НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»
Завідувач кафедри
_____ Наталія АУШЕВА
“___”_____2026 р.

Дипломна робота

на здобуття ступеня бакалавр

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Вебзастосунок для проведення онлайн-конференцій”

Виконала: студентка 4 курсу, групи ТР-21

Зелінська Вероніка Владиславівна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник к.т.н., доцент, Кублій Лариса Іванівна _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент доцент каф. ТАЕ, к.т.н., Олександр СІРИЙ _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль ст.викладач Ольга БЕСПАЛА _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

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

Київ - 2026

Національний технічний університет України

“Київський політехнічний інститут

імені Ігоря сікорського”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти — перший (бакалаврський)

спеціальність 122 “Комп’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА
(підпис)
“___” _____2026 р.

ЗАВДАННЯ
на дипломну роботу студенту

_____ Зелінській Вероніці Владиславівні _____
(прізвище, ім’я, по батькові)

1. Тема роботи “Вебзастосунок для проведення онлайн-конференцій”
Науковий керівник Кублій Лариса Іванівна, к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “28” травня 2026 року № 1992
2. Термін подання студентом роботи 04.06.2026
3. Вихідні дані до роботи мова програмування JavaScript, TypeScript, React, Node.js, фреймворк Express.js, Agora SDK, Supabase, середовище розробки VS Code
4. Перелік питань, які потрібно розробити _____

1) аналіз наявних платформ і технологій відеозв’язку;
2) визначення функціональних вимог до системи;
3) проєктування архітектури і розробка вебзастосунку;
4) реалізація аудіо- та відеозв’язку в реальному часі із сервісом Agora;
5) розробка зручного інтерфейсу користувача.

5. Орієнтований перелік ілюстративного матеріалу: діаграма прецедентів (ролей), скріншоти роботи користувачів з вебзастосунком.

6. Дата видачі завдання 15.09.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	02.06.2025	Виконано
2.	Аналіз методів та засобів розв’язання задачі	05.04.2026	Виконано
3.	Розробка архітектури та загальної структури системи	12.04.2026	Виконано
4.	Розробка окремих підсистем	24.04.2026	Виконано
5.	Програмна реалізація системи	10.05.2026	Виконано
6.	Оформлення пояснювальної записки	18.05.2026	Виконано
7.	Захист програмного забезпечення	12.05.2026	Виконано
8.	Передзахист	02.06.2026	Виконано
9.	Захист	18.06.2026	Виконано

Студент _____ Вероніка ЗЕЛІНСЬКА
(підпис) (прізвище та ініціали)

Керівник _____ Лариса КУБЛІЙ
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 47 сторінках, містить 12 ілюстрації, 8 таблиць, 25 джерел у переліку посилань, 2 додатки.

Метою роботи є розробка вебзастосунку для проведення онлайн-конференцій із підтримкою планування подій та інструментів взаємодії між учасниками.

У роботі проведено аналіз наявних платформ для відеоконференцій — Zoom, Google Meet і Microsoft Teams — і визначено їхні обмеження щодо спеціалізації інтерфейсу й функціональних можливостей. На основі цього аналізу сформульовано функціональні вимоги до системи та обрано технологічний стек.

Застосунок реалізовано за клієнт-серверною архітектурою. Клієнтську частину розроблено з використанням бібліотеки React, мови програмування TypeScript та бібліотеки UI-компонентів MUI і розгорнуто на хмарній платформі хостингу вебзастосунків Vercel. Серверну частину побудовано у середовищі Node.js та з використанням веб фреймворку Express.js і розгорнуто на сервісі для хостингу Heroku. Для зберігання даних та автентифікації використано BaaS-платформу (Backend-as-a-Service) Supabase, що побудована на основі PostgreSQL. Для аудіо- та відеозв’язку в реальному часі використано сервіс Agora Video Calling.

Застосунок забезпечує реєстрацію та авторизацію користувачів, створення і планування конференцій, запрошення учасників за кодом або зі списку, проведення відеоконференцій із керуванням аудіо- та відео потоком, ведення спільних нотаток, а також інтерактивне тестування з відображенням статистики для власника.

Визначено перспективні напрямки розвитку системи: реалізація чату та рефакторинг нотаток через сервіс Agora Messaging, додавання інтерактивної спільної дошки, інтеграція інструментів штучного інтелекту для автоматичного занотовування та підсумовування обговорень.

Ключові слова: технологія, застосунок, мова програмування, вебзастосунок, онлайн-конференція, відеозв’язок, React, Node.js, Supabase, Agora, WebRTC, SPA, REST API.

ANNOTATION

The thesis is completed on 47 pages and contains 12 illustrations, 8 tables, and 25 references.

The objective of this work is to develop a web application for conducting online conferences with support for event scheduling and participant interaction tools.

The work includes an analysis of existing video conferencing platforms — Zoom, Google Meet, and Microsoft Teams — identifying their limitations in terms of interface specialization and feature sets. Based on this analysis, functional requirements for the system were defined and a technology stack was selected.

The application follows a client-server architecture. The client side was built using React library, programming language TypeScript, and MUI UI-component library, and deployed on Vercel. The server side was built on Node.js with Express.js webframework and deployed on a cloud hosting service Heroku. Supabase (PostgreSQL) was used for data storage and authentication, and Agora Video Calling was integrated for real-time audio and video communication.

The application provides user registration and authentication, conference creation and scheduling, participant invitation via code or user list, video conferencing with audio and video controls, shared notes editing, and interactive testing with live statistics for the host.

Directions for further development include chat implementation and notes refactoring via Agora Messaging, an interactive shared whiteboard, and AI-powered tools for automatic note-taking and discussion summarization.

Keywords: technology, application, programming language, online-conference, video calling, web application, online conference, video communication, React, Node.js, Supabase, Agora, WebRTC, SPA, REST API.

ЗМІСТ

ВСТУП

Дистанційний формат роботи та навчання став невід'ємною частиною сучасного життя, що зумовило стрімкий розвиток засобів онлайн-комунікації. Попит на якісні інструменти для проведення відеоконференцій постійно зростає, а вимоги користувачів до таких систем стають дедалі вищими.

Наявні рішення — Zoom, Google Meet, Microsoft Teams — охоплюють широку аудиторію і забезпечують базові функції відео- та аудіозв'язку. Проте через універсальність своєї цільової аудиторії ці платформи мають обмежену гнучкість: їхній функціонал і дизайн не мають чіткої спеціалізації, що не дає можливість повною мірою задовольнити потреби конкретних категорій користувачів. Крім того, дослідники фіксують проблему "ефекту Zoom-виснаження" [1] — зниження концентрації та втоми від тривалого використання відеоконференцій, що є наслідком, зокрема, недосконалості інтерфейсних рішень.

Метою роботи є розробка вебзастосунку для проведення онлайн-конференцій із підтримкою планування подій та інструментів взаємодії між учасниками.

Для досягнення поставленої мети вирішуються такі задачі:

— аналіз наявних платформ та технологій відеозв'язку;

— визначення функціональних вимог до системи;

— Проєктування архітектури та розробка вебзастосунку;

— реалізація аудіо- та відеозв'язку в реальному часі із сервісом Agora;

— розробка зручного інтерфейсу користувача.

Результати роботи пройшли апробацію на XXIII Міжнародній науково-практичній конференції молодих вчених та студентів "Сучасні проблеми наукового забезпечення енергетики" (Додаток А). Практичну придатність розробленого застосунку підтверджено шляхом проведення реальних відеоконференцій із підключенням з різних пристроїв та облікових записів, а

також перевіркою роботи всіх реалізованих функціональних модулів у реальних умовах.

Дипломна робота складається з п’яти розділів, містить 12 рисунків, 8 таблиць, 2 додатки та 25 джерел у переліку посилань. У першому розділі обґрунтовується актуальність теми, формулюється задача розробки та визначаються функціональні вимоги системи. Другий розділ містить огляд і порівняльний аналіз наявних платформ та технологій відеозв’язку. У третьому розділі описано технологічний стек та обґрунтовано вибір засобів розробки. Четвертий розділ присвячено проєктуванню та програмній реалізації системи, а також дослідженню технологій для подальшого розвитку. П’ятий розділ описує роботу користувача з застосунком та наводить приклади інтерфейсу.

1. РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ПРОВЕДЕННЯ ОНЛАЙН-КОНФЕРЕНЦІЙ

Розробка будь-якого програмного продукту розпочинається з ґрунтовного аналізу предметної області та чіткого формулювання задачі. Без розуміння контексту, в якому функціонуватиме система, та потреб її майбутніх користувачів неможливо прийняти обґрунтовані архітектурні та проектні рішення. У цьому розділі розглядається актуальність дистанційної комунікації як явища, що стрімко трансформує сучасне робоче та навчальне середовище, формулюється задача розробки вебзастосунку та визначається перелік функціональних вимог, які є основою для подальшого Проєктування системи.

1.1. Аналіз актуальності дистанційної комунікації та онлайн-конференцій

Дедалі більше людей працюють і навчаються віддалено. Пандемія COVID-19 стала каталізатором масового переходу до дистанційного формату, проте цей процес не зупинився разом із нею — навпаки, він закріпив нові стандарти організації праці та освіти, які продовжують розвиватися. Дистанційний формат співпраці став новим стандартом для команд у різних куточках світу, а очна взаємодія поступово відходить на задній план або набуває гібридного характеру. Це стало потужним поштовхом до швидкого розвитку засобів аудіо- та відеозв’язку в реальному часі — галузі, що ще десятиліття тому була нішевою.

Попит на інструменти для проведення онлайн-конференцій продовжує зростати як у корпоративному, так і в освітньому середовищі. Разом з тим зростають і очікування користувачів: від сучасних платформ очікують не лише стабільного відеозв’язку, а й гнучкого набору інструментів для взаємодії, зручного планування подій та інтерфейсу, що не перевантажує увагу. Водночас наявні рішення не завжди задовольняють ці вимоги — їхній функціонал орієнтований на якомога ширшу аудиторію, що неминуче призводить до компромісів у зручності та

спеціалізації. Це свідчить про актуальність розробки спеціалізованих вебзастосунків, орієнтованих на конкретні сценарії використання та цільові групи користувачів.

1.2 Формулювання задачі створення вебзастосунку

Метою роботи є розробка вебзастосунку для проведення онлайн-конференцій із підтримкою планування подій та інструментів взаємодії між учасниками.

Для досягнення мети необхідно вирішити такі задачі:

— проаналізувати наявні платформи та технології відеозв’язку;

— визначити функціональні вимоги до системи;

— спроектувати архітектуру застосунку та обрати технологічний стек;

— реалізувати клієнтську та серверну частини застосунку;

— інтегрувати сервіс аудіо- та відеозв’язку в реальному часі;

— розробити зручний інтерфейс користувача.

Розробка вебзастосунку передбачає реалізацію повного циклу роботи з онлайн-конференціями — від реєстрації користувача та планування події до проведення відеоконференції з набором інтерактивних інструментів.

Особливу увагу приділено зручності інтерфейсу та мінімізації когнітивного навантаження на учасників, що є безпосередньою відповіддю на виявлені недоліки наявних рішень. Результатом роботи є функціональний вебзастосунок, розгорнутий у хмарному середовищі та готовий до використання в реальних умовах.

1.3 Визначення функціональних вимог до системи

На основі аналізу предметної області та поставленої задачі визначено такі функціональні вимоги до системи:

— реєстрація та авторизація користувачів із підтвердженням електронної пошти;

— створення та планування конференцій із зазначенням дати й часу проведення;

— запрошення учасників через унікальний код або додавання зареєстрованих користувачів до списку запрошених;

— відображення запланованих конференцій на домашній сторінці в хронологічному порядку;

— аудіо- та відеозв’язок у реальному часі з можливістю керування мікрофоном і камерою;

— ведення спільних нотаток під час конференції.

Визначені вимоги формують мінімально достатній функціональний базис системи, що забезпечує повноцінний досвід проведення онлайн-конференції — від організаційного етапу до безпосередньої взаємодії учасників. На основі цього переліку здійснюється проєктування архітектури системи та вибір технологічного стеку, що детально описано в наступних розділах.

2. АНАЛІЗ СУЧАСНИХ СИСТЕМ ОНЛАЙН-КОНФЕРЕНЦІЙ ТА ТЕХНОЛОГІЙ ВІДЕОЗВ'ЯЗКУ

Перед початком розробки власного рішення важливо дослідити наявних ринок інструментів та зрозуміти, які підходи вже реалізовані, а які потреби користувачів залишаються незадоволеними. Такий аналіз дає можливість уникнути повторення чужих помилок, виявити прогалини в наявних рішеннях та сформувати обґрунтовану концепцію власного продукту. У цьому розділі розглядаються найпоширеніші платформи для проведення онлайн-конференцій, аналізуються їхні ключові характеристики та визначаються обмеження, які стали передумовою для розробки спеціалізованого застосунку.

2.1 Огляд платформ для проведення онлайн-конференцій

Серед найпоширеніших платформ для проведення онлайн-конференцій виділяють Zoom, Google Meet і Microsoft Teams. Кожна з них має свої особливості та цільову аудиторію.

Платформа Zoom — одна з найпопулярніших платформ для відеоконференцій. Підтримує великі зустрічі, вебінари, має розвинену екосистему інтеграцій та інструменти для навчальних заходів. Проте більшість спеціалізованих функцій є платними, а безкоштовний план обмежує тривалість групових дзвінків до 40 хвилин, що суттєво звужує його практичне застосування.

Платформа Google Meet — вебзастосунок від Google, інтегрований із сервісами Google Workspace. Не потребує встановлення додаткового програмного забезпечення, доступний безпосередньо в браузері та орієнтований на простоту використання. Проте саме через надмірну універсальність платформа позбавлена будь-якої спеціалізації — вона однаково підходить для всіх і не є оптимальною ні для кого.

Застосунок Microsoft Teams — платформа, тісно інтегрована з екосистемою Microsoft 365. Поєднує функції відеозв’язку, чату та спільної роботи з документами, переважно орієнтована на корпоративне середовище. Попри широкий функціонал, платформа відзначається громіздкою структурою навігації та перевантаженим інтерфейсом, що збільшує когнітивне навантаження та зорову втому при тривалому використанні.

2.2 Порівняльний аналіз наявних рішень

Попри широке розповсюдження, усі три платформи мають спільний недолік — вони розраховані на максимально широку аудиторію, через що їхній функціонал і дизайн не мають чіткої спеціалізації. Це обмежує їхню придатність для специфічних сценаріїв використання, зокрема для проведення навчальних заходів або тематичних конференцій.

Порівняльний аналіз основних характеристик платформ наведено далі в таблиці 2.1.

Таблиця 2.1 — Порівняльний аналіз платформ для онлайн-конференцій

Характеристика	Zoom	Google Meet	Microsoft Teams
Робота в браузері	Частково	Так	Частково
Планування подій	Так	Так	Так
Спільні нотатки	Ні	Ні	Так
Запрошення за кодом	Так	Так	Так
Спеціалізація інтерфейсу	Ні	Ні	Ні
Безкоштовний доступ	Обмежений	Так	Обмежений

Таким чином, можна зробити висновок про недоліки й потенціал наявних платформ і використати ці дані для покращення продукту.

2.3 Огляд технологій відеозв’язку та обґрунтування вибору Agora

В основі більшості сучасних платформ для відеоконференцій лежить технологія WebRTC [2] — відкритий стандарт для передачі аудіо, відео та довільних даних між браузерами в режимі реального часу без встановлення додаткового програмного забезпечення. WebRTC підтримується всіма сучасними браузерами та є основою як для самостійних реалізацій, так і для хмарних сервісів відеозв’язку.

Технологія WebRTC розв’язує фундаментальну задачу встановлення peer-to-peer з’єднання між пристроями, що знаходяться за різними мережевими екранами та NAT. Для цього використовуються протоколи ICE, STUN та TURN: STUN-сервер допомагає пристрою визначити свою публічну IP-адресу, а TURN-сервер виступає ретранслятором у випадках, коли пряме з’єднання неможливе. За різними оцінками, до 15–20% з’єднань потребують ретрансляції через TURN, що робить його обов’язковим компонентом будь-якого продакшн-рішення на базі WebRTC.

Самостійна реалізація відеозв’язку на базі WebRTC потребує розгортання та підтримки кількох серверних компонентів. Кожен опублікований застосунок вимагає щонайменше чотирьох інфраструктурних шарів: сервер сигналізації, STUN/TURN-сервери, медіасервер для багатокористувацьких дзвінків та інфраструктуру для запису сесій. При цьому самостійна реалізація додає значну складність, ще один сервер для підтримки та ще одну точку відмови. Налаштування медіасервера, STUN і TURN, написання сервера сигналізації та тестування NAT-traversal за різних мережевих умов є зусиллям мінімум на кілька тижнів [3-5].

Як альтернативу самостійній реалізації було розглянуто використання хмарних CPaaS-платформ (Communications Platform as a Service), зокрема Agora та Twilio. Серед них обрано Agora з таких міркувань:

— сервіс Agora використовує власну мережу SD-RTN (Software Defined Real-time Network) замість чистого WebRTC, що забезпечує наскрізну затримку менше 200 мс у стандартних мережевих умовах [6];

— за умов нестабільного з’єднання з втратою пакетів 25% та варіацією часу затримки 600 мс сервіс Agora стабільно забезпечує вищу частоту кадрів порівняно з альтернативними рішеннями [7];

— документація та SDK (Software Development Kit) Agora вирізняються зручністю для розробників: детальними прикладами коду, навчальними матеріалами та чіткою структурою документації, що скорочує час на інтеграцію [8];

— сервіс Agora має розвинену екосистему розширень із підтримкою AI-шумоподавлення, користувацьких фонів та UI Kit для швидкого старту на різних платформах [9].

Важливим фактором також є модель ціноутворення Agora: платформа надає безкоштовний місячний ліміт, достатній для розробки та тестування застосунку, а подальша тарифікація базується на фактичному обсязі використання. Це робить Agora привабливим вибором не лише з технічної, а й з економічної точки зору для проєктів на ранніх етапах розробки.

Таким чином, використання Agora дає можливість зосередитися на розробці функціоналу застосунку, делегуючи всю інфраструктуру медіа передачі хмарній платформі.

3. ЗАСОБИ Й ТЕХНОЛОГІЇ РОЗРОБКИ ВЕБЗАСТОСУНКУ

Вибір технологічного стеку є одним із найважливіших рішень на етапі Проєктування програмної системи. Він визначає не лише технічні можливості майбутнього продукту, а й швидкість розробки, зручність супроводу коду та потенціал для масштабування. Кожна технологія обирається виходячи з конкретних вимог системи, досвіду команди розробників та зрілості інструменту. У цьому розділі описано та обґрунтовано вибір засобів розробки клієнтської та серверної частин застосунку, бази даних, сервісу аутентифікації та платформи для реалізації аудіо та відеозв'язку в реальному часі.

3.1 Використання бібліотеки React для клієнтської частини

Клієнтську частину вебзастосунку реалізовано з використанням бібліотеки React [10] — однієї з найпопулярніших JavaScript-бібліотек для побудови користувацьких інтерфейсів. React використовує компонентний підхід, що дає можливість розбивати інтерфейс на незалежні повторно використовувані частини та ефективно керувати станом застосунку. Компонентна модель добре поєднується з модульною архітектурою проекту — кожен feature-модуль складається з набору React-компонентів із чітко визначеною відповідальністю.

Серед альтернатив розглядалися Vue та Angular. Vue є простішим у початковому освоєнні, проте має меншу екосистему та спільноту. Angular пропонує більш строгу архітектуру, однак є надлишково складним для застосунку такого масштабу та має значно вищий поріг входу. React займає оптимальну позицію між цими підходами: він достатньо гнучкий для швидкої розробки та водночас має зрілу екосистему бібліотек, великий обсяг документації та активну спільноту.

Мову TypeScript обрано як основну для написання клієнтського коду. TypeScript є надбудовою JavaScript із підтримкою статичної типізації, що

підвищує надійність коду, спрощує його підтримку та дає можливість виявляти помилки на етапі компіляції, а не під час виконання.

Для реалізації користувацького інтерфейсу використано бібліотеку компонентів Material UI (MUI) [11] — реалізацію системи дизайну Material Design для React. Вона надає готовий набір компонентів, що легко налаштовуються. Це значно що прискорює розробку інтерфейсу та забезпечує його візуальну узгодженість. Використання MUI як основи дає можливість зосередитися на логіці застосунку, не витрачаючи час на реалізацію базових елементів інтерфейсу з нуля.

Клієнтську частину розгорнуто на платформі Vercel [12]. Vercel спеціалізується саме на хостингу фронтенд-застосунків і забезпечує автоматичне розгортання при кожному push до репозиторію, глобальну CDN-мережу для швидкої доставки статичних ресурсів та зручний попередній перегляд для кожної гілки розробки. Усе це суттєво прискорює цикл розробки та тестування.

3.2 Використання платформи Node.js для серверної частини

Серверну частину вебзастосунку побудовано на платформі Node.js [13] — середовищі виконання JavaScript поза браузером, що базується на двигуні V8. Використання JavaScript як єдиної мови для клієнтської та серверної частин дає можливість повторно використовувати типи та допоміжні функції, спрощує переключення між частинами кодової бази та знижує загальну складність проекту. Node.js добре підходить для побудови серверних застосунків завдяки асинхронній моделі виконання та великій екосистемі пакетів.

Для організації серверної логіки використано фреймворк Express [14] — мінімалістичний веб фреймворк для Node.js, який забезпечує зручну маршрутизацію HTTP-запитів та підключення проміжного програмного забезпечення. Express не нав'язує жорсткої структури проекту, що дає можливість організувати код відповідно до потреб конкретної системи.

Серверну частину розгорнуто на платформі Heroku [15] — хмарній PaaS-платформі, що підтримує автоматичне розгортання з Git-репозиторію та не

потребує налаштування серверної інфраструктури. Heroku забезпечує швидкий старт без необхідності адміністрування сервера, що є суттєвою перевагою на етапі активної розробки.

3.3 Використання бази даних і засобів зберігання інформації

Для зберігання даних використано Supabase [16] — відкриту платформу, що надає хмарну реляційну базу даних на основі PostgreSQL, а також вбудовані засоби аутентифікації користувачів. Вибір Supabase обумовлено кількома факторами:

— вбудована автентифікація з підтримкою підтвердження електронної пошти, керування сесіями та JWT-токенами дає можливість уникнути необхідності реалізовувати цей шар самостійно;

— в основі Supabase лежить PostgreSQL — реляційна СУБД з підтримкою складних запитів, зовнішніх ключів та транзакцій, що забезпечує надійність та цілісність даних;

— платформа надає зручний клієнтський SDK для взаємодії з базою даних безпосередньо з клієнтської та серверної частин застосунку, а також інтуїтивний веб інтерфейс для керування даними;

— безкоштовний план Supabase є достатнім для розробки та тестування застосунку без додаткових витрат.

Серед альтернатив розглядався Firebase від Google — популярна NoSQL-платформа з вбудованою аутентифікацією та синхронізацією даних у реальному часі. Проте реляційна модель даних з чіткими зв’язками між таблицями є більш природною для предметної області цього застосунку, ніж документо-орієнтований підхід Firebase. Крім того, PostgreSQL забезпечує суворішу перевірку цілісності даних через зовнішні ключі та обмеження, що є важливим для коректної роботи зв’язків між конференціями, учасниками та тестами.

Аутентифікацію користувачів реалізовано засобами Supabase Auth із підтвердженням електронної пошти через одноразовий код.

3.4 Інтеграція сервісу Agora

Для реалізації аудіо та відеозв’язку в реальному часі інтегровано сервіс Agora [17] — хмарну PaaS-платформу для передачі мультимедійних даних у реальному часі. Сервіс Agora базується на технології WebRTC [2] і забезпечує стабільну передачу аудіо та відео, підтримку багатокористувацьких сесій та низьку затримку зв’язку.

У застосунку використовується Agora Video Calling — режим, у якому всі учасники конференції є рівноправними й можуть одночасно передавати власний аудіо та відеопотік. Інтеграцію реалізовано через офіційний Agora Web SDK на стороні клієнтської частини, а серверна частина відповідає за генерацію токенів доступу для підключення до каналів Agora.

4. ПРОЄКТУВАННЯ І ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ОНЛАЙН-КОНФЕРЕНЦІЙ

Проєктування програмної системи передбачає прийняття низки взаємопов'язаних рішень — від вибору загальної архітектури до деталей реалізації окремих модулів. Якість цих рішень безпосередньо впливає на надійність, розширюваність та зручність супроводу готового продукту. У цьому розділі описано архітектуру вебзастосунку, структуру клієнтської та серверної частин, деталі реалізації ключових функціональних модулів — автентифікації, керування конференціями, аудіо та відеозв’язку, спільних нотаток і тестування, — а також структуру бази даних та зв’язки між її таблицями.

4.1 Архітектура вебзастосунку

Вебзастосунок побудовано за клієнт-серверною архітектурою з чітким розмежуванням відповідальності між частинами системи. Клієнтська частина відповідає за відображення інтерфейсу користувача та керування станом застосунку. Серверна частина реалізує бізнес-логіку, обробляє запити клієнта та взаємодіє із зовнішніми сервісами. Зберігання даних забезпечується хмарною базою даних Supabase. Аудіо- та відеозв’язок реалізовано через сервіс Agora, до якого клієнт підключається безпосередньо після отримання токена від серверної частини.

Взаємодія між частинами системи відбувається за такою схемою. Клієнт надсилає HTTP-запити до серверної частини через REST API. Сервер, отримавши запит, дістає з заголовку запиту JWT-токен автентифікації та передає його на перевірку Supabase Auth.

Якщо підтверджено авторизований статус користувача, сервер виконує бізнес-логіку та звертається до бази даних Supabase або зовнішніх сервісів як-от Agora. Результат повертається клієнту у форматі JSON.

Окремим каналом взаємодії є аудіо- та відеозв’язок. Для підключення до конференції клієнт спочатку запитує у сервера токен доступу Agora, після чого встановлює пряме з’єднання з хмарною інфраструктурою Agora. Медіа потоки між учасниками передаються безпосередньо через Agora, не навантажуючи серверну частину застосунку.

Така архітектура забезпечує чітке розділення відповідальності: сервер не обробляє медіадані і зосереджений виключно на бізнес-логіці, а клієнт взаємодіє з Agora напряму після одноразової авторизації через сервер. Це знижує навантаження на серверну частину та спрощує масштабування системи.

4.2 Проєктування структури клієнтської частини

Клієнтська частина організована як односторінковий застосунок (SPA) з клієнтською маршрутизацією. Структуру маршрутів наведено в таблиці 4.1.

Таблиця 4.1 — Маршрути клієнтської частини

Шлях	Сторінка	Призначення
/	—	Перенаправлення на /welcome
/welcome	WelcomePage	Стартова сторінка
/sign-up	SignUpPage	Реєстрація
/home	HomePage	Домашня сторінка користувача
/conference/create	CreateConferencePage	Створення конференції
/conference/:id	ConferencePage	Проведення конференції
/user/:userId	UserProfilePage	Профіль користувача
*	ErrorPage	Сторінка помилки

Кодову базу побудовано за модульною архітектурою: кожна функціональна область застосунку (реєстрація, домашня сторінка, конференція тощо) інкапсульована у власному feature-модулі з власними компонентами та логікою.

Route-компоненти у шарі pages/ є тонкими обгортками, які лише підключають відповідний модуль, не містячи власної логіки. Такий підхід має низку переваг [18]:

— пришвидшує процес розробки, спрощує підтримку та орієнтацію в коді в подальшій роботі;

— знижує зв’язність між частинами застосунку, дає можливість перевикористати значно більшу частину коду;

— має кращі показники Core Web Vitals і SEO, що напряму впливає на позиції в Google. Модульний код легше оптимізувати, а сторінки швидше проходять перевірки PageSpeed.

Спільні компоненти та обгортки над елементами бібліотеки MUI винесено в окремі шари, що забезпечує візуальну та стилістичну узгодженість інтерфейсу в межах усього застосунку.

Для керування глобальним станом використано бібліотеку Zustand [19] — легковісну альтернативу Redux [20, 21], яка не потребує шаблонного коду та добре інтегрується з React-хуками, що для даного проекту є найбільш підходящою опціє. Стан організовано у два логічно відокремлені стори: один відповідає за дані профілю користувача, інший — за стан конференцій і тестування.

Захист маршрутів реалізовано на рівні API-запитів: axios-інтерцептор перехоплює відповіді з кодами 401 і 403, після чого виконує оновлення токена або перенаправляє користувача на сторінку входу.

4.3 Проєктування серверної частини

Серверну частину реалізовано як REST API на основі Node.js та Express із класичним поділом на шари: точка входу, middleware, маршрути та бібліотечні модулі.

Точка входу index.js відповідає за реєстрацію глобальних middleware та підключення маршрутів. Глобально застосовуються вбудований парсер тіла

express.json() та модифікований CORS-middleware, що обмежує список дозволених джерел через змінну середовища ALLOWED_ORIGINS.

Маршрути згруповано за ресурсами у чотири файли: users.js, conferences.js, conferenceTests.js та agora.js. Порядок їхнього підключення має значення з точки зору безпеки: публічні маршрути /users та /agora реєструються до застосування middleware автентифікації, захищені маршрути /conferences, /conference-tests та /profile — після нього. Проміжне програмне забезпечення аутентифікації (auth.js) перевіряє наявність заголовка x-user-id у вхідному запиті та передає ідентифікатор користувача до обробників маршрутів.

Допоміжні модулі винесено до директорії lib: supabase.js містить ініціалізований клієнт для взаємодії з базою даних, generateAgoraToken.js відповідає за генерацію токенів доступу для підключення до каналів Agora, respond.js — уніфікована функція формування HTTP-відповідей.

Запити для роботи з конференціями згруповано у шість запитів, що охоплюють повний цикл керування конференцією — від створення до видалення і керування учасниками. Перелік наведено в таблиці 4.2.

Таблиця 4.2 — Запити для роботи з конференціями

Метод	URL	Опис
GET	/conferences	Отримати список конференцій
GET	/conferences?code=&connectionId=	Приєднатися за кодом
POST	/conferences	Створити конференцію
PATCH	/conferences?id=	Оновити конференцію
DELETE	/conferences?id=	Видалити конференцію
DELETE	/conferences/participants?conferenceId=	Відхилити участь

Запити для роботи з тестами забезпечують отримання, створення та оновлення тесту в межах конференції, враховуючи те, що тільки один тест може відповідати конкретній конференції. Перелік наведено в таблиці 4.3.

Таблиця 4.3 — Запити для роботи з тестами

Метод	URL	Опис
GET	/conference-tests?conferenceId=	Отримати тест конференції
POST	/conference-tests	Створити тест
PATCH	/conference-tests	Надіслати відповідь

Запити для роботи з користувачами охоплюють операції з профілем поточного користувача, пошук та керування обліковими записами. Перелік наведено в таблиці 4.4.

Таблиця 4.4 — Запити для роботи з користувачами

Метод	URL	Опис
GET	/profile	Профіль поточного користувача
GET	/users	Список усіх користувачів
GET	/users?id=	Користувач за ID
GET	/users/search?query=	Пошук користувачів
POST	/users	Створити користувача
PATCH	/users	Оновити поточного користувача
DELETE	/users	Видалити поточного користувача

Наведені запити охоплюють усі необхідні операції для повноцінної роботи з даними користувачів. Групування запитів за ресурсами — конференції, тести, користувачі — забезпечує зрозумілу та передбачувану структуру API, що спрощує його підтримку та розширення в майбутньому. Кожен запит виконує єдину чітко визначену функцію, що відповідає принципам Проєктування REST API та знижує ймовірність помилок при інтеграції клієнтської частини із серверною.

4.4 Реалізація системи автентифікації та керування користувачами

Автентифікацію реалізовано засобами Supabase Auth. Під час реєстрації користувач вводить електронну пошту та пароль, після чого на пошту надсилається одноразовий код підтвердження. Після підтвердження створюється запис у таблиці users із базовими даними профілю.

Авторизація наступних запитів відбувається через JWT-токен, який Supabase видає після успішного входу. Токен передається у заголовку запитів до серверної частини, де перевіряється перед виконанням бізнес-логіки.

Користувач може переглядати та редагувати власний профіль: ім’я, займенники, роль, часовий пояс, біографію та аватар.

4.5 Реалізація механізму створення та планування конференцій

Створення конференції відбувається через форму на сторінці /conference/create. Користувач може вказати назву, порядок денний, дату й час проведення та тривалість. Після створення конференція зберігається в таблиці conferences, а її творець автоматично додається до таблиці conference_participants із позначкою is_host = true.

Запрошення учасників реалізовано двома способами: через унікальний код конференції, який інші користувачі можуть ввести для приєднання, або через додавання зареєстрованих користувачів до списку запрошених. В обох випадках учасник додається до таблиці conference_participants, і конференція відображається на його домашній сторінці в хронологічному порядку.

4.6. Реалізація аудіо- та відеозв’язку в реальному часі

Підключення до конференції реалізовано як єдиний узгоджений flow, що поєднує отримання даних конференції з авторизацією в Agora. Коли користувач

переходить на сторінку /conference/:id, хук useConferenceController викликає запит GET /conferences?code=&connectionId=, передаючи ідентифікатор каналу з URL та унікальний connectionId користувача з профілю. Сервер повертає об'єкт конференції разом із згенерованим токеном доступу [22] Agora та списком учасників. Таким чином отримання даних конференції та авторизація в Agora відбуваються за один мережевий запит.

Після отримання відповіді хук useJoin з Agora Web SDK ініціює підключення до відеоканалу. Підключення активується лише після того, як токен, connectionId та ідентифікатор каналу одночасно доступні — це забезпечується прапорцем isReady, який передається другим аргументом до useJoin. Такий підхід унеможливлює передчасне підключення до каналу з неповними даними.

Паралельно з підключенням ініціалізуються локальні медіа треки через хуки useLocalMicrophoneTrack та useLocalCameraTrack. Після їхньої готовності хук usePublish публікує обидва треки у канал одночасно. Вмикання та вимикання мікрофона і камери під час конференції реалізовано через метод setEnabled відповідного треку, що дає можливість призупиняти передачу потоку без його повного закриття.

Відображення відеопотоків інших учасників реалізовано у компоненті VideoGrid. Хук useRemoteUsers повертає актуальний список підключених учасників, кожен з яких відображається через компонент RemoteUser з Agora SDK, що самостійно підписується на відповідні аудіо та відео потоки. Зіставлення між Agora-користувачем та даними профілю відбувається через порівняння поля uid з Agora та поля connectionId з таблиці users — ці значення є ідентичними, що забезпечує коректне відображення імені та аватара учасника поряд із його відео потоком.

4.7 Реалізація спільних нотаток

Спільні нотатки реалізовано у вигляді поля agenda в таблиці conferences. Поле підтримує збереження тексту з HTML-розміткою, що дає можливість

власнику дзвінка структурувати порядок денний із форматуванням — заголовками, списками тощо. Нотатки створюються та редагуються власником до початку конференції через форму на сторінці /conference/create або під час редагування події.

Під час конференції нотатки доступні всім учасникам у режимі перегляду. Текст рендериться як HTML безпосередньо на клієнті.

Такий підхід дає можливість учасникам зосередитися на перебігу конференції, маючи постійний доступ до порядку денного в межах одного вікна браузера — без необхідності відкривати додаткові вкладки чи зовнішні документи.

4.8 Реалізація тестування під час конференції

Функціонал тестування реалізовано як інтерактивний інструмент для перевірки розуміння матеріалу безпосередньо під час конференції. Власник створює питання з варіантами відповідей через окрему форму — тест містить текстове питання, від двох до десяти варіантів відповідей та одну правильну. Після збереження тест записується до таблиці conference_tests і стає доступним усім учасникам.

Виявлення нового тесту на стороні клієнта відбувається через регулярні запити до ендпоінту GET /conference-tests?conferenceId= з інтервалом у 5 секунд. Щойно тест з’являється у відповіді, він відображається учаснику у вигляді питання з варіантами для вибору.

Учасник обирає один варіант і надсилає відповідь через PATCH /conference-tests. Сервер оновлює лічильники total_answers та total_correct у таблиці conference_tests та повертає результат. Клієнт підсвічує обраний варіант зеленим або червоним залежно від його правильності, що забезпечує миттєвий зворотний зв’язок для учасника.

Власник спостерігає за статистикою відповідей також через polling з інтервалом у 5 секунд — актуальні значення total_answers та total_correct відображаються в реальному часі. За потреби власник може замінити поточний

тест новим, після чого учасники отримають оновлене питання на наступному циклі опитування.

4.9 Дослідження технологій для подальшого розвитку системи

У процесі розробки застосунку було досліджено ряд технологій, впровадження яких є перспективним для подальшого розвитку системи, проте виходить за межі поточного етапу реалізації.

4.9.1 Сервіс Agora Messaging для реалізації чату

Для реалізації текстового чату між учасниками конференції було досліджено Agora Messaging SDK [23]. Ця технологія є природним розширенням вже інтегрованого у застосунок Agora Video Calling — обидва інструменти працюють у межах єдиної екосистеми та використовують спільну інфраструктуру каналів. Це суттєво спрощує інтеграцію порівняно зі стороннім рішенням і дає можливість уникнути необхідності підтримувати окремий сервіс обміну повідомленнями. Дослідження охоплювало вивчення документації SDK, моделі каналів та цінової політики сервісу.

4.9.2. Протокол WebSocket та Agora Signaling для синхронізації в реальному часі

Поточний механізм доставки тестів та нотаток учасникам базується на регулярному polling-запитах з інтервалом у 5 секунд. Як альтернативу було розглянуто використання WebSocket-з’єднань або Agora Signaling — протоколів, що забезпечують двосторонній зв’язок між сервером і клієнтом у реальному часі без необхідності ініціювати запит з боку клієнта. Такий підхід дав би можливість знизити кількість мережевих запитів та забезпечити миттєву доставку оновлень. Проте оскільки синхронізація контенту не є основним фокусом роботи та поточна реалізація повністю задовольняє функціональні вимоги системи, впровадження цих технологій відкладено до наступного етапу розробки.

4.9.3 Agora AI та інтерактивна спільна дошка

Окремо було досліджено розширені інструменти платформи Agora — AI-модулі [24] та Whiteboard SDK [25]. AI-інструменти надають можливості автоматичного занотовування перебігу конференції та генерації підсумків обговорення, що є особливо актуальним для навчальних заходів. Сервіс Interactive Whiteboard забезпечує реалізацію інтерактивної спільної дошки з підтримкою малювання, додавання фігур та текстових анотацій у реальному часі, що відкриває значний потенціал для проведення більш різнопланових занять, як от уроки з математики з живим процесом вирішення. Обидва інструменти розглядаються як пріоритетні напрямки розвитку системи та потребують окремого етапу дослідження й інтеграції.

4.10 Структура бази даних і зв’язки між таблицями

База даних містить чотири таблиці. Структуру таблиць наведено далі в таблицях 4.5–4.8.

Реляційна модель даних обрана з огляду на чітку структуру предметної області: між користувачами, конференціями та тестами існують формалізовані зв’язки, які природно виражаються через зовнішні ключі та обмеження цілісності. PostgreSQL як основа Supabase забезпечує суворе дотримання цих обмежень на рівні бази даних, що унеможливлює появу некоректних або осиротілих записів. Такий підхід знижує ймовірність помилок у бізнес-логіці та спрощує написання запитів завдяки можливості використання JOIN-операцій між таблицями.

Таблиця users є центральною сутністю системи — вона зберігає облікові дані та профільну інформацію кожного зареєстрованого користувача. Структуру таблиці наведено в таблиці 4.5.

Таблиця 4.5 — Структура таблиці users

Поле	Тип	Опис

id	uuid	Первинний ключ
email	text	Електронна пошта
name	text	Ім’я користувача
pronouns	text	Займенники
role	text	Роль (студент, викладач тощо)
timezone	text	Часовий пояс
bio	text	Біографія
avatar_url	text	Посилання на аватар
connection_id	integer	Agora connection ID
created_at	timestamp	Дата створення

Таблиця conferences зберігає всі дані про конференцію — від назви та нотаток (порядку денного) до часових параметрів і коду для приєднання. Поле creator_id пов’язує конференцію з її автором у таблиці users. Структуру таблиці наведено в таблиці 4.6.

Таблиця conference_participants реалізує зв’язок багато-до-багатьох між конференціями та користувачами. Вона використовується для визначення списку користувачів конференції та їх ролей. Складений первинний ключ із полів conference_id та user_id унеможливлює дублювання записів (користувач не може двічі бути доданий як учасник однієї конференції), а поле is_host визначає роль учасника в межах конкретної конференції. Структуру таблиці наведено в таблиці 4.7.

Таблиця 4.6 — Структура таблиці conferences

Поле	Тип	Опис
id	uuid	Первинний ключ
name	text	Назва конференції
agenda	text	Порядок денний / нотатки
date	timestamp	Дата та час проведення
duration	integer	Тривалість у хвилинах

code	text	Код для приєднання
creator_id	uuid	FK — users.id
created_at	timestamp	Дата створення
ended_at	timestamp	Дата завершення

Таблиця 4.7 — Структура таблиці conference_participants

Поле	Тип	Опис
conference_id	uuid	FK — conferences.id
user_id	uuid	FK — users.id
is_host	boolean	Ознака власника

Таблиця conference_tests пов’язана з конференцією відношенням один-до-одного — кожна конференція може мати щонайбільше один активний тест. Замість збереження нових тестів як окремих сутностей, поточний тест замінюється новим у разі створення тесту власником. Поле question зберігає структуру питання у форматі JSON, який буде повністю парситись на клієнтській стороні, а лічильники total_answers та total_correct оновлюються в реальному часі в міру надходження відповідей від учасників для забезпечення огляду статистики для учасника, що створив конференцію та тести в ній — власника. Таким чином реалізовано швидкий зворотній зв’язок між викладачем та студентом. Структуру таблиці наведено в таблиці 4.8.

Таблиця 4.8 — Структура таблиці conference_tests

Поле	Тип	Опис
conference_id	uuid	PK / FK — conferences.id
question	json	Питання: заголовок, варіанти, правильна відповідь
total_answers	integer	Загальна кількість відповідей
total_correct	integer	Кількість правильних відповідей

Зв’язки між таблицями: таблиця conferences пов’язана з таблицею users через поле creator_id. Таблиця conference_participants пов’язує conferences та users відношенням багато-до-багатьох і має складений первинний ключ (conference_id, user_id). Таблиця conference_tests пов’язана з conferences відношенням один-до-одного через поле conference_id.

4.11 Діаграма прецедентів

Діаграму прецедентів наведено на рисунку 4.1. Система передбачає двох акторів: учасника та власника. Учасник може:

— реєструватися й авторизуватися;

— переглядати список конференцій;

— приєднуватися до конференції за кодом;

— брати участь у відеоконференції;

— переглядати спільні нотатки;

— відповідати на тести.

Власник, крім перерахованого, також може:

— створювати та видаляти конференції

— керувати списком учасників;

— створювати тести й переглядати їхню статистику.

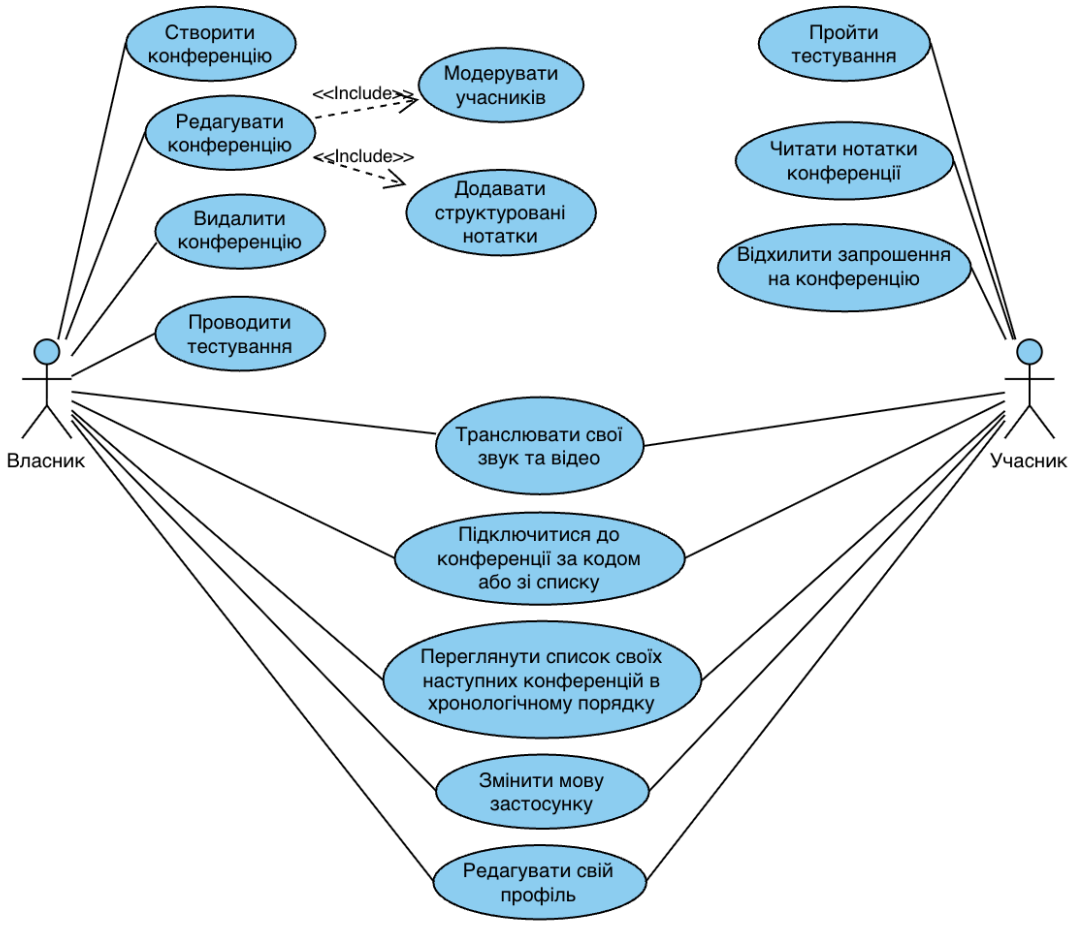


Рисунок 4.1. — Діаграма прецедентів

Таким чином, увесь базовий функціонал для навчального заняття у форматі онлайн покрито, хоч потенціал для додаткових функцій лишається широким.

5. РОБОТА КОРИСТУВАЧА З ВЕБЗАСТОСУНКОМ

Функціональність програмного продукту найповніше розкривається через призму досвіду кінцевого користувача. Навіть технічно досконала система втрачає свою цінність, якщо взаємодія з нею є незручною або незрозумілою. У цьому розділі розглядається робота користувача з розробленим застосунком: від системних вимог та процесу реєстрації до використання інструментів взаємодії під час конференції. Наводяться приклади інтерфейсу, описуються основні сценарії використання та окреслюються напрямки подальшого розвитку і вдосконалення системи.

5.1 Системні вимоги для роботи вебзастосунку

Оскільки застосунок є вебзастосунком, що працює в браузері, він не потребує встановлення додаткового програмного забезпечення. Для коректної роботи необхідні сучасний браузер з підтримкою WebRTC (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari останніх версій), стабільне інтернет з’єднання, а також мікрофон і камера у випадку, якщо користувач бажає брати активну участь у відеоконференції. Якщо ж він підключається виключно як слухач, то остання вимога не є обов’язковою.

5.2. Реєстрація та авторизація користувача

Для початку роботи з застосунком користувач переходить на стартову сторінку /welcome, де може обрати реєстрацію або вхід. Сторінка виконана у двоколонковому макеті: ліва частина містить форму автентифікації, права — декоративну панель з візуальним прикладом роботи застосунку. Для входу в наявний обліковий запис користувач вводить електронну пошту та пароль і натискає кнопку “Увійти”. Повторний вхід не потребує додаткового

підтвердження — достатньо коректних облікових даних. Інтерфейс наведено на рисунку 5.1.

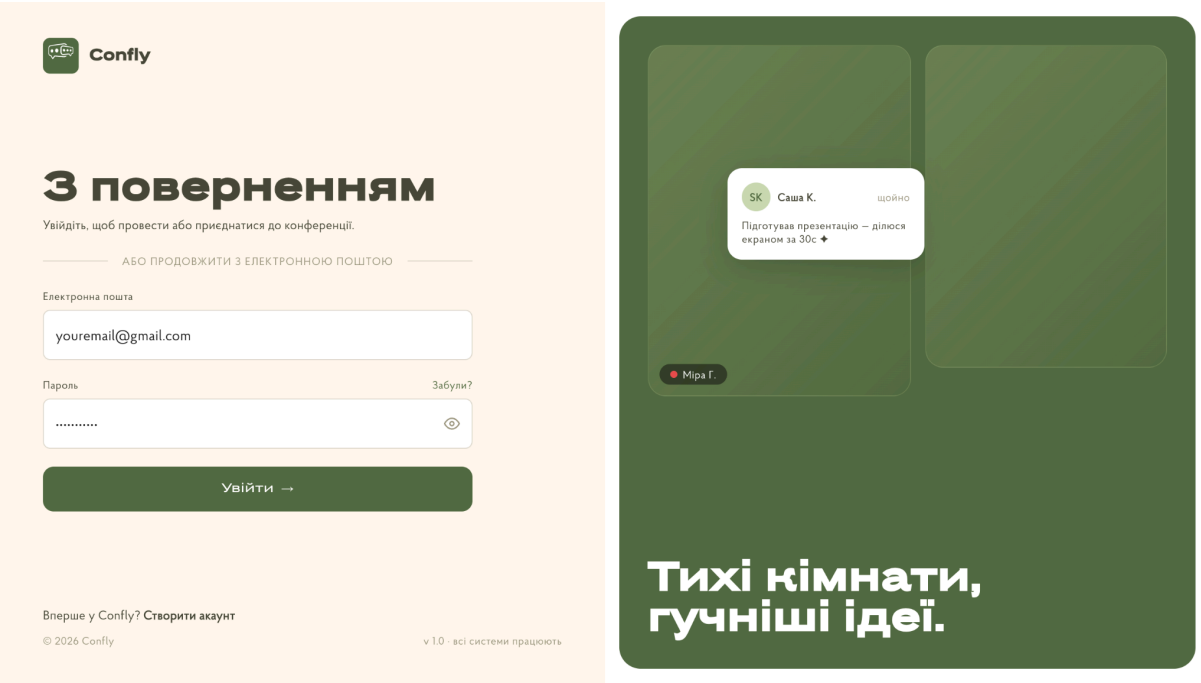


Рисунок 5.1 — Сторінка входу

Реєстрація нового облікового запису відбувається на сторінці /sign-up. Цей етап відображено на рисунку 5.2.

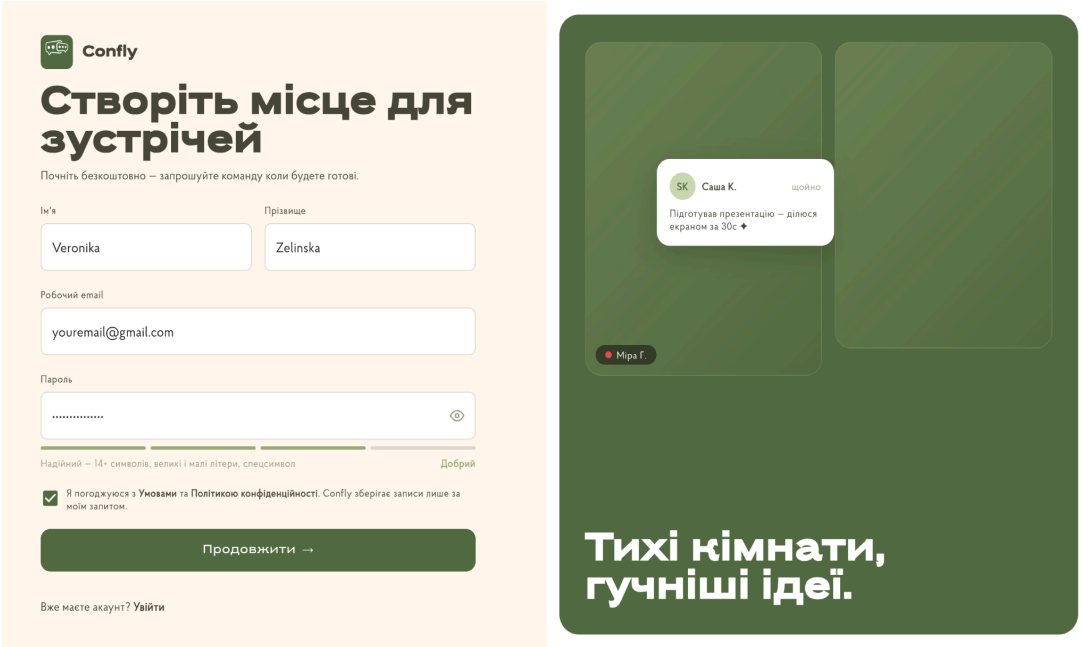


Рисунок 5.2 — Сторінка реєстрації

Користувач заповнює форму з полями імені, прізвища, електронної пошти та пароля. Індикатор надійності пароля в реальному часі підказує користувачу, чи відповідає введений пароль вимогам безпеки.

Після натискання кнопки “Продовжити” на вказану електронну пошту надсилається 8-значний код підтвердження з терміном життя — година. Це дає користувачу більш ніж достатньо часу на реєстрацію і зменшує кількість потенційних перенадсилань коду, що в результаті збільшувало б бюджет проекту.

На наступному екрані користувач вводить отриманий код. Інтерфейс даного етапу видно на рисунку 5.3. Після успішного підтвердження обліковий запис активується і користувач автоматично отримує доступ до застосунку.

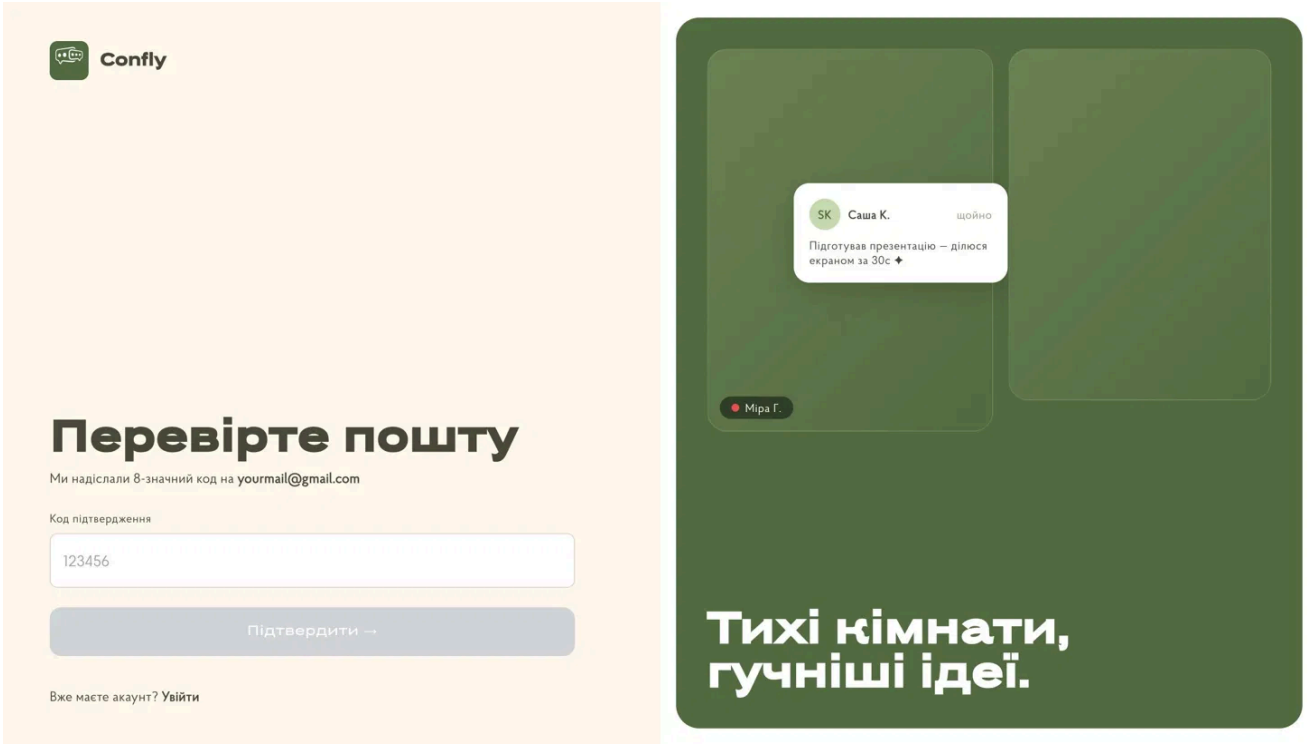


Рисунок 5.3 — Підтвердження електронної пошти

Такий підхід до реєстрації — з обов’язковим підтвердженням електронної пошти через одноразовий код — забезпечує базовий рівень верифікації

користувачів та унеможливлює створення облікових записів на неіснуючі адреси. Це є важливим як з точки зору безпеки системи, так і з точки зору якості даних: кожен зареєстрований користувач у таблиці users гарантовано має підтверджену та дійсну електронну пошту, що є передумовою для коректної роботи механізму запрошень до конференцій.

5.3 Створення й планування онлайн-конференції

Після авторизації користувач потрапляє на домашню сторінку /home, звідки, як видно на рисунку 5.4, в один клік можна дістатися до всього найважливішого функціоналу.

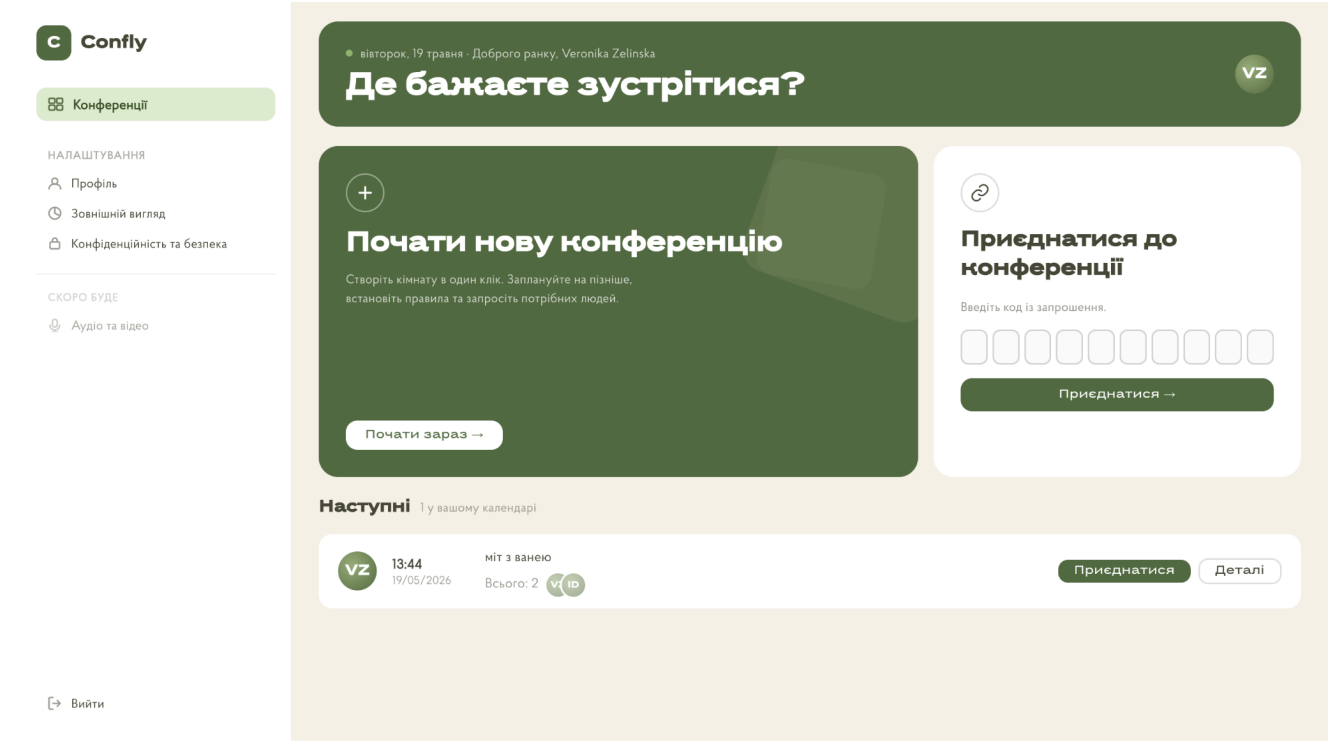


Рисунок 5.4. — Головна (домашня) сторінка

У верхньому лівому кутку знаходиться картка для створення конференцій, що переносить користувача на сторінку, продемонстровану на рисунку 5.5. Для створення нової конференції користувач переходить на сторінку /conference/create, де заповнює форму: вказує назву, порядок денний, дату й час проведення та

тривалість. Після збереження конференція з’являється на домашній сторінці та стає доступною для запрошених учасників.

У верхньому правому кутку на рисунку 5.4 — картка приєднання до конференції за 10-значним кодом, що складається з латинських літер та цифр. Варто знову зауважити, що користувача не допустить навіть за правильним кодом, якщо його не було запрошено власником на дану конференцію.

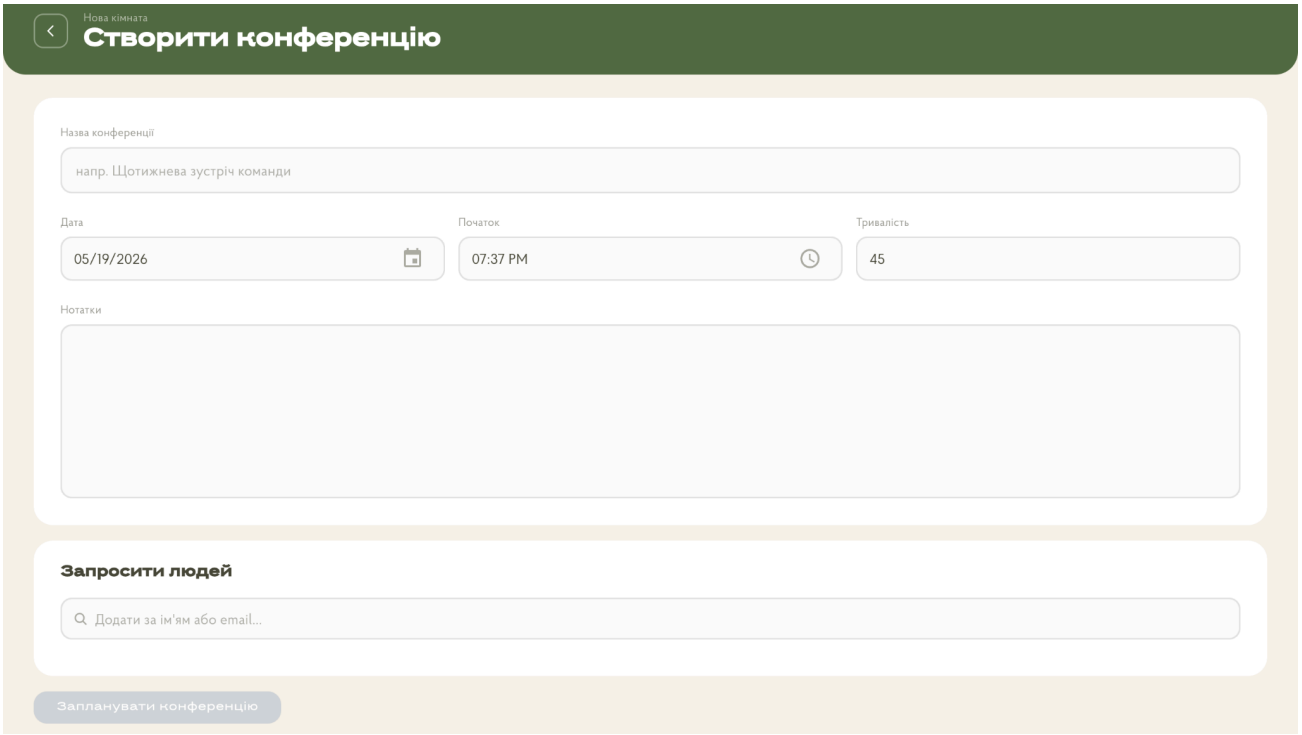


Рисунок 5.5. — Сторінка створення конференції

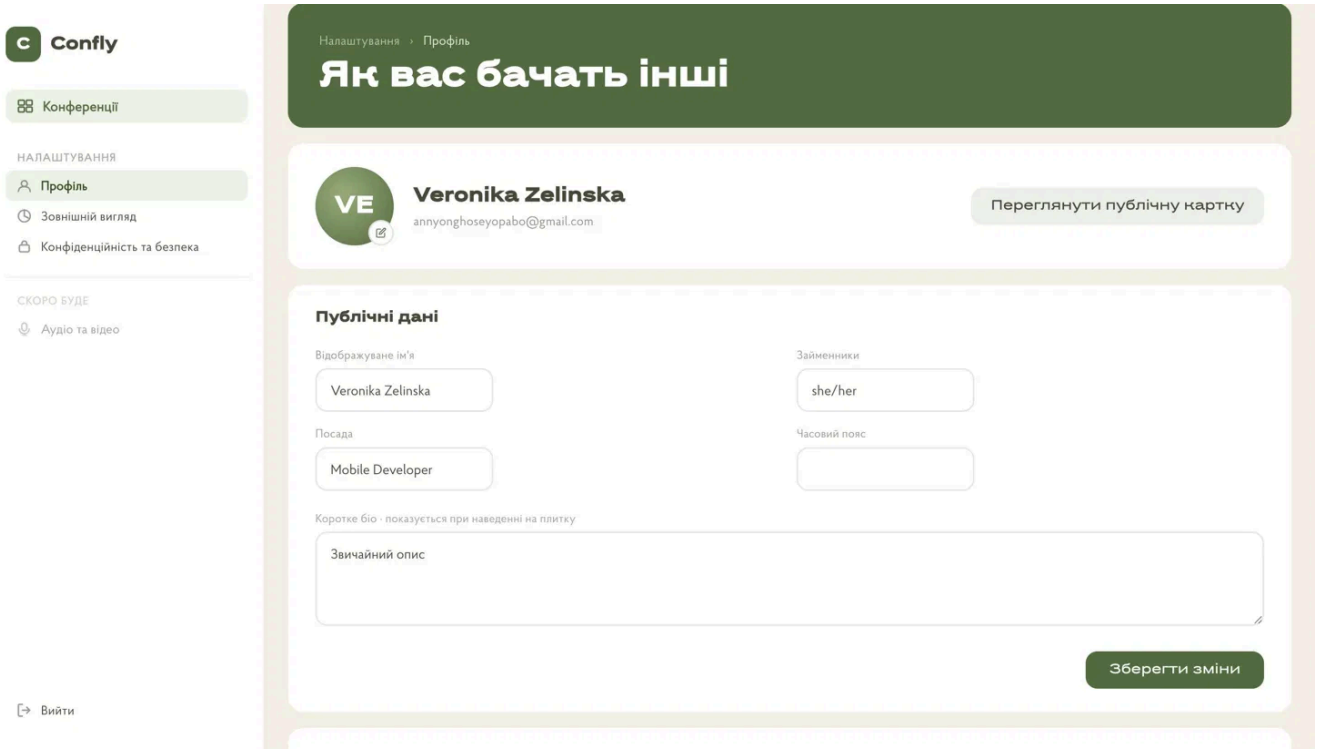
Знизу на рисунку 5.4 відображається список “Наступні”. В ньому в хронологічному порядку відображаються усі сьогоднішні та майбутні заплановані дзвінки. Така система фільтрації надає можливість приєднатися до дзвінка, на який користувач запізнився, чи повернутися групою в дзвінок, що відбувся сьогодні раніше, щоб продовжити дискусію тієї ж теми — функціонал, важливий для динамічних процесів і проактивних груп.

На самій картці конференції можна побачити список учасників і перейти на їхні профілі, дату та час, а також виконати дії по відношенню до самого дзвінка — відхилити чи скасувати його, редагувати, приєднатися (в залежності від ролі).

Така організація домашньої сторінки відображає ключовий принцип проєктування інтерфейсу застосунку — мінімізацію кількості дій, необхідних для досягнення результату. Найважливіші операції доступні безпосередньо зі списку конференцій без необхідності переходити на окремі сторінки, що скорочує час взаємодії та знижує когнітивне навантаження на користувача. Цей підхід є свідомою відповіддю на виявлену у ході аналізу надмірну складність навігації у наявних платформах, що призводить до втоми при тривалому використанні.

5.4 Керування профілем користувача

Після реєстрації користувач може налаштувати власний профіль на сторінці /user/:userId. Сторінка профілю, що відображена на рисунку 5.6, організована як



панель налаштувань із бічною навігацією, що містить розділи профілю, зовнішнього вигляду та конфіденційності.

Рисунок 5.6 — Сторінка профілю користувача

У розділі профілю користувач може заповнити публічні дані: відображуване ім’я, займенники, посаду, часовий пояс та коротке біо, яке відображається при наведенні на плитку учасника під час конференції. Аватар налаштовується безпосередньо на сторінці профілю. Усі зміни зберігаються після натискання кнопки “Зберегти зміни”. Кнопка “Переглянути публічну картку” дає можливість побачити, як профіль виглядає для інших учасників.

5.5 Підключення до конференції за запрошенням або кодом

Для зручності користувачів, було додано можливість доєднатися до конференції двома способами.

Перший — власник додає користувача до списку запрошених під час створення або редагування конференції, після чого вона автоматично з’являється на домашній сторінці запрошеного.

Другий — користувач вводить унікальний код конференції, отриманий від власника, і приєднується до неї самостійно. В обох випадках після підключення користувач потрапляє на сторінку конференції /conference/:id.

Загальний вигляд онлайн-конференції продемонстровано на рисунку 5.7.

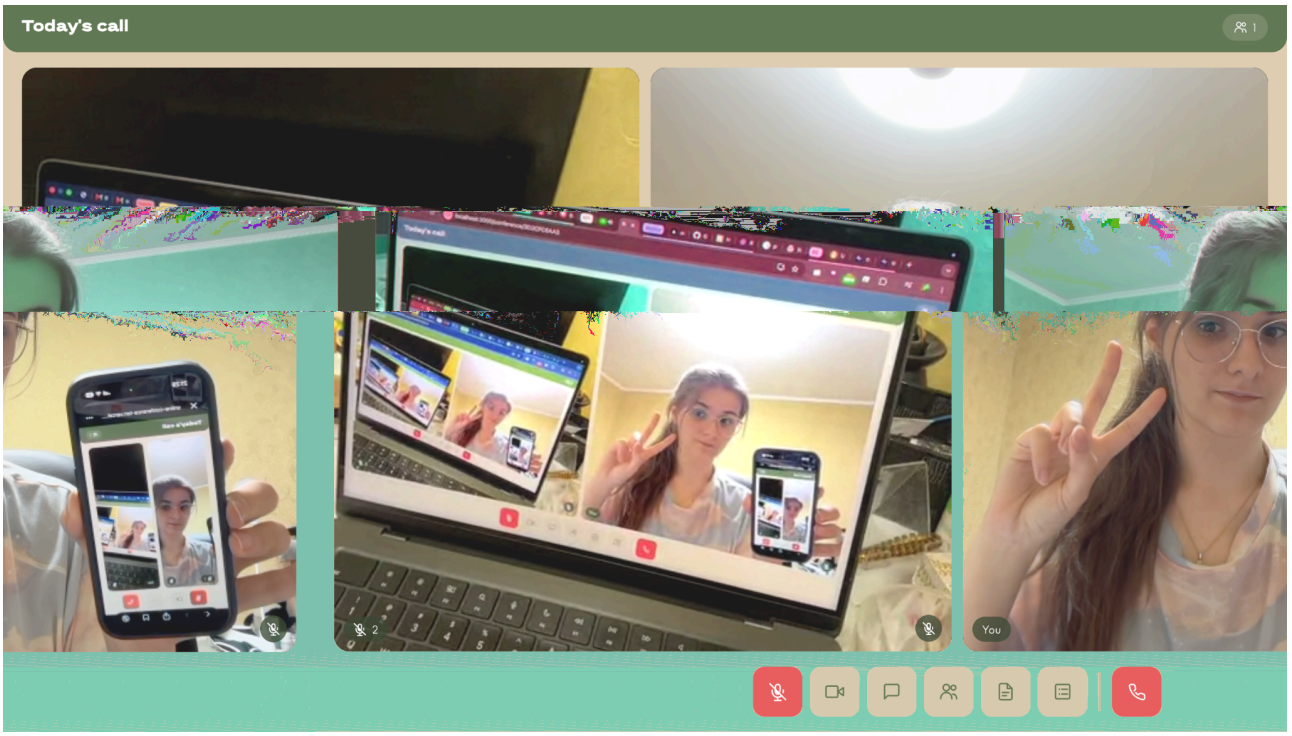


Рисунок 5.7. — Сторінка конференції (підключення виконано з ноутбуку та телефону на різних облікових записах)

Варто зауважити, що для конференцій з більшою кількістю учасників, розташування плиток відео буде змінено в цілях відображення якнайбільшої кількості людей без викривлення чи сильного обрізання їхнього відео.

5.6 Використання інструментів взаємодії під час конференції

Під час конференції учасникам доступні такі інструменти:

— керування мікрофоном і камерою — користувач може вмикати та вимикати власний аудіо- та відеопотік;

— спільні нотатки — усі учасники можуть переглядати нотатки у зручному для читання форматі (з HTML-форматуванням, якщо власником було додано відповідний текст). Приклад відображено на рисунку 5.8.

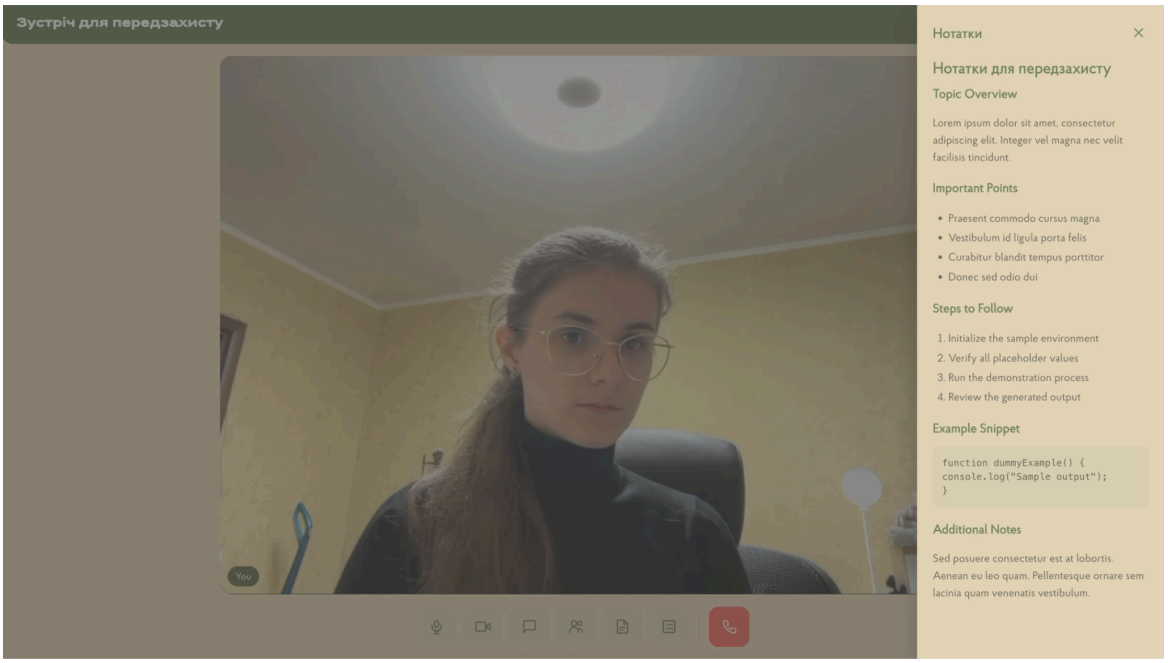


Рисунок 5.8 — Вигляд нотаток під час онлайн-конференції

— тестування — власник може створити питання з варіантами відповідей, яке одразу стає доступним усім учасникам. Після відповіді учасників власник бачить статистику: загальну кількість відповідей та кількість правильних. Власник може замінити поточне питання новим у будь-який момент. Приклад тестування з обох перспектив відображено на рисунках 5.9-5.11.

На рисунку 5.9 відображено процес створення власником тесту: заповнення самого питання та варіантів відповіді, вибір правильної та публікація запитання. Форма створення тесту відкривається у бічній панелі, не перекриваючи відеопотік — власник може одночасно стежити за конференцією та формулювати питання. Варіанти відповідей додаються динамічно, а правильна позначається натисканням безпосередньо на відповідний варіант. Після натискання кнопки “Опублікувати тест” питання стає доступним усім учасникам на наступному циклі polling-запиту.

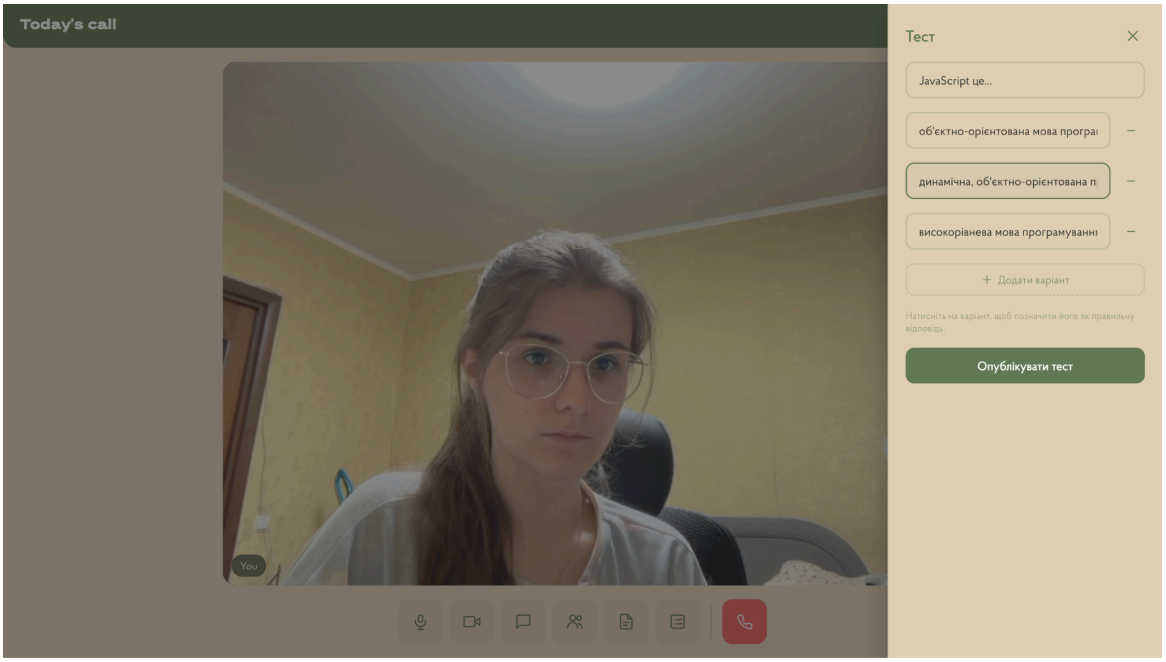


Рисунок 5.9 — Створення тесту власником

На рисунку 5.10 продемонстровано вигляд щойно створеного тесту учаснику конференції, що зайшов з іншого пристрою (телефону). Інтерфейс тесту адаптований для мобільного екрану — варіанти відповідей відображаються у вигляді чітко розділених карток, зручних для натискання на сенсорному екрані.

Після вибору варіанту та надсилання відповіді учасник отримує миттєвий зворотний зв’язок: правильна відповідь підсвічується зеленим кольором, а під варіантами з’являється текстове підтвердження результату. Це дає можливість учаснику одразу оцінити власний рівень розуміння матеріалу без необхідності чекати на коментар власника.

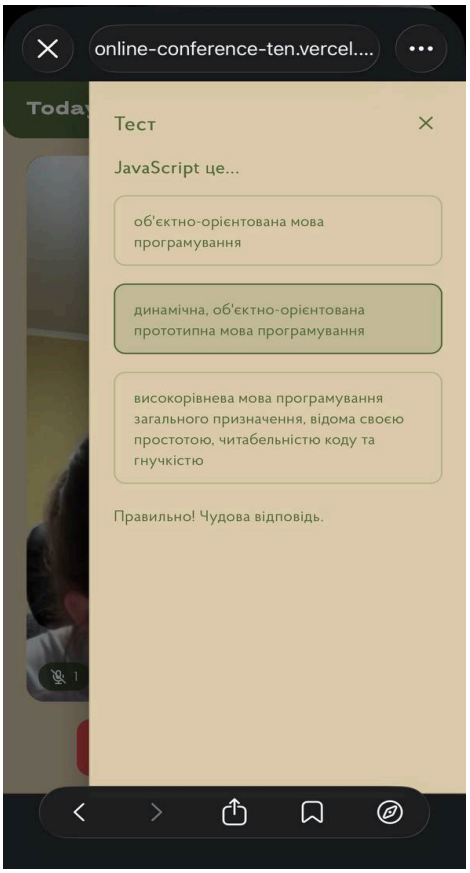


Рисунок 5.10 — Вигляд тесту з точки зору учасника після надання відповіді (учасник підключений з телефону)

Також на рисунку 5.10 продемонстровано частину мобільного адаптиву — в межах проєкту було забезпечено якісний досвід використання вебзастосунку як користувачами персональних ком’ютерів, так і тим, хто заходить з телефону.

Як видно на рисунку 5.11, паралельно з тим, як учасник відповідає на тест, хост бачить оновлення статистики в реальному часі. Панель тесту в інтерфейсі хоста відрізняється від учасницької: замість можливості обрати відповідь відображаються лічильники загальної кількості відповідей та кількості

правильних. Правильний варіант підсвічується зеленим кольором безпосередньо в списку — це дозволяє хосту одночасно бачити і саме питання, і поточний результат, не перемикаючись між режимами.

Подібний підхід до тестування не перериває потік заняття, навпаки — підтримує високий рівень залученості усіх присутніх та спонукає до підняття питань у разі їх виникнення.

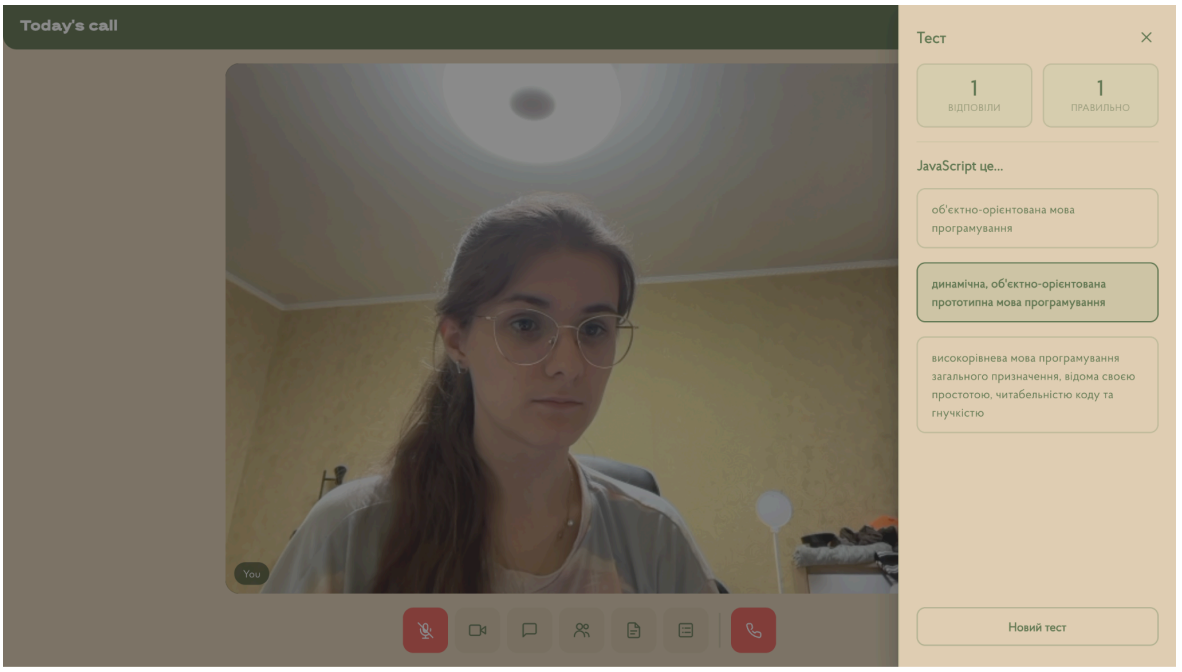


Рисунок 5.11 — Вигляд тесту з точки зору власника після отримання відповіді

На рисунку 5.11 видно результат проведеного тестування — було зібрано відповіді присутніх учасників і відображено статистику для власника. У наведеному прикладі один учасник надав відповідь, яка виявилася правильною — лічильники показують “1 відповіли / 1 правильно”. Після перегляду результатів власник може натиснути кнопку “Новий тест”, щоб створити та опублікувати наступне питання. Це дає можливість проводити кілька тематичних запитань поспіль у межах однієї конференції, перевіряючи розуміння матеріалу поетапно.

ВИСНОВКИ

На основі аналізу наявних платформ для проведення онлайн-конференцій — Zoom, Google Meet та Microsoft Teams — визначено їхній спільний недолік: орієнтацію на максимально широку аудиторію без чіткої спеціалізації інтерфейсу й функціоналу. Проведено огляд технологій відеозв'язку, зокрема WebRTC і хмарних CPaaS-платформ, на основі якого обґрунтовано вибір сервісу Agora Video Calling як засобу реалізації аудіо та відеозв'язку в реальному часі.

У результаті дослідження програмних засобів обґрунтовано використання React, TypeScript і MUI для клієнтської частини, Node.js і Express.js для серверної частини, Supabase як платформи для зберігання даних та автентифікації, а також Vercel і Heroku як середовищ розгортання клієнтської і серверної частин відповідно.

Розроблено вебзастосунок для проведення онлайн-конференцій з клієнт-серверною архітектурою, що забезпечує повний цикл роботи з конференціями: реєстрацію та авторизацію користувачів із підтвердженням електронної пошти, створення й планування конференцій, запрошення учасників за кодом або зі списку зареєстрованих користувачів, проведення відеоконференцій із керуванням аудіо- та відео потоком, перегляд спільних нотаток, а також інтерактивне тестування учасників із відображенням статистики для власника.

Результати роботи пройшли апробацію на XXIII Міжнародній науково-практичній конференції молодих вчених та студентів "Сучасні проблеми наукового забезпечення енергетики" (Додаток А). Коректність роботи застосунку підтверджено практичним тестуванням у реальних умовах — проведено сеанси відеоконференцій із підключенням з різних пристроїв та облікових записів.

Визначено перспективні напрямки подальшого розвитку системи: реалізація текстового чату і рефакторинг механізму синхронізації нотаток і тестів із використанням Agora Messaging або WebSocket, додавання інтерактивної спільної дошки на основі Agora Whiteboard SDK, а також інтеграція інструментів штучного інтелекту для автоматичного занотовування й підсумовування обговорень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Four causes of ‘Zoom fatigue’ and their causes. URL: https://news.stanford.edu/stories/2021/02/four-causes-zoom-fatigue-solutions
2. What is WebRTC: A Beginner’s Guide. URL: https://medium.com/@nakrani/what-is-webrtc-6e9cc2a51222
3. Best Tech Stack for WebRTC App Development: вебсайт. URL: https://sheerbit.com/best-tech-stack-for-webrtc-app-development/
4. How Much Does It Really Cost to Build and Run a WebRTC Application?: вебсайт. URL: https://webrtc.ventures/2025/10/how-much-does-it-really-cost-to-build-and-run-a-webrtc-application/
5. Why System Integrators Choose VideoSDK Over Open-Source WebRTC: вебсайт. URL: https://www.videosdk.live/blog/system-integrators-videosdk-vs-opensource
6. VideoSDK vs Twilio vs Agora: Which is Best for BFSI Video KYC?: вебсайт. URL: https://www.videosdk.live/blog/videosdk-vs-twilio-vs-agora-bfsi-video-kyc
7. Testing Agora vs. Twilio for Multi-Party Web Video Calls: вебсайт. URL: https://www.agora.io/en/blog/testing-agora-vs-twilio-for-multi-party-web-video-calls/
8. twilio vs agora: вебсайт. URL: https://www.stackshare.io/stackups/twilio-vs-agora
9. Top 5 Alternatives to Twilio Programmable Video: вебсайт. URL: https://getstream.io/blog/twilio-video-alternatives/
10. React Documentation: вебсайт. URL: https://react.dev/
11. Material UI Documentation: вебсайт. URL: https://mui.com/material-ui
12. Vercel Documentation: вебсайт. URL: https://vercel.com/docs
13. Node.js Documentation : вебсайт. URL: https://nodejs.org/en
14. Express.js Documentation: вебсайт. URL: https://expressjs.com/en/5x/guide/routing/

15. Heroku Documentation for Node.js: вебсайт. URL: https://www.heroku.com/nodejs/
16. Supabase Documentation: вебсайт. URL: https://supabase.com/docs
17. Agora Real-Time Voice and Video Engagement : вебсайт. URL: https://www.agora.io/en/
18. Модульна архітектура: як побудувати сайт, що росте разом із вашим бізнесом: вебсайт. URL: https://cases.media/en/article/modulna-arkhitektura-yak-pobuduvati-sait-sho-roste-razom-iz-vashim-biznesom?srsltid=AfmBOophdhDWhYeukoF4l6CVEdnax57Ex_wuieRC3RP1QUBAIqMTdNI3
19. Zustand documentation. Updating State page: вебсайт. URL: https://zustand.docs.pmnd.rs/learn/guides/updating-state
20. State Management in React: Comparing Redux Toolkit vs. Zustand: вебсайт. URL: https://dev.to/hamzakhan/state-management-in-react-comparing-redux-toolkit-vs-zustand-3no
21. Comparison: How Zustand stacks up against similar libraries: вебсайт. URL: https://zustand.docs.pmnd.rs/learn/getting-started/comparison
22. Building a Token Server for Agora Applications using Node.js: вебсайт. URL: https://www.agora.io/en/blog/how-to-build-a-token-server-for-agora-applications-using-nodejs/
23. Enable and configure Chat. URL: https://docs.agora.io/en/agora-chat/get-started/enable
24. Conversational AI Engine (Agora Documentation): вебсайт. URL: https://docs.agora.io/en/conversational-ai/overview/product-overview
25. Interactive Whiteboard (Agora Documentation): вебсайт. URL: https://docs.agora.io/en/interactive-whiteboard/overview/product-overview

ДОДАТОК А

Апробація

Матеріали XXIII Міжнародної науково-практичної конференції молодих вчених і студентів “Сучасні проблеми наукового забезпечення енергетики” див. на наступній сторінці
До 40-річчя Чорнобильської катастрофи

“Вебзастосунок для проведення онлайн-конференцій”

УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ІАТЕ_ЦТЕ_ТР_15

Аркушів 6

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

СУЧАСНІ ПРОБЛЕМИ НАУКОВОГО ЗАБЕЗПЕЧЕННЯ ЕНЕРГЕТИКИ:

БЕЗПЕКА, СТІЙКІСТЬ, ІТ ТА ЕКОЛОГІЧНИЙ МОНІТОРИНГ

XXIII Міжнародна науково-практична конференція
молодих вчених та студентів

До 40-річчя Чорнобильської катастрофи

17–22 квітня 2026 року, м. Київ

Київ-2026

УДК 004.77:005.745

[1] Бакалаврант 4 курсу Зелінська В. В.
[1] Доц., к.т.н. Кублій Л. І.
https://scholar.google.com.ua/citations?hl=en&user=rXA1eMIAAAAJ

[1] КПІ ім. Ігоря Сікорського

ВЕБЗАСТОСУНОК ДЛЯ ПРОВЕДЕННЯ ОНЛАЙН-КОНФЕРЕНЦІЙ

Постановка проблеми та її актуальність. Дедалі більше людей працюють і навчаються віддалено. Дистанційний формат співпраці став новим стандартом, а очна взаємодія відходить на задній план. Це стало поштовхом до швидкого розвитку значної кількості технологій, зокрема засобів аудіо- та відеозв'язку в реальному часі. Наявні рішення не завжди задовольняють вимоги користувачів щодо гнучкості функціональних можливостей, інтеграції додаткових інструментів для взаємодії, зручності організації запланованих подій.

Аналіз останніх досліджень. Сучасні підходи до проведення онлайн-конференцій зосереджуються на забезпеченні якісного аудіо- та відеозв'язку в реальному часі за допомогою вебтехнологій. Одним із ключових стандартів є WebRTC (Real-Time Communication) – відкрита технологія, яка дає змогу браузерам здійснювати передавання відео й аудіо з мінімальною затримкою, гарантує безпечну й масштабовану комунікацію [1]. Завдяки цьому технологія WebRTC лежить в основі багатьох сучасних систем відеоконференцій. Також набули популярності хмарні сервіси на базі WebRTC. Прикладом є PaaS-продукти (Platform-as-a-Service), такі як платформа для автоматизації вбудовування в застосунки відео- та аудіозв'язку в реальному часі Agora [2] чи хмарна комунікаційна платформа Twilio [3], а також готові SaaS-продукти (Software-as-a-Service), наприклад, Zoom. Використання цих рішень спрощує впровадження відеозв'язку, проте забезпечення стабільності й масштабованості залишається предметом активних досліджень. Зокрема, в роботах аналізується вплив навантаження на якість досвіду користувачів [4] – розробляються підходи до стрес-тестування WebRTC-систем і визначаються метрики для запобігання погіршенню якості при великій кількості учасників [5].

Популярними для проведення онлайн-конференцій є такі платформи, як Zoom, Google Meet і Microsoft Teams. Вони забезпечують базові функції відео- та аудіозв'язку, проте через різноманітність їхньої цільової аудиторії вони спрощені, їхній дизайн і функціональні можливості не мають чіткої спеціалізації, через що не повністю покривають потреби онлайн-конференцій і навчальних заходів.

Формулювання мети. Метою дослідження є розробка веб-застосунку для проведення онлайн-конференцій із можливістю планування подій і взаємодії між учасниками.

Основна частина. Розроблений вебзастосунок має клієнт-серверну архітектуру з чітким розмежуванням функцій. Клієнтська частина відповідає за інтерфейс користувача і керування станом конференції. Вона забезпечує підключення учасників до конференцій, керування мікрофоном і камерою, а також доступ до інструментів взаємодії. Серверна частина реалізує бізнес-логіку застосунку, обробку запитів клієнта, створення й планування конференцій, взаємодію з сервісом аудіо- й відеозв'язку, забезпечуючи централізоване керування процесами і можливість масштабування системи.

Реалізовано механізм створення онлайн-конференцій із прив'язкою до конкретного часу проведення і можливістю надання доступу учасникам за унікальним посиланням або ідентифікатором конференції.

Функціональні можливості аудіо- та відеозв'язку в реальному часі реалізовано за допомогою інтеграції зі стороннім сервісом Agora [2] з використанням формату трансляції в реальному часі Interactive Streaming [6], що забезпечує стабільну передачу мультимедійних даних і підтримку багатокористувацьких сесій.

Вебзастосунок призначений для проведення онлайн-конференцій у режимі реального часу й забезпечує створення, планування й проведення відеозустрічей. Застосунок підтримує передачу аудіо та відео, керування мікрофоном і камерою, підняття руки, можливість додавання спільних нотаток, обмін повідомленнями.

Інтерфейс користувача спроєктований із врахуванням вимог зручності та інтуїтивності. Структура елементів керування дає можливість швидко підключатися до конференцій, керувати налаштуваннями і взаємодіяти з іншими учасниками без необхідності додаткового навчання. До уваги взято недоліки платформ-конкурентів, такі як «ефект Zoom-виснаження» [7].

Клієнтську частину веб-застосунку реалізовано з використанням бібліотеки React [8], що загалом є популярним рішенням для фронтенд-частини застосунку. Серверну частину, яка відповідає за обробку запитів і підключення сторонніх сервісів, побудовано на платформі Node.js [9]. Для реалізації аудіо- та відеозв'язку інтегровано сервіс Agora [2], який забезпечує стабільну передачу мультимедійних даних і гладку масштабованість при низькій ціні.

Висновки. Створений вебзастосунок для проведення онлайн-конференцій забезпечує стабільний відео- та аудіозв'язок, підтримує планування подій і надає додаткові інструменти для взаємодії учасників, зручний у використанні. Розроблене рішення може бути основою для подальшого розвитку й розширення шляхом додавання нових функцій таких, як опитування в реальному часі, користувацькі фони. Перспективними напрямами також є реалізація інтерактивної дошки для спільної роботи учасників у реальному часі, інтеграція інструментів штучного інтелекту для автоматичного занотовування, підсумовування обговорень або асистування під час конференцій. Використання таких рішень може підвищити ефективність взаємодії учасників і розширити можливості застосунку без суттєвих змін базової архітектури.

Перелік посилань:

1. What is WebRTC: A Beginner's Guide. URL: https://medium.com/@nakrani/what-is-webrtc-6e9cc2a51222 (дата звернення: 09.02.2026).

2. Agora Real-Time Voice and Video Engagement : веб-сайт. URL: https://www.agora.io/en/ (дата звернення: 09.02.2026).

3. Twilio : веб-сайт. URL: https://www.twilio.com/en-us (дата звернення: 09.02.2026).

4. QoS vs QoE: Measuring and Optimizing RTC User Experience. URL: https://trtc.io/blog/details/qos-vs-qoe (дата звернення: 09.02.2026).

5. Improving End-User Experience through Analytics: Quality of Experience (QoE) and Quality of Service (QoS) for RTC. URL: https://www.agora.io/en/blog/quality-of-service-and-quality-of-experience-for-rtc/ (дата звернення: 09.02.2026).

6. Nesler Tyler. What Is Interactive Streaming? URL: https://www.streamingmedia.com/Articles/Editorial/Short-Cuts/What-Is-Interactive-Streaming-157505.aspx (дата звернення: 09.02.2026).

7. Four causes of 'Zoom fatigue' and their causes. URL: https://news.stanford.edu/stories/2021/02/four-causes-zoom-fatigue-solutions (дата звернення: 09.02.2026).

8. React Documentation : веб-сайт. URL: https://react.dev/ (дата звернення: 09.02.2026).

9. Node.js Documentation : веб-сайт. URL: https://nodejs.org/en (дата звернення: 09.02.2026).

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ДИПЛОМ

ІІІ с т у п е н я

ЗА АКТИВНУ УЧАСТЬ У

XXIII МІЖНАРОДНІЙ НАУКОВО-ПРАКТИЧНІЙ КОНФЕРЕНЦІЇ МОЛОДИХ ВЧЕНИХ ТА СТУДЕНТІВ «СУЧАСНІ ПРОБЛЕМИ НАУКОВОГО ЗАБЕЗПЕЧЕННЯ ЕНЕРГЕТИКИ: БЕЗПЕКА, СТІЙКІСТЬ, ІТ ТА ЕКОЛОГІЧНИЙ МОНІТОРИНГ (ДО 40-РІЧЧЯ ЧОРНОБИЛЬСЬКОЇ КАТАСТРОФИ)»

нагороджується студентка гр. ТР – 21

кафедри ЦТЕ

Зелінська Вероніка Владиславівна

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» Навчально-науковий інститут атомної та теплової енергетики

В.о. директора НН ІАТЕ **Ольга ЧЕРНОУСЕНКО**

2026

ДОДАТОК Б

Фрагменти коду реалізації серверної та клієнтської логіки для взаємодії з онлайн-конференціями

“Вебзастосунок для проведення онлайн-конференцій”

УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ІАТЕ_ЦТЕ_ТР_21

Мови програмування — JavaScript / TypeScript

Аркушів 14

Реалізація серверної логіки для керування онлайн-конференціями

Програмні засоби:

— мова програмування — Javascript;

— середовище виконання — Node.js;

— фреймворк для маршрутизації — Express.js;

```
const { Router } = require('express');
const crypto = require('crypto');
const supabase = require('../lib/supabase');
const respond = require('../lib/respond');
const generateAgoraToken = require('../lib/generateAgoraToken');

const router = Router();

router.get('/', async (req, res) => {
 if (req.query.code && req.query.connectionId) {
   const { data: conference, error } = await supabase
     .from('conferences')
     .select('*, conference_participants!inner(user_id, is_host)')
     .eq('code', req.query.code)
     .eq('conference_participants.user_id', req.userId)
     .single();

   if (error ) return respond(res, { data: null, error });
      if (conference === undefined) return respond(res, { data: null, error: "No matching conference found." });

   const agoraToken = generateAgoraToken(conference.code, req.query.connectionId);
   console.log({ channelName: conference.code, uid: req.query.connectionId, agoraToken });
   return respond(res, { data: { ...conference, token: agoraToken }, error: null });
 }

 const { data, error } = await supabase
```

```
    .from('conferences')
    .select('*, conference_participants!inner(user_id, is_host)')
    .eq('conference_participants.user_id', req.userId)
    .gte('date', new Date(new Date().toDateString()).toISOString())
    .order('date', { ascending: true });

  if (error) return respond(res, { data: null, error });

  const ids = data.map((c) => c.id);
  const { data: participants } = await supabase
    .from('conference_participants')
    .select('conference_id, user_id, users(id, name, connection_id)')
    .in('conference_id', ids);

  const participantsMap = (participants || []).reduce((acc, row) => {
    if (!acc[row.conference_id]) acc[row.conference_id] = [];
      acc[row.conference_id].push({ userId: row.user_id, name: row.users?.name ?? null,
connectionId: row.users?.connection_id ?? null });
    return acc;
  }, {});

  const enriched = data.map((c) => ({
    ...c,
    participants: participantsMap[c.id] ?? [],
    participant_count: (participantsMap[c.id] ?? []).length,
  }));
  respond(res, { data: enriched, error: null });
});

router.post('/', async (req, res) => {
  const code = crypto.randomBytes(5).toString('hex').slice(0, 10).toUpperCase();

  const result = await supabase
    .from('conferences')
    .insert({
      name: req.body.name,
```

```
      date: req.body.date,
      duration: req.body.duration,
      agenda: req.body.agenda,
      creator_id: req.userId,
      code
    })
    .select()
    .single();

  if (result.data) {
    const { participantIds } = req.body;

    const rows = [
      { conference_id: result.data.id, user_id: req.userId, is_host: true },
          ...(Array.isArray(participantIds) ? participantIds.map((uid) => ({ conference_id: result.data.id, user_id: uid, is_host: false })) : []),
    ];

    await supabase.from('conference_participants').insert(rows);
  }

  respond(res, result);
});

router.patch('/', async (req, res) => {
  const result = await supabase
    .from('conferences')
    .update(req.body)
    .eq('id', req.query.id)
    .eq('creator_id', req.userId)
    .select()
    .single();

  respond(res, result);
});
```

```
router.delete('/', async (req, res) => {
 const result = await supabase
   .from('conferences')
   .delete()
   .eq('id', req.query.id)
   .eq('creator_id', req.userId)
   .select()
   .single();

 respond(res, result);
});

router.delete('/participants', async (req, res) => {
 const result = await supabase
   .from('conference_participants')
   .delete()
   .eq('conference_id', req.query.conferenceId)
   .eq('user_id', req.userId)
   .eq('is_host', false)
   .select()
   .single();

 respond(res, result);
});

module.exports = router;
```

Реалізація серверної логіки для генерації токенів для підключення до онлайн- конференції

Програмні засоби:

— мова програмування — Javascript;

— середовище виконання — Node.js;

— фреймворк для маршрутизації — Express.js;

```
const { Router } = require('express');
const { RtcTokenBuilder, RtcRole } = require('agora-access-token');
const generateAgoraToken = require('../lib/generateAgoraToken');

const router = Router();

const nocache = (_req, res, next) => {
 res.header('Cache-Control', 'private, no-cache, no-store, must-revalidate');
 res.header('Expires', '-1');
 res.header('Pragma', 'no-cache');
 next();
};

router.get('/rtc/:channel/:role/:tokentype/:uid', nocache, (req, res) => {
 const { channel, role: roleParam, tokentype, uid } = req.params;

 if (!channel) return res.status(400).json({ error: 'channel is required' });
 if (!uid) return res.status(400).json({ error: 'uid is required' });

 let role;
 if (roleParam === 'publisher') role = RtcRole.PUBLISHER;
 else if (roleParam === 'audience') role = RtcRole.SUBSCRIBER;
 else return res.status(400).json({ error: 'role is incorrect' });

 const expireTime = parseInt(req.query.expiry, 10) || 3600;
 const privilegeExpireTime = Math.floor(Date.now() / 1000) + expireTime;
```

```
    const { APP_ID, APP_CERTIFICATE } = process.env;

    let token;
    if (tokentype === 'userAccount') {
      token = generateAgoraToken(channel, uid, expireTime);
    } else if (tokentype === 'uid') {
       token = RtcTokenBuilder.buildTokenWithUid(APP_ID, APP_CERTIFICATE, channel, uid,
role, privilegeExpireTime);
    } else {
      return res.status(400).json({ error: 'token type is invalid' });
    }

    res.json({ rtcToken: token });
  });

  module.exports = router;
```

Реалізація підключення до конференції на клієнтській стороні

Програмні засоби:

— мова програмування — Typescript;

— інтерфейсна бібліотека — React.js;

Порядок вкладення компонентів наступний: AgoraWrapper, ConferenceSectionInner, VideoGrid.

Компонент AgoraWrapper:

```
import { FC, memo, useMemo } from 'react';
import AgoraRTC, { AgoraRTCProvider } from 'agora-rtc-react';

interface AgoraWrapperProps {
 children: React.ReactNode;
};

const AgoraWrapperComponent: FC<AgoraWrapperProps> = ({children}) => {
 const client = useMemo(() => AgoraRTC.createClient({ mode: "rtc", codec: "vp8" }), []);

 return (
   <AgoraRTCProvider client={client}>
     {children}
   </AgoraRTCProvider>
 );
};

export const AgoraWrapper = memo(AgoraWrapperComponent);
```

Контроллер у ConferenceSectionInner, в якому відбувається основна логіка підключення:

```
import { useNavigate, useParams } from "react-router-dom";
import { useEffect, useMemo, useState } from "react";
import { useJoin, useLocalCameraTrack, useLocalMicrophoneTrack, usePublish, useRTCClient } from "agora-rtc-react";
import { useConferenceController } from "hooks/useConferenceController";
import { useProfileStore } from "stores/profileStore";

const appId = process.env.REACT_APP_AGORA_APP_ID || "";

export const useConferenceSectionController = () => {
 const { id: channelName } = useParams<{ id: string }>();
    const { currentConference, setCurrentConference, joinConference } = useConferenceController(true);
 const { profile } = useProfileStore();

 const [isConnected, setIsConnected] = useState(false);
 const [isLoading, setIsLoading] = useState(true);

 const [micOn, setMic] = useState(true);
 const [cameraOn, setCamera] = useState(true);
 const [notesOpen, setNotesOpen] = useState(false);
 const [testOpen, setTestOpen] = useState(false);

 const client = useRTCClient();
  const navigate = useNavigate();

 useEffect(() => {
   if (channelName && profile?.connectionId) {
     joinConference(channelName, profile.connectionId).then((success) => {
       if (!success) {
         navigate("/home");
       }
```

```
      });
    }
  }, [profile?.connectionId]);

  const isReady = useMemo(() =>
    !!currentConference && !!profile?.connectionId && !!channelName,
  [currentConference, profile?.connectionId, channelName]);

  const result = useJoin({
    appid: appId,
    channel: channelName || "",
    token: currentConference?.token || "",
    uid: profile?.connectionId || 0,
  }, isReady);

  useEffect(() => {
    setIsConnected(result.isConnected);
    setIsLoading(result.isLoading);
  }, [result])

  const { localMicrophoneTrack } = useLocalMicrophoneTrack(true);
  const { localCameraTrack } = useLocalCameraTrack(true);

   useEffect(() => { localMicrophoneTrack?.setEnabled(micOn); }, [localMicrophoneTrack,
micOn]);
    useEffect(() => { localCameraTrack?.setEnabled(cameraOn); }, [localCameraTrack,
cameraOn]);

  usePublish([localMicrophoneTrack, localCameraTrack]);

  const handleMute = () => {
    setMic((prev) => !prev);
  };

  const handleCamera = () => {
    setCamera((prev) => !prev);
```

```
};

const handleNotes = () => {
  setNotesOpen((prev) => !prev);
};

const handleTest = () => {
  setTestOpen((prev) => !prev);
};

const handleLeave = async () => {
  localCameraTrack?.stop();
  localCameraTrack?.close();
  localMicrophoneTrack?.stop();
  localMicrophoneTrack?.close();
  await client.leave();
  setCurrentConference(undefined);
  navigate("/home");
};


return {
  channelName,
  localMicrophoneTrack,
  localCameraTrack,
  micOn,
  cameraOn,
  notesOpen,
  testOpen,
  isLoading,
  isConnected,
  actions: {
    handleMute,
    handleCamera,
    handleNotes,
    handleTest,
```

```
    handleLeave,
  }
 };
};
```

Компонент ConferenceSectionInner:

```
import { FC, memo } from "react";
import { CircularProgress, SxProps } from "@mui/material";

import { Box } from "ui/Box";
import ConferenceTopBar from "../ConferenceTopBar";
import ConferenceToolbar from "../ConferenceToolbar";
import { styles } from "./styles";
import { useConferenceSectionController } from "./useConferenceSectionController";
import { AgoraWrapper } from "../AgoraWrapper";
import VideoGrid from "../VideoGrid";
import NotesPanel from "../NotesPanel";
import TestPanel from "../TestPanel";
import { useConferenceStore } from "stores/conferenceStore";
import { useConferenceTestController } from
"modules/ConferenceModule/useConferenceTestController";

const ConferenceSectionInner: FC = () => {
 const {
  localMicrophoneTrack,
  localCameraTrack,
  micOn,
  cameraOn,
  notesOpen,
  testOpen,
  channelName,
  actions,
  isLoading,
  isConnected,
```

```
} = useConferenceSectionController();
const { currentConference, currentTest } = useConferenceStore();
const { submitAnswer, refetchTest, createTest } = useConferenceTestController(currentConference?.id, currentConference?.isHost ?? false);

return (...)
};

const ConferenceSection: FC = () => (
 <AgoraWrapper>
  <ConferenceSectionInner />
 </AgoraWrapper>
);

export default memo(ConferenceSection);
```

Компонент VideoGrid:

```
import { FC } from "react";
import { SxProps } from "@mui/material";

import { Box } from "ui/Box";
import ParticipantTile from "../ParticipantTile";
import { styles } from "./styles";
import { getVideoGridLayout, getVideoWidth } from "./helpers";
import { ILocalAudioTrack, ILocalVideoTrack, LocalUser, RemoteUser, useRemoteUsers } from "agora-rtc-react";
import { JoinableConference } from "types/conference";

interface Props {
 currentConference: JoinableConference | undefined,
 localMicrophoneTrack: ILocalAudioTrack | null;
 localCameraTrack: ILocalVideoTrack | null;
 micOn: boolean;
 cameraOn: boolean;
}
```

```
const VideoGrid: FC<Props> = ({ currentConference, localMicrophoneTrack, localCameraTrack, micOn, cameraOn }) => {
  const remoteUsers = useRemoteUsers();
  const totalCount = remoteUsers.length + 1;
  const { maxRemoteTiles, columns, rows } = getVideoGridLayout(totalCount);
  const widthSx = getVideoWidth(totalCount, rows);
  const visibleRemote = remoteUsers.slice(0, maxRemoteTiles);

  return (
    <Box sx={[styles.root, widthSx, { gridTemplateColumns: `repeat(${columns}, 1fr)`, gridTemplateRows: `repeat(${rows}, 1fr)` }] as SxProps}>
      {visibleRemote.map((user) => {
        const participant = currentConference?.participants.find((p) => p.connectionId === Number(user.uid));

        return (
          <ParticipantTile
            key={user.uid}
            participant={{
              id: String(user.uid),
              name: participant?.name ?? String(user.uid),
              isMuted: !user.hasAudio,
              isCameraOff: !user.hasVideo,
            }}
          >
            <RemoteUser user={user} playAudio={user.hasAudio} style={{ width: "100%", height: "100%" }} />
          </ParticipantTile>
        )
      })}

      <ParticipantTile
        participant={{
          id: "local",
          name: "You",
```

```
          isYou: true,
          isMuted: !micOn,
          isCameraOff: !cameraOn,
        }}
      >
        <LocalUser
          cameraOn={cameraOn}
          micOn={micOn}
          videoTrack={localCameraTrack}
          style={{ width: "100%", height: "100%", minHeight: 0 }}
        />
      </ParticipantTile>
    </Box>
  )
};

export default VideoGrid;
```