Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-месенджера у реальному часі з використанням Node.js, Typescript та MongoDB

Виконав(ла): студент(ка) IV курсу, групи СП-41
спеціальності 121
«Інженерія програмного забезпечення»
(шифр і назва спеціальності)

		Бурда О. Б.
	(підпис)	(прізвище та ініціали)
Керівник		Мудрик І. Я.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Стоянов Ю. М.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М. Р.
	(підпис)	(прізвище та ініціали)
Рецензент		
	(підпис)	(прізвище та ініціали)

Тернопіль
2025

АНОТАЦІЯ

Розробка веб-месенджера у реальному часі з використанням Node.js, Typescript та MongoDB // Кваліфікаційна робота освітнього рівня «Бакалавр» // Бурда Олександр Богданович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-41 // Тернопіль, 2025 // Сторінок – 65, рисунків – 22, таблиць – 14, додатків – 3, посилань – 21, лістингів – 1.

Ключові слова: архітектура програмного забезпечення, веб-месенджер, вебсокети, монорепозиторій, Node.js, Prisma, React, Redis, tRPC, TypeScript.

Метою роботи є дослідження сучасних архітектурних патернів, проектування та розробка високопродуктивної програмної системи для комунікації в реальному часі.

Методи розробки базуються на застосуванні монорепозиторної архітектури, мови програмування TypeScript, платформи Node.js та бібліотеки React. Взаємодія між клієнтом та сервером реалізована за допомогою фреймворку tRPC, а комунікація в реальному часі забезпечується через Вебсокети у поєднанні з брокером повідомлень Redis.

В результаті роботи було розроблено програмну реалізацію веб-месенджера, що представляє собою клієнт-серверний додаток з наскрізною типізацією. Система включає серверну частину на Node.js з API на базі tRPC, клієнтську частину на React та декаплінговану архітектуру реального часу, що використовує Redis Pub/Sub.

ABSTRACT

Development of a real-time web messenger using Node.js, TypeScript, and MongoDB // Bachelor's Qualification Thesis // Burda Oleksandr Bogdanovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, Group SP-41 // Ternopil, 2025 // Pages – 65, figures – 22, tables – 14, appendices – 3, references – 21, listings – 1.

Keywords: software architecture, web messenger, WebSockets, monorepo, Node.js, Prisma, React, Redis, tRPC, TypeScript

The objective of the thesis is the investigation of modern architectural patterns, and the design and development of a high-performance software system for real-time communication.

The development methods are based on the use of a monorepo architecture, the TypeScript programming language, the Node.js platform, and the React library. Client-server interaction is implemented using the tRPC framework, while real-time communication is ensured via WebSockets combined with the Redis message broker.

As a result of the work, a software implementation of a web messenger was developed, which is a client-server application with end-to-end type safety. The system includes a Node.js backend with a tRPC-based API, a React frontend, and a decoupled real-time architecture that utilizes Redis Pub/Sub.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

tRPC (англ. TypeScript Remote Procedure Call) – бібліотека для побудови типізованих API, яка дозволяє викликати функції на сервері з клієнта без необхідності ручного створення HTTP-запитів.

API (англ. Application Programing Interface) – прикладний програмний інтерфейс.

JSON (англ. JavaScript Object Notation) – нотація об’єкту JavaScript.

ORM (англ. Object-relational mapping) – технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».

Pub/Sub (англ. Publish/Subscribe) – шаблон обміну повідомленнями, в якому відправники (видавці, publishers) не надсилають повідомлення безпосередньо отримувачам (підписникам, subscribers).

Декаплінгована система (англ. Decoupled System) – це архітектурний підхід, при якому компоненти системи (модулі, сервіси) розробляються так, щоб бути максимально незалежними один від одного. Взаємодія між ними відбувається через чітко визначені та стабільні інтерфейси (наприклад, API), а не через прямі виклики чи залежності від внутрішньої реалізації інших компонентів.

PNPM (англ. Performant Node Package Manager) – менеджер пакетів для JavaScript, який є альтернативою NPM та Yarn.

PK (англ. Primary Key) – первинний ключ; унікальний ідентифікатор, який використовується для однозначного визначення кожного документа в колекції.

FK (англ. Foreign Key) – зовнішній ключ; поле або набір полів у документі, яке посилається на первинний ключ (PK) іншого документа, дозволяючи створювати логічні зв'язки між даними у різних колекціях.

ЗМІСТ

ВСТУП 8
1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ВЕБ-МЕСЕНДЖЕРІВ 10
1.1 Аналіз предметної області веб-месенджерів 10
1.2 Формування вимог до веб-месенджера для комунікації в реальному часі . 11
1.3 Опис варіантів використання веб-месенджера 12
1.4 Вибір технологій розробки системи 14
2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБ-МЕСЕНДЖЕРА 16
2.1 Проектування бази даних 16
2.2 Моделювання архітектури системи 31
2.3 Розробка серверної частини 36
2.4 Розробка клієнтської частини 40
2.5 Тестування системи 43
3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ 49
3.1 Фізіологічний вплив факторів існування на життєдіяльність людини 49
3.2 Підбирання оптимальних параметрів мікроклімату на робочих місцях 51
ВИСНОВКИ 54
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 56
ДОДАТКИ 58
ДОДАТОК А – ТЕЗИ КОНФЕРЕНЦІЇ 59
ДОДАТОК Б – ЛІСТИНГ ФУНКЦІЇ “WITHRATELIMIT” ДЛЯ ОБМЕЖЕННЯ ЧАСТОТИ ЗАПИТІВ 62
ДОДАТОК В – ДИСК З РОБОТОЮ 65

ВСТУП

У сучасному світі цифрових технологій комунікація в інтернеті стала невід'ємною частиною повсякденного та професійного життя. Веб-месенджери, що забезпечують миттєвий обмін інформацією, відіграють ключову роль у підтримці соціальних та робочих зв'язків. Зі зростанням вимог до швидкості, надійності та безпеки, виникає потреба у створенні архітектурно досконалих та масштабованих комунікаційних платформ, що використовують передові інженерні практики.

Дана кваліфікаційна робота присвячена комплексному дослідженню, проектуванню та реалізації високопродуктивного веб-месенджера для комунікації в реальному часі. В основі проекту лежить сучасна монорепозиторна архітектура, керована за допомогою Turborepo, яка дозволила забезпечити високий рівень узгодженості коду. Технологічний стек було обрано з метою досягнення максимальної надійності та ефективності розробки: серверна частина реалізована на Node.js з використанням TypeScript, а клієнтська – на бібліотеці React. Ключовою особливістю є синергія між tRPC та Prisma ORM, що забезпечує наскрізну типізацію від бази даних MongoDB до кінцевого користувацького інтерфейсу.

Центральним елементом архітектури є декаплінгована система комунікації в реальному часі, побудована за патерном «Видавець-Підписник» (Publisher/Subscriber). Замість прямої передачі даних, вона використовує Вебсокети у поєднанні з брокером повідомлень Redis Pub/Sub. Такий підхід дозволяє відокремити логіку обробки запитів від логіки доставки подій, що є критично важливим для горизонтального масштабування та відмовостійкості системи. Використання Prisma ORM як шару абстракції над базою даних MongoDB дозволило не лише спростити запити, але й автоматично генерувати типи, що є фундаментом для наскрізної типізації tRPC.

Робота охоплює повний цикл розробки: від проектування моделі даних та

системної архітектури до послідовної реалізації функціональних модулів та їх тестування. Було реалізовано ключовий функціонал, що включає систему автентифікації, механізм соціальних зв'язків, групову комунікацію з ролями та каналами, а також обмін особистими повідомленнями. Особливу увагу було приділено реалізації складних інженерних завдань, таких як безпечне завантаження файлів у хмарне сховище та впровадження системи обмеження частоти запитів для захисту від зловживань.

У процесі розробки було застосовано ітеративний підхід, що дозволило послідовно нарощувати функціонал, забезпечуючи стабільність системи на кожному етапі. Фінальним етапом стало комплексне тестування, що включало модульне тестування серверної логіки за допомогою Jest та тестування ключових користувацьких сценаріїв для перевірки наскрізної функціональності системи.

Цей проект не лише демонструє створення функціонального програмного продукту, але й слугує практичним дослідженням ефективності сучасних архітектурних патернів та технологій для побудови складних веб-додатків, які маніпулюють даними в умовах реального часу. Отриманий досвід та знання є цінним внеском у професійний розвиток та можуть бути використані для подальшого вдосконалення та розширення подібних систем.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ВЕБ-МЕСЕНДЖЕРІВ

1.1 Аналіз предметної області веб-месенджерів

Веб-месенджери стали невід'ємним інструментом цифрової комунікації, що забезпечує швидкий обмін інформацією та підтримку соціальних зв’язків як у професійній діяльності, так і в особистому житті. З розвитком технологій вони еволюціонували у багатофункціональні платформи, які об'єднують текстову, голосову та відео-комунікацію, а також інтеграцію з іншими сервісами. Проте, незважаючи на різноманіття існуючих рішень, вибір оптимальної платформи залишається викликом для багатьох організацій та спільнот. Часто наявні месенджери є складними у використанні, не гарантують належного рівня безпеки даних або мають суттєві функціональні обмеження у безкоштовних версіях.

Ці недоліки зумовлюють актуальність розробки нових комунікаційних платформ, здатних задовольнити зростаючі потреби користувачів. Створення веб-месенджера на основі сучасного технологічного стеку, що включає tRPC, вебсокети, Node.js, TypeScript та MongoDB, дозволяє вирішити ключові проблеми існуючих аналогів, забезпечивши високу продуктивність, масштабованість та надійний захист даних [1, 3]. Для того, щоб створити конкурентоспроможний продукт, необхідно провести детальний аналіз провідних гравців ринку, визначити їхні сильні та слабкі сторони та зрозуміти потреби цільової аудиторії.

Однією з найпопулярніших платформ для корпоративної комунікації є Slack. Її перевагами є зручна організація спілкування через тематичні канали, широкі можливості інтеграції з такими сервісами, як Google Drive та Trello, а також підтримка ботів для автоматизації робочих процесів. Однак головним недоліком Slack є значне обмеження функціоналу у безкоштовній версії, що може стати перешкодою для невеликих команд.

Іншим популярним рішенням є Discord, який, попри початкову орієнтацію на геймерів, здобув популярність серед різноманітних спільнот. Платформа забезпечує високу якість голосової та відео-комунікації, дозволяє створювати

сервери з гнучкою системою каналів та налаштовувати права доступу для учасників. Більшість функцій Discord є безкоштовними, але його функціональність та імідж менш орієнтовані на формальне корпоративне використання.

У свою чергу, Google Chat, як частина екосистеми Google Workspace, орієнтований переважно на корпоративних клієнтів. Глибока інтеграція з іншими сервісами Google, такими як Gmail та Google Drive, забезпечує зручність у роботі з даними та високий рівень безпеки. Проте його функціонал є обмеженим для користувачів, які не входять до екосистеми Google Workspace, що знижує його універсальність.

Таким чином, проведений аналіз Slack, Discord та Google Chat демонструє, що кожна з платформ займає свою нішу, маючи як унікальні переваги, так і суттєві недоліки. Slack є ефективним для корпоративних команд, Discord – для спільнот, що потребують якісного голосового зв'язку, а Google Chat – для організацій, інтегрованих у середовище Google. Це створює ринкову можливість для розробки нового веб-месенджера, який би поєднав переваги існуючих рішень, уникнувши їхніх недоліків, та запропонував користувачам збалансовану, безпечну і функціональну платформу.

1.2 Формування вимог до веб-месенджера для комунікації в реальному часі

Розробка веб-месенджера для спілкування в реальному часі вимагає фундаментального підходу до планування. Ключовим етапом є ретельне визначення функціональних, нефункціональних та інтерфейсних вимог. Саме цей аналіз закладає основу для створення платформи, яка буде не лише технічно надійною, але й орієнтованою на користувача, забезпечуючи інтуїтивну та ефективну взаємодію.

Однією з основних функціональних вимог є забезпечення миттєвого обміну

повідомленнями між користувачами. Реактивність системи має гарантувати швидке оновлення даних без необхідності перезавантаження сторінки. Це досягається завдяки використанню вебсокетів, які забезпечують двосторонній зв’язок між клієнтом і сервером. Крім того, важливо передбачити можливість групового спілкування, що дозволяє користувачам взаємодіяти в рамках спільних чатів.

Інтуїтивно зрозумілий дизайн є важливою нефункціональною вимогою. Інтерфейс веб-месенджера повинен бути простим у використанні, з чіткою організацією елементів, що дозволяє користувачам легко орієнтуватися в системі. Особливу увагу слід приділити адаптивності дизайну, щоб забезпечити зручність використання на різних пристроях, включаючи смартфони, планшети та комп’ютери.

Продуктивність веб-месенджера визначається швидкістю обробки повідомлень та стабільністю роботи системи при великій кількості користувачів [7, 11]. Важливо забезпечити масштабованість платформи, щоб вона могла ефективно функціонувати навіть при значному зростанні навантаження. Це включає оптимізацію роботи бази даних, яка повинна швидко обробляти запити та забезпечувати доступ до інформації без затримок.

1.3 Опис варіантів використання веб-месенджера

Діаграма варіантів використання є важливим інструментом для візуалізації того, як система буде використовуватися різними категоріями користувачів, а також для визначення функціональних вимог [12]. Вона дозволяє зрозуміти, які дії користувачі можуть виконувати в системі, і як вони взаємодіють із її компонентами. На діаграмі варіантів використання зображено основні дії, які користувачі можуть виконувати, а також зв’язки між актором і варіантами використання.

У веб-месенджері виділено два типи акторів:

– Неавторизований користувач: Представляє користувачів, які ще не виконали авторизацію в системі. Цей актор має доступ лише до функції реєстрації за допомогою акаунта Google.

– Авторизований користувач: Представляє всіх користувачів системи, які виконали авторизацію через Google і мають доступ до базового функціоналу веб-месенджера. Залежно від ролі в групах, авторизований користувач може виконувати дії адміністратора або модератора.

На діаграмі варіантів використання (див. рис. 1.1) будуть зображені основні функції веб-месенджера, включаючи авторизацію через Google, управління списком друзів, обмін особистими повідомленнями, створення та редагування груп, управління каналами в межах групи. Ця діаграма допоможе краще зрозуміти, як система буде функціонувати в реальному середовищі та як користувачі будуть взаємодіяти з нею.

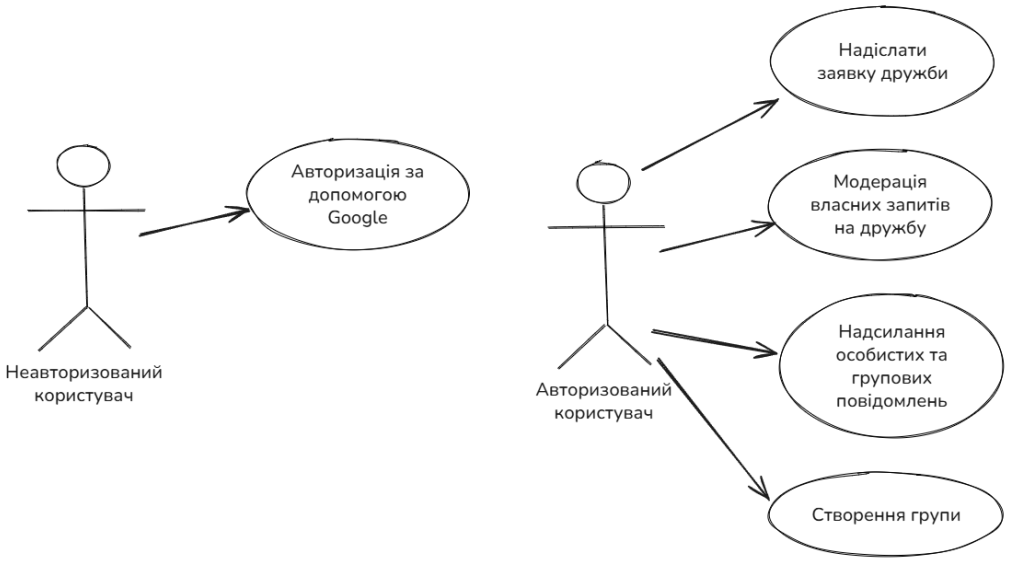

Рисунок 1.1 – Діаграма варіантів використання

Опис типових сценаріїв використання:

– Авторизація через Google: Користувач входить у систему, використовуючи свій Google-акаунт. Це дозволяє отримати доступ до

функціоналу веб-месенджера.

– Управління списком друзів: Авторизований користувач може здійснювати пошук друзів за електронною поштою, надсилати заявки в друзі, переглядати статус заявок (в очікуванні або відправлені), а також видаляти друзів зі списку. Ця функція сприяє організації персональних контактів і забезпечує зручність у спілкуванні.

– Особисті повідомлення: Авторизований користувач може надсилати та отримувати повідомлення від своїх друзів у форматі прямого спілкування. Ця функція дозволяє вести приватні розмови, забезпечуючи швидкий обмін інформацією.

– Створення та управління групами: Авторизований користувач може створювати групи, які дозволяють організовувати спілкування між учасниками. У межах групи користувач може виконувати дії адміністратора, такі як редагування групи, створення текстових і голосових каналів та управління учасниками.

1.4 Вибір технологій розробки системи

Вибір технологій розробки є важливим етапом у створенні веб-месенджера для комунікації в реальному часі. Це рішення впливає на продуктивність, масштабованість, безпеку та зручність підтримки проекту.

Для створення користувацького інтерфейсу обрано технологію TypeScript у поєднанні з React. TypeScript забезпечує статичну типізацію, що дозволяє зменшити кількість помилок у коді та полегшує підтримку проекту. React, завдяки своїй компонентній архітектурі, дозволяє створювати незалежні та багаторазові компоненти, що спрощує розробку, тестування та масштабування інтерфейсу [13]. Використання React забезпечує високу продуктивність завдяки віртуальному DOM, який оптимізує оновлення інтерфейсу користувача [4]. Крім того, велика спільнота розробників React надає доступ до численних бібліотек та інструментів,

що прискорює процес розробки.

Для реалізації серверної логіки було обрано стек технологій Node.js та TypeScript [3]. Перевагою такого підходу є уніфікація мови програмування (JavaScript) для фронтенду та бекенду, що робить процес розробки більш цілісним. Завдяки своїй неблокуючій, подієво-орієнтованій природі, Node.js чудово підходить для створення високопродуктивних та масштабованих систем. Додатковою перевагою є доступ до величезної кількості готових модулів через менеджер пакунків pnpm, що полегшує впровадження складного функціоналу, зокрема організацію двосторонньої комунікації між сервером і клієнтом за допомогою вебсокетів.

Для зберігання даних було обрано базу даних MongoDB, яка належить до типу документних баз. Вона зберігає інформацію у вигляді документів, схожих на формат JSON [14]. Такий підхід забезпечує високу гнучкість у зміні структури даних без необхідності складних міграцій, що робить її чудовим вибором для динамічних веб-додатків. Завдяки механізму індексування MongoDB дозволяє швидко отримувати доступ до даних і виконувати складні запити. Крім того, підтримка вкладених документів спрощує організацію даних для функцій веб-месенджера, таких як списки друзів, повідомлення чи групи.

2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБ-МЕСЕНДЖЕРА

2.1 Проектування бази даних

Проектування бази даних є фундаментальним етапом у життєвому циклі розробки програмного забезпечення, оскільки від структури даних безпосередньо залежить продуктивність, масштабованість та гнучкість усієї системи. Цей процес визначає, яка інформація буде зберігатися, як вона буде організована та які зв'язки існуватимуть між її елементами. Для даного проекту було обрано документо-орієнтовану NoSQL базу даних MongoDB, а для взаємодії з нею – Prisma ORM. Такий вибір зумовлений здатністю MongoDB ефективно працювати з ієрархічними та напівструктурованими даними, а також її горизонтальною масштабованістю, що є критично важливим для систем з високим навантаженням та великою кількістю одночасних користувачів [14].

З огляду на комплексність предметної області, для забезпечення чіткості та послідовності викладу, опис моделі даних організовано за функціональними групами. Кожна група об'єднує сутності, що відповідають за конкретний аспект функціоналу веб-месенджера.

Розгляд моделі даних доцільно розпочати з ключової функціональної групи, що відповідає за користувача та автентифікацію. Ці сутності є ядром системи, оскільки вони керують ідентифікацією, безпечними сесіями та зберіганням базової інформації про користувачів. Центральною сутністю цієї групи є модель User, яка агрегує всю інформацію, пов'язану з конкретним користувачем системи. Детальний опис її атрибутів наведено в таблиці 2.1.

Таблиця 2.1 – Атрибути сутності User

Назва атрибута	Тип даних	Призначення	Особливості

Продовження таблиці 2.1

id	ObjectId	Унікальний ідентифікатор користувача	PK
email	String	Електронна пошта користувача	Унікальне, індексоване поле
name	String	Ім'я користувача	Опційне поле
image	String	URL-адреса аватара	Опційне поле
lastSeen	DateTime	Час останньої активності	Опційне поле

Для інтеграції зі сторонніми сервісами автентифікації, такими як Google, використовується сутність Account. Вона виконує роль моста, пов'язуючи зовнішній профіль провайдера з внутрішнім записом User у системі та зберігаючи необхідні токени доступу. Структура даної сутності представлена в таблиці 2.2.

Таблиця 2.2 – Атрибути сутності Account

Назва атрибута	Тип даних	Призначення	Особливості
id	ObjectId	Унікальний ідентифікатор користувача	PK
providerId	String	Ідентифікатор провайдера	
userId	ObjectId	Ідентифікатор користувача в системі	FK до колекції User

Продовження таблиці 2.2

accessToken	String	Токен доступу від провайдера	Опційне поле

Після успішної автентифікації система створює безпечну сесію для користувача. За управління цими сесіями відповідає сутність Session. Кожен запис у цій колекції представляє активний вхід користувача в систему, містить унікальний токен сесії та метадані для підвищення безпеки. Детальний опис атрибутів наведено в таблиці 2.3.

Таблиця 2.3 – Атрибути сутності Session

Назва атрибута	Тип даних	Призначення	Особливості
id	ObjectId	Унікальний ідентифікатор користувача	PK
token	String	Унікальний токен сесії для автентифікації	Унікальне, індексоване поле
expiresAt	DateTime	Час, коли сесія стає недійсною	
userId	ObjectId	Ідентифікатор користувача	FK до колекції User

Для забезпечення додаткових рівнів безпеки та підтвердження дій, таких як верифікація електронної пошти, використовується сутність Verification. Вона зберігає тимчасові коди, які надсилаються користувачеві для підтвердження його

особи. Структура цієї моделі описана в таблиці 2.4.

Таблиця 2.4 – Атрибути сутності Verification

Назва атрибута	Тип даних	Призначення	Особливості
id	ObjectId	Унікальний ідентифікатор користувача	PK
identifier	String	Ідентифікатор, що верифікується	
value	String	Секретний код або токен для верифікації	
expiresAt	DateTime	Час, коли код стає недійсним	

Для наочної демонстрації взаємозв'язків між описаними сутностями було розроблено діаграму класів (див. рис. 2.1). На ній візуалізовано центральну роль моделі User та її відношення «один-до-багатьох» із сутностями Account та Session, що в сукупності формують цілісний механізм управління доступом та ідентифікацією користувачів.

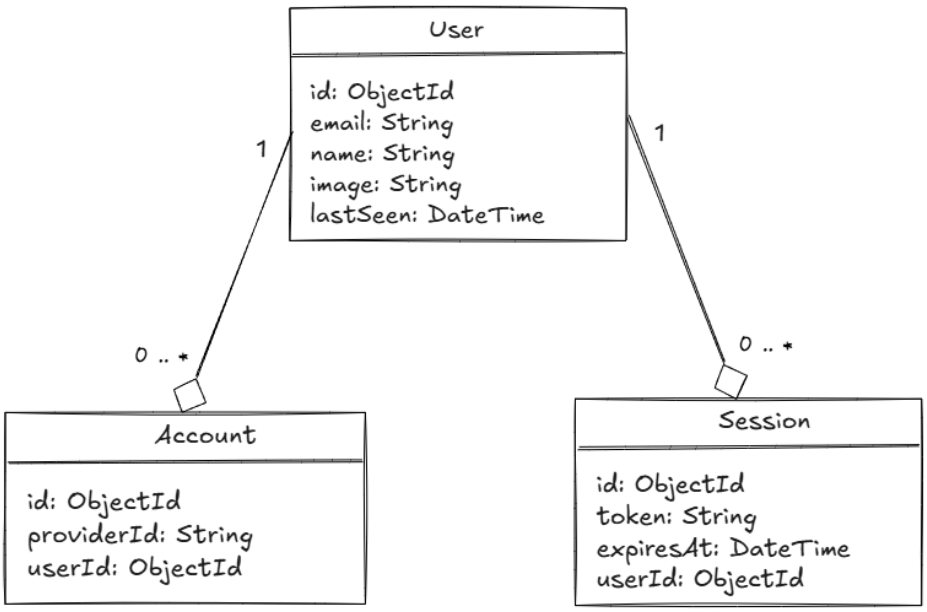

Рисунок 2.1 – Діаграма класів для сутностей користувача та автентифікації

Наступна логічна група сутностей реалізує функціонал соціальних зв'язків. Вона складається з моделей, що дозволяють користувачам надсилати запити на дружбу та формувати список друзів. Процес встановлення дружби починається із запиту, який моделюється сутністю FriendRequest. Вона відстежує життєвий цикл запиту від його створення до прийняття або відхилення. Детальна структура сутності наведена в таблиці 2.5.

Таблиця 2.5 – Атрибути сутності FriendRequest

Назва атрибута	Тип даних	Призначення	Особливості
id	ObjectId	Унікальний ідентифікатор запиту	PK
status	Enum	Статус запиту (PENDING, ACCEPTED, REJECTED)	Значення за замовчуванням PENDING

Продовження таблиці 2.5

senderId	ObjectId	Ідентифікатор відправника запиту	FK до колекції User
receiverId	ObjectId	Ідентифікатор отримувача запиту	FK до колекції User

Після того, як запит на дружбу отримує статус ACCEPTED, створюється запис у колекції Friend. Ця сутність представляє вже встановлений, двосторонній зв'язок дружби між двома користувачами, що дозволяє ефективно отримувати списки контактів. Її атрибути описані в таблиці 2.6.

Таблиця 2.6 – Атрибути сутності FriendRequest

Назва атрибута	Тип даних	Призначення	Особливості
id	ObjectId	Унікальний ідентифікатор запису дружби	PK
userId	ObjectId	Ідентифікатор першого користувача	FK до колекції User
friendId	ObjectId	Ідентифікатор другого користувача	FK до колекції User

Взаємодія цих двох сутностей візуалізована на діаграмі класів (див. рис. 2.2). Вона демонструє, як модель FriendRequest слугує тимчасовим об'єктом для ініціації зв'язку, тоді як модель Friend представляє постійний, встановлений зв'язок між двома екземплярами User.

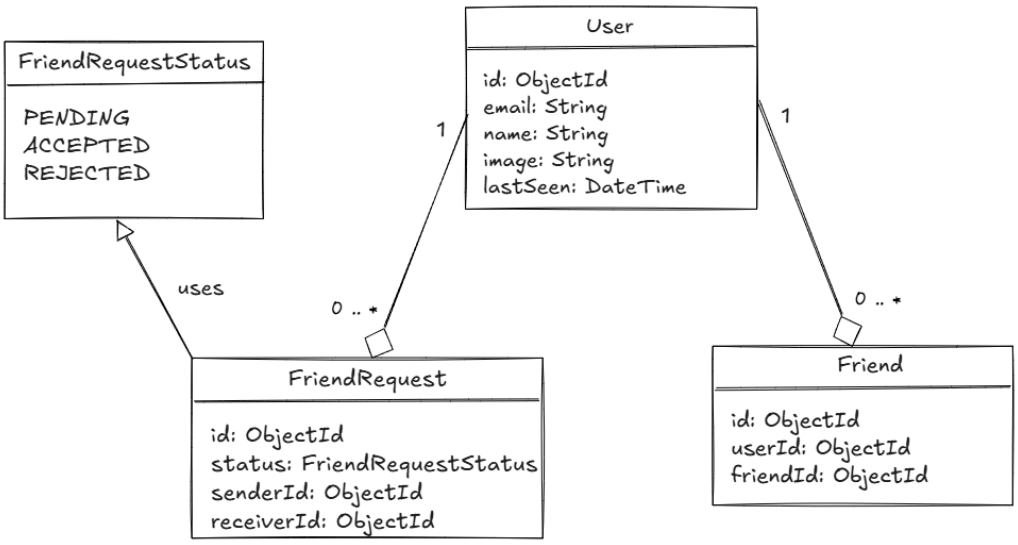

Рисунок 2.2 – Діаграма класів для сутностей соціальних зв'язків

Для організації спілкування у спільнотах спроектовано групу сутностей, що відповідає за групову комунікацію. Вона є аналогом функціоналу платформ Discord або Slack і включає моделі для створення груп, управління їх учасниками та організації тематичних каналів. Основою будь-якої спільноти є сама група, що представлена сутністю Group. Її атрибути описані в таблиці 2.7.

Таблиця 2.7 – Атрибути сутності Group

Назва атрибута	Тип даних	Призначення	Особливості
id	ObjectId	Унікальний ідентифікатор групи	PK
name	String	Назва групи	
description	String	Короткий опис групи	Опційне поле
image	String	URL-адреса іконки групи	Опційне поле

Продовження таблиці 2.7

inviteCode	String	Унікальний код для запрошення	Унікальне, індексоване поле

Для реалізації зв'язку «багато-до-багатьох» між користувачами та групами, а також для управління правами доступу, використовується проміжна сутність GroupMember. Вона не лише пов'язує користувача з групою, але й зберігає його роль у цій спільноті. Структура даної сутності представлена в таблиці 2.8.

Таблиця 2.8 – Атрибути сутності Group

Назва атрибута	Тип даних	Призначення	Особливості
id	ObjectId	Унікальний ідентифікатор членства	PK
role	Enum	Роль учасника (OWNER, ADMIN, MEMBER)	
userId	ObjectId	Ідентифікатор користувача	FK до колекції User
groupId	ObjectId	Ідентифікатор групи	FK до колекції Group

Спілкування всередині групи структурується за допомогою тематичних каналів, які моделюються сутністю GroupChannel. Кожен канал належить до певної групи і має визначений тип, що дозволяє розділяти текстові обговорення та голосові розмови. Атрибути цієї сутності наведені в таблиці 2.9.

Таблиця 2.9 – Атрибути сутності GroupChannel

Назва атрибута	Тип даних	Призначення	Особливості
id	ObjectId	Унікальний ідентифікатор каналу	PK
name	String	Ідентифікатор першого користувача	
type	Enum	Тип каналу (TEXT, VOICE)	
groupId	ObjectId	Ідентифікатор групи, до якої належить канал	FK до колекції Group
createdBy	ObjectId	Ідентифікатор користувача, що створив канал	FK до колекції User

Ієрархічна структура цього модуля візуалізована на діаграмі класів (див. рис. 2.3). На ній показано, як сутність Group виступає контейнером, що має відношення «один-до-багатьох» із сутностями GroupChannel та GroupMember. У свою чергу, GroupMember реалізує зв'язок між Group та User, формуючи цілісну модель для управління спільнотами.

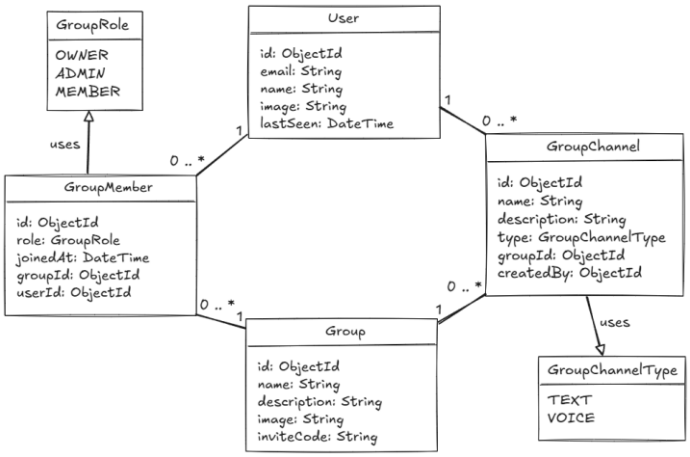

Рисунок 2.3 – Діаграма класів для сутностей групової комунікації

Основну функцію месенджера реалізує група сутностей, що відповідає за обмін повідомленнями та контентом. Вона включає моделі для особистих та групових повідомлень, а також сутності для розширення їх функціоналу. Спілкування в групових каналах складається з окремих повідомлень, що представлені сутністю ChannelMessage, структура якої описана в таблиці 2.10.

Таблиця 2.10 – Атрибути сутності ChannelMessage

Назва атрибута	Тип даних	Призначення	Особливості
id	ObjectId	Унікальний ідентифікатор повідомлення	PK
content	String	Текстовий вміст повідомлення	
senderId	ObjectId	Ідентифікатор автора повідомлення	FK до колекції User

Продовження таблиці 2.10

groupChannelId	ObjectId	Ідентифікатор каналу, де надіслано повідомлення	FK до колекції GroupChannel

Паралельно з груповою комунікацією система підтримує приватні розмови між двома користувачами, які моделюються сутністю DirectMessage. Її структура подібна до ChannelMessage, але замість посилання на канал вона містить посилання на отримувача.

Таблиця 2.11 – Атрибути сутності DirectMessage

Назва атрибута	Тип даних	Призначення	Особливості
id	ObjectId	Унікальний ідентифікатор повідомлення	PK
content	String	Текстовий вміст повідомлення	
senderId	ObjectId	Ідентифікатор відправника	FK до колекції User
receiverId	ObjectId	Ідентифікатор отримувача	FK до колекції User

Для збагачення комунікації користувачі можуть додавати емодзі-реакції до повідомлень. Цей функціонал реалізовано за допомогою поліморфної сутності Reaction, яка може бути пов'язана як з груповим, так і з особистим повідомленням.

Таблиця 2.12 – Атрибути сутності Reaction

Назва атрибута	Тип даних	Призначення	Особливості
id	ObjectId	Унікальний ідентифікатор реакції	PK
emoji	String	Символ емодзі	
userId	ObjectId	Ідентифікатор користувача, що поставив реакцію	FK до колекції User
channelMessageId	ObjectId	Ідентифікатор повідомлення в каналі	FK до колекції ChannelMessage, може бути опційним
directMessageId	ObjectId	Ідентифікатор особистого повідомлення	FK до колекції DirectMessage, може бути опційним

Окрім текстової комунікації та реакцій, важливою частиною функціоналу є можливість обміну файлами. Для реалізації цього завдання було спроектовано архітектуру, що розділяє метадані самого файлу та інформацію про його використання. Перша сутність, File, відповідає за зберігання інформації про завантажений цифровий актив як такий, незалежно від того, де і скільки разів він буде використаний у системі. Такий підхід дозволяє уникнути дублювання даних та оптимізувати їх зберігання. Структура цієї сутності наведена в таблиці 2.13.

Таблиця 2.13 – Атрибути сутності File

Назва атрибута	Тип даних	Призначення	Особливості
id	ObjectId	Унікальний ідентифікатор файлу	PK
name	String	Назва файлу	
path	String	Шлях до файлу у сховищі	
type	String	MIME-тип файлу	
userId	ObjectId	Ідентифікатор користувача, що завантажив файл	FK до колекції User

Однак, простого зберігання файлу недостатньо; необхідно також відстежувати його використання в контексті конкретних розмов. Для вирішення цього завдання введено другу, проміжну сутність – FileUpload. Вона виконує роль зв'язуючої ланки, яка асоціює конкретний екземпляр файлу (з колекції File) з повідомленням, до якого він був прикріплений. Такий підхід забезпечує гнучкість, дозволяючи один і той самий завантажений файл використовувати у різних групових або особистих чатах без необхідності його повторного завантаження. Детальний опис атрибутів цієї поліморфної сутності наведено в таблиці 2.14.

Таблиця 2.14 – Атрибути сутності FileUpload

Назва атрибута	Тип даних	Призначення	Особливості

Продовження таблиці 2.14

id	ObjectId	Унікальний ідентифікатор файлу	PK
fileId	ObjectId	Ідентифікатор файлу	FK до колекції File
channelMessageId	ObjectId	Ідентифікатор повідомлення в каналі	FK до колекції ChannelMessage, може бути опційним
directMessageId	ObjectId	Ідентифікатор особистого повідомлення	FK до колекції DirectMessage, може бути опційним

Діаграма класів для цього модуля (див. рис. 2.4) ілюструє, як сутності ChannelMessage та DirectMessage виступають центральними елементами, до яких через відношення «один-до-багатьох» приєднуються сутності Reaction та FileUpload, розширюючи їхній базовий функціонал.

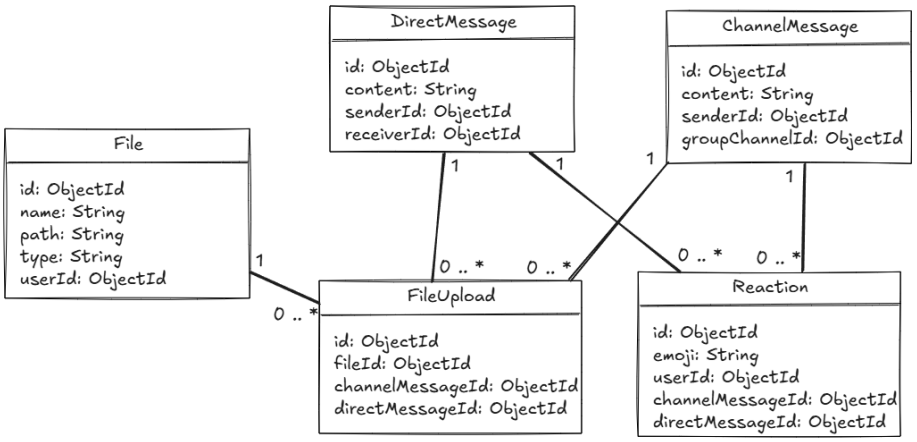

Рисунок 2.4 – Діаграма класів для сутностей обміну повідомленнями

На завершення, для надання цілісного уявлення про модель даних системи, було розроблено загальну діаграму класів (див. рис. 2.5). Вона інтегрує всі описані вище функціональні групи, візуалізуючи складні взаємозв'язки між ними та демонструючи центральну роль сутності User як сполучної ланки для всіх аспектів функціоналу веб-месенджера.

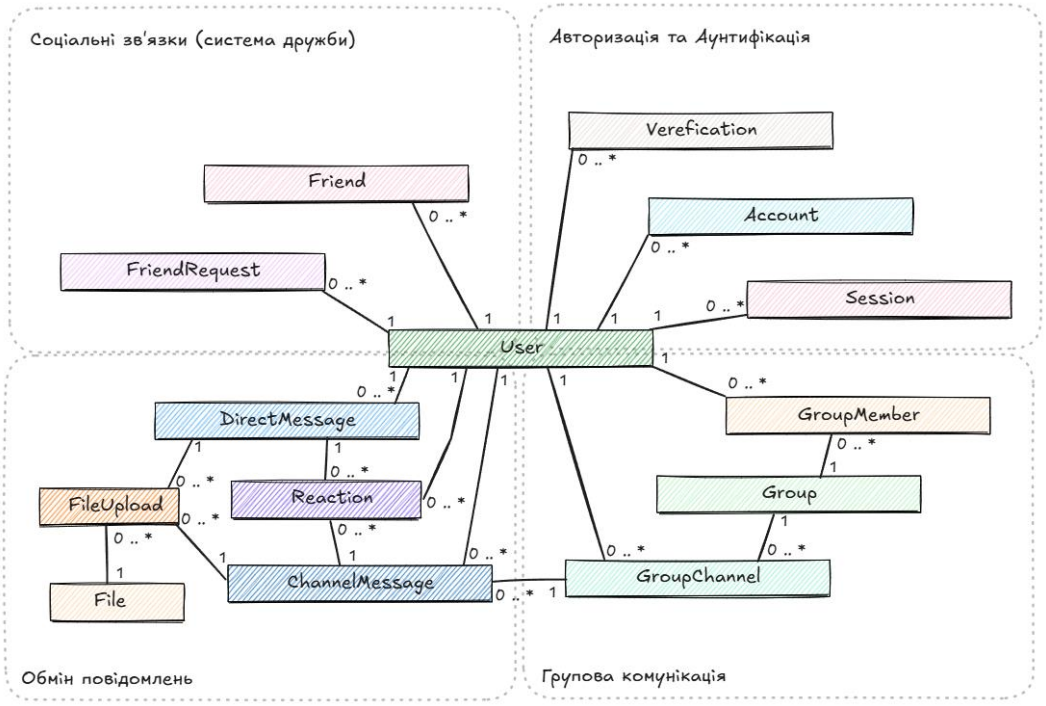

Рисунок 2.5 – Загальна діаграма класів системи веб-месенджера

2.2 Моделювання архітектури системи

Моделювання архітектури є критично важливим етапом проектування, що визначає високорівневу структуру програмної системи, її основні компоненти та принципи їхньої взаємодії. Правильно спроектована архітектура закладає фундамент для створення надійного, масштабованого та легкого в підтримці продукту. Для розробки веб-месенджера було обрано класичну клієнт-серверну архітектуру, оскільки вона забезпечує чітке розділення відповідальності: клієнтська частина відповідає за представлення та взаємодію з користувачем, а серверна – за бізнес-логіку, обробку та зберігання даних. Такий підхід дозволяє розробляти, тестувати та масштабувати обидві частини системи незалежно одна від одної [11].

Ключовим архітектурним рішенням, що визначає організацію кодової бази, стало використання монорепозиторію [2]. Цей вибір був стратегічним, спрямованим на вирішення поширених проблем у розробці full-stack додатків. На відміну від традиційної моделі з кількома окремими репозиторіями, монорепозиторій, керований за допомогою менеджера пакетів pnpm та його функціоналу Workspaces, надає низку суттєвих переваг. По-перше, він забезпечує уніфіковану кодову базу та значно спрощує повторне використання коду через спільні пакети, що є критично важливим для забезпечення консистентності типів між клієнтом та сервером. По-друге, він дозволяє вносити атомарні зміни в одному коміті, що є незамінним при рефакторингу. Нарешті, такий підхід спрощує управління залежностями. Для оптимізації процесів розробки було інтегровано інструмент Turborepo, основна перевага якого полягає в інтелектуальному кешуванні результатів виконання задач, що радикально скорочує час розробки та збірки.

Весь життєвий цикл розробки проекту керувався за допомогою системи контролю версій Git. Для централізованого зберігання кодової бази та управління версіями було використано платформу GitHub [21]. На рисунку 2.6 показано

головну сторінку репозиторію проекту, що демонструє його активність та структуру.

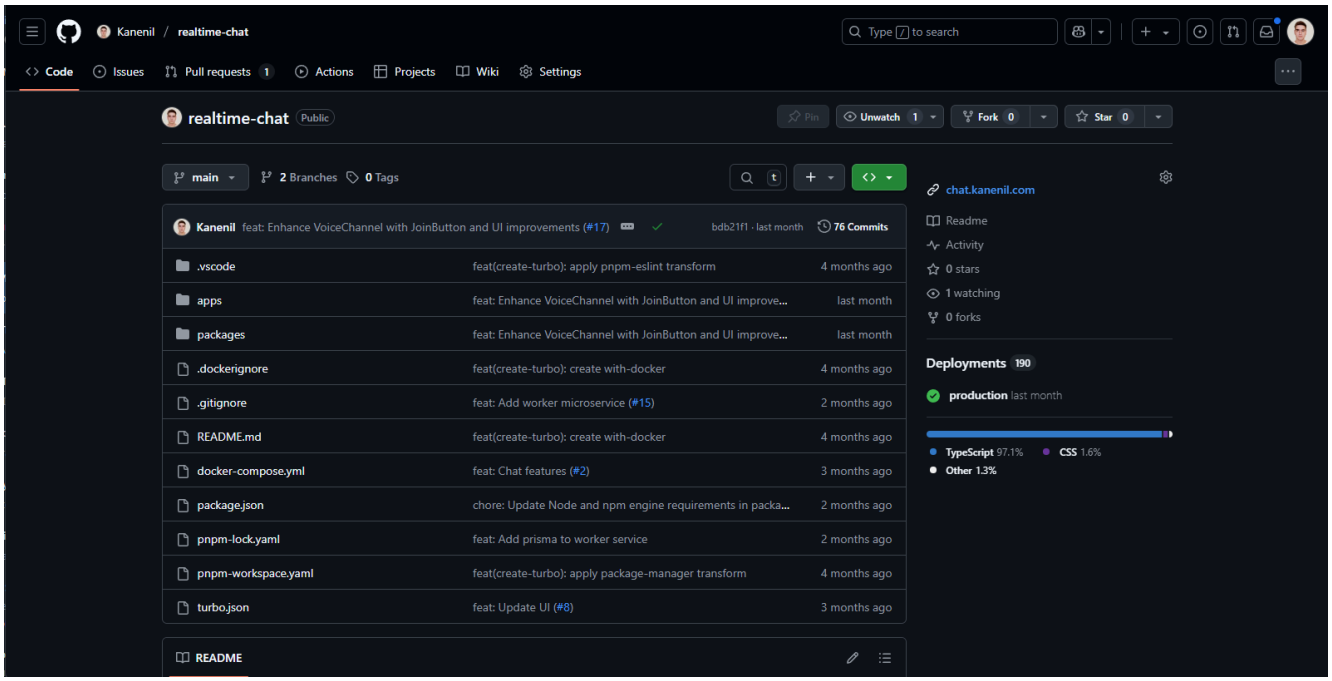

Рисунок 2.6 – Головна сторінка репозиторію проекту

Використання Git дозволило ефективно відстежувати всі зміни в кодовій базі, безпечно експериментувати з новим функціоналом в ізольованих гілках та забезпечувати цілісність проекту. Було застосовано модель розгалуження Feature Branch Workflow, де кожна нова функція або виправлення розроблялися в окремій гілці. Практичне застосування цієї моделі можна побачити на візуалізації історії комітів (див. рис. 2.7). Граф чітко ілюструє, як від основної гілки розробки відгалужуються окремі гілки для реалізації конкретних завдань (наприклад, feat/setting-page), а після їх завершення вони зливаються назад, зберігаючи історію розробки чистою та зрозумілою.

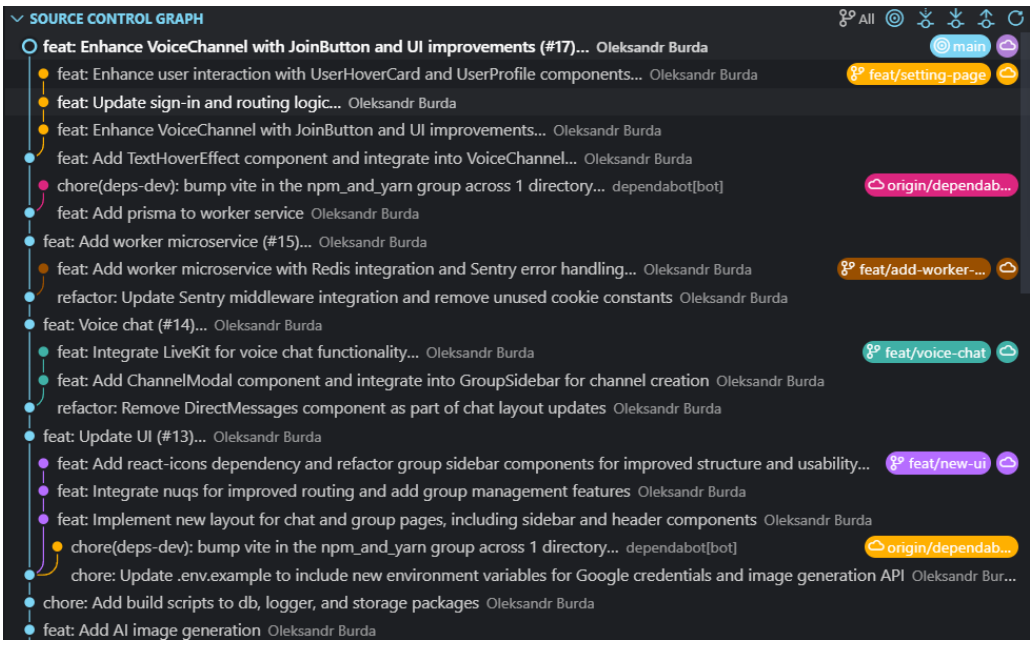

Рисунок 2.7 – Граф історії комітів, що демонструє модель розгалуження

Для наочної демонстрації фізичної організації кодової бази в середовищі розробки Cursor було зроблено знімок екрана (див. рис. 2.8). Він візуалізує кореневу структуру проекту, включаючи директорії для розгортаємих додатків (apps), спільних пакетів (packages) та глобальних конфігураційних файлів.

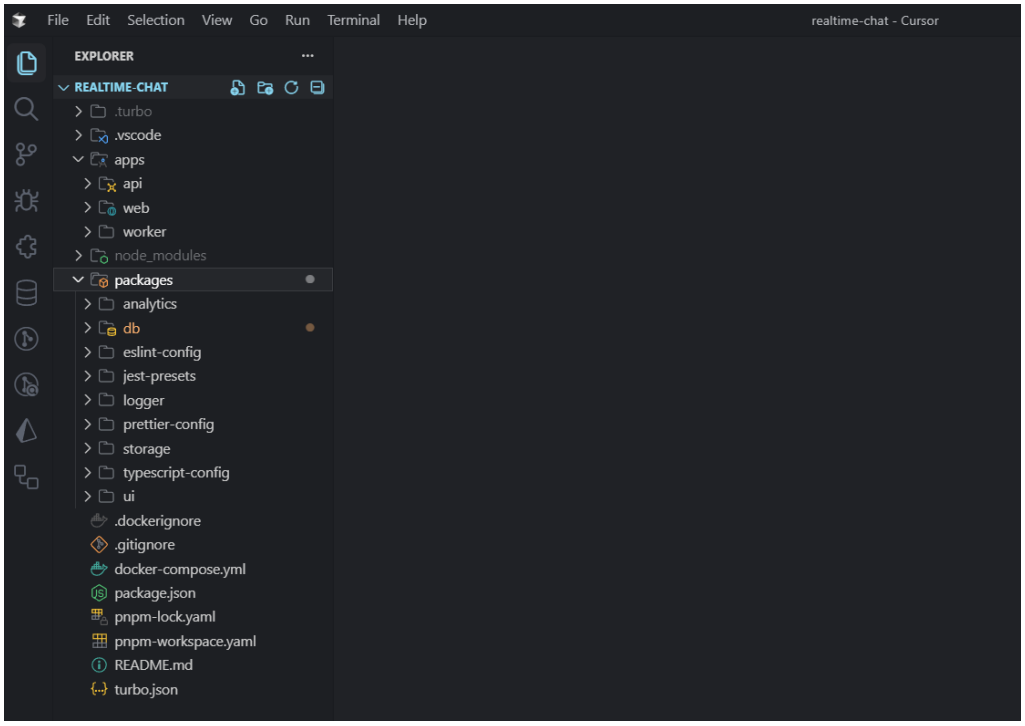

Рисунок 2.8 – Структура проекту в середовищі розробки

Як видно на рисунку 2.8, система має чітку компонентну структуру. В директорії apps знаходяться два основні додатки. Клієнтська частина apps/web є односторінковим додатком (SPA), розробленим з використанням бібліотеки React та сучасного інструментарію Vite. Вибір React зумовлений його компонентною архітектурою, що дозволяє створювати складні інтерфейси з незалежних, перевикористовуваних блоків та ефективно керувати їхнім станом [4]. Серверна частина apps/api розроблена на платформі Node.js [3]. Її неблокуюча, подієво-орієнтована природа є критично важливою для обробки великої кількості одночасних вебсокет-з'єднань з мінімальними затримками. Використання TypeScript в обох частинах проекту забезпечило статичну типізацію, що є фундаментальним для створення великих, надійних систем, оскільки дозволяє виявляти помилки на етапі компіляції, а не під час виконання.

Особливу роль в архітектурі відіграє директорія packages, яка є серцем механізму повторного використання коду. Пакет packages/db повністю інкапсулює логіку взаємодії з базою даних, надаючи типізований клієнт Prisma для серверної частини. Пакет packages/ui містить спільну бібліотеку UI-компонентів на основі shadcn/ui, що гарантує візуальну консистентність усього додатку. Інші пакети, такі як packages/logger та packages/storage, вирішують наскрізні завдання, а конфігураційні пакети (eslint-config, typescript-config) забезпечують єдині стандарти якості коду для всього проекту.

Взаємодія між клієнтом та сервером реалізована за допомогою двох спеціалізованих протоколів. Для всіх стандартних запитів та мутацій даних використовується технологія tRPC [5]. Її було обрано через унікальну здатність забезпечувати наскрізну типізацію без етапу генерації коду, що є ключовою перевагою порівняно з REST або GraphQL в умовах повністю типізованого монорепозиторію.

Для комунікації в реальному часі застосовано протокол Вебсокетів (WebSockets). На відміну від HTTP-пулінгу, який створює значне надлишкове навантаження, вебсокети забезпечують ефективний та ресурсоощадний постійний канал зв'язку для миттєвих сповіщень, таких як доставка нових повідомлень чи

оновлення статусів користувачів.

Для візуального представлення логічної архітектури та потоків даних було розроблено високорівневу діаграму (див. рис. 2.9). На ній зображено, як клієнтський додаток web взаємодіє з серверним api через два незалежні канали: tRPC для запитів та вебсокети для подій реального часу [5]. У свою чергу, серверна частина використовує спільний пакет db для доступу до бази даних MongoDB, демонструючи цілісність та взаємозв'язок усіх компонентів системи.

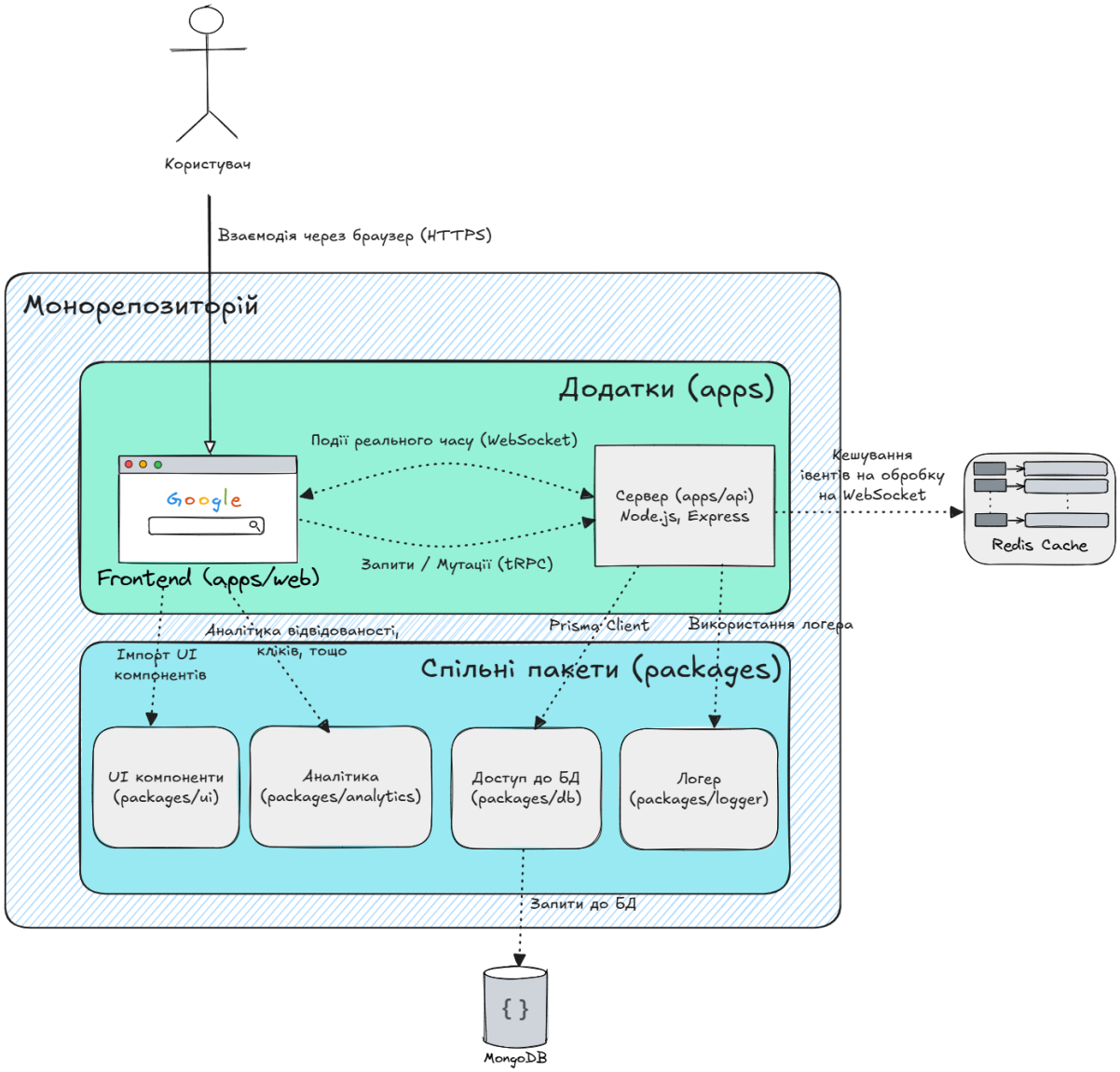

Рисунок 2.9 – Високорівнева архітектурна діаграма системи

2.3 Розробка серверної частини

Розробка серверної частини велася за послідовним, ітеративним підходом, що дозволило систематично нарощувати функціональність системи, забезпечуючи стабільність та архітектурну цілісність на кожному етапі. Процес можна умовно розділити на кілька ключових фаз: створення інфраструктурного фундаменту, реалізація базової функціональності користувача, розробка механізмів соціальної взаємодії та, нарешті, впровадження складних систем комунікації в реальному часі.

Розробка розпочалася зі створення архітектурного скелету додатку. Важливим кроком стало впровадження системи конфігурації через змінні середовища з валідацією за допомогою бібліотеки Zod, що гарантує надійність та безпеку запуску додатку.

Після налаштування архітектурного скелету, першочерговим завданням стала реалізація ключової сутності системи – користувача, та механізмів його автентифікації. Було створено базову модель User у схемі Prisma та інтегровано бібліотеку Better Auth для обробки автентифікації через соціальні провайдери, зокрема Google. Для повноцінної роботи цього модуля було додано підтримуючі моделі Session, Account та Verification.

На цьому ж етапі було закладено основу для API, ініціалізувавши tRPC. Було створено контексту, що передає інформацію про сесію в кожну процедуру, та створено функцію isAuthed для захисту приватних ендпоінтів. Це дозволило сформувати патерн «захищеної процедури» (protectedProcedure) [6], який став стандартом для всіх подальших розробок, що вимагають авторизації. На рисунку 2.10 показано реалізацію цього механізму, що є наріжним каменем безпеки системи.

```
export const isAuthed: MiddlewareFunction = async ({ ctx, next }) => {
  if (!ctx.session || !ctx.session.user) {
    throw new TRPCError({ code: 'UNAUTHORIZED' });
  }

  return next({ ctx });
};

```

Рисунок 2.10 – Створення захищеної процедури для забезпечення авторизації

Маючи в системі поняття автентифікованого користувача, розробка перейшла до створення механізмів соціальної взаємодії. Спочатку було реалізовано базові операції з користувачами, такі як пошук та отримання інформації про профіль. Потім було розроблено систему дружби, що включає моделі FriendRequest та Friend. Ця логіка вимагала реалізації більш складної бізнес-логіки, а саме використання транзакцій бази даних при прийнятті запиту на дружбу для забезпечення атомарності операції.

Паралельно було налаштовано HTTP-сервер на базі Express та інтегровано з ним tRPC та WebSocket сервери, що підготувало платформу для наступного, найважливішого етапу [5].

Впровадження системи комунікації в реальному часі був центральний етап розробки, що вимагав реалізації складної архітектури. Спочатку було розроблено функціонал прямих повідомлень, за яку відповідає модель DirectMessage. На її базі було побудовано CRUD-операції.

Ключовим архітектурним рішенням в інтеграції вебсокет-з’єднання стало використання Redis, як брокера повідомлень. Коли мутація sendDirectMessage успішно зберігає повідомлення в базу даних, вона не намагається напряму сповістити клієнта, замість цього, вона публікує подію у відповідний канал Redis [8]. Паралельно, tRPC-підписка onDirectMessageChanged підписується на цей канал, яка отримавши подію з Redis, надсилає дані конкретному клієнту через двохстороннє вебсокет-з'єднання. Такий підхід забезпечує високу

масштабованість та відмовостійкість. На рисунку 2.11 показано ключовий фрагмент коду, що демонструє цей механізм.

```
128     await redisPubSub.publish<DirectMessageChange>(
129       EVENTS.DIRECT_MESSAGE_UPDATED,
130       {
131         directMessage: {
132           ...message,
133           fileUploads
134         },
135         status: 'ADDED'
136       }
137     );
138
139     return message;
140   });
```

Рисунок 2.11 – Публікація події в Redis після успішного виконання мутації

На базі цієї системи було реалізовано додатковий функціонал: реакції (з універсальною моделлю Reaction) та завантаження файлів (з моделями File та FileUpload).

На основі стабільного функціоналу особистих повідомлень, було впроваджено більш складну групову комунікацію. Це включало розробку моделей Group, GroupChannel та ChannelMessage, а також реалізацію системи ролей (GroupMember) для управління правами доступу.

На завершальних етапах розробки фокус змістився з простої реалізації функціоналу на забезпечення надійності, безпеки та стабільності системи в робочому середовищі. Одним з ключових механізмів, впроваджених на цьому етапі, стала система обмеження частоти запитів (Rate Limiting). Її основна мета – захист сервера від зловживань, таких як спам-атаки або спроби перевантаження системи (DoS-атаки), а також забезпечення справедливого розподілу ресурсів між усіма користувачами.

Для реалізації цього механізму було створено універсальну та гнучку функцію withRateLimit. Архітектурним рішенням стало використання Redis, як сховища для лічильників запитів. Вибір Redis зумовлений його високою

продуктивністю, оскільки він є in-memory сховищем, та наявністю атомарних операцій, таких як INCR, що є критично важливим для коректної роботи лічильників в умовах одночасних запитів [8].

В основі механізму лежить алгоритм «Фіксованого вікна з лічильником», де для кожного користувача та для кожної конкретної дії створюється унікальний ключ у Redis (наприклад, ratelimit:message:send:USER_ID) [9]. При кожному запиті лічильник для цього ключа збільшується. Якщо це перший запит у «вікні», для ключа встановлюється час життя (expire), що дорівнює тривалості вікна. Якщо кількість запитів перевищує встановлений ліміт, функція генерує помилку TOO_MANY_REQUESTS і перериває виконання процедури.

На рисунку 2.12 наведено код фабричної функції withRateLimit, яка приймає параметри limit (кількість запитів) та windowInSeconds (тривалість вікна).

```
34  export const withRateLimit = (options: RateLimitOptions) => {
35    const { limit, windowInSeconds, identifier } = options;
36
37    return t.middleware(async ({ ctx, next, path }) => {
38  >   if (!ctx.session?.user) {…
43      }
44
45      const rateLimitKey = `ratelimit:${identifier || path}:${ctx.session.user.id}`;
46
47      const currentCount = await redis.incr(rateLimitKey);
48
49      if (currentCount === 1) {
50        await redis.expire(rateLimitKey, windowInSeconds);
51      }
52
53      if (currentCount > limit) {      Oleksandr Burda, 3 months ago via  PR #6 • fea
54        await redis.decr(rateLimitKey);
55
56        const ttl = await redis.ttl(rateLimitKey);
57        const { minutes, seconds, resetTimestamp } = formatTimeRemaining(ttl);
58
59        let errorMessage: string;
60  >     if (minutes > 0) {…
64        } else {
65          errorMessage = i18n.t('rateLimit.errors.tooManyRequests_seconds', {
66            seconds
67          });
68        }
```

Рисунок 2.12 – Реалізація функції для обмеження частоти запитів

Важливою особливістю реалізації є надання користувачеві чіткого зворотного зв'язку. У випадку перевищення ліміту, система не просто блокує запит, а й повертає інформативне повідомлення про те, через скільки хвилин або секунд можна буде повторити спробу, що значно покращує користувацький досвід.

Для зручності та чистоти коду, на основі цієї фабричної функції було створено кілька специфічних екземплярів під різні дії, таких як withMessageRateLimit або withFileUploadRateLimit. Це дозволяє декларативно застосовувати різні ліміти до різних процедур, просто додаючи відповідну функцію до ланцюжка їх виконання.

Для розуміння та відслідковування помилок, які виникають під час роботи програми в контейнері, було інтегровано систему моніторингу помилок Sentry та налаштовано структуроване логування.

2.4 Розробка клієнтської частини

Клієнтська частина є точкою взаємодії користувача з системою, тому її розробка була зосереджена на створенні швидкого та інтуїтивно зрозумілого інтерфейсу. Вона розроблена як односторінковий додаток (Single Page Application, SPA) з використанням бібліотеки React, яка дозволила застосувати декларативний підхід до побудови інтерфейсу, де UI є функцією від стану додатку.

Розробка розпочалася зі створення міцного архітектурного фундаменту, було реалізовано систему з чітким розділенням відповідальності за допомогою механізму контекстів (React Context API), створивши набір глобальних «провайдерів» [4]. Такий підхід є реалізацією патерну «Впровадження залежностей» (Dependency Injection) [6], який дозволяє уникнути глобальних змінних та забезпечує чистий, структурований доступ до сервісів з будь-якого компонента. На рисунку 2.13 показано, як головний компонент додатку

обгортається у провайдери, що надають доступ до tRPC, системи кешування та інших сервісів.

```tsx
const App = () => {
  const [queryClient] = useState(() => new QueryClient());
  const [trpcClient] = useState(createTRPCClient);

  return (
    <trpc.Provider client={trpcClient} queryClient={queryClient}>
      <QueryClientProvider client={queryClient}>
        <NuqsAdapter>
          <UserProvider>
            <NewGroupProvider>
              <OnlineProvider>      Oleksandr Burda, 2 months ago
                <FriendProvider>
                  <EmojiPickerProvider>
                    <VoiceChatProvider>
                      <AppRouter />
                    </VoiceChatProvider>
                  </EmojiPickerProvider>
                </FriendProvider>
              </OnlineProvider>
            </NewGroupProvider>
          </UserProvider>
        </NuqsAdapter>
      </QueryClientProvider>
    </trpc.Provider>
  );
};
```

Рисунок 2.13 – Ініціалізація клієнтського додатку з використанням провайдерів

Для керування глобальним станом додатку, таким як інформація про поточного користувача, було обрано бібліотеку Zustand, яка дозволяє створити легкі та високопродуктивні сховища даних.

Першим функціональним модулем, з яким стикається користувач, є система автентифікації. Для захисту приватних розділів додатку було реалізовано механізм захищених маршрутів. Це компонент вищого порядку, який перевіряє наявність активної сесії користувача, якщо сесія відсутня, він автоматично

перенаправляє користувача на сторінку входу, забезпечуючи безпеку даних.

Для забезпечення консистентності даних та синхронізації стану між усіма клієнтами було реалізовано подієво-керовану модель оновлення інтерфейсу. На відміну від патерну «оптимістичного оновлення», де інтерфейс оновлюється негайно, в даній архітектурі клієнт виступає в ролі пасивного слухача, а сервер є єдиним джерелом правди.

Коли користувач виконує дію, наприклад, відправляє запит на дружбу, клієнт викликає відповідну tRPC-мутацію. Однак, він не змінює локальний стан одразу. Замість цього, він очікує на подію від сервера. Сервер, обробивши мутацію, публікує подію через вебсокет, а в свою чергу, клієнт, який підписаний на цю подію, отримує нові дані (наприклад, об'єкт нового запиту на дружбу) і лише тоді оновлює свій локальний кеш даних. Це гарантує, що інтерфейс завжди відображає актуальний, підтверджений сервером стан. На рисунку 2.14 показано приклад tRPC-підписки, яка слухає події та оновлює кеш.

```
50  trpc.directMessage.onDirectMessageChanged.useSubscription(
51    { userId: userId || '' },
52    {
53      enabled: !!userId,
54      onData: async ({ directMessage, status }) => {
55        utils.directMessage.findDirectMessages.setInfiniteData(
56 >        {…
59          },
60          (data) => {
61 >          if (!data) {…
66            }
67
68            return {      Oleksandr Burda, 3 months ago via  PR
69              ...data,
70              pages: data.pages.map((page, index) =>
71                updatePageDirectMessages(
72                  page,
73                  {
74                    ...directMessage,
75                    status
76                  },
77                  index
78                )
79              )
80            };
81          }
82        );
```

Рисунок 2.14 – Оновлення інтерфейсу на основі даних, отриманих через вебсокет-підписку

Для забезпечення плавного перегляду довгої історії повідомлень було реалізовано механізм нескінченної прокрутки, система завантажує повідомлення невеликими порціями за допомогою хука useInfiniteQuery від tRPC. Коли користувач прокручує чат догори, автоматично ініціюється запит на завантаження наступної, старішої порції даних.

Отримання нових повідомлень в реальному часі також слідує подієво-керованій моделі. Хук useEffect використовується для підписки на відповідну вебсокет-подію при монтуванні компонента чату. Коли надходить нова подія, вона оновлює кеш даних, що завдяки декларативній природі React, автоматично призводить до оновлення інтерфейсу.

2.5 Тестування системи

Тестування є невід'ємним етапом життєвого циклу розробки програмного забезпечення, що має на меті перевірку коректності роботи системи, виявлення дефектів та забезпечення відповідності функціональним та нефункціональним вимогам. Для даного проекту було застосовано багаторівневу стратегію тестування, що дозволило всебічно перевірити розроблену систему як на рівні окремих компонентів, так і в контексті цілісних користувацьких сценаріїв.

Для забезпечення надійності серверної логіки на компонентному рівні було розроблено комплексний набір модульних тестів (unit tests). Метою цього підходу є ізольована перевірка окремих частин коду – в даному випадку, tRPC-процедур – для підтвердження їхньої коректної роботи. Для тестування було використано фреймворк Jest [10], а для ізоляції від зовнішніх залежностей, таких як база даних чи Redis сховище, застосовано техніку мокування залежностей. Виконання всього набору тестів відбувається в автоматичному режимі в консольному середовищі, що дозволяє швидко перевіряти стабільність кодової бази після внесення змін.

Результати виконання, показані на рисунку 2.15, демонструють, що всі тестові сценарії для перевірених модулів успішно пройдені, що свідчить про високу якість та надійність кодової бази серверної частини.

```
api:test: PASS src/procedures/groupChannel/__tests__/subscriptions.test.ts
api:test: PASS src/procedures/reaction/__tests__/subscriptions.test.ts
api:test: PASS src/procedures/user/__tests__/mutations.test.ts
api:test: PASS src/procedures/friend/__tests__/subscriptions.test.ts
api:test: PASS src/procedures/file/__tests__/queries.test.ts
api:test: PASS src/procedures/reaction/__tests__/mutations.test.ts
api:test: PASS src/procedures/directMessage/__tests__/subscriptions.test.ts
api:test: PASS src/procedures/file/__tests__/mutations.test.ts
api:test: PASS src/procedures/ai/__tests__/mutations.test.ts
api:test: PASS src/procedures/group/__tests__/subscriptions.test.ts
api:test: PASS src/procedures/directMessage/__tests__/mutations.test.ts
api:test: PASS src/procedures/user/__tests__/subscriptions.test.ts
api:test:
api:test: Test Suites: 19 passed, 19 total
api:test: Tests:       219 passed, 219 total
api:test: Snapshots:   0 total
api:test: Time:        3.297 s, estimated 4 s
api:test: Ran all test suites.

 Tasks:    1 successful, 1 total
Cached:    0 cached, 1 total
  Time:    5.831s
```

Рисунок 2.15 – Результат виконання модульних тестів у консолі

Хоча модульні тести підтверджують коректність окремих компонентів, вони не гарантують правильної роботи системи в цілому з точки зору кінцевого користувача. Тому другим рівнем стратегії стало тестування ключових користувацьких сценаріїв. Цей підхід фокусується на перевірці наскрізної функціональності, імітуючи реальні дії користувача в системі. Для формалізації та візуалізації цих сценаріїв було використано діаграми діяльності.

Першим і найбільш фундаментальним сценарієм, що підлягав перевірці, був процес авторизації користувача. Взаємодія між клієнтом, сервером та стороннім провайдером автентифікації для цього сценарію детально показана на діаграмі послідовності (див. рис. 2.16).

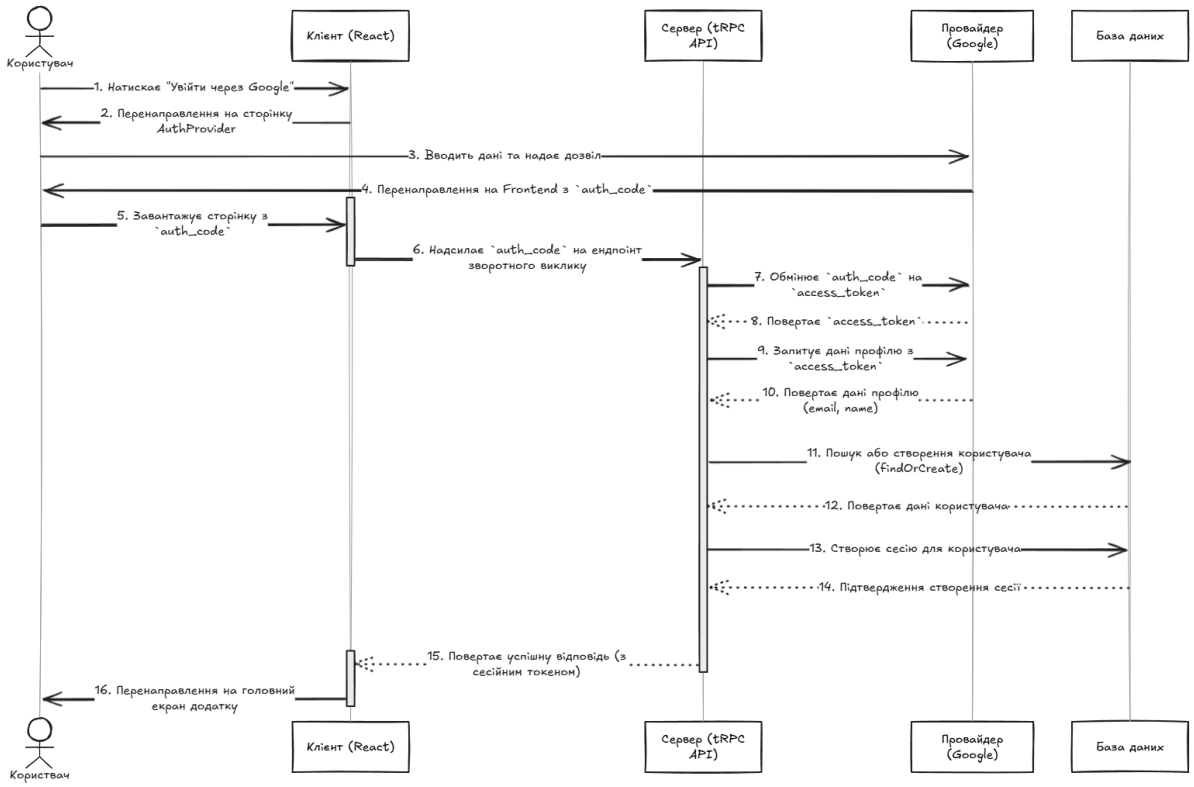

Рисунок 2.16 – Діаграма послідовності для сценарію «Авторизація користувача»

Тестування починається на головній сторінці додатку, яка для неавторизованого користувача є екраном входу, як показано на рисунку 2.17.

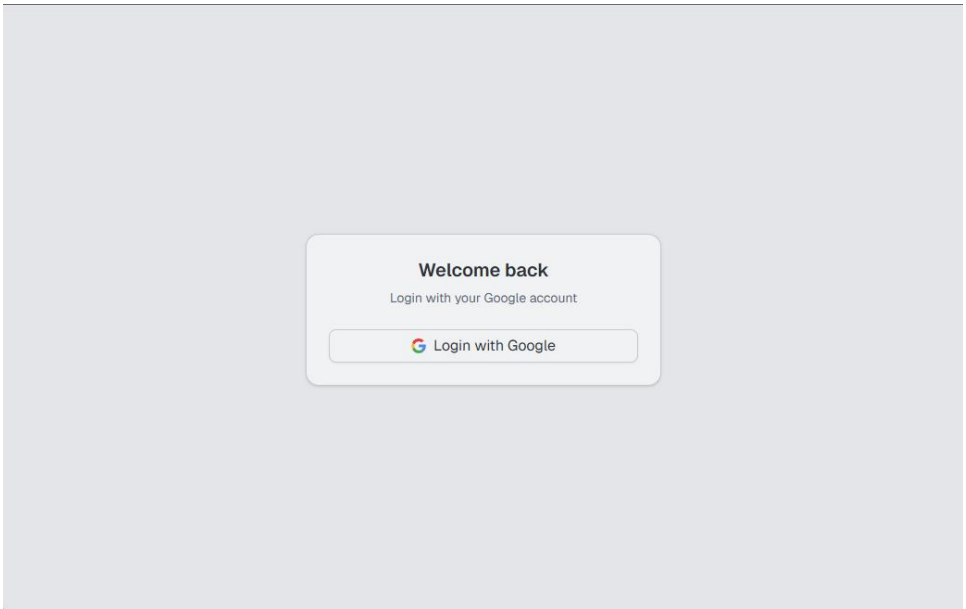

Рисунок 2.17 – Початковий екран для неавторизованого користувача

Після натискання кнопки «Login with Google» та успішного підтвердження особи, користувач перенаправляється назад у додаток і бачить головний інтерфейс системи. Цей результат, зображений на рисунку 2.18, що підтверджує успішне завершення сценарію авторизації.

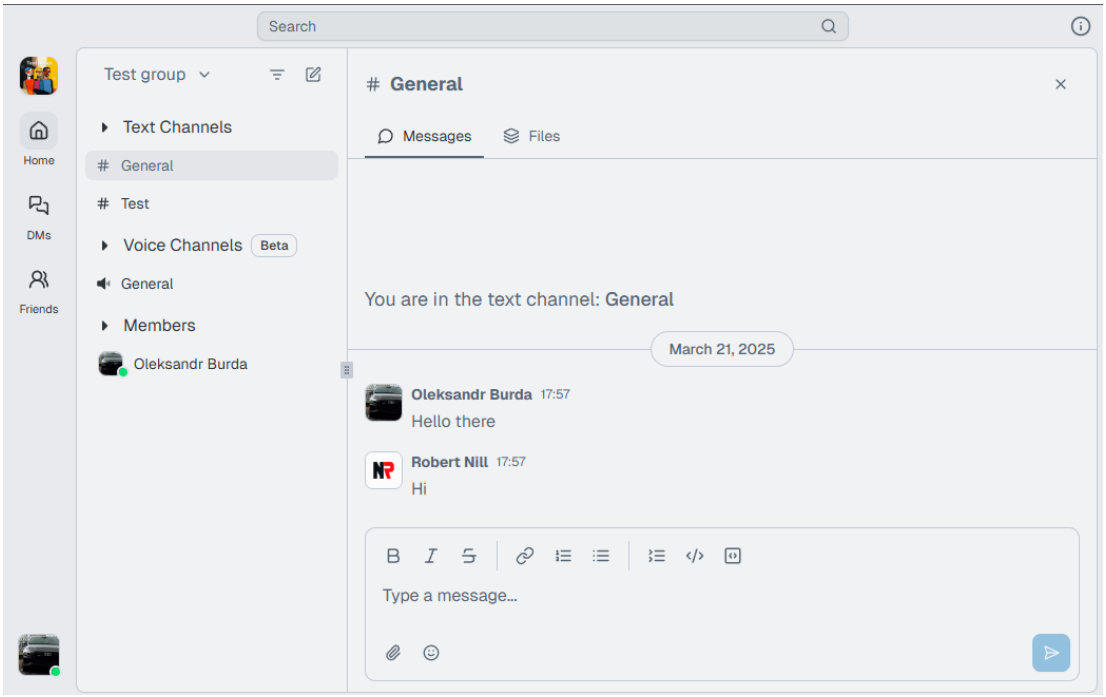

Рисунок 2.18 – Головний екран після успішної авторизації

Другим ключовим сценарієм було тестування основного функціоналу додатку – відправки повідомлення в груповому каналі. На діаграмі послідовності (див. рис. 2.19) показано повний ланцюжок взаємодій: від дії користувача на клієнті, через обробку запиту на сервері, до отримання події в реальному часі через вебсокет.

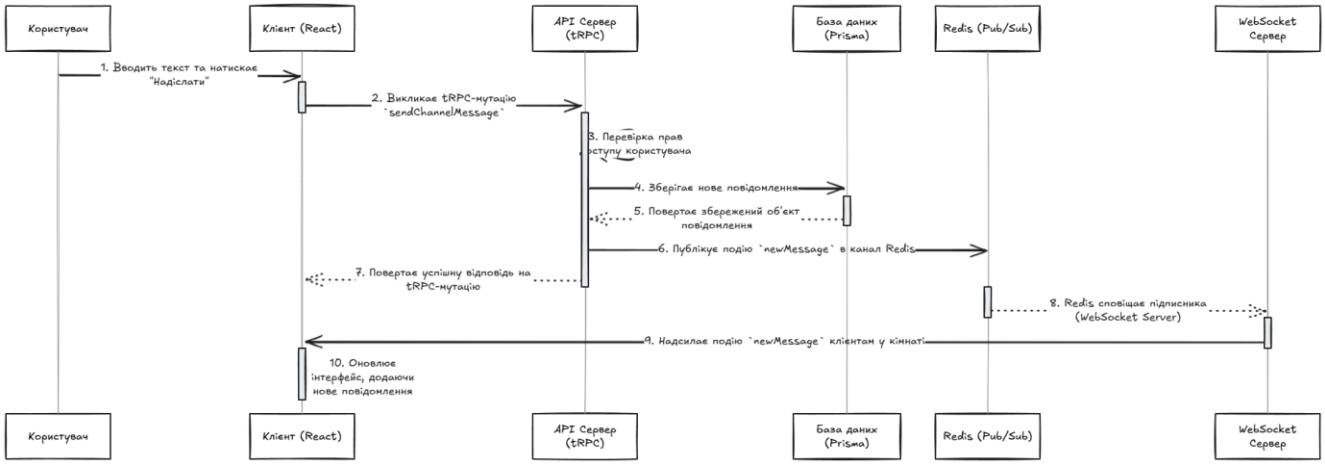

Рисунок 2.19 – Діаграма послідовності для сценарію відправки повідомлення

Авторизований користувач обирає сервер та текстовий канал. Інтерфейс відображає історію повідомлень для обраного каналу, як показано на рисунку 2.20.

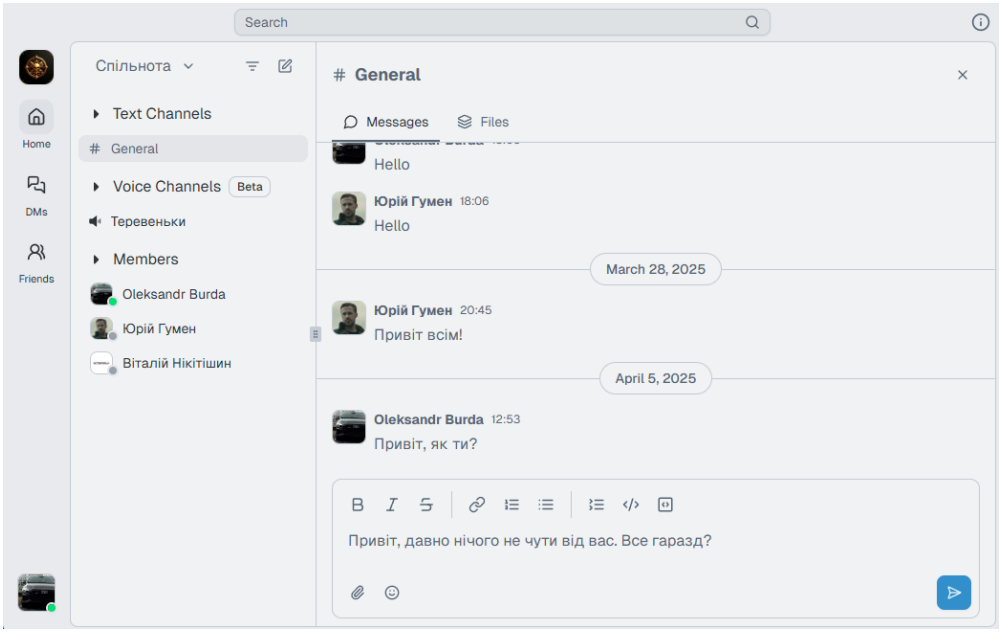

Рисунок 2.20 – Інтерфейс обраного каналу перед відправкою

Після введення тексту та натискання клавіші «Enter», система миттєво відображає нове повідомлення у вікні чату. Цей результат (див. рис. 2.21)

підтверджує коректну роботу всього ланцюжка комунікації в реальному часі, візуалізованого на діаграмі послідовності.

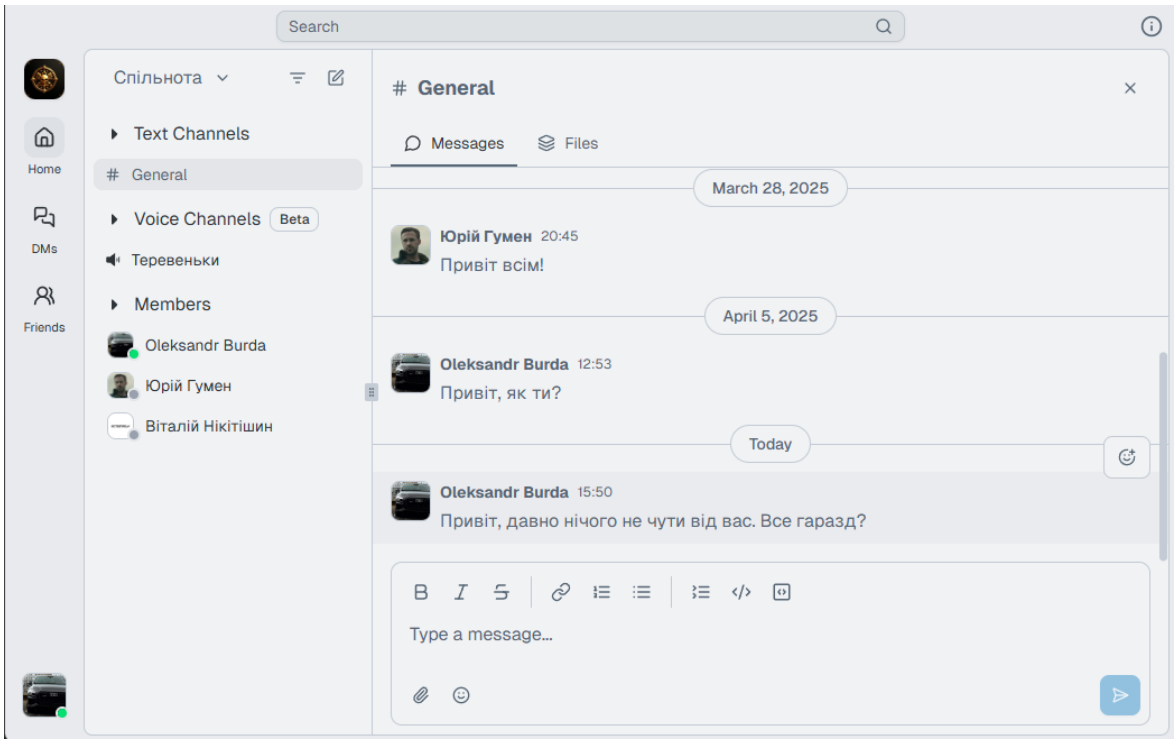

Рисунок 2.21 – Інтерфейс після відправки нового повідомлення

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Фізіологічний вплив факторів існування на життєдіяльність людини

Життєдіяльність людини нерозривно пов'язана із взаємодією з навколишнім середовищем, яке містить численні фактори, здатні чинити прямий фізіологічний вплив на організм. Ці фактори класифікуються на природні, техногенні та соціально-політичні, і кожен з них може викликати специфічні патологічні стани, травми або захворювання, порушуючи нормальне функціонування органів та систем [16, 18].

Природні фактори чинять потужний, часто руйнівний вплив на людину. Тектонічні явища, такі як землетруси, призводять до руйнування будівель, що стає причиною важких механічних травм: переломів кісток, ушкоджень м'яких тканин, черепно-мозкових травм та синдрому тривалого стискання, коли тривалий тиск на частини тіла викликає масивне отруєння організму продуктами розпаду м'язової тканини [16]. Метеорологічні стихійні лиха також мають значний фізіологічний вплив. Урагани та смерчі спричиняють травми через падіння предметів та уламків. Екстремальні температури призводять до перегрівання організму (тепловий та сонячний удари), що характеризується головним болем, нудотою, підвищенням температури тіла та, у важких випадках, втратою свідомості, або до обмороження, яке викликає пошкодження тканин внаслідок дії низьких температур, від некрозу шкіри до змертвіння кінцівок [16].

Техногенні фактори, пов'язані з діяльністю людини, є джерелом різноманітних фізіологічних уражень. Аварії з викидом радіоактивних речовин призводять до іонізаційного опромінення, що викликає променеву хворобу. Її наслідки включають ураження кровотворної, ендокринної, імунної та нервової систем, а також розвиток онкологічних захворювань у довгостроковій перспективі. Аварії з витоком сильнодіючих отруйних речовин (СДОР), таких як аміак, хлор чи метилізоционат, викликають гострі отруєння. Токсини проникають в організм через дихальні шляхи, шкіру або шлунково-кишковий тракт, уражаючи

життєво важливі органи та викликаючи асфіксію (задуху), опіки дихальних шляхів та системну інтоксикацію [16]. Пожежі та вибухи поєднують у собі декілька уражуючих факторів: дія високої температури спричиняє термічні опіки різного ступеня, а вдихання продуктів горіння (чадний газ, синильна кислота) призводить до важких отруєнь та кисневого голодування. Ураження електричним струмом викликає судоми м'язів, порушення роботи серця та дихання, а також специфічні електроопіки в місцях входу та виходу струму [16, 18].

До соціальних факторів, що мають руйнівний фізіологічний вплив, належить наркоманія. Вживання наркотичних речовин викликає стан хронічного отруєння організму та формування двох типів залежності. Психічна залежність проявляється у непереборному бажанні повторного прийому наркотику. Фізична залежність є станом адаптації, коли наркотик вбудовується у процеси обміну речовин. Припинення його вживання викликає важкий фізіологічний стан – абстинентний синдром («ломку»), що супроводжується сильними м'язовими болями, судомами, розладами травлення та серцево-судинної системи [16].

У відповідь на будь-яку важку травму (механічну, опікову, хімічну), що супроводжується сильним болем та крововтратою, в організмі розвивається загальна складна реакція – травматичний шок. Це патологічний процес, що характеризується глибоким пригніченням функцій нервової системи, кровообігу, дихання та обміну речовин. Шок є прямою загрозою для життя і вимагає негайних протишокових заходів [16, 18].

Таким чином, фізіологічний вплив факторів існування є багатогранним і охоплює широкий спектр реакцій організму – від локальних ушкоджень до системних розладів та смерті. Розуміння цих механізмів є основою для розробки заходів безпеки, профілактики та надання ефективної долікарської допомоги [18].

3.2 Підбирання оптимальних параметрів мікроклімату на робочих місцях

Підбирання та підтримання оптимальних параметрів мікроклімату є одним із ключових завдань охорони праці, спрямованим на реалізацію конституційного права працівників на безпечні та здорові умови праці, що закріплено у статті 6 Закону України «Про охорону праці» [17]. Мікроклімат виробничих приміщень – це комплекс фізичних факторів, що впливають на тепловий обмін людини з навколишнім середовищем. Згідно з ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень», основними параметрами, що його характеризують, є температура повітря (t, ℃), відносна вологість (φ, %), швидкість руху повітря (v, м/с) та інтенсивність теплового опромінення ($E_{\text{опр}}$, Вт/м^2) [16, 18]. Здатність організму підтримувати сталу температуру тіла (36,6 ℃) при зміні зовнішніх умов називається терморегуляцією. Значні коливання параметрів мікроклімату призводять до її порушення, що може спричинити перегрів або переохолодження організму, зниження працездатності та розвиток професійних захворювань [16, 18].

Нормативні значення параметрів мікроклімату встановлюються диференційовано, залежно від періоду року (теплий та холодний) та категорії робіт за енерговитратами. Розглянемо конкретний приклад: підбір оптимальних параметрів мікроклімату для офісного працівника, робота якого пов'язана з використанням комп'ютерної техніки. Така діяльність, згідно з класифікацією, належить до категорії Іа (легка фізична робота), що виконується сидячи і не потребує фізичного напруження, з енерговитратами до 139 Вт [18].

Для теплого періоду року на постійному робочому місці для даної категорії робіт встановлюються наступні оптимальні та допустимі параметри мікроклімату [18]:

- Температура повітря: оптимальна – $23 - 25$ ℃; допустима – $22 - 28$ ℃.
- Відносна вологість повітря: оптимальна – $40 - 50$ %; допустима – не

більше 55 % при температурі 28 ℃.

- Швидкість руху повітря: оптимальна – не більше 0,1 м/с; допустима – не більше 0,2 м/с.

Для холодного періоду року норми для цієї ж категорії робіт будуть іншими [18]:

- Температура повітря: оптимальна – $22 - 24$ ℃; допустима – $21 - 25$ ℃.
- Відносна вологість повітря: оптимальна – $40 - 50$ %; допустима – не більше 75 %.
- Швидкість руху повітря: оптимальна – не більше 0,1 м/с; допустима – не більше 0,1 м/с.

Для оцінки відповідності фактичних умов нормативним вимогам проводиться систематичний контроль за допомогою спеціальних приладів: температуру вимірюють термометрами, відносну вологість – психрометрами (Августа, Ассмана), швидкість руху повітря – анемометрами, а інтенсивність теплового випромінювання – актинометрами [16].

Забезпечення нормативних параметрів мікроклімату досягається за допомогою комплексу інженерно-технічних заходів [18]. У холодний період року для компенсації тепловтрат та підтримання заданої температури використовуються системи опалення. Найбільш ефективними та гігієнічними є системи водяного опалення [16]. Для видалення надлишків тепла, вологи та шкідливих речовин, а також для подачі чистого повітря застосовуються системи вентиляції. За способом переміщення повітря вентиляція буває природною (аерація) та механічною (з використанням вентиляторів). За зоною дії розрізняють загально обмінну, яка забезпечує повітрообмін у всьому приміщенні, та місцеву, що видаляє шкідливі речовини безпосередньо від джерела їх утворення (витяжні зонти, кожухи, бортові відсмоктувачі) або локально подає повітря (повітряні душі). Місцева витяжна вентиляція є найбільш ефективною та економічною [16]. Розрахунок необхідного повітрообміну (L, $м^3/год$) виконується на основі розбавлення надлишкової теплоти або шкідливих речовин до гранично

допустимих концентрацій [16].

Найбільш досконалим методом створення та автоматичного підтримання оптимальних параметрів повітряного середовища є кондиціонування. Системи кондиціонування дозволяють не лише нагрівати чи охолоджувати повітря, але й регулювати його вологість та чистоту, що особливо важливо для високотехнологічних виробництв або приміщень зі значними тепловиділеннями [16]. У випадках, коли інженерно-технічні заходи не дозволяють повністю усунути вплив несприятливих метеорологічних умов, обов'язковим є застосування засобів індивідуального захисту (ЗІЗ): спецодягу для захисту від високих чи низьких температур, захисних окулярів від теплового випромінювання тощо [16, 18]. Таким чином, підбирання оптимального мікроклімату є комплексним завданням, що вимагає аналізу умов праці, проведення контрольних вимірювань та впровадження адекватних інженерних рішень для створення безпечного та комфортного робочого середовища [17, 18].

ВИСНОВКИ

Дана кваліфікаційна робота представляє результат комплексної розробки програмної системи – веб-месенджера для комунікації в реальному часі, що охоплює повний інженерний цикл від аналізу вимог до реалізації та тестування. Цей проект став практичним дослідженням сучасних підходів до створення складних, високонавантажених веб-додатків.

Фундаментальним архітектурним вибором стало застосування монорепозиторію, що дозволило централізовано керувати кодовою базою та забезпечити високий рівень консистентності між різними частинами системи. Це рішення, у поєднанні з синергією TypeScript, tRPC та Prisma, дозволило реалізувати ключову мету – досягнення повної безпеки типів на всіх рівнях системи, від запиту до бази даних до рендерингу компонента в браузері, що мінімізувало потенційні помилки при розробці.

Особливу увагу було приділено проектуванню серверної частини, здатної витримувати високі навантаження. Було спроектовано та реалізовано слабозв'язану (декаплінговану) архітектуру для комунікації в реальному часі на основі брокера повідомлень Redis, що забезпечує горизонтальну масштабованість. Реалізація таких складних механізмів, як безпечне завантаження файлів у хмарне сховище та система обмеження частоти запитів, демонструє глибоке розуміння інженерних викликів сучасних веб-систем.

Розробка клієнтської частини на базі бібліотеки React була спрямована на створення динамічного та зручного користувацького інтерфейсу. Було впроваджено сучасні патерни взаємодії, зокрема нескінченну прокрутку для ефективного перегляду історії повідомлень та подієво-керовану модель оновлення, що забезпечило плавну та чутливу до дій користувача роботу додатку.

Для підтвердження надійності та коректності роботи системи було проведено всебічне тестування. Стратегія включала як модульне тестування окремих компонентів серверної логіки за допомогою Jest, так і тестування

наскрізних користувацьких сценаріїв, що гарантувало відповідність продукту функціональним вимогам.

Створений програмний продукт є не лише завершеним рішенням, але й міцним фундаментом для подальшого розширення, такого як впровадження відеодзвінків чи розробка мобільних додатків. З метою демонстрації виконаної роботи та забезпечення прозорості, вихідний код проекту доступний у відкритому репозиторії на платформі GitHub.

Таким чином, цей проект став ключовим етапом у формуванні інженерних компетенцій, поглибивши практичні навички у проектуванні, реалізації та підтримці складних програмних систем, що відповідають сучасним стандартам якості та продуктивності.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 с.
2. Potvin R., Levenberg J. Why Google Stores Billions of Lines of Code in a Single Repository. Communications of the ACM. 2016. Vol. 59, No. 7. С. 78–87.
3. Node.js Foundation. Node.js Documentation. [Електронний ресурс]. URL: https://nodejs.org/en/docs/.
4. Meta Platforms, Inc. React Official Documentation. [Електронний ресурс]. URL: https://react.dev/.
5. tRPC Team. tRPC: End-to-end typesafe APIs made easy. [Електронний ресурс]. URL: https://trpc.io/.
6. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994. 395 с.
7. Petryk, M.R., Boyko, I.V., Khimich, O.M. et al. High-Performance Supercomputer Technologies of Simulation and Identification of Nanoporous Systems with Feedback for n-Component Competitive Adsorption. Cybern Syst Anal 57, 316–328 (2021). https://doi.org/10.1007/s10559-021-00357-7.
8. Redis. Redis Pub/Sub. [Електронний ресурс]. URL: https://redis.io/docs/manual/pubsub/.
9. Google Cloud. Rate limiting strategies and techniques. [Електронний ресурс]. URL: https://cloud.google.com/architecture/rate-limiting-strategies-techniques.
10. Jest. Jest: Delightful JavaScript Testing. [Електронний ресурс]. URL: https://jestjs.io/.
11. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. O’Reilly Media, 2020.
12. Bittner K., Spence I. Use Case Modeling. Addison-Wesley, 2003.
13. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. O’Reilly Media, 2020.

14. Bradshaw S., Brazil E., Chodorow K. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage. O'Reilly Media, 2020.

15. Grigorik I. High Performance Browser Networking. O'Reilly Media, 2013.

16. Курс «Безпека життєдіяльності та основи охорони праці» [Електронний ресурс] / Електронне навчання ТНТУ. – 2025. – Режим доступу: https://dl.tntu.edu.ua/index.php.

17. Закон України «Про охорону праці» від 14.10.1992 № 2694-XII [Електронний ресурс] / Верховна Рада України. – 2025. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2694-12.

18. Безпека життєдіяльності : навчальний посібник [Електронний ресурс] / Т.Є. Стиценко, Г. В. Пронюк, Н. М. Сердюк, І. І. Хондак. – Харків : ХНУРЕ, 2018. – С. 8-15, 27-43, 96-114, 130-136, 231-238, 243-252, 275-284, 295-301. – Режим доступу: https://os.nure.ua/wp-content/uploads/2021/04/posibnik-bgd_2018.pdf.

19. Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: https://dl.tntu.edu.ua/bounce.php?course=5329.

20. Методичні вказівки до виконання дипломної роботи освітнього рівня - бакалавр студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення/ Укладачі : Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

21. Репозиторій GitHub [Електронний ресурс]. – 2025. – Режим доступу до ресурсу: https://github.com/Kanenil/realtime-chat.

ДОДАТКИ

ДОДАТОК А – Тези конференції

Міністерство освіти і науки України
Тернопільська обласна військова адміністрація
Тернопільський національний технічний університет імені Івана Пулюя
Фізико-механічний інститут ім. Г.В. Карпенка НАН України
Інститут прикладних проблем механіки і математики
ім. Я. С. Підстригача НАН України
Інститут електродинаміки НАН України
Інститут фізики напівпровідників ім. В.Є. Лашкарьова НАН України
Наукове товариство імені Шевченка
Vilnius Gediminas Technical University (Литва)
Warsaw University of Technology Lighting Technology Division (Польща)
Iskenderun Technical University (Туреччина);
Lublin University of Technology (Польща)
Technical University of Košice (Словаччина)
Асоціація випускників ТНТУ

МАТЕРІАЛИ

Міжнародної науково-технічної конференції

«ФУНДАМЕНТАЛЬНІ ТА ПРИКЛАДНІ ПРОБЛЕМИ СУЧАСНИХ ТЕХНОЛОГІЙ»

присвячена

180-річчю з дня народження Івана Пулюя та 65-річчю з дня заснування Тернопільського національного технічного університету імені Івана Пулюя

28-29 травня 2025 року
Тернопіль

УДК 004.4
О. Бурда, І. Мудрик
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ПЕРЕВАГИ ВИКОРИСТАННЯ TYPESCRIPT У ПОЄДНАННІ З MONGODB ДЛЯ РОЗРОБКИ СУЧАСНИХ ВЕБ-ДОДАТКІВ

O. Burda, I. Mudryk

ADVANTAGES OF USING TYPESCRIPT IN CONJUNCTION WITH MONGODB FOR DEVELOPING MODERN WEB APPLICATIONS

Зростаюча складність сучасних веб-додатків вимагає інструментів, які забезпечують надійність коду та ефективну роботу з даними. Гнучкість NoSQL баз даних, таких як MongoDB, є привабливою для швидкої розробки та масштабування, але може створювати труднощі у підтримці цілісності даних та зрозумілості структури в кодовій базі динамічно типізованих мов. TypeScript, як надмножина JavaScript зі своєю статичною типізацією, пропонує вирішення багатьох цих проблем.

TypeScript додає статичну типізацію до JavaScript, що дозволяє заздалегідь визначати типи даних і виявляти помилки ще до запуску програми. Це значно знижує ризик виникнення багів під час виконання коду. Завдяки чітко визначеним типам, код стає більш зрозумілим, а його підтримка та рефакторинг — простішими. Крім того, редактори коду та IDE використовують інформацію про типи для покращення автодоповнення та підказок, що сприяє ефективності розробки [1].

MongoDB є популярною документо-орієнтованою NoSQL базою даних, яка зберігає дані у форматі, схожому на JSON [2]. Ключовою особливістю MongoDB є відсутність жорстко фіксованої схеми на рівні бази даних; документи в одній колекції можуть мати різний набір полів. MongoDB розроблена з урахуванням горизонтального масштабування та забезпечує високу швидкість операцій. Хоча гнучка схема сприяє швидкій ітерації, вона може ускладнити підтримку консистентності даних та вимагати ретельнішої валідації на рівні додатку [3].

Використання TypeScript разом із MongoDB дозволяє об'єднати переваги обох технологій. TypeScript дає змогу визначати структуру документів MongoDB через інтерфейси та типи, створюючи своєрідну "схему" на рівні коду додатку. Це компенсує відсутність схеми на рівні бази даних і допомагає підтримувати консистентність даних. Крім того, TypeScript забезпечує коректну роботу з полями документів, гарантуючи правильність типів і запобігаючи помилкам під час виконання програми [4]. Сильна типізація дозволяє уникнути поширених помилок, які виникають через некоректну роботу з даними [4].

Застосування TypeScript у поєднанні з MongoDB є обґрунтованим вибором для розробки сучасних додатків. TypeScript забезпечує структуру та безпеку при роботі з динамічними даними MongoDB, зменшує кількість помилок та значно спрощує розробку та підтримку додатків, що взаємодіють з цією базою даних. Ця синергія технологій призводить до створення більш надійних, підтримуваних та легших у розробці рішень, що використовують переваги гнучкості та масштабованості MongoDB.

Сучасні підходи до аналізу складних систем часто базуються на використанні гібридних моделей, які поєднують різні методи для досягнення більш точних результатів [5].

Література:

1. FOXMINDED.UA. Що таке Typescript і навіщо він потрібен. URL: https://foxminded.ua/typescript/
2. GeeksforGeeks. MongoDB: An introduction. URL: https://www.geeksforgeeks.org/mongodb-an-introduction/
3. MongoDB. Introduction: Schema flexibility vs. "schemaless" databases. URL: https://www.mongodb.com/resources/basics/unstructured-data/schemaless

4. MongoDB. How to Use TypeScript with MongoDB Atlas. URL: https://www.mongodb.com/resources/products/compatibilities/using-typescript-with-mongodb-tutorial
5. Bachynskyi, M., Petryk, M., Brevus, V., Mudryk, I., Glova, B. 3D-hybrid mathematical model for analysis of abnormal neurological movements for the purposes of diagnosis and treatment of limb tremor. 3rd Interna tional Workshop on Information Technologies: Theoretical and Applied Problems, ITTAP 2023. Ternopil. 22–23 November 2023. CEUR Workshop Proceedings, 2023, 183–196.

ДОДАТОК Б – Лістинг функції “withRateLimit” для обмеження частоти запитів

```
/**
 * Creates a rate limiting middleware for TRPC procedures
 *
 * @param options Rate limiting options
 * @returns TRPC middleware that applies rate limiting
 */
export const withRateLimit = (options: RateLimitOptions) => {
  const { limit, windowInSeconds, identifier } = options;

  return t.middleware(async ({ ctx, next, path }) => {
    if (!ctx.session?.user) {
      throw new TRPCError({
        code: 'UNAUTHORIZED',
        message: i18n.t('auth.errors.unauthorized')
      });
    }

    const rateLimitKey = `ratelimit:${identifier ||
   path}:${ctx.session.user.id}`;

    const currentCount = await redis.incr(rateLimitKey);

    if (currentCount === 1) {
      await redis.expire(rateLimitKey, windowInSeconds);
    }

    if (currentCount > limit) {
     await redis.decr(rateLimitKey);
```

Продовження лістингу Додатку Б

```
   const ttl = await redis.ttl(rateLimitKey);

   const { minutes, seconds, resetTimestamp } =
formatTimeRemaining(ttl);


   let errorMessage: string;

   if (minutes > 0) {

     errorMessage =
i18n.t('rateLimit.errors.tooManyRequests_minutes', {

       count: minutes

     });

   } else {

     errorMessage =
i18n.t('rateLimit.errors.tooManyRequests_seconds', {

       seconds

     });

   }


   throw new TRPCError({

     code: 'TOO_MANY_REQUESTS',

     message: errorMessage,

     cause: {

       resetTimestamp,

       rateLimit: {

         limit,

         remaining: 0,

         reset: resetTimestamp

       }

     }

   });

 }


 const ttl = await redis.ttl(rateLimitKey);
```

Продовження лістингу Додатку Б

```
    const resetTime = Math.floor(Date.now() / 1000) + ttl;
    const remaining = limit - currentCount;

    return next({
      ctx: {
        ...ctx,
        rateLimit: {
          limit,
          remaining,
          reset: resetTime
        }
      }
    });
  });
};
```

ДОДАТОК В – Диск з роботою