МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

на тему: «Розробка кросплатформеного мобільного додатку для служби психологічної підтримки студентів Національного університету «Острозька академія» на базі Flutter »

Виконав: студент 4 курсу, групи КН-42
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Поліщук Максим Борисович

Керівник: *старший викладач кафедри ЕММІТ,*
Клебан Юрій Вікторович

Рецензент: *кандидат технічних наук, доцент,*
доцент кафедри прикладної математики
Донецького національного університету
імені Василя Стуса
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри інформаційних технологій та аналітики даних _____
(проф., д.е.н. Кривицька О.Р.)

Протокол No 11 від 20 травня 2026 р.

Острог, 2026

АНОТАЦІЯ

кваліфікаційної роботи

на здобуття освітнього ступеня бакалавра

Тема: *«Розробка кросплатформеного мобільного додатку для служби психологічної підтримки студентів Національного університету «Острозька академія» на базі Flutter»*

Автор: *Поліщук Максим Борисович*

Науковий керівник: *науковий ступінь, вчене звання, посада, ПІБ керівника*

Захищена «___»_____ 2026 року.

Пояснювальна записка до кваліфікаційної роботи: 40 с., 2 рис. (мінімум), 10 табл. (мінімум), 2 додатків (мінімум), 12 джерел.

Ключові слова: *інформаційна система, кросплатформний додаток, наскрізне шифрування (E2EE), AES-256-GCM, CQRS, SignalR, Flutter, React, ASP.NET Core.*

Короткий зміст праці:

Кваліфікаційна робота присвячена проектуванню та розробці інформаційної системи психологічної підтримки OstrohHelp для здобувачів вищої освіти. Основною метою є створення сучасної, масштабованої та конфіденційної платформи для онлайн-консультування із застосуванням наскрізного шифрування (E2EE).

Backend системи реалізовано на ASP.NET Core 8.0 із використанням PostgreSQL, JWT-автентифікації та архітектурного патерну CQRS. Для забезпечення миттєвої взаємодії між користувачами використано технологію SignalR. Конфіденційність повідомлень забезпечується алгоритмом AES-256-GCM, а ключі шифрування генеруються за допомогою HKDF-SHA256 і зберігаються лише в оперативній пам'яті клієнтів.

Розроблено два клієнтські застосунки: мобільний на Flutter та веб-застосунок на React. Система підтримує ролі студента, психолога та керівника служби, а також містить адміністративний функціонал, аудит дій користувачів і механізм Rate Limitin.

Практична цінність роботи полягає у створенні готової системи OstrohHelp, яка може бути використана для організації безпечної та конфіденційної психологічної підтримки в Національному університеті «Острозька академія».

The qualification work is devoted to the design and development of the OstrohHelp psychological support information system for higher education students. The main goal of the project is to create a modern, scalable, and confidential platform for online counseling using End-to-End Encryption (E2EE).

The backend of the system is implemented using ASP.NET Core 8.0 with PostgreSQL, JWT authentication, and the CQRS architectural pattern. SignalR technology is used to provide real-time communication between users. Message confidentiality is ensured through the AES-256-GCM encryption algorithm, while encryption keys are generated using HKDF-SHA256 and stored only in the clients' RAM.

Two client applications were developed: a mobile application using Flutter and a web application using React. The system supports the roles of Student, Psychologist, and Head of Service, and includes administrative functionality, user activity audit logging, and a Rate Limiting mechanism.

The practical value of the work lies in the creation of the ready-to-use OstrohHelp system, which can be applied to provide secure and confidential psychological support at the National University of Ostroh Academy.

ЗМІСТ

ВСТУП

Кваліфікаційна робота на тему «Розробка кросплатформенного мобільного додатку для служби психологічної підтримки студентів Національного університету «Острозька академія» на базі Flutter» присвячена створенню багатокомпонентної інформаційної системи OstrohHelp. Актуальність теми зумовлена зростаючою потребою здобувачів вищої освіти у доступних, конфіденційних та сучасних механізмах психологічної допомоги. Традиційні методи консультування мають обмеження, пов'язані з графіком та психологічними бар'єрами, які ефективно долаються за допомогою цифрового крос-платформного рішення. Основний виклик полягає у забезпеченні максимальної конфіденційності, що досягається застосуванням наскрізного шифрування (E2EE).

Об'єкт дослідження - процес розробки та забезпечення безпеки кросплатформної інформаційної системи психологічної підтримки.

Предмет дослідження - архітектурні рішення, методики реалізації наскрізного шифрування та real-time комунікації у крос-платформному середовищі.

Мета кваліфікаційної роботи полягає в узагальненні та систематизації знань і практичних навиків, набутих під час навчання, а також їх застосуванні у процесі прийняття проєктних рішень для створення багатокомпонентної інформаційної системи.

Задля досягнення поставленої мети визначено такі основні завдання:

1. Систематизація, закріплення та розширення теоретичних знань і практичних навиків у сферіпроектування інформаційних систем, зокрема у сфері веб- та мобільної розробки.
2. Обґрунтування архітектурного рішення, що забезпечує високий рівень конфіденційності даних за допомогою наскрізного шифрування AES-256-GCM та детермінованої генерації ключів HKDF-SHA256.
3. Удосконалення вміння застосовувати сучасні технології програмування та патерни для розробки клієнт-серверних застосувань.
4. Перевірка вміння застосовувати технології розробки інформаційних

5. систем на практиці та обробляти й систематизувати результати досліджень.

Методи дослідження: Для виконання роботи використано методи системного аналізу, об'єктно-орієнтованого програмування, архітектурний патерн CQRS, а також криптографічні методи AES-256-GCM та HKDF-SHA256 для забезпечення E2EE. Практична цінність роботи полягає у створенні готового до впровадження програмного продукту - системи OstrohHelp, яка може бути використана для забезпечення конфіденційної психологічної підтримки здобувачів вищої освіти Національного університету «Острозька академія». Структура роботи: Кваліфікаційна робота складається зі вступу, чотирьох основних розділів, висновку, списку використаних джерел та додатків. Основні розділи:

1. Теоретична частина (Теоретичне обґрунтування архітектури).
2. REST API (Проєктування backend-системи).
3. Фронт Flutter (Реалізація мобільного застосунку).
4. Фронт React (Реалізація веб-застосунку).

Виконання цієї роботи демонструє здатність до проектування та розробки програмного забезпечення із застосуванням різних парадигм програмування, а також розуміння концепції інформаційної безпеки та принципів безпечного проектування програмного забезпечення.

РОЗДІЛ 1. ЗАГАЛЬНІ ПОЛОЖЕННЯ ТА ТЕОРЕТИЧНЕ ОБҐРУНТУВАННЯ АРХІТЕКТУРИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

1.1. Актуальність теми та мета кваліфікаційної роботи

Актуальність розробки інформаційної системи психологічної підтримки, представленої проєктом OstrohHelp, зумовлена необхідністю забезпечення доступних, конфіденційних та сучасних механізмів консультування для здобувачів вищої освіти. Традиційні форми психологічної допомоги часто мають обмеження, пов'язані з графіком, місцем розташування та психологічними бар'єрами для звернення, які ефективно долаються за допомогою цифрових рішень.

Система OstrohHelp створюється як крос-платформний застосунок, що гарантує високий рівень безпеки та конфіденційності даних завдяки застосуванню наскрізного шифрування (E2EE) та технологій реального часу (Real-time, SignalR).

Мета кваліфікаційної роботи полягає в узагальненні та систематизації знань і практичних навиків, набутих під час навчання, а також їх застосуванні у процесі прийняття проєктних рішень для створення багатокомпонентної інформаційної системи.

Завдання кваліфікаційної роботи:

1. Систематизація, закріплення та розширення теоретичних знань і практичних навиків у сфері проектування інформаційних систем, зокрема у сфері веб- та мобільної розробки.
2. Обґрунтування архітектурного рішення, що забезпечує високий рівень конфіденційності даних за допомогою наскрізного шифрування та детермінованої генерації ключів HKDF-SHA256.
3. Удосконалення вміння застосовувати сучасні технології програмування, такі як ASP.NET Core 8.0, Flutter та React, разом із патернами CQRS і SignalR, для розробки клієнт-серверних застосувань.
4. Перевірка вміння застосовувати технології розробки інформаційних систем на практиці.

Виконання цієї роботи демонструє здатність до проектування та розробки програмного забезпечення із застосуванням різних парадигм програмування (СК8), зокрема об'єктно-орієнтованого програмування та архітектурного патерну CQRS, що дозволило ефективно розділити логіку читання та запису. Крім того, робота підтверджує розуміння концепції інформаційної безпеки та принципів безпечного проектування програмного забезпечення (ПР15, СК5, СК7), що було реалізовано через впровадження наскрізного шифрування (E2EE) за допомогою алгоритму AES-256-GCM та детермінованої генерації ключів HKDF-SHA256, що є фундаментальним для забезпечення найвищого рівня конфіденційності даних.

1.2. Огляд предметної області та функціональність системи

Інформаційна система OstrohHelp розроблена для підтримки психологічних служб університету та має три основні ролі користувачів, кожна з яких має чітко визначений набір дозволів та функціональних потоків. Це забезпечує структурований доступ до конфіденційних даних та гарантує виконання усіх адміністративних і консультаційних функцій, необхідних для ефективної роботи служби психологічної підтримки.

Таблиця 1.1. “Ролі в додатку OstrohHelp”

Роль	Призначення	Ключовий функціонал
Student	Здобувач вищої освіти	Створення анкет , перегляд своїх консультацій , безпечний чат.
Psychologist	Психолог служби підтримки	Перегляд та прийняття анкет , створення та управління консультаціями, чат зі студентами.
HeadOfService	Керівник служби	Повний адміністративний контроль.

Основний потік роботи (User Flow):

1. Автентифікація.
    a. Користувач здійснює вхід через Google OAuth 2.0.
2. Створення анкети.
    a. Студент заповнює анкету, яка отримує статус Pending.
3. Прийняття анкети.
    a. Психолог переглядає список анкет зі статусом Pending (GET /api/Questionnaire/all) та приймає одну, викликаючи POST /api/Consultations/Accept-Questionnaire. Це створює запис Consultation.
4. Чат та Real-time.
    a. Учасники консультації (Студент і Психолог) приєднуються до чату, отримують ключ шифрування через SignalR та обмінюються повідомленнями в режимі реального часу.

1.3. Обґрунтування архітектурного підходу та моделі безпеки

Backend системи OstrohHelpApi побудований на ASP.NET Core 8.0 і дотримується принципів чистої архітектури, використовуючи паттерн CQRS (Command Query Responsibility Segregation) для управління бізнес-логікою.

CQRS забезпечує розділення операцій, що змінюють стан системи (Commands), від операцій, що лише отримують дані (Queries).

1. Commands - виконуються асинхронно, ініціюють транзакції та зміни в базі даних.
2. Queries - оптимізовані для швидкого читання даних, можуть використовувати кешування, але ніколи не змінюють стану системи.

Це розділення дозволяє незалежно оптимізувати операції читання (швидке завантаження історії чату) та операції запису (Commands), що критично важливо для масштабованості та підтримки складного домену, де безпека та швидкість є

пріоритетами, а також значно спрощує подальше впровадження нових бізнес-функцій і забезпечує високу транзакційну цілісність при зміні стану системи.

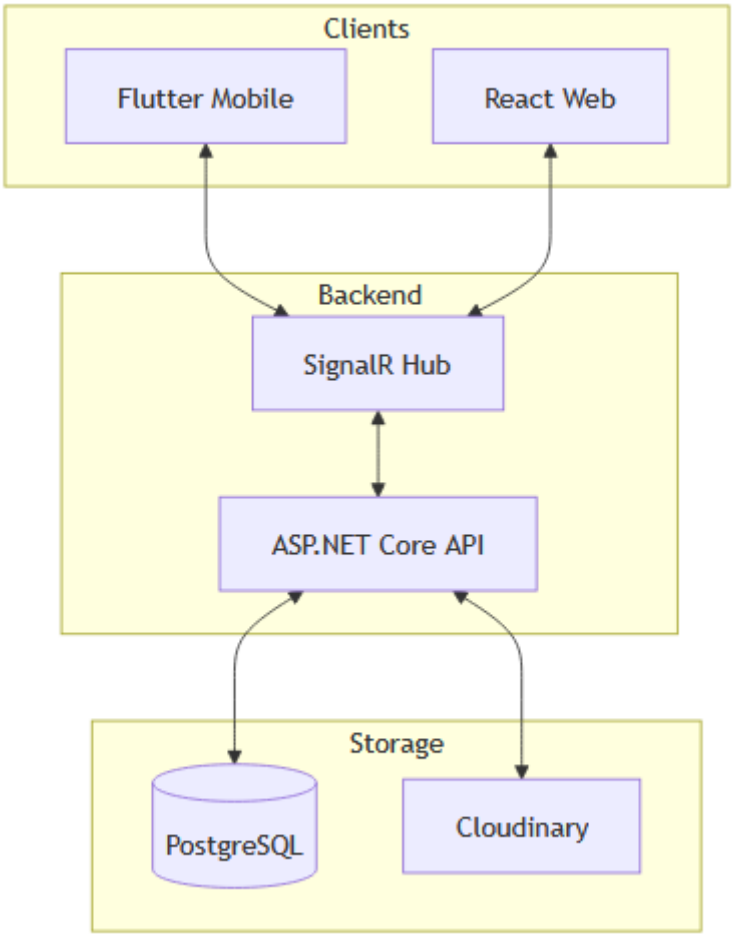

Рис 1.1 “Діаграма взаємодії компонентів”
Джерело: розробка автора

Для забезпечення миттєвого обміну повідомленнями та статусами (online/offline, прочитання) використовується технологія SignalR (WebSocket). Це рішення замінило застарілий та ресурсоємний підхід REST API Polling.

Таблиця 1.2. “Технічні характеристики REST Polling vs SignalR”

Параметр	REST API Polling (Старий)	SignalR (WebSocket, Новий)
Затримка (Latency)	2-5 секунд (інтервал опитування)	< 100 мілісекунд

З'єднання	Багато коротких HTTP-запитів	Одне постійне з'єднання (Single Persistent Connection)
Ефективність	Високе навантаження на сервер, швидка розрядка батареї	Дуже низьке навантаження, економія ресурсів
Масштабованість	Низька, швидко досягає ліміту сервера	Висока (5000+ користувачів на одному хабі)

Ключовий функціонал, що реалізується через SignalR:

1. Передача ключа шифрування (ReceiveConsultationKey).
2. Миттєва доставка повідомлень (ReceiveMessage).
3. Оновлення статусу присутності (UserStatusChanged).
4. Оновлення статусу прочитання (ReceiveMarkedAsRead).

1.4. Архітектура наскрізного шифрування (E2EE)

Конфіденційність в OstrohHelp досягається завдяки наскрізному шифруванню (End-to-End Encryption, E2EE) повідомлень, де шифрування та дешифрування відбуваються виключно на клієнтській стороні (Flutter/React). Сервер OstrohHelpApi зберігає повідомлення лише у вигляді зашифрованих бінарних даних.

Для шифрування обрано надійний алгоритм AES-256-GCM (Advanced Encryption Standard in Galois/Counter Mode):

1. AES-256: Забезпечує високу криптографічну стійкість.
2. GCM: Режим роботи, який забезпечує не лише конфіденційність, але й аутентифікацію даних. Це означає, що разом із зашифрованим контентом генерується Аутентифікаційний тег (AuthTag), що запобігає будь-яким несанкціонованим змінам даних під час передачі чи зберігання (Tamper Detection).

Кожне повідомлення в базі даних (Message entity) містить три Base64-кодовані поля, які були згенеровані клієнтом:

1. EncryptedContent: Зашифрований текст повідомлення.
2. IV (Initialization Vector): 12 унікальних байтів, згенерованих випадковим чином для кожного повідомлення, що необхідні для безпечної роботи GCM.
3. AuthTag: 16 байтів, що використовуються для перевірки цілісності даних при дешифруванні.

У разі спроби дешифрування скомпрометованого повідомлення, клієнт отримує помилку AuthenticationTagMismatchException, гарантуючи, що користувач не побачить змінений текст. Таким чином, система суворо дотримується принципу цілісності даних (Tamper Detection), запобігаючи витоку інформації у випадку компрометації каналу.

Консультаційний ключ, необхідний для шифрування/дешифрування, ніколи не зберігається в базі даних і не передається через REST API. Він генерується сервером динамічно та детерміновано за допомогою HKDF-SHA256 (HMAC-based Extract-and-Expand Key Derivation Function, RFC 5869).

Потік генерації ключа:

1. Вхідні дані (Input): Сервер використовує два ключові компоненти:
2. Master Key (K): 256-бітний ключ, що зберігається у файлі конфігурації.env на сервері і ніколи не покидає середовище backend.
3. ConsultationId (Salt): Унікальний GUID консультації, який виступає як сіль.
4. Виведення (Output): HKDF-SHA256 детерміновано виводить 256-бітний Consultation Key.

Передача ключа відбувається тільки через захищене SignalR з'єднання:

1. Крок 1: Запит. Клієнт викликає SignalR метод JoinConsultation, ініціюючи процес підключення до групи чату консультації.
2. Крок 2: Перевірка. Сервер перевіряє права користувача за

допомогою інтерфейсу IConsultationAccessChecker на участь у поточній консультації.

3. Крок 3: Генерація. Сервер генерує Consultation Key за допомогою HKDF-SHA256.
4. Крок 4: Передача. Ключ надсилається клієнту через приватну SignalR-подію ReceiveConsultationKey у форматі Base64.
5. Крок 5: Зберігання. Клієнт декодує ключ та зберігає його у RAM-пам'яті (не на диску), використовуючи для подальших операцій шифрування/дешифрування.
6. Крок 6: Обмін. Повідомлення шифруються локально клієнтом та надсилаються на backend, який їх зберігає і транслює іншому учаснику через SignalR ReceiveMessage.

```
{
 ConsultationId: "abc-123...",
 Key: "XtLurkNiKAseW287L...", // BASE64-кодований 256-бітний ключ
 Algorithm: "AES-256-GCM"
}
```

Лістинг 1.1. “Приклад Event'у для передачі ключа”

Джерело: розробка автора

1.5. Обґрунтування вибору крос-платформних рішень

Для максимальної доступності, система використовує два клієнти, що взаємодіють з єдиним backend OstrohHelpApi. Це ключове для кросплатформної розробки, оскільки дозволяє охопити мобільну (Android/iOS) та веб-аудиторію. Таким чином, здобувачі вищої освіти мають зручний доступ до конфіденційної підтримки незалежно від пристрою.

Таблиця 1.3. “Обґрунтування вибору Flutter”

Причина вибору	Пояснення
Крос-платформність	Створення єдиної кодової бази та уніфікованого UI/UX для обох мобільних операційних систем.
Продуктивність	Flutter забезпечує високу швидкість роботи, близьку до нативної.
Інтеграція SignalR (signalr_netcore)	Використовується пакет для real-time комунікації та передачі ключа шифрування.
E2EE (End-to-End Encryption)	Реалізація AES-256-GCM шифрування здійснюється за допомогою бібліотеки pointycastle.

Обґрунтування вибору Flutter для мобільного клієнта (OstrohHelpApp) базується на необхідності забезпечити уніфікований та високопродуктивний застосунок для платформ Android та iOS. Крос-платформність Flutter дозволила створити єдину кодову базу, що мінімізує витрати на розробку та гарантує єдиний користувацький інтерфейс (UI/UX) для обох мобільних операційних систем.

Крім високої продуктивності, близької до нативної, Flutter також забезпечує необхідну інтеграцію для архітектури безпеки:

1. Real-time комунікація: Для роботи з SignalR ChatHub backend використовується пакет signalr_netcore, який відповідає за миттєву доставку повідомлень та захищену передачу 256-бітного ключа шифрування.
2. Криптографія: Реалізація наскрізного шифрування (E2EE) за допомогою алгоритму AES-256-GCM здійснюється через Dart-бібліотеку pointycastle. Ця бібліотека дозволяє виконувати шифрування та

дешифрування повідомлень безпосередньо на пристрої клієнта, забезпечуючи аутентифікацію даних через AuthTag, що критично важливо для конфіденційності.

Таблиця 4 “Обґрунтування архітектурних рішень для веб-клієнта ”

Причина вибору	Пояснення
Стандарт індустрії	React є провідним рішенням для створення односторінкових застосунків (SPA) з високою інтерактивністю.
Екосистема	Дозволяє легко інтегрувати такі необхідні бібліотеки, як react-router-dom (маршрутизація), Axios (HTTP-запити) та @microsoft/signalr (WebSocket).
UX/UI	Використовується Tailwind CSS для швидкої та адаптивної стилізації, що забезпечує коректне відображення на різних пристроях.
E2EE	Веб-версія реалізує той самий криптографічний потік (отримання ключа через SignalR, шифрування перед відправкою) за допомогою відповідних JavaScript-бібліотек.

Веб-версія містить ключові сторінки, такі як LoginPage, ConsultationsPage, AdminPanelPage та MyQuestionnairesPage.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ (REST API)

Розділ 2 детально описує архітектуру backend-системи OstrohHelpApi, яка є ядром інформаційної системи психологічної підтримки. Серверна частина розроблена на фреймворку ASP.NET Core 8.0 і використовує PostgreSQL 16 як базу даних, забезпечуючи взаємодію з клієнтськими застосунками (Flutter та React) через стандартизований REST API та технології реального часу SignalR.

2.1. Архітектурний патерн Command Query Responsibility Segregation (CQRS)

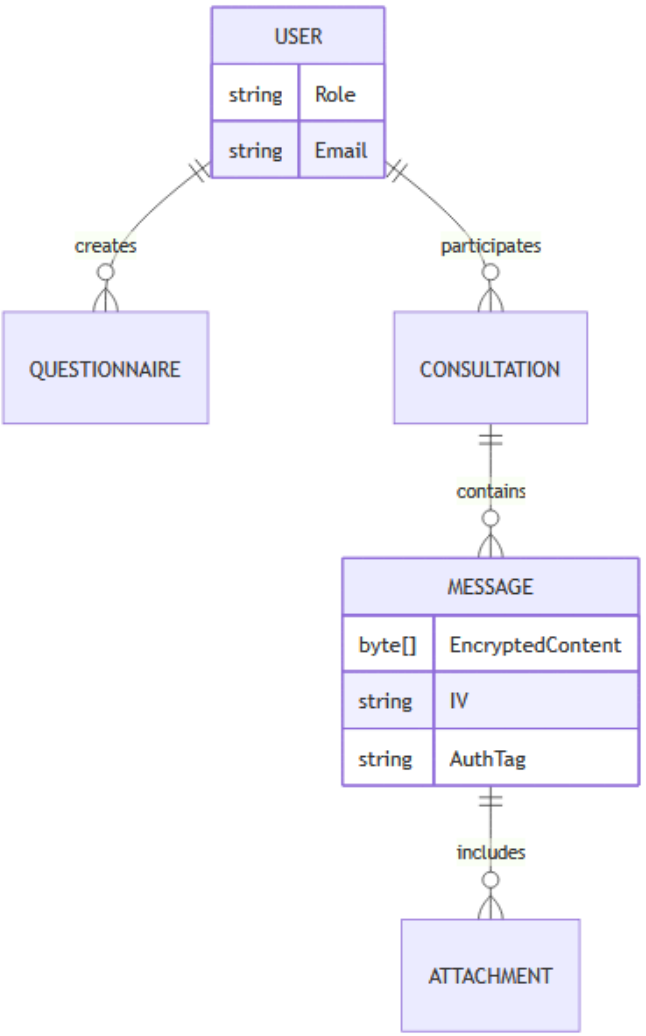

Рис 2.1 “Діаграма сутності-зв'язку (ER-діаграма) Бази Даних”
Джерело: розробка автора

Система використовує патерн CQRS (Command Query Responsibility Segregation), який забезпечує чітке розділення відповідальності між операціями читання (Queries) та операціями зміни стану системи (Commands). Таке розділення дозволяє незалежно оптимізувати обидва потоки: Command-частина може бути оптимізована для транзакційної цілісності та аудиту, тоді як Query-частина — для швидкості читання та кешування.

Commands – це об'єкти, що інкапсулюють намір змінити стан системи (наприклад, створення повідомлення, видалення користувача). Кожна команда обробляється відповідним Handler, який містить бізнес-логіку, виконує валідацію, працює з репозиторіями та забезпечує аудит. Для надсилання повідомлення використовується команда, яка приймає вже зашифровані клієнтом дані (Encrypted Content, IV, Auth Tag) як Base64-рядки, що є наріжним каменем архітектури наскрізного шифрування OstrohHelp.

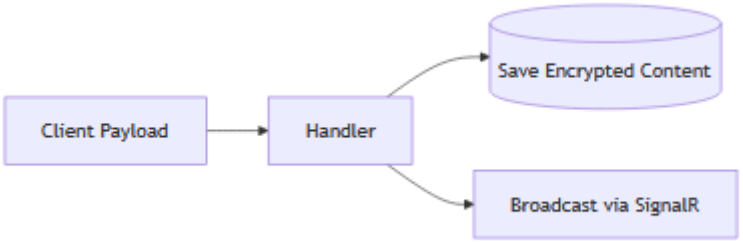

Рис 2.2 “Схема процесу зберігання зашифрованих даних від клієнта”
Джерело: розробка автора

Queries використовуються виключно для отримання даних і не змінюють стану системи. Вони можуть бути оптимізовані для роботи з read-only моделями даних (DTO) або кешуванням, що підвищує швидкість завантаження списків та історії чату, мінімізуючи обчислювальні витрати на швидке обслуговування великої кількості запитів на читання. Це дозволяє застосовувати додаткові оптимізації, такі як read-replica, та суттєво прискорює серіалізацію пакетів, оскільки на клієнт передаються лише необхідні зашифровані поля.

```
public class GetMessagesQuery : IRequest<List<MessageDto>>
{
    public Guid IdConsultation { get; set; }
}
public class GetMessagesQueryHandler : IRequestHandler<GetMessagesQuery,
List<MessageDto>>
{
    public async Task<List<MessageDto>> Handle(GetMessagesQuery request,
CancellationToken cancellationToken)
    {
        // 1. Обов'язкова перевірка членства в консультації
        await
_accessChecker.CheckConsultationMemberAccess(request.IdConsultation);
        // 2. Виконання оптимізованого запиту до БД
        var messages = await _messageRepository
            .GetMessagesForConsultationAsync(request.IdConsultation);
        // 3. Повернення списку DTO
        return _mapper.Map<List<MessageDto>>(messages);
    }
}
```

Лістинг 2.1. “Приклад Query”

Джерело: розробка автора

Таким чином, ключова відмінність Query від Command полягає у відсутності побічних ефектів та гарантії, що операція не призведе до зміни стану системи або бази даних. Це дозволяє команді розробки застосовувати додаткові оптимізації, такі як read-replica, і використовувати легко-вагомі моделі даних (DTO), мінімізуючи обчислювальні витрати на швидке обслуговування великої кількості запитів на читання, що критично для відображення списків консультацій та завантаження історії чату.

Використання DTO у Query-частині є важливим для конфіденційності, оскільки дозволяє передавати на клієнт лише необхідні зашифровані поля, уникаючи експорту зайвих даних із домену. Це також суттєво прискорює серіалізацію та десеріалізацію пакетів. Таким чином, уся критична логіка дешифрування повідомлень залишається виключно на клієнтській стороні, що є наріжним каменем

архітектури наскрізного шифрування OstrohHelp та виключає можливість доступу сервера до оригінального вмісту повідомлень.

2.2. Загальні характеристики API та Модель безпеки

Backend OstrohHelpApi використовує JWT Bearer Token для автентифікації, який видається після успішного входу через Google OAuth 2.0.

Таблиця 2.1 ”Основні технічні характеристики”

Параметр	Значення
Framework	ASP.NET Core 8.0
Базова URL	https://localhost:7123/api
Real-time	SignalR (WebSocket)
База даних	PostgreSQL 16
Автентифікація	JWT Bearer Token (термін дії 7 днів)
Архітектура	CQRS + Чиста архітектура

Доступ до ендпоїнтів суворо контролюється через три ролі: Student, Psychologist, HeadOfService.

Критична безпека в чатах забезпечується на рівні репозиторію та хабу через інтерфейс IConsultationAccessChecker. Цей механізм гарантує, що користувач може отримати доступ до консультації чи повідомлення лише у випадку, якщо він є її безпосереднім учасником (StudentId або PsychologistId).

2.3. Наскрізне Шифрування (E2EE) та Детермінована Генерація Ключа

Конфіденційність повідомлень забезпечується наскрізним шифруванням, де сервер OstrohHelpApi не бере участі у процесі дешифрування і зберігає лише зашифровані дані.

Використовується алгоритм AES-256-GCM (Advanced Encryption Standard in Galois/Counter Mode).

1. AES-256 - для криптографічної стійкості (256-бітний ключ).
2. GCM - забезпечує аутентифікацію даних через Аутентифікаційний тег (AuthTag, 16 байтів), що запобігає модифікації повідомлення під час передачі чи зберігання (Tamper Detection).

Backend зберігає три Base64-кодовані компоненти, отримані від клієнта:

1. EncryptedContent: Зашифрований текст.
2. IV (Initialization Vector): 12 унікальних байтів для кожного повідомлення.
3. AuthTag: 16 байтів для перевірки цілісності.

Ключ консультації (256-бітний) генерується динамічно за допомогою HKDF-SHA256 (HMAC-based Extract-and-Expand Key Derivation Function, RFC 5869). Це дозволяє уникнути зберігання ключа в базі даних.

Процес детермінованого виведення ключа:

1. Input Master Key (K): 256-бітний ключ, що зберігається в змінній середовища (.env) на сервері і ніколи не передається клієнту.
2. Input Salt (S): Унікальний ідентифікатор консультації (ConsultationId) використовується як сіль.
3. Output: HKDF-SHA256 детерміновано виводить 256-бітний Consultation Key.

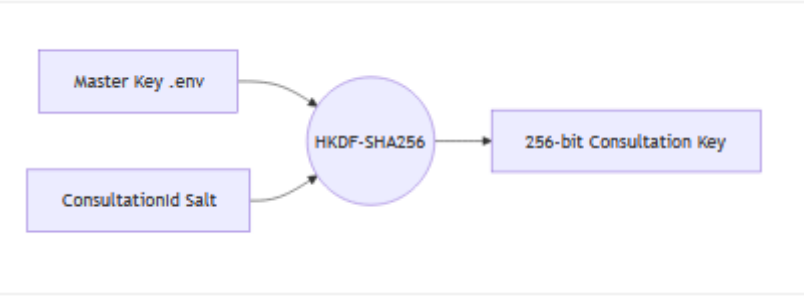

Рис 2.3 “Блок-схема Детермінованої Генерації Ключа”

Джерело: розробка автора

Завдяки детермінізму HKDF-SHA256, ключ для даної консультації завжди буде однаковим, що дозволяє клієнтам, які повторно приєднуються, миттєво відновити сесію чату та отримати доступ до всієї зашифрованої історії повідомлень, навіть у разі обриву Real-time з'єднання або перевипуску JWT токена.

Ключ передається виключно через захищене SignalR з'єднання при виклику клієнтом методу JoinConsultation(consultationId), що виключає перехоплення ключа через стандартні HTTP-запити REST API. Це забезпечує найвищий рівень безпеки каналу, оскільки ключові дані не проходять через менше захищені статичні ендпоїнти, а передаються лише після успішної перевірки прав доступу на сервері.

```
public async Task JoinConsultation(Guid consultationId)
{
    // 1. Генерація ключа
    byte[] consultationKey = _keyDerivationService.DeriveKeyForConsultation(
        masterKeyFromEnvironment,
        consultationId // Використовується як сіль (Salt)
    );
    string base64Key = Convert.ToBase64String(consultationKey);
    // 2. Приватна передача ключа клієнту (Clientes.Caller)
    await Clients.Caller.SendAsync("ReceiveConsultationKey", new {
        ConsultationId = consultationId,
        Key = base64Key, // BASE64, зберігається клієнтом у RAM
        Algorithm = "AES-256-GCM"
    });
}
```

Лістинг 2.2. “Приклад ChatHub”

Джерело: розробка автора

Клієнт отримує ключ у Base64, декодує його та зберігає виключно в оперативній пам'яті (RAM), що мінімізує ризик його компрометації. Такий підхід гарантує, що ключ шифрування ніколи не буде записаний на постійний носій (диск чи локальне сховище), що є фундаментальним принципом безпеки E2EE. Зберігання в RAM забезпечує його автоматичне знищення після завершення сесії або виходу користувача.

2.4. Real-Time Комунікація (SignalR Hub)

Для real-time функціональності використовується SignalR ChatHub. Це рішення повністю замінює застарілий та ресурсоємний підхід REST API Polling. Використання технології SignalR (WebSocket) гарантує миттєву доставку повідомлень, оскільки затримка складає менше 100 мілісекунд, та забезпечує постійне з'єднання замість численних коротких HTTP-запитів. Така оптимізація критично знижує навантаження на CPU сервера та значно підвищує ефективність використання батареї мобільних пристроїв.

Таблиця 2.2 “Порівняння Real-Time методів”

Параметр	REST API Polling (Старий)	SignalR (WebSocket, Новий)
Затримка	2-5 секунд (інтервал опитування)	Менше 100 мілісекунд
З'єднання	Багато коротких HTTP-запитів	Одне постійне з'єднання
Ефективність	Високе навантаження на CPU сервера	Дуже низьке навантаження
Масштабованість	Низька, через N+1 запитів	Висока (5000+ користувачів на хабі)

Ключові Real-Time функції ChatHub:

1. Key Distribution: Передача ключа шифрування. Ця подія є критично важливою, оскільки вона надає клієнту 256-бітний ключ для початку наскрізного шифрування (E2EE).
2. Message Delivery: Миттєва доставка нових повідомлень. Забезпечує негайну трансляцію зашифрованих пакетів між учасниками консультації.
3. Read Receipts: Оновлення статусу прочитання (MarkAsRead). Дозволяє учасникам чату бачити, що співрозмовник ознайомився з останніми надісланими повідомленнями в реальному часі.

4. Presence: Оновлення статусу присутності. Інформує клієнтів про те, чи перебуває інший учасник чату в мережі.

2.5. Детальний опис REST API Контрактів

Керує життєвим циклом користувача та його авторизацією.

Таблиця 2.3 ”Ендпоїнти контролера AuthController”

Ендпоїнт	Метод	Призначення	Обмеження
/google-login	POST	Автентифікація через Google OAuth, видача JWT	Анонімний
/all	GET	Отримання списку всіх користувачів	Психолог, Керівник
/User-Delete	DELETE	Видалення користувача з системи (hard delete)	Тільки Керівник служби
/User-Role-Update	PUT	Зміна ролі користувача	Тільки Керівник служби

Контролер AuthController керує авторизацією через Google OAuth та видачею JWT-токенів. Ендпоїнт /all дозволяє Психологу та Керівнику отримати список усіх користувачів.

```
// Request: PUT /api/auth/User-Role-Update
{
 "userId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
```

```
 "roleId": "00000000-0000-0000-0000-000000000002" // ID ролі
'Psychologist'
}
```

Лістинг 2.3. “Приклад запиту на зміну ролі”

Джерело: розробка автора

Працює з повідомленнями та вкладеннями, включаючи мультимедійні файли. Усі операції з даними повідомлень оперують виключно зашифрованим контентом, забезпечуючи End-to-End Encryption. Це стосується навіть завантаження файлів через Cloudinary, а сервер OstrohHelpApi зберігає лише зашифровані Base64-кодовані компоненти, унеможливлюючи доступ до оригінального вмісту повідомлень і вкладень, що гарантує абсолютну конфіденційність і виключає можливість компрометації чутливої інформації на серверному рівні.

Таблиця 2.4 “Ендпоїнти контролера MessageController”

Ендпоїнт	Метод	Призначення	Обмеження
/Recive	GET	Історія повідомлень (повертає зашифровані Base64-поля)	Членство в консультації
/Send	POST	Надсилання нового зашифрованого повідомлення	Авторизований Rate Limited
/BatchUpload	POST	Завантаження вкладень до Cloudinary (до 600 MB)	Авторизований Rate Limited

/Delete	DELETE	Логічне видалення повідомлення (Soft Delete та очищення вмісту)	Власник повідомлення
/Attachment/{attachmentId}	DELETE	Логічне видалення одного вкладення	Власник повідомлення

На відміну від фізичного видалення (hard delete), при Soft Delete:

1. Встановлюється прапор IsDeleted = true;
2. Поля EncryptedContent, Iv, AuthTag обнуляються (null) для забезпечення конфіденційності.

Керує життєвим циклом консультації, що є контейнером для чату, включаючи створення запису після прийняття анкети психологом та забезпечення безпечного обміну повідомленнями.

Створення консультації (POST /Accept-Questionnaire):

Цей ендпоїнт використовується психологом для прийняття анкети зі статусом "Pending" та створення нового запису в таблиці Consultations, автоматично пов'язуючи Студента та Психолога в новій безпечній консультаційній сесії.

```
// Request: POST /api/Consultations/Accept-Questionnaire
{
 "questionaryId": "8fa85f64-5717-4562-b3fc-2c963f66afab",
 "psychologistId": "2fa85f64-5717-4562-b3fc-2c963f66afa1",
 "scheduledTime": "T10:00:00Z"
}
```

Лістинг 2.4.

Джерело: розробка автора

Ця дія оновлює статус анкети на "Accepted" та логується як "ConsultationCreated", оскільки з точки зору системи це є ключовим моментом, що

ініціює життєвий цикл нової консультації. Це гарантує, що факт створення чату між Студентом і Психологом буде зафіксовано в системі аудиту з точною відміткою часу, а подальші операції (наприклад, надсилання повідомлень) будуть асоційовані з цим новим ID консультації.

Таблиця 2.5 “Ендпоїнти контролера QuestionnaireController”

Ендпоїнт	Метод	Призначення	Обмеження
/Create-Questionnaire	POST	Створення нової анкети (статус "Pending")	Авторизований
/all	GET	Список усіх анкет зі статусом "Pending" (для психологів)	Психолог, Керівник
/Update-Status	PUT	Зміна статусу анкети (Accepted/Rejected)	Психолог, Керівник

2.6. Аудит та Обмеження частоти запитів (Rate Limiting)

Система OstrohHelpApi використовує інфраструктуру Audit Logging для реєстрації всіх критичних дій користувачів. Логування вбудоване в обробники CQRS Commands та ChatHub.

Приклади логованих дій:

1. MessageSent: Надсилання повідомлення (REST/SignalR).
2. UserRoleUpdated: Зміна ролі адміністратором.
3. ConsultationCreated: Створення консультації.
4. AttachmentUploaded: Завантаження вкладень з деталями (кількість файлів, загальний розмір).

Усі записи зберігаються в таблиці audit_logs і включають user_id, action, resource та JSON-об'єкт details для додаткової інформації.

Для захисту від атак та перевантаження системи використовується механізм Rate Limiting на основі алгоритму Token Bucket. Обмеження застосовується на критичні ендпоїнти, такі як POST /api/Message/Send та POST /api/Message/BatchUpload.

У разі перевищення ліміту сервер повертає HTTP-статус 429 Too Many Requests та заголовок Retry-After, який вказує, через який час клієнт може повторити запит. Такий підхід, реалізований на основі алгоритму Token Bucket, є необхідним заходом для захисту критичних ендпоїнтів від DoS-атак та гарантує стабільність роботи системи, не дозволяючи одному користувачеві чи боту перевантажити ресурс.

```
// Приклад відповіді при перевищенні ліміту
HTTP/1.1 429 Too Many Requests
Retry-After: 60
// ...
{
 "error": "Rate limit exceeded",
 "message": "Too many requests to SendMessage. Please try again
in 60 seconds.",
 "retryAfter": 60
}
```

Лістинг 2.5.

Джерело: розробка автора

Приклад ілюструє приклад стандартизованої відповіді від backend-системи OstrohHelpApi у випадку, коли клієнт перевищує встановлений ліміт частоти запитів (Rate Limiting).

Цей механізм захисту застосовується на критичних ендпоїнтах, зокрема POST /api/Message/Send та POST /api/Message/BatchUpload, для захисту системи від атак типу "Відмова в обслуговуванні" (DoS) та гарантування її стабільності.

РОЗДІЛ 3. КЛІЄНТСЬКА ЧАСТИНА: РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ (FLUTTER)

Розділ 3 детально описує мобільний застосунок OstrohHelpApp, розроблений на крос-платформному фреймворку Flutter. Цей застосунок є клієнтським інтерфейсом для всіх трьох категорій користувачів -Student, Psychologist, HeadOfService -і забезпечує реалізацію наскрізного шифрування (E2EE) та real-time комунікації на мобільному пристрої.

3.1. Технологічний стек та Архітектура

Вибір Flutter (Dart 3.x) для мобільного клієнта забезпечує єдину кодову базу для нативних застосунків на Android та iOS. Це значно прискорює розробку та уніфікує користувацький досвід (UX).

Таблиця 3.1

Компонент	Технологія / Пакет	Призначення
Управління станом	Bloc / Cubit	Управління станом, розділення логіки та UI
Real-Time	signal_netcore	Інтеграція з SignalR ChatHub backend
Автентифікація	google_sign_in	Обробка Google OAuth 2.0 на мобільному пристрої
Шифрування	pointycastle	Реалізація криптографічних алгоритмів.
HTTP-клієнт	http	Виконання REST API запитів на стороні клієнту

Для організації коду та підтримки його масштабованості використовується архітектурний патерн Bloc/Cubit. Цей патерн дозволяє інкапсулювати бізнес-логіку у Bloc або Cubit, які приймають події (Events) та випускають нові стани (States), на які реагує UI.

Приклад організації модулів:

1. Presentation Layer (UI): Сторінки (ChatPage, ConsultationListPage) та віджети, які викликають методи в Cubit/Bloc та відображають поточний стан.
2. Business Logic Layer (Bloc/Cubit): Містить логіку, керує станом (наприклад, ChatCubit зберігає поточний ключ шифрування та список повідомлень).
3. Data Layer (Services/API Clients): Містить сервіси (AuthApiService, MessageApiService), які взаємодіють з backend через REST API або SignalR.

3.2. Real-Time комунікація та E2EE

Ключова функціональність чату реалізується через ChatService, який обгортає пакет signalr_netcore та відповідає за управління WebSocket-з'єднанням, передачу ключа шифрування та обробку real-time подій.

Після успішного підключення до ChatHub через Real-time з'єднання, клієнт зобов'язаний отримати унікальний 256-бітний ключ шифрування для даної консультації, який є критично важливим для наскрізного шифрування (E2EE).

Послідовність дій на Flutter-клієнті:

1. Виклик методу: Клієнт викликає SignalR метод connection.invoke('JoinConsultation', args: [consultationId]).
2. Отримання події: Клієнт підписується на подію connection.on('ReceiveConsultationKey', (args) { ... }).

3. Зберігання: Отриманий ключ у форматі Base64 декодується в байтовий масив та зберігається у RAM-пам'яті застосунку (наприклад, у ChatCubit), ігноруючи локальне сховище (localStorage).

Для шифрування та дешифрування повідомлень використовується Dart-бібліотека pointycastle. Клієнт виконує шифрування безпосередньо перед надсиланням повідомлення на backend.

Процес шифрування повідомлення:

1. Підготовка: З RAM-пам'яті витягується 256-бітний Consultation Key.
2. Генерація IV: Генерується унікальний 12-байтовий Initialization Vector (IV) для поточного повідомлення.
3. Шифрування: Алгоритм AES-256-GCM застосовується до вихідного тексту (plaintext).
4. Складання пакету: На виході отримуємо: зашифрований контент (EncryptedContent) IV та AuthTag. Всі три компоненти кодуються в Base64 та відправляються на сервер через REST API або SignalR.

Приклад логіки шифрування (Dart/pointycastle):

```
// Функція шифрування
Uint8List encryptMessage(Uint8List plaintext, Uint8List key, Uint8List iv)
{
  final cipher = GcmBlockCipher(AesEngine());
  // Налаштування режиму шифрування
  cipher.init(
    true, // true = шифрування
    AEADParameters(
      KeyParameter(key),
      16 * 8, // Розмір AuthTag у бітах (16 bytes)
      iv,
      null, // Додаткові аутентифіковані дані (AAD)
    )
  );
  // Обробка та повернення шифротексту та AuthTag
```

```
  final encryptedContent = cipher.process(plaintext);
  final authTag = cipher.getMac();
  // Клієнт об'єднує та кодує це для відправки на сервер
  return encryptedContent;
}
```

Лістинг 3.1.

Джерело: розробка автора

Процес дешифрування повідомлення:

1. При отриманні історії (GET /Recive) або real-time повідомлення (ReceiveMessage), клієнт декодує Base64-компоненти.
2. Використовує Consultation Key, IV та AuthTag для дешифрування.
3. Якщо AuthTag невалідний (дані змінені), виникає виняток InvalidCipherTextException, що захищає користувача від перегляду скомпрометованого контенту.

Для відображення статусу Online/Offline співрозмовника використовується SignalR-подія UserStatusChanged.

PresenceService (в Flutter) ініціалізується при успішному вході користувача і підтримує постійне з'єднання. При зміні статусу користувача backend надсилає подію:

```
// Приклад Receive Event від ChatHub
{
 "userId": "1fa85f64-5717-4562-b3fc-2c963f66afa0",
 "isOnline": true
}
```

Лістинг 3.2.

Джерело: розробка автора

Flutter-застосунок оновлює стан (наприклад, у OnlineUsersNotifier), який відображається на ConsultationListPage та Chat Page.

3.3. Основні функціональні модулі та User Flows

Мобільний застосунок розділений на функціональні модулі, які обслуговують потреби кожної ролі. Це забезпечує чітку логіку для User Flows, від створення студентом анкети та доступу до чату до прийняття запиту психологом, а також керування користувачами через адміністративну панель для керівника служби, гарантуючи, що кожен користувач має доступ лише до релевантного функціоналу та даних, що забезпечує як безпеку, так і кращий користувацький досвід та цілісність даних.

Основний потік зосереджений на створенні анкети, перегляді консультацій та чаті.

Таблиця 3.2

Сторінка	Призначення	Ключові API-виклики
LoginPage	Вхід через Google OAuth	POST /api/auth/google-login
QuestionnairePage	Створення нової анкети	POST /api/Questionnaire/Create-Questionnaire
ConsultationListPage	Список консультацій, перегляд online-статусу	GET /api/Consultations/Get-All-Consultations-By-UserId/{Id}
ChatPage	Real-time чат з E2EE	GET /api/Message/Recive; SignalR - SendMessage

Для адміністративних ролей доступна AdminPanelPage, звідки можна перейти до управління користувачами, ролями та анкетами, а також виконувати критичні операції, такі як видалення користувачів або зміна їхнього курсу.

AdminQuestionnairesPage (Прийняття анкет):

1. Психолог отримує список усіх анкет зі статусом *Pending* (GET /api/questionnaire/all).
2. При прийнятті анкети створюється консультація:
   POST /api/Consultations/Accept-Questionnaire.

AdminUsersPage (Управління користувачами):

1. Керівник служби може отримати повний список користувачів (
   GET /api/auth/all).
2. Критичні операції видалення та зміни ролі доступні лише для цієї ролі і логуються на backend.

Завантаження файлів реалізовано через MessageApiService.batchUpload(), який викликає /api/Message/BatchUpload.

Потік завантаження файлів:

1. Flutter-клієнт збирає файли у формат multipart/form-data.
2. Клієнт надсилає запит на backend.
3. Backend завантажує файли до Cloudinary та генерує прев'ю-посилання.
4. Клієнт отримує ці посилання та відображає прев'ю в чаті, використовуючи спеціальні віджети Flutter.
5. Цей механізм завантаження вкладень є частиною забезпечення End-to-End Encryption, оскільки лише зашифровані посилання зберігаються в БД, а не сам контент.
6. Backend логує факт завантаження вкладень як *AttachmentUploaded* в таблицю *audit_logs*, з деталями про кількість файлів та загальний розмір.
7. Ендпоїнт /api/Message/BatchUpload захищений механізмом Rate Limiting на основі алгоритму Token Bucket для запобігання перевантаженню системи.
8. Обробка прев'ю-посилань на Flutter-клієнті дозволяє користувачам

бачити мініатюри зображень чи відеопостери, не завантажуючи повний файл до моменту його перегляду.

9. Для веб-версії (React) використовується той самий ендпоїнт, але обробка вкладень інтегрована у компонент *ChatWindow*.

3.4. Діаграма взаємодії Flutter та Backend

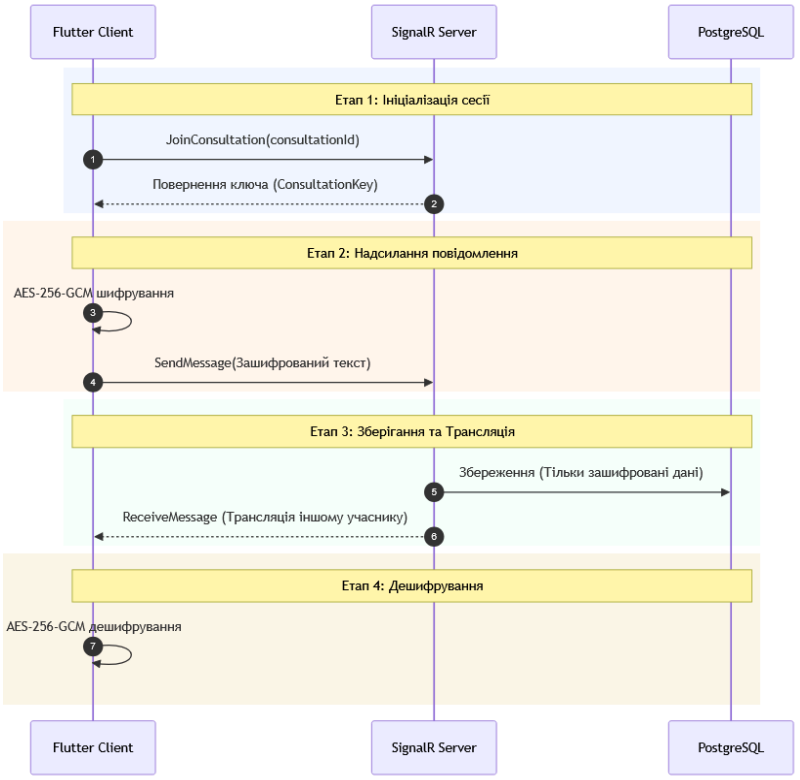

Рис 3.1

Джерело: розробка автора

Наступна діаграма ілюструє основний потік даних між Flutter-клієнтом та backend, підкреслюючи роль SignalR та клієнтського шифрування E2EE.

РОЗДІЛ 4. КЛІЄНТСЬКА ЧАСТИНА: РЕАЛІЗАЦІЯ ВЕБ-ЗАСОСУНКУ (REACT)

Розділ 4 детально описує веб-версію інформаційної системи OstrohHelp Web, розроблену з використанням бібліотеки React 19 та мови TypeScript. Веб-застосунок, реалізований як односторінковий додаток (SPA), забезпечує повний функціонал для всіх трьох ролей користувачів, з особливим акцентом на адміністративній панелі для Керівника служби та Психолога.

4.1. Технологічний стек та архітектура React-застосунку

Вибір React забезпечує високу інтерактивність, швидку розробку компонентів та доступ до широкої екосистеми інструментів, що є важливим для створення професійного веб-порталу.

Таблиця 3.2 “Технології використані в Веб версії”

Компонент	Технологія / Пакет	Призначення
Фреймворк	React	Створення декларативного та компонентного інтерфейсу
Управління станом	React Context / Redux Toolkit	Управління глобальними даними.
HTTP-клієнт	Axios	Виконання REST API запитів до OstrohHelpApi
Real-Time	@microsoft/signalr	Клієнтська інтеграція з SignalR ChatHub

Шифрування (E2EE)	crypto-js	Реалізація AES-256-GCM та Base64 кодування на JavaScript
Стилізація	Tailwind CSS	Утилітарно-орієнтований CSS для швидкої та адаптивної розробки UI
Маршрутизація	react-router-dom	Керування маршрутами та відображенням різних сторінок (SPA)

React-застосунок побудований на принципах компонентної архітектури. Організація коду передбачає розділення на Smart/Container Components та Dumb/Presentational Components. Взаємодія з backend інкапсульована у Services, а глобальне сховище стану забезпечується через Context/Store.

Організація коду:

1. Components: Розділені на Smart/Container Components (містить логіку) та Dumb/Presentational Components (відповідає лише за відображення).
2. Services: Інкапсулюють взаємодію з backend.
3. Context/Store: Глобальне сховище стану (наприклад, AuthContext зберігає JWT токен та профіль користувача).

4.2. Реалізація наскрізного шифрування (E2EE) у JavaScript

Веб-клієнт повинен повторити точний криптографічний потік, реалізований у Flutter та визначений backend. Для цього використовуються стандартні JavaScript-бібліотеки.

Ключ шифрування не зберігається у localStorage або sessionStorage. Він зберігається лише в оперативній пам'яті (RAM) веб-сторінки, що досягається зберіганням ключа у стані React Context, який не є персистентним.

Потік:

1. Клієнт ініціює з'єднання SignalR та викликає connection.invoke.
2. Сервер надсилає ключ через подію ReceiveConsultationKey .
3. React-компонент декодує Base64 та зберігає ключ у ChatContext.Key.

```
connection.on('ReceiveConsultationKey', (data) => {
    // Декодування Base64 ключа
    const key = CryptoJS.enc.Base64.parse(data.Key);
    // Зберігання ключа в оперативній пам'яті (state)
    setChatState({ ...chatState, consultationKey: key });
    console.log('Encryption key received and stored in RAM.');
});
```

Лістинг 4.1. Фрагмент логіки обробки ключа в ChatContext

Джерело: розробка автора

Для шифрування використовується бібліотека crypto-js, яка підтримує надійний алгоритм AES-256-GCM для реалізації наскрізного шифрування (E2EE) на стороні веб-клієнта (React). Це гарантує, що повідомлення шифруються локально, а ключ не зберігається на диску, оскільки він розміщується лише в оперативній пам'яті (RAM) веб-сторінки, зазвичай у стані React Context. Вибір GCM режиму також забезпечує аутентифікацію даних через AuthTag, гарантуючи цілісність повідомлення. Це виключає можливість доступу сервера чи проміжного посередника до відкритого вмісту. Після завершення сесії або виходу користувача цей ключ автоматично знищується.

Шифрування (перед надсиланням):

1. Надійно генерується випадковий 12-байтовий IV (Initialization Vector).
2. Вихідний текст (оригінальний) (plaintext) шифрується за допомогою ключа, IV та алгоритму AES-256-GCM.
3. На виході отримуємо: зашифрований текст (EncryptedContent) та AuthTag.
4. Всі три компоненти кодуються в Base64.

```
const encryptMessage = (plaintext, key) => {
    const iv = CryptoJS.lib.WordArray.random(12); // 12 bytes IV
    const encrypted = CryptoJS.AES.encrypt(plaintext, key, {
        iv: iv,
        mode: CryptoJS.mode.GCM,
        padding: CryptoJS.pad.NoPadding,
        tagLength: 128 // 16 bytes AuthTag
    });
    return {
        encryptedContent:
encrypted.ciphertext.toString(CryptoJS.enc.Base64),
        iv: iv.toString(CryptoJS.enc.Base64),
        authTag: encrypted.tag.toString(CryptoJS.enc.Base64)
    };
};
```

Лістинг 4.2. Приклад функції шифрування

Джерело: розробка автора

Дешифрування (при отриманні):

1. Отримані Base64-рядки декодуються.
2. Використовується збережений ключ та отримані IV і AuthTag для відновлення plaintext.
3. Якщо AuthTag не збігається, crypto-js поверне помилку або невалідний контент, що захищає від скомпрометованих повідомлень.

4.3. Реалізація Real-Time комунікації (SignalR)

Веб-клієнт використовує офіційну бібліотеку @microsoft/signalr для підключення до ChatHub.

Ініціалізація з'єднання: З'єднання встановлюється в корені застосунку або у ChatContext, з передачею JWT токена.

Обробка подій: React-компоненти підписуються на події SignalR через connection.on(...) і оновлюють стан програми, викликаючи перемальовування UI.

1. ReceiveMessage: Отримує зашифрований пакет, дешифрує його локально та додає до списку повідомлень, забезпечуючи наскрізне шифрування (E2EE).
2. UserStatusChanged: Оновлює індикатор присутності (online/offline) на ConsultationListPage, інформуючи про активність співрозмовника.
3. ReceiveConsultationKey: Обробляє критично важливу подію передачі 256-бітного ключа шифрування, який зберігається виключно в RAM-пам'яті клієнта.
4. MessageRead / ReceiveMarkedAsRead: Забезпечує оновлення статусу прочитання повідомлень в реальному часі.

Реалізація real-time зв'язку через SignalR замінила застарілий та ресурсоємний підхід REST API Polling, мінімізуючи затримку до менше 100 мілісекунд.

4.4. Ключові функціональні модулі веб-версії

Веб-версія OstrohHelp Web є основним інструментом для Психологів та Керівників служби завдяки розширеній адміністративній функціональності.

Після успішного входу JWT токен зберігається у стані AuthContext. Маршрутизація (react-router-dom) використовує Protected Routes для обмеження доступу:

1. Обмеження за авторизацією: <Route element={<RequireAuth />} />
2. Обмеження за роллю:
   <Route element={<RequireRole allowedRoles={['Psychologist', 'HeadOfService']} />} />

Цей модуль є центральним для ролей Психолога та Керівника служби. Він надає розширену функціональність для управління системою, включаючи огляд Questionnaire Management (прийняття очікуючих анкет) та User Management (зміна ролей, видалення користувачів), що є критично важливим для підтримки безпеки та організації роботи служби психологічної підтримки.

Questionnaire Management (для Психологів):

1. Отримання очікуючих анкет: ET /api/Questionnaire/all
2. Відображення: Анкети відображаються у таблиці з можливістю сортування за датою подачі та фільтрації.
3. Дія: Кнопка "Прийняти" яка викликає POST /Accept-Questionnaire.

User Management (для Керівника служби):

1. Перегляд усіх користувачів: GET /api/auth/all
2. Можливість зміни ролі: Модальне вікно для виклику PUT /api/auth/User-Role-Update та вибору нової ролі з GET /api/Role .
3. Видалення користувача: Кнопка "Видалити" (DELETE /api/auth/User-Delete) з обов'язковим попередженням та підтвердженням.

ChatWindow є ключовим компонентом, який керує:

1. Завантаження історії: Виклик GET /api/Message/Recive з подальшим послідовним дешифруванням кожного повідомлення.
2. Обробка вводу: Збір тексту, локальне шифрування (з IV та AuthTag) та надсилання через SignalR SendMessage.
3. Вкладення: Інтерфейс для вибору та завантаження файлів (POST /api/Message/BatchUpload).

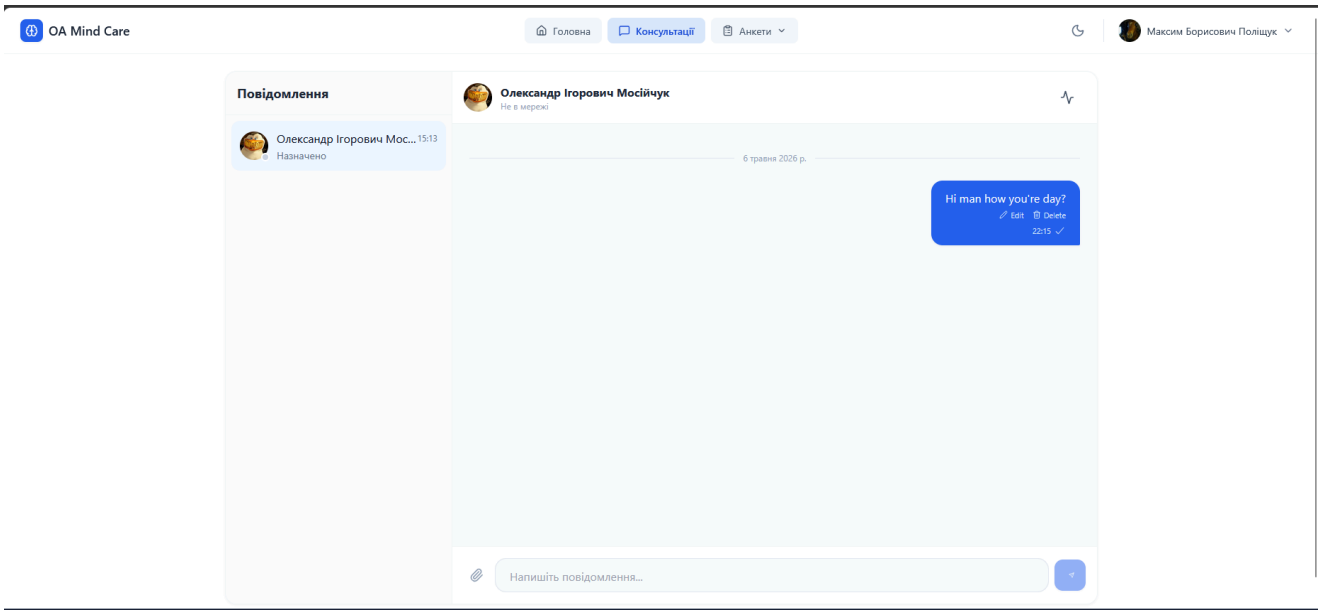

Рис 4.1 “Інтерфейс чату з сайту OstrohHelp”

Джерело: розробка автора

4.5. Інтеграція з REST API (Axios)

Для забезпечення консистентності запитів та обробки помилок використовується HTTP-клієнт Axios. Усі запити автоматично додають JWT токен у заголовок (Authorization: Bearer <JWT_TOKEN>).

Обробка HTTP-помилок:

Axios перехоплювачі (Interceptors) використовуються для централізованої обробки статус-кодів:

1. 401 Unauthorized: Автоматичне перенаправлення на LoginPage.
2. 403 Forbidden: Відображення повідомлення "У вас немає прав доступу" та, можливо, перенаправлення на 403-сторінку.
3. 429 Too Many Requests: Відображення попередження користувачу про перевищення ліміту та час, через який можна повторити спробу (з заголовка Retry-After).

Реалізація системи на крос-платформних технологіях Flutter та React дозволила охопити як мобільну, так і веб-аудиторію, зберігаючи при цьому єдиний високий стандарт безпеки (E2EE) та якості сервісу (Real-time SignalR). Flutter забезпечує нативну продуктивність на мобільних пристроях, тоді як React надає гнучкість та функціональність, необхідні для роботи з адміністративними панелями та звітністю.

ВИСНОВОК

Кваліфікаційна робота присвячена проєктуванню, розробці та детальному обґрунтуванню архітектури багатокомпонентної інформаційної системи психологічної підтримки OstrohHelp.

Мета, що полягала в узагальненні теоретичних знань та практичних навиків, а також їхньому застосуванні для створення багатокомпонентної інформаційної системи, була повністю досягнута. У процесі роботи були вирішені всі поставлені завдання.

Обґрунтовано архітектуру безпеки (Розділ 1). Досягнуто високого рівня конфіденційності даних шляхом реалізації наскрізного шифрування (E2EE) для всіх повідомлень у чаті. Для цього використано криптографічний алгоритм AES-256-GCM, який забезпечує не лише конфіденційність, але й аутентифікацію даних (AuthTag), запобігаючи їх модифікації. Ключ шифрування для кожної консультації генерується детерміновано за допомогою HKDF-SHA256 та передається клієнтам виключно через захищений канал SignalR, ніколи не зберігаючись у базі даних.

Розроблено масштабований backend (Розділ 2). Backend-частина системи OstrohHelpApi реалізована на ASP.NET Core 8.0 із застосуванням архітектурного патерну CQRS (Command Query Responsibility Segregation). Це дозволило ефективно розділити логіку читання (Queries) та запису (Commands), забезпечуючи незалежну оптимізацію та масштабованість. Також було реалізовано деталізований контроль доступу на основі ролей (Student, Psychologist, HeadOfService) та впроваджено систему Audit Logging і Rate Limiting для моніторингу та захисту критичних ендпоїнтів.

Забезпечено Real-Time комунікацію (Розділ 2). Замість застарілого підходу REST API Polling було інтегровано технологію SignalR (WebSocket), що забезпечило миттєву доставку повідомлень (затримка менше 100 мс) та значно зменшило навантаження на сервер і підвищило ефективність використання батареї мобільних пристроїв.

Створено крос-платформну клієнтську частину, описану в Розділах 3 та 4. Для забезпечення максимальної доступності були розроблені два повністю синхронізовані клієнтські застосунки, які взаємодіють з єдиним backend. Це мобільний клієнт на Flutter, що гарантує нативну продуктивність на платформах Android та iOS, та використовує патерн Bloc/Cubit для управління станом, а також веб-клієнт на React 19, який надає повноцінний інтерфейс для доступу через браузер, включаючи розширений функціонал адміністративної панелі для ролей Psychologist та HeadOfService.

Таким чином, у ході кваліфікаційної роботи була розроблена сучасна, безпечна та багатокомпонентна інформаційна система OstrohHelp. Це успішно демонструє вміння застосовувати технології програмування, що відповідають вимогам (СК8), та розуміння концепцій інформаційної безпеки (ПР15, СК5, СК7).

Практична цінність роботи полягає у створенні готового до впровадження програмного продукту, який може бути використаний для забезпечення конфіденційної психологічної підтримки здобувачів вищої освіти, а також у детальному обґрунтуванні та документуванні архітектурних рішень, що можуть служити основою для подальшого розвитку та масштабування системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. .NET Core Versions [Електронний ресурс] - Режим доступу до ресурсу: 01.02.2025: https://versionsof.net/core/
2. React Documentation [Електронний ресурс] - Режим доступу до ресурсу: 15.06.2025: https://react.dev/
3. SignalR in ASP.NET Core & React [Електронний ресурс] - Режим доступу до ресурсу: 26.05.2025: https://www.roundthecode.com/dotnet-tutorials/using-signalr-in-asp-net-core-react-to-send-messages
4. MediatR for CQRS [Електронний ресурс] - Режим доступу до ресурсу: 27.06.2025: https://github.com/jbogard/MediatR
5. Clean Architecture CQRS Event Sourcing [Електронний ресурс] - Режим доступу до ресурсу: 01.07.2025: https://github.com/jeangatto/ASP.NET-Core-Clean-Architecture-CQRS-Event-Sourcing
6. React Context API [Електронний ресурс] - Режим доступу до ресурсу: 06.07.2025: https://react.dev/reference/react/createContext
7. Axios [Електронний ресурс] - Режим доступу до ресурсу: 09.07.2025: https://axios-http.com/docs/intro
8. Cloudinary Documentation [Електронний ресурс] - Режим доступу до ресурсу: 29.07.2025: https://cloudinary.com/documentation
9. Scalable .NET 8 CQRS Web API [Електронний ресурс] - Режим доступу до ресурсу: 25.08.2025: https://dev.to/iamcymentho/building-a-scalable-net-8-web-api-clean-architecture-cqrs-jwt-postgresql-redis-a-5bpj
10. Google OAuth 2.0 Docs [Електронний ресурс] - Режим доступу до ресурсу: 02.09.2025: https://developers.google.com/identity/protocols/oauth2
11. CQRS MediatR ASP.NET Core Guide [Електронний ресурс] - Режим доступу до ресурсу: 03.09.2025:

https://codewithmukesh.com/blog/cqrs-and-mediatr-in-aspnet-core/

12. Flutter E2EE Package [Електронний ресурс] - Режим доступу до ресурсу: 16.09.2025: https://pub.dev/packages/flutter_e2e
13. SignalR Wiki [Електронний ресурс] - Режим доступу до ресурсу: 19.09.2025: https://github.com/signalr/signalr/wiki
14. ASP.NET Core 8.0 Documentation [Електронний ресурс] - Режим доступу до ресурсу: 28.09.2025: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0
15. MediatR Library [Електронний ресурс] - Режим доступу до ресурсу: 30.10.2025: https://github.com/jbogard/MediatR
16. AES-256-GCM Introduction [Електронний ресурс] - Режим доступу до ресурсу: 16.10.2025: https://www.cincopa.com/learn/introduction-to-aes-gcm-mode-and-its-advantages
17. Flutter Secure Storage [Електронний ресурс] - Режим доступу до ресурсу: 22.10.2025: https://pub.dev/packages/flutter_secure_storage
18. ASP.NET Core Release Notes 8.0 [Електронний ресурс] - Режим доступу до ресурсу: 25.10.2025: https://github.com/dotnet/AspNetCore.Docs/blob/main/aspnetcore/release-notes/aspnetcore-8.0.md
19. E2EE in Flutter Chat [Електронний ресурс] - Режим доступу до ресурсу: 08.11.2025: https://stackoverflow.com/questions/65956155/how-to-have-end-to-end-encryption-in-flutter-chat-app
20. SignalR Authentication Tutorial [Електронний ресурс] - Режим доступу до ресурсу: 13.11.2025: https://www.youtube.com/watch?v=8g34hhmiI_Q
21. CQRS Mediator .NET 8 Web API [Електронний ресурс] - Режим доступу до ресурсу: 15.11.2025: https://www.linkedin.com/pulse/implementing-cqrs-mediator-pattern-net-8-web-api-edson-martinez-reyae

22. Dart Documentation [Електронний ресурс] - Режим доступу до ресурсу: 21.11.2025: https://dart.dev/guides
23. SignalR JWT Bearer Auth [Електронний ресурс] - Режим доступу до ресурсу: 11.12.2025: https://stackoverflow.com/questions/75440299/how-to-use-jwtbearer-to-authenticate-signalr-client
24. Rate Limiting ASP.NET Core [Електронний ресурс] - Режим доступу до ресурсу: 20.12.2025: https://learn.microsoft.com/en-us/aspnet/core/performance/rate-limit?view=aspnetcore-8.0
25. Flutter Documentation [Електронний ресурс] - Режим доступу до ресурсу: 05.01.2026: https://docs.flutter.dev/
26. Event Sourcing CQRS PostgreSQL [Електронний ресурс] - Режим доступу до ресурсу: 07.01.2026: https://github.com/cjrequena/event-sourcing-cqrs-postgresql-sample
27. Virgil E3Kit Flutter [Електронний ресурс] - Режим доступу до ресурсу: 10.01.2026: https://github.com/cardoso/virgil-e3kit-flutter
28. Crypto-js [Електронний ресурс] - Режим доступу до ресурсу: 18.01.2026: https://www.npmjs.com/package/crypto-js
29. Flutter Bloc Documentation [Електронний ресурс] - Режим доступу до ресурсу: 24.01.2026: https://bloclibrary.dev/
30. CQRS in ASP.NET Core [Електронний ресурс] - Режим доступу до ресурсу: 31.01.2026: https://www.reddit.com/r/dotnet/comments/gpiyeq/cqrs_pattern_in_aspnet_core_31_explained/
31. SignalR JWT Claims Security [Електронний ресурс] - Режим доступу до ресурсу: 04.02.2026: https://pub.dev/packages/flutter_secure_storage
32. SignalR JWT Authentication [Електронний ресурс] - Режим доступу до ресурсу: 06.02.2026: https://github.com/mikebridge/IdentityServer4SignalR
33. SignalR Documentation (@microsoft/signalr) [Електронний ресурс] -

Режим доступу до ресурсу: 11.02.2026: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction

34. Tailwind CSS Documentation [Електронний ресурс] - Режим доступу до ресурсу: 22.02.2026: https://tailwindcss.com/docs
35. Practical ASP.NET Core samples [Електронний ресурс] - Режим доступу до ресурсу: 07.03.2026: https://github.com/dodyg/practical-aspnetcore/blob/net8.0/README.md
36. Cloudinary Flutter SDK [Електронний ресурс] - Режим доступу до ресурсу: 17.03.2026: https://cloudinary.com/documentation/flutter_integration
37. PostgreSQL CQRS Read/Write Splitting [Електронний ресурс] - Режим доступу до ресурсу: 27.03.2026: https://kitemetric.com/blogs/supercharge-your-postgresql-database-with-cqrs-read-write-splitting-for-blazing-fast-queries
38. PointyCastle [Електронний ресурс] - Режим доступу до ресурсу: 03.04.2026: https://pub.dev/packages/pointycastle
39. Encrypt Package Flutter [Електронний ресурс] - Режим доступу до ресурсу: 09.04.2026: https://pub.dev/packages/encrypt
40. JWT in ASP.NET Core [Електронний ресурс] - Режим доступу до ресурсу: 12.04.2026: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity-api-authorization?view=aspnetcore-8.0
41. CQRS in .NET Microservices [Електронний ресурс] - Режим доступу до ресурсу: 18.04.2026: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/cqrs-microservice-reads
42. E2EE Flutter Secure Messaging [Електронний ресурс] - Режим доступу до ресурсу: 29.04.2026: https://connectycube.com/2024/08/08/implementing-end-to-end-encryption-in-flutter-chat-apps-a-guide-to-secure-messaging/

43. HKDF-SHA256 RFC 5869 [Електронний ресурс] - Режим доступу до ресурсу: 04.05.2026: https://datatracker.ietf.org/doc/html/rfc5869
44. CQRS Теорія ASP.NET Core [Електронний ресурс] - Режим доступу до ресурсу: 08.05.2026: https://habr.com/ru/articles/543828/
45. PostgreSQL Documentation [Електронний ресурс] - Режим доступу до ресурсу: 09.05.2026: https://www.postgresql.org/docs/
46. CQRS Architecture .NET Core 8 [Електронний ресурс] - Режим доступу до ресурсу: 23.05.2025: https://www.codingvila.com/2024/12/creating-cqrs-architecture-in-net-core-8.html

ДОДАТКИ