сервісу для потокової передачі відео. Основна ідея полягала у впровадженні системи, яка дозволяла б користувачам створювати власні трансляції, подібні до Twitch або YouTube Live [23]. Такий сервіс мав включати наступні основні функції:

1. Підтримка трансляцій у реальному часі:
   1.1. Використання сучасних технологій потокової передачі відео.
   1.2. Низька затримка між транслятором і глядачами.
2. Гнучкість управління доступом:
   2.1. Можливість створювати публічні та приватні трансляції.
   2.2. Інтеграція механізмів аутентифікації та авторизації для безпеки даних.
3. Взаємодія між транслятором і глядачами:
   3.1. Інтерактивні елементи: чати, реакції, голосування тощо.
4. Масштабованість:
   4.1. Забезпечення можливості одночасного обслуговування великої кількості користувачів.

Реалізація та новий вектор розвитку

Під час розробки було впроваджено основну архітектуру для стрімінгово веб-сервісу, яка базувалася на RTMP [24] (Real-Time Messaging Protocol) та HLS.

Продюсер (транслятор) передавав свої відео- та аудіопотоки на сервер через RTMP, який забезпечує стабільну передачу медіаданих із мінімальними

вимогами до мережі. Сервер, у свою чергу, конвертував потоки у формат HL, який розбиває потоки на короткі сегменти (зазвичай 2-10 секунд) і генерує плейлист формату .m3u8 [25].

Це рішення дозволяло масштабувати стрімінг за допомогою CDN [26] (Content Delivery Network), що зменшує навантаження на сервер і забезпечує швидку доставку глядачам по всьому світу. Глядачі отримували відео через HLS та відтворювали його у своїх браузерах.

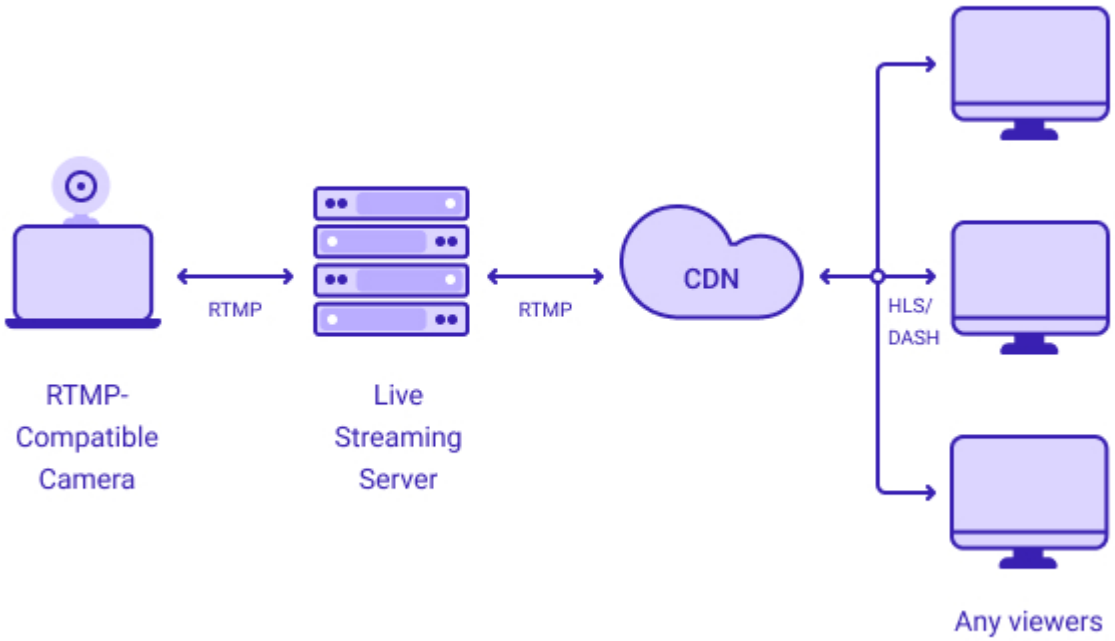

Рисунок 2.1 - Архітектура стрімінгово сервісу на основі RTMP і HLS

Однак, у процесі роботи з'ясувалося, що впровадження базового функціоналу трансляцій є менш цікавим (в технічному плані), ніж передбачалося. Використання RTMP для продюсера та HLS для глядачів добре відоме і не потребує значних інновацій.

У зв'язку з цим було вирішено вирішити більш цікаву проблему - розробці системи відеоконференцій із інтерактивними можливостями, притаманними, наприклад, Google Meet. Для цього було реалізовано новий підхід із використанням WebRTC для передачі медіа-потоків між учасниками конференцій.

Проблематика інтерактивних відеоконференцій.

Розробка системи для інтерактивних відеоконференцій є складним завданням, що пов’язано з кількома ключовими факторами:

1. Двосторонній обмін медіа-потоками: На відміну від стрімінгових сервісів, де один продюсер транслює потік багатьом глядачам, у відеоконференціях кожен учасник є одночасно продюсером (генерує потік) і консюмером (отримує потік від інших учасників). Це створює значне навантаження як на клієнтській пристрої, так і на серверну інфраструктуру.
2. Масштабованість: У міру збільшення кількості учасників конференції зростає кількість переданих потоків. У деяких архітектурах це призводить до експоненційного росту навантаження, що є серйозним викликом для продуктивності системи.
3. Різноманітність клієнтських пристроїв: Учасники можуть використовувати пристрої з різною обчислювальною потужністю (від мобільних пристроїв до потужних комп’ютерів) та мережевими можливостями (від повільного мобільного інтернету (3G-4G) до швидкісного Wi-Fi або гігабітного Ethernet-підключення). Це вимагає адаптивного підходу до передачі потоків, щоб забезпечити належну якість для всіх.
4. Забезпечення низької затримки: Для підтримки природної взаємодії між учасниками затримка має бути мінімальною (менше 200мс). Це виключає використання деяких традиційних протоколів для передачу потоків, таких як HLS, які мають високу затримку через сегментацію.
5. Інтерактивність: Система повинна підтримувати функції, які роблять конференції інтерактивними, такі як текстові чати, реакції, демонстрації екрану, голосування, тощо.

2.2 Розробка та докладний опис алгоритму роботи програми

Для побудови ефективної системи для передачі потокового відео в реальному часі важливо правильно обрати архітектуру WebRTC, яка забезпечить баланс між продуктивністю, масштабованістю та зручністю використання. Рішення залежить від конкретно поставленої задачі та вимог, таких як кількість користувачів, якість відео, вимоги до затримки та доступні ресурси. Зважаючи на вищезазначені виклики, було розглянуто три основні архітектури WebRTC [27]: Mesh, MCU, SFU.

1. Mesh: Усі учасники напряму підключаються один до одного, передаючи потоки через P2P-з’єднання.

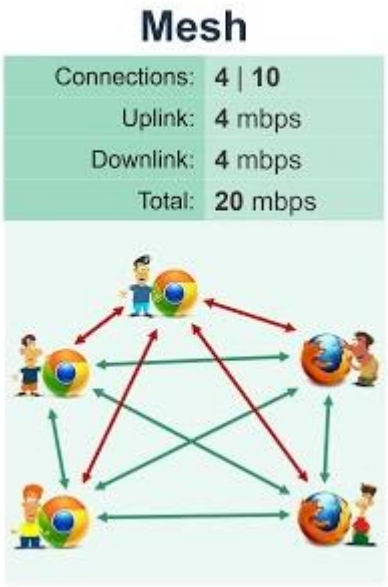

Рисунок 2.2 - WebRTC Mesh архітектура

Проблема: Цей підхід добре працює лише для дуже малих груп (до 3-5 учасників). Для великих конференцій кожен клієнт має обробляти надто багато потоків, що швидко перевантажує обчислювальні ресурси пристрою та мережу.

2. MCU: Усі потоки передаються на сервер, де вони перекодовуються в один комбінований потік, який потім надсилається кожному клієнту.

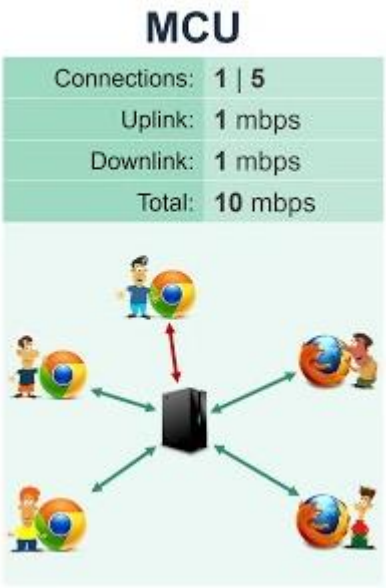

Рисунок 2.3 - WebRTC MCU архітектура

Проблема: MCU створює навантаження на сервер, особливо для великих конференцій, де потоки потрібно перекодовувати в реальному часі. Крім того, це збільшує затримку, що є критични для інтерактивних додатків.

3. SFU: Сервер приймає всі потоки від учасників і пересилає їх без перекодування іншим учасникам.

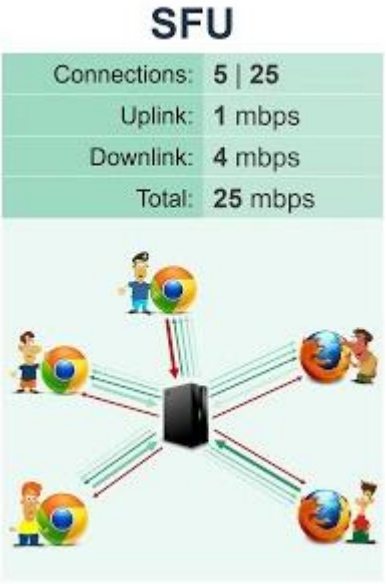

Рисунок 2.4 - WebRTC SFU архітектура

Проблема: Хоча SFU знижує навантаження на сервер, клієнт повинен обробляти кілька потоків одночасно. Для слабких пристроїв із низькою швидкістю мережі це може створювати проблеми.

З огляду на наведені архітектури, мої доступні ресурси та вимоги для реалізації потокового відео в межах інтерактивної конференц-платформи було обрано архітектуру SFU. Вона забезпечує оптимальний баланс між навантаженням на сервер та клієнтськими пристроями, що є критично важливим для масштабованих додатків. Завдяки використанню SFU:

- Затримка залишається мінімальною, оскільки сервер не займається перекодуванням потоків.
- Навантаження на сервер знижується порівняно з MCU, що дозволяє підтримувати більшу кількість одночасних користувачів.
- Користувачі отримують окремі потоки, що дає можливість контролювати якість кожного відео окремо (наприклад, вибирати між HD і SD залежно від пропускної здатності мережі).

Впровадження SFU-архітектури вимагає ретельного проектування серверної та клієнтської частин, зокрема:

- На серверній стороні: необхідно реалізувати маршрутизацію потоків власноруч або за допомогою спеціалізованих бібліотек, таких як Mediasoup або Janus. Ці рішення забезпечують масштабованість та інтеграцію із сучасними WebRTC API [2].
- На клієнтській стороні: потрібно оптимізувати обробку кількох відео- та аудіо потоків для забезпечення плавної роботи навіть на пристроях з обмеженими ресурсами.

Для реалізації ефективної архітектури WebRTC важливу роль відіграють допоміжні сервери, такі як STUN [34] (Session Traversal Utilities for NAT) і TURN [35] (Traversal Using Relays around NAT), а також правильне використання медіа серверів. Вони забезпечують додаткові можливості для передачі потоків, особливо в умовах складних мережевих конфігурацій:

1. STUN: STUN-сервери допомагають визначити публічну IP-адресу клієнта, необхідну для встановлення прямого з’єднання між учасниками (P2P). Це особливо актуально коли клієнти знаходяться за NAT [36] (Network address translation).
    - Приклад: клієнт звертається до STUN-сервера, отримує свою публічну IP-адресу і передає її іншому клієнту для встановлення з’єднання.
    - Обмеження: STUN працює тільки у випадках, коли NAT дозволяє прямий зв’язок.
2. TURN: TURN-сервери виступають як ретранслятор для медіапотоків у тих випадках, коли пряме з’єднання неможливе через жорсткі NAT або корпоративні фаєрволи.
    - Переваги: TURN забезпечує стабільність з’єднання навіть у складних мережах. Потоки передаються через сервер, гарантуючи, що зв’язок буде встановлено.
    - Недолік: TURN значно збільшує затримку та навантаження на сервери через необхідність ретрансляції всіх даних.

Також варто враховувати додаткові аспекти, такі як:

1. Сигнальний сервер: усі архітектури WebRTC потребують сигнального сервера для обміну метаданими між учасниками, такими як SDP [37] (Session Description Protocol) та ICE кандидати [38]. Це ключовий елемент для ініціалізації P2P-з’єднань.
2. Механізми адаптації якості: використання таких технік, як simulcast (передача одного потоку в декількох якостях) та SVC [39] (Scalable Video Coding)
3. Моніторинг та аналітика: для забезпечення стабільності роботи системи важливо впроваджувати інструменти моніторингу, які дозволяють в реальному часі відслідковувати якість з’єднання, затримки та пропускну здатність.

4. Безпека: усі потоки в WebRTC за замовчуванням шифруються за допомогою DTLS і SRTP, що забезпечує захист даних від стороннього втручання.

Хотілось би акцентувати увагу на simulcast і тому для чого він потрібен [40]. Це техніка, при якій один і той же відеопотік відправляється в декількох версіях з різною якістю, а сервер SFU (в даному випадку Mediasoup) може потім пересилати певну версію відеопотоку кожному клієнту в залежності від його пропускної здатності або продуктивності пристрою. Ось алгоритм за яким він працює:

1. Продюсер обирає цільову вихідну роздільну здатність.
2. Simulcast визначає трі різні варіанти бітрейту на основі встановленого алгоритму з використанням цільової вихідної роздільної здатності.
3. Simulcast кодує три потоки, по одному для кожного параметра бітрейту.
4. Кожен альтернативний потік надсилається на сервер SFU. Якщо пропускна здатність між продюсером і SFU занизька для надсилання всіх трьох потоків, вони будуть перекодовані з нижчими бітрейтами.
5. Сервер SFU надсилає один альтернативний потік на кожен пристрій кінцевого користувача на основі доступної пропускної здатності.
6. Користувачі відчувають найкраще поєднання надійності та якості.

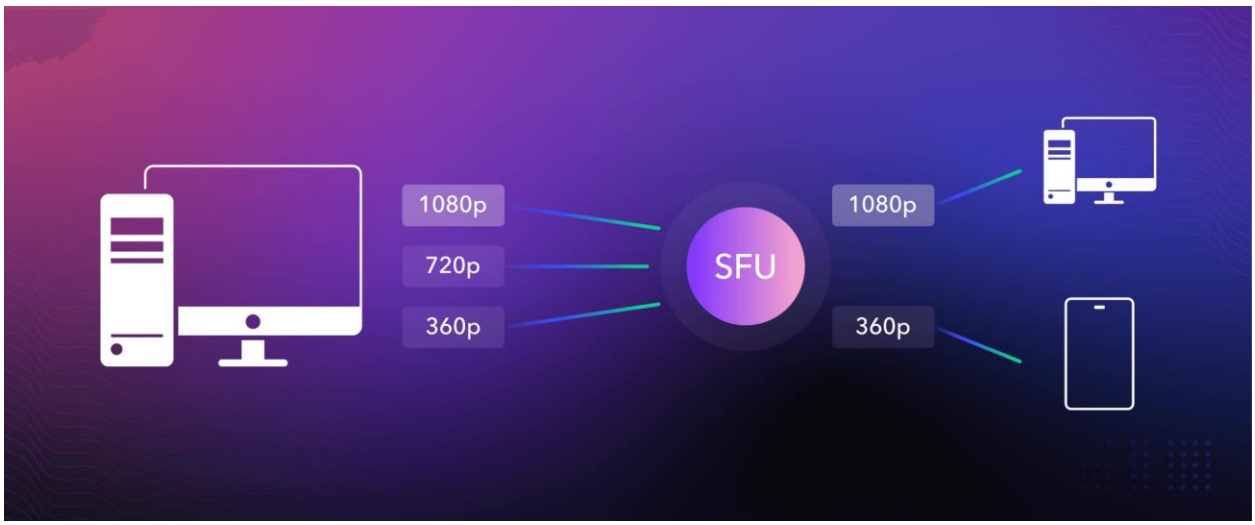

Рисунок 2.5 - Принцип роботи simulcast

У двох словах це дуже простий та ефективний інструмент для адаптивності відео- та аудіо потоків. Simulcast підтримується у всіх сучасних

браузерах через WebRTC API. Найкраще він працює з відеокодеками [41], які підтримують багатошаровість (наприклад, VP8, VP9), а кодек H.264 [42] підтримує simulcast з обмеженнями.

У випадку з mediasoup все ще простіше - simulcast підтримується "з коробки", що значно спрощує реалізацію цієї функціональності. Це забезпечує не лише стабільність з'єднання, але й покращує загальний користувацький досвід. Таким чином, обраний підхід дозволяє створити систему, яка відповідає сучасним вимогам до потокового відео, забезпечуючи високу якість передачі, мінімальну затримку та можливість масштабування для підтримки великої кількості користувачів.

2.3. Розробка інтерфейсу програми

Інтерфейс користувача повинен забезпечувати просте та інтуїтивне керування основними можливостями системи:

1. Навігаційна панель:

Рисунок 2.6 - Навігаційна панель

1.1. Логотип та назва системи: при натисканні користувач повертається на головну сторінку.

1.2. Поле пошуку: користувач може ввести запит у текстове поле для пошуку конференцій, учасників або записів. Праворуч в полі знаходиться іконка лупи, при натисканні на яку виконується пошук.

1.3. Кнопка "Log In": тільки для неавторизованих користувачів, при натисканні відкривається модальне вікно для авторизації.

1.4. Кнопка "Sign Up": тільки для неавторизованих користувачів, при натисканні відкривається модальне вікно для реєстрації.

1.5. Іконка профілю: при натисканні з’являється випадаюче меню, що надає кілька основних опцій

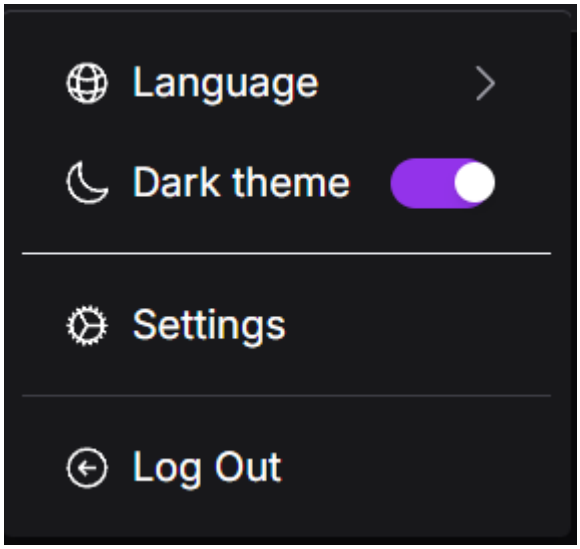

Рисунок 2.7 - Випадаюче меню

1.5.1. Language: при натисканні відкривається підменю з доступними мовами.

1.5.2. Dark theme: перемикач, що дозволяє перемикати інтерфейс між світлою і темною темами. Стан теми зберігається в межах браузера (профілю).

1.5.3. Log In: тільки для неавторизованих користувачів, при натисканні відкривається модальне вікно для авторизації.

1.5.4. Settings: натискання відкриває сторінку налаштувань профілю користувача.

1.5.5. Log Out: натискання виконує вихід із системи та перенаправляє користувача на головну сторінку.

2. Головна сторінка: виконує функцію інформаційного центру, де користувачі можуть переглядати списки категорій конференцій, а також доступні в них поточні та майбутні зустрічі.

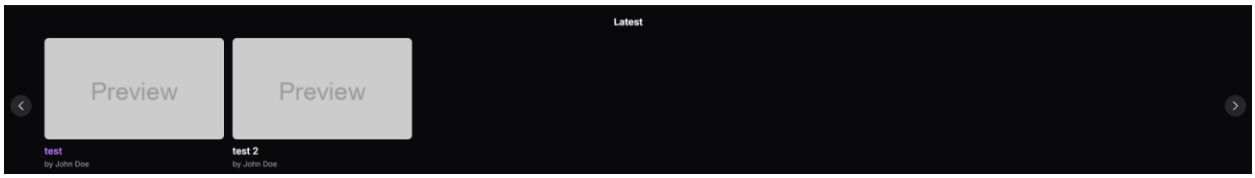

Рисунок 2.8 - Список активних зустрічей на головній сторінці

2.1. При натисканні на зображення або назву зустрічі відкривається сторінка зустрічі, де можна переглянути деталі або приєднатися до неї.

2.2. При натисканні на ім’я користувача, який створив зустріч відбувається навігація на сторінку користувача, де можна побачити зустрічі, які він створював та записи цих зустрічей (якщо вони є і публічно доступні).

2.3. При натисканні на категорію відкривається сторінка зі списком зустрічей за обраною категорією.

3. Авторизація:

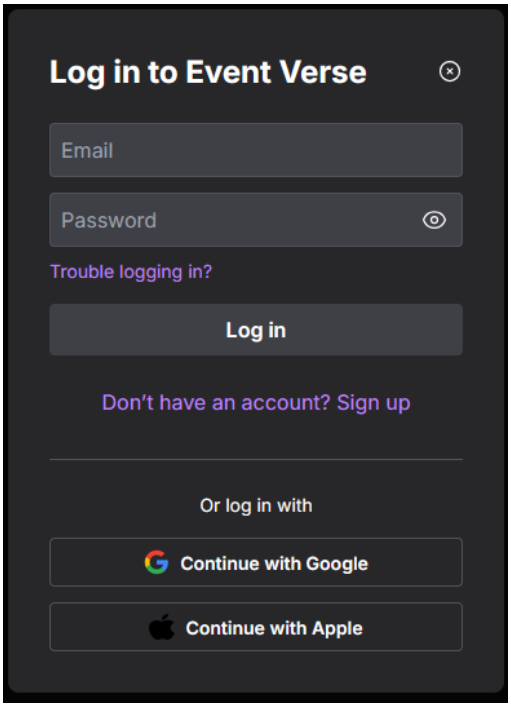

Рисунок 2.9 - Модальне вікно авторизації

3.1. Логін: модальне вікно відкривається при натисканні "Log In" на головній сторінці.

3.1.1. Поля:

3.1.1.1. Електронна пошта або логін.

3.1.1.2. Пароль.

3.1.2. Кнопки:

3.1.2.1. Trouble logging in?: при натисканні відкривається сторінка для відновлення доступу до аккаунту.

3.1.2.2. Log in: доступна лише після заповнення полів. При натисканні відправляє запит на авторизацію.

3.1.2.3. Don`t have an account? Sign up: при натисканні закривається вікно авторизації та відкривається вікно реєстрації.

3.1.2.4. Continue with Google/Apple: авторизація через сторонніх провайдерів.

3.2. Реєстрація: модальне вікно відкривається при натисканні "Sign Up" на головній сторінці. Майже всі поля і кнопки аналогічно з модальним вікном авторизації.

4. Сторінка зустрічі:

4.1. Компонент підготовки: можна переглянути інформацію про зустріч, побачити список учасників (або напис "No one is here yet", якщо на зустрічі ще нікого немає), перевірити стан мікрофону та камери. За допомогою кнопки "Join" можна приєднатися до зустрічі.

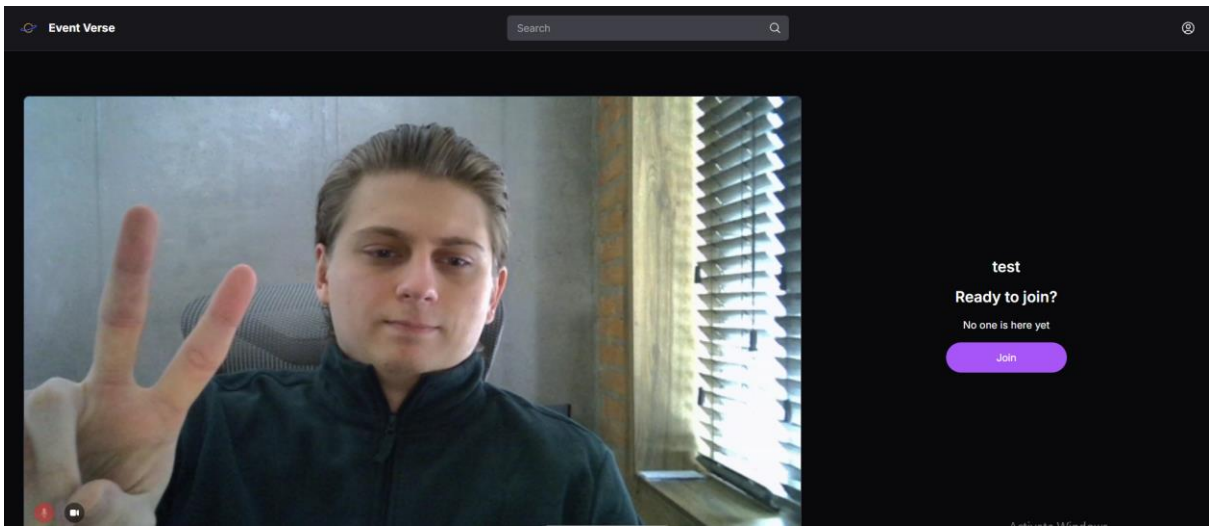

Рисунок 2.10 - Компонент підготовки до зустрічі

4.2. Компонент конференції: відображає список учасників, їх відеопотоки. Дуже важливою складовою тут є панель управління, яка включає в себе наступні іконки:

4.2.1. Мікрофон: при натисканні вмикає або вимикає мікрофон.

4.2.2. Камера: при натисканні вмикає або вимикає камеру.

4.2.3. Екран: при натисканні дозволяє обрати вкладку, вікно або екран для показу іншим учасникам. Після початку показу камера стає недоступною до кінця показу.

4.2.4. Шестерня: представляю собою налаштування, при натисканні відкриває модальне вікно в якому можна обрати діапазон якості відео, обрати вхідні та вихідні пристрої.

4.2.5. Телефон (на червоному фоні): при натисканні користувач покидає зустріч та перенаправляється на компонент підготовки до цієї ж зустрічі. Така логіка роботи зумовлена тим, що іноді користувачі можуть випадково на неї натиснути.

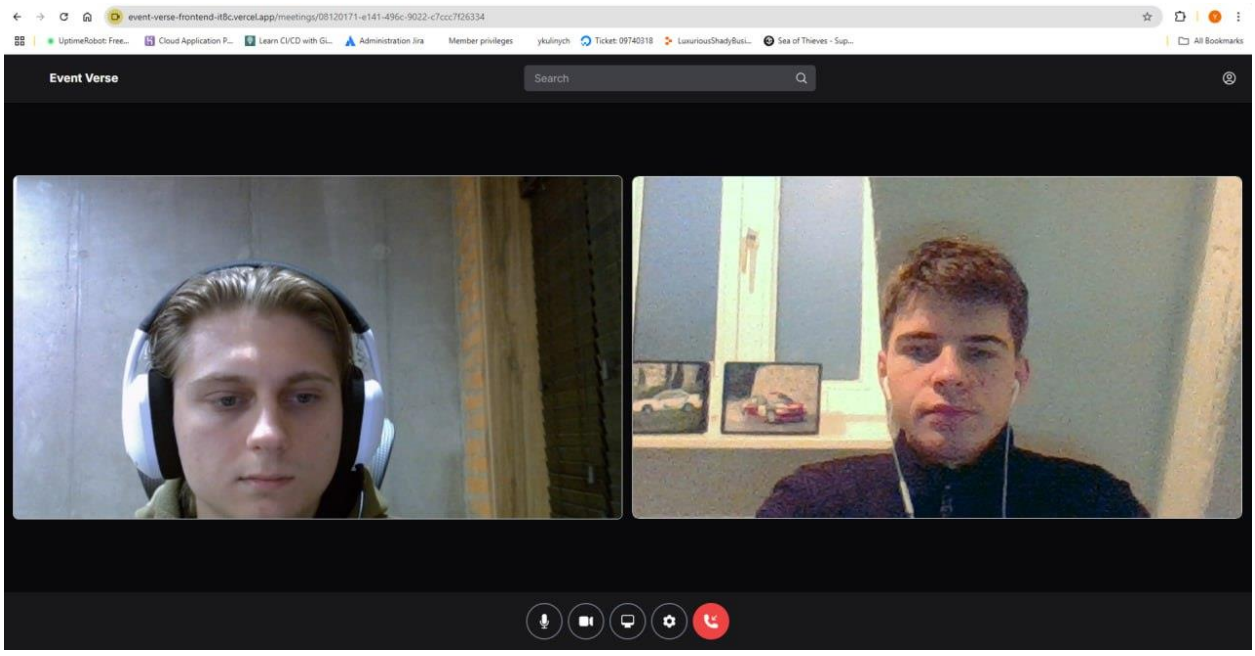

Рисунок 2.11 - Компонент конференції

Окрім цих основних сторінок та компонентів усюди (сторінка зустрічі є виключенням) доступна кнопка у вигляді "плюса", яка відповідає за створення

конференції. При її натисканні відкривається модальне вікно з наступними полями:

1. Meeting Title: назва зустрічі, необов’язкове для заповнення поле.
2. Start Time: дата і час початку зустрічі, обов’зякове для заповнення.
3. End Time: дата і час закінчення зустрічі, необов’язкове.

Рисунок 2.12 - Кнопка відкриття модального вікна для створення зустрічі

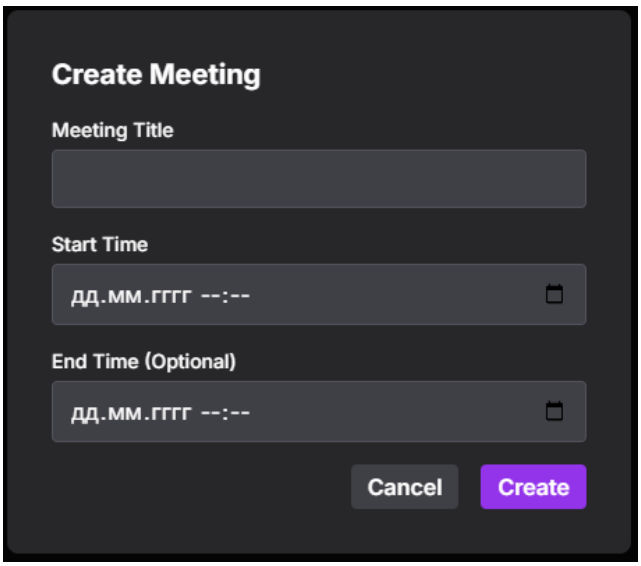

Рисунок 2.13 - Модальне вікно для створення зустрічі

Інтерфейс додатку вже зараз є функціональним і зрозумілим для користувачів завдяки мінімалістичному дизайну, логічній структурі, зрозумілій організації контенту та простоті навігації. Однак впровадження більшої інформативності, адаптивності та нових функцій, таких як фільтрація, різного роду статистика або інтеграції інших сервісів, значно покращить користувацький досвід і зробить платформу більш конкурентоспроможною та привабливою для ширшого кола користувачів.

3 РОЗРОБКА БАЗИ ДАНИХ І ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз обраної середи програмування

При розробці інтерактивного веб-додатку для відеоконференцій постають такі ключові вимоги до технологій:

1. Низька затримка та висока ефективність роботи з потоковими даними.
2. Зручність та простота розробки, легкість рефакторингу, типобезпека та автоматизація процесів.
3. Масштабованість та продуктивність.
4. Простота інтеграції з додатковими модулями та сторонніми сервісами.
5. Підтримка сучасних стандартів веб-розробки.

Враховуючи ці критерії було обрано наступний стек технологій.

Мова програмування та середовище виконання:

1. TypeScript [48]: надбудова над JavaScript, яка забезпечує статичну типізацію. Використання TypeScript дозволяє:
    1.1. Виявити логічні помилки на етапі компіляції.
    1.2. Зрозумілий та самодокументований код через визначення інтерфейсів, типів та класів.
    1.3. Простішу командну розробку, оскільки інші розробники отримують чітку інформацію про типи вхідних та вихідних даних

функцій. Це знижує ризик помилки і полегшує рефакторинг, що є критично важливим у складних додатках.

2. Node.js [49]: середовище виконання JavaScript на стороні сервера. Основні переваги Node.js:
    2.1. Неблокуюча модель вводу/виводу, оптимізована для обробки великої кількості одночасних з'єднань.
    2.2. Велика екосистема пакетів (npm [50]), яка дозволяє швидко інтегрувати сторонні модулі.
    2.3. Велика спільнота та активна підтримка, завдяки чому легко знайти розв'язання складних проблем.
    2.4. Здатність ефективно працювати з real-time технологіями, що є ключовим для вимог додатку.

Система передачі медіаданих:

1. WebRTC та Mediasoup: WebRTC - стандартна технологія для передачі аудіо та відео в реальному часі безпосередньо між браузерами. Для вирішення селективної маршрутизації потоків використовується Mediasoup. Переваги:
    1.1. Низька затримка, що дозволяє створити відчуття присутності всіх учасників в реальному часі.
    1.2. Гнучкість: можна легко додавати або видаляти учасників, змінювати якість потоку, керувати конференціями з багатьма користувачами.
    1.3. Масштабованість: Mediasoup оптимізує маршрутизацію потоків, не перекодовуючи їх, заощаджуючи ресурси процесора та покращуючи продуктивність.

Для реалізації серверної сторони я обрав NestJS [51], який підтримує TypeScript "з коробки" та пропонує добре структуровану архітектуру, має вбудовані засоби для інтеграції з системами безпеки та іншими сервісами. До переваг ще варто додати модульну архітектуру, яка полегшує масштабування

та розподіл коду на окремі функціональні блоки, має зрозумілу файлово-каталогову структуру.

Бібліотека для стану та комунікацій в реальному часі:

1. Redux Toolkit: надбудова над Redux для спрощення керування станом клієнтської частини. Дозволяє зосередитися на логіці, мінімізуючи шаблонний код і помилки.
2. Socket.IO: для сигналізації та передачі допоміжних даних у режимі реального часу між клієнтом та сервером, крім медіапотоків, що передаються по WebRTC. Socket.IO забезпечує двонапрямлене, низько-латентне, подіє-орієнтоване з’єднання між клієнтом і серрвером та суттєво полегшує керування подіями та забезпечує механізми для автоматичного перепідключення та масштабування.

Для реалізації клієнтської частини найкращим варіантом є Next.js - фреймворк для React, який покращує продуктивність, SEO та швидкість завантаження сторінок. До його переваг можна додати простоту маршрутизації та організації коду, статичний рендерінг та простоту інтеграції з екосистемою React.

Обраний стек технологій легко горизонтально масштабувати, додаючи нові вузли та балансуючи навантаження між ними. Така конфігурація задовольняє вимоги до продуктивності, інтерактивності, масштабованості та надійності, забезпечуючи водночас зручний та структурований підхід до розробки, підтримки та оновлення системи.

3.2 Розробка бази даних

При проектуванні бази даних були враховані ключові аспекти додатку:

1. Зберігання інформації про користувачів: профільні дані, аватар, інформація про використані способи авторизації (OAuth).

2. Гнучкість у підтримці декількох способів авторизації: інтеграція через сторонніх постачальників (Google, Apple, GitHub) або власний логін через email і пароль.
3. Збереження інформації про сесії: управління токенами доступу, термінами їх дії, відновлення доступу.
4. Збереження чат-повідомлень, історії інтерактивних дій (дошка, реакції) та допоміжних даних.
5. Забезпечення референційної цілісності та можливості масштабування: таблиці та зв’язки спроектовано для уникнення дублювання даних та помилок, забезпечення унікальності користувачів та зв’язків із постачальниками авторизації.

Для розробки бази даних було обрано PostgreSQL як основну СКБД та Prisma як ORM [43] для взаємодії з базою даних. Цей вибір був обумовлений такими перевагами та недоліками:

Переваги PostgreSQL:

1. Розширюваність: PostgreSQL підтримує складні типи даних (JSON, XML, hstore), користувацькі функції тощо, що робить її гнучкою для складних додатків.
2. Референційна цілісність: завдяки підтримці зовнішніх ключів, обмежень унікальності та каскадних операцій забезпечується висока якість даних.
3. Масштабованість: підтримка шардінгу [44], реплікації та високої доступності.
4. Активна спільнота: PostgreSQL має широку підтримку в спільноті розробників і регулярні оновлення.

Недоліки PostgreSQL:

1. Ресурсозатратність: У порівнянні з іншими СКБД, такими як MySQL, PostgreSQL може бути більш вимогливою до ресурсів при великих навантаженнях. Але цей недолік компенсується масштабованістю за домогою додаткових інструментів, таких як Patroni [45] або інших.

Переваги Prisma:

1. Типізація: Prisma генерує TypeScript-типи на основі схеми бази даних, що забезпечує статичну перевірку та знижує кількість помилок під час розробки.
2. Автоматизація: генерація CRUD-операцій спрощує створення запитів і знижує кількість шаблонного коду.
3. Міграції бази даних: Prisma має вбудовану підтримку міграцій, що дозволяє легко оновлювати структуру бази даних.
4. Підтримка складних запитів: завдяки Prisma Client можно легко створювати запити з реляційними зв’язками без потреби писати SQL вручну.
5. Підтримка views [46] (представленнями): на даний момент Prisma лише частково підтримує представленнями [47] і деякі специфічні функції можуть потребувати додаткової роботи з SQL, але це вже набагато полегшує взаємодію зі складними агрегованими даними.

Недоліки Prisma:

1. Продуктивність: Для складних запитів Prisma може бути повільнішою, оскільки генерує проміжний SQL-код.
2. Обмежена підтримка складних SQL-функцій: у деяких випадках складні функції PostgreSQL (або іншої СКБД) не можна реалізувати через Prisma. Цей недолік пов’язаний з тим, що Prisma є кросплатформеною, що ускладнює реалізацію одних і тих самих функцій у різних СКБД.

В результаті аналізу були сформовані основні сутності системи, які описують її логіку та забезпечують зручну роботу з даними:

1. User (Користувач): зберігає базову інформацію про користувача.
   1.1. id (PK) - унікальний ідентифікатор користувача.
   1.2. email (унікальний) - електронна пошта користувача.
   1.3. password - хешований пароль (тільки для користувачів, які авторизовуються через email/password)

1.4. full_name - повне ім’я користувача.

1.5. avatar - URL до аватару користувача (опціонально).

1.6. created_at, updated_at - часові мітки створення та останнього оновлення профілю.

2. AuthProvider (Постачальник авторизації): зберігає інформацію про способи авторизації, які використовує користувач.

2.1. email - електронна пошта, зареєстрована у постачальника.

2.2. provider - тип постачальника (наприклад, Google, Apple, GitHub).

2.3. provider_id - унікальний ідентифікатор користувача у стороннього постачальника.

2.4. access_token - токен доступу для взаємодії з API постачальника.

2.5. refresh_token - токен для оновлення доступу.

2.6. created_at - час створення запису.

2.7. user_id - ідентифікатор користувача. зв’язок із користувачем, який використовує цього постачальника.

2.8. [provider, provider_id] - ідентифікатор таблиці.

2.9. [email, provider] - унікальність пари.

2.10. [user_id, provider, provider_id] - унікальність пари.

3. Session (Сесія): зберігає інформацію про активні сесії користувачів.

3.1. id (PK) - унікальний ідентифікатор сесії.

3.2. token (unique) - токен доступу для авторизації запитів.

3.3. expires_at - час закінчення дії сесії.

3.4. created_at, updated_at - часові мітки створення та останнього оновлення сесії.

3.5. user_id - ідентифікатор користувача. зв’язок із користувачем, для якого створена сесія.

3.6. [auth_provider, auth_provider_id] - зв’язок із постачальником авторизації, якщо сесію створено через OAuth.

4. Meeting (зустріч):

4.1. id (PK) - унікальний ідентифікатор зустрічі.

4.2. title - назва зустрічі.

4.3. start_time - дата і час початку зустрічі.

4.4. end_time - дата і час закінчення зустрічі.

4.5. created_at, updated_at - часові мітки створення та останнього оновлення зустрічі.

5. Category (категорія):

5.1. id (PK) - унікальний ідентифікатор категорії.

5.2. name - назва категорії

5.3. created_at, updated_at - часові мітки створення та останнього оновлення категорії.

6. MeetingCategory (m2m):

6.1. meeting_id - ідентифікатор зустрічі. зв’язок з мітингом, якому присвоєна категорія.

6.2. category_id - ідентифікатор категорії. зв’язок із категорією, яку присвоюють мітингу.

6.3. created_at - часова сітка створння зв’язку.

7. BoardAction (Дії на дошці):

7.1. id (PK) - унікальний ідентифікатор дії.

7.2. action_type - тип дії (наприклад, "draw", "erase", "text" тощо).

7.3. payload - JSON з координатами, кольорами, текстом.

7.4. timestamp - час виконання дії.

7.5. meeting_id - ідентифікатор зустрічі. зв’язок з мітингом, якому належить дія.

7.6. user_id - ідентифікатор користувача. зв’язок із користувачем, якому належить дія.

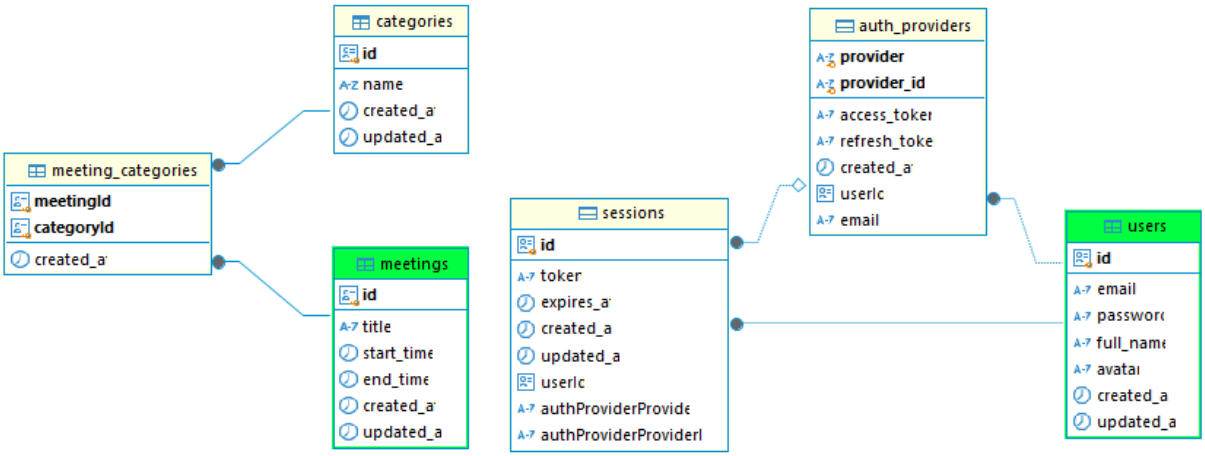

Рисунок 3.1 - ER діаграма основних сутностей системи

Поєднання PostgreSQL та Prisma дозволило реалізувати складну структуру бази даних із підтримкою реляційних зв’язків, референційної цілісності. При цьому Prisma спростила розробку та забезпечила безпеку даних завдяки типізації. Однак у майбутньому, для покращення продуктивності, може виникнути потреба у використанні чистого SQL у вузьких місцях.

4 АНАЛІЗ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ІННОВАЦІЇ

4.1 Розрахунок собівартості програмної інновації

Собівартість розробки програмного забезпечення складається з таких основних компонентів:

1. Витрати на оплату праці розробників.
2. Витрати на інфраструктуру: сервери, хостинги, інструменти.
3. Витрати на ліцензії та додаткові програмні засоби.
4. Витрати на тестування та забезпечення якості.

Враховуючи специфіку та складність проекту команда розробників складається має складатися з:

Таблиця 4.1- Команда розробників

Роль	Кількість	Зарплата на місяць	Зайнятість
Backend-розробник	2	$2700	100%
Frontend-розробник	2	$2300	100%
DevOps-інженер	1	$2800	100%

Отже приблизні загальні витрати на оплату праці на місяць складають приблизно 2 * 2700 + 2 * 2300 + 2800 * 0.5 = $11400. Орієнтовні заробітні плати згідно обраного стеку були взяті з Djinni [53].

Розрахувати витрати на інфраструктуру набагато складніше, оскільки для продакшен (production) середовища важливо врахувати механізми динамічного масштабування серверів. Це дозволяє уникнути перевантаження окремих серверів, забезпечити високу доступність сервісу та якість передачі даних у реальному часі. Враховуючи це я б додав 20% до собівартості серверів для правильності розрахунків. Враховуючи важливість стабільної роботи системи для продакшен середовища потрібно:

1. Основний сервер та сервер сигналізації: буде достатньо простого сервера, який може витримувати велику кількість з’єднань. $48/mo
2. Сервер для сервісу аторизації. $24/mo
3. Медісервер: найбільше навантаження буде саме на цей сервер, отже тут потрібен найкращий. 192$/mo

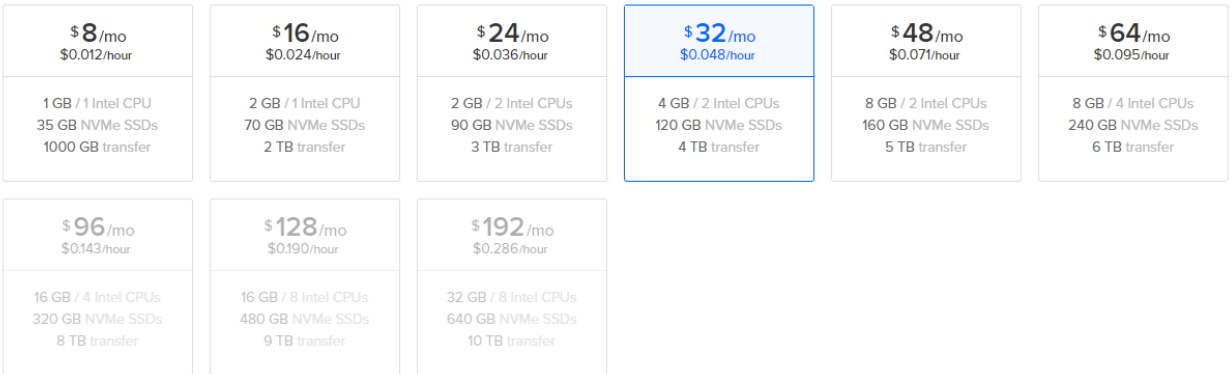

Рисунок 4.1 - Вартість серверів DigitalOcean

4. Домен: перший рік на умовному namecheap [54] коштує близько $20.
5. Бази даних: на перший час - безкоштовно (в Docker [55]), не має сенсу оскільки навантаження на базу не будуть великими.
6. Інструменти для CI/CD:
    6.1. GitHub Actions - безкоштовно (до певних лімітів).
    6.2. DockerHub [56]: платний план $10/mo.
    6.3. Container Registry: Basic плану будет достатньо (на початку) - $5/mo.
7. Моніторинг та логування: Grafana, Prometheus, ElasticSearch, Kibana - всі ці інструменти можна використовувати безкоштовно.

Враховуючи, що основні витрати у інфраструктурі припадають на сервери - їх можна скоротити до нуля скориставшись AWS Activate [57] або аналогічними сервісами, а отже і сумарні витрати на інфраструктуру без витрати коштів на сервери будуть приблизно $20 на місяць.

До витрат на ліцензії та додаткові інструменти можна віднести:

1. GitHub (GitHub Teams): $4/користувач * 5 користувачів = $20/mo
2. Jira: безкоштовно до 10 користувачів.

Отже в цій категорії витрати всього $20 на місяць.

Для тестування ж досталь безкоштовних інструментів (таких як Jest, Puppeteer), тому в цій категорії нуль витрат. У разі необхідності можна замовити ручне тестування, але його я поки не буду враховувати у витратах.

Підводячи підсумки приблизна вартість розробки на місяць приблизно $11500.

4.2 Розрахунок ефективності впровадження програмної інновації

При розробці Для оцінки економічної ефективності впровадження веб-сервісу розглянемо основні показники:

1. Очікуваний дохід від впровадження.
2. Період окупності (Payback Period, PP).
3. Рентабельність інвестицій (Return on Investment, ROI).

Для аналізу економічної ефективності впровадження веб-сервісу розглянуто модель, у якій основна функціональність сервісу є безкоштовною для індивідуальних користувачів. Генерація доходів забезпечується через:

1. Корпоративні підписки (B2B модель) [58].
2. Інтеграція реклами від сторонніх рекламодавців для індивідуальних користувачів без підписки (лише на ранніх етапах).
3. Комісія за доходи від інтегрованих сервісів (наприклад, продаж платних квитків на події, доступ до преміального контенту).

Передбачено, що корпоративні клієнти, які потребують розширеного функціоналу, зокрема аналітики зустрічей і розширеної безпеки, сплачуватимуть $50 на місяць за команду до 100 осіб. Очікувана кількість корпоративних підписок у перший рік = 20. Отже річний дохід становить: $50 * 12 місяців * 20 підписок = $12000 на рік.

Інтеграція реклами це не найкращий спосіб отримувати дохід для стартапу який хоче конкурувати з гігантами, але його варто врахувати. Безкоштовні користувачі під час відеозустрічей або в інтерфейсі можуть бачити рекламу, яку розміщують сторонні рекламодавці. Вартість реклами розраховується за моделлю CPM (вартість за 1000 показів). Середній CPM = $5. Очікувана кількість переглядів реклами: 50000 на місяць. Річний дохід від реклами: 50000 показів * $5 / 1000 * 12 місяців = $3000 на рік.

Сервіс дозволяє організаторам заходів продавати квитки на події черех інтегровані сервіси. Комісія платформи становить 5% від доходу. Очікуваний

обсяг продажу квитків - 10000 на рік. Середня ціна квитка - $10. Річний дохід від комісій: 10000 квитків * $10 * 5% = $5000 на рік.

Склавши всі джерела доходу отримуємо $20000 у перший рік. Період окупності визначаємо за формулою:

□□ = □/□,

де А - собівартість розробки,

В - щорічний дохід.

Для розробки MVP [59] потрібно хоча б 3 місяці. Отже собівартість буде приблизно $34500, але варто врахувати що для залучення нових корпоративних клієнтів потрібно розробляти новий функціонал, тому я додам принаймні ще 10% до собівартості. Отже А = $37950, а

$$□□ = \frac{37950}{20000} = 1.9 \text{ року або приблизно 23 місяці.}$$

Тепер розрахуємо рентабельність інвестицій [60]:

$$□□□ = \frac{□ - □}{□} \times 100\% = \frac{20000 - 37950}{37950} \times 100\% = -47.3\%$$

Рентабельність інвестицій у перший рік становить -47.3%, що свідчить про початковий період збитковості проекту. Однак подальше розширення функціоналу, зростання корпоративних підписок та збільшення користувацької бази дозволить досягти позитивної рентабельності вже у наступні роки. Але також потрібно не забувати, що для збільшення прибутки необхідно рекламувати проект, що теж коштує грошей.

Аналіз економічної ефективності показал, що для розробки MVP веб-сервісу необхідно $37950. Очікуваний річний дохід у перший рік становить $20000 що дозволяє окупити інвестиції за 1.9 року, тобто 23 місяці. У перший рік проект працюватиме зі збитковістю (ROI = -47.3%), але з розвитком платформи очікується приріст доходів через розширення бази корпоративних клієнтів, збільшення попиту на інтерактивні платні події та зростання активності індивідуальних користувачів, що дозволить підвищити економічну ефективність.

З огляду на довгострокові перспективи, впровадження цього проекту є доцільним, оскільки він має потенціал стати конкурентоспроможним рішенням у сфері інтерактивних сервісів. Завдяки чіткому плану розвитку та поступовому вдосконаленню функціоналу проект може стати рентабельним у довгостроковій перспективі, адже пропонує універсальне рішення для організації платних заходів.

ВИСНОВОК

У процесі виконання кваліфікаційної роботи було проаналізовано сучасні технології, розроблено ефективний прототип системи та оцінено її доцільність. Результати дослідження та розробки підтвердили актуальність обраної теми, оскільки стрімінгові сервіси та платформи для відеозв’язку сьогодні є невід’ємною частиною цифрового світу.

У роботі було проаналізовано популярні технології для передачі відео в реальному часі, такі як WebRTC, DASH, HLS, SRT та інші. Проведено порівняльний аналіз їх переваг і недоліків. У результаті WebRTC було обрано як основну технологію для забезпечення низької затримки, високої якості аудіо- та відеозв'язку та можливості масштабування. Реалізацію WebRTC забезпечено за допомогою Mediasoup, що виступає як SFU, оптимізуючи маршрутизацію потоків та мінімізуючи затримки.

Дослідження показало, що WebRTC є найоптимальнішою технологією для передачі відео в реальному часі, враховуючи вимоги до низької затримки, якості передачі даних і масштабованості. Розроблений прототип демонструє можливість створення конкурентоспроможного продукту, який може знайти своє місце на ринку. Проект має потенціал для розвитку у сфері освіти, бізнесу та розваг, що забезпечує його довгострокову актуальність і можливість масштабування.

Результати роботи можуть бути використані для створення нових стрімінгових сервісів та платформ відеозв’язку. Створена архітектура системи є модульною та легко масштабованою, що дозволяє інтегрувати нові функції без значних витрат.

ПЕРЕЛІК ПОСИЛАНЬ

1. WebRTC [Електронний ресурс] – Режим доступу до ресурсу: https://webrtc.org
2. WebRTC API [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API
3. Ristic D. Learning WebRTC. Birmingham: Packt Publishing, 2015, 186 p.
4. Sergiienko A. WebRTC Cookbook. Birmingham: Packt Publishing, 2015, 230 p.
5. Secure Real-Time Transport Protocol [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Secure_Real-time_Transport_Protocol
6. Ineractive Connectivity Establishment [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Interactive_Connectivity_Establishment
7. Datagram Transport Layer Security [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Datagram_Transport_Layer_Security
8. Grigorik I. High Performance Browser Networking: What every web developer should know about networking and web performance. Sebastopol: O`Reilly Media, 2013, 398 p.
9. Secure Reliable Transport [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Secure_Reliable_Transport
10. DASH [Електронний ресурс] – Режим доступу до ресурсу: https://www.mpeg.org/standards/MPEG-DASH
11. HLS [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/http-live-streaming/hls-authoring-specification-for-apple-devices
12. Comparing video streaming protocols: A Comprehensive analysis [Електронний ресурс] – Режим доступу до ресурсу:

https://sendbird.com/developer/tutorials/comparing-video-streaming-protocols-a-comprehensive-analysis

13. mediasoup [Електронний ресурс] – Режим доступу до ресурсу: https://mediasoup.org/
14. Janus - General purpose WebRTC server [Електронний ресурс] – Режим доступу до ресурсу: https://janus.conf.meetecho.com/docs
15. Intoduction | Jitsi Meet [Електронний ресурс] – Режим доступу до ресурсу: https://jitsi.github.io/handbook/docs/architecture
16. mediasoup :: Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://mediasoup.org/documentation/
17. One platform to connect | Zoom [Електронний ресурс] – Режим доступу до ресурсу: https://www.zoom.com/en
18. Google Meet [Електронний ресурс] – Режим доступу до ресурсу: https://meet.google.com/landing
19. Twitch [Електронний ресурс] – Режим доступу до ресурсу: https://www.twitch.tv/
20. Zoom User Stats: How Many People Use Zoom in 2024? [Електронний ресурс] – Режим доступу до ресурсу: https://backlinko.com/zoom-users
21. Twitch Usage and Growth Statistics: How Many People Use Twitch? [Електронний ресурс] – Режим доступу до ресурсу: https://backlinko.com/twitch-users
22. Twitch users by age 2024 [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/634057/twitch-user-age-worldwide
23. YouTube [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/live
24. Adobe RTMP Specification [Електронний ресурс] – Режим доступу до ресурсу: https://rtmp.veriskope.com/docs/spec
25. M3U [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/M3U

26. Content delivery network [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Content_delivery_network
27. The multiple faces of WebRTC N-peer calling: Mesh, MCU, SFU [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/christosalexiou/the-multiple-faces-of-webrtc-n-peer-calling-mesh-mcu-and-sfu-39dg
28. NestJS - A progressive Node.js framework [Електронний ресурс] – Режим доступу до ресурсу: https://nestjs.com/
29. Prisma | Simplify working and interacting with databases [Електронний ресурс] – Режим доступу до ресурсу: https://www.prisma.io/
30. PostgreSQL: The world`s most advanced open source database [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/
31. Next.js by Vercel - The React Framework [Електронний ресурс] – Режим доступу до ресурсу: https://nextjs.org/
32. Redux Toolkit [Електронний ресурс] – Режим доступу до ресурсу: https://redux-toolkit.js.org/
33. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML [Електронний ресурс] – Режим доступу до ресурсу: https://tailwindcss.com/
34. STUN [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/STUN
35. Traversal Using Relays around NAT [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Traversal_Using_Relays_around_NAT
36. Network address translation [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Network_address_translation
37. SDP - MDN Web Docs Glossary: Definitions of Web-related terms [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/SDP

38. RTCIceCandidate - Web APIs [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/RTCIceCandidate
39. Scalable Video Coding (SVC) Extension for WebRTC [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/TR/webrtc-svc/
40. WebRTC Simulcast: What It Is and How It Works [Електронний ресурс] – Режим доступу до ресурсу: https://www.wowza.com/blog/webrtc-simulcast-what-it-is-and-how-it-works
41. Web video codec guide - Web media technologies [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Video_codecs
42. Encode/H.264 - FFmpeg [Електронний ресурс] – Режим доступу до ресурсу: https://trac.ffmpeg.org/wiki/Encode/H.264
43. Object-relation mapping [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping
44. Understanding partitioning and sharding in Postgres and Citus [Електронний ресурс] – Режим доступу до ресурсу: https://techcommunity.microsoft.com/blog/adforpostgresql/understanding-partitioning-and-sharding-in-postgres-and-citus/3891629
45. Introduction - Patroni 4.0.4 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://patroni.readthedocs.io/en/latest/
46. What is a Materialized View? - Materialized View Explained - AWS [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/materialized-view/
47. How to include views in your Prisma schema | Prisma Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.prisma.io/docs/orm/prisma-schema/data-model/views
48. TypeScript: JavaScript With Syntax For Types [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/

49. Node.js - Run JavaScript Everywhere [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en
50. npm | Home [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/
51. NestJS - A Progressive Node.js framework [Електронний ресурс] – Режим доступу до ресурсу: https://nestjs.com/
52. npm | Home [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/
53. Статистика зарплат - Djinni [Електронний ресурс] – Режим доступу до ресурсу: https://djinni.co/salaries/
54. Namecheap - Domains & Hosting [Електронний ресурс] – Режим доступу до ресурсу: https://www.namecheap.com
55. postgres - Official Image | Docker Hub [Електронний ресурс] – Режим доступу до ресурсу: https://hub.docker.com/_/postgres
56. Picing | Docker [Електронний ресурс] – Режим доступу до ресурсу: https://www.docker.com/pricing/
57. Build Your Startup [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/startups#start
58. Business-to-Business (B2B): What It Is and How It`s Used [Електронний ресурс] – Режим доступу до ресурсу: https://www.investopedia.com/terms/b/btob.asp
59. Minimum Viable Product - What is a MVP and why is it important? [Електронний ресурс] – Режим доступу до ресурсу: https://www.productplan.com/glossary/minimum-viable-product/
60. ROI [Електронний ресурс] – Режим доступу до ресурсу: https://www.investopedia.com/articles/basics/10/guide-to-calculating-roi.asp

ДОДАТОК А - КОД ПРОГРАМИ

event-verse-backend/src/common/decorators/auth/auth.guard.ts

```
import { CanActivate, Type } from '@nestjs/common';
import { UserAuthGuard } from './user';

export class AuthGuard {
  static USER: Type<CanActivate> = UserAuthGuard;
}
```

event-verse-backend/src/common/decorators/auth/auth.guard.ts

```
import { CanActivate, ExecutionContext, Injectable } from
'@nestjs/common';
import { Reflector } from '@nestjs/core';
import { PrismaService } from
'../../../modules/prisma/prisma.service';

@Injectable()
export class UserAuthGuard implements CanActivate {
  constructor(
    protected reflector: Reflector,
    protected prismaService: PrismaService,
  ) {}

  async canActivate(context: ExecutionContext): Promise<boolean>
{
    const request = context.switchToHttp().getRequest();
    const { st } = request.cookies || {};
    if (!st) return false;

    const session = await
this.prismaService.session.findUnique({
      where: { token: st },
      include: {
        user: {
          include: {
            authProviders: true,
          },
        },
      },
    });

    if (!session) return false;
    if (session.expiresAt < new Date()) {
      const refreshed = await
this.refreshSessionIfPossible(session);
      if (!refreshed) {
        return false;
      }
    }
```

```
    if (session?.user) {
      request.user = session.user;
    }

    return true;
  }

  private async refreshSessionIfPossible(session: any):
Promise<boolean> {
    const { user } = session;
    if (!user?.authProviders || user.authProviders.length === 0)
{
      return false;
    }

    for (const provider of user.authProviders) {
      const { provider: providerType, accessToken, refreshToken
} = provider;

      if (!refreshToken) continue;

      switch (providerType) {
        case 'GOOGLE':
          if (await this.validateGoogleToken(accessToken,
refreshToken)) {
            return true;
          }
          break;
        case 'APPLE':
          if (await this.validateAppleToken(accessToken,
refreshToken)) {
            return true;
          }
          break;
        case 'GITHUB':
          if (await this.validateGithubToken(accessToken,
refreshToken)) {
            return true;
          }
          break;
        default:
          continue;
      }
    }

    return false;
  }
}
```

event-verse-backend/src/common/decorators/current-user.ts

```
import { createParamDecorator, ExecutionContext } from
'@nestjs/common';

export const CurrentUser = createParamDecorator((_, ctx:
ExecutionContext) => {
  return ctx.switchToHttp().getRequest().user || null;
});
```

event-verse-backend/src/common/decorators/index.ts

```
export * from '../decorators/auth/auth.guard';
export * from './current-user';
```

event-verse-backend/src/common/index.ts

```
export * from './decorators';
```

event-verse-backend/src/config/configuration.ts

```
export default () => ({
  nodeEnv: process.env.NODE_ENV as 'production' | 'development',
  port: parseInt(process.env.PORT, 10),
  session: {
    secret: process.env.SESSION_SECRET,
  },
  frontend: {
    url: process.env.FRONTEND_URL,
  },
});
```

event-verse-backend/src/config/mediasoup.config.ts

```
import {
  RtpCodecCapability,
  TransportListenInfo,
  WorkerLogTag,
  WorkerSettings,
} from 'mediasoup/node/lib/types';
import { cpus } from 'node:os';

export const config = {
  listenIp: '0.0.0.0',
  listenPort: 3016,

  worker: {
    numWorkers: Object.keys(cpus()).length,
    worker: {
      rtcMinPort: 10000,
      rtcMaxPort: 10100,
      logLevel: 'debug',
```

```
      logTags: ['info', 'ice', 'dtls', 'rtp', 'srtp', 'rtcp'] as
WorkerLogTag[],
    },
  } as WorkerSettings,

  router: {
    mediaCodecs: [
      {
        kind: 'audio',
        mimeType: 'audio/opus',
        clockRate: 48000,
        channels: 2,
      },
      {
        kind: 'video',
        mimeType: 'video/VP8',
        clockRate: 90000,
        parameters: {
          'x-google-start-bitrate': 1000,
        },
      },
    ] as RtpCodecCapability[],
  },

  webRtcTransport: {
    listenIps: [
      {
        ip: '0.0.0.0',
        announcedIp: '172.28.122.145',
      },
    ] as TransportListenInfo[],
    maxIncomeBitrate: 1500000,
    initialAvailableOutgoingBitrate: 1000000,
  },
} as const;
```

event-verse-backend/src/modules/auth/strategies/google.strategy.ts

```
import { Injectable } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { Strategy, VerifyCallback } from 'passport-google-
oauth20';

@Injectable()
export class GoogleStrategy extends PassportStrategy(Strategy,
'google') {
  constructor() {
    super({
      clientID: process.env.GOOGLE_CLIENT_ID,
```

```
      clientSecret: process.env.GOOGLE_CLIENT_SECRET,
      callbackURL: process.env.GOOGLE_CALLBACK_URL,
      scope: ['email', 'profile', 'openid'],
    });
  }

  authorizationParams(): { [key: string]: string } {
    return {
      access_type: 'offline',
    };
  }

  async validate(
    accessToken: string,
    refreshToken: string,
    profile: any,
    done: VerifyCallback,
  ): Promise<any> {
    const { id, displayName, name, emails, photos } = profile;

    const user = {
      id,
      email: emails[0].value,
      displayName,
      name,
      avatar: photos[0].value,
      accessToken,
      refreshToken,
    };
    done(null, user);
  }
}
```

event-verse-backend/src/modules/auth/auth.controller.ts

```
import { Response } from 'express';
import {
  Body,
  Controller,
  Get,
  Post,
  Req,
  Res,
  UseGuards,
} from '@nestjs/common';
import { AuthService } from './auth.service';
import { AuthGuard } from '@nestjs/passport';
import { User } from '@prisma/client';
import { CurrentUser, AuthGuard as LocalAuthGuard } from
'../../common';
```

```
@Controller('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Get('google')
  @UseGuards(AuthGuard('google'))
  googleAuth() {}

  @Get('google/callback')
  @UseGuards(AuthGuard('google'))
  googleAuthRedirect(@Req() req, @Res() res) {
    return this.authService.googleAuthRedirect(req, res);
  }

  @Post('validate')
  async validateToken(
    @Body() body: any,
    @Res({ passthrough: true }) res: Response,
  ) {
    const { user, session } = await
this.authService.validateToken(body);

    res.cookie('st', session.token, {
      // todo: sameSite
      httpOnly: true,
      sameSite: 'none',
      secure: true,
    });

    return user;
  }

  @UseGuards(LocalAuthGuard.USER)
  @Get('me')
  getUser(@CurrentUser() user: User) {
    return user;
  }
}
```

event-verse-backend/src/modules/auth/auth.module.ts

```
import { Module } from '@nestjs/common';
import { AuthController } from './auth.controller';
import { AuthService } from './auth.service';
import { PassportModule } from '@nestjs/passport';
import { GoogleStrategy } from './strategies/google.strategy';
import { PrismaModule } from '../prisma/prisma.module';
import { ConfigModule } from '@nestjs/config';

@Module({
  imports: [
    ConfigModule,
    PrismaModule,
```

```
    PassportModule.register({ defaultStrategy: 'google' }),
  ],
  controllers: [AuthController],
  providers: [AuthService, GoogleStrategy],
})
export class AuthModule {}
```

event-verse-backend/src/modules/auth/auth.service.ts

```
import { HttpException, HttpStatus, Injectable } from
'@nestjs/common';
import { PrismaService } from '../prisma/prisma.service';
import { AuthProviderType } from '@prisma/client';
import { randomUUID } from 'node:crypto';
import * as bcrypt from 'bcrypt';
import { ConfigService } from '@nestjs/config';

interface GoogleUser {
  id: string;
  email: string;
  displayName: string;
  name: {
    givenName: string;
    familyName: string;
  };
  avatar: string;
  accessToken: string;
  refreshToken?: string;
}

const generatePassword = (length: number): string => {
  const chars =

'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!
@#$%^&*()';
  let password = '';
  for (let i = 0; i < length; i++) {
    const randomIndex = Math.floor(Math.random() *
chars.length);
    password += chars[randomIndex];
  }
  return password;
};

const hashPassword = (password: string): Promise<string> => {
  const saltRounds = 10;
  return bcrypt.hash(password, saltRounds);
};

@Injectable()
export class AuthService {
  constructor(
    private readonly prismaService: PrismaService,
```

```
  private readonly configService: ConfigService,
) {}

async googleAuthRedirect(req: any, res: any) {
  const {
    id: providerId,
    email,
    displayName: fullName,
    avatar,
    refreshToken,
    accessToken,
  } = req.user as GoogleUser;

  const provider = AuthProviderType.GOOGLE;

  let user = await this.prismaService.user.findFirst({
    where: {
      authProviders: {
        some: {
          provider,
          providerId,
        },
      },
    },
  });

  if (!user) {
    const password = generatePassword(8);
    const hashedPassword = await hashPassword(password);
    user = await this.prismaService.user.create({
      data: {
        email,
        password: hashedPassword,
        fullName,
        avatar,
        authProviders: {
          create: {
            email,
            provider,
            providerId,
            accessToken,
            refreshToken,
          },
        },
      },
    });

    // todo: send email notification with password
  }

  const session = await this.prismaService.session.create({
    data: {
      token: randomUUID(),
```

```
        expiresAt: new Date(Date.now() + 60 * 60 * 1000),
        userId: user.id,
        authProviderProvider: provider,
        authProviderProviderId: providerId,
      },
    });

    return res.redirect(

`${this.configService.get('frontend.url')}/auth/validate?token=$
{session.token}`,
    );
  }

  async validateToken(body: any) {
    const { token } = body;
    const session = await
this.prismaService.session.findUnique({
      where: {
        token,
      },
      include: {
        user: {
          include: {
            authProviders: true,
          },
        },
        authProvider: true,
      },
    });

    if (!session) {
      throw new HttpException('Invalid token.',
HttpStatus.BAD_REQUEST);
    }

    if (session.authProvider) {
      // todo: check access | refresh tokens
    }

    const { user } = session;

    return { user, session };
  }
}
```

event-verse-backend/src/modules/mediasoup/mediasoup.module.ts

```
import { Module } from '@nestjs/common';
import { MediasoupService } from './mediasoup.service';

@Module({
  providers: [MediasoupService],
```

```
  exports: [MediasoupService],
})
export class MediasoupModule {}
```

event-verse-backend/src/modules/mediasoup/mediasoup.service.ts

```
import { Injectable } from '@nestjs/common';
import { createWorker } from 'mediasoup';
import { Router, Worker } from 'mediasoup/node/lib/types';
import { config } from '../../config/mediasoup.config';

@Injectable()
export class MediasoupService {
  private worker: Worker;

  async createMediasoupWorker() {
    const { worker: workerConfig } = config;
    this.worker = await createWorker(workerConfig);
    return this.worker;
  }

  createWebRtcTransport = async (router: Router) => {
    const { maxIncomeBitrate, ...webRtcTransportConfig } =
      config.webRtcTransport;
    const transport = await
router.createWebRtcTransport(webRtcTransportConfig);

    if (maxIncomeBitrate) {
      try {
        await transport.setMaxIncomingBitrate(maxIncomeBitrate);
      } catch (error) {
        console.error(error);
      }
    }

    return {
      transport,
      params: {
        id: transport.id,
        iceParameters: transport.iceParameters,
        iceCandidates: transport.iceCandidates,
        dtlsParameters: transport.dtlsParameters,
      },
    };
  };
}
```

event-verse-backend/src/modules/meeting/dto/create-meeting.dto.ts

```
import { IsNotEmpty, IsString, IsDate, IsOptional } from 'class-
validator';
```

```
export class CreateMeetingDto {
  @IsString()
  @IsNotEmpty()
  title: string;

  @IsDate()
  @IsNotEmpty()
  startTime: Date;

  @IsDate()
  @IsOptional()
  endTime?: Date;
}
```

event-verse-backend/src/modules/meeting/dto/index.ts

```
export * from './create-meeting.dto';
```

event-verse-backend/src/modules/meeting/meeting.controller.ts

```
import {
  Controller,
  Get,
  Post,
  Delete,
  Param,
  Body,
  Patch,
} from '@nestjs/common';
import { MeetingService } from './meeting.service';
import { CreateMeetingDto } from './dto';

@Controller('meeting')
export class MeetingController {
  constructor(private readonly meetingService: MeetingService)
{}

  @Post()
  async createMeeting(@Body() data: CreateMeetingDto) {
    return this.meetingService.createMeeting(data);
  }

  @Get(':id')
  async getMeetingById(@Param('id') id: string) {
    return this.meetingService.getMeetingById(id);
  }

  @Get()
  async getAllMeetings() {
    return this.meetingService.getAllMeetings();
  }

  @Patch(':id')
```

```
  async updateMeeting(@Param('id') id: string, @Body() data:
CreateMeetingDto) {
    return this.meetingService.updateMeeting(id, data);
  }

  @Delete(':id')
  async deleteMeeting(@Param('id') id: string) {
    return this.meetingService.deleteMeeting(id);
  }
}
```

event-verse-backend/src/modules/meeting/meeting.gateway.ts

```
import {
  WebSocketGateway,
  WebSocketServer,
  OnGatewayConnection,
  OnGatewayDisconnect,
  OnGatewayInit,
  SubscribeMessage,
  ConnectedSocket,
  MessageBody,
} from '@nestjs/websockets';
import { Server, Socket } from 'socket.io';
import { Logger } from '@nestjs/common';
import {
  Consumer,
  DtlsParameters,
  Producer,
  Router,
  RtpCapabilities,
  Transport,
  Worker,
  MediaKind,
  RtpParameters,
  AppData,
} from 'mediasoup/node/lib/types';
import { MediasoupService } from
'../mediasoup/mediasoup.service';
import { config } from '../../config/mediasoup.config';

interface User {
  id: string;
  producerTransport?: Transport;
  consumerTransport?: Transport;
  producers: Map<string, Producer>;
  consumers: Map<string, Consumer>;
}

interface Room {
  router: Router;
  users: Map<string, User>;
}
```

```
@WebSocketGateway(8000, { cors: { origin: '*' } })
export class MeetingGateway
  implements OnGatewayInit, OnGatewayConnection,
OnGatewayDisconnect
{
  @WebSocketServer() server: Server;
  private logger: Logger = new Logger('MeetingGateway');

  private clientConnections: Map<string, string> = new Map();

  private worker: Worker;
  private rooms: Map<string, Room> = new Map();

  constructor(private readonly mediasoupService:
MediasoupService) {}

  async afterInit() {
    this.worker = await
this.mediasoupService.createMediasoupWorker();
  }

  handleConnection(client: Socket) {
    const userId = client.handshake.query.userId as string;

    if (!userId) {
      this.logger.warn(`Client connected without userId:
${client.id}`);
      client.disconnect();
      return;
    }

    this.clientConnections.set(client.id, userId);
    this.logger.debug(`Client connected: ${client.id}, userId:
${userId}`);
  }

  handleDisconnect(client: Socket) {
    const userId = this.clientConnections.get(client.id);

    if (!userId) {
      this.logger.error(
        `Client disconnected: ${client.id}, but userId not
found`,
      );
      return;
    }

    this.clientConnections.delete(client.id);
    this.logger.error(
      `Client disconnected: ${client.id}, User: ${userId}
removed from connections`,
    );
```

```
    let meetingId: string | undefined;

    this.rooms.forEach((room, id) => {
      if (room.users.has(client.id)) {
        meetingId = id;
        const user = room.users.get(client.id);

        if (user) {
          user.producers.forEach((producer) => {
            this.logger.log(`Closing producer: ${producer.id}`);
            producer.close();
          });

          if (user.producerTransport) {
            this.logger.log(`Closing producer transport for
user: ${userId}`);
            user.producerTransport.close();
          }

          if (user.consumerTransport) {
            this.logger.log(`Closing consumer transport for
user: ${userId}`);
            user.consumerTransport.close();
          }

          user.consumers.forEach((consumer) => {
            this.logger.log(`Closing consumer: ${consumer.id}`);
            consumer.close();
          });
        }

        room.users.delete(client.id);
      }
    });

    if (meetingId) {
      const room = this.rooms.get(meetingId);
      if (room) {
        client.to(meetingId).emit('userLeft', { userId });
        this.logger.log(
          `User ${userId} leave from room ${meetingId},
notification sent to others`,
        );
      }
    }
  }

  private getRoomAndUser(
    client: Socket,
    roomId: string,
  ): { room?: Room; user?: User; error?: string } {
    const room = this.rooms.get(roomId);
```

```
    if (!room) {
      this.logger.error(`Room not found: ${roomId}`);
      return { error: `Room not found: ${roomId}` };
    }

    const user = room.users.get(client.id);
    if (!user) {
      this.logger.error(
        `User not found in room: ${roomId}, clientId:
${client.id}`,
      );
      return { error: `User not found in room: ${roomId}` };
    }

    return { room, user };
  }

  @SubscribeMessage('joinRoom')
  async handleJoinRoom(
    @MessageBody() message: { meetingId: string },
    @ConnectedSocket() client: Socket,
  ): Promise<{
    rtpCapabilities: RtpCapabilities;
    producers: { userId: string; producerId: string }[];
  }> {
    const { meetingId } = message;
    let room = this.rooms.get(meetingId);

    if (!room) {
      const { mediaCodecs } = config.router;
      const router = await this.worker.createRouter({
mediaCodecs });
      room = {
        router,
        users: new Map(),
      };
      this.rooms.set(meetingId, room);
      this.logger.log(`Created new room: ${meetingId}.`);
    }

    const userId = this.clientConnections.get(client.id);
    if (!userId) {
      this.logger.warn(`Client ${client.id} has no associated
userId`);
      client.disconnect();
      return;
    }

    const user: User = {
      id: userId,
      producers: new Map(),
      consumers: new Map(),
    };
```

```
    room.users.set(client.id, user);
    client.join(meetingId);

    this.logger.debug(`Client ${client.id} joined meet
${meetingId}`);

    const producers: { userId: string; producerId: string }[] =
[];
    room.users.forEach((roomUser, userClientId) => {
      if (userClientId !== client.id) {
        roomUser.producers.forEach((producer, producerId) => {
          producers.push({
            userId: roomUser.id,
            producerId,
          });
        });
      }
    });

    return { rtpCapabilities: room.router.rtpCapabilities,
producers };
  }

  @SubscribeMessage('leaveRoom')
  handleLeaveRoom(
    @MessageBody() message: { meetingId: string },
    @ConnectedSocket() client: Socket,
  ): void {
    const { meetingId } = message;
    const { room, user, error } = this.getRoomAndUser(client,
meetingId);
    if (error) {
      return;
    }

    user.producers.forEach((producer) => producer.close());
    user.consumers.forEach((consumer) => consumer.close());

    if (user.producerTransport) {
      user.producerTransport.close();
    }
    if (user.consumerTransport) {
      user.consumerTransport.close();
    }

    room.users.delete(client.id);
    this.logger.log(`User ${user.id} left room ${meetingId}`);

    client.broadcast.to(meetingId).emit('userLeft', {
      userId: user.id,
    });

    if (room.users.size === 0) {
```

```
      this.rooms.delete(meetingId);
      this.logger.log(`Room ${meetingId} deleted as it is now
empty.`);
    }
  }

  @SubscribeMessage('createProducerTransport')
  async handleCreateProducerTransport(
    @MessageBody() message: { meetingId: string },
    @ConnectedSocket() client: Socket,
  ) {
    const { meetingId } = message;
    const { room, user, error } = this.getRoomAndUser(client,
meetingId);
    if (error) {
      return { error };
    }

    try {
      const { transport, params } =
        await
this.mediasoupService.createWebRtcTransport(room.router);
      user.producerTransport = transport;
      return {
        params,
      };
    } catch (error) {
      this.logger.error(error);
      return {
        error,
      };
    }
  }

  @SubscribeMessage('connectProducerTransport')
  async handleConnectProducerTransport(
    @MessageBody()
    message: { dtlsParameters: DtlsParameters; meetingId: string
},
    @ConnectedSocket() client: Socket,
  ) {
    const { dtlsParameters, meetingId } = message;
    const { user, error } = this.getRoomAndUser(client,
meetingId);
    if (error) {
      return { error };
    }

    if (!user.producerTransport) {
      const error = 'Producer transport not found for the user';
      this.logger.error(error);
      return { error };
    }
```

```
    await user.producerTransport.connect({
      dtlsParameters,
    });
    this.logger.log('Producer Transport connected!');
    return { success: true };
  }

  @SubscribeMessage('produce')
  async handleProduce(
    @MessageBody()
    message: {
      kind: MediaKind;
      rtpParameters: RtpParameters;
      appData: AppData;
      meetingId: string;
    },
    @ConnectedSocket() client: Socket,
  ) {
    const { kind, rtpParameters, appData, meetingId } = message;
    const { user, error } = this.getRoomAndUser(client,
meetingId);
    if (error) {
      return { error };
    }

    const producer = await user.producerTransport.produce({
      kind,
      rtpParameters,
      appData,
    });

    user.producers.set(producer.id, producer);

    client.broadcast.to(meetingId).emit('newProducer', {
      producerId: producer.id,
      kind: producer.kind,
      userId: user.id,
    });

    this.logger.log('PRODUCE');
    return { id: producer.id };
  }

  @SubscribeMessage('createConsumerTransport')
  async handleCreateConsumerTransport(
    @MessageBody() message: { meetingId: string },
    @ConnectedSocket() client: Socket,
  ) {
    const { meetingId } = message;
    const { room, user, error } = this.getRoomAndUser(client,
meetingId);
    if (error) {
```

```
      return { error };
    }

    try {
      const { transport, params } =
        await
this.mediasoupService.createWebRtcTransport(room.router);
      user.consumerTransport = transport;
      return {
        params,
      };
    } catch (error) {
      this.logger.error(error);
      return {
        error,
      };
    }
  }

  @SubscribeMessage('connectConsumerTransport')
  async handleConnectConsumerTransport(
    @MessageBody()
    message: { meetingId: string; dtlsParameters: DtlsParameters
},
    @ConnectedSocket() client: Socket,
  ) {
    const { meetingId, dtlsParameters } = message;
    const { user, error } = this.getRoomAndUser(client,
meetingId);
    if (error) {
      return { error };
    }

    if (!user.producerTransport) {
      const error = 'Producer transport not found for the user';
      this.logger.error(error);
      return { error };
    }

    await user.consumerTransport.connect({ dtlsParameters });
    return { success: true };
  }

  @SubscribeMessage('resume')
  async handleResume(
    @MessageBody() messageBody: { meetingId: string; consumerId:
string },
    @ConnectedSocket() client: Socket,
  ) {
    const { meetingId, consumerId } = messageBody;
    const { user, error } = this.getRoomAndUser(client,
meetingId);
    if (error) {
```

```
      return { error };
    }

    const consumer = user.consumers.get(consumerId);
    if (!consumer) {
      return { error: 'Consumer not found' };
    }

    await consumer.resume();
    return {
      success: true,
    };
  }

  @SubscribeMessage('consume')
  async handleConsume(
    @MessageBody()
    messageBody: {
      meetingId: string;
      producerId: string;
      rtpCapabilities: RtpCapabilities;
    },
    @ConnectedSocket() client: Socket,
  ) {
    const { meetingId, producerId, rtpCapabilities } =
messageBody;

    const { room, user, error } = this.getRoomAndUser(client,
meetingId);
    if (error) {
      return { error };
    }

    if (!user.consumerTransport) {
      const error = 'Consumer transport not found for the user';
      this.logger.error(error);
      return { error };
    }

    const router = room.router;
    if (!router.canConsume({ producerId, rtpCapabilities })) {
      return { error: 'Cannot consume this producer' };
    }

    try {
      const consumer = await user.consumerTransport.consume({
        producerId,
        rtpCapabilities,
        paused: true,
      });

      user.consumers.set(consumer.id, consumer);
```

```
      return {
        id: consumer.id,
        producerId: consumer.producerId,
        kind: consumer.kind,
        rtpParameters: consumer.rtpParameters,
      };
    } catch (error) {
      this.logger.error(error);
      return { error: 'Failed to create consumer' };
    }
  }
}
```

event-verse-backend/src/modules/meeting/meeting.module.ts

```
import { Module } from '@nestjs/common';
import { MeetingService } from './meeting.service';
import { MeetingController } from './meeting.controller';
import { PrismaModule } from '../prisma/prisma.module';
import { MeetingGateway } from './meeting.gateway';
import { MediasoupModule } from '../mediasoup/mediasoup.module';

@Module({
  imports: [PrismaModule, MediasoupModule],
  controllers: [MeetingController],
  providers: [MeetingService, MeetingGateway],
})
export class MeetingModule {}
```

event-verse-backend/src/modules/meeting/meeting.service.ts

```
import { Injectable } from '@nestjs/common';
import { PrismaService } from '../prisma/prisma.service';
import { CreateMeetingDto } from './dto';

@Injectable()
export class MeetingService {
  constructor(private readonly prisma: PrismaService) {}

  async createMeeting(data: CreateMeetingDto) {
    return this.prisma.meeting.create({
      data,
    });
  }

  async getMeetingById(id: string) {
    return this.prisma.meeting.findUnique({
      where: { id },
    });
  }
```

```
  async getAllMeetings() {
    return this.prisma.meeting.findMany();
  }

  async updateMeeting(id: string, data: CreateMeetingDto) {
    return this.prisma.meeting.update({
      where: { id },
      data,
    });
  }

  async deleteMeeting(id: string) {
    return this.prisma.meeting.delete({
      where: { id },
    });
  }
}
```

event-verse-backend/src/modules/prisma/prisma.module.ts

```
import { Module } from '@nestjs/common';

import { PrismaService } from './prisma.service';

@Module({
  providers: [PrismaService],
  exports: [PrismaService],
})
export class PrismaModule {}
```

event-verse-backend/src/modules/prisma/prisma.service.ts

```
import { Injectable, OnModuleDestroy, OnModuleInit } from
'@nestjs/common';
import { PrismaClient } from '@prisma/client';

@Injectable()
export class PrismaService
  extends PrismaClient
  implements OnModuleInit, OnModuleDestroy
{
  async onModuleInit() {
    await this.$connect();
  }

  async onModuleDestroy() {
    await this.$disconnect();
  }
}
```

event-verse-backend/src/app.module.ts

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { ConfigModule } from '@nestjs/config';
import configuration from './config/configuration';
import { PrismaModule } from './modules/prisma/prisma.module';
import { AuthModule } from './modules/auth/auth.module';
import { MeetingModule } from
'./modules/meeting/meeting.module';

@Module({
  imports: [
    ConfigModule.forRoot({
      load: [configuration],
      isGlobal: true,
    }),
    PrismaModule,
    AuthModule,
    MeetingModule,
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

event-verse-backend/src/main.ts

```
import { NestFactory } from '@nestjs/core';
import { ConfigService } from '@nestjs/config';
import { AppModule } from './app.module';
import { NestExpressApplication } from '@nestjs/platform-
express';
import { Logger } from '@nestjs/common';

import * as cookieParser from 'cookie-parser';

async function bootstrap() {
  const app = await
NestFactory.create<NestExpressApplication>(AppModule);
  app.enableCors({
    origin: true,
    credentials: true,
  });

  const globalPrefix = 'api';
  app.setGlobalPrefix(globalPrefix);

  initCookieMiddleware(app);

  const port = process.env.PORT || 3001;
  await app.listen(port);

  const url = await app.getUrl();
```

```
  Logger.debug(`Application is running on: ${url}`);
}

function initCookieMiddleware(app: NestExpressApplication) {
  const sessionConfig = app.get(ConfigService).get('session');

  app.use(cookieParser(sessionConfig.secret));
}

bootstrap();
```

event-verse-backend/prisma/schema.prisma

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id            String    @id @default(uuid()) @db.Uuid
  email         String    @unique
  password      String
  fullName      String    @map("full_name")
  avatar        String?

  createdAt     DateTime  @default(now())  @db.Timestamp()
@map("created_at")
  updatedAt     DateTime  @updatedAt       @db.Timestamp()
@map("updated_at")

  authProviders AuthProvider[]
  session       Session[]

  @@map("users")
}

enum AuthProviderType {
  GOOGLE
  APPLE
  GITHUB
}

model AuthProvider {
  email        String
  provider     AuthProviderType
  providerId   String             @map("provider_id")
```

```
  accessToken  String?            @map("access_token")
  refreshToken String?            @map("refresh_token")

  createdAt    DateTime           @default(now())
@db.Timestamp() @map("created_at")

  user         User               @relation(fields: [userId],
references: [id])
  userId       String             @db.Uuid
  sessions     Session[]

  @@unique([email, provider])
  @@unique([userId, provider,  providerId])
  @@id([provider, providerId])
  @@map("auth_providers")
}

model Session {
  id                    String             @id @default(uuid())
@db.Uuid
  token                 String             @unique
  expiresAt             DateTime           @map("expires_at")
@db.Timestamp()
  createdAt             DateTime           @default(now())
@map("created_at") @db.Timestamp()
  updatedAt             DateTime           @updatedAt
@map("updated_at") @db.Timestamp()

  user                  User               @relation(fields:
[userId], references: [id])
  userId                String             @db.Uuid

  authProvider          AuthProvider?      @relation(fields:
[authProviderProvider, authProviderProviderId], references:
[provider, providerId])
  authProviderProvider   AuthProviderType?
  authProviderProviderId String?

  @@map("sessions")
}

model Meeting {
  id            String      @id @default(uuid())  @db.Uuid
  title         String
  startTime     DateTime    @map("start_time")  @db.Timestamp()
  endTime       DateTime?   @map("end_time")    @db.Timestamp()

  createdAt     DateTime    @default(now()) @map("created_at")
@db.Timestamp()
  updatedAt     DateTime    @updatedAt      @map("updated_at")
@db.Timestamp()

  categories  MeetingCategory[]
```

```
  @@map("meetings")
}

model Category {
  id          String      @id @default(uuid()) @db.Uuid
  name        String      @unique

  createdAt   DateTime    @default(now()) @db.Timestamp()
@map("created_at")
  updatedAt   DateTime    @updatedAt @db.Timestamp()
@map("updated_at")

  meetings    MeetingCategory[]

  @@map("categories")
}

model MeetingCategory {
  meeting     Meeting     @relation(fields: [meetingId],
references: [id])
  meetingId   String      @db.Uuid
  category    Category    @relation(fields: [categoryId],
references: [id])
  categoryId  String      @db.Uuid

  createdAt   DateTime    @default(now()) @db.Timestamp()
@map("created_at")

  @@id([meetingId, categoryId])
  @@map("meeting_categories")
}
```

event-verse-backend/src/docker-compose.yml

```
version: '3.9'

services:
  postgres:
    image: postgres:15
    container_name: ev_postgres
    environment:
      POSTGRES_USER: ev_user_fdrzqund
      POSTGRES_PASSWORD: j@k1jNkvdkVz^uhy
      POSTGRES_DB: event_verse__db_1b34336fe583
    ports:
      - '5432:5432'
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

event-verse-frontend/src/app/auth/validate/page.tsx

```
"use client";

import { useEffect } from "react";
import { useRouter, useSearchParams } from "next/navigation";
import { useAppDispatch } from "@/hooks/redux";
import { validateToken } from "@/features/auth/slice";

const ValidatePage = () => {
  const dispatch = useAppDispatch();
  const router = useRouter();
  const searchParams = useSearchParams();

  useEffect(() => {
    const handleValidateToken = async () => {
      const token = searchParams.get("token");

      console.log(token);

      if (!token) {
        router.push("/");
        return;
      }

      try {
        await dispatch(validateToken(token)).unwrap();
      } catch (error) {
        console.error("Token validation failed:", error);
      } finally {
        router.push("/");
      }
    };

    handleValidateToken();
  }, [searchParams, router, dispatch]);

  return (
    <div className="fixed inset-0 flex items-center justify-
center bg-black bg-opacity-50 z-50">
      <div className="animate-spin rounded-full h-16 w-16
border-t-4 border-blue-500 border-solid"></div>
    </div>
  );
};

export default ValidatePage;

event-verse-frontend/src/app/interfaces/meeting.ts

export interface RemoteMediaStreams {
  [userId: string]: {
    videoStream: MediaStream;
    audioStream: MediaStream;
```

```
  };
}
```

event-verse-frontend/src/app/meetings/[id]/page.tsx

```
"use client";

import React, { useEffect, useRef, useState } from "react";
import { useParams } from "next/navigation";
import {
  Device,
  Device as DeviceType,
  MediaKind,
  RtpCapabilities,
  RtpParameters,
  Transport,
} from "mediasoup-client/lib/types";
import { useAppDispatch } from "@/hooks/redux";
import { useWebSocket } from "@/context/WebSocketProvider";
import {
  CreateTransportResponse,
  CreateTransportResponseParams,
} from "@/types/Meeting";
import { RemoteMediaStreams } from "@/app/interfaces/meeting";
import { fetchMeetingById } from "@/features/meeting/slice";
import { delay } from "@/utils";

import PreparationScreen from
"@/components/Meeting/PreparationScreen";
import ConferenceScreen from
"@/components/Meeting/ConferenceScreen";

const MeetingPage: React.FC = () => {
  const dispatch = useAppDispatch();
  const { id: meetingId } = useParams();
  const [joined, setJoined] = useState<boolean>(false);

  useEffect(() => {
    if (typeof meetingId === "string")
dispatch(fetchMeetingById(meetingId));

    return () => {
      socket?.emit("leaveRoom", { meetingId });
    };
  }, [meetingId]);

  const { socket } = useWebSocket();
  const device = useRef<DeviceType | null>(null);
  const sendTransport = useRef<Transport | null |
undefined>(null);
  const recvTransport = useRef<Transport | null |
undefined>(null);
```

```
  const [localMediaStream, setLocalMediaStream] =
useState<MediaStream | null>(
    null
  );
  const [remoteMediaStreams, setRemoteMediaStreams] =
    useState<RemoteMediaStreams>({});

  useEffect(() => {
    socket?.on("newProducer", async ({ producerId, kind, userId
}) => {
      await consume(producerId, userId);
    });

    socket?.on("userLeft", ({ userId }: { userId: string }) => {
      console.log(`User ${userId} disconnected`);
      setRemoteMediaStreams((prev) => {
        const updatedStreams = { ...prev };
        delete updatedStreams[userId];
        return updatedStreams;
      });
    });
  }, [socket]);

  const consume = async (producerId: string, userId: string) =>
{
    if (!recvTransport.current || !device.current) {
      await delay(1000);
      await consume(producerId, userId);
      return;
    }

    socket?.emit(
      "consume",
      {
        meetingId,
        producerId,
        rtpCapabilities: device.current.rtpCapabilities,
      },
      async ({
        id: consumerId,
        producerId,
        kind,
        rtpParameters,
      }: {
        id: string;
        producerId: string;
        kind: MediaKind;
        rtpParameters: RtpParameters;
      }) => {
        if (!consumerId) {
          console.warn("Consume request failed!");
          return;
        }
```

```
        const consumer = await recvTransport.current?.consume({
          id: consumerId,
          producerId,
          kind,
          rtpParameters,
        });

        const stream = new MediaStream();
        if (consumer?.track) stream.addTrack(consumer.track);
        setRemoteMediaStreams((prev) => {
          const userStreams = prev[userId] || {};
          return {
            ...prev,
            [userId]: {
              ...userStreams,
              ...(kind === "video"
                ? { videoStream: stream }
                : { audioStream: stream }),
            },
          };
        });

        socket.emit("resume", { meetingId, consumerId }, () => {
          //
        });
      }
    );
  };

  const createDevice = async (routerRtpCapabilities:
RtpCapabilities) => {
    device.current = new Device();
    await device.current.load({ routerRtpCapabilities });
    console.log("Device created");
  };

  const startProducing = async (trackType: "video" | "audio") =>
{
    if (!sendTransport.current || !localMediaStream) {
      console.error("No send transport or local media stream
available");
      return;
    }

    console.log(`Starting to produce ${trackType}...`);

    try {
      if (trackType === "video") {
        const videoTrack = localMediaStream.getVideoTracks()[0];

        if (videoTrack) {
```

```
          const videoProducer = await
sendTransport.current.produce({
            track: videoTrack,
          });
          console.log("Video producer created:", videoProducer);
        } else {
          console.warn("No video track available");
        }
      } else if (trackType === "audio") {
        const audioTrack = localMediaStream.getAudioTracks()[0];

        if (audioTrack) {
          const audioProducer = await
sendTransport.current.produce({
            track: audioTrack,
          });

          // audioProducer.on("trackended", () => {
          //   console.log("audio trackended");
          // });
          // audioProducer.on("transportclose", () => {
          //   console.log("audio transportclose");
          // });

          console.log("Audio producer created:", audioProducer);
        } else {
          console.warn("No audio track available");
        }
      }
    } catch (error) {
      console.error("Failed to produce media:", error);
    }
  };

  const createSendTransport = async () => {
    socket?.emit(
      "createProducerTransport",
      { meetingId },
      (data: CreateTransportResponse) => {
        if ("error" in data) {
          if (typeof data.error === "string") {
            console.log(data.error);
          } else {
            console.log(data.error);
          }
          return;
        }

        sendTransport.current =
device.current?.createSendTransport(
          (data as CreateTransportResponseParams).params
        );
        console.log("send transport created.");
```

```
        sendTransport.current?.on(
          "connect",
          async ({ dtlsParameters }, callback, errback) => {
            try {
              socket.emit(
                "connectProducerTransport",
                { dtlsParameters, meetingId },
                () => {
                  console.log("send transport connected.");
                  callback();
                }
              );
            } catch (error: any) {
              errback(error);
            }
          }
        );

        sendTransport.current?.on(
          "produce",
          async ({ kind, rtpParameters, appData }, callback,
errback) => {
            try {
              socket.emit(
                "produce",
                { kind, rtpParameters, appData, meetingId },
                ({ id }: { id: string }) => {
                  callback({ id });
                }
              );
            } catch (error: any) {
              errback(error);
            }
          }
        );

        startProducing("audio");
        startProducing("video");
      }
    );
  };
  const createRecvTransport = async () => {
    socket?.emit(
      "createConsumerTransport",
      { meetingId },
      (data: CreateTransportResponse) => {
        if ("error" in data) {
          if (typeof data.error === "string") {
            console.log(data.error);
          } else {
            console.log("error is not string");
            console.log(data.error);
```

```
          }
          return;
        }

        recvTransport.current =
device.current?.createRecvTransport(
          (data as CreateTransportResponseParams).params
        );
        console.log("recv transport created.");

        recvTransport.current?.on(
          "connect",
          ({ dtlsParameters }, callback, errback) => {
            try {
              socket.emit(
                "connectConsumerTransport",
                { dtlsParameters, meetingId },
                () => {
                  console.log("consumer transport connected.");
                  callback();
                }
              );
            } catch (error: any) {
              errback(error);
            }
          }
        );
      }
    );
  };

  const joinRoom = () => {
    if (!meetingId || !socket) return;

    socket.emit(
      "joinRoom",
      { meetingId },
      async ({
        rtpCapabilities,
        producers,
      }: {
        rtpCapabilities: RtpCapabilities;
        producers: { userId: string; producerId: string }[];
      }) => {
        console.log(`join room: ${meetingId}`);
        await createDevice(rtpCapabilities);
        await createSendTransport();
        await createRecvTransport();
        setJoined(true);

        for (const { userId, producerId } of producers) {
          await consume(producerId, userId);
        }
```

```
      }
    );
  };

  const setMediaStream = (stream: MediaStream | null) => {
    setLocalMediaStream(stream);
  };

  return joined ? (
    <ConferenceScreen
      localMediaStream={localMediaStream}
      remoteMediaStreams={remoteMediaStreams}
    />
  ) : (
    <PreparationScreen onJoin={joinRoom}
setMediaStream={setMediaStream} />
  );
};

export default MeetingPage;
```

event-verse-frontend/src/app/global.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;

@layer utilities {
  /* Hide scrollbar for Chrome, Safari and Opera */
  .no-scrollbar::-webkit-scrollbar {
    display: none;
  }
  /* Hide scrollbar for IE, Edge and Firefox */
  .no-scrollbar {
    -ms-overflow-style: none; /* IE and Edge */
    scrollbar-width: none; /* Firefox */
  }
}
```

event-verse-frontend/src/app/layout.tsx

```
import type { Metadata } from "next";
import { Providers } from "./providers";
import { Inter } from "next/font/google";
import "./globals.css";
import Navbar from "@/components/Navbar/Navbar";
import AuthWrapper from "@/components/AuthWrapper";
import ThemeInitializer from "@/components/ThemeInitializer";

const inter = Inter({ subsets: ["latin"] });

export const metadata: Metadata = {
  title: "Event Verse",
```

```
  description: "todo",
};

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <html className={inter.className}>
      <body className="bg-zinc-100 text-black dark:bg-zinc-950
dark:text-white h-screen flex flex-col">
        <Providers>
          <AuthWrapper>
            <ThemeInitializer />

            <Navbar />
            <main className="flex-grow flex">{children}</main>
          </AuthWrapper>
        </Providers>
      </body>
    </html>
  );
}
```

event-verse-frontend/src/app/page.tsx

```
"use client";

import React, { useEffect, useState } from "react";
import { PlusIcon } from "@heroicons/react/24/solid";
import { useAppDispatch, useAppSelector } from "@/hooks/redux";
import { selectMeetings } from "@/features/meeting";
import { fetchMeetings } from "@/features/meeting/slice";
import { selectMe } from "@/features/auth";

import CreateMeetingModal from
"@/components/CreateMeetingModal";
import HorizontalMeetingList from
"@/components/HorizontalMeetingList";

export default function Home() {
  const dispatch = useAppDispatch();

  const me = useAppSelector(selectMe);
  const meetings = useAppSelector(selectMeetings);

  useEffect(() => {
    dispatch(fetchMeetings());
  }, [dispatch]);

  const [isModalOpen, setIsModalOpen] = useState(false);
```

```
  const handleOpenModal = () => setIsModalOpen(true);
  const handleCloseModal = () => setIsModalOpen(false);

  return (
    <>
      {me && (
        <>
          <button
            onClick={handleOpenModal}
            className="fixed bottom-4 right-4 p-3 bg-purple-500
text-white rounded-full shadow-lg hover:bg-purple-600
transition"
          >
            <PlusIcon className="h-6 w-6" />
          </button>
          <CreateMeetingModal isOpen={isModalOpen}
onClose={handleCloseModal} />
        </>
      )}

      <div className="w-full mt-4 px-4">
        <h2 className="text-lg font-bold mb-4 cursor-pointer
hover:text-purple-400 text-center">
          Latest
        </h2>
        <HorizontalMeetingList meetings={meetings} />
      </div>
    </>
  );
}
```

event-verse-frontend/src/app/providers.tsx

```
"use client";

import store from "../store";
import { Provider as ReduxProvider } from "react-redux";
import { AuthModalProvider } from "@/context/AuthModalContext";
import { WebSocketProvider } from "@/context/WebSocketProvider";

export function Providers({ children }: { children:
React.ReactNode }) {
  return (
    <ReduxProvider store={store}>
      <AuthModalProvider>
        <WebSocketProvider>{children}</WebSocketProvider>
      </AuthModalProvider>
    </ReduxProvider>
  );
}
```

event-verse-frontend/src/components/Meeting/ConferenceScreen.tsx

```
import React, { useState } from "react";

import {
  MicrophoneIcon,
  VideoCameraIcon,
  ComputerDesktopIcon,
  Cog6ToothIcon,
  PhoneArrowDownLeftIcon,
  VideoCameraSlashIcon,
} from "@heroicons/react/24/solid";
import { useRouter } from "next/navigation";
import { RemoteMediaStreams } from "@/app/interfaces/meeting";

interface ConferenceScreenProps {
  localMediaStream: MediaStream | null;
  remoteMediaStreams: RemoteMediaStreams;
}

const ConferenceScreen: React.FC<ConferenceScreenProps> = ({
  localMediaStream,
  remoteMediaStreams,
}) => {
  const [audioEnabled, setAudioEnabled] = useState(true);
  const [videoEnabled, setVideoEnabled] = useState(true);
  const router = useRouter();

  const toggleAudio = () => {
    if (localMediaStream) {
      const audioTracks = localMediaStream.getAudioTracks();
      if (audioTracks.length === 0) {
        alert("Please allow access to the audio device and
reload the page.");
        return;
      }
      audioTracks.forEach((track) => (track.enabled =
!track.enabled));
      setAudioEnabled(!audioEnabled);
    }
  };

  const toggleVideo = () => {
    if (localMediaStream) {
      const videoTracks = localMediaStream.getVideoTracks();
      if (videoTracks.length === 0) {
        alert("Please allow access to the video device and
reload the page.");
        return;
      }
      videoTracks.forEach((track) => (track.enabled =
!track.enabled));
      setVideoEnabled(!videoEnabled);
    }
  };
```

```
  const startScreenShare = async () => {
    try {
      const displayStream = await
navigator.mediaDevices.getDisplayMedia({
        video: true,
      });
      console.log("Screen sharing started:", displayStream);
    } catch (error) {
      console.error("Error starting screen sharing:", error);
    }
  };

  const disconnect = () => {
    router.push("/");
  };

  const participantCount =
    Object.keys(remoteMediaStreams).length + (localMediaStream ?
1 : 0);

  const getGridColumns = () => {
    if (participantCount <= 1) return "grid-cols-1";
    if (participantCount === 2) return "grid-cols-2";
    if (participantCount <= 4) return "grid-cols-2";
    if (participantCount <= 6) return "grid-cols-3";
    if (participantCount <= 9) return "grid-cols-3";
    return "grid-cols-4";
  };

  return (
    <div className="flex flex-col h-full w-full">
      <div
        className={`grid ${getGridColumns()} gap-4 p-4 flex-grow
h-[65%] overflow-auto place-items-center`}
      >
        {localMediaStream && (
          <div className="relative w-full aspect-video bg-black
border rounded-lg overflow-hidden">
            <video
              ref={(video) => {
                if (video) {
                  video.srcObject = localMediaStream;
                  video.muted = true;
                }
              }}
              autoPlay
              playsInline
              className="w-full h-full object-cover"
              disablePictureInPicture
            />
          </div>
        )}
```

```
        {Object.entries(remoteMediaStreams).map(([userId,
streams]) => (
          <div
            key={userId}
            className="relative w-full aspect-video bg-black
border rounded-lg overflow-hidden"
          >
            {!streams.videoStream && (
              <h3 className="absolute inset-0 flex items-center
justify-center text-white text-xl font-bold">
                {`User: ${userId}`}
              </h3>
            )}

            {streams.videoStream && (
              <video
                ref={(video) => {
                  if (video) {
                    video.srcObject = streams.videoStream;
                  }
                }}
                autoPlay
                playsInline
                className="w-full h-full object-cover"
                disablePictureInPicture
              />
            )}

            {streams.audioStream && (
              <audio
                ref={(audio) => {
                  if (audio) {
                    audio.srcObject = streams.audioStream;
                  }
                }}
                autoPlay
              />
            )}
          </div>
        ))}
      </div>

      <div className="w-full p-4 flex justify-center items-
center gap-2 bg-zinc-900">
        <button
          onClick={toggleAudio}
          className="relative flex items-center justify-center
w-14 h-14 rounded-full border-2 border-gray-500 text-white
hover:bg-zinc-800"
        >
          <MicrophoneIcon
            className={`bg-opacity-50 rounded-full w-6 h-6 ${
```

```
              audioEnabled ? "" : "opacity-50"
            }`}
          />
          {!audioEnabled && (
            <div className="absolute inset-0 bg-red-700 rounded-
full opacity-50" />
          )}
        </button>

        <button
          onClick={toggleVideo}
          className="relative flex items-center justify-center
w-14 h-14 rounded-full border-2 border-gray-500 text-white
hover:bg-zinc-800"
        >
          {videoEnabled ? (
            <VideoCameraIcon className="h-6 w-6" />
          ) : (
            <VideoCameraSlashIcon className="h-6 w-6 text-red-
500" />
          )}
        </button>

        <button
          onClick={startScreenShare}
          className="relative flex items-center justify-center
w-14 h-14 rounded-full border-2 border-gray-500 text-white
hover:bg-zinc-800"
        >
          <ComputerDesktopIcon className="h-6 w-6" />
        </button>

        <button className="relative flex items-center justify-
center w-14 h-14 rounded-full border-2 border-gray-500 text-
white hover:bg-zinc-800">
          <Cog6ToothIcon className="h-6 w-6" />
        </button>

        <button
          onClick={disconnect}
          className="flex items-center justify-center w-14 h-14
rounded-full border-2 border-red-500 bg-red-500 text-white"
        >
          <PhoneArrowDownLeftIcon className="h-6 w-6" />
        </button>
      </div>
    </div>
  );
};

export default ConferenceScreen;
```

event-verse-frontend/src/components/Meeting/PreparationScreen.tsx

```
import React, { useEffect, useRef, useState } from "react";
import {
  MicrophoneIcon,
  //   MicrophoneSlashIcon,
  VideoCameraIcon,
  VideoCameraSlashIcon,
  ExclamationCircleIcon,
} from "@heroicons/react/24/solid";
import { useAppSelector } from "@/hooks/redux";
import { selectMeetingById } from "@/features/meeting";

interface PreparationScreenProps {
  onJoin: () => void;
  setMediaStream: (stream: MediaStream | null) => void;
}

const PreparationScreen: React.FC<PreparationScreenProps> = ({
  onJoin,
  setMediaStream,
}) => {
  const meeting = useAppSelector(selectMeetingById);

  const [permissionsGranted, setPermissionsGranted] =
useState(false);
  const [audioEnabled, setAudioEnabled] = useState(true);
  const [videoEnabled, setVideoEnabled] = useState(true);
  const [videoAvailable, setVideoAvailable] = useState(true);
  const videoRef = useRef<HTMLVideoElement | null>(null);
  const mediaStream = useRef<MediaStream | null>(null);

  const getUserMedia = async () => {
    try {
      const audioStream = await
navigator.mediaDevices.getUserMedia({
        audio: {
          echoCancellation: false,
          noiseSuppression: true,
          autoGainControl: true,
          sampleRate: 48000,
        },
      });
      setPermissionsGranted(true);
      try {
        const videoStream = await
navigator.mediaDevices.getUserMedia({
          video: true,
        });
        mediaStream.current = new MediaStream([
          ...audioStream.getTracks(),
          ...videoStream.getTracks(),
```

```
        ]);
        setVideoAvailable(true);
      } catch (videoError) {
        console.warn("Video not available:", videoError);
        mediaStream.current = audioStream;
        setVideoAvailable(false);
      }

      if (videoRef.current && mediaStream.current) {
        videoRef.current.srcObject = mediaStream.current;
      }

      setMediaStream(mediaStream.current);
    } catch (audioError) {
      console.error("Failed to get audio permissions:",
audioError);
      setPermissionsGranted(false);
    }
  };

  const toggleAudio = () => {
    if (mediaStream.current) {
      const audioTracks = mediaStream.current.getAudioTracks();
      audioTracks.forEach((track) => (track.enabled =
!track.enabled));
      setAudioEnabled(!audioEnabled);
    }
  };

  const toggleVideo = () => {
    if (mediaStream.current && videoAvailable) {
      const videoTracks = mediaStream.current.getVideoTracks();
      videoTracks.forEach((track) => (track.enabled =
!track.enabled));
      setVideoEnabled(!videoEnabled);
    }
  };

  useEffect(() => {
    getUserMedia();
  }, []);

  return (
    <div className="flex h-full p-8 items-center justify-center
w-full">
      <div className="relative flex flex-col items-center space-
y-4 w-2/3">
        <div className="w-full h-auto border rounded-lg bg-gray-
800 relative overflow-hidden border-gray-300 dark:border-gray-
700">
          <video
            ref={videoRef}
            autoPlay
```

```
            playsInline
            className="w-full h-full object-cover rounded-lg
aspect-video"
            disablePictureInPicture
          />
          {!permissionsGranted && (
            <div className="absolute inset-0 flex items-center
justify-center text-white text-lg font-semibold">
              Camera not found
            </div>
          )}
          <div className="absolute bottom-4 left-4 flex space-x-
4">
            <button
              onClick={toggleAudio}
              className="text-white text-2xl relative"
            >
              <MicrophoneIcon
                className={`p-2 bg-black bg-opacity-50 rounded-
full w-8 h-8 ${
                  audioEnabled ? "" : "opacity-50"
                }`}
              />
              {!audioEnabled && (
                <div className="absolute inset-0 bg-red-500
rounded-full opacity-50" />
              )}
            </button>
            <button onClick={toggleVideo} className="text-white
text-2xl">
              {videoEnabled && videoAvailable ? (
                <VideoCameraIcon className="p-2 bg-black bg-
opacity-50 rounded-full w-8 h-8" />
              ) : (
                <VideoCameraSlashIcon className="p-2 bg-red-500
rounded-full w-8 h-8" />
              )}
            </button>
          </div>
          {!videoAvailable && (
            <ExclamationCircleIcon className="absolute bottom-4
right-4 text-red-500 w-8 h-8" />
          )}
        </div>
      </div>

      <div className="flex flex-col items-center justify-center
w-1/3 pl-8">
        <h2 className="text-2xl font-semibold mb-
4">{meeting?.title}</h2>
        <h2 className="text-2xl font-semibold mb-4">Ready to
join?</h2>
        <p className="mb-4">No one is here yet</p>
```

```
        <button
          onClick={onJoin}
          className="px-6 py-3 bg-purple-500 text-white rounded-
full w-48"
        >
          Join
        </button>
      </div>
    </div>
  );
};

export default PreparationScreen;
```

event-verse-frontend/src/components/Navbar/DropdownMenu.tsx

```
"use client";

import React from "react";
import {
  MoonIcon,
  CogIcon,
  GlobeAltIcon,
  ArrowRightCircleIcon,
  ArrowLeftCircleIcon,
  ChevronRightIcon,
} from "@heroicons/react/24/outline";
import { useAppSelector } from "@/hooks/redux";
import { selectMe } from "@/features/auth";

interface DropdownMenuProps {
  handleOpenLoginModal: () => void;
}

const DropdownMenu: React.FC<DropdownMenuProps> = ({
  handleOpenLoginModal,
}) => {
  const me = useAppSelector(selectMe);

  const toggleTheme = () => {
    const html = document.documentElement;
    if (html.classList.contains("dark")) {
      html.classList.remove("dark");
      localStorage.setItem("theme", "light");
    } else {
      html.classList.add("dark");
      localStorage.setItem("theme", "dark");
    }
  };

  return (
```

```
    <div className="absolute right-0 px-2 mt-4 bg-white text-
black rounded shadow-lg border-t border-zinc-200 dark:border-
zinc-700 dark:bg-zinc-900 dark:text-white">
      <ul className="py-2 w-max">
        <li className="flex items-center justify-between px-4
py-2 hover:bg-zinc-200 dark:hover:bg-zinc-700 cursor-pointer">
          <div className="flex items-center">
            <GlobeAltIcon className="w-5 h-5 mr-2" />
            <span>Language</span>
          </div>
          <ChevronRightIcon className="w-5 h-5 text-zinc-500
dark:text-zinc-400" />
        </li>

        <li
          className="flex items-center px-4 py-2 hover:bg-zinc-
200 dark:hover:bg-zinc-700 cursor-pointer"
          onClick={toggleTheme}
        >
          <MoonIcon className="w-5 h-5 mr-2" />
          <span className="mr-2">Dark theme</span>

          <div className="mx-2 w-10 h-5 flex items-center
rounded-full p-1 cursor-pointer border-2 border-transparent
dark:border-transparent bg-zinc-300 dark:bg-purple-600
transition-all duration-300">
            <div className="bg-white dark:bg-white w-4 h-4
rounded-full shadow-md transform duration-300 ease-in-out
translate-x-0 dark:translate-x-5"></div>
          </div>
        </li>
        <hr className="my-2" />
        {me ? (
          <>
            <li className="flex items-center px-4 py-2 hover:bg-
zinc-200 dark:hover:bg-zinc-700 cursor-pointer">
              <CogIcon className="w-5 h-5 mr-2" />
              <span>Settings</span>
            </li>
            <hr className="my-2 border-zinc-200 dark:border-
zinc-700" />
            <li className="flex items-center px-4 py-2 hover:bg-
zinc-200 dark:hover:bg-zinc-700 cursor-pointer">
              <ArrowLeftCircleIcon className="w-5 h-5 mr-2" />
              <span>Log Out</span>
            </li>
          </>
        ) : (
          <li
            className="flex items-center px-4 py-2 hover:bg-
zinc-200 dark:hover:bg-zinc-700 cursor-pointer"
            onClick={handleOpenLoginModal}
          >
```

```
            <ArrowRightCircleIcon className="w-5 h-5 mr-2" />
            <span>Log In</span>
          </li>
        )}
      </ul>
    </div>
  );
};

export default DropdownMenu;
```

event-verse-frontend/src/components/Navbar/Navbar.tsx

```
"use client";

import React, { useState } from "react";
import Lottie from "lottie-react";
import { useRouter } from "next/navigation";
import {
  MagnifyingGlassIcon,
  UserCircleIcon,
} from "@heroicons/react/24/outline";
import { useAppSelector } from "@/hooks/redux";
import { selectMe } from "@/features/auth";
import { useAuthModal } from "@/context/AuthModalContext";

import logo from "@/assets/logo.json";

import DropdownMenu from "./DropdownMenu";
import Button from "../ui/Button";

const Navbar = () => {
  const router = useRouter();

  const { openLoginModal } = useAuthModal();
  const me = useAppSelector(selectMe);
  const [isDropdownOpen, setIsDropdownOpen] = useState(false);

  const handleOpenLoginModal = () => {
    setIsDropdownOpen(false);
    openLoginModal();
  };

  const handleNavigateToMain = () => {
    router.push("/");
  };

  const toggleDropdown = () =>
setIsDropdownOpen(!isDropdownOpen);

  return (
    <nav className="flex items-center justify-between px-6 py-4
bg-white dark:bg-zinc-900 transition-colors duration-300 shadow-
```

```
md dark:shadow-lg border-b border-zinc-200 dark:border-zinc-
800">
      <div
        className="flex items-center space-x-4 cursor-pointer
transition duration-200 hover:opacity-80"
        onClick={handleNavigateToMain}
      >
        <Lottie animationData={logo} className="w-8 h-8"
loop={true} />

        <span className="text-lg font-bold">Event Verse</span>
      </div>

      <div className="flex-1 max-w-md mx-4">
        <div className="relative">
          <input
            type="text"
            placeholder="Search"
            className="w-full px-4 py-2 rounded-md bg-zinc-100
dark:bg-zinc-700 placeholder-zinc-500 dark:placeholder-zinc-400
focus:outline-none focus:ring-2 focus:ring-purple-500"
          />
          <MagnifyingGlassIcon className="absolute w-5 h-5
right-3 top-2.5" />
        </div>
      </div>

      <div className="flex items-center space-x-4">
        {!me && (
          <>
            <Button onClick={handleOpenLoginModal}>Log
In</Button>
            <Button className="bg-purple-600 hover:bg-purple-500
text-white">
              Sign Up
            </Button>
          </>
        )}

        <div className="relative">
          <button
            onClick={toggleDropdown}
            className="flex items-center space-x-2
focus:outline-none hover:text-purple-500"
          >
            <UserCircleIcon className="w-6 h-6" />
          </button>

          {isDropdownOpen && (
            <DropdownMenu
handleOpenLoginModal={handleOpenLoginModal} />
          )}
        </div>
```

```
      </div>
    </nav>
  );
};

export default Navbar;
```

event-verse-frontend/src/components/ui/Button.tsx

```
import React from "react";

interface ButtonProps extends
React.ButtonHTMLAttributes<HTMLButtonElement> {
  className?: string;
}

const Button: React.FC<ButtonProps> = ({ children, className,
...props }) => {
  const baseStyles =
    "font-semibold py-1 px-3 rounded transition-colors duration-
300";

  const computedClassName = `${baseStyles} ${
    className
      ? `${className}`
      : `bg-zinc-300 hover:bg-zinc-400 dark:bg-zinc-700
dark:hover:bg-zinc-600`
  }`;

  return (
    <button className={computedClassName} {...props}>
      {children}
    </button>
  );
};

export default Button;
```

event-verse-frontend/src/components/AuthWrapper.tsx

```
"use client";

import React, { useEffect, useState } from "react";
import { useAppDispatch } from "@/hooks/redux";
import { fetchMe } from "@/features/auth/slice";

const AuthWrapper: React.FC<{ children: React.ReactNode }> = ({
children }) => {
  const dispatch = useAppDispatch();
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    const handleFetchMe = async () => {
```

```
      try {
        await dispatch(fetchMe()).unwrap();
      } catch (error) {
        console.log(error);
      } finally {
        setIsLoading(false);
      }
    };
    handleFetchMe();
  }, [dispatch]);

  if (isLoading) {
    return (
      <div className="fixed inset-0 flex items-center justify-
center bg-black bg-opacity-50 z-50">
        <div className="animate-spin rounded-full h-16 w-16
border-t-4 border-blue-500 border-solid"></div>
      </div>
    );
  }

  return <>{children}</>;
};

export default AuthWrapper;
```

event-verse-frontend/src/components/CreateMeetingModal.tsx

```
"use client";

import React from "react";
import { useForm } from "react-hook-form";
import { yupResolver } from "@hookform/resolvers/yup";
import * as Yup from "yup";
import { useAppDispatch } from "@/hooks/redux";
import { createMeeting } from "@/features/meeting/slice";
import Button from "./ui/Button";

interface CreateMeetingModalProps {
  isOpen: boolean;
  onClose: () => void;
}

interface CreateMeetingFormData {
  title: string;
  startTime: Date;
  endTime?: Date | null;
}

const schema = Yup.object().shape({
  title: Yup.string().required("Title is required"),
  startTime: Yup.date().required("Start time is required"),
  endTime: Yup.date()
```

```
    .nullable()
    .transform((value, originalValue) => (originalValue === "" ?
null : value)),
});

const CreateMeetingModal: React.FC<CreateMeetingModalProps> = ({
  isOpen,
  onClose,
}) => {
  const dispatch = useAppDispatch();

  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm<CreateMeetingFormData>({
    resolver: yupResolver(schema),
  });

  const onSubmit = async (data: CreateMeetingFormData) => {
    await dispatch(
      createMeeting({
        title: data.title,
        startTime: data.startTime.toISOString(),
        endTime: data.endTime ? data.endTime.toISOString() :
null,
      })
    );

    onClose();
  };

  if (!isOpen) return null;

  return (
    <div className="fixed inset-0 flex items-center justify-
center bg-black bg-opacity-50 z-50">
      <div className="bg-white dark:bg-zinc-800 p-8 rounded-lg
shadow-lg w-full max-w-md">
        <h2 className="text-xl font-bold mb-4">Create
Meeting</h2>
        <form onSubmit={handleSubmit(onSubmit)}
className="space-y-4">
          <div>
            <label className="block text-sm font-medium mb-1">
              Meeting Title
            </label>
            <input
              {...register("title")}
              type="text"
              className="w-full px-3 py-2 border border-zinc-300
rounded dark:bg-zinc-700 dark:border-zinc-600 dark:text-white
focus:outline-none focus:ring-2 focus:ring-purple-500"
```

```
            />
            {errors.title && (
              <div className="text-red-500 text-sm mt-1">
                {errors.title.message}
              </div>
            )}
          </div>

          <div>
            <label className="block text-sm font-medium mb-
1">Start Time</label>
            <input
              {...register("startTime")}
              type="datetime-local"
              className="w-full px-3 py-2 border border-zinc-300
rounded dark:bg-zinc-700 dark:border-zinc-600 dark:text-white
focus:outline-none focus:ring-2 focus:ring-purple-500"
            />
            {errors.startTime && (
              <div className="text-red-500 text-sm mt-1">
                {errors.startTime.message}
              </div>
            )}
          </div>

          <div>
            <label className="block text-sm font-medium mb-1">
              End Time (Optional)
            </label>
            <input
              {...register("endTime")}
              type="datetime-local"
              className="w-full px-3 py-2 border border-zinc-300
rounded dark:bg-zinc-700 dark:border-zinc-600 dark:text-white
focus:outline-none focus:ring-2 focus:ring-purple-500"
            />
            {errors.endTime && (
              <div className="text-red-500 text-sm mt-1">
                {errors.endTime.message}
              </div>
            )}
          </div>

          <div className="flex justify-end space-x-4">
            <Button onClick={onClose} className="bg-zinc-300
dark:bg-zinc-700">
              Cancel
            </Button>
            <Button
              type="submit"
              className="bg-purple-600 hover:bg-purple-500 text-
white"
            >
```

```
              Create
            </Button>
          </div>
        </form>
      </div>
    </div>
  );
};

export default CreateMeetingModal;
```

event-verse-frontend/src/components/HorizontalMeetingList.tsx

```
"use client";

import React, { useRef } from "react";
import { useRouter } from "next/navigation";
import { Meeting } from "@/types/Meeting";
import { useAuthModal } from "@/context/AuthModalContext";
import { selectMe } from "@/features/auth";
import { useAppSelector } from "@/hooks/redux";
import { ChevronLeftIcon, ChevronRightIcon } from
"@heroicons/react/24/solid";

interface MeetingListProps {
  meetings: Meeting[];
}

const HorizontalMeetingList: React.FC<MeetingListProps> = ({
meetings }) => {
  const router = useRouter();
  const { openLoginModal } = useAuthModal();
  const me = useAppSelector(selectMe);

  const containerRef = useRef<HTMLDivElement>(null);

  const navigateToMeeting = (id: string) => {
    if (!me) {
      openLoginModal();
      return;
    }
    router.push(`/meetings/${id}`);
  };

  const navigateToProfile = (userId: string) => {
    router.push(`/profile/${userId}`);
  };

  const navigateToTag = (tag: string) => {
    router.push(`/search?tag=${tag}`);
  };

  const scrollLeft = () => {
```

```
    if (containerRef.current) {
      const firstChild = containerRef.current.children[0] as
HTMLElement;
      const scrollAmount = firstChild.offsetWidth + 13;
      containerRef.current.scrollBy({
        left: -scrollAmount,
        behavior: "smooth",
      });
    }
  };

  const scrollRight = () => {
    if (containerRef.current) {
      const firstChild = containerRef.current.children[0] as
HTMLElement;
      const scrollAmount = firstChild.offsetWidth + 13;
      containerRef.current.scrollBy({
        left: scrollAmount,
        behavior: "smooth",
      });
    }
  };

  return (
    <div className="relative w-full">
      <div
        className="absolute left-0 top-1/2 transform -translate-
y-1/2 bg-zinc-800 hover:bg-purple-400 p-2 rounded-full cursor-
pointer z-10"
        onClick={scrollLeft}
      >
        <ChevronLeftIcon className="h-6 w-6 text-white" />
      </div>
      <div
        className="absolute right-0 top-1/2 transform -
translate-y-1/2 bg-zinc-800 hover:bg-purple-400 p-2 rounded-full
cursor-pointer z-10"
        onClick={scrollRight}
      >
        <ChevronRightIcon className="h-6 w-6 text-white" />
      </div>

      <div
        ref={containerRef}
        className="flex overflow-x-scroll gap-4 mx-16 no-
scrollbar"
      >
        {meetings.map((meeting: Meeting) => (
          <div key={meeting.id} className="min-w-[337px] max-w-
[300px]">
            <div
              className="relative aspect-video overflow-hidden
rounded-lg"
```

```
              onClick={() => navigateToMeeting(meeting.id)}
            >
              <img

src="https://via.placeholder.com/256x144.png?text=Preview"
                alt="Meeting Preview"
                className="w-full h-full object-cover cursor-
pointer"
              />
            </div>

            <div className="mt-2">
              <h3
                className="text-lg font-semibold truncate
cursor-pointer hover:text-purple-400"
                onClick={() => navigateToMeeting(meeting.id)}
              >
                {meeting.title}
              </h3>

              <p
                className="text-sm text-zinc-400"
                //   onClick={() =>
navigateToProfile(meeting.creatorId)}
              >
                by{" "}
                <span className="cursor-pointer hover:text-
purple-400">
                  John Doe
                </span>
                {/* <span className="text-
white">{meeting.creatorName}</span> */}
              </p>

              <div className="flex flex-wrap gap-2 mt-2">
                {/* {meeting.tags.map((tag) => (
              <span
                key={tag}
                className="px-2 py-1 text-xs bg-zinc-700
rounded-full hover:bg-purple-400"
                onClick={() => navigateToTag(tag)}
              >
                {tag}
              </span>
            ))} */}
              </div>
            </div>
          </div>
        ))}
      </div>
    </div>
  );
};
```

```
export default HorizontalMeetingList;

event-verse-frontend/src/components/LoginModal.tsx

"use client";

import React, { useState } from "react";
import {
  XCircleIcon,
  EyeIcon,
  EyeSlashIcon,
} from "@heroicons/react/24/outline";

const LoginModal = ({ onClose }: { onClose: () => void }) => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [isPasswordVisible, setIsPasswordVisible] =
useState(false);

  const togglePasswordVisibility = () => {
    setIsPasswordVisible(!isPasswordVisible);
  };

  const isFormValid = email.length > 0 && password.length > 0;

  return (
    <div className="fixed inset-0 flex items-center justify-
center bg-black bg-opacity-50 z-50">
      <div className="bg-white dark:bg-zinc-800 p-8 rounded-lg
shadow-lg w-96 relative">
        <div className="flex items-center justify-between mb-6">
          <h2 className="text-2xl font-bold">Log in to Event
Verse</h2>
          <button onClick={onClose}>
            <XCircleIcon className="w-5 h-5" />
          </button>
        </div>

        <input
          type="text"
          placeholder="Email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          className="w-full mb-3 p-2 border border-zinc-300
rounded focus:outline-none focus:ring-2 focus:ring-purple-500
dark:bg-zinc-700 dark:border-zinc-600 dark:focus:ring-purple-500
dark:text-white"
        />
        <div className="relative w-full mb-2">
          <input
            type={isPasswordVisible ? "text" : "password"}
            placeholder="Password"
```

```
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            className="w-full p-2 border border-zinc-300 rounded
focus:outline-none focus:ring-2 focus:ring-purple-500 dark:bg-
zinc-700 dark:border-zinc-600 dark:focus:ring-purple-500 pr-10
dark:text-white"
          />
          <button
            type="button"
            onClick={togglePasswordVisibility}
            className="absolute inset-y-0 right-3 flex items-
center"
          >
            {isPasswordVisible ? (
              <EyeSlashIcon className="w-5 h-5" />
            ) : (
              <EyeIcon className="w-5 h-5" />
            )}
          </button>
        </div>

        <p className="text-sm text-purple-400 dark:text-purple-
400 hover:underline cursor-pointer mb-4">
          Trouble logging in?
        </p>

        <button
          disabled={!isFormValid}
          className={`w-full py-2 font-semibold rounded text-
white ${
            isFormValid
              ? "bg-purple-400 hover:bg-purple-500"
              : "bg-zinc-700 cursor-not-allowed"
          }`}
        >
          Log in
        </button>
        <button className="mt-4 w-full py-2 rounded text-purple-
400 hover:text-zinc-800 hover:bg-zinc-300 dark:hover:bg-zinc-600
dark:hover:text-white">
          Don’t have an account? Sign up
        </button>

        <div className="my-6 border-t border-zinc-300
dark:border-zinc-600"></div>
        <p className="text-center text-sm mb-4">Or log in
with</p>

        <div className="space-y-3">
          <button
            onClick={() => {
              window.location.href =
`${process.env.NEXT_PUBLIC_BACKEND_URL}/auth/google`;
```

```
            }}
            className="flex items-center justify-center space-x-
2 border border-zinc-300 rounded p-2 w-full dark:border-zinc-600
hover:bg-zinc-100 dark:hover:bg-zinc-700"
          >
            <img

src="https://upload.wikimedia.org/wikipedia/commons/thumb/c/c1/G
oogle_%22G%22_logo.svg/150px-Google_%22G%22_logo.svg.png"
              alt="Google"
              className="w-5 h-5"
            />
            <span className="text-sm font-semibold">Continue
with Google</span>
          </button>

          <button className="flex items-center justify-center
space-x-2 border border-zinc-300 rounded p-2 w-full dark:border-
zinc-600 hover:bg-zinc-100 dark:hover:bg-zinc-700">
            <img

src="https://upload.wikimedia.org/wikipedia/commons/f/fa/Apple_l
ogo_black.svg"
              alt="Apple"
              className="w-5 h-5"
            />
            <span className="text-sm font-semibold">Continue
with Apple</span>
          </button>
        </div>
      </div>
    </div>
  );
};

export default LoginModal;
```

event-verse-frontend/src/components/ThemeInitializer.tsx

```
"use client";

import { useEffect } from "react";

const ThemeInitializer = () => {
  useEffect(() => {
    const html = document.documentElement;
    const savedTheme = localStorage.getItem("theme");

    if (savedTheme === "dark") {
      html.classList.add("dark");
    } else {
      html.classList.remove("dark");
    }
```

```
  }, []);

  return null;
};

export default ThemeInitializer;
```

event-verse-frontend/src/context/AuthModalContext.tsx

```
"use client";

import React, { createContext, useContext, useState } from
"react";
import LoginModal from "@/components/LoginModal";

interface AuthModalContextProps {
  isLoginModalOpen: boolean;
  openLoginModal: () => void;
  closeLoginModal: () => void;
}

const AuthModalContext = createContext<AuthModalContextProps |
undefined>(
  undefined
);

export const AuthModalProvider: React.FC<{ children:
React.ReactNode }> = ({
  children,
}) => {
  const [isLoginModalOpen, setIsLoginModalOpen] =
useState(false);

  const openLoginModal = () => setIsLoginModalOpen(true);
  const closeLoginModal = () => setIsLoginModalOpen(false);

  return (
    <AuthModalContext.Provider
      value={{ isLoginModalOpen, openLoginModal, closeLoginModal
}}
    >
      {children}
      {isLoginModalOpen && <LoginModal onClose={closeLoginModal}
/>}
    </AuthModalContext.Provider>
  );
};

export const useAuthModal = () => {
  const context = useContext(AuthModalContext);
  if (!context) {
    throw new Error("useAuthModal must be used within an
AuthModalProvider");
```

```
  }
  return context;
};

event-verse-frontend/src/context/WebSocketProvider.rsx

"use client";

import { selectMe } from "@/features/auth";
import { useAppSelector } from "@/hooks/redux";
import React, { createContext, useContext, useEffect, useState }
from "react";
import { io, Socket } from "socket.io-client";

interface SocketContextValue {
  socket: Socket | null;
}

const WebSocketContext = createContext<SocketContextValue |
null>(null);

export const useWebSocket = () => {
  const context = useContext(WebSocketContext);
  if (!context) {
    throw new Error("useWebSocket must be used within a
WebSocketProvider");
  }
  return context;
};

export const WebSocketProvider: React.FC<{ children:
React.ReactNode }> = ({
  children,
}) => {
  const me = useAppSelector(selectMe);
  const [socket, setSocket] = useState<Socket | null>(null);

  useEffect(() => {
    if (me && !socket) {
      const socketInstance = io(process.env.NEXT_PUBLIC_WS_URL,
{
        query: { userId: me.id },
        transports: ["websocket"],
      });

      socketInstance.on("connect", () => {
        console.log("Socket.IO connected");
      });

      socketInstance.on("disconnect", () => {
        console.log("Socket.IO disconnected");
      });
```

```
      socketInstance.on("message", (data) => {
        console.log("Message received:", data);
      });

      setSocket(socketInstance);

      return () => {
        socketInstance.disconnect();
        setSocket(null);
      };
    }
  }, [me]);

  return (
    <WebSocketContext.Provider value={{ socket }}>
      {children}
    </WebSocketContext.Provider>
  );
};
```

event-verse-frontend/src/features/auth
event-verse-frontend/src/features/meeting

event-verse-frontend/src/hooks/redux.ts

```
import { TypedUseSelectorHook, useDispatch, useSelector } from
"react-redux";
import { AppDispatch, RootState } from "../store";

export const useAppDispatch = () => useDispatch<AppDispatch>();
export const useAppSelector: TypedUseSelectorHook<RootState> =
useSelector;
```

event-verse-frontend/src/lib/axios.ts

```
import axios, { CreateAxiosDefaults } from "axios";

const axiosDefaults: CreateAxiosDefaults = {};
if (process.env.NEXT_PUBLIC_BACKEND_URL) {
  axiosDefaults.baseURL = process.env.NEXT_PUBLIC_BACKEND_URL;
}

const api = axios.create(axiosDefaults);

api.interceptors.request.use((config) => {
  Object.keys(config.params || {}).map((key) => {
    if (
      config.params[key] === undefined ||
      config.params[key] === null ||
      (typeof config.params[key] === "string" &&
        !(config.params[key] as string).length)
    ) {
      delete config.params[key];
```

```
    }
  });

  config.withCredentials = true;
  config.headers["Access-Control-Allow-Credentials"] = true;
  config.headers["Cache-Control"] = "no-cache";
  return config;
});

api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response) {
      return Promise.reject(error.response.data);
    }
    return Promise.reject(error);
  }
);

export default api;
```

event-verse-frontend/src/store/index.ts

```
import authReducer from "@/features/auth";
import meetingReducer from "@/features/meeting";
import { configureStore } from "@reduxjs/toolkit";

const store = configureStore({
  reducer: {
    auth: authReducer,
    meeting: meetingReducer,
  },
});

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;

export default store;
```

event-verse-frontend/src/types/Meeting.ts

```
import {
  DtlsParameters,
  IceCandidate,
  IceParameters,
} from "mediasoup-client/lib/types";

export type Meeting = {
  id: string;
  title: string;
  startTime: string;
  endTime: string;
};
```

```
export type CreateTransportResponseParams = {
  params: {
    id: string;
    iceParameters: IceParameters;
    iceCandidates: IceCandidate[];
    dtlsParameters: DtlsParameters;
  };
};
export type CreateTransportResponseError = { error: string |
unknown };
export type CreateTransportResponse =
  | CreateTransportResponseParams
  | CreateTransportResponseError;
```

event-verse-frontend/src/types/User.ts

```
export type AuthProviderType = "GOOGLE" | "APPLE" | "GITHUB";

export type User = {
  id: string;
  email: string;
  fullName: string;
  avatar: string;
  authProviders: AuthProvider[];
};

export type AuthProvider = {
  email: string;
  provider: AuthProviderType;
  providerId: string;
};
```

event-verse-frontend/src/utils/delay.ts

```
export const delay = (ms: number) =>
  new Promise((resolve) => setTimeout(resolve, ms));
```

event-verse-frontend/src/utils/formatter.ts

```
export const dateFormatter = new Intl.DateTimeFormat("en-GB", {
  year: "numeric",
  month: "long",
  day: "numeric",
  hour: "2-digit",
  minute: "2-digit",
  second: "2-digit",
  timeZone: "UTC",
});
```

event-verse-frontend/src/utils/index.ts

```
export * from "./delay";
```

```
export * from "./formatter";
```

event-verse-frontend/tailwind.config.ts

```
import type { Config } from "tailwindcss";

export default {
  content: [
    "./src/pages/**/*.{js,ts,jsx,tsx,mdx}",
    "./src/components/**/*.{js,ts,jsx,tsx,mdx}",
    "./src/app/**/*.{js,ts,jsx,tsx,mdx}",
  ],
  darkMode: "class",
  theme: {
    extend: {},
  },
  plugins: [],
} satisfies Config;
```

event-verse-frontend/vercel.json

```
{
  "build": {
    "env": {
      "NPM_FLAGS": "--legacy-peer-deps"
    }
  }
}
```