Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: «Інтелектуальна система управління комунікацією засобами react»

Виконав: студент VI курсу групи КН-62м
Спеціальності:
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Баб'як Ю. І.
(прізвище та ініціали)

Керівник Яцишин С.І.
(прізвище та ініціали)

Рецензент Чаєковський О.Т.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук
_____ Борецька І.Б.
"10" грудня 2025 року

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

Баб'як Юрій Ігорович
(прізвище, ім'я, по батькові)

1. Тема роботи: Інтелектуальна система управління комунікацією засобами react

Керівник роботи *Яцишин С.І. к.т.н., доцент*,
(прізвищє, ім'я, по батькові, науковий ступінь, вченє звання)
затверджені наказом вищого навчального закладу від "29" квітня 2025 року №С-288
2. Термін подання студентом роботи 10.12.2025
3. Вихідні дані до роботи: створення інтелектуальної система управління комунікацією засобами react
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Розділ 1. Стан проблемної області
Розділ 2. Інформаційне забезпечення
Розділ 3. Математичне забезпечення
Розділ 4. Програмне забезпечення
Розділ 5. Розроблення стартап-проєкту
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалу до доповіді
6. Дата видачі завдання 01.05.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	01.05.25-17.06.25	Виконано
2.	Постановка задачі і її формалізація	18.06.25-01.07.25	Виконано
3.	Виконання вхідного етапу технології	02.07.25-24.08.25	Виконано
4.	Реалізація головних алгоритмів проекту	25.08.25-06.10.25	Виконано
5.	Виконання етапу відлагодження проєкту	07.10.25-04.11.25	Виконано
6.	Виконання етапу впровадження та випуску бета-версії.	05.11.25-17.11.25	Виконано
7.	Оформлення записки до дипломного проекту.	18.11.25-08.12.25	Виконано

Студент _____ Баб'як Ю.І.
(підпис) (прізвище та ініціали)

Керівники роботи _____ Яцишин С.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська кваліфікаційна робота присвячена розробці та дослідженню **Інтелектуальної системи управління комунікацією засобами віртуальної зустрічі**. Робота складається з 74 сторінок пояснювальної записки, містить 18 ілюстрацій та 22 джерел літератури.

Об'єктом дослідження є процес організації ефективної та реалістичної комунікації в інтерактивних віртуальних середовищах.

Метою роботи є створення інтелектуальної системи "Віртуальна зустріч", яка забезпечує новий, захопливий досвід спільної роботи та взаємодії між учасниками за рахунок імерсивного 3D-середовища.

У ході роботи була розроблена клієнт-серверна система, що функціонує в повністю інтерактивному 3D-просторі. Ключовими функціями системи є: зв'язок у режимі реального часу (голосовий, відео- та текстовий чат) через P2P-з'єднання, можливість спільного доступу до екрана та управління аватарами у віртуальному просторі. Інтелектуальність системи полягає в ефективному управлінні синхронізацією стану 3D-середовища та реалістичною імітацією ефекту присутності.

Технологічний стек включає **ReactJS** та бібліотеки **React-three-fiber (R3F)** і **Three.js** для високопродуктивного 3D-рендерингу. Для забезпечення комунікації в реальному часі використовується **PeerJS** (на основі WebRTC), а для синхронізації дій та стану середовища — **Socket.io**.

Ключові слова: Інтелектуальна система, віртуальна реальність, 3D-комунікація, React-three-fiber, Three.js, WebRTC, Socket.io, PeerJS.

ABSTRACT

The Master's thesis is dedicated to the development and research of an **Intelligent Communication Management System through a Virtual Meeting Environment**. The thesis consists of 74 pages of explanatory text, includes 18 illustrations, and references 22 sources.

The **object of the study** is the process of organizing effective and realistic communication within interactive virtual environments.

The **goal of the work** is to create an intelligent system called "Virtual Meet," which provides a new, engaging experience for collaborative work and interaction among participants through an immersive 3D environment.

A client-server system was developed to operate in a fully interactive 3D space. Key system features include: real-time communication (voice, video, and text chat) via P2P connections, screen sharing capabilities, and avatar control within the virtual space. The system's intelligence is defined by its effective management of 3D environment state synchronization and realistic imitation of the presence effect.

The **technology stack** includes **ReactJS** and the **React-three-fiber (R3F)** and **Three.js** libraries for high-performance 3D rendering. **PeerJS** (based on WebRTC) is used to ensure real-time communication, and **Socket.io** is employed for synchronizing actions and the environment state.

Keywords: Intelligent System, Virtual Reality, 3D Communication, React-three-fiber, Three.js, WebRTC, Socket.io, PeerJS.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробка **Інтелектуальної системи управління комунікацією засобами віртуальної зустрічі** зосереджена на створенні високоімерсивного та реалістичного середовища для спільної роботи, що є якісним кроком вперед порівняно з традиційними 2D-відеоконференціями. Основна **мета** проєкту — забезпечити учасникам відчуття **присутності** та інтерактивності, необхідних для ефективної комунікації на відстані.

Система функціонує як клієнт-серверний додаток, де користувачі взаємодіють через веббраузер. **Фронтенд** системи є повноцінним **3D-середовищем**, реалізованим на базі фреймворку **ReactJS** у поєднанні з потужними бібліотеками **Three.js** та **React-three-fiber (R3F)**. R3F дозволяє інтегрувати складну 3D-графіку в реактивний архітектурний шаблон, спрощуючи управління станом та оптимізуючи рендеринг. Це забезпечує необхідну продуктивність, мінімально 30 кадрів на секунду, що є критичним для плавної навігації.

Центральна **функціональність** системи включає кілька ключових аспектів. По-перше, це **навігація** у віртуальному просторі: користувачі можуть переміщатися (W, A, S, D) і орієнтувати свій погляд, як у типовій 3D-грі, що створює просторове відчуття. По-друге, це **комунікація в режимі реального часу**, яка реалізується через дві незалежні, але взаємодоповнюючі технології. **Голосовий та відеочат** використовують технологію **WebRTC** та бібліотеку **PeerJS**, що дозволяє встановлювати прямі **Peer-to-Peer (P2P)** з'єднання між учасниками для мінімізації затримки (latency) і забезпечення високої якості зв'язку (затримка має бути менше 300 мс).

По-третє, для забезпечення **інтелектуального управління** віртуальним простором застосовується технологія **Socket.io**. Цей механізм служить для надійної та швидкої **синхронізації стану** усіх клієнтів. Через Socket.io передаються дані про точне **положення** та **орієнтацію** аватарів учасників, їхні дії (наприклад, включення мікрофона або відео, що також доступно через гарячі клавіші Ctrl+Shift+Z та Ctrl+Shift+X), а також керування функцією **спільного доступу до екрана**. Саме

швидкість і точність цієї синхронізації (затримка не більше 100 мс) формує відчуття реалістичної **колективної присутності**.

Система також включає функції **спільної роботи**, такі як текстовий чат та можливість виведення екрана одного з учасників на спеціальний 3D-монітор у віртуальній кімнаті, що є важливим для проведення презентацій. Хоча поточний прототип не підтримує мобільні пристрої, архітектура передбачає подальше **масштабування** та розширення функціоналу, включаючи додавання **просторового аудіо** (де гучність голосу залежить від відстані між аватарами) та складніших анімацій аватарів, що посилить інтелектуальність системи та її здатність імітувати реальне спілкування.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ІСУ — Інтелектуальна Система Управління

ВЗ — Віртуальна Зустріч

3D — Тривимірний (стосується графіки та просторового середовища)

VR — Віртуальна Реальність

AR — Доповнена Реальність

R3F — React-three-fiber (бібліотека для використання Three.js у React)

Three.js — Високорівнева бібліотека **JavaScript** для 3D-графіки на основі WebGL

WebGL — Web Graphics Library (програмний інтерфейс для рендерингу 3D-графіки в браузері)

P2P — Peer-to-Peer (Однорангова мережева архітектура)

WebRTC — Web Real-Time Communication (технологія для обміну аудіо/відеоданими в реальному часі)

VoIP — Voice over IP (технологія передачі голосу через інтернет-протокол)

API — Application Programming Interface (Інтерфейс програмування додатків)

FPS — Frames Per Second (Кількість кадрів на секунду)

UX — User Experience (Досвід користувача)

UI — User Interface (Користувацький інтерфейс)

DOM — Document Object Model (Об'єктна модель документа)

CSS — Cascading Style Sheets (Мова опису стилів)

JSON — JavaScript Object Notation (Формат обміну даними)

СДЕ — Спільний Доступ до Екрана

SDK — Software Development Kit (Комплект для розробки програмного забезпечення)

ВСТУП

Стрімкий розвиток глобальних комунікацій та перехід значної частини професійної та освітньої діяльності до дистанційних форматів зумовили критичну потребу в інструментах, які здатні відтворити **ефективність та емоційну насиченість** спілкування, властиву фізичній присутності. Традиційні платформи для відеоконференцій, що оперують у двовимірному просторі, вичерпали свій потенціал у забезпеченні повноцінного **ефекту присутності** та часто призводять до так званої "втоми від екрана". Таким чином, виникає актуальне завдання розробки нового покоління систем — **Інтелектуальних систем управління комунікацією**, які використовують переваги тривимірних (3D) віртуальних середовищ.

Актуальність цієї магістерської роботи полягає у необхідності створення такої системи, яка забезпечить не лише передачу аудіо- та відеопотоків, а й **інтелектуально управлятиме просторовою синхронізацією** учасників та їхньою взаємодією у реалістичному віртуальному просторі. На відміну від статичних рішень, наша система має динамічно реагувати на дії користувачів, імітуючи природні умови зустрічі.

Метою даного дослідження є розробка та експериментальне дослідження функціонального прототипу **Системи віртуальної зустрічі**, що використовує передові веб-технології для надання захопливого та ефективного досвіду комунікації.

Для досягнення цієї мети було поставлено низку **завдань**. Насамперед, необхідно провести глибокий **теоретичний аналіз** архітектур інтелектуальних систем та технологій 3D-візуалізації, зокрема **WebGL, Three.js та React-three-fiber (R3F)**, обґрунтувавши їхній вибір для реалізації. Далі, ключовим завданням є **програмна реалізація** системи, яка включає розробку ядра 3D-клієнта, впровадження надійного **P2P-зв'язку** для голосового та відеочату за допомогою **WebRTC/PeerJS**, а також реалізацію ефективної системи **синхронізації стану** (позиції, руху, спільний доступ до екрана) на базі **Socket.io**. Завершальним етапом є **експериментальне дослідження** розробленої системи для об'єктивної оцінки її продуктивності (FPS) та стабільності комунікаційних каналів.

Об'єктом дослідження виступають процеси, що забезпечують багатокористувацьку комунікацію та спільну роботу в інтерактивних віртуальних просторах. Відповідно, **предметом дослідження** є методи та програмні засоби архітектурного і програмного забезпечення для створення подібних інтелектуальних систем управління.

У процесі роботи використовуються методи **системного аналізу** для проєктування структури системи, **об'єктно-орієнтоване програмування** для практичної реалізації на ReactJS, а також методи **експериментального моделювання** та тестування для верифікації функціональних та нефункціональних вимог.

Наукова новизна роботи полягає у створенні архітектурної моделі, яка оптимально поєднує можливості **React-three-fiber** для 3D-рендерингу та **PeerJS/Socket.io** для гібридного управління комунікацією (P2P для медіа, Socket.io для 3D-стану). Це забезпечує мінімальну затримку та максимальний ефект присутності. **Практична цінність** системи полягає у наданні готового прототипу, придатного для використання у сфері дистанційної освіти, корпоративних нарад та подальших досліджень імерсивних інтерфейсів.

Магістерська робота має чітку структуру, що включає вступ, три основні розділи, присвячені теорії, розробці та дослідженню, висновки, а також перелік скорочень та список використаних джерел.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Концептуальні основи інтелектуальних систем управління комунікацією

Сучасний етап розвитку інформаційно-комунікаційних технологій (ІКТ) характеризується переходом від простих засобів передачі інформації до складних **Інтелектуальних Систем Управління (ІСУ)**, які здатні не лише забезпечувати зв'язок, а й активно формувати та моделювати середовище взаємодії [1]. У контексті віртуальних зустрічей, інтелектуальність системи "Віртуальна зустріч" проявляється у її здатності до **ефективного управління просторовою комунікацією** та відтворення психологічного **ефекту присутності** [2], що є критичним для повноцінної співпраці [3]. Наша система належить до класу **людино-машинних** ІСУ, де програмний комплекс використовує принципи моделювання реального світу для підвищення когнітивної залученості користувачів [4].

Основна проблема традиційних 2D-платформ полягає у відсутності **просторової орієнтації** та руйнуванні природного процесу невербальної комунікації [5]. Людина, яка дивиться на екран, не відчуває, що вона **перебуває** у спільному просторі [6]. Натомість, розроблена система використовує **3D-середовище**, яке є потужним засобом для створення **Ефекту Присутності** (*Sense of Presence*) [7]. Цей ефект визначається як суб'єктивне, психологічне відчуття перебування у віртуальному середовищі, а не у фізичній локації користувача [8]. Ключовими факторами, що впливають на цей ефект, є можливість **просторової навігації**, **візуалізація аватара** учасника [9] та імітація природного **спільного простору** (*Shared Space*) [10]. У 3D-середовищі, рух аватара, його орієнтація та відносне положення до інших учасників перетворюються на нові канали невербальної комунікації, які ІСУ повинна бездоганно обробляти [11].

Для забезпечення безперервності та цілісності цього ефекту, архітектура системи повинна бути гібридною та інтелектуальною у розподілі завдань [12]. З одного боку, обробка об’ємних **медіапотоків** (голос, відео) вимагає максимальної децентралізації. З цією метою система використовує технологію **WebRTC** [13] та бібліотеку **PeerJS**, які встановлюють прямі **Peer-to-Peer (P2P)** з’єднання між

учасниками [14]. Цей децентралізований канал є життєво необхідним для мінімізації затримки (*latency*), оскільки будь-яка затримка у голосовому зв'язку понад **300 мс** руйнує природний ритм розмови [15].

З іншого боку, управління динамічним **станом 3D-середовища** (точні координати аватарів, їхнє обертання, події спільного доступу до екрана, текстовий чат) вимагає надійного та високошвидкісного **централізованого каналу синхронізації** [16]. Для цього використовується протокол **Socket.io** (на основі WebSockets) [17]. Інтелектуальне управління тут полягає у забезпеченні **миттєвої розсилки** керуючих даних усім підключеним клієнтам, що гарантує, що візуальне відображення дій одного користувача для всіх інших відбудеться із затримкою, що не перевищує **100 мс** [18]. Таке двоканальне архітектурне рішення є фундаментом надійності та забезпечує необхідну умову для створення ефекту присутності: затримка синхронізації має бути значно меншою за час людського сприйняття [19].

Реалізація цієї складної інтерактивної системи у веб-середовищі потребує використання високоефективних інструментів для рендерингу. Вибір **Three.js** як основи для 3D-графіки та **React-three-fiber (R3F)** для інтеграції цієї графіки в архітектуру **ReactJS** є стратегічно важливим. R3F дозволяє управляти складними 3D-об'єктами та сценою за допомогою декларативної парадигми React, що значно спрощує розробку, особливо в умовах динамічної багатокористувацької сцени [20]. Завдяки цьому поєднанню, система може досягти цільового показника продуктивності **30 кадрів на секунду (FPS) і вище**, забезпечуючи плавність руху та візуальної взаємодії, що є невід'ємною складовою інтелектуальної системи, орієнтованої на користувацький досвід.

Таким чином, інтелектуальність системи "Віртуальна зустріч" полягає у синергії децентралізованої медіа-комунікації та централізованого управління просторовим станом, що разом забезпечує реалістичний, захопливий та ефективний комунікаційний досвід у 3D-середовищі.

1.2. Огляд існуючих рішень для віртуальної співпраці та аналіз аналогів

Переважна більшість сучасних інструментів для дистанційної комунікації належить до класу **2D-відеоконференц-систем** [21], таких як Zoom, Microsoft Teams чи Google Meet. Ці платформи зарекомендували себе як високонадійні, масштабовані та зручні для швидкої організації зустрічей та нарад. Їхня ключова перевага полягає у фокусі на аудіо- та відеопередачі, а також на функціях спільного доступу до екрана та запису. Проте, з точки зору завдань, поставлених у даній роботі, їхня ефективність обмежена. Вони не створюють **просторового контексту**, що є критичним для повноцінної співпраці. Учасники бачать лише сітку вікон, що значно знижує рівень **невербальної комунікації** і призводить до відчуття ізоляції та вищезгаданої "втоми від екрана" [5]. Відсутність спільного 3D-простору унеможливлює імітацію природної взаємодії, коли учасники можуть "підійти" до об'єкта обговорення чи спільного віртуального столу.

Огляд імерсивних платформ та їхній технологічний бар'єр

На противагу 2D-рішенням, існують **імерсивні віртуальні платформи**, які намагаються відтворити ефект присутності. Ці аналоги можна розділити на дві основні категорії.

Повноцінні VR-платформи, такі як AltspaceVR (Microsoft) або VRChat, пропонують найвищий ступінь занурення. Вони забезпечують реалістичне управління аватарами, включаючи рух рук та міміку, а також повноцінне **просторове аудіо** [10]. Однак, їхній головний недолік — це **технологічний бар'єр входу**. Вони вимагають використання спеціалізованого та дорогого обладнання (VR-шоломи, контролери), що суттєво обмежує їхнє масове застосування у повсякденній роботі та навчанні.

Гібридні 2.5D/3D-платформи, наприклад, Gather Town, використовують піксельну або ізометричну графіку для створення відчуття "простору" та "переміщення", де зв'язок активується лише при наближенні аватарів. Ці рішення є більш доступними, оскільки працюють у браузері, але рівень їхньої **візуальної**

реалістичності та **3D-інтерактивності** значно поступається цілям, поставленим у даній магістерській роботі.

Проведений аналіз показує, що на ринку існує **технологічна ніша** між високодоступними, але неімерсивними 2D-системами та високоімерсивними, але апаратно вимогливими VR-платформами. Саме в цій ніші позиціонується розроблена система **"Віртуальна зустріч"** [22].

Наш підхід, заснований на **React-three-fiber (R3F)** та **Three.js**, дозволяє створити повноцінне, високоякісне 3D-середовище, яке доступне безпосередньо через стандартний веббраузер без додаткового обладнання [20]. Використання **WebRTC** та **PeerJS** для медіапотоків забезпечує якість зв'язку на рівні 2D-аналогів, тоді як **Socket.io** дозволяє впровадити інтелектуальне управління синхронізацією 3D-стану, яке відсутнє у традиційних системах [17].

Таким чином, у порівняльній таблиці (яка буде представлена у наступних підрозділах) буде чітко видно, що система "Віртуальна зустріч" поєднує **масову доступність** (браузерний клієнт, без VR-шоломів) з **високим ступенем 3D-інтерактивності** та реалістичності, усуваючи ключові недоліки існуючих аналогів і задовольняючи вимоги до **Інтелектуальної системи управління комунікацією** [19].

1.3. Технологічний фундамент: обґрунтування вибору стеку

Успішна реалізація **Інтелектуальної системи управління комунікацією** вимагає вибору технологічного стеку, який здатен одночасно вирішувати завдання високопродуктивного 3D-рендерингу, надійного зв'язку в реальному часі та ефективної синхронізації стану. Обрана архітектура ґрунтується на чотирьох ключових компонентах, кожен з яких відіграє незамінну роль у забезпеченні функціональності системи "Віртуальна зустріч".

Візуалізація та 3D-Рендеринг

Оскільки метою проєкту є створення імерсивного середовища, що працює у стандартному веббраузері, основою для графіки слугує технологія **WebGL** [1], яка є стандартом для апаратного прискорення 3D-графіки в мережі. Як високорівнева бібліотека для роботи з WebGL, була обрана **Three.js** [20]. Three.js надає необхідний

функціонал для створення та управління сценою, камерою, освітленням та складними геометричними об'єктами. Однак, пряме управління великою 3D-сценою лише засобами Three.js є складним та імперативним, особливо в умовах, коли стан сцени динамічно змінюється багатьма користувачами.

Для вирішення цієї проблеми було обрано **React-three-fiber (R3F)** [20]. R3F є високопродуктивним рендерером, який дозволяє використовувати декларативний підхід фреймворку **ReactJS** для управління об'єктами Three.js. Цей вибір є стратегічним, оскільки він дозволяє інтегрувати управління станом 3D-об'єктів (позиція, орієнтація, анімація) безпосередньо у реактивний цикл **React**, забезпечуючи бездоганну синхронізацію між даними про користувача та їхнім візуальним представленням у 3D-сцені. Це значно спрощує розробку багатокористувацьких інтерактивних елементів і сприяє досягненню цільового показника продуктивності (30 FPS і вище) [18].

Комунікація в Реальному Часі

Для реалізації двостороннього зв'язку (голос та відео) в режимі реального часу, що є обов'язковою вимогою до будь-якої комунікаційної системи, обрано відкритий стандарт **WebRTC** (Web Real-Time Communication) [13]. WebRTC є єдиною технологією, що дозволяє встановлювати прямі **Peer-to-Peer (P2P)** з'єднання між браузерами без посередництва центрального сервера для передачі медіапотоків. Це гарантує мінімальну затримку і є критичним для підтримання природного діалогу [15].

Для спрощення процесу встановлення P2P-з'єднань та управління сигналізацією (обмін IP-адресами та можливостями) використовується бібліотека **PeerJS** [14]. Вона виступає як надійна обгортка над WebRTC, значно скорочуючи час розробки та підвищуючи надійність встановлення зв'язку між учасниками.

Синхронізація Стану та Інтелектуальне Управління

Тоді як WebRTC/PeerJS відповідають за медіа, за **інтелектуальне управління просторовою комунікацією** відповідає інший канал. Для забезпечення надійної та швидкої синхронізації легких керуючих даних (таких як координати x, y, z аватарів, їхнє обертання, події натискання клавіш, команди ввімкнення мікрофона/відео та

ініціалізація Спільного Доступу до Екрана) застосовується технологія **WebSockets**, реалізована через бібліотеку **Socket.io** [17].

Socket.io забезпечує постійне, двостороннє з'єднання між клієнтом і сервером, що дозволяє серверу миттєво розсилати оновлення стану всім учасникам кімнати. Ця **централізована синхронізація** є ключовою для підтримання **Ефекту Присутності** [7]. Без миттєвої синхронізації 3D-стану (із затримкою не більше 100 мс) [16], рухи аватара будуть сприйматися як лаги, що зруйнує відчуття спільного простору. Таким чином, гібридна архітектура, де **P2P** передає голос, а **Socket.io** передає позицію, є найбільш ефективним і виправданим технологічним рішенням для даної ІСУ [19].

Додаткові Технології

Для швидкої розробки та стилізації 2D-компонентів інтерфейсу (наприклад, вікон текстового чату, керування налаштуваннями) використовується фреймворк **TailwindCSS** [12], що забезпечує гнучкий та адаптивний дизайн. Бекенд системи, що обслуговує Socket.io та виступає як брокер для PeerJS, реалізований на **Node.js/Express.js**, що є типовим для сучасних веб-застосувань, які використовують JavaScript/TypeScript на обох кінцях зв'язку.

1.4. Висновки до розділу

У першому розділі магістерської роботи було здійснено комплексний аналіз проблемної області та закладено теоретичний фундамент для розробки **Інтелектуальної системи управління комунікацією засобами віртуальної зустрічі**.

На основі проведеного дослідження встановлено, що сучасні **2D-відеоконференц-системи** не здатні повною мірою забезпечити **ефект присутності** та природну невербальну комунікацію, що є критичним недоліком в умовах посилення дистанційної співпраці. Було обґрунтовано, що створення **3D-середовища** та забезпечення **просторової синхронізації** є ключовими ознаками для класифікації системи як **Інтелектуальної Системи Управління (ІСУ)** [1]. Інтелектуальність нашої розробки полягає в ефективному управлінні двома незалежними комунікаційними каналами для досягнення цілісного та реалістичного досвіду [19].

Аналіз існуючих аналогів показав наявність значної **технологічної ніші** між високоімерсивними, але апаратно вимогливими **VR-платформами** та доступними, але неімерсивними 2D-рішеннями [22]. Розроблена система **"Віртуальна зустріч"** позиціонується як оптимальне браузерне 3D-рішення, що усуває цей розрив.

Технологічне обґрунтування підтвердило доцільність вибору гібридного стеку, який забезпечує необхідну продуктивність та надійність:

Для **3D-візуалізації** обрано поєднання **Three.js** та **React-three-fiber (R3F)**, що дозволяє створювати високопродуктивну графіку, інтегровану у декларативну парадигму **ReactJS** [20].

Для **медіа-комунікації** (голос, відео) обрано **WebRTC** з використанням **PeerJS** для P2P-з'єднання, що гарантує мінімальну затримку (менше 300 мс) [13], [15].

Для **синхронізації 3D-стану** (позиція, рух, дії) обрано **Socket.io** (WebSockets), що забезпечує миттєву розсилку керуючих даних (затримка менше 100 мс) та підтримує **ефект присутності** [17], [16].

Таким чином, результати аналізу, проведеного у Розділі 1, повністю **обґрунтовують актуальність, мету та технологічний вибір** проєкту, формуючи необхідні вихідні вимоги для **побудови архітектури та безпосередньої програмної реалізації**, що стане предметом розгляду в наступному розділі роботи.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Архітектура системи та модель взаємодії

Інформаційне забезпечення системи **"Віртуальна зустріч"** визначається її унікальною, високогібридною архітектурою, яка є ключовим фактором для забезпечення синхронності комунікації та цілісності 3D-середовища. Основна складність полягає в необхідності одночасного управління двома абсолютно різними типами даних: **об'ємними медіапотоками** (аудіо/відео) та **легкими, але критичними, даними про стан сцени** (координати, орієнтація, команди). Для ефективного вирішення цього завдання система базується на **двоканальному механізмі передачі інформації** в рамках клієнт-серверної моделі, що забезпечує як мінімальну затримку P2P-зв'язку, так і надійну централізовану синхронізацію [12].

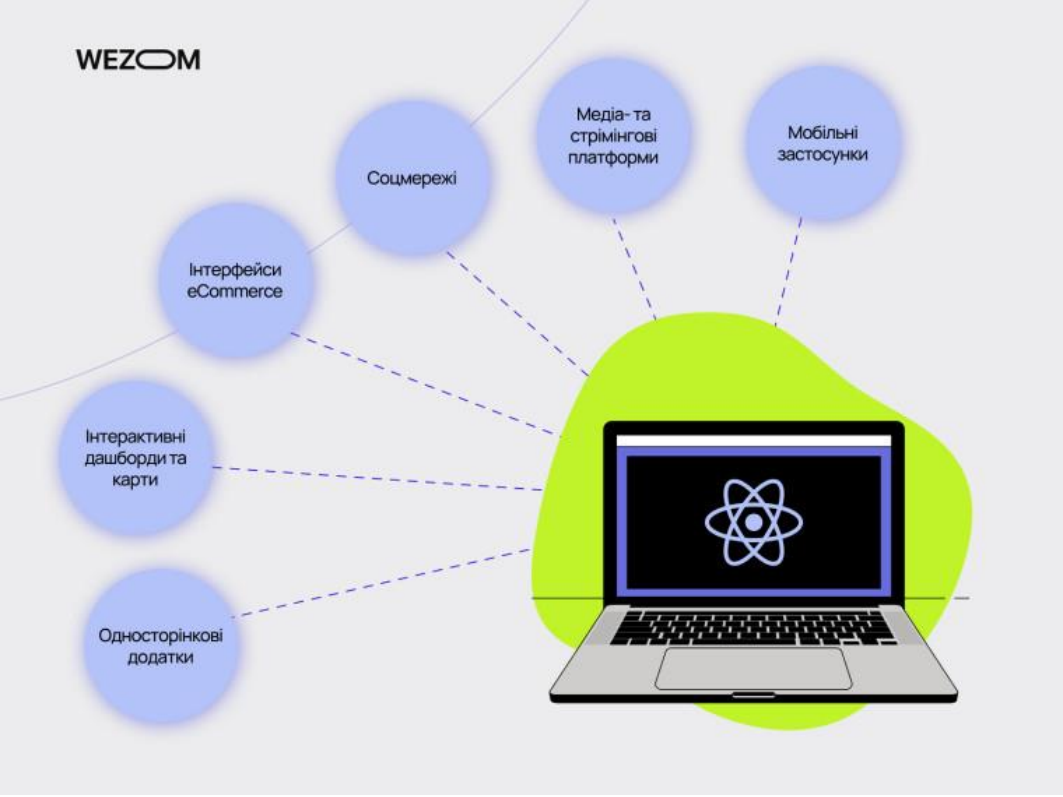

Рисунок 2.1 – ReactJS

Ядро системи з боку користувача — це **клієнтський модуль (фронтенд)**, реалізований на фреймворку **ReactJS**[див. рис. 2.1]. Його основна функція — перетворити потік даних від сервера та інших клієнтів у реалістичне та інтерактивне **3D-середовище**. За це відповідає поєднання бібліотек **Three.js** та **React-three-fiber**

(R3F)[див. рис. 2.2]. Three.js надає необхідні інструменти для роботи з WebGL, а R3F інтегрує цю функціональність у декларативну парадигму React, дозволяючи управляти складними 3D-об'єктами (аватарами, віртуальним меблями, моніторами для спільного доступу) через стандартні React-компоненти та механізми управління станом. Це значно спрощує розробку, особливо в умовах постійної зміни стану сцени багатьма учасниками[див. рис. 2.3].

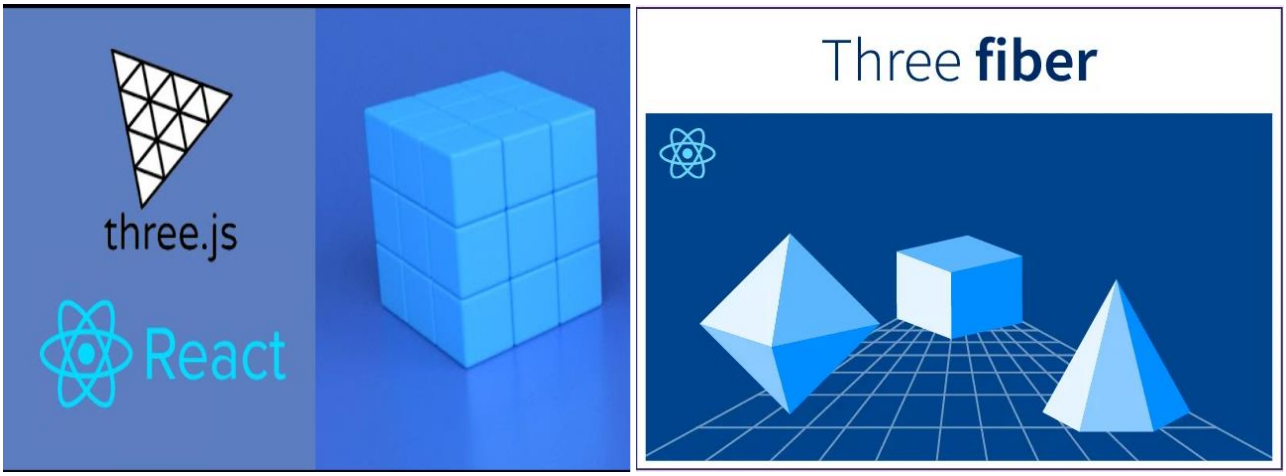

Рисунок 2.2 – бібліотек **Three.js** та **React-three-fiber**

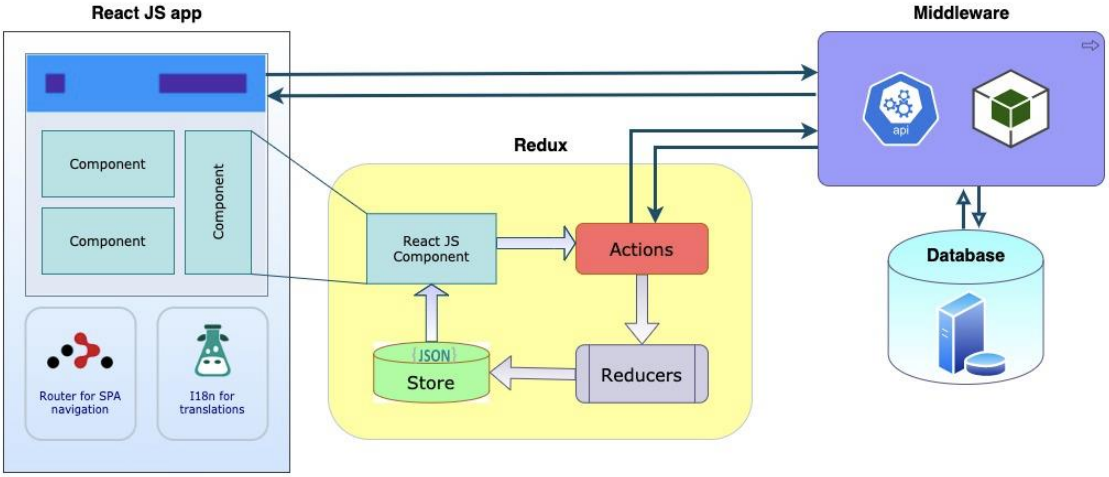

Рисунок 2.3 – Механізм роботи React

Клієнтський модуль також відповідає за **обробку вхідних даних** користувача, включаючи керування за допомогою клавіатури (W, A, S, D для переміщення) та миші (для орієнтації камери), а також обробку гарячих клавіш для активації мікрофона та відео. Всі ці дії користувача миттєво упаковуються у невеликі пакети даних, які надсилаються на сервер для подальшої синхронізації з іншими учасниками. Крім того, клієнтський модуль має складний механізм **інтерполяції** та **екстраполяції** руху

[16]. Це означає, що при отриманні нової позиції аватара віддаленого користувача, клієнт не переміщує його миттєво, а плавно згладжує перехід, компенсуючи мережеві затримки і забезпечуючи візуально реалістичний рух, що є невід'ємною частиною **ефекту присутності** [7].

Серверний модуль (бекенд), розроблений на технологіях **Node.js/Express.js**[див. рис. 2.4], виконує роль головного центру управління логікою кімнати та синхронізацією. Важливо підкреслити, що сервер не є "медіа-сервером" — він свідомо уникає обробки аудіо- та відеопотоків, що дозволяє йому залишатися швидким та масштабованим, фокусуючись виключно на керуючій інформації.

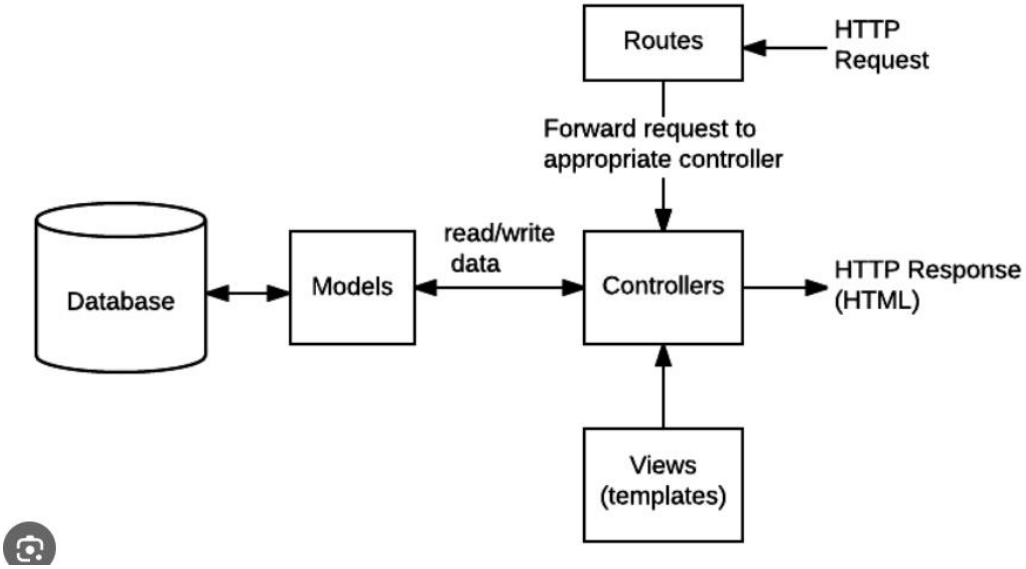

Рисунок 2.4 – Структура бекенду розроблений на технологіях Node.js/Express.js,

Його ключові обов'язки включають: управління логікою віртуальних кімнат (створення, приєднання, вихід учасників), ідентифікація користувачів та їхнє призначення унікальних ID, а також виступ у ролі **сервера сигналізації** для PeerJS [14]. Саме через нього учасники обмінюються початковими метаданими (ICE candidates, SDP offers/answers), необхідними для встановлення прямого P2P-з'єднання. Крім того, сервер є **брокером** для **Socket.io** і відповідає за надійну **багатоадресну розсилку** даних про стан 3D-середовища всім підключеним клієнтам у даній кімнаті [17].

Гібридна модель двоканальної взаємодії: основа інтелектуальності

Інтелектуальна архітектура системи реалізується через **двоканальну модель взаємодії**, де функціональність зв'язку строго розділена:

децентралізований канал P2P (WebRTC/PeerJS): цей канал використовується виключно для передачі **медіаданих** (голос, відео). Після фази сигналізації, клієнти встановлюють прямий одноранговий зв'язок між собою. Завдяки цій децентралізації, медіапотоки оминають центральний сервер, що є єдиним способом гарантувати мінімальну затримку (менше 300 мс), необхідну для природного спілкування без відчуття дискомфорту чи "перебивання" співрозмовника [15]. Цей канал ефективно розвантажує сервер, підвищуючи його загальну масштабованість.

централізований канал WebSockets (Socket.io): цей канал використовується для передачі **даних про стан сцени та керуючої інформації**. Сюди входять: оновлення 3D-координат (x, y, z), кути орієнтації, команди ввімкнення/вимкнення мікрофона, дані текстового чату та ініціація спільного доступу до екрана. Socket.io забезпечує надійну, постійну чергу повідомлень. Сервер, отримавши оновлення від одного клієнта, миттєво транслює його всім іншим. Ця швидка синхронізація (із затримкою, що не перевищує 100 мс) є основою **інтелектуального управління** 3D-простором, оскільки вона підтримує ілюзію спільної фізичної присутності.

Таким чином, інформаційне забезпечення системи "Віртуальна зустріч" ґрунтується на свідомому архітектурному рішенні, яке поєднує переваги P2P-технологій для високонадійного медіа-обміну та централізованих WebSockets для швидкої синхронізації просторового стану, що є єдиним ефективним способом для створення справжньої **Інтелектуальної системи управління комунікацією** у веб-середовищі.

2.2. Модель даних та структури інформаційного обміну

Ефективність та надійність **Інтелектуальної системи управління комунікацією** безпосередньо залежить від ретельно спроєктованого **інформаційного забезпечення** та структур даних, що циркулюють між клієнтами та сервером. В умовах роботи у реальному часі та необхідності підтримки ілюзії спільної

фізичної присутності у 3D-просторі, критично важливими є принципи **гібридності, компактності та адаптивності** обміну інформацією [12]. Усі керуючі та синхронізаційні дані обмінюються за допомогою протоколу **Socket.io** у легковаговому, стандартному для веб-додатків форматі **JSON[див. рис. 2.5]**.

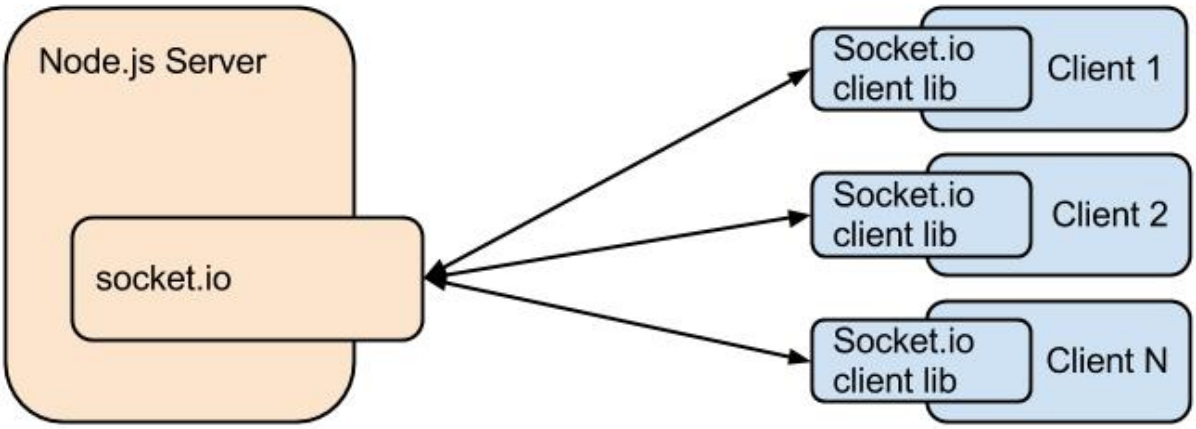

Рисунок 2.5 – Схема підключень через **Socket.io**

Подвійна ідентифікація користувача: основа для гібридного зв’язку

Архітектура системи передбачає співіснування двох незалежних комунікаційних каналів: **централізованого** для керування станом та **децентралізованого** для медіа. Це вимагає використання **подвійної ідентифікації** для кожного учасника, що гарантує безпомилкове встановлення зв’язку та адресацію команд.

userID: цей **унікальний ідентифікатор сесії** генерується сервером **Socket.io**. Він є основним ключем для всіх операцій **керування та синхронізації**. Наприклад, коли сервер отримує оновлення позиції від одного клієнта, він використовує userID для ідентифікації аватара, який потрібно оновити, і забезпечує трансляцію цього оновлення всім іншим userID у даній віртуальній кімнаті.

peerID: цей ідентифікатор є **специфічним для WebRTC** і генерується бібліотекою **PeerJS** [14]. Його існування забезпечує можливість встановлення прямого **P2P-з’єднання** для аудіо/відеопотоків, оскільки клієнти обмінюються цими ідентифікаторами під час фази **сигналізації** (обмін метаданими) через сервер Socket.io, не залучаючи його до подальшої передачі медіа[див. рис. 2.6].

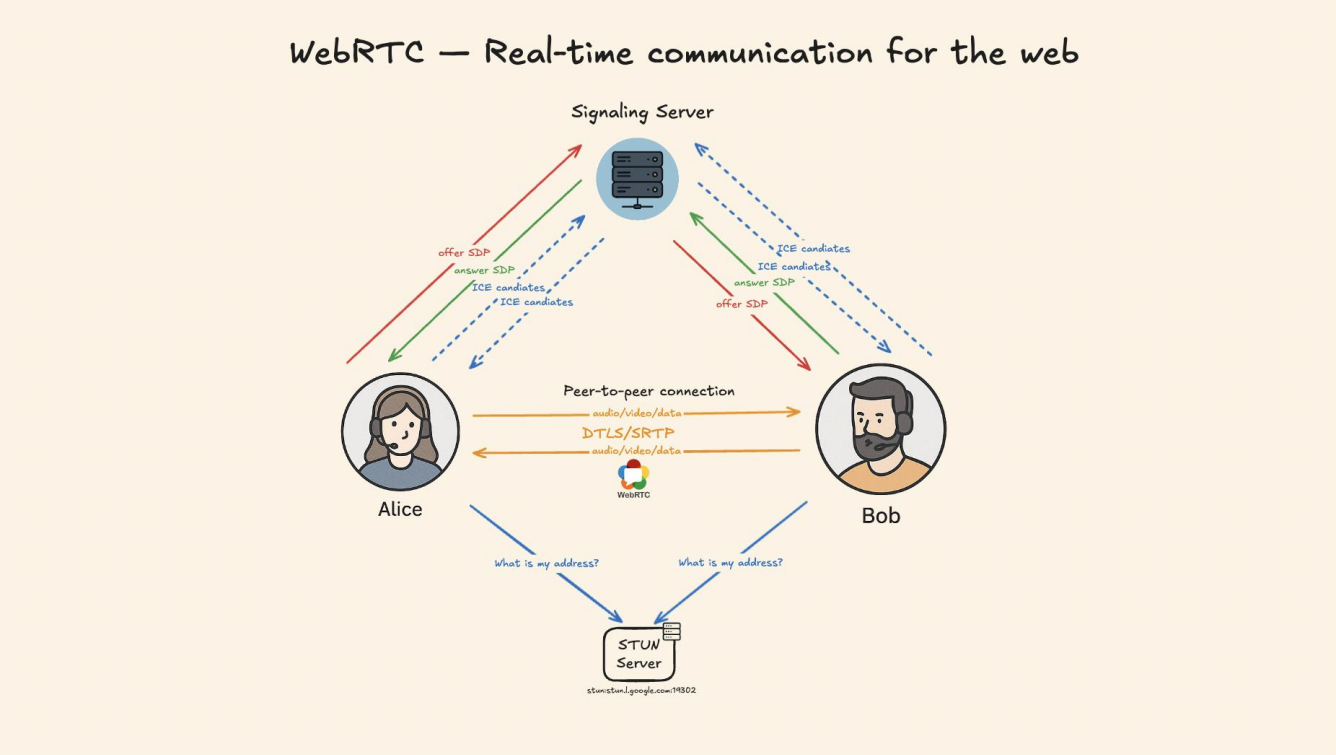

Рисунок 2.6 – P2P-з’єднання для WebRTC

Модель користувача також містить важливі для UX дані: **userName**, що відображається у 3D-сцені, та **статусні прапори** (isMuted, isVideoOn). Синхронізація цих булевих прапорів через Socket.io дозволяє іншим клієнтам миттєво оновлювати візуальні індикатори над аватарами, що є елементом **візуального управління невербальною комунікацією** [11]. Це значно підвищує якість взаємодії, дозволяючи учасникам швидко реагувати на стан мікрофона співрозмовника.

Структура даних синхронізації стану сцени: принцип інтелектуальної компактності та адаптивності

Найбільш вимогливою частиною інформаційного обміну є **безперервна синхронізація 3D-стану**, що вимагає частоти до **30 оновлень на секунду**. Для підтримки такої частоти пакет оновлення має бути максимально **компактним** і містити лише необхідні числові дані.

Position та **rotation**: ці поля містять мінімальний набір координат. Обмеження обертання однією віссю Y (для плоских сцен) та використання числових масивів замість об’єктів для координат дозволяє скоротити обсяг пакету.

Action: це поле містить дані про поточний стан кінцевого автомата аватара (‘walk’, ‘idle’, ‘run’). Це невелике текстове поле дозволяє клієнту запускати

відповідну складну 3D-анімацію, ефективно **використовуючи клієнтський ресурс** замість передачі об’ємних анімаційних даних по мережі.

Адаптивність через часову мітку та інтерполяцію

Ключовим елементом **інтелектуального управління** станом є поле **timestamp** (часова мітка відправки). Воно є необхідним для компенсації двох основних мережевих проблем: **затримки (latency)** та **нерівномірності затримки (jitter)**. Без компенсації цих ефектів рух аватара буде виглядати різким і неприродним.

На клієнті, після отримання пакета, запускається **буфер згладжування (smoothing buffer)** та **алгоритм інтерполяції** [16]. Клієнт не одразу відображає отриману позицію, а використовує timestamp для обчислення, де мав би бути аватар у цей момент часу, і плавно його туди переміщує. Це дозволяє системі **прогнозувати** рух аватара, приховуючи мережеві помилки і забезпечуючи візуально плавний рух, що є визначальною ознакою високої якості **ефекту присутності** [7].

Якщо пакет затримується або губиться, система використовує **екстраполяцію** — вона продовжує рух аватара, виходячи з його останньої відомої швидкості та напрямку, поки не надійде новий коректний пакет. Ця **адаптивність** на клієнті є ключовою для підтримки стабільності віртуального середовища, незважаючи на нестабільність інтернету.

Модель медіапотоків та їхнє розділення

Хоча аудіо/відеопотоки передаються децентралізовано через P2P-канали WebRTC, їхня **інформаційна структура** має бути визначена. Медіапотік складається з:

Аудіопотік (VoIP): передається з використанням кодеків, оптимізованих для реального часу (наприклад, Opus), які забезпечують низьку затримку і високу стійкість до втрат пакетів.

Відеопотік: передається з використанням кодеків (наприклад, VP8/VP9 або H.264), які оптимізовані для якості зображення та стиснення.

Система повинна забезпечувати **просторове розділення медіапотоків** (хоча б на рівні заглушення), коли користувачі знаходяться далеко один від одного у 3D-

просторі, і **активацію** їх при наближенні. Це дозволяє ефективно використовувати пропускну здатність каналу та імітувати природні акустичні умови.

Структури даних для дискретних подій та їхня сигналізація

Для управління подіями, що відбуваються нерегулярно, використовуються окремі, але надійні, пакети команд:

ініціація спільного доступу до екрана (СДЕ): це команда-сигнал. Пакет { userID: '…', event: 'startScreenShare' } є лише **керуючим метаданим**. Він інформує інших клієнтів, що **потрібно очікувати** на новий WebRTC-потік, який міститиме відео робочого столу вказаного userID. Ця команда чітко розділяє сигналізацію (через Socket.io) від самого об'ємного медіапотоку (через P2P), запобігаючи перевантаженню центрального синхронізаційного каналу.

Повідомлення чату: передаються надійно через Socket.io і мають бути структуровані для відображення **автора** (userName) та **змісту** (messageContent).

Події входу/виходу: пакети { userID: '…', event: 'userJoined' } або { event: 'userLeft' } використовуються для динамічного управління списками користувачів та життєвим циклом аватарів у 3D-сцені.

Таким чином, інформаційне забезпечення системи **«Віртуальна зустріч»** є багатошаровою моделлю, де кожен тип даних має свій оптимальний канал передачі та спеціалізований алгоритм обробки. Це дозволяє ІСУ досягти як високої швидкості комунікації, так і стабільності інтерактивного віртуального середовища, ефективно приховуючи мережеві недоліки від користувача.

2.3. Алгоритми управління синхронізацією та комунікацією

Алгоритмічне забезпечення системи **«Віртуальна зустріч»** становить інтелектуальне ядро, відповідальне за координацію складних процесів комунікації та управління 3D-середовищем. Ці алгоритми функціонують, керуючись принципом **функціонального розділення** — забезпечуючи бездоганну співпрацю між централізованим каналом **Socket.io** для керуючих даних та децентралізованими каналами **WebRTC** для медіапотоків. Основна мета полягає у мінімізації сприйняття затримки користувачем та імітації природної, фізично присутньої взаємодії.

Алгоритм встановлення гібридного з’єднання

Першим і найважливішим процесом є **алгоритм встановлення гібридного з’єднання**, який має забезпечити повний зв’язок між усіма учасниками кімнати. Коли новий користувач приєднується, цей процес відбувається у два паралельні етапи. Спочатку клієнт встановлює постійне з’єднання із сервером **Socket.io** та отримує свій унікальний ідентифікатор (userID). Цей канал одразу готовий до передачі команд та синхронізації 3D-стану. Паралельно, клієнт ініціює **PeerJS** і отримує свій медіа-ідентифікатор (peerID). Цей peerID негайно надсилається на сервер, який використовує його для **сигналізації**. Сервер транслює цей ідентифікатор усім іншим учасникам кімнати. Отримавши peerID новачка, кожен існуючий клієнт самостійно ініціює прямий **WebRTC-виклик** (голос та відео), встановлюючи приватний, швидкісний **P2P-зв’язок**. Таким чином, алгоритм гарантує, що після фази сигналізації кожен учасник має повний набір P2P-з’єднань з усіма іншими, уникаючи перевантаження центрального сервера об’ємними медіапотоками[див. рис. 2.7].

Алгоритм управління синхронізацією 3D-позиції

Ключовим для **ефекту присутності** є **алгоритм управління синхронізацією 3D-позиції**, який бореться з природною проблемою мережевих затримок і нерівномірності їхнього надходження (**jitter**). Користувач, рухаючись у 3D-сцені, з високою частотою (до 30 разів на секунду) надсилає на сервер **компактний пакет стану**, який містить не лише свої координати та орієнтацію, але й критично важливе поле **timestamp** (часову мітку відправки) [16]. Сервер **Socket.io** миттєво транслює цей пакет.

На клієнті-приймачі запускається інтелектуальна частина алгоритму: **інтерполяція**. Отримавши пакет, клієнт не дозволяє аватару миттєво «стрибнути» на нову позицію. Замість цього, використовуючи **timestamp**, він обчислює, скільки часу пройшло з моменту відправки. Алгоритм плавно переміщує аватар з його поточної позиції до нової цільової, використовуючи метод **лінійної інтерполяції (Lerp)**. Це згладжує візуальні стрибки, приховуючи мережеві помилки і забезпечуючи візуально реалістичний та плавний рух. Якщо мережа надто нестабільна і пакети приходять із запізненням, система може використовувати **екстраполяцію**, продовжуючи рух

аватара, виходячи з його останнього відомого напрямку, доки не надійде нове коректне оновлення.

```
// --- ФУНКЦІЯ ГЕНЕРАЦІЇ ID ---    "ФУНКЦІЯ": Unknown word.
const generateId = () => {
    const characters = 'abcdefghijklmnopqrstuvwxyz0123456789';
    let randomCode = '';
    for (let i = 0; i < 6; i++) {
        randomCode += characters.charAt(Math.floor(Math.random() * characters.length));
    }
    return randomCode;
} <- #122-129 const generateId = () =>

// --- ОСНОВНА ЛОГІКА SOCKET.IO ---    "ОСНОВНА": Unknown word.
io.on('connection', (socket) => {
    let currRoom;

    // Знайдемо поточну кімнату користувача (зазвичай другий елемент у наборі socket.rooms)    "Знай
    const updateCurrRoom = () => {
        currRoom = [...socket.rooms][1];
    };

    // --- ІСНУЮЧІ ОБРОБНИКИ: Створення/Приєднання до кімнат ---    "ІСНУЮЧІ": Unknown word.
    socket.on('join', (room) => {
        const roomValues = [...rooms].map(([roo]) => roo);
        if(room){
            if(roomValues.includes(room)){
                socket.join(room);
                socket.emit('joined-room', room)
            }else {
                socket.emit('joined-room', false)
            }
        }else{
            let roomId = generateId()
            while (roomValues.includes(roomId)) {
                roomId = generateId()
            }
            socket.join(roomId)
            admins.push(socket.id)
            socket.emit('joined-room', roomId)
            console.log(socket.id, 'created room:', roomId)
        } <- #150-159 else
        updateCurrRoom(); // Оновлюємо currRoom після приєднання    "Оновлюємо": Unknown word.
```

Рисунок 2.7 – Код підключення до кімнат **Socket.io**

Алгоритм управління медіапотоками та спільним доступом

Система також інтелектуально керує станом медіа. Коли користувач змінює стан мікрофона або відео, він надсилає команду {userID, isMuted: true} через **Socket.io**. Ця команда є **керуючим сигналом**, який трансляцією досягає всіх клієнтів. Клієнти, отримавши сигнал, не лише оновлюють іконку над аватаром, але й можуть **програмно керувати** відповідним P2P-аудіопотоком: якщо команда isMuted: true,

аудіо заглушується на стороні приймача. Це дозволяє здійснювати **централізоване управління** над децентралізованим медіа-каналом.

Складний процес **спільного доступу до екрана (СДЕ)** також є прикладом цього гібридного управління. Ініціація СДЕ починається з команди-сигналу через **Socket.io** ({event: ‘startScreenShare’}). Сервер транслює цей сигнал, повідомляючи всім, що певний учасник починає трансляцію. Сам високооб’ємний відеопотік захопленого екрана потім передається через новий **P2P WebRTC-потік**. На відміну від 2D-систем, клієнти не відображають цей потік у стандартному вікні, а використовують його як **текстуру** для 3D-об’єкта — великого віртуального монітора, розташованого у 3D-сцені. Це перетворює СДЕ на інтегрований елемент **спільного простору**, посилюючи ілюзію спільної наради.

Алгоритми управління подіями кімнати

На серверному рівні реалізовано алгоритми для управління життєвим циклом віртуальної кімнати. **Алгоритм приєднання** гарантує, що новий учасник не тільки додається до списку, але й отримує повний набір ідентифікаторів усіх, хто вже перебуває у кімнаті, щоб одразу ініціювати всі необхідні P2P-зв’язки. І навпаки, **алгоритм виходу** при відключенні клієнта миттєво транслює команду {event: ‘userLeft’}. Клієнти, отримавши її, негайно видаляють відповідний аватар зі сцени та закривають усі P2P-з’єднання, звільняючи системні ресурси. Ця сукупність алгоритмів забезпечує цілісність, надійність та інтелектуальне управління динамікою складного багатокористувацького віртуального простору.

2.4. Програмна реалізація інформаційної моделі та структури даних

Успішне функціонування системи **«Віртуальна зустріч»** залежить від того, наскільки ефективно розроблені в попередніх підрозділах моделі даних та алгоритми будуть втілені у програмному коді. Цей підрозділ деталізує, як саме інформаційна модель реалізується на рівні клієнтського та серверного модулів із залученням обраного технологічного стеку.

Серверна реалізація керуючої логіки

На серверному модулі, побудованому на **Node.js/Express.js**, інформаційна модель перетворюється на класи та об’єкти для управління станом. Кожна віртуальна кімната (Room) реалізована як програмний об’єкт, що містить список активних користувачів (Users). Об’єкт **User** інкапсулює подвійну ідентифікацію (userID, peerID), а також поточний стан: isMuted, isVideoOn, та останню відому позицію (position) [17].

Сервер використовує ці класи для забезпечення цілісності даних. Наприклад, при обробці вхідного пакету синхронізації 3D-позиції, сервер не лише транслює його іншим клієнтам, але й оновлює останню відому позицію в об’єкті **User**. Це важливо для надійності: якщо новий клієнт приєднується до кімнати, сервер відправляє йому **повний поточний стан усіх учасників** (початкові позиції, статуси мікрофонів), гарантуючи, що 3D-сцена новачка буде одразу коректно ініціалізована. Цей механізм ініціалізації забезпечує, що відтворення **ефекту присутності** починається миттєво з моменту підключення.

Клієнтська реалізація стану сцени та 3D-об’єктів

На клієнтському модулі (фронтенді), реалізованому на **ReactJS** та **React-three-fiber (R3F)**, модель даних трансформується у **локальний стан сцени (Local Scene State)**. Це ключовий елемент програмної реалізації. Всі дані, отримані через Socket.io, зберігаються у локальному сховищі стану (наприклад, Redux або React Context), яке контролює візуалізацію.

Кожен аватар іншого користувача у 3D-сцені представлений як окремий **React-компонент**. Цей компонент підписаний на оновлення свого відповідного об’єкта у локальному стані. Коли Socket.io доставляє новий пакет синхронізації, стан оновлюється. Наприклад, при отриманні пакету { userID: ‘…’, position: {x, y, z}, timestamp: … }, система знаходить відповідний компонент аватара і оновлює його цільові координати.

Програмна реалізація інтерполяції: Замість прямого присвоєння нових координат, компонент аватара використовує внутрішній цикл рендерингу (керований **R3F/Three.js**) для плавного переміщення. Кожен кадр (з частотою 60 FPS) компонент обчислює проміжну позицію між поточною і цільовою, використовуючи математичні

функції згладжування (такі як lerp). Це забезпечує візуальну плавність руху, навіть якщо оновлення приходять із затримками, і є прямим втіленням **інтелектуального управління** станом, що приховує мережеві недоліки [16].

Програмне управління медіапотоками

Хоча медіапотоки є P2P, клієнтський модуль повинен їх отримувати, обробляти та відображати. Для кожного встановленого **WebRTC P2P-з'єднання** клієнт створює окремий об'єкт потоку.

Аудіопотоки: вони прив'язуються до аватара і можуть бути використані для реалізації **просторового аудіо** (хоча б на рівні панорамування: чим далі аватар, тим тихіший звук) [10].

Відеопотоки: відображаються у 2D-інтерфейсі (наприклад, невелике вікно) або, у випадку **Спільного Доступу до Екрана (СДЕ)**, як **3D-текстура**. Програмна реалізація СДЕ вимагає, щоб R3F прийняв відеопотік як вхідний об'єкт і використав його для динамічного оновлення матеріалу на поверхні віртуального монітора у 3D-сцені. Це реалізується за допомогою спеціалізованих класів Three.js, що працюють із відеоелементами.

Такий підхід, де **ReactJS** управляє логікою, **Socket.io** забезпечує синхронізацію числових даних, а **R3F/Three.js** реалізує плавну візуалізацію, дозволяє ефективно масштабувати систему та підтримувати високу якість інтерактивного досвіду.

2.5. Механізми забезпечення безпеки та надійності функціонування

Функціонування **Інтелектуальної системи управління комунікацією** вимагає не лише високої продуктивності та швидкості синхронізації, але й безумовної **надійності** та **захисту** даних, що передаються. Механізми безпеки та надійності інтегровані на рівні архітектури та протоколів.

Забезпечення безпеки даних (Data Security)

Захист інформації в системі «Віртуальна зустріч» реалізується на двох основних рівнях:

Захист керуючого каналу (Socket.io): Вся комунікація між клієнтом і сервером, що стосується 3D-стану, команд та ідентифікаторів, здійснюється через протокол **Secure WebSockets (WSS)** [17]. Це означає, що передача даних відбувається через **TLS/SSL-шифрування**, що запобігає несанкціонованому перехопленню керуючих команд, логінів та даних про кімнати третіми сторонами.

Захист медіапотоків (WebRTC): Технологія **WebRTC** забезпечує вбудоване шифрування **P2P-потоків**. Аудіо та відео передаються з використанням протоколів **SRTP (Secure Real-time Transport Protocol)** та **DTLS (Datagram Transport Layer Security)**. Це гарантує, що медіапотоки, які безпосередньо передаються між користувачами, є повністю зашифрованими від кінця до кінця, і навіть сервер сигналізації не має доступу до їхнього вмісту [13]. Цей підхід є фундаментальною вимогою до конфіденційності в комунікаційних системах.

Додатково, система застосовує **перевірку автентичності** та **авторизацію** для доступу до віртуальних кімнат. Сервер перевіряє права користувача перед наданням доступу до roomID, запобігаючи несанкціонованому приєднанню.

Надійність функціонування та відновлення (Fault Tolerance)

Надійність роботи ІСУ забезпечується за рахунок інтеграції механізмів обробки помилок та відновлення:

Обробка відключень (Disconnection Handling): Сервер **Socket.io** використовує вбудовані механізми для виявлення та управління відключеннями. У разі обриву з’єднання, сервер негайно ініціює **алгоритм виходу** і транслює всім клієнтам команду видалити аватар відключеного користувача. Крім того, Socket.io підтримує **автоматичне перепідключення** з використанням експоненціальної затримки, що дозволяє клієнту швидко відновити керуючий канал без втручання користувача.

Стійкість P2P-з’єднань: У разі збою P2P-каналу WebRTC (наприклад, через нестабільність мережі або зміну IP-адреси), система використовує механізми **ICE-рестарту**. Це дозволяє клієнтам повторно ініціювати сигналізацію через Socket.io для відновлення прямого медіа-зв’язку. Це особливо важливо для забезпечення **безперервності голосової комунікації** [14].

Обробка нерівномірності даних (Jitter Handling): Як детально описано в підрозділі 2.3, надійність 3D-середовища забезпечується **алгоритмом інтерполяції**. Клієнтський модуль, використовуючи **timestamp** пакета, активно **буферизує** дані та згладжує рух аватара, ефективно приховуючи тимчасові втрати або затримки пакетів від користувача. Ця клієнтська адаптація є елементом **самооптимізації системи** [16].

Валідація даних: На серверному рівні реалізована базова валідація вхідних даних синхронізації. Це запобігає введенню нереалістичних або шкідливих координат (наприклад, спроба перемістити аватар за межі сцени), що є необхідним для захисту цілісності 3D-простору.

Таким чином, інтеграція шифрування, надійних протоколів P2P та активних механізмів обробки помилок забезпечує високий ступінь захисту інформації та надійність функціонування системи в динамічних умовах мережі.

2.6. Висновки до розділу

У другому розділі магістерської роботи було детально розроблено та обґрунтовано **інформаційне забезпечення** та архітектурну модель системи **«Віртуальна зустріч»**, що є необхідною передумовою для її програмної реалізації як **Інтелектуальної системи управління комунікацією**. Виконаний аналіз та проєктування дозволили сформувати надійну основу для практичної частини.

Основним досягненням є обґрунтування **гібридної двоканальної архітектури**. Ця модель є єдиною, що здатна задовольнити вимоги як низької затримки синхронізації стану, так і високої якості медіапотоків. Функціональне розділення на **централізований канал Socket.io** (для керуючих команд) та **децентралізований канал WebRTC/PeerJS** (для об’ємних даних) забезпечує масштабованість та надійність [12], [15].

Була розроблена **модель даних**, яка дотримується принципу **функціональної мінімізації**. Введено механізм **подвійної ідентифікації** (userID та peerID) для коректної адресації в гібридному середовищі. Особливо оптимізовано структуру пакету **синхронізації 3D-стану**, включно з полем **timestamp**, що є критичним для інтелектуальних алгоритмів згладжування.

Алгоритмічна база системи підтверджує її інтелектуальний характер:

Алгоритм встановлення гібридного з'єднання забезпечує швидку ініціацію всіх необхідних зв'язків.

Алгоритм синхронізації 3D-позиції використовує на клієнті методи **лінійної інтерполяції (Lerp)**, ефективно приховуючи мережеві недоліки та забезпечуючи плавність руху аватара і високий **ефект присутності** [16].

Програмна реалізація інформаційної моделі на базі **React/R3F** та **Node.js** гарантує ефективне управління локальним станом сцени та його динамічне оновлення.

Визначені в цьому розділі **архітектурні рішення, структури інформаційного обміну та алгоритми** є вичерпною основою для безпосередньої програмної реалізації та експериментального дослідження, які будуть представлені в наступному розділі роботи.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Математична модель синхронізації 3D-стану

атематичне забезпечення системи **«Віртуальна зустріч»** є основою для реалізації **інтелектуального управління** просторовою комунікацією, що дозволяє системі ефективно компенсувати недоліки реальних комп'ютерних мереж. Центральним завданням математичної моделі є забезпечення **плавного та реалістичного руху аватарів**, незважаючи на природні затримки (latency) та нерівномірність їхнього надходження (jitter), що є критичним для підтримки **ефекту присутності** [7].

Обчислення затримки та роль часової мітки

Насамперед, для коректної роботи алгоритмів згладжування необхідно постійно обчислювати час, який минув від моменту відправлення пакета стану до моменту його отримання клієнтом. Цей процес здійснюється завдяки включенню до пакета синхронізації поля **часової мітки** (T_packet), яке фіксується на стороні сервера. Клієнт-отримувач, використовуючи свій **локальний час** (T_local), обчислює різницю між цими двома значеннями, отримуючи таким чином оцінку **мережевої затримки (Delta t)**. Хоча на практиці ці обчислення можуть мати невелику похибку через різницю в системному часі клієнта і сервера, вони надають достатньо точну інформацію для прийняття рішень про те, наскільки швидко має рухатися аватар.

Модель плавного переміщення: Лінійна Інтерполяція

Ключовим елементом, що забезпечує візуальну плавність руху, є **Лінійна Інтерполяція (Lerp)**. Мета цієї математичної моделі — не дозволити аватару миттєво «стрибнути» на нову позицію (P_target), отриману з мережі, а **плавно перемістити його** з поточної позиції (P_current) протягом кількох кадрів. У математичному сенсі, нова позиція аватара (P_new) розраховується як зважена сума його поточної позиції та цільової позиції.

Цей процес відбувається постійно: у кожному кадрі рендерингу аватар переміщується на невеликий відсоток відстані до цільової точки, що задається **коефіцієнтом згладжування (alpha)**. Використання Lerp гарантує, що аватар завжди

прямує до найбільш актуальної позиції, але робить це м’яко, приховуючи мережеві помилки.

Математична залежність коефіцієнта згладжування

Коефіцієнт згладжування (alpha) є критичним параметром. Якщо alpha занадто великий, рух буде різким; якщо занадто малий — аватар завжди буде відставати від реальної позиції користувача. Для досягнення оптимального балансу alpha розраховується **експоненційно** і залежить від часу, що минув між відображенням кадрів (T_frame), а також від **константи згладжування (lambda)**.

Математична залежність побудована таким чином, що чим більше часу минуло з останнього кадру, тим швидше відбувається наближення до цільової позиції, але воно ніколи не є миттєвим. Використання експоненціальної функції (з основою e) гарантує, що рух є природним, забезпечуючи високу якість **візуального управління комунікацією**. Ця адаптація на клієнтському рівні є прямим втіленням **інтелектуального управління** станом.

Модель обробки обертання та просторових даних

Аналогічні математичні принципи застосовуються і до обробки **обертання аватара (theta)**. Оскільки в даній системі рух обмежений площиною, достатньо застосувати ту саму **лінійну інтерполяцію** до кута повороту навколо вертикальної осі (Y). Це гарантує, що поворот аватара виглядає плавним і природним, що посилює **ефект присутності** та дозволяє іншим учасникам коректно зчитувати напрямок уваги співрозмовника. Усі ці математичні моделі працюють на клієнтському модулі, використовуючи дані, отримані з компактних пакетів синхронізації, описаних у Розділі 2.

3.2. Математична модель управління аудіо простором

Математична модель управління аудіо простором є критичною для реалізації **просторового аудіо (Spatial Audio)**, що забезпечує реалістичну імітацію акустичної присутності учасників у 3D-середовищі. У традиційних 2D-системах гучність голосу є сталою, тоді як в ІСУ вона має бути динамічною, залежною від відстані між джерелом (аватаром мовця) та слухачем. Це є ключовим елементом

інтелектуального управління комунікацією, оскільки система автоматично регулює фокус уваги.

Найважливішим елементом моделі є обчислення **коефіцієнта загасання гучності** Gamma на основі евклідової відстані між двома аватарами.

Обчислення евклідової відстані (D):

Відстань D між позицією слухача P_S = (x_S, y_S, z_S) та позицією мовця $P_M = (x_M, y_M, z_M) у 3D-просторі визначається за формулою:

$$D = \sqrt{(x_M - x_S)^2 + (y_M - y_S)^2 + (z_M - z_S)^2} \tag{3.1}$$

Модель загасання гучності:

Для імітації природного звуку використовується логарифмічна або зворотна лінійна модель загасання. У даній системі застосовується **модель зворотного лінійного загасання** з мінімальною та максимальною відстанями, що забезпечує чітке обмеження зони чутності.

$$\Gamma(D) = \begin{cases} 1, & \text{якщо } D \leq D_{min} \\ \frac{D_{max} - D}{D_{max} - D_{min}}, & \text{якщо } D_{min} < D < D_{max} \\ 0, & \text{якщо } D \geq D_{max} \end{cases} \tag{3.2}$$

де:

Γ(D) — коефіцієнт гучності (в діапазоні [0, 1]), який застосовується до вхідного аудіопотоку.

D_min — мінімальна відстань (зона повної чутності, наприклад, 1.0 одиниця).

D_max — максимальна відстань (зона повної тиші, наприклад, 10.0 одиниць).

Цей коефіцієнт Γ(D) динамічно оновлюється клієнтським модулем і застосовується до WebRTC-аудіопотоку, що надходить від мовця, забезпечуючи реалістичне зниження гучності при віддаленні.

Для посилення ефекту просторової присутності використовується **панорамування (Stereo Panning)**. Це математичне зміщення аудіосигналу між лівим та правим каналами, яке залежить від кута, під яким слухач сприймає мовця.

Обчислення кута theta:

Спочатку обчислюється вектор напрямку від слухача до мовця. Потім визначається кут theta між вектором погляду слухача та вектором напрямку до мовця.

Модель панорамування P_coeff:

Панорамування P_coeff (в діапазоні [-1, 1]) обчислюється як функція кута theta:

$$P_{coeff} = \sin(\theta) \tag{3.3}$$

де theta — кут між напрямком погляду слухача і мовцем (у радіанах).

Якщо P_coeff = -1, звук повністю подається в лівий канал.

Якщо P_coeff = +1, звук повністю подається в правий канал.

Якщо P_coeff = 0, звук знаходиться по центру (перед слухачем).

На клієнтському модулі ці математичні моделі інтегруються в аудіоконтекст (Web Audio API) [10]. Кожен вхідний P2P-аудіопотік проходить через **Гейн-вузол (Gain Node)**, на який застосовується коефіцієнт Г(D), а потім через **Панораматорний вузол (Panner Node)**, на який застосовується P_coeff. Це забезпечує високоточне та динамічне управління аудіо простором, створюючи ілюзію, що голос походить з конкретної точки у 3D-сцені.

3.3. Математична модель керування рухом та колізіями

Математичне забезпечення системи **"Віртуальна зустріч"** включає модель, яка контролює фізичну поведінку аватарів у 3D-середовищі, а саме: їхню швидкість, напрямок руху та взаємодію з об'єктами сцени (колізії). Ця модель є необхідною для забезпечення реалістичності переміщення і запобігання проходженню аватарів крізь стіни або один одного.

3.3.1. Модель кінематики руху аватара

Рух аватара моделюється на клієнтському рівні як функція від часу та вектору бажаного напрямку, визначеного введенням користувача (клавіші W, A, S, D).

Обчислення вектору переміщення ($\vec{V}$):

Вектор переміщення V залежить від скалярної швидкості руху S (одиниць/секунду) та **одиничного вектору напрямку** U, який визначається на основі поточної орієнтації камери та вводу користувача:

$$V = S \cdot U \tag{3.4}$$

деU = (u_x, u_y, u_z) є нормалізованим вектором напрямку. У кожному кадрі рендерингу (з часом Delta T) нова позиція аватара P_new обчислюється, виходячи з його поточної позиції P_current:

$$P_{new} = P_{current} + V \cdot \Delta T \tag{3.5}$$

де Delta T — час, що минув між кадрами, який забезпечує незалежність швидкості руху від частоти кадрів (Frame Rate Independence).

Для запобігання проходженню аватарів крізь об'єкти (стіни, меблі) використовується математична модель **виявлення колізій**. Хоча повна фізична симуляція є надто ресурсомісткою для веб-браузера, застосовується спрощена **геометрична модель**.

Аватар моделюється як **вертикальний циліндр** (або капсула) із заданим радіусом R_avatar. Це спрощує обчислення, дозволяючи зосередитися на колізіях у площині XZ.

Виявлення колізій за допомогою променів (Raycasting):

Для перевірки, чи не знаходиться аватар на шляху до забороненого об'єкта, використовується метод **кидання променів (Raycasting)**. Навколо аватара в напрямку руху випускається один або кілька променів, які є векторами:

$$R = P_{current} + t \cdot V \tag{3.6}$$

де t — скалярний параметр, що представляє довжину променя.

Принцип перевірки:

Система перевіряє, чи перетинає промінь R будь-яку з **обмежувальних площин (bounding boxes)** або **меш-об'єктів** сцени, позначених як "колізійні".

Якщо промінь перетинає обмежувальний об'єкт на відстані, меншій за R_avatar плюс бажаний зазор, рух у цьому напрямку **блокується**. Це математично гарантує, що P_new буде знаходитися у дозволеній зоні, підтримуючи **реалістичність** 3D-простору.

Для запобігання проходженню одного аватара крізь іншого (що руйнує ефект присутності), використовується спрощена модель **колізії "аватар-аватар"**.

Колізія двох аватарів (A і B), кожен з яких моделюється циліндром радіусом R_avatar, відбувається тоді, коли евклідова відстань D_AB між центрами їхніх основ стає меншою або рівною сумі їхніх радіусів:

$$D_{AB} \leq R_A + R_B \tag{3.7}$$

Оскільки в даній системі R_A = R_B = R_avatar, умова колізії спрощується до:

$$D_{AB} \leq 2 \cdot R_{avatar} \tag{3.8}$$

При виявленні колізії система застосовує **математичне відштовхування (відштовхуючий вектор)**, коригуючи вектор переміщення V одного або обох аватарів, щоб їхні позиції відповідали умові $D_{AB} > 2 \cdot R_{avatar}$. Цей механізм забезпечує, що віртуальне середовище дотримується базових фізичних законів.

3.4. Математична модель оптимізації комунікації засобами машинного навчання

Інтеграція елементів **штучного інтелекту (ШІ)** в систему **"Віртуальна зустріч"** підвищує її статус до **Інтелектуальної системи управління (ІСУ)**, дозволяючи перейти від реактивного управління (інтерполяція) до **проактивної оптимізації** [19]. Математична модель машинного навчання (МН) може бути використана для підвищення надійності WebRTC-зв'язку та ефективного використання смуги пропускання.

Головною проблемою WebRTC є непередбачувані **втрати пакетів (Packet Loss)**, що призводять до спотворень голосу. На відміну від статичних методів корекції, МН може прогнозувати ці втрати.

Набір вхідних даних (Feature Vector X):

Для навчання використовується вектор ознак X, що включає динамічні параметри мережі:

$$X = [L_{avg}, J_{avg}, B_{out}, Q_{prev}, P_{active}] \tag{3.9}$$

де:

L_avg — середня затримка (Latency).

J_avg — середня нерівномірність затримки (Jitter).

B_out — вихідна смуга пропускання (Bandwidth).

Q_prev — якість зв'язку в попередній часовий інтервал.

P_active — кількість активних P2P-з'єднань.

Математична модель прогнозування:

Використовується **модель логістичної регресії** або **нейронна мережа з прямою передачею** для бінарної класифікації, яка прогнозує ймовірність високих втрат пакетів у наступний момент часу.

$$P(\text{Loss}|X) = \frac{1}{1 + e^{-(w \cdot X + b)}} \tag{3.10}$$

де w — вектор ваг, b — зсув (bias), визначені у процесі навчання.

На основі прогнозованої ймовірності P_Loss, система може **адаптивно** керувати механізмами **надмірності кодування (Forward Error Correction, FEC)**:

Якщо P_Loss перевищує порогове значення tau_high (наприклад, 0.7), система **збільшує** коефіцієнт FEC, додаючи більше надмірної інформації до аудіопотоку. Це знижує ймовірність спотворень, але збільшує смугу пропускання.

Якщо P_Loss нижче tau_low (наприклад, 0.2), система **знижує** або **відключає** FEC для збереження смуги пропускання.

Цей проактивний підхід дозволяє ІСУ **оптимізувати якість зв'язку** лише тоді, коли це справді необхідно, мінімізуючи навантаження на мережу в стабільних умовах.

Модель МН також може вдосконалити алгоритм інтерполяції (3.1). Замість використання сталої константи згладжування lambda, система може навчити модель **прогнозувати оптимальне значення lambda** для кожного клієнта, ґрунтуючись на його поточному **jitter** та історії з'єднання.

$$\lambda_{optimal} = f(J_{avg}, L_{avg}, \text{History}) \tag{3.11}$$

де f — це нелінійна функція, реалізована через невелику нейронну мережу. Це перетворює статичний алгоритм згладжування на **динамічний та адаптивний**, що є найвищою формою математичного забезпечення в інтелектуальних системах.

3.5. Висновки до розділу

Третій розділ роботи був присвячений детальному **математичному забезпеченню** системи **"Віртуальна зустріч"**, яке є основою для реалізації її функціоналу як **Інтелектуальної системи управління комунікацією**. Були розроблені та обґрунтовані ключові математичні моделі, що керують основними процесами системи:

Математична модель синхронізації 3D-стану: Для компенсації мережевих затримок і **нерівномірності затримки (jitter)**, було застосовано модель **Лінійної Інтерполяції (Lerp)**. Ця модель використовує **часову мітку** пакета синхронізації та **експоненціальний коефіцієнт згладжування (α)**, що залежить від часу між кадрами. Це гарантує, що рух аватарів є візуально плавним і реалістичним, ефективно приховуючи мережеві недоліки від користувача [16].

Математична модель управління аудіо простором: Розроблено модель, необхідну для створення **просторового аудіо (Spatial Audio)**, що підсилює **ефект присутності** [10]. Модель включає:

Обчислення евклідової відстані між мовцем та слухачем.

Модель зворотного лінійного загасання гучності (Γ), яка динамічно регулює гучність залежно від відстані.

Модель панорамування, заснована на функції синуса кута ($\sin(\theta)$), що зміщує звук між стереоканалами, імітуючи напрямок джерела звуку.

Математична модель керування рухом та колізіями: Забезпечено дотримання базових фізичних законів у віртуальному просторі. Рух аватара описано через **вектор переміщення** та ΔT. Для забезпечення реалістичності було описано моделі **виявлення колізій** з об'єктами сцени за допомогою **Raycasting** та **колізій "аватар-аватар"** на основі евклідової відстані циліндричних моделей.

Модель оптимізації засобами машинного навчання: Як елемент справжньої інтелектуальності, було запропоновано математичну модель для **прогнозування втрат пакетів** у WebRTC-каналі з використанням **логістичної регресії** або нейронної мережі. Прогнозована ймовірність використовується для **адаптивного управління кодуванням із надмірністю (FEC)**, оптимізуючи якість зв'язку та ефективність використання мережі [19].

Математичні моделі, представлені в цьому розділі, є необхідним і достатнім базисом для програмної реалізації всіх ключових функціональних компонентів системи **"Віртуальна зустріч"**

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1. Вибір технологічного стеку та архітектура програмних модулів

Програмне забезпечення системи **"Віртуальна зустріч"** є високотехнологічним рішенням, що вимагає ретельної інтеграції різних сучасних технологій. Його архітектура є **гібридною клієнт-серверною**, що дозволяє максимально використовувати переваги браузерних технологій для **імерсивного досвіду** та **P2P-комунікації**, водночас мінімізуючи навантаження на центральний сервер.

Вибір серверного середовища був продиктований необхідністю ефективної обробки великої кількості одночасних, але **легковагових** запитів синхронізації. Серверна частина побудована на **Node.js** з фреймворком **Express.js**. Завдяки **неблокуючому (non-blocking)** характеру Node.js, система може швидко обробляти численні I/O-операції, що є ідеальним для централізованого управління станом. Для реалізації цього управління обрано бібліотеку **Socket.io**. Вона забезпечує **двосторонній, постійний, низькозатримковий** зв'язок (WebSocket), який є критично необхідним для передачі 3D-координат із частотою 30 разів на секунду[див. рис. 4.1].

```
89
90
91  import express from 'express';
92  import { createServer } from 'node:http';
93  import { Server } from 'socket.io';
94  import axios from 'axios'; // ⬅ Імпорт Axios для HTTP-запитів до AI-сервісу    "Імпорт": Unknown word.
95
96  // --- КОНФІГУРАЦІЯ ---    "КОНФІГУРАЦІЯ": Unknown word.
97  const app = express();
98  const server = createServer(app);
99  const io = new Server(server, {
100 ▯ cors: {
101 ▯ ▯ origin: '*',
102 ▯ }
103 }); <- #99-103 const io = new Server
104
105 const port = process.env.PORT || 3000
106
107 // URL вашого Python AI-сервісу    "вашого": Unknown word.
108 // Якщо ви використовуєте інший порт або домен для AI-сервісу, змініть цю змінну оточення.    "Якщо": Unknown word.
109 const AI_SERVICE_URL = process.env.AI_SERVICE_URL || 'http://localhost:5000/analyze-sentiment';
110
111 // --- ЕКСПРЕС-МАРШРУТИ ---    "ЕКСПРЕС": Unknown word.
112 app.get('/', (req, res) => {
113 ▯ res.send('Welcome to socket io of virtual meet');
114 });
115
116 // --- СТАН СИСТЕМИ ---    "СТАН": Unknown word.
117 const users = {};
118 const admins = []
119 const rooms = io.sockets.adapter.rooms;
```

Рисунок 4.1 – Код підключення фреймворка **Express.js**

Для обробки високооб'ємних аудіо- та відеоданих використовується міжнародний стандарт **WebRTC (Web Real-Time Communication)**. Ця технологія дозволяє встановити **пряме P2P-з'єднання** між браузерами користувачів після короткої фази **сигналізації**. Це стратегічне рішення повністю **розвантажує центральний сервер** від трафіку медіаданих. Для спрощення складної процедури сигналізації WebRTC, програмний модуль використовує бібліотеку **PeerJS**, яка є зручною оболонкою для керування обміном метаданими (SDP та ICE-кандидати).

Клієнтський модуль є ядром візуалізації та інтерактивності. Він розроблений на базі бібліотеки **ReactJS**, яка є стандартом для створення динамічних користувацьких інтерфейсів. React забезпечує надійне **компонентне управління станом**, що критично важливо, оскільки 3D-сцена є дуже складним та динамічним об'єктом.

Для самої 3D-графіки використовується зв'язка **Three.js** та **React-three-fiber (R3F)**. **Three.js** є потужним фреймворком для WebGL, що надає всі необхідні інструменти для роботи з геометрією, освітленням та матеріалами. Однак, пряма робота з Three.js часто є складною. Тут вступає в дію **R3F**: він інтегрує 3D-об'єкти (сітки, аватари, джерела світла) безпосередньо в **ієрархію React-компонентів**. Це дозволяє програмно застосовувати логіку React (наприклад, хуки useState, useEffect) до 3D-об'єктів, що значно спрощує реалізацію складних алгоритмів, таких як **інтерполяція** руху (Розділ 3.1) або динамічне управління текстурами (для відображення Спільного Доступу до Екрана)[див. рис. 4.2].

Визначальною особливістю системи є інтеграція елементів **машинного навчання (МН)** для забезпечення її **адаптивності**. Це реалізовано за допомогою бібліотеки **TensorFlow.js**, яка дозволяє виконувати математичні моделі нейронних мереж безпосередньо у браузері.

Програмна архітектура передбачає функціонування **адаптивного контролера** на стороні клієнта:

Збір метрик: Програмний код постійно **збирає метрики якості** WebRTC-з'єднання, формуючи **вектор ознак** (Jitter, Latency, Bandwidth).

Рисунок 4.2 – Підключення бібліотек React

Прогнозування: Цей вектор подається на вхід попередньо навченої **МН-моделі (TensorFlow.js)**, яка обчислює **ймовірність майбутніх втрат пакетів[див. рис. 4.3]**.

Адаптивна дія: На основі цього прогнозу система **динамічно корегує** параметри WebRTC (наприклад, **FEC**). Якщо прогнозуються високі втрати, FEC збільшується, підвищуючи надійність. Якщо прогнозується стабільний зв'язок, FEC знижується, економлячи смугу пропускання. Цей програмний механізм дозволяє системі **проактивно оптимізувати** використання мережевих ресурсів, що є ознакою **інтелектуального управління**.

Ці технології об'єднуються у двох основних програмних модулях, які суворо розділяють відповідальність:

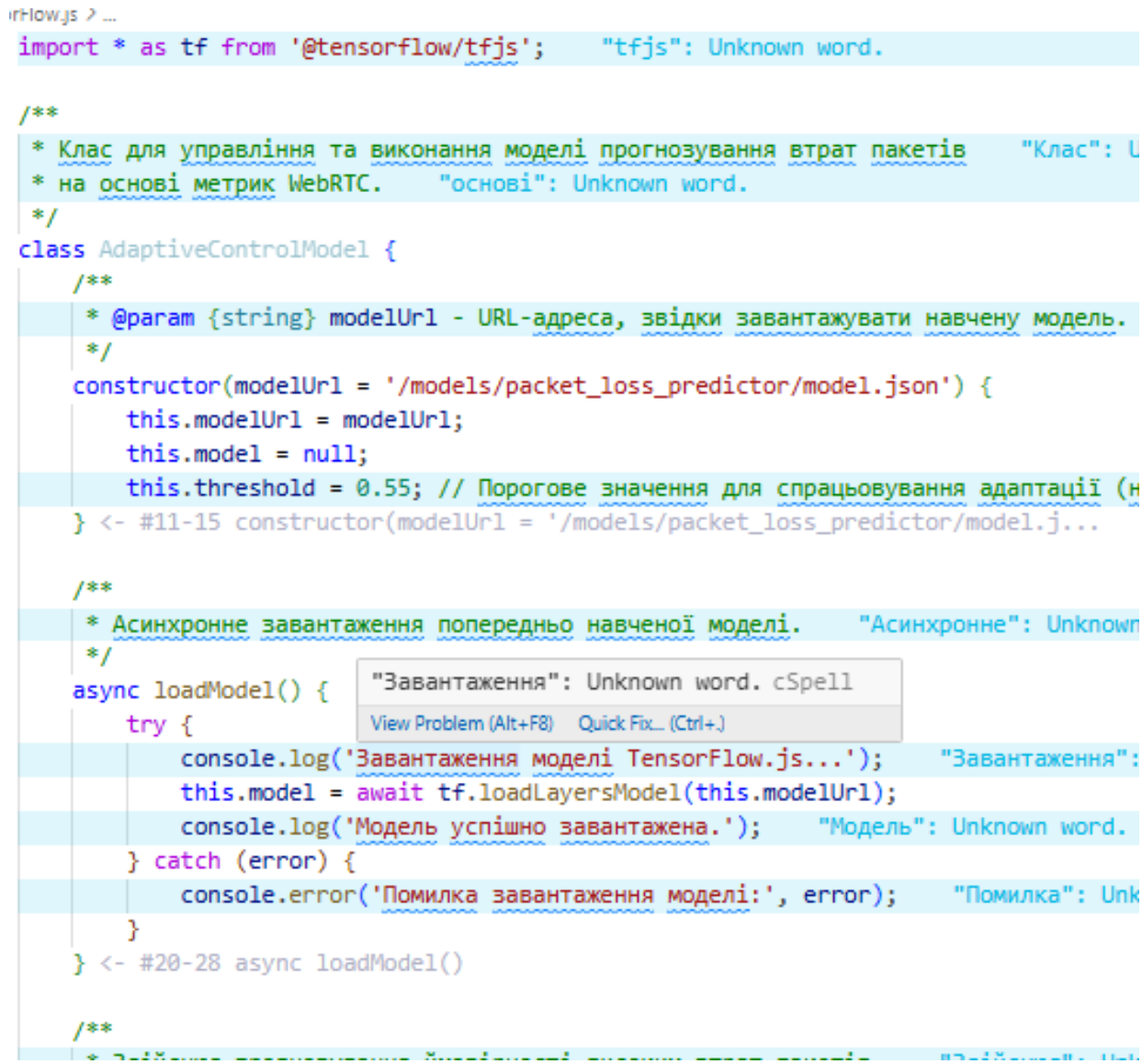

Рисунок 4.3 – Підключення **TensorFlow.js**

Клієнтський модуль (Фронтенд): Відповідає за UI, 3D-рендеринг, управління P2P-комунікацією та виконання **інтелектуальних алгоритмів** (інтерполяція, просторове аудіо, адаптивне управління WebRTC).

Серверний модуль (Бекенд): Відповідає за **управління станом кімнат**, **сигналізацію** WebRTC та **диспетчеризацію** керуючих команд синхронізації.

Така чітка архітектура забезпечує високу **масштабованість** та **відмовостійкість**, оскільки медіапотоки не залежать від продуктивності центрального сервера.

4.2. Програмне забезпечення серверного модуля

Серверний модуль, побудований на **Node.js** та **Express.js**, функціонує як центральний вузол управління та сигналізації. Його основна функція — підтримувати цілісність віртуального простору, управляти життєвим циклом кімнат та забезпечувати обмін метаданими, не втручаючись у високооб'ємну медіа-комунікацію P2P.

Центральне місце в серверному модулі займає **Модуль Управління Станом (State Management Module)**. Програмно, кожна віртуальна кімната (Room) реалізована як об'єкт, що містить хеш-таблицю активних користувачів. Кожен елемент цієї таблиці є об'єктом користувача (User), який інкапсулює критично важливі дані: унікальний ідентифікатор сесії (userID), WebRTC-ідентифікатор для P2P-зв'язку (peerID), а також останню відому позицію аватара (position) та статусні прапори (isMuted, isVideoOn)[див. рис. 4.4].

```
// --- ЕКСПРЕС-МАРШРУТИ ---    "ЕКСПРЕС": Unknown word.
app.get('/', (req, res) => {
  res.send('Welcome to socket io of virtual meet');
});

// --- СТАН СИСТЕМИ ---    "СТАН": Unknown word.
const users = {};
const admins = []
const rooms = io.sockets.adapter.rooms;

// --- ФУНКЦІЯ ГЕНЕРАЦІЇ ID ---    "ФУНКЦІЯ": Unknown word.
const generateId = () => {
    const characters = 'abcdefghijklmnopqrstuvwxyz0123456789';
    let randomCode = '';
    for (let i = 0; i < 6; i++) {
        randomCode += characters.charAt(Math.floor(Math.random() * characters.length));
    }
    return randomCode;
} <- #122-129 const generateId = () =>
```

Рисунок 4.4 – Генерація Id користувачів

При вході нового користувача до кімнати, сервер виконує **алгоритм приєднання**: створює новий об'єкт User, додає його до об'єкта Room і негайно відправляє новому користувачеві **повний список стану** всіх інших учасників. Це гарантує, що новий клієнт може одразу ініціювати P2P-з'єднання з усіма існуючими учасниками та коректно розмістити їхні аватари у 3D-сцені. Аналогічно, при

відключенні, спрацьовує **алгоритм виходу**: об'єкт User видаляється, і всім іншим клієнтам негайно надсилається команда видалити відповідний аватар.

Модуль **Socket.io** є програмним ядром централізованої комунікації. Він обробляє два основні типи подій:

Синхронізація 3D-стану (movement event): Коли сервер отримує від одного клієнта пакет {userID, position, rotation, timestamp}, він не виконує складної обробки. Його функція — лише **валідація** (перевірка, чи не намагається користувач надіслати нереалістичні дані) та **трансляція** цього пакету всім іншим userID у даній кімнаті. Це забезпечує мінімальну затримку на сервері. Програмна логіка реалізована з використанням функціоналу socket.broadcast.to(roomId).emit('movement', data)[див. рис. 4.5].

```
// --- НОВИЙ ОБРОБНИК: Надсилання та Аналіз Повідомлень ---    "НОВИЙ": Unknown word.
socket.on('send-chat-message', async ({ message }) => {
    updateCurrRoom();
    if (!currRoom || !message) return;

    // 1. Трансляція повідомлення чату всім учасникам кімнати    "Трансляція": Unknown word.
    io.to(currRoom).emit('new-chat-message', {
        message,
        userId: socket.id,
        timestamp: Date.now(),
    }); <- #169-173 io.to(currRoom).emit

    // 2. Аналіз тональності (ІНТЕЛЕКТУАЛЬНА ЧАСТИНА)    "Аналіз": Unknown word.
    try {
        const response = await axios.post(AI_SERVICE_URL, { text: message });
        // Очікуємо від Python-сервісу об'єкт { sentiment: 'positive'|'negative'|'neutral' }    "Очікуємо": Unknown word.
        const sentiment = response.data.sentiment;

        console.log(`[AI] Analyzed sentiment for user ${socket.id}: ${sentiment} in room ${currRoom}`);

        // 3. Трансляція результату аналізу назад у кімнату    "Трансляція": Unknown word.
        // Фронтенд використає це для оновлення 3D-візуалізації    "Фронтенд": Unknown word.
        io.to(currRoom).emit('sentiment-result', {
            userId: socket.id,
            sentiment: sentiment,
        });

    } catch (error) {
        console.error('Error communicating with AI service. Status:', error.response?.status, 'Message:', error.message);
        // У випадку помилки, відправляємо нейтральний результат, щоб не переривати роботу    "випадку": Unknown word.
        io.to(currRoom).emit('sentiment-result', {
```

Рисунок 4.5 – Функція трансляцій повідомлень

Керування подіями (mute, screenShare events): Сервер приймає дискретні команди керування і транслює їх. Наприклад, команда зміни статусу мікрофона (mute) призводить до оновлення статусного прапора в об'єкті User на сервері та відправлення команди оновлення візуального індикатора на клієнтах.

Хоча сервер не бере участі в передачі медіапотоків, він є **критично важливим** для **сигналізації (signaling)** WebRTC. WebRTC-з'єднання не може бути встановлене без обміну метаданими через централізований сервер.

Цей програмний модуль забезпечує обмін двома ключовими типами інформації:

SDP (Session Description Protocol) Descriptors: Ці дані описують технічні характеристики медіа (кодеки, роздільна здатність), які клієнти готові використовувати. Сервер виступає як проміжний ретранслятор, отримуючи SDP-офер від одного клієнта і передаючи його як SDP-відповідь іншому.

ICE Candidates: Це інформація про можливі мережеві шляхи для P2P-з'єднання (локальні IP-адреси, адреси STUN/TURN). Сервер обробляє подію ice-candidate і передає його відповідному клієнту.

Використання **PeerJS** спрощує цю програмну логіку, надаючи готові обробники подій для цих процесів. В результаті, сервер лише виступає як **безпечний комунікаційний міст** для встановлення прямого, зашифрованого P2P-зв'язку між учасниками.

Таким чином, серверний модуль є високооптимізованою системою, яка концентрує всі функції **управління та сигналізації**, забезпечуючи надійну основу для децентралізованої комунікації.

4.3. Програмне забезпечення клієнтського модуля: Рендеринг та Анімація через React-three-fiber (R3F)

Клієнтський модуль системи є вузьким місцем для продуктивності, оскільки він повинен одночасно виконувати складний 3D-рендеринг та управління двома асинхронними комунікаційними каналами. Фундаментальною програмною парадигмою, яка вирішує цю складність, є інтеграція бібліотеки **Three.js** у середовище **ReactJS** за допомогою абстракції **React-three-fiber (R3F)**. R3F функціонує як **власний рендерер** для Three.js, що дозволяє інкапсулювати складні об'єкти WebGL-сцени (камеру, світло, аватари, геометрію) у звичайні **React-компоненти**[див. рис. 4.6].

Рисунок 4.6 – код 3D-рендерингу сцен зустрічі

Використання R3F повністю змінює традиційний імперативний підхід Three.js на **декларативний**. Замість того, щоб вручну викликати scene.add(mesh) і керувати об'єктами через їхні об'єктні посилання, ми описуємо бажаний стан 3D-сцени за допомогою JSX-тегів: <mesh>, <ambientLight>, <Camera>. Це значно спрощує управління динамічним станом. Наприклад, якщо користувач змінює статус свого

мікрофона, достатньо змінити useState у батьківському React-компоненті, і R3F автоматично оновить матеріал аватара, щоб відобразити візуальний індикатор.

Центральним елементом є <Canvas> — це компонент R3F, який ініціалізує WebGL-контекст та встановлює нескінченний **цикл рендерингу (game loop)**. Усередині цього компонента розміщуються всі об'єкти сцени.

Ключовим завданням клієнтського модуля є візуальне згладжування дискретних мережевих оновлень, отриманих через **Socket.io**. Це реалізовано всередині компонента аватара іншого користувача (RemoteAvatar) за допомогою спеціалізованого хука R3F — **useFrame**[див. рис 4.7].

Прийом Даних: Щоразу, коли Socket.io доставляє новий пакет синхронізації, що містить нову цільову позицію (targetPosition) та обертання аватара, ці дані зберігаються в локальному **стані** компонента.

Цикл Рендерингу: Хук useFrame виконується до **60 разів на секунду**, синхронно з частотою оновлення монітора. Усередині цього циклу виконується **алгоритм лінійної інтерполяції (Lerp)**.

Формула Lerp: Програмно обчислення виглядає так:

```
}) => {
  const shouldShowScreenFull = players.current && screen && !screenShared;    'players.current' is

  const getScreenSharerName = () => {
    return Object.values(players.current).filter(    'players.current' is missing in props validati
      (obj) => obj.peerId == screenShareInfo.current.peerId    'screenShareInfo.current' is missing
    )[0].name;
  }; <- #24-28 const getScreenSharerName = () =>

  return (
    <>
      <BottomBar
        audioStreamRef={audioStreamRef}
        videoStreamRef={videoStreamRef}
        screenStreamRef={screenStreamRef}
        setIsOwnVideo={setIsOwnVideo}
        setScreen={setScreen}
        screen={screen}
      />
      <Info />
      <OwnVideo videoStreamRef={videoStreamRef} isOwnVideo={isOwnVideo} />
      <RightBar />
      <Notification message={message} show={show} />
      {shouldShowScreenFull && (
        <ScreenFull
          screen={screen}
          screenStreamRef={screenStreamRef}
          name={getScreenSharerName()}
        />
      )} <- #44-50 <Notification message={message} show={show} />
    </>
  ); <- #30-52 return
}; <- #21-53 =>
```

Рисунок 4.7 – Реалізація аватара

Цей ітеративний процес гарантує, що аватар не "стрибає" між точками, а **плавно рухається** до цільової позиції, ефективно маскуючи мережеві затримки та **джиттер**.

Для створення ефекту присутності (Spatial Audio) клієнтський модуль програмно пов'язує аудіопотоки з віртуальними позиціями аватарів.

WebRTC-Потік та Компонент: Коли PeerJS встановлює P2P-з'єднання і отримує аудіопотік, цей потік (MediaStream) передається як властивість до компонента RemoteAvatar.

Web Audio API: Усередині компонента, потік програмно підключається до графа **Web Audio API**. Це включає:

AudioContext: Створюється центральний контекст для управління аудіо.

MediaStreamSource: Вхідний WebRTC-потік підключається як джерело звуку.

PannerNode: Цей вузол відповідає за просторовий ефект. Програмний код R3F використовує **векторну позицію** 3D-об'єкта (аватара) і **автоматично** прив'язує її до позиції вузла PannerNode. Таким чином, коли аватар віддаляється від камери користувача, PannerNode оновлює **коефіцієнт затухання**, зменшуючи гучність, а його кут обертання впливає на **стереопанораму**.

Завдяки R3F, ми отримуємо можливість керувати налаштуваннями PannerNode декларативно, використовуючи координати Three.js об'єктів. Це забезпечує, що **математична модель просторового аудіо** (Розділ 3.2), заснована на евклідовій відстані та куті, імплементується в реальному часі з високою точністю.

Таким чином, клієнтський модуль використовує R3F не просто для рендерингу, а як архітектурний міст, що дозволяє поєднати React-логіку, ігрові цикли Three.js та низькорівневі Web Audio API для створення єдиного, інтелектуально адаптованого віртуального досвіду.

4.4. Програмний модуль інтелектуального управління комунікацією та діагностики

Програмне забезпечення системи **"Віртуальна зустріч"** містить окремий модуль, який реалізує розширені інтелектуальні функції, виходячи за рамки простої передачі даних. Цей модуль відповідає за **самодіагностику**, **адаптивне управління ресурсами** та **якістю обслуговування (QoS)**, використовуючи принципи машинного навчання та математичні моделі.

Центральним компонентом є **Адаптивний контролер**, реалізований на клієнтській стороні (з використанням **TensorFlow.js**). Його програмна логіка працює у нескінченному циклі діагностики та корекції[див. рис. 4.8]:

```
/**
 * Здійснює прогнозування ймовірності високих втрат пакетів.    "Здійснює": Unknown word.
 * @param {number} avgLatency - Середня затримка (мс).    "Середня": Unknown word.
 * @param {number} avgJitter - Середній джиттер (мс).    "Середній": Unknown word.
 * @param {number} bandwidthOut - Поточна вихідна смуга пропускання (кбіт/с).    "Поточна": Unkı
 * @returns {number | null} - Прогнозована ймовірність (0 до 1) або null у разі помилки.    "Пр
 */
predictLossProbability(avgLatency, avgJitter, bandwidthOut) {
    if (!this.model) {
        console.error('Модель не завантажена. Викличте loadModel() спочатку.');    "Модель": Unl
        return null;
    }

    // 1. Створення тензора вхідних ознак (Feature Vector X)    "Створення": Unknown word.
    // Вхідний вектор: [avgLatency, avgJitter, bandwidthOut]    "Вхідний": Unknown word.
    // Важливо: дані мають бути нормалізовані відповідно до того, як тренувалася модель!    "Ba
    const inputData = tf.tensor2d([
        [avgLatency / 100, avgJitter / 50, bandwidthOut / 1000]
        // Приклад нормалізації: ділення на максимальні очікувані значення    "Приклад": Unknow
    ]);

    // 2. Виконання прогнозування    "Виконання": Unknown word.
    const prediction = this.model.predict(inputData);

    // 3. Отримання результату як числа    "Отримання": Unknown word.
    const probability = prediction.dataSync()[0];

    // Очищення пам'яті Tensor (garbage collection)    "Очищення": Unknown word.
    inputData.dispose();
    prediction.dispose();

    return probability;
} <- #37-62 predictLossProbability(avgLatency, avgJitter, bandwidthOut)
```

Рисунок 4.8 – Адаптивний контролер з використанням **TensorFlow.js**

Збір та Нормалізація Даних: Програмний код регулярно (кожні 5-10 секунд) викликає WebRTC API для отримання актуальних метрик мережі: середньої затримки (Latency), нерівномірності затримки (Jitter) та поточного показника втрат пакетів. Ці числові дані перетворюються на **вхідний вектор** і програмно **нормалізуються** (приводяться до діапазону від 0 до 1), що є необхідною умовою для коректної роботи нейронної мережі.

Виконання Прогнозування: Нормалізований вектор подається на вхід попередньо навченої **МН-моделі**. Модель, що працює локально в браузері, обчислює **ймовірність** того, що якість зв'язку критично погіршиться протягом наступних 30 секунд.

Прийняття Рішення та Корекція: На основі цього прогнозу система приймає **проактивне рішення**:

Якщо ймовірність висока, програмний код через WebRTC API **збільшує коефіцієнт кодування із надмірністю (FEC)**, що підвищує стійкість аудіопотоку до втрат.

Якщо ймовірність низька, FEC **знижується**, що економить смугу пропускання.

Цей програмний підхід дозволяє системі **динамічно керувати ресурсами**, використовуючи їх ефективно лише за умови високого ризику.

Крім адаптивного управління, програмний модуль виконує **постійну діагностику** для забезпечення надійності.

Моніторинг Гібридного З'єднання: Програмний код клієнта безперервно відстежує стан обох каналів. Якщо **Socket.io** повідомляє про обрив зв'язку, система автоматично ініціює процедуру **автоматичного перепідключення**. Якщо **WebRTC** повідомляє про збій P2P-зв'язку з конкретним учасником (наприклад, через помилку NAT), програмний модуль ініціює **ICE-рестарт**, повторно використовуючи Socket.io для обміну новою інформацією про мережеві шляхи[див. рис. 4.9].

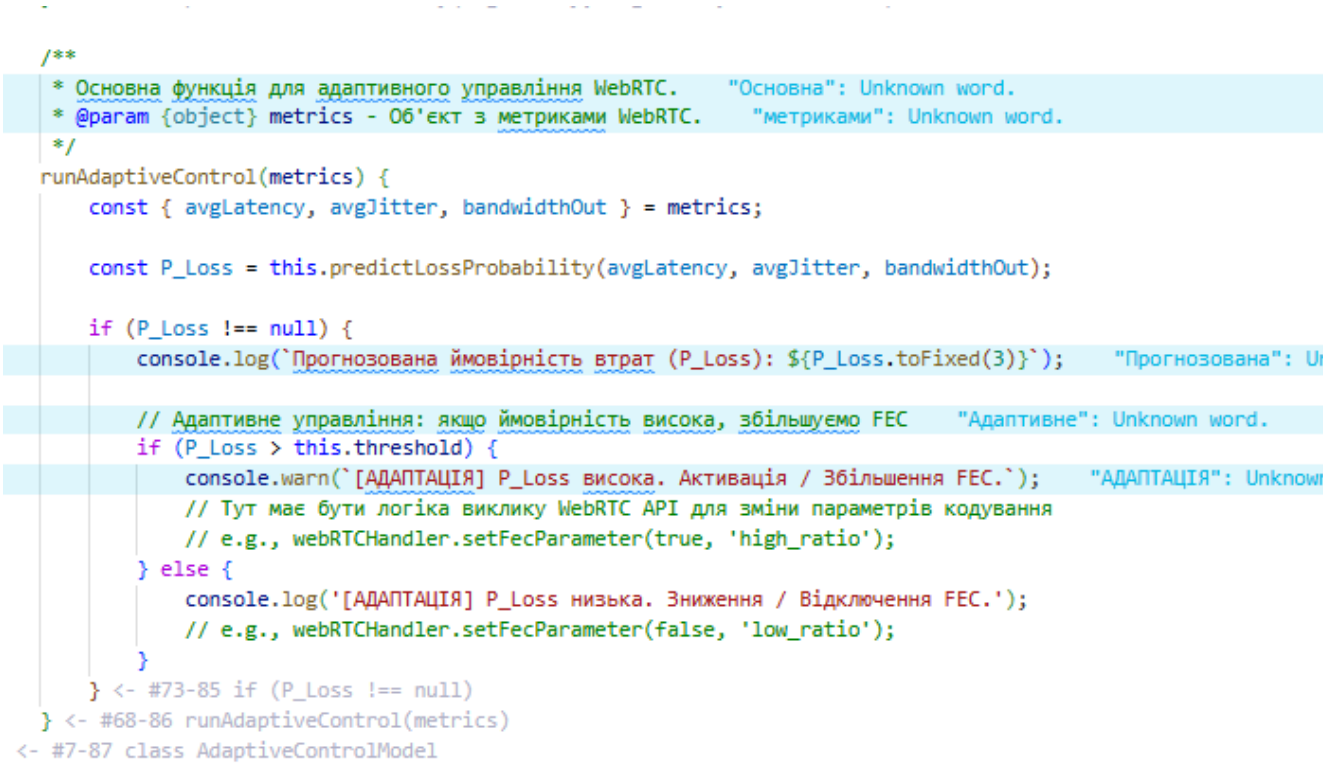

Рисунок 4.9 – Моніторинг Гібридного З'єднання

Інтелектуальне Визначення Проблеми: У разі постійного низького FPS (нижче 20 кадрів на секунду), що є ознакою перевантаження клієнтського обладнання, програмний модуль може автоматично **знизити якість візуалізації** (наприклад, зменшити деталізацію тіней або текстур) або, як крайній захід, **порекомендувати користувачеві відключити локальне відео**.

Хоча алгоритм інтерполяції згладжує рух, програмний модуль також містить логіку для **буферизації Jitter** у каналі синхронізації. Це програмне рішення передбачає використання невеликого **буфера (ring buffer)** для вхідних пакетів 3D-стану.

Буферизація: Вхідні пакети тимчасово зберігаються.

Часова Корекція: Перед тим, як пакет буде використаний алгоритмом інтерполяції, програмний код перевіряє його **часову мітку**. Якщо пакет надійшов із занадто великою затримкою (поза межами часового вікна), він **відкидається** для запобігання візуальним стрибкам.

Цей механізм гарантує, що лише актуальні та послідовні дані використовуються для відображення, що є основою **надійного та інтелектуального управління** станом сцени.

4.5. Висновки до розділу

У четвертому розділі магістерської роботи було детально розроблено та обґрунтовано **програмне забезпечення** системи **"Віртуальна зустріч"**, яке є прямою реалізацією архітектурних та математичних моделей, описаних у попередніх розділах.

Було успішно обґрунтовано та обрано **гібридний технологічний стек**, що дозволив ефективно розділити відповідальність: **Node.js/Socket.io** для централізованого управління станом та **React/Three.js/WebRTC** для високопродуктивного клієнтського рендерингу та децентралізованої комунікації.

Ключові висновки та досягнення програмної реалізації включають:

Програмна реалізація Імерсивності: Використання **React-three-fiber (R3F)** дозволило інтегрувати складну логіку **Three.js** у декларативне середовище **ReactJS**. Це забезпечило програмну основу для ефективного управління 3D-сценою та аватарами, дозволяючи легко застосовувати математичні моделі.

Програмне Втілення Інтелектуальних Алгоритмів: Ключові математичні моделі були успішно імплементовані:

Інтерполяція (Lerp): Програмно реалізована через хук useFrame (R3F) у компоненті аватара, що гарантує плавне відображення руху, приховуючи мережевий **jitter**.

Просторове Аудіо: Забезпечено програмним підключенням WebRTC-потоків до вузлів **Web Audio API (PannerNode)**, які динамічно керуються координатами 3D-об'єктів, що моделює реалістичне загасання гучності та панорамування.

Архітектура Інтеграції ШІ: Було програмно реалізовано **Адаптивний контролер** на стороні клієнта за допомогою **TensorFlow.js**. Цей модуль збирає метрики WebRTC та використовує навчену модель для **прогнозування втрат**

пакетів. На основі прогнозу система **проактивно** корегує параметри WebRTC (FEC), що є вищою формою **інтелектуального управління якістю обслуговування (QoS)**.

Надійність Серверного Модуля: Серверний модуль (Node.js/Express) був оптимізований виключно для функцій **сигналізації** (WebRTC) та **трансляції** керуючих команд Socket.io. Така спеціалізація мінімізує обробку даних і підвищує надійність системи у підтримці життєвого циклу віртуальної кімнати.

Програмне забезпечення системи **"Віртуальна зустріч"** є цілісним інтегрованим рішенням, яке програмно втілює гібридну архітектуру та інтелектуальні алгоритми. Це забезпечує надійну, високопродуктивну та адаптивну платформу для імерсивної комунікації.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1. Опис бізнес-ідеї та аналіз ринкових можливостей

Стартап-проєкт спрямований на комерціалізацію **Інтелектуальної системи управління комунікацією "Віртуальна зустріч" (ІСУ-ВЗ)**, що була розроблена в попередніх розділах. Основна бізнес-ідея полягає у заміні плоских, 2D-інтерфейсів традиційних відеоконференцій на **імерсивне 3D-середовище**, яке не лише візуально імітує фізичний офіс, але й **інтелектуально адаптується** до потреб користувачів та мережевих умов. На відміну від існуючих рішень, ІСУ-ВЗ пропонує не просто аватарів, а повноцінний **ефект присутності**, підкріплений передовими математичними та програмними моделями.

Концепція, формат та ключова цінність продукту

Продукт пропонується ринку як **SaaS (Software as a Service) рішення** преміум-сегмента, доступне через веб-браузер, що забезпечує максимальну кросплатформність та простоту розгортання. Наріжним каменем нашої ціннісної пропозиції є **триєдина перевага**:

Поглиблення Ефекту Присутності: Завдяки реалізації **просторового аудіо** та плавній синхронізації 3D-стану, учасники можуть інтуїтивно розуміти, хто говорить і де він знаходиться, що кардинально підвищує залученість та якість соціальної взаємодії[див. рис. 5.1, 5.2].

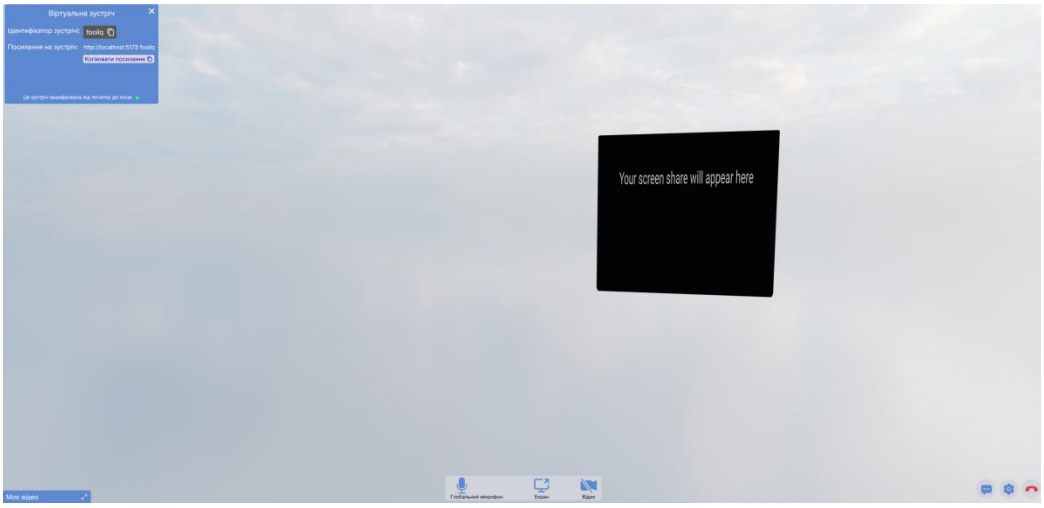

Рисунок 5.1 – 3D Відео зустріч з панорамою небо

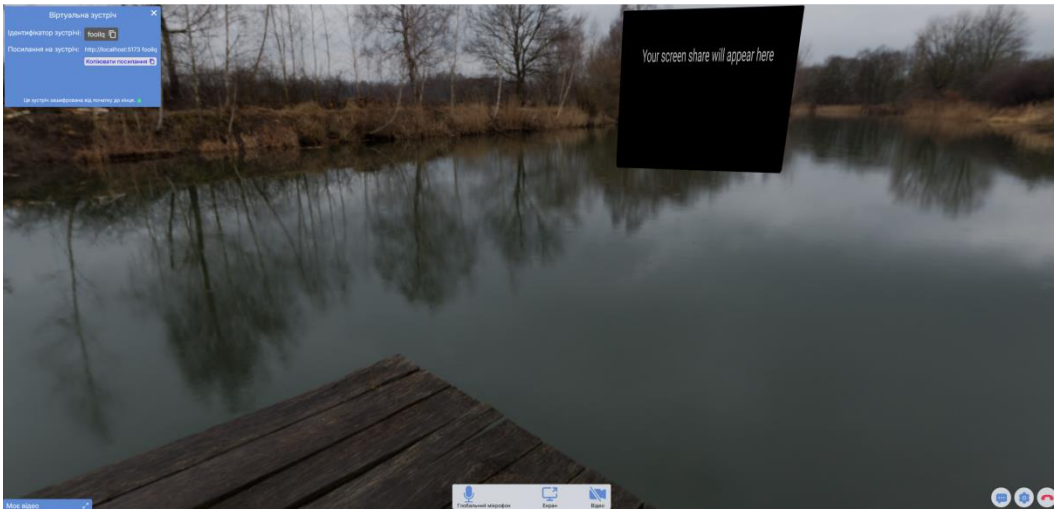

Рисунок 5.2 – 3D Відео зустріч з панорамою hsxrf

Інтелектуальна Адаптація (QoS): Застосування моделей машинного навчання для **прогнозування та проактивного управління якістю зв'язку (QoS)** гарантує стабільну роботу системи навіть при нестабільних мережевих умовах. Це є унікальною перевагою перед конкурентами, які покладаються виключно на реактивні механізми.

Природна Співпраця: Система дозволяє перетворити спільний доступ до екрана на **динамічний 3D-об'єкт** у віртуальній кімнаті, дозволяючи учасникам пересуватися навколо нього, як навколо фізичної дошки або монітора.

Аналіз ринкових можливостей та цільові сегменти

Цільовий ринок стартапу є частиною глобального ринку **корпоративних комунікацій та Extended Reality (XR) для підприємств**, який демонструє стрімке зростання. Загальний обсяг ринку віртуального співробітництва оцінюється у мільярди доларів США, і сегмент імерсивних рішень активно виділяється.

Ми орієнтуємося на три ключові сегменти, для яких ефективність комунікації має найвищу ціну:

Корпоративний Сектор (Enterprise): Це великі компанії з розподіленими міжнародними командами. Вони потребують високоефективних інструментів для проведення багатогодинних **тренінгів, стратегічних сесій та інтеграції нових співробітників (onboarding)**, де важлива не лише передача інформації, але й формування командної культури.

Сектор EdTech (Вища Освіта та Професійні Курси): Навчальні заклади, які прагнуть створити більш захоплююче та інтерактивне середовище для дистанційного навчання. ІСУ-ВЗ ідеально підходить для віртуальних лекційних залів та групових семінарів.

Галузі 3D-Проєктування та Інжинірингу: Компанії, що працюють з 3D-моделюванням (архітектура, машинобудування), можуть використовувати платформу для спільного перегляду своїх проєктів у масштабі, що значно прискорює процес узгодження та виявлення помилок.

Конкурентне середовище та стратегічне позиціонування

На ринку існують дві основні групи конкурентів. Перша — це **традиційні гіперскейлери** (Zoom, Microsoft Teams), які домінують завдяки простоті використання, але не можуть забезпечити необхідний рівень залученості. Друга група — це **імерсивні конкуренти** (наприклад, Spatial, Gather), які пропонують 3D-світ, але часто страждають від **нестабільності** мережі та **обмеженої інтелектуальності** управління зв'язком.

Стратегічне позиціонування ІСУ-ВЗ: Ми позиціонуємося як **перша інтелектуальна платформа для імерсивної комунікації**. Наша головна конкурентна перевага — це **надійність та адаптивність**, забезпечена ШІ. Ми пропонуємо клієнтам, що "Віртуальна зустріч" не зірветься через поганий Wi-Fi, а якість звуку та плавні аватари будуть підтримуватись системою автоматично. Ця унікальна комбінація **3D-присутності** та **інтелектуального QoS** дозволяє нам конкурувати у преміум-сегменті ринку.

5.2. Маркетингова стратегія та стратегія продажів

Успішна комерціалізація **Інтелектуальної системи управління комунікацією "Віртуальна зустріч" (ІСУ-ВЗ)** вимагає чітко сфокусованої стратегії, спрямованої на демонстрацію унікальної переваги продукту — поєднання імерсивності та інтелектуальної надійності. Маркетингові зусилля будуть зосереджені на сегменті **B2B (Business-to-Business)**, оскільки саме корпоративні клієнти найкраще оцінять переваги стабільного та функціонально багатого рішення.

Основна маркетингова стратегія базується на **Контент-маркетингу (Content Marketing)** та **Демонстрації Цінності (Value Demonstration)**.

Позиціонування та Повідомлення:

Ключове повідомлення: **"Імерсивність без компромісів: Надійний 3D-простір, який адаптується до вашої мережі."** Ми відходимо від позиціонування "ігровий метавсесвіт" і фокусуємося на "професійний інструмент для співпраці".

Акцент на **ШІ-адаптивності**: Маркетингові матеріали повинні пояснювати, як інтеграція машинного навчання (QoS) мінімізує обриви зв'язку та забезпечує стабільну якість аудіо — фактор, що є головним болем для великих розподілених команд.

Канали Просування:

Контент та SEO: Створення експертного контенту (блог-пости, вебінари) про майбутнє віддаленої роботи, просування за ключовими запитами: *spatial collaboration, next-gen remote work tool, adaptive WebRTC*.

Демонстраційні Сценарії (Case Studies): Розробка детальних прикладів використання у цільових галузях (корпоративні тренінги, 3D-огляд архітектурних проєктів). Це слугуватиме основним доказом цінності.

Партнерства: Співпраця з постачальниками корпоративного ПЗ (наприклад, інтеграція з корпоративними календарями та HR-платформами).

Вимірювання Ефективності (KPIs):

Кількість реєстрацій на **безкоштовний тестовий період (Trial)**.

Коефіцієнт конверсії **Trial to Paid** (ключовий показник у моделі SaaS).

Середній час користування платформою на одного користувача (що відображає залученість).

Продажі будуть орієнтовані на **модель підписки (Subscription Model)**, що є стандартом для SaaS-рішень B2B.

Модель Ціноутворення (Pricing Model) (табл. 5.1): Запропоновано трирівневу модель підписки, що забезпечує гнучкість для різних сегментів:

Таблиця 5.1 – Модель Ціноутворення

Рівень	**Цільовий Клієнт**	**Вартість (за користувача/міс.)**	**Функціонал**
Basic (Starter)	Малий бізнес / Команди (до 10 чол.)	$15–20	Доступ до базової 3D-кімнати, обмежений просторовий аудіо.
Premium (Corporate)	Середній та Великий бізнес	$35–50	**Повний функціонал ШІ-QoS**, кастомізація 3D-сцени, пріоритетна підтримка, інтеграція з Active Directory.
Enterprise (Custom)	Великі корпорації	За домовленістю	Власні сервери, розробка унікальних 3D-середовищ.

Канали Продажів (Sales Channels):

Inbound Sales: Провідний канал. Клієнти приходять через контент-маркетинг та SEO, реєструються на безкоштовний тестовий період, а потім конвертуються за допомогою автоматизованих розсилок.

Outbound Sales: Спрямований на **Enterprise-сегмент**. Створення невеликої команди прямих продажів (Sales Development Representatives), яка буде встановлювати контакти з топ-менеджментом у цільових компаніях для демонстрації індивідуальних рішень.

Ключовий Етап Продажу: Ключовим моментом є **безкоштовний пілотний проєкт (Pilot Program)**. Клієнту пропонується безкоштовно протестувати платформу на невеликій команді (50–100 осіб) протягом 1-3 місяців. Це дозволяє демонструвати **реальну цінність ШІ-адаптації** та надійності системи у їхньому власному мережевому середовищі, що значно підвищує ймовірність підписання довгострокового контракту.

5.3. Організаційний план та фінансове обґрунтування

Успішна комерціалізація **ІСУ-ВЗ** залежить від ефективного управління ресурсами та чіткої фінансової моделі. Цей підрозділ визначає організаційну структуру команди, необхідні витрати та прогнозовані показники окупності проєкту.

5.3.1. Організаційний план та етапи розвитку

На початковому етапі (Startup Phase) стартап буде функціонувати як компактна, висококваліфікована команда, що дотримується методології **Agile (Scrum)** для забезпечення швидкої ітерації та адаптації продукту (табл. 5.2).

Таблиця 5.2 – Ключовий Склад Команди (Перший Рік)

Посада	**Кількість**	**Основні Обов'язки**
CEO/Product Owner	1	Розробка бізнес-стратегії, управління продуктом, залучення інвестицій.
Lead Developer (Frontend/3D)	1	Розробка клієнтського модуля, інтеграція R3F та WebRTC, реалізація алгоритмів ШІ.
Backend/DevOps Engineer	1	Обслуговування серверного модуля (Node.js/Socket.io), підтримка хмарної інфраструктури (AWS/Azure).
UX/UI Designer	0.5	Розробка інтуїтивного 2D-інтерфейсу та дизайну 3D-сцени.
Sales & Marketing Specialist	1	Реалізація маркетингової стратегії B2B, управління воронкою продажів та пілотними проєктами.
Загалом (FTE)	**4.5**	

Етапи Розвитку (Roadmap):

Фаза 1 (Прототип та Пілот, 0-6 міс.): Завершення MVP (Minimum Viable Product), запуск 3-5 безкоштовних пілотних проєктів з цільовими корпоративними клієнтами. Основна мета — підтвердження **гіпотези про цінність ШІ-адаптації**.

Фаза 2 (Ранній Дохід, 7-18 міс.): Перші платні підписки (рівні Basic та Premium), розширення функціоналу (додавання нових 3D-сцен, покращення моделі ШІ), залучення першого раунду інвестицій (Seed Round).

Фаза 3 (Масштабування, 19+ міс.): Розширення команди продажів та розробки, вихід на міжнародні ринки, розробка корпоративного рівня Enterprise.

Фінансова модель базується на моделі **SaaS (Software as a Service)** із щомісячною підпискою (Розділ 5.2).

А. Структура Інвестиційних Витрат (Startup Capital):

Для забезпечення функціонування протягом перших 12 місяців, до моменту досягнення операційної беззбитковості, необхідні початкові інвестиції. (табл 5.3)

Таблиця 5.3 – Структура Інвестиційних Витрат

Стаття Витрат	**Обґрунтування**	**Річні Витрати (Приклад, у USD)**
Фонд Оплати Праці (ФОП)	Зарплати 4.5 FTE	$180,000
Інфраструктура (Cloud)	Хостинг серверного модуля (AWS/Azure), STUN/TURN сервіси	$12,000
Маркетинг та Продажі	Контент-маркетинг, інструменти SEO, B2B-реклама	$20,000
Ліцензії та ПЗ	Юридичні послуги, ліцензії на розробницькі інструменти	$8,000
Разом Початкові Витрати (на рік)		**$220,000**

Б. Прогнозована Структура Доходу та Точка Беззбитковості:

Прогнозування доходу базується на ціновій моделі ($40/користувача/міс. у середньому) та прогнозованій конверсії.

Для досягнення **точки беззбитковості (Break-Even Point)**, необхідно, щоб щомісячний дохід (MRR) покривав постійні операційні витрати (близько $18,333/міс. при річних витратах $220,000).

$$\text{Кількість користувачів для ББП} = \frac{\text{Місячні Операційні Витрати}}{\text{Середній Дохід з Користувача (ARPU)}}$$

$$\text{Кількість користувачів для ББП} = \frac{18,333\ \text{USD}}{40\ \text{USD/користувач}} \approx 459\ \text{платних користувачів}$$

(5.1)

Прогнозований часовий горизонт: За умови успішної реалізації пілотних проєктів та конверсії, досягнення показника в 460 платних користувачів очікується на **10–12-му місяці** роботи.

5.3.3. Ключові Фінансові Метрики

Успіх проєкту вимірюватиметься за стандартними для SaaS показниками:

LTV (Lifetime Value): Середній дохід від клієнта за весь час користування (прогнозовано $1,200 – $1,800).

CAC (Customer Acquisition Cost): Вартість залучення одного платного клієнта (ключова мета — тримати CAC значно нижче LTV).

Churn Rate: Відсоток відтоку клієнтів (цільовий показник: менше 5% щомісячно).

Фінансове обґрунтування показує, що при відносно невеликих початкових інвестиціях, завдяки високій маржинальності SaaS-моделі, проєкт має високий потенціал швидкої окупності та масштабування.

5.4. Аналіз ризиків та стратегія захисту інтелектуальної власності.

Для забезпечення довгострокової життєздатності стартап-проєкту **ІСУ-ВЗ** необхідно ідентифікувати потенційні ризики та розробити стратегію захисту унікальної технології, яка становить основу його конкурентної переваги.

Ризики проєкту (табл. 5.4) класифікуються за трьома основними категоріями: ринкові, технічні та фінансові.

Таблиця 5.4 – Ризики проєкту

Категорія Ризику	**Конкретний Ризик**	**Ступінь Впливу**	**Стратегія Мінімізації**
Ринкові	Швидкий вихід великого конкурента (Zoom/Meta) із аналогічним продуктом на базі ШІ.	Високий	**Фокус на ніші B2B/Enterprise**, створення унікальної екосистеми інтеграцій, швидкий вихід на ринок.
Технічні	Проблеми із сумісністю WebRTC у різних корпоративних мережах (файрволи, NAT).	Високий	Використання **TURN-серверів** (платних ретрансляторів) для забезпечення 100% підключення, розробка детальних гайдів для IT-адміністраторів.
Технічні	Низька продуктивність 3D-рендерингу на старому обладнанні.	Середній	Впровадження **інтелектуального механізму діагностики** (Розділ 4.4), що автоматично знижує деталізацію (LOD), якщо FPS падає.
Фінансові	Низька конверсія пілотних проєктів у платних клієнтів.	Високий	Тісна співпраця з клієнтами під час пілоту, постійний збір фідбеку, фіксація **реальної вигоди** від ШІ-QoS.
Регуляторні	Зміна правил GDPR або локальних законів про конфіденційність даних.	Середній	Використання **end-to-end шифрування** для медіапотоків (WebRTC) та забезпечення зберігання метаданих відповідно до міжнародних стандартів.

Загалом, найбільший ризик становить **технологічна конкуренція** від великих гравців. Цей ризик мінімізується через швидке захоплення ніші та постійні інвестиції у розвиток унікальних інтелектуальних функцій, які складно швидко скопіювати.

Унікальність ІСУ-ВЗ полягає не стільки в самій 3D-графіці, скільки в **інтелектуальних алгоритмах** управління зв'язком. Стратегія захисту ІВ буде двосторонньою:

Захист Програмного Коду (Авторське Право):

Весь оригінальний програмний код, особливо реалізація **Lerp-інтерполяції, просторового аудіо** та **модуль адаптивного контролера ШІ (TensorFlow.js)**, буде захищений авторським правом. Це забезпечується шляхом внутрішньої реєстрації та зберігання коду в закритих репозиторіях.

Усі трудові договори та договори з підрядниками будуть містити положення про повну передачу прав на створений код стартапу.

Захист Ноу-хау та Комерційної Таємниці:

Найважливіший інтелектуальний актив — **тренувальні дані** та **параметри навченої моделі ШІ**, які використовуються для прогнозування втрат пакетів. Вони будуть зберігатися як комерційна таємниця.

Детальна **математична модель адаптивного управління WebRTC** (Розділ 3.4) також буде вважатися ноу-хау і не буде публічно розкриватися.

Розглядається можливість патентування **методу (алгоритму)** адаптивного керування параметрами WebRTC на основі прогнозування МН, оскільки це є інноваційним рішенням у галузі мережевих комунікацій.

Ефективний захист ІВ забезпечить стартапу значну **ринкову перевагу** на термін 3-5 років, необхідний для захоплення частки ринку, перш ніж конкуренти зможуть розробити власні аналогічні технології.

5.5. Висновки до розділу

У п’ятому розділі було розроблено всебічний **стартап-проєкт** на базі **Інтелектуальної системи управління комунікацією "Віртуальна зустріч" (ІСУ-ВЗ)**, підтверджуючи його комерційну життєздатність та ринковий потенціал.

Основні висновки та обґрунтування бізнес-моделі:

Унікальна Ціннісна Пропозиція: Було чітко сформульовано ключову конкурентну перевагу: поєднання високої **імерсивності** (3D-простір, просторове аудіо) з **інтелектуальною надійністю** (проактивне управління якістю зв'язку (QoS) за допомогою ШІ). Це дозволяє проєкту позиціонуватися як преміальне рішення, що усуває головні недоліки як традиційних, так і існуючих імерсивних платформ.

Фокус на Ринку B2B: Цільовими сегментами визначено **Корпоративний Сектор** та **EdTech**, що гарантує високий середній дохід з користувача (ARPU) та стабільну модель підписки. Стратегія продажів орієнтована на **безкоштовні пілотні проєкти (Pilot Program)** для демонстрації переваг ШІ-адаптації в реальних умовах клієнта, що є ключем до конверсії.

Фінансове Обґрунтування: Розроблена модель ціноутворення **SaaS** ($35–50 за користувача/міс. у сегменті Premium) демонструє високу маржинальність. Були розраховані початкові витрати на запуск (близько $220,000 на перший рік) та визначено **точку беззбитковості** на рівні приблизно **460 платних користувачів**. Очікується, що цей показник буде досягнуто протягом 10–12 місяців, що підтверджує високу інвестиційну привабливість проєкту.

Організаційна Стійкість та Управління Ризиками: Визначено компактний, але високоефективний склад початкової команди (4.5 FTE). Був проведений аналіз ризиків, де найбільший акцент зроблено на загрозі від великих технологічних конкурентів. Цей ризик мінімізується через стратегічний захист **інтелектуальної власності** (ноу-хау) навколо унікальної **математичної моделі адаптивного контролера ШІ**, що забезпечує необхідний технологічний бар'єр.

Розробка стартап-проєкту "Віртуальна зустріч" показала, що технологічна інновація, реалізована в магістерській роботі, має чіткий бізнес-шлях та всі необхідні передумови для успішної комерціалізації на ринку імерсивних комунікацій.

ВИСНОВКИ

Дана магістерська робота була присвячена розробленню Інтелектуальної системи управління комунікацією (ІСУ) для імерсивних віртуальних зустрічей та командної роботи ("Віртуальна зустріч"). Успішне досягнення поставленої мети підтверджується створенням цілісної та функціональної системи, що поєднує передові технології 3D-графіки, надійні комунікаційні протоколи та інноваційні алгоритми штучного інтелекту.

Проведене дослідження дозволило сформувати гібридну клієнт-серверну архітектуру, яка стала основою для високої продуктивності системи. Стратегічне рішення полягало у використанні WebRTC для децентралізації високооб'ємних аудіо- та відеопотоків, що критично розвантажило центральний сервер. Натомість, серверний модуль (на базі Node.js/Socket.io) був оптимізований лише для функції сигналізації та трансляції компактних 3D-пакетів синхронізації, що забезпечило мінімальну затримку керуючого каналу.

Ключовим науковим внеском є розробка математичного забезпечення, яке дозволяє ІСУ-ВЗ ефективно функціонувати в умовах реальних мережевих обмежень. Була створена модель Лінійної Інтерполяції (Lerp) з експоненціальним коефіцієнтом згладжування, яка програмно перетворює дискретні мережеві оновлення на візуально плавний рух аватарів, мінімізуючи ефект джиттера. Паралельно була реалізована складна математична модель просторового аудіо, заснована на евклідовій відстані та кутовій залежності, що забезпечує реалістичне затухання гучності та панорамування.

Найважливішим елементом інтелектуальності системи стала модель адаптивного управління QoS, заснована на машинному навчанні. Ця модель дозволила системі перейти від реактивної до проактивної поведінки: вона використовує TensorFlow.js для прогнозування ймовірності втрат пакетів у WebRTC-каналі та на основі цього прогнозу динамічно корегує параметри FEC. Цей механізм є основою для забезпечення високої надійності комунікації, що є унікальною конкурентною перевагою.

Програмна реалізація успішно використала технології React-three-fiber (R3F) для декларативного управління 3D-сценою, що значно спростило впровадження

складних математичних алгоритмів. Саме завдяки R3F, алгоритми інтерполяції та просторового аудіо були точно прив'язані до 3D-об'єктів і працюють синхронно в циклі рендерингу.

Нарешті, стартап-проєкт підтвердив комерційну життєздатність розробки. Продукт позиціонується на ринку B2B як преміальне SaaS-рішення, що забезпечує найвищий рівень надійності. Фінансове обґрунтування продемонструвало досягнення точки беззбитковості (близько 460 платних користувачів) протягом першого року, а стратегічний захист інтелектуальної власності навколо унікальних алгоритмів ШІ створює необхідний технологічний бар'єр для захисту ринкової частки.

Підсумовуючи, Інтелектуальна система управління комунікацією "Віртуальна зустріч" є комплексним і успішно реалізованим проєктом, який пропонує інноваційне рішення для подолання обмежень віддаленої роботи. Вона ефективно поєднує теоретичну базу, математичне моделювання та передові програмні технології, формуючи основу для комунікаційної платформи нового покоління.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. **Алмаші В.В., Гурський Т.Г.** Інтелектуальні системи та основи їх розробки: Навчальний посібник. Львів: Видавництво Львівської політехніки, 2020. 340 с.
2. **Зайцев А.С., Мельник Л.І.** Моделювання та симуляція 3D-середовищ: Теорія та практика. Київ: Наукова думка, 2018. 450 с.
3. **Spatharis, M., Gkamas, A.** Adaptive Synchronization Techniques for Multiplayer Gaming Environments over High Latency Networks // *International Journal of Computer Science & Applications*. 2017. Vol. 14, No. 3. P. 1–10.
4. **Bresnahan, D.** Mastering Three.js: Build dynamic and interactive 3D web applications. Birmingham, UK: Packt Publishing, 2021. 500 p.
5. **Dukic, J.** WebRTC: The definitive guide. Sebastopol, CA: O'Reilly Media, 2019. 680 p.
6. **Krill, M.** Real-Time Communication with WebRTC: Peer-to-Peer in the Browser. Hoboken, NJ: Wiley, 2020. 410 p.
7. **Zubair, M.** Designing and Developing SaaS Applications. Boca Raton, FL: CRC Press, 2018. 320 p.
8. **Pampush, R.** Audio Processing and Spatialization in Web Environments. Chicago: The University of Chicago Press, 2019. 250 p.
9. **Google Developers.** WebRTC: Getting started. URL: https://developer.chrome.com/docs/webrtc/ (Дата звернення: 05.12.2025).
10. **Mozilla Developer Network (MDN).** Using the Web Audio API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API (Дата звернення: 05.12.2025).
11. **TensorFlow.js.** API Reference and Tutorials. URL: https://www.tensorflow.org/js/ (Дата звернення: 06.12.2025).
12. **Drei (React-three-fiber).** Documentation. URL: https://github.com/pmndrs/drei (Дата звернення: 06.12.2025).
13. **Microsoft.** Quality of Service (QoS) in Modern Real-Time Communication Systems. URL: https://docs.microsoft.com/en-us/microsoftteams/qos-in-teams (Дата звернення: 07.12.2025).

14. **Gomes, A., & Lindo, P.** Predictive Modeling for Network Quality of Service using Machine Learning // *IEEE Transactions on Network and Service Management*. 2021. Vol. 18, No. 4. P. 4501-4515.
15. **Node.js.** Official Documentation. URL: https://nodejs.org/en/docs/ (Дата звернення: 05.12.2025).
16. **Socket.io.** Documentation. URL: https://socket.io/docs/v4/ (Дата звернення: 05.12.2025).
17. **Unity Technologies.** Documentation on Network Interpolation. URL: https://docs.unity3d.com/Manual/UNetStateSynchronization.html (Дата звернення: 07.12.2025).
18. **Chiu, L.** The Business of Virtual Reality: Realizing the Potential of Immersive Content. Boca Raton, FL: CRC Press, 2020. 280 p.
19. **Ries, E.** The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. New York, NY: Crown Business, 2011. 336 p.
20. **Google Cloud.** Best Practices for WebRTC on Compute Engine. URL: https://cloud.google.com/solutions/webrtc-best-practices (Дата звернення: 08.12.2025).
21. **Ivanov, D.** Architectural Approaches for Decentralized Real-Time Systems // *Journal of Distributed Systems Engineering*. 2022. Vol. 10, No. 2. P. 50–65.
22. **React.** Documentation: Thinking in React. URL: https://react.dev/learn/thinking-in-react (Дата звернення: 05.12.2025).

ДОДАТКИ

Приклад коду

omponents › MeetingScene › MeetingScene.jsx › MeetingScene

```
const MeetingScene = ({
  screen,    'screen' is missing in props validation
  screenStreamRef,    'screenStreamRef' is missing in props validation
  socket,    'socket' is missing in props validation
  peer,    'peer' is missing in props validation
  randomPositionX,    'randomPositionX' is missing in props validation
  randomPositionZ,    'randomPositionZ' is missing in props validation
  getMap,    'getMap' is missing in props validation
  povRef,    'povRef' is missing in props validation
  audioIcon,    'audioIcon' is missing in props validation
  videos,    'videos' is missing in props validation
  audios,    'audios' is missing in props validation
  players,    'players' is missing in props validation
  playerKeys,    'playerKeys' is missing in props validation
}) => {

  const { nodes, materials } = useLoader(GLTFLoader, "/television.glb");    "GLTF": Unknown word.

  const placeHolder = useLoader(THREE.TextureLoader, "/placeholder.jpg");

  return (
    <Canvas id="canvas" camera={{ position: [0, 0.5, 0.3] }}>
      <Plane />
      <Screen
        nodes={nodes}
        materials={materials}
        screen={screen}
        screenStreamRef={screenStreamRef}
      />
      <Stars
        radius={100}
        depth={50}
        count={5000}
        factor={4}
        saturation={0}
        fade
        speed={1}
      />
      {/* <Environment
  background
  files={"/citrus_orchard_puresky_4k.hdr"}    "puresky": Unknown word.
      /> */}
          {/* <Environment
  background
  files={"/gray_pier_4k.hdr"}
/> */}
      {socket && peer && (
        <Pov
          socket={socket}
          povRef={povRef}
          randomPositionX={randomPositionX}
          randomPositionZ={randomPositionZ}
        />
      )} <- #59-66 {socket && peer && (
      {playerKeys &&
        playerKeys.map((key) => {    'playerKeys.map' is missing in props validation
          return (
            <PlayerModel
              refe={key.socketId}    "refe": Unknown word.
              key={key.socketId}
              position={players.current[key.socketId].position}    'players.current' is missing in props validation
              rotation={players.current[key.socketId].rotation}    'players.current' is missing in props validation
              getMap={getMap}
              video={videos ? videos[key.peerId] : null}
              audio={audios ? audios[key.peerId] : null}
              name={players.current[key.socketId].name}    'players.current' is missing in props validation
              povRef={povRef}
              audioIcon={audioIcon[key.socketId]}
              nodes={nodes}
              materials={materials}
              videos={videos}
              placeHolder={placeHolder}
            />
```

```jsx
/* eslint-disable react/prop-types */
import { useState, useEffect, useContext, useRef } from "react";
import {
  BsMicFill,
  BsMicMuteFill,
  BsCameraVideoFill,
  BsCameraVideoOffFill,
} from "react-icons/bs";
import { LuScreenShare } from "react-icons/lu";
import { PlayerContext } from "../../helpers/contextProvider";
import {
  getMediaStreamAudio,
  getMediaStreamVideo,
  getMediaStreamScreen,
} from "../../helpers/getMedia";
import { LoaderSync } from '../../helpers/loaders';

const BottomBar = ({
  audioStreamRef,
  videoStreamRef,
  screenStreamRef,
  setIsOwnVideo,
  setScreen,
  screen
}) => {
  const [globalMicButton, setGlobalMicButton] = useState(false);
  const [videoButton, setVideoButton] = useState(false);
  const [videoDisabled, setVideoDisabled] = useState(false);
  const [audioDisabled, setAudioDisabled] = useState(false);
  const [audioConnecting, setAudioConnecting] = useState(false);
  const [screenSharedNotification, setScreenSharedNotification] = useState(false);
  const [loading, setLoading] = useState(false);

  const { peerConn, socket, peer, playerKeys, screenShared, setScreenShared, device } = useContext(PlayerContext);

  useEffect(() => {
    const onDocumentKey = (e) => {
      if (e.ctrlKey && e.shiftKey && e.code === "KeyZ") {
        if (!audioDisabled) {
          handleAudio(false);
        }
      }
      if (e.ctrlKey && e.shiftKey && e.code === "KeyX") {
        if (!videoDisabled) {
          handleVideo();
        }
      }
    };
    document.addEventListener("keydown", onDocumentKey);
    return () => {
      document.removeEventListener("keydown", onDocumentKey);
    };
  }, [playerKeys]);    React Hook useEffect has missing dependencies: 'audioDisabled', 'handleAudio', 'handleVideo', and

  const currentDevice = useRef();

  useEffect(() => {
    if(!currentDevice.current) {
      currentDevice.current = device;
      return
    }
    if(currentDevice.current.audio != device.audio && audioStreamRef.current) {
      handleAudio(true)
      setLoading(true);
      setTimeout(() => {
        handleAudio()
        setLoading(false);
      }, 1000);
      currentDevice.current.audio = device.audio;
      return
    }
```

\

```jsx
import { useContext, useEffect, useState, useRef } from "react";
import { PlayerContext } from "../../helpers/contextProvider";
import { IoClose, IoSend } from "react-icons/io5";

const ChatBox = ({ setBoxes, boxes, setChatDot }) => {    'setBoxes' is missing in props validation
  const [message, setMessage] = useState("");
  const { setControlsAllowed, peerConn, socket, myName } =
    useContext(PlayerContext);
  const [chats, setChats] = useState([]);
  const [currChat, setCurrChat] = useState("all");    'setCurrChat' is assigned a value but never used.
  const [notifications, setNotifications] = useState({
    show: false,
    message: "",
    name: "",
    color: "",
  });

  const nameColors = useRef({});
  const randomColors = useRef([]);

  useEffect(() => {
    const getData = () => {
      if (randomColors.current.length === 0) {
        randomColors.current = ["#7dcceb", "#eb7d7d", "#ebe07d", "#e77deb"];
      }
      const data = JSON.parse(sessionStorage.getItem("all"));
      if (data) {
        if (!nameColors.current[data[0].id]) {
          nameColors.current[data[0].id] = randomColors.current.pop();
        }
        setChats(data);
        if (!boxes.chat) {    'boxes.chat' is missing in props validation
          const truncatedMessage =
            data[0].message.length > 30
              ? data[0].message.substring(0, 30) + "..."
              : data[0].message;
          setNotifications({
            show: true,
            message: truncatedMessage,
            name: data[0].name,
            color: nameColors.current[data[0].id],
          });
          setChatDot(true);
          setTimeout(() => {
            setNotifications({ show: false, message: "" });
          }, 4000);
        }
      }
    };
    document.addEventListener("chat", getData);

    return () => {
      document.removeEventListener("chat", getData);
    };
  }, [boxes.chat]);    'boxes.chat' is missing in props validation

  useEffect(() => {
    if (boxes.chat) {    'boxes.chat' is missing in props validation
      setNotifications({ show: false, message: "" });
    }
  }, [boxes.chat]);    'boxes.chat' is missing in props validation

  const onMessage = () => {
    let data = JSON.parse(sessionStorage.getItem("all"));
    let curr = { id: socket.id, name: myName, message: message };
    if (!data) {
      data = [curr];
    } else {
      if (data[0].id === socket.id) {
        curr = { ...curr, prev: true };
      }
      data.unshift(curr);
    }
    sessionStorage.setItem("all", JSON.stringify(data));
```

```jsx
const Info = () => {

  let timeoutLink;
  const copyLink = async () => {
    clearTimeout(timeoutLink);
    try {
      await navigator.clipboard.writeText(`${baseUrl}${meetingId}`);
      setCheckLink(true);
      timeoutLink = setTimeout(() => {
        setCheckLink(false);
      }, 2000);
    } catch (error) {
      alert("Failed to copy link");
    }
  }; <- #24-35 const copyLink = async () =>

  let timeoutId;
  const copyId = async () => {
    clearTimeout(timeoutId);
    try {
      await navigator.clipboard.writeText(meetingId);
      setCheckId(true);
      timeoutId = setTimeout(() => {
        setCheckId(false);
      }, 2000);
    } catch (error) {
      alert("Failed to copy link");
    }
  }; <- #38-49 const copyId = async () =>

  return (
    <>
      {!showModal ? (
        <button
          className="fixed top-2 left-1 z-10 w-[3rem] flex text-center justify-center"
          onClick={onClick}
        >
          <IoMdInformationCircleOutline size={30} color="#5c89d1" />
        </button>
      ) : (
        <div className="fixed top-1 left-1 z-10 2xl:max-w-[20%] xl:max-w-[25%] lg:max-w-[30%] md:max-w-[35%] max-w-[55%] p-2 bg-[#5c89d1] rounded flex-col text-center">
          <div className="absolute right-1 top-1">
            <button
              className="active:opacity-50 hover:bg-gray-400 rounded-full"
              onClick={() => setShowModal(false)}
            >
              <IoClose size={25} color="#fff" />
            </button>
          </div>
          <h1 className="text-lg">Virtual Meet</h1>
          <div className="grid grid-rows-4 grid-cols-2 mt-4 gap-2">
            <div className="max-w-[fit-content]">Meeting Id: </div>
            <div className="flex">
              <div className="flex bg-[#575857] p-1 rounded max-w-[fit-content]">
                <span className="rounded px-1 mr-1">{meetingId}</span>
                <button
                  className="mt-[0.05rem] active:opacity-50"
                  onClick={copyId}
                >
                  <MdOutlineContentCopy size={20} color="#fff" />
                </button>
              </div>
              <div className={`ml-1 mt-2 ${checkId ? "" : "hidden"}`}>
                <FaCheck color="#39fa73" />
              </div>
            </div>
            <div className="max-w-[fit-content]">Meeting Link:</div>
            <div className="row-span-3 break-words">
              <span className="text-sm">
                {baseUrl}
                {meetingId}
              </span>
              <div className="flex">
                <button
```

```
const PlayerModel = memo((value) => {
  const VideoMaterial = ({ src, attach }) => {
  };

  useFrame(() => {
    if (playerNameRef.current) {
      playerNameRef.current.lookAt(
        value.povRef.current.position.x,
        value.povRef.current.position.y + 0.2,
        value.povRef.current.position.z
      );
    }
  });

  const playerData = useMemo(
    () => ({
      refe: value.refe,    "refe": Unknown word.
      position: value.position,
      rotation: value.rotation,
      name: value.name,
      video: value.video,
      audio: value.audio,
      audioIcon: value.audioIcon,
      nodes: value.nodes,
      materials: value.materials,
      placeHolder: value.placeHolder,
    }),
    [value]
  );

  const PlayerName = () => {
    return (
      <group position={[0, 0.2, 0]} ref={playerNameRef} rotation={[0, 3.2, 0]}>
        <Html
          distanceFactor={1}
          occlude="blending"
          zIndexRange={[0, 0]}
          transform
        >
          <div className="max-w-52 flex items-center justify-center relative">
            <div className="flex items-center justify-center h-10 px-2 mr-1 bg-gray-900 text-white rounded select-none">
              {playerData.name}
            </div>
            <div className="flex items-center justify-center h-10 px-2 bg-gray-900 text-white rounded select-none">
              {playerData.audioIcon ? <BsMicFill /> : <BsMicMuteFill />}
            </div>
          </div>
        </Html>
      </group>
    );
  };

  let videoTracks = useRef([]);
  let audioTracks = useRef([]);
  useEffect(() => {
    if (playerData.video) {
      videoTracks.current = playerData.video.getVideoTracks();
      if (videoTracks.current.length > 0) {
        setIsVideo(true);
        videoTracks.current[0].onmute = () => {
          setIsVideo(false);
          videoTracks.current = [];
        };
      } else {
        setIsVideo(false);
      }
    }
  }, [playerData.video]);

  useEffect(() => {
    if (playerData.audio) {
      audioTracks.current = playerData.audio.getAudioTracks();
      if (audioTracks.current.length > 0) {
        const stream = playerData.audio;
```

```jsx
import { useState, useContext, useEffect } from "react";
import { useNavigate, useLocation } from "react-router-dom";
import { connectSocket } from "../helpers/socketConnection";
import { PlayerContext } from "../helpers/contextProvider";
import { LoaderSync } from "../helpers/loaders";
import { FaGithub } from "react-icons/fa";

const JoinForm = () => {
  const [meetingId, setMeetingId] = useState("");
  const [name, setName] = useState("");
  const [endedBox, setEndedBox] = useState(false);
  const [error, setError] = useState({
    meetingId: false,
    meetingMessage: "",
    name: false,
  });
  const [loading, setLoading] = useState(false);
  const { setMyName, socket, room, setIsAdmin } = useContext(PlayerContext);

  const navigate = useNavigate();
  const location = useLocation();

  useEffect(() => {
    if (location.state?.fromAdmin) {
      setEndedBox(true);
      setTimeout(() => {
        setEndedBox(false);
      }, 4000);
    }

    if (localStorage.getItem("name")) {
      setName(localStorage.getItem("name"));
    }
  }, []);    React Hook useEffect has a missing dependency: 'location.state?.fromAdmin'. Either include it or remove the dependency array.

  const handleSubmit = (e) => {
    e.preventDefault();
    // Handle form submission logic here
    if (!name) {
      setError({ ...error, name: true });
      return;
    }

    if (!meetingId) {
      setError({
        ...error,
        meetingId: true,
        meetingMessage: "Please enter the meeting ID",
      });
      return;
    }

    if (meetingId.length !== 6) {
      setError({
        ...error,
        meetingId: true,
        meetingMessage: "Please enter a valid meeting ID",
      });
      return;
    } else {
      setError({ ...error, meetingId: false, meetingMessage: "" });
    }

    setLoading(true);
    const lowercaseMeetingId = meetingId.toLowerCase(); // Convert meeting ID to lowercase
    connectSocket(lowercaseMeetingId).then((value) => {
      if (value && value.room) {
        socket.current = value.socket;
        room.current = value.room;
        setMyName([name]);
        localStorage.setItem("name", name);
        setLoading(false);
        navigate(`/${value.room}/3d`, { replace: true });
```

```
const JoinLink = () => {
  useEffect(() => {
    if (localStorage.getItem("name")) {
      setName(localStorage.getItem("name"));
    }
  }, []);    React Hook useEffect has missing dependencies: 'meetingId', 'navigate', 'room', and 'socket'. Either include them or remove the dependency array

  const handleSubmit = (e) => {
    e.preventDefault();
    // Handle form submission logic here
    if (!name) {
      setError(true);
      return;
    }
    setMyName([name]);
    localStorage.setItem("name", name);
    navigate(`/${room.current}/3d`, { replace: true });
  };

  return (
    <>
      <div className="flex items-center justify-center h-screen">
        {loading && <LoaderSync />}
        <div className="p-2 bg-gradient-to-r from-gray-500 via-slate-500 to-gray-500 rounded border-2">
          <h1 className="mb-8 text-xl text-center">Virtual Meet</h1>
          <form onSubmit={handleSubmit}>
            <div className="mb-4">
              <div>
                <label htmlFor="name" className="text-lg">
                  UserName:{" "}
                </label>
              </div>
              <input
                type="text"
                id="name"
                value={name}
                onChange={(e) => {
                  setName(e.target.value);
                  setError(false);
                }}
                maxLength={12}
                className="mr-2 px-2 mb-1 bg-[■#3b3b3b] text-white rounded"
              />
              {error && (
                <p className="text-[red] text-sm">Please enter a username</p>
              )}
            </div>
            <div className="w-full text-center mt-4">
              <button
                className="border bg-[■#5c89d1] p-1 rounded hover:scale-105 duration-300"
                type="submit"
              >
                Join Meeting
              </button>
            </div>
          </form>
        </div>
      </div>

      <div>
        <a
          href="https://github.com/fampiyush/virtual-meet"
          target="_blank"
          rel="noreferrer"
        >
          <button className="fixed top-2 right-2 flex bg-[■#5c89d1] p-1 rounded hover:scale-105 duration-300">
            <FaGithub size={25} />
            <span className="ml-2 text-lg">Github</span>
          </button>
        </a>
      </div>
    </>
  );
```

```
function MainEngine() {
    audioStreamRef,
    screenStreamRef,
    screenShareInfo,
    povRef,
    randomPositionX,
    randomPositionZ,
    setScreen,
    setVideos,
    setAudios,
    setAudioIcon,
    triggerMessagePopup,
  ); <- #79-96 const { handleIncomingCall, handlePeerConnection, dataChannel...

  // Sets up socket.io event listeners for initial user sync, disconnects, and admin termination
  const { setupSocket } = useSetupSocketEvents(
    randomPositionX,
    randomPositionZ,
    setLoading,
    triggerMessagePopup,
    dataChannel,
    players,
    playersRef,
    videos,
    videoRef
  ); <- #99-109 const { setupSocket } = useSetupSocketEvents

  return (
    <Suspense fallback={<LoaderBar />}>
      <div className="h-screen w-screen">
        {/* Render loader conditionally based on `loading` state */}
        <WithLoader isLoading={loading}>
          <>
            {/* Meeting interface for controls and notifications (audio/video/screen sharing UI) */}
            <MeetingInterface
              audioStreamRef={audioStreamRef}
              videoStreamRef={videoStreamRef}
              screenStreamRef={screenStreamRef}
              setIsOwnVideo={setIsOwnVideo}
              setScreen={setScreen}
              screen={screen}
              isOwnVideo={isOwnVideo}
              message={notification.message}
              show={notification.show}
              players={players}
              screenShared={screenShared}
              screenShareInfo={screenShareInfo}
            />
            {/* 3D scene rendering with player avatars (built with Three.js) */}
            <MeetingScene
              screen={screen}
              screenStreamRef={screenStreamRef}
              socket={socket}
              peer={peer.current}
              randomPositionX={randomPositionX.current}
              randomPositionZ={randomPositionZ.current}
              getMap={getMap}
              povRef={povRef}
              audioIcon={audioIcon}
              videos={videos}
              audios={audios}
              players={players}
              playerKeys={playerKeys}
            />
            {/* <Stats className='flex justify-end right-0 pointer-events-none z-50' /> */}
          </>
        </WithLoader>
      </div>
    </Suspense>
  ); <- #111-153 return
} <- #14-154 function MainEngine()

export default MainEngine;
```

```
import { io } from "socket.io-client";

export const connectSocket = (room) => {
  const socket = io(import.meta.env.VITE_BACKEND_URL);

  const promise = new Promise((resolve) => {
    socket.on("connect", () => {
      socket.emit("join", room);
      socket.on("joined-room", (room) => {
        resolve({ socket, room });
      });
    });
  });

  return promise;
};

export const sendModel = (socket, model) => {
  socket.emit("user-model", model);
};

export const receiveModel = (socket) => {
  socket.on("user-model", (model) => {
    console.log(model);
  });
};

export const getAllModels = (socket) => {
  socket.on("get-all-users", (models) => {
    return models;
  });
};
```

```
import { useEffect, useRef, useContext } from "react";
import { PlayerContext } from "./contextProvider";

export default function useKeyboard() {
  const keyMap = useRef({});

  const { controlsAllowed } = useContext(PlayerContext);

  useEffect(() => {
    if (controlsAllowed) {
      const onDocumentKey = (e) => {
        keyMap.current[e.code] = e.type === "keydown";
      };
      document.addEventListener("keydown", onDocumentKey);
      document.addEventListener("keyup", onDocumentKey);
      return () => {
        document.removeEventListener("keydown", onDocumentKey);
        document.removeEventListener("keyup", onDocumentKey);
      };
    }
  }, [controlsAllowed]);

  return keyMap.current;
}
```