МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітньо-професійна програма: «Безпека комп'ютерних систем і мереж»

Група: 4КБ-02

Дипломний проект

здобувача освіти денної форми навчання
КБ.02.26.000.ДП

ШУМІЛО
МАКСИМА СЕРГІЙОВИЧА

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: «**Безпека комп'ютерних систем і мереж»**

Група: **4КБ-02**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Розробка застосунку для обміну повідомленнями з можливістю шифрування

Проектний матеріал складається з пояснювальної записки на 80 сторінках та графічного (презентаційного) матеріалу на 15 аркушах (слайдах)

Дипломник _____ (Шуміло М.С.)

Керівник _____ (Нестеренко В.Д.)

Консультанти:

з економічного розділу _____ (Канський М.Ю.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Краснокутська К.Г.)

Захист «21» червня 2025 р. Протокол ЕК № 2

Оцінка ЕК 4 (добре) / 85.

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Безпека комп'ютерних систем і мереж»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"19" 05 2025 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Шуміло Максиму Сергійовичу*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка застосунку для обміну повідомленнями з можливістю шифрування*

затверджена наказом по коледжу від "*14*" *листопада* 2024р. № *246*

2. Термін здачі закінченого проекту

3. Вихідні данні до проекту *1. Передбачити сучасний UI та дружній UX. 2. Обрати найоптимальніший фреймворк для реалізації поставленої задачі. 3. Визначити та застосувати формули для розрахунку електроенергії. 4. Передбачити будування графіків для отриманих розрахунків.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

1. Аналіз предметної області. 2. Технології та засоби розробки. 3. Проєктування архітектури системи обміну повідомленнями. 4. Розробка клієнтського веб-застосунку. 5. Розробка серверної частини. 6. Реалізація шифрування повідомлень 7. Економічний розрахунок. 8. Аспекти охорони праці та техніки безпеки

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Титул; Актуальність теми; Визначення процесу шифрування; Різновиди алгоритмів шифрування; Поняття комбінованого шифрування; Загальна архітектура застосунку; Використані алгоритми; Створення публічного та приватного ключів; Шифрування повідомлення; Процес дешифрування; Структура бази даних; Передача даних в режимі реального часу; Тестування функцій; Ключові аспекти безпеки; Висновки;

6. Консультанти по проекту, із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Нестеренко В.Д.*		
Економічний розділ	*Канський М.Ю.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 12.05.2025

Керівник *Нестеренко В.Д.* _____
(підпис)

Завдання прийняв до виконання *Шуміло М. С.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Формування вступу*	15.05.2025	Виконано
2	*Дослідження предметної області*	16.05.2025	Виконано
3	*Пошук і аналіз технічної літератури щодо шифрування*	17.05.2025	Виконано
	та захисту		
4	*Аналіз сучасних технологій веб-розробки, вибір стеку*	19.05.2025	Виконано
5	*Проєктування архітектури веб-застосунку*	22.05.2025	Виконано
6	*Розробка клієнтської частини*	26.05.2025	Виконано
7	*Розробка серверної частини*	01.06.2025	Виконано
8	*Реалізація алгоритмів шифрування та дешифрування повідомлень*	06.06.2025	Виконано
9	*Оформлення пояснювальної записки*	10.06.2025	Виконано
10	*Тестування застосунку в реальному часі*	11.06.2025	Виконано
11	*Підготовка графічних матеріалів*	13.06.2025	Виконано
12	*Економічний розрахунок*	13.06.2025	Виконано
13	*Опис аспектів охорони праці та техніки безпеки*	14.06.2025	Виконано
14	*Підведення висновків*	15.06.2025	Виконано
15	*Підготовка доповіді для захисту*	16.06.2025	Виконано

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			КБ 02. 26 000. 00 ДП	Дипломний проект		
А4			КБ 02. 26 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	КБ 02. 26 000. 00 ДП		
Розробив		Шуміло М.С.		16.06.25	Розробка застосунку для обміну повідомленнями з можливістю шифрування	Літ.	Аркуш
Перевірив		Нестеренко В.Д.		16.06.25		Н Д П	4 / 80
Н. контр.		Петрашова В.І.		16.06.25		ВСП "ОТФК ОНТУ" гр.4КБ-02	
Затверд.		Кривченко Ю.В.		18.06.25			

ЗМІСТ

ВСТУП

В Україні та суспільстві, де цифрові технології займають ключове місце в усіх сферах життя, питання безпеки інформації набуває особливої актуальності. З кожним роком обсяги даних, що передаються через мережу Інтернет, стрімко зростають, що зумовлює необхідність забезпечення конфіденційності, цілісності та доступності інформації. Особливо це стосується обміну повідомленнями між користувачами, оскільки особисті або корпоративні дані можуть стати об’єктом несанкціонованого доступу, перехоплення або маніпуляцій. У зв’язку з цим, розробка безпечних систем комунікації є надзвичайно важливою складовою розвитку інформаційного суспільства.

Одним із найбільш ефективних підходів до захисту даних при передачі є використання методів шифрування. Маючи доступ к криптографічним алгоритмам повідомлення можуть бути перетворені у форму, недоступну для читання сторонніми особами, що суттєво знижує ризик витоку інформації. У контексті повсякденного спілкування через мобільні або веб-застосунки, це забезпечує користувачам додаткову впевненість у безпеці їхніх даних. Ми можемо констатувати що, потреба в інструментах для захищеного обміну повідомленнями є не лише актуальною, а й вкрай необхідною в умовах сучасних цифрових викликів.

Метою даної дипломної роботи є розробка застосунку для обміну повідомленнями з можливістю шифрування, який дозволяє здійснювати безпечну комунікацію між користувачами. Основна ідея полягає в поєднанні зручного користувацького інтерфейсу з надійними алгоритмами шифрування, які забезпечують високий рівень захисту інформації. Реалізація такого рішення дозволить користувачам обмінюватися повідомленнями, не турбуючись про можливість їх перехоплення або несанкціонованого доступу до даних.

Для реалізації даного проекту було обрано сучасний стек технологій, включаючи JavaScript, Node.js, бібліотеки для реалізації шифрування (наприклад, CryptoJS або WebCrypto API), та вбудовані інструменти для створення клієнтського інтерфейсу.

У процесі розробки передбачено реалізацію функціональності реєстрації та автентифікації користувачів, формування чатів, обміну повідомленнями, при шифруванні текстових даних на стороні клієнта перед передачею на сервер.

Результатом роботи стане інтерактивний застосунок, що дозволяє користувачам вести конфіденційне спілкування у захищеному середовищі. Особлива увага приділяється зручності використання, мінімізації складності взаємодії з криптографічними механізмами, на забезпеченню продуктивності системи. Застосунок може бути корисним як для приватних осіб, що бажають зберегти приватність переписки, так і для організацій, які потребують внутрішнього засобу комунікації з підвищеним рівнем безпеки.

Отже, тема дипломної роботи є надзвичайно актуальною в умовах сучасного інформаційного середовища. Запропонований підхід до створення застосунку для обміну повідомленнями з можливістю шифрування дозволяє поєднати високі стандарти безпеки із зручністю користування, що є запорукою ефективної та надійної цифрової комунікації.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Огляд сфери застосування та ключових рішень

1.1.1 Визначення ключових викликів для безпечного листування

У цифрову епоху комунікація між людьми дедалі більше переноситься в онлайн-простір. Від особистих розмов до ділового листування та мільйони повідомлень щодня передаються через інтернет. Проте паралельно з розвитком засобів зв'язку зростає і кількість кіберзагроз. Несанкціонований доступ до повідомлень, зломи акаунтів, прослуховування трафіку - це реальність сучасного користувача, незалежно від його технічної обізнаності. Особливо гостро проблема стоїть у країнах, де тривають воєнні дії або інформаційна війна, таких як Україна. Саме тому потреба у захищеному обміні повідомленнями набуває не лише технічного, а й суспільного значення.

Серед основних викликів, що стоять перед розробниками месенджерів або комунікаційних платформ, може бути забезпечення конфіденційності, цілісності та автентичності переданої інформації. Конфіденційність передбачає, що лише відправник і отримувач мають доступ до змісту повідомлення. Цілісність гарантує, що дані не були змінені під час передачі. Автентичність - підтвердження того, що повідомлення надійшло саме від того користувача, який заявлений як відправник. Досягнення всіх трьох цілей можливе лише за умови застосування сучасних криптографічних протоколів [1].

Однак реалізація шифрування у звичайному застосунку має свої труднощі. Користувачі часто не готові жертвувати зручністю заради безпеки. Якщо процес входу, обміну ключами або відновлення повідомлень є надто складним, більшість користувачів або не користуватимуться програмою, або обиратимуть менш захищені альтернативи. Існують думки що, один із ключових викликів знайти баланс між високим рівнем захисту та простотою використання. І саме на цьому перетині зручності та безпеки виникає запит на нові рішення.

Ще однією важливою потребою є відкритість коду або принаймні прозорість архітектури. Користувачі дедалі частіше ставлять під сумнів безпеку закритих

месенджерів, які не розкривають, як саме працює їхнє шифрування [2].

У відповідь на це набувають популярності рішення на кшталт Signal або Element, які демонструють відкриту криптографію, дозволяють зовнішній аудит і мають високий рівень довіри у спільноті. У межах дипломного проєкту доцільним є реалізація застосунку, який був би не тільки безпечним, а й технологічно відкритим для розширення та перевірки з боку інших розробників.

На рис. 1.1 зображена структура моделі CIA, яка розкриває три фундаментальні принципи кіберпезбеки.

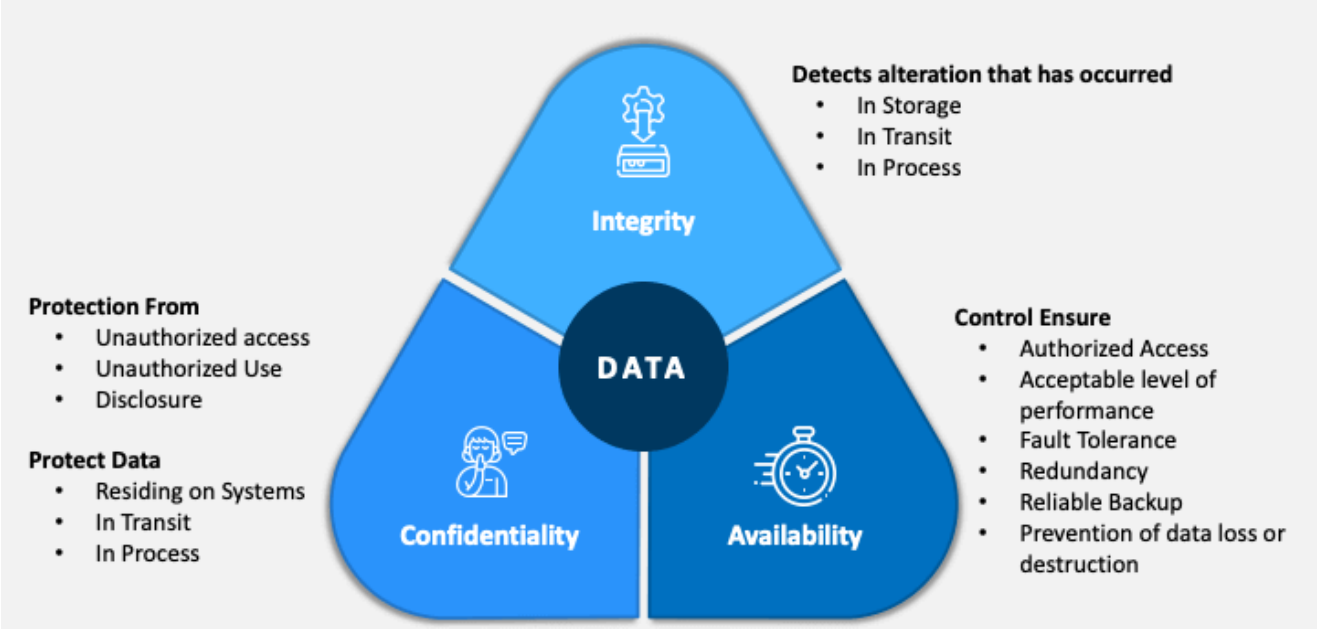


Рисунок 1.1. Діаграма моделі CIA (Confidentiality, Integrity, Availability)

Окрім того, існує потреба у створенні універсальних інструментів комунікації, які були б придатними для локального використання як у приватному, так і в державному секторі. Більшість наявних рішень розроблені за кордоном і не враховують локальну специфіку українських користувачів: обмеження законодавства, мову інтерфейсу, специфічні сценарії використання (наприклад, в умовах війни чи евакуації). Водночас українські стартапи часто не мають ресурсів для повноцінної розробки захищеного чату з нуля. Це створює реальний попит на легковагові, безпечні, локалізовані рішення [3].

Важливим є і той факт, що цифрова безпека все ще часто розглядається як другорядна функція. Наприклад, деякі стартапи починають з MVP, у якому взагалі відсутнє шифрування, і тільки пізніше намагаються його «прикрутити». Такий

підхід небезпечний, адже чим глибше система інтегрована з користувачами, тим складніше забезпечити її надійний захист без втрати сумісності або стабільності. Отже, безпека має бути закладена на рівні архітектури з перших етапів розробки. Це ще один виклик, а саме навчити студентів, молодих розробників та підприємців мислити «за замовчуванням безпечно».

Не менш важливою потребою є створення механізмів збереження та передачі ключів шифрування. В ідеалі система повинна використовувати наскрізне (end-to-end) шифрування, при якому навіть сервер не має доступу до змісту повідомлень. Це виключає можливість «зливу» інформації через вразливості або внутрішні витоки. Проте реалізація такого підходу вимагає ретельного управління ключами, синхронізації пристроїв та збереження сесій. У контексті дипломного проєкту це може бути реалізовано на основі бібліотек Web Crypto API, а для більшої автономії яка із підтримкою локального зберігання ключів у браузері користувача.

Рисунок 1.2. Обмін криптографічними ключами / End-to-End шифрування

Нарешті, варто відзначити потребу у гнучкій модульній архітектурі, яка дозволяє поступово додавати нові функції: обмін файлами, голосовий зв'язок, відеодзвінки, інтеграцію з календарем або таск-трекером. Початкове ядро системи має бути побудоване так, щоб не потребувалося повне переписування логіки при додаванні нових компонентів. Це дозволяє не лише масштабувати рішення, а й адаптувати його до різних цільових груп від студентських спільнот до

корпоративних користувачів [4].

Аналіз сучасного стану цифрових комунікацій дозволяє окреслити основні виклики: необхідність високої безпеки при збереженні зручності, прозорість і відкритість архітектури, локалізація рішень для українських реалій, забезпечення наскрізного шифрування, гнучке управління ключами та масштабованість системи. Усі ці потреби мають бути враховані при розробці застосунку, який не лише відповідатиме сучасним стандартам, а й стане реально корисним для широкого кола користувачів в Україні.

Одним із додаткових викликів є зростаюча кількість пристроїв, які користувач застосовує одночасно: смартфон, ноутбук, планшет, стаціонарний комп'ютер. У такому середовищі важливо забезпечити синхронізацію зашифрованих повідомлень між усіма пристроями користувача, не знижуючи при цьому рівень захисту. Це потребує створення складної логіки збереження ключів, контролю сесій та безпечної передачі даних між клієнтськими додатками. Якщо така синхронізація реалізована некоректно, це може призвести до втрати доступу до історії повідомлень або компрометації ключів.

Ще однією потребою сучасних застосунків є надання користувачу контролю над своїми даними. В ідеалі він має мати можливість самостійно видаляти повідомлення, обмежувати час зберігання інформації, встановлювати правила автоматичного очищення історії. Такі функції не лише підвищують довіру до системи, а й відповідають вимогам міжнародних стандартів захисту персональних даних, зокрема GDPR. Вбудовані механізми приватності мають бути зрозумілими, гнучкими й доступними для кожного користувача, незалежно від його технічного рівня.

Як ми можемо бачити, актуальною є потреба у створенні середовища, яке враховує можливості використання застосунку людьми з особливими потребами. Це включає підтримку екранних читачів, адаптивний дизайн, коректну роботу при масштабуванні інтерфейсу, кольорові контрасти, та голосові команди для навігації. Та слід зазначити що, захищена цифрова комунікація стає доступною для всіх категорій користувачів, включно з людьми похилого віку, слабозорими або особами

з моторними обмеженнями.

Слід враховувати і постійне оновлення нормативно-правової бази у сфері інформаційної безпеки, як в Україні, так і на міжнародному рівні. Застосунок повинен бути гнучким до змін вимог, наприклад, у сфері зберігання метаданих, тривалості сесій або журналів аудиту. Реалізація чіткої моделі відповідності вимогам, наприклад, ISO/IEC 27001 або національного законодавства, забезпечить готовність системи до аудиту та подальшого комерційного використання у державному чи корпоративному секторі [5].

Нарешті, критичною потребою є підвищення обізнаності користувачів щодо цифрової безпеки. Навіть найзахищеніший застосунок не здатен протистояти соціальній інженерії чи халатному ставленню користувача до власної безпеки. Тому система повинна містити елементи навчання підказки, інформаційні банери, покрокові інструкції вони повністю формують базову культуру захищеної поведінки. Саме головне, безпечна комунікація перетворюється не лише на технологічне рішення, а й на інструмент цифрової просвіти.

1.1.2 Принципи роботи систем обміну повідомленнями

Системи обміну повідомленнями є фундаментальними складовими сучасних цифрових платформ, які забезпечують передачу інформації між користувачами в реальному або наближеному до реального часу. Принципи їх роботи формуються на основі декількох базових концепцій: встановлення каналу зв’язку, автентифікації учасників, передавання даних, їх обробки та збереження, має гарантій безпеки комунікації. Ефективність такої системи визначається її здатністю обробляти велику кількість одночасних запитів із мінімальними затримками. Крім того, надзвичайно важливо забезпечити цілісність і конфіденційність повідомлень.

Типова архітектура таких систем включає клієнтську частину (інтерфейс користувача), серверну частину (яка керує повідомленнями, підключеннями, авторизацією тощо) та протокол реального часу, через який відбувається безпосередній обмін. У більшості сучасних рішень використовується архітектура з постійним двостороннім з’єднанням між клієнтом і сервером, яка зокрема, на базі WebSocket, SignalR, що забезпечує миттєву доставку повідомлень до клієнта [6].

Застосунок, розроблений у рамках цієї дипломної роботи, реалізує обмін повідомленнями із використанням бібліотеки SignalR, що входить до складу ASP.NET Core і дозволяє забезпечити високопродуктивне двостороннє з’єднання між браузером і сервером. Ця бібліотека дає клієнтам обмінюватися повідомленнями без затримок, обіймаючи класичні обмеження HTTP-запитів. SignalR автоматично управляє повторним з’єднанням, масштабуванням через кілька серверів та вибором оптимального транспортного механізму (WebSocket, Long Polling тощо).

Ключовою вимогою до систем обміну повідомленнями є безпека переданих даних. Для цього на стороні клієнта реалізовано механізми шифрування за допомогою Web Crypto API, а саме у файлі crypto.ts. Повідомлення шифрується перед надсиланням на сервер, що дозволяє досягти конфіденційності навіть у разі компрометації серверної частини. Отримувач повідомлення розшифровує його локально, використовуючи відповідний ключ, що відповідає концепції end-to-end encryption (E2EE). Загалом алгоритм виглядає наступним чином:

- Створюється симетричний ключ, який шифрується публічними ключами всіх учасників переписки;
- На сервер відправляються зашифровані симетричні ключі;
- Відправник шифрує повідомлення симетричним ключем та відправляє його отримувачу;
- Отримувач розшифровує симетричний ключ своїм приватним ключем;
- Симетричним ключем отримується доступ до повідомлення.

Оглянемо наступну таблицю, яка містить порівняльні характеристики технологій для передачі даних в режимі реального часу:

Таблиця 1.1 - Протоколи реального часу

Характеристика	**WebSocket**	**Long Polling**	**SignalR**
Тип з'єднання	Постійне двостороннє з’єднання	Періодичне HTTP-запитування	Адаптивне (WebSocket / SSE / Polling)
Підтримка двостороннього зв'язку	Так	Умовно (імітація)	Так

Продовження таблиці 1.1

Затримка передачі	Низька	Висока	Низька
Складність реалізації	Середня	Проста	Низька (високий рівень абстракції)
Підтримка масштабування	Залежить від реалізації	Складна	Вбудована підтримка
Автоматичний вибір транспорту	Ні	Ні	Так
Підтримка старих браузерів	Низька	Висока	Середня
Навантаження на сервер	Низьке	Високе	Оптимізоване

Окрім самого обміну повідомленнями, критично важливими аспектами сучасної системи комунікації є збереження історії спілкування, ефективне управління станами сесій та надійна підтримка асинхронної доставки (наприклад, коли один із користувачів перебуває офлайн). Для забезпечення цих функцій серверна частина системи виступає в ролі центрального диспетчера. Вона не лише маршрутизує повідомлення, але й надійно зберігає їх у базі даних. Це дозволяє системі гарантувати повторну доставку повідомлень, як тільки офлайн-клієнт знову підключається до мережі, забезпечуючи безперервність комунікації.

З технічної точки зору, система працює за подієвою моделлю, де повідомлення надсилається через SignalR-хаб і ретранслюється всім відповідним клієнтам. Шифрування реалізується на рівні повідомлення перед його передачею, а отже, дані не передаються в чистому вигляді ані по мережі, ані не зберігаються у відкритому вигляді на сервері. Такий підхід дозволяє мінімізувати ризики перехоплення інформації та забезпечує відповідність сучасним стандартам захисту персональних даних. Та саме головне, що це досягається шляхом застосування сучасних криптографічних алгоритмів.

SignalR здатен автоматично масштабувати дані та відправляти їх до серверу. На рис. 1.4 зображений принцип використання системи SignalR для безперервної передачі даних між клієнтом та сервером.

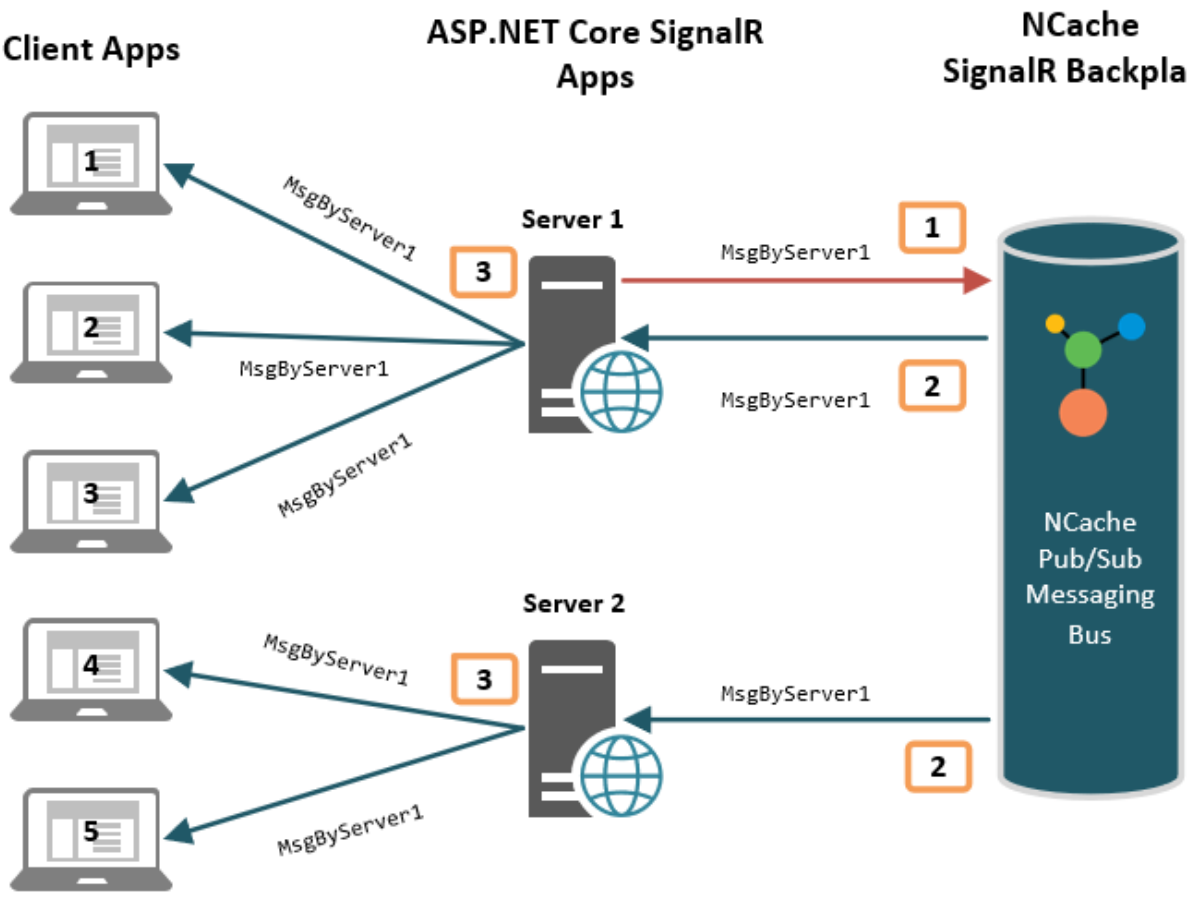


Рисунок 1.4. Відправка даних з серверу керувачам

На рисунку наведено схему роботи масштабованої системи обміну повідомленнями на основі ASP.NET Core SignalR із використанням Backplane-механізму (наприклад, NCache) для синхронізації повідомлень між кількома серверами. Клієнтські застосунки підключаються до різних серверів (Server 1, Server 2), які обмінюються повідомленнями з використанням технології SignalR. Щоб забезпечити консистентну доставку повідомлень усім підключеним клієнтам незалежно від того, до якого сервера вони підключені, використовується NCache Pub/Sub Messaging Bus це проміжний рівень, що виступає у ролі каналу передачі повідомлень між серверами. Можна насамперед використовувати та реалізувати у проектах горизонтальне масштабування системи та гарантувати однакову обробку подій у всіх вузлах, зберігаючи при цьому низьку затримку передачі повідомлень [7][13].

Можна оглядати, системи обміну повідомленнями, зокрема реалізовані у даному застосунку, базуються на поєднанні технологій реального часу, криптографії та розподілених архітектур. Забезпечення швидкості обміну,

масштабованості та захисту інформації є ключовими викликами, що визначають підхід до проєктування подібних рішень.

1.1.3 Основи криптографічного захисту повідомлень у цифрових системах

Забезпечення конфіденційності переданої інформації є однією з ключових функцій сучасних систем обміну повідомленнями. Саме можливість шифрування повідомлень дозволяє гарантувати, що дані не потраплять до сторонніх осіб під час передавання через мережу. У цьому розділі розглянемо основні підходи до криптографічного захисту, типи шифрування, має принципи їх реалізації в контексті програмного забезпечення.

Існує два основних типи шифрування: симетричне та асиметричне [8]. У симетричному шифруванні той самий ключ використовується як для шифрування, так і для розшифрування повідомлення. Це робить його швидким і ефективним для роботи з великими обсягами даних, проте виникає проблема безпечного обміну ключами між користувачами. Асиметричне шифрування, натомість, використовує пару ключів - це публічний (public key) і приватний (private key). Повідомлення, зашифроване публічним ключем, може бути розшифроване лише відповідним приватним, що дозволяє безпечно передавати дані навіть через незахищені канали.

У реальних системах часто використовується гібридний підхід, де асиметричне шифрування використовується для обміну симетричними сесійними ключами, а вже потім усі повідомлення передаються із використанням симетричного алгоритму, наприклад, AES (Advanced Encryption Standard). Такий підхід дозволяє досягти балансу між безпекою та продуктивністю.

Принципи розробки програмного забезпечення зі шифруванням базуються на концепції end-to-end encryption (E2EE). У цьому випадку повідомлення шифрується безпосередньо на пристрої відправника й розшифровується лише на пристрої одержувача, а сервер виступає лише як ретранслятор зашифрованих даних, не маючи доступу до їх вмісту [9].

Використання коду вимагає реалізації локального шифрування (наприклад, за допомогою Web Crypto API у браузері) та управління ключами на стороні

клієнта. Розглянемо таблицю порівняння основних криптографічних алгоритмів:

Таблиця 1.2 - Порівняння основних криптографічних алгоритмів

Алгоритм	Тип шифрування	Призначення	Переваги	Недоліки
AES	Симетричне	Шифрування великих обсягів даних	Висока швидкість, підтримка в більшості платформ	Необхідно безпечно передати ключ
RSA	Асиметричне	Обмін ключами, шифрування коротких повідомлень	Надійність, не потребує попереднього обміну ключами	Низька швидкість, великі розміри ключів
ECC	Асиметричне	Захищене шифрування з меншими ключами	Схожа безпека як у RSA, але при менших розмірах ключів	Складніша реалізація, обмежена підтримка
AES-GCM	Симетричне (режим)	Шифрування + автентифікація (E2EE)	Вбудована перевірка цілісності, сучасний стандарт	Потребує унікального IV для кожного повідомлення

Також слід зазначити настіпні правила:

1. AES використовується у більшості систем E2EE (наприклад, Signal, WhatsApp) для основного шифрування.
2. RSA самий класичний приклад для обміну ключами.
3. ECC (еліптична криптографія) часто використовується у мобільних застосунках через ефективність.
4. HMAC зазвичай доповнює AES або GCM, перевіряючи, що повідомлення не було змінено.

Крім алгоритмів шифрування, важливим компонентом є управління ключами (key management). Алгоритм включає генерацію, збереження, оновлення та знищення ключів. Помилки у цьому процесі можуть повністю нівелювати ефективність навіть найсильнішого шифрування. Для більшої безпеки можна використовувати сховища ключів (наприклад, IndexedDB у браузері), апаратні

токени або зовнішні сервіси на кшталт HSM (Hardware Security Module).

У межах дипломного проєкту шифрування реалізовано на клієнтському рівні та у модулі crypto.ts. Зокрема, для шифрування повідомлень використовуються функції генерації ключів, шифрування та дешифрування на основі алгоритмів AES-GCM та RSA. Він може захистити дані навіть у разі, якщо зловмисник перехопить трафік або отримає доступ до серверної частини. При цьому система зберігає простоту використання для кінцевого користувача, а саме усі криптографічні операції відбуваються у фоновому режимі, без потреби втручання користувача.

Однією з головних переваг використання end-to-end шифрування є те, що навіть розробники або адміністратори сервера не можуть переглянути вміст повідомлень. Це дозволяє усунути «людський фактор» як можливу вразливість, та зменшує юридичну відповідальність компаній за зберігання приватних даних користувачів. У більшості сучасних протоколів, зокрема у протоколі Signal, реалізується forward secrecy, який включає принцип, за якого компрометація одного сеансу або ключа не дає змоги розшифрувати попередні повідомлення.

Важливою частиною криптографічних систем є використання ініціалізаційних векторів (IV) та сольових значень (salt) [5]. Вони додають до кожного шифрування унікальні дані, що дозволяє уникати ситуацій, коли однакові повідомлення генерують однаковий зашифрований вихід. Це унеможливлює статистичний аналіз та суттєво підвищує криптостійкість системи. У практичному застосунку кожен блок повідомлення супроводжується окремим IV, що зберігається разом із шифротекстом або передається окремо.

Окремої уваги заслуговує механізм перевірки цілісності повідомлень. У сучасних реалізаціях найчастіше використовується алгоритм HMAC (Hash-based Message Authentication Code) або режим шифрування AES-GCM, який поєднує шифрування та автентифікацію. Алгоритм може відправляти одержувачу не лише розшифрувати повідомлення, але й переконатися, що воно не було змінено під час передавання. Саме головне, навіть якщо дані перехоплено, зловмисник не зможе їх підробити без відповідного ключа. На рис. 1.5 Зображений графічний принцип роботи цієї системи.

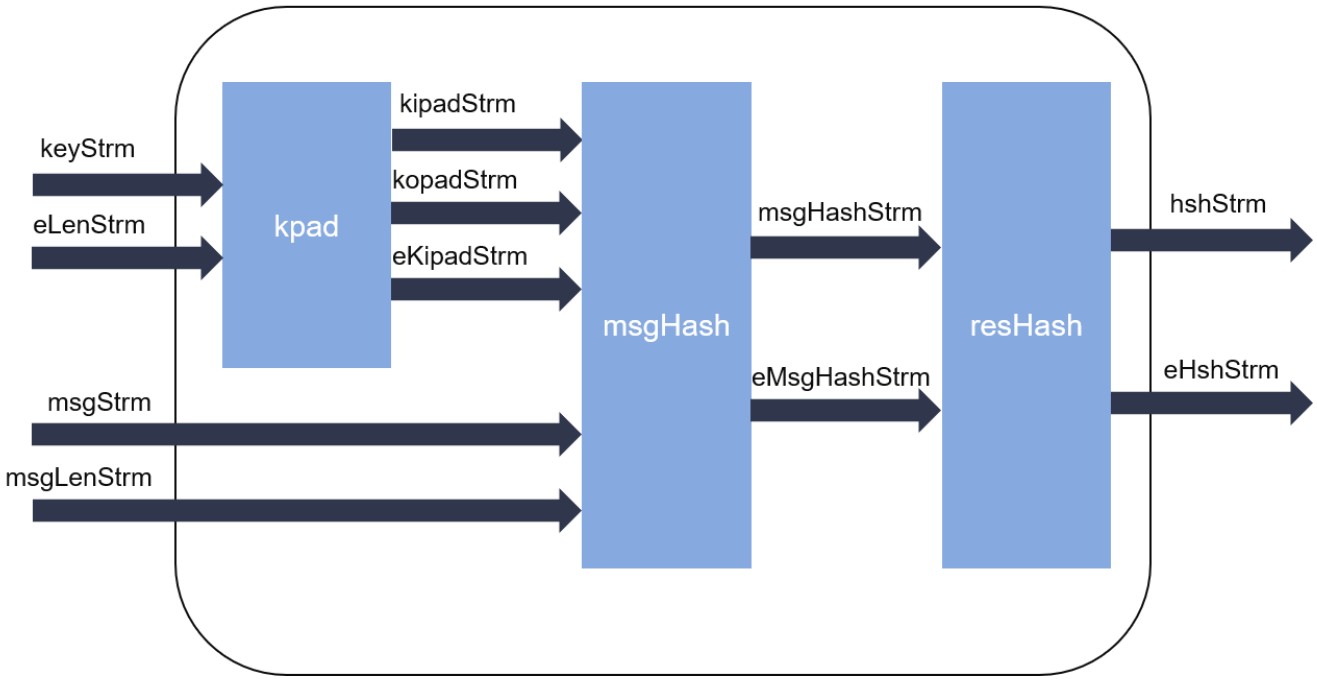


Рисунок 1.5. Принцип роботи алгоритму HMAC

У багатьох країнах, включаючи Україну, питання шифрування також регулюється на рівні законодавства. Наприклад, Закон України «Про захист персональних даних» вимагає вживання заходів для захисту конфіденційної інформації, а використання криптографічних засобів вважається одним із таких методів. У деяких випадках розробники мають дотримуватись стандартів, зокрема рекомендацій Державної служби спеціального зв'язку та захисту інформації або міжнародних протоколів (наприклад, ISO/IEC 27001, NIST SP 800-57).

Розробка власного криптографічного функціоналу в межах дипломного проєкту дозволила глибше зануритися у технічні аспекти реалізації шифрування на практиці.

Воно включало генерацію пар ключів, обробку виключень, перетворення форматів даних (наприклад, ArrayBuffer ↔ Base64), і оптимізацію швидкодії на клієнтській стороні. Крім того, вивчення сучасних бібліотек (наприклад, Web Crypto API) допомогло переконатися, що навіть у браузерному середовищі можна досягти високого рівня безпеки без необхідності використання сторонніх інструментів або серверної криптографії [10][11].

Реалізація шифрування в системах обміну повідомленнями ґрунтується на поєднанні перевірених криптографічних алгоритмів, правильної архітектури

E2EE, надійного управління ключами та захисту клієнтського середовища. Якій створює основу для безпечного цифрового спілкування, яке відповідає сучасним вимогам до конфіденційності та стійкості до загроз.

1.1.4 Технічні вимоги до безпечного месенджера

Створення системи захищеного обміну повідомленнями вимагає врахування низки технічних вимог, які забезпечують конфіденційність, надійність та стабільну роботу сервісу. Зважаючи на те, що цифрова комунікація часто містить персональні, конфіденційні або навіть критично важливі дані, недотримання базових вимог може призвести до витоків інформації, зловживань або повної втрати довіри з боку користувачів. У цьому підрозділі буде розглянуто основні вимоги до архітектури безпечного месенджера, які мають бути враховані при його проектуванні та реалізації.

Однією з ключових вимог є реалізація наскрізного шифрування (end-to-end encryption). Усі повідомлення мають шифруватися на пристрої відправника й розшифровуватися виключно на пристрої одержувача. Він дає нагрузку на сервер даних, також воно допускає алгоритм, що сервер, через який проходять повідомлення, не повинен мати доступу до їхнього вмісту. Для цього потрібна реалізація локального шифрування, обмін публічними ключами та захищене зберігання приватних ключів на клієнтській стороні. У нашому застосунку ці функції частково реалізуються у модулі crypto.ts, та проходять через модуль шифрування, із використанням Web Crypto API та вбудованним алгоритмом перехопленя ключей до доступу к даним.

Ще однією вимогою є механізм автентифікації та управління сесіями. Система повинна надійно ідентифікувати користувача, захищаючи його обліковий запис від несанкціонованого доступу. Варто впровадити щонайменше базову автентифікацію за паролем, з можливістю додавання двофакторної автентифікації (2FA) в майбутньому. Також важливо контролювати тривалість та активність сесій, мати змогу примусово завершити сеанс, якщо він підозрілий.

Не менш важливою є підтримка асинхронної доставки повідомлень, зокрема, коли один із користувачів тимчасово офлайн. У таких випадках система повинна

зберегти зашифроване повідомлення на сервері до моменту, коли одержувач знову з’явиться онлайн. Повідомлення має бути зашифроване вже на момент відправлення, а сервер повинен зберігати його лише в зашифрованому вигляді, не маючи змоги його прочитати або змінити.

Серйозну увагу потрібно приділити масштабованості та стабільності сервісу. У випадку зростання кількості користувачів або запуску застосунку в умовах високого навантаження (наприклад, під час інформаційних атак чи вірусного поширення месенджера), система має зберігати працездатність. Перехоплення досягається через реалізацію балансування навантаження, підтримку кількох серверів (кластеризацію), кешування часто використовуваних даних та використання backplane-технологій для синхронізації подій (наприклад, як у SignalR).

Останнім, але не менш важливим аспектом є взаємодія з користувачем (UX) у контексті безпеки. Інтерфейс має бути зрозумілим, не вимагати від користувача технічних знань, але водночас надавати достатньо контролю: наприклад, попереджати про зміну публічного ключа співрозмовника, дозволяти вручну перевірити автентичність ключа, повідомляти про вхід з нового пристрою. Простота й прозорість у поєднанні з безпечним за замовчуванням (secure-by-default) підходом – фундаментальні UX-вимоги до сучасних захищених месенджерів.

1.1.5 Основи протоколу Signal та його роль у сучасній криптографії

Протокол Signal сьогодні заслужено вважається одним із найнадійніших і найбільш вивчених протоколів захищеного обміну повідомленнями. Він використовується як основа у таких популярних застосунках, як Signal Messenger, WhatsApp, Facebook Messenger (у режимі "секретного чату") та підтримується відкритою спільнотою дослідників у сфері криптографії. Головна особливість протоколу Signal полягає в тому, що він здатен гарантувати високий рівень конфіденційності, автентичності та цілісності обміну повідомленнями навіть у відкритому середовищі. При цьому користувачеві не потрібно мати спеціальні знання чи виконувати складні дії – система зробить все автоматично.

Таблиця 1.3 - Схема основних механізмів протоколу Signal

Компонент	Призначення	Ключові властивості
X3DH	Початковий обмін ключами між користувачами	Працює асинхронно, дозволяє почати діалог з офлайн-користувачем
PreKeys	Набір публічних ключів, що публікується на сервері	Забезпечують можливість встановлення першого з'єднання
Double Ratchet	Оновлення симетричних ключів після кожного повідомлення	Forward Secrecy + Future Secrecy
Identity Keys	Сталі пари ключів для кожного користувача	Використовуються для верифікації автентичності
Session Keys	Тимчасові симетричні ключі для конкретного діалогу	Змінюються з кожним повідомленням
Message Keys	Ключ для конкретного повідомлення, що генерується з session key	Одноразовий, використовується тільки для одного шифрування

Основною особливістю протоколу Signal є використання гібридної криптографічної моделі, яка поєднує переваги симетричного та асиметричного шифрування. Суть підходу полягає в тому, що для кожного сеансу спілкування динамічно генеруються сесійні ключі (симетричні), які захищаються з використанням асиметричних ключів, що дозволяє забезпечити Forward Secrecy тобто, навіть у разі компрометації одного з ключів, інші сесії залишаються захищеними.

Архітектура протоколу включає кілька криптографічних механізмів. Один із ключових - це X3DH (Extended Triple Diffie-Hellman) який використовується для безпечного початкового встановлення зв’язку між користувачами [12]. Він дозволяє обмінятися ключами навіть тоді, коли один із співрозмовників тимчасово офлайн. Іншим важливим елементом є Double Ratchet Algorithm, який відповідає за оновлення ключів після кожного повідомлення, забезпечуючи як forward secrecy, так і future secrecy (захист майбутніх повідомлень у разі зламу теперішнього сеансу). На рис.1.6 зображений механізм безпечної передачі даних між двома користувачами.

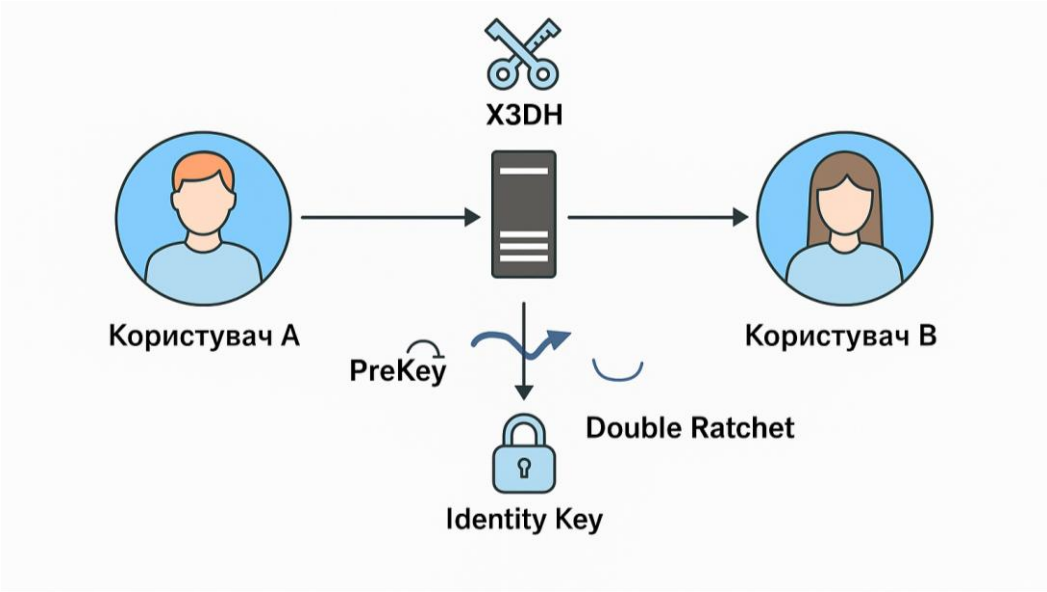


Рисунок 1.6. Передача протоколу по механізмах скрізь X3DH

Окрім цього, протокол використовує так звані PreKeys попередньо завантажені на сервер відкриті ключі, що дозволяють асинхронно ініціювати розмову. Кожен користувач публікує певну кількість таких ключів, і коли інший користувач хоче надіслати повідомлення, він використовує один із доступних PreKey для початку сеансу. Алгоритм дозволяє уникати необхідності одночасного перебування обох сторін онлайн.

У контексті дипломного проєкту вивчення протоколу Signal відіграє важливу роль, оскільки його структура та принципи можуть бути адаптовані у спрощеному вигляді для реалізації на клієнтській стороні. Так, зокрема, концепція оновлення сесійних ключів, розділення транспортного рівня та рівня шифрування, обмежене зберігання історії ключів, усе це може бути застосовано для підвищення стійкості власного рішення до атак. Хоча повна реалізація протоколу Signal є досить складною й вимагає глибоких криптографічних знань, її поетапне впровадження дозволяє досягти якісно нового рівня безпеки у цифрових комунікаційних системах.

1.2 Розробка додатку

1.2.1 Архітектура та компоненти системи

Розробка застосунку для захищеного обміну повідомленнями потребує чіткої, модульної архітектури, яка дозволяє ефективно розділяти відповідальності,

забезпечувати масштабованість, безпеку та зручність супроводу коду. У межах дипломного проєкту було реалізовано повноцінну клієнт-серверну систему, яка складається з трьох основних рівнів: клієнтська частина, серверна частина та база даних.

Клієнтська частина побудована з використанням технологій React + TypeScript. Інтерфейс реалізовано як односторінковий застосунок (SPA), що забезпечує швидку взаємодію з користувачем без постійного перезавантаження сторінок. React використовується для компонентної організації інтерфейсу, а TypeScript дозволяє впровадити статичну типізацію, зменшуючи ймовірність помилок під час виконання. Основні функціональні елементи, такі як ChatView.tsx, обробляють взаємодію користувача з чатами, тоді як crypto.ts відповідає за шифрування повідомлень на стороні клієнта.

Серверна частина реалізована з використанням платформи ASP.NET Core. Вона включає окремі проєкти:

1. SecureMessages.Service - логіка обробки запитів,
2. SecureMessages.DataAccess - доступ до бази даних,
3. SecureMessages - точка входу (web API).

Комунікація між клієнтом і сервером відбувається через HTTP-запити та SignalR, який забезпечує двосторонній зв’язок у реальному часі. Сервер відповідає за автентифікацію користувачів, маршрутизацію повідомлень, логіку обміну ключами, а також управління сесіями.

База даних створена на основі SQL Server, структура якої збережена у вигляді .bacpac-файлу (messenger.bacpac). Вона містить основні таблиці для зберігання інформації про користувачів, повідомлення, сесії та ключі шифрування (у зашифрованому вигляді або в хеш-формі). Модель даних побудована відповідно до принципів нормалізації (1NF–3NF) з урахуванням можливості масштабування в майбутньому [14].

На концептуальному рівні архітектура відповідає трирівневій моделі:

1. Presentation layer (UI) - це інтерфейс користувача, реалізований у React.
2. Application layer (API + SignalR) - це логіка обробки запитів, керування

повідомленнями.

3. Data layer (SQL Server) - це зберігання структурованих даних, логіка запитів.

У якості каналу зв’язку між клієнтом і сервером використовується HTTPS, що захищає передані дані на транспортному рівні. Проте додатково вся комунікація між користувачами шифрується на клієнті у якому згідно з принципами end-to-end encryption, що гарантує захист навіть у разі компрометації сервера або перехоплення трафіку.

Архітектура застосунку реалізує принципи модульності, безпеки та гнучкості. Вона дозволяє у подальшому розширювати функціональність, підключати нові сервіси (наприклад, відеозв’язок або групові чати), масштабувати систему під збільшення навантаження та підтримувати різні типи клієнтів (мобільні, десктопні, веб).

1.2.2 Реалізація аутентифікації та керування користувачами

Аутентифікація користувачів є критично важливою частиною будь-якого захищеного застосунку, оскільки саме вона гарантує, що доступ до особистих даних отримує лише авторизована особа. У межах цього дипломного проєкту реалізовано базовий, але захищений механізм автентифікації, який дозволяє користувачам реєструватися, входити в систему та підтримувати сесії доступу з використанням токенів.

На серверній стороні застосунку, розробленого з використанням ASP.NET Core, реалізовано контролери, що відповідають за процеси реєстрації та авторизації користувачів. Коли користувач проходить реєстрацію, він вводить логін і пароль, які надсилаються на сервер для обробки. Пароль у жодному разі не зберігається у відкритому вигляді — замість цього він хешується за допомогою надійного алгоритму bcrypt. Цей алгоритм додає унікальну соль та виконує складне обчислення, що робить результуючий хеш стійким до злому навіть при використанні сучасних засобів для brute-force атак.

Завдяки такому підходу, навіть якщо база даних буде скомпрометована, зловмисник не зможе отримати реальні паролі, адже відновити їх з хешів практично неможливо. Після успішного хешування, дані користувача разом із

зашифрованим паролем зберігаються в базі даних. У разі подальшої авторизації система перевіряє надані облікові дані, порівнюючи пароль, введений користувачем, з раніше збереженим хешем. Якщо перевірка проходить успішно, користувачу видається токен автентифікації (JWT), який містить всю необхідну інформацію для підтвердження особи при подальших запитах до API.

Такий підхід дозволяє забезпечити високий рівень захисту персональних даних користувачів та гарантує безпечне зберігання паролів, відповідно до сучасних стандартів безпеки веб-застосунків.

Для авторизації застосовується механізм JWT (JSON Web Token). Після успішного входу користувач отримує токен, який містить основну інформацію про особу (наприклад, ID, логін, ролі) і підписаний сервером. Цей токен передається клієнтом у заголовку Authorization: Bearer під час подальших запитів до API. При цьому забезпечується статусна авторизація без необхідності зберігати сесію на сервері, що особливо зручно для масштабованих рішень.

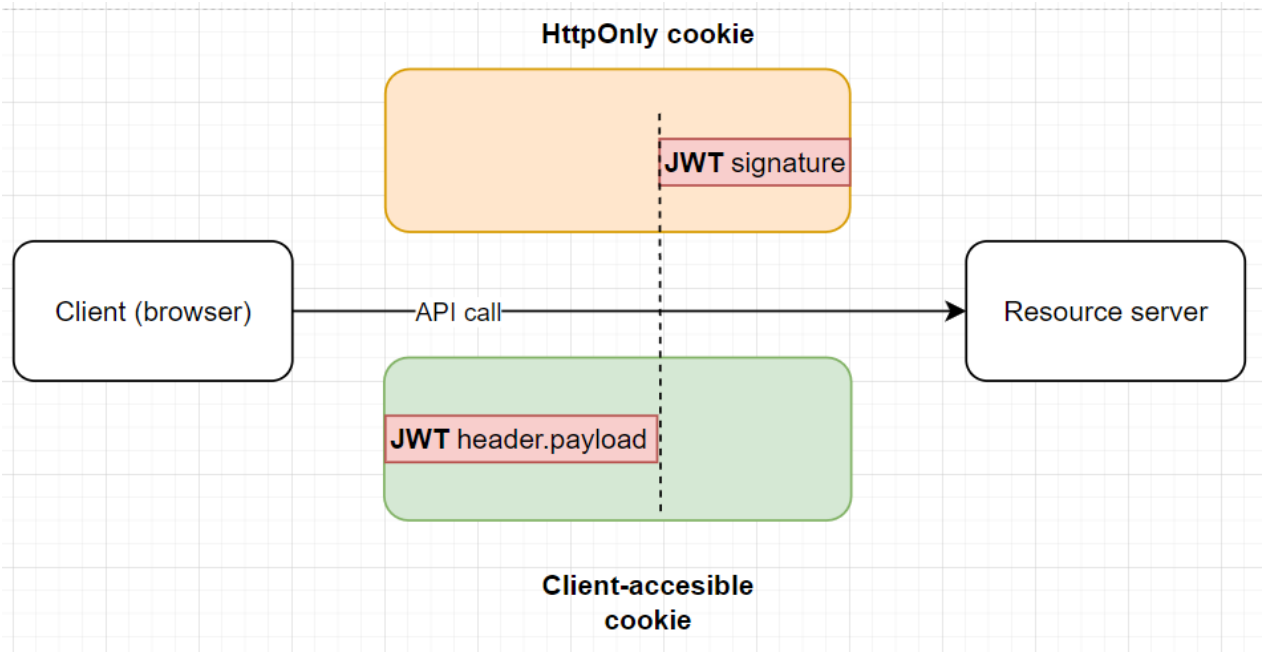


Рисунок 1.7. Приклад виконання запиту до ресурсу

У застосунках на основі ASP.NET Core, авторизація реалізується за допомогою вбудованого middleware, який підключається у файлі Program.cs або Startup.cs. Основна логка полягає у тому, що механізм працює у зв’язці з JWT, дозволяючи перевіряти кожен вхідний HTTP-запит на наявність авторизаційного токена.

Типова конфігурація виглядає так:

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = "SecureMessages",
            ValidAudience = "SecureMessagesClient",
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes("YourSuperSecretKey"))
        };
    });
```

Розглянемо алгоритм детальніше:

1. Використовується схема JWT Bearer, тобто очікується, що користувач надсилатиме токен у заголовку Authorization: Bearer {token}.
2. Налаштовано перевірку підпису, дати дії, Issuer та Audience, що підвищує захищеність.
3. Ключ "1234567890123456" є симетричним і використовується для підпису токенів (в реальному проєкті має бути довшим і зберігатись у Secrets або Azure Vault).

Цей код налаштовує JWT-автентифікацію, перевіряючи підпис токена, дату його дії (Lifetime), відповідність видавця (Issuer) і клієнта (Audience). Ключ, яким підписано токен, зберігається локально на сервері.

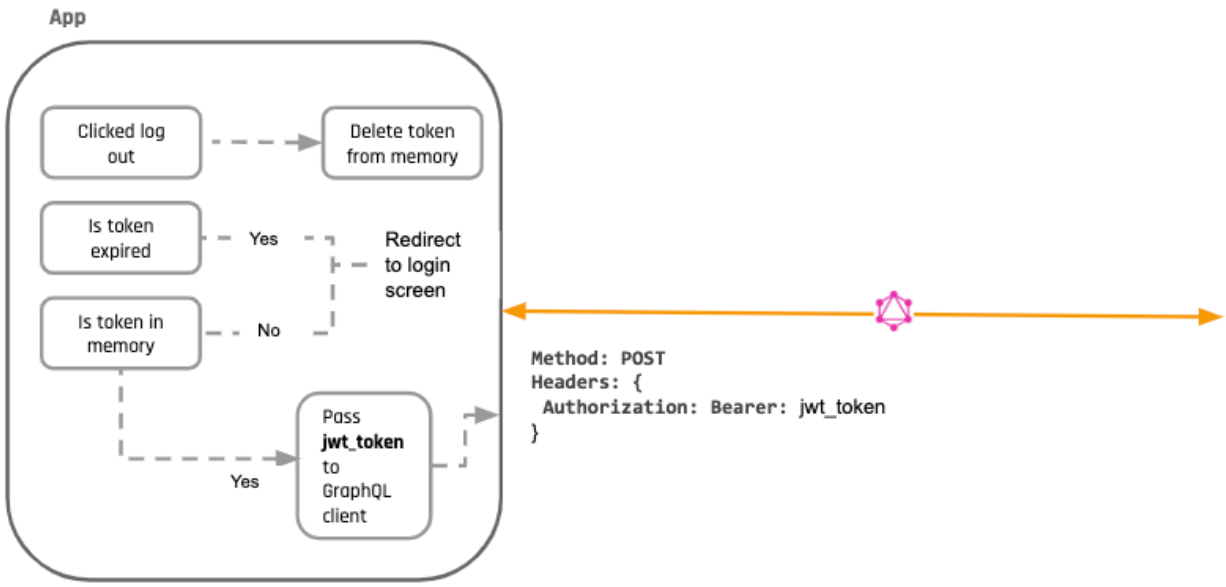


Рисунок 1.8. Відправка запросу з токеном, та його перевірка

Після налаштування сервісу потрібно підключити middleware у пайплайн обробки запитів:

```
app.UseAuthentication();
app.UseAuthorization();
```

1. UseAuthentication() перевіряє токен у кожному запиті – чи є він, чи дійсний.
2. UseAuthorization() перевіряє, чи має користувач доступ до ресурсу згідно з політиками або ролями.

Ці виклики розміщуються у Program.cs після app.UseRouting() і перед app.MapControllers(). UseAuthentication() відповідає за зчитування та перевірку токена з заголовка запиту, а UseAuthorization() відповідає за перевірку прав доступу до конкретного ресурсу, згідно з ролями або політиками.

Результатом є захищений API, до якого користувач може звертатися лише після успішної автентифікації. Якщо токен відсутній, недійсний або прострочений, сервер автоматично поверне відповідь 401 Unauthorized.

Використяння дозволяє централізовано керувати доступом до всіх частин застосунку, забезпечуючи як гнучкість, так і високу безпеку, без необхідності вручну перевіряти права в кожному контролері.

На стороні клієнта (React + TypeScript) реалізовано форми для входу та реєстрації, логіку зберігання JWT у локальному сховищі (наприклад, localStorage або sessionStorage), а також захищену навігацію де користувач не може потрапити до головного інтерфейсу обміну повідомленнями без наявного та валідного токена. У разі його відсутності або завершення терміну дії користувача автоматично перенаправляє на сторінку входу.

У системі також реалізовано перевірку автентичності на кожному запиті дає запит на бекенд аналізує токен, перевіряє його підпис, дату створення та інші метадані. На рівні бекенду кожен вхідний запит проходить перевірку токена, що виконується посередниками (middleware) або вбудованими фільтрами ASP.NET Core. Перевірка включає такі основні етапи

- Аналіз структури токена – перевіряється, чи відповідає токен очікуваному формату.
- Перевірка цифрового підпису – завдяки криптографічному підпису

можливо впевнитися, що токен не був змінений або підроблений сторонніми особами.

- Перевірка терміну дії – аналізується час створення та час закінчення дії токена (поля iat, exp), щоб виключити можливість використання простроченого токена.
- Перевірка метаданих – такі як ідентифікатор користувача, роль (claims), Device ID.

У разі, якщо будь-який із цих етапів не проходить успішно — наприклад, токен є простроченим, зміненим або зовсім відсутнім — система автоматично генерує виняток, який обробляється глобальним механізмом обробки помилок. Клієнту повертається відповідь з HTTP-кодом 401 Unauthorized, що однозначно вказує на помилку автентифікації. У відповіді також може міститися додаткова інформація у форматі JSON, яка допомагає клієнтському інтерфейсу інформувати користувача про проблему (наприклад, "Сесія завершена, увійдіть повторно").

Такий підхід дозволяє не лише гарантувати безпеку доступу до API, але й чітко контролювати, які користувачі можуть виконувати певні дії. Це особливо важливо для функціональності, що пов’язана із приватним листуванням, обробкою конфіденційних даних, або зміною профілю користувача.

Завдяки автоматизованій перевірці автентичності на кожному запиті, система SecureMessages підтримує високий рівень захисту, запобігаючи несанкціонованому доступу та зберігаючи цілісність користувацьких даних у режимі реального часу.

Для керування користувачами передбачено мінімальні адміністративні функції, наприклад, перевірка існування користувача, валідація логіна, та можливість реалізації додаткових ролей у майбутньому. У структурі бази даних це реалізовано через таблицю Users, яка зберігає інформацію про облікові записи, дати реєстрації, хешовані паролі та інші атрибути.

Реалізований механізм автентифікації поєднує в собі сучасні стандарти безпеки з простотою інтеграції в SPA-застосунок. Він забезпечує надійний контроль доступу, масштабованість та зручність для користувачів без потреби в

складних зовнішніх сервісах (як-от OAuth або LDAP), залишаючи можливість для їх додавання в майбутньому.

Одним із важливих принципів безпечної автентифікації є зберігання лише хешованих паролів. У системі паролі не зберігаються у відкритому вигляді, а обробляються за допомогою криптографічного хешування. Типовий підхід включає використання алгоритму PBKDF2, bcrypt або Argon2, які спеціально призначені для захисту паролів і мають параметри для уповільнення обчислень, що ускладнює атаки перебором.

На рівні API авторизації можна використати такий підхід:

```
var passwordHash = Convert.ToBase64String(KeyDerivation.Pbkdf2(
    password: request.Password,
    salt: user.Salt,
    prf: KeyDerivationPrf.HMACSHA256,
    iterationCount: 10000,
    numBytesRequested: 256 / 8));
```

Цей код демонструє, як створити хеш пароля з використанням PBKDF2 - одного з рекомендованих алгоритмів. Він також використовує сіль (salt) - це унікальний випадковий набір байтів, що додається до кожного пароля для уникнення атак за попередньо обчисленими хешами (наприклад, rainbow tables). Він забезпечує високий рівень захисту навіть при використанні простих паролів.

У контексті автентифікації, після порівняння хешів, сервер генерує JWT-токен. У JWT зазвичай включається userId, login, можливо за роль. Токен підписується секретним ключем, що зберігається у конфігурації (appsettings.json) і недоступний зовні. Наприклад:

```
var token = new JwtSecurityToken(
    issuer: "SecureMessages",
    audience: "SecureMessagesClient",
    claims: claims,
    expires: DateTime.UtcNow.AddHours(1),
    signingCredentials: credentials);
```

Тут користувач отримує токен, який автоматично додається до запитів у заголовку. Сервер перевіряє його на кожному запиті та якщо він був невалідний, прострочений або модифікований, запит блокується. Це гарантує захист від підробки сесій або несанкціонованого доступу.

Також важливо передбачити механізми для відновлення доступу, наприклад, запити на скидання пароля, які реалізуються через тимчасові токени або коди,

надіслані на email у вигляді пар хешу, та розкриваються через алгоритм пар ключ-значення, які вимагають сучастні алгоритми доступу до даних, вони на мють уяви про пароль, а прорівнюють ключи, а архітектура дозволяє легко додати її в майбутньому без радикальних змін.

1.2.3 Реалізація алгоритму шифрування та де-шифрування

Питання захисту персональних даних стоїть на першому плані, реалізація надійного алгоритму шифрування є критично важливою складовою будь-якого засобу обміну повідомленнями. Застосунок, розроблений у межах цієї дипломної роботи, демонструє практичну реалізацію високозахищеної системи обміну повідомленнями, у якій особлива увага приділена шифруванню та дешифруванню переданих даних.

В основі реалізованої системи лежить використання асиметричного шифрування RSA-OAEP. Даний підхід дозволяє гарантувати, що повідомлення, надіслане одним користувачем, зможе прочитати лише інший користувач, який володіє відповідним приватним ключем. Усе це реалізовано за допомогою сучасного Web Cryptography API, що підтримується у більшості браузерів та дозволяє виконувати криптографічні операції без потреби у зовнішніх бібліотеках.

Одним із перших етапів ініціалізації системи шифрування є створення пари відкритого та закритого ключів:

```
export async function generateKeyPair(): Promise<CryptoKeyPair> {
  return await window.crypto.subtle.generateKey(
    {
      name: "RSA-OAEP",
      modulusLength: 2048,
      publicExponent: new Uint8Array([1, 0, 1]),
      hash: "SHA-256"
    },
    true,
    ["encrypt", "decrypt"]
  );
}
```

Цей код виконує генерацію пари ключів RSA з довжиною модуля 2048 біт, що є загальноприйнятим стандартом для забезпечення високого рівня криптостійкості. Параметр hash: "SHA-256" свідчить про використання SHA-256 як допоміжної хеш-функції, яка підвищує стійкість до атак типу padding oracle. Основна математична основа RSA включає використання двох великих простих

чисел для створення модуля n = p × q, а також визначення відкритої експоненти e та закритої експоненти d, що задовольняє рівняння:

$$e \times d \equiv 1 \ (mod\ \varphi(n)),\ де\ \varphi(n) = (p-1)(q-1). \quad (1.1)$$

Після створення ключів, їх потрібно експортувати для зберігання чи передачі:

```
export async function exportPublicKey(key: CryptoKey): Promise<string> {
  const exported = await window.crypto.subtle.exportKey("spki", key);
  return btoa(String.fromCharCode(...new Uint8Array(exported)));
}
```

У цьому фрагменті коду ми експортуємо публічний ключ у форматі SPKI (Subject Public Key Info), а потім перетворюємо його у формат base64 для простоти збереження та передачі. Основна криптографічна операція – власне, шифрування за допомогою публічного ключа RSA:

```
export async function encryptKeyWithPublicKey(symKey: CryptoKey, publicKey:
CryptoKey): Promise<string> {
    const exportedKey = await crypto.subtle.exportKey("raw", symKey);

    const encrypted = await crypto.subtle.encrypt(
        { name: "RSA-OAEP" },
        publicKey,
        exportedKey
    );

    return btoa(String.fromCharCode(...new Uint8Array(encrypted)));
}
```

Повідомлення перетворюється в байтовий формат, а далі – шифрується відкритим ключем користувача. Це забезпечує, що тільки користувач, який володіє відповідним приватним ключем, зможе прочитати повідомлення. Алгоритм RSA-OAEP реалізує шифрування за формулою:

C = RSAEncrypt(M, e, n), де M це закодоване повідомлення, а C - шифртекст, який надсилається. На боці отримувача повідомлення виконується дешифрування:

```
export async function decryptSymmetricKey(encryptedBase64: string, privateKeyPem:
string): Promise<CryptoKey> {
    const privateKey = await importPrivateKey(privateKeyPem);
    const encrypted = Uint8Array.from(atob(encryptedBase64), c => c.charCodeAt(0));
    const decryptedRawKey = await crypto.subtle.decrypt(
        { name: "RSA-OAEP" },
        privateKey,
        encrypted
    );
    return await crypto.subtle.importKey(
        "raw",
        decryptedRawKey,
        { name: "AES-GCM" },
        true,
        ["encrypt", "decrypt"]
```

```
        );
    }
```

Ключ використовується для розшифрування зашифрованого байтового масиву. Це гарантує, що тільки авторизований користувач має змогу отримати доступ до змісту повідомлення.

Основна перевага полягає у використанні пари ключів: відкритого для шифрування і приватного - для дешифрування. Такий підхід усуває потребу у безпечному каналі для передачі самого ключа, що є критичним у відкритих мережах, таких як Інтернет. При застосуванні схеми OAEP (Optimal Asymmetric Encryption Padding), алгоритм набуває додаткового рівня захисту від атак на основі обраного шифртексту (chosen ciphertext attacks), що робить його практично невразливим до класичних криптоаналітичних підходів.

Передбачено реалізацію шифрування повідомлень із використанням алгоритму AES (Advanced Encryption Standard). Цей алгоритм належить до класу симетричних криптографічних методів, тобто для процесів шифрування та дешифрування використовується один і той самий ключ. Однією з головних переваг AES є його висока швидкодія, що дозволяє ефективно обробляти великі обсяги даних без значних витрат ресурсів. Проте, водночас, симетричне шифрування має певні недоліки в аспекті безпеки, зокрема у випадках, коли ключ може бути перехоплений або скомпрометований.

Саме тому в розробленому підході реалізовано принцип подвійного шифрування. На першому етапі повідомлення шифрується за допомогою AES, що забезпечує швидкість та компактність. Для підвищення рівня захищеності кожне шифрування супроводжується генерацією унікального вектора ініціалізації (IV — Initialization Vector), який формує випадковий набір байтів. Цей вектор додає елемент непередбачуваності, навіть якщо вміст повідомлення залишається незмінним. Тобто, два однакові повідомлення, зашифровані з різними IV, даватимуть абсолютно різні результати шифрування.

Після завершення процесу шифрування отримані байти зашифрованого тексту об'єднуються з масивом байтів IV. Це об'єднання необхідне для того, щоб при подальшому дешифруванні можна було правильно відтворити оригінальне

повідомлення. У такому вигляді (зашифроване повідомлення + IV) дані зберігаються в базі даних, що робить їх надійними та стійкими до спроб несанкціонованого доступу.

```
export async function encryptMessage(message: string, key: CryptoKey):
Promise<string> {
    const iv = crypto.getRandomValues(new Uint8Array(12));
    const encoded = new TextEncoder().encode(message);

    const ciphertext = await crypto.subtle.encrypt(
        { name: "AES-GCM", iv },
        key,
        encoded
    );

    const combined = new Uint8Array(iv.length + ciphertext.byteLength);
    combined.set(iv, 0);
    combined.set(new Uint8Array(ciphertext), iv.length);

    return bufferToBase64(combined.buffer);
}
```

Процес дешифрування є зворотним до шифрування. Спочатку алгоритм виділяє збережений IV та основний зашифрований текст. Зазвичай ці два елементи зберігаються як одна строка, де IV може бути записано на початку або в кінці. Потім за допомогою криптографічної бібліотеки та збереженого симетричного ключа здійснюється розшифрування повідомлення. На останньому етапі масив байтів, отриманий у результаті дешифрування, перетворюється назад у текстову строку, яку можна читати. Оскільки для розшифрування використовується той самий ключ, що й для шифрування, дуже важливо забезпечити безпечне зберігання цього ключа, наприклад, у спеціалізованому сховищі або за допомогою додаткових методів захисту.

```
export async function decryptMessage(encrypted: string, key: CryptoKey):
Promise<string> {
    const combined = new Uint8Array(base64ToBuffer(encrypted));
    const iv = combined.slice(0, 12);
    const data = combined.slice(12);

    const decrypted = await crypto.subtle.decrypt(
        { name: "AES-GCM", iv },
        key,
        data
    );

    return new TextDecoder().decode(decrypted);
}
```

Висока криптостійкість, кросплатформеність, можливість експорту/імпорту ключів і простота розгортання роблять цей алгоритм ідеальним для впровадження

у системах безпечного обміну повідомленнями, де довіра, надійність та стабільність мають першочергове значення.

Також серед переваг реалізованого алгоритму варто відзначити:

1. Асиметричність відсутність потреби у передачі ключів шифрування у відкритому вигляді;
2. Використання AES + RSA - гарантія криптостійкості та захисту від атак на цілісність даних;
3. Універсальність Web Cryptography API дозволяє використовувати дану реалізацію на більшості пристроїв без зовнішніх залежностей;
4. Експорт/імпорт ключів система дозволяє переносити ключі між пристроями або зберігати їх у зашифрованому вигляді на сервері.

Дана реалізація ідеально підходить для безпечного обміну повідомленнями, що є ключовою потребою в умовах кіберзагроз. Вона може бути адаптована до більш складних протоколів, зокрема до реалізації моделі end-to-end encryption (E2EE), що є стандартом у таких застосунках, як Signal чи WhatsApp. При прозорій структурі коду, її легко масштабувати або вдосконалювати, а саме додавши підтримку симетричних методів шифрування (AES) для обробки великих обсягів даних.

Для того, щоб розпочати чат, необхідно знати унікальний ідентифікатор співрозмовника або кожного з учасників групового чату. Цей ідентифікатор дозволяє Клієнтській частині виконати запит на сервер для отримання відповідного публічного ключа кожного користувача. Публічні ключі потрібні для безпечного обміну даними, оскільки саме вони використовуються на етапі початкового шифрування.

Після успішного отримання публічних ключів, Клієнтська частина автоматично генерує симетричний ключ у форматі AES (Advanced Encryption Standard). Цей ключ є спільним для всіх учасників конкретного чату й використовується для шифрування та дешифрування усіх текстових повідомлень, файлів або іншого контенту, який передається в межах чату. Використання симетричного шифрування забезпечує високу швидкість обробки даних при

збереженні достатнього рівня безпеки.

Однак, щоб учасники чату могли скористатися цим симетричним ключем, необхідно його передати їм у захищеному вигляді. Для цього використовується метод гібридного шифрування. Кожен симетричний AES-ключ шифрується окремо для кожного учасника чату з використанням його індивідуального публічного RSA ключа. У результаті цього процесу утворюється N зашифрованих копій симетричного ключа, де N — кількість учасників чату.

Ці зашифровані копії надсилаються на сервер, який зберігає їх та передає відповідним клієнтам. Кожен учасник, отримавши свою зашифровану копію, використовує свій приватний ключ для її розшифрування та отримання оригінального симетричного AES-ключа. Таким чином, забезпечується, що тільки авторизовані учасники мають доступ до шифрування і розшифрування повідомлень у чаті. Як наслідок, створюється захищене середовище для обміну повідомленнями, в якому дані залишаються конфіденційними навіть у випадку перехоплення трафіку. Відповідна блок-схема зображена на рис. 1.9.

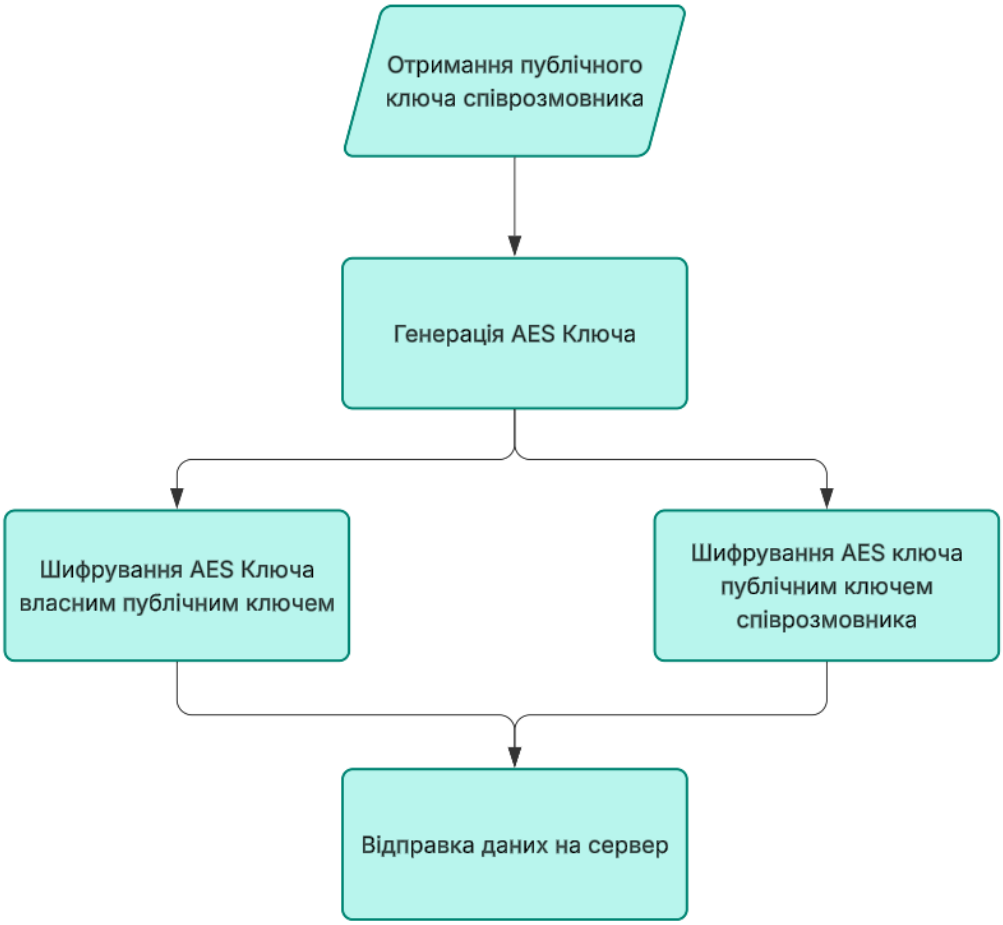


Рисунок 1.9 Алгоритм початку чату з співрозмовником

Коли один із учасників входить до чату, йому необхідно мати змогу розшифрувати усі раніше надіслані повідомлення, щоб побачити повну історію розмови. Для цього виконується низка кроків у зворотному порядку до процесу початкового шифрування.

Насамперед клієнтська частина надсилає запит на сервер для отримання зашифрованих повідомлень, а також своєї копії симетричного ключа. Ця копія була попередньо зашифрована з використанням публічного ключа цього конкретного учасника під час ініціалізації або приєднання до чату.

Після отримання даних клієнт починає процес дешифрування. На першому етапі відбувається розшифрування симетричного ключа. Для цього застосовується приватний ключ користувача, який зберігається лише на його пристрої і недоступний нікому іншому, включно із сервером. Цей крок критично важливий, оскільки тільки власник приватного ключа може отримати доступ до дійсного симетричного AES-ключа.

Як тільки симетричний ключ успішно відновлено, клієнт може перейти до наступного етапу — розшифрування самих повідомлень. Кожне повідомлення, що надходить у зашифрованому вигляді, обробляється з використанням отриманого AES-ключа. В результаті користувач бачить повідомлення у відкритому вигляді, саме так, як їх було написано іншими учасниками. Дешифрування самих повідомлень проходить у циклі і відбувається N разів, де N – кількість повідомлень. Саме тому в алгоритм було додано алгоритм AES, адже таке дешифрування відбувається значно швидше аніж за допомогою асиметричних алгоритмів, що критично важливо з точки зору кінцевого користувача, який не бажає чекати довгий час на дешифрування.

Завдяки такій архітектурі забезпечується, що навіть при повторному вході до чату або зміні пристрою (за умови приватного ключа) користувач може безпечно відновити доступ до історії листування. Водночас система гарантує, що жоден сторонній користувач або навіть сервер не зможе здійснити розшифрування, оскільки для цього необхідний приватний ключ, що залишається під контролем користувача. На рис. 1.10 зображено блок-схему алгоритму процесу дешифрування

Рисунок 1.10 Блок-схема процесу дешифрування

Впроваджений криптографічний модуль не лише відповідає вимогам безпеки, а й демонструє високу ефективність, зручність та адаптивність до майбутніх змін архітектури застосунку.

1.2.4 Структура бази даних та логіка запитів

Проєкт "SecureMessages" передбачає реалізацію простої, але ефективної схеми зберігання даних, що відповідає потребам системи захищеного обміну повідомленнями. Архітектура бази даних побудована так, щоб забезпечити швидкий доступ до інформації, високий рівень безпеки, простоту масштабування та зрозумілу логіку взаємодії між таблицями. База даних складається з двох ключових таблиць:

Users – таблиця, що містить інформацію про зареєстрованих користувачів.

Messages – таблиця, що зберігає зашифровані повідомлення між користувачами, слід наголосити, що вся інформація вже зашифрована у таблиці.

Таблиця Users має наступну структуру:

1. Id ціле число (INT), первинний ключ, автоінкремент;
2. Username текстове поле для логіна користувача;
3. PublicKey публічний ключ користувача у форматі NVARCHAR(MAX);
4. PasswordHash хеш паролю користувача.

```
CREATE TABLE Users (
    Id INT PRIMARY KEY IDENTITY(1,1),
    Username NVARCHAR(100) NOT NULL,
    PublicKey NVARCHAR(MAX) NOT NULL,
    PasswordHash NVARCHAR(MAX) NOT NULL
);
```

Така структура дозволяє ефективно і безпечно зберігати облікові дані, водночас публічний ключ використовується для шифрування повідомлень.

Таблиця Messages:

1. Id первинний ключ повідомлення;
2. SenderId зовнішній ключ, що посилається на Users.Id;
3. ReceiverId зовнішній ключ на Users.Id;
4. EncryptedText зашифроване повідомлення;
5. SentAt дата та час відправлення повідомлення.

```
CREATE TABLE Messages (
    Id INT PRIMARY KEY IDENTITY(1,1),
    SenderId INT NOT NULL,
    ReceiverId INT NOT NULL,
    EncryptedText NVARCHAR(MAX) NOT NULL,
    SentAt DATETIME DEFAULT GETDATE(),
    FOREIGN KEY (SenderId) REFERENCES Users(Id),
    FOREIGN KEY (ReceiverId) REFERENCES Users(Id)
);
```

Після створення таблиць головним завданням є реалізація запитів, що обслуговують ключові процеси: реєстрацію, авторизацію, відправлення повідомлень, отримання історії чату. Реєстрація користувача має наступний вигляд:

```
INSERT INTO Users (Username, PublicKey, PasswordHash)
VALUES ('john_doe', '<BASE64_PUBLIC_KEY>', '<HASHED_PASSWORD>');
```

Цей запит вставляє новий запис користувача в базу. Публічний ключ зберігається у відкритому вигляді, оскільки використовується для шифрування.

Авторизація відбувається не SQL-запитом, а шляхом перевірки хешу паролю на стороні сервера. Наприклад:

```
SELECT Id, Username FROM Users
WHERE Username = 'john_doe';
```

Далі порівнюється PasswordHash з хешем введеного користувачем пароля на сервері. Перед тим як зберегти повідомлення, воно шифрується за допомогою публічного ключа отримувача. Це унеможливлює доступ до вмісту повідомлення навіть у разі компрометації бази даних. При цьому забезпечується високий рівень захисту персональних даних та відповідність сучасним стандартам безпеки. Потім запит виглядає так:

```
INSERT INTO Messages (SenderId, ReceiverId, EncryptedText)
VALUES (1, 2, 'wN42jB93...');
SELECT * FROM Messages
WHERE (SenderId = 1 AND ReceiverId = 2)
   OR (SenderId = 2 AND ReceiverId = 1)
ORDER BY SentAt ASC;
```

Цей запит виводить всі повідомлення між двома користувачами у хронологічному порядку. Для підвищення продуктивності доступу до повідомлень, можна створити індекси:

```
CREATE INDEX IDX_Sender ON Messages(SenderId);
CREATE INDEX IDX_Receiver ON Messages(ReceiverId);
```

Такі індекси покращують швидкість вибірки при фільтрації повідомлень по відправнику або отримувачу. Також опишемо структуру, додавши таблиці:

1. UserSessions для зберігання сесій JWT;
2. Devices для підтримки входу з кількох пристроїв;
3. DeletedMessages логування видалених повідомлень з міркувань безпеки;
4. MessageStatus (наприклад: відправлено, отримано, прочитано);
5. Attachments якщо буде реалізовано передавання файлів.

Усі сутності (entity-класи), які використовуються для представлення таблиць у базі даних, описуються за допомогою файлів з розширенням .cs, тобто у вигляді C#-класів. Такий підхід є стандартною практикою при роботі з ORM (Object-Relational Mapping) системою Entity Framework, яка широко використовується у розробці застосунків на платформі .NET. Кожен клас відповідає окремій таблиці у базі даних, а його властивості — стовпцям цієї таблиці. Така структура дозволяє розробникам зручно працювати з даними на рівні об’єктів, без необхідності

безпосередньо писати SQL-запити.

Однією з важливих переваг використання Entity Framework є можливість автоматичного створення, оновлення та виконання міграцій. Міграції дозволяють трекати зміни у структурах класів-сутностей та відповідно оновлювати структуру бази даних. Завдяки цьому підтримується консистентність між кодом застосунку та реальною БД. Наприклад, якщо до сутності додається нова властивість, достатньо згенерувати міграцію, яка створить відповідний стовпець у таблиці бази даних.

Ще однією суттєвою перевагою є формування проміжного шару (абстракції) між серверною логікою застосунку та безпосередньо базою даних. Це дозволяє розробникам працювати з даними у вигляді об'єктів, не піклуючись про низькорівневі деталі взаємодії з SQL-сервером. Такий підхід зменшує ризик помилок, сприяє кращій підтримуваності коду та пришвидшує процес розробки.

З точки зору безпеки, Entity Framework також забезпечує базовий захист від деяких поширених типів кібератак. Наприклад, при правильному використанні ORM значно знижується ризик SQL-ін’єкцій, оскільки всі запити до БД формуються через скомпільовані методи з параметрами, а не через ручне зшивання строк. Крім того, EF може автоматично управляти транзакціями, що допомагає зберігати цілісність даних навіть у випадку складних операцій або виникнення помилок.

```
public class Chat
{
    public Guid Id { get; set; }
    public Guid CreatedByUserId { get; set; }
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    [Required]
    public Guid DeviceAId { get; set; }
    public Device? DeviceA { get; set; }
    [Required]
    public Guid DeviceBId { get; set; }
    public Device? DeviceB { get; set; }
    public string EncryptedSymmetricKeyForA { get; set; } = string.Empty;
    public string EncryptedSymmetricKeyForB { get; set; } = string.Empty;
    public ICollection<Message> Messages { get; set; } = new List<Message>();
```

```
}
```

У системі "SecureMessages" одним із ключових елементів безпеки є процес зберігання паролів користувачів. Для цього використовується стійкий до атак метод хешування із застосуванням криптографічної "солі", що унеможливлює зворотне відновлення пароля навіть у разі компрометації бази даних. Алгоритми типу bcrypt або PBKDF2 дозволяють багаторазово хешувати вхідне значення, що істотно ускладнює перебір варіантів шляхом атаки типу "brute force". Це є загальновизнаним стандартом у галузі інформаційної безпеки й гарантує високий рівень захисту персональних даних.

Ще одним важливим принципом побудови безпечної системи є жорстке обмеження прямого доступу до бази даних. Уся взаємодія з нею має здійснюватися виключно через серверне API, яке виконує роль посередника між клієнтською частиною та сховищем даних. Такий підхід повністю усуває ризики, пов’язані з потенційним витоком даних або несанкціонованими спробами доступу до бази з боку клієнта.

Завдяки цьому модель взаємодії стає контрольованою: клієнт не має жодної змоги напряму формувати SQL-запити чи змінювати структуру бази даних. Уся логіка обробки даних, перевірки прав доступу, фільтрації та валідації вхідних значень реалізується на серверному рівні.

Крім того, у межах такої архітектури обмежується можливість маніпуляцій з хешами, токенами або іншими важливими даними, які можуть передаватися клієнтом. Сервер перевіряє їхню автентичність і валідність, не покладаючись на сам клієнт, що дозволяє зберігати цілісність і достовірність усіх транзакцій.

Забезпечення безперервності роботи системи також включає реалізацію політики резервного копіювання. Зберігання таких копій бажано організовувати на окремих серверах або в надійних хмарних середовищах, де вони можуть бути оперативно відновлені у випадку збоїв або атак. Це дозволяє не лише убезпечити дані, а й забезпечити безперебійний доступ до сервісу в разі аварійних ситуацій. На рис. 1.11 зображена схема бази даних, згенерована автоматично на основі існуючих в ній таблиць.

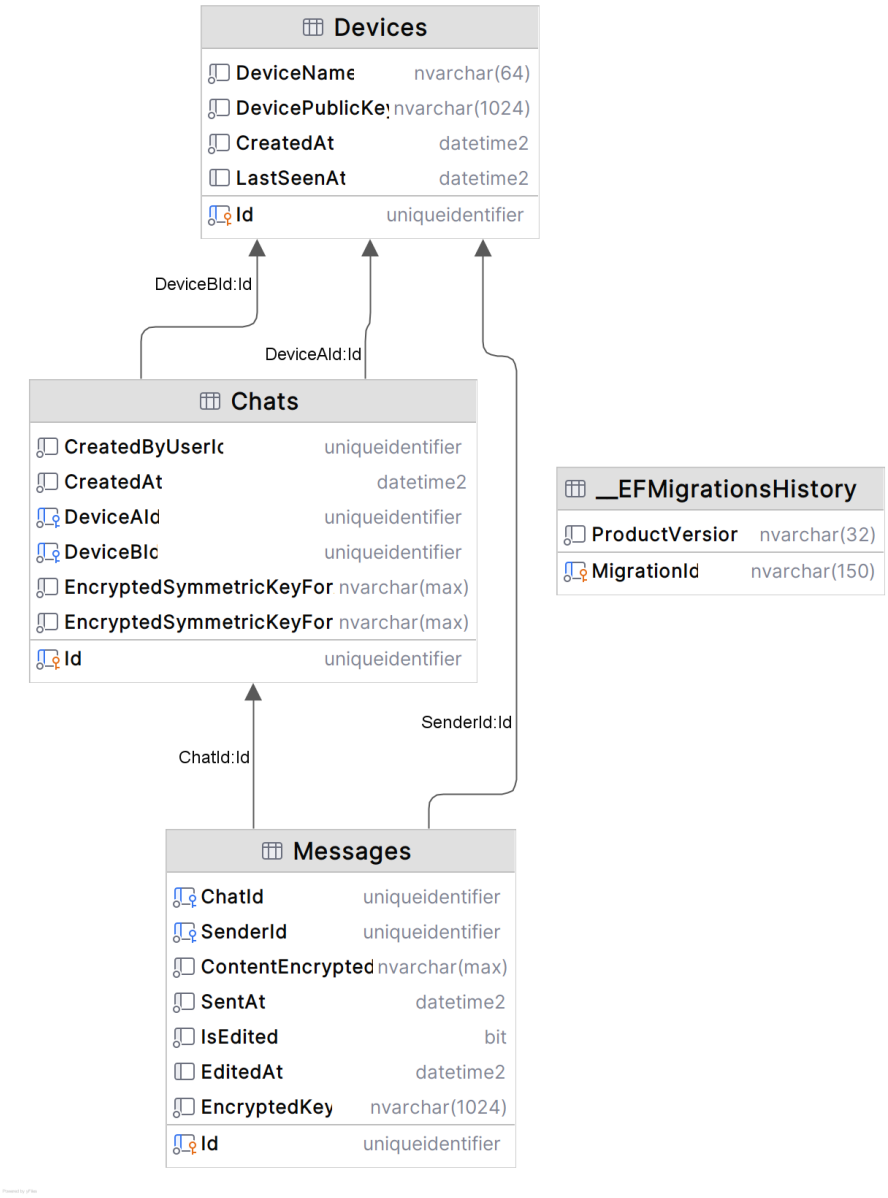


Рисунок 1.11. Схема бази даних

Структура бази даних є простою, але гнучкою й безпечною для реалізації ключової функціональності застосунку. При чіткому розділенню обов'язків таблиць та оптимальній схемі зв'язків забезпечується масштабованість, продуктивність і відповідність найкращим практикам зберігання персональних даних у сучасних інформаційних системах.

На рис. 1.12 наведено приклад Execution plan при створенні бази даних – це діаграма, за допомогою якої можна оцінити складність та вкладеність будь-якого запиту. Створення бази даних можна поділити на дві частини: створення таблиць та індексів поміж ними.

Рисунок 1.12. Execution plan додатку.

1.2.5 Взаємодія в реальному часі через SignalR

Однією з ключових переваг сучасних месенджерів є можливість забезпечити обмін повідомленнями в режимі реального часу. У проєкті "SecureMessages" цю задачу вирішено шляхом інтеграції бібліотеки SignalR технології від Microsoft, яка дозволяє встановлювати постійне з'єднання між клієнтом і сервером для миттєвої передачі даних.

SignalR використовує веб-сокети як основний протокол, однак автоматично може переходити до інших транспортів (наприклад, Server-Sent Events або Long Polling), якщо браузер не підтримує WebSockets. Така адаптивність гарантує високу доступність функціональності на різних пристроях і в різних умовах мережі. У загальному вигляді архітектура роботи SignalR у SecureMessages виглядає так:

1. Клієнт підключається до хабу SignalR при вході в застосунок;
2. Хаб авторизує користувача, додаючи його до відповідної групи на основі ID;
3. При відправленні повідомлення воно викликається метод хабу, який пересилає зашифроване повідомлення іншому користувачу, та шифрується ключем безпеки SHA256;
4. Клієнт, що приймає, отримує повідомлення у своєму браузері та розшифровує його локально.

```
public class ChatHub : Hub
{
    public async Task SendMessage(string receiverUsername, string encryptedMessage)
    {
        var receiverConnection = ConnectedUsers.GetConnectionId(receiverUsername);
        if (receiverConnection != null)
```

```
            {
                await    Clients.Client(receiverConnection).SendAsync("ReceiveMessage",
encryptedMessage);
            }
        }

        public override Task OnConnectedAsync()
        {
            var username = Context.User.Identity.Name;
            ConnectedUsers.Add(username, Context.ConnectionId);
            return base.OnConnectedAsync();
        }

        public override Task OnDisconnectedAsync(Exception? exception)
        {
            var username = Context.User.Identity.Name;
            ConnectedUsers.Remove(username);
            return base.OnDisconnectedAsync(exception);
        }
    }
```

У проєкті SecureMessages клас ChatHub виступає як центральний вузол обміну повідомленнями між користувачами в режимі реального часу. Основна логіка зосереджена у методі SendMessage, який приймає ім’я отримувача та зашифроване повідомлення. Після цього сервер перевіряє, чи є підключення до вказаного користувача в словнику ConnectedUsers, і якщо так миттєво надсилає йому повідомлення через метод Clients.Client(...).SendAsync(...). При такому підходу не потрібно опитувати сервер кожні кілька секунд, а обмін відбувається миттєво.

Ключовим моментом у забезпеченні надійності комунікації є методи OnConnectedAsync і OnDisconnectedAsync. Вони автоматично спрацьовують при підключенні або відключенні користувача та оновлюють словник ConnectedUsers, де зберігається відповідність між іменами користувачів і їх SignalR-з'єднаннями (ConnectionId). Це дозволяє підтримувати актуальну карту активних сесій і надсилати повідомлення точно за призначенням. Така зв’язка авторизації, реєстрації з’єднань і точкового надсилання повідомлень формує надійний і масштабований фундамент для системи захищеного обміну повідомленнями в реальному часі, миттєво.

Клас ChatHub тісно інтегрується з механізмами авторизації ASP.NET – Context.User.Identity.Name автоматично містить ім’я автентифікованого користувача, що дозволяє гарантувати, що тільки авторизовані особи можуть підключатись до хабу.

Даний хаб дозволяє передавати повідомлення між користувачами, зберігаючи їхні з'єднання в кеші ConnectedUsers. Це мінімізує час затримки та забезпечує стабільний зв’язок.

```
const connection = new signalR.HubConnectionBuilder()
    .withUrl("/chathub")
    .build();

connection.on("ReceiveMessage", function(encryptedMessage) {
    const decrypted = decryptMessage(encryptedMessage);
    displayMessage(decrypted);
});

await connection.start();
```

Клієнт підключається до хабу, очікує повідомлення, розшифровує його і відображає на екрані користувача. Це дозволяє досягти реального часу в комунікації, що важливо для будь-якого месенджера.

Незважаючи на використання веб-сокетів, SignalR у SecureMessages працює виключно через HTTPS-з'єднання, що виключає ризик перехоплення трафіку.

Повідомлення передаються вже в зашифрованому вигляді, тому навіть у разі втрати шифрування каналу, зловмисник не зможе розшифрувати вміст. Переваги використання SignalR

1. Мінімальні затримки між надсиланням і відображенням повідомлень;
2. Масштабованість SignalR можна поєднувати з Redis для розподілених хабів;
3. Гнучкість підтримка fallback-транспортів для старих браузерів;
4. Інтеграція з ASP.NET дозволяє зручно реалізовувати авторизацію, логіку викликів та логування.

Інтеграція SignalR у проєкті стала важливим кроком до побудови повноцінного месенджера, здатного забезпечувати взаємодію між користувачами без затримок.

SignalR вирізняється високою продуктивністю та надійністю, а також достатньо простою інтеграцією в екосистему ASP.NET Core, що дозволяє швидко реалізувати стабільну двосторонню взаємодію між клієнтом і сервером. Водночас ця технологія автоматично обирає найбільш ефективний транспорт для передавання даних, орієнтуючись на можливості браузера та мережеве оточення, що робить її універсальною для широкого спектра пристроїв.

1.3 Тестування додатку

1.3.1 Тестування та відправка повідомлень

Одним із ключових завдань у процесі тестування застосунку стало перевіряння коректності механізму відправлення та прийому повідомлень між користувачами. Оскільки основна функціональність системи полягає у безпечній та стабільній передачі даних у режимі реального часу, особливу увагу було приділено взаємодії між клієнтами через вебсокети, а саме — через бібліотеку SignalR, яка використовується у проєкті.

Для того щоб максимально наблизити тестування до умов реального використання, було прийнято рішення використовувати два одночасно відкриті браузери — Google Chrome та Microsoft Edge. Кожен із них виступав окремим незалежним користувачем у системі. Такий підхід дозволив емулювати живу розмову між двома пристроями, ніби два реальні користувачі знаходяться в різних місцях і надсилають один одному повідомлення.

На початковому етапі тестування кожен з браузерів генерував унікальний Device ID — ідентифікатор пристрою, який використовується для первинної ініціалізації з'єднання через SignalR. Цей Device ID слугує додатковим засобом ідентифікації конкретної сесії користувача, що дозволяє системі правильно маршрутизувати повідомлення між відповідними клієнтами та забезпечити точну синхронізацію даних.

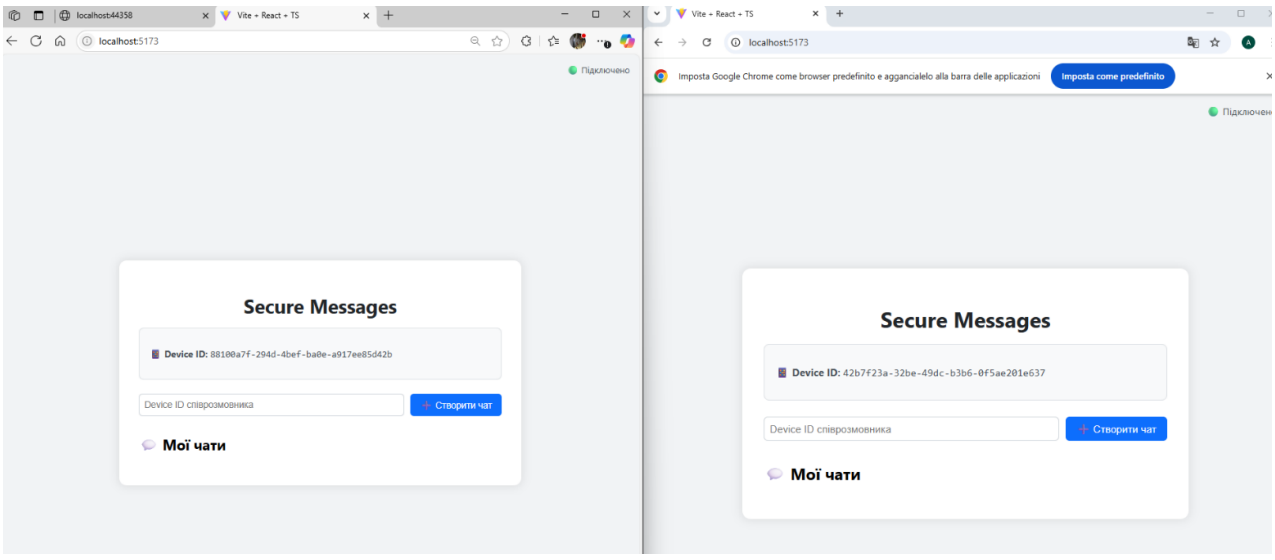


Рисунок 1.13. Початкова сторінка додатку

На сторінці користувача реалізовано функціональність для ініціації нового чату, яка починається з введення ID співрозмовника у спеціальне поле. Цей ідентифікатор використовується для пошуку іншого користувача в системі та формування зв’язку між двома пристроями. Після натискання кнопки «Створити чат» запускається логіка, що відповідає за перевірку існування відповідного запису в базі даних і, за потреби, створення нового чату, який прив’язується до ID обох учасників.

Встановлення зв’язку між користувачами реалізується на рівні структури бази даних, де чат зберігає посилання на обидва пристрої або акаунти, які беруть участь у спілкуванні. Після успішного створення такого зв’язку, новий запис одразу з’являється в інтерфейсі обох учасників — у розділі «Мої чати». У цьому списку відображається ID іншого пристрою або користувача, що дозволяє швидко переходити до переписки.

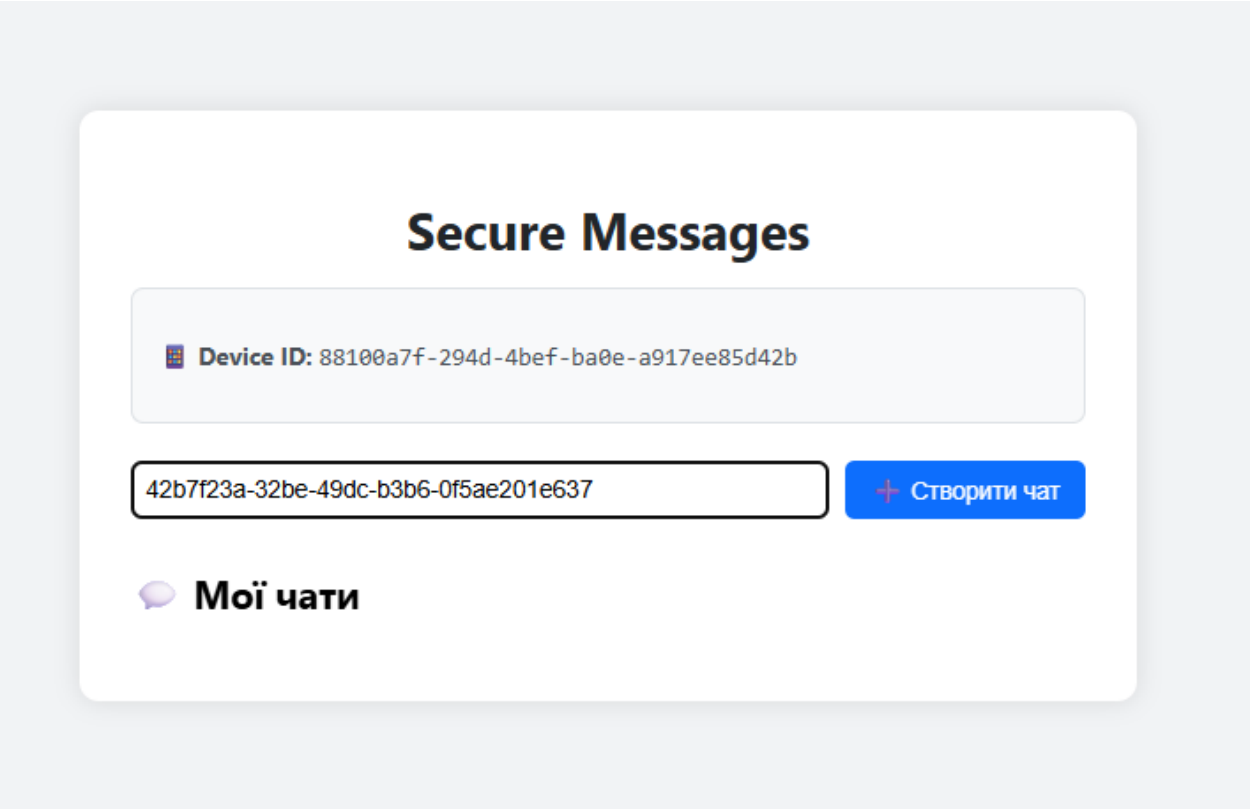


Рисунок 1.14. Додавання коду користувача до чату

Кожен із клієнтів має змогу перейти до відповідного діалогу, в якому реалізовано розділення повідомлень за джерелом – свої та співрозмовника відображаються по-різному: з відповідним стилем та підписом.

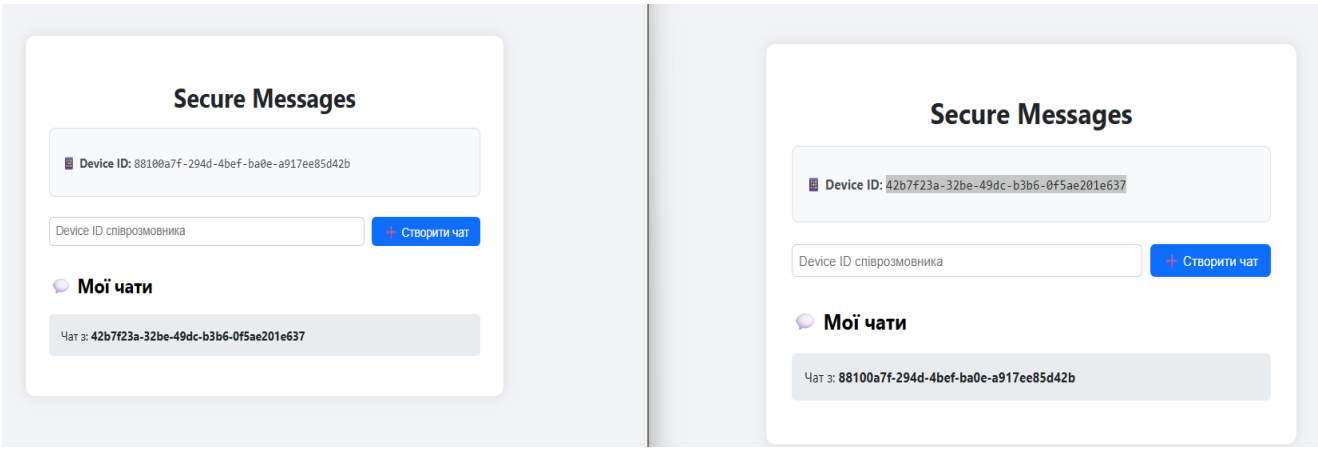


Рисунок 1.15. Вигляд вікон після додавання користувача

Для перевірки миттєвості доставки повідомлень користувачі по черзі вводили повідомлення в поле введення та спостерігали за появою їх у вікні співрозмовника. Як видно, затримка практично відсутня, а повідомлення передається збереженим та коректно розшифрованим.

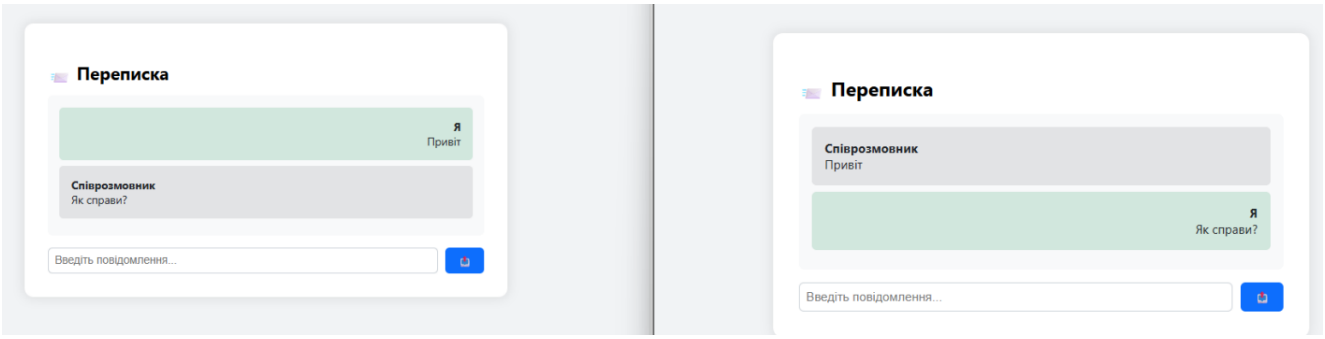


Рисунок 1.16. – Вигляд вікон з повідомленнями

Це підтверджує, що механізм реального часу через SignalR працює стабільно навіть при одночасній роботі з двома клієнтами локально.

Також у консолі браузера фіксується лог підключення до WebSocket з деталями з’єднання.

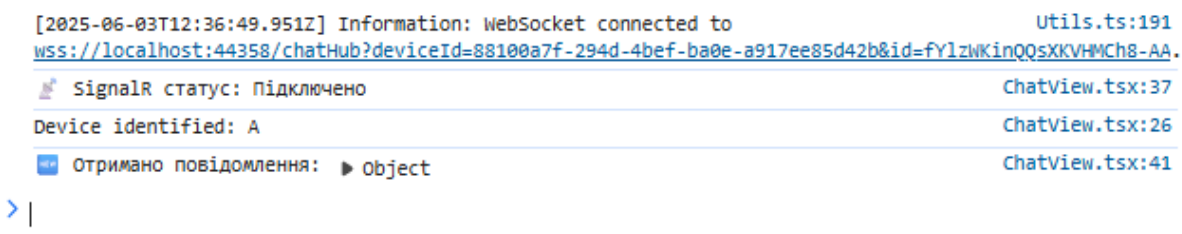


Рисунок 1.15. Вигляд респонсу після підключення користувача

Видно, що клієнт підключається до хабу через захищене з’єднання

wss://localhost:<порт>, передаючи параметри deviceId і id, що дозволяє серверу правильно ідентифікувати клієнта та направити повідомлення за призначенням. Це надзвичайно важливий момент у системі, яка покладається на унікальність пристрою для ідентифікації користувача.

1.3.2 Обробка запиту та нетворкінг

На рівні мережевої взаємодії застосунок демонструє ефективну та злагоджену інтеграцію з бекендом, реалізованим на платформі ASP.NET Core. Така архітектура дозволяє забезпечити як надійність зберігання даних, так і їхню оперативну передачу у режимі реального часу, що є критично важливим для сучасного месенджера, орієнтованого на безпечне спілкування.

Під час створення або надсилання нового повідомлення на клієнтському боці формується POST-запит до RESTful API за адресою POST /api/chat/message. Цей запит містить у тілі наступні параметри:

- chatId – ідентифікатор чату, до якого належить повідомлення;
- encryptedText – зашифрований вміст повідомлення, що гарантує конфіденційність навіть у разі перехоплення трафіку;
- senderDeviceId – унікальний ідентифікатор пристрою, з якого було надіслано повідомлення.

Саме SignalR тут відіграє роль каналу для миттєвого сповіщення, дозволяючи отримувачеві побачити нове повідомлення одразу після його надсилання, без необхідності ручного оновлення сторінки або опитування сервера. Це створює ефект "живого чату", де повідомлення надходять та з’являються практично миттєво, навіть за умов високого навантаження.

На стороні сервера запит обробляється контролером, де запит валідується, створюється запис в базі даних та відправляє повідомлення в реальному часі.

Таке поєднання надійного REST API для обробки та збереження даних разом із SignalR для обміну в реальному часі забезпечує високу масштабованість, швидкодію і безпеку. Подібна гібридна модель дозволяє ефективно розділити відповідальність між різними шарами системи: API відповідає за бізнес-логіку та роботу з базою, тоді як SignalR за динамічну комунікацію між користувачами.

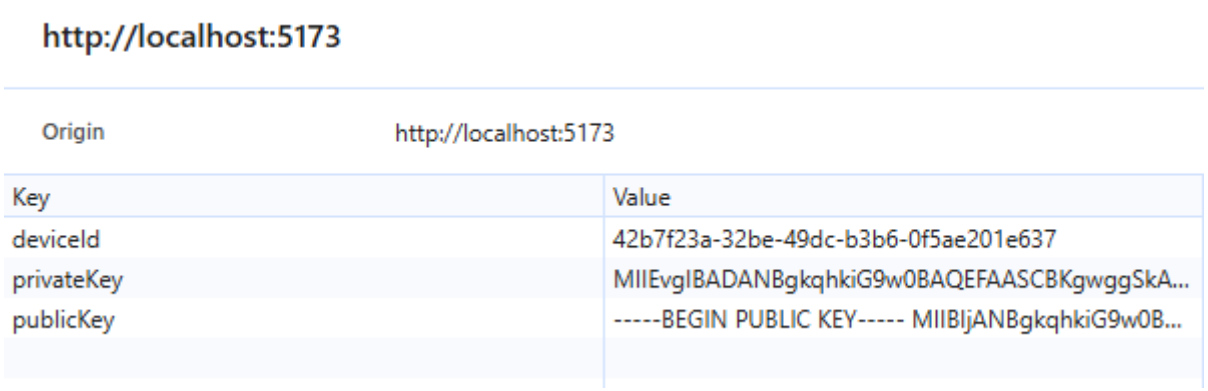


Рисунок 1.17. Збереження даних у сховищі

У заголовках запиту фіксується коректне використання CORS, підтримка credentials та дозволені методи що свідчить про дотримання безпечної міждоменної взаємодії.

У вікні вкладки Response видно результат обробки запиту сервером, який зображено на рис. 1.18.

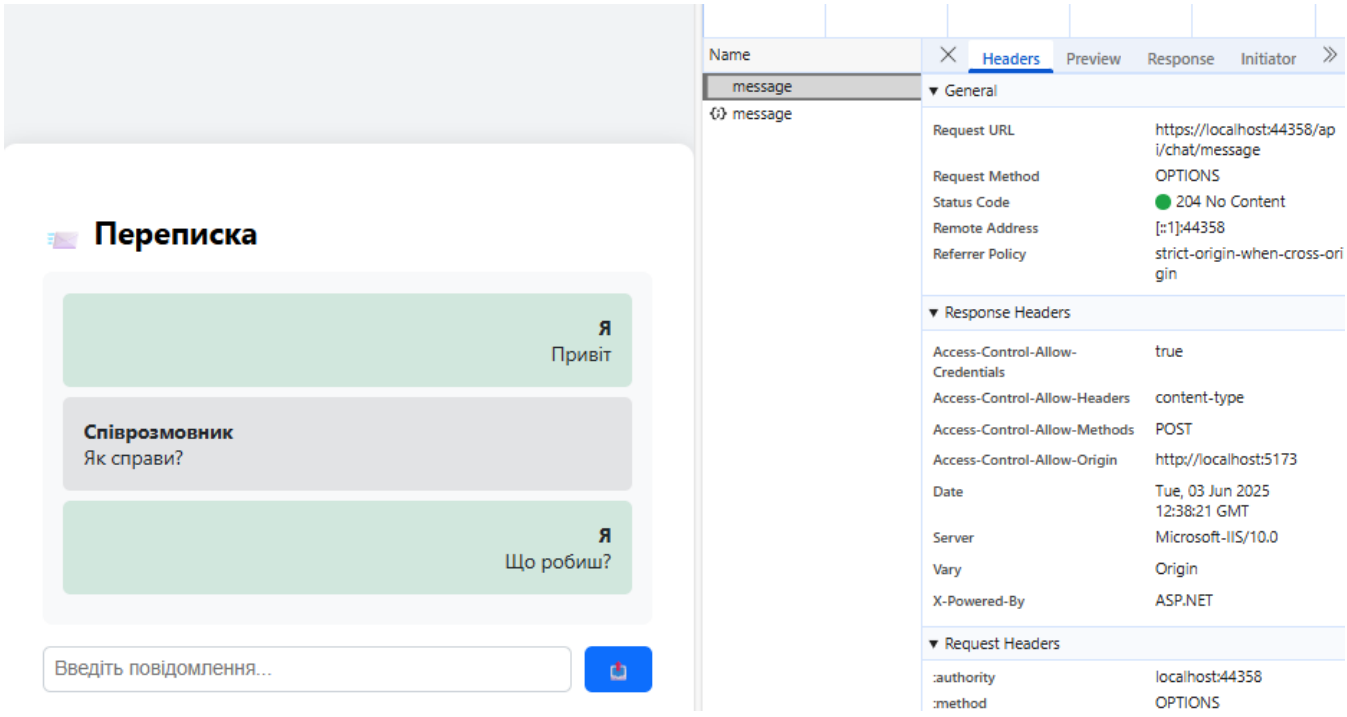


Рисунок 1.18. Відправка даних до серверу

Сервер повертає об'єкт повідомлення, який включає всі метадані: час надсилання, ідентифікатор відправника, зашифрований контент і Device ID автора. Це дозволяє клієнту без зайвих запитів оновити інтерфейс і відобразити нове повідомлення. Аналогічна структура представлена й у вкладці Preview, де видно об'єкт у структурованому вигляді.

Рисунок 1.19. Вигляд респонсу з серверу після відправки повідомлення

У Payload можна побачити точний зміст відправленого запиту ID чату, зашифроване повідомлення, та Device ID користувача, що надсилає повідомлення. Приклад запиту на сервер зображено на рис. 1.19.

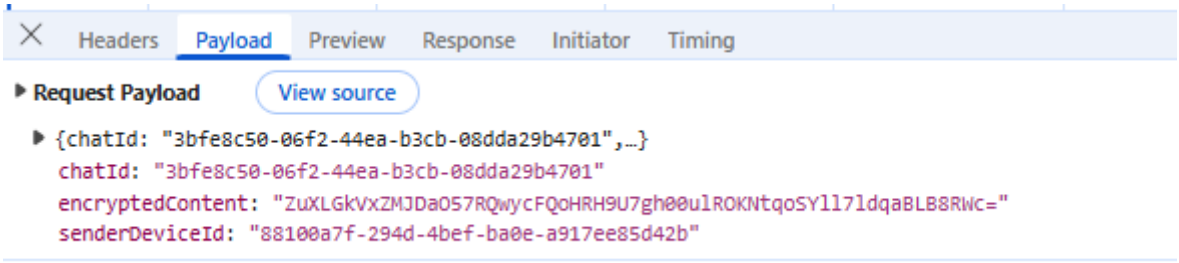


Рисунок 1.20. Вигляд відправки даних з клієнту

Це дозволяє серверу не лише зберегти запис у БД, але й одразу передати його в SignalR-хаб для миттєвої доставки іншому клієнту.

Механізм запитів та відповіді в SecureMessages працює максимально прозоро і ефективно. Всі етапи від натискання кнопки «Відправити» до отримання повідомлення співрозмовником відбуваються в межах кількох мілісекунд. Це досягається до поєднанні правильної структури API, оптимізованої роботи SignalR та асинхронної обробки запитів на сервері. Використання сучасних стандартів HTTPS, WebSocket і локального шифрування повідомлень гарантує як швидкість, так і конфіденційність спілкування користувачів.

Для того, щоб переконатися в тому, що шифрування повідомлень дійсно відбувається на практиці, можна здійснити перевірку безпосередньо через доступ до бази даних, у якій зберігається вміст чатів. Це можна зробити, використовуючи SQL-запити для вибірки відповідних записів. На рис. 1.20 наведено приклад такої вибірки, з якого чітко видно, що всі повідомлення в базі даних зберігаються у

зашифрованому вигляді.

Особливо важливо підкреслити, що сервер, на якому розміщена база даних, не має доступу до жодного з ключів, необхідних для розшифрування цих повідомлень. Уся логіка шифрування й дешифрування реалізується на стороні клієнта, що повністю виключає можливість прочитання повідомлень з боку серверної інфраструктури. Це означає, що навіть у разі несанкціонованого доступу до самої бази даних. Наприклад, у результаті злому або витоку зловмисник не зможе витягнути жодної корисної інформації зі вмісту повідомлень.

Шифровані дані, які він потенційно може отримати, не мають сенсу без відповідного симетричного ключа, який знаходиться лише у користувачів, що беруть участь у чаті. Завдяки цьому підходу реалізується принцип наскрізного шифрування (end-to-end encryption), при якому доступ до відкритого тексту повідомлень мають виключно кінцеві учасники комунікації.

ChatId	ContentEncrypted
62D1F0A0-E57D-4A1B-B5E5-08DDA817F156	5Cr/kkcg4zFEionz/qDik0opvanPt0zgjCW+sHoKrDSc
62D1F0A0-E57D-4A1B-B5E5-08DDA817F156	x2yXN5iOU4Jzok2zQGg/G9u8cc2VFFkp11eNhwroEcBk
62D1F0A0-E57D-4A1B-B5E5-08DDA817F156	4NUA8WuuyjqX181jHmSplsetD7LXelFXjYh1sUcgErkBTcVboOys
62D1F0A0-E57D-4A1B-B5E5-08DDA817F156	EcwxEfls+Mb1tkss9m7TY7HoXB84l6GCI+qT2jPke9l=
62D1F0A0-E57D-4A1B-B5E5-08DDA817F156	bZvhcwpG7mZ4LNmW3kLD0dWwAuAdDQOZiZ41imQ=
62D1F0A0-E57D-4A1B-B5E5-08DDA817F156	3eLFbRF8YiKhj3EiWWH8DMWFa/y3djAx7UZjKnZP6XoERifDF...
62D1F0A0-E57D-4A1B-B5E5-08DDA817F156	ZjLSkC7k906g8xnRCLBVaS95iOfxaKUamRF9paHwSw3V0w==
62D1F0A0-E57D-4A1B-B5E5-08DDA817F156	9YYRjUG4Tf8WhCPEa6ezDHridnQ1tMoRVAm4Vqa82Zb1NK8y...
62D1F0A0-E57D-4A1B-B5E5-08DDA817F156	XyGPAXf3ffG3PBdSgehLjeHw0QbcqgNPoxyRWKFNFDY=
0E19CF97-D389-4CD1-B5E6-08DDA817F156	r524GpgC4KqTmknM8xc385QjKCc+ZVir4SU0jsDM
0E19CF97-D389-4CD1-B5E6-08DDA817F156	GQxkuSOu8fmLX/k7AhKuoa9qK5RepxGjytbLX/mz
0E19CF97-D389-4CD1-B5E6-08DDA817F156	QLwZl7TD99xxJuRlrNVDzv6hRAsR+5SAbMFjMV0/atJgg1o=
0E19CF97-D389-4CD1-B5E6-08DDA817F156	bmV0LLTSg9AfJPbsL5WgRYlhGCFaXtEWP5EUiFJ1MVYJEg==
0E19CF97-D389-4CD1-B5E6-08DDA817F156	6u/JZV8dMNU68ZeprsDdtI7rChgQISSIs36Lr4RFLn4xKQf4
0E19CF97-D389-4CD1-B5E6-08DDA817F156	k3AIF84Md+J6WrWjclrAINAOtt6dVyHXMGcd/2KqZYVxM6Ag
0E19CF97-D389-4CD1-B5E6-08DDA817F156	/x5bVQsnFBu7KSYOWheSeWFlfGqwhAtFjv1bJxKdC4/S49o=

Рисунок 1.20 Зашифрований контент повідомлень

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

У дипломному проєкті створено застосунок для обміну повідомлень між пристроями з просунутим шифруванням до кінцевого користувача. Розроблена клієнтська та серверна частина дозволяють безпечно обмінюватися повідомленнями, які можуть бути розшифровані лише відправником або отримувачем, що дозволяє забезпечити найвищий рівень безпеки та унеможливити компрометацію у випадку перехоплення повідомлень.

2.2 Визначення трудомісткості розробки ПЗ

Тривалість розробки програмного продукту залежить від його обсягу, складності, кваліфікації розробників і встановлених ринком термінів. Метод структурної аналогії дозволяє оцінити обсяг у тисячах умовних машинних команд на основі подібного програмного забезпечення.

Табл. 2.1 містить аналоги ПЗ з подібними функціями; обраний варіант виділено сірим.

Таблиця 2.1. Каталог аналогів

Найменування ПП	**Обсяг функції ПП – V_o, умовних. машинних командах**
1. ПП автоматизації засобів по каталогу	680 – 7000
2. ПП автоматизованих розрахунків	1300 – 8600
3. ПП введення інформації	1060 – 5750

Після вибору аналога з обсягом Vo (умовні машинні команди), трудомісткість визначається за табл. 2.2.

Таблиця 2.2. Обсяг ПП

Обсяг ПП, тис.умов.машинних команд	**Норма часу, люд/год**
1.00	229
2.00	244
3.00	262
4.00	283
5.00	306
6.00	330
7.00	357

Продовження таблиці 2.1

Обсяг ПП, тис.умов.машинних команд	Норма часу, люд/год
8.00	385
9.00	414
10.00	445

На основі отриманого значення за довідником визначають укрупнену норму часу, скориговану коефіцієнтом $К_к$ = 0,7–0,8 для умов розробки на комп'ютері:

$$T_{ар} = 306 \text{ x } 0,8 = 244,8 \text{ (люд/годин)} \quad (2.1)$$

Трудомісткість визначається для кожного етапу окремо, з урахуванням складності, новизни та використання стандартних модулів, за відповідними формулами:

$$T_{ТЗ} = T^a p \times L_1 \times K_H \quad (2.2)$$

$$T_{ТП} = T^a p \times L_2 \times K_H \quad (2.3)$$

$$T_{РП} = T^a p \times L_3 \times K_H \times K_T \quad (2.4)$$

Для розрахунку використовуються такі коефіцієнти:

- Li – частка і-го етапу (табл. 2.3);
- Кн – коефіцієнт новизни (табл. 2.4);
- Кт – коефіцієнт використання типових програм (табл. 2.5).

Наш варіант виділено сірим.

Таблиця 2.3. Питомі коефіцієнти трудомісткості стадії у загальній трудомісткості розробки ПП

Код стадії	Ступінь новизни		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Таблиця 2.4. Значення коефіцієнта новизни

Код ступеня новизни	Ступінь новизни	Значення $К_н$
А	Принципово новий ПП	1,75 – 1,2
Б	ПП – розвиток визначеного параметричного ряду	1,0 – 0,8

Продовження таблці 2.4

Код ступеня новизни	Ступінь новизни	Значення $К_н$
В	ПП, яке має аналог	0,7

Таблиця 2.5. Значення коефіцієнта використання типових програм

Ступінь охоплення реалізованих функцій розроблювального ПП типовими програмами, %	Значення $К_т$
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

Тепер розраховуємо трудомісткість для всіх етапів і зводимо у табл. 2.6:

Трудомісткість технічного завдання:

$$T_{тз}=Ta*L_1*K_н=244,8*0,12*0,8= 23,50 \text{ (люд/годин)} \quad (2.2)$$

Трудомісткість розробки технічного проєкту:

$$T_{тп}=Ta*L_2*K_н =244,8*0,15*0,8 = 29,37 \text{ (люд/годин)} \quad (2.3)$$

Трудомісткість розробки робочого проєкту:

$$T_{рп}=Ta*L_3*K_н*K_т=244,8*0,58*0,8*0,7= 79,51 \text{ (люд/годин)} \quad (2.4)$$

Для розрахунків визначили обсяг документації по етапах:

- технічне завдання $N_{тз}$=2 (стор);
- розробка ТП $N_{тп}$=43 (стор);
- розробка робочого проєкту $N_{рп}$=9 (стор);
- пояснювальна записка відповідно $N_{пз}$=15 (стор).

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, годин		
1.ТЗ	$T_{Pтз}$=23,50	$T_{кк}=0,7*N_{тз}$= 0,7*2 =1,4	$T_{нк}=0,15*N_{тз}$ =0,15*2=0,30
2.Розробка ТП	ТРтп=29,37	Ткк=0,7*Nтп= 0,7*43=30,1	Тнк=0,15*Nтп =0,15*43=6,45
3.Розробка РП	ТРрп= 79,51	Ткк=0,7*Nрп= 0,7*9=6,3	Тнк=0,15*Nрп= 0,15*9=1,35
4.Розробка ПЗ	Тпз=1,5*Nпз= 1,5*15 =22,5	Ткк=0,7*Nтз= 0,7*15=10,5	Тнк=0,15*Nпз= 0,15*15 =2,25

Продовження таблиці 2.6

Усього, в т.ч.:	213,53		
- на розробку	Тр=154,88		
- контроль керівника		Ткк=48,3	
- нормоконтроль			Тнк=10,35

2.3 Розрахунок ціни програмного продукту

Розраховуємо основну зарплату виконавців, матеріальні та загальні витрати на розробку ПП. Зарплата наведена в табл. 2.7. З 1 квітня 2024 мінімальна місячна зарплата – 8000 грн, погодинна ставка – 46 грн (згідно зі ст. 8 Закону про Держбюджет України).

Таблиця 2.7. Розрахунок основної заробітної плати виконавців

Найменування робіт	**Трудомісткість робіт, години**	**Погодинна тарифна ставка, грн.**	**Розрахунок, грн.**
1.Розробка ПП	154,88	55,35	8573,48
2.Контроль керівника	48,3	120,28	5810,00
3.Нормоконтроль	10,35	120,28	1245,00
Усього	-	-	Зо= 15628,48

Розрахуємо матеріальні витрати на розробку ПП та наведемо їх у табл. 2.8.

Таблиця 2.8. Розрахунок матеріальних витрат на розробку

Найменування матеріальних витрат	**Тип, модель**	**Кількість**	**Ціна одиниці, грн.**	**Вартість, грн.**
Папір	Лист А4	70	4.0	280,0
Разом	-	-	-	$В_{мі}$ =280,0
Транспортно – заготівельні Витрати (10%)				$В_{тр\_з}$ = 0.1 * $В_{м1}$ = 0,1*280 =28,00
Усього				$В_м$=$В_{мі}$+$В_{тр\ з=}$ 308,00

За отриманими даними складена калькуляція планової собівартості ПП, наведена в табл. 2.9.

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	**Значення, грн.**	**Формула розрахунку**
1. Матеріали	308,00	$В_м$ (див. табл. 2.8)
2. Основна заробітна плата	15628,48	$З_о$ (див. табл. 2.7)
3.Додаткова заробітна плата	1562,84	$З_д = 0.1 * З_о = 15628,48*0,1$
4.Відрахування до єдиного фонду соціального внеску	3864,37	$В_{є.с.в.} = 0.22 * (З_о + З_д) = 0,22*$ (15628,48+1562,84)
5. Накладні витрати	6251,39	$В_{нак.} = 0.4 * З_о = 0.4 * 15628,48$
6. Повна собівартість	27681,08	$С_{пов} = В_м + З_о + З_д + В_{є.с.в.} + В_{нак.} =$ 308,00+15628,48+1562,84+3864,37 + 6251,39

Розмір прибутку розраховується за формулою:

$$П=(С_п*Р)/100=(27681,08*10)/100=2768,11 \text{ грн.} \quad (2.5)$$

Де р – плановий рівень рентабельності (10-15%).

Оптова ціна розраховується за формулою:

$$Ц_о=С_п+П=27681,08+2768,11=30449,19 \text{ грн.} \quad (2.6)$$

За отриманими даними, ціна реалізації ПП за формулою становить:

$$Ц_р=Ц_о+ПДВ=30449,19+30449,19*0.2=36539,03 \text{ грн.} \quad (2.7)$$

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

3.1 Резюме

Забезпечення безпеки праці є не лише основою створення комфортних умов для працівників, а й ключовим фактором збереження їхнього здоров'я та працездатності, що, у свою чергу, впливає на ефективність роботи та прибутковість підприємства. Досягти належного рівня безпеки можливо лише за умови суворого дотримання норм трудового законодавства, державних стандартів України, а також галузевих вимог, спрямованих на охорону праці.

У цьому розділі дипломної роботи розглядаються умови праці програміста (оператора ПК), які мають бути забезпечені на підприємстві для організації безпечного робочого процесу, а також потенційні небезпечні чинники, санітарно-гігієнічні умови та інші аспекти, пов'язані з розробкою веб-системи.

3.2 Оцінка ризиків та впливу шкідливих чинників на здоров'я оператора ПК під час створення програмного забезпечення

До небезпечних факторів належать такі умови праці, які за певних обставин можуть стати причиною виробничих травм або раптового погіршення стану здоров'я працівника. Якщо ж фактор викликає поступове погіршення здоров'я, зокрема розвиток хронічних захворювань або зниження працездатності, його відносять до шкідливих. Варто зазначити, що за тривалого чи інтенсивного впливу шкідливі фактори можуть перетворитися на небезпечні.

У процесі роботи за персональним комп'ютером працівник зазнає впливу низки небезпечних і шкідливих чинників, серед яких:

- невідповідність мікроклімату встановленим нормам;
- недостатня освітленість робочої зони;
- ризик ураження електричним струмом;
- дія статичної електрики;
- нераціональне розташування обладнання та незручна організація робочого місця тощо.

3.3 Нормативні вимоги до виробничого середовища

Відповідно до вимог Правил охорони праці при роботі з електронно-обчислювальною технікою, робоче місце користувача персонального комп'ютера має бути організоване таким чином, щоб забезпечити максимально ефективну та безпечну діяльність. Далі розглянемо основні нормативні положення щодо умов виробничого середовища.

3.3.1 Вимоги до приміщень з відеотерміналами

Приміщення, в яких встановлені відеотермінали, доцільно орієнтувати вікнами на північ або північний схід. Вікна повинні бути обладнані регульованими жалюзі або шторами, що дозволяють повністю затемнювати приміщення за потреби. Освітлення має відповідати вимогам ДБН В.2.5-28-2018 «Природне і штучне освітлення» та включати як денне, так і штучне світло.

Окрім цього, в робочих зонах необхідно передбачити спеціальні місця для відпочинку та психологічного розвантаження працівників.

Робоче місце оператора ПК повинно мати не менше 6 м² площі та забезпечувати щонайменше 20 м³ повітряного об'єму.

3.3.2 Освітлення

Робоче місце користувача персонального комп'ютера повинно бути оснащене комбінованим освітленням, що поєднує природні та штучні джерела світла. Для загального освітлення доцільно використовувати люмінесцентні лампи типу ЛД. Освітленість поверхні, на якій виконується робота, має відповідати нормативним значенням – у межах 300-500 люкс.

3.3.3 Шум

Під час виконання робіт, що потребують підвищеної концентрації уваги, рівень шуму в приміщенні не повинен перевищувати 50 децибел. Для зменшення впливу шумових і вібраційних навантажень обладнання встановлюється на спеціальні амортизуючі основи. Якщо джерелом шуму є конструктивні елементи приміщення, наприклад стіни, доцільно використовувати звукоізоляційні матеріали.

3.3.4 Мікроклімат

Порушення мікрокліматичних умов може негативно впливати на працездатність працівників, знижувати їхню реакцію і сприяти виникненню помилок у роботі. Тому важливо підтримувати оптимальні параметри: температура повітря в межах 22-25 °С, відносна вологість – 40-60%, швидкість повітряного руху – 0,1-0,2 м/с.

Приміщення має бути обладнане системами опалення, вентиляції та кондиціювання, які забезпечують рівномірне теплопостачання, якісну циркуляцію повітря, а також ефективне очищення повітря від пилу й шкідливих домішок.

3.3.5 Електробезпека

Проходження електричного струму через людське тіло може призвести до термічних опіків, електролітичних порушень та серйозних біологічних ушкоджень. Для запобігання таким ураженням необхідно суворо дотримуватись ряду заходів електробезпеки, а саме:

- неухильно дотримуватись правил безпечної експлуатації електрообладнання;
- виключити можливість контакту працівників із відкритими або неізольованими струмопровідними частинами, що перебувають під напругою;
- застосовувати якісні ізоляційні матеріали для надійного електрозахисту;
- забезпечити ефективне заземлення всіх металевих елементів;

3.3.6 Вимоги до організації робочого місця

Робочі місця необхідно організовувати таким чином, щоб у полі зору працівника не з’являлися блискучі або відбивні поверхні, а також пряме світло від вікон чи освітлювальних приладів. Відеотермінали слід розміщувати під кутом 90–100° до вікон, щоб природне світло потрапляло збоку. Недопустимо, щоб працівник сидів обличчям або спиною до вікна, незалежно від типу освітлення.

Робочий стіл має мати можливість регулювання висоти в межах 680–800 мм та бути достатньо широким для зручного розміщення всіх необхідних предметів. Рекомендовані розміри: висота – 725 мм, ширина – від 600 до 1400 мм, глибина –

800–1000 мм. Стілець повинен бути оснащений механізмами налаштування висоти сидіння, кута нахилу та висоти спинки. Усі ці параметри мають регулюватися незалежно один від одного, легко налаштовуватись і надійно фіксуватись.

Монітор ПК доцільно встановлювати під кутом приблизно +30° відносно вертикальної лінії погляду користувача — для зручного візуального сприйняття інформації. Клавіатура має розташовуватись на робочій поверхні на відстані 100–300 мм від її переднього краю.

Усі елементи робочого місця повинні бути розташовані відповідно до ергономічних вимог, що сприятиме комфортній і продуктивній роботі користувача. На рис. 3.1 представлено робоче місце і робоча поза користувача комп'ютера.

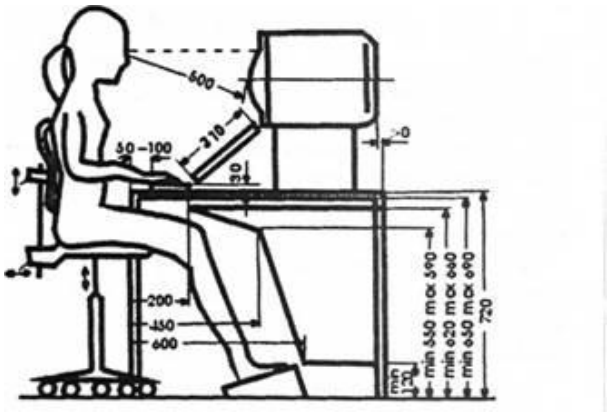


Рисунок 3.1. Робоче місце і робоча поза користувача комп'ютера

Робоче місце і робоча поза користувача комп'ютера включає:

1 – кут екрана; 2 – кут огляду (зору); 3 – відстань огляду; 4 – висота середини екрана; 5 – висота клавіатури; 6 – висота столу; 7 – відстань колін від столу; 8 – підставка для ніг; 9 – підставка для документів; 10 – положення рук; 11 – кут ліктів; 12 – спинка крісла; 13 – підлокітник; 14 – опора для попереку; 15 – кут колін; 16 – кут спинки крісла; 17 – висота сидіння

3.4 Пожежна безпека

Вимоги щодо пожежної безпеки в приміщеннях, де експлуатуються електричні мережі та електронно-обчислювальна техніка, регламентуються стандартами ГОСТ 12.1.033-81 та ГОСТ 12.1.004-85. Такі приміщення повинні відповідати пожежній категорії Д, що передбачає використання негорючих

матеріалів у конструкціях та відсутність легкозаймистих речовин у звичайних умовах експлуатації.

Система протипожежної безпеки охоплює низку заходів, спрямованих на попередження виникнення пожеж, включає засоби активного і пасивного захисту, а також організаційні й технічні рішення, що знижують рівень пожежної небезпеки.

До основних засобів захисту належать автоматизовані системи пожежної сигналізації, відповідні типи вогнегасників і забезпечення умов для швидкої та безпечної евакуації працівників.

Для ліквідації займання на початковому етапі використовують первинні засоби пожежогасіння, такі як порошкові та вуглекислотні вогнегасники, спеціальні протипожежні покривала з вогнетривких матеріалів, а також контейнери з водою або ящики з піском.

На рис. 3.2 представлено ілюстрацію типового пожежного щита з первинними засобами пожежогасіння.

Рисунок 3.2. Первинні засоби пожежогасіння

ВИСНОВКИ

У процесі виконання дипломної роботи було досліджено актуальність проблеми безпечної цифрової комунікації в умовах стрімкого зростання обсягів переданої інформації та зростаючих кіберзагроз. На основі вивчення сучасних тенденцій у сфері інформаційної безпеки було обґрунтовано доцільність розробки застосунку для обміну повідомленнями з вбудованою системою шифрування.

Під час реалізації проєкту було використано сучасні технології веб-розробки, зокрема .NET для серверної логіки, TypeScript та React для клієнтської частини, а також бібліотеки шифрування на основі Web Crypto API. Було враховано основні принципи побудови захищених систем: конфіденційність, цілісність та доступність. Особливу увагу приділено зручності користувача та простоті взаємодії із застосунком.

В даному проекті, усі поставлені цілі дипломної роботи було досягнуто. Розроблений застосунок демонструє приклад реалізації простої та водночас безпечної системи обміну повідомленнями, яка може бути використана як основа для створення масштабованого продукту.

Можливі напрями розширення функціоналу застосунку включають інтеграцію підтримки мультимедійних файлів, що дозволить користувачам обмінюватися зображеннями, відео та документами у зашифрованому вигляді. Це розширить сферу використання системи не лише для текстових повідомлень, а й для більш комплексної комунікації. Додатково доцільним є впровадження функції групових чатів та відеозв’язку, що зробить застосунок придатним для командної роботи, онлайн-конференцій або приватного спілкування з кількома співрозмовниками одночасно.

Результати дипломної роботи можуть бути використані як практична основа для навчання, комерційної реалізації або подальших наукових досліджень у сфері інформаційної безпеки та криптографії.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Стадник, В. В. Програмування мовою C#. – К. : Кондор, 2020. – 328 с.
2. Бондар, І. В. Об'єктно-орієнтоване програмування на C#. – Львів : Новий Світ – 2000, 2019. – 384 с.
3. Diffie, W., & Hellman, M. (1976). New directions in cryptography. IEEE Transactions on Information Theory, 22(6), 644–654.
4. Stallings W. Cryptography and Network Security: Principles and Practice. — Pearson Education, 2020. — 752 p.
5. Paar C., Pelzl J. Understanding Cryptography. — Springer, 2010. — 372 p.
6. Tanenbaum A., Wetherall D. Computer Networks. — Pearson, 2013. — 960 p.
7. Rescorla E. SSL and TLS: Designing and Building Secure Systems. — Addison-Wesley, 2001. — 320 p.
8. Leron P. Efficient Frontend Development with Angular and TypeScript. — Apress, 2022. — 320 p.
9. Krill M. Angular Development with TypeScript. — Manning Publications, 2021. — 400 p.
10. Дія. Захист персональних даних та безпека користувачів. — Офіційний портал Дія. — [Веб-сайт] URL: https://diia.gov.ua/services – Дата звернення: 12.05.2025
11. Закон України «Про захист інформації в інформаційно-телекомунікаційних системах» № 80/94-ВР від 05.07.1994 р. - [Веб-Сайт]. - URL: https://zakon.rada.gov.ua/laws/show/80/94-вр – Дата звернення: 10.05.2025.
12. ДСТУ 3396.2-96. Захист інформації. Терміни та визначення. - [Електронний ресурс]. - Режим доступу: https://online.budstandart.com/ua/catalog/doc-page?id_doc=5746 – Дата звернення: 12.05.2025.
13. Національний технічний університет України «КПІ ім. Ігоря Сікорського». Курс «Інформаційна безпека». - [Веб-сайт]. - URL: https://kafit.org.ua/courses/info-security – Дата звернення: 12.05.2025.
14. Криптографічні методи захисту інформації — навчальний посібник. Харків: ХНУРЕ, 2021. — 142 с

ДОДАТОК А. Фрагмент програмного коду шифрування

```
export async function generateKeyPair() {
    const keyPair = await window.crypto.subtle.generateKey(
        {
            name: "RSA-OAEP",
            modulusLength: 2048,
            publicExponent: new Uint8Array([1, 0, 1]),
            hash: "SHA-256",
        },
        true,
        ["encrypt", "decrypt"]
    );
    return keyPair;
}
export async function decryptSymmetricKey(encryptedBase64: string, privateKeyPem:
string): Promise<CryptoKey> {
    const privateKey = await importPrivateKey(privateKeyPem);
    const encrypted = Uint8Array.from(atob(encryptedBase64), c => c.charCodeAt(0));
    const decryptedRawKey = await crypto.subtle.decrypt(
        { name: "RSA-OAEP" },
        privateKey,
        encrypted
    );
    return await crypto.subtle.importKey(
        "raw",
        decryptedRawKey,
        { name: "AES-GCM" },
        true,
        ["encrypt", "decrypt"]
    );
}

export async function encryptMessage(message: string, key: CryptoKey): Promise<string> {
    const iv = crypto.getRandomValues(new Uint8Array(12));
    const encoded = new TextEncoder().encode(message);

    const ciphertext = await crypto.subtle.encrypt(
        { name: "AES-GCM", iv },
        key,
        encoded
    );

    const combined = new Uint8Array(iv.length + ciphertext.byteLength);
    combined.set(iv, 0);
    combined.set(new Uint8Array(ciphertext), iv.length);

    return bufferToBase64(combined.buffer);
}

export async function decryptMessage(encrypted: string, key: CryptoKey): Promise<string>
{
    const combined = new Uint8Array(base64ToBuffer(encrypted));
    const iv = combined.slice(0, 12);
    const data = combined.slice(12);

    const decrypted = await crypto.subtle.decrypt(
        { name: "AES-GCM", iv },
        key,
        data
    );

    return new TextDecoder().decode(decrypted);
```

```
}

export async function encryptKeyWithPublicKey(symKey: CryptoKey, publicKey: CryptoKey):
Promise<string> {
    const exportedKey = await crypto.subtle.exportKey("raw", symKey);

    const encrypted = await crypto.subtle.encrypt(
        { name: "RSA-OAEP" },
        publicKey,
        exportedKey
    );

    return btoa(String.fromCharCode(...new Uint8Array(encrypted)));
}

function bufferToBase64(buffer: ArrayBuffer): string {
    const binary = String.fromCharCode(...new Uint8Array(buffer));
    return btoa(binary);
}

function base64ToBuffer(base64: string): ArrayBuffer {
    const binary = atob(base64);
    const bytes = new Uint8Array(binary.length);
    for (let i = 0; i < binary.length; i++) {
        bytes[i] = binary.charCodeAt(i);
    }
    return bytes.buffer;
}
export function onChatCreated(callback: (chat: any) => void) {
    connection.on("ChatCreated", callback);
}

export function onNewMessage(callback: (msg: any) => void) {
    connection.off("NewMessage");
    connection.on("NewMessage", callback);
}
const BASE_URL = "https://localhost:44358";

export async function createDevice(deviceName: string, devicePublicKey: string) {
    const res = await fetch(`${BASE_URL}/api/chat/device`, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ deviceName, devicePublicKey })
    });
    if (!res.ok) throw new Error("Failed to create device");
    return res.json();
}

export async function getChats(deviceId: string) {
    const res = await fetch(`${BASE_URL}/api/chat/device/${deviceId}/chats`);
    return res.json();
}

export async function getDevicePublicKey(deviceId: string) {
    const res = await fetch(`${BASE_URL}/api/chat/device/${deviceId}/public-key`);
    return res.text();
}

export async function createChat(data: {
    deviceAId: string;
    deviceBId: string;
    encryptedSymmetricKeyForA: string;
    encryptedSymmetricKeyForB: string;
}) {
    const res = await fetch(`${BASE_URL}/api/chat/chat`, {
```

```
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(data)
    });
    if (!res.ok) throw new Error("Failed to create chat");
    return res.json();
}

export async function getChatById(chatId: string) {
    const res = await fetch(`${BASE_URL}/api/chat/chat/${chatId}`);
    return res.json();
}

export async function getMessages(chatId: string) {
    const res = await fetch(`${BASE_URL}/api/chat/chat/${chatId}/messages`);
    return res.json();
}

export async function sendMessage(data: {
    chatId: string;
    senderDeviceId: string;
    encryptedContent: string;
}) {
    const res = await fetch(`${BASE_URL}/api/chat/message`, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(data)
    });
    return res.json();
}

function App() {
    const [deviceId, setDeviceId] = useState<string | null>(null);
    const [chats, setChats] = useState<any[]>([]);
    const [peerId, setPeerId] = useState<string>('');
    const [connected, setConnected] = useState<boolean>(false);

    useEffect(() => {
        const setupDevice = async () => {
            const existingDeviceId = localStorage.getItem("deviceId");
            const privateKey = localStorage.getItem("privateKey");

            if (existingDeviceId && privateKey) {
                setDeviceId(existingDeviceId);
                return;
            }

            const keys = await generateKeyPair();
            const publicKeyPem = await exportPublicKey(keys.publicKey);
            const privateKeyPem = await exportPrivateKey(keys.privateKey);

            const device = await createDevice("Браузер", publicKeyPem);

            localStorage.setItem("deviceId", device.id);
            localStorage.setItem("publicKey", publicKeyPem);
            localStorage.setItem("privateKey", privateKeyPem);
            setDeviceId(device.id);
        };

        setupDevice();
    }, []);

    useEffect(() => {
        if (!deviceId) return;
```

```
        getChats(deviceId).then(setChats);

        const connection = startConnection(deviceId, (status) => {
            setConnected(status);
            console.log("📡 SignalR статус:", status ? "Підключено" : "Відключено");
        });

        onChatCreated((chat) => {
            console.log("🆕 Отримано новий чат:", chat);

            const newChat = {
                ...chat,
                deviceAId: chat.fromDeviceId,
                deviceBId: deviceId
            };

            setChats(prev => [newChat, ...prev]);
        });


        onNewMessage((msg) => {
            console.log("💬 Нове повідомлення:", msg);
        });
    }, [deviceId]);

    const handleCreateChat = async () => {
        if (!deviceId || !peerId) return;

        try {
            const peerPublicKeyPem = await getDevicePublicKey(peerId);
            const myPublicKeyPem = localStorage.getItem("publicKey")!;

            if (!peerPublicKeyPem || !myPublicKeyPem) {
                throw new Error("Не вдалося завантажити публічні ключі");
            }

            const peerPublicKey = await importPublicKey(peerPublicKeyPem);
            const myPublicKey = await importPublicKey(myPublicKeyPem);

            const aesKey = await window.crypto.subtle.generateKey(
                { name: "AES-GCM", length: 256 },
                true,
                ["encrypt", "decrypt"]
            );

            const encryptedKeyForA = await encryptKeyWithPublicKey(aesKey, myPublicKey);
            const encryptedKeyForB = await encryptKeyWithPublicKey(aesKey,
peerPublicKey);
            const chat = await createChat({
                deviceAId: deviceId,
                deviceBId: peerId,
                encryptedSymmetricKeyForA: encryptedKeyForA,
                encryptedSymmetricKeyForB: encryptedKeyForB
            });

            setChats(prev => [chat, ...prev]);
            setPeerId('');
        } catch (err) {
            console.error("❌ Помилка створення чату:", err);
            alert("Не вдалося створити чат. Перевірте публічні ключі.");
        }
    };

    return (
```

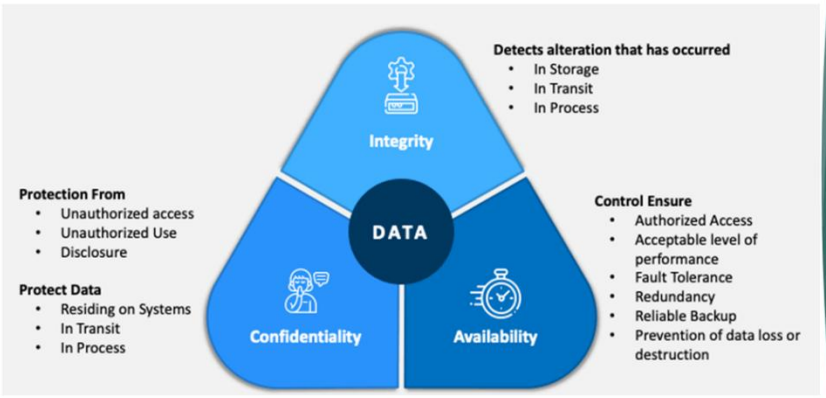
```
        <div className="container">
            <div className="status-indicator">
                {connected ? "🟢 Підключено" : " Відключено"}
            </div>

            <h1 className="title">Secure Messages</h1>

            <div className="device-info">
                {deviceId ? (
                    <p><b>📱 Device ID:</b> <code>{deviceId}</code></p>
                ) : (
                    <p>🔐 Ініціалізація пристрою...</p>
                )}
            </div>

            <div className="chat-create-box">
                <input
                    type="text"
                    placeholder="Device ID співрозмовника"
                    value={peerId}
                    onChange={e => setPeerId(e.target.value)}
                    className="input"
                />
                <button onClick={handleCreateChat} className="button">
                    ➕ Створити чат
                </button>
            </div>

            <h2>💬 Мої чати</h2>
            <ul className="chat-list">
                {deviceId && chats.map(chat => {
                    const otherId = chat.deviceAId === deviceId ? chat.deviceBId :
chat.deviceAId;
                    return (
                        <li
                            key={chat.id}
                            className="chat-item"
                            onClick={() => window.location.href = `/chat/${chat.id}`}
                            style={{cursor: 'pointer'}}
                        >
                            Чат з: <b>{otherId}</b>
                        </li>
                    );
                })}
            </ul>

        </div>
    );
}

export default App;
```

ДОДАТОК Б. Слайди мультимедійної презентації

ВИКОНАВ СТУДЕНТ:
ШУМІЛОВ М.С.

КЕРІВНИК:
НЕСТЕРЕНКО В. Д.

Розробка застосунку для обміну повідомленнями з можливістю шифрування



Актуальність

Що таке шифрування?

Шифрування — це процес перетворення інформації у спеціальний формат, зрозумілий лише тим, хто має відповідний ключ

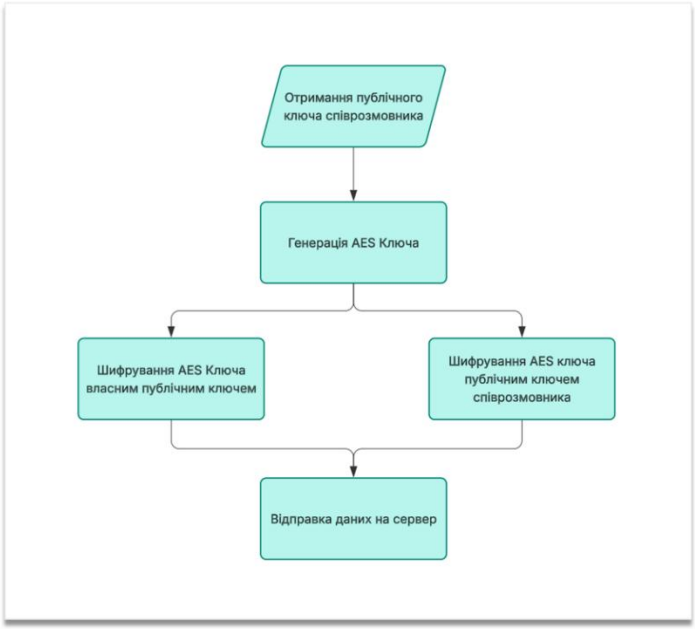
Види шифрування

Симетричне

- Використовується один ключ
- Швидке
- Потребує небагато ресурсів
- Менш захищене

Асиметричне

- Використовує два ключа
- Потребує більше ресурсів
- Надійніше

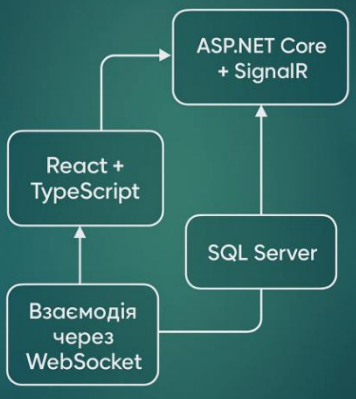


Комбіноване шифрування

Загальна архітектура застосунку

- Client: React + TypeScript
- Backend: ASP.NET Core + SignalR
- База даних: SQL Server
- Взаємодія через WebSocket
- Шифрування — на клієнтській стороні
- Реальний час через SignalR

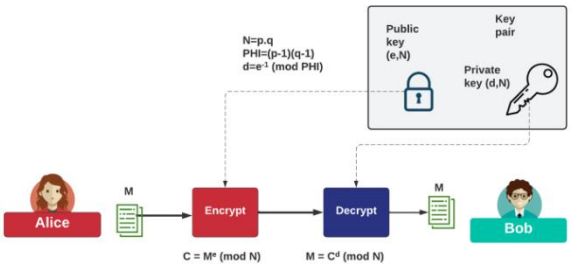
Загальна архітектура застосунку



Шифрування — на клієнтській стороні
Реальний час через SignalR

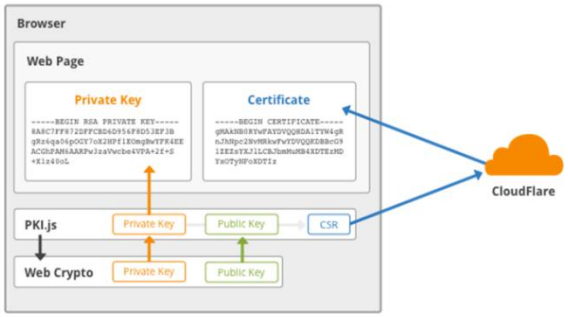
Використані алгоритми

У рамках реалізації безпечного обміну повідомленнями було використано потужну криптографічну комбінацію: алгоритм RSA-OAEP для асиметричного шифрування та SHA-256 для хешування, що є надійним і перевіреним стандартом у галузі.

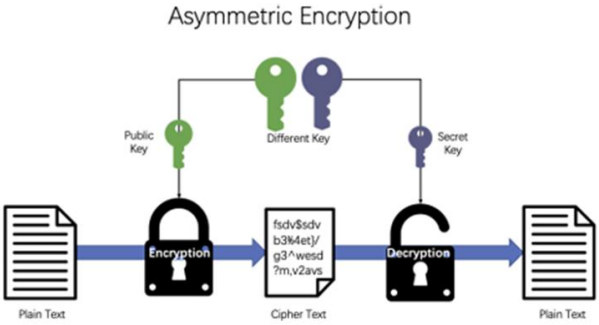


Створення ключів

Ключовим етапом шифрування є створення криптографічної пари ключів. У системі SecureMessages кожен пристрій генерує свою унікальну пару — відкритий (public) і закритий (private) ключ. Це відбувається через Web Crypto API, який надає надійні й безпечні методи генерації ключів згідно з сучасними криптографічними стандартами.



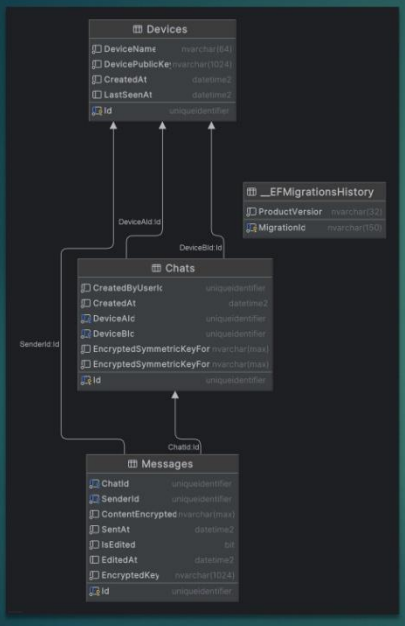
Шифрування повідомлення



Шифрування реалізовано з використанням RSA-OAEP — алгоритму, який є одним з найнадійніших способів захисту даних. При надсиланні повідомлення система спочатку перетворює його на формат байтів, після чого викликає функцію encryptMessage(), яка приймає відкритий ключ отримувача та саме повідомлення. Алгоритм RSA гарантує, що лише власник відповідного закритого ключа зможе прочитати вміст. Унікальність підходу полягає в тому, що навіть сервер не може дешифрувати передану інформацію — він бачить лише зашифровану форму без доступу до ключів.

Дешифрування

Зворотний процес шифрування — дешифрування — реалізовано через приватний ключ, який ніколи не покидає пристрій користувача. Використовуючи метод decryptMessage(), клієнт розшифровує повідомлення, що було зашифроване публічним ключем відправника.



База даних

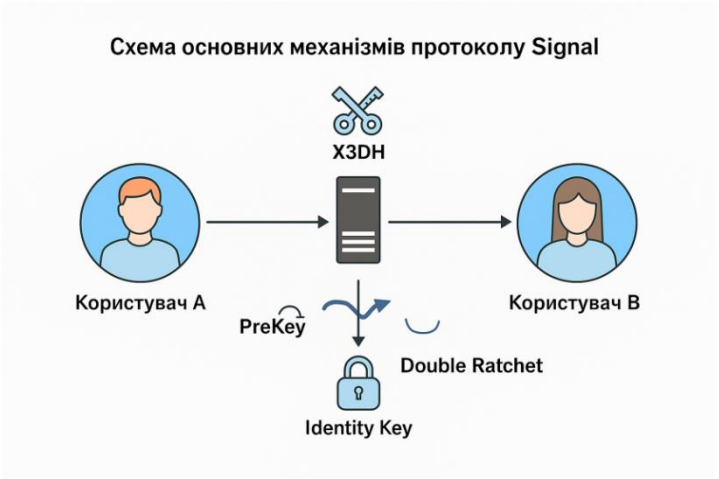
SQL Server. 2 основні таблиці:

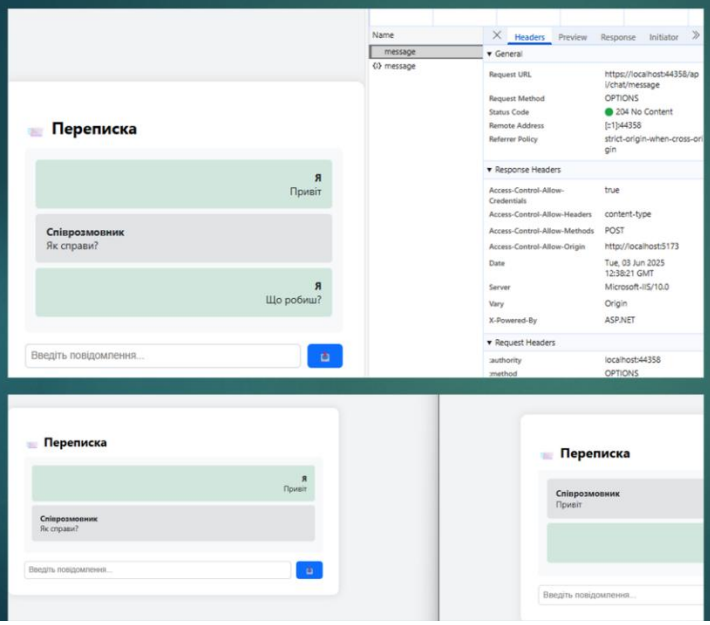
- **Users**: Username, PublicKey, PasswordHash
- **Messages**: SenderId, ReceiverId, EncryptedText, SentAt
  Дані зберігаються в зашифрованому вигляді
  Індексація — для швидкого пошуку

Передача в реальному часі

SignalR — двосторонній канал через WebSocket

- Користувач приєднується до хабу
- Повідомлення миттєво передається співрозмовнику
- Висока швидкість без навантаження на сервер





Тестування функцій

Ключові аспекти безпеки

- Шифрування на клієнті
- HTTPS + WebSocket
- Захист токенів авторизації (JWT)
- Обмеження доступу до БД
- Хешування паролів (bcrypt)

Висновки

У ході дипломної роботи було реалізовано повноцінний прототип системи захищеного обміну повідомленнями, який поєднує в собі простий інтерфейс і складну внутрішню логіку. Особливу увагу приділено питанню безпеки — від генерації ключів до шифрування повідомлень і їх передачі в реальному часі. Система працює стабільно, підтримує багатокористувацьку взаємодію, шифрує повідомлення ще до моменту надсилання та не дозволяє жодній третій стороні отримати доступ до вмісту. Проєкт має практичну цінність та потенціал до подальшого розвитку як у напрямку мобільних додатків, так і B2B-рішень.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Шуміло Максима Сергійовича

(прізвище, ім`я та по батькові)

Спеціальність *123 «Комп'ютерна інженерія»*

Освітня програма *«Безпека комп'ютерних систем та мереж»*

Керівник дипломного проекту (роботи) *Нестеренко Володимир Дмитрович*

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка застосунку для обміну повідомленнями з можливістю шифрування*

Обсяг розрахунково-пояснювальної записки *80* сторінок

Обсяг графічної (презентаційної) частини *15* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту завданню

Представлений на рецензію дипломний проект відповідає затвердженій темі та виконаний відповідно технічному завданню. Дипломний проект присвячений проблемі захищеності обміну повідомлень та складається з пояснювальної записки, додатку з програмним кодом та мультимедійної презентації, що містить приклади роботи програми.

б) характеристика виконання кожного розділу дипломного проекту

Пояснювальна записка складається з основного розділу (аналізу предметної області, проектування застосунку, реалізації застосунку, тестування застосунку), економічного розділу, розділу охорони праці та додатків. Перелічені розділи поетапно охоплюють розробку, виконані докладно та обґрунтовано. Розділ охорони праці містить загальну інформацію та вимоги до техніки безпеки оператора КТ. Економічний розділ проекту містить розрахунок витрат на НДР та реалізацію проекту.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту

Графічна частина складається з 15 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять ілюстративні схеми, скріншоти роботи програмного застосунку, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм. Якість виконання графічної частини проекту та пояснювальної записки добра, розробку виконано у повному обсязі.

г) перелік позитивних якостей дипломного проекту *Реалізовано end-to-end шифрування о основі алгоритмів AES та RSA, що забезпечує високий рівень конфіденційності. Проект охоплює актуальну тему інформаційної безпеки, зокрема захищеного обміну повідомленнями.*

Система підтримує реальний час обміну повідомлень за допомогою SignalR.

д) основні недоліки дипломного проекту

Відсутній механізм резервного копіювання або синхронізації ключів між кількома пристроями одного користувача. RSA-2048 – доволі громіздкий, у клієнтському JS-середовищі може гальмувати. Немає жодного Pentest-звіту чи автоматизованих сканів (SAST/DAST).

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Добре*

Прізвище, ім'я, по батькові рецензента *к.т.н. Шибаєва Наталя Олегівна*

Місце роботи і посада рецензента *Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій*

Підпис:

«*22*» *червня* 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Шуміло Максима Сергійовича

(прізвище, ім`я та по батькові)

Спеціальність: *123 «Комп'ютерна інженерія»*

Освітня програма: *«Безпека комп'ютерних систем і мереж»*

Тема дипломного проекту: *Розробка застосунку для обміну повідомленнями з можливістю шифрування*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка до дипломного проекту містить 80 сторінок. У пояснювальній записці описано етапи розробки Back-End та Front-End частини веб-застосунку для обміну повідомленнями засобами ASP.NET та React. Графічна частина складається з окремих слайдів, оформлених у вигляді презентації, передбачених технічним завданням. Якість виконання пояснювальної записки та слайдів добра.*

б) самостійність роботи над проектом: *Протягом виконання дипломного проекту здобувач освіти Шуміло Максим поступово та послідовно виконував всі етапи, проявляв ініціативу в створенні загальної концепції та реалізації роботи. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника.*

в) теоретична підготовка випускника (випускниці): *Здобувач освіти Шуміло Максим під час роботи над дипломним проектом вивчив достатньо багато літературних та інтернет-джерел за даною тематикою. Вважаю, що теоретична підготовка дипломника достатня і він готовий до захисту проекту.*

г) вміння розв'язувати виробничі та конструкторські питання *Під час виконання дипломного проекту здобувач освіти Шуміло Максим показав вміння організовано працювати над поставленим завданням, застосовувати знання у галузі програмування та криптографії, розробляти, та налаштовувати спеціалізоване програмне забезпечення, оформлювати слайди та складати презентації, користуючись сучасними комп'ютерними програмними засобами, такими як MS VS Code, MS VS, SSMS, Microsoft PowerPoint, Microsoft Visio та ін.*

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Добре*

Загальна оцінка *Добре*

Прізвище, ім'я, по батькові керівника дипломного проекту *Нестеренко Володимир Дмитрович*

Місце роботи і посада керівника дипломного проекту *ВСП «Одеський технічний фаховий коледж ОНТУ», викладач спецдисциплін циклової комісії комп'ютерної техніки та програмної інженерії*

Підпис

«16» червня 2025 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Шуміло Максим Сергійович,
здобувач освіти гр. 4КБ-02, та

Нестеренко Володимир Дмитрович ,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проекту фахового молодшого бакалавра на тему:

«Розробка застосунку для обміну повідомленнями з можливістю шифрування» (автор роботи – Шуміло М.С., керівник роботи – Нестеренко В.Д.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Шуміло М.С. /

Керівник _____ / Нестеренко В.Д. /

«16» червня 2025 р.

Д О В І Д К А

циклової комісії КТ та ПІ
про допуск до захисту дипломного проєкту
здобувача (здобувачки) освіти IV курсу
відділення комп'ютерних систем групи 4КБ-02

Шуміла Максима Сергійовича

на тему *Розробка застосунку для обміну повідомленнями з можливістю шифрування*

Висновок відповідальної особи за проведення нормоконтролю:
пояснювальна записка до дипломного проєкту виконана з некритичними порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування

(підпис) *16.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 13.06.2025 р. значення коефіцієнту подібності в роботі становить 14,51%, коефіцієнт цитування – 1,56%.*

(підпис) *16.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Шуміла М.С.* (П.І.Б.)

проведена «*16*» *червня* 20*25* р.

Висновки *Пояснювальна записка до дипломного проєкту виконана у повному обсязі. Випускна кваліфікаційна робота (дипломний проєкт) відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Голова ЦК КТ та ПІ (підпис) *Кривченко Ю.В.* (П.І.Б.)

StrikePlagiarism.com

Дата звіту 6/13/2025
Дата редагування 6/13/2025

Документ прийнятий

Звіт подібності

метадані

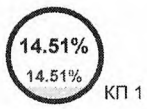
Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Розробка застосунку для обміну повідомленнями з можливістю шифрування

Автор
Шуміло Максим Сергійович
Науковий керівник / Експерт
Нестеренко Володимир Дмитрович

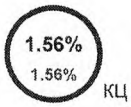
підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КЦ

25	14855	124911
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	B	28
Інтервали	A⟶	0
Мікропробіли	¶	0
Білі знаки	B	5
Парафрази (SmartMarks)	a	101

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	60 0.40 %
2	https://card-file.ontu.edu.ua/server/api/core/bitstreams/995bdcec-4e4d-4321-8070-4d6badcb8e49/content	44 0.30 %
3	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	42 0.28 %
4	https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download	38 0.26 %
5	https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download	37 0.25 %

6	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	35 0.24 %
7	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	34 0.23 %
8	https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download	32 0.22 %
9	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	30 0.20 %
10	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	29 0.20 %

з домашньої бази даних (0.32 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності [redacted] 6/12/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	47 (5) 0.32 %

з програми обміну базами даних (1.97 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Zakhliebaiev_mag_rob.docx [redacted] 12/10/2023 Sumy State University (Кафедра комп'ютерних наук)	164 (16) 1.10 %
2	MP_K23-2м_Лісова І.О. [redacted] 1/10/2025 University of Customs and Finance (University of Customs and Finance)	40 (3) 0.27 %
3	Новий Документ Microsoft Word (2)_compressed.pdf [redacted] 12/15/2024 East Ukrainian National University named after Volodymyr Dahl (East Ukrainian National University named after Volodymyr Dahl)	28 (4) 0.19 %
4	Розробка плагіну для захисту інтелектуальної власності в 3D моделюванні [redacted] 6/17/2024 Odessa National Polytechnic University (ІІБРТ, Каф. кібербезпеки та програмного забезпечення)	25 (2) 0.17 %
5	bitstream_12de591d-f150-4815-8037-f8e10a46fe58 [redacted] 12/8/2024 National Technical University "Kharkiv Polytechnic Institute" students papers (National Technical University "Kharkiv Polytechnic Institute" students papers)	23 (2) 0.15 %
6	Розробка системи управління контентом для веб-сайту дитячого будинку з реалізацією панелі адміністратора та використанням принципів “чистого коду” Роберта Мартіна [redacted] 5/21/2025 Volodymyr Vynnychenko Central Ukrainian State Pedagogical University (кафедра інформатики, програмування, штучного інтелекту та технологічної освіти)	12 (2) 0.08 %

з Інтернету (12.22 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/server/api/core/bitstreams/44c16132-5f53-48e2-b6c0-61e9a2f0fd75/content	527 (44) 3.55 %
2	**https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download**	**354 (24) 2.38 %**
3	**https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download**	**139 (7) 0.94 %**

4	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	115 (4) 0.77 %
5	https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download	76 (5) 0.51 %
6	https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download	70 (2) 0.47 %
7	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	65 (7) 0.44 %
8	https://card-file.ontu.edu.ua/server/api/core/bitstreams/995bdcec-4e4d-4321-8070-4d6badcb8e49/content	56 (2) 0.38 %
9	https://gist.github.com/homeronkis	52 (2) 0.35 %
10	https://card-file.ontu.edu.ua/bitstreams/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download	45 (4) 0.30 %
11	https://card-file.ontu.edu.ua/bitstreams/21173711-5b67-4b87-b17f-6302c25e7a31/download	36 (2) 0.24 %
12	https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download	33 (3) 0.22 %
13	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	30 (3) 0.20 %
14	https://card-file.ontu.edu.ua/server/api/core/bitstreams/214d43de-5031-4ab6-849f-efa001b5416b/content	25 (2) 0.17 %
15	https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download	23 (1) 0.15 %
16	https://card-file.ontu.edu.ua/bitstreams/62baa43e-b968-4993-bb54-8cf8761a89b2/download	22 (2) 0.15 %
17	https://card-file.ontu.edu.ua/bitstreams/5240e379-7721-49f0-8ee8-27140b0b473a/download	21 (1) 0.14 %
18	https://elartu.tntu.edu.ua/bitstream/lib/41648/2/Dyplom_Kostiuk_K_O_2023.pdf	20 (2) 0.13 %
19	https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download	17 (1) 0.11 %
20	https://card-file.ontu.edu.ua/bitstreams/0e6c3361-ffbf-4469-86a1-fe84a1fe21cd/download	13 (1) 0.09 %
21	https://www.ijfmr.com/papers/2024/3/19395.pdf	13 (1) 0.09 %
22	https://dspace.znu.edu.ua/jspui/bitstream/12345/20020/1/%D0%94%D0%B8%D0%BF%D0%BB%D0%BE%D0%BC_%D0%AE%D1%80%D1%8C%D1%94%D0%B2.pdf	13 (1) 0.09 %
23	https://www.kursak.com/bezpechni-pryyomy-pratsi-bukhhalterii/	13 (2) 0.09 %
24	https://journals.dut.edu.ua/index.php/dataprotect/article/download/2302/2275/	12 (1) 0.08 %
25	https://essuir.sumdu.edu.ua/bitstream-download/123456789/92741/1/Bac_Pryhodina.pdf	9 (1) 0.06 %
26	https://openarchive.nure.ua/bitstreams/4c9330ad-cdce-4f79-a164-3b193ad05dc3/download	6 (1) 0.04 %
27	https://card-file.ontu.edu.ua/bitstreams/0e72a3b9-bdd7-4711-a3c6-dedc1d4287cc/download	6 (1) 0.04 %
28	https://ela.kpi.ua/bitstream/123456789/42797/1/Davydenko_bakalavr.pdf	5 (1) 0.03 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»
Освітньо-професійна програма «Безпека комп'ютерних систем і мереж» Група: 4КБ- 02