

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

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»

Завідувач кафедри

_____ Наталія АУШЕВА

“ ” _____ 2026 р.

**Дипломна робота
на здобуття ступеня бакалавр**

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Реактивна система сповіщень із гарантованою доставкою повідомлень”

Виконав: студент 4 курсу, групи ТР-25

Тіхонов Олександр Станіславович

(прізвище, ім’я, по батькові)

_____ (підпис)

Керівник доцент каф. ЦТЕ, к.т.н. Володимир ТИХОХОД

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Рецензент

ст. викладач каф. ТАЕ, доктор філософії, Ольга ВЛАСЕНКО

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Н.контроль асистент Костянтин ЗДОР

(посада, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

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

Студент _____

Київ - 2026

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)

спеціальність 122 “Комп’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ

Завідувач кафедри ЦТЕ

Наталія АУШЕВА

(підпис)

“ ” 2026 р.

ЗАВДАННЯ

на дипломну роботу студенту

Тіхонов Олександр Станіславович

(прізвище, ім’я, по батькові)

1. Тема роботи “Реактивна система сповіщень із гарантованою доставкою повідомлень”

Науковий керівник Тихоход Володимир Олександрович, к.т.н., доцент

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “ ” 2026 року №

2. Термін подання студентом роботи 04.06.2026

3. Вихідні дані до роботи: вимоги до швидкодії (затримка не більше 2-3 секунд), протокол WebSocket, необхідність обробки нестабільного з'єднання, патерн At-least-once delivery, PostgreSQL для збереження станів.

4. Зміст роботи:

- 1) аналіз методів передачі даних у реальному часі та існуючих рішень;
- 2) обґрунтування вибору технологічного стека та середовища розробки;

- 3) проектування архітектури системи за принципами DDD та чистої архітектури;
 - 4) розробка реляційної моделі даних та алгоритму гарантованої доставки;
 - 5) створення відмовостійкої черги фонової обробки;
 - 6) тестування системи та розробка стартап-проєкту.
5. схема інформаційних потоків реактивної системи, архітектурна модель системи (Clean Architecture), ER-діаграма сутностей реляційної бази даних, діаграма послідовності процесу доставки та підтвердження сповіщення, діаграма послідовності синхронізації, схема алгоритму черги повторних спроб.
6. Дата видачі завдання 15.09.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	15.09.25 – 30.09.25	Виконано
2.	Аналіз методів та засобів розв’язання задачі	01.10.25-15.11.25	Виконано
3.	Розробка архітектури та загальної структури системи	16.11.25-31.12.25	Виконано
4.	Розробка окремих підсистем	01.02.26-31.03.26	Виконано
5.	Програмна реалізація системи	01.04.26-10.05.26	Виконано
6.	Оформлення пояснювальної записки	11.05.26-25.05.26	Виконано
7.	Захист програмного забезпечення	26.05.26-31.05.26	Виконано
8.	Передзахист	01.06.26-07.06.26	
9.	Захист	10.06.26-20.06.26	

Студент

(підпис)

Олександр ТІХОНОВ

(прізвище та ініціали)

Керівник

(підпис)

Володимир ТИХОХОД

(прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 81 сторінці, містить 23 ілюстрації, 11 таблиць, 1 додаток, 19 джерел в переліку посилань. Метою роботи є розробка реактивної системи сповіщень, здатної забезпечити гарантовану доставку повідомлень в умовах нестабільного з'єднання. Застосовано принципи предметно-орієнтованого проектування, методи наскрізної нумерації, дедуплікації та управління чергами з експоненційним відступом. Програмно реалізована система сповіщень реального часу з алгоритмами фонові синхронізації суттєво мінімізує ризики втрати чи дублювання інформації при обривах зв'язку. Розроблений програмний комплекс є придатним для забезпечення надійного каналу інформування в корпоративних системах. Ключові слова: РЕАКТИВНА СИСТЕМА, ГАРАНТОВАНА ДОСТАВКА, ІДЕМПОТЕНТНІСТЬ, ДЕДУПЛІКАЦІЯ, ВІДМОВОСТІЙКІСТЬ.

ABSTRACT

The thesis is executed on 81 pages, contains 23 figures, 11 tables, 1 appendix, 19 sources in the reference list. The purpose of the work is the development of a reactive notification system capable of providing guaranteed message delivery under unstable connections. Domain-driven design principles, end-to-end numbering, deduplication, and exponential backoff queues are applied. The implemented real-time notification system with background synchronization algorithms significantly minimizes data loss or duplication risks during connection drops. The developed software system is suitable for providing a reliable notification channel in corporate systems. Keywords: REACTIVE SYSTEM, GUARANTEED DELIVERY, IDEMPOTENCY, DEDUPLICATION, FAULT TOLERANCE.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ.....	7
ВСТУП.....	8
1 ЗАДАЧА ГАРАНТОВАНОЇ ДОСТАВКИ ПОВІДОМЛЕНЬ У РЕАКТИВНИХ СИСТЕМАХ ТА ОГЛЯД ЗАСОБІВ ЇЇ РОЗВ'ЯЗАННЯ	9
1.1 Постановка прикладної задачі	9
1.1.1 Зміст задачі гарантованої доставки повідомлень.....	10
1.1.2 Вимоги до системи для розв'язання задачі	11
1.1.3 Опис вхідної та вихідної інформації	12
1.2 Огляд методів розв'язання задачі.....	14
1.2.1 Технології передачі даних у режимі реального часу	14
1.2.2 Аналіз готових сервісів сповіщень	16
1.2.3 Патерни забезпечення гарантованої доставки.....	18
1.3 Огляд програмних засобів реалізації	19
1.3.1 Засоби серверної частини	19
1.3.2 СУБД та засоби фонові обробки.....	20
1.3.3 Засоби клієнтської частини	21
1.3.4 Інфраструктурне середовище.....	22
2 МЕТОДИ ТА АЛГОРИТМИ ЗАБЕЗПЕЧЕННЯ ГАРАНТОВАНОЇ ДОСТАВКИ ПОВІДОМЛЕНЬ У РЕАКТИВНИХ СИСТЕМАХ	23
2.1 Архітектурні принципи побудови системи	23
2.2 Алгоритми відстеження статусів та дедуплікації повідомлень	24
2.2.1 Формальна модель життєвого циклу повідомлення	24
2.2.2 Динамічне обчислення статусів та дедуплікація повідомлень	27
2.3 Логічна модель синхронізації та черги повторних спроб.....	29
2.3.1 Алгоритм синхронізації станів після відновлення з'єднання	29
2.3.2 Модель експоненційного відступу.....	31
2.4 Алгоритми розмежування доступу та криптографічного захисту	33
2.4.1 Алгоритм перевірки прав доступу.....	33
2.4.2 Принципи криптографічного захисту повідомлень	35

3 ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ РЕАКТИВНОЇ СИСТЕМИ СПОВІЩЕНЬ ІЗ ГАРАНТОВАНОЮ ДОСТАВКОЮ ПОВІДОМЛЕНЬ	36
3.1 Функціональна модель системи	36
3.2 Архітектурна структура програмного комплексу	38
3.3 Реляційна модель даних	39
3.4 Структура серверних програмних модулів	45
3.5 Структура клієнтських програмних модулів	48
3.6 Реалізація відмовостійкої черги фонові обробки	50
3.7 Програмні модулі підсистеми безпеки	52
3.7.1 Програмні модулі авторизації HTTP-запитів	53
3.7.2 Програмні модулі авторизації сокет-з'єднань	55
3.7.3 Програмні модулі криптографічного захисту вмісту повідомлень	56
4 РОЗГОРТАННЯ, ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ РЕАКТИВНОЇ СИСТЕМИ СПОВІЩЕНЬ ІЗ ГАРАНТОВАНОЮ ДОСТАВКОЮ ПОВІДОМЛЕНЬ	58
4.1 Розгортання програмного комплексу	58
4.1.1 Системні вимоги	58
4.1.2 Архітектура розгортання	59
4.1.3 Введення системи в експлуатацію	61
4.2 Демонстрація реалізованих сценаріїв роботи системи	61
4.2.1 Реєстрація та вхід у систему	62
4.2.2 Керування каналами	63
4.2.3 Відправлення сповіщень, статуси доставки та гілки відповідей	64
4.2.4 Робота з push-сповіщеннями	66
4.3 Тестування алгоритмів гарантованої доставки	67
4.3.1 Методика тестування	67
4.3.2 Результати тестування	68
ВИСНОВКИ	71
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	73
ДОДАТОК А	75

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API (Application Programming Interface) — прикладний програмний інтерфейс.

ACK (Acknowledgement) — сигнал підтвердження отримання даних.

DDD (Domain-Driven Design) — предметно-орієнтоване проектування.

FCM (Firebase Cloud Messaging) — хмарний сервіс обміну повідомленнями.

JWT (JSON Web Token) — відкритий стандарт для створення токенів доступу.

ORM (Object-Relational Mapping) — технологія об'єктно-реляційного відображення баз даних.

PWA (Progressive Web Application) — прогресивний вебзастосунок.

RBAC (Role-Based Access Control) — управління доступом на основі ролей.

SSE (Server-Sent Events) — технологія надсилання подій від сервера до клієнта.

UUID (Universally Unique Identifier) — універсальний унікальний ідентифікатор.

ВСТУП

Своєчасне доставлення сповіщень є важливою вимогою для сучасних інформаційних систем. Незважаючи на використання повнодуплексного зв'язку (зокрема WebSocket), розриви мережевого з'єднання часто призводять до втрати або помилкового дублювання повідомлень під час перепідключення клієнтів. Існуючі рішення обмежують можливості керування доставкою та відновленням після збоїв, що потребує реалізації нового архітектурного підходу.

Метою роботи є розробка реактивної системи сповіщень із гарантованою доставкою повідомлень при нестабільному з'єднанні.

Для досягнення мети вирішено такі завдання:

1. Аналіз підходів, методів та алгоритмів розв'язання задачі гарантованої доставки.
2. Аналіз програмних засобів для реалізації системи.
3. Проєктування та розробка програмного забезпечення з механізмами пакетної синхронізації та відмовостійкої черги фонові обробки.
4. Тестування розробленого програмного забезпечення.

Розроблений програмний комплекс забезпечує надійний канал інформування і може використовуватися як альтернатива стороннім сервісам або як компонент корпоративних систем.

Результати дипломної роботи не публікувалися та не доповідалися на наукових конференціях.

Дипломна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел (19 найменувань) та 1 додатка. Загальний обсяг становить 81 сторінку. Робота містить 23 ілюстрації та 11 таблиць.

1 ЗАДАЧА ГАРАНТОВАНОЇ ДОСТАВКИ ПОВІДОМЛЕНЬ У РЕАКТИВНИХ СИСТЕМАХ ТА ОГЛЯД ЗАСОБІВ ЇЇ РОЗВ'ЯЗАННЯ

Розділ присвячено формулюванню прикладної задачі забезпечення гарантованої доставки повідомлень у реактивних системах сповіщень, аналізу теоретичних підходів до її розв'язання та огляду програмних засобів, придатних для реалізації запропонованих рішень. Спочатку конкретизовано задачу через опис інформаційного середовища, у якому виникають втрати та дублювання повідомлень, та визначено мінімальний набір вимог, потрібний для її коректної постановки. Далі розглянуто протоколи передачі даних у режимі реального часу, готові хмарні сервіси інформування та класичні моделі гарантій доставки в розподілених системах. На основі проведеного аналізу обґрунтовано вибір засобів реалізації — з обов'язковим зв'язком кожного інструмента з вимогами поставленої задачі.

1.1 Постановка прикладної задачі

Реактивна система сповіщень — це програмний комплекс, у якому сервер ініціативно надсилає повідомлення про події підключеним клієнтам у міру їх виникнення. Користувач очікує, що кожне сповіщення дійде до нього без втрат, у правильній послідовності та без зайвих повторів, навіть якщо мережеве з'єднання тимчасово розірвалось. Саме на цій вимозі ґрунтується задача гарантованої доставки повідомлень, яку розв'язує дипломна робота.

У підрозділі деталізовано зміст самої задачі, сформульовано вимоги, необхідні для її постановки, та описано структуру вхідних і вихідних інформаційних потоків системи. Технічні рішення (вибір протоколу, конкретні алгоритми, архітектурні шаблони) у цьому підрозділі не розглядаються — їх обґрунтуванню присвячено наступні підрозділи розділу 1 та розділ 2.

1.1.1 Зміст задачі гарантованої доставки повідомлень

Розглянемо предметну область. Учасники системи (відправники та одержувачі) обмінюються повідомленнями через комунікаційний сервер, який зберігає історію та координує доставку. Логічно повідомлення згруповані в тематичні канали; учасник може бути підписаний на декілька каналів і отримувати повідомлення в реальному часі лише з тих, до яких має доступ. Загальну модель предметної області наведено на рисунку 1.1.

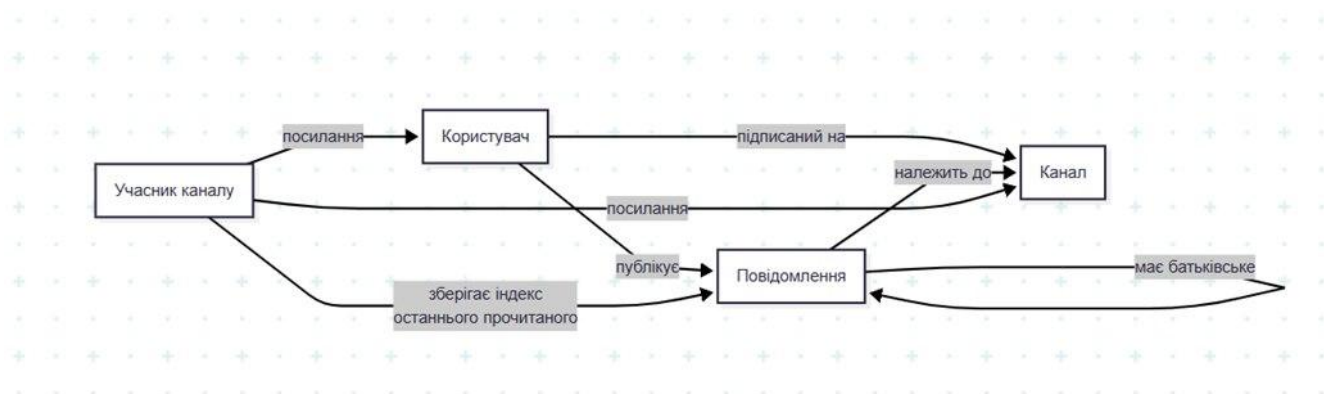


Рисунок 1.1 — Модель предметної області системи сповіщень

Задача гарантованої доставки виникає тому, що передача даних у комп'ютерних мережах не є абсолютно надійною. Аналіз життєвого циклу повідомлення дозволяє виділити три типові ситуації, у яких порушується цілісність інформаційного потоку:

— втрата повідомлення під час розриву з'єднання — повідомлення сформоване й опубліковане сервером, але одержувач не отримує його, оскільки в момент трансляції з'єднання вже розірвано, а штатних засобів повторної передачі немає;

— поява дублікатів — повторно надіслані під час синхронізації повідомлення можуть накладатися на вже наявні в історії клієнта;

— порушення хронології — повідомлення надходять у послідовності, відмінній від хронологічної, або в історії клієнта залишаються «пропуски», які він не виявляє самостійно.

У сукупності описані ситуації формують центральну проблему — стан невизначеності між сервером і клієнтом щодо того, які саме повідомлення було успішно доставлено. Метою постановки задачі є усунення такої невизначеності, тобто побудова такого протоколу взаємодії, у якому клієнт може однозначно визначити, які повідомлення він отримав, які пропустив і чи містить його локальна історія дублікати.

Розв'язання задачі вимагає одночасного забезпечення трьох властивостей: (а) виявлення пропусків в історії одержаних повідомлень, (б) гарантованої повторної доставки пропущених повідомлень після відновлення зв'язку, (в) розпізнавання повторних копій одного й того самого повідомлення на стороні сервера та клієнта. Саме на ці три властивості орієнтовано подальший огляд патернів та засобів.

1.1.2 Вимоги до системи для розв'язання задачі

Для коректної постановки задачі сформулюємо мінімально необхідний набір вимог до проєктованої системи. У цьому підрозділі подано лише ті вимоги, що впливають безпосередньо з умови задачі гарантованої доставки. Деталі реалізації (інтерфейсні рішення, моделі шифрування, конкретні протоколи) розглядаються в подальших розділах роботи.

Першу групу складають вимоги до самого процесу доставки повідомлень. Кожне опубліковане повідомлення має бути надіслане одержувачам хоча б один раз, у правильному хронологічному порядку, та не має породжувати дублікатів у локальній історії клієнта незалежно від кількості повторних спроб передачі. У разі тимчасового розриву з'єднання система зобов'язана зберегти інформацію про невручене повідомлення та виконати його доставку після відновлення зв'язку — без втручання користувача.

Другу групу складають вимоги до відновлення сесії клієнта. Після повторного встановлення з'єднання клієнт повинен мати можливість запросити в сервера усі повідомлення, які з'явилися в каналі після певної відомої клієнту позиції. Це означає, що система повинна забезпечувати однозначну ідентифікацію кожного повідомлення та можливість відновлення їх правильної послідовності на стороні клієнта.

Третю групу складають вимоги до структури комунікації, без яких задача не може бути сформульована повноцінно. До них належить підтримка тематичних каналів як ізольованих логічних просторів обміну повідомленнями, можливість групування повідомлень у гілки відповідей навколо батьківського повідомлення та розмежування прав учасників каналу (право на публікацію та на читання). Ці елементи безпосередньо впливають на алгоритми вибірки та доставки пропущених повідомлень, тому повинні фіксуватися ще на етапі постановки задачі.

Нефункціональні характеристики (обмеження на час доставки, обсяг паралельних з'єднань, моделі безпеки, обчислювальні ресурси) уточнюються на етапі проєктування й розглядаються в розділах 2 та 4.

1.1.3 Опис вхідної та вихідної інформації

Інформаційну взаємодію в системі утворюють потоки структурованих повідомлень між клієнтом і сервером. Логічну схему цих потоків наведено на рисунку 1.2.

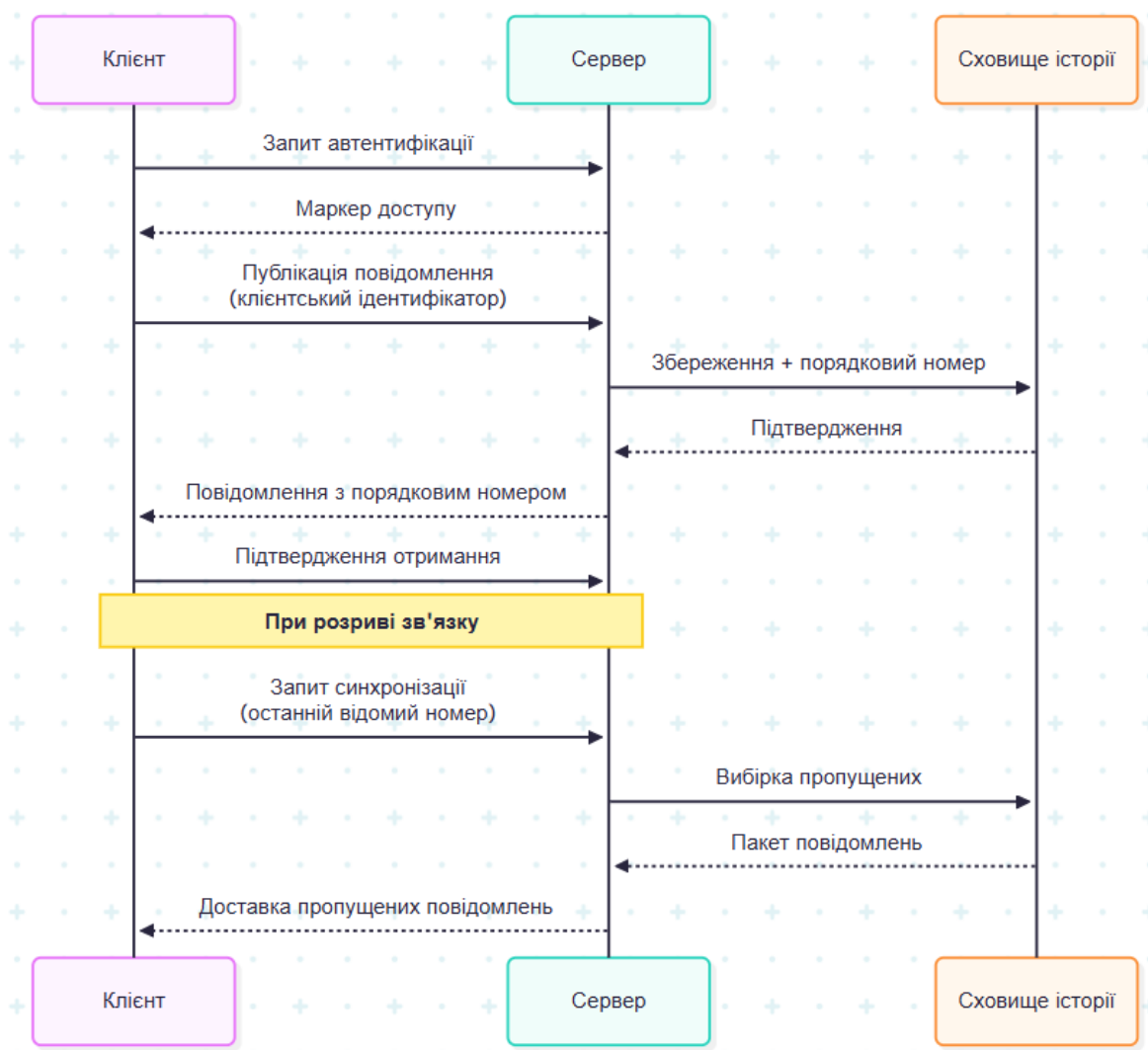


Рисунок 1.2 — Схема інформаційних потоків системи

Вхідну інформацію, що надходить від клієнта до сервера, утворюють три логічні групи:

1. Дані ідентифікації та автентифікації користувача й цільового каналу.
2. Корисне навантаження повідомлення (текстовий зміст, рівень пріоритету, посилання на вкладення, а також службові метадані для ідентифікації та організації гілок відповідей).
3. Службові запити (підтвердження отримання повідомлень та запити на синхронізацію після розриву з'єднання).

Вихідну інформацію, що надсилається сервером до клієнта, також утворюють три групи:

1. Сформовані повідомлення, доповнені серверними атрибутами (ідентифікаторами, часовими мітками та даними про їх послідовність).
2. Метадані каналів (перелік активних учасників, інформація про ролі та кількість непрочитаних повідомлень).
3. Службові відповіді (підтвердження прийняття повідомлення сервером, повідомлення про помилки або відмову в доступі).

Така структура потоків у поєднанні з наскрізною нумерацією та обмеженням унікальності клієнтських ідентифікаторів утворює основу для розв'язання задачі гарантованої та ідемпотентної обробки повідомлень.

1.2 Огляд методів розв'язання задачі

Для побудови розв'язання поставленої задачі необхідно проаналізувати існуючі підходи на двох рівнях: транспортному (як саме доставляти повідомлення між клієнтом і сервером) та прикладному (як гарантувати цілісність повідомлень за наявності розривів зв'язку). У підрозділі також розглянуто наявні готові сервіси інформування з метою з'ясування, наскільки вони задовольняють вимоги поставленої задачі.

1.2.1 Технології передачі даних у режимі реального часу

Класична модель «запит — відповідь» протоколу HTTP не передбачає ініціативної передачі даних від сервера, що ускладнює реалізацію реактивних сповіщень. Для усунення цього обмеження сформовано три основні підходи, які відрізняються напрямком передачі, обсягом службового трафіку та поведінкою під час розривів зв'язку. Узагальнене порівняння цих підходів за критеріями, важливими для розв'язання задачі гарантованої доставки, наведено в таблиці 1.1.

Таблиця 1.1 — Порівняння технологій передачі даних у режимі реального часу

Критерій	Тривале опитування (Long Polling)	Події від сервера (SSE)	Двосторонній зв'язок (WebSocket)
Напрямок передачі	Однобічний (емуляція)	Однобічний (від сервера)	Двобічний
Обсяг службового трафіку	Високий (HTTP-заголовки на кожному циклі)	Середній	Низький (легкі фрейми)
Затримка передачі	Висока	Низька	Низька
Відновлення після розриву	Через новий HTTP-запит	Вбудоване (на рівні браузера)	На рівні застосунку
Підтримка підтвердження отримання від клієнта	Окремими запитами	Лише через окремий канал	У межах того самого з'єднання
Сумісність зі старими проміжними системами	Висока	Середня	Низька

Тривале опитування [1] реалізується як HTTP-запит, який сервер тримає відкритим до появи нових даних або вичерпання таймауту. Підхід зберігає сумісність із застарілими проксі-серверами та фаєрволами, проте на кожному циклі передає повний обсяг службових заголовків і відкриває нове з'єднання, що створює надлишкове навантаження за великої кількості клієнтів.

Технологія подій від сервера [2] (Server-Sent Events, SSE) використовує одне тривале HTTP-з'єднання, через яке сервер транслює клієнту потік повідомлень у текстовому форматі. Перевагою є наявність вбудованого механізму повторного підключення на рівні браузера. Принциповим обмеженням для поставленої задачі є однобічність протоколу: для надсилання сигналів підтвердження отримання та запитів синхронізації необхідно піднімати окремі HTTP-запити, що ускладнює координацію станів.

Протокол WebSocket [3] забезпечує двобічну передачу даних у межах одного постійного з'єднання після початкового рукостискання. Це дозволяє в межах того самого каналу передавати як основний потік повідомлень, так і службові сигнали (підтвердження, запити синхронізації, службові пакети підтримки активності) без додаткових запитів. Логіка повторного підключення після розриву реалізовується на прикладному рівні, але саме це й створює простір для проєктування механізмів гарантованої доставки. Для задач, у яких клієнт активно надсилає підтвердження отримання, WebSocket є найбільш придатним вибором серед розглянутих варіантів.

1.2.2 Аналіз готових сервісів сповіщень

З метою з'ясування доцільності розроблення власної системи проведено порівняння трьох поширених хмарних сервісів інформування за критеріями, що відображають вимоги поставленої задачі: контроль над послідовністю доставки, можливість реалізації власної логіки дедуплікації, придатність для двобічної взаємодії та доступність для самостійного розгортання.

Більшість готових хмарних платформ орієнтовані на масову розсилку за моделлю «щонайбільше один раз» і суттєво обмежують прикладний контроль розробника над чергами. Натомість використання чистого протоколу WebSocket переносить відповідальність на рівень застосунку, що дозволяє гнучко проєктувати власні алгоритми точкової пакетної синхронізації та суворо контролювати порядок доставки, уникаючи при цьому критичної залежності від сторонньої інфраструктури. Результати порівняння подано в таблиці 1.2.

Таблиця 1.2 — Аналіз готових сервісів сповіщень

Критерій	Firebase Cloud Messaging	Pusher	OneSignal
Орієнтація сервісу	Мобільні push-сповіщення	Двосторонні канали реального часу	Маркетингові розсилки
Контроль над послідовністю доставки	Обмежений	Закритий алгоритм	Не передбачено
Підтримка наскрізної нумерації	Не передбачено	Не передбачено	Не передбачено
Підтримка дедуплікації за клієнтським ключем	Не передбачено	Не передбачено	Не передбачено
Двобічна взаємодія з підтвердженням	Часткова	Так	Ні
Можливість самостійного розгортання	Ні	Ні	Ні
Модель тарифікації	Безкоштовна (з обмеженнями)	Платна (за з'єднанням)	Платна (за пристроєм)

З наведеного порівняння випливає, що жоден із розглянутих сервісів не надає прикладного контролю над механізмами наскрізної нумерації та дедуплікації повідомлень, що є ключовими для розв'язання поставленої задачі. За критерієм контролю над послідовністю доставки та можливості налаштування власної логіки повторної доставки сервіси FCM і OneSignal не відповідають вимогам, а сервіс Pusher, маючи відповідні можливості на абстрактному рівні, реалізує їх у вигляді закритого програмного інтерфейсу, що не дозволяє інтегрувати їх із прикладною моделлю даних розроблюваної системи. На основі цього, для розв'язання задачі обґрунтовано розроблення власного програмного комплексу з самостійним розгортанням, що забезпечує повний контроль над алгоритмами доставки.

1.2.3 Патерни забезпечення гарантованої доставки

У теорії розподілених систем виділяють три класичні моделі гарантій доставки повідомлень [4], які відрізняються поведінкою у разі мережових збоїв та накладеними обмеженнями на обчислювальну вартість. Стисло проаналізуємо кожную модель з огляду на її придатність до поставленої задачі.

Модель «щонайбільше один раз» (at-most-once) передбачає, що повідомлення надсилається без очікування підтвердження. Підхід забезпечує найвищу пропускну здатність, проте будь-який мережовий збій спричиняє безповоротну втрату повідомлення, що для систем критичного інформування є неприйнятним.

Модель «щонайменше один раз» (at-least-once) передбачає, що сервер очікує від клієнта сигналу підтвердження отримання, а в разі його відсутності протягом визначеного інтервалу повторно надсилає повідомлення. Така модель усуває втрати, проте за певних умов (наприклад, втрати сигналу підтвердження в каналі) допускає появу дублікатів на стороні клієнта.

Модель «рівно один раз» (exactly-once) передбачає доставку повідомлення до одержувача рівно один раз без втрат та дублювання. Втім, у розподілених системах загального призначення повна реалізація цієї моделі вимагає узгодженого протоколу між усіма вузлами та обмежує пропускну здатність системи. Тому в інженерній практиці її, як правило, реалізують не як властивість транспортного рівня, а як ефект, що досягається сукупністю механізмів на кількох рівнях системи.

На основі проведеного аналізу для розв'язання поставленої задачі доцільно використовувати комбіновану модель, яка поєднує гарантії доставки на транспортному рівні з ідемпотентною обробкою операцій на прикладному рівні. Детальний опис обраного підходу та конкретних механізмів його реалізації розглядається в другому розділі.

1.3 Огляд програмних засобів реалізації

Підрозділ містить огляд програмних засобів, придатних до реалізації запропонованих рішень. Для кожного засобу акцент зроблено на тому, як саме його властивості відповідають вимогам поставленої задачі. Описи, що не мають безпосереднього зв'язку з гарантованою доставкою, скорочено до мінімуму.

1.3.1 Засоби серверної частини

Сервер реактивної системи сповіщень підтримує велику кількість одночасних з'єднань та реагує на події з низькою затримкою. Цим вимогам найкраще відповідає середовище виконання з неблокуючою обробкою введення-виведення. Серед розглянутих варіантів — Java, Go, Python, Node.js — для реалізації обрано Node.js, оскільки його подієва модель обробки запитів дозволяє ефективно утримувати тисячі постійних з'єднань у межах одного системного процесу, без виділення окремого потоку на кожен сокет. Додатковим аргументом є можливість використання спільної мови програмування (TypeScript) на серверній та клієнтській сторонах, що зменшує ймовірність розходжень у форматах обміну даними.

На відміну від багатопотокових платформ, де утримання багатьох постійних з'єднань призводить до значних витрат оперативної пам'яті на перемикання контексту між потоками, подієва архітектура Node.js мінімізує інфраструктурні витрати. Це дозволяє виділити більше обчислювальних ресурсів безпосередньо під логіку буферизації та первинної фільтрації пакетів.

Архітектурною основою серверної частини обрано фреймворк NestJS [5]. Поряд із мікрофреймворками (Express.js, Fastify) він пропонує сувору модульну структуру, побудовану на принципах інверсії залежностей. Для задачі гарантованої доставки ця властивість важлива тим, що дозволяє чітко відокремити прикладні сценарії (підтвердження отримання, синхронізація після відновлення з'єднання,

дедуплікація) від інфраструктурних адаптерів (з'єднання з базою даних, керування чергами, транспортний шар). NestJS також надає вбудовані абстракції для роботи з протоколом WebSocket, що скорочує обсяг технічного коду, необхідного для підтримки постійних з'єднань.

Для роботи з реляційною базою даних обрано Prisma ORM [6]. Її особливість — генерація строго типізованого клієнта на основі декларативної схеми даних. Це безпосередньо відповідає вимогам поставленої задачі: будь-яке порушення цілісності схеми (наприклад, спроба зберегти повідомлення без клієнтського ідентифікатора або з порушенням обмеження унікальності) виявляється ще на етапі компіляції, що знижує ризик некоректної реалізації механізмів дедуплікації та наскрізної нумерації.

1.3.2 СУБД та засоби фонові обробки

Сховище даних має задовольняти двом ключовим для задачі вимогам: суворо підтримувати атомарність та послідовність генерації порядкових номерів повідомлень у межах кожного каналу та забезпечувати швидку пакетну вибірку повідомлень, що належать певному діапазону порядкових номерів. Цим вимогам відповідають реляційні СУБД із підтримкою принципів ACID. Серед розглянутих варіантів (PostgreSQL, MySQL, MongoDB) обрано PostgreSQL [7], оскільки документо-орієнтовані сховища (MongoDB) не гарантують суворої транзакційності за умови складних оновлень, які виходять за межі одного документа, що для задачі генерації порядкових номерів є критичним.

Порівняно з іншими реляційними системами керування базами даних, PostgreSQL пропонує більш гнучкі механізми блокувань на рівні рядків та суворої ізоляції транзакцій. Це є важливим для запобігання виникненню ситуацій конкуренції під час одночасного оновлення показників прочитання повідомлень багатьма учасниками каналу.

Для координації стану між кількома серверними вузлами та реалізації фонові обробки повторних спроб обрано Redis [8]. На відміну від класичних кеш-сховищ (Memcached), Redis підтримує структури даних, придатні для організації черг, та механізм публікації/підписки. Це дозволяє використовувати його одночасно як швидку шину сповіщень між вузлами серверного кластера та як основу для організації відмовостійкої черги повторних спроб.

Для організації власне фонові черги повторних спроб обрано бібліотеку BullMQ [9], яка функціонує поверх Redis. Серед розглянутих альтернатив — промислових брокерів повідомлень (RabbitMQ, Apache Kafka) — BullMQ обрано тому, що вона не потребує окремого серверного компонента та надає вбудовану підтримку моделі експоненційного збільшення інтервалів повторних спроб і пріоритезації завдань — саме тих механізмів, які потрібні для прикладної задачі повторної доставки невручених повідомлень.

1.3.3 Засоби клієнтської частини

Клієнтська частина системи має оперативно відображати зміни стану кожного повідомлення (надіслано, доставлено, прочитано) та підтримувати локальне виявлення пропусків в історії. Для побудови інтерфейсу обрано бібліотеку React [10] — її компонентна модель та механізм віртуального представлення документа дозволяють обмежувати оновлення інтерфейсу лише тими елементами, статус яких фактично змінився, що особливо важливо за умови інтенсивного потоку вхідних подій.

Для взаємодії з серверним протоколом WebSocket обрано бібліотеку Socket.IO [11]. У задачі гарантованої доставки її головною перевагою є вбудована логіка автоматичного відновлення з'єднання після розриву, на основі якої будується клієнтський алгоритм запиту синхронізації, а також можливість прозорого переходу на тривале опитування, якщо проміжне мережеве обладнання блокує протокол WebSocket.

Використання вбудованих низькорівневих механізмів контролю активності з'єднання за допомогою службових сигналів забезпечує своєчасне виявлення прихованих обривів мережі ще до того, як операційна система зафіксує втрату зв'язку, що дозволяє миттєво перевести клієнтський застосунок у режим автономної роботи.

Для отримання сповіщень за умови, коли вкладку браузера закрито або застосунок переведено у фоновий режим, клієнтську частину доповнено механізмом фонових повідомлень браузера [12] на основі службового робітника [13] (service worker). Це дозволяє системі доставляти критичні сповіщення без активної сокет-сесії, доповнюючи основну модель доставки додатковим резервним каналом.

1.3.4 Інфраструктурне середовище

З метою забезпечення відтворюваності розгортання та незалежності від конкретної серверної платформи обрано засоби контейнеризації Docker [14]. Кожен інфраструктурний компонент системи (база даних, сховище в оперативній пам'яті, сервер застосунку, статичні файли інтерфейсу) виконується в окремому контейнері, а опис їхньої взаємодії здійснюється декларативно через файл оркестрації.

Інші елементи інфраструктури розробника (інтегроване середовище розробки, система контролю версій, об'єктне сховище для медіафайлів) не мають безпосереднього впливу на властивості розв'язку задачі гарантованої доставки і застосовуються в стандартній конфігурації, тому їх докладний опис у роботі опущено.

2 МЕТОДИ ТА АЛГОРИТМИ ЗАБЕЗПЕЧЕННЯ ГАРАНТОВАНОЇ ДОСТАВКИ ПОВІДОМЛЕНЬ У РЕАКТИВНИХ СИСТЕМАХ

Розділ присвячено опису методів та алгоритмів, що забезпечують розв'язання сформульованої в розділі 1 задачі гарантованої доставки. Спочатку обґрунтовано архітектурні принципи побудови системи, далі формалізовано алгоритми відстеження життєвого циклу повідомлення та обчислення його статусів, алгоритм синхронізації станів після відновлення з'єднання, модель експоненційного відступу для черги повторних спроб, а також алгоритми розмежування доступу та принципи криптографічного захисту. Програмна реалізація запропонованих алгоритмів — структура модулів, конкретні засоби СУБД та конфігурація інфраструктури — розглядається в розділі 3.

2.1 Архітектурні принципи побудови системи

Для розв'язання задачі гарантованої доставки принциповим є відокремлення бізнес-логіки (правил коректної поведінки) від інфраструктурних деталей (транспортного протоколу, СУБД, фреймворку). За такого відокремлення алгоритми наскрізної нумерації, підтвердження отримання та дедуплікації формулюються незалежно від конкретних технологій, що дозволяє їх верифікувати та модифікувати без втручання в інфраструктурний шар.

В основу системи покладено два архітектурні принципи: предметно-орієнтоване проєктування [15] та чиста архітектура [16]. Їх використання безпосередньо сприяє вирішенню задачі гарантованої доставки та забезпеченню відмовостійкості.

Застосування предметно-орієнтованого проєктування дозволяє ізолювати складну бізнес-логіку (правила обчислення статусів повідомлень, алгоритми перевірки на дублікати та наскрізну нумерацію) у незалежних доменних сутностях.

Це мінімізує ймовірність помилок при паралельній обробці великого потоку повідомлень.

Водночас чиста архітектура забезпечує суворе відокремлення цих критичних алгоритмів від нестабільного інфраструктурного шару. Завдяки цьому механізми відстеження станів та черги повторної доставки працюють повністю ізольовано від розривів мережеских WebSocket-з'єднань або тимчасових збоїв бази даних. Це створює надійний фундамент для відмовостійкості системи: при падінні інфраструктурного вузла логіка доставки не порушується, а лише ставиться на паузу.

2.2 Алгоритми відстеження статусів та дедуплікації повідомлень

Розв'язання задачі гарантованої доставки потребує формалізації трьох взаємопов'язаних механізмів: відстеження життєвого циклу повідомлення, обчислення статусів доставки та прочитання на основі порядкових номерів і дедуплікації повторних запитів. У підрозділі ці механізми описано в термінах предметної області, незалежно від конкретних технологій реалізації.

2.2.1 Формальна модель життєвого циклу повідомлення

Кожне повідомлення в системі моделюється як скінченний автомат із чотирма станами. У стані Очікування перебуває повідомлення, сформоване відправником, але ще не зафіксоване сервером. У стані Відправлено — повідомлення, збережене в історії та надіслане одержувачам без підтвердження доставки. У стані Доставлено — повідомлення, для якого отримано підтвердження прийняття принаймні від одного одержувача. У стані Прочитано — повідомлення, факт ознайомлення з яким зафіксовано принаймні одним учасником каналу, окрім автора.

На рівні взаємодії клієнта з інтерфейсом застосовується метод оптимістичного оновлення: повідомлення відображається у списку історії одразу після ініціювання відправки, без очікування відповіді сервера. На сторону сервера повідомлення надходить разом із тимчасовим клієнтським ідентифікатором, а після успішного збереження в історії воно отримує постійний ідентифікатор і порядковий номер, які повертаються відправнику для заміни тимчасового запису. Такий метод дозволяє відокремити сприйманий користувачем час відгуку від реального часу збереження повідомлення, що особливо важливо при тимчасових мережових затримках.

Важливою властивістю запропонованої моделі скінченного автомата є суворі односпрямованість (монотонність) переходів після успішної фіксації повідомлення на сервері. Оскільки в реактивному середовищі сигнали підтвердження можуть надходити з порушенням хронологічного порядку через різну швидкість мережових маршрутів, система повинна коректно обробляти такі колізії. Завдяки автомату станів система автоматично ігнорує запізнілі або дубльовані події нижчого рівня. Наприклад, якщо для певного повідомлення вже зафіксовано стан «Прочитано», будь-які наступні запізнілі сигнали про його базову доставку відхиляються. Це дозволяє усунути стан гонитви при паралельній обробці запитів та гарантувати цілісність відображення історії.

Переходи між станами ініціюються чотирма типами подій: збереження повідомлення в історії, підтвердження отримання, підтвердження перегляду та розрив зв'язку до моменту збереження. У разі настання останньої події повідомлення повертається у стан очікування й передається до черги повторних спроб (див. підрозділ 2.3.2). Діаграму станів життєвого циклу повідомлення наведено на рисунку 2.1.



Рисунок 2.1 — Діаграма станів життєвого циклу повідомлення

Перевага скінченно-автоматної моделі полягає в тому, що алгоритм опрацювання повідомлення не залежить від конкретного транспорту, через який надходять події, ані від механізму виявлення видимості повідомлень на стороні одержувача — ці елементи виступають як зовнішні джерела подій. Конкретні засоби генерування таких подій (механізм виявлення видимості елементів [17] інтерфейсу на клієнті, обробники сокет-подій на сервері) розглянуто в розділі 3.

2.2.2 Динамічне обчислення статусів та дедуплікація повідомлень

Для зменшення обсягу даних, що зберігаються в історії, у системі застосовано підхід, за якого статус прочитання повідомлень обчислюється динамічно, а не зберігається окремо для кожної пари «повідомлення — учасник». Замість проміжної таблиці зв'язку у схемі даних використовуються два числові поля: наскрізний порядковий номер повідомлення в межах каналу та показчик останнього прочитаного повідомлення для кожного учасника. Повідомлення вважається прочитаним учасником, якщо його порядковий номер не перевищує цього показчика.

Така математична основа дозволяє реалізувати пакетне оновлення статусів на стороні клієнта. При отриманні події про зміну показчика прочитання будь-яким учасником клієнтський застосунок одноразово перераховує стани локального кешу: всі повідомлення з порядковим номером, меншим або рівним новому значенню показчика, набувають статусу «прочитано цим учасником». Це дозволяє уникнути надсилання окремих подій для кожного елемента історії й знижує мережеве навантаження за умови інтенсивного оновлення статусів.

Цей же принцип використовується для миттєвого обчислення множини учасників, які прочитали конкретне повідомлення. Замість зберігання списку «хто прочитав» система виконує фільтрацію множини учасників каналу за умовою «показчик прочитання учасника $\geq$ порядковий номер повідомлення», виключаючи з результату автора публікації. Таке обчислення виконується на стороні клієнта без додаткових запитів до сервера.

Для запобігання появі дублікатів в історії застосовується алгоритм дедуплікації, що ґрунтується на унікальному клієнтському ідентифікаторі, який генерується відправником до моменту надсилання та зберігається разом із повідомленням під обмеженням унікальності в межах каналу. При надходженні запиту на публікацію сервер визначає, чи існує в історії повідомлення з таким самим ідентифікатором; якщо існує — повертає посилання на наявне повідомлення

без створення нового запису. Атомарність перевірки забезпечується на рівні моделі даних, що унеможливлює одночасне збереження двох копій того самого повідомлення навіть за умови повторних запитів.

Описаний механізм є основою для алгоритму синхронізації станів після відновлення з'єднання: оскільки клієнт постійно знає свій показчик останнього прочитаного повідомлення, він може запросити в сервера всю упущену пачку повідомлень одним запитом. Загальну схему взаємодії компонентів у межах механізму гарантованої доставки наведено на рисунку 2.2.

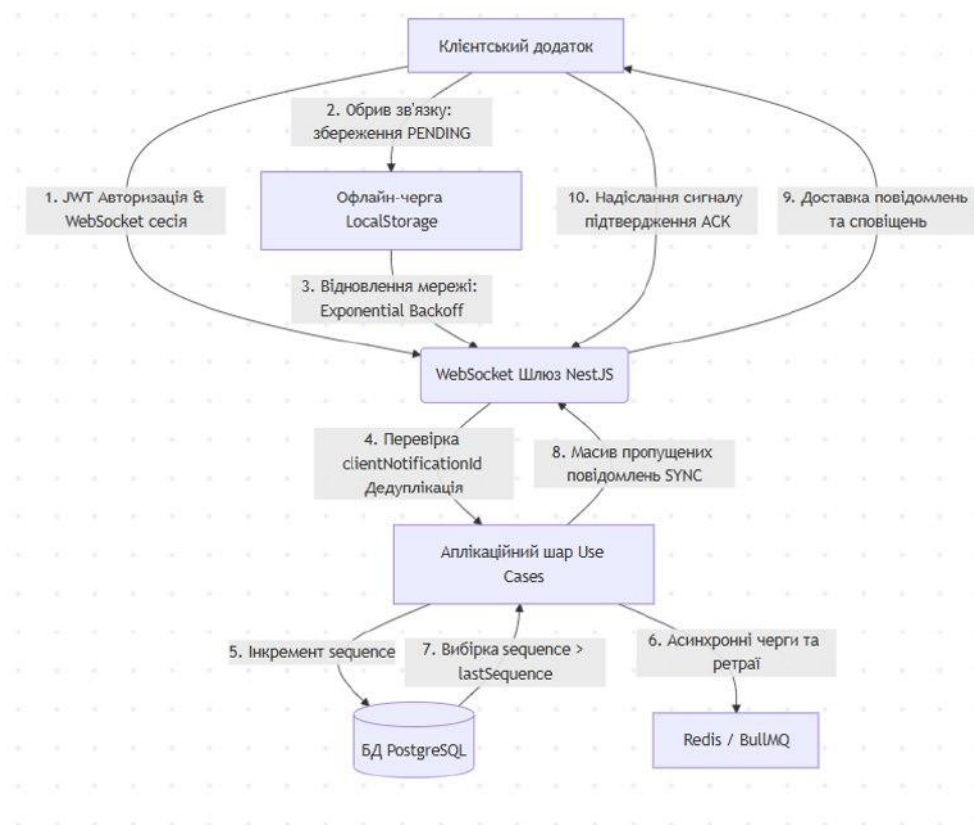


Рисунок 2.2 — Схема інформаційних потоків та механізму гарантованої доставки

Поєднання наскрізної нумерації, динамічного обчислення статусів та дедуплікації за клієнтським ідентифікатором утворює замкнений алгоритмічний

контур, у межах якого подальші алгоритми синхронізації та повторної доставки (підрозділ 2.3) працюють як надбудова над спільним набором числових полів.

2.3 Логічна модель синхронізації та черги повторних спроб

У підрозділі формалізовано два алгоритми: синхронізації пропущених повідомлень після відновлення з'єднання та повторних спроб збереження повідомлень за тимчасової недоступності інфраструктури. Перший забезпечує цілісність історії з боку клієнта, другий — стійкість процесу публікації з боку сервера.

2.3.1 Алгоритм синхронізації станів після відновлення з'єднання

Алгоритм синхронізації активується у двох випадках: автоматично — при встановленні нового з'єднання після розриву; та явно — за запитом клієнта, який під час обміну даними самотійно виявив пропуск у послідовності порядкових номерів власної історії. В обох випадках алгоритм ґрунтується на тому, що клієнт для кожного каналу зберігає останній відомий йому порядковий номер, а сервер має змогу пакетно вибрати з історії всі повідомлення з більшими номерами.

Запропонований підхід суттєво зменшує навантаження на пропускну здатність мережі, оскільки замість повного перезавантаження історії клієнтський застосунок отримує виключно різницю між локальним та серверним станами. Крім того, пакетна передача гарантує суворий хронологічний порядок відновлення, що є необхідною умовою для коректного оновлення інтерфейсу без ризику виникнення конфліктів чи візуального дублювання інформації.

Логіку роботи процесу відновлення втрачених даних формалізовано у вигляді схеми алгоритму пакетної синхронізації пропущених повідомлень (рис. 2.3).

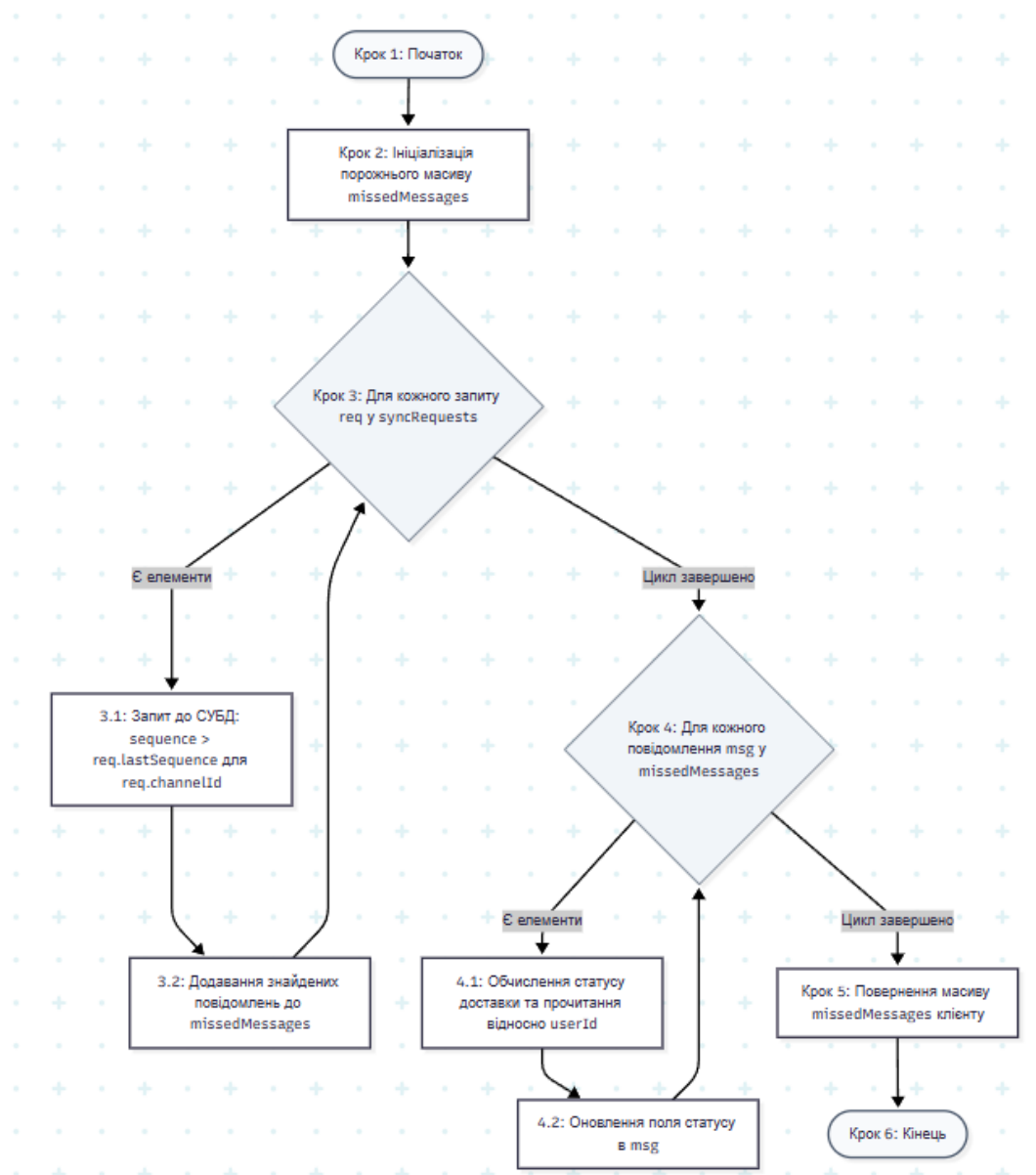


Рисунок 2.3 — Схема алгоритму пакетної синхронізації пропущених повідомлень

У режимі явної синхронізації за запитом клієнта алгоритм виконується аналогічно, з тією відмінністю, що замість покажчика прочитаного у вхідних параметрах використовується конкретний номер, починаючи з якого клієнт виявив пропуск. Послідовність кроків алгоритму та взаємодію клієнтського застосунку, сокет-шлюзу й сховища історії наведено на рисунку 2.4.

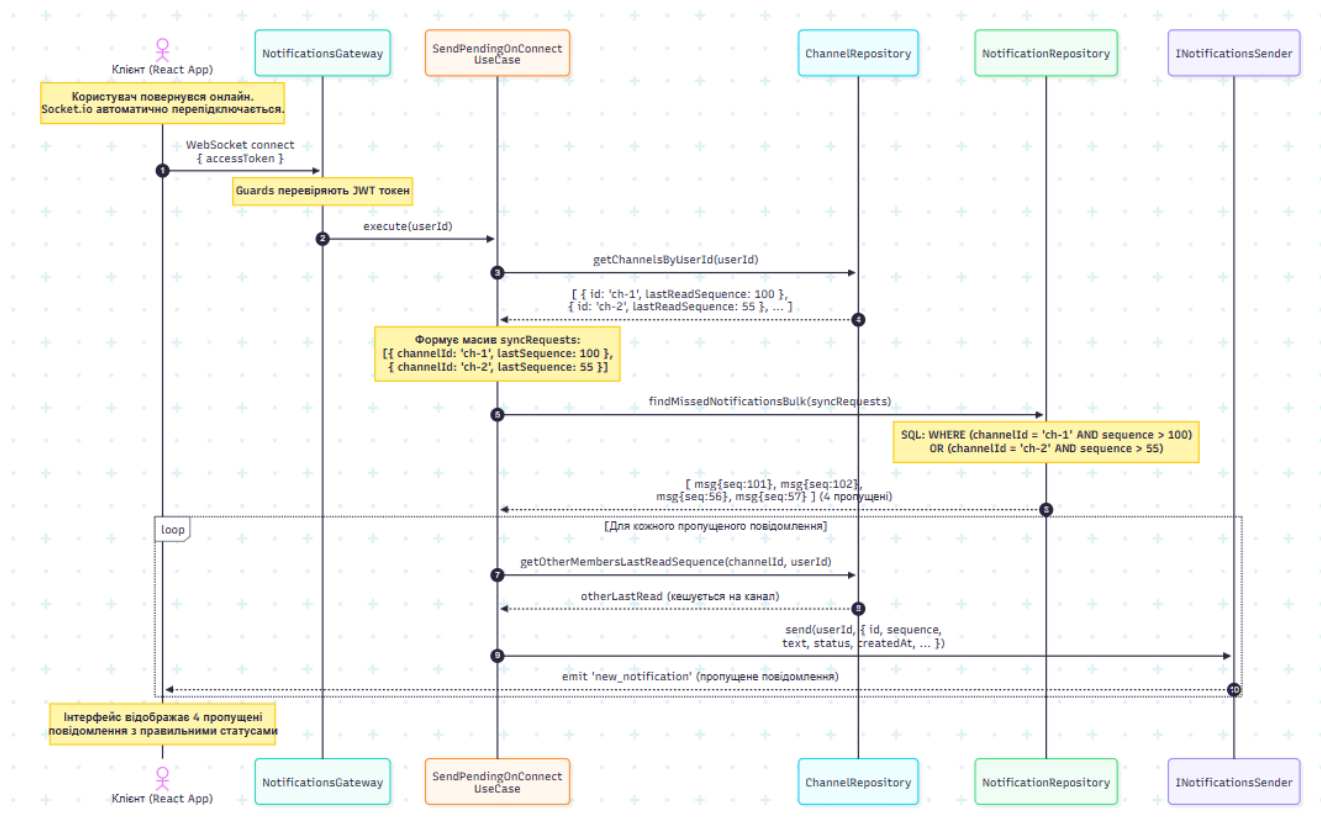


Рисунок 2.4 — Діаграма послідовності синхронізації пропущених сповіщень

Пакетна вибірка за умовою «порядковий номер більший за останній відомий клієнту» забезпечує, що кількість запитів до сховища лінійно залежить від кількості клієнтів, а не від обсягу пропущених повідомлень. Це принципово для випадку масових одночасних перепідключень після відновлення серверного вузла, коли кожен клієнт повертається в систему одночасно з іншими.

2.3.2 Модель експоненційного відступу

Якщо під час публікації повідомлення виникає тимчасовий збій інфраструктури (наприклад, недоступність сховища історії або затримка при взаємодії із зовнішніми компонентами), система не повертає помилку відправнику, а повторює спробу збереження через певний час. Безпосереднє повторення з фіксованим інтервалом створює ризик лавиноподібного навантаження на

інфраструктуру в момент її відновлення, тому застосовується модель з експоненційним збільшенням інтервалів очікування [18]: кожен наступний інтервал є подвоєнням попереднього.

Ці значення є параметрами налаштування системи та можуть гнучко змінюватися залежно від умов експлуатації. Вибір базових значень (початково 5 секунд, максимум 5 спроб) зумовлений пошуком балансу між необхідністю якомога швидше доставити повідомлення та наданням інфраструктурі достатнього часу на перезапуск після короткочасного збою (наприклад, перезавантаження СУБД чи відновлення контейнера).

За базових налаштувань початковий інтервал очікування перед першою повторною спробою становить 5 секунд; кожен наступний інтервал подвоюється. У результаті послідовність інтервалів між спробами має вигляд $5 \rightarrow 10 \rightarrow 20 \rightarrow 40 \rightarrow 80$ секунд, а сумарний час від моменту першого збою до останньої спроби не перевищує 155 секунд. Такий підхід дає змогу уникнути лавиноподібного перевантаження системи під час масового перепідключення.

Окрім самої моделі повторних спроб, у систему вбудовано механізм запобіжника, що активується в умовах тривалого збою інфраструктури. У стані «запобіжник увімкнено» оброблення черги повторних спроб призупиняється — щоб накопичені завдання не вичерпали ліміт спроб упродовж аварії — а паралельно з періодичністю 10 секунд виконуються перевірки доступності сховища. У момент успішної перевірки запобіжник вимикається, і черга поновлює оброблення накопичених завдань. Такий комбінований підхід гарантує, що система не витратить обчислювальні ресурси на завідомо неуспішні запити до неробочої інфраструктури. Крім того, це запобігає незворотній втраті повідомлень, ліміт повторень яких міг би вичерпатися виключно через тривалість аварійної ситуації, а не через логічні помилки обробки. Загальну схему алгоритму повторних спроб з урахуванням моделі експоненційного відступу та запобіжника наведено на рисунку 2.5.

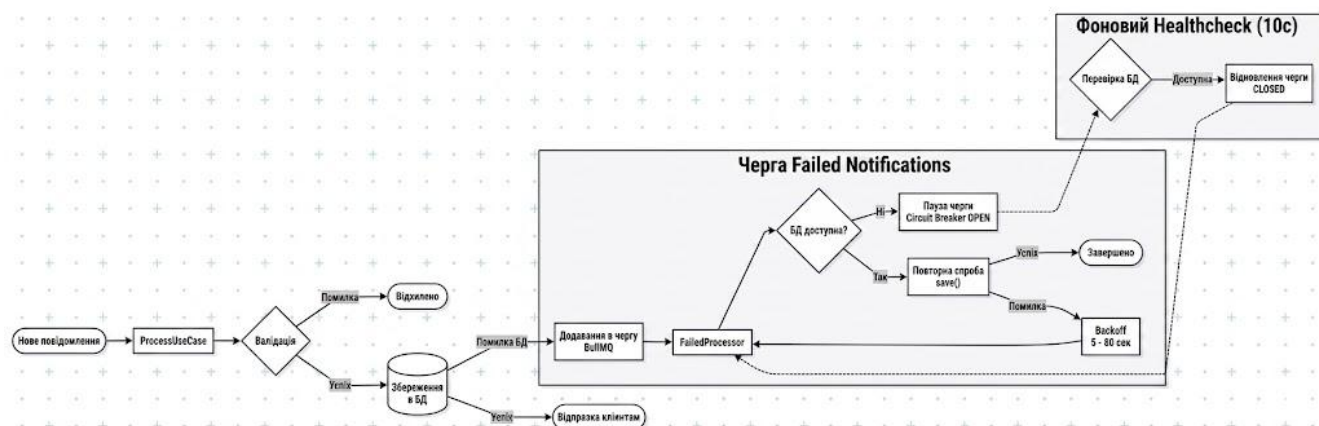


Рисунок 2.5 — Схема алгоритму черги повторних спроб та механізму відмовостійкості

Поєднання експоненційного зростання інтервалів і запобіжника дозволяє розділити дві задачі: модель повторних спроб обмежує сумарну тривалість очікування для окремого повідомлення, а запобіжник запобігає вичерпанню цього ліміту під час тривалих аварій інфраструктури за рахунок тимчасового призупинення оброблення черги.

2.4 Алгоритми розмежування доступу та криптографічного захисту

У підрозділі описано два класи алгоритмів захисту: алгоритм перевірки прав доступу при виконанні операцій над повідомленнями та принципи криптографічного захисту вмісту.

2.4.1 Алгоритм перевірки прав доступу

Розмежування повноважень здійснюється за рольовою моделлю з трьома рівнями. Підписник має право читати канал та відповідати в наявних гілках обговорень. Публікатор, окрім того, має право створювати кореневі повідомлення

для масової розсилки. Адміністратор має повний контроль над каналом, включно з модерацією, видаленням будь-яких повідомлень, зміною конфігурації та керуванням складом учасників.

Принциповою вимогою до алгоритму перевірки прав є інкапсуляція правил безпосередньо в доменній сутності каналу: рішення про дозвіл або заборону приймається на доменному рівні незалежно від джерела запиту, що забезпечує однакову поведінку для запитів, отриманих через різні транспортні канали (сокет-канал, фоновий обробник черги, REST-маршрут). Алгоритм перевірки виконується перед виконанням кожної операції над повідомленням і є частиною прикладного сценарію використання, а не транспортного шару.

Послідовність перевірок при отриманні запиту на операцію над повідомленням наведено на рисунку 2.6.

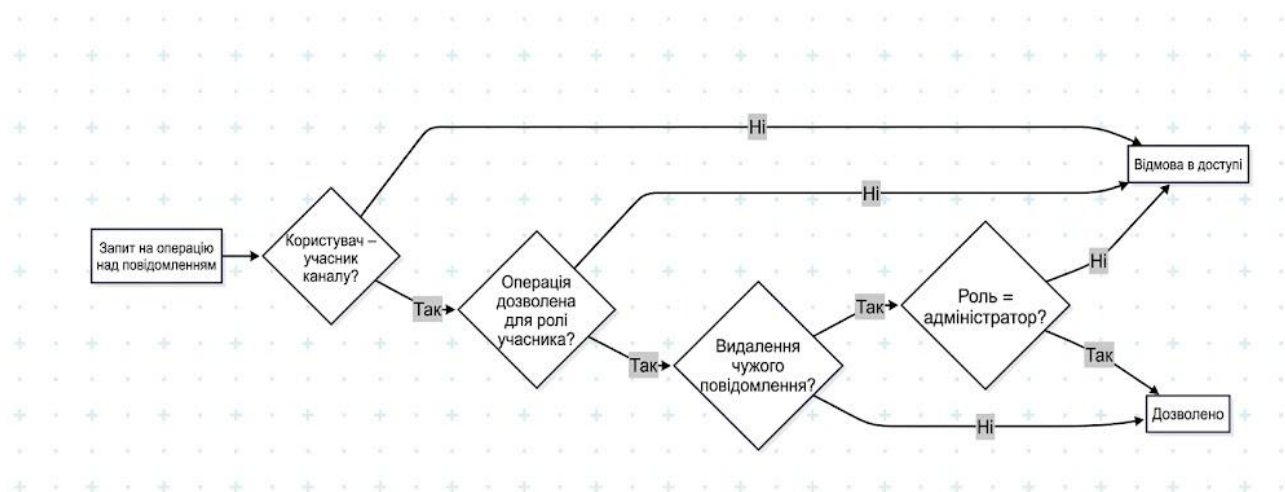


Рисунок 2.6 — Алгоритм перевірки прав доступу при операціях над повідомленням

У разі повернення ознаки «відмова» прикладний сценарій формує помилку, що передається ініціатору запиту через відповідний інфраструктурний адаптер; у разі ознаки «дозволено» — операція виконується далі за загальним алгоритмом обробки повідомлення.

2.4.2 Принципи криптографічного захисту повідомлень

Для забезпечення конфіденційності інформаційних потоків та захисту облікових даних у системі застосовуються чотири незалежні криптографічні механізми, кожен з яких відповідає за окрему загрозу.

Перший — симетричне аутентифіковане шифрування вмісту повідомлень. Воно забезпечує одночасно конфіденційність вмісту та виявлення спроб його підміни, що забезпечує додатковий захист історії переписки від компрометації сховища: навіть у разі несанкціонованого доступу до бази даних зломисник отримує лише шифротекст, що суттєво ускладнює його несанкціоноване прочитання або непомітну модифікацію.

Другий — однонаправлене хешування паролів користувачів адаптивною функцією, стійкою до підбору на спеціалізованому обчислювальному обладнанні. Принципом захисту є те, що з хешу практично неможливо відновити вихідний пароль, а вартість обчислення кожної спроби підбору штучно завищена. Цей механізм нейтралізує загрозу розкриття облікових даних у разі витоку інформації.

Третій — однонаправлене хешування довготривалих маркерів оновлення сесії. Принцип той самий, що й для паролів: з хешу математично вкрай складно відновити оригінальний маркер, тому це значно знижує ризик використання записів з бази даних сторонніми особами для входу від імені користувача.

Четвертий — наскрізне шифрування корисного навантаження фонових сповіщень та автентифікація сервера застосунку перед зовнішнім сервісом доставки. Перший аспект забезпечує захист від прочитання вмісту сповіщень проміжними вузлами на шляху доставки; другий — знижує ймовірність перехоплення або підміни адресатів сторонніми сервісами.

3 ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ РЕАКТИВНОЇ СИСТЕМИ СПОВІЩЕНЬ ІЗ ГАРАНТОВАНОЮ ДОСТАВКОЮ ПОВІДОМЛЕНЬ

Розділ присвячено програмній реалізації методів та алгоритмів, сформульованих у розділі 2. Спочатку наведено функціональну модель системи у вигляді діаграми варіантів використання; далі описано загальну архітектурну структуру програмного комплексу, реляційну модель даних, склад серверних і клієнтських програмних модулів, побудову відмовостійкої черги фонові обробки та програмні модулі підсистеми безпеки. Самі алгоритми (наскрізна нумерація, дедуплікація, синхронізація, експоненційний відступ, перевірка прав) уже формалізовано в розділі 2, тому тут увагу зосереджено на структурі реалізації та обґрунтуванні прийнятих проєктних рішень.

3.1 Функціональна модель системи

Перш ніж переходити до опису внутрішньої структури системи, доцільно зафіксувати її функціональну модель — перелік основних дійових осіб (акторів) та сценаріїв використання, які повинна підтримувати реалізована система. У розробленій моделі виділено дві ключові ролі: ініціатор, який формує інформаційне наповнення сповіщень та керує їх розсилкою, і кінцевий отримувач, чий клієнтський застосунок приймає повідомлення та автоматично генерує сигнали підтвердження. Така абстракція дозволяє чітко окреслити межі програмного комплексу, відокремивши внутрішні процеси маршрутизації та синхронізації від зовнішніх джерел інформації. Загальну функціональну модель у вигляді діаграми варіантів використання наведено на рисунку 3.1.

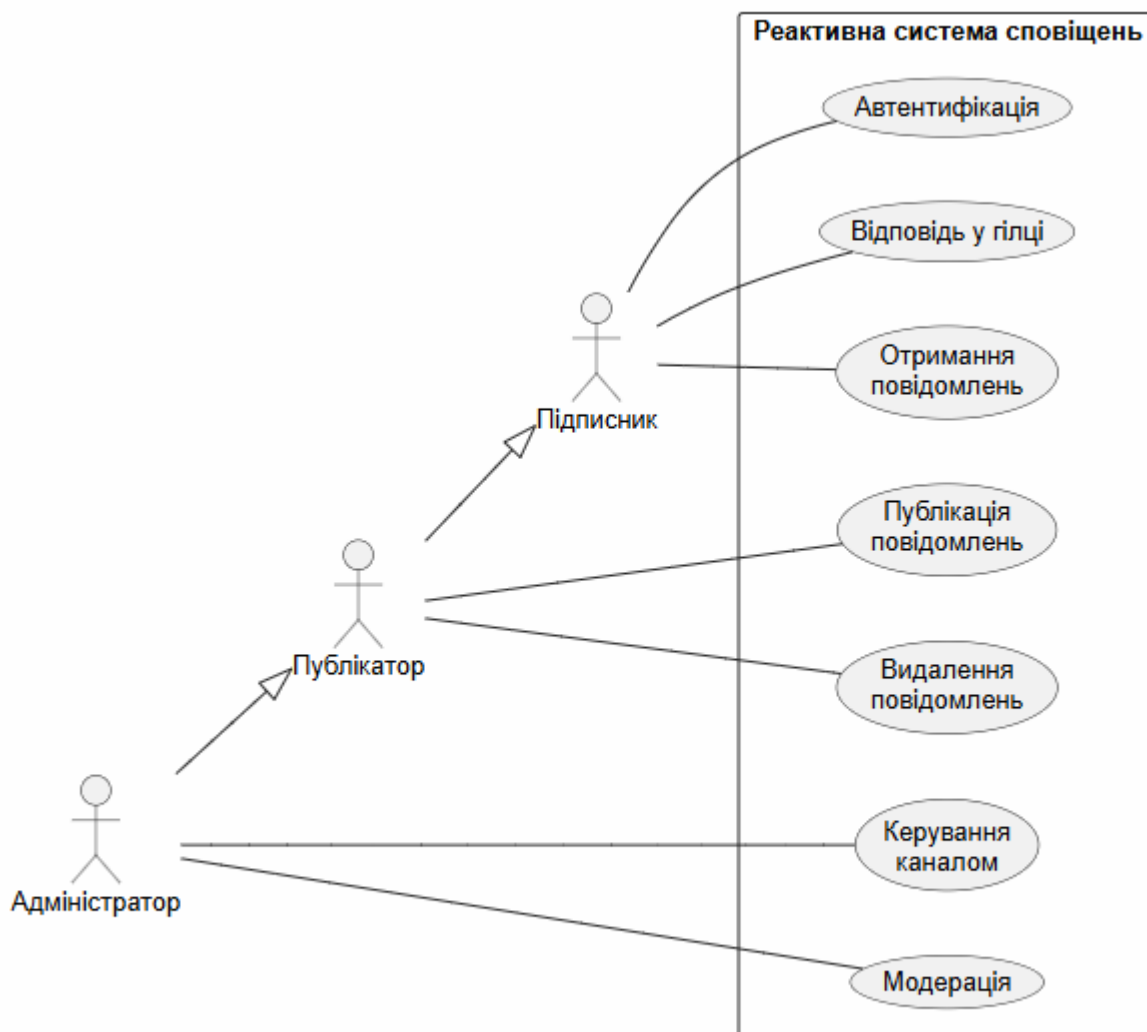


Рисунок 3.1 — Діаграма варіантів використання системи

У моделі визначено трьох акторів, що відповідають ролям користувачів у каналах (підрозділ 2.4.1). Підписник має доступ до базових сценаріїв: автентифікація, перегляд історії каналу, отримання нових сповіщень у реальному часі, отримання фонових сповіщень за відсутньої сесії та відповідь у наявних гілках обговорень. Публікатор додатково отримує право створювати кореневі сповіщення в каналі та видаляти власні повідомлення. Адміністратор додатково має право

створювати й налаштовувати канали, керувати складом учасників, призначати ролі та модерувати будь-які повідомлення в каналі.

Поведінка системи в кожному з варіантів використання визначається алгоритмами розділу 2: для сценаріїв публікації, отримання та синхронізації застосовуються алгоритми наскрізної нумерації й дедуплікації (підрозділ 2.2), для відновлення після розривів — алгоритм пакетної синхронізації (підрозділ 2.3.1), для будь-яких операцій над повідомленнями — алгоритм перевірки прав доступу (підрозділ 2.4.1).

3.2 Архітектурна структура програмного комплексу

Архітектурні принципи побудови системи обґрунтовано в підрозділі 2.1; у цьому підрозділі описано, як саме ці принципи реалізовано в коді розробленого програмного комплексу. Відповідно до концепції чистої архітектури, системні компоненти жорстко розподілено на чотири ізольовані рівні: предметний, прикладний, інфраструктурний та рівень представлення. Ключовим правилом організації взаємодії є суворе дотримання односпрямованості залежностей — виключно від зовнішніх технічних деталей до внутрішнього ядра бізнес-логіки. Зокрема, предметний рівень повністю інкапсулює правила життєвого циклу повідомлень та наскрізної нумерації, тоді як інфраструктурний шар забезпечує фізичну взаємодію з реляційним сховищем і мережевими підключеннями. Такий підхід гарантує, що зміни у механізмах збереження даних чи транспортному середовищі не вимагатимуть перероблення критично важливих алгоритмів гарантованої доставки. Загальну архітектурну модель з відображенням шарів, складу компонентів та напрямку залежностей наведено на рисунку 3.2.

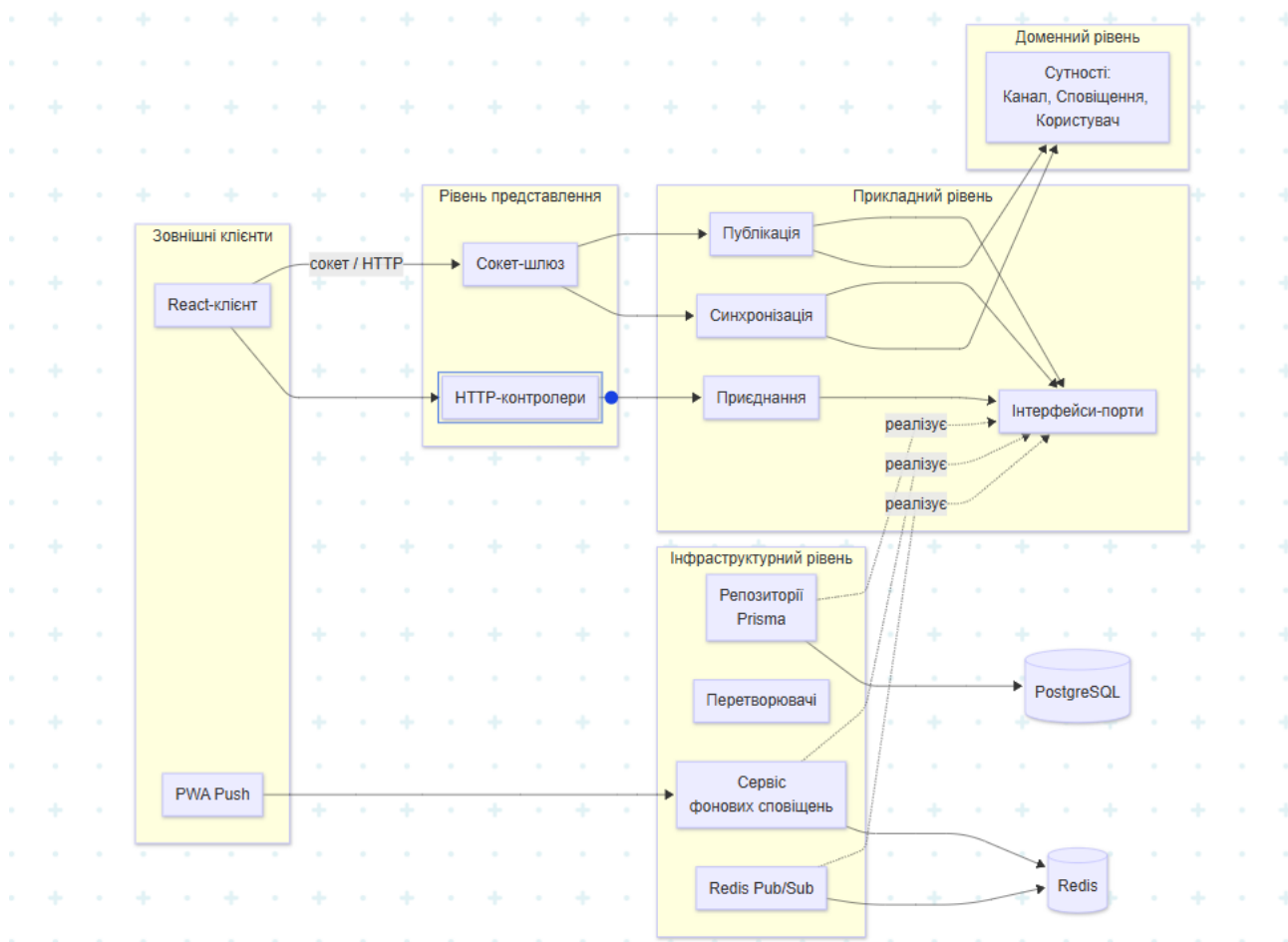


Рисунок 3.2 — Архітектурна модель системи на основі чистої архітектури

Доменний шар реалізовано у вигляді класів-сутностей (Користувач, Інформаційний канал, Сповіщення, Учасник каналу) без зовнішніх залежностей, що дозволяє переносити їх між середовищами виконання без модифікацій. Ключова реалізаційна особливість — інкапсуляція бізнес-правил у методах сутностей: правила публікації та видалення повідомлень зосереджено в методах сутності каналу, що забезпечує однакову поведінку незалежно від джерела запиту.

Прикладний шар містить класи сценаріїв використання, кожен з яких відповідає одному пункту функціональної моделі з підрозділу 3.1. Сценарії взаємодіють із зовнішнім світом виключно через інтерфейси-порти, що дозволяє

запускати один і той самий сценарій через різні транспорти (сокет-шлюз і фонову чергу повторних спроб) без дублювання бізнес-логіки.

Інфраструктурний шар містить реалізації портів: репозиторії бази даних, перетворювачі між таблицями та доменними сутностями, сокет-шлюзи, адаптери до сховища в оперативній пам'яті та модулі безпеки. Такий розподіл локалізує зміни в інфраструктурному шарі (наприклад, при заміні СУБД або транспортного протоколу) та спрощує тестування — у тестах інфраструктурні порти замінюються на заглушки без потреби піднімати реальну базу даних чи сокет-сервер.

3.3 Реляційна модель даних

Реляційну модель даних розроблено таким чином, щоб обмеження цілісності на рівні системи керування базами даних безпосередньо підтримували алгоритми гарантованої доставки. Загальну логічну схему сутностей та зв'язків між ними наведено на рисунку 3.3.

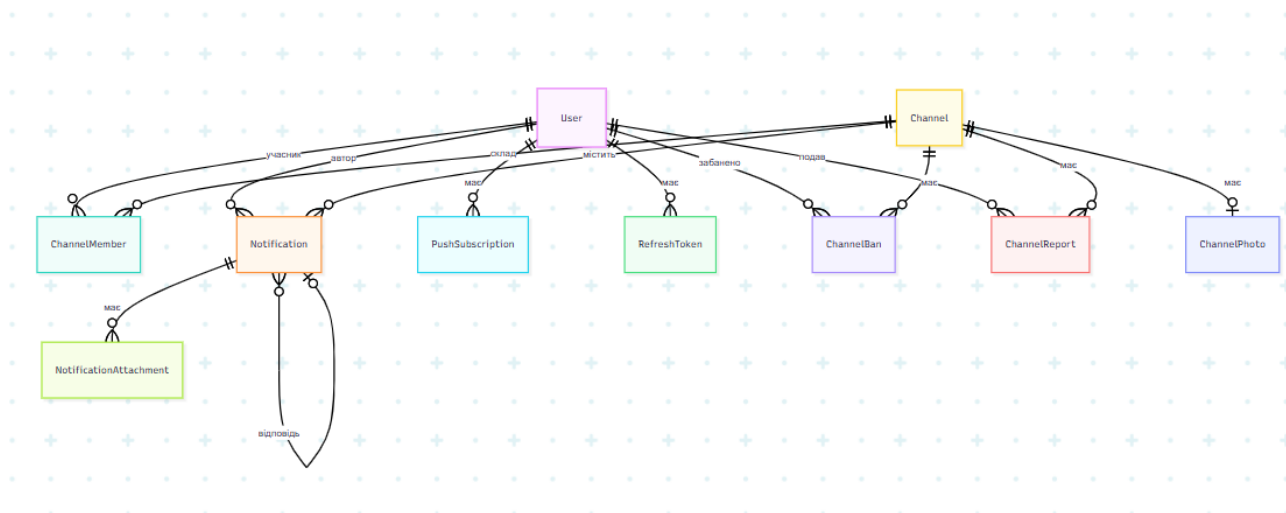


Рисунок 3.3 — ER-діаграма сутностей реляційної бази даних

Як видно з наведеної діаграми, базовим елементом реляційної моделі є сутність «Сповідання». Вона зберігає не лише текстовий вміст, але й набір службових метаданих, які є критично важливими для роботи алгоритмів дедуплікації та пакетної синхронізації при відновленні зв'язку. Детальну структуру та опис полів цієї сутності наведено в таблиці 3.1.

Таблиця 3.1 — Структура сутності Сповідання

Поле	Тип	Обов'язковість	PK / FK / UNIQUE	Призначення
id	UUID	Так	PK	Унікальний ідентифікатор повідомлення
channel_id	UUID	Так	FK → Канал	Канал, у якому опубліковано повідомлення
sender_id	UUID	Так	FK → Користувач	Автор повідомлення
client_id	UUID	Так	UNIQUE у межах channel_id	Клієнтський ідентифікатор для дедуплікації
sequence	INTEGER	Так	UNIQUE у межах channel_id	Наскрізний порядковий номер повідомлення в каналі
parent_id	UUID	Ні	FK → Сповідання	Посилання на батьківське повідомлення для гілок відповідей
text	TEXT	Так	—	Вміст повідомлення (зашифрований за схемою AES-256-GCM)
priority	ENUM	Так	—	Рівень пріоритету повідомлення
created_at	TIMESTAMP	Так	—	Час створення повідомлення

Серед прийнятих проєктних рішень принциповим є одночасне використання трьох ідентифікаторів повідомлення (id, client_id, sequence), кожен з яких розв'язує власну задачу:

id (UUID, первинний ключ) формується на клієнтській стороні до моменту збереження. Це дозволяє клієнту відразу мати ідентифікатор для подальших

операцій (наприклад, для оптимістичного оновлення інтерфейсу) без очікування синхронної відповіді сервера.

`client_id` — унікальний ідентифікатор повідомлення, який клієнт генерує перед першою спробою надсилання й використовує без змін при будь-яких повторних спробах. Накладене на нього обмеження унікальності в межах каналу є основою алгоритму дедуплікації: повторне надходження повідомлення з тим самим `client_id` відхиляється самою системою керування базами даних на рівні атомарної транзакції, що виключає появу дублікатів навіть за умови одночасних повторних спроб.

`sequence` — цілочисельний автоінкрементний номер, монотонність та атомарність генерування якого забезпечуються транзакційним механізмом СУБД. Це поле є основою алгоритмів виявлення пропусків (клієнт може математично порівняти послідовність отриманих номерів) і пакетної синхронізації (одним SQL-запитом за умовою «`sequence > останній відомий клієнту`» вибираються всі пропущені повідомлення).

Сумісне використання трьох ідентифікаторів дозволяє розділити три незалежні задачі моделі даних: глобальну ідентифікацію запису (`id`), детерміновану дедуплікацію повторних надходжень (`client_id`) та впорядкування історії для алгоритмів синхронізації (`sequence`). Жодне з цих полів не дублює функцію іншого: кожен з них має різну природу формування — клієнтський генератор для `id` та `client_id`, серверний транзакційний інкремент для `sequence` — та різні гарантії унікальності. Завдяки цьому розподілу обов'язків відповідні алгоритми гарантованої доставки можуть розвиватися незалежно один від одного, не вимагаючи перепроєктування решти полів моделі.

Для забезпечення багатекористувацької взаємодії та контролю доступу в системі передбачено сутність «Користувач». Вона зберігає облікові дані та слугує точкою прив'язки для визначення авторства повідомлень і перевірки прав доступу. Детальну структуру цієї сутності наведено в таблиці 3.2.

Таблиця 3.2 — Структура сутності Користувач

Поле	Тип	Обов'язковість	PK / FK / UNIQUE	Призначення
id	UUID	Так	PK	Унікальний ідентифікатор користувача
username	VARCHAR	Так	UNIQUE	Унікальне ім'я користувача
email	VARCHAR	Так	UNIQUE	Електронна адреса
password_hash	VARCHAR	Так	—	Хеш пароля з вбудованою сіллю (Argon2)
created_at	TIMESTAMP	Так	—	Дата та час створення облікового запису

Комунікація між користувачами організовується за допомогою ізольованих просторів — інформаційних каналів. Сутність «Інформаційний канал» визначає глобальні налаштування доступу та вказує на необхідність шифрування для певної групи повідомлень. Її структуру наведено в таблиці 3.3.

Таблиця 3.3 — Структура сутності Інформаційний канал

Поле	Тип	Обов'язковість	PK / FK / UNIQUE	Призначення
id	UUID	Так	PK	Унікальний ідентифікатор каналу
slug	VARCHAR	Так	UNIQUE	Текстовий ідентифікатор каналу
name	VARCHAR	Так	—	Відображувана назва
is_private	BOOLEAN	Так	—	Ознака приватності каналу
is_encrypted	BOOLEAN	Так	—	Ознака шифрування вмісту повідомлень
created_at	TIMESTAMP	Так	—	Дата створення

Зв'язок між конкретними користувачами та каналами реалізується через сполучну сутність «Учасник каналу». Її структуру наведено в таблиці 3.4.

Винесення поля `last_read_sequence` саме у цю сполучну сутність (а не в окрему таблицю індивідуальних статусів) є прямим наслідком алгоритму динамічного обчислення статусу прочитання. Для встановлення факту ознайомлення з повідомленням достатньо порівняти це поле з порядковим номером повідомлення, що суттєво знижує обсяг збережених даних.

Таблиця 3.4 — Структура сутності Учасник каналу

Поле	Тип	Обов'язковість	PK / FK / UNIQUE	Призначення
<code>id</code>	UUID	Так	PK	Унікальний ідентифікатор запису учасника
<code>channel_id</code>	UUID	Так	FK → Канал, UNIQUE з <code>user_id</code>	Канал, до якого належить учасник
<code>user_id</code>	UUID	Так	FK → Користувач, UNIQUE з <code>channel_id</code>	Користувач, що є учасником
<code>role</code>	ENUM	Так	—	Роль учасника в каналі (Адміністратор / Публікатор / Підписник)
<code>last_read_sequence</code>	INTEGER	Так	—	Показчик останнього прочитаного повідомлення в каналі
<code>joined_at</code>	TIMESTAMP	Так	—	Час приєднання до каналу

Для підтримки передачі мультимедійних даних або документів передбачено сутність «Вкладення». Вона зберігає посилання на файли, що фізично розміщені у зовнішньому об'єктному сховищі, та пов'язує їх із конкретним повідомленням. Структуру цієї сутності наведено в таблиці 3.5.

Таблиця 3.5 — Структура сутності Вкладення

Поле	Тип	Обов'язковість	PK / FK / UNIQUE	Призначення
id	UUID	Так	PK	Унікальний ідентифікатор вкладення
notification_id	UUID	Так	FK → Сповіщення	Повідомлення, до якого прикріплено вкладення
url	TEXT	Так	—	Посилання на об'єкт у зовнішньому об'єктному сховищі
created_at	TIMESTAMP	Так	—	Час прикріплення

Цілісність моделі підтримується обмеженнями зовнішніх ключів із каскадним видаленням: при видаленні каналу автоматично видаляються його учасники, історія повідомлень та вкладення; при видаленні повідомлення — його вкладення. Це запобігає появі «осиротілих» записів та порушенню логіки пакетної синхронізації. Окрім каскадного видалення, цілісність змін у межах окремої публікації забезпечується транзакційним механізмом СУБД: вставка повідомлення, генерування порядкового номера та перевірка унікальності клієнтського ідентифікатора виконуються як одна атомарна транзакція, що унеможливорює появу часткових станів моделі при паралельних запитах.

3.4 Структура серверних програмних модулів

Серверну частину реалізовано як сукупність модулів, розподілених за чотирма шарами архітектури: транспортним, прикладним, доменним та інфраструктурним. Структуру модулів та напрямок залежностей між ними наведено на рисунку 3.4.

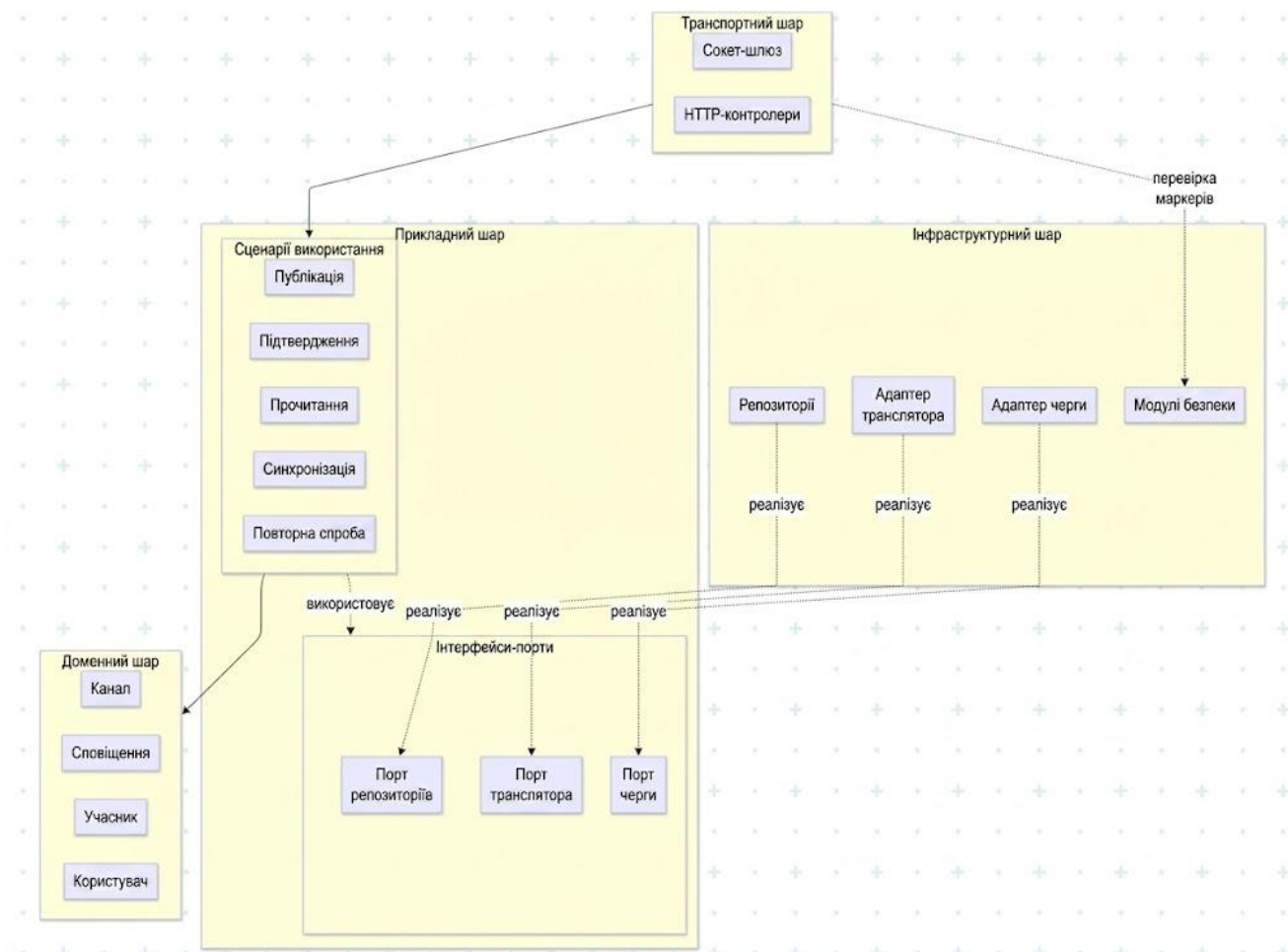


Рисунок 3.4 — Діаграма компонентів серверної частини

Розподіл відповідальності між шарами реалізовано таким чином, щоб одну й ту саму бізнес-операцію можна було ініціювати з будь-якого транспорту без зміни бізнес-логіки. Транспортний шар лише виконує валідацію вхідних об'єктів та делегує управління відповідному сценарію використання; прикладний шар оркеструє виклики до доменних сутностей через інтерфейси-порти; доменний шар містить правила цілісності; інфраструктурні адаптери реалізують конкретну взаємодію з базою даних та зовнішніми системами. Ключове проєктне рішення — інкапсуляція правил перевірки прав безпосередньо в методах доменної сутності каналу: завдяки цьому сценарій публікації, викликаний як зі сокет-шлюзу (за

прямою дією користувача), так і з фонові черги повторних спроб (без активного користувача), застосовує ті самі правила без дублювання коду.

Послідовність взаємодії модулів під час типового сценарію обробки нового сповіщення наведено на рисунку 3.5.

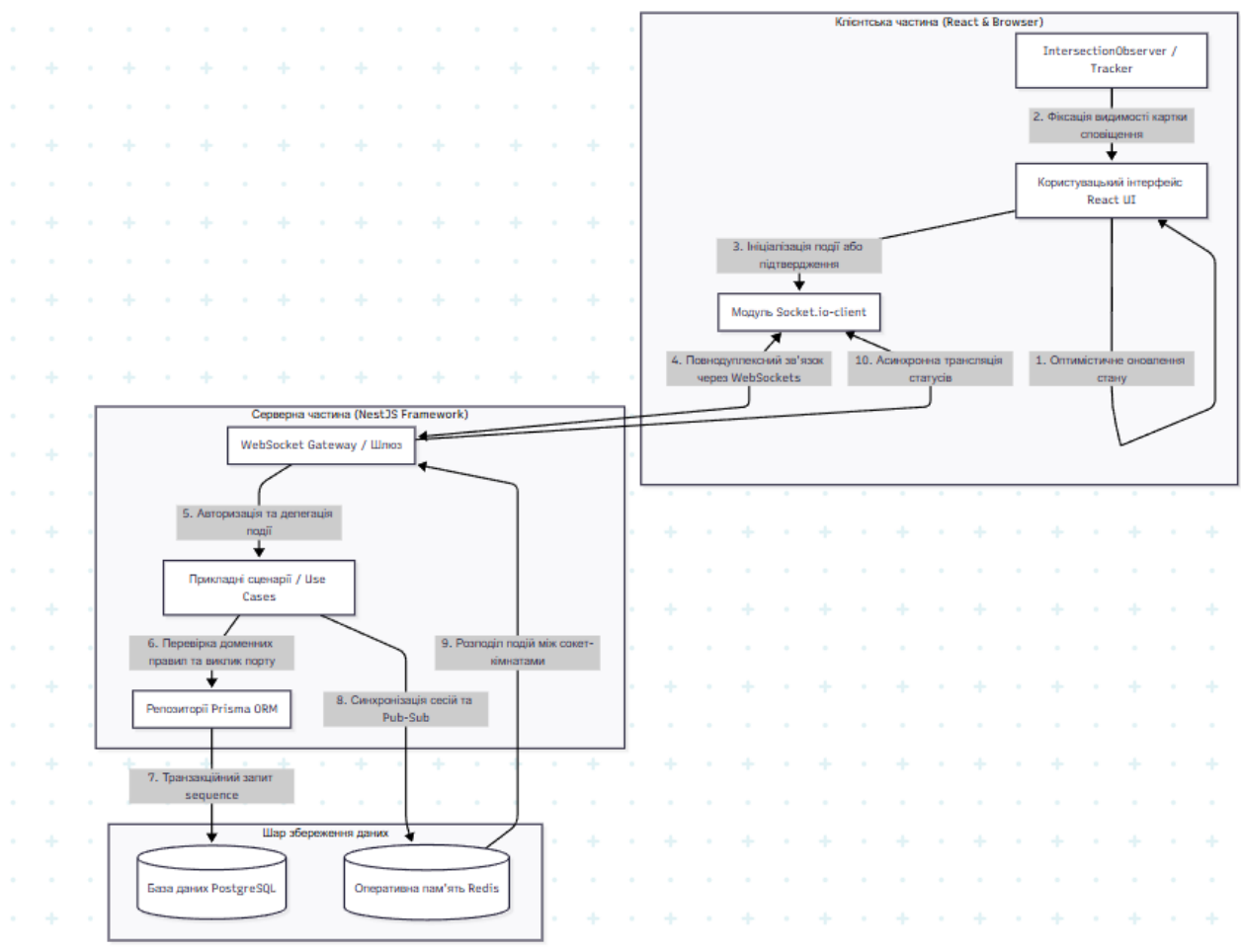


Рисунок 3.5 — Схема взаємодії програмних модулів та сокет-шлюзів

Така послідовність обрана з огляду на дві вимоги: мінімізувати кількість викликів до бази даних на один запит (одна перевірка прав, одне збереження, одна трансляція) та забезпечити природне місце для роботи дедуплікації — атомарне збереження з обмеженням унікальності клієнтського ідентифікатора, яке здійснює сам адаптер репозиторію.

3.5 Структура клієнтських програмних модулів

Клієнтську частину реалізовано як одно-сторінковий вебзастосунок з чотиришаровою структурою: візуальні компоненти інтерфейсу, шар керування станом, шар сервісів взаємодії із сервером та утилітарні модулі. Структуру модулів наведено на рисунку 3.6.

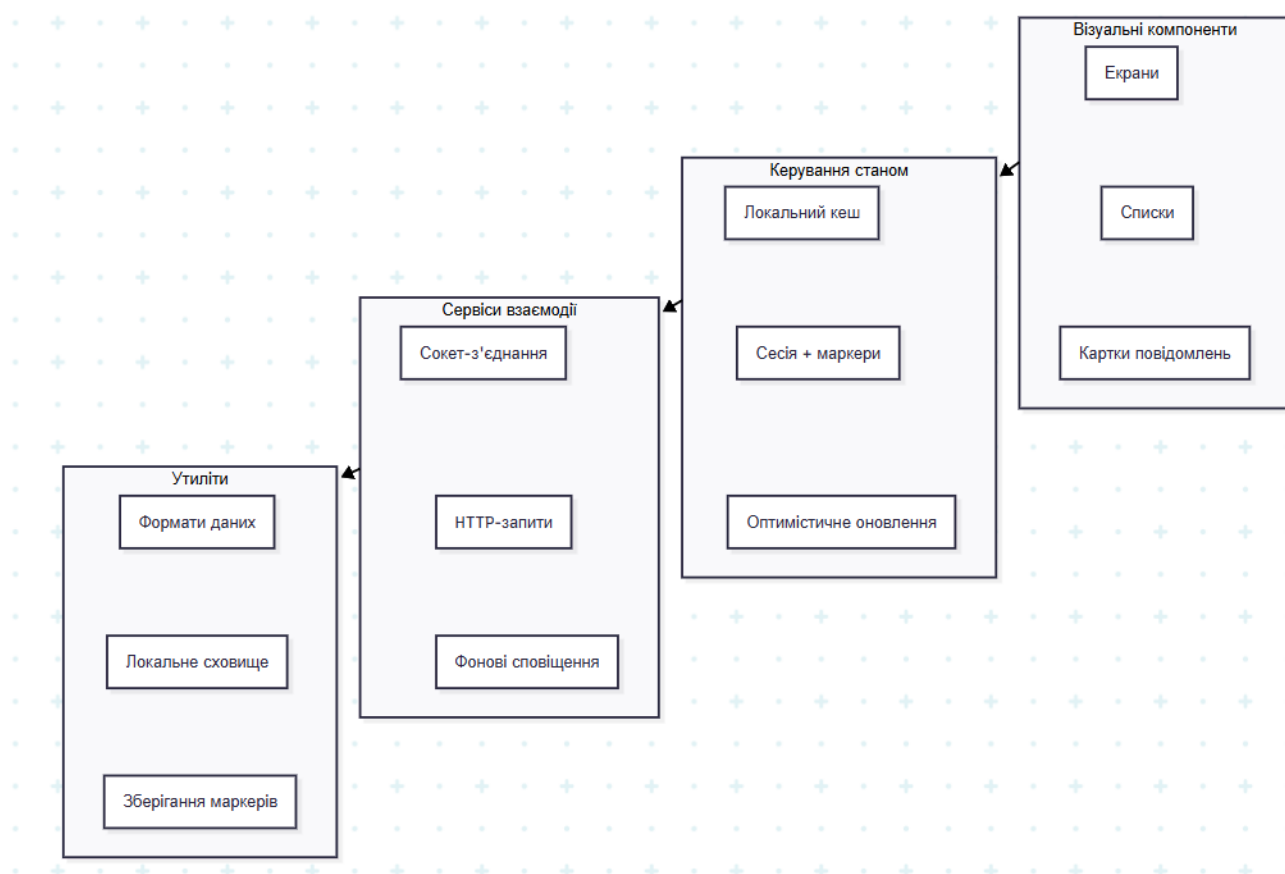


Рисунок 3.6 — Структура програмних модулів клієнтської частини

Прийняте проєктне рішення — реалізація шару керування станом як єдиного централізованого сховища (а не локального стану на рівні візуальних компонентів) — обґрунтоване специфікою задачі: повідомлення в межах одного каналу можуть одночасно відображатися на кількох екранах (списки, гілки, push-сповіщення), а статус прочитання змінюється глобально для всіх учасників. Розподілений

локальний стан у таких умовах призвів би до неузгодженостей. Централізоване сховище також природно реалізує патерн оптимістичного оновлення: тимчасовий запис додається в кеш одразу, а після відповіді сервера замінюється на постійний без додаткової логіки в інтерфейсі.

Шар сервісів взаємодії із сервером відокремлено від шару керування станом для того, щоб виклики до сервера могли бути перехоплені та змінені (наприклад, для прозорого виконання ротації маркерів автентифікації при отриманні помилки авторизації) без зміни коду візуальних компонентів. Шар утилітарних модулів містить функції, не специфічні для жодного зі сценаріїв: перетворення форматів обміну, локальне зберігання сесійного стану в захищеному вигляді та допоміжні засоби форматування.

Описана чотиришарова структура клієнтського застосунку повторює архітектурну модель серверної частини, наведену в підрозділі 3.2, що забезпечує єдиний підхід до проєктування обох сторін системи. Така узгодженість дозволяє ізольовано тестувати кожен шар: візуальні компоненти можна перевіряти на штучних вхідних даних, шар керування станом — без активного мережевого з'єднання, а сервіси взаємодії — на тестових заглушках серверної частини. Узгодженість архітектурних моделей сторін також забезпечує єдину семантику обміну даними: формат корисного навантаження, структура службових повідомлень та інтерпретація ідентифікаторів повідомлень однакові з обох боків каналу зв'язку, що зменшує ймовірність помилок інтеграції. Збереження чіткого розподілу відповідальності між шарами клієнтської частини спрощує її подальший супровід та внесення змін у візуальне оформлення без модифікації бізнес-логіки.

3.6 Реалізація відмовостійкої черги фонові обробки

Чергу повторних спроб реалізовано як набір ізольованих програмних модулів, що працюють поверх сховища в оперативній пам'яті. Структуру компонентів та їх взаємодію наведено на рисунку 3.7.

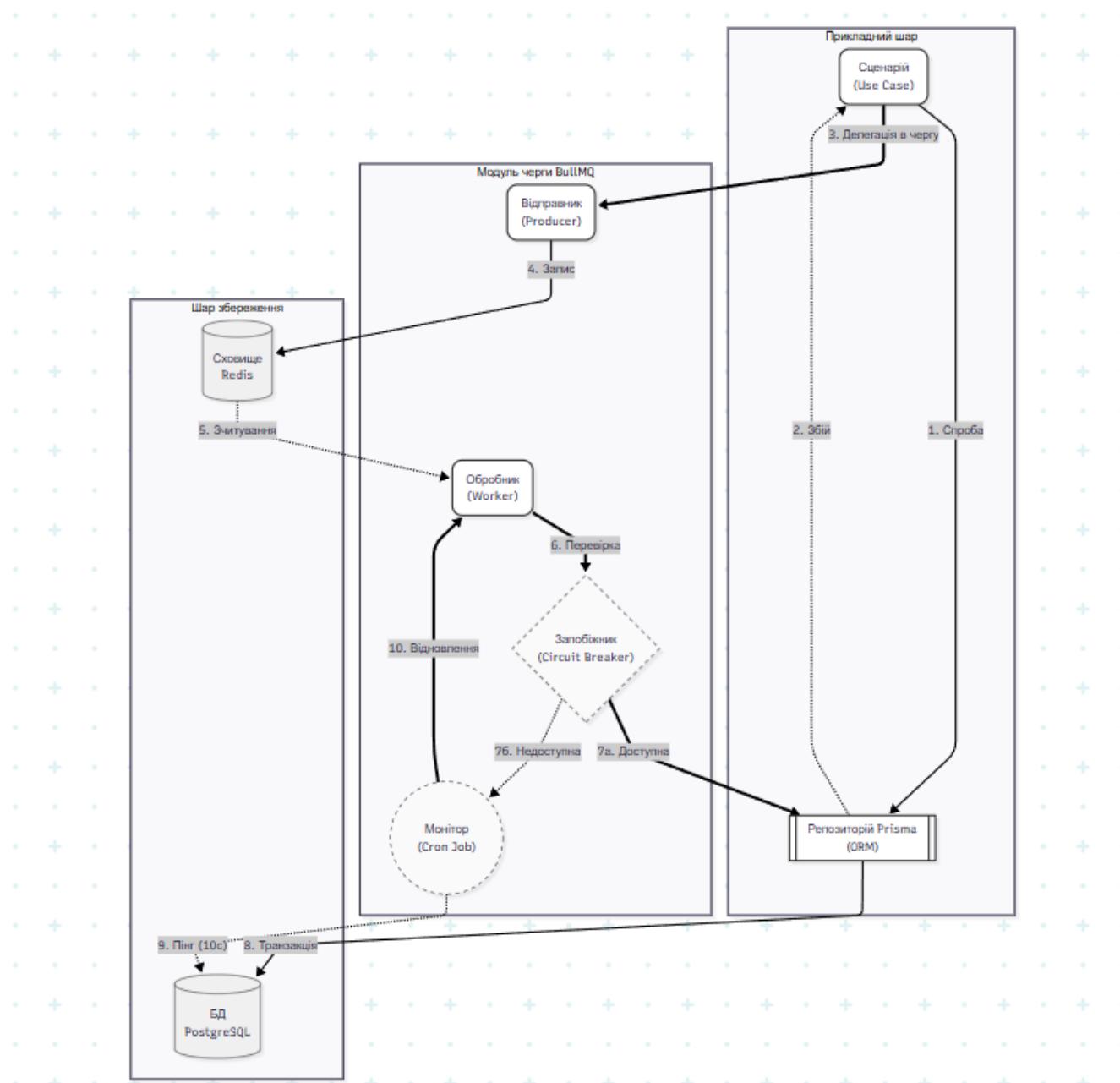


Рисунок 3.7 — Схема архітектури та взаємодії компонентів фонові черги

Модуль виробника завдань інтегровано безпосередньо в прикладний сценарій публікації сповіщення. При виникненні інфраструктурної помилки під час збереження повідомлення перехоплювач помилок передає метадані повідомлення модулю виробника, який формує об'єкт завдання та поміщає його в чергу. Модуль обробника завдань працює в автономному режимі: безперервно опитує сховище черги, отримує наступне завдання та викликає той самий прикладний сценарій збереження, що і первинна спроба. Модуль запобіжника функціонує паралельно з обробником: при виявленні недоступності сховища призупиняє оброблення черги, а паралельно запускається модуль періодичної перевірки доступності.

Параметри роботи черги задано на основі наступних міркувань. Максимальна кількість повторних спроб встановлена на рівні 5 — це достатньо для подолання типових короточасних інфраструктурних збоїв (тайм-ауту бази даних, мережевої затримки, перезапуску компонента), не призводячи водночас до надмірного накопичення невідправлених завдань у разі тривалої аварії. Початковий інтервал очікування 5 секунд є компромісом між швидкістю відновлення для відправника та запасом часу для відновлення інфраструктури: менші значення спричиняли б повторне навантаження на компонент, що ще не відновився, більші — затримували б відправлення в типових сценаріях короточасних збоїв. Стратегія експоненційного зростання інтервалів зі множителем 2 ($5 \rightarrow 10 \rightarrow 20 \rightarrow 40 \rightarrow 80$ с) відповідає стандартній рекомендації для уникнення лавиноподібного навантаження після відновлення інфраструктури. Інтервал перевірки доступності бази даних запобіжником 10 секунд обрано з огляду на те, щоб після успішного відновлення сховища система відновлювала роботу з мінімальною додатковою затримкою, не створюючи водночас надмірного навантаження легковажними тестовими запитамі в період аварії. Усі вказані значення є параметрами налаштування системи та можуть змінюватися у файлі конфігурації відповідно до умов експлуатації.

Модуль моніторингу подій черги фіксує системні сигнали про успішне завершення завдань, помилки обробки та вичерпання ліміту повторних спроб, що використовується для ведення журналу аудиту й подальшого аналізу збоїв.

3.7 Програмні модулі підсистеми безпеки

Підсистему безпеки реалізовано як сукупність модулів, що покривають дві ортогональні задачі: захист звичайних HTTP-маршрутів і захист повнодуплексних сокет-з'єднань. Обидві задачі ґрунтуються на спільній моделі автентифікації за допомогою криптографічно підписаних маркерів доступу та спільній рольовій моделі розмежування прав. Структуру рольової моделі наведено на рисунку 3.8.

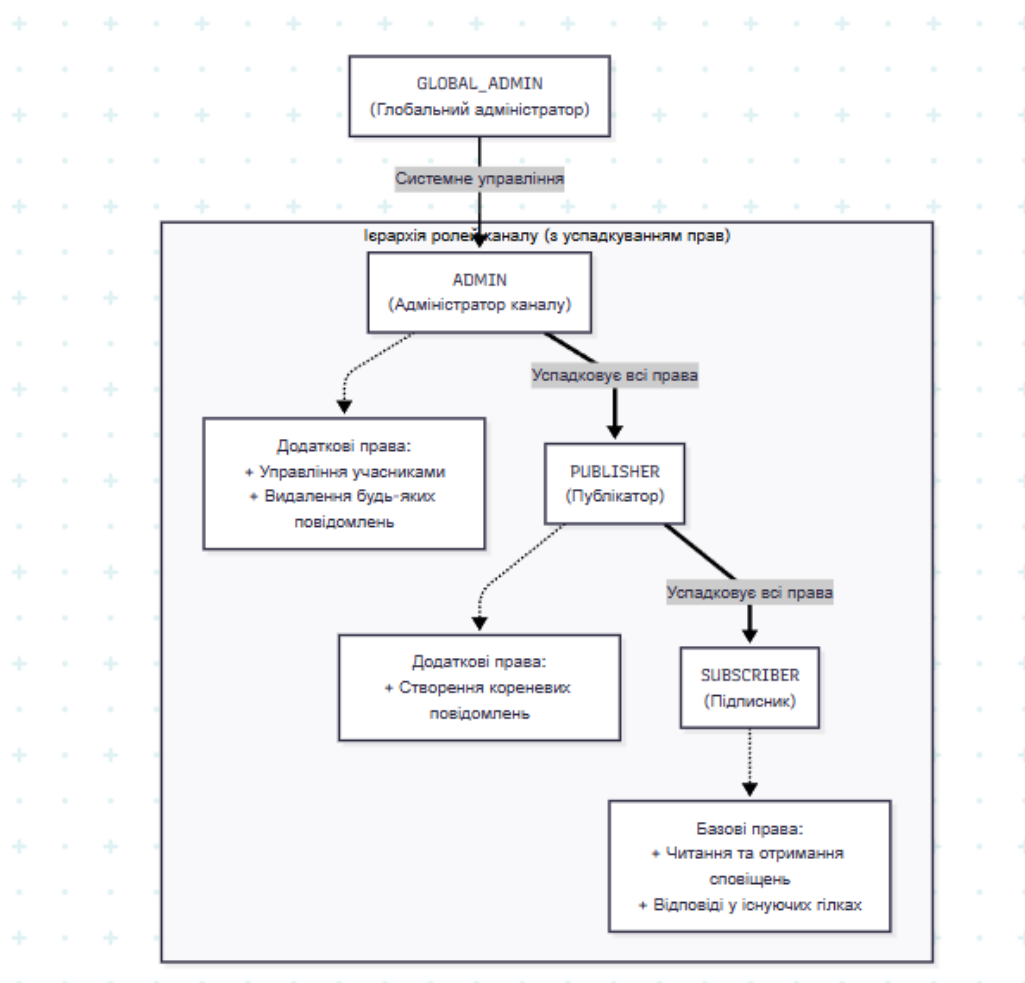


Рисунок 3.8 — Рольова модель доступу до інформаційних каналів

Структуру рольових записів у моделі даних реалізовано у сутності Учасник каналу через перерахування з трьома значеннями (Адміністратор, Публікатор,

Підписник). Перевірка прав, формалізована як алгоритм у підрозділі 2.4.1, інкапсульована безпосередньо в методах доменної сутності каналу, тому одна й та сама перевірка виконується незалежно від джерела запиту — і з HTTP-контролера, і з сокет-шлюзу, і з обробника фонові черги.

3.7.1 Програмні модулі авторизації HTTP-запитів

Для захисту прикладних кінцевих точок серверної частини реалізовано бездержавну схему автентифікації на основі криптографічно підписаних маркерів доступу [19]. Підсистема складається з трьох програмних модулів. Модуль видавання маркерів реалізовано як прикладний сценарій, що при успішній перевірці облікових даних формує пару маркерів. Перевірка пароля користувача під час входу делегована окремому модулю верифікації облікових даних, який працює на основі алгоритму Argon2. Відкритий пароль у системі не зберігається: у моделі даних користувача присутнє лише поле хешу з вбудованою сіллю, що суттєво ускладнює відновлення пароля навіть у разі компрометації сховища. Аналогічно тривалий маркер оновлення сесії зберігається в моделі даних не в оригінальному вигляді, а як хеш SHA-256: оригінальний маркер відомий лише клієнту, тому відновлення його з вмісту бази у разі несанкціонованого доступу є вкрай ускладненим. Сформована пара маркерів складається з короткоживучого маркера доступу (час життя 15 хвилин, містить ідентифікатор користувача та глобальну роль) і тривалого маркера оновлення (час життя 7 діб, зберігається у вигляді хешу SHA-256 у базі даних і у вигляді захищеного куки-файлу на клієнті — таке зберігання суттєво ускладнює використання записів з бази для входу від імені користувача, оскільки відновлення оригінального маркера з його хешу є практично нездійсненним). Модуль перехоплення запитів реалізовано як стандартний серверний охоронець фреймворку, який перевіряє підпис маркера доступу без звернення до бази даних. Модуль ротації маркерів реалізовано як окремий контролер, що приймає маркер оновлення з куки-файлу, порівнює його хеш з тим,

що збережений у базі даних, і у разі збігу видає нову пару маркерів, оновлюючи запис у базі (одноразова ротація).

Послідовність кроків процесу авторизації при HTTP-запиті, включно з прозорою ротацією маркерів після завершення терміну дії короткоживучого маркера, наведено на рисунку 3.9.

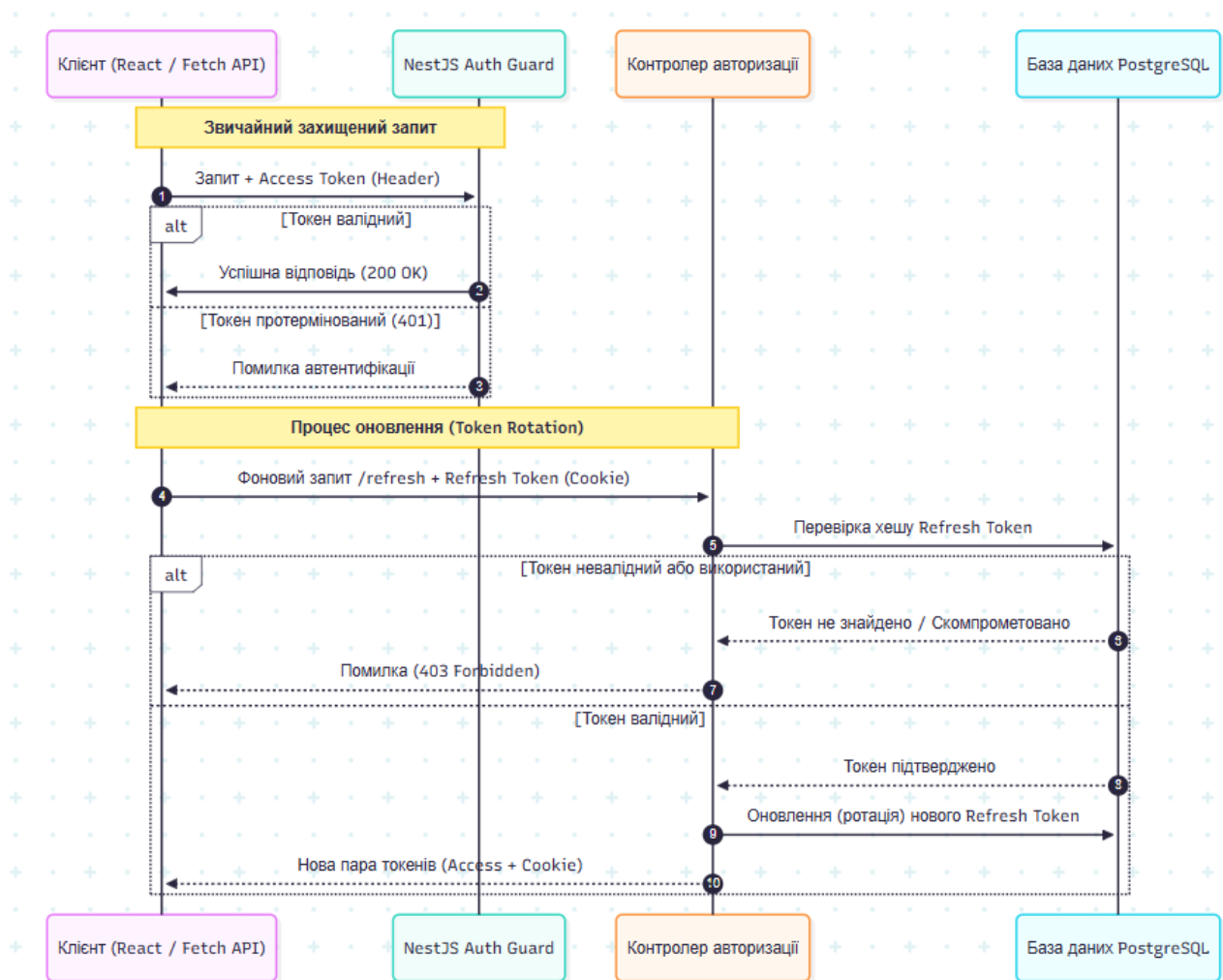


Рисунок 3.9 — Діаграма послідовності процесу HTTP-авторизації користувача

Клієнтський модуль захисту маркерів зберігає маркер оновлення лише у вигляді захищеного куки-файлу з прапорами обмеження доступу, що суттєво ускладнює його зчитування з JavaScript-середовища. Короткоживучий маркер

доступу зберігається в оперативній пам'яті клієнтського застосунку та автоматично прикріплюється до кожного вихідного запиту в заголовку авторизації. Така структура мінімізує кількість операцій введення-виведення на сервері: основний потік запитів проходить через швидку криптографічну перевірку без звернення до бази даних, а важкі запити до реляційного сховища відбуваються лише під час оновлення сесії.

3.7.2 Програмні модулі авторизації сокет-з'єднань

Захист сокет-з'єднань відрізняється від HTTP-шару тим, що з'єднання встановлюється один раз, а далі підтримується тривалий час; стандартні проміжні перевірки на кожен пакет недоступні. Для розв'язання цієї проблеми реалізовано двоетапну схему авторизації як два окремі програмні модулі.

Перший модуль — обробник етапу встановлення з'єднання — отримує контроль у момент ініціювання сокета. Маркер доступу передається клієнтом у спеціалізованому об'єкті конфігурації авторизації під час рукостискання. Обробник передає маркер сервісу криптографічного аналізу; у разі негативного результату з'єднання примусово розривається, у разі позитивного — об'єкт користувача з ідентифікатором та ролями прикріплюється до контексту сокет-сесії, утворюючи ізольований контекст безпеки на час життя з'єднання.

Другий модуль — перехоплювач сокет-подій — отримує контроль при кожній спробі взаємодії клієнта з конкретним каналом (приєднання до кімнати каналу, публікація сповіщення). Модуль витягує ідентифікатор користувача з контексту сесії та цільовий ідентифікатор каналу з пакета події, виконує асинхронний запит до репозиторію для перевірки членства й, у разі позитивного результату, передає роль учасника доменній сутності каналу для перевірки прав на конкретну операцію. У разі відмови клієнту повертається сокет-помилка про відсутність прав; у разі дозволу — операція виконується далі за алгоритмом, описаним у підрозділі 2.4.1.

Загальну схему алгоритму перевірки маркерів та кімнатної авторизації на рівні сокет-шлюзу наведено на рисунку 3.10.

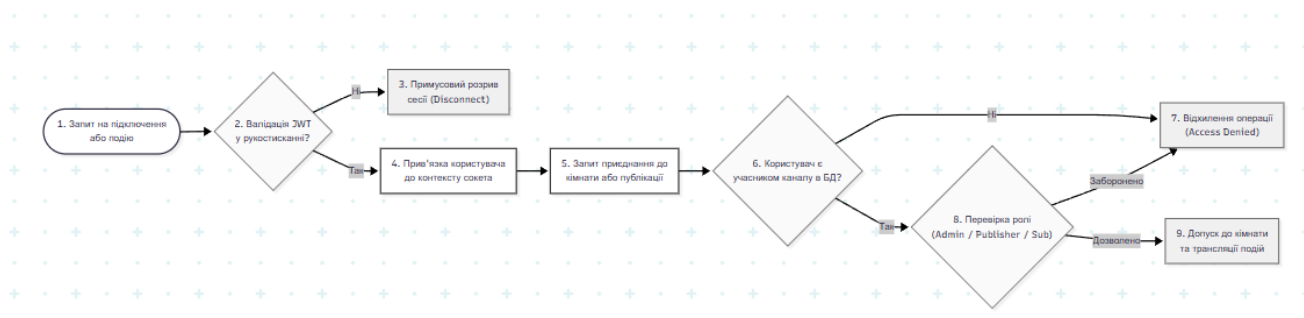


Рисунок 3.10 — Схема алгоритму авторизації та розмежування прав у сокет-шлюзі

Поєднання модулів автентифікації на етапі рукописання та перехоплювача сокет-подій забезпечує захист від несанкціонованих з'єднань і несанкціонованого доступу до окремих каналів для обох транспортів (HTTP і сокет-з'єднання) при суттєвій відмінності в їхній моделі життєвого циклу.

3.7.3 Програмні модулі криптографічного захисту вмісту повідомлень

Криптографічний захист на серверній стороні реалізовано двома ізольованими модулями: модулем симетричного шифрування вмісту повідомлень та модулем автентифікації фонового каналу доставки сповіщень. Обидва модулі прозорі для прикладних сценаріїв — жодного коду шифрування не міститься у класах сценаріїв або доменних сутностях.

Модуль симетричного шифрування інтегровано в інфраструктурні адаптери репозиторіїв повідомлень та каналів. Шифрування виконується перед збереженням запису в реляційному сховищі, розшифрування — під час вибірки. Застосовано алгоритм аутентифікованого шифрування AES-256-GCM, який забезпечує захист від несанкціонованого читання вмісту та виявлення спроб його підміни. Симетричний ключ формується при запуску застосунку через функцію

криптографічної деривації на основі параметрів зі змінних середовища й на клієнт не передається. Для кожного запису генерується випадковий вектор ініціалізації, а зашифрований результат разом із тегом автентифікації зберігається в полі текстового типу у форматі з шістнадцятковим кодуванням.

Модуль доставки фонових сповіщень забезпечує захист на двох рівнях: шифрування корисного навантаження для конкретного одержувача та автентифікація сервера застосунку перед зовнішнім сервісом доставки. Корисне навантаження кожного сповіщення шифрується безпосередньо для одержувача за специфікацією RFC 8291 на основі двох параметрів його підписки — публічного ключа еліптичної кривої P-256 (p256dh) та секрету автентифікації (auth). Це забезпечує захист від прочитання вмісту проміжними вузлами на шляху доставки. Автентифікація сервера перед сервісом доставки виконується за стандартом ідентифікації прикладного сервера: пара ключів VAPID (закритий ключ сервера та відомий сервісу доставки відкритий ключ) використовується для підпису запитів, що дозволяє сервісу доставки приймати лише запити, відправлені легітимним сервером застосунку. Дані кожної підписки (адреса призначення, p256dh, auth) зберігаються з прив'язкою до конкретного авторизованого користувача, що суттєво ускладнює перехоплення або підміну адресатів сторонніми сервісами.

У сукупності описані модулі підсистеми безпеки забезпечують захист системи на трьох незалежних рівнях: автентифікації користувачів та керування сесіями, розмежування доступу до окремих інформаційних каналів через сокет-транспорт і криптографічного захисту вмісту повідомлень як у сховищі, так і під час доставки через зовнішні сервіси. Ізоляція цих модулів один від одного локалізує будь-які зміни в межах відповідної підсистеми: оновлення алгоритму шифрування не зачіпає логіку авторизації, а зміна моделі сесій не впливає на криптографію вмісту. Усі модулі підсистеми безпеки інтегровані в загальну архітектурну модель комплексу через шар інфраструктурних адаптерів, що дозволяє підставляти у прикладні сценарії як виробничі, так і тестові реалізації без модифікації бізнес-логіки.

4 РОЗГОРТАННЯ, ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ РЕАКТИВНОЇ СИСТЕМИ СПОВІЩЕНЬ ІЗ ГАРАНТОВАНОЮ ДОСТАВКОЮ ПОВІДОМЛЕНЬ

Розділ присвячено практичному впровадженню розробленого програмного комплексу та його експериментальній перевірці. Спочатку описано вимоги до апаратного й програмного забезпечення, на якому може функціонувати система, та її контейнерну архітектуру розгортання. Далі наведено демонстрацію реалізованих сценаріїв роботи системи, що підтверджує виконання функціональних вимог, сформульованих у розділі 1. Окремий підрозділ присвячено методиці та результатам тестування алгоритмів гарантованої доставки, описаних у розділі 2, в умовах змодельованих мережевих та інфраструктурних збоїв.

4.1 Розгортання програмного комплексу

4.1.1 Системні вимоги

Програмний комплекс реалізовано як набір окремих сервісів, що працюють у контейнеризованому середовищі, тому ключовою умовою його розгортання є наявність на цільовому вузлі засобів контейнеризації. Серверна частина потребує помітніших ресурсів, оскільки на ній лежить навантаження від підтримки повнодуплексних з'єднань, транзакційного збереження повідомлень та обробки фонові черги повторних спроб.

Мінімальна конфігурація серверного вузла, на якій система коректно функціонує під помірним навантаженням, передбачає двоядерний процесор, чотири гігабайти оперативної пам'яті, двадцять гігабайтів вільного дискового простору, операційну систему сімейства Linux із довгостроковою підтримкою та встановлені засоби контейнеризації. Найбільш чутливою до обсягу оперативної

пам'яті є комбінація реляційної бази даних, сховища в оперативній пам'яті та середовища виконання серверного коду, які функціонують одночасно; саме тому обсяг чотирьох гігабайтів зазначено як нижню межу, а для виробничих навантажень рекомендована конфігурація з чотириядерним процесором та восьми гігабайтами оперативної пам'яті.

Клієнтська частина не висуває специфічних вимог до обчислювальних ресурсів. Достатньо сучасного веббраузера з підтримкою повнодуплексного протоколу зв'язку та механізмів фонових повідомлень, а також стабільного підключення до мережі для первинного завантаження застосунку. Подальша робота можлива за умови тимчасових розривів зв'язку завдяки реалізованим у системі алгоритмам синхронізації пропущених повідомлень (підрозділ 2.3.1).

Зазначені системні вимоги задовольняються більшістю сучасних серверних платформ — як на власному обладнанні, так і у хмарних середовищах, — що дозволяє організації вибирати варіант розгортання залежно від наявних ресурсів та політики безпеки. Завдяки контейнеризованому виконанню компонентів перехід між різними інфраструктурними середовищами (локальне розгортання, приватна або публічна хмара) не потребує модифікації вихідного коду застосунку: змінюється лише конфігурація запуску контейнерів. Така портативність розгортання спрощує також підготовку тестових та стейджингових середовищ, ідентичних виробничому, що знижує ризик розбіжностей у поведінці системи при переході від розробки до експлуатації.

4.1.2 Архітектура розгортання

Для усунення проблем сумісності залежностей та забезпечення відтворюваності розгортання обрано архітектуру на основі засобів контейнеризації. Кожен логічний компонент системи виконується в ізольованому контейнері з власним набором системних бібліотек, що дозволяє переносити комплекс між середовищами без зміни вихідного коду. Загальну схему контейнерного

розгортання з відображенням компонентів та інформаційних потоків між ними наведено на рисунку 4.1.

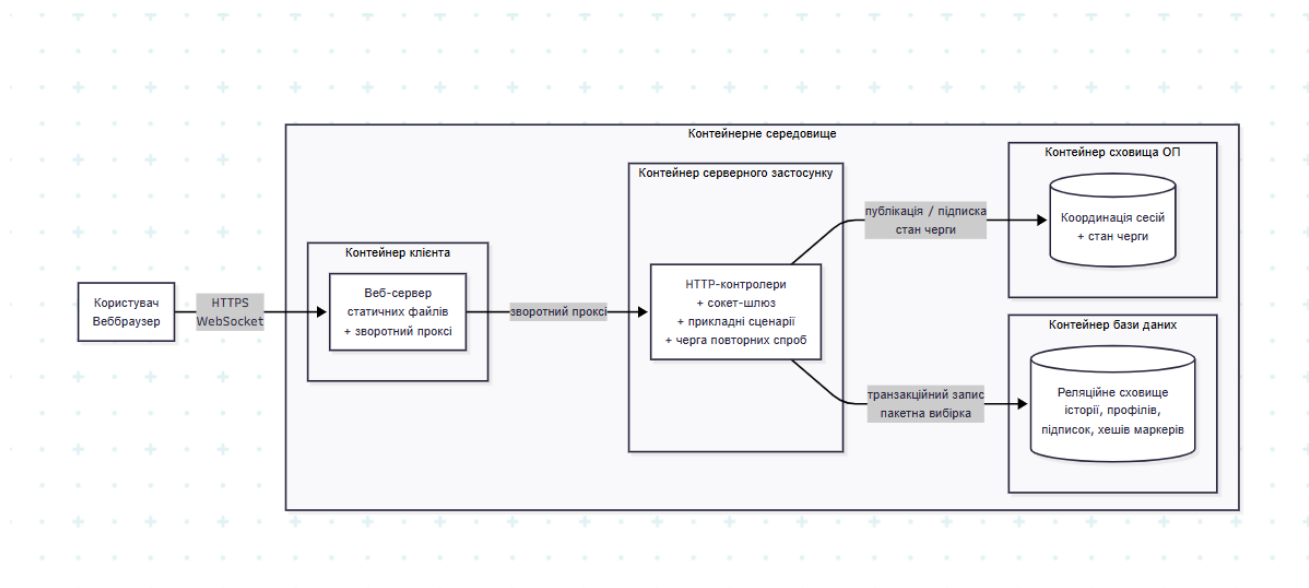


Рисунок 4.1 — Архітектура контейнерного розгортання програмного комплексу

Розгортання складається з чотирьох контейнерів, об'єднаних у спільну приватну мережу. Контейнер реляційної бази даних відповідає за персистентне зберігання історії повідомлень, профілів користувачів, складу каналів та хешів довготривалих маркерів сесій; саме на нього лягає підтримка наскрізної нумерації та обмежень унікальності клієнтських ідентифікаторів повідомлень, що є основою алгоритмів дедуплікації й виявлення пропусків (підрозділ 2.2). Контейнер сховища в оперативній пам'яті використовується для координації сокет-сесій між процесами серверного застосунку та для роботи фонові черги повторних спроб; його роль у розв'язанні задачі — забезпечити дешеву й швидку шину сповіщень між компонентами та зберігати стан черги без додаткового навантаження на реляційне ядро. Контейнер серверного застосунку містить транспортний шар, прикладні сценарії та інфраструктурні адаптери (підрозділ 3.4); він є вхідною точкою для всіх запитів клієнтів. Контейнер клієнтського застосунку обслуговує статичні файли інтерфейсу та виконує функцію зворотного проксі для запитів до серверного

контейнера, що дозволяє уникнути обмежень політики спільного походження на рівні браузера.

Конфігурація всіх контейнерів централізована через єдиний декларативний файл оркестрації. Він описує залежності контейнерів (наприклад, серверний застосунок стартує лише після ініціалізації бази даних та сховища в оперативній пам'яті), параметри мережі та політику відновлення після відмов окремих компонентів. Це дозволяє розгорнути всю інфраструктуру однією командою на будь-якому сумісному вузлі.

4.1.3 Введення системи в експлуатацію

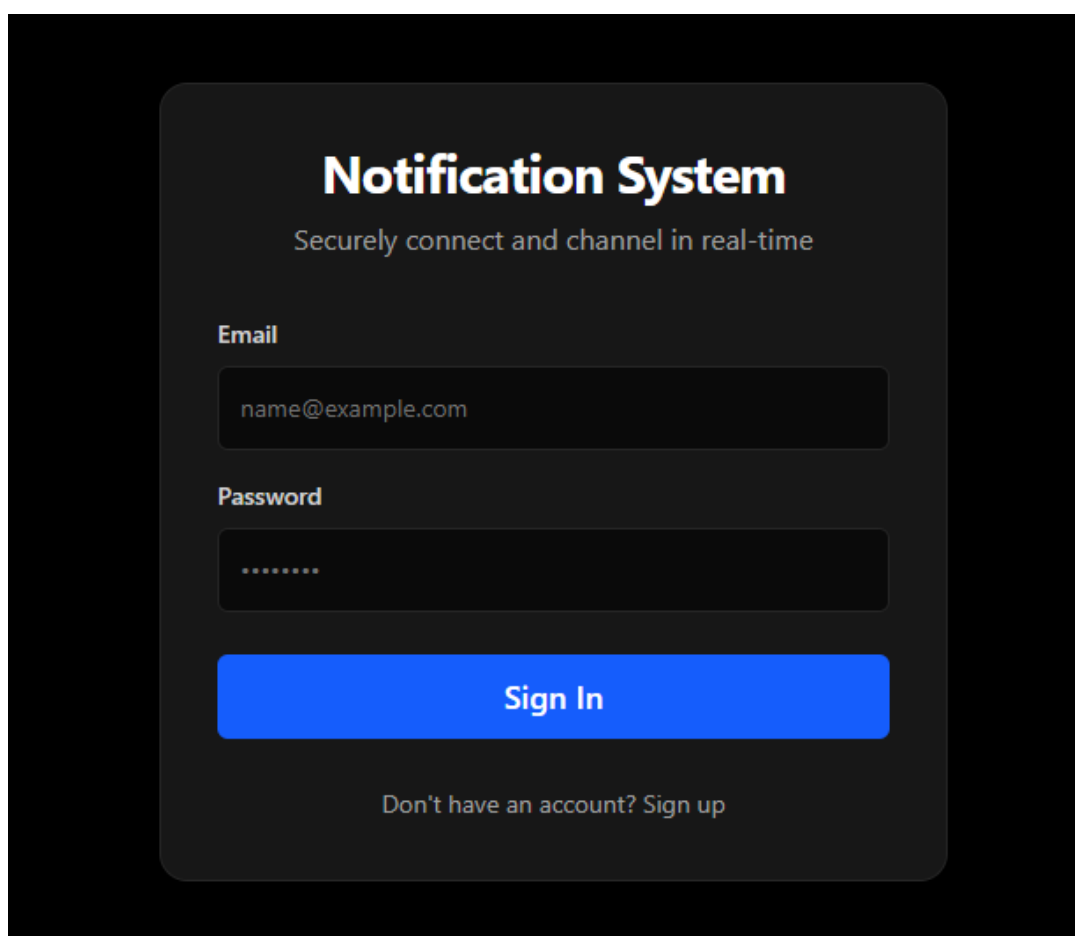
Введення системи в експлуатацію складається з шести етапів: отримання вихідного коду з системи контролю версій, конфігурування змінних середовища (секретні ключі для маркерів доступу, ключі автентифікації фонових каналів доставки, реквізити доступу до зовнішнього об'єктного сховища), збирання образів контейнерів, запуск інфраструктури в фоновому режимі, автоматичне застосування міграцій схеми бази даних та верифікація працездатності за журналами контейнерів.

4.2 Демонстрація реалізованих сценаріїв роботи системи

Підрозділ містить демонстрацію роботи реалізованої системи у межах ключових сценаріїв використання, які відповідають функціональним вимогам, сформульованим у розділі 1. Алгоритмічні підстави цих сценаріїв (життєвий цикл статусів повідомлення, дедуплікація, синхронізація, перевірка прав) формалізовано в розділі 2, а їх програмна реалізація описана в розділі 3, тому тут зосереджено увагу на спостережуваній поведінці системи з точки зору кінцевого користувача.

4.2.1 Реєстрація та вхід у систему

Сценарій автентифікації демонструє виконання вимоги до безпечного доступу. Користувач вводить електронну адресу та пароль у формі входу; за позитивної перевірки облікових даних серверний застосунок видає пару маркерів (короткоживучий маркер доступу та довготривалий маркер оновлення сесії), після чого клієнтський застосунок ініціює сокет-з'єднання та переходить до головного робочого простору. При тимчасовій втраті мережевого з'єднання повторне введення облікових даних не потрібне: збережений на клієнті маркер автоматично використовується для відновлення сесії — це демонструє реалізацію вимоги до прозорого відновлення стану клієнта (підрозділ 1.1.2). Вигляд форми входу наведено на рисунку 4.2.



The image shows a user authentication form titled "Notification System" with the subtitle "Securely connect and channel in real-time". The form is set against a dark background. It contains two input fields: "Email" with the placeholder text "name@example.com" and "Password" with masked characters "*****". Below these fields is a prominent blue "Sign In" button. At the bottom of the form, there is a link that reads "Don't have an account? Sign up".

Рисунок 4.2 — Інтерфейс форми авторизації користувача

Графічне компонування форми мінімізує когнітивне навантаження на користувача й безпосередньо інтегроване з підсистемою захищеної сесії, описаною в підрозділі 3.7.1.

4.2.2 Керування каналами

Для перевірки реалізації вимоги до структурованої комунікації (підрозділ 1.1.2) у системі реалізовано панель керування каналами, доступну користувачам з відповідною роллю. Сценарії включають створення нового каналу із заданим рівнем приватності, запрошення учасників за електронною адресою, динамічне призначення ролей (адміністратор, публікатор, підписник) та активацію режиму шифрування вмісту повідомлень (підрозділ 3.7.3). Усі зміни конфігурації каналу транслюються активним учасникам у реальному часі через сокет-канал. Розподіл повноважень за ролями підсумовано в таблиці 4.1.

Таблиця 4.1 — Матриця прав доступу учасників каналу

Тип операції	Адміністратор	Публікатор	Підписник
Перегляд історії повідомлень	Дозволено	Дозволено	Дозволено
Публікація кореневих сповіщень	Дозволено	Дозволено	Заборонено
Відповідь у наявних гілках	Дозволено	Дозволено	Дозволено
Видалення повідомлень	Усі	Лише власні	Заборонено
Зміна ролей учасників	Дозволено	Заборонено	Заборонено

Дані матриці підтверджують, що реалізована рольова модель забезпечує суворе розмежування дій між трьома рівнями повноважень: тільки адміністратор має повний контроль над каналом і його учасниками; публікатор обмежений лише власними повідомленнями при модерації; підписник позбавлений можливості створювати кореневі повідомлення, що відповідає вимозі ізоляції каналів масової розсилки від можливості несанкціонованого створення повідомлень. Інтерфейс панелі керування наведено на рисунку 4.3.

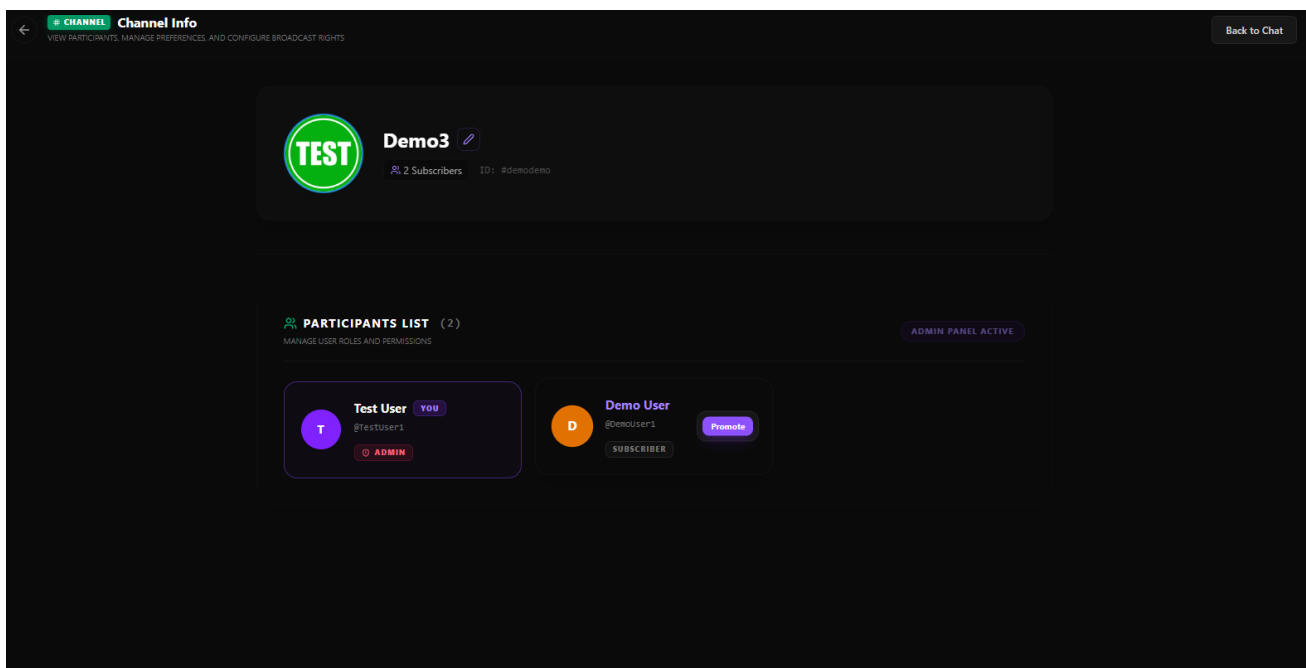


Рисунок 4.3 — Панель керування інформаційним каналом

Інтерфейс дозволяє адміністратору налаштовувати канал та склад учасників безпосередньо в режимі роботи системи, без необхідності перезапуску або повторного входу.

4.2.3 Відправлення сповіщень, статуси доставки та гілки відповідей

Сценарій публікації повідомлень є центральною демонстрацією реалізованих алгоритмів гарантованої доставки. Користувач, що має право публікації, вводить текст у поле редактора та ініціює відправлення; завдяки методу оптимістичного оновлення (підрозділ 3.5) повідомлення з'являється в історії одразу зі статусом Очікування. Подальші зміни статусу демонструють роботу скінченно-автоматної моделі життєвого циклу повідомлення (підрозділ 2.2.1): після збереження сервером статус змінюється на Відправлено, після підтвердження отримання одним із одержувачів — на Доставлено, після фіксації фактичного

перегляду — на Прочитано. Перелік усіх можливих станів та умов їх настання підсумовано в таблиці 4.2.

Таблиця 4.2 — Стани повідомлень у клієнтському інтерфейсі

Стан	Графічний індикатор	Умова настання
Очікування	Іконка годинника	Повідомлення додане локально, відповідь сервера не отримана
Відправлено	Без індикатора	Сервер зберіг повідомлення, але одержувачі його не підтвердили
Доставлено	Одна позначка	Принаймні один одержувач підтвердив прийняття
Прочитано	Дві позначки	Принаймні один одержувач (окрім автора) ознайомився з вмістом
Помилка мережі	Іконка попередження	Розрив зв'язку до моменту фіксації повідомлення на сервері

У разі розриву зв'язку до моменту збереження повідомлення в історії, повідомлення залишається в локальній черзі клієнта зі статусом Помилка мережі та автоматично надсилається повторно після відновлення зв'язку — це демонструє роботу інтеграції клієнтської офлайн-черги з серверною чергою повторних спроб.

Для запобігання змішуванню різних тем у головній стрічці каналу реалізовано сценарій гілок відповідей. Відповідь у гілці виконується тим самим алгоритмом гарантованої доставки, з тією відмінністю, що повідомлення отримує посилання на батьківське (підрозділ 3.3). Серверний шлюз спрямовує оновлення лише тим клієнтам, які активно переглядають відповідну гілку, що зменшує мережевий трафік для інших учасників каналу.

Загальний вигляд робочого простору з історією повідомлень та індикаторами статусів наведено на рисунку 4.4.

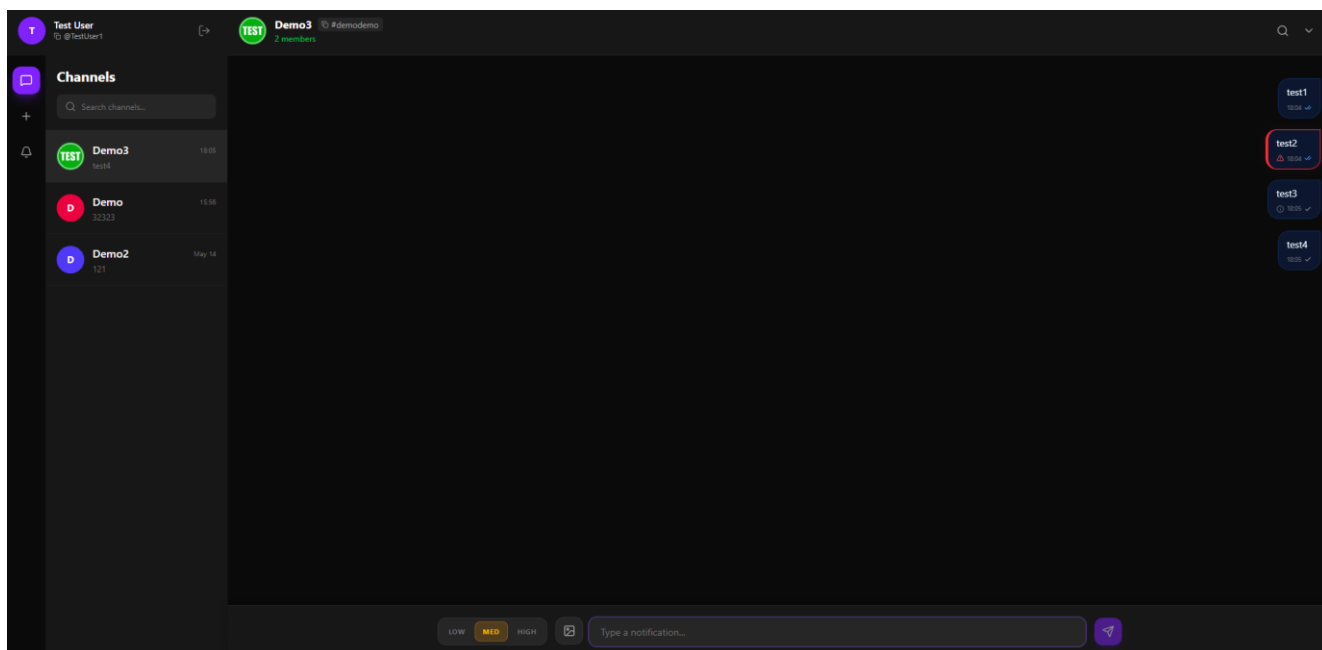


Рисунок 4.4 — Робочий простір системи та візуалізація статусів доставки

Реалізовані графічні індикатори надають користувачу зворотний зв'язок щодо кожного етапу обробки його повідомлень без потреби явних запитів про стан.

4.2.4 Робота з push-сповіщеннями

Для виконання вимоги до гарантованого інформування за умови, коли активна сесія користувача у системі відсутня (закрита вкладка браузера, заблокований пристрій), реалізовано підтримку фонових повідомлень. При першому вході в систему користувачу пропонується дозволити отримання таких повідомлень; у разі згоди клієнтський застосунок реєструє підписку, яка пов'язується з його обліковим записом (підрозділ 3.7.3). За наявності нового повідомлення в каналі, в якому в одержувача немає активної сокет-сесії, серверний застосунок перенаправляє його через зовнішній сервіс доставки фонових повідомлень із попереднім шифруванням корисного навантаження. Приклад відображення такого сповіщення на рівні операційної системи наведено на рисунку 4.5.

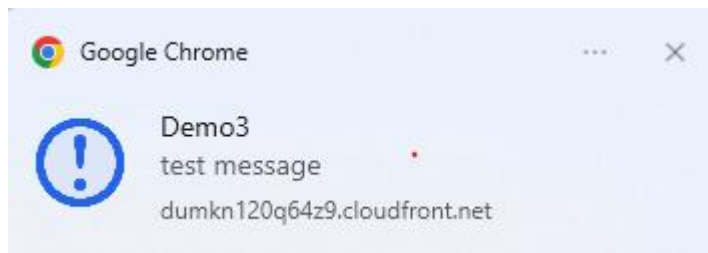


Рисунок 4.5 — Відображення фонового сповіщення в операційній системі

Натискання на сповіщення відкриває застосунок, виконує синхронізацію пропущених повідомлень за алгоритмом з підрозділу 2.3.1 і фокусує робочий простір на відповідному каналі. Це демонструє наскрізну інтеграцію механізму фонових повідомлень з основною підсистемою гарантованої доставки.

4.3 Тестування алгоритмів гарантованої доставки

Метою тестування була експериментальна перевірка коректності роботи алгоритмів, формалізованих у розділі 2, в умовах змодельованих мережових та інфраструктурних збоїв. Підрозділ містить опис методики тестування з визначенням кількісних показників, що вимірювалися, та результати проходження тестових сценаріїв.

4.3.1 Методика тестування

Тестування проводилося в контейнерному середовищі (підрозділ 4.1.2), що дозволяло точно контролювати стан кожного компонента інфраструктури та імітувати його відмови без впливу на сусідні компоненти. Для імітації мережових умов на клієнтській стороні використовувалися вбудовані засоби обмеження пропускної здатності та розриву з'єднання веббраузера.

Під час кожного сценарію фіксувалися такі кількісні показники: кількість надісланих тестових повідомлень, кількість повідомлень, що досягли одержувача,

кількість зафіксованих дублікатів, час до завершення синхронізації клієнта після відновлення з'єднання, кількість повторних спроб збереження в фоновій черзі та час спрацьовування механізму запобіжника. Перелік сценаріїв тестування разом з умовами випробувань, вхідними параметрами та критеріями успішності наведено в таблиці 4.3.

Таблиця 4.3 — Сценарії тестування та критерії оцінювання

Сценарій	Умови випробування	Вимірюваний показник	Критерій успішності
Втрата зв'язку клієнта під час публікації	Розрив мережі тривалістю до 60 с, 50 повідомлень, 5 паралельних клієнтів	Кількість втрачених повідомлень; час до завершення повторного надсилання після відновлення зв'язку	Втрати = 0; час повторного надсилання ≤ 5 с
Тимчасова недоступність бази даних	Зупинка контейнера БД на 60 с під час інтенсивного надсилання	Кількість збережених транзакцій після відновлення; кількість повторних спроб на одне повідомлення	Збережено ≥ 100 % надісланих повідомлень; повторних спроб ≤ 5 на повідомлення
Масове перепідключення клієнтів	Одночасне перепідключення 10 клієнтів, у кожного 50 пропущених повідомлень	Час до повної синхронізації останнього клієнта; кількість запитів до БД	Час ≤ 5 с; запитів до БД пропорційно кількості клієнтів, а не повідомлень

Кожен сценарій виконувався тричі для усереднення результатів та виключення випадкових відхилень.

4.3.2 Результати тестування

За результатами проведених випробувань зафіксовано таку поведінку системи в межах визначених умов.

У сценарії втрати зв'язку клієнта повідомлення зберігалися в локальній черзі застосунку та автоматично надсилалися після відновлення зв'язку; у проведених тестах втрати повідомлень не зафіксовано, а повторно синхронізовані повідомлення не утворювали дублікатів завдяки дедуплікації за клієнтським ідентифікатором; час повторного надсилання після відновлення зв'язку становив від 2 до 3 секунд.

У сценарії тимчасової недоступності бази даних повідомлення переходили до фоновій черги повторних спроб згідно з моделлю експоненційного відступу (підрозділ 2.3.2); після відновлення доступності бази даних усі повідомлення, надіслані під час недоступності, було збережено в межах п'яти повторних спроб.

У сценарії масового перепідключення клієнтів синхронізація пропущених повідомлень виконувалася одним пакетним запитом на кожного клієнта незалежно від обсягу пропущеної історії; час до завершення синхронізації для всіх десяти клієнтів становив від 1 до 2 секунд.

Усереднені за трьома запусками результати всіх сценаріїв наведено в таблиці 4.4; вони узгоджуються з очікуваними значеннями, заявленими в методиці тестування (підрозділ 4.3.1).

Таблиця 4.4 — Кількісні результати тестування

Сценарій	Надіслано повідомлень	Доставлено	Втрачено	Дублікатів	Час відновлення/ синхронізації
Втрата зв'язку клієнта (50 × 5 клієнтів)	250	250	0	0	2–3с після відновлення
Недоступність бази даних 60 с	100	100	0	0	5–10с після відновлення
Масове перепідключення 10 клієнтів	500	500	0	0	1-2с до повної синхронізації

Отримані показники часу синхронізації виявилися меншими за цільові значення (підрозділ 4.3.1), що пояснюється застосуванням пакетної вибірки пропущених повідомлень за порядковими номерами (підрозділ 2.3.1): кількість запитів до бази даних залежить від числа клієнтів, а не від обсягу пропущеної історії. Відсутність дублікатів у всіх сценаріях підтверджує коректність дедуплікації за клієнтським ідентифікатором. На відміну від готових хмарних сервісів (підрозділ 1.2.2), розроблений комплекс забезпечує прикладний контроль над послідовністю доставки та повторною синхронізацією без залежності від сторонньої інфраструктури.

Зазначені результати характеризують поведінку системи в межах описаної методики тестування та змодельованих умов і не можуть розглядатися як гарантія аналогічної поведінки в усіх можливих експлуатаційних умовах. Зокрема, перевірені сценарії охоплюють тривалість збоїв до однієї хвилини, обсяг тестових даних до п'ятисот повідомлень та до десяти паралельних клієнтів. Для висновків щодо поведінки системи за вищих навантажень або триваліших аварій потрібне додаткове промислове тестування з ширшим набором параметрів.

З точки зору досягнутого рівня гарантій доставки, реалізована модель забезпечує ефект, у літературі позначений як *effectively exactly-once*: повторну доставку пропущених повідомлень реалізовано за звійкою наскрізних порядкових номерів, на прикладному — ідемпотентність обробки за рахунок дедуплікації за клієнтським ідентифікатором повідомлення (підрозділ 2.2.2). У межах проведених тестів цей механізм забезпечив відсутність як втрат, так і дублікатів повідомлень у клієнтських інтерфейсах, що відповідає функціональним вимогам, сформульованим у розділі 1.

ВИСНОВКИ

У дипломній роботі розв'язано актуальне науково-прикладне завдання щодо забезпечення надійної передачі даних у розподілених системах реального часу шляхом розробки програмного комплексу «Реактивна система сповіщень із гарантованою доставкою повідомлень». За результатами проведених досліджень та програмної реалізації сформульовано такі основні висновки:

1. На основі аналізу існуючих підходів обґрунтовано використання комбінованої моделі гарантованої доставки — повторної доставки пропущених повідомлень за звіркою порядкових номерів разом із прикладною ідемпотентністю операцій. Обраний підхід виявився доцільним для розв'язання поставленої задачі, оскільки дозволяє мінімізувати ризики втрати та дублювання повідомлень без істотного зниження пропускної здатності системи.

2. За результатами аналізу програмних засобів встановлено, що для розв'язання поставленої задачі найбільш придатним є поєднання неблокуючого серверного середовища Node.js із фреймворком NestJS, реляційної СУБД PostgreSQL із засобом об'єктно-реляційного відображення Prisma ORM, сховища в оперативній пам'яті Redis із бібліотекою керування чергами BullMQ, а також клієнтської бібліотеки React у поєднанні з бібліотекою Socket.IO. Зазначені засоби разом покривають усі ключові вимоги до системи, сформульовані в розділі 1.

3. Розроблено реактивну систему сповіщень, у якій реалізовано сформульовані алгоритми забезпечення гарантованої доставки. Досягнутим результатом є створення програмного комплексу, що забезпечує надійну доставку повідомлень користувачам та автоматичне відновлення повноти історії після тимчасового розриву мережевого з'єднання.

4. За результатами проведених випробувань встановлено, що реалізовані алгоритми мінімізують ризик втрати або дублювання повідомлень у межах визначених умов тестування. У зафіксованих сценаріях не виявлено випадків

втрати або дублювання повідомлень, а час повної фонові синхронізації клієнта після перепідключення становить до трьох секунд.

Таким чином, мету дипломної роботи досягнуто за рахунок проведеного аналізу методів та програмних засобів забезпечення гарантованої доставки повідомлень у реактивних системах, формалізації відповідних алгоритмів, програмної реалізації механізмів наскрізної нумерації, підтвердження отримання, повторної доставки та дедуплікації повідомлень, а також експериментальної перевірки розроблених рішень.

Практичне значення одержаних результатів полягає в можливості використання розробленого програмного комплексу як основи для побудови надійного каналу сповіщень у корпоративних інформаційних системах, системах моніторингу інфраструктури та інших застосунках, що потребують надійного обміну повідомленнями в режимі реального часу.

Перспективними напрямками подальшого розвитку є проведення навантажувального та промислового тестування системи на більших обсягах даних та триваліших аварійних умовах, розширення механізмів моніторингу й аудиту роботи черги повторних спроб, а також інтеграція системи з наявними рішеннями автентифікації та керування доступом у межах конкретних організаційних середовищ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. RFC 6202: Known Issues and Best Practices for the Use of Long Polling and Streaming in Web. Internet Engineering Task Force (IETF). URL: <https://datatracker.ietf.org/doc/html/rfc6202> (дата звернення: 17.05.2026).
2. Server-Sent Events. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events (дата звернення: 17.05.2026).
3. WebSocket API. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 15.05.2026).
4. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol : O'Reilly Media, 2017. 616 p.
5. NestJS — A progressive Node.js framework. NestJS Documentation. URL: <https://docs.nestjs.com/> (дата звернення: 14.05.2026).
6. Prisma | Next-generation Node.js and TypeScript ORM. Prisma Documentation. URL: <https://www.prisma.io/docs/> (дата звернення: 15.05.2026).
7. PostgreSQL: The World's Most Advanced Open Source Relational Database. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs/15/> (дата звернення: 12.05.2026).
8. Redis. Redis Documentation. URL: <https://redis.io/docs/> (дата звернення: 16.05.2026).
9. BullMQ — Message Queue and Batch processing for NodeJS based on Redis. BullMQ Documentation. URL: <https://docs.bullmq.io/> (дата звернення: 15.05.2026).
10. React – A JavaScript library for building user interfaces. React Documentation. URL: <https://react.dev/> (дата звернення: 16.05.2026).
11. Socket.IO. Socket.IO Documentation. URL: <https://socket.io/docs/v4/> (дата звернення: 14.05.2026).

12. Web Push API. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Push_API (дата звернення: 15.05.2026).
13. Service Worker API. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API (дата звернення: 17.05.2026).
14. Docker Documentation. Docker Docs. URL: <https://docs.docker.com/> (дата звернення: 16.05.2026).
15. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley Professional, 2003. 560 p.
16. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
17. Intersection Observer API. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API (дата звернення: 17.05.2026).
18. Hohpe G., Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston : Addison-Wesley, 2003. 736 p.
19. RFC 7519: JSON Web Token (JWT). Internet Engineering Task Force (IETF). URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата звернення: 14.05.2026).

ДОДАТОК А

Програмний код серверної частини реактивної системи сповіщень із гарантованою доставкою

Програмні засоби: мова програмування — TypeScript; середовище виконання — Node.js; фреймворки та бібліотеки — NestJS, Prisma ORM, BullMQ, Socket.IO; база даних — PostgreSQL.

TypeScript

// Лістинг А.1 — Транспортний шлюз для забезпечення двостороннього зв'язку

```
@WebSocketGateway({
  path: '/api/socket.io/',
  cors: { origin: process.env.FRONTEND_URL, credentials: true },
  transports: ['websocket'],
})
export class NotificationsGateway implements OnGatewayInit, OnGatewayConnection
{
  @WebSocketServer()
  server: Server;

  constructor(
    private readonly connectionTracker: ConnectionTrackerService,
    private readonly acknowledgeUseCase: AcknowledgeNotificationUseCase,
    private readonly readUseCase: ReadNotificationUseCase,
    private readonly syncNotificationsUseCase: SyncNotificationsUseCase,
    private readonly wsConnectionService: WsConnectionService,
  ) {}
```

```
afterInit(server: Server) {
  this.connectionTracker.setServer(server);
}
```

```
async handleConnection(client: Socket) {
  try {
    await this.wsConnectionService.authenticateAndJoinRooms(client);
  } catch (e) {
    client.disconnect();
  }
}
```

```
@UseGuards(WsAuthGuard)
@SubscribeMessage(SocketEvent.ACK)
async handleAck(@ConnectedSocket() client: Socket, @MessageBody() data: {
notificationId: string }) {
  const userId = client.data.user?.sub;
  if (userId) await this.acknowledgeUseCase.execute(userId, data.notificationId);
}
```

```
@UseGuards(WsAuthGuard)
@SubscribeMessage(SocketEvent.READ)
async handleRead(@ConnectedSocket() client: Socket, @MessageBody() data: {
notificationId: string }) {
  const userId = client.data.user?.sub;
  if (userId) await this.readUseCase.execute(userId, data.notificationId);
}
```

```
@UseGuards(WsAuthGuard)
```

```

@SubscribeMessage(SocketEvent.SYNC_NOTIFICATIONS)
async handleSyncNotifications(
  @ConnectedSocket() client: Socket,
  @MessageBody() data: { syncRequests: { channelId: string; lastSequence: number }[]
}
) {
  const userId = client.data.user?.sub;
  if (!userId) return { error: 'Unauthorized' };
  const notifications = await this.syncNotificationsUseCase.execute(userId,
data.syncRequests);
  return { notifications };
}
}

```

// Лістинг А.2 — Сервіс авторизації та маршрутизації підключень

```

@Injectable()
export class WsConnectionService {
  constructor(
    @Inject(TOKEN_SERVICE) private readonly tokenService: ITokenService,
    @Inject('CHAT_REPO') private readonly channelRepository: IChannelRepository,
  ) {}

  async authenticateAndJoinRooms(client: Socket): Promise<string> {
    const token = this.extractToken(client);
    if (!token) throw new WsException('Unauthorized');
    const payload = await this.tokenService.verifyAccessToken(token);
    client.data.user = payload;
    const userId = payload.sub;

```

```

await client.join(userId);
if (payload.role === 'GLOBAL_ADMIN') await client.join('admin');

const channelIds = await this.channelRepository.findUserChannelIds(userId);
for (const channelId of channelIds) {
    await client.join(channelId);
}
return userId;
}

private extractToken(client: Socket): string | null {
    const authHeader = client.handshake.headers.authorization;
    return client.handshake.auth.token || (authHeader?.startsWith('Bearer ') ?
authHeader.split(' ')[1] : null);
}
}

// Лістинг А.3 — Прикладний сценарій гарантованої публікації сповіщення
@Injectable()
export class ProcessNotificationUseCase {
    constructor(
        @Inject('MESSAGE_REPO') private readonly repository: INotificationRepository,
        @Inject(MESSENGER_SENDER) private readonly sender: INotificationsSender,
        @Inject('CHAT_REPO') private readonly channelRepository: IChannelRepository,
        @Inject(ADMIN_REPOSITORY) private readonly adminRepository:
IAdminRepository,
        private readonly eventEmitter: EventEmitter2,
        @Inject(FAILED_NOTIFICATIONS_QUEUE) private readonly failedQueue:
IFailedNotificationsQueue,

```

```
) {}
```

```
async execute(
  channelId: string, senderId: string, text: string, clientNotificationId?: string,
  priority?: NotificationPriority, parentNotificationId?: string, attachments?: string[],
  throwOnError = false
): Promise<{ success: boolean; status: 'sent' | 'queued'; notification?: any }> {

  const activeBan = await this.adminRepository.getActiveBan(channelId);
  if (activeBan) throw new ForbiddenException(`Channel is banned. Reason:
  ${activeBan.reason}`);

  try {
    if (clientNotificationId) {
      const existing = await
this.repository.findByClientNotificationId(clientNotificationId);
      if (existing) return { success: true, status: 'sent', notification:
this.formatNotificationPayload(existing, 'DELIVERED') };
    }

    const role = await this.channelRepository.getMemberRole(channelId, senderId);
    if (!Channel.canMemberPost(role, !!parentNotificationId)) return { success: false,
status: 'sent' };

    const notification = parentNotificationId
      ? Notification.createReply(parentNotificationId, channelId, senderId, text,
clientNotificationId, attachments)
      : Notification.createRoot(channelId, senderId, text, priority, clientNotificationId,
attachments);
```

```

    const savedNotification = await this.repository.save(notification);
    this.sender.send(channelId,      this.formatNotificationPayload(savedNotification,
'DELIVERED'));
    this.eventEmitter.emit(NotificationDomainEvent.CREATED,      new
NotificationCreatedEvent(savedNotification));

    return      {      success:      true,      status:      'sent',      notification:
this.formatNotificationPayload(savedNotification, 'DELIVERED') };
  } catch (error) {
    if (throwOnError) throw error;

    await this.failedQueue.enqueue({
      channelId, senderId, text, clientNotificationId, priority, parentNotificationId,
attachments,
    });
    return { success: true, status: 'queued' };
  }
}

private formatNotificationPayload(n: any, status: string) {
  return {
    id: n.id, channelId: n.channelId, senderId: n.senderId, text: n.text, status,
    createdAt:      n.createdAt,      sequence:      n.sequence,      clientNotificationId:
n.clientNotificationId,
    priority:      n.priority,      parentNotificationId:      n.parentNotificationId,      attachments:
n.attachments,
  };
}

```

```
}
```

// Лістинг А.4 — Інфраструктурний репозиторій із підтримкою шифрування
@Injectable()

```
export class PrismaNotificationRepository implements INotificationRepository {
  constructor(
    private readonly prisma: PrismaService,
    private readonly encryptionService: EncryptionService,
  ) {}
```

```
  async save(notification: Notification): Promise<Notification> {
    const persistenceData = NotificationMapper.toPersistence(notification);
    const data = await this.prisma.notification.create({
      data: {
        ...persistenceData,
        text: this.encryptionService.encrypt(persistenceData.text),
        attachments: { create: notification.attachments?.map(url => ({ url })) || [] }
      },
      include: { attachments: true }
    });
    return this.mapToDomain(data);
  }
```

```
  private mapToDomain(data: any): Notification {
    if (data.text) data.text = this.encryptionService.decrypt(data.text);
    return NotificationMapper.toDomain(data);
  }
}
```