ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ

Пояснювальна записка

до кваліфікаційної роботи *магістра*

на тему: “Моделювання та реалізація веб-системи для анонімної комунікації”

Виконав: студент *6-го* курсу, групи *6ПР1*, *денної* форми навчання, спеціальності *121 - “Інженерія програмного забезпечення”*

Половюк Олег Сергійович

Керівник: *к.т.н., доцент Огнєва О.Є.*

Рецензент: *к.т.н., доцент Конрніловська Н.В.*

Хмельницький - 2025 р.

ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет: *Інформаційних технологій та дизайну*.

Кафедра: *Програмних засобів і технологій*.

Освітньо-кваліфікаційний рівень: *магістр*.

Галузі знань: *12 - Інформаційні технології*.

Освітньо-професійна програма: *Програмне забезпечення систем*.

Спеціальність: *121 – Інженерія програмного забезпечення*.

ЗАТВЕРДЖУЮ

Завідувач кафедри *програмних засобів і технологій* к.т.н., доцент О. Є. Огнєва.

“____” _____ 2025 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Половюку Олегу Сергійовичу

1. Тема роботи: *“Моделювання та реалізація веб-системи для анонімної комунікації”*. Керівник роботи: *к.т.н., доцент Огнєва Оксана Євгенівна*. Затверджені наказом ХНТУ від *15 вересня* 2025 року № *416-с*.
2. Строк подання студентом роботи: *22.12.2025*.
3. Вихідні дані до роботи: *ДСТУ з обробки інформації, літературні та періодичні джерела, матеріали практики*.
4. Зміст пояснювальної записки: *аналіз предментної області (характеристика предметної області, аналіз існуючих аналогів та рішень, вибір технологій та інструментів розробки, вимоги до програмного забезпечення); проектування програмного забезпечення (діаграма прецедентів, діаграма діяльності, діаграма послідовності, архітектура інформаційної системи, діаграма потоків даних, діаграма сутностей та зв’язків, прототип користувацького інтерфейсу, файлова структура); реалізація програмного забезпечення (аналіз та налаштування конфігурації проекту, формування структури бази даних,*

організація роботи серверної логіки, організація роботи клієнтської логіки); тестування програмного забезпечення (підготовка середовища для тестування, функціональне тестування користувацького інтерфейсу, тестування API з використанням Postman, аналіз бази даних після виконання тестів).

5. Перелік графічного матеріалу: *сімдесят два зображення, тринадцять креслень та одна комп'ютерна презентація*.
6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання: *15.09.2025*.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	*Отримання завдання*	15.09.2025	Виконано
2	*Аналіз предметної області*	22.09.2025-05.10.2025	Виконано
3	*Проектування програмного забезпечення*	06.10.2025-19.10.2025	Виконано
4	*Реалізація програмного забезпечення*	27.10.2025-23.20.2025	Виконано
5	*Тестування програмного забезпечення*	24.11.2025-30.11.2025	Виконано
6	*Оформлення пояснювальної записки*	01.12.2025-14.12.2025	Виконано
7	*Захист кваліфікаційної роботи*	22.12.2025	

Студент: _____ *Половюк О.С.*
(підпис) (прізвище та ініціали)

Керівник роботи: _____ *Огнєва О.Є.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська кваліфікаційна робота присвячена моделюванню та реалізації веб-системи для анонімної текстової комунікації користувачів. Робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

Перший розділ присвячений аналізу предметної області анонімних комунікаційних систем, в якому розглянуто особливості анонімної взаємодії користувачів у цифровому середовищі, проаналізовано існуючі програмні аналоги та підходи до реалізації подібних рішень, обґрунтовано вибір технологій та інструментів розробки, а також сформульовано вимоги до ПЗ.

Другий розділ відповідає за проектування програмного забезпечення, у межах якого побудовано та описано основні моделі системи, зокрема діаграми прецедентів, діяльності, послідовності, потоків даних, архітектури інформаційної системи та діаграму сутностей та зв’язків. Крім того представлено прототип користувацького інтерфейсу та файлову структуру програмного забезпечення..

Третій розділ присвячений реалізації програмного забезпечення, в якому розглянуто налаштування конфігурації проекту, формування структури бази даних, організацію серверної та клієнтської логіки веб-системи.

Четвертий розділ описує тестування програмного забезпечення, в якому наведено підготовку середовища для тестування, демонстрацію функціональних можливостей користувацького інтерфейсу, перевірку роботи API за допомогою Postman, а також аналіз стану бази даних після виконання тестових сценаріїв.

Кожен розділ кваліфікаційної роботи завершується окремими висновками, у яких узагальнюються результати виконаного аналізу, проектування, реалізації та тестування програмного забезпечення.

ABSTRACT

The master's qualification thesis is devoted to the modeling and implementation of a web-based system for anonymous text communication between users. The thesis consists of an introduction, four chapters, conclusions, a list of references, and appendices.

The first chapter is devoted to the analysis of the subject area of anonymous communication systems. It examines the features of anonymous user interaction in a digital environment, analyzes existing software analogs and approaches to the implementation of similar solutions, substantiates the selection of technologies and development tools, and formulates the requirements for the software.

The second chapter focuses on software design. Within this chapter, the main system models are developed and described, including use case diagrams, activity diagrams, sequence diagrams, data flow diagrams, information system architecture, and an entity–relationship diagram. In addition, a user interface prototype and the file structure of the software are presented.

The third chapter is dedicated to the implementation of the software. It considers the configuration setup of the project, the formation of the database structure, and the organization of the server-side and client-side logic of the web system.

The fourth chapter describes software testing. It presents the preparation of the testing environment, the demonstration of the functional capabilities of the user interface, the verification of API operation using Postman, and the analysis of the database state after the execution of test scenarios.

Each chapter of the qualification thesis concludes with separate conclusions that summarize the results of the performed analysis, design, implementation, and testing of the software.

ЗМІСТ

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням обсягів цифрової комунікації та трансформацією способів міжособистісної взаємодії. Веб-орієнтовані інформаційні системи дедалі частіше виступають не лише як інструменти обміну інформацією, але й як середовище формування соціальних зв'язків, дискусійних просторів та неформальної комунікації. Особливе місце серед таких систем займають платформи анонімного спілкування, які дозволяють користувачам взаємодіяти без розкриття особистих даних, що, у свою чергу, впливає на характер поведінки, стиль спілкування та рівень відкритості учасників.

Анонімна текстова комунікація являється актуальним явищем у сучасному цифровому суспільстві, оскільки відповідає потребі користувачів у свободі висловлювання, зниженні соціального тиску та можливості взаємодії без попередньої ідентифікації. Водночас аналіз наявних веб-рішень свідчить про те, що більшість існуючих платформ або орієнтовані на асинхронну взаємодію у форматі форумів, або реалізують синхронне спілкування з обов'язковою реєстрацією чи прив'язкою до особистих облікових записів. Окрему нішу займають сервіси випадкової комунікації, проте вони часто використовують відеоформат, що значно підвищує бар'єр входу та не відповідає потребам користувачів, зацікавлених саме у текстовій формі анонімного спілкування.

В даному контексті постає науково-практична проблема створення веб-системи, яка поєднує переваги синхронної комунікації в режимі реального часу з принципами повної анонімності та мінімального порогу входу. Вирішення цієї проблеми потребує комплексного підходу, що охоплює аналіз предметної області, проектування архітектури системи, реалізацію серверної та клієнтської логіки, а також перевірку коректності функціонування в реальних умовах використання. Саме необхідність такого системного дослідження зумовлює доцільність виконання даної магістерської кваліфікаційної роботи.

Актуальність теми зумовлена зростаючою потребою у веб-орієнтованих системах анонімної комунікації, які забезпечують миттєвий доступ до взаємодії без процедур реєстрації чи автентифікації, підтримують обмін повідомленнями в режимі реального часу та дозволяють користувачам взаємодіяти у тимчасових комунікаційних середовищах. Додаткову актуальність темі надає відсутність універсальних рішень, що поєднують функціональність форумних систем та сервісів випадкової комунікації у межах єдиного текстового середовища, орієнтованого на веб-технології.

Метою дослідження являється моделювання та реалізація веб-системи для анонімної текстової комунікації, яка забезпечує створення та використання комунікаційних середовищ у режимі реального часу без застосування механізмів реєстрації користувачів.

Для досягнення поставленої мети в роботі необхідно вирішити слідуючі **завдання**: проаналізувати предметну область анонімної текстової комунікації та існуючі програмні аналоги; визначити вимоги до програмного забезпечення; виконати проектування інформаційної системи із використанням відповідних діаграм і моделей; реалізувати серверну та клієнтську частини веб-системи з урахуванням вимог до анонімності та взаємодії в реальному часі; здійснити тестування функціональних можливостей програмного забезпечення та проаналізувати результати його роботи.

Об’єктом дослідження є процеси анонімної текстової комунікації користувачів у веб-орієнтованих інформаційних системах, тоді як **предметом дослідження** являються методи, моделі та програмні засоби проектування та реалізації веб-системи для анонімної комунікації в режимі реального часу.

Наукова новизна одержаних результатів полягає у формуванні цілісної моделі веб-системи анонімного текстового спілкування, що поєднує принципи тимчасової ідентифікації користувачів, динамічного створення комунікаційних середовищ і синхронного обміну повідомленнями без використання механізмів персоніфікації. Запропонований підхід дозволяє розглядати анонімність не як

додаткову функцію, а як базову властивість системи, інтегровану на всіх рівнях її функціонування.

Практичне значення отриманих результатів полягає у можливості використання розробленої веб-системи як готового програмного рішення для організації анонімної текстової комунікації або як основи для подальшого розвитку подібних інформаційних систем. Отримані проектні та програмні рішення можуть бути застосовані у навчальному процесі, дослідницькій діяльності, а також під час створення веб-орієнтованих сервісів комунікаційного призначення.

Апробація результатів магістерської кваліфікаційної роботи здійнювалася через наукові публікації та матеріалали конференцій, тематика яких безпосередньо пов’язана з проблематикою анонімної веб-комунікації. Основні положення дослідження викладені у тезах доповідей, присвячених аналізу сучасного інструментарію для моделювання та реалізації веб-систем анонімної комунікації, а також застосуванню криптографічних підходів у процесі проектування та реалізації анонімних веб-рішень. Зазначені результати оприлюднені в межах VIII Всеукраїнської науково-практичної конференції “Сучасні інформаційні системи та технології” й X Міжнародної наукової конференції “Проблеми та перспективи реалізації та впровадження міждисциплінарних наукових досягнень”, що підтверджує практичну спрямованість виконаного дослідження.

Загалом виконане дослідження спрямоване на розв’язання актуальної науково-практичної задачі, пов’язаної зі створенням сучасних веб-орієнтованих систем анонімної комунікації, та поєднує теоретичні положення з їх практичним застосуванням. Запропонований підхід демонструє можливість побудови цілісного програмного рішення, у якому анонімність, інтерактивність і простота використання виступають взаємопов’язаними складовими єдиної концепції. Отримані результати підтверджують доцільність обраних методів і технологій, а також створюють передумови для подальших досліджень у напрямі розвитку веб-систем комунікаційного призначення з урахуванням соціальних, технічних і поведінкових аспектів взаємодії користувачів.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Аналіз предметної області передбачає дослідження теоретичних і практичних засад, що визначають умови створення та функціонування програмних систем певного призначення, а також виявлення особливостей, обмежень і закономірностей, властивих цій сфері. У межах такого аналізу розглядаються характеристики об’єкта дослідження, наявні рішення та підходи, технічні засоби й технології, здатні забезпечити ефективну реалізацію програмного продукту, а також формулюються вимоги, необхідні для подальшого проектування та впровадження системи.

1.1. Характеристика предметної області

Предметна область анонімної текстової комунікації у веб-середовищі охоплює сукупність соціальних, інформаційних та технологічних процесів, спрямованих на забезпечення взаємодії між невідомими один одному користувачами без необхідності розкриття персональних даних. Інтерес до подібних систем формується під впливом розвитку цифрових комунікацій, зростання попиту на конфіденційність та бажання користувачів брати участь у відкритих або спонтанних дискусіях без процедур ідентифікації. У межах таких середовищ відбувається моделювання специфічної форми соціальної взаємодії, де акцент переноситься з індивідуальних характеристик користувача на сам зміст повідомлень і динаміку їхнього обміну [4, 5, 10, 27].

Анонімна комунікація має низку відмінних рис, які визначають її особливе місце серед інших форм цифрової взаємодії. Вона усуває бар’єри, пов’язані з ретельною підготовкою профілю, і мінімізує психологічні обмеження, що властиві ідентифікованим формам спілкування. У подібних системах користувачі можуть вступати у взаємодію без тривалих попередніх процедур, що спричиняє високу динамічність комунікаційних потоків. Саме миттєвість доступу та відсутність зобов’язань щодо розкриття особистості створюють передумови для розвитку текстових середовищ, у яких головним чинником є зміст повідомлення, а не авторство [5, 10, 27].

Зародження ідеї розроблення програмного забезпечення для анонімного спілкування часто пов’язане з культурними чи медійними джерелами, де моделюється подібна форма соціальної взаємодії. Одним із прикладів є художні твори, в яких описуються сценарії обміну повідомленнями між невідомими особами в умовах повної анонімності. Попри художнє походження подібних концепцій, вони формують реальне технологічне підґрунтя для створення інструментів, що задовольняють сучасні потреби швидкого та необтяжливого комунікування у цифровому просторі. Переосмислення таких ідей у контексті веб-середовища дає змогу поєднати принципи форумів, які забезпечують структурованість комунікації, та механізми спонтанного підбору співрозмовників, характерні для динамічних чат-сервісів [4, 5, 36].

Розглядаючи анонімну текстову комунікацію як предметну область, слід окреслити її ключові компоненти. До них належать користувачі як учасники процесу; комунікаційні простори (кімнати), що виступають тимчасовими або постійними середовищами взаємодії; механізми маршрутизації та передавання повідомлень; а також алгоритми керування доступом і регулювання навантаження. Важливу роль відіграють і засоби синхронізації, які забезпечують можливість обміну повідомленнями у режимі реального часу. Технологічною складовою предметної області є платформи та протоколи, що підтримують подібні форми взаємодії, зокрема засоби веб-розробки, інструменти роботи з даними та сервіси реального часу [15, 18, 35, 38].

Таблиця 1

Основні характеристики предметної області анонімної текстової комунікації [5, 15, 36]

Характеристика	Змістовий опис
Анонімність	Відсутність необхідності у реєстрації, підтвердженні особи або розкритті персональних даних
Миттєвий доступ	Можливість увійти до середовища взаємодії без додаткових процедур підготовки

Тимчасові комунікаційні простори	Кімнати, що створюються для окремих сесій взаємодії та можуть бути видалені після завершення
Синхронність взаємодії	Передавання повідомлень у режимі реального часу між невідомими учасниками
Непередбачуваність складу учасників	Користувачі не мають інформації про співрозмовників і не контролюють процес їхнього добору
Відсутність соціального тиску	Комунікація відбувається без прив’язки до особистої репутації чи соціального контексту

У сучасних умовах стрімкого розвитку веб-технологій попит на анонімні текстові системи комунікації продовжує зростати. Одним із чинників є прагнення частини користувачів до безпечного середовища, де відсутні соціальні ризики, пов’язані з персональною ідентифікацією. Крім того, такі системи використовуються для швидкого обміну інформацією, проведення дискусій або отримання спонтанного зворотного зв’язку. Текстові інтерфейси у цьому контексті виступають універсальним способом комунікації, що не залежить від наявності аудіо- чи відеообладнання та не накладає додаткових вимог на технічні ресурси користувача [10, 17, 40].

Особливістю предметної області також є потреба в ефективних алгоритмах, що регулюють навантаження та керують короткочасними комунікаційними сесіями. На відміну від традиційних форумів, де взаємодія має переважно асинхронний характер, або відеочатів, що використовують високоресурсні канали, текстові анонімні системи ґрунтуються на швидкому обміні невеликими обсягами даних. Це зумовлює застосування технологій, які забезпечують низькі часові затримки, гарантовану доставку повідомлень і можливість масштабування системи без погіршення якості взаємодії [15, 18, 35, 38].

Характеристика предметної області дозволяє сформувати цілісне уявлення про функціональний, соціальний та технологічний контекст, у якому відбувається розроблення веб-системи анонімного спілкування. Визначення ключових

властивостей та умов функціонування таких систем створює передумови для подальшого аналізу аналогів, обґрунтованого вибору технологій та формулювання вимог до програмного забезпечення у наступних підрозділах [29, 36].

1.2. Аналіз існуючих аналогів та рішень

Анонімна цифрова комунікація являється поширеною складовою сучасного веб-середовища, що реалізується у вигляді різних платформ, орієнтованих на приховане листування або спонтанну взаємодію між користувачами. Аналіз наявних аналогів дозволяє визначити функціональні властивості, які стали стандартом у сфері анонімних сервісів, а також виявити обмеження, характерні для існуючих рішень. Даний аналіз формує цінні орієнтири для розуміння логіки побудови анонімних систем та окреслює проблему, яку має вирішувати розроблюване програмне забезпечення [5, 10, 36].

Серед найбільш відомих аналогів можна виокремити сервіси спонтанного підбору співрозмовників, такі як Omegle та Chatroulette, які забезпечують анонімне спілкування між випадковими користувачами. Основою таких платформ є відсутність реєстрації та формування тимчасової сесії спілкування між двома невідомими користувачами. Проте дані сервіси переважно орієнтовані на відео- або аудіокомунікацію, що створює додаткові технічні вимоги та знижує ступінь фактичної анонімності через непряме розкриття особистої інформації за допомогою зображення або голосу. Незважаючи на свою популярність, такі рішення не забезпечують повністю текстової моделі взаємодії та не передбачають гнучкої структури кімнат у межах одного ресурсу [5, 10, 38].

Другу групу аналогів становлять анонімні форуми та текстові дошки, зокрема 4chan, Dread та інші платформи, засновані на використанні псевдонімів або одноразових ідентифікаторів для кожної публікації чи коментаря. Основною перевагою таких сервісів являється можливість повного приховування особистості, проте модель взаємодії є суто асинхронною. Вона орієнтована на тривалі дискусії, а не на швидкий обмін повідомленнями. Крім того, структура форумів не передбачає

автоматизованого формування закритих комунікаційних просторів із динамічним складом учасників, що є важливим у контексті анонімних чат-систем [4, 27, 36].

Окрему групу становлять текстові месенджери з підтримкою прихованих або тимчасових кімнат, серед яких можна виокремити Telegram (анонімні коментарі та канали), Discord (тимчасові приватні канали), а також менш поширені анонімні мобільні додатки. Хоча вказані платформи й забезпечують певний рівень анонімності, вони все одно базуються на механізмах реєстрації або ідентифікації через обліковий запис, номер телефону чи інтегровані сервіси. Таким чином, вони не можуть бути віднесені до повністю анонімних систем, оскільки будь-яка форма реєстрації створює залежність між користувачем та платформою [10, 27, 40].

Таблиця 2

Порівняння найбільш поширених аналогів анонімного спілкування [5, 10, 36]

Програмний аналог	Тип взаємодії	Механізм анонімності	Основні обмеження
Omegle	Спонтанний підбір пари, відео/текст	Відсутність реєстрації, тимчасові сесії	Орієнтація на відео, низька керованість комунікаційними просторами
Chatroulette	Спонтанний підбір співрозмовників	Тимчасові ідентифікатори	Основна взаємодія — відео, відсутність текстових просторів
4chan	Асинхронні текстові дискусії	Одноразові ідентифікатори для дописів	Немає режиму реального часу, відсутність динамічних кімнат
Discord	Закриті канали	Приватні кімнати, але через акаунт	Не являється анонімним, залежність від реєстрації
Telegram	Канали та групи	Можливість прихованих коментарів	Прив’язка до номера телефону, часткова анонімність

Аналіз аналогів свідчить, що наявні рішення демонструють лише часткову реалізацію властивостей, притаманних анонімним текстовим сервісам. Сервіси на кшталт Omegle та Chatroulette забезпечують спонтанність і невизначеність вибору співрозмовника, але орієнтовані на відеоформат і не реалізують структуру чат-кімнат. Анонімні форуми гарантують високий рівень прихованості, але не підтримують синхронного обміну повідомленнями. Месенджери з приватними каналами забезпечують гнучкість комунікаційних структур, проте залежать від реєстрації, що суперечить вимозі повної анонімності [5, 10, 38].

Незважаючи на різноманітність існуючих рішень, залишається прогалина між сервісами спонтанного знайомства та анонімними текстовими платформами: відсутнє програмне забезпечення, яке б одночасно забезпечувало миттєвий доступ, відсутність реєстрації, динамічні текстові кімнати й обмін повідомленнями в режимі реального часу. Саме цю нішу займає система, у якій анонімність реалізується за допомогою тимчасових ідентифікаторів та гнучкої структури кімнат, що робить її концептуально новим поєднанням форумної логіки, елементів чат-рулетки та текстових комунікаційних сервісів [15, 27, 36].

1.3. Вибір технологій та інструментів розробки

Вибір технологій для створення веб-системи анонімної текстової комунікації визначається властивостями предметної області, яка передбачає динамічний обмін повідомленнями, високу варіативність сценаріїв взаємодії та необхідність оперативної передачі даних без помітних затримок. З огляду на те, що комунікація у такій системі відбувається між невідомими користувачами та має короткочасний характер, технологічний стек повинен забезпечувати підтримку великої кількості паралельних підключень, швидку обробку подій, адаптивність до змін у структурі даних та стабільну роботу в умовах реального часу. Крім того, до технологій ставляться вимоги щодо зручності інтеграції різних компонентів, простоти масштабування та гнучкості при розробленні інтерфейсу [15, 29, 36, 38].

Платформи для серверної розробки мають різний підхід до організації оброблення запитів. Серед доступних варіантів виділяються середовища, що використовують неблокувальний механізм виконання, завдяки якому зменшуються затримки під час опрацювання запитів і підвищується пропускна здатність системи. Такий механізм є важливим для забезпечення синхронної взаємодії між клієнтами, де кожне повідомлення повинно негайно надходити до всіх учасників сеансу. Саме тому середовище, орієнтоване на подієву модель роботи, часто розглядається як природний варіант для систем, які функціонують за принципом миттєвого обміну даними [25, 26, 38].

Додаткового обґрунтування потребує вибір технологій, що стосуються обміну інформацією у режимі реального часу. Протоколи традиційних HTTP-запитів недостатньо ефективні для багатократного миттєвого обміну короткими повідомленнями, тому доцільно розглядати інструменти, що дозволяють підтримувати постійний канал зв’язку між клієнтом та сервером. Такі механізми дають змогу негайно передавати повідомлення усім учасникам певного комунікаційного простору та реагувати на події без здійснення повторних запитів від клієнта. Саме ця властивість являється необхідною для моделей, у яких зміна кількості активних користувачів чи надходження нового повідомлення має бути відображено миттєво [18, 34, 35].

Серед засобів збереження даних актуальним є підхід, що дозволяє працювати зі структурами, які можуть змінюватися у процесі розроблення або експлуатації. Текстова анонімна комунікація створює динамічні набори документів: онлайн-кімнати виникають та зникають у довільний момент, повідомлення формуються у нестабільних за обсягом потоках, а тривалість існування інформації може бути короткою. Тому вибір системи керування базами даних має ґрунтуватися на можливості гнучко оперувати даними та швидко обробляти великі колекції документів без необхідності жорсткого регламентування їхньої структури. Документноорієнтована модель роботи з даними відповідає вимогам такого середовища [15, 21, 22, 33].

Для клієнтської частини, актуальним являється застосування інструментів, здатних підтримувати швидке оновлення інтерфейсу у відповідь на події, які надходять у реальному часі. Компонентний підхід дозволяє побудувати інтерфейс із чітким поділом на логічні складові, кожна з яких відповідає за власний стан і спосіб відображення. Оскільки зміни у таких системах відбуваються постійно (надходження повідомлень, оновлення кількості користувачів, зміна активного комунікаційного простору), інструменти інтерфейсної розробки мають забезпечувати мінімальні витрати на рендеринг та оновлення стану. Важливим також є застосування механізмів форматування стилів, які спрощують підтримку інтерфейсу та дозволяють структурувати великі обсяги стилізаційних правил [30, 31, 32].

Окремої уваги потребує вибір допоміжних інструментів, що використовуються на етапах тестування, контролю версій, відлагодження та аналізу проміжних результатів. Тестування API потребує засобів, що дозволяють формувати та надсилати запити у контрольованих умовах, а робота з даними — інструментів візуального перегляду стану бази. Розробка інтерфейсу й серверних компонентів вимагає використання зручного середовища програмування, тоді як керування проектом передбачає застосування системи контролю версій [9, 20, 28, 39].

Таблиця 3

Обґрунтування вибору технологій та інструментів розробки веб-системи

Технологія / інструмент	Обґрунтування доцільності використання
Node.js	Подієва неблокувальна модель дозволяє обробляти значну кількість паралельних підключень та зменшує час реакції системи [25, 26]
Express	Гнучка структура для маршрутизації та організації серверної логіки, що полегшує інтеграцію з іншими компонентами [6, 7]
Socket.io	Підтримка постійного двонапрямного зв’язку між клієнтом та сервером, необхідного для

	режиму реального часу [35]
MongoDB	Можливість зберігати динамічні структури даних, що відповідає природі короткочасних комунікаційних просторів [21, 22]
Mongoose	Декларативне визначення структур даних та контроль над взаємодією з документами [23, 24]
React	Компонентний підхід та швидке оновлення інтерфейсу у відповідь на події [31]
Fetch API	Проста та стандартизована модель роботи з HTTP-запитами [16, 30]
Socket.io Client	Забезпечення синхронізації між клієнтом та сервером у режимі реального часу [34]
SCSS	Структурована організація стилів та можливість розширення стилізаційних правил [32]
Visual Studio Code	Зручна робота з кодом та підтримка численних інструментів розробки [39]
MongoDB Compass	Візуальний аналіз стану бази даних та зручне дослідження документів [20]
Postman	Формування та тестування HTTP-запитів під час розробки [28]
Microsoft Edge	Перевірка коректності роботи інтерфейсу та поведінки клієнтської частини у браузері [19]
Git	Керування версіями проекту та командна взаємодія [9]

Сукупність розглянутих технологій та інструментів формує основу, що відповідає вимогам системи анонімної текстової комунікації. Їхні властивості забезпечують поєднання високої продуктивності, гнучкості структури даних, можливості динамічного оновлення інтерфейсу та ефективного керування проектом. Завдяки цьому технологічний стек здатний підтримувати швидкі комунікаційні процеси, обробляти інформацію у режимі реального часу та адаптуватися до змін,

які виникають під час короткочасних або інтенсивних сесій взаємодії. Такий вибір сприяє створенню функціонального, масштабованого й енергоефективного програмного забезпечення, що відповідає особливостям предметної області [15, 29, 36].

1.4. Вимоги до програмного забезпечення

Формування вимог до програмного забезпечення є ключовим етапом аналізу предметної області, оскільки саме на ньому визначаються властивості системи, її функціональне призначення, обмеження, модель взаємодії з користувачем та технічні характеристики, необхідні для стабільної роботи. Вимоги відображають специфіку веб-системи анонімної комунікації, для якої характерні динамічність, відсутність реєстрації, висока швидкість передавання повідомлень і короткочасність комунікаційних сесій. Окрему увагу приділено забезпеченню коректної взаємодії серверної та клієнтської частин системи [11, 29, 36].

Система повинна забезпечувати можливість створення комунікаційних кімнат та підключення до них інших користувачів у режимі реального часу без попередньої автентифікації. Оскільки анонімність являється суттєвою характеристикою предметної області, проект передбачає використання механізму тимчасової ідентифікації, що дозволяє здійснювати обмін повідомленнями без збереження персональних даних. Система також має підтримувати роботу з кількома режимами комунікації, які відрізняються кількістю учасників та характером взаємодії. Важливими вимогами є коректне відображення кількості активних користувачів, надання можливості переглядати повідомлення, які були створені до моменту підключення, та забезпечення безпечного видалення кімнат відповідно до встановлених правил [11, 14, 27].

Оскільки система функціонує як інтерактивна веб-платформа, вимоги охоплюють також характеристики клієнтського інтерфейсу. Інтерфейс повинен бути інтуїтивним, реагувати на події негайно та не створювати додаткових затримок під час обміну даними. Адаптація до різних розмірів екранів, швидке рендерування компонентів, чітка структура інтерфейсних модулів та можливість відображення

системних повідомлень є невід'ємними складовими. Окремі вимоги стосуються механізму передавання повідомлень і структур даних, що повинні забезпечувати синхронність роботи всіх користувачів однієї кімнати [14, 17, 36].

Технічні вимоги визначають характеристики середовища, у якому має функціонувати система, а також вимоги до оброблення інформаційних потоків. Інтерактивний характер комунікації потребує підтримки постійного каналу зв'язку між клієнтом та сервером, що виключає можливість використання лише традиційних механізмів запит-відповідь. База даних повинна забезпечувати швидкий запис і вибірку коротких документів з можливістю масштабування у разі збільшення кількості користувачів. Вимоги до продуктивності включають обмеження на кількість одночасних кімнат, кількість повідомлень, що відображаються користувачеві, а також швидкість оброблення підключення та відключення учасників [14, 15, 38].

Таблиця 4

Вимоги до веб-системи анонімної комунікації [11, 24, 36]

Тип вимоги	Зміст вимоги
Функціональні вимоги	Система повинна забезпечувати створення нових комунікаційних кімнат у різних режимах; можливість підключення до наявних кімнат; передавання та отримання повідомлень у режимі реального часу; відображення кількості активних користувачів; формування тимчасового ідентифікатора для кожного учасника; можливість перегляду попередніх повідомлень; механізм коректного виходу з кімнати; можливість ініціювати видалення кімнати відповідно до встановлених обмежень
Нефункціональні вимоги	Час реакції системи повинен бути мінімальним; обмін повідомленнями не повинен мати відчутних затримок; інтерфейс має бути адаптивним, інтуїтивно зрозумілим та

	візуально цілісним; структура компонентів інтерфейсу повинна забезпечувати швидке оновлення стану; система має підтримувати одночасну роботу багатьох користувачів; інтерфейс повинен відображати системні повідомлення, що стосуються змін стану комунікації
Технічні вимоги	Система повинна підтримувати двонапрямну взаємодію між клієнтом та сервером; база даних має забезпечувати швидку обробку документів та підтримувати динамічну структуру даних; обмеження на максимальну кількість кімнат та повідомлень мають бути встановлені конфігураційно; механізм тимчасової ідентифікації повинен формувати унікальні ідентифікатори стосовно кімнати; сервер повинен коректно обробляти підключення та відключення користувачів; канали взаємодії повинні бути стабільними та витривалими до розривів з'єднання
Організаційні вимоги	Система повинна бути придатною для запуску у локальному середовищі та у мережевих конфігураціях; процес розгортання повинен включати встановлення залежностей, запуск серверної частини та запуск клієнтської частини; тестування має виконуватися за допомогою спеціалізованих інструментів; перегляд стану бази даних повинен здійснюватися за допомогою візуальних засобів; система повинна підтримувати можливість подальшого розширення

Сукупність визначених вимог створює формальний опис властивостей, яким повинна відповідати веб-система анонімної текстової комунікації. Вони охоплюють

як функціональну поведінку системи, так і характеристики її продуктивності, надійності, організації даних та загальної взаємодії користувача з інтерфейсом. Завдяки структурованому підходу до формування вимог забезпечується можливість подальшого проектування архітектури програмного забезпечення, розроблення його логічної структури та побудови компонентів, здатних підтримувати інтерактивну комунікацію у режимі реального часу [11, 29, 36].

1.5. Висновки до розділу 1

Проведений аналіз предметної області дав змогу окреслити особливості анонімної комунікації та визначити ключові характеристики, яким повинно відповідати програмне забезпечення для підтримки швидкої взаємодії користувачів у режимі реального часу. Дослідження існуючих аналогів засвідчило відсутність рішень, що поєднують механізми анонімності, динамічне створення комунікаційних середовищ та синхронний обмін повідомленнями в інтерактивній формі, що підкреслює доцільність розроблення нового проекту. Аналіз технологічних можливостей дав підстави визначити інструменти та середовища, здатні забезпечити необхідні властивості системи, а формування вимог дозволило структурувати очікувану функціональність й технічні параметри майбутнього програмного забезпечення. Отримані результати створюють концептуальну основу для подальшого проектування архітектури та моделювання структурних компонентів.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Проектування програмного забезпечення охоплює побудову логічної та структурної основи системи, визначення її архітектурних компонентів, опис взаємодії між ними та формалізацію процесів, що забезпечують виконання функціональних вимог. На цьому етапі встановлюються моделі поведінки, структури даних та механізми оброблення подій, що дозволяє сформувати цілісну концепцію роботи системи та забезпечує узгодженість її складових. Проектування створює фундамент для подальшої реалізації, забезпечуючи коректність, масштабованість і передбачуваність функціонування програмного забезпечення.

2.1. Діаграма прецедентів (use case diagram)

Діаграма прецедентів відображає загальну модель взаємодії між користувачем та програмним забезпеченням, а також внутрішні дії системи, що виконуються у відповідь на ініційовані запити [3, 8]. У межах даної моделі виділено два основних актори: користувач як зовнішній учасник процесу комунікації та система як внутрішній виконавець логіки оброблення подій. Такий підхід дозволяє чітко розмежувати дії, що ініціюються безпосередньо користувачем, і процеси, які автоматично виконуються системою для забезпечення коректного функціонування комунікаційного середовища.

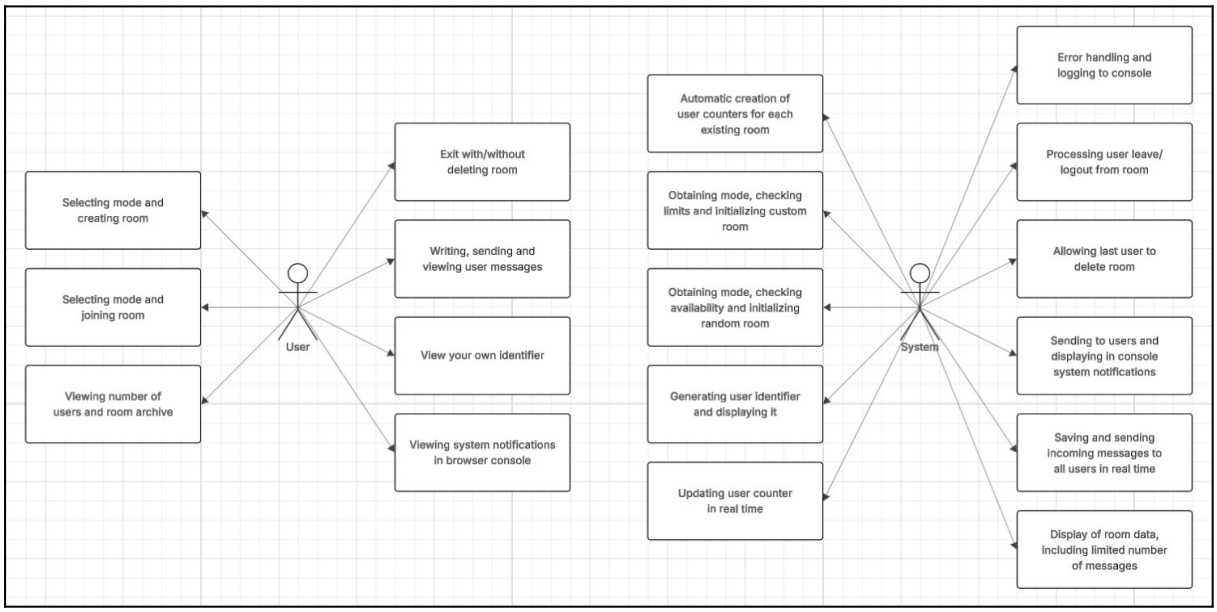


Рис. 1. Діаграма прецедентів (use case diagram)

Користувач взаємодіє із системою через обмежений набір дій, спрямованих на організацію та участь у текстовій комунікації. Початковим етапом взаємодії є вибір режиму роботи, після чого можливе створення нового комунікаційного середовища або підключення до вже існуючого. У межах активної сесії користувач має можливість переглядати кількість активних учасників, отримувати доступ до архіву повідомлень, а також здійснювати безпосередній обмін текстовими повідомленнями. Важливою складовою взаємодії є можливість перегляду власного тимчасового ідентифікатора, який використовується виключно в межах поточної комунікаційної сесії та не пов’язується з персональними даними. Окрім цього, користувач отримує системні повідомлення, що відображаються у середовищі браузера та інформують про події, пов’язані зі змінами стану кімнати або діями інших учасників.

Завершення взаємодії з комунікаційним середовищем здійснюється шляхом виходу з кімнати, який може відбуватися як із збереженням середовища, так і з його видаленням за умови виконання встановлених обмежень. Така модель дозволяє забезпечити гнучке керування життєвим циклом комунікаційних просторів без необхідності централізованого адміністрування або втручання сторонніх учасників.

Система, у свою чергу, виконує низку внутрішніх процесів, які не потребують безпосередньої участі користувача, але є критично важливими для стабільної роботи програмного забезпечення. До таких процесів належить автоматичне створення лічильників користувачів для кожного комунікаційного середовища, перевірка обмежень під час створення або підключення до кімнати, а також ініціалізація відповідного режиму взаємодії. Система забезпечує формування тимчасового ідентифікатора користувача та його відображення у клієнтському середовищі, що дозволяє розрізняти повідомлення учасників без порушення принципів анонімності.

Під час активної комунікації система відповідає за оновлення кількості учасників у режимі реального часу, збереження повідомлень та їх розповсюдження

між всіма підключеними користувачами. Окрему роль відіграє оброблення подій виходу користувача з кімнати або завершення сесії, що супроводжується відповідним коригуванням внутрішніх лічильників і формуванням системних повідомлень. У разі виконання умов, за яких видалення кімнати є допустимим, система надає можливість завершити існування комунікаційного середовища та очистити пов’язані з ним дані.

Діаграма прецедентів у такому вигляді дозволяє узагальнено відобразити основні сценарії взаємодії та визначити межі відповідальності між користувачем і системою. Вона формує концептуальну основу для подальшого деталізованого моделювання процесів, поведінки компонентів та потоків даних, що забезпечують функціонування програмного забезпечення [3, 8].

2.2. Діаграма діяльності (activity diagram)

Діаграма діяльності відображає послідовність дій та умов переходів, що визначають динаміку роботи програмного забезпечення від моменту ініціалізації до завершення взаємодії користувача з комунікаційним середовищем [3, 8]. Модель охоплює як дії, що виконуються автоматично системою, так і етапи, які ініціюються користувачем у процесі вибору режиму, підключення до кімнати та обміну повідомленнями. Такий підхід дозволяє формалізувати загальний сценарій функціонування системи з урахуванням альтернативних гілок й умов виконання.

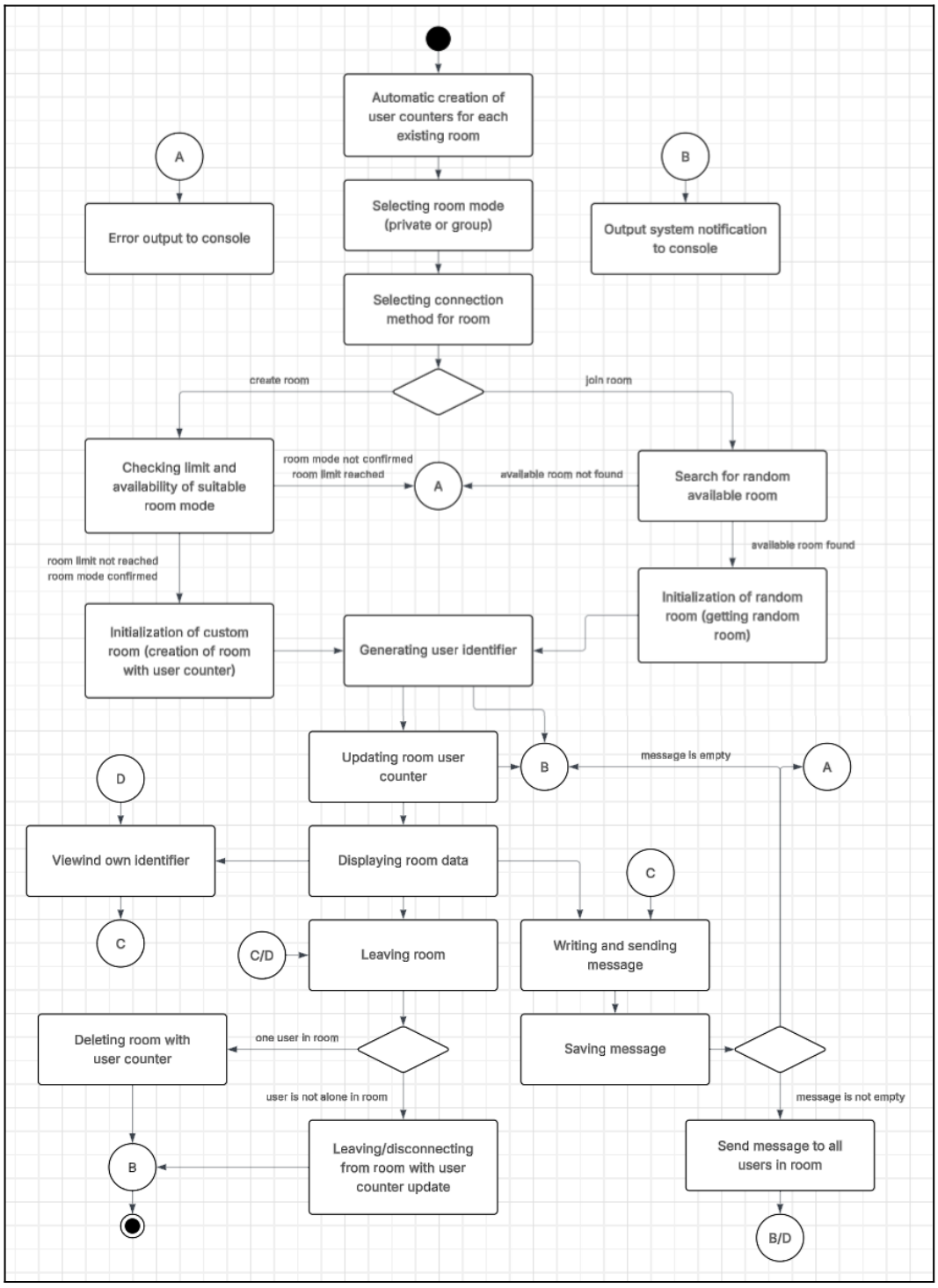


Рис. 2. Діаграма діяльності (activity diagram)

Початковий стан процесу пов’язаний з підготовкою внутрішнього середовища, зокрема з автоматичним формуванням лічильників користувачів для всіх наявних комунікаційних просторів. Після цього відбувається вибір режиму взаємодії, що визначає подальший характер комунікації, а також спосіб

підключення до кімнати. На цьому етапі формується точка розгалуження, яка розділяє сценарій створення нового комунікаційного середовища та сценарій підключення до вже існуючого.

Під час ініціації custom-кімнати здійснюється перевірка відповідності обраного режиму встановленим обмеженням і доступності такого типу середовища. За умови виконання всіх критеріїв відбувається ініціалізація нового комунікаційного простору з одночасним створенням лічильника користувачів. Якщо ж обмеження не дотримані, процес переривається з відповідним переходом до оброблення помилкової ситуації. Альтернативна гілка передбачає пошук доступного комунікаційного середовища випадковим чином, де у разі відсутності відповідних варіантів також ініціюється завершення поточного сценарію.

Після успішного створення або підключення до кімнати виконується формування тимчасового ідентифікатора користувача, що забезпечує його розпізнавання в межах активної сесії без використання персональних даних. Надалі відбувається оновлення лічильника учасників та відображення інформації про комунікаційне середовище, включно з доступними повідомленнями. На цьому етапі користувач може переглядати власний ідентифікатор, що не впливає на загальний перебіг процесу, але являється складовою інформаційної взаємодії.

Подальша діяльність зосереджується на обміні повідомленнями, який включає введення тексту, перевірку його коректності та збереження в системі. За умови, що повідомлення містить змістовні дані, воно передається всім активним учасникам комунікаційного середовища, що забезпечує синхронність взаємодії. Якщо ж повідомлення не відповідає встановленим умовам, процес переходить до альтернативної гілки без виконання операції розповсюдження.

Завершальним етапом діяльності є вихід користувача з комунікаційного середовища. На цьому етапі відбувається перевірка кількості активних учасників, що визначає подальший сценарій. У разі, якщо користувач являється єдиним учасником — ініціюється видалення кімнати разом із пов’язаними з нею даними. Якщо ж у середовищі залишаються інші учасники, то виконується коректне

завершення сесії з оновленням відповідного лічильника. Після цього процес переходить у фінальний стан, що означає завершення взаємодії користувача з програмним забезпеченням.

2.3. Діаграма послідовності (sequence diagram)

Діаграми послідовності відображають часову впорядкованість обміну повідомленнями між основними логічними учасниками системи та дозволяють формалізувати сценарії взаємодії, що охоплюють ініціалізацію комунікаційного середовища, участь у ньому, обмін повідомленнями та завершення сесії [3, 8]. У межах моделі розглядається чотиристороння взаємодія між користувачем, клієнтським інтерфейсом, серверною частиною та сховищем даних, де кожен елемент виконує чітко визначену роль у загальному процесі.

Перед виконанням користувацьких сценаріїв передбачається ініціалізаційний етап, під час якого серверна частина синхронізує внутрішній стан із даними сховища. На цьому початковому етапі здійснюється отримання переліку наявних комунікаційних середовищ із бази даних, після чого для кожного з них формується відповідний лічильник активних користувачів, що зберігається у внутрішньому середовищі виконання. Така попередня підготовка забезпечує коректне відстеження кількості учасників у кожній кімнаті одразу після встановлення з’єднань і дозволяє виконувати подальші операції підключення, виходу та синхронізації стану без додаткових запитів на створення лічильників під час активної взаємодії.

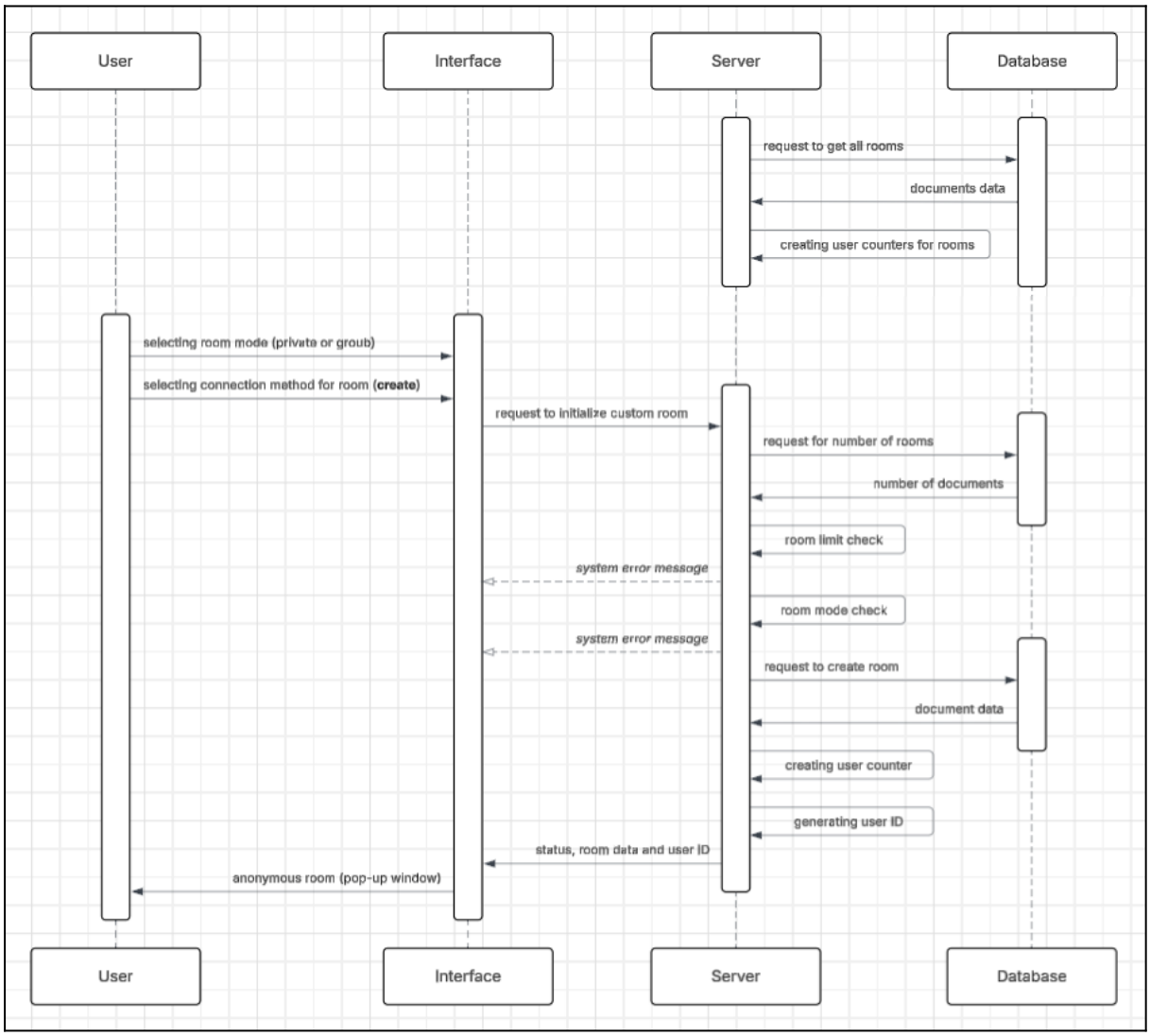


Рис. 3. Діаграма послідовності (sequence diagram). Нульовий етап та перший сценарій взаємодії з системою

Перший сценарій описує послідовність дій під час створення нового комунікаційного середовища. Користувач ініціює вибір режиму взаємодії та спосіб підключення, після чого відповідний запит передається через клієнтський інтерфейс до серверної частини. Сервер, у свою чергу, взаємодіє зі сховищем даних для отримання інформації про наявні комунікаційні простори та перевірки встановлених обмежень. За умови коректності параметрів здійснюється створення нового середовища, ініціалізація внутрішнього лічильника та формування тимчасового ідентифікатора користувача, після чого узагальнена інформація про стан системи передається до інтерфейсу та відображається користувачеві.

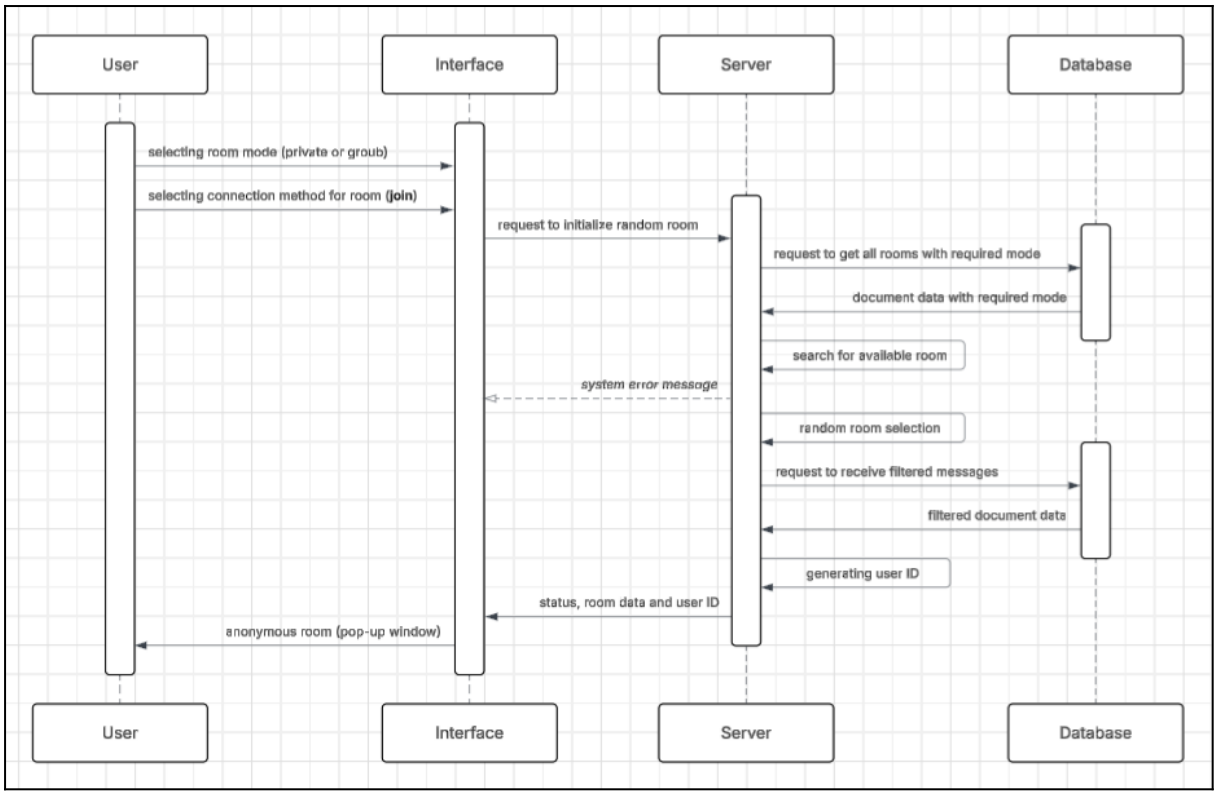


Рис. 4. Діаграма послідовності (sequence diagram). Другий сценарій взаємодії з системою

Другий сценарій відображає послідовність дій, пов’язаних із підключенням до випадкового доступного комунікаційного середовища. Після вибору режиму взаємодії — інтерфейс ініціює відповідний запит до серверної частини, який здійснює вибірку з бази даних комунікаційних просторів із заданими параметрами. Далі виконується пошук доступного середовища та відбір повідомлень, що підлягають відображенню відповідно до встановлених обмежень. Паралельно формується тимчасовий ідентифікатор користувача, після чого сервер передає до інтерфейсу інформацію про підключене середовище та поточний стан взаємодії.

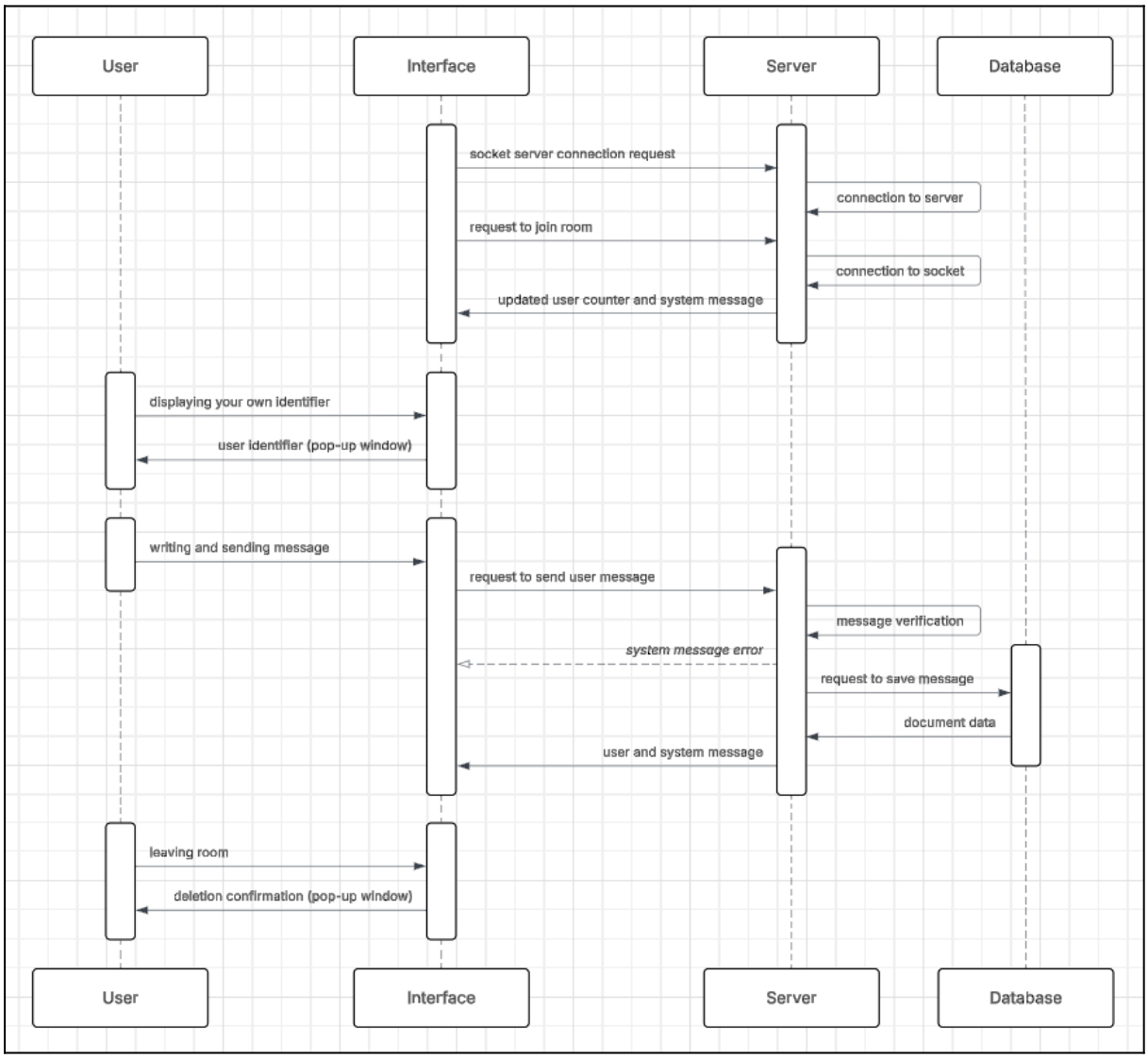


Рис. 5. Діаграма послідовності (sequence diagram). Третій сценарій взаємодії з системою

Третій сценарій описує процес безпосередньої участі користувача в комунікаційному середовищі, включно з підключенням до каналу реального часу, переглядом власного ідентифікатора та обміном повідомленнями. Клієнтський інтерфейс ініціює встановлення з’єднання з сервером, після чого виконується приєднання до відповідного комунікаційного простору. Серверна частина забезпечує оновлення внутрішнього лічильника та надсилання системних повідомлень іншим учасникам. У процесі обміну повідомленнями введений користувачем текст передається на сервер для перевірки та збереження у сховищі даних, після чого синхронно розповсюджується між усіма активними учасниками середовища.

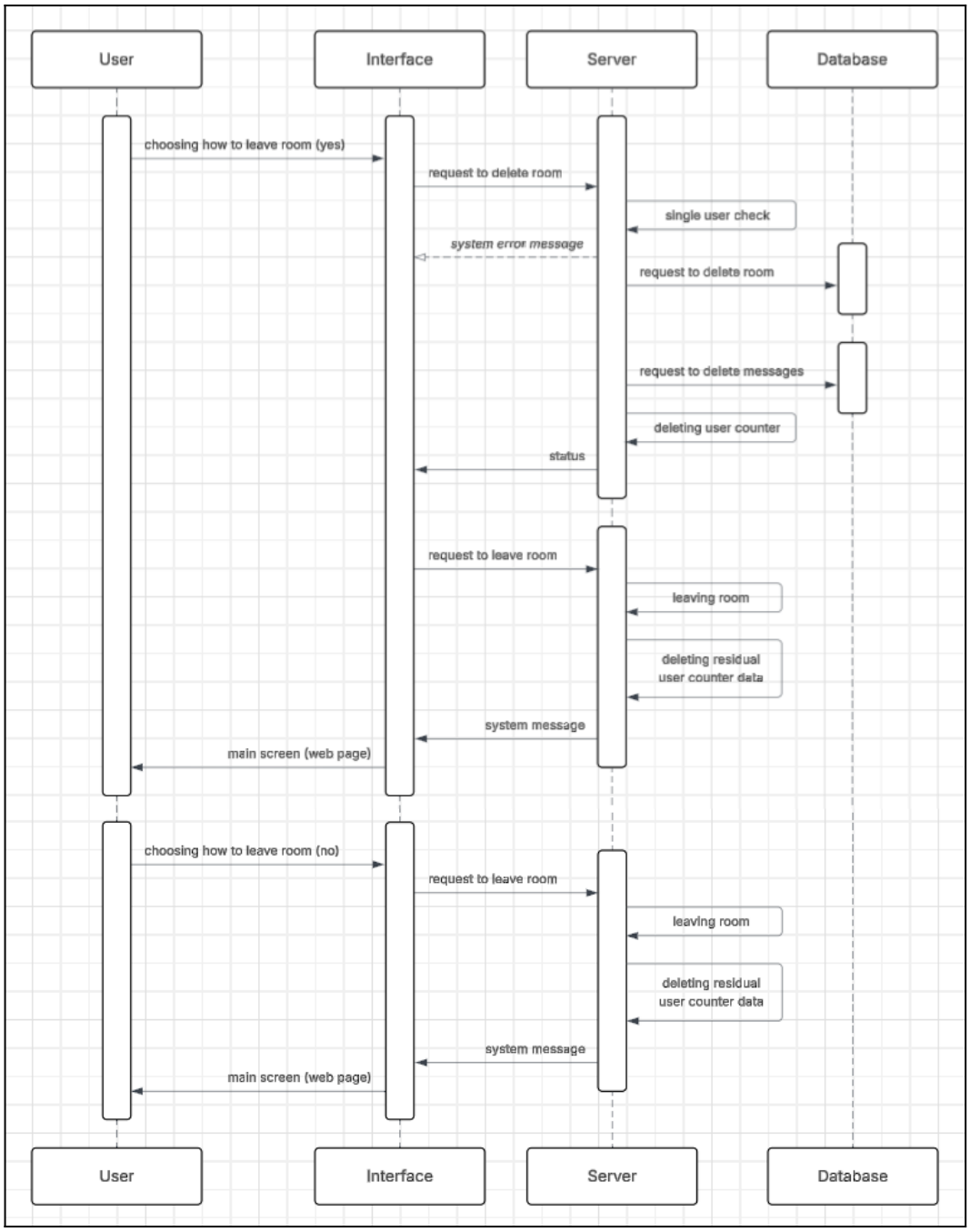


Рис. 6. Діаграма послідовності (sequence diagram). Четвертий сценарій взаємодії з системою

Четвертий сценарій відображає завершення взаємодії користувача з комунікаційним середовищем та охоплює альтернативні варіанти виходу. Користувач через інтерфейс визначає спосіб завершення сесії, після чого відповідні запити передаються до серверної частини. Сервер перевіряє умови, за яких можливе видалення комунікаційного середовища, та взаємодіє зі сховищем даних для

очищення пов’язаних записів. У разі стандартного виходу здійснюється коректне завершення з’єднання з оновленням внутрішнього лічильника та передачею системних повідомлень. Після завершення обраного сценарію інтерфейс повертає користувача до початкового стану взаємодії.

Сукупність наведених вище діаграм послідовності дозволяє цілісно описати часову логіку взаємодії між основними компонентами системи та формалізувати ключові сценарії роботи програмного забезпечення, що забезпечують коректну організацію анонімної текстової комунікації.

2.4. Архітектура інформаційної системи (information system architecture)

Архітектура інформаційної системи відображає структурну організацію програмного забезпечення та взаємодію його основних компонентів, що забезпечують функціонування анонімної текстової комунікації у веб-середовищі [2, 29, 36]. Модель архітектури побудована за принципом чіткого розмежування клієнтського інтерфейсу, серверної логіки та рівня зберігання даних, що дозволяє забезпечити масштабованість, логічну ізоляцію відповідальностей і стабільну роботу системи за умов паралельної взаємодії користувачів.

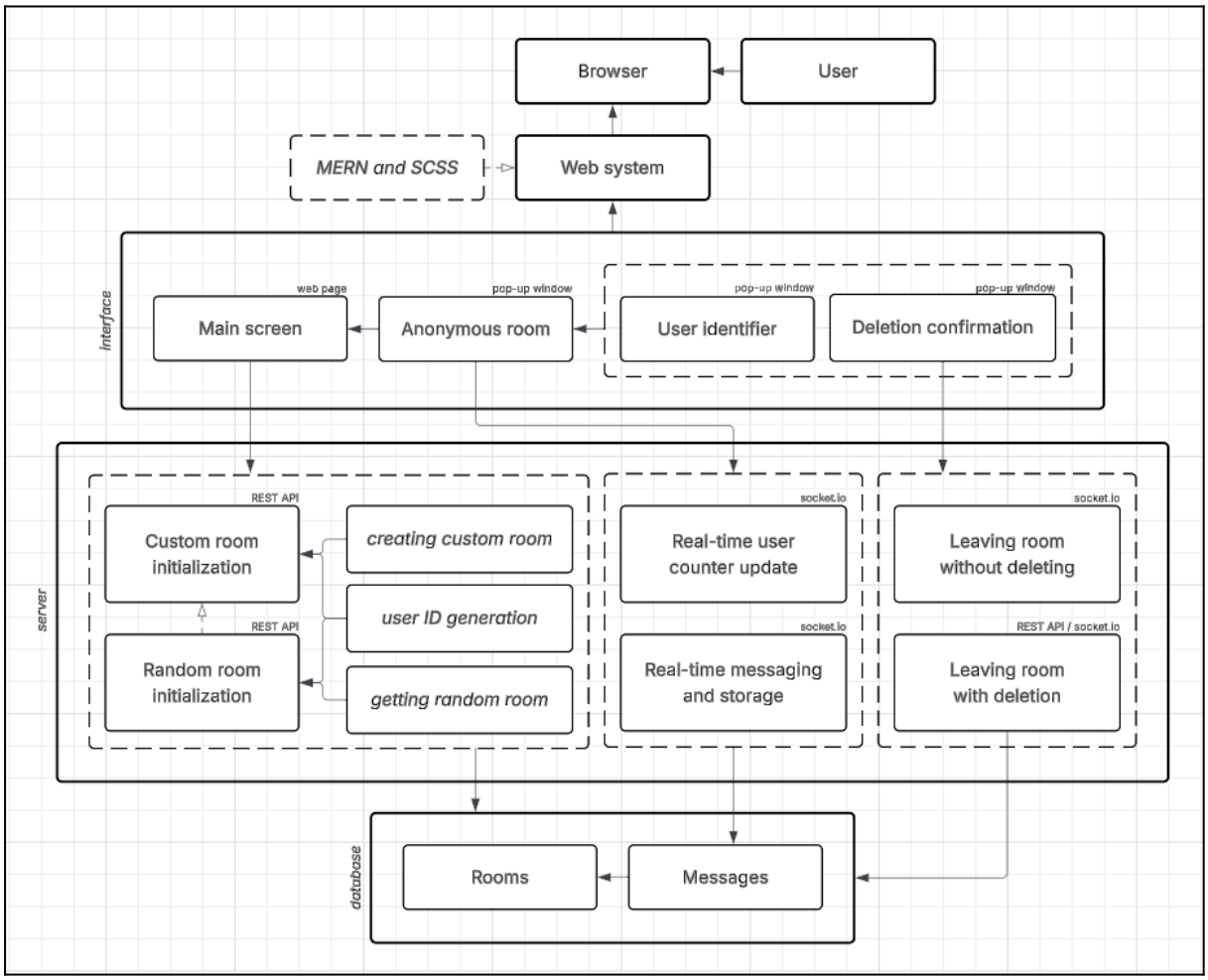


Рис. 7. Архітектура інформаційної системи (information system architecture)

Зовнішній рівень архітектури представлений користувачем, який взаємодіє з програмним забезпеченням через веб-браузер. Браузер виконує роль середовища виконання клієнтської частини та забезпечує доступ до інтерфейсу системи. Веб-система, що функціонує в межах браузера, реалізує клієнтський рівень архітектури та відповідає за відображення стану комунікаційних середовищ, оброблення користувацьких дій та ініціацію запитів до серверної частини. Клієнтський інтерфейс логічно поділено на основний екран, який виконує роль початкової точки взаємодії, та модальні компоненти, що використовуються для відображення додаткової інформації, зокрема тимчасового ідентифікатора користувача та підтвердження дій, пов’язаних із завершенням взаємодії.

Серверний рівень архітектури зосереджує в собі основну бізнес-логіку системи та забезпечує координацію між клієнтським інтерфейсом і сховищем даних.

У межах даного рівня виокремлено функціональні підсистеми, відповідальні за ініціалізацію комунікаційних середовищ, генерацію тимчасових ідентифікаторів, керування підключеннями користувачів та підтримку обміну повідомленнями в режимі реального часу. Частина взаємодій між клієнтською та серверною складовими реалізується за запитно-відповідною моделлю для виконання операцій ініціалізації, тоді як синхронізація повідомлень і стану учасників здійснюється через постійний двонапрямний канал зв’язку.

Окреме місце в архітектурі займає механізм оброблення виходу користувачів із комунікаційного середовища, який підтримує альтернативні сценарії завершення взаємодії. Серверна частина координує як стандартне відключення користувача з коректним оновленням внутрішнього стану, так і повне видалення комунікаційного середовища разом із пов’язаними даними за умови виконання встановлених обмежень. Такий підхід дозволяє уникнути накопичення неактуальних даних і підтримувати узгодженість стану системи.

Рівень зберігання даних представлений сховищем, у якому окремо зберігається інформація про комунікаційні середовища та текстові повідомлення. Архітектурне розділення цих сутностей дозволяє оптимізувати доступ до даних і забезпечити гнучке керування життєвим циклом інформації. Взаємодія серверної частини зі сховищем даних виконується лише в межах чітко визначених операцій, що сприяє збереженню цілісності даних та спрощує контроль за станом системи.

Загалом архітектура інформаційної системи формує цілісну модель взаємодії між користувачем, клієнтським інтерфейсом, серверною логікою та рівнем зберігання даних, забезпечуючи узгоджену роботу компонентів та створюючи стабільну основу для реалізації анонімної текстової комунікації у веб-середовищі.

2.5. Діаграма потоків даних (data flow diagram)

Діаграма потоків даних відображає рух інформації між основними процесами системи та дозволяє формалізувати логіку передавання даних у межах ключових сценаріїв функціонування програмного забезпечення [29, 36]. В даній моделі акцент зроблено на інформаційних потоках, що виникають під час ініціалізації

комунікаційного середовища, обміну повідомленнями в режимі реального часу та завершення взаємодії користувача з кімнатою, включно з варіантами її видалення.

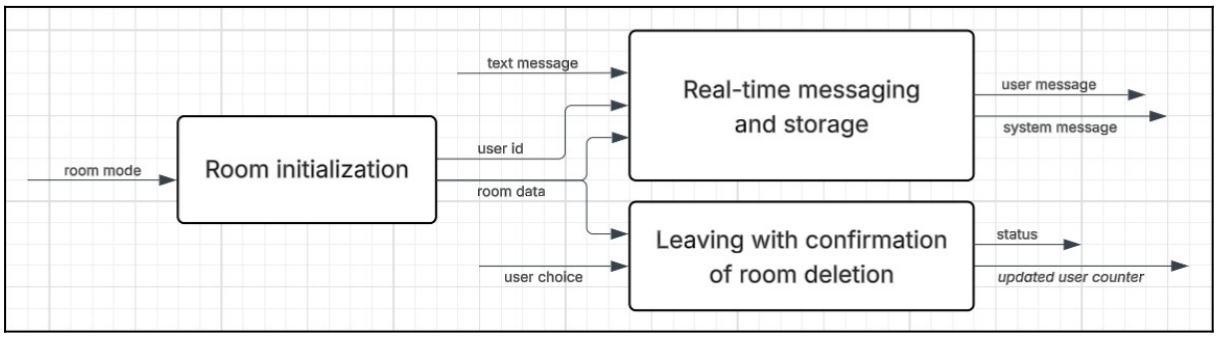


Рис. 8. Діаграма потоків даних (data flow diagram)

Початковим елементом моделі являється процес ініціалізації комунікаційного середовища, до якого надходять дані про обраний режим взаємодії. На основі цього вхідного потоку формується сукупність даних, що описують параметри кімнати та пов’язаний із нею тимчасовий ідентифікатор користувача. Сформовані дані передаються до наступних процесів системи та виступають основою для подальшої взаємодії в межах активної сесії.

Центральним елементом діаграми є процес обміну повідомленнями та їх зберігання, який обробляє вхідні текстові повідомлення разом із ідентифікаційними даними користувача та параметрами комунікаційного середовища. У результаті виконання цього процесу формуються два типи вихідних потоків: повідомлення, створені користувачами, та системні повідомлення, що відображають зміни стану комунікаційного середовища. Обидва типи даних передаються для подальшого розповсюдження між учасниками та підтримують синхронність взаємодії в режимі реального часу.

Окремий інформаційний потік формується під час завершення взаємодії користувача з кімнатою. В даному випадку вхідними даними виступають як вибір користувача щодо способу виходу, так і дані комунікаційного середовища, необхідні для коректного визначення подальшого сценарію. На основі цих даних активується процес підтвердження видалення комунікаційного середовища, за

результатами якого передається статусна інформація, а також, за потреби, оновлені дані лічильника користувачів, що характерно для сценарію виходу без видалення кімнати. Такий підхід забезпечує коректне відображення поточного стану середовища та підтримує узгодженість внутрішніх даних під час завершення сесії.

Загалом діаграма потоків даних узагальнює логіку передавання інформації між ключовими процесами системи та демонструє, яким чином дані трансформуються і розповсюджуються в межах анонімної текстової комунікації. Така модель дозволяє чітко визначити залежності між процесами та забезпечує цілісне розуміння інформаційних потоків, що лежать в основі функціонування програмного забезпечення [29, 36].

2.6. Діаграма сутностей та зв’язків (entity–relationship diagram)

Діаграма сутностей та зв’язків відображає логічну модель структури даних програмного забезпечення та визначає основні інформаційні сутності, їх атрибути й характер взаємозв’язків між ними. Модель зосереджена на описі даних, необхідних для функціонування анонімної текстової комунікації, та забезпечує формалізоване уявлення про спосіб зберігання і організації інформації в межах системи [33].

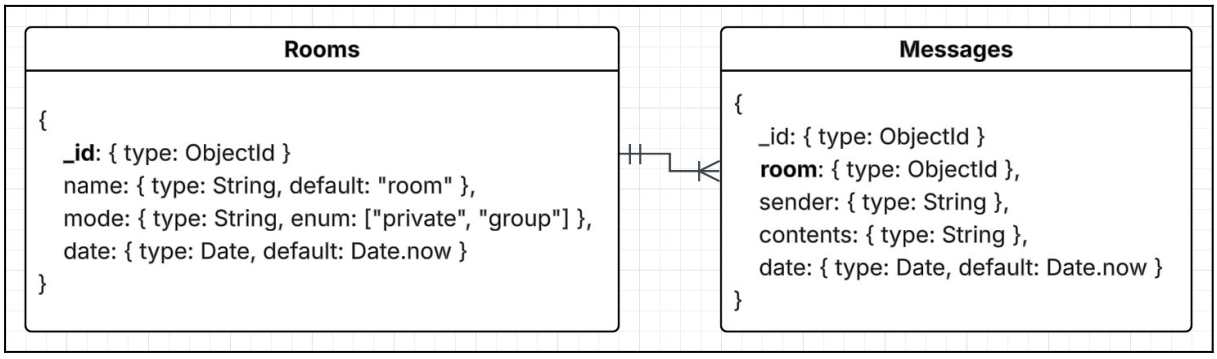


Рис. 9. Діаграма сутностей та зв’язків (entity–relationship diagram)

Центральною сутністю моделі являється комунікаційне середовище, яке репрезентує окрему кімнату для взаємодії користувачів. Дана сутність містить унікальний ідентифікатор, текстову назву, параметр режиму взаємодії та часову мітку створення. Сукупність атрибутів дозволяє однозначно ідентифікувати

комунікаційне середовище, визначити його тип та відстежувати момент ініціалізації, що є важливим для керування життєвим циклом кімнат.

Пов’язаною з описаною вище сутністю є повідомлення, які представляють одиниці текстової взаємодії між учасниками. Кожне повідомлення має власний унікальний ідентифікатор, текстовий вміст, часову мітку створення та ідентифікатор відправника, який використовується для розрізнення учасників у межах конкретної комунікаційної сесії. Окремим атрибутом повідомлення є посилання на відповідне комунікаційне середовище, що встановлює логічний зв’язок між повідомленням і кімнатою, в якій воно було створене.

Між сутностями комунікаційного середовища та повідомлення встановлено зв’язок типу “один до багатьох”, що означає можливість існування множини повідомлень, пов’язаних з однією кімнатою. Такий характер зв’язку відображає реальну модель взаємодії, за якої кожне повідомлення належить лише одному комунікаційному середовищу, тоді як кожна кімната може містити довільну кількість повідомлень у межах встановлених обмежень, що забезпечує цілісність даних та спрощує виконання операцій вибірки, збереження та видалення інформації.

Загалом діаграма сутностей та зв’язків формує логічну основу для реалізації рівня зберігання даних та визначає чіткі залежності між основними інформаційними об’єктами системи. Така модель сприяє узгодженості структури даних із логікою функціонування програмного забезпечення та забезпечує надійну основу для подальшого проектування та реалізації сховища інформації [33].

2.7. Прототип користувацького інтерфейсу (user interface prototype)

Прототип користувацького інтерфейсу відображає логічну організацію візуальних елементів та сценарії взаємодії користувача з програмним забезпеченням у межах веб-середовища [3, 8]. Модель інтерфейсу побудована з урахуванням принципів мінімалізму, однозначності дій та зниження когнітивного навантаження, що є особливо важливим для систем анонімної комунікації, де користувач повинен мати можливість розпочати взаємодію без попередньої підготовки або реєстрації.

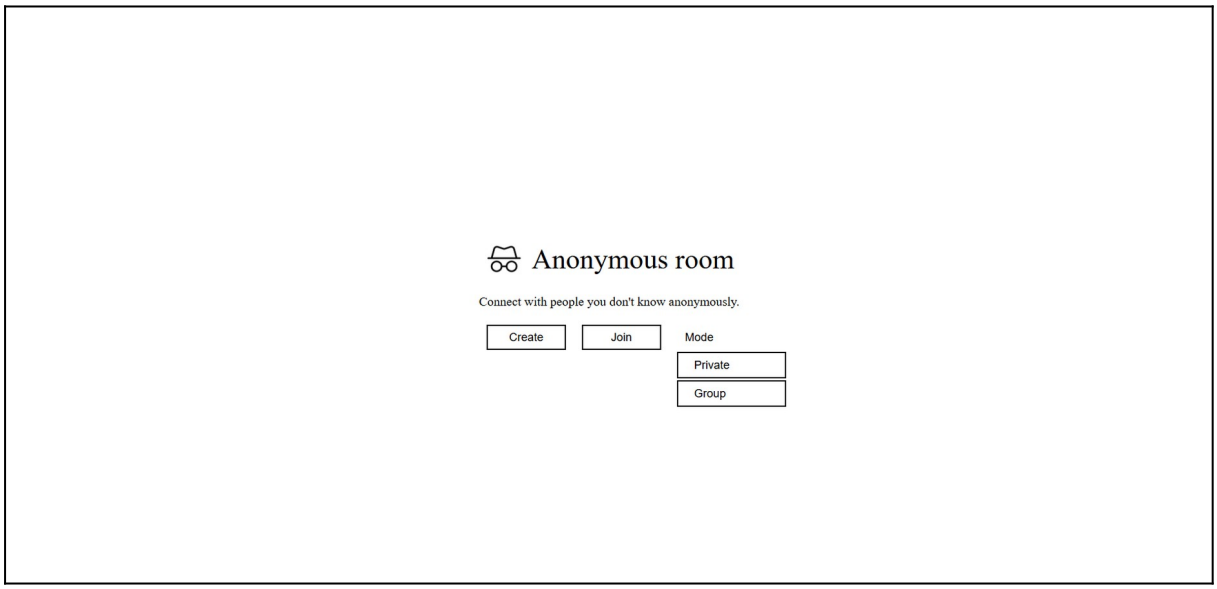


Рис. 10. Прототип користувацького інтерфейсу (user interface prototype). Головний екран

Початковий стан інтерфейсу представлений головним екраном, який виконує роль центральної точки входу до системи. На цьому етапі користувачеві надається можливість вибору режиму взаємодії та способу підключення до комунікаційного середовища. Елементи керування згруповані таким чином, щоб забезпечити логічну послідовність дій та виключити неоднозначність вибору. Візуальна структура головного екрана не перевантажена другорядною інформацією, що дозволяє зосередитися виключно на ініціалізації комунікації.

Після переходу до активного комунікаційного середовища інтерфейс трансформується відповідно до нового контексту взаємодії. У верхній частині розміщується інформаційний блок, який містить дані про комунікаційне середовище, часові параметри його створення та кількість активних учасників. Така організація дозволяє користувачеві постійно орієнтуватися в поточному стані взаємодії без необхідності виконання додаткових дій.

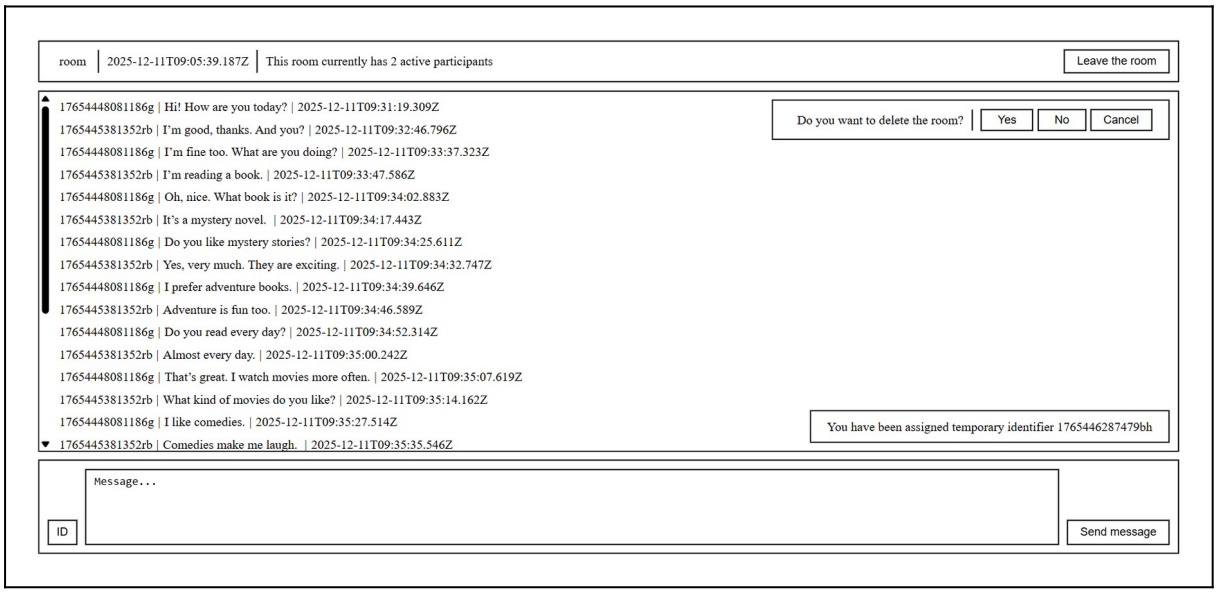


Рис. 11. Прототип користувацького інтерфейсу (user interface prototype). Анонімна кімната

Центральну частину інтерфейсу займає область відображення повідомлень, де текстова інформація подається у хронологічному порядку з чітким розмежуванням повідомлень різних учасників. Кожне повідомлення супроводжується ідентифікаційними та часовими атрибутами, що забезпечує зрозумілість діалогу та дозволяє відстежувати послідовність обміну інформацією. Інтерфейс не передбачає використання візуальних маркерів, пов’язаних з особистістю користувача, що узгоджується з концепцією анонімної взаємодії.

У нижній частині інтерфейсу розташовано елементи для введення та надсилання текстових повідомлень, а також елемент доступу до відображення тимчасового ідентифікатора користувача. Така організація підтримує безперервність комунікації та дозволяє виконувати основні дії без зміни контексту взаємодії. Відображення ідентифікатора реалізовано у вигляді окремого інформаційного блоку, що з’являється лише за ініціативою користувача, тим самим не перевантажуючи основний інтерфейс.

Окремим елементом інтерфейсу являється механізм завершення взаємодії з комунікаційним середовищем, який реалізовано через модальне вікно підтвердження. Такий підхід дозволяє чітко розмежувати дії, пов’язані зі звичайним виходом, та дії, що можуть призвести до видалення комунікаційного середовища.

Модальне вікно фокусує увагу користувача на прийнятті рішення та зменшує ймовірність випадкового завершення взаємодії.

Загалом прототип користувацького інтерфейсу формує цілісну модель взаємодії, у якій кожен елемент підпорядкований єдиній логіці роботи системи [3, 8]. Така структура забезпечує інтуїтивну навігацію, підтримує принципи анонімності та створює зручні умови для реалізації текстової комунікації в режимі реального часу.

2.8. Файлова структура (file structure)

Файлова структура інформаційної системи побудована за принципом чіткого розмежування клієнтської та серверної частин, що відповідає архітектурі веб-орієнтованих застосунків із розподіленою логікою. Такий підхід забезпечує зрозумілу організацію коду, спрощує супровід та масштабування системи, а також дозволяє ізолювати відповідальності між різними рівнями обробки даних і взаємодії з користувачем [3]. Загальна структура представлена у вигляді кореневого каталогу, в межах якого виділено два основні підкаталоги, що відповідають серверній та клієнтській частинам системи.

Рис. 12. Файлова структура (file structure). Серверна частина системи

Серверна частина організована у вигляді логічно згрупованих директорій, кожна з яких відповідає окремому аспекту обробки запитів та управління даними. Виділення шару конфігурацій дозволяє централізовано зберігати параметри середовища виконання, не змішуючи їх із прикладною логікою. Модуль моделей відповідає за опис структури даних, що зберігаються у базі даних, та забезпечує узгодженість між логічним представленням інформації та її фізичним зберіганням. Окремий рівень сервісів концентрує бізнес-логіку, пов’язану з ініціалізацією комунікаційних середовищ, керуванням лічильниками користувачів, генерацією тимчасових ідентифікаторів та обробкою повідомлень. Маршрутизація запитів реалізована у виділеному модулі, що забезпечує чітке розмежування між точками доступу та їх внутрішньою реалізацією. Додатково виділено компонент, відповідальний за обробку подій реального часу, що підкреслює асинхронний характер взаємодії у системі.

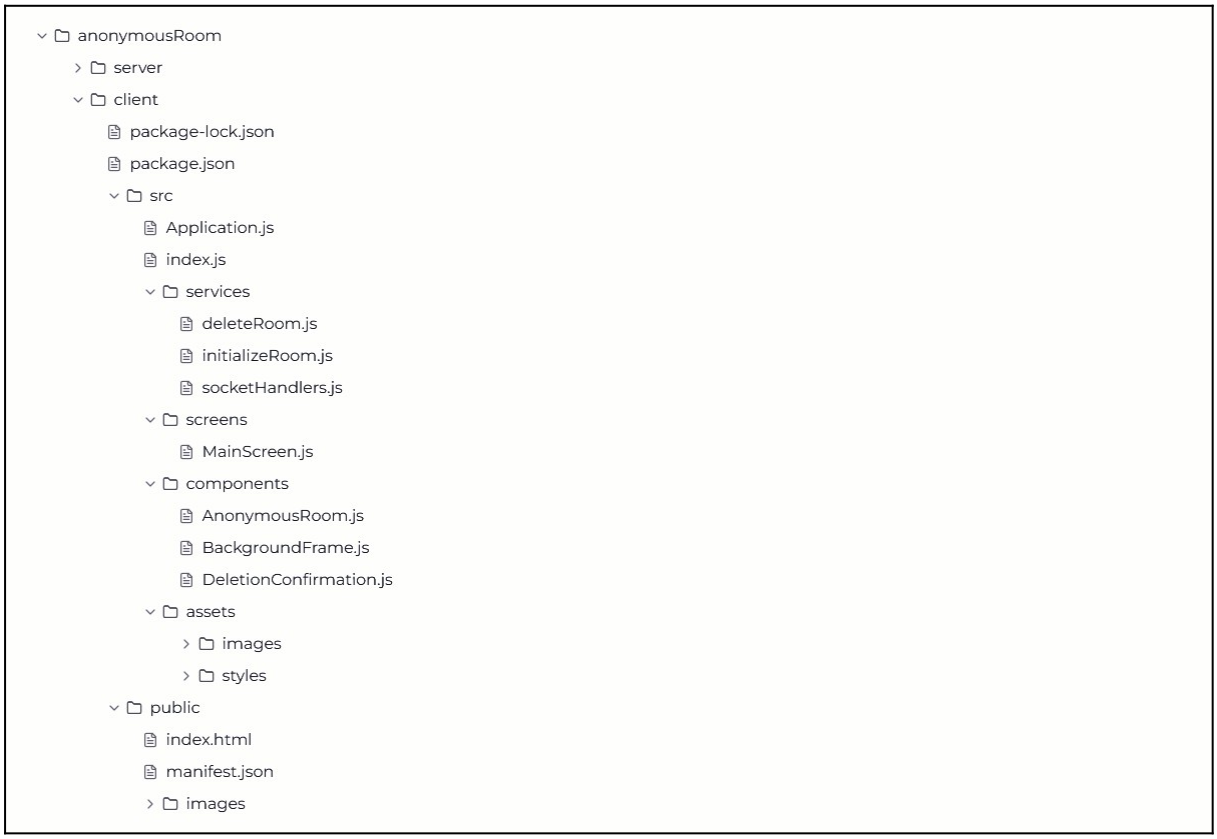


Рис. 13. Файлова структура (file structure). Клієнтська частина системи

Клієнтська частина має структуру, орієнтовану на компонентний підхід та розділення логіки представлення [8]. Основний вхідний файл забезпечує ініціалізацію застосунку та підключення кореневого інтерфейсу, тоді як окремі директорії призначені для екранів, повторно використовуваних компонентів та сервісів взаємодії з серверною частиною. Така організація дозволяє ізолювати логіку інтерфейсу від мережевих операцій та спрощує подальше розширення функціональних можливостей. Статичні ресурси, зокрема стилі та зображення, винесені в окремий каталог, що сприяє підтримці єдиного візуального стилю та впорядкованості файлової системи. Публічна частина клієнтського застосунку містить базові файли, необхідні для початкового завантаження у браузері, що відповідає загальноприйнятій практиці побудови сучасних веб-інтерфейсів.

Загалом запропонована файлова структура відображає архітектурні рішення, прийняті під час проектування інформаційної системи, та забезпечує логічну цілісність, зрозумілість і зручність роботи з кодовою базою. Вона створює основу для подальшого розвитку програмного забезпечення без порушення вже реалізованих механізмів взаємодії між його складовими [3].

2.9. Висновки до розділу 2

Проектування програмного забезпечення виконано з позиції системного підходу, що передбачає формалізацію ключових процесів, структур і взаємодій, необхідних для функціонування інформаційної системи анонімного текстового спілкування. Побудовані діаграми узгоджено відображають поведінку користувачів і системи, логіку виконання основних сценаріїв, послідовність обміну даними між компонентами, архітектурну організацію та модель зберігання інформації. Запропоновані проектні рішення формують цілісне уявлення про майбутню реалізацію програмного забезпечення та створюють концептуальну основу для

подальшого переходу до етапу реалізації з урахуванням визначених функціональних та інформаційних зв’язків.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Реалізація програмного забезпечення передбачає безпосереднє втілення розроблених моделей, алгоритмів і технічних рішень у вигляді узгоджених програмних компонентів, що забезпечують функціонування веб-системи для анонімної комунікації. На цьому етапі здійснюється практичне застосування обраних технологій, налаштування програмного середовища, побудова структури даних та логіки обробки запитів, а також організація взаємодії між клієнтською та серверною частинами з урахуванням вимог до надійності, безпеки та продуктивності для подальшого огляду й тестування готової системи.

3.1. Аналіз та налаштування конфігурації проекту

3.1.1. Конфігурація серверної сторони проекту

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "development": "nodemon index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "Poloviuk Oleh <poloviukoleh@gmail.com>",
  "license": "ISC",
  "dependencies": {
    "config": "^4.1.1",
    "cors": "^2.8.5",
    "express": "^5.1.0",
    "mongoose": "^9.0.0",
    "socket.io": "^4.8.1"
  },
  "devDependencies": {
    "nodemon": "^3.1.11"
  }
}
```

Файл конфігурації серверного середовища *package.json* визначає базові параметри програмного оточення та регламентує механізми виконання сервера. У

конфігурації зазначено ідентифікаційні характеристики програмного модуля, визначено стартовий файл та набір сценаріїв, які забезпечують запуск у стандартному та розробницькому режимах, а також формальне тестове виконання. Окремими блоками визначено перелік зовнішніх бібліотек, що забезпечують роботу з конфігураційними параметрами, міждоменною взаємодією, серверною інфраструктурою, системою зберігання даних та каналами обміну повідомленнями у режимі реального часу. Додатково виокремлено комплект інструментів, призначених для автоматизованого оновлення під час розроблення, що оптимізує процес зміни та перевірки серверної логіки [6, 23, 25, 35].

```
{
    "serverPort": 5000,
    "mongoDatabaseURI": "mongodb://localhost:27017/anonymousChat",
    "roomCreationLimit": 1000,
    "displayMessageLimit": 100,
    "minimumUsersToDeleteRoom": 1,
    "privateRoomUserLimit": 2,
    "groupRoomUserLimit": 9
}
```

Файл конфігурації серверного середовища *default.json* містить набір параметрів, що визначають вихідні умови функціонування веб-системи. У конфігурації встановлено мережевий порт, на якому здійснюється оброблення запитів, та адресу підключення до сховища даних, що забезпечує взаємодію з локальною інстанцією документно-орієнтованої бази. Окремі параметри регламентують кількісні обмеження щодо створення комунікаційних просторів, максимальної кількості відображуваних повідомлень, мінімальної чисельності учасників, за якої можливе видалення простору, а також граничні величини для приватних та колективних каналів взаємодії. Сукупність цих параметрів формує базові правила роботи серверної частини та забезпечує контроль за масштабуванням і поведінкою системи [22, 25].

3.1.2. Конфігурація клієнтської сторони проекту

```
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/dom": "^10.4.1",
    "@testing-library/jest-dom": "^6.8.0",
    "@testing-library/react": "^16.3.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^19.1.1",
    "react-dom": "^19.1.1",
    "react-scripts": "5.0.1",
    "socket.io-client": "^4.8.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Файл конфігурації клієнтського середовища *package.json* визначає характеристики програмного модуля, параметри його запуску та перелік інструментів, необхідних для функціонування інтерфейсної частини веб-системи. У

конфігурації зазначено ідентифікаційні атрибути клієнтського модуля та встановлено залежності, що забезпечують роботу інтерфейсного фреймворка, механізмів віртуалізації компонентів, засобів тестування, а також клієнтської частини каналу обміну даними у режимі реального часу. Окремо визначено набір сценаріїв для розгортання, компіляції, тестування та модифікації конфігурації середовища виконання. Додаткові параметри описують правила статичного аналізу коду та цільові групи браузерів для середовища промислової експлуатації й етапу розроблення, що забезпечує відповідність інтерфейсної частини вимогам сумісності та оптимальної продуктивності [30, 34].

```
{
  "name": "Anonymous room",
  "short_name": "A-Room",
  "icons": [
    { "src": "./images/logo_16x16.png", "type": "image/png", "sizes": "16x16" },
    { "src": "./images/logo_24x24.png", "type": "image/png", "sizes": "24x24" },
    { "src": "./images/logo_32x32.png", "type": "image/png", "sizes": "32x32" },
    { "src": "./images/logo_64x64.png", "type": "image/png", "sizes": "64x64" },
        { "src": "./images/logo_128x128.png", "type": "image/png", "sizes":
"128x128" },
        { "src": "./images/logo_256x256.png", "type": "image/png", "sizes":
"256x256" },
        { "src": "./images/logo_512x512.png", "type": "image/png", "sizes":
"512x512" }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#673ab7",
  "background_color": "#2196f3"
}
```

Файл *manifest.json* формує конфігурацію прогресивного веб-застосунку, визначаючи його зовнішній вигляд, поведінку та правила інтеграції з інтерфейсом кінцевого пристрою. У конфігурації задаються повна та скорочена назви застосунку, що використовуються для відображення у системних меню та на домашньому екрані. Перелік графічних ресурсів містить різні варіанти піктограм, забезпечуючи коректне масштабування та відповідність вимогам різних платформ і роздільних

здатностей. Додатково зазначено початкову адресу завантаження, режим відображення, домінантний колір інтерфейсу та фонове оформлення, що визначає загальний стиль застосунку та те, як він виглядатиме у режимі окремого вікна без елементів браузера. Сукупність цих параметрів забезпечує можливість установлення веб-застосунку на пристрої користувача, підвищує його інтегрованість зі середовищем операційної системи та створює передумови для роботи у форматі автономного інтерфейсного модуля [17, 30, 40].

3.2. Формування структури бази даних

```
const { Schema, model } = require("mongoose");

const schema = new Schema({
    name: { type: String, default: "room", required: true },
    mode: { type: String, enum: ["private", "group"], required: true },
    date: { type: Date, default: Date.now }
});

module.exports = model("Room", schema);
```

Модель Room описує структуру сутності, що використовується для збереження параметрів комунікаційних просторів у документно-орієнтованій базі даних. У схемі визначено атрибути, які характеризують найменування простору, тип взаємодії між користувачами та часову позначку його створення. Тип взаємодії обмежується наперед визначеним переліком режимів, що забезпечує контроль коректності збережених даних та унеможливлює використання невалідних значень. Часова характеристика формується автоматично під час створення об’єкта, що дає змогу фіксувати момент ініціації відповідного комунікаційного простору. Сукупність цих параметрів забезпечує узгоджене представлення сутності в системі та визначає її основні властивості під час подальшої обробки [22, 23, 24].

```
const { Schema, Types, model } = require("mongoose");

const schema = new Schema({
    room: { type: Types.ObjectId, ref: "Room" },
    sender: { type: String, required: true },
```

```
    contents: { type: String, required: true },
    date: { type: Date, default: Date.now }
});

module.exports = model("Message", schema);
```

Модель Message визначає структуру сутності, що використовується для зберігання текстових повідомлень у межах визначених комунікаційних просторів. У схемі передбачено атрибут, який формує зв’язок між повідомленням і відповідним простором взаємодії через посилання на унікальний ідентифікатор пов’язаної сутності. Додатково описано характеристики відправника, зміст повідомлення та часову позначку його створення. Часова характеристика встановлюється автоматично під час формування об’єкта, що дозволяє відстежувати хронологію комунікації. Така структура забезпечує впорядковане зберігання інформації, підтримує зв’язність між повідомленнями та комунікаційними просторами і створює основу для відновлення послідовності обміну даними [22, 23, 24].

3.3. Організація роботи серверної логіки

```
const express = require("express");
const mongoose = require("mongoose");
const { Server } = require("socket.io");
const config = require("config");
const http = require("http");
const cors = require("cors");

const routesOfInteractionWithRoom = require("./routes/interactWithRoom");
const createUserCounterForEachRoom =
require("./services/createUserCounterForEachRoom");
const initializeSocketHandlers = require("./sockets/initializeSocketHandlers");

const SERVER_PORT = config.get("serverPort");
const MONGO_DATABASE_URI = config.get("mongoDatabaseURI");

const application = express();
const httpServer = http.createServer(application);
const socketServer = new Server(httpServer, {
    cors: { origin: "*", methods: ["GET", "POST"] }
});

application.use(cors());
```

```
application.use(express.json());
application.use("/api/interact-with-room", routesOfInteractionWithRoom);

mongoose.connect(MONGO_DATABASE_URI)
    .then(async () => {
        console.log("Database connected...");

        await createUserCounterForEachRoom();
        initializeSocketHandlers(socketServer);

        httpServer.listen(SERVER_PORT, () => {
            console.log(`Server started on port ${SERVER_PORT}...`);
        });
    })
    .catch((error) => console.log(error.message));
```

Серверний модуль ініціалізації основної логіки забезпечує формування інфраструктури для оброблення запитів, керування з'єднаннями у режимі реального часу та взаємодії з базою даних. На початковому етапі здійснюється імпорт бібліотек, що відповідають за побудову мережевої взаємодії, створення транспортного рівня, керування подієвими каналами, оброблення конфігураційних параметрів та застосування політики міждоменного доступу. Додатково підключаються внутрішні модулі, які реалізують маршрутизацію для операцій із комунікаційними просторами, ініціалізацію механізмів підрахунку користувачів та налаштування обробників подій для каналів у режимі реального часу. Після отримання конфігураційних параметрів створюється екземпляр інтерфейсного застосунку, формуються серверні структури для оброблення звичайних та подієвих запитів і визначаються правила міжмережевої взаємодії [6, 7, 25, 35].

Після налаштування проміжного програмного шару активуються маршрути, призначені для роботи з комунікаційними просторами, та встановлюються механізми оброблення структурованих даних. Далі ініціюється підключення до сховища даних, що у разі успішного встановлення з'єднання дозволяє виконати підготовчі процедури, пов'язані з формуванням лічильників користувачів у межах кожного комунікаційного простору, а також активувати обробники подій, які регулюють функціонування каналів у режимі реального часу. Завершальним етапом

являється запуск серверної інфраструктури на вказаному мережевому порту, що робить систему доступною для зовнішніх запитів та забезпечує її подальшу операційну роботу [6, 7, 23, 25].

```
const { Router } = require("express");

const generateUserIdentifier = require("../services/generateUserIdentifier");
const createCustomRoom = require("../services/createCustomRoom");
const getRandomRoom = require("../services/getRandomRoom");
const deleteRoom = require("../services/deleteRoom");

const router = Router();

router.post("/initialize-custom-room", async (request, response) => {
    try {
        const { roomMode } = request.body;

        const userIdentifier = generateUserIdentifier();
        const { customRoom, customRoomMessages } = await
createCustomRoom(roomMode);

        response.status(201).json({
            message: "Custom room initialization completed successfully",
            userIdentifier, customRoom, customRoomMessages
        });

    } catch (error) {
        response.status(500).json({ message: "Error initializing custom
room" });
        console.log(error.message);
    }
});

router.post("/initialize-random-room", async (request, response) => {
    try {
        const { roomMode } = request.body;

        const userIdentifier = generateUserIdentifier();
        const { randomRoom, randomRoomMessages } = await
getRandomRoom(roomMode);

        response.status(201).json({
            message: "Random room initialization completed successfully",
            userIdentifier, randomRoom, randomRoomMessages
        });
```

```
    } catch (error) {
        response.status(500).json({ message: "Error initializing random
room" });
        console.log(error.message);
    }
});

router.delete("/delete-room/:roomIdentifier", async (request, response) => {
    try {
        await deleteRoom(request.params.roomIdentifier);
        response.status(200).json({
            message: "Room deletion completed successfully"
        });

    } catch (error) {
        response.status(500).json({ message: "Error deleting room" });
        console.log(error.message);
    }
});

module.exports = router;
```

Маршрутизатор серверної частини керує обробленням запитів, пов’язаних із формуванням, отриманням та видаленням комунікаційних просторів, забезпечуючи узгоджену взаємодію між клієнтським інтерфейсом і внутрішніми сервісними компонентами. У модулі створюється спеціалізований об’єкт маршрутизації, до якого підключаються допоміжні процедури, що генерують анонімні ідентифікатори користувачів, формують нові комунікаційні простори, реалізують вибір існуючих просторів за визначеними умовами та забезпечують їхнє видалення [6, 7, 25].

Перша група маршрутів призначена для ініціації комунікаційних просторів фіксованого типу. Після отримання структурованих даних від клієнта активується механізм створення унікального користувацького маркера та викликається процедура формування нового простору, яка повертає інформацію про створений об’єкт разом із набором пов’язаних повідомлень. У відповідь надсилається повідомлення про успішне створення середовища для взаємодії разом із необхідними даними для подальшої роботи [6, 7, 25].

Друга група маршрутів реалізує механізм вибору випадкових комунікаційних просторів на основі визначеного режиму взаємодії. Використовується алгоритм формування унікального маркера користувача та процедура отримання відповідного простору, після чого створюється відповідь, що містить інформацію про вибраний об’єкт і пов’язані з ним дані [6, 7, 25].

Окрема маршрутна конструкція відповідає за видалення комунікаційних просторів. Після отримання ідентифікатора простору активується сервісна процедура очищення, яка виконує операцію видалення відповідного об’єкта зі сховища даних. Результатом є повідомлення про успішне завершення операції або повідомлення про помилку у разі порушення працездатності сервісної частини. Сукупність цих механізмів формує ядро HTTP-взаємодії серверної логіки та забезпечує повноцінну підтримку операцій із комунікаційними просторами [6, 7, 25].

```
const Room = require("../models/Room");
const { createUserCounterForRoom } = require("./interactWithRoomUserCounter");

async function createUserCounterForEachRoom() {
    try {
        const allRooms = await Room.find();
        allRooms.forEach((room) => createUserCounterForRoom(room._id));

    } catch (error) { throw error }
}

module.exports = createUserCounterForEachRoom;
```

Сервісний модуль ініціалізації лічильників користувачів для всіх комунікаційних просторів забезпечує відновлення внутрішнього стану системи під час запуску серверної частини. Після звернення до сховища даних виконується отримання повного переліку комунікаційних просторів, що існують у системі, що дозволяє визначити множину актуальних об’єктів, які потребують синхронізації з внутрішніми сервісними структурами. Для кожного виявленого простору активується механізм формування окремого лічильника користувачів, завдяки чому

забезпечується коректне встановлення початкового стану обліку учасників незалежно від кількості вже існуючих записів у базі даних. Такий підхід дозволяє гарантовано відновити консистентність сервісних структур після запуску системи й забезпечує можливість подальшого коректного керування кількістю активних користувачів у кожному комунікаційному середовищі [21, 23, 25].

```
const saveUserMessage = require("../services/saveUserMessage");
const { getUserCountForRoom, updateUserCounterForRoom } =
require("../services/interactWithRoomUserCounter");

const socketToRoom = new Map();

function initializeSocketHandlers(socketServer) {
    socketServer.on("connection", (socket) => {

        socket.on("join-room", ({ roomIdentifier }) => {
            socket.join(roomIdentifier);
            socketToRoom.set(socket.id, roomIdentifier);

            updateUserCounterForRoom(roomIdentifier, 1);
            const userCountForRoom = getUserCountForRoom(roomIdentifier);

            socketServer.to(roomIdentifier).emit("get-user-count",
userCountForRoom);
            socket.broadcast.to(roomIdentifier).emit("receive-system-message",
"Anonymous user has joined the room");
        });

        socket.on("send-user-message", async ({ roomIdentifier, userIdentifier,
userMessageContent }) => {
            try {
                const userMessage = await saveUserMessage(roomIdentifier,
userIdentifier, userMessageContent);

                socketServer.to(roomIdentifier).emit("receive-user-message",
userMessage);
                socket.broadcast.to(roomIdentifier).emit("receive-system-
message", "Anonymous user wrote and sent a message");

            } catch (error) {
                socket.emit("receive-system-message", "Error sending and saving
message");
                console.log(error.message);
            }
```

```
        });

        socket.on("leave-room", ({ roomIdentifier }) => {
            socket.leave(roomIdentifier);

            updateUserCounterForRoom(roomIdentifier, -1);
            const userCountForRoom = getUserCountForRoom(roomIdentifier);

            socketServer.to(roomIdentifier).emit("get-user-count",
userCountForRoom);
            socket.broadcast.to(roomIdentifier).emit("receive-system-message",
"Anonymous user has leave the room");

            socketToRoom.delete(socket.id);
        });

        socket.on("disconnect", () => {
            const roomIdentifier = socketToRoom.get(socket.id);

            if(roomIdentifier) {
                updateUserCounterForRoom(roomIdentifier, -1);
                const userCountForRoom = getUserCountForRoom(roomIdentifier);

                socketServer.to(roomIdentifier).emit("get-user-count",
userCountForRoom);
                socket.broadcast.to(roomIdentifier).emit("receive-system-
message", "Anonymous user has disconnected");

                socketToRoom.delete(socket.id);
            }
        });

    });
}

module.exports = initializeSocketHandlers;
```

Модуль ініціалізації обробників подій для каналу взаємодії у режимі реального часу формує повноцінну інфраструктуру для динамічного обміну даними між учасниками комунікаційних просторів. На початковому етапі створюється внутрішня структура відображення, яка дозволяє пов’язувати окремі з’єднання з відповідними просторами взаємодії. Це забезпечує можливість точно визначати, до

якого середовища належить кожен активний канал, а також здійснювати контроль за кількістю користувачів у кожному з них [25, 35].

Після встановлення з’єднання між клієнтом і сервером активується обробник подій, що регулює приєднання до комунікаційного простору. У межах цієї операції клієнт підключається до групового каналу, який відповідає обраному простору, виконується оновлення лічильника активних користувачів, а також трансляція системних повідомлень та актуального стану кількості учасників іншим підключеним користувачам. Така процедура забезпечує узгоджене інформування всіх учасників про зміни у структурі присутності [35].

Окремий обробник призначено для передачі повідомлень у межах простору. Після отримання даних від клієнта активується сервісний механізм збереження повідомлення у сховищі даних, що гарантує фіксацію історії взаємодії. Після успішного проведення операції формується широкомовна подія, яка доставляє нове повідомлення всім учасникам відповідного середовища, а також надсилаються додаткові системні сповіщення, що сигналізують про активність користувача [35].

Механізм оброблення виходу з комунікаційного простору передбачає виключення користувача зі структури каналу, корекцію значення лічильника активних учасників і трансляцію оновлених даних усім залишеним користувачам. Після виконання цих операцій внутрішня інформація про зв’язок між каналом та простором очищається, що забезпечує коректність подальшого управління з’єднаннями [35].

Останній обробник регулює ситуації неявного відключення, коли канал завершує роботу без надсилання спеціального повідомлення про вихід. У такому випадку визначається, з яким середовищем був пов'язаний канал, після чого виконується корекція кількості активних учасників та надсилання відповідних системних повідомлень. Завдяки цьому вдається підтримувати узгодженість стану комунікаційних просторів навіть за непередбачуваних змін підключень. Сукупність розглянутих механізмів формує основу динамічної взаємодії в реальному часі та забезпечує стабільність роботи комунікаційної підсистеми [35].

```
const userCountersForAllRooms = new Map();

function createUserCounterForRoom(roomIdentifier) {
    const roomUserCounterKey = String(roomIdentifier);
    userCountersForAllRooms.set(roomUserCounterKey, 0);
}

function getUserCountForRoom(roomIdentifier) {
    const roomUserCounterKey = String(roomIdentifier);
    const userCountForRoom = userCountersForAllRooms.get(roomUserCounterKey);

    return userCountForRoom;
}

function updateUserCounterForRoom(roomIdentifier, value) {
    const roomUserCounterKey = String(roomIdentifier);
    const userCountForRoom = getUserCountForRoom(roomIdentifier);
    userCountersForAllRooms.set(roomUserCounterKey, userCountForRoom + value);
}

function deleteUserCounterForRoom(roomIdentifier) {
    const roomUserCounterKey = String(roomIdentifier);
    userCountersForAllRooms.delete(roomUserCounterKey);
}

module.exports = {
    createUserCounterForRoom,
    getUserCountForRoom,
    updateUserCounterForRoom,
    deleteUserCounterForRoom
};
```

Сервісний модуль керування лічильниками користувачів у комунікаційних просторах формує внутрішній механізм оперативного відстеження кількості активних учасників без звернення до зовнішнього сховища даних. У модулі використовується спеціалізована структура відображення, що забезпечує асоціативний доступ до значень лічильників за унікальними ідентифікаторами просторів. Такий підхід дозволяє підтримувати актуальний стан підключених користувачів у режимі реального часу та мінімізувати затримки, пов’язані з частими операціями читання й запису [25].

Окремі процедури забезпечують створення початкового лічильника для нового комунікаційного простору, отримання поточного значення кількості користувачів, оновлення його відповідно до подій підключення або відключення та повне видалення лічильника у разі усунення відповідного простору. Операції оновлення виконуються шляхом інкрементування або декрементування поточного значення, що забезпечує послідовність та узгодженість внутрішнього стану. Завдяки такій архітектурі модуль відіграє ключову роль у підтриманні правильності логіки взаємодії в реальному часі, оскільки дозволяє достовірно визначати доступність комунікаційних просторів, контролювати їхнє навантаження та гарантувати відповідність системним обмеженням щодо максимальної кількості учасників [25].

```
const { random } = Math;

function generateUserIdentifier() {
    const base = 36;
    const timestamp = Date.now().toString();
    const randomCharacters = random().toString(base).substring(2, 4);
    const userIdentifier = timestamp + randomCharacters;

    return userIdentifier;
}

module.exports = generateUserIdentifier;
```

Сервісний модуль формує унікальний анонімний ідентифікатор користувача шляхом поєднання часової позначки та випадкової компоненти, що забезпечує високий рівень варіативності сформованих значень. У процесі побудови ідентифікатора використовується числове представлення поточного часу, яке гарантує монотонність та однозначність у межах послідовних викликів. Доповнення часової компоненти випадковим фрагментом, згенерованим на основі числового перетворення у символьну форму, підвищує стійкість до повторень та забезпечує можливість масового створення незалежних маркерів у межах коротких проміжків часу. Об’єднання цих двох джерел ентропії дозволяє отримати значення, придатне

для використання в анонімних комунікаційних сценаріях, де не здійснюється прив’язка до реальних персональних даних [25, 27].

```
const config = require("config");

const Room = require("../models/Room");
const { createUserCounterForRoom } = require("./interactWithRoomUserCounter");

const ROOM_CREATION_LIMIT = config.get("roomCreationLimit");

async function createCustomRoom(roomMode) {
    try {
        const numberOfRooms = await Room.countDocuments();
        if(numberOfRooms >= ROOM_CREATION_LIMIT)
            throw new Error("Room creation failed: document creation limit
reached in the `rooms` collection");

        const customRoom = await Room.create({ mode: roomMode });
        createUserCounterForRoom(customRoom._id);

        return { customRoom, customRoomMessages: [] };

    } catch (error) { throw error }
}

module.exports = createCustomRoom;
```

Сервісний модуль створення індивідуалізованого комунікаційного простору забезпечує формування нового об’єкта у сховищі даних із урахуванням системних обмежень та ініціації допоміжних структур, необхідних для подальшого обслуговування користувачів. Перед створенням простору виконується підрахунок наявних записів, що дозволяє контролювати дотримання встановленої межі щодо максимальної кількості можливих комунікаційних середовищ. У разі перевищення граничного значення формування нового запису блокується з відповідною генерацією помилки, що запобігає неконтрольованому розширенню структури даних [21, 23, 25].

Після перевірки системних обмежень здійснюється створення нового простору із зазначенням режиму взаємодії, який визначає тип подальшого використання.

Одразу після формування об’єкта ініціюється допоміжний механізм обліку користувачів для щойно створеного середовища, що забезпечує можливість подальшого контролю кількості активних учасників. Результатом роботи модуля є повернення даних про новостворений комунікаційний простір разом із початковим порожнім набором повідомлень, що відображає початковий стан взаємодії в новому середовищі [21, 23, 25].

```
const { floor, random } = Math;
const config = require("config");

const Room = require("../models/Room");
const Message = require("../models/Message");
const { getUserCountForRoom } = require("./interactWithRoomUserCounter");

const DISPLAY_MESSAGE_LIMIT = config.get("displayMessageLimit");
const PRIVATE_ROOM_USER_LIMIT = config.get("privateRoomUserLimit");
const GROUP_ROOM_USER_LIMIT = config.get("groupRoomUserLimit");

const ROOM_USER_LIMITS = {
    private: PRIVATE_ROOM_USER_LIMIT,
    group: GROUP_ROOM_USER_LIMIT
};

async function getRandomRoom(roomMode) {
    try {
        const roomUserLimit = ROOM_USER_LIMITS[roomMode];
        const roomCollection = await Room.find({ mode: roomMode });

        const freeRooms = roomCollection.filter((room) => {
            const userCountForRoom = getUserCountForRoom(room._id);
            return userCountForRoom < roomUserLimit;
        });
        if(freeRooms.length === 0)
            throw new Error("Room search failed: no documents found in the
`rooms` collection");

        const randomRoomIndex = floor(random() * freeRooms.length);
        const randomRoom = freeRooms[randomRoomIndex];

        const randomRoomMessages = await Message.find({ room: randomRoom._id })
            .sort({ date: -1 }).limit(DISPLAY_MESSAGE_LIMIT);
        const sortedRandomRoomMessages = randomRoomMessages.reverse();
```

```
        return { randomRoom, randomRoomMessages: sortedRandomRoomMessages };

    } catch (error) { throw error }
}

module.exports = getRandomRoom;
```

Сервісний модуль вибору випадкового комунікаційного простору реалізує механізм пошуку середовища для взаємодії на основі зазначеного режиму та поточного навантаження, що забезпечує рівномірний розподіл користувачів і запобігає перевищенню встановлених обмежень. На початковому етапі визначається гранична кількість учасників для обраного режиму, після чого здійснюється отримання повного набору просторів відповідного типу зі сховища даних. Наступним кроком формується підмножина доступних просторів шляхом відфільтровування тих, у яких кількість активних учасників не перевищує встановлене системою значення. За відсутності придатних середовищ генерується помилка, що сигналізує про неможливість задовольнити запит [21, 23, 25].

Серед доступних просторів здійснюється стохастичний вибір одного елемента, що забезпечує непередбачуваність і рівномірність розподілу користувачів між можливими варіантами. Після отримання ідентифікованого комунікаційного простору виконується вибірка пов’язаних повідомлень зі сховища даних із використанням обмеження на максимальну кількість відображуваних записів. Вибрані повідомлення впорядковуються у хронологічній послідовності, що дозволяє коректно відтворювати історію взаємодії. Підсумком роботи модуля є надання структурованих даних про знайдений комунікаційний простір та впорядкованого набору пов’язаних повідомлень, які відображають попередній перебіг комунікації [21, 23, 25].

```
const config = require("config");

const Room = require("../models/Room");
const Message = require("../models/Message");
const { getUserCountForRoom, deleteUserCounterForRoom } =
require("./interactWithRoomUserCounter");
```

```
const MINIMUM_USERS_TO_DELETE_ROOM = config.get("minimumUsersToDeleteRoom");

async function deleteRoom(roomIdentifier) {
    try {
        const userCountForRoom = getUserCountForRoom(roomIdentifier);
        if(userCountForRoom > MINIMUM_USERS_TO_DELETE_ROOM)
            throw new Error("Room deletion failed: number of users exceeds
minimum required to delete");

        await Room.findByIdAndDelete(roomIdentifier);
        await Message.deleteMany({ room: roomIdentifier });
        deleteUserCounterForRoom(roomIdentifier);

    } catch (error) { throw error }
}

module.exports = deleteRoom;
```

Сервісний модуль видалення комунікаційного простору реалізує механізм усунення структур даних, пов’язаних із конкретним середовищем взаємодії, за умови дотримання визначених системних обмежень. На початковому етапі здійснюється отримання інформації про кількість активних учасників відповідного простору, що дозволяє визначити можливість виконання операції. Якщо фактична кількість учасників перевищує встановлене граничне значення, процес видалення блокується шляхом генерації помилки, що сприяє запобіганню некоректного припинення активних сесій [25, 29, 36].

У разі відповідності умовам видалення виконується усунення запису, який описує комунікаційний простір у сховищі даних, а також видалення всіх повідомлень, що були з ним пов’язані. Після очищення основних структур даних ініціюється процедура скасування механізму обліку користувачів для відповідного простору, що забезпечує підтримання внутрішньої узгодженості сервісних структур. Завершення сукупності цих операцій призводить до повного усунення простору та пов’язаних із ним даних, що відновлює узгоджений стан системи відповідно до встановленої логіки її функціонування [21, 23, 25].

```
const Message = require("../models/Message");

async function saveUserMessage(roomIdentifier, userIdentifier,
userMessageContent) {
    try {
        if (userMessageContent.trim().length === 0)
            throw new Error("Message saving failed: content is missing");

        const userMessage = await Message.create({
            room: roomIdentifier,
            sender: userIdentifier,
            contents: userMessageContent
        });

        return userMessage;

    } catch (error) { throw error }
}

module.exports = saveUserMessage;
```

Сервісний модуль збереження користувацьких повідомлень реалізує механізм фіксації результатів комунікації у сховищі даних з урахуванням вимог до валідності вхідних даних. Перед виконанням операції здійснюється перевірка наявності змістовної частини повідомлення, що дає змогу запобігти збереженню порожніх або формально невизначених записів. У разі порушення цієї умови генерується помилка, яка блокує подальшу обробку [25, 29, 36].

Після успішної перевірки проводиться створення нового документа, який містить інформацію про простір взаємодії, у рамках якого було сформовано повідомлення, ідентифікатор відправника та текстову частину повідомлення. Створений об’єкт повертається як результат роботи модуля, що забезпечує можливість його негайного поширення серед інших учасників відповідного середовища. Такий підхід гарантує структурованість даних, забезпечує відтворюваність історії комунікації та підтримує операційну цілісність процесу обміну інформацією [21, 23, 25].

3.4. Організація роботи клієнтської логіки

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" href="./favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="theme-color" content="#673ab7" />
    <meta name="description" content="Web application Anonymous room implemented
on MERN stack" />
    <link rel="apple-touch-icon" href="./images/logo_256x256.png" />
    <link rel="manifest" href="./manifest.json" />
    <title>A-Room</title>
  </head>
  <body>
    <noscript>
      JavaScript is disabled. The application cannot run.
    </noscript>
    <div id="root"></div>
  </body>
</html>
```

Файл *index.html* у клієнтському середовищі визначає базову структуру документа, яка слугує статичним контейнером для подальшого відтворення інтерфейсної частини веб-системи. У метаданих документа, який було описано раніше, задаються параметри кодування, масштабування для мобільних пристроїв, колірна тема та текстовий опис призначення застосунку, що забезпечує коректне відображення сторінки у браузері та підвищує відповідність вимогам сучасних веб-стандартів. Додатково вказуються ресурси, пов’язані з іконками та маніфестом прогресивного веб-додатка, що дає змогу оптимізувати роботу застосунку у середовищах із різним рівнем підтримки веб-технологій. У тілі документа розміщується контейнер, у який динамічно монтується інтерфейсна частина, сформована засобами бібліотеки компонентного підходу. Окремо передбачено повідомлення для випадку, коли динамічні можливості браузера недоступні, що гарантує коректне інформування користувача про неможливість запуску системи за відсутності програмної підтримки необхідних технологій [17, 30, 40].

```
import React from 'react';
import ReactDOM from 'react-dom/client';

import './assets/styles/css/index.css'

import Application from './Application';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Application />
  </React.StrictMode>
);

reportWebVitals();
```

Клієнтський модуль початкової ініціалізації формує точку входу інтерфейсної частини веб-системи та забезпечує монтування компонентної структури у статичний контейнер базового документа. У модулі підключаються засоби компонентного рендерингу, стилізаційні ресурси та головний компонент застосунку, який слугує кореневою ланкою інтерфейсної ієрархії. Після визначення цільового елемента у структурі документа створюється спеціалізований інтерфейсний вузол, на який проєктується компонента, що реалізує функціональну логіку клієнтської частини. Процес рендерингу виконується в режимі підвищеного контролю, що сприяє виявленню потенційних відхилень під час розроблення та експлуатації. Додатково активується модуль аналітичного спостереження за продуктивністю, що дає змогу проводити моніторинг показників роботи інтерфейсної частини за потреби її оптимізації [30, 31].

```
import "./assets/styles/css/application.css"

import MainScreen from "./screens/MainScreen";
import BackgroundFrame from "./components/BackgroundFrame";

function Application() {
  return (
    <div className="application">
      <MainScreen />
```

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

export default Application;
```

Клієнтський модуль, що формує кореневу структуру інтерфейсної частини, визначає композицію основних візуальних компонентів, які забезпечують відображення програмної логіки на рівні користувацького середовища. У модулі підключаються стилізаційні ресурси, що регулюють оформлення зовнішньої оболонки застосунку, а також два функціональні компоненти, один з яких відповідає за основну взаємодію користувача з системою, тоді як інший забезпечує декоративне та структурне оформлення фонової частини інтерфейсу. Структурне поєднання цих елементів у межах єдиного контейнера формує базовий каркас клієнтської логіки, на який накладаються функціональні та візуальні механізми інших компонентів застосунку [30, 31].

```
import "../assets/styles/css/backgroundFrame.css"

function BackgroundFrame() {
    return (
        <div className="background-frame">
            <div className="background-frame__corner background-frame__corner--
top-left" />
            <div className="background-frame__corner background-frame__corner--
top-right" />
            <div className="background-frame__corner background-frame__corner--
bottom-right" />
            <div className="background-frame__corner background-frame__corner--
bottom-left" />
        </div>
    );
}

export default BackgroundFrame;
```

Компонент декоративного оформлення інтерфейсу формує статичну структурну оболонку, яка виконує роль візуального тла для основних елементів

клієнтської частини. У межах компонента підключаються стилізаційні ресурси, що визначають графічне оформлення та просторове розміщення його складових. Структурно компонент містить контейнер із чотирма допоміжними елементами, кожен з яких представляє кутову декоративну фігуру, призначену для візуального підсилення композиції інтерфейсу. Завдяки такій організації компонент забезпечує формування цілісного фоново-декоративного каркаса, який не впливає на функціональну логіку застосунку, але підвищує візуальну впорядкованість та завершеність інтерфейсного середовища [31].

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

import "../assets/styles/css/mainScreen.css"

import logoIcon from "../assets/images/logo.png";
import AnonymousRoom from "../components/AnonymousRoom";
import initializeRoom from "../services/initializeRoom"

function MainScreen() {
    const [roomMode, setRoomMode] = useState("");
    const [roomData, setRoomData] = useState(null);
    const [roomArchive, setRoomArchive] = useState([]);
    const [userIdentifier, setUserIdentifier] = useState(null);
    const [isRoomModeList, setRoomModeList] = useState(false);
    const [isAnonymousRoom, setAnonymousRoom] = useState(false);

    const changeStateOfRoomModeList = () => setRoomModeList(!isRoomModeList);
    const changeStateOfAnonymousRoom = () => setAnonymousRoom(!isAnonymousRoom);
    const changeRoomMode = (currentMode) =>
        setRoomMode(previousMode => (previousMode === currentMode ? "" :
currentMode));

    const initializeCustomRoom = async () => {
        try {
            const data = await initializeRoom(
                "http://localhost:5000/api/interact-with-room/initialize-custom-
room",
                roomMode
            );

            console.log(data.message);
            setRoomData(data.customRoom);
            setRoomArchive(data.customRoomMessages);
```

```
            setUserIdentifier(data.userIdentifier);

            changeStateOfRoomModeList();
            changeStateOfAnonymousRoom();
            setRoomMode("");

        } catch (error) {
            console.log(error.message);
        }
    };

    const initializeRandomRoom = async () => {
        try {
            const data = await initializeRoom(
                "http://localhost:5000/api/interact-with-room/initialize-random-
room",
                roomMode
            );

            console.log(data.message);
            setRoomData(data.randomRoom);
            setRoomArchive(data.randomRoomMessages);
            setUserIdentifier(data.userIdentifier);

            changeStateOfRoomModeList();
            changeStateOfAnonymousRoom();
            setRoomMode("");

        } catch (error) {
            console.log(error.message);
        }
    };

    return (
        <div className="main-screen">
            <div className="main-screen__branding">
                <img className="main-screen__logo" src={logoIcon} alt="logo.png"
/>
                <div className='main-screen__title'>Anonymous room</div>
            </div>
            <div className="main-screen__description">Connect with people you
don't know anonymously.</div>
            <div className="main-screen__controls">
                <button className="main-screen__create-button"
onClick={initializeCustomRoom}>Create</button>
                <button className="main-screen__join-button"
onClick={initializeRandomRoom}>Join</button>
                <div className="main-screen__mode-section">
```

```
                    <button className="main-screen__mode-button"
onClick={changeStateOfRoomModeList}>Mode</button>
                    {isRoomModeList && (
                        <div className="main-screen__mode-list">
                            <button className="main-screen__private-button"
data-active={roomMode === "private"} onClick={() =>
changeRoomMode("private")}>Private</button>
                            <button className="main-screen__group-button" data-
active={roomMode === "group"} onClick={() =>
changeRoomMode("group")}>Group</button>
                        </div>
                    )}
                </div>
            </div>
            {isAnonymousRoom && (
                <AnonymousRoom
                    roomData={roomData}
                    roomArchive={roomArchive}
                    userIdentifier={userIdentifier}
                    changeStateOfAnonymousRoom={changeStateOfAnonymousRoom}
                />
            )}
        </div>
    );
}

export default MainScreen;
```

Компонент, що формує основний інтерфейс початкового екрана, виконує роль центрального вузла клієнтської логіки, відповідального за вибір режиму взаємодії, ініціювання створення або приєднання до комунікаційного простору та відображення структур, необхідних для подальшої анонімної комунікації. У межах компонента підтримується низка внутрішніх станів, що регулюють вибір режиму роботи, збереження отриманих даних про комунікаційний простір, формування архіву повідомлень, визначення анонімного ідентифікатора користувача та перемикання між інтерфейсними режимами відображення [30, 31].

Логіка взаємодії включає механізми зміни вибраного режиму, керування видимістю переліку доступних варіантів та активацію процедур створення або вибору комунікаційного простору. Під час ініціації взаємодії здійснюється надсилання структурованого запиту до серверної частини із зазначенням вибраного

режиму, після чого в разі успішної відповіді компонента оновлює внутрішні стани відповідно до отриманих даних: зберігається інформація про комунікаційний простір, історія його використання та анонімний маркер користувача. Після оновлення станів активуються механізми перемикання інтерфейсного відображення, що забезпечують перехід від початкового екрана до інтерфейсу анонімної кімнати [16, 30].

Структурно інтерфейс компонента містить блоки brand-ідентифікації, короткий опис призначення застосунку та елементи керування, що забезпечують вибір режиму й виклик відповідних процедур. Після підтвердження взаємодії та отримання актуальних даних відображається окрема структура, яка реалізує компонент анонімного комунікаційного простору. У результаті компонент формує повноцінний логічний міст між початковим користувацьким взаємодією та функціонуванням внутрішніх механізмів, забезпечуючи послідовний перехід до операційного інтерфейсу системи [30, 31].

```
async function initializeRoom(url, roomMode) {
    try {
        const response = await fetch(url, {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ roomMode })
        });
        const data = await response.json();
        if(!response.ok) throw new Error(data.message);
        return data;

    } catch(error) { throw error }
}

export default initializeRoom;
```

Сервісний модуль для ініціалізації комунікаційного простору реалізує механізм обміну даними між клієнтською та серверною частинами з метою створення або вибору середовища взаємодії відповідно до зазначеного режиму. Під час виконання процедури формується мережевий запит із використанням

структурованого формату передавання даних, у межах якого до серверної частини передається інформація про вибраний режим роботи. Після відправлення запиту модуль очікує на відповідь, у якій містяться дані щодо комунікаційного простору, його історії повідомлень та сформованого анонімного ідентифікатора користувача [16, 30].

Отримана відповідь проходить перевірку на коректність за допомогою оцінювання статусу мережевої операції. У разі виявлення відхилень формується виняткова ситуація, що передається в зовнішній обробник, тим самим забезпечуючи контроль цілісності процесу ініціалізації. Якщо ж відповідь є коректною, дані повертаються до викликаного компонента, де вони використовуються для оновлення інтерфейсного стану та подальшої активації механізмів анонімної комунікації. Такий підхід створює узгоджену модель взаємодії між клієнтом і сервером та гарантує передання лише валідних структур даних [16, 30].

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

import "../assets/styles/css/anonymousRoom.css";

import DeletionConfirmation from "./DeletionConfirmation";
import socketHandlers from "../services/socketHandlers";

function AnonymousRoom({ roomData, roomArchive, userIdentifier,
changeStateOfAnonymousRoom }) {
    const [roomUserCounter, setRoomUserCounter] = useState(0);
    const [roomMessages, setRoomMessages] = useState(roomArchive);
    const [userMessageContent, setUserMessageContent] = useState("");
    const [isDeletionConfirmation, setDeletionConfirmation] = useState(false);
    const [isUserIdentifier, setUserIdentifier] = useState(false);

    useEffect(() => {
        socketHandlers.connectSocket();
        socketHandlers.emitJoinRoom(roomData._id);

        socketHandlers.onGetUserCount(setRoomUserCounter);
        socketHandlers.onReceiveUserMessage((currentMessage) =>
            setRoomMessages((previousMessages) => [...previousMessages,
currentMessage]));
        socketHandlers.onReceiveSystemMessage(console.log);
        socketHandlers.onDisconnect(changeStateOfAnonymousRoom);
```

```
        return () => {
            socketHandlers.offGetUserCount();
            socketHandlers.offReceiveUserMessage();
            socketHandlers.offReceiveSystemMessage();
            socketHandlers.offDisconnect();

            socketHandlers.disconnectSocket();
        };
    }, [roomData._id]);

    const changeStateOfUserIdentifier = () => setUserIdentifier(!
isUserIdentifier);
    const changeStateOfDeletionConfirmation = (value) =>
setDeletionConfirmation(value);
    const inputChangeProcessing = (event) =>
setUserMessageContent(event.target.value);

    const sendUserMessage = () => {
        if (!userMessageContent.trim()) return;
        socketHandlers.emitSendUserMessage(roomData._id, userIdentifier,
userMessageContent);
        setUserMessageContent("");
    };

    return (
        <div className="anonymous-room">
            <div className="anonymous-room__header">
                <div className="anonymous-room__primary-information">
                    <div className="anonymous-room__title">{roomData.name}</div>
                    <div className="anonymous-room__creation-
date">{roomData.date}</div>
                    <div className="anonymous-room__user-counter">This room
currently has {roomUserCounter} active participants</div>
                </div>
                <button className="anonymous-room__leave-button" onClick={() =>
changeStateOfDeletionConfirmation(true)}>Leave the room</button>
            </div>
            <div className="anonymous-room__messages">
                {roomMessages.length === 0 && (
                    <div className="anonymous-room__status">No messages</div>
                )}
                {roomMessages.map(roomMessage => (
                    <div className="anonymous-room__message"
key={roomMessage._id}>
                        <span className="anonymous-
room__sender">{roomMessage.sender}</span>
```

```
                        <span className="anonymous-
room__contents">{roomMessage.contents}</span>
                        <span className="anonymous-
room__time">{roomMessage.date}</span>
                    </div>
                ))}
            </div>
            <div className="anonymous-room__interaction-panel">
                <div className="anonymous-room__identifier-section">
                    <button className="anonymous-room__identifier-button"
onClick={changeStateOfUserIdentifier}>ID</button>
                    {isUserIdentifier && (
                        <div className="anonymous-room__user-identifier">You
have been assigned temporary identifier {userIdentifier}</div>
                    )}
                </div>
                <div className="anonymous-room__message-section">
                    <textarea className="anonymous-room__input-field"
value={userMessageContent} onChange={inputChangeProcessing} spellCheck="true"
placeholder="Message..."/>
                    <button className="anonymous-room__send-button"
onClick={sendUserMessage}>Send message</button>
                </div>
            </div>
            {isDeletionConfirmation && (
                <DeletionConfirmation
                    roomIdentifier={roomData._id}
                    leaveSocketRoom={socketHandlers.emitLeaveRoom}
                    changeStateOfAnonymousRoom={changeStateOfAnonymousRoom}

changeStateOfDeletionConfirmation={changeStateOfDeletionConfirmation}
                />
            )}
        </div>
    );
}

export default AnonymousRoom;
```

Компонент анонімного комунікаційного простору формує динамічну інтерфейсну структуру, у межах якої реалізується обмін повідомленнями, відстеження активності учасників та керування процесом завершення взаємодії. У компоненті підтримуються внутрішні стани, що описують кількість активних користувачів, перелік отриманих повідомлень, вміст введеного користувачем тексту,

режим відображення службових елементів та стан підтвердження виходу з комунікаційного середовища. Така система станів забезпечує можливість реагування інтерфейсу на події, що надходять від користувача або від зовнішніх джерел [30, 31].

Під час монтування компонента ініціюється підключення до каналу взаємодії у режимі реального часу, після чого активується процедура приєднання до відповідного комунікаційного простору. Паралельно відбувається реєстрація обробників подій, що відповідають за отримання інформації про кількість активних користувачів, надходження нових повідомлень, оброблення системних сповіщень та опрацювання неявного завершення взаємодії. Реєстр обробників синхронізує стан інтерфейсу з даними, що надходять від серверної частини, забезпечуючи коректне відображення кількісних змін, доповнення архіву повідомлень та обробку випадків примусового припинення з’єднання. Під час демонтування компонента виконується від’єднання від каналу та скасування зареєстрованих обробників, що унеможливлює виникнення помилкових оновлень стану після завершення роботи відповідного інтерфейсного середовища [30, 34].

Функціональна частина компонента містить механізми керування режимами відображення службового ідентифікатора користувача, оброблення введених текстових даних та передачі повідомлень до зовнішнього каналу. Перед відправленням текст підлягає перевірці на наявність змістовної частини, що запобігає появі порожніх записів. Відправлені повідомлення зберігаються у структурі станів, а їх оброблення на серверному боці дозволяє актуалізувати інформацію для всіх учасників середовища [30, 34].

Інтерфейсна структура компонента складається з декількох логічних зон: заголовка з відомостями про комунікаційне середовище та кнопкою завершення взаємодії, області відображення отриманих повідомлень, панелі взаємодії для введення тексту та службових елементів, а також окремого блоку підтвердження виходу з комунікаційного простору. Така організація забезпечує послідовне та чітке

відтворення функціональної логіки анонімної комунікації та створює повноцінне середовище для обміну даними у реальному часі [18, 30, 31].

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

const socket = io("http://localhost:5000", {
    autoConnect: false,
    reconnection: false
});

const connectSocket = () => socket.connect();
const disconnectSocket = () => socket.disconnect();

const emitJoinRoom = (roomIdentifier) => socket.emit("join-room", {
roomIdentifier });
const emitLeaveRoom = (roomIdentifier) => socket.emit("leave-room", {
roomIdentifier });
const emitSendUserMessage = (roomIdentifier, userIdentifier, userMessageContent)
=>
    socket.emit("send-user-message", { roomIdentifier, userIdentifier,
userMessageContent });

const onGetUserCount = (callback) => socket.on("get-user-count", callback);
const onReceiveUserMessage = (callback) => socket.on("receive-user-message",
callback);
const onReceiveSystemMessage = (callback) => socket.on("receive-system-message",
callback);
const onDisconnect = (callback) => socket.on("disconnect", callback);

const offGetUserCount = () => socket.off("get-user-count");
const offReceiveUserMessage = () => socket.off("receive-user-message");
const offReceiveSystemMessage = () => socket.off("receive-system-message");
const offDisconnect = () => socket.off("disconnect");

export default {
    connectSocket, disconnectSocket,
    emitJoinRoom, emitLeaveRoom, emitSendUserMessage,
    onGetUserCount, onReceiveUserMessage, onReceiveSystemMessage, onDisconnect,
    offGetUserCount, offReceiveUserMessage, offReceiveSystemMessage,
offDisconnect
};
```

Сервісний модуль керування взаємодією з каналом обміну даними у режимі реального часу формує уніфікований інтерфейс для встановлення з’єднання,

передавання подій та отримання оновлень від серверної частини. У модулі створюється клієнтський екземпляр каналу, для якого заборонено автоматичне підключення та повторні спроби встановлення з’єднання, що забезпечує суворий контроль над моментами ініціації та завершення взаємодії. Такий підхід дозволяє компонентам інтерфейсу самостійно визначати умови підключення відповідно до логіки роботи застосунку [30, 34, 35].

Операції над каналом згруповано за функціональним призначенням. До першої групи належать механізми встановлення та розриву зв’язку, які забезпечують можливість активації або завершення взаємодії з комунікаційним простором у визначені моменти. Друга група охоплює механізми передавання подій, що відповідають за приєднання та вихід користувача з комунікаційного простору, а також за передавання текстових повідомлень. Передавання даних здійснюється у структурованому форматі, який дозволяє серверній частині однозначно інтерпретувати інформацію про простір, користувача та зміст повідомлення [34, 35].

Третю групу становлять механізми реєстрації обробників подій, які фіксують реакцію клієнтського інтерфейсу на зміни, що надходять від серверної частини. Серед таких подій — оновлення кількості активних учасників, отримання повідомлень від інших користувачів, надходження системних сповіщень та оброблення неявного завершення взаємодії. Завдяки цим механізмам компонентна структура інтерфейсу підтримує актуальний стан, синхронізуючись із зовнішніми змінами у реальному часі [30, 34, 35].

Завершальну групу становлять механізми скасування реєстрації обробників, що використовується під час демонтування компонентів інтерфейсу. Це запобігає накопиченню зайвих викликів, забезпечує стабільність роботи клієнтської частини та унеможливлює оновлення стану після завершення взаємодії. Сукупність цих інструментів формує цілісну підсистему управління комунікацією в реальному часі та забезпечує надійну інтеграцію між інтерфейсною й серверною логікою [30, 34, 35].

```
import "../assets/styles/css/deletionConfirmation.css"
import deleteRoom from "../services/deleteRoom";

function DeletionConfirmation({ roomIdentifier, leaveSocketRoom,
    changeStateOfAnonymousRoom, changeStateOfDeletionConfirmation }) {

    const leaveRoomWithDeletion = async () => {
        try {
            const data = await deleteRoom(
                `http://localhost:5000/api/interact-with-room/delete-room/$
{roomIdentifier}`
            );

            console.log(data.message);
            leaveSocketRoom(roomIdentifier);
            changeStateOfDeletionConfirmation(false);
            changeStateOfAnonymousRoom();

        } catch (error) {
            console.log(error.message);
        }
    };

    const leaveRoomWithoutDeletion = () => {
        leaveSocketRoom(roomIdentifier);
        changeStateOfDeletionConfirmation(false);
        changeStateOfAnonymousRoom();
    };

    return (
        <div className="deletion-confirmation">
            <div className="deletion-confirmation__question">Do you want to
delete the room?</div>
            <div className="deletion-confirmation__answer-section">
                <button className="deletion-confirmation__confirm-button"
onClick={leaveRoomWithDeletion}>Yes</button>
                <button className="deletion-confirmation__decline-button"
onClick={leaveRoomWithoutDeletion}>No</button>
                <button className="deletion-confirmation__cancel-button"
onClick={() => changeStateOfDeletionConfirmation(false)}>Cancel</button>
            </div>
        </div>
    );
}

export default DeletionConfirmation;
```

Компонент підтвердження видалення комунікаційного простору виконує роль проміжної інтерфейсної ланки, що регулює процес завершення взаємодії користувача із середовищем анонімної комунікації та визначає подальшу долю відповідного простору. У компонент передаються функціональні механізми, які відповідають за ініціювання видалення простору на серверній стороні, припинення участі у каналі взаємодії та зміну станів інтерфейсу, що визначають видимість структур, пов'язаних із комунікаційним середовищем [30, 31].

Логіка компонента передбачає два альтернативні сценарії виходу: з видаленням комунікаційного простору та без його усунення. У першому випадку формується мережевий запит до серверної частини, що ініціює процес видалення відповідного об'єкта та пов'язаних даних у сховищі. Після одержання підтвердження виконуються завершальні дії на клієнтському боці, зокрема припинення участі у каналі взаємодії та повернення інтерфейсу до початкового стану. У другому випадку відсутня взаємодія із сервером: інтерфейс завершує участь у комунікаційному середовищі, а його видимість вимикається, при цьому сам простір зберігається [16, 30].

Інтерфейсна частина компонента складається зі службового запитання щодо видалення простору та набору керуючих елементів, що дозволяють користувачеві вибрати бажаний сценарій або скасувати дію, повертаючись до поточного стану комунікації. Така структура забезпечує контрольованість і передбачуваність завершальних етапів взаємодії, мінімізує ризики випадкового видалення та підтримує узгодженість між клієнтською й серверною частинами системи [30, 31].

```
async function deleteRoom(url) {
    try {
        const response = await fetch(url, {
            method: "DELETE"
        });
        const data = await response.json();
        if(!response.ok) throw new Error(data.message);
        return data;

    } catch(error) { throw error }
```

```
}

export default deleteRoom;
```

Сервісний модуль, що забезпечує ініціювання процесу видалення комунікаційного простору, реалізує мережеву взаємодію між клієнтською і серверною частинами із використанням структурованого запиту, орієнтованого на здійснення операції усунення даних. Під час виконання процедури формується мережевий запит із зазначенням методу, який відповідає операції видалення. У відповідь від серверної частини надходить структурований результат, що містить інформацію про підсумок операції [16, 30].

Після отримання відповіді модуль проводить оцінювання коректності виконання операції за допомогою аналізу статусу мережевого запиту. У разі, якщо відповідь свідчить про виникнення помилки, формується виняткова ситуація, що передається до зовнішнього обробника. Якщо ж виконання операції було успішним, результат повертається до викликаного компонента, де використовується для подальшої зміни інтерфейсного стану. Такий підхід забезпечує чітку й контрольовану реалізацію сценарію видалення та гарантує відповідність клієнтської логіки результатам, які надходять від серверної частини [16, 30].

```
const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};

export default reportWebVitals;
```

Модуль аналітичного спостереження за показниками продуктивності забезпечує можливість вимірювання параметрів, що характеризують ефективність

завантаження та відтворення інтерфейсної частини веб-системи. Механізм роботи модуля ґрунтується на динамічному підключенні спеціалізованої бібліотеки, яка містить алгоритми оцінювання ключових метрик, пов’язаних із відображенням елементів інтерфейсу, обробленням подій та затримками під час відтворення контенту. Після підтвердження наявності функції оброблення результатів активуються процедури обчислення показників стабільності розмітки, затримки першої взаємодії, швидкості формування початкового контенту, відображення найбільшого об’єкта та часу встановлення зв’язку. Кожна з отриманих величин передається зовнішньому обробнику, що забезпечує можливість подальшої оцінки та оптимізації продуктивності клієнтської частини на основі фактичних даних експлуатації [17, 30, 31, 40].

3.5. Висновки до розділу 3

Розглянутий розділ охопив практичну реалізацію веб-системи для анонімної комунікації, у межах якої було сформовано цілісну інфраструктуру серверної та клієнтської частин. На серверному боці реалізовано механізми конфігурації, структурування даних, оброблення подій у режимі реального часу та забезпечення взаємодії з базою даних, що створює основу для стабільного функціонування системи та підтримки динамічних процесів обміну повідомленнями. Клієнтська частина забезпечує відтворення інтерфейсних компонентів, організацію взаємодії користувача із середовищем анонімної комунікації та синхронізацію з подіями, що надходять від серверної інфраструктури. Сукупність реалізованих програмних рішень формує функціонально узгоджену систему, здатну забезпечувати анонімний обмін інформацією та підтримувати цілісність процесів комунікації в умовах динамічного навантаження.

Посилання на GitHub: https://github.com/PoloviukOS/anonymousRoom

РОЗДІЛ 4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програмного забезпечення охоплює процес перевірки його функціональної цілісності шляхом послідовного відтворення характерних сценаріїв роботи та аналізу реакції системи на стандартні й нестандартні дії. У межах цього етапу досліджуються механізми оброблення запитів, поведінка підсистем під час взаємодії з користувачем, а також стійкість до помилкових або непередбачуваних ситуацій. Такий підхід забезпечує підтвердження надійності реалізованої логіки, демонструє фактичну працездатність програмного продукту та формує підґрунтя для оцінювання його готовності до експлуатації.

4.1. Підготовка середовища для тестування

Підготовка середовища для тестування передбачає послідовне отримання програмного забезпечення, інсталяцію необхідних залежностей та активацію серверної й клієнтської частин у відповідних середовищах виконання. Для початку необхідно виконати завантаження програмного коду шляхом ініціювання отримання віддаленого репозиторію через відповідну команду у середовищі термінала, що забезпечує формування локальної копії проекту разом зі всіма його складовими [9, 39].

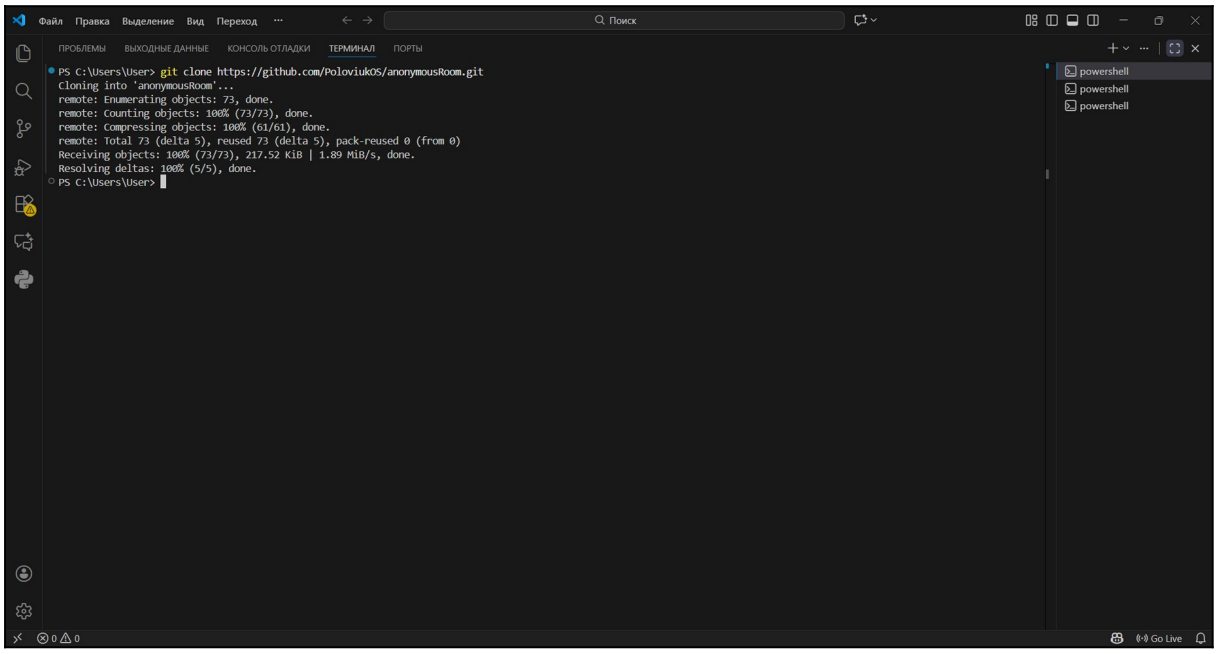


Рис. 14. Клонування віддаленого GitHub-репозиторію

Після завершення отримання коду слід перейти до каталогу, у якому розміщено структуру серверної частини, та ініціювати встановлення необхідних компонентів, які забезпечують можливість подальшого виконання програмної логіки. Для цього послідовно здійснюється перехід між каталогами та виклик операції інсталяції залежностей, що формує повноцінне серверне середовище [25, 26, 39].

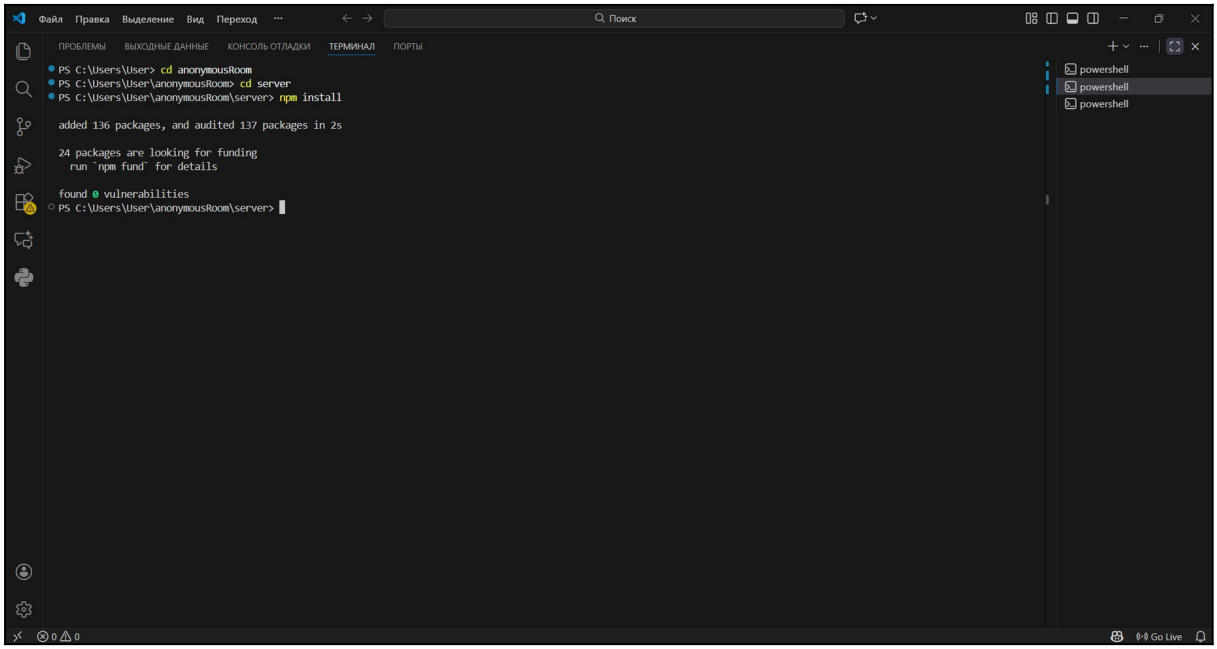


Рис. 15. Встановлення server-залежностей

Аналогічні дії необхідно виконати для клієнтської частини: перейти до каталогу інтерфейсної складової та ініціювати встановлення компонентів, необхідних для коректної роботи клієнтської логіки, що забезпечує підготовку інтерфейсного середовища до подальшого запуску та взаємодії з серверною частиною [26, 30, 39].

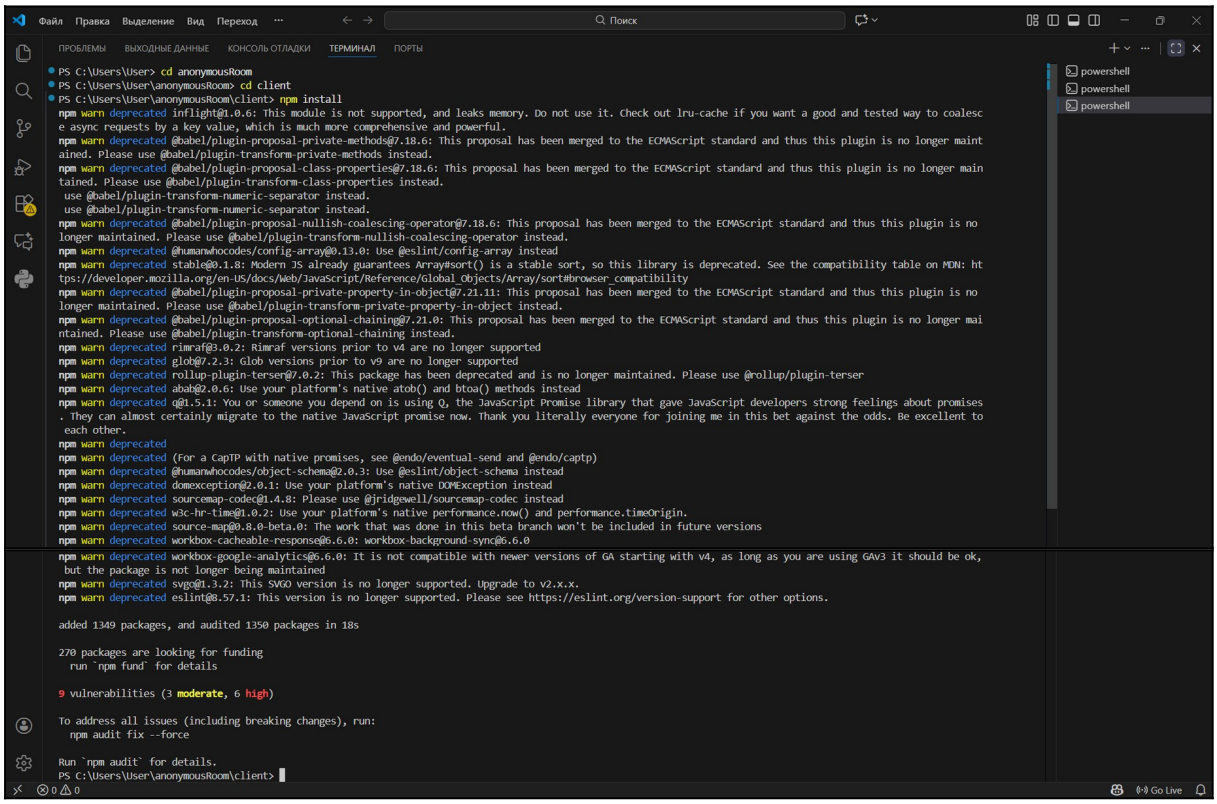


Рис. 16-17. Встановлення client-залежностей

Після формування середовища слід активувати серверну частину шляхом виклику відповідної команди у терміналі, відкритому в каталозі серверної складової проекту, що призводить до запуску серверної інфраструктури у режимі розроблення, що дозволяє спостерігати за її роботою та забезпечує можливість подальшої взаємодії клієнтської частини з сервером [25, 26, 39].

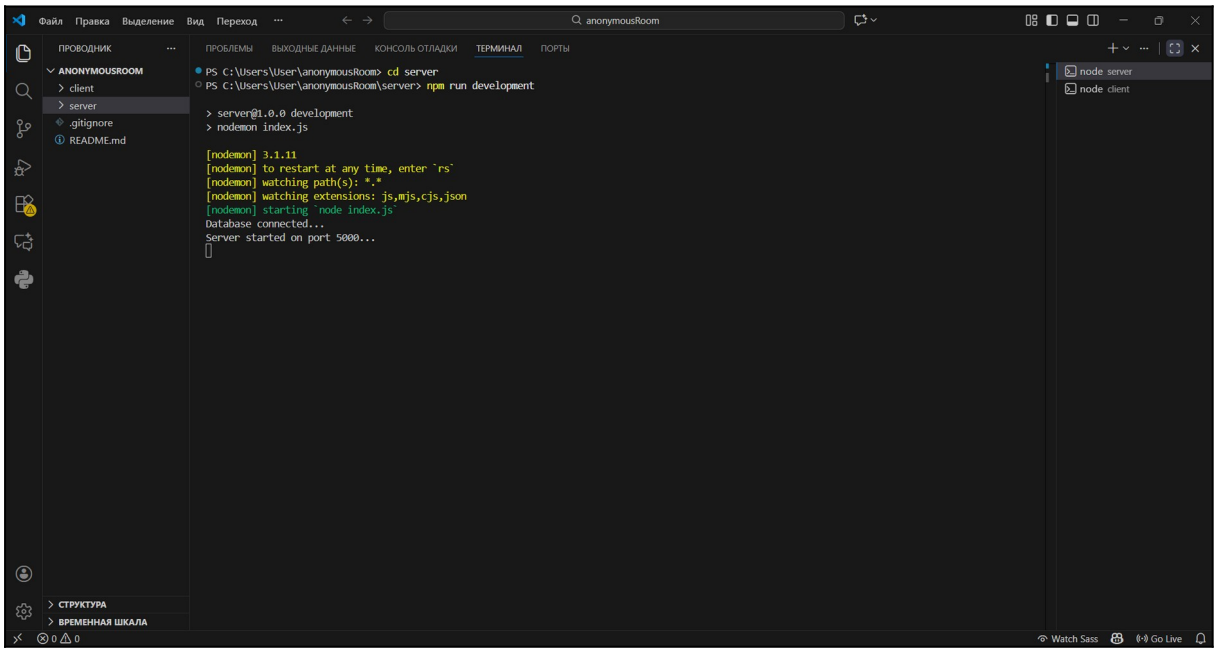


Рис. 18. Запуск server-частини проекту

Наступним кроком є запуск клієнтської складової у середовищі термінала, відкритого в каталозі інтерфейсної частини проекту, що забезпечує розгортання користувацького інтерфейсу у браузері та створює умови для виконання подальших тестових операцій [26, 30, 39].

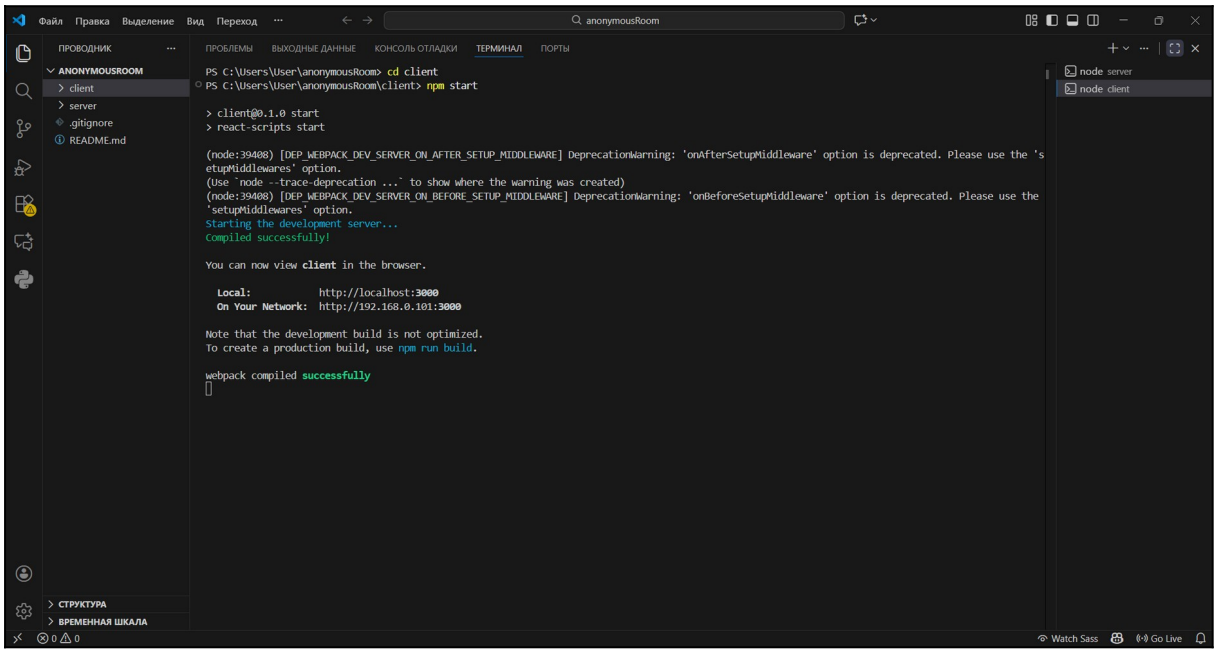


Рис. 19. Запуск client-частини проекту

Завершальним етапом підготовки являється перевірка стану сховища даних шляхом відкриття відповідної бази у засобі візуалізації, що дозволяє переконатися у правильності підключення та доступності структур, пов’язаних із роботою програмного забезпечення [20, 22].

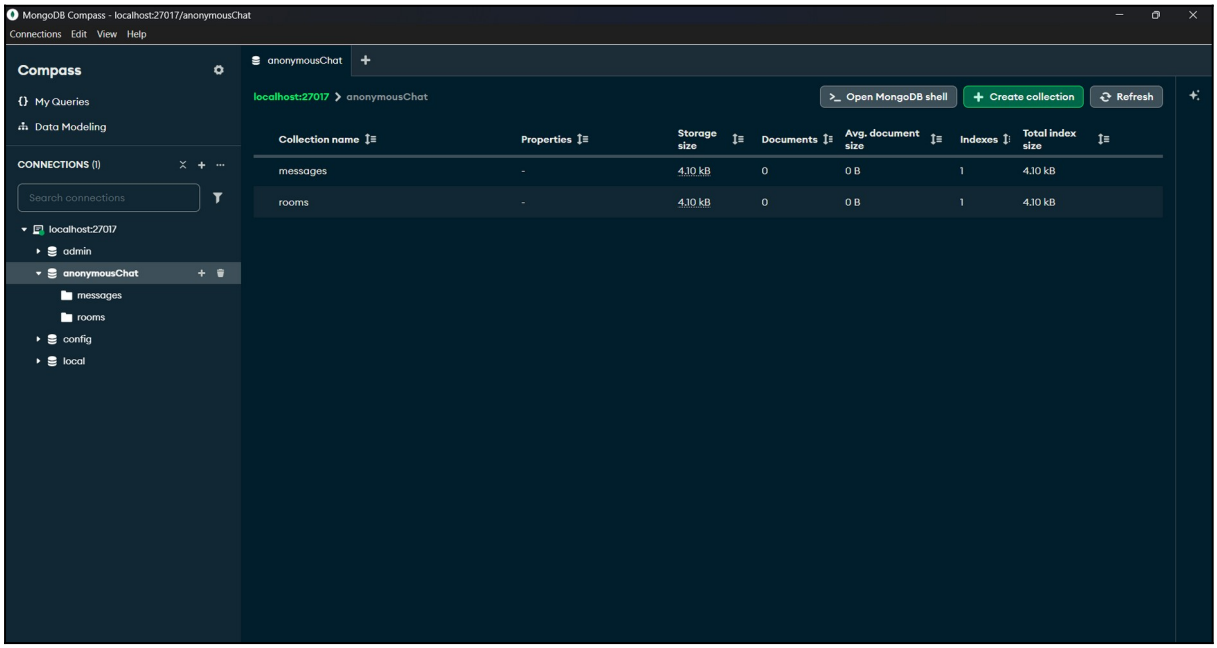


Рис. 20. Вміст автоматично створенної бази даних

4.2. Функціональне тестування користувацького інтерфейсу

Функціональне тестування користувацького інтерфейсу здійснюється шляхом послідовного відтворення дій, які демонструють роботу основних механізмів системи та підтверджують коректність взаємодії між інтерфейсною частиною, серверною складовою та каналом обміну даними. Під час тестування користувач працює у браузері з відкритою консоллю, де відображаються інформаційні повідомлення, що супроводжують виконання кожної операції [19, 29, 36].

Для отримання комунікаційного середовища необхідно обрати бажаний режим взаємодії, після чого ініціювати створення кімнати. Після формування середовища система відображає службові дані, зокрема тимчасовий ідентифікатор, який дозволяє користувачеві брати участь у подальшій комунікації.

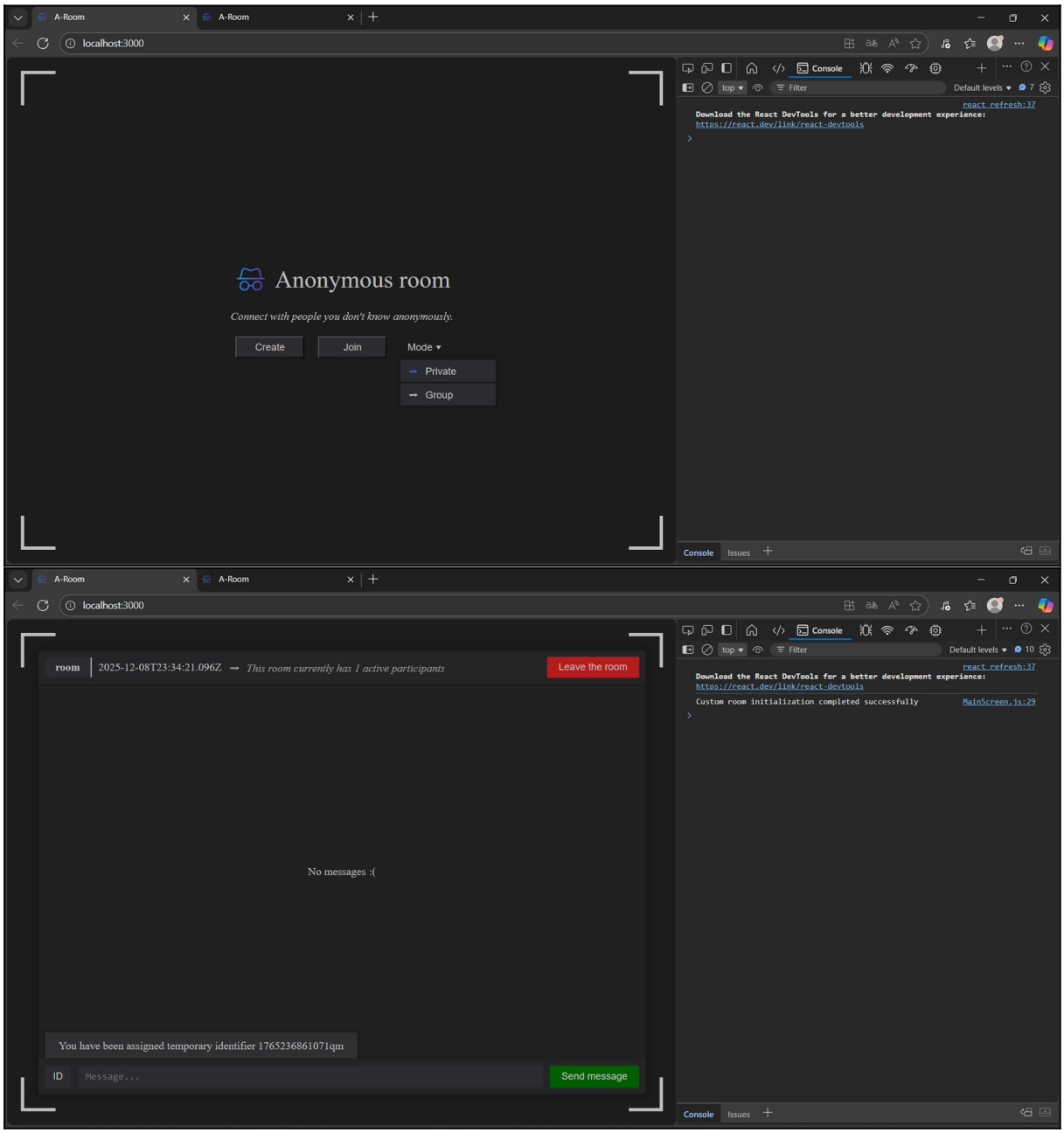


Рис. 21-22. Створення custom-кімнати

Альтернативним способом є вибір режиму та підключення до вже наявного середовища. У процесі підключення користувач отримує службову інформацію про доступ до кімнати та присвоєний ідентифікатор, що підтверджує успішний вхід.

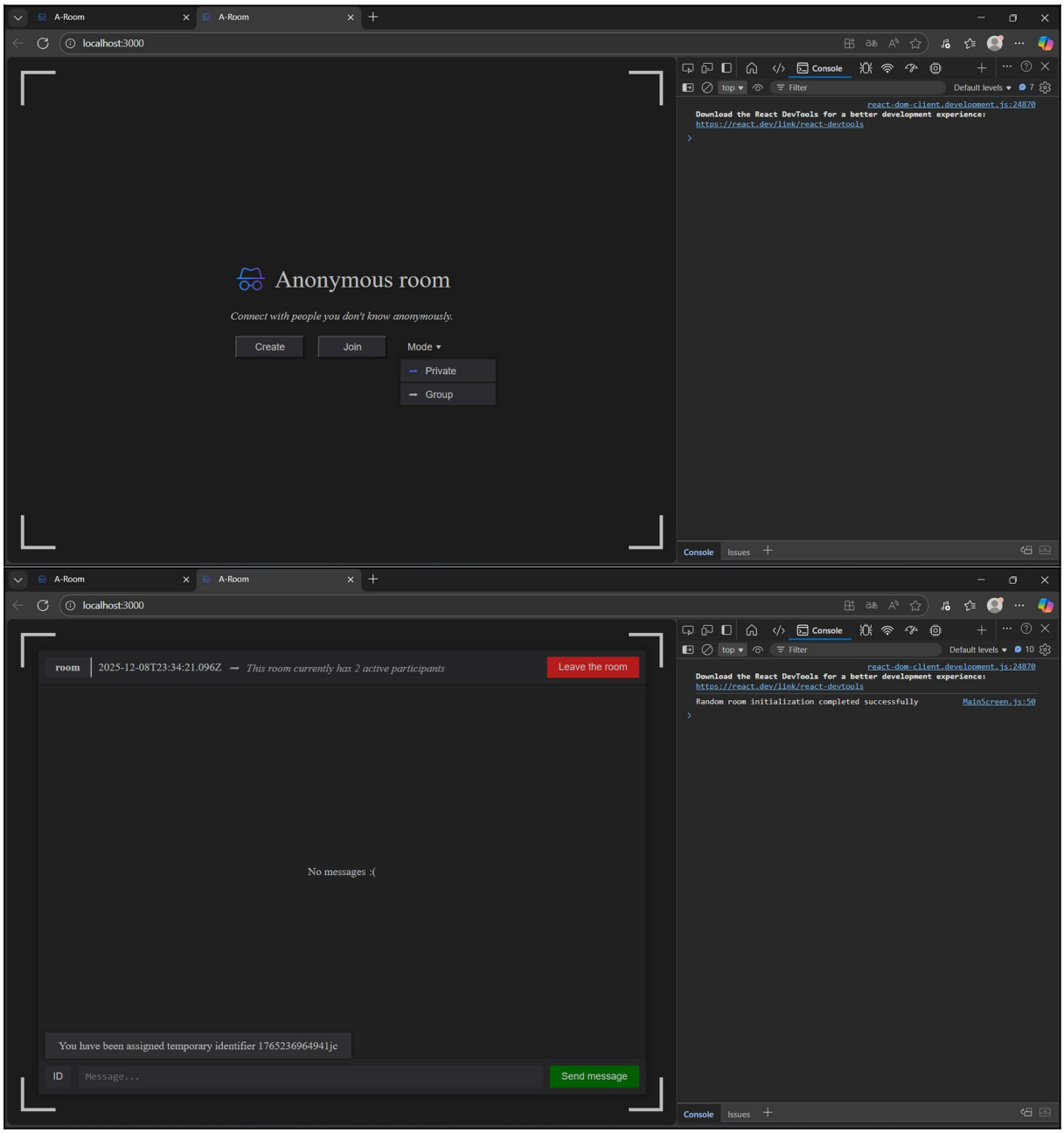


Рис. 23-24. Підключення до випадкової вімнати

Для здійснення комунікації користувач вводить текстове повідомлення у відповідне поле та активує надсилання. Після цього повідомлення відображається у структурі інтерфейсу та дублюється у консолі, що засвідчує правильність його оброблення та передачі.

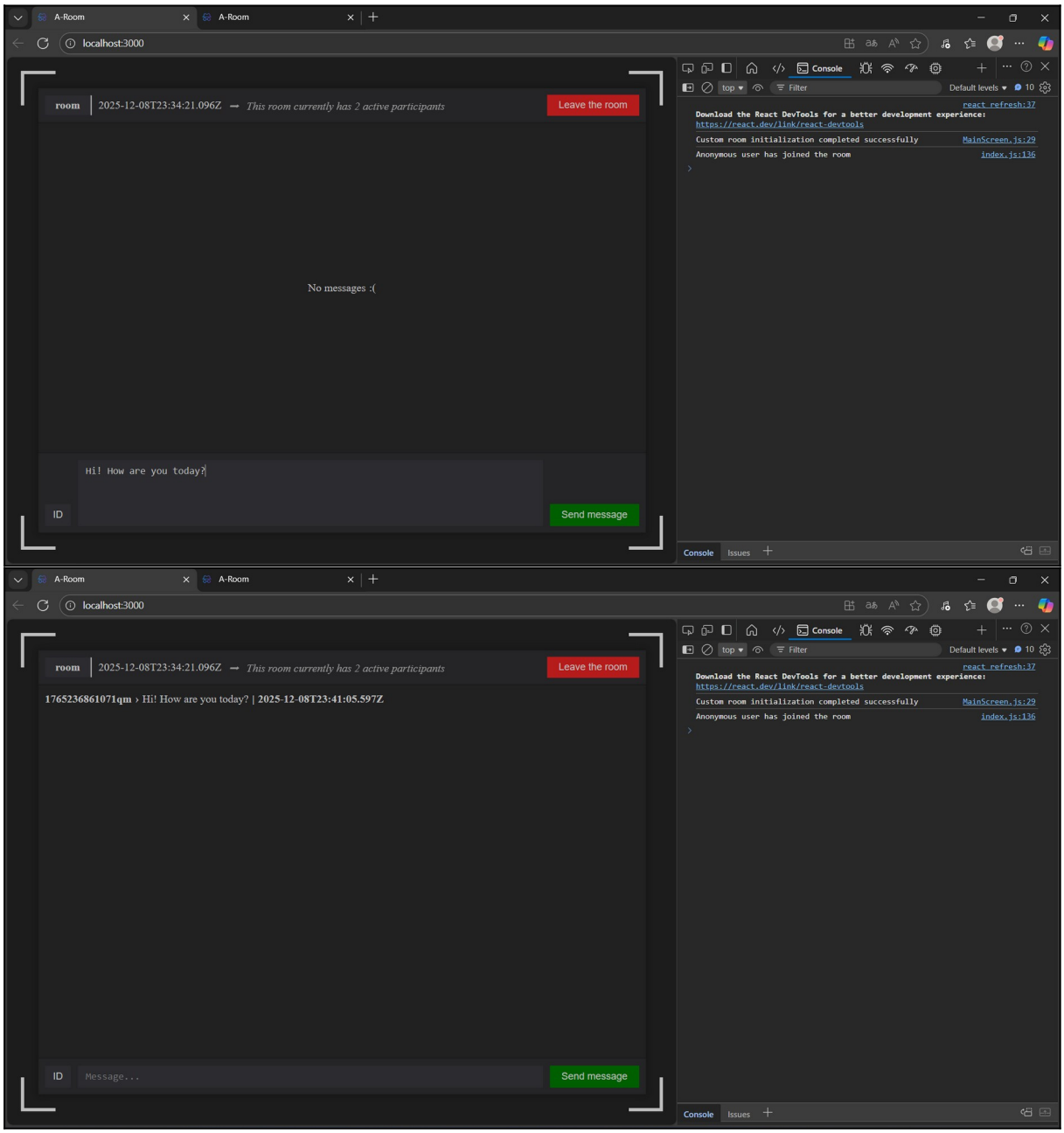


Рис. 25-26. Написання та відправка повідомлення

Під час одночасної взаємодії декількох учасників система забезпечує обмін даними у режимі реального часу. Кожен користувач отримує повідомлення від інших учасників у структурі інтерфейсу, а відповідні службові сигнали надходять до консолі, що підтверджує роботу каналу live-взаємодії.

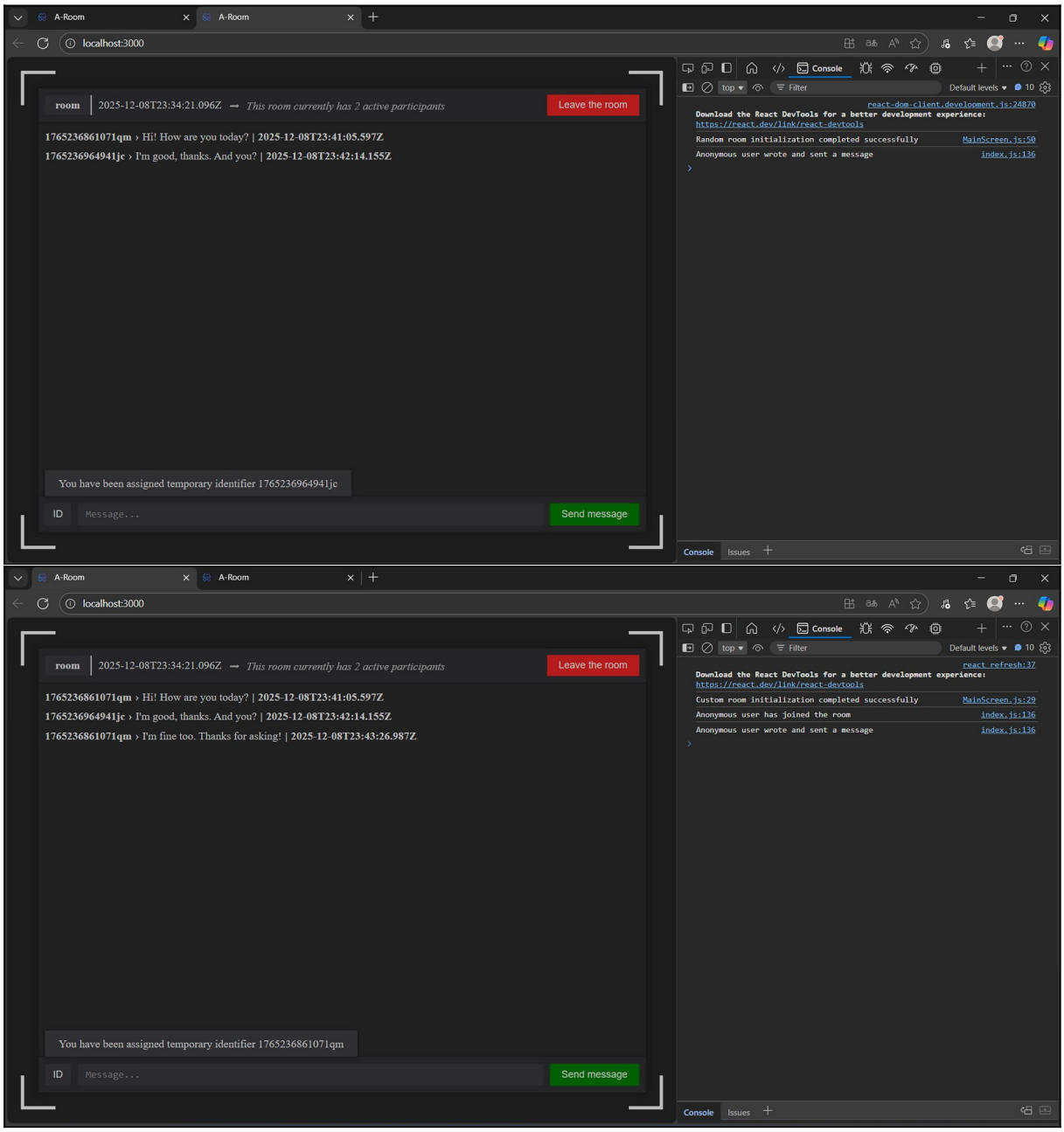


Рис. 27-28. Обмін повідомленнями між користувачами в режимі реального часу

У разі завершення роботи в середовищі користувач може залишити кімнату без її видалення. У цьому випадку система припиняє участь у каналі обміну, інтерфейс повертається до початкового стану, а в консолі відображається повідомлення про ручний вихід. Така ж логіка спрацьовує і в разі закриття вкладки браузера, однак відповідні службові сигнали у консолі мають інший зміст.

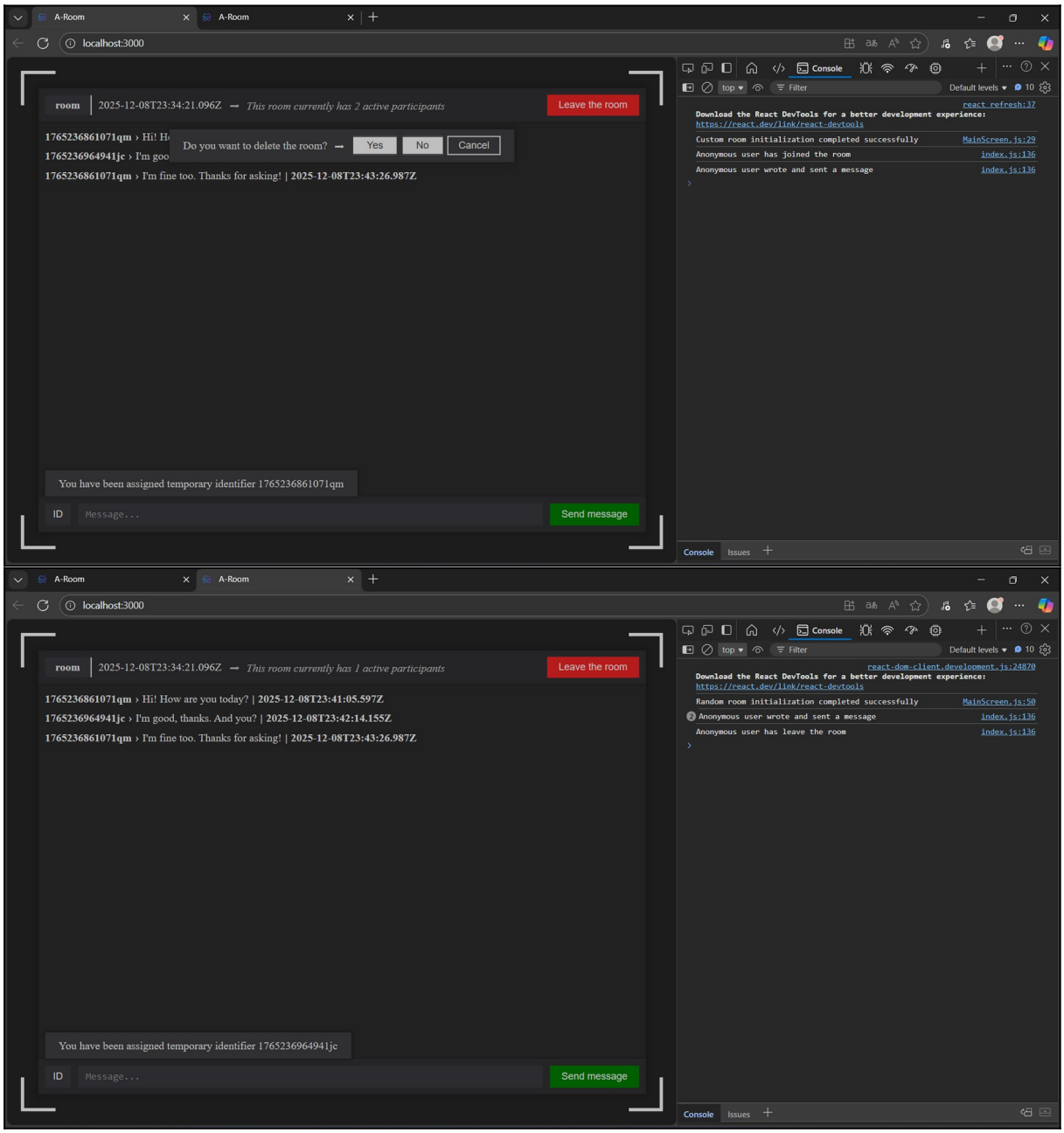


Рис. 29-30. Вихід з кімнати без її видалення

Користувач також може завершити роботу з одночасним видаленням середовища. У цьому випадку система ініціює видалення кімнати та пов’язаних даних, після чого завершує участь у каналі взаємодії та повертається до початкового інтерфейсного стану.

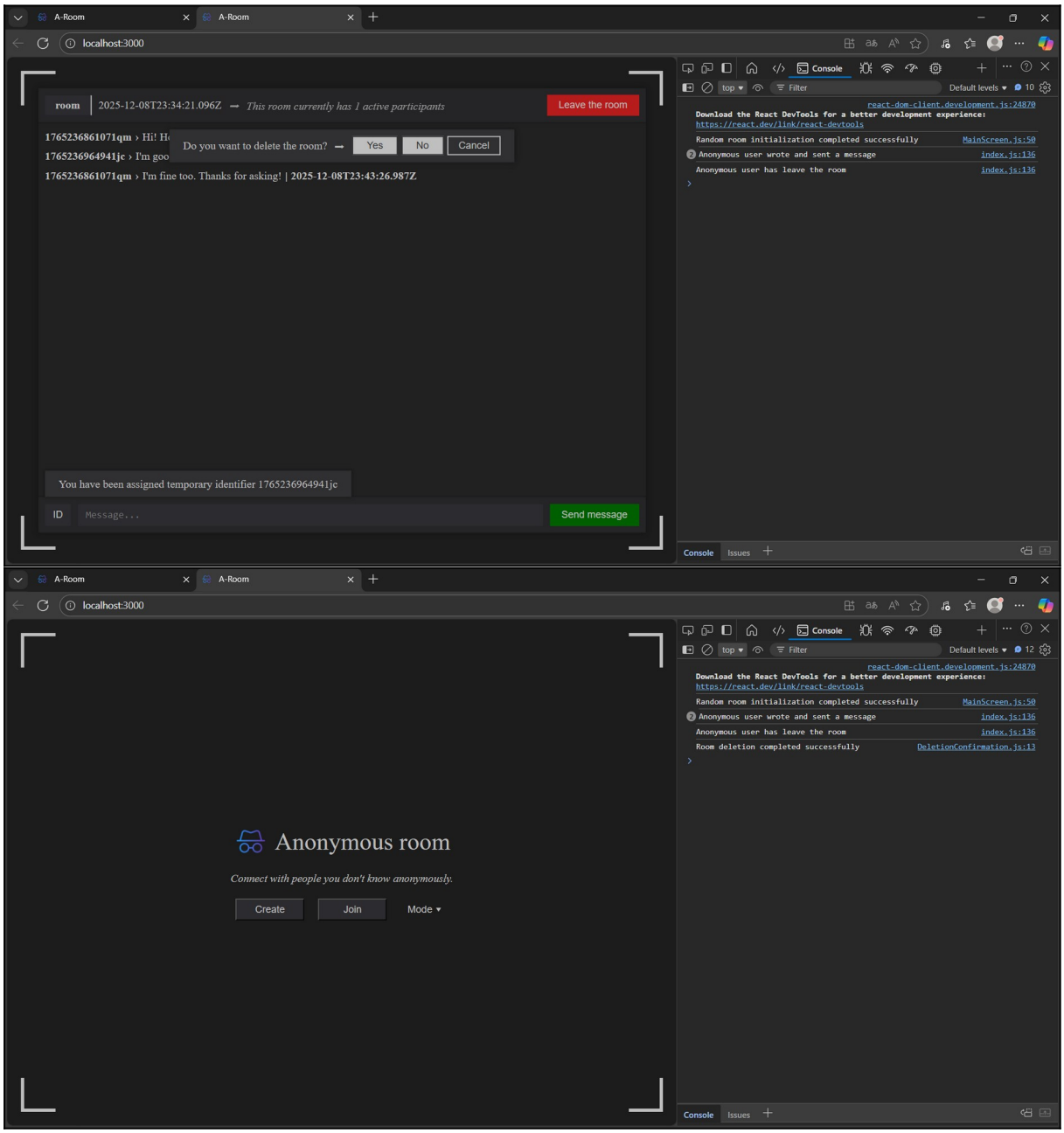


Рис. 31-32. Вихід з видаленням кімнати

Функціональні тести користувацького інтерфейсу в даному підрозділі були виконані без помилок, тоді як зображення, що демонструють оброблення некоректних ситуацій, розміщено у відповідному додатку (додаток до функціонального UI-тестування).

4.3. Тестування API з використанням Postman

Тестування прикладного програмного інтерфейсу здійснюється шляхом послідовного виконання запитів, що відтворюють основні сценарії взаємодії між клієнтською логікою та серверною частиною. Для цього використовується інструмент, який дозволяє формувати запити, надсилати їх на відповідні адреси та переглядати структуру отриманих відповідей. У процесі тестування аналізуються дані, що повертаються сервером, а також коректність виконання запитів відповідно до закладеної логіки роботи системи [28].

Для створення нового комунікаційного середовища необхідно сформувати запит на ініціалізацію custom-кімнати, зазначивши режим взаємодії у структурі тіла запиту. Після його виконання сервер повертає success-повідомлення, дані, що містять інформацію про створене середовище, початковий архів повідомлень та ідентифікатор користувача. Отримана відповідь підтверджує коректність роботи механізмів створення кімнат і доступності структур, пов’язаних із її подальшим використанням.

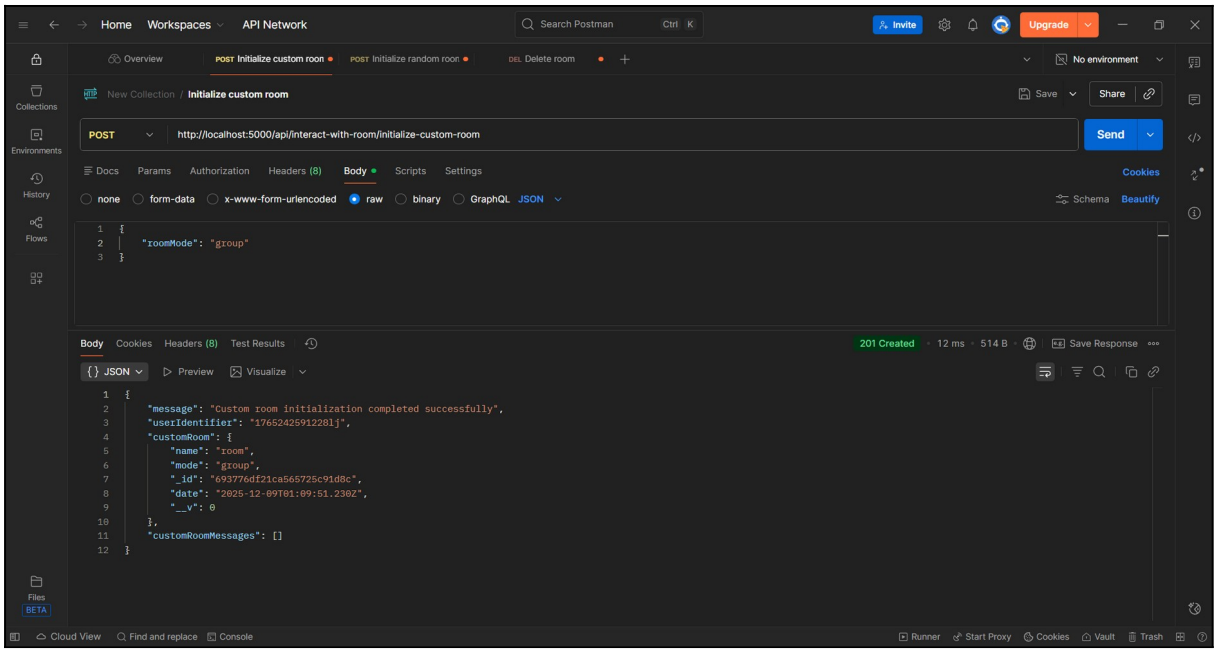


Рис. 33. Виконання запиту ініціалізації custom-кімнати

Для підключення до наявного середовища використовується запит ініціалізації випадкової кімнати, у якому також зазначається бажаний режим взаємодії. Після виконання запиту сервер повертає структуру даних, що містить success-повідомлення, інформацію про вибране середовище, впорядкований архів повідомлень та сформований ідентифікатор користувача. Така відповідь демонструє працездатність механізму добору доступних кімнат і правильність формування даних для початкового підключення.

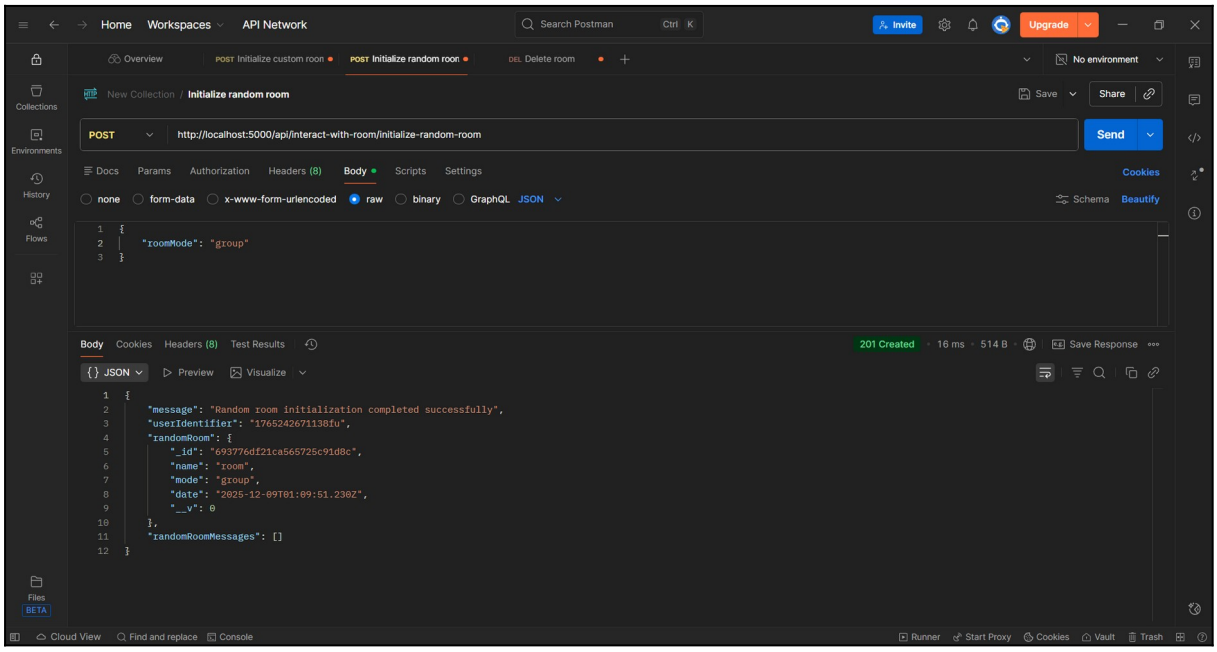


Рис. 34. Виконання запиту ініціалізації випадкової кімнати

Для завершення роботи із середовищем використовується запит видалення кімнати, у якому передається відповідний ідентифікатор. Після виконання запиту сервер підтверджує успішне усунення середовища, що включає видалення даних про кімнату та пов’язаних повідомлень. Отримана відповідь підтверджує коректність роботи механізмів керування життєвим циклом комунікаційних просторів.

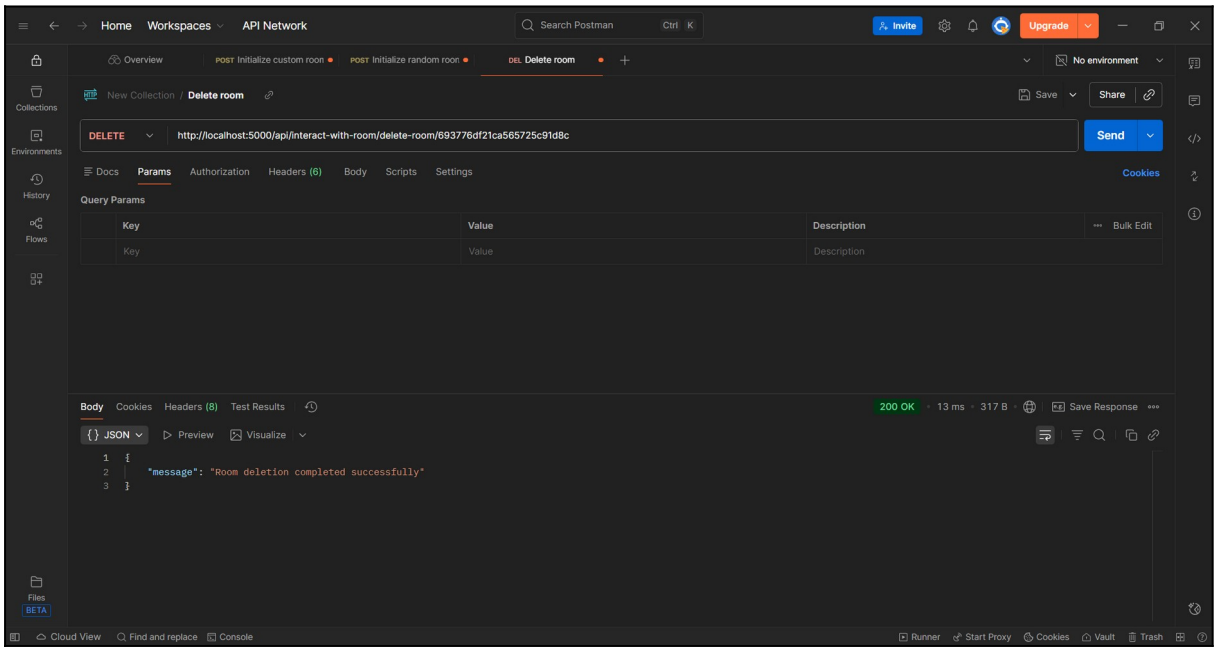


Рис. 35. Виконання запиту видалення кімнати

API-тести з використанням Postman були виконані в даному підрозділі без помилок, тоді як зображення, що демонструють оброблення виняткових ситуацій, розміщено у відповідному додатку (додаток до тестування API з використанням Postman).

4.4. Аналіз бази даних після виконання тестів

Аналіз стану бази даних після проведення тестування дає можливість простежити зміни, що відбулися у структурі збережених даних під час виконання різних сценаріїв взаємодії. Послідовний розгляд результатів виконання операцій дозволяє визначити відповідність фактичного вмісту сховища очікуваним наслідкам роботи програмного забезпечення та підтвердити коректність механізмів створення, використання та видалення комунікаційних просторів і пов’язаних із ними повідомлень [20, 22].

Після виконання операцій зі створення двох комунікаційних просторів база даних містить два записи у відповідній колекції. Один із них сформовано під час функціонального тестування користувацького інтерфейсу, другий — під час здійснення запиту в межах тестування прикладного програмного інтерфейсу.

Обидва записи відображають структуру даних, сформовану механізмами створення кімнат, і засвідчують коректне функціонування програмної логіки.

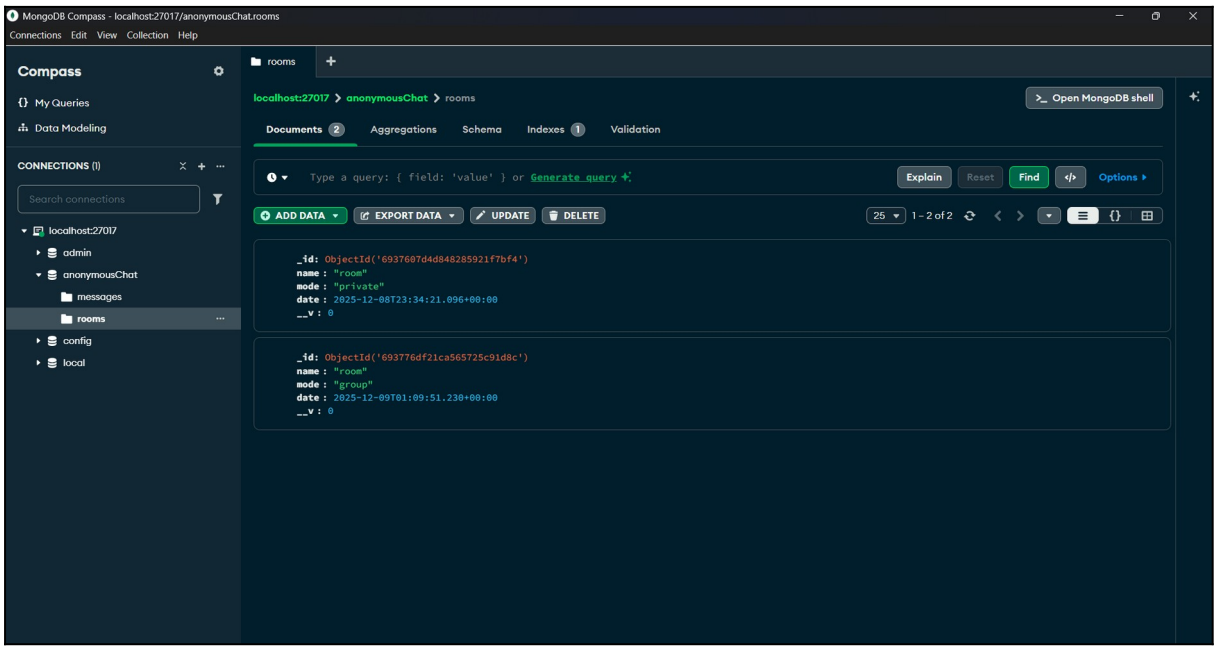


Рис. 36. Вміст rooms-колекції (два створені room-документи)

У процесі обміну повідомленнями між користувачами було сформовано три записи, які відображено у колекції збережених повідомлень. Кожен документ містить інформацію про автора, зміст та часову позначку створення, що демонструє правильність роботи механізмів передачі та збереження даних у режимі реального часу. Відповідність структури документів очікуваному формату підтверджує стабільність роботи інфраструктури оброблення повідомлень.

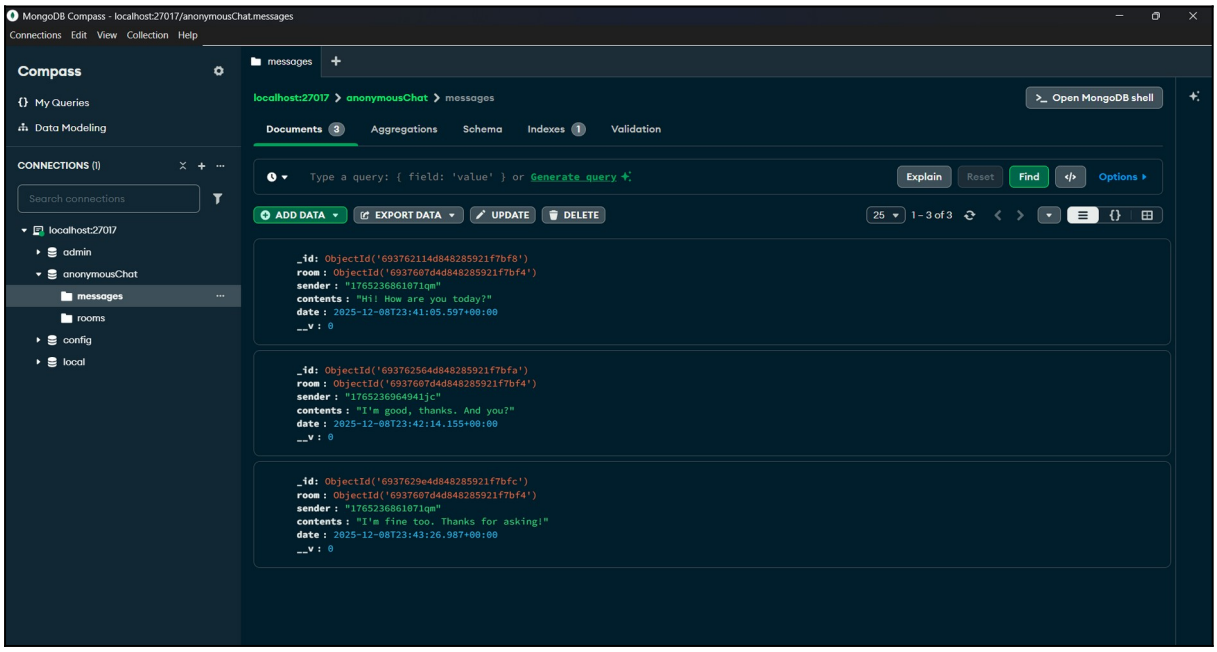


Рис. 37. Вміст messages-колекції (три створені message-документи)

Після завершення тестування та виконання операцій з видалення комунікаційних просторів база даних більше не містить записів у колекції кімнат, оскільки один документ було усунено під час функціонального тестування користувацького інтерфейсу, а інший — у межах тестування прикладного програмного інтерфейсу. Водночас очищено і колекцію повідомлень, оскільки їхні записи були пов’язані саме з тими середовищами, що підлягали видаленню. Таким чином, структура бази даних після завершення всіх операцій відповідає очікуваному стану та підтверджує коректність механізмів очищення.

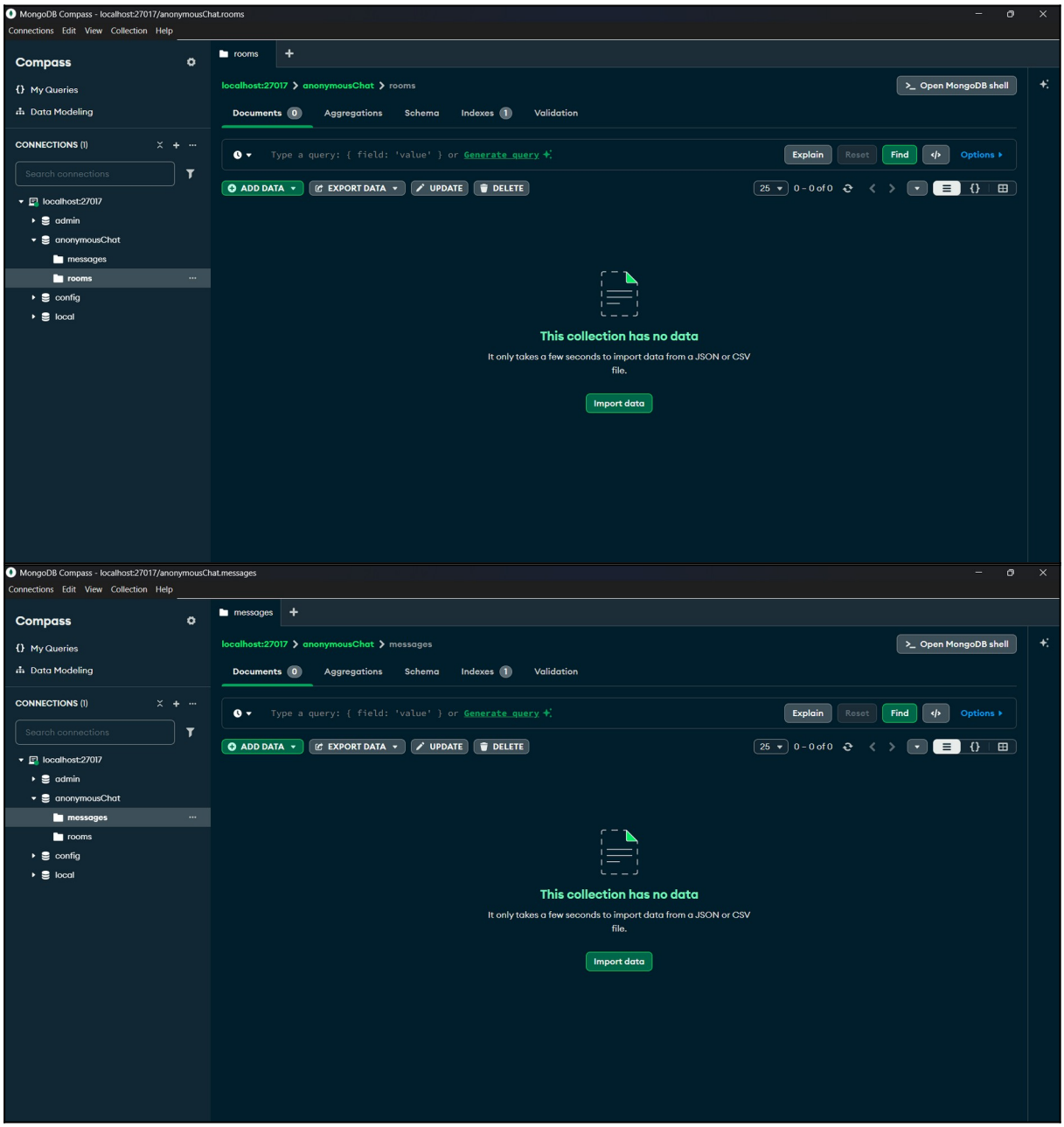


Рис. 38-39. Вміст усіх колекцій бази даних після видалення відповідних документів

4.5. Висновки до розділу 4

Тестування програмного забезпечення підтвердило коректність функціональної логіки веб-системи, стабільність взаємодії між її клієнтською та серверною частинами і правильність оброблення даних на всіх етапах роботи. Послідовне відтворення сценаріїв взаємодії через інтерфейс та прикладний програмний інтерфейс продемонструвало здатність системи коректно виконувати створення, використання та завершення роботи комунікаційних просторів,

забезпечуючи їх синхронізацію в режимі реального часу та збереження фактичних результатів у базі даних. Аналіз стану сховища після завершення тестів засвідчив відповідність отриманих даних очікуваному результату, що свідчить про узгодженість механізмів зберігання, оновлення та видалення інформації. Сукупність проведених тестів підтвердила працездатність програмного продукту та його готовність до подальшої експлуатації.

ВИСНОВКИ

Проведене ноукове дослідження охоплює повний цикл створення веб-орієнтованої інформаційної системи для анонімного текстового спілкування — від теоретичного аналізу предметної області до практичної реалізації та перевірки функціонування програмного забезпечення. У межах виконаної роботи послідовно розглянуто ключові аспекти, що визначають специфіку анонімної комунікації у сучасному цифровому середовищі.

Аналіз предметної області анонімної текстової комунікації дозволив визначити її соціальні та технологічні особливості, а також окреслити фактори, які впливають на поведінку користувачів у подібних системах. Дослідження існуючих програмних аналогів виявило обмеження поширених підходів, зокрема орієнтацію на асинхронну взаємодію або обов’язкову персоніфікацію користувачів, що підтвердило доцільність розроблення веб-системи, заснованої на принципах повної анонімності та синхронної текстової комунікації. На основі результатів аналізу сформульовано вимоги до програмного забезпечення, які охоплюють функціональні, технічні та експлуатаційні характеристики системи. Вимоги враховують необхідність забезпечення взаємодії в режимі реального часу, відсутність процедур реєстрації, коректне керування комунікаційними середовищами та узгоджену роботу клієнтської і серверної частин. Сформований набір вимог став основою для подальшого проектування інформаційної системи.

Проектування програмного забезпечення виконано із застосуванням системного підходу, що дозволило формалізувати логіку взаємодії між користувачами та системою, визначити внутрішні процеси обробки даних і структуру зберігання інформації. Побудовані діаграми прецедентів, діяльності, послідовності, потоків даних, архітектури інформаційної системи та діаграма сутностей і зв’язків забезпечили цілісне уявлення про функціонування системи та створили концептуальну основу для її реалізації.

Реалізація веб-системи виконана з урахуванням вимог до анонімності, модульності та підтримки інтерактивної взаємодії. Серверна та клієнтська частини

логічно розмежовані, а обмін даними організований з використанням як запитно-відповідного підходу, так і постійного двостороннього з’єднання. Запропоноване рішення забезпечує коректну ідентифікацію повідомлень у межах окремих комунікаційних середовищ без використання персональних даних користувачів.

Перевірка працездатності програмного забезпечення підтвердила коректність реалізованих функціональних можливостей та стабільність взаємодії між основними компонентами системи. Тестування виконувалося на рівні користувацького інтерфейсу та серверних інтерфейсів взаємодії, а аналіз стану бази даних після виконання тестових сценаріїв засвідчив цілісність інформації та правильність виконання операцій створення, обміну та видалення даних.

Узагальнюючи результати виконаної роботи, можна зробити висновок, що поставлена мета магістерської кваліфікаційної роботи була досягнута шляхом послідовного вирішення всіх визначених завдань, пов’язаних з аналізом, проектуванням, реалізацією та тестуванням веб-системи анонімного текстового спілкування. Отримані результати підтверджують можливість створення функціонально завершеного та технологічно обґрунтованого програмного забезпечення, яке може використовуватися як самостійне рішення або як основа для подальшого розвитку подібних інформаційних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ACM Digital Library [Електронний ресурс]. — Режим доступу: https://dl.acm.org/

2. Bass L., Clements P., Kazman R. Software Architecture in Practice. — 4th ed. — Boston: Addison-Wesley, 2021. — 560 p.

3. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. — 2nd ed. — Boston: Addison-Wesley, 2005. — 496 p.

4. Chaum D. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms // Communications of the ACM. — 1981. — Vol. 24. — No. 2. — P. 84-90. — Режим доступу: https://dl.acm.org/doi/10.1145/358549.358563

5. Danezis G., Diaz C. A Survey of Anonymous Communication Channels. — Microsoft Research, 2008 [Електронний ресурс]. — Режим доступу: https://www.microsoft.com/en-us/research/publication/a-survey-of-anonymous-communication-channels/

6. Express API Reference [Електронний ресурс]. — Режим доступу: https://expressjs.com/en/4x/api.html

7. Express.js Guide — Routing [Електронний ресурс]. — Режим доступу: https://expressjs.com/en/guide/routing.html

8. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. — 3rd ed. — Boston: Addison-Wesley, 2004. — 208 p.

9. Git — Documentation [Електронний ресурс]. — Режим доступу: https://git-scm.com/docs

10. Green M., Smith J. Anonymity and Privacy in Online Communication Systems // Journal of Information Security. — 2016.

11. IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications [Електронний ресурс]. — Режим доступу: https://standards.ieee.org/standard/830-1998.html

12. IEEE Xplore Digital Library [Електронний ресурс]. — Режим доступу: https://ieeexplore.ieee.org/Xplore/home.jsp

13. ISO/IEC 12207:2017. Systems and software engineering — Software life cycle processes [Електронний ресурс]. — Режим доступу: https://www.iso.org/standard/63711.html

14. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) [Електронний ресурс]. — Режим доступу: https://www.iso.org/standard/35733.html

15. Kleppmann M. Designing Data-Intensive Applications. — Sebastopol: O'Reilly, 2017. — 616 p.

16. MDN Web Docs — Fetch API [Електронний ресурс]. — Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

17. MDN Web Docs — Web Architecture [Електронний ресурс]. — Режим доступу: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Architecture

18. MDN Web Docs — WebSockets API [Електронний ресурс]. — Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

19. Microsoft Edge for Developers [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/microsoft-edge/

20. MongoDB Compass — Manual [Електронний ресурс]. — Режим доступу: https://www.mongodb.com/docs/compass/current/

21. MongoDB CRUD Operations [Електронний ресурс]. — Режим доступу: https://www.mongodb.com/docs/manual/crud/

22. MongoDB Manual — Getting Started [Електронний ресурс]. — Режим доступу: https://www.mongodb.com/docs/manual/introduction/

23. Mongoose Guide — Getting Started [Електронний ресурс]. — Режим доступу: https://mongoosejs.com/docs/index.html

24. Mongoose Schemas and Models [Електронний ресурс]. — Режим доступу: https://mongoosejs.com/docs/guide.html

25. Node.js Documentation — About API [Електронний ресурс]. — Режим доступу: https://nodejs.org/api/

26. Node.js Guides — Getting Started [Електронний ресурс]. — Режим доступу: https://nodejs.org/en/docs/guides/

27. Pfitzmann A., Hansen M. A Terminology for Talking About Privacy by Data Minimization. — Technical Report, 2010 [Електронний ресурс]. — Режим доступу: https://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.31.pdf

28. Postman Learning Center — API Testing [Електронний ресурс]. — Режим доступу: https://learning.postman.com/docs/getting-started/introduction/

29. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach. — 9th ed. — New York: McGraw-Hill, 2020. — 960 p.

30. React API Reference [Електронний ресурс]. — Режим доступу: https://react.dev/reference/react

31. React Documentation — Main Concepts [Електронний ресурс]. — Режим доступу: https://react.dev/learn

32. SCSS Reference — Sass Basics [Електронний ресурс]. — Режим доступу: https://sass-lang.com/guide

33. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. — 7th ed. — New York: McGraw-Hill, 2020. — 1376 p.

34. Socket.IO Client API [Електронний ресурс]. — Режим доступу: https://socket.io/docs/v4/client-api/

35. Socket.IO Documentation — Introduction [Електронний ресурс]. — Режим доступу: https://socket.io/docs/v4/

36. Sommerville I. Software Engineering. — 10th ed. — Boston: Pearson, 2016. — 816 p.

37. Stack Overflow Developer Insights [Електронний ресурс]. — Режим доступу: https://insights.stackoverflow.com/survey

38. Tanenbaum A. S., van Steen M. Distributed Systems: Principles and Paradigms. — 2nd ed. — Upper Saddle River: Pearson, 2007. — 686 p.

39. Visual Studio Code — User Guide [Електронний ресурс]. — Режим доступу: https://code.visualstudio.com/docs

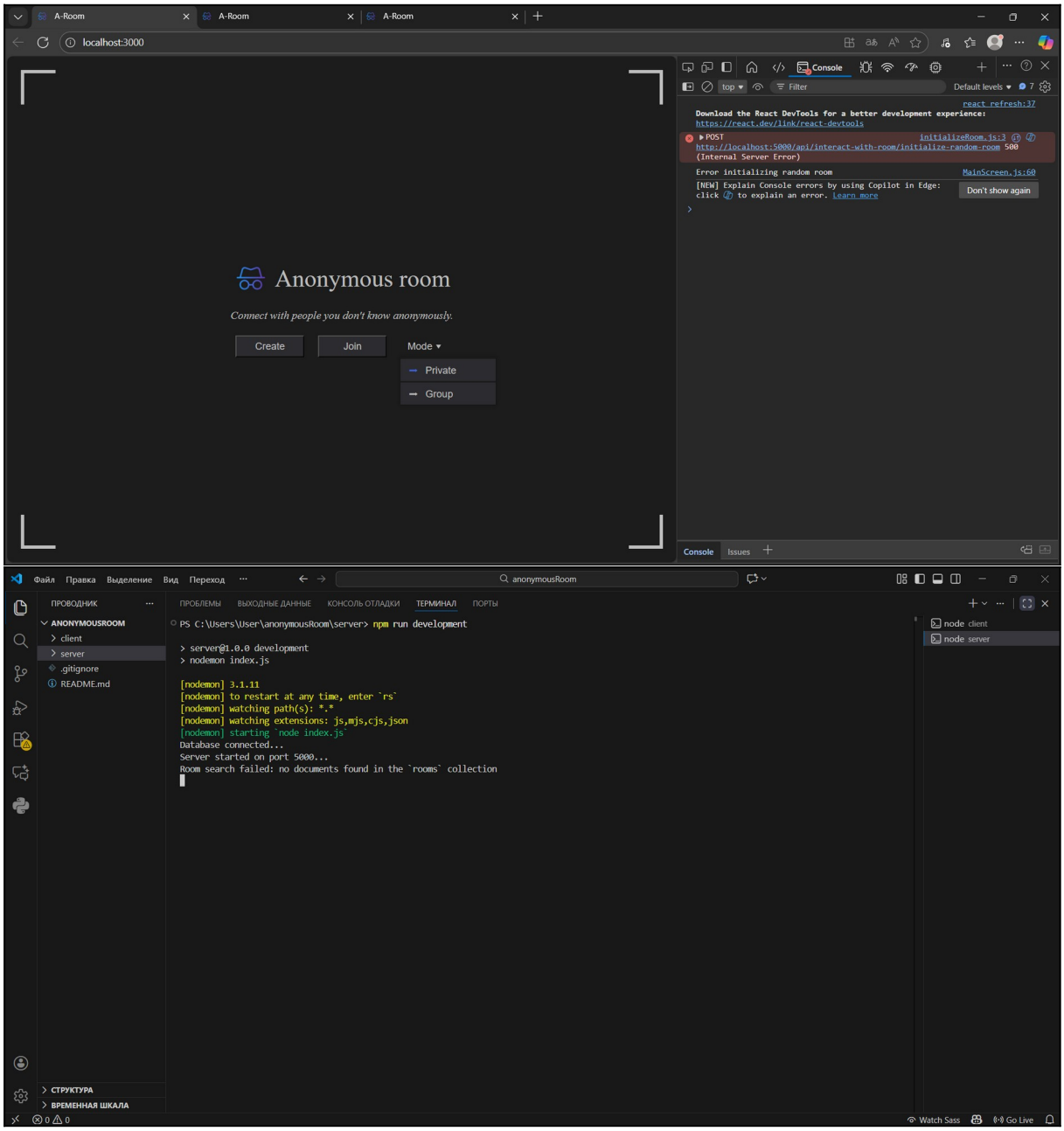
40. W3C Web Architecture [Електронний ресурс]. — Режим доступу: https://www.w3.org/TR/webarch/

ДОДАТКИ

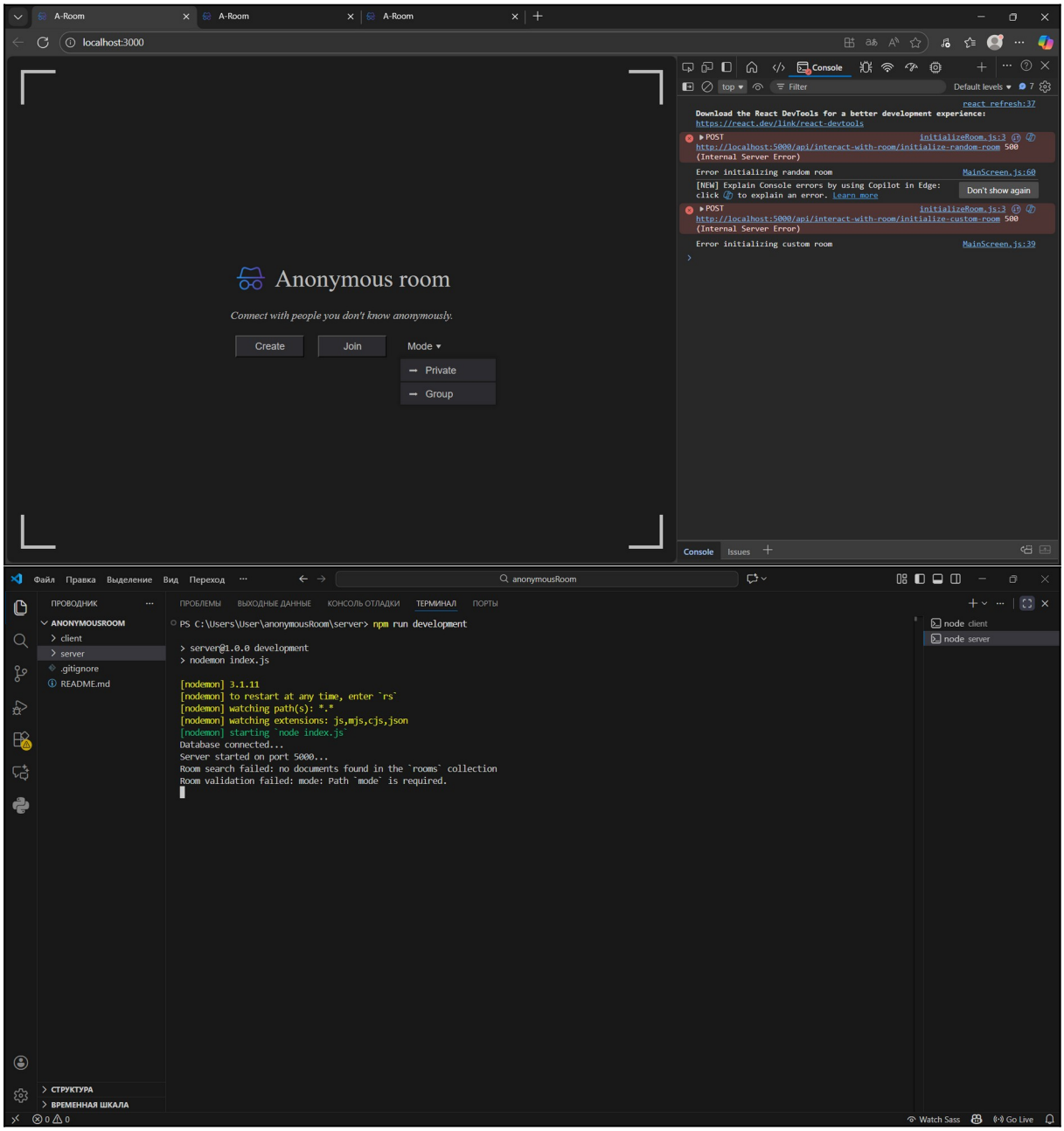
Додаток 1

Функціональне UI-тестування

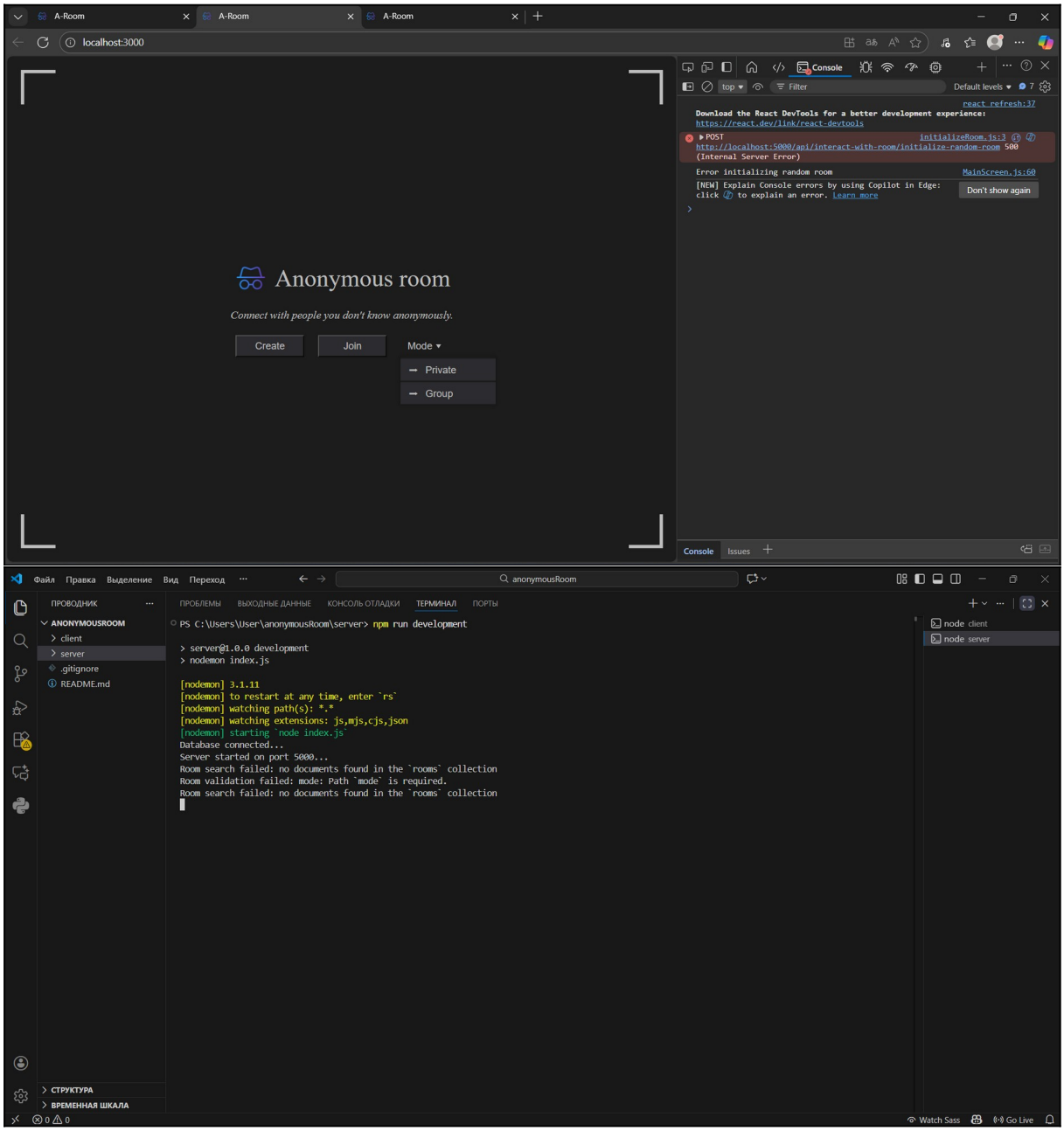
Спроба ініціалізації випадкової кімнати (приєднання) за відсутності кімнат



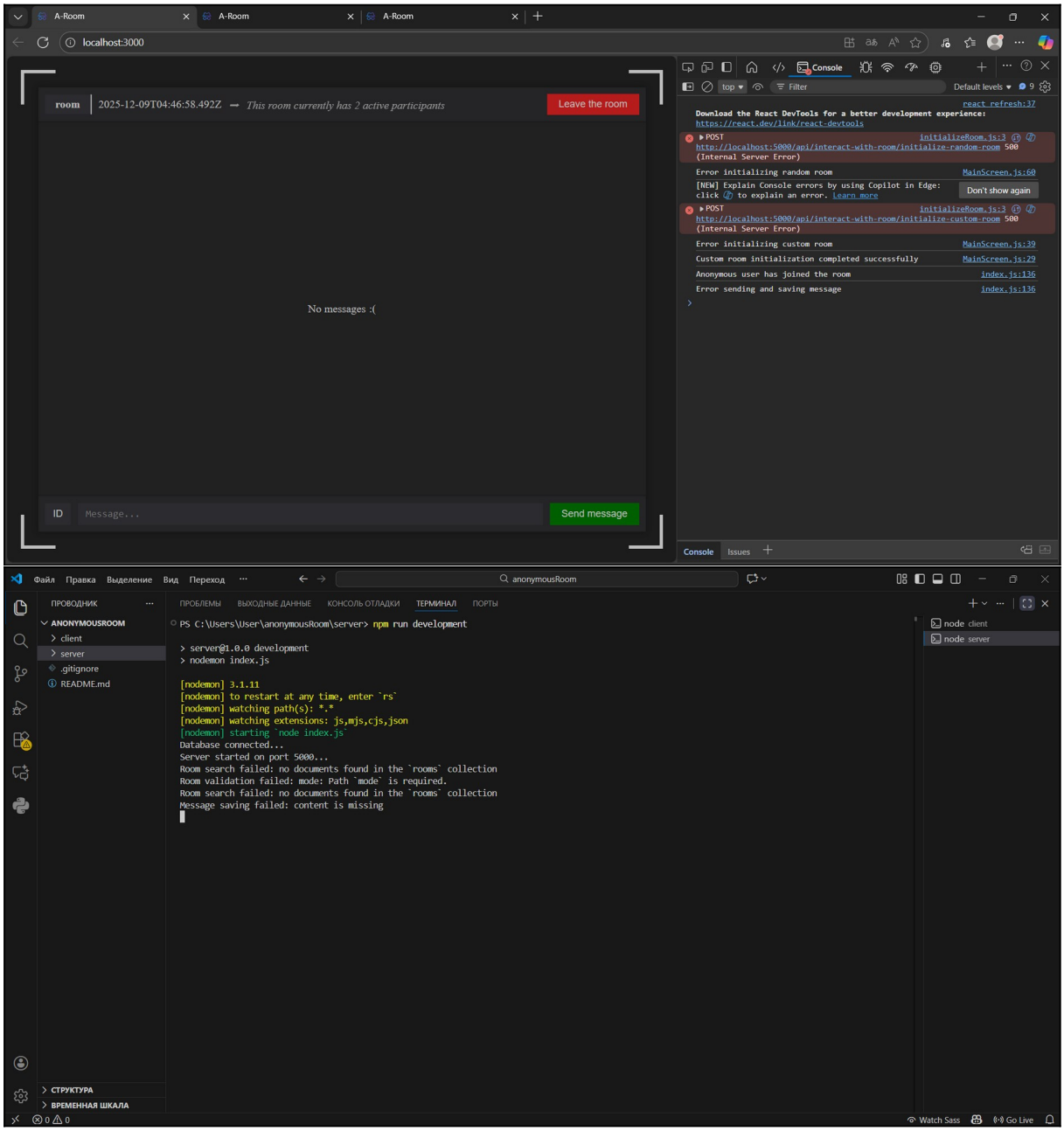
Спроба ініціалізації custom-кімнати (створення) без вибору режиму



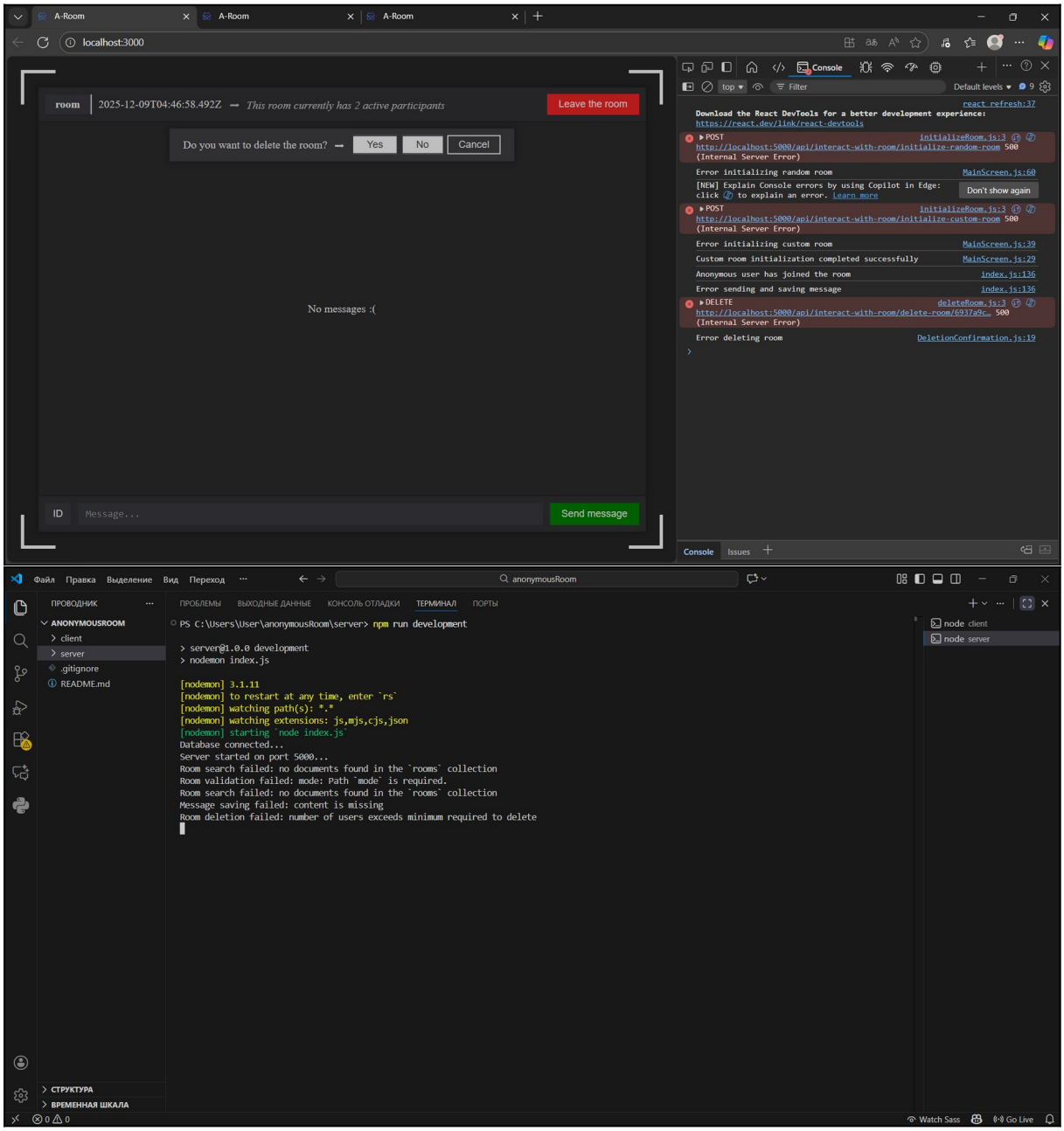
Спроба ініціалізації випадкової кімнати (приєднання) без вибору режиму



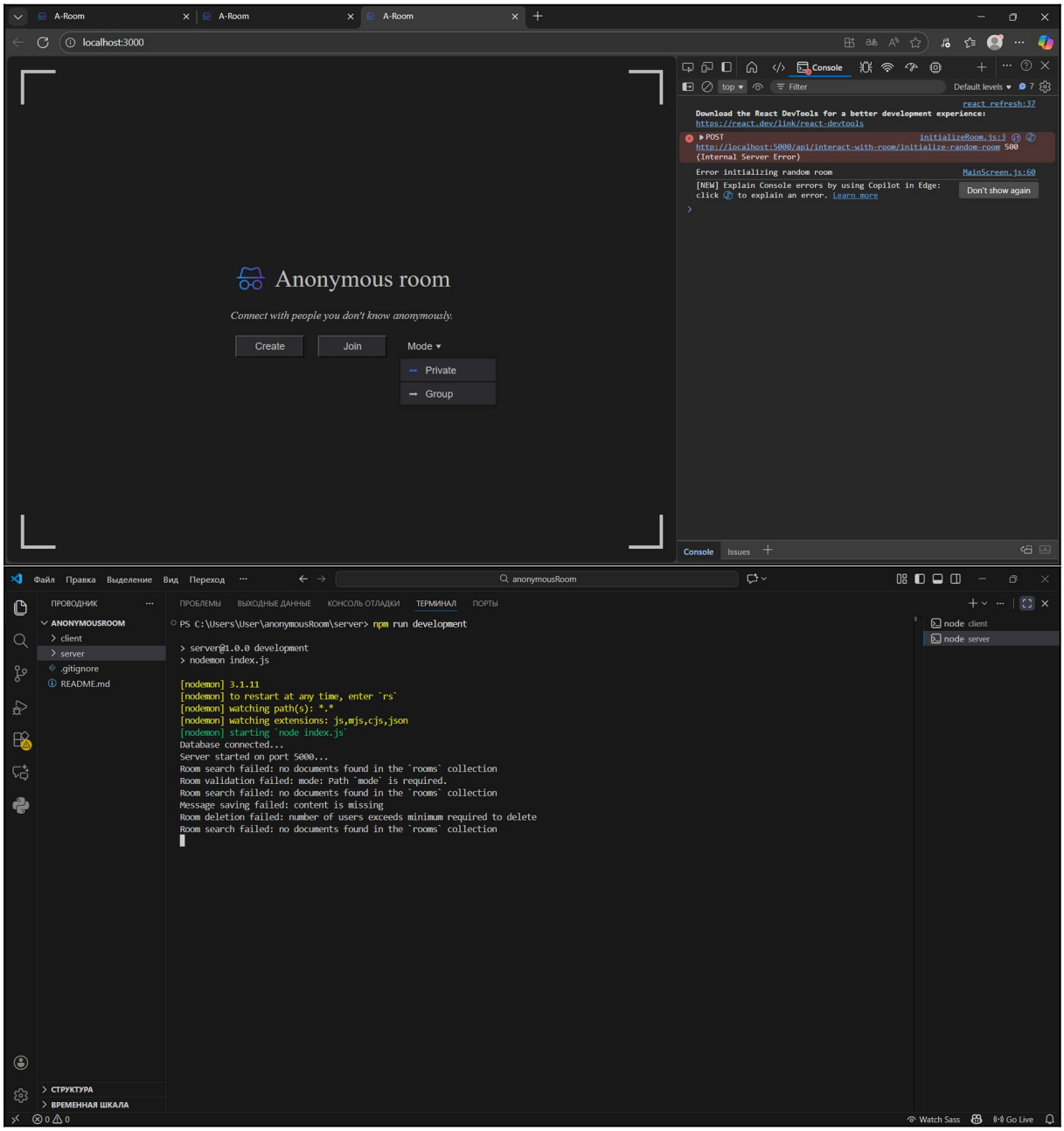
Спроба відправлення порожнього повідомлення в кімнаті



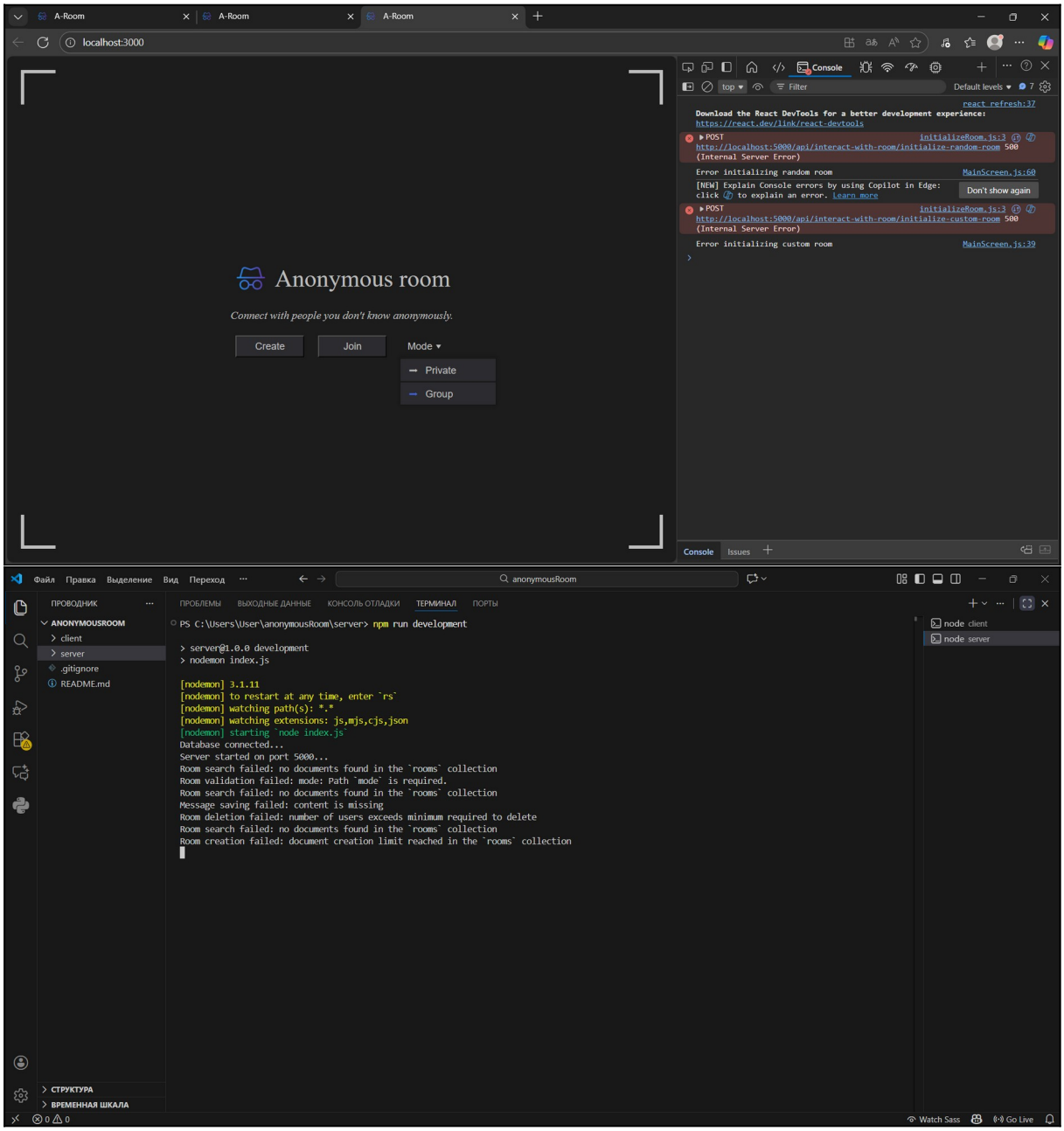
Спроба видалення кімнати незважаючи на обмеження



Спроба ініціалізації випадкової кімнати (приєднання) незважаючи на обмеження



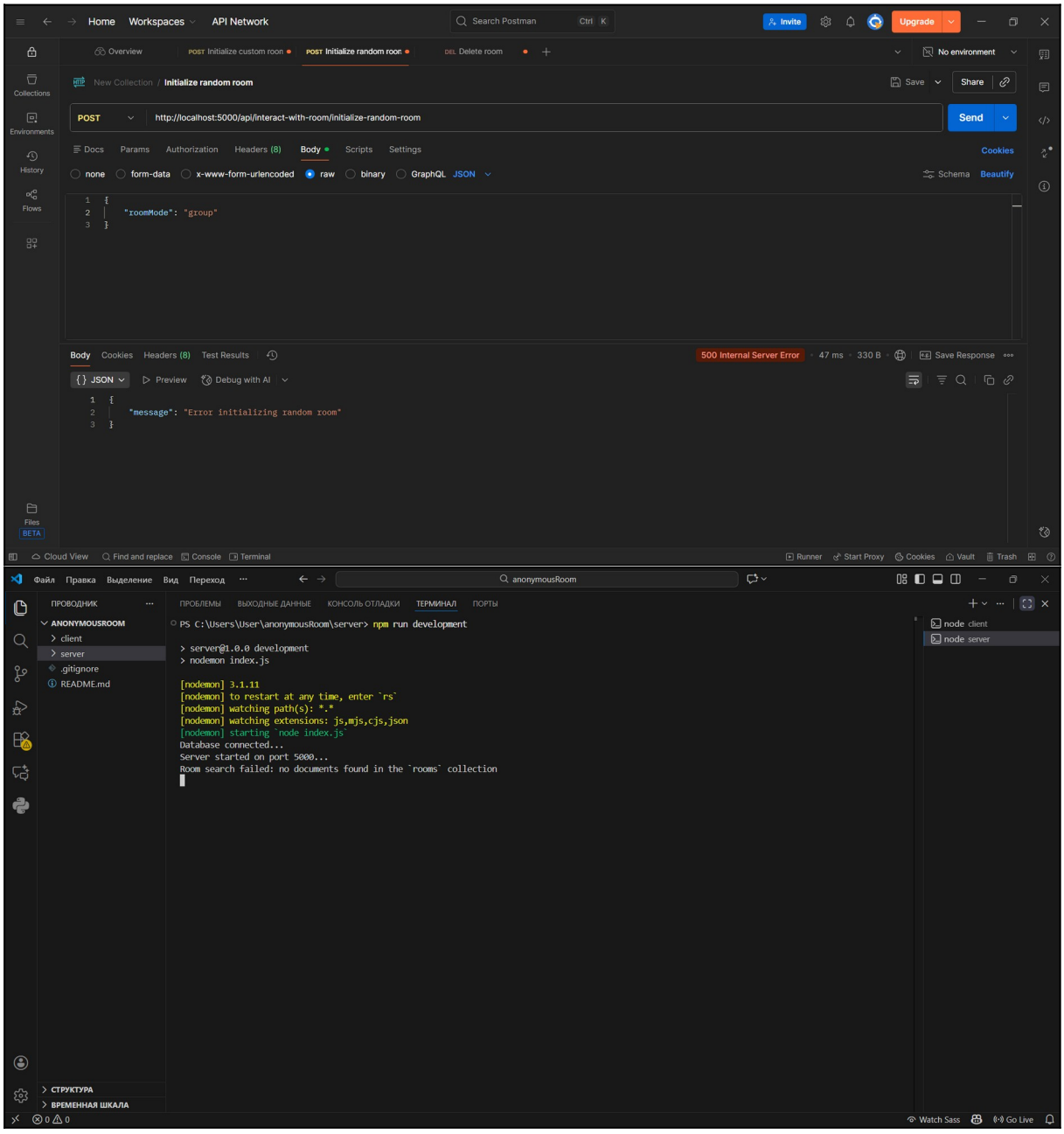
Спроба ініціалізації custom-кімнати (створення) незважаючи на обмеження



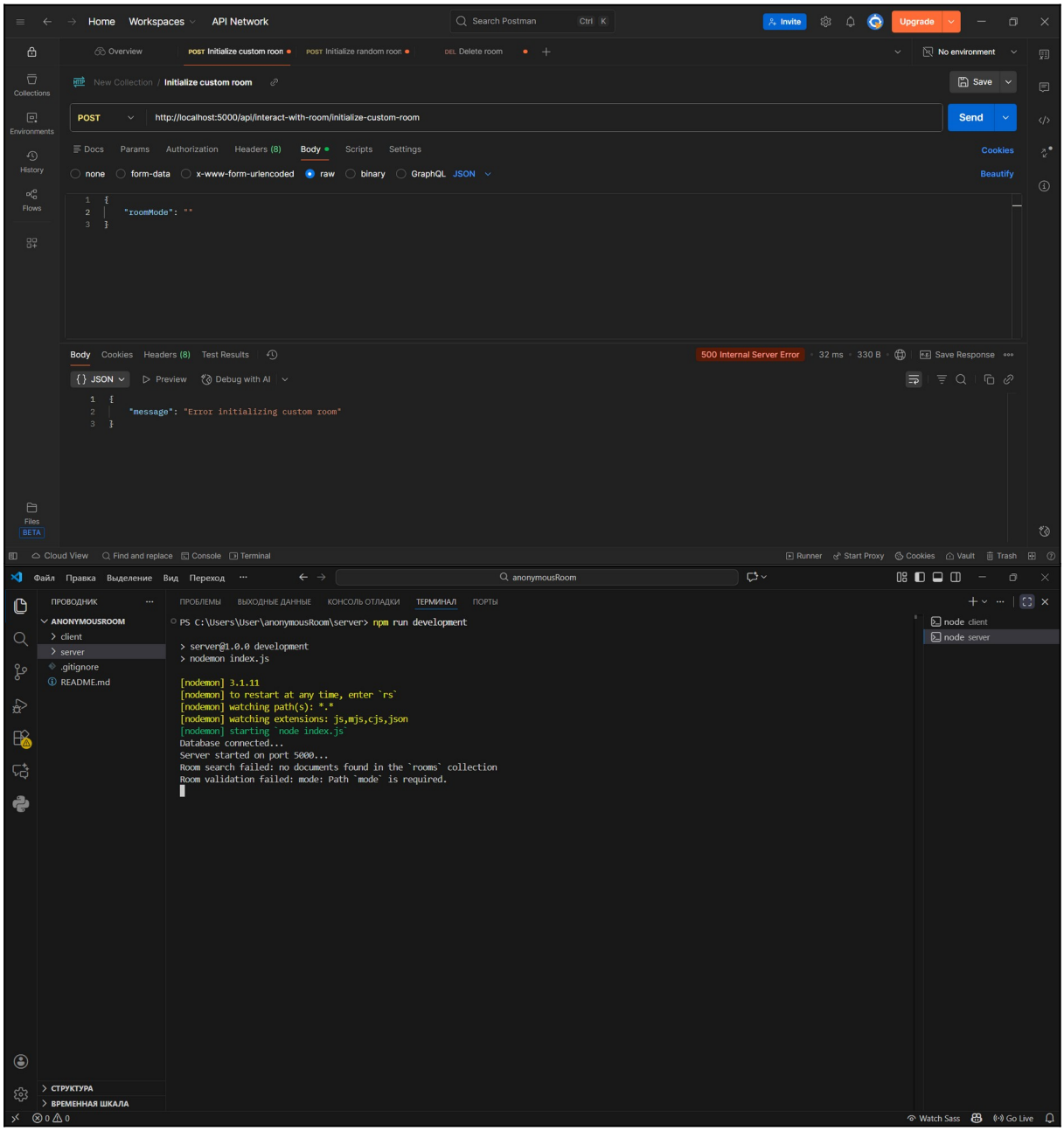
Додаток 2

Тестування API з використанням Postman

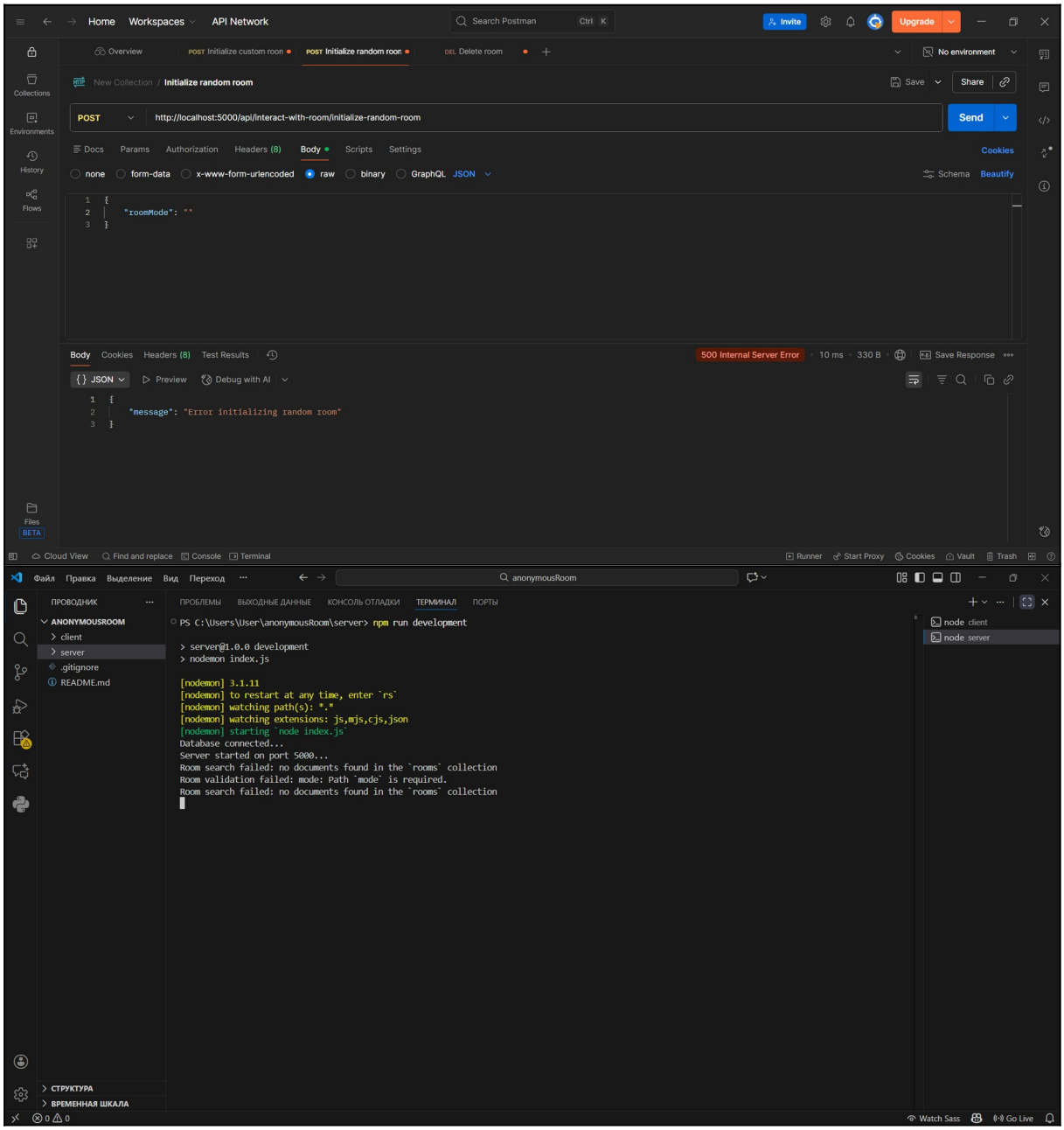
Спроба ініціалізації випадкової кімнати (приєднання) за відсутності кімнат



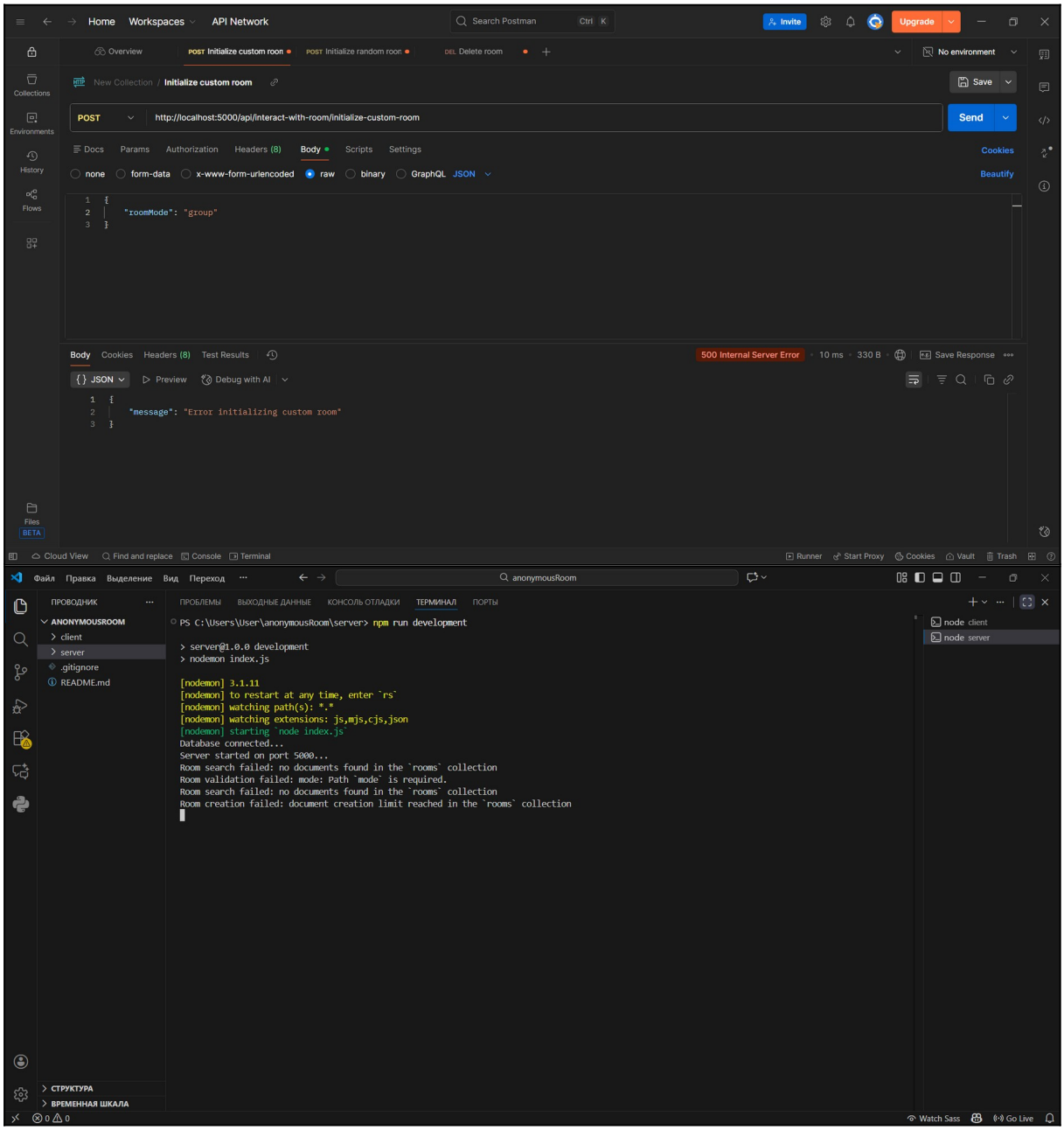
Спроба ініціалізації custom-кімнати (створення) без зазначення режиму



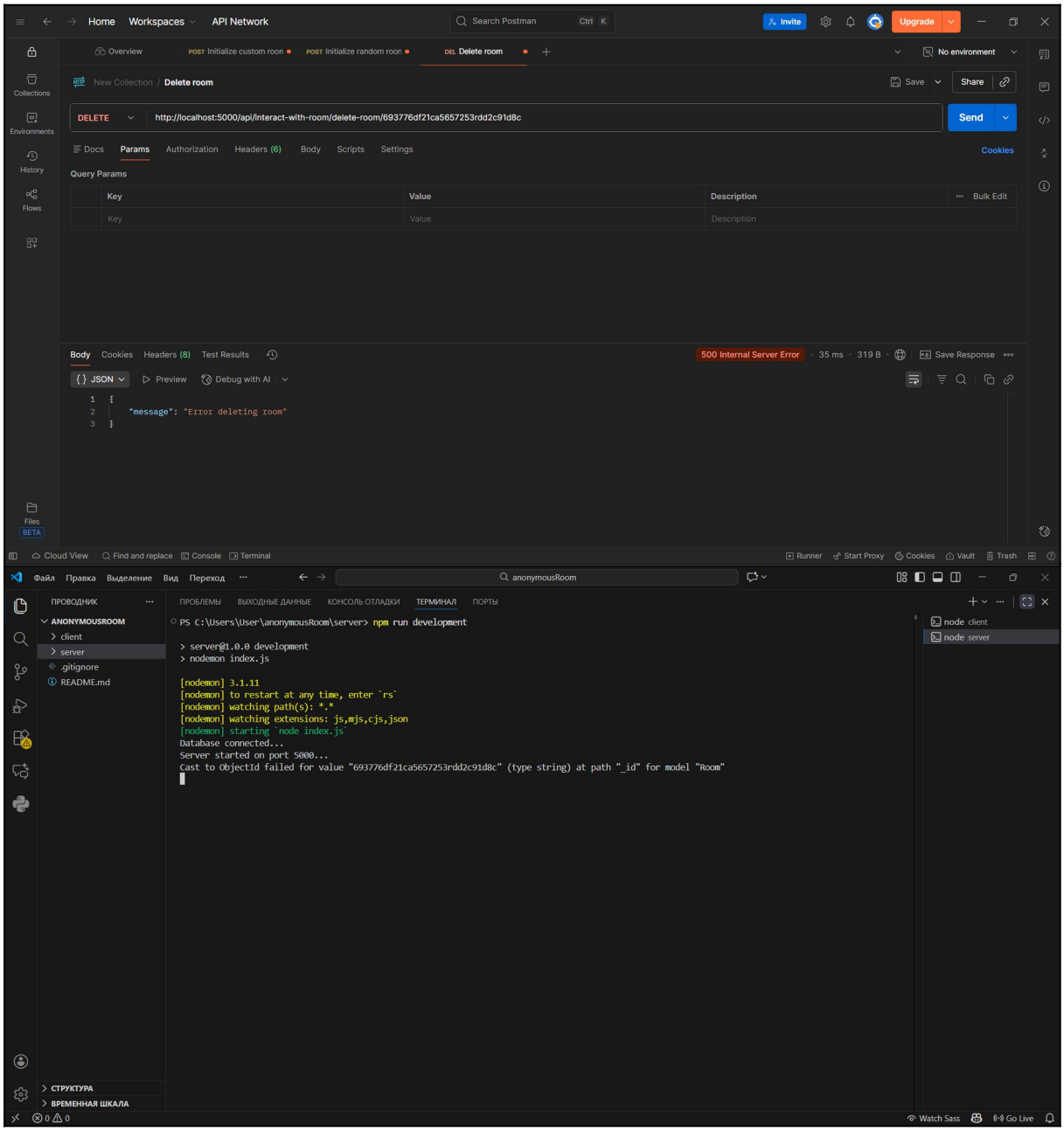
Спроба ініціалізації випадкової кімнати (приєднання) без зазначення режиму



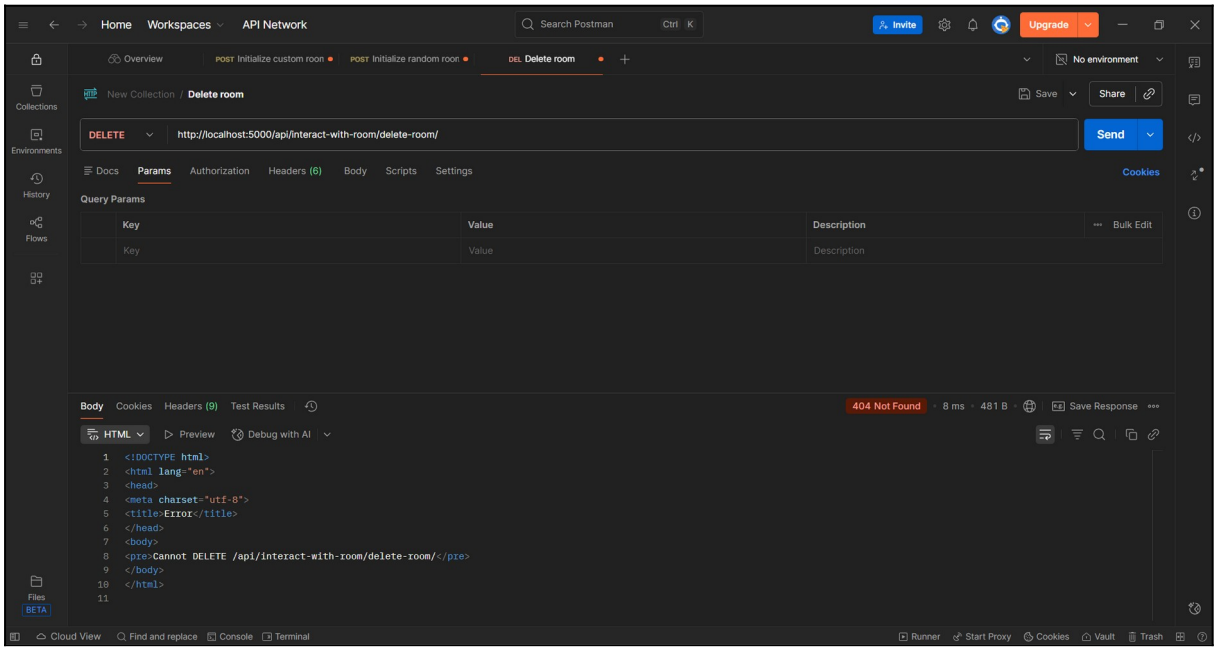
Спроба ініціалізації custom-кімнати (створення) незважаючи на обмеження



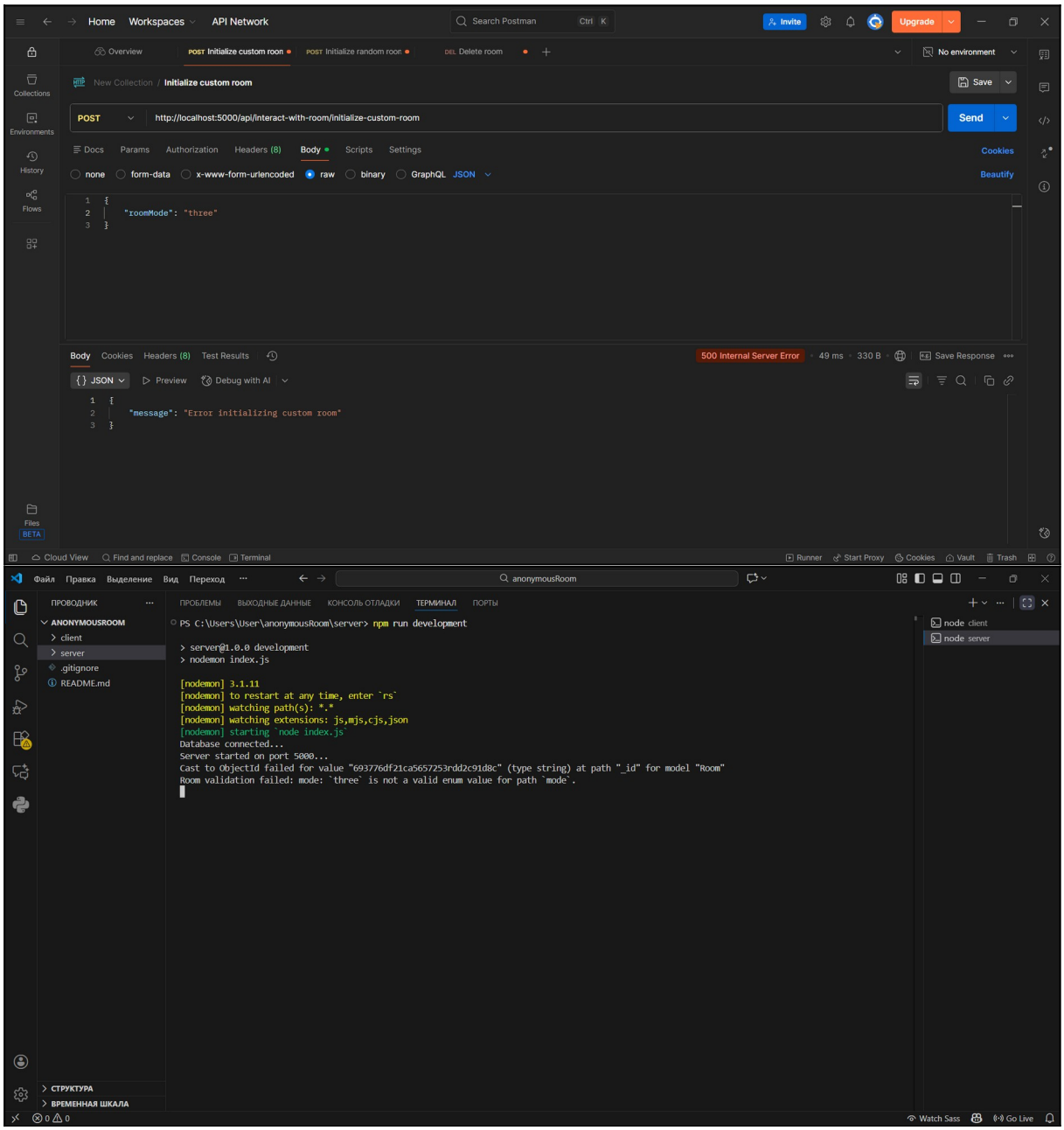
Спроба видалення неіснуючої кімнати (неправильно вказаний ідентифікатор)



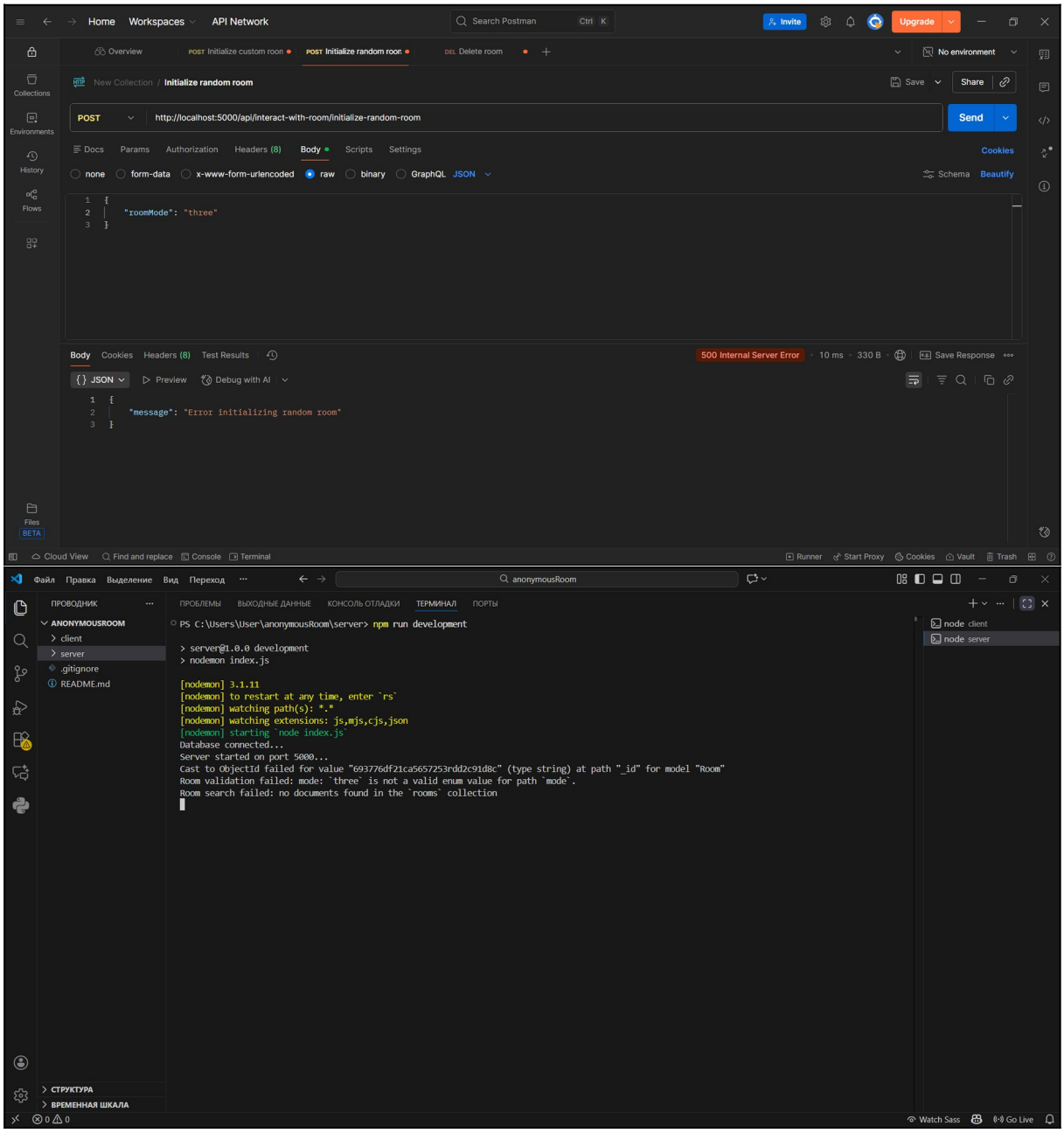
Спроба видалити кімнату не вказуючи ідентифікатор



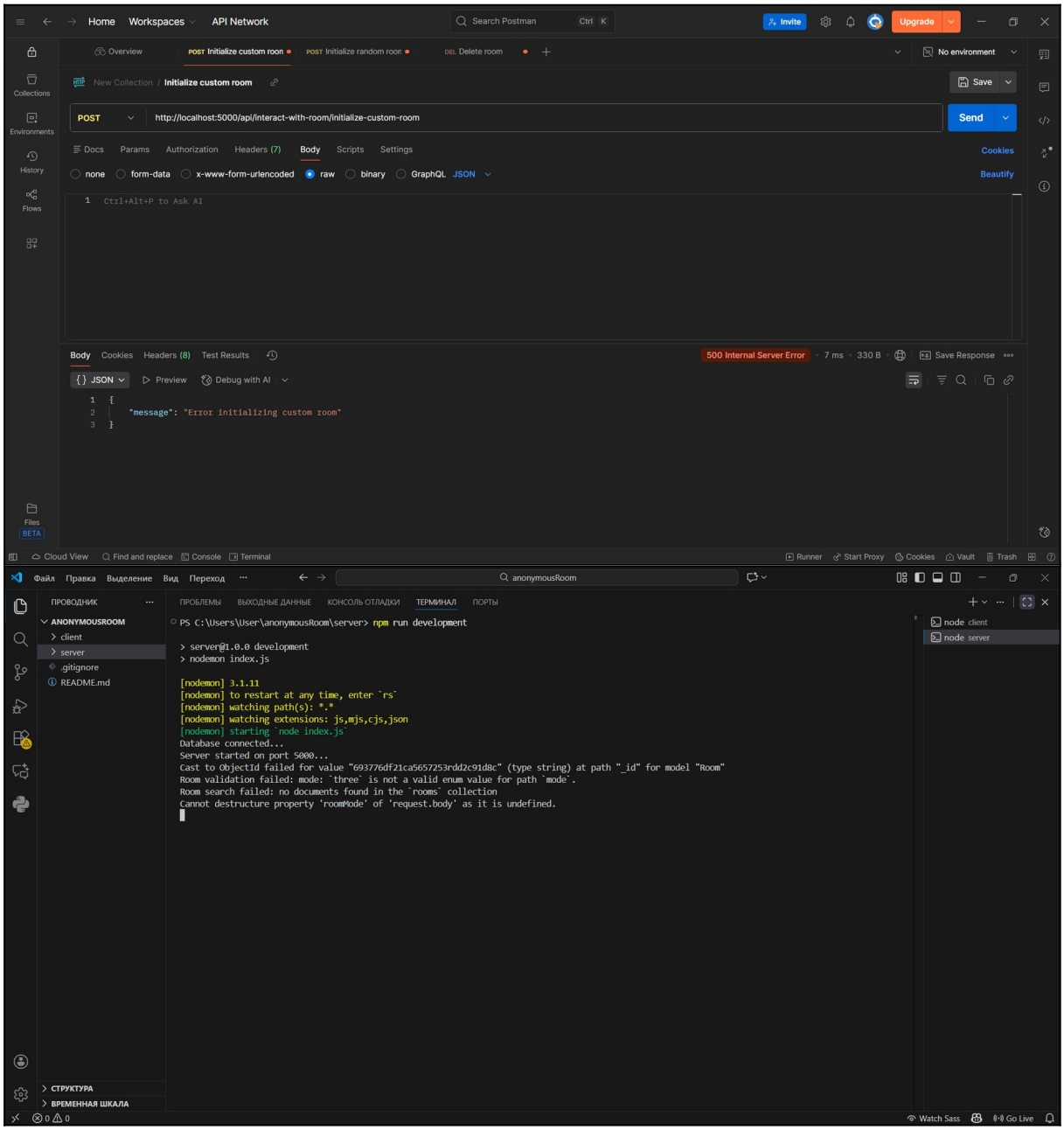
Спроба ініціалізації custom-кімнати (створення) з незареєстрованим режимом



Спроба ініціалізації випадкової кімнати (приєднання) з незареєстрованим режимом



Спроба ініціалізації custom-кімнати (створення) без тіла запиту



Спроба ініціалізації випадкової кімнати (приєднання) без тіла запиту