

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Веб-технології»)

на тему:

«Веборієнтований програмний комплекс стеганографічного захисту
інформації у цифрових зображеннях»

Виконав студент групи ВТ-22-1
Пилипчук Святослав Анатолійович

Керівник роботи:
Концидайло Андрій Михайлович

Рецензент:
Власенко Олег Васильович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **Пилипчук Святослав Анатолійович**

Керівник роботи: **Концидайло Андрій Михайлович**

Тема роботи: **«Веборієнтований програмний комплекс стеганографічного захисту інформації у цифрових зображеннях»,**

затверджена

Наказом закладу вищої освіти від **«27» лютого 2026 р., №92**

Вихідні дані для роботи: об'єкт дослідження — процес прихованої передачі текстової інформації цифровими каналами зв'язку; предмет дослідження — методи цифрової стеганографії просторової області, алгоритми χ^2 -стегааналізу та криптографічного захисту, а також їх реалізація у вигляді веб-застосу-нку.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розді-лів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Концидайло А. М.	10.04.2026р.	20.04.2026р
2	Концидайло А. М.	20.04.2026р	25.04.2026р
3	Концидайло А. М.	01.05.2026р	20.05.2026р

РЕФЕРАТ

Кваліфікаційна робота складається з програмного комплексу StegoVault та пояснювальної записки. Пояснювальна записка містить 108 сторінок, 40 рисунків та 14 таблиць.

Метою роботи є проектування та реалізація веб-орієнтованого програмного комплексу для стеганографічного захисту текстових повідомлень у растрових зображеннях із вбудованим модулем χ^2 -стегоаналізу та криптографічним захистом AES-256-CBC.

У роботі проведено аналіз актуальних загроз інформаційній безпеці, досліджено методи цифрової стеганографії просторової області та виконано порівняльний аналіз існуючих програмних аналогів. Розроблено клієнт-серверну архітектуру на основі REST API та SPA з використанням стеку TypeScript / Node.js + Express / React / MongoDB. Реалізовано три методи стеганографії — LSB, LSBM та Adaptive (Edge-Aware з оператором Собеля), механізм Data Whitening на основі DEFLATE та XOR-скремблювання, а також модуль χ^2 -стегоаналізу з ковзним вікном. Система підтримує мультикористувацький режим із JWT-автентифікацією, пакетну обробку зображень, самознищувальні повідомлення та адміністративний інтерфейс. Функціональне тестування за методом чорної скриньки охопило 46 сценаріїв із 100% успішністю проходження.

Ключові слова: СТЕГANOГРАФІЯ, LSB, LSBM, АДАПТИВНИЙ МЕТОД, AES-256, СТЕГОАНАЛІЗ, CHI-SQUARE, DATA WHITENING, WEB-ЗАСТОСУНОК, ЗАХИСТ ІНФОРМАЦІЇ.

					ІПЗ.КР.Б-121-26-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Пилипчук С.А.			Веборієнтований програмний комплекс стеганографічного захисту інформації у цифрових зображеннях	Літ.	Арк.	Аркушів
Керівник		Концидайло А.М.					3	108
Реценз.		Власенко О.В.				Житомирська політехніка, група ВТ-22-1		
Н. контр.								
Затверд.								

ABSTRACT

The bachelor's thesis consists of the StegoVault software system and an explanatory note. The explanatory note contains 108 pages, 40 illustrations and 14 tables.

The goal of the work is to design and implement a web-based software system for steganographic protection of text messages in raster images, incorporating a built-in χ^2 -steganalysis module and AES-256-CBC cryptographic protection.

The work presents an analysis of current information security threats, a study of spatial-domain digital steganography methods, and a comparative analysis of existing software analogues. A client-server architecture was designed using a REST API and SPA approach with a TypeScript / Node.js + Express / React / MongoDB stack. Three steganography methods were implemented — LSB, LSBM, and Adaptive (Edge-Aware using Sobel operator) — along with a Data Whitening mechanism based on DEFLATE compression and XOR-scrambling, and a χ^2 -steganalysis module with a sliding window. The system supports multi-user mode with JWT authentication, batch image processing, self-destructing messages, and an administrative interface. Functional black-box testing covered 46 test scenarios with a 100% pass rate.

Keywords: STEGANOGRAPHY, LSB, LSBM, ADAPTIVE METHOD, AES-256, STEGANALYSIS, CHI-SQUARE, DATA WHITENING, WEB APPLICATION, INFORMATION SECURITY.

		Пилипчук С.А.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AES (Advanced Encryption Standard) — стандарт симетричного блокового шифрування

API (Application Programming Interface) — інтерфейс програмування застосунків

ASCII (American Standard Code for Information Interchange) — стандартна таблиця кодування символів

BMP (Bitmap) — формат растрового зображення без стиснення

JSON (Binary JSON) — двійковий формат зберігання документів у MongoDB

CBC (Cipher Block Chaining) — режим зчеплення блоків шифрування

CERT-UA — Команда реагування на комп'ютерні надзвичайні події України

CORS (Cross-Origin Resource Sharing) — механізм спільного доступу до ресурсів між різними джерелами

CRUD (Create, Read, Update, Delete) — базові операції з даними

DCT (Discrete Cosine Transform) — дискретне косинусне перетворення

DEFLATE — алгоритм стиснення даних без втрат

DFT (Discrete Fourier Transform) — дискретне перетворення Фур'є

DLP (Data Loss Prevention) — система запобігання витоку даних

DOM (Document Object Model) — об'єктна модель документа

DWT (Discrete Wavelet Transform) — дискретне вейвлет-перетворення

HMAC (Hash-based Message Authentication Code) — код автентифікації повідомлень на основі хеш-функції

HTTP/HTTPS (HyperText Transfer Protocol / Secure) — протокол передачі гіпертексту

IV (Initialization Vector) — вектор ініціалізації

JSON (JavaScript Object Notation) — текстовий формат обміну даними

JWT (JSON Web Token) — компактний токен автентифікації з цифровим підписом

LSB (Least Significant Bit) — молодший значущий біт

LSBM (Least Significant Bit Matching) — метод підбору молодшого значущого біта

MERN (MongoDB, Express, React, Node.js) — стек веб-технологій

MSE (Mean Squared Error) — середньоквадратична похибка

NoSQL (Not only SQL) — клас нереляційних систем управління базами даних

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		5

ODM (Object Document Mapper) — бібліотека відображення об'єктів на документи бази даних

PBKDF2 (Password-Based Key Derivation Function 2) — функція формування ключа з пароля

PNG (Portable Network Graphics) — формат растрового зображення зі стисненням без втрат

PoV (Pairs of Values) — пари значень (у χ^2 -стегааналізі)

PRNG (Pseudo-Random Number Generator) — генератор псевдовипадкових чисел

PSNR (Peak Signal-to-Noise Ratio) — пікове відношення сигналу до шуму

PVD (Pixel Value Differencing) — метод стеганографії на основі різниці значень пікселів

RBAC (Role-Based Access Control) — рольова модель управління доступом

REST (Representational State Transfer) — архітектурний стиль побудови API

RGB (Red, Green, Blue) — трикомпонентна колірна модель

RGBA (Red, Green, Blue, Alpha) — колірна модель RGB із каналом прозорості

SPA (Single Page Application) — односторінковий застосунок

TTL (Time To Live) — час життя запису в базі даних

UI (User Interface) — інтерфейс користувача

UML (Unified Modeling Language) — уніфікована мова моделювання

UTF-8 (Unicode Transformation Format, 8-bit) — формат кодування символів Unicode

UX (User Experience) — досвід взаємодії користувача

XOR (Exclusive OR) — побітова операція «виключне АБО»

БД — база даних

ІКТ — інформаційно-комунікаційні технології

КБ — кілобайт

МБ — мегабайт

НФВ — нефункціональна вимога

ПЗ — програмне забезпечення

СУБД — система управління базами даних

ФВ — функціональна вимога

χ^2 — статистика хі-квадрат (критерій Пірсона)

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ЗМІСТ

РЕФЕРАТ.....	4
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	5
ВСТУП.....	8
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ЗАСОБІВ	10
1.1 Постановка задачі та аналіз проблеми захисту даних	10
1.2 Огляд методів цифрової стеганографії та стеганоаналізу	14
1.3 Аналіз існуючих програмних аналогів	19
1.4 Вибір архітектури та інструментальних засобів	25
Висновки до першого розділу	29
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ	30
2.1 Архітектура системи та варіанти використання	30
2.2 Проєктування бази даних	35
2.3 Розробка алгоритмів стеганографії та криптозахисту	38
2.4 Реалізація програмного комплексу	52
Висновки до другого розділу	57
РОЗДІЛ 3. ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ	58
3.1 Структура інтерфейсу та функціональні можливості	58
3.2 Експериментальне дослідження стійкості до атак	67
3.3 Тестування функціоналу та інструкція користувача	75
Висновок до третього розділу	79
ВИСНОВКИ	81
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	83
ДОДАТКИ.....	86

ВСТУП

Актуальність теми. Стрімкий розвиток інформаційно-комунікаційних технологій зумовив кардинальне збільшення обсягу конфіденційних даних, що передаються цифровими каналами зв'язку. За даними IBM «Cost of a Data Breach 2024», середня вартість одного витоку даних досягла 4,88 млн доларів США — найбільше щорічне зростання з часів пандемії. В Україні CERT-UA у 2025 році опрацювала 5 927 кіберінцидентів, що на 37,4% більше, ніж у 2024 році. Наведені дані свідчать про нагальну потребу в ефективних засобах захисту інформації при її передачі цифровими каналами.

Традиційна криптографія не вирішує проблему аналізу трафіку: зашифроване повідомлення є очевидним індикатором передачі цінної інформації. Стеганографія усуває цей недолік, приховуючи сам факт існування повідомлення. Комбінація обох підходів формує двошаровий захист, стійкіший порівняно з кожним методом окремо. Проте існуючі інструменти (OpenStego, Steghide, SilentEye, OpenPuff) мають спільні системні обмеження: відсутність веб-доступності, підтримка лише одного методу вбудовування та відсутність повноцінного модуля статистичного стегоаналізу. Це підтверджує доцільність розробки власної веб-платформи, що усуває виявлені прогалини.

Метою роботи є проектування та реалізація програмного комплексу для стеганографічного захисту текстових повідомлень у растрових зображеннях із вбудованим модулем χ^2 -стегоаналізу та криптографічним захистом AES-256-CBC.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

1. Проаналізувати актуальні загрози інформаційній безпеці та сформулювати вимоги до системи.
2. Дослідити методи стеганографії просторової області (LSB, LSBM, Adaptive) та χ^2 -стегоаналізу.
3. Провести порівняльний аналіз існуючих програмних аналогів та обґрунтувати власну розробку.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

4. Спроекувати клієнт-серверну архітектуру на основі REST API і SPA.
5. Реалізувати алгоритмічне ядро: методи стеганографії, Data Whitening та χ^2 -аналіз із ковзним вікном.
6. Розробити мультикористувацьку веб-платформу з JWT-автентифікацією, пакетною обробкою та адміністративним інтерфейсом.
7. Провести експериментальне дослідження якості зображень (PSNR, MSE) та стійкості методів до χ^2 -атаки.
8. Виконати функціональне тестування за сценаріями чорної скриньки.

Об'єкт дослідження — процес прихованої передачі текстової інформації цифровими каналами зв'язку із забезпеченням конфіденційності та невиявлюваності.

Предмет дослідження — методи цифрової стеганографії просторової області, алгоритми χ^2 -стегоаналізу та засоби криптографічного захисту в контексті їх реалізації у вигляді веб-застосунку.

Методи дослідження. У роботі використано: аналіз і синтез — для дослідження предметної області; порівняльний аналіз — для оцінки аналогів; методи цифрової обробки зображень — для реалізації стеганографічних алгоритмів; математична статистика — для χ^2 -тесту; UML-моделювання — для проектування архітектури; метод чорної скриньки — для функціонального тестування.

Практична цінність результатів. Розроблений комплекс StegoVault реалізує три методи стеганографії, AES-256-CBC шифрування, Data Whitening та χ^2 -стегоаналіз у єдиній веб-платформі з мультикористувацьким режимом і пакетною обробкою до 10 зображень. Результати можуть бути застосовані у задачах корпоративної безпеки, цифрової криміналістики та захищеної комунікації в умовах цензури або збройного конфлікту.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ЗАСОБІВ

1.1 Постановка задачі та аналіз проблеми захисту даних

Обсяг конфіденційної інформації, що передається відкритими цифровими каналами, невпинно зростає, а проблема її захисту від несанкціонованого доступу лише загострюється. За даними звіту IBM «Cost of a Data Breach 2024», середня вартість одного витоку даних у 2024 році досягла 4,88 млн доларів США, що на 10% перевищує показник попереднього року та є найбільшим щорічним зростанням із часів пандемії [1]. Згідно зі звітом Verizon «Data Breach Investigations Report 2024», було зафіксовано 10 626 підтверджених випадків витоку даних — найбільше значення за всю 17-річну історію публікації звіту [2]. В Україні ситуація є ще більш критичною: CERT-UA у 2024 році опрацювала 4 315 кіберінцидентів, а у 2025 році — вже 5 927, що становить зростання на 37,4%. Зокрема, у другому півріччі 2024 року зафіксовано зростання кількості інцидентів на 48% порівняно з попереднім аналогічним періодом [3][4]. Наведені дані свідчать про нагальну потребу в ефективних засобах захисту інформації при її передачі цифровими каналами.

Сукупність загроз інформаційній безпеці в контексті передачі даних можна класифікувати за трьома основними категоріями: перехоплення, модифікація та аналіз трафіку. Перехоплення передбачає несанкціоноване отримання даних зловмисником у процесі їх передачі по каналу зв'язку. Модифікація — цілеспрямовану зміну переданого повідомлення з метою дезінформації або порушення цілісності. Аналіз трафіку (traffic analysis) є найбільш прихованою загрозою: навіть без розшифрування вмісту зловмисник може встановити факт комунікації між сторонами, визначити її частоту та обсяг, і на підставі цього зробити висновки про характер діяльності учасників. Загальна схема стеганографічної комунікації з урахуванням наведених загроз представлена (рис. 1.1).

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

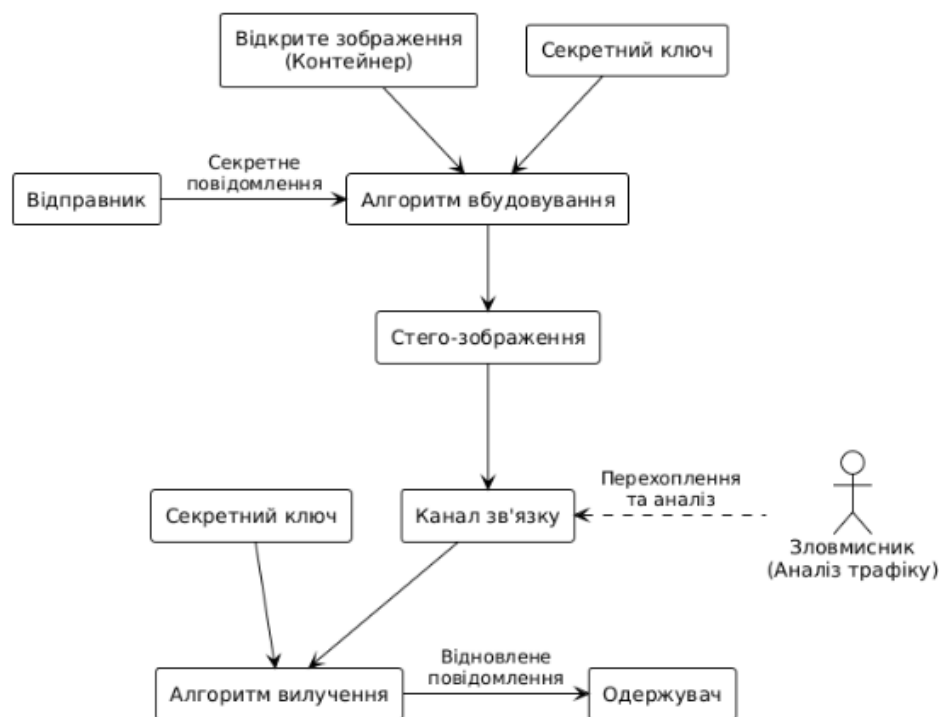


Рис. 1.1 – Загальна схема стеганографічної комунікації

Традиційним засобом захисту від перехоплення та модифікації є криптографія — наука про методи перетворення інформації з метою забезпечення її конфіденційності та цілісності. Проте криптографія не вирішує проблему аналізу трафіку: зашифроване повідомлення, навіть будучи нечитабельним для зловмисника, є очевидним індикатором того, що передається щось цінне. У середовищах з посиленням моніторингом (авторитарні режими, корпоративні мережі під наглядом, зони активних бойових дій) сам факт використання шифрування може привернути підвищену увагу або навіть слугувати підставою для примусової видачі ключів дешифрування. Порівняльну характеристику криптографії та стеганографії за ключовими параметрами ілюструє (рис. 1.2).



Рис. 1.2 – Порівняння криптографії та стеганографії

Стеганографія (від грецьк. *steganos* — прихований, *graphia* — написання) — це наука про приховування самого факту передачі інформації шляхом вбудовування секретного повідомлення у нейтральний контейнер (зображення, аудіо, відео, текстовий файл). На відміну від криптографії, яка приховує зміст повідомлення, стеганографія приховує факт його існування. Для зловмисника або системи моніторингу передача виглядає як звичайний обмін мультимедійним файлом.

Комбінація цих двох підходів формує двошаровий захист, який є більш стійким порівняно з кожним підходом окремо. Саме такий підхід реалізовано в розробленій системі: повідомлення спочатку шифрується алгоритмом AES-256, а вже зашифровані дані вбудовуються у зображення методами LSB-стеганографії.

Практичне застосування стеганографічного захисту охоплює широкий спектр сценаріїв. У задачах захищеної комунікації в умовах цензури журналісти та правозахисники в країнах з обмеженим доступом до інформації використовують стеганографію для передачі матеріалів без ризику виявлення. У сфері корпоративної безпеки системи Data Loss Prevention (DLP) застосовують стеганографічні водяні знаки для відстеження джерел витоку конфіденційних документів. Цифрові водяні знаки захищають авторські права на фотографії та медіаконтент: у зображення вбудовуються невидимі ідентифікатори власника. Нарешті, в умовах збройного конфлікту стеганографія дозволяє передавати координати й оперативні дані через відкриті канали зв'язку без привернення уваги.

Таким чином, розробка програмного комплексу, що реалізує стеганографічний захист з вбудованим модулем стегоаналізу та криптографічним шифруванням, є актуальним науково-практичним завданням.

Аналіз загроз та сценаріїв застосування дозволив сформулювати функціональні та нефункціональні вимоги до системи, які наведені (табл. 1.1) та (табл. 1.2) відповідно.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Концидайло А.М.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

Функціональні вимоги до системи

ID	Вимога	Пріоритет
ФВ-01	Система повинна забезпечувати кодування текстового повідомлення у растрове зображення форматів PNG та BMP	Високий
ФВ-02	Система повинна забезпечувати декодування прихованого повідомлення із стего-зображення	Високий
ФВ-03	Система повинна підтримувати щонайменше три методи стеганографії: LSB, LSBM, Adaptive	Високий
ФВ-04	Система повинна виконувати обов'язкове шифрування повідомлення алгоритмом AES-256-CBC перед вбудовуванням	Високий
ФВ-05	Система повинна здійснювати статистичний аналіз зображень на наявність прихованих даних (χ^2 -тест)	Високий
ФВ-06	Система повинна обчислювати та відображати метрики якості зображення: PSNR та MSE	Середній
ФВ-07	Система повинна підтримувати реєстрацію, автентифікацію та авторизацію користувачів	Високий
ФВ-08	Система повинна зберігати історію операцій кодування та декодування	Середній
ФВ-09	Система повинна підтримувати пакетну обробку до 10 зображень одночасно	Середній
ФВ-10	Система повинна забезпечувати функцію самознищувальних повідомлень з обмеженням кількості переглядів	Низький
ФВ-11	Система повинна надавати адміністративний інтерфейс управління користувачами та системними журналами	Середній

Таблиця 1.2

Нефункціональні вимоги до системи

ID	Категорія	Вимога
НФВ-01	Продуктивність	Час кодування зображення розміром до 1920×1080 пікселів не повинен перевищувати 5 секунд
НФВ-02	Продуктивність	Час відповіді API на запити стегоаналізу — не більше 10 секунд
НФВ-03	Безпека	Автентифікація реалізується на основі JWT (access token 15 хв, refresh token 7 діб)
НФВ-04	Безпека	Обмеження частоти запитів (rate limiting): не більше 200 запитів за 15 хвилин з однієї IP-адреси
НФВ-05	Безпека	Паролі користувачів зберігаються у хешованому вигляді (bcrypt, cost factor 12)
НФВ-06	Надійність	Система повинна коректно обробляти некоректні вхідні дані та повертати зрозумілі повідомлення про помилки

НФВ-07	Зручність	Веб-інтерфейс повинен підтримувати сучасні браузері (Chrome, Firefox, Edge актуальних версій)
НФВ-08	Сумісність	Система повинна функціонувати на операційних системах Windows 10+, Ubuntu 22.04+
НФВ-09	Масштабованість	Архітектура повинна допускати горизонтальне масштабування без зміни бізнес-логіки

Аналіз наведених вимог дозволяє окреслити межі системи та визначити її місце у просторі інформаційних технологій безпеки. Розроблювана система орієнтована на веб-середовище, що забезпечує кросплатформну доступність без необхідності встановлення додаткового програмного забезпечення. Наявність модуля стегоаналізу перетворює систему на інструмент подвійного призначення: не лише для приховування, а й для виявлення прихованих даних — що є критично важливим у задачах цифрової криміналістики та корпоративної безпеки.

1.2 Огляд методів цифрової стеганографії та стеганоаналізу

Цифрова стеганографія оперує поняттями контейнера (cover medium) та стего-об'єкта (stego object). Ефективність системи визначається трьома конкуруючими параметрами: місткістю (embedding capacity), непомітністю (imperceptibility) та стійкістю (robustness). Збільшення будь-якого з них, як правило, погіршує два інших, що формує простір компромісних рішень.

Усі методи поділяються на дві категорії: просторова область (spatial domain) — оперують безпосередньо значеннями пікселів (LSB, LSBM, Adaptive, PVD); частотна область (frequency domain) — вбудовують дані в коефіцієнти DCT/DWT/DFT перетворень (F5, JSteg). Методи просторової області простіші у реалізації та мають більшу ємність, але є вразливішими до статистичних атак. У розробленій системі реалізовано методи просторової області, оскільки вони забезпечують достатню ємність для текстових повідомлень у форматах PNG та BMP.

Наведена класифікація охоплює методи просторової області (LSB, LSBM, адаптивний) та методи частотної області, що оперують коефіцієнтами перетворень (рис. 1.3).

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				14
Змн.	Арк.	№ докум.	Підпис	Дата		



Рис. 1.3 – Класифікація методів стеганографії

Метод LSB базується на нечутливості людського зору до змін молодшого біта кольорового каналу [5]. Максимальна ємність для зображення $W \times H$ пікселів:

$$C = \left\lfloor \frac{W \times H \times 3}{8} \right\rfloor \text{ байтів}$$

Операція вбудовування біта b у піксель p та вилучення:

$$p' = (p \& 0xFE) | b \quad b = p' \& 0x01$$

Переваги: висока ємність (1 біт/канал), PSNR > 50 дБ при малому заповненні, простота реалізації. Недолік: вбудовування вирівнює частоти пар значень $\{2k, 2k + 1\}$ у гістограмі, що є основою χ^2 -атаки Вестфельда–Пфітцмана [6]. Структуру молодшого біта в пікселі показано на біт-діаграмі (рис. 1.4).

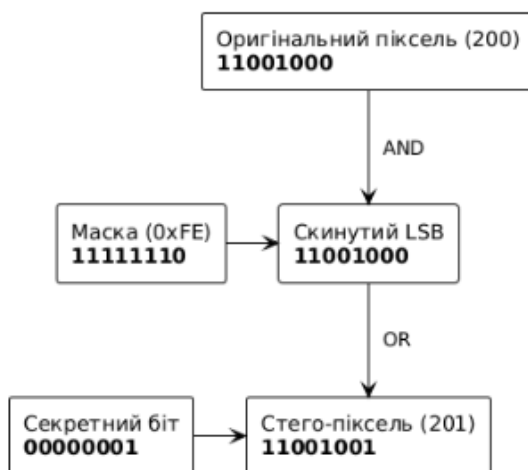


Рис. 1.4 – Біт-діаграма пікселя (LSB)

Метод LSBM усуває статистичну аномалію LSB: у разі незбігу молодшого біта замість прямої заміни виконується додавання або віднімання 1, обрані випадково:

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Концидайло А.М.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

$$p' = p + \delta, \delta \in \{-1, +1\} \text{ — випадково}$$

Граничні випадки: якщо $p = 0$ — лише $+1$; якщо $p = 255$ — лише -1 . Оскільки обидва члени пари $\{2k, 2k + 1\}$ модифікуються рівноімовірно, характерне вирівнювання гістограми не виникає, що суттєво підвищує стійкість до χ^2 -атаки при збереженні тієї ж ємності [7]. Відповідні гістограми наведено (рис. 1.5).

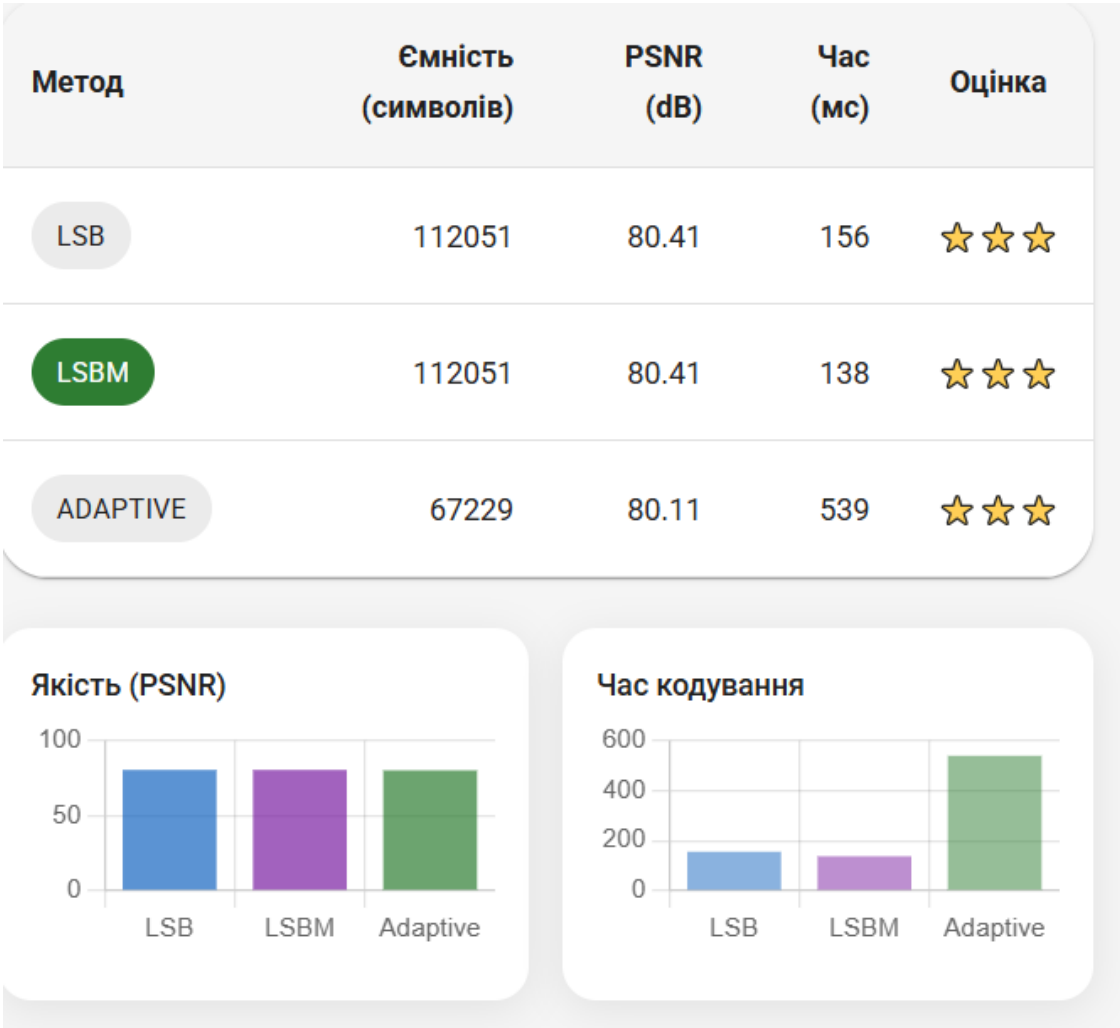


Рис. 1.5 – Гістограми

Адаптивний метод (Edge-Aware) концентрує вбудовування у текстурованих ділянках, де зміни пікселів найменш помітні для ока. Для виявлення країв застосовується оператор Собеля (матриці згортки 3×3):

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, G_y = G_x^T$$

$$G(x, y) = \sqrt{G_x^2 + G_y^2}$$

Алгоритм адаптивного вбудовування виконується за такою послідовністю кроків:

1. Конвертувати зображення у grayscale: $L = 0.299R + 0.587G + 0.114B$
2. Застосувати оператор Собеля $\rightarrow$ карта країв
3. Відсортувати пікселі за спаданням G
4. Відібрати позиції вище 40-го перцентилію ($\sim 60\%$ пікселів)
5. Вбудувати біти методом LSB у відсортованому порядку

Адаптивний метод досягає кращих значень PSNR порівняно з LSB при однаковому обсязі даних завдяки концентрації змін у менш чутливих ділянках [8][9]. Карту відмінностей пікселів для адаптивного методу показано (рис. 1.6).

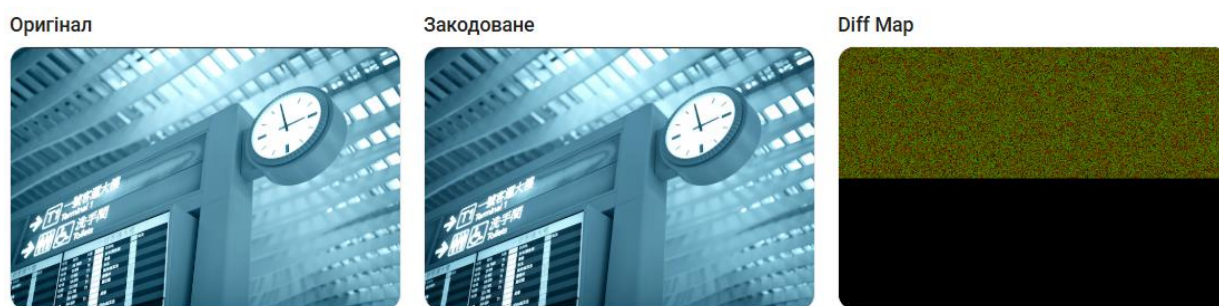


Рис. 1.6 – Pixel Diff Map

Data Whitening - Нерівномірний розподіл бітів текстового payload (ASCII-символи мають нульовий старший біт, кирилиця — характерні патерни) є додатковим індикатором для стегоаналізу. Data Whitening перетворює дані на рівномірну бітову послідовність ($\approx 50\%$ нулів і одиниць) за три кроки.

Спочатку текст кодується у UTF-8 — перетворюється на байтовий масив. Далі байти стискаються алгоритмом DEFLATE (zlib, рівень 9) [10], що усуває кореляції між байтами та зменшує розмір payload на 30–60%. Нарешті виконується XOR-скремблювання [11]: байти XOR-уються з псевдовипадковим потоком на основі HMAC-SHA256 у режимі лічильника: $K_i = \text{HMAC-SHA256}(K_0, i)$, де K_0 — фіксований ключ

Результат наближається до розподілу Бернуллі $p = 0.5$, що знімає статистичні аномалії незалежно від обраного методу вбудовування. Блок-схему конвеєра Data Whitening наведено (рис. 1.7).

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Концидайло А.М.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

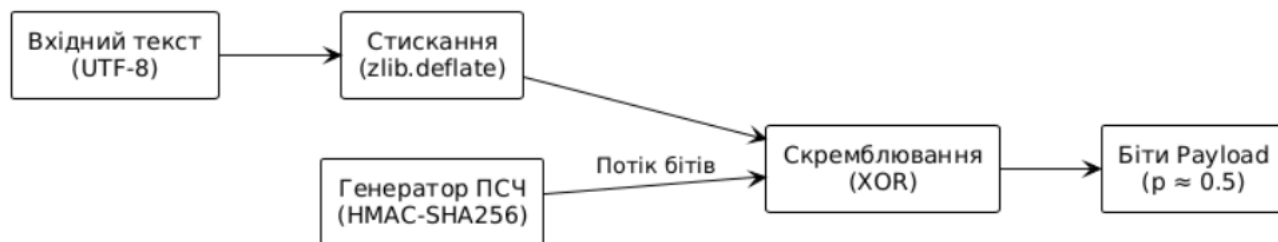


Рис. 1.7 – Блок-схема Data Whitening

Якість приховування інформації оцінюється двома метриками — середньо-квадратичною похибкою MSE та піковим відношенням сигналу до шуму PSNR, які обчислюються за формулами:

$$\text{MSE} = \frac{1}{W \cdot H \cdot 3} \sum_{x,y,c} (I(x,y,c) - I'(x,y,c))^2 \quad \text{PSNR} = 10 \cdot \log_{10} \left(\frac{255^2}{\text{MSE}} \right)$$

Граничне значення непомітності у системі: $\text{PSNR} \geq 40$ дБ.

Стегоаналіз: χ^2 -атака Вестфельда–Пфітцмана

χ^2 -атака виявляє LSB-стеганографію через аналіз пар значень (PoV). При вбудовуванні частоти O_{2k} та O_{2k+1} вирівнюються до очікуваного значення $E_k = (O_{2k} + O_{2k+1})/2$. Статистика Пірсона:

$$\chi^2 = \sum_{k=0}^{K-1} \frac{(O_{2k} - E_k)^2}{E_k} + \frac{(O_{2k+1} - E_k)^2}{E_k}, \quad \text{df} = K - 1$$

p-value обчислюється через верхню неповну гамма-функцію:

$$p = Q \left(\frac{\text{df}}{2}, \frac{\chi^2}{2} \right)$$

При $p < 0.05$ — зображення чисте; при $p > 0.9$ — висока ймовірність вбудовування. Для виявлення часткового вбудовування застосовується ковзне вікно ($\max(1000, \text{totalPixels}/20)$ пікселів, $\text{stride} = \text{windowSize}/2$, тобто 50% перекриття); фінальна ймовірність:

$$p_{\text{final}} = \max(p_{\text{overall}}, \max_i p_{\text{window}_i})$$

Порівняльна характеристика методів стеганографії

Критерій	LSB	LSBM	Adaptive
Ємність (біт/піксель)	3 (R+G+B)	3 (R+G+B)	≤ 3 (лише краї)
PSNR при малому заповн.	~83 дБ	~83 дБ	~84 дБ
Стійкість до χ^2 -атаки	Низька	Середня	Низька*
Залежність від структури	Ні	Ні	Так (Собель)
Складність реалізації	Низька	Низька	Середня
Вразливість до WS-атаки	Висока	Середня	Низька

Жоден із методів не є ідеальним за всіма критеріями. У розробленій системі реалізовано всі три, що дозволяє обирати оптимальний баланс між ємністю та стійкістю. Підвищення стійкості до χ^2 -атаки незалежно від методу досягається попереднім застосуванням Data Whitening.

1.3 Аналіз існуючих програмних аналогів

Перед розробкою власної системи необхідно провести детальний аналіз існуючих програмних рішень у галузі цифрової стеганографії. Мета аналізу полягає у виявленні функціональних прогалин наявних інструментів, обґрунтуванні доцільності власної розробки та формуванні вимог, що забезпечать конкурентні переваги. Для порівняння обрано чотири найпоширеніших інструментів, що охоплюють різні типи рішень: десктопні застосунки (OpenStego, Steghide, SilentEye, OpenPuff).

OpenStego — безкоштовний застосунок з відкритим кодом, написаний на мові Java, що забезпечує крос-платформну сумісність (Windows, Linux, macOS). Є одним із найбільш цитованих у науковій літературі [12] стеганографічних інструментів. Реалізує два основних режими роботи: приховування даних (data hiding) та цифровий водяний знак (digital watermarking) на основі алгоритму Дугада. Інтерфейс програми показано (рис. 1.8).

Introduction

Welcome to the homepage of OpenStego, the free steganography solution.

OpenStego provides two main functionalities:

- **Data Hiding:** It can hide any data within a cover file (e.g. images).
- **Watermarking (beta):** Watermarking files (e.g. images) with an invisible signature. It can be used to detect unauthorized file copying.

Please see [Concepts](#) page to learn more.

Using OpenStego

Using OpenStego is pretty straightforward. There are two modes of operation - data hiding and watermarking.

Data Hiding

In this mode, you can either hide the data (file) inside an image or extract the data from the image. Check screenshots below to see how it can be done:

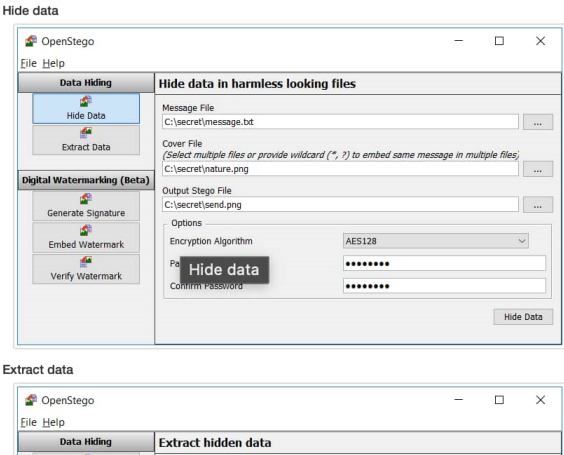


Рис. 1.8 – Інтерфейс OpenStego

За методом вбудовування OpenStego використовує RandomLSB (рандомізований LSB) — єдиний доступний плагін для стеганографії. Підтримуються формати PNG та BMP (виключно растрові зображення без стиснення з втратами), а шифрування виконується алгоритмами AES-128 та AES-256 на основі пароля. Власний модуль стегоаналізу відсутній. Інтерфейс десктопний графічний (Java Swing) і потребує встановлення JRE; програма розповсюджується за ліцензією GNU GPL v2.

До основних недоліків OpenStego належать відсутність стегоаналітичного модуля, підтримка лише одного методу стеганографії, застарілий інтерфейс і неможливість використання через веб-браузер. Архітектура також не підтримує мультикористувацький режим та збереження історії операцій.

Steghide — консольний інструмент із відкритим кодом для операційних систем Linux та Windows, орієнтований на досвідчених користувачів. Використовує граф-теоретичний підхід до стеганографії [13]: алгоритм вбудовування розглядає зображення як граф і шукає оптимальне паросполучення для мінімізації кількості змінених пікселів. Інтерфейс Steghide наведено (рис. 1.9).

```

(Luv@kali)-[/home/kali/Desktop]
$ steghide info luv.jpeg
"luv.jpeg":
  format: Windows 3.x bitmap
  capacity: 163.5 KB
Try to get information about embedded data ? (y/n) y
Enter passphrase:
  embedded file "luv.txt":
    size: 20.0 Byte
    encrypted: rijndael-128, cbc
    compressed: yes

```

Рис. 1.9 – Інтерфейс Steghide

Steghide реалізує власний алгоритм вбудовування — граф-теоретичний LSB з мінімальною кількістю змін. Підтримуються формати JPEG, BMP, WAV та AU, тоді як PNG не підтримується. Шифрування вбудоване (Rijndael-128, Triple-DES, Blowfish та інші), а модуль стегоаналізу відсутній. Робота здійснюється виключно через CLI, а ліцензія — GNU GPL v2.

Головним недоліком Steghide є відсутність підтримки формату PNG — найпоширенішого веб-формату зображень без стиснення з втратами; крім того, відсутні графічний інтерфейс, модуль стегоаналізу та можливість веб-використання. Відсутність підтримки PNG є суттєвим обмеженням для сучасних сценаріїв застосування, оскільки PNG є стандартним форматом для обміну якісними зображеннями в мережі.

SilentEye — крос-платформний застосунок з відкритим кодом (Windows, macOS, Linux), що відрізняється дружнім графічним інтерфейсом та підтримкою плагінів для нових алгоритмів. Орієнтований на початківців, які потребують мінімального порогу входу [14]. Інтерфейс програми показано (рис. 1.10).

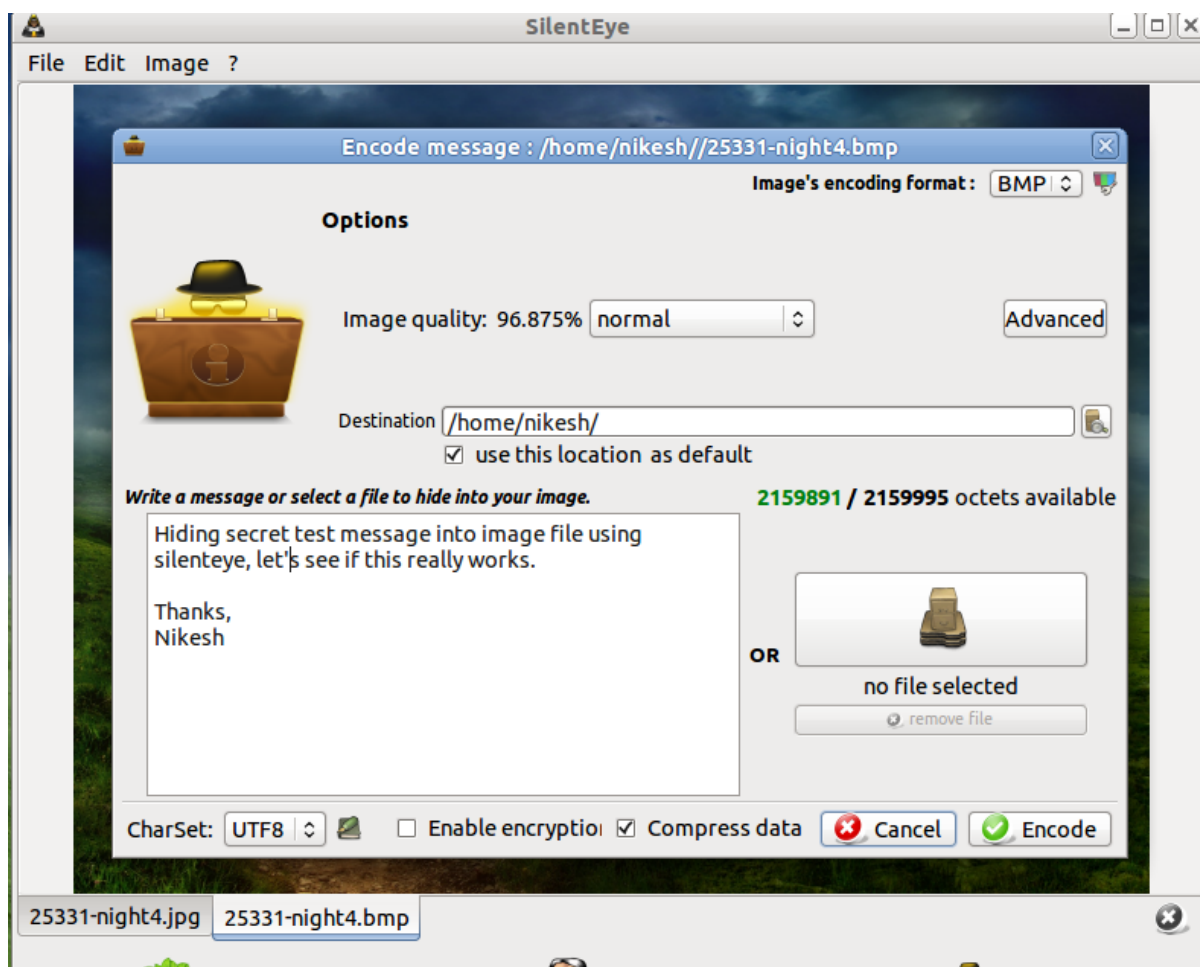


Рис. 1.10 – Інтерфейс SilentEye

SilentEye вбудовує дані методом LSB у зображення та аудіо. Підтримуються формати BMP (для зображень) і WAV (для аудіо), тоді як JPEG та MP3 були заплановані, але так і не реалізовані. Шифрування доступне через плагін (AES), а модуль стегоаналізу відсутній. Інтерфейс графічний (Qt), простий і зручний; ліцензія GNU GPL v3, проте репозиторій не оновлювався з 2014 року.

Серед недоліків SilentEye — підтримка лише BMP-формату для зображень (великий розмір файлів і непрактичність), відсутність PNG та стегоаналізу, а також те, що проєкт фактично не підтримується з 2014 року. Використання застарілого незбалансованого інструменту у виробничому середовищі несе ризики безпеки.

OpenPuff — найфункціональніший серед розглянутих десктопних інструментів, позиціонується як професійне рішення для прихованої передачі даних [15]. Реалізує концепцію ланцюжків носіїв (carrier chains): дані розбиваються

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

між кількома файлами-носіями, що суттєво ускладнює виявлення. Інтерфейс OpenPuff наведено (рис. 1.11).



Рис. 1.11 – Інтерфейс OpenPuff

OpenPuff застосовує адаптивне нелінійне кодування з вибором кількості бітів на носій. Підтримується широкий набір форматів: зображення (BMP, JPG, PNG, TGA), аудіо (AIFF, MP3, NEXT/SND, WAV), відео (3GP, MP4, MOV, FLV) та Flash (SWF, PDF). Шифрування реалізоване 16 алгоритмами, серед яких AES, 3DES та RC4, а безпеку посилюють три паролі, апаратно ініціалізований CSPRNG і відхиляльна стеганографія (deniable steganography). Реалізовано Data Whitening (рівномірний розподіл бітів) та заявлено стійкість до χ^2 -аналізу, однак власний модуль стегоаналізу стороннього зображення відсутній. Інтерфейс десктопний, лише для Windows; остання версія 4.01 датована 2018 роком.

Попри розширений функціонал, OpenPuff має низку недоліків: він не містить модуля стегоаналізу, не є веб-застосунком, не підтримує мультикористувачський режим, не зберігає історію операцій і не оновлювався з 2018 року.

Зведені результати порівняльного аналізу наведено (табл. 1.4).

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

Таблиця 1.4

Порівняльна характеристика програмних аналогів

Критерій	OpenStego	Steghide	SilentEye	OpenPuff	Наша система
Веб-інтерфейс	-	-	-	-	+
Підтримка PNG	+	-	-	+	+
Підтримка BMP	+	+	+	+	+
Метод LSB	+	+	+	+	+
Метод LSBM	-	-	-	-	+
Adaptive (Sobel)	-	-	-	-	+
AES-256 шифрування	+	+	+	+	+
Data Whitening	-	-	-	+	+
Стегоаналіз (χ^2)	-	-	-	-	+
Метрики PSNR/MSE	-	-	-	-	+
Мультикористув. режим	-	-	-	-	+
Пакетна обробка	-	-	-	-	+
Самознищ. повідомлення	-	-	-	-	+
Активна підтримка	+	+-	-	-	+

Аналіз (табл. 1.4) дозволяє виокремити три системні недоліки, притаманні всім розглянутим аналогам.

По-перше, більшість інструментів є десктопними застосунками з необхідністю встановлення, що унеможлиблює їх використання на мобільних платформах та у корпоративних середовищах з обмеженнями на встановлення стороннього ПЗ. Єдиний веб-інструмент (StegOnline [16]) позбавлений шифрування, авторизації та будь-якої серверної логіки.

По-друге, жоден із аналогів не реалізує одночасно три методи стеганографії — LSB, LSBM та Adaptive, — що позбавляє користувача можливості вибору оптимального алгоритму залежно від сценарію: максимальна ємність (LSB), стійкість до χ^2 -атаки (LSBM), або мінімальна помітність при стиснених зображеннях (Adaptive).

По-третє, жоден аналог не надає повноцінного модуля статистичного стегоаналізу з обчисленням χ^2 , p-value та аналізом у ковзному вікні. OpenPuff лише

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

заявляє стійкість до χ^2 -атаки, але не надає інструменту для незалежної верифікації цього твердження. Відсутній також розрахунок метрик якості PSNR та MSE, без яких неможливо об'єктивно оцінити ступінь непомітності вбудовування.

Виявлені прогалини підтверджують доцільність розробки власної системи, що поєднує в єдиній веб-платформі: три методи стеганографії, шифрування AES-256 з Data Whitening, повноцінний χ^2 -стегааналіз, метрики PSNR/MSE та мультикористувацький режим. Функціональні та нефункціональні вимоги до системи сформульовано у підрозділі 1.1.

1.4 Вибір архітектури та інструментальних засобів

Обґрунтований вибір архітектурного підходу та технологічного стеку є фундаментальним рішенням, що визначає масштабованість та підтримуваність системи. У цьому підрозділі аналізуються альтернативні рішення та обґрунтовується вибір конкретних технологій відповідно до вимог підрозділу 1.1.

Для системи обрано клієнт-серверну архітектуру з розподілом на Frontend (SPA) та Backend (REST API). REST API базується на стандартних HTTP-методах та принципі statelessness: кожен запит містить усі необхідні дані, що забезпечує горизонтальне масштабування. SPA реалізує навігацію без перезавантаження сторінки, що забезпечує плавний UX.

Серед розглянутих альтернатив монолітну SSR-архітектуру відхилено, оскільки вона ускладнює логіку стеганографічних обчислень і знижує гнучкість API; мікросервісну архітектуру — як надлишково складну для проєкту обсягом близько 6300 рядків коду, що розробляється однією особою; serverless-підхід — через обмеження на розмір бінарних payload і відсутність постійного з'єднання з MongoDB. Як основну мову програмування обрано TypeScript [17] — надмножину JavaScript зі статичною типізацією, яка посідає 3-тє місце в рейтингу Stack Overflow Developer Survey 2024 [18]; вона застосовується для обох частин системи (backend і frontend), що дозволяє повторно використовувати API-типи й уникати помилок невідповідності контракту між клієнтом і сервером.

Статична типізація виявляє помилки на етапі компіляції, що критично при роботі з бітовими операціями та буферами пікселів. Інтерфейси чітко описують структури PixelBuffer, EmbedOptions і SteganalysisResult. Використання єдиної мови для frontend і backend усуває контекстне перемикання та дозволяє застосовувати спільні типи.

Як серверний фреймворк обрано Node.js з Express — асинхронне серверне середовище на рушії V8. Express.js залишається найпопулярнішим серверним фреймворком [19] за Stack Overflow Developer Survey 2024.

Переваги Node.js + Express для даної системи.

Асинхронна обробка введення-виведення забезпечує ефективне паралельне опрацювання кількох запитів стеогоаналізу без блокування потоку. Вбудований Buffer API надає низькорівневу роботу з бінарними даними (масивами пікселів) без сторонніх бібліотек. NPM-екосистема включає Jimp для обробки зображень, а також jsonwebtoken, bcrypt і mongoose.

Таблиця 1.5

Порівняння серверних технологій

Критерій	Node.js Express	Python FastAPI	Java + Spring Boot	PHP + Laravel
Мова frontend	TypeScript (спільна)	Різні	Різні	Різні
Робота Buffer/бінарними даними	Нативна	Нативна	Нативна	Обмежена
Асинхронність	Event Loop	AsyncIO	Threads	Обмежена
NPM-бібліотека Jimp	Нативна	-	-	-
Поріг входу	Низький	Низький	Високий	Середній
Популярність	№1 серверний	Зростає	Корпоративний	Знижується
Розгортання	Просте	Просте	JVM overhead	Просте

Клієнтською бібліотекою обрано React — декларативну бібліотеку для побудови інтерфейсів [20] (Meta, 2013). Її використовують 44,7% розробників згідно зі Stack Overflow Developer Survey 2024, що робить її найпопулярнішим веб-фреймворком.

Компонентна архітектура дозволяє виділити ізольовані компоненти ImageUploader, SteganalysisChart, PixelDiffMap та HistogramOverlay. Virtual DOM

оптимізує оновлення інтерфейсу при роботі з великими зображеннями, а React Hooks (useState, useEffect, useCallback) зменшують обсяг шаблонного коду. Еко-система React надає Chart.js для гістограм і графіків стеогоаналізу та React Router для навігації.

У порівнянні з Angular — менший час завантаження та відсутність жорсткої структури. У порівнянні з Vue.js — ширша екосистема та краща підтримка TypeScript.

Базою даних обрано MongoDB — документно-орієнтовану NoSQL СУБД (формат BSON) [21], якій надано перевагу над реляційними альтернативами (PostgreSQL, MySQL) через природу даних системи.

Таблиця 1.6

Порівняння MongoDB та PostgreSQL для даного проєкту

Критерій	MongoDB	PostgreSQL
Модель даних	Документна (BSON/JSON)	Реляційна (таблиці/рядки)
Гнучкість схеми	Висока (schema-less)	Жорстка (migrations)
JOIN-операції	Через \$lookup	Нативні
JSON-документи	Нативні	JSONB поле
TypeScript ORM/ODM	Mongoose	Prisma / TypeORM
Горизонтальне масштаб.	Sharding нативний	Складніше
ACID-транзакції	(з v4.0, документ-рівень)	Повні
Оптимальність для проєкту	Висока	Середня

Ключові аргументи на користь MongoDB: колекція messages має варіативну структуру залежно від методу кодування, що у реляційній БД потребувало б додаткових таблиць; Mongoose ODM забезпечує схемну валідацію, pre/post-хуки та TypeScript-типізацію; нативний JSON усуває перетворення структур при формуванні API-відповідей.

Для автентифікації застосовано JWT — компактний токен з цифровим підписом [22], що складається з Header (алгоритм), Payload (userId, role, exp) та Signature (HMAC-SHA256). Порівняно із сесійною автентифікацією він забезпечує stateless-перевірку без звернення до бази даних на кожен запит, а поле role дозволяє реалізувати RBAC без додаткових запитів.

У системі реалізовано двотокенну схему автентифікації. Короткоживучий access token зберігає чинність протягом 15 хвилин, тоді як refresh token діє впродовж 7 діб; обидва токени зберігаються у браузерному сховищі localStorage. У

разі отримання статусу 401 механізм автоматичного оновлення (silent refresh), реалізований через response-interceptor бібліотеки Axios, отримує новий access token, не вимагаючи повторного входу користувача.

Для обробки зображень використано бібліотеку Jimp — рішення для Node.js без жодних нативних залежностей [23] (ліцензія MIT, версія 0.22.12).

Бібліотека надає прямий доступ до сирого масиву пікселів: image.bitmap.data повертає Uint8Array зі значеннями RGBA, а індекс пікселя (x, y) обчислюється через getPixelIndex(x, y) — це фундаментальна операція для методів LSB, LSBM та Adaptive. Метод .scan() виконує ітерацію по всіх пікселях з координатами (x, y) та індексом idx і відкриває доступ до каналів R(idx), G(idx+1) і B(idx+2). Формати PNG та BMP зберігаються без стиснення з втратами, що гарантує цілісність вбудованих стеганографічних бітів. Нарешті, відсутність нативних залежностей (на відміну від Sharp, який потребує компіляції C++-розширення libvips) дозволяє виконувати npm ci без помилок незалежно від середовища.

Вибір Jimp замість Sharp обґрунтовується пріоритетами: для стеганографії критичним є побітовий доступ до кожного пікселя, а не пропускна здатність потокової обробки. API bitmap.data[idx] забезпечує пряму модифікацію буфера без проміжних перетворень, тоді як потокова модель Sharp орієнтована на масштабування та ресайз — надлишковий для поставленого завдання функціонал.

Обраний технологічний стек системи охоплює мову TypeScript, серверне середовище Node.js з Express, клієнтську бібліотеку React, базу даних MongoDB, автентифікацію на основі JWT та бібліотеку обробки зображень Jimp (рис. 1.12).

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

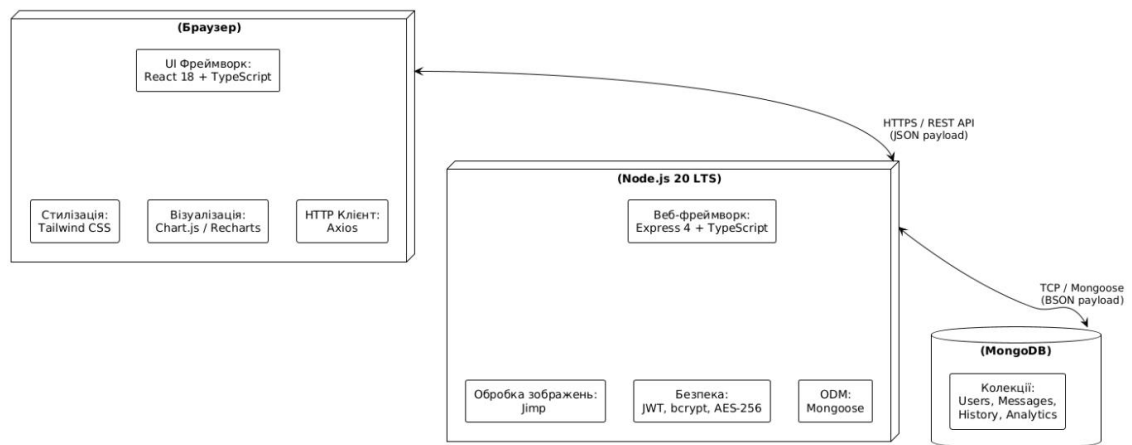


Рис. 1.12 – Технологічний стек системи

Взаємодія між компонентами стеку детально розглядається у підрозділі 2.1 на основі UML Component та Deployment діаграм.

Висновки до першого розділу

У першому розділі проведено комплексний аналіз предметної області та виконано підготовчі дослідження для проектування системи.

За результатами аналізу предметної області (підрозділ 1.1) сформульовано мету роботи та визначено перелік функціональних (ФВ-01 — ФВ-11) і нефункціональних (НФВ-01 — НФВ-09) вимог до системи.

Аналіз методів стеганографії (підрозділ 1.2) обґрунтував вибір трьох методів вбудовування — LSB, LSBM та Adaptive (Sobel), — а також підтвердив доцільність реалізації χ^2 -стегааналізу та метрик якості PSNR і MSE для об'єктивної оцінки непомітності.

Порівняльний аналіз чотирьох програмних аналогів (підрозділ 1.3) виявив спільні системні недоліки існуючих рішень: відсутність веб-доступності, обмеженість одним методом стеганографії та відсутність повноцінного модуля статистичного стегааналізу, що підтвердило доцільність власної розробки.

Обґрунтування технологічного стеку (підрозділ 1.4) визначило архітектуру Client-Server REST API + SPA та повний набір інструментів: TypeScript, Node.js + Express, React, MongoDB, JWT, Jimp та AES-256-CBC.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

2.1 Архітектура системи та варіанти використання

Узагальнена архітектура системи — система побудована на основі клієнт-серверної архітектури з чітким розподілом на Frontend (SPA) та Backend (REST API). Взаємодія між рівнями здійснюється виключно через HTTP/HTTPS з передачею даних у форматі JSON, що забезпечує незалежність клієнтської та серверної частин і можливість їх окремого розгортання.

Компонентна діаграма системи (рис. 2.1) відображає чотири підсистеми та зв'язки між ними.

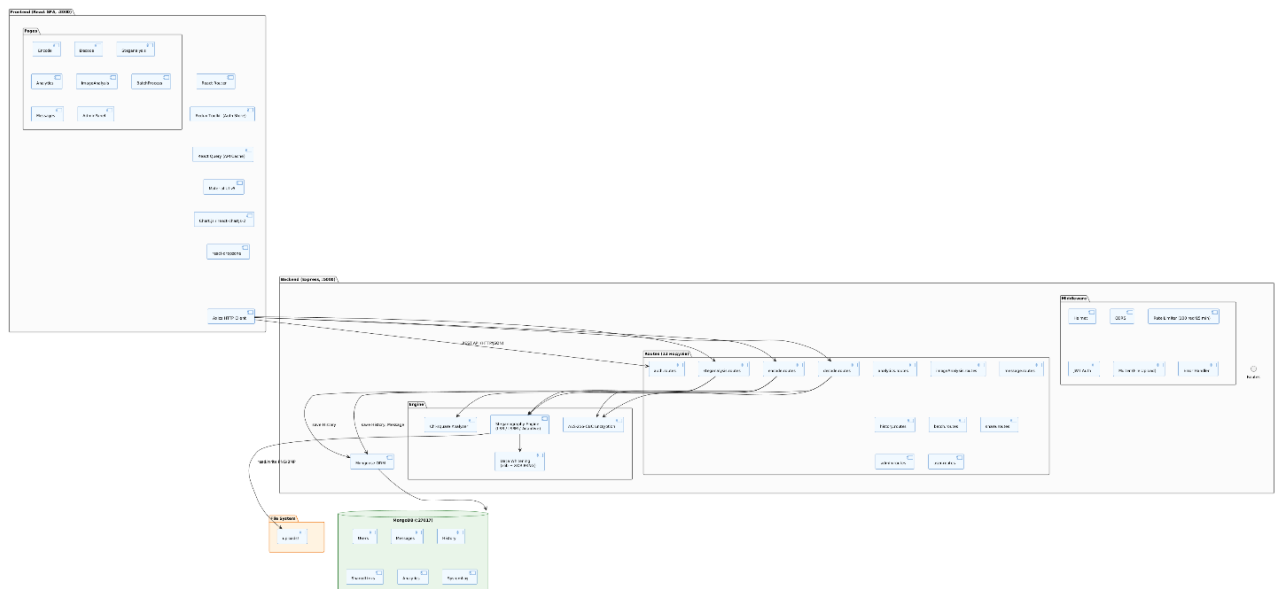


Рис. 2.1 – UML Component діаграма

Frontend — це односторінковий вебзастосунок (SPA) на основі бібліотеки React, який працює на порту 3000 і складається з кількох взаємопов'язаних компонентів. Маршрутизацію між сторінками без перезавантаження забезпечує React Router, централізоване управління станом автентифікації — Redux Toolkit (Auth Store), а кешування й інвалідацію API-відповідей — React Query (API Cache). Транспортний рівень реалізовано через Axios HTTP Client з interceptors, які автоматично додають JWT-токен до кожного запиту. Інтерфейс користувача побудовано на бібліотеці компонентів Material UI v5, засобах візуалізації Chart.js і react-chartjs-2 для графіків та гістограм, а також компоненті react-dropzone для завантаження файлів.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Концидайло А.М.				30
Змн.	Арк.	№ докум.	Підпис	Дата		

Backend — це серверний застосунок на фреймворку Express, що працює на порту 5000 й організований як конвеєр проміжних обробників (middleware) та багаторівнева архітектура. Конвеєр middleware охоплює модуль Helmet, що встановлює захисні HTTP-заголовки, механізм CORS, обмежувач частоти запитів Rate Limiter (до 200 запитів за 15 хвилин), middleware перевірки JWT-токенів, модуль Multer для завантаження файлів та централізований обробник помилок Error Handler. Маршрутизацію розподілено між дванадцятьма модулями: auth, encode, decode, steganalysis, analytics, imageAnalysis, message, history, batch, share, admin і user. Обчислювальне ядро (Engine) об'єднує стеганографічний движок (LSB, LSBM, Adaptive), механізм Data Whitening (zlib і XOR-PRNG), аналізатор χ^2 -тесту та модуль шифрування AES-256-CBC, тоді як рівень доступу до даних реалізовано через Mongoose ODM поверх MongoDB.

Базу даних реалізовано на СУБД MongoDB, яка приймає підключення на порту 27017 і містить шість колекцій: Users, Messages, History, SharedLinks, Analytics та SystemLog. Оригінальні та стего-зображення зберігаються у файловій системі сервера, у директорії uploads/.

Діаграма розгортання (рис. 2.2) описує фізичну топологію системи та комунікаційні зв'язки між вузлами.

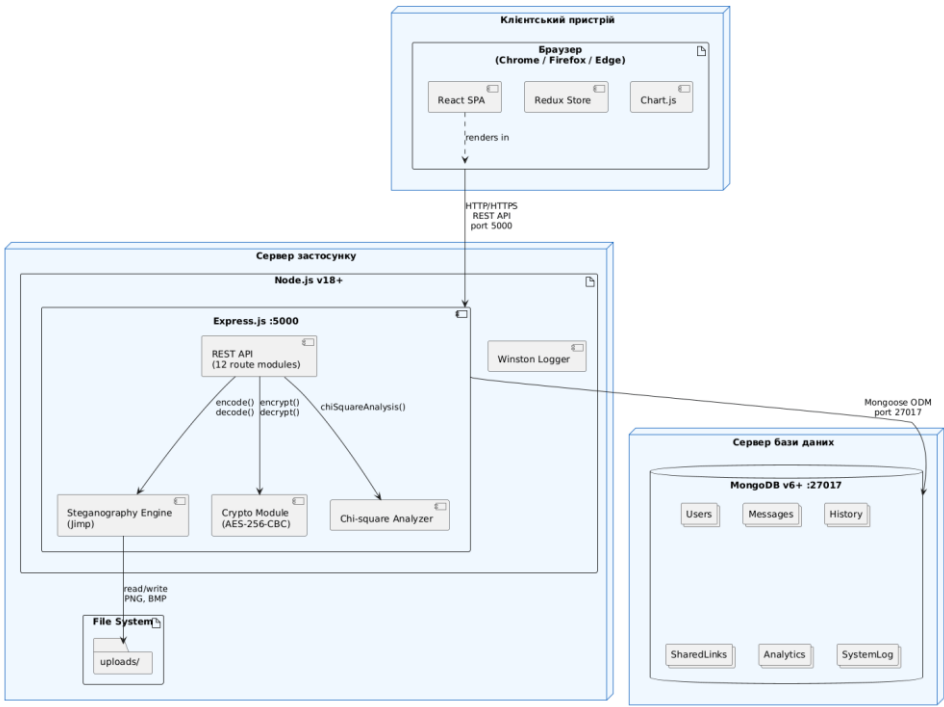


Рис. 2.2 – UML Deployment діаграма

Система розгортається на трьох вузлах. Клієнтський пристрій (браузер Chrome/Firefox/Edge) виконує React SPA, Redux Store та Chart.js. Сервер застосунку запускає Node.js 18+ з Express на порту 5000; у його складі — REST API (12 route-модулів), Steganography Engine (Jimp), Crypto Module (AES-256-CBC), Chi-square Analyzer та Winston Logger. Сервер бази даних виконує MongoDB 6+ на порту 27017.

Браузер взаємодіє з сервером застосунку через REST API за протоколом HTTP/HTTPS на порту 5000, а сам сервер підключається до бази даних MongoDB через Mongoose ODM на порту 27017. Steganography Engine виконує операції читання та запису у файловій системі (директорія uploads/) для збереження зображень у форматах PNG і BMP.

Use Case діаграма (рис. 2.3) описує функціональні можливості системи з точки зору трьох зовнішніх акторів.

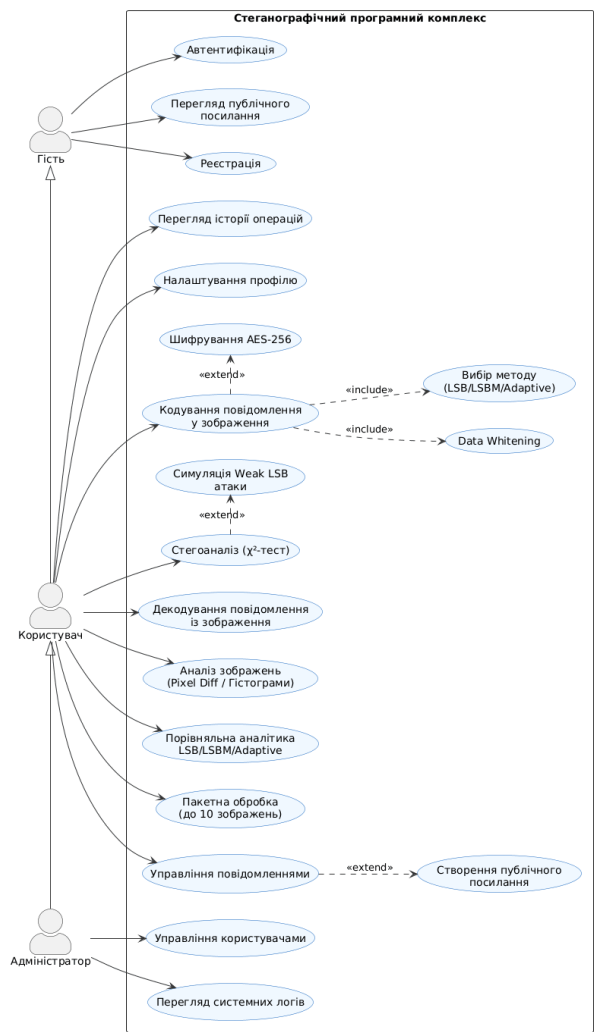


Рис. 2.3 – UML Use Case діаграма

Система визначає три актори з ієрархією успадкування: Гість → Користувач → Адміністратор. Зв'язок Admin --|> User --|> Guest означає, що кожен наступний рівень успадковує всі права попереднього та доповнює їх власними.

Гість має доступ до трьох варіантів використання: реєстрація, автентифікація та перегляд публічного посилання (без авторизації). Користувач отримує доступ до дев'яти функцій: кодування повідомлень у зображенні, декодування повідомлень, стегоаналіз (χ^2 -тест), аналіз зображень (Pixel Diff / гістограми), порівняльна аналітика методів LSB/LSBM/Adaptive, пакетна обробка (до 10 зображень), управління повідомленнями, перегляд історії операцій, налаштування профілю. Адміністратор додатково виконує управління обліковими записами користувачів та перегляд системних логів.

Між варіантами використання діаграма фіксує чотири відношення типів «include» та «extend». Відношення «include» між «Кодуванням» і «Вибором методу» означає, що вибір одного з методів LSB, LSBM або Adaptive є обов'язковим, а відношення «include» між «Кодуванням» і «Data Whitening» — що рівномірний розподіл бітів виконується завжди. Натомість відношення «extend» між «Кодуванням» і «AES-256» активується лише за наявності пароля, а відношення «extend» між «Стегоаналізом» і «Симуляцією Weak LSB» є необов'язковою розширеною функцією.

На основі Use Case діаграми визначено три основні сценарії роботи системи.

Перший сценарій — кодування повідомлення. Автентифікований користувач завантажує зображення-контейнер (PNG або BMP), вводить текст, обирає метод стеганографії (LSB, LSBM або Adaptive) та за потреби задає пароль для шифрування AES-256. Система повертає стего-зображення та метрики якості (PSNR, MSE).

Другий сценарій — стегоаналіз. Користувач завантажує підозріле зображення для перевірки на наявність прихованих даних. Система виконує χ^2 -тест по трьох каналах (R, G, B) з ковзним вікном, виносить вердикт про наявність чи відсутність ознак стеганографії та будує графік розподілу χ^2 -статистики.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

Третій сценарій — адміністрування. Адміністратор через захищену адміністративну панель (маршрут /admin) переглядає список зареєстрованих користувачів, блокує або видаляє облікові записи та переглядає системні журнали рівнів info, warning і error для моніторингу безпеки.

Структуру REST API організовано у дванадцять модулів маршрутів під спільним префіксом /api (рис. 2.4).

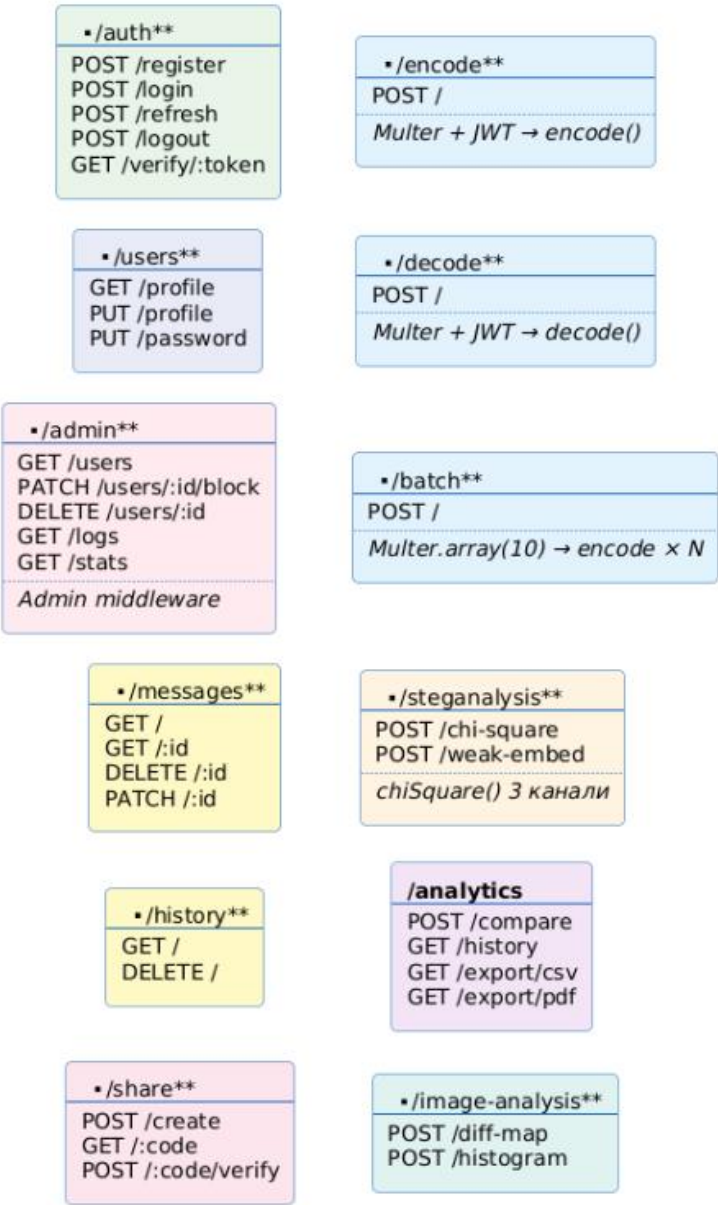


Рис. 2.4 – Структура REST API маршрутів

Спроектована клієнт-серверна архітектура та деталізований REST API повністю покривають усі функціональні вимоги до системи, визначені на етапі аналізу. Розподіл системи на ізольовані модулі забезпечує високу масштабованість

та полегшує подальшу підтримку коду. Сценарії використання чітко розмежовують права доступу між гостями, користувачами та адміністратором. Для забезпечення коректної роботи описаної бізнес-логіки та збереження стану системи (історії операцій, метрик, облікових записів) необхідно розробити надійну структуру зберігання даних, що розглядається у наступному підрозділі під час проектування бази даних.

2.2 Проектування бази даних

Проектування бази даних системи здійснено з урахуванням документно-орієнтованої природи MongoDB та специфіки предметної області: варіативної структури стеганографічних метаданих, необхідності TTL-автовидалення записів та гнучкого зберігання вкладених об'єктів результатів аналізу. Логічна модель даних охоплює шість колекцій та реалізована у вигляді Mongoose-схем у директорії backend/src/models/

Логічна модель даних (рис. 2.5) відображає шість колекцій MongoDB, повний перелік полів кожної колекції та зв'язки між ними через ObjectId-посилання.

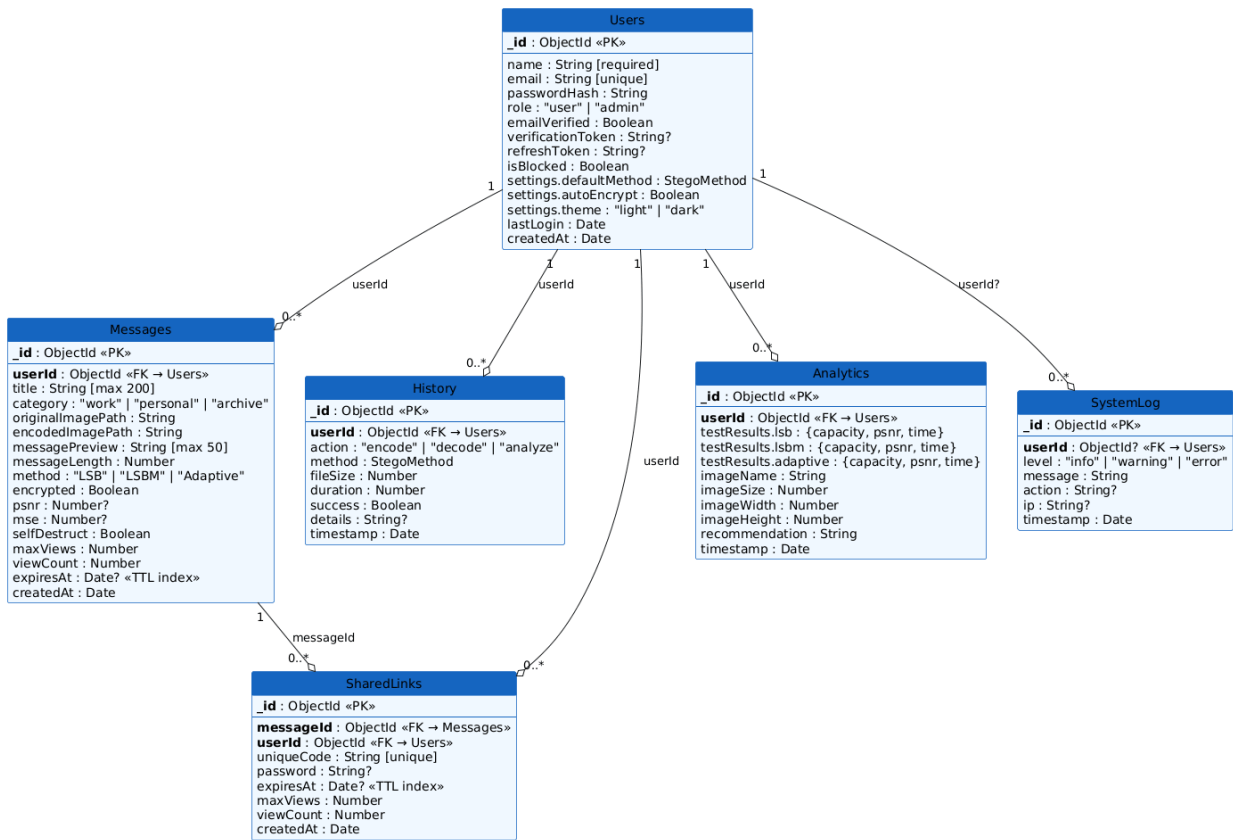


Рис. 2.5 – Логічна модель даних (ObjectId references)

Центральною є колекція Users, з якою пов'язані всі п'ять інших колекцій через поле `userId`. Колекція Messages додатково є батьківською для SharedLinks через поле `messageId`. Такий підхід відповідає принципу посилань (reference pattern) у MongoDB і дозволяє уникнути надмірного дублювання даних при збереженні гнучкості запитів.

Колекція Users зберігає облікові записи всіх зареєстрованих користувачів системи та є основою ролівої моделі доступу (RBAC).

Унікальний індекс `{ email: 1 }` забезпечує унікальність адреси та прискорює пошук при автентифікації, а звичайний індекс `{ role: 1 }` прискорює вибірку користувачів за роллю в адміністративній панелі.

Поле `passwordHash` зберігає виключно `bcrypt`-хеш з `cost factor 12` [24]; оригінальний пароль ніколи не зберігається. Поле `refreshToken` зберігається хешованим та очищується при виході (`POST /auth/logout`). Вкладений об'єкт `settings` реалізований як `subdocument` без окремої колекції, що дозволяє отримати налаштування користувача єдиним запитом без JOIN-операцій.

Колекція Messages є ключовою з точки зору предметної області — вона зберігає метадані кожного стеганографічного повідомлення, включаючи шляхи до зображень, параметри кодування та налаштування самознищення.

Складений індекс `{ userId: 1, createdAt: -1 }` покриває найпоширеніший запит — список повідомлень користувача, відсортований за датою у спадному порядку. TTL-індекс `{ expiresAt: 1 }` (`expireAfterSeconds: 0`) автоматично видаляє документ після досягнення значення `expiresAt`. Індекс `{ category: 1 }` використовується для фільтрації за категорією повідомлень (`work`, `personal`, `archive`), а текстовий індекс `{ title: 'text', messagePreview: 'text' }` — для повнотекстового пошуку за назвою та прев'ю повідомлення.

Механізм самознищення реалізовано через дві взаємодоповнюючі стратегії: TTL-індекс (автоматичне видалення за датою) та лічильник переглядів `viewCount / maxViews`. При кожному декодуванні або перегляді через `SharedLink middleware` виконує `viewCount++` та перевіряє умову `viewCount >=`

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

`maxViews && selfDestruct === true` — у цьому разі документ видаляється негайно через `Message.findByIdAndDelete()`.

Колекція `History` автоматично фіксує кожну операцію стеганографії, виконану користувачем, та формує журнал аудиту для відображення у розділі «Історія операцій».

Індекс: `{ userId: 1, timestamp: -1 }` — складений індекс для пагінованого виведення хронологічного журналу. Запис у `History` виконується автоматично у кожному `route-handler` через `History.create()` після завершення операції, незалежно від її результату (`success: true/false`).

Колекція `SharedLinks` реалізує механізм публічних посилань на стего-зображення з можливістю обмеження доступу паролем, лімітом переглядів та терміном дії.

Унікальний індекс `{ uniqueCode: 1 }` дозволяє знаходити запис за час $O(\log n)$ при переході за посиланням `GET /api/share/:code`, а TTL-індекс `{ expiresAt: 1 }` автоматично очищує прострочені посилання.

Публічне посилання має формат `/share/:code` та доступне без автентифікації. Якщо встановлено пароль, клієнт надсилає запит `POST /api/share/:code/access` з паролем, а бекенд порівнює його через `bcrypt.compare()` та повертає тимчасовий доступ.

Колекція `Analytics` зберігає результати порівняльного тестування трьох методів стеганографії на одному зображенні та формує основу для сторінки «Аналітика».

Вкладений об'єкт `testResults` є прикладом `bucket pattern` у `MongoDB`: три підоб'єкти з однаковою структурою зберігаються в одному документі, що дозволяє отримати повні результати порівняльного тесту єдиним запитом без агрегації.

Колекція `SystemLog` зберігає структуровані записи журналу сервера. Записи у `SystemLog` створюються безпосередньо у `route-handler`'ах через `SystemLog.create()` (наприклад, при кодуванні, реєстрації), а `Winston Logger` паралельно веде файлове та консольне логування.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

Складений індекс { level: 1, timestamp: -1 } призначено для фільтрованого перегляду логів в адміністративній панелі, а індекс { userId: 1 } — для пошуку логів за конкретним користувачем.

Поле userId є необов'язковим, оскільки частина системних подій (наприклад, неавторизовані запити або помилки сервера) не пов'язана з конкретним користувачем. Запис рівня «error» завжди містить трасування стеку у полі details, що полегшує діагностику.

Зв'язки між колекціями та стратегія запитів. У MongoDB відсутні нативні JOIN-операції; зв'язки між документами реалізовано через ObjectId references (reference pattern) з явним populate через Mongoose.

Ключові принципи організації зв'язків у системі.

Зв'язок Users з колекціями Messages, History та Analytics реалізовано через поле userId у дочірніх колекціях: при запиті списку повідомлень користувача виконується Message.find({ userId }).sort({ createdAt: -1 }) без populate, оскільки дані колекції Users у цьому контексті не потрібні. Зв'язок Messages із SharedLinks використовує SharedLink.findOne({ uniqueCode }).populate('messageId') для отримання шляхів до зображень при відкритті публічного посилання. Каскадне видалення реалізовано явно: при видаленні користувача пов'язані документи з колекцій Messages, History, Analytics та SharedLinks вилучаються окремими запитамі deleteMany({ userId }). Автоматичне очищення за часом життя забезпечує MongoDB TTL Worker, який кожні 60 секунд видаляє прострочені документи з колекцій Messages і SharedLinks без участі прикладного коду.

Усі шість Mongoose-схем зберігаються у директорії backend/src/models/ (файли User.ts, Message.ts, History.ts, SharedLink.ts, Analytics.ts, SystemLog.ts) і підключаються до Express-застосунку через єдину точку ініціалізації config/database.ts.

2.3 Розробка алгоритмів стеганографії та криптозахисту

Алгоритмічне ядро системи реалізовано у файлі backend/src/engine/steganography.ts (~940 рядків) та

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Концидайло А.М.				38
Змн.	Арк.	№ докум.	Підпис	Дата		

backend/src/engine/encryption.ts. Усі алгоритми розроблено з урахуванням двох ключових вимог: непомітності вбудовування (максимізація PSNR) та стійкості до статистичного стегааналізу (χ^2 -тест).

Перед вбудовуванням будь-яким із трьох методів текстовий payload обов'язково проходить через Data Whitening pipeline, а після формування бітового потоку до нього додається 2-бітний ідентифікатор методу та 32-бітна довжина payload, що дозволяє декодеру автоматично визначити метод без додаткових метаданих.

Pipeline Data Whitening (рис. 2.6) — це попередня обробка payload, що усуває статистичні закономірності вхідного тексту та забезпечує рівномірний розподіл бітів (~50/50 нулів і одиниць) перед вбудовуванням.

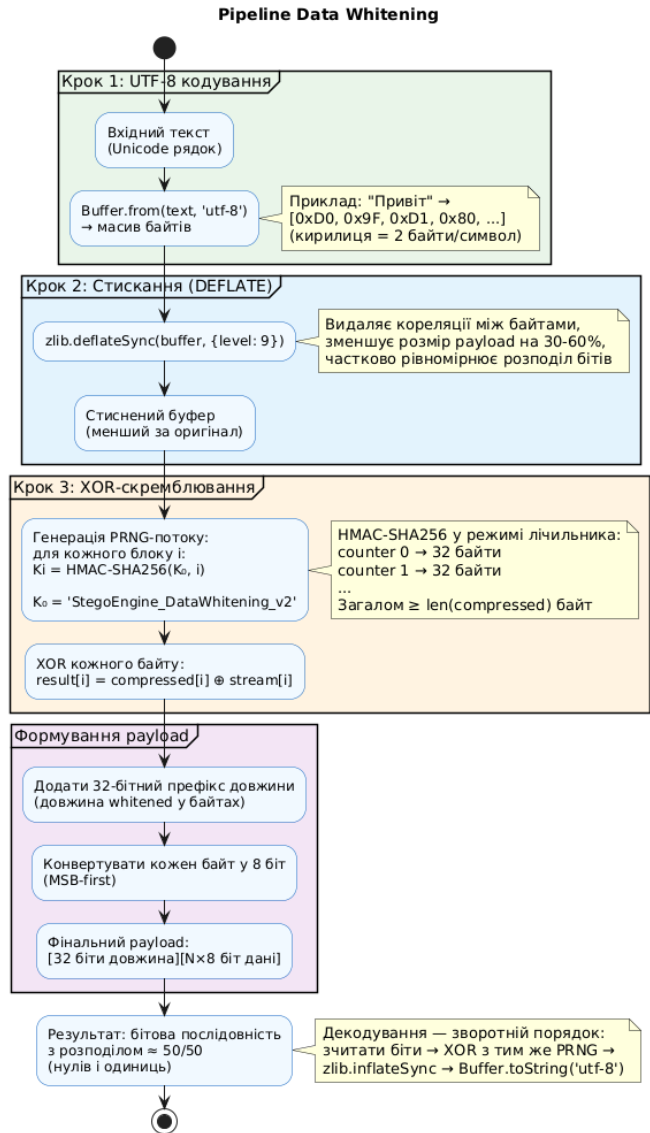


Рис. 2.6 – Pipeline Data Whitening

Спочатку вхідний Unicode-рядок кодується у UTF-8 і перетворюється на масив байтів через `Buffer.from(text, 'utf-8')`; кирилиця при цьому займає по два байти на символ (наприклад, «П» → `[0xD0, 0x9F]`), що враховується при обчисленні ємності контейнера. Потім байтовий буфер стискається алгоритмом DEFLATE через `zlib.deflateSync(buffer, {level: 9})`: це водночас зменшує розмір payload на 30–60% і руйнує лінгвістичні кореляції між байтами (повторювані символи, характерні частоти літер) і частково вирівнює розподіл.

Нарешті стиснений буфер XOR-ується з псевдовипадковим потоком, що генерується у режимі лічильника: $K_i = \text{HMAC-SHA256}(K_0, i)$ де $K_0 = \text{"StegoEngine_DataWhitening_v2"}$ — фіксований ключ-константа, i — номер 32-байтного блоку. Кожна ітерація генерує 32 байти ключового потоку; генерація продовжується до накопичення кількості байтів $\geq \text{len}(\text{compressed})$. Операція: $\text{result}[i] = \text{compressed}[i] \oplus \text{stream}[i]$

Готовий payload утворюється так: до результату XOR-скремблювання додається 32-бітний префікс довжини (довжина whitened-буфера у байтах), після чого кожен байт конвертується у вісім бітів у порядку MSB-first. Підсумкова структура бітового потоку має вигляд `[32 біти: довжина][N×8 біт: дані]`.

Декодування виконується у зворотному порядку: зчитати біти → відновити байти → XOR з тим самим PRNG (K_0 ідентичний) → `zlib.inflateSync()` → `Buffer.toString('utf-8')`.

Алгоритм LSB кодування — базовий алгоритм просторової стеганографії, що замінює молодший значущий біт кожного колірному каналу пікселя на відповідний біт payload. Блок-схему алгоритму LSB-кодування наведено (рис. 2.7).

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				40
Змн.	Арк.	№ докум.	Підпис	Дата		

Блок-схема алгоритму LSB кодування

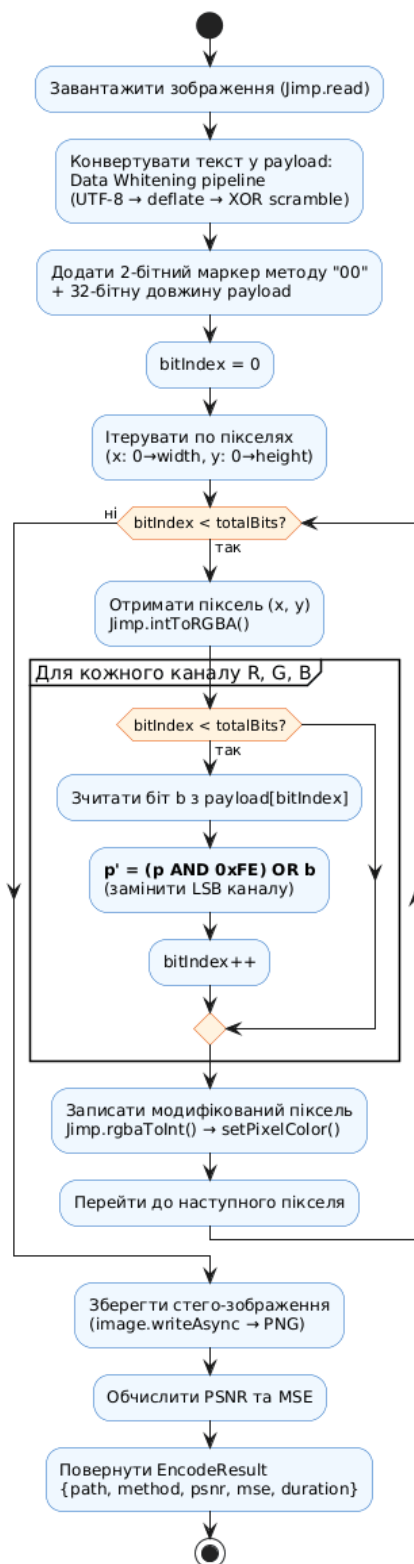


Рис. 2.7 – Блок-схема алгоритму LSB кодування

Після проходження payload через Data Whitening pipeline до бітового потоку додається 2-бітний маркер методу "00", який записується у канали R та G

		Пилипчук С.А.		
		Концидайло А.М.		
Змн.	Арк.	№ докум.	Підпис	Дата

ІПЗ.КР.Б-121-26-ІЗ

Арк.

41

першого пікселя (0,0). Ітерація виконується по пікселях у порядку рядок-за-рядком (x: 0→width, y: 0→height); для кожного пікселя послідовно обробляються три канали R, G, B.

Вбудовування одного біта b у колірний канал p виконується за допомогою двох побітових операцій:

$$p' = (p \& 0xFE) | b$$

де $0xFE = 11111110_2$ — маска обнулення LSB. Операція AND $0xFE$ обнуляє молодший біт каналу, операція OR b встановлює його в значення поточного біта payload. Таким чином, значення каналу змінюється максимум на ± 1 , що є невидимим для людського ока.

Максимальна ємність контейнера обчислюється за формулою:

$$C_{\text{LSB}} = \left\lfloor \frac{W \times H \times 3}{8} \right\rfloor \text{ байтів}$$

де W — ширина зображення, H — висота, коефіцієнт 3 — кількість каналів (R, G, B), знаменник 8 — кількість бітів у байті. Зображення 1000×1000 пікселів теоретично містить до 375 000 байтів (~ 366 КБ) payload. Фактична ємність дещо менша через 2-бітний маркер та 32-бітний префікс довжини.

Недолік методу LSB — пряма заміна LSB порушує природний розподіл пар значень $\{2k, 2k+1\}$ у гістограмі каналу, що робить зображення вразливим до χ^2 -стегааналізу.

Алгоритм LSBM кодування усуває головну вразливість LSB, замінюючи детерміновану операцію присвоєння LSB на випадкову зміну значення каналу на ± 1 . Блок-схему методу наведено (рис. 2.8).

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

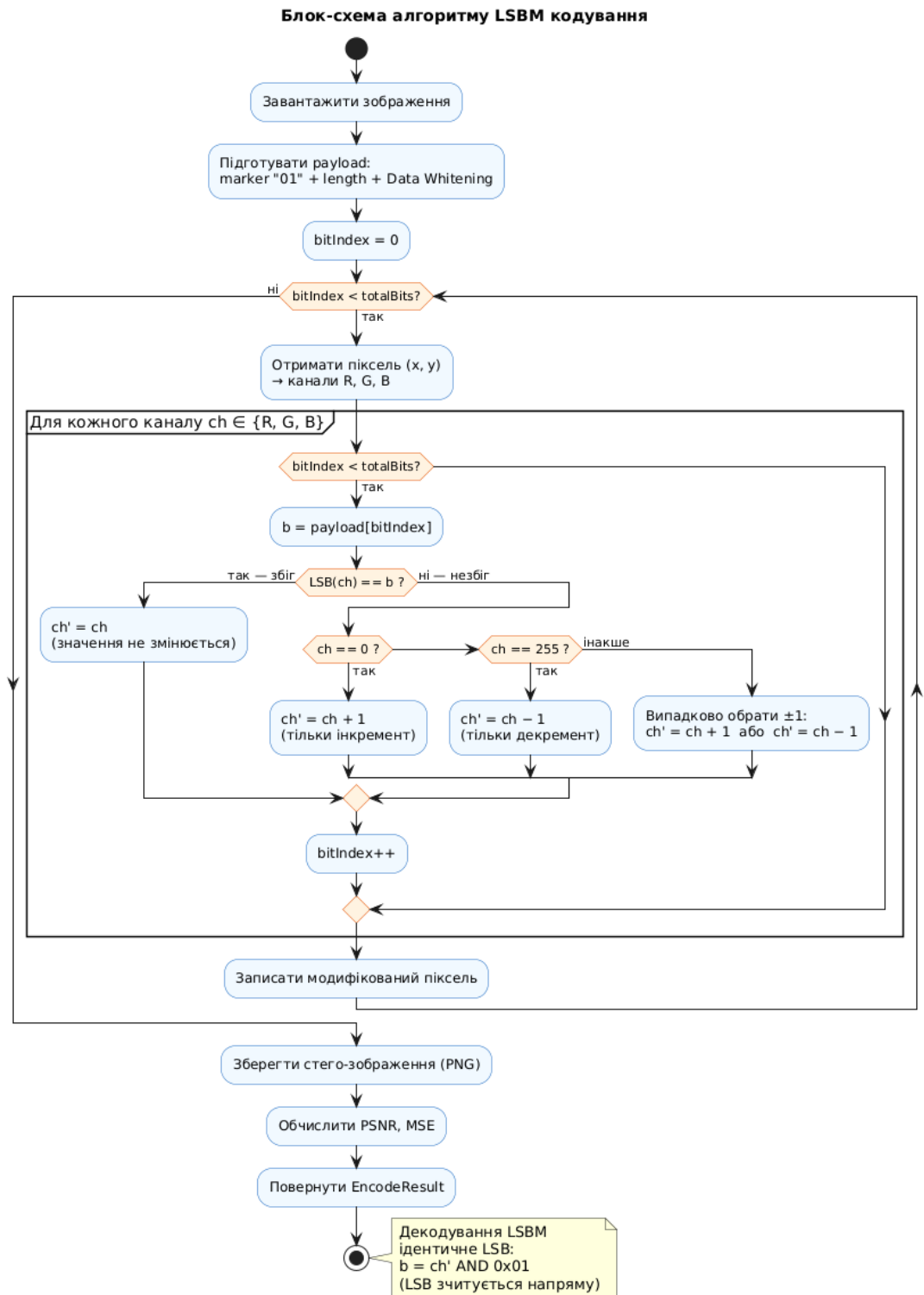


Рис. 2.8 – Блок-схема алгоритму LSBM кодування

Payload маркується двобітовим ідентифікатором методу «01», після чого під час обробки кожного каналу ch для запису біта b виконується умовне розгалуження. Якщо молодший біт каналу вже дорівнює потрібному ($LSB(ch) == b$), значення каналу не змінюється. Якщо ж біти не збігаються і канал перебуває на

межі діапазону, виконується лише однобічна корекція: при $ch == 0$ значення збільшується на одиницю ($ch' = ch + 1$), а при $ch == 255$ — зменшується на одиницю ($ch' = ch - 1$). В усіх інших випадках значення змінюється на ± 1 з випадковим вибором напрямку ($ch' = ch \pm 1$).

Обробка граничних значень 0 та 255 є обов'язковою для запобігання виходу за діапазон $[0, 255]$. Випадковість вибору ± 1 у загальному випадку забезпечує, що результуюча гістограма не демонструє характерного «вирівнювання» пар $\{2k, 2k+1\}$, на якому базується χ^2 -тест.

Декодування методу LSBM повністю ідентичне декодуванню звичайного LSB і виконується тією самою операцією $b = ch' \text{ AND } 0x01$, тобто шляхом зчитування молодшого значущого біта колірному каналу. Це можливо завдяки тому, що операція зміни значення каналу на ± 1 ніколи не порушує відповідності молодшого біта вбудованому. Якщо потрібний біт уже збігається з молодшим бітом каналу, значення пікселя залишається незмінним; якщо ж не збігається, то додавання або віднімання одиниці змінює парність значення каналу, унаслідок чого його молодший біт стає рівним саме цільовому біту b . Отже, незалежно від обраного напрямку модифікації ($+1$ чи -1), молодший біт модифікованого каналу завжди містить корисну інформацію. Завдяки цьому декодеру не потрібен окремий алгоритм для методу LSBM — він застосовує ту саму процедуру побітового зчитування, що й для LSB, а сам метод вбудовування визначає автоматично за двобітовим маркером, записаним на початку контейнера.

Порівнюючи LSB і LSBM, варто зазначити, що обидва методи вносять однакову максимальну зміну ± 1 на канал, тому метрики PSNR та MSE є практично ідентичними. Ключова відмінність полягає у стійкості LSBM до χ^2 -атаки при рівні значущості $\alpha = 0,05$, що підтверджується результатами стегоаналізу у розділі 3.

Адаптивний алгоритм кодування (рис. 2.9) розширює LSB просторовим відбором пікселів — вбудовування концентрується у текстурованих ділянках зображення (краях, границях об'єктів), де модифікація пікселів є найменш помітною.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

Блок-схема адаптивного методу кодування (Edge-Aware)

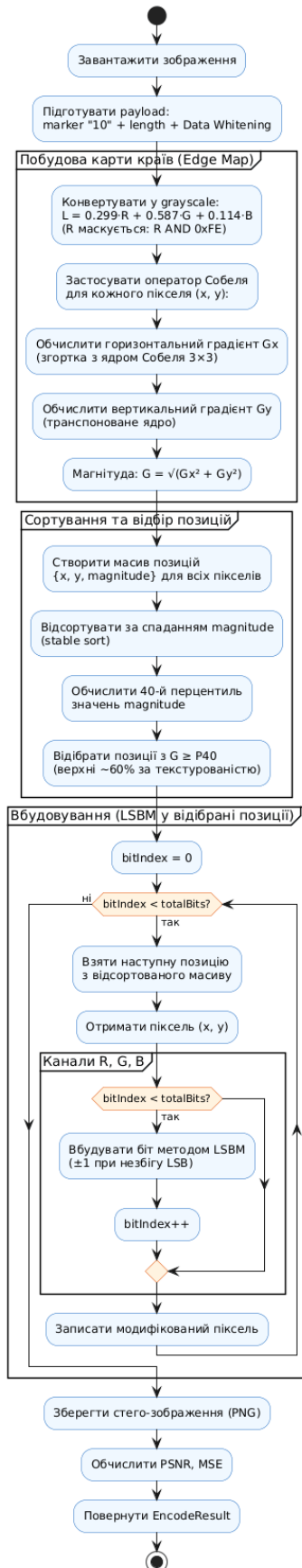


Рис. 2.9 – Блок-схема адаптивного методу кодування (Edge-Aware)

Payload маркується ідентифікатором "10". Алгоритм складається з трьох фаз.

Перша фаза — Побудова карти країв (Edge Map). Зображення конвертується у відтінки сірого за формулою:

$$L = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

де коефіцієнти відповідають відносній чутливості людського зору до трьох каналів. Зауважимо, що при конвертації маскуються LSB усіх трьох каналів (R, G, B) через операцію $\& 0xFE$, що забезпечує ідентичність карти країв на оригінальному та закодованому зображеннях (оскільки вбудовування змінює LSB усіх каналів). Для кожного пікселя (x, y) обчислюються горизонтальний G_x та вертикальний G_y градієнти через операцію згортки з ядром оператора Собеля 3×3 . Результуюча магнітуда градієнта:

$$G(x, y) = \sqrt{G_x^2 + G_y^2}$$

характеризує «текстурованість» пікселя — чим вищий G , тим різкіша зміна яскравості навколо пікселя.

Друга фаза — Сорткування та відбір позицій. Формується масив усіх пікселів $\{x, y, \text{magnitude}\}$, відсортований за спаданням magnitude . Обчислюється 40-й перцентиль P_{40} значень магнітуди. До вбудовування допускаються лише пікселі з $G \geq P_{40}$, тобто верхні $\sim 60\%$ за рівнем текстурованості. Такий поріг відбору є компромісом між ємністю контейнера (чим більше пікселів допущено, тим більше payload) та непомітністю (вбудовування у гладкі ділянки є візуально помітним).

Третя фаза — Вбудовування. У відібрані позиції застосовується пряма заміна LSB: $\text{pixel}[\text{ch}] = (\text{pixel}[\text{ch}] \& 0xFE) \mid \text{bit}$. Захист від стегоаналізу забезпечується просторовою стратегією: вбудовування зосереджено виключно у ділянках з високою текстурованістю (краї, границі об'єктів), де природна варіативність значень пікселів маскує модифікацію LSB.

Алгоритм AES-256-CBC [25] реалізує двосторонній криптозахист payload з використанням стандарту AES-256 у режимі CBC через вбудований модуль

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

Node.js crypto без сторонніх залежностей. Блок-схему процесу шифрування та дешифрування наведено (рис. 2.10).

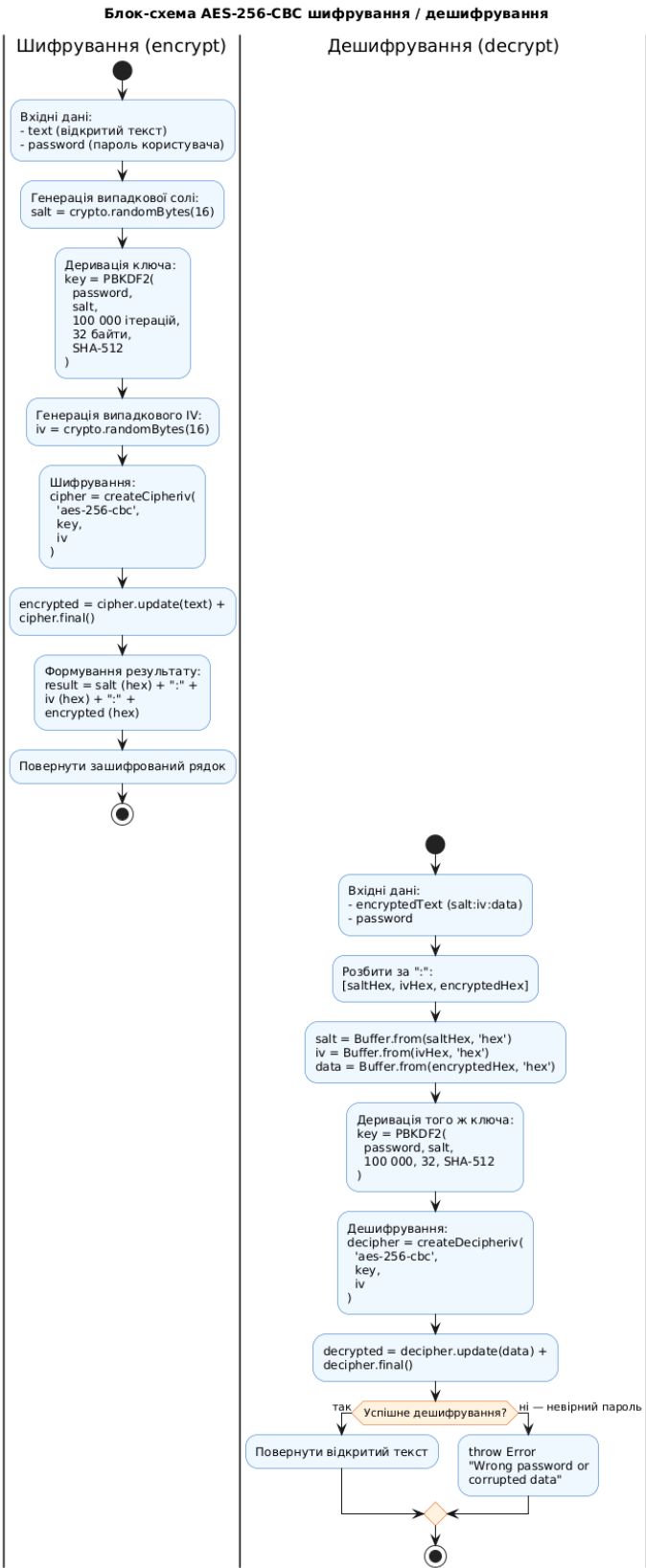


Рис. 2.10 – Блок-схема AES-256-CBC шифрування/дешифрування

Процедура шифрування encryptMessage(text, password) виконує такі кроки.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

Спочатку генерується випадкова сіль `salt = crypto.randomBytes(16)` завдовжки 128 біт. Далі з пароля та солі деривується ключ за алгоритмом PBKDF2 [26]:

`key = PBKDF2(password, salt, 100000, 32, SHA-512)`

де 100 000 — кількість ітерацій (захист від brute-force атак), 32 — довжина ключа у байтах (256 біт для AES-256)

Потім генерується випадковий вектор ініціалізації `iv = crypto.randomBytes(16)` завдовжки 128 біт, унікальний для кожного шифрування. Після цього виконується власне шифрування: `cipher = createCipheriv('aes-256-cbc', key, iv)`, `encrypted = cipher.update(text) + cipher.final()`. Нарешті формується результат у вигляді рядка `salt(hex) + ':' + iv(hex) + ':' + encrypted(hex)`, що містить усі параметри, потрібні для подальшого дешифрування.

Процедура дешифрування `decryptMessage(cipherText, password)` розбиває рядок за роздільником ":", відновлює `salt`, `iv`, `data` через `Buffer.from(hex, 'hex')`, дериває той самий ключ через PBKDF2 з тими самими параметрами та виконує `createDecipheriv('aes-256-cbc', key, iv)`. При неправильному паролі `decipher.final()` генерує виняток "Wrong password or corrupted data".

Криптографічна стійкість: 256-бітний ключ AES-256-CBC робить атаку повного перебору обчислювально неможливою, оскільки простір можливих ключів становить 2^{256} варіантів. Унікальні `salt` та `IV` при кожному шифруванні гарантують, що однаковий текст з однаковим паролем кожного разу генерує різний шифртекст.

Алгоритм декодування є єдиним для всіх трьох методів (LSB, LSBM, Adaptive), оскільки маркер методу зчитується з перших двох бітів стего-зображення.

Процедура `decode(imagePath, password?)` виконується у такому порядку.

Спочатку зображення завантажується через `Jimp.read(imagePath)`. Потім зчитується маркер методу — два молодші біти каналів R та G пікселя (0,0): значення «00» відповідає методу LSB, «01» — LSBM, «10» — Adaptive. Далі зчитується довжина `payload` (32 біти LSB з наступних 11 пікселів), що задає кількість байтів `whitened`-буфера. Вибір позицій для зчитування залежить від методу: для

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

LSB та LSBM біти зчитуються послідовно по всіх пікселях, а для Adaptive відновлюється Edge Map (ті самі обчислення оператора Собеля, той самий 40-й перцентиль і той самий відсортований масив позицій). Зчитування виконується як $b = ch \text{ AND } 0x01$ (молодший біт поточного каналу) для заданої кількості бітів, після чого застосовується зворотний Data Whitening: відновлені байти XOR-уються з тим самим PRNG, далі виконується `zlib.inflateSync()` і `Buffer.toString('utf-8')`. За наявності пароля додатково викликається `decryptMessage(text, password)` з модуля `encryption.ts`.

χ^2 -аналіз (рис. 2.11) — це статистичний метод виявлення слідів LSB-стеганографії, що ґрунтується на дослідженні розподілу попарних значень яскравості (Pairs of Values, PoV) у гістограмі кольорового каналу. Метод запропоновано А. Вестфельдом і А. Пфітцманом, і він є одним з класичних інструментів пасивного стегоаналізу. В його основу покладено таку закономірність: у природному (немодифікованому) зображенні частоти сусідніх значень яскравості, що утворюють пари $\{2k, 2k+1\}$, як правило, помітно відрізняються, тоді як послідовне LSB-вбудовування «переключає» пікселі між парним і непарним значеннями кожної пари й поступово вирівнює ці частоти. Чим більшу частину контейнера заповнено прихованими даними, тим сильнішим є таке вирівнювання. Для кількісної оцінки ступеня вирівнювання обчислюється статистика χ^2 Пірсона, на основі якої визначається ймовірність (p-value) того, що зображення містить вбудоване повідомлення: значення p-value, близьке до одиниці, свідчить про високу ймовірність стеганографії, а близьке до нуля — про «чисте» зображення. У розробленому модулі χ^2 -аналіз виконується як у глобальному режимі (по всьому зображенню), так і за допомогою ковзного вікна, що дозволяє не лише виявити сам факт вбудовування, а й локалізувати ділянки зображення з найвищою концентрацією прихованих даних.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

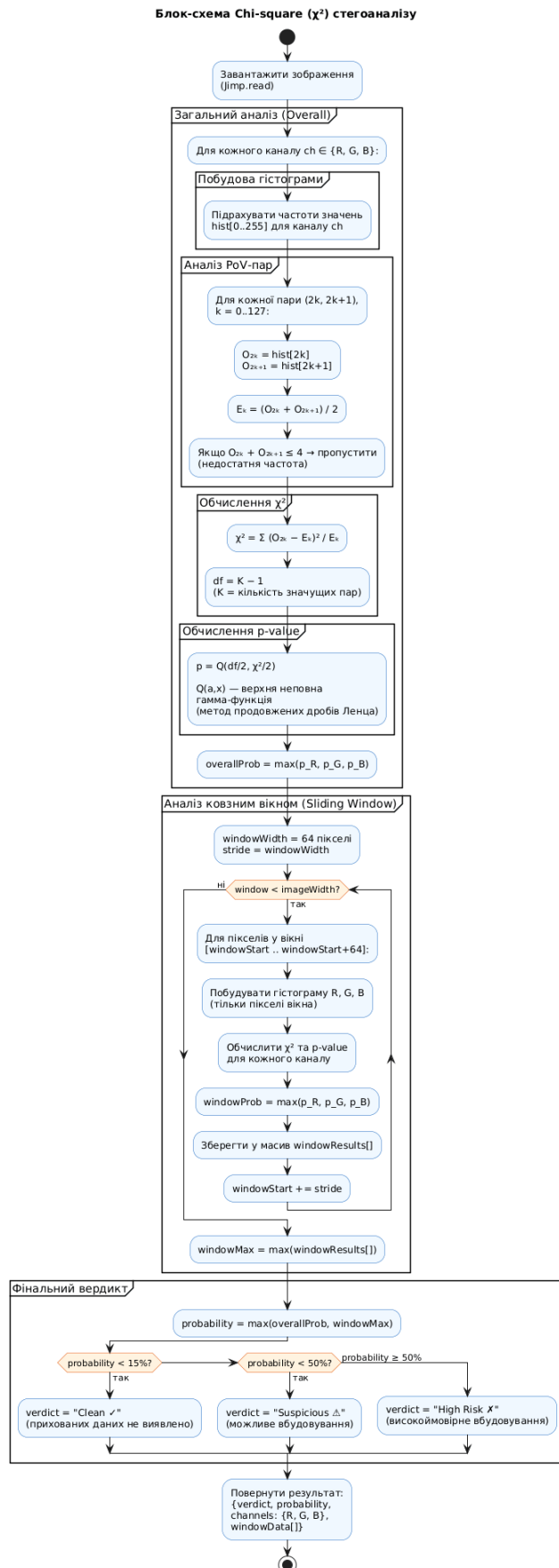


Рис. 2.11 – Блок-схема χ^2 -стегоаналізу

Теоретичну основу методу становить такий ефект: при вбудовуванні даних методом LSB частоти пар значень $\{2k, 2k+1\}$ у гістограмі каналу вирівнюються, наближаючись до однакових значень, оскільки вбудовування «переключає» пікселі між парними та непарними значеннями. Чим більша область вбудовування, тим сильніше вирівнювання і тим вища статистика χ^2 .

Глобальний аналіз (Overall). Для кожного каналу $ch \in \{R, G, B\}$:

1. Побудова гістограми $hist[0..255]$ — підрахунок частот усіх 256 значень яскравості
2. Для кожної пари $(2k, 2k+1)$, де $k = 0..127$:
 - Спостережувані значення: $O_{2k} = hist[2k], O_{2k+1} = hist[2k + 1]$
 - Очікуване значення: $E_k = \frac{O_{2k} + O_{2k+1}}{2}$
 - Якщо $O_{2k} + O_{2k+1} = 0$ — пара пропускається (нульова частота)
3. Обчислення χ^2 -статистики:

$$\chi^2 = \sum_{k=0}^{K-1} \frac{[(O_{2k} - E_k)^2 + (O_{2k+1} - E_k)^2]}{E_k}$$

де K — кількість значущих пар, $df = K$ — ступені свободи

4. Обчислення p-value через верхню неповну гамма-функцію:

$$p = Q\left(\frac{df}{2}, \frac{\chi^2}{2}\right)$$

де $Q(a, x)$ обчислюється методом продовжених дробів Ленца

Для виявлення часткового вбудовування застосовано аналіз ковзним вікном (Sliding Window): зображення сканується вікном розміром $windowSize = \max(1000, \lfloor totalPixels/20 \rfloor)$ пікселів з кроком $step = \lfloor windowSize/2 \rfloor$ (50% перекриття). Для кожного вікна окремо обчислюються χ^2 та p-value по трьох каналах, а значення $windowProb = \max(pR, pG, pB)$ зберігається у масив $curve[]$. Такий підхід дозволяє локалізувати зону вбудовування — на графіку Chart.js у клієнтській частині пікові значення $windowProb$ відповідають областям зображення з найвищою концентрацією вбудованих даних.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				51
Змн.	Арк.	№ докум.	Підпис	Дата		

Клієнт отримує структуру {verdict, probability, channels: {r, g, b}, windowData[]} та відображає Verdict Card, Probability Bar, каналні гістограми та графік ковзного вікна.

Розроблені та програмно реалізовані алгоритми формують надійне математичне та логічне ядро системи. Поєднання криптографічного перетворення (AES-256), ентропійного відбілювання (Data Whitening) та адаптивного стеганографічного вбудовування дозволило створити комплексний механізм прихованої передачі даних, який є стійким до сучасних методів статистичного аналізу. Водночас наявність власного модуля χ^2 -тестування забезпечує можливість об'єктивної перевірки сформованих контейнерів на стороні сервера. Наступним кроком є інтеграція цих ізольованих алгоритмів у єдиний веб-орієнтований застосунок, фізична організація програмного коду та налаштування взаємодії між клієнтом і сервером, що детально розглядається у наступному підрозділі.

2.4 Реалізація програмного комплексу

Програмний комплекс реалізовано у вигляді монорепозиторію stego-app/, що складається з двох незалежних підпроектів — серверного backend/, побудованого на платформі Node.js із фреймворком Express та мовою TypeScript, і клієнтського frontend/ на React та TypeScript. Обидва підпроекти мають власні файли package.json і tsconfig.json та запускаються незалежно.

Фізичну організацію коду розподілено за директоріями обох підпроектів (рис. 2.12). Серверну частину структуровано у п'ять основних директорій, кожна з яких відповідає за окремий рівень функціональності системи.

У директорії config/ зосереджено файли конфігурації системи: модуль database.ts відповідає за підключення до бази даних MongoDB за допомогою Mongoose ODM, а модуль logger.ts налаштовує систему журналювання Winston з одночасним виведенням повідомлень у консоль та у файли error.log і combined.log. Директорія engine/ містить алгоритмічне ядро застосунку — файл steganography.ts обсягом близько 940 рядків, у якому реалізовано методи вбудовування LSB, LSBM та Adaptive, алгоритм χ^2 -стегоаналізу й механізм Data

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				52
Змн.	Арк.	№ докум.	Підпис	Дата		

Whitening, а також файл encryption.ts, що забезпечує криптографічний захист повідомлень за алгоритмом AES-256-CBC. У директорії middleware/ розміщено проміжні обробники запитів: модуль auth.ts здійснює перевірку JWT-токенів, rateLimiter.ts обмежує частоту звернень (до 200 запитів за 15 хвилин у загальному випадку, 20 запитів за 15 хвилин для операцій автентифікації та 10 запитів за хвилину для кодування), upload.ts керує завантаженням файлів засобами Multer, а errorHandler.ts централізовано опрацьовує помилки. У директорії models/ зберігаються шість Mongoose-схем даних (User, Message, History, SharedLink, Analytics та SystemLog), а в директорії routes/ — дванадцять модулів маршрутизації; точкою входу до серверної частини є файл server.ts, що запускає застосунок на порту 5000.

Клієнтську частину організовано за аналогічним принципом. У директорії components/ зосереджено три багаторазові компоненти інтерфейсу — Layout, PrivateRoute та AdminRoute; директорія pages/ містить чотирнадцять сторінок застосунку, директорія services/ — налаштований екземпляр HTTP-клієнта Axios для взаємодії із сервером, а директорія store/ — глобальне сховище стану Redux разом зі зрізом authSlice, що відповідає за стан автентифікації користувача.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

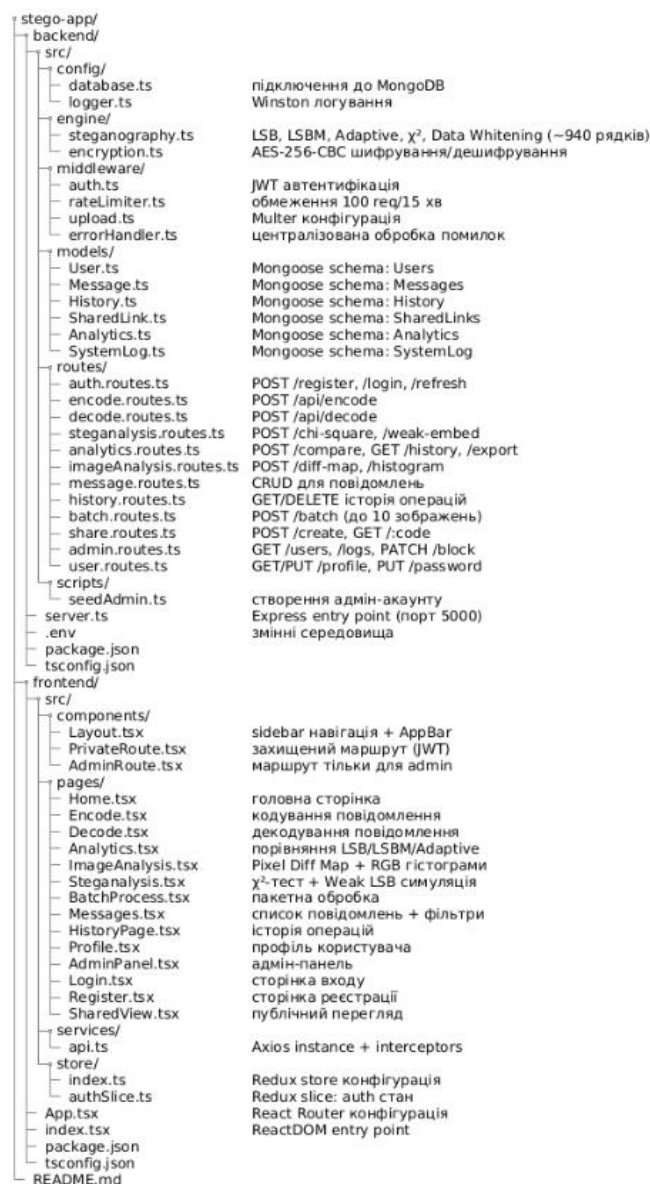


Рис. 2.12 – Структура каталогів проекту

Ініціалізацію сервера реалізовано у файлі `server.ts`: Express-застосунок підключає `middleware` у фіксованому порядку — `Helmet` → `CORS` → `Rate Limiter` → `express.json()` → `route`-модулі → `Error Handler`. Порядок є принциповим: `Helmet` та `Rate Limiter` повинні стояти до будь-яких маршрутів, а `Error Handler` — останнім, щоб перехоплювати всі необроблені винятки.

Перевірку автентифікації реалізовано у проміжному обробнику `auth` (файл `auth.ts`). Кожен захищений маршрут проходить через цей обробник, який витягує `Bearer`-токен із заголовка `Authorization`, верифікує його через `jwt.verify()`, завантажує повний об'єкт користувача з бази даних через `User.findById()` та записує його у `req.user` (тип `IUser`), а рядковий ідентифікатор — у `req.userId`. При

невалідному токени повертається 401 Unauthorized, при заблокованому акаунті — 403 Forbidden. Перевірку ролі admin реалізовано окремим обробником adminOnly, який додатково перевіряє умову req.user.role === 'admin' і при невідповідності повертає 403 Forbidden.

Оброблювач маршруту кодування реалізовано у файлі encode.routes.ts: він приймає дані форми (FormData) з полями image (файл), text, method, encrypt і password та застосовує конвеєр auth (JWT) → encodeLimiter (до 10 запитів за хвилину) → Multer → логіка encode.

```
router.post('/', auth, encodeLimiter, upload.single('image'), async (req, res) => {
  const { message, method, encrypt: shouldEncrypt, password } = req.body;
  let finalMessage = message;
  if (shouldEncrypt === 'true') {
    finalMessage = encryptMessage(message, password);
  }
  const result = await encode(req.file.path, finalMessage, method, encodedDir,
outputFilename);

  await Message.create({ userId: req.user!._id, title, method, ...});
  await History.create({ userId: req.user!._id, action: 'encode',
    method, duration: Date.now() - startTime, success: true });
  res.json({ psnr: result.psnr, mse: result.mse,
    encodedImageUrl: `/uploads/encoded/${outputFilename}` });
});
```

Централізовану обробку помилок реалізовано у файлі errorHandler.ts. Чотирипараметровий middleware (err, req, res, next) перехоплює всі необроблені помилки, логує їх через Winston logger.error() (у файл та консоль), класифікує за типом (ValidationError → 400, JsonWebTokenError → 401, MulterError → 400) та повертає уніфіковану відповідь {error: message}. Записи у SystemLog створюються безпосередньо в оброблювачах маршрутів при успішних операціях, що виключає дублювання конструкцій try/catch у кожному з них.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

Пакетну обробку реалізовано у файлі `batch.routes.ts`. Завантажувач Multer налаштовано на приймання до 10 файлів одночасно (`upload.array('images', 10)`). Оброблювач послідовно опрацьовує зображення у циклі `for`: для кожного файлу окремо викликається `encode()`, результат зберігається через `Message.create()`, а помилки агрегуються окремо. Завдяки цьому невдала обробка одного зображення не зупиняє решту (ізоляція помилок).

Управління станом покладено на Redux Toolkit, який використовується виключно для зберігання стану автентифікації. Файл `authSlice.ts` визначає стан `{user, token, refreshToken, loading, error}` та три синхронні reducer-дії: `logout` (очищення стану і `localStorage`), `clearError` (скидання помилки) та `updateUser` (часткове оновлення об'єкта `user`). Асинхронні операції (`register`, `login`, `loadUser`) реалізовано через `createAsyncThunk`.

Решта серверних даних управляється через React Query (кешування, інвалідація, повторні запити).

HTTP-клієнт (`services/api.ts`). Axios instance налаштований з `baseUrl`: `process.env.REACT_APP_API_URL`. Два interceptors: `request` — автоматично додає заголовок `Authorization: Bearer <token>` з `localStorage.getItem('accessToken')` до кожного запиту; `response` — при отриманні статусу 401 автоматично виконує `POST /auth/refresh` для отримання нового `accessToken` і повторює оригінальний запит, реалізуючи silent token refresh без участі користувача.

Захищену маршрутизацію реалізовано через компоненти-обгортки: `PrivateRoute.tsx` перевіряє наявність `token` у Redux store і, якщо токен відсутній, перенаправляє на `/login`, а `AdminRoute.tsx` додатково перевіряє умову `user.role === 'admin'` і при невідповідності перенаправляє на головну сторінку.

Найбільш складним з точки зору обчислювальної логіки є функція побудови Edge Map у `steganography.ts`.

Маскування LSB у всіх трьох каналах (R, G, B) через операцію `& 0xFE` при обчисленні яскравості `L` є обов'язковим: оскільки вбудовування модифікує LSB

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

усіх трьох каналів, без маскуванню побудована Edge Map при декодуванні відпрізнялася б від тієї, що використовувалася при кодуванні, що призвело б до неправильного відновлення позицій пікселів та втрати повідомлення

Висновки до другого розділу

У другому розділі виконано повний цикл проєктування та реалізації стега-нографічного комплексу. Розроблено клієнт-серверну архітектуру на стеку MERN з розподілом на клієнтський React-застосунок, що працює на порту 3000, та серверний REST API на Express на порту 5000; визначено три ролі користувачів і дванадцять модулів маршрутів з JWT-автентифікацією. Спроектовано схему бази даних MongoDB із шести колекцій зі складеними та TTL-індексами для оптимізації запитів і автоматичного видалення прострочених записів.

Реалізовано математичне ядро системи: алгоритми LSB, LSBM та Adaptive (оператор Собеля), конвеєр Data Whitening (DEFLATE + XOR HMAC-SHA256), криптографічний захист AES-256-CBC (PBKDF2, 100 000 ітерацій) та χ^2 -стего-аналіз у двох режимах — глобальному та ковзним вікном. Якість вбудовування оцінюється метриками PSNR і MSE з пороговим значенням $\text{PSNR} \geq 40$ дБ. Оцінка ефективності розроблених алгоритмів та тестування системи розглядаються у наступному розділі.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Концидайло А.М.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1 Структура інтерфейсу та функціональні можливості

У цьому підрозділі описано структуру інтерфейсу програмного комплексу StegoVault, перелік реалізованих функціональних модулів та принципи розмежування доступу між ролями користувача і адміністратора.

Інтерфейс системи StegoVault реалізовано як односторінковий застосунок (SPA) на базі React 18 з бібліотекою компонентів Material UI v5. Маршрутизація здійснюється засобами React Router v6 з декларативним описом маршрутів. Загальна структура інтерфейсу побудована за принципом Layout-Outlet: спільний каркас (AppBar, бокова панель навігації, область вмісту) зберігається при переході між сторінками, а оновлюється лише центральна зона контенту, що забезпечує плавний користувацький досвід без повного перезавантаження.

Верхня панель містить логотип платформи, назву «StegoVault» та меню користувача з аватаром, іменем та опціями «Профіль», «Вихід», а для адміністратора — додатковий пункт «Адмін-панель». Панель оформлена у кольорах корпоративної палітри платформи, що формує впізнавану візуальну ідентичність застосунку.

Бокова панель навігації відображає перелік усіх доступних розділів із відповідними піктограмами Material Icons: Головна, Кодування, Декодування, Пакетна обробка, Аналіз зображень, Аналітика, Стегоаналіз (χ^2), Повідомлення, Історія, Профіль. Пункт меню, що відповідає поточній сторінці, підсвічується кольором primary. На мобільних пристроях бокова панель автоматично згортається та відкривається за натисканням відповідної іконки у лівому кутку верхньої панелі, що забезпечує адаптивність інтерфейсу.

Розмежування доступу на рівні інтерфейсу реалізовано через компоненти-обгортки PrivateRoute та AdminRoute. PrivateRoute перевіряє наявність токена автентифікації у Redux-сховищі та перенаправляє неавтентифікованого користувача на сторінку входу. AdminRoute додатково перевіряє роль користувача та перенаправляє на головну сторінку у разі відсутності адміністративних привілеїв.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

Інтерфейс системи побудовано за принципом єдиного каркаса (Layout) зі сталими верхньою панеллю й бічним меню навігації та змінною областю вмісту (рис. 3.1).

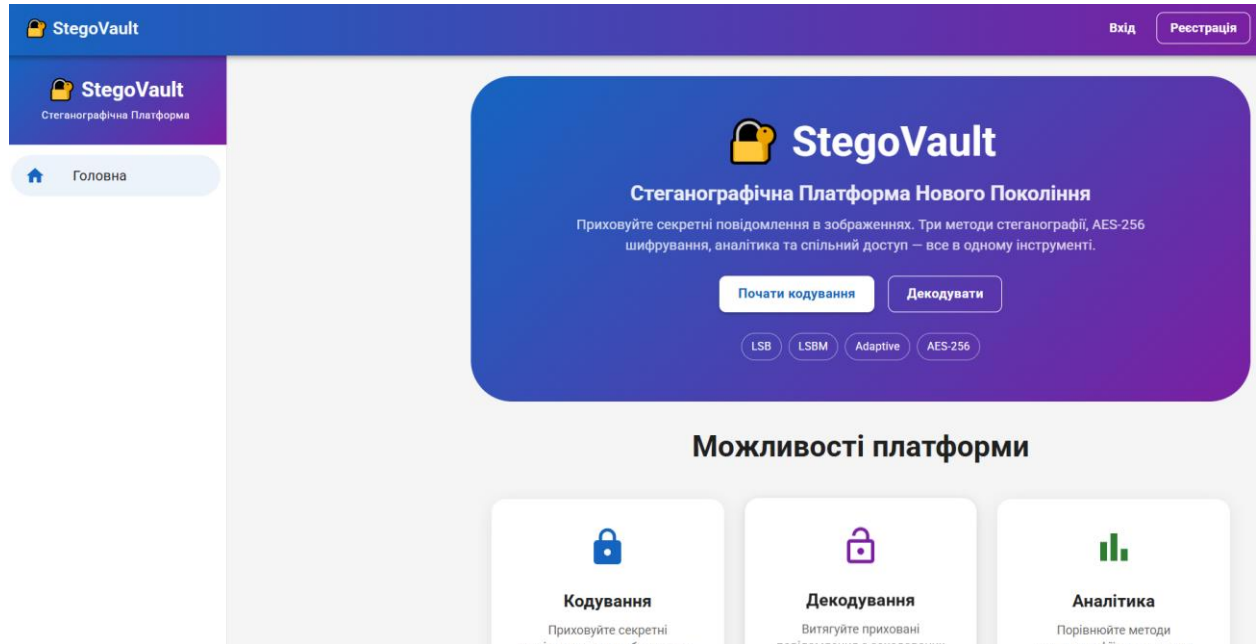


Рис. 3.1 – Головна сторінка системи StegoVault

Головна сторінка (Home) виконує роль landing page та містить три функціональні блоки. Hero-секція відображає назву платформи, короткий опис та дві кнопки швидкого доступу — «Почати кодування» та «Декодувати», а також перелік підтримуваних методів (LSB, LSBM, Adaptive, AES-256). Блок «Можливості платформи» містить шість карток з піктограмами, що описують основні функції: Кодування, Декодування, Аналітика, AES-256 Шифрування, Метрики якості, Спільний доступ. Блок «Як це працює?» візуалізує процес у три кроки: завантаження зображення, введення повідомлення, отримання результату.

За автентифікацію відповідають сторінки входу (Login) та реєстрації (Register), які реалізовано як окремі маршрути поза основним Layout, що унеможливорює доступ до навігаційної панелі до проходження автентифікації. Форма входу містить поля email та пароль із валідацією обов’язкових полів, а форма реєстрації додатково містить поле імені та підтвердження паролю з інтерактивною перевіркою збігу в реальному часі. Обидві форми відображають повідомлення про помилки через компонент Alert.

Центральним функціональним модулем системи є сторінка кодування повідомлення Encode (рис. 3.2). Зона завантаження підтримує drag-and-drop з обмеженням форматів (PNG, BMP) та розміру файлу. Після завантаження зображення автоматично виконується запит до ендпоінту `/api/encode/analyze`, який повертає місткість зображення для кожного методу стеганографії (LSB, LSBM, Adaptive) у символах, а результати відображаються у вигляді чіпів під прев'ю зображення.

Форма кодування містить: поле введення секретного повідомлення з лічильником символів; перемикач методу стеганографії (LSB, LSBM, Adaptive); параметр шифрування AES-256 із полем пароллю; назву та категорію повідомлення (personal, work, secret, other); параметри самознищення — максимальну кількість переглядів та термін дії посилання у днях.

Після успішного кодування відображається панель результатів із метриками: PSNR (дБ), MSE, кількість використаних бітів, ємність зображення та час виконання. Користувачу доступні кнопки «Завантажити» (збереження закодованого зображення), «Зберегти» (додавання до бібліотеки повідомлень) та «Поділитися» (генерація публічного посилання).

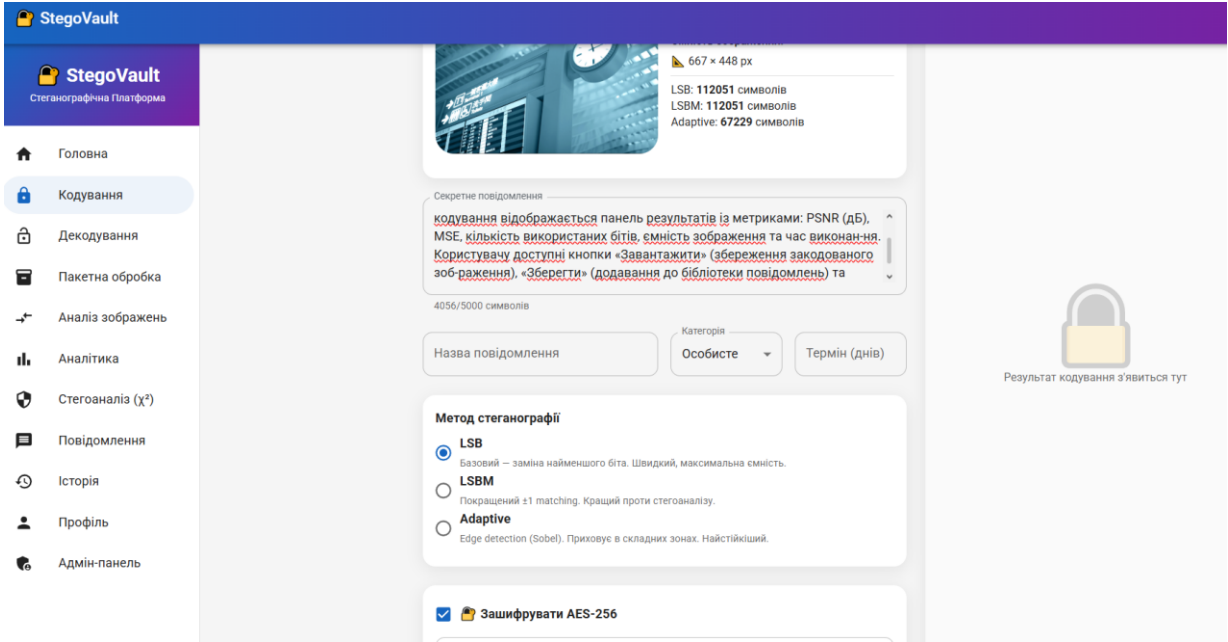


Рис. 3.2 – Сторінка кодування повідомлення

Зворотний процес реалізує сторінка декодування Decode (рис. 3.3). Після завантаження стего-зображення система автоматично надсилає його на ендпоінт

/api/decode/detect для визначення методу вбудовування, а виявлений метод відображається як відповідний індикатор. Форма містить перемикач зашифрованого повідомлення та поле паролю, а результат декодування відображається у форматному блоці з можливістю копіювання у буфер обміну та експорту у текстовий файл.

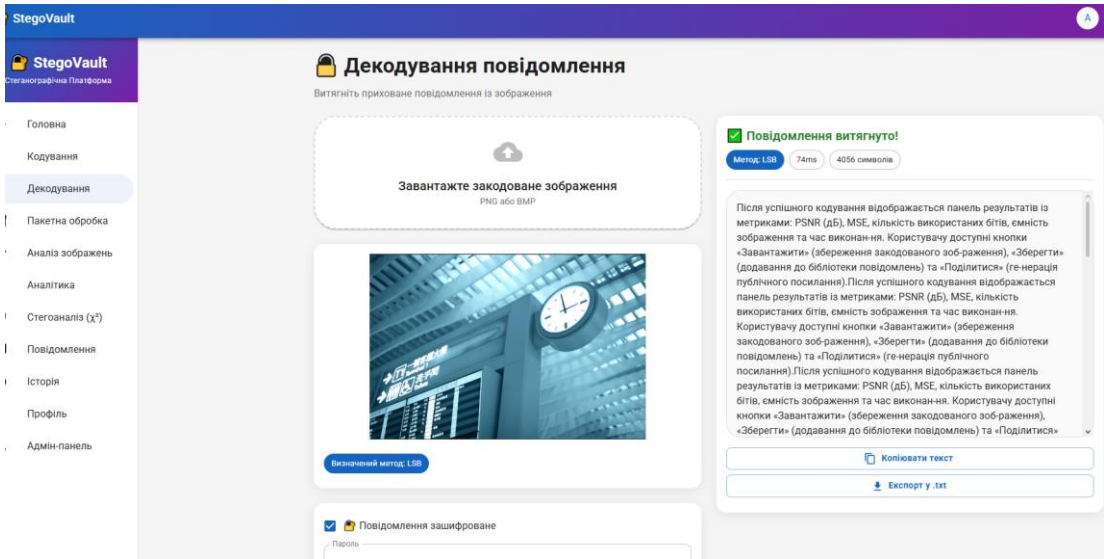


Рис. 3.3 – Сторінка декодування повідомлення

Сторінка порівняння методів Analytics (рис. 3.4) забезпечує автоматичне тестування всіх трьох методів стеганографії на одному зображенні з однаковим тестовим повідомленням. Після завантаження зображення система послідовно виконує кодування кожним методом і повертає порівняльну таблицю з метриками PSNR, часу виконання та ємності. Результати візуалізуються у вигляді стовпчикових діаграм і супроводжуються текстовою рекомендацією щодо оптимального методу.

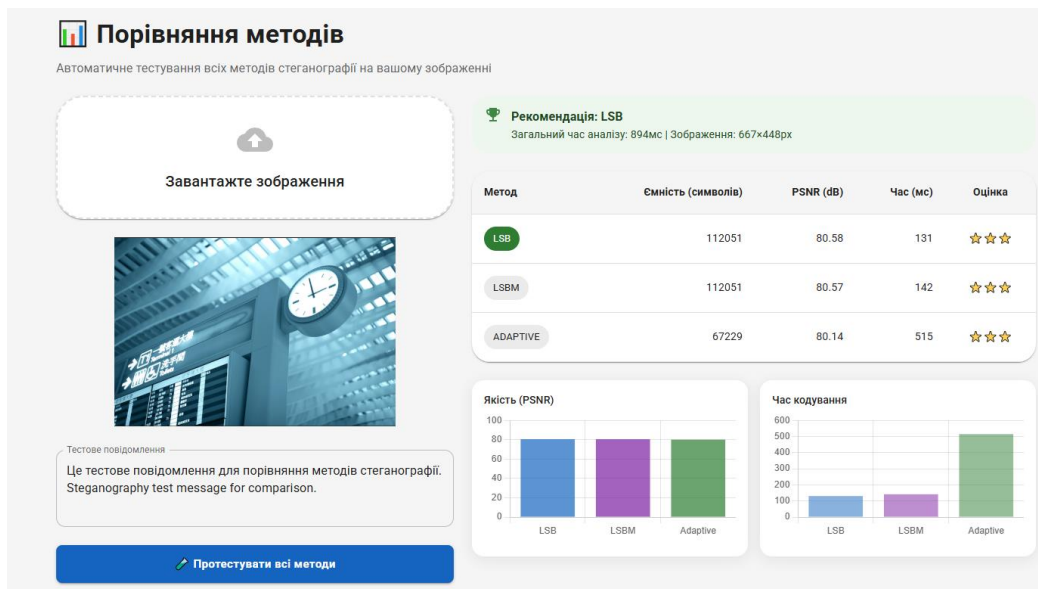


Рис. 3.4 – Сторінка порівняння методів стеганографії

Сторінка стегоаналізу Steganalysis (рис. 3.5) реалізує два режими роботи. Перший — χ^2 -аналіз завантаженого зображення: система надсилає зображення на ендпоінт `/api/steganalysis/chi-square` та отримує результати, які візуалізуються через кольоровий індикатор рівня ризику (clean / low / medium / high), текстовий вердикт, таблицю χ^2 -статистики по кожному каналу (R, G, B) із значеннями χ^2 , p-value та кількістю проаналізованих пар, а також лінійний графік ковзного вікна — криву ймовірності вбудовування вздовж зображення, що дозволяє локалізувати область вбудовування.

Другий режим — «Слабке вбудовування» (Weak Embed): користувач завантажує чисте зображення, система виконує найвне LSB-вбудовування без шифрування та Data Whitening і автоматично завантажує результат. Цей режим призначений для демонстрації вразливості незахищеного LSB до χ^2 -атаки.

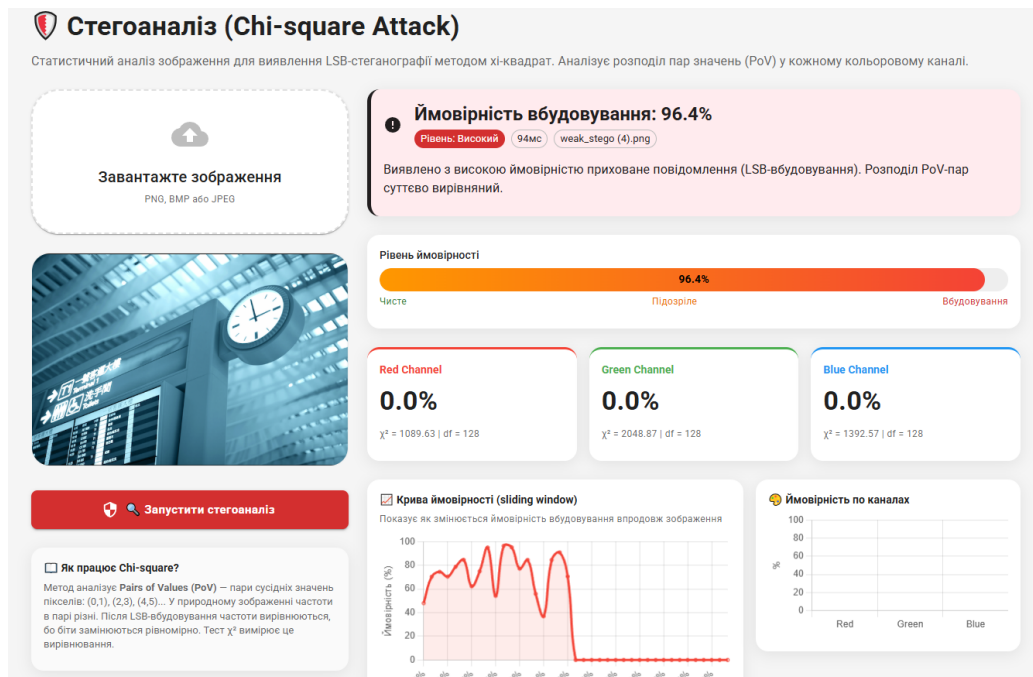


Рис. 3.5 – Сторінка стеогоаналізу з результатами χ^2 -тесту

Сторінка аналізу зображень ImageAnalysis організована у дві вкладки. Перша вкладка «Pixel Diff Map» приймає два зображення — оригінал та закодоване — і генерує карту відмінностей (ендпоінт `/api/image-analysis/diff`) з регульованим коефіцієнтом підсилення; на виході відображаються три зображення (оригінал, закодоване та Diff Map) і метрики PSNR та MSE. Друга вкладка «Порівняння гістограм» відображає гістограми розподілу яскравості кожного каналу (R, G, B) у вигляді стовпчикових діаграм, що дозволяє візуально оцінити зміни у статистичному розподілі значень пікселів після вбудовування. Результати такого аналізу зображень показано (рис. 3.6).

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				63
Змн.	Арк.	№ докум.	Підпис	Дата		

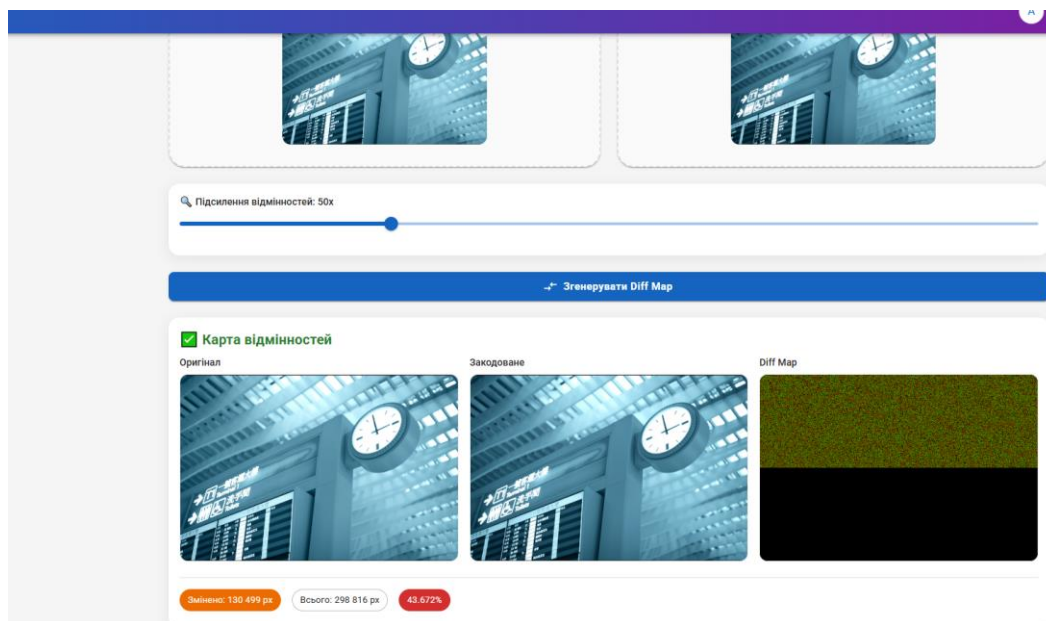


Рис. 3.6 – Аналіз зображень: Pixel Diff Map та гістограми каналів

Сторінка пакетної обробки BatchProcess дозволяє одночасно закодувати одне повідомлення у декілька зображень (до 10 файлів). Список завантажених файлів відображається із зазначенням імені та розміру з можливістю видалення кожного файлу окремо, форма параметрів ідентична формі одиночного кодування, а прогрес обробки відображається відповідним індикатором. Результат містить таблицю з показниками для кожного файлу (PSNR, MSE, час) та підсумкову статистику успішних операцій.

Сторінка управління повідомленнями Messages (рис. 3.7) відображає бібліотеку збережених закодованих повідомлень у табличному форматі з пагінацією. Кожен запис містить назву, категорію, метод стеганографії, довжину повідомлення, ознаку шифрування, значення PSNR та дату створення. Доступні фільтри за категорією та методом, а також текстовий пошук. Для кожного повідомлення реалізовано перегляд у модальному вікні, завантаження закодованого зображення, створення посилання для спільного доступу та видалення, а також пакетне видалення кількох записів одночасно.

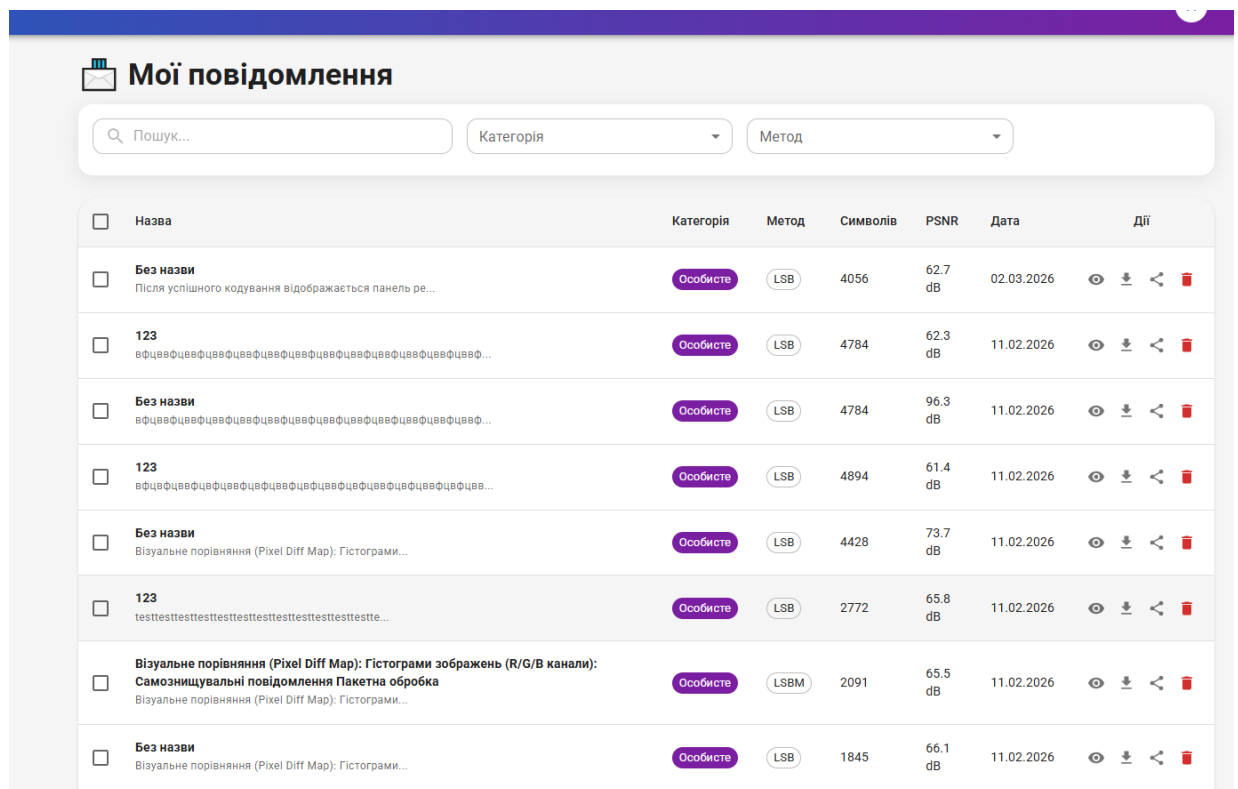


Рис. 3.7 – Сторінка управління повідомленнями

Сторінка історії операцій HistoryPage відображає хронологічний журнал усіх дій користувача із підсумковими картками (загальна кількість кодувань, декодувань, аналізів) та фільтрацією за типом операції. Таблиця містить тип дії, метод, розмір файлу, тривалість, статус виконання та час операції. Передбачено експорт історії у CSV-формат через ендпоінт `/api/history/export/csv`.

Сторінка профілю користувача Profile містить три секції: персональні дані (зміна імені та email), зміну паролю та налаштування (метод стеганографії за замовчуванням, автоматичне шифрування). Додатково відображається графік активності користувача за останній період, побудований на основі даних ендпоінту `/api/history/stats`.

Сторінка адміністративної панелі AdminPanel (рис. 3.8) доступна лише користувачам з роллю адміністратора та організована у три вкладки. Вкладка «Dashboard» відображає зведену статистику системи: загальну кількість користувачів, повідомлень та операцій, кругову діаграму розподілу методів стеганографії та стовпчикову діаграму щоденної активності. Вкладка «Користувачі»

містить таблицю з пошуком та можливістю блокування і розблокування облікових записів, а вкладка «Логи» відображає системні журнали з фільтрацією за рівнем (info, warn, error).

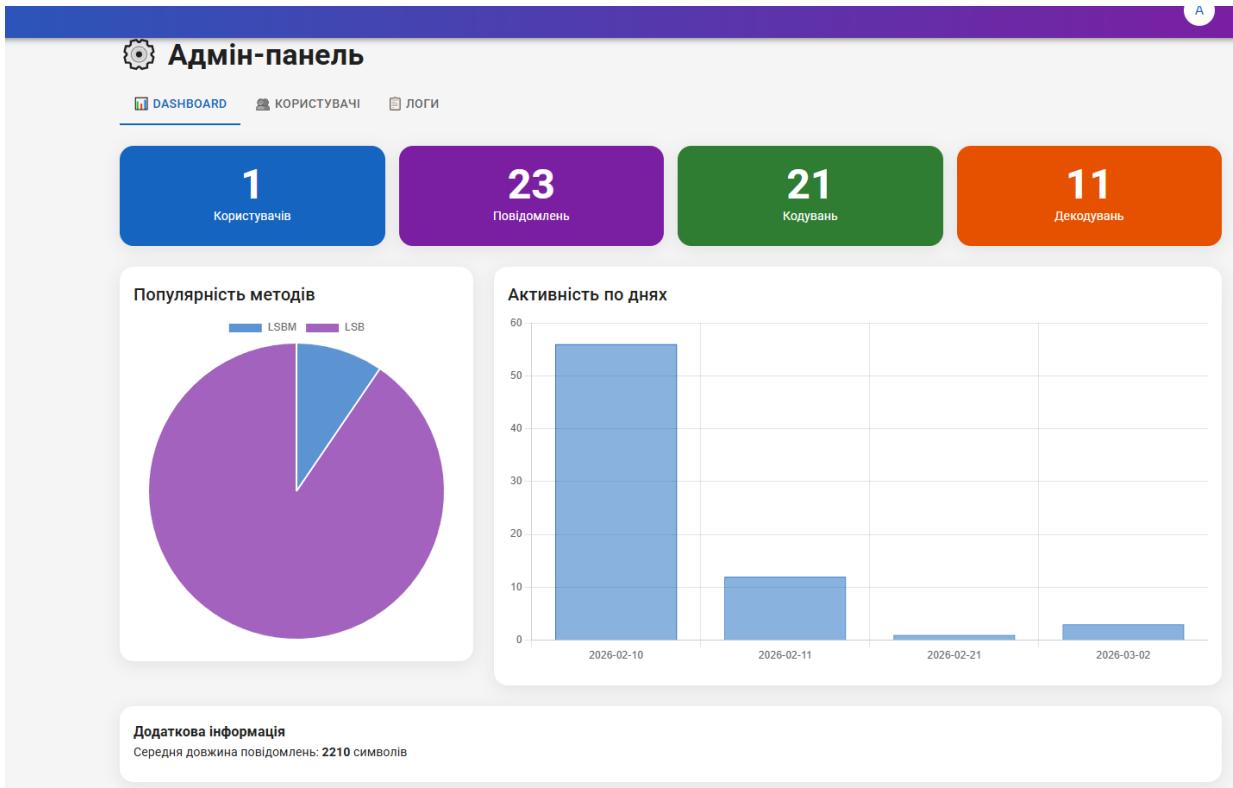


Рис. 3.8 – Адміністративна панель: вкладка Dashboard

Сторінка спільного доступу SharedView є публічно доступною та призначена для перегляду повідомлень за унікальним посиланням формату /shared/:code. При відкритті посилання система перевіряє його валідність: якщо посилання захищене паролем, відображається форма введення, а якщо термін дії або ліміт переглядів вичерпано — відповідне повідомлення. Після успішного доступу користувач може переглянути декодоване повідомлення, скопіювати його та завантажити закодоване зображення.

Перелік усіх сторінок інтерфейсу з їх функціональним призначенням та рівнем доступу наведено (табл. 3.1).

Таблиця 3.1

Функціональні сторінки інтерфейсу системи

Сторінка	Маршрут	Рівень доступу	Призначення
Головна	/	Публічний	Landing page з описом можливостей
Вхід	/login	Публічний	Автентифікація користувача
Реєстрація	/register	Публічний	Створення облікового запису
Кодування	/encode	Авторизований	Вбудовування повідомлення у зображення
Декодування	/decode	Авторизований	Витягнення повідомлення із зображення
Пакетна обробка	/batch	Авторизований	Кодування до 10 зображень одночасно
Аналіз зображень	/image-analysis	Авторизований	Diff Map та порівняння гістограм
Аналітика	/analytics	Авторизований	Порівняння методів стеганографії
Стегоаналіз	/steganalysis	Авторизований	χ^2 -тест та демонстрація вразливості LSB
Повідомлення	/messages	Авторизований	Бібліотека закодованих повідомлень
Історія	/history	Авторизований	Журнал операцій із експортом CSV
Профіль	/profile	Авторизований	Особисті дані та налаштування
Адмін-панель	/admin	Адміністратор	Статистика, користувачі, логи системи
Спільний доступ	/shared/:code	Публічний	Перегляд повідомлення за посиланням

Таким чином, інтерфейс системи StegoVault охоплює 14 функціональних сторінок із чітким розмежуванням доступу на трьох рівнях (публічний, авторизований, адміністратор), що відповідає сформованим функціональним і нефункціональним вимогам (розділ 2.1) та забезпечує повний цикл роботи з прихованою передачею інформації — від кодування та декодування до стеганоаналізу й адміністрування системи.

3.2 Експериментальне дослідження стійкості до атак

Метою експериментального дослідження є кількісна оцінка стійкості реалізованих методів стеганографії до статистичної χ^2 -атаки Вестфельда–Пфітцмана, визначення практичного порогу виявлюваності залежно від ступеня заповнення зображення, а також верифікація ефективності механізму Data Whitening

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

у складі повного конвеєру захисту. Усі експерименти виконано засобами платформи StegoVault; загальний час виконання всіх серій вимірювань склав 51.6 секунд.

Для проведення досліджень сформовано тестовий набір із чотирьох зображень формату PNG із різними характеристиками: фотографія з високою текстурністю (пейзаж, 1920×1080), фотографія з однорідними областями (портрет, 1280×960), штучне зображення з різкими переходами (графіка, 800×600) та фотографія середньої складності (побутова сцена, 1600×1200).

Перед основними вимірюваннями виконано χ^2 -аналіз усіх чотирьох зображень без будь-якого вбудовування — для встановлення базового рівня (baseline). Результати наведено (табл. 3.2).

Таблиця 3.2

Контрольний χ^2 -аналіз чистих зображень (без вбудовування)

Зображення	p-value (R)	p-value (G)	p-value (B)	p-value (max)	Рівень ризику
Пейзаж (1920×1080)	0.000	0.000	0.000	0.939	high
Портрет (1280×960)	0.000	0.000	0.000	0.000	clean
Графіка (800×600)	0.000	0.000	0.000	0.000	clean
Побутова (1600×1200)	0.000	0.000	0.000	0.000	clean

Три з чотирьох зображень демонструють p-value = 0.000 (рівень «clean») у всіх каналах, що підтверджує коректність базового стану. Зображення «Пейзаж», однак, показує p-value = 0.939 навіть без вбудовування. Цей результат є прикладом хибнопозитивного спрацьовування χ^2 -тесту, зумовленого природним розподілом градієнтів у текстурному зображенні: dense-texture фотографія з великою кількістю дрібних деталей (листя, трава) містить природний розподіл значень пікселів, який у деяких вікнах sliding window нагадує статистичний профіль LSB-вбудовування. Це є відомим обмеженням χ^2 -тесту Вестфельда, яке детально описане у літературі. Таким чином, для подальших серій вимірювань зображення «Пейзаж» є непридатним носієм для оцінки невиявлюваності, а всі висновки щодо стійкості базуються на трьох придатних зображеннях (Портрет, Графіка, Побутова).

Стійкість до χ^2 -атаки: наївне LSB vs Data Whitening, для демонстрації вразливості незахищеної LSB-стеганографії використано режим Weak Embed, який виконує пряме вбудовування псевдовипадкових бітів у 50% ємності зображення без шифрування, компресії та XOR-скремблювання. Такий підхід систематично вирівнює частоти Pairs of Values $\{2k, 2k+1\}$ у гістограмі кожного каналу — характерна ознака, яку детектує χ^2 -тест. Результати наведено (табл. 3.3).

Таблиця 3.3

Результати χ^2 -аналізу при слабкому вбудовуванні (без Data Whitening)

Зображення	Розмір (px)	p-value (R)	p-value (G)	p-value (B)	p-value (max)
Пейзаж	1920×1080	0.000	0.000	0.000	0.990
Портрет	1280×960	0.000	0.000	0.000	0.999
Графіка	800×600	0.000	0.000	0.000	0.980
Побутова	1600×1200	0.000	0.000	0.000	0.994

У всіх чотирьох випадках p-value перевищує 0.98, що відповідає рівню ризику «high». Характерно, що по каналах R, G, B окремо p-value = 0.000, тоді як фінальне значення (max серед усіх вікон sliding window) залишається критично високим — це пояснюється тим, що вирівнювання PoV-пар концентрується у конкретних вікнах, де щільність вбудовування найвища, і саме ці вікна формують пік p-value. Результат підтверджує: наївна LSB-стеганографія без захисту payload є повністю вразливою до χ^2 -атаки. Відповідний результат χ^2 -аналізу наведено (рис. 3.9).

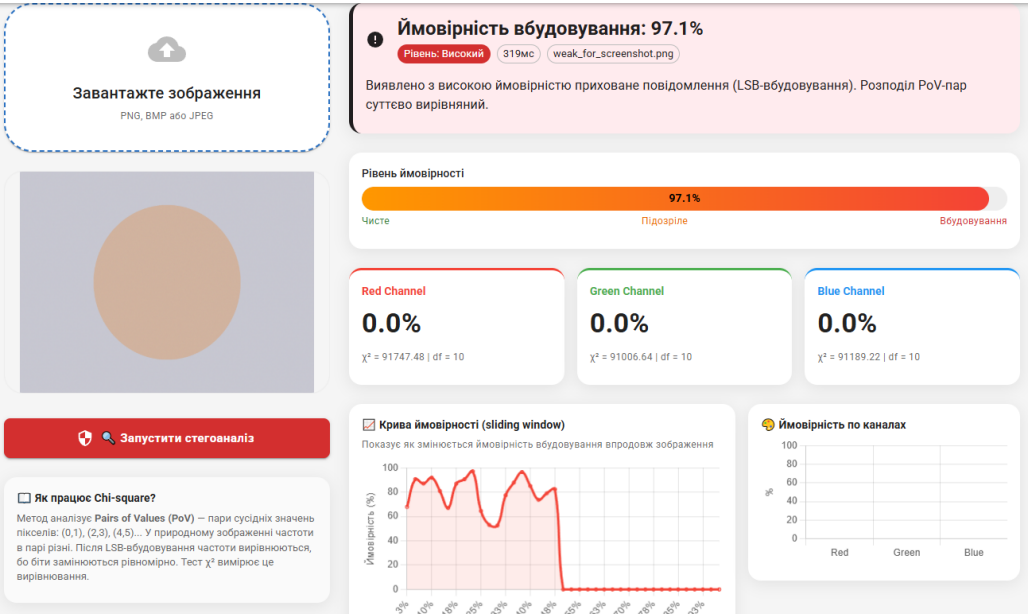


Рис. 3.9 – Результат χ^2 -аналізу зображення зі слабким вбудовуванням

Для перевірки ефективності повного конвеєру Data Whitening кожне тестове зображення закодовано з повідомленням довжиною 100 символів кирилицею трьома методами з увімкненим AES-256-CBC та Data Whitening. Результати наведено (табл. 3.4).

Таблиця 3.4

Результати χ^2 -аналізу при кодуванні з Data Whitening (100 символів)

Зображення	Метод	p-value (max)	Рівень ризику	PSNR (дБ)
Портрет (1280×960)	LSB	0.0000	clean	83.80
Портрет (1280×960)	LSBM	0.0000	clean	83.74
Портрет (1280×960)	Adaptive	0.0000	clean	83.63
Графіка (800×600)	LSB	0.0000	clean	79.59
Графіка (800×600)	LSBM	0.0000	clean	79.42
Графіка (800×600)	Adaptive	0.0000	clean	79.62
Побутова (1600×1200)	LSB	0.0000	clean	85.51
Побутова (1600×1200)	LSBM	0.0000	clean	85.53
Побутова (1600×1200)	Adaptive	0.0000	clean	85.49

У всіх дев'яти випадках (три придатних зображення × три методи) p-value = 0.0000, що відповідає рівню «clean». Порівняння з (табл. 3.3) ($p > 0.98$ без Data Whitening) демонструє радикальну відмінність: XOR-скремблювання на базі HMAC-SHA256 PRNG перетворює структурований текстовий payload на рівноімовірну бітову послідовність, яка не створює статистичного вирівнювання PoV-пар. Значення PSNR при малому payload (100 символів) є надзвичайно високим — 79–86 дБ, що значно перевищує поріг непомітності 40 дБ і свідчить про мінімальний вплив вбудовування на якість зображення при малих обсягах даних.

Для визначення практичного порогу виявлюваності проведено серію кодувань з повідомленнями зростаючої довжини на зображенні «Портрет» (1280×960, ємність LSB: ~460 795 символів, baseline $p = 0.000$). Результати наведено (табл. 3.5).

Таблиця 3.5

Залежність p-value та PSNR від обсягу payload (Портрет, LSB, з Data Whitening)

Обсяг волів)	(сим- волив)	Заповнення (%)	p-value (max)	Рівень ризику	PSNR (дБ)
100		0.02	0.0000	clean	83.54
500		0.11	0.0000	clean	77.63

1 000	0.22	0.0000	clean	74.74
5 000	1.09	0.0000	clean	67.75
10 000	2.17	0.0000	clean	64.72
50 000	10.85	0.9676	high	57.71
100 000	21.70	0.9798	high	54.70

Результати виявляють чіткий пороговий ефект: при заповненні до 2.17% ємності (10 000 символів) p-value залишається рівним 0.0000 (рівень «clean»), а PSNR — вище 64 дБ. При переході від 10 000 до 50 000 символів (заповнення 2.17% → 10.85%) p-value стрибкоподібно зростає до 0.9676 (рівень «high»). При 100 000 символів (21.70%) p-value досягає 0.9798, що порівнянне з результатами наївного Weak Embed. Характерно, що PSNR при цьому залишається на рівні 54–57 дБ, що перевищує поріг 40 дБ — тобто зображення є візуально якісним, але статистично вразливим. Це підкреслює, що χ^2 -тест виявляє статистичні аномалії задовго до появи видимих артефактів.

Таким чином, практичний поріг безпечного використання системи становить приблизно 2% ємності зображення при увімкненому Data Whitening, що для зображення формату 1280×960 відповідає ~9 200 символам — достатньо для передачі типових текстових повідомлень. Графічну залежність p-value та PSNR від ступеня заповнення представлено (рис. 3.10).

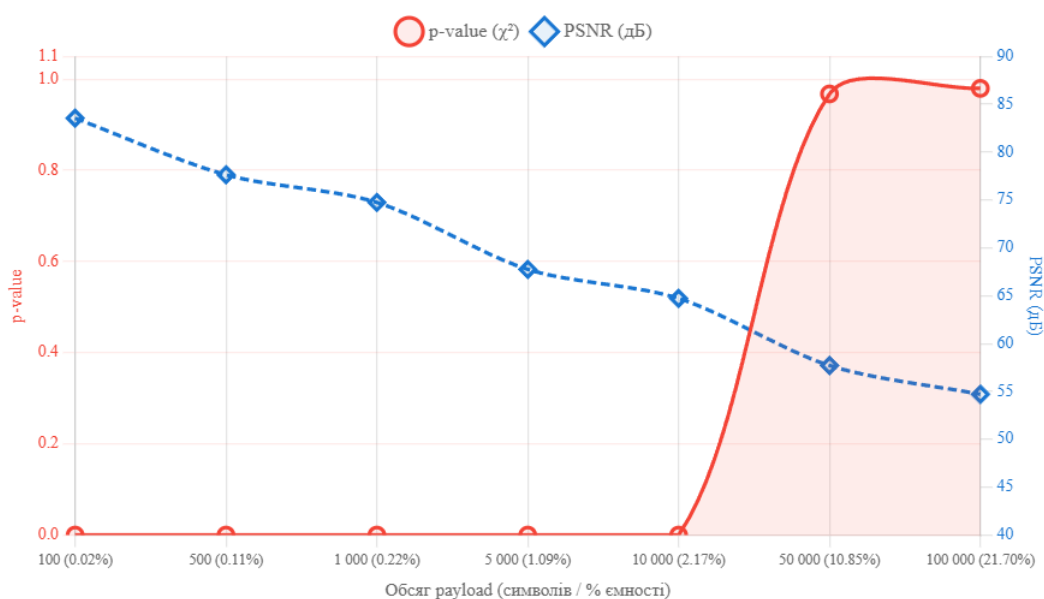


Рис. 3.10 – Залежність p-value та PSNR від ступеня заповнення зображення

Зведені результати порівняння трьох методів на всіх чотирьох тестових зображеннях наведено (табл. 3.6).

Таблиця 3.6

Порівняння методів стеганографії за метриками якості (100 символів)

Зображення	Метрика	LSB	LSBM	Adaptive
Пейзаж (1920×1080)	PSNR (дБ)	85.86	85.86	85.93
	Час (мс)	746	736	3 277
	Ємність (симв.)	777 595	777 595	466 555
Портрет (1280×960)	PSNR (дБ)	83.61	83.60	83.79
	Час (мс)	421	446	2 014
	Ємність (симв.)	460 795	460 795	276 475
Графіка (800×600)	PSNR (дБ)	79.61	79.60	79.67
	Час (мс)	164	173	593
	Ємність (симв.)	179 995	179 995	107 995
Побутова (1600×1200)	PSNR (дБ)	85.47	85.47	85.80
	Час (мс)	713	657	2 939
	Ємність (симв.)	719 995	719 995	431 995

Аналіз даних дозволяє сформулювати такі закономірності.

По-перше, різниця у значеннях PSNR між методами є мінімальною і не перевищує 0.3 дБ у межах одного зображення — усі три методи забезпечують практично однакову візуальну якість при малих обсягах payload.

По-друге, Adaptive метод є стабільно найповільнішим: час кодування у 4–5.5 разів перевищує час LSB завдяки обчисленню карти країв Собеля та сортуванню пікселів за градієнтом.

По-третє, ємність Adaptive становить ~60% від LSB/LSBM через використання лише текстурованих пікселів (вище 40-го перцентилу за градієнтом).

По-четверте, LSB та LSBM мають практично ідентичні показники PSNR, часу та ємності — перевага LSBM проявляється лише у стійкості до атак, що аналізують напрямок модифікації (WS-атака, Sample Pairs), а не до χ^2 -тесту. Візуалізацію порівняння PSNR представлено (рис. 3.11).

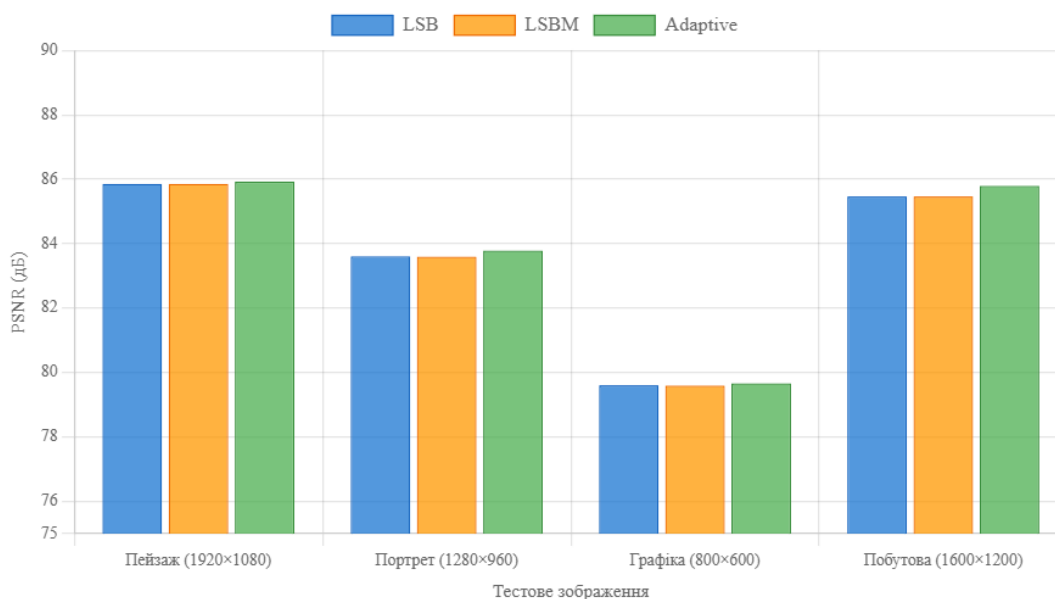


Рис. 3.11 – Порівняння PSNR методів стеганографії (модуль Analytics)

Для візуалізації просторового розподілу змін виконано аналіз Pixel Diff Map для зображення «Пейзаж» (1920×1080, 100 символів). Кількість змінених пікселів склала 607 для LSB та LSBM і 602 для Adaptive, що становить 0.029% від загальної кількості пікселів (2 073 600). Різниця у кількості змінених пікселів між методами мінімальна і визначається виключно обсягом payload. Принципова відмінність полягає у просторовому розподілі: LSB та LSBM модифікують пікселі послідовно від початку зображення (рядок за рядком, зліва направо), тоді як Adaptive концентрує зміни у текстурованих областях за картою Собеля. На Diff Map з коефіцієнтом підсилення 50 це відображається як компактна суцільна смуга у верхньому лівому куті (LSB/LSBM) проти розсіяних точок вздовж контурів об'єктів (Adaptive), що наочно підтверджує вищу стеганографічну якість адаптивного методу (рис. 3.12).

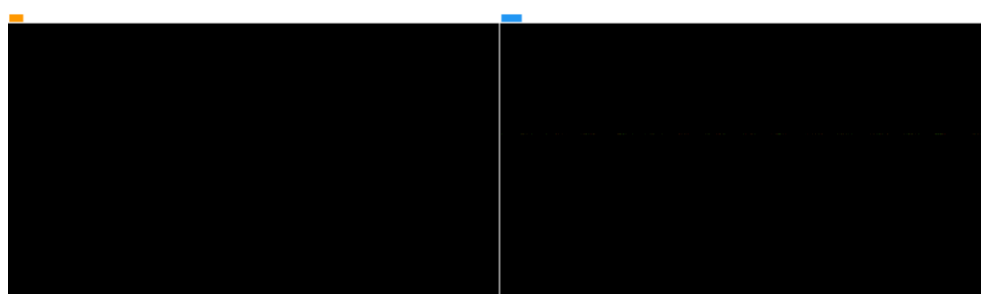


Рис. 3.12 –Pixel Diff Map: LSB (зліва) та Adaptive (справа), 100 символів, 1920×1080

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

Зведену порівняльну характеристику стійкості методів до χ^2 -атаки наведено (табл. 3.7).

Таблиця 3.7

Зведена характеристика стійкості методів стеганографії

Характеристика	LSB	LSBM	Adaptive
χ^2 -стійкість (з Data Whitening, до 2% заповнення)	Висока	Висока	Висока
χ^2 -стійкість (без Data Whitening)	Низька	Середня	Низька
PSNR при малому payload (дБ)	79–86	79–86	79–86
PSNR при великому payload (> 10%)	54–58	54–58	56–60
Просторовий розподіл змін	Послідовний	Послідовний	По краях/текстурах
Швидкість кодування	Висока	Висока	У 4–5 разів повільніше
Ємність (відносно LSB)	100%	100%	~60%
Рекомендований сценарій	Максимальна ємність, швидкість	Запас стійкості до WS-атаки	Мінімальна просторова помітність

На підставі проведених експериментів сформульовано такі висновки. По-перше, Data Whitening є визначальним фактором стійкості: без нього χ^2 -тест виявляє вбудовування з $p\text{-value} > 0.98$ для всіх чотирьох зображень, тоді як з увімкненим Data Whitening та AES-256-CBC усі три методи демонструють $p\text{-value} = 0.0000$ при типових обсягах payload до 2% ємності. По-друге, практичний поріг безпечного заповнення становить ~2% ємності: до цього порогу система забезпечує повну невиявлюваність при χ^2 -аналізі; понад 10% — система стає вразливою незалежно від методу. По-третє, природний розподіл пікселів деяких зображень (зокрема, текстурних фотографій типу «Пейзаж») може спричиняти хибно-позитивне спрацювання χ^2 -тесту навіть на чистих зображеннях — цей факт необхідно враховувати при виборі носія. По-четверте, усі три методи задовольняють встановлену нефункціональну вимогу НФВ-06 щодо якості зображення: $\text{PSNR} > 40$ дБ досягається при заповненні до 21.7% ємності.

3.3 Тестування функціоналу та інструкція користувача

Метою даного підрозділу є верифікація коректності реалізованих функціональних вимог шляхом ручного тестування основних сценаріїв використання платформи, а також формулювання стислої інструкції користувача для основних модулів системи. Тестування проведено методом чорної скриньки у браузері Google Chrome на локальному сервері, де серверна частина працювала на порту 5000, а клієнтська — на порту 3000.

Перевірку функціоналу виконано за 46 тестовими сценаріями, згрупованими за модулями системи. Зведені результати наведено (табл. 3.8).

Таблиця 3.8

Зведені результати функціонального тестування

Модуль	Сценаріїв	Пройдено	Ключові перевірені сценарії
Автентифікація	9	9	Реєстрація, вхід, захист маршрутів, зміна пароля, валідація форм
Кодування	12	12	LSB/LSBM/Adaptive, AES-256, drag&drop, ліміт розміру/формату, самознищення, shared link
Декодування	6	6	Відновлення тексту, невірний пароль, автовизначення методу, експорт .txt
Пакетна обробка	3	3	Кодування 5 файлів, ліміт 10 файлів, завантаження всіх результатів
Стегоаналіз	3	3	χ^2 -аналіз чистого та закодованого зображення, Weak Embed
Аналітика / Аналіз зображень	2	2	Порівняння методів, Pixel Diff Map, RGB-гістограми
Повідомлення	3	3	Фільтрація, масове видалення, shared link
Спільний доступ	2	2	Декодування за посиланням, вичерпаний ліміт переглядів
Історія	2	2	Журнал з фільтрацією, експорт CSV
Адмін-панель	4	4	Dashboard, блокування користувача, перегляд логів
Разом	46	46	—

Усі 46 тестових сценаріїв пройдено успішно (100%). Система коректно валідує вхідні дані на стороні клієнта та сервера, обробляє граничні випадки (невірний пароль, перевищення ліміту розміру, вичерпаний ліміт переглядів) та відображає інформативні повідомлення про помилки через компонент Alert.

Для доступу до платформи StegoVault користувач відкриває веб-застосунок у браузері та створює обліковий запис на сторінці реєстрації (ім'я, email, пароль не менше 6 символів). Після авторизації стають доступними всі функціональні модулі через бокову панель навігації.

Для кодування повідомлення користувач переходить на сторінку «Кодування» та виконує кілька дій. Спочатку він завантажує зображення формату PNG або BMP розміром до 10 МБ у зону dropzone — система автоматично відображає попередній перегляд та ємність носія для кожного методу (LSB, LSBM, Adaptive). Далі вводить текст повідомлення у відповідне поле, обирає метод стеганографії та за потреби активує шифрування AES-256-CBC з введенням пароля. Додатково можна задати назву, категорію, термін дії посилання та режим самознищення з обмеженням кількості переглядів. Після натискання кнопки «Закодувати повідомлення» система відображає закодоване зображення, метрики якості (PSNR, MSE, час виконання) та кнопки завантаження і створення посилання для спільного доступу (рис. 3.13).

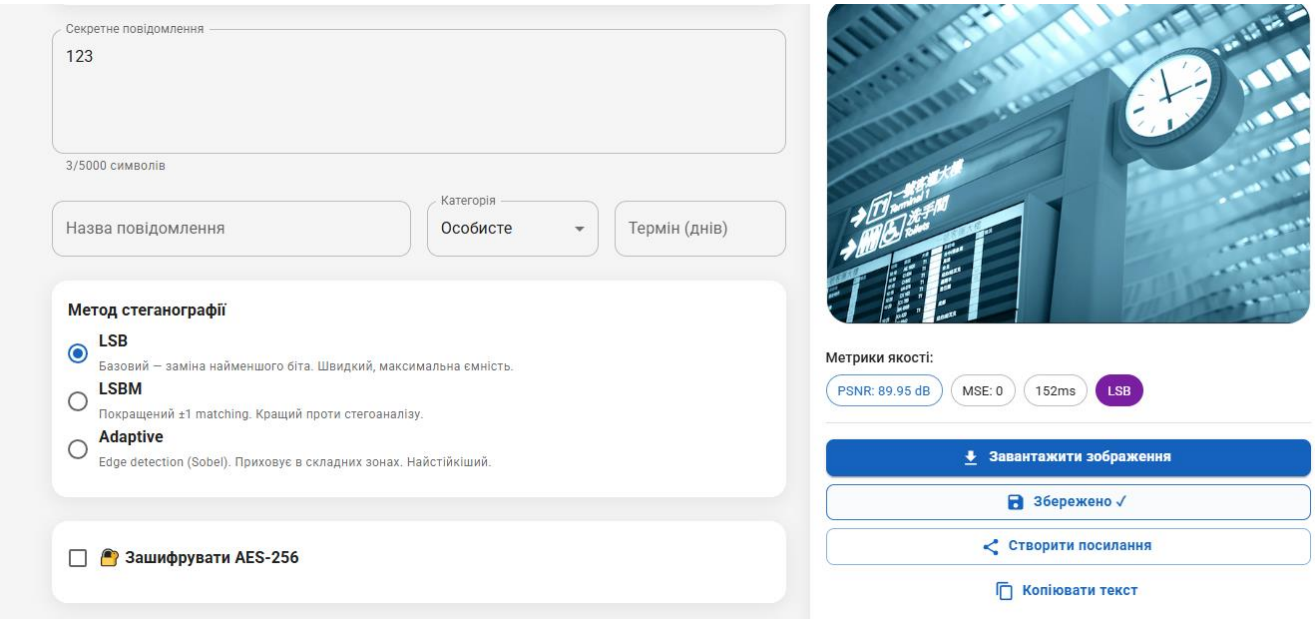


Рис. 3.13 – Інтерфейс кодування: завантажене зображення, форма параметрів та результат з метриками

Для декодування користувач переходить на сторінку «Декодування» та завантажує стего-зображення. Система автоматично визначає використаний метод

вбудовування за маркером у структурі даних, а за наявності шифрування вводиться пароль. Після натискання «Декодувати» у правій панелі відображається відновлений текст із зазначенням методу, часу обробки та кількості символів; текст можна скопіювати або експортувати у файл .txt (рис. 3.14).

Рис. 3.14 – Інтерфейс декодування: завантажене зображення та відновлений текст

Згенероване під час кодування посилання формату /shared/:code може бути передано отримувачу. Отримувач відкриває посилання у браузері без авторизації, за наявності пароля вводиться його у форму верифікації та переглядає декодоване повідомлення. Посилання з увімкненим самознищенням автоматично деактивується після вичерпання ліміту переглядів.

Для стегоаналізу на сторінці «Стегоаналіз» користувач завантажує зображення та натискає «Аналізувати». Система відображає кольоровий індикатор рівня ризику (clean / low / medium / high), таблицю χ^2 -статистики по каналах R, G, B та криву ковзного вікна для локалізації вбудовування. Режим «Weak Embed» дозволяє згенерувати навмисно вразливе зображення для демонстрації ефективності χ^2 -тесту. Результат χ^2 -стегоаналізу зображення показано (рис. 3.15).

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				77
Змн.	Арк.	№ докум.	Підпис	Дата		

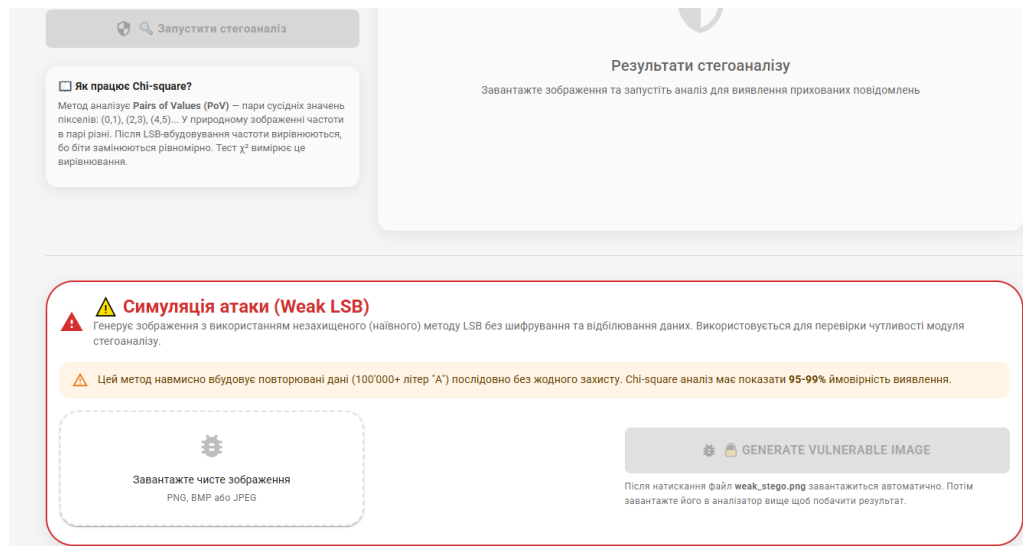


Рис. 3.15 – Результат χ^2 -стегоаналізу: індикатор ризику, таблиця каналів та крива ковзного вікна

Решта модулів доповнюють основний функціонал. Сторінка «Порівняння методів» автоматично тестує LSB, LSBM та Adaptive на одному зображенні і формує таблицю з метриками та рекомендацію оптимального методу. Сторінка «Аналіз зображень» дозволяє побудувати Pixel Diff Map та порівняти RGB-гістограми оригіналу й стего-зображення. Сторінка «Пакетна обробка» забезпечує одночасне кодування до 10 зображень. Сторінки «Повідомлення» та «Історія» надають доступ до бібліотеки збережених повідомлень та журналу операцій відповідно з можливістю фільтрації та експорту у CSV. Адміністратор додатково має доступ до зведеної статистики системи, управління обліковими записами та перегляду системних логів (рис. 3.16).

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

DASHBOARD КОРИСТУВАЧІ ЛОГИ				
Bci info warning error				
Рівень	Повідомлення	Дія	Користувач	Дата
info	Encoding: LSB, 5 chars	encode	admin@stego.com	02.03.2026, 10:36:54
info	Encoding: LSB, 3 chars	encode	admin@stego.com	02.03.2026, 10:35:41
info	Encoding: LSB, 4402 chars	encode	admin@stego.com	02.03.2026, 10:23:03
info	Encoding: LSB, 4056 chars	encode	admin@stego.com	02.03.2026, 09:32:11
info	Encoding: LSB, 4784 chars	encode	admin@stego.com	11.02.2026, 14:11:46
info	Encoding: LSB, 4784 chars	encode	admin@stego.com	11.02.2026, 14:09:38
info	Encoding: LSB, 4894 chars	encode	admin@stego.com	11.02.2026, 10:31:46
info	Encoding: LSB, 4428 chars	encode	admin@stego.com	11.02.2026, 00:33:42
info	Encoding: LSB, 2772 chars	encode	admin@stego.com	11.02.2026, 00:32:21
info	Encoding: LSBM, 2091 chars	encode	admin@stego.com	11.02.2026, 00:13:26

Рис. 3.16 – Dashboard адміністративної панелі: статистика, розподіл методів та активність

Таким чином, усі реалізовані модулі платформи StegoVault протестовано та підтверджено їх відповідність функціональним вимогам. Система забезпечує повний цикл стеганографічного захисту — від кодування з шифруванням до безпечного спільного доступу та стегоаналізу — з інтуїтивним інтерфейсом, що не потребує спеціальних технічних знань від кінцевого користувача.

Висновок до третього розділу

У третьому розділі виконано комплексну верифікацію розробленої платформи StegoVault, що охоплює опис структури інтерфейсу, експериментальне дослідження стійкості до стегоаналізу та функціональне тестування системи.

У підрозділі 3.1 описано архітектуру інтерфейсу системи, реалізованого як односторінковий застосунок на базі React 18 з 14 функціональними сторінками та тривірневим розмежуванням доступу (публічний, авторизований, адміністратор). Встановлено, що структура інтерфейсу повністю відповідає переліку функціональних вимог, визначених у розділі 2.

У підрозділі 3.2 проведено експериментальне дослідження стійкості трьох реалізованих методів стеганографії до χ^2 -атаки Вестфельда–Пфітцмана на тестовому наборі з чотирьох зображень. Ключовим результатом є підтвердження ефективності механізму Data Whitening: без нього χ^2 -тест виявляє LSB-вбудовування з $p\text{-value} > 0.98$ для всіх зображень, тоді як з увімкненим Data Whitening

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Концидайло А.М.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

та шифруванням AES-256-CBC усі три методи демонструють $p\text{-value} = 0.0000$ при заповненні до 2% ємності носія. Додатково виявлено пороговий ефект виявлюваності: при заповненні понад 10% $p\text{-value}$ стрибкоподібно зростає до рівня «high» навіть за наявності Data Whitening. Зафіксовано, що зображення з природно-текстурним розподілом пікселів (тип «Пейзаж») можуть давати хибнопозитивні результати χ^2 -тесту незалежно від вбудовування, що є відомим обмеженням методу. Всі три методи (LSB, LSBM, Adaptive) забезпечують $\text{PSNR} > 40$ дБ при заповненні до 21.7% ємності, задовольняючи нефункціональну вимогу НФВ-06.

У підрозділі 3.3 проведено ручне тестування методом чорної скриньки за 46 тестовими сценаріями, розподіленими між 10 функціональними модулями системи. Усі 46 сценаріїв пройдено успішно (100%), що підтверджує коректну реалізацію всіх функціональних вимог, у тому числі граничних випадків та механізмів валідації.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Концидайло А.М.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання — розроблено програмний комплекс StegoVault для стеганографічного захисту текстових повідомлень у растрових зображеннях із вбудованим модулем χ^2 -стеґоаналізу та криптографічним захистом AES-256-CBC.

За результатами виконаної роботи зроблено такі висновки:

1. Проведений аналіз предметної області підтвердив нагальну потребу в ефективних засобах прихованої передачі інформації: кількість кіберінцидентів в Україні зросла на 37,4% у 2025 році порівняно з 2024 роком, а середня вартість витоку даних у світі досягла 4,88 млн доларів США. Сформульовано 11 функціональних та 9 нефункціональних вимог до системи.

2. Досліджено та систематизовано методи цифрової стеганографії просторової області. Встановлено, що метод LSB забезпечує максимальну ємність (3 біти/піксель), LSBM — підвищену стійкість до χ^2 -атаки завдяки рівноімовірній модифікації пар значень, а Adaptive (Edge-Aware з оператором Собеля) — кращі показники PSNR при однаковому обсязі вбудованих даних завдяки концентрації змін у текстурованих ділянках.

3. Порівняльний аналіз чотирьох програмних аналогів (OpenStego, Steghide, SilentEye, OpenPuff) виявив три системних недоліки наявних рішень: відсутність веб-доступності, підтримка лише одного методу стеганографії та відсутність повноцінного модуля статистичного стеґоаналізу, що обґрунтувало доцільність власної розробки.

4. Спроектовано клієнт-серверну архітектуру системи на основі REST API (12 маршрутних модулів) та SPA з чітким розподілом відповідальностей між рівнями. Побудовано UML-діаграми компонентів, розгортання та варіантів використання, що описують три рівні доступу: гість, користувач і адміністратор.

5. Реалізовано алгоритмічне ядро системи: три методи вбудовування (LSB, LSBM, Adaptive), механізм Data Whitening (UTF-8 → DEFLATE → XOR-скремблювання на основі HMAC-SHA256), що усуває статистичні аномалії

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

payload незалежно від методу, та χ^2 -аналізатор із ковзним вікном (50% перекриття), що забезпечує виявлення часткового вбудовування.

6. Розроблено мультикористувацьку веб-платформу на стеку TypeScript, Node.js з Express, React та MongoDB з двотокенною JWT-автентифікацією, пакетною обробкою до 10 зображень, механізмом самознищувальних повідомлень та адміністративним інтерфейсом управління користувачами і системними журналами.

7. Експериментальне дослідження показало, що при типових обсягах вбудовування (до приблизно 2% ємності) усі три методи забезпечують PSNR значно вищий за пороговий критерій непомітності 40 дБ — на рівні 79–86 дБ, а навіть при заповненні до 21,7% ємності PSNR залишається в межах 54–58 дБ. За показниками якості зображення методи LSB, LSBM та Adaptive практично рівноцінні (різниця не перевищує часток децибела). Застосування Data Whitening знижує p-value χ^2 -тесту до рівня статистично незначущих значень незалежно від методу вбудовування.

8. Функціональне тестування за методом чорної скриньки охопило 46 тестових сценаріїв і підтвердило коректність роботи системи в усіх перевірених режимах із 100% успішністю проходження.

Практична цінність роботи полягає в тому, що розроблений комплекс StegoVault є першим серед розглянутих аналогів, що поєднує в єдиній веб-платформі три методи стеганографії, AES-256-CBC шифрування з Data Whitening, χ^2 -стегоаналіз та мультикористувацький режим. Система може бути застосована в задачах корпоративної безпеки, цифрової криміналістики та захищеної комунікації в умовах цензури або збройного конфлікту.

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				82
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. IBM Security. Cost of a Data Breach Report 2024. URL: <https://www.ibm.com/reports/data-breach> (дата звернення: 12.03.2026).
2. Verizon. 2024 Data Breach Investigations Report (DBIR). URL: <https://www.verizon.com/business/resources/reports/dbir/> (дата звернення: 12.03.2026).
3. CERT-UA. Звіт про кіберінциденти в Україні за 2024 рік. URL: <https://cert.gov.ua/> (дата звернення: 15.03.2026).
4. Держспецзв'язку України. Огляд кіберінцидентів за 2025 рік. URL: <https://cip.gov.ua/> (дата звернення: 15.03.2026).
5. Provos N., Honeyman P. Hide and Seek: An Introduction to Steganography. IEEE Security & Privacy. 2003. Vol. 1, № 3. P. 32–44.
6. Westfeld A., Pfitzmann A. Attacks on Steganographic Systems. Information Hiding. LNCS 1768. Springer, 2000. P. 61–76.
7. Mielikainen J. LSB Matching Revisited. IEEE Signal Processing Letters. 2006. Vol. 13, № 5. P. 285–287.
8. Gonzalez R. C., Woods R. E. Digital Image Processing. 4th ed. New York : Pearson, 2018. 1192 p.
9. Luo W., Huang F., Huang J. Edge Adaptive Image Steganography Based on LSB Matching Revisited. IEEE Transactions on Information Forensics and Security. 2010. Vol. 5, № 2. P. 201–214.
10. Deutsch P. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951. IETF, 1996.
11. Krawczyk H., Bellare M., Canetti R. HMAC: Keyed-Hashing for Message Authentication. RFC 2104. IETF, 1997.
12. OpenStego — Free Steganography Solution. URL: <https://www.openstego.com/> (дата звернення: 20.03.2026).
13. Steghide. URL: <https://steghide.sourceforge.net/> (дата звернення: 20.03.2026).

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

- 14.SilentEye — Cross-platform steganography application. URL: <https://achorein.github.io/silenteye/> (дата звернення: 22.03.2026).
- 15.OpenPuff Steganography & Watermarking. URL: https://embeddedsd.net/OpenPuff_Steganography_Home.html (дата звернення: 22.03.2026).
- 16.StegOnline — Web-based steganography tool. URL: <https://stegonline.georgeom.net/> (дата звернення: 24.03.2026).
- 17.TypeScript Documentation. URL: <https://www.typescriptlang.org/docs/> (дата звернення: 05.04.2026).
- 18.Stack Overflow. Developer Survey 2024. URL: <https://survey.stackoverflow.co/2024/> (дата звернення: 05.04.2026).
- 19.Express — Fast, unopinionated, minimalist web framework for Node.js. URL: <https://expressjs.com/> (дата звернення: 08.04.2026).
- 20.React — The library for web and native user interfaces. URL: <https://react.dev/> (дата звернення: 08.04.2026).
- 21.MongoDB Documentation. URL: <https://www.mongodb.com/docs/> (дата звернення: 10.04.2026).
- 22.Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. IETF, 2015.
- 23.Jimp — JavaScript Image Manipulation Program. URL: <https://github.com/jimp-dev/jimp> (дата звернення: 12.04.2026).
- 24.Provos N., Mazieres D. A Future-Adaptable Password Scheme. USENIX Annual Technical Conference. 1999. P. 81–91.
- 25.NIST. Advanced Encryption Standard (AES). FIPS PUB 197. 2001.
- 26.Moriarty K., Kaliski B., Rusch A. PKCS #5: Password-Based Cryptography Specification Version 2.1. RFC 8018. IETF, 2017.
- 27.Boutell T. et al. PNG Specification, Version 1.2. URL: <https://www.w3.org/TR/PNG/> (дата звернення: 18.04.2026).
- 28.OWASP. Web Security Testing Guide. URL: <https://owasp.org/www-project-web-security-testing-guide/> (дата звернення: 06.05.2026).

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидайло А.М.				84
Змн.	Арк.	№ докум.	Підпис	Дата		

29.MDN Web Docs. Web APIs. URL: <https://developer.mozilla.org/en-US/docs/Web/API> (дата звернення: 06.05.2026).

30.Material UI — React component library. URL: <https://mui.com/> (дата звернення: 10.05.2026).

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				85
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

Додаток А містить вихідний код ключових модулів платформи стеганографічного захисту зображень «StegoVault», розробленої з використанням технологій TypeScript, Node.js (Express), React та MongoDB.

А.1. Стеганографічний движок (backend/src/engine/steganography.ts)

// Стеганографічний движок - 3 методи: LSB, LSBM, Adaptive

// Модуль реалізує кодування, декодування та χ^2 -стегоаналіз зображень

```
import Jimp from 'jimp';
import path from 'path';
import zlib from 'zlib';
import crypto from 'crypto';
import fs from 'fs';
```

// типи

```
export type StegoMethod = 'LSB' | 'LSBM' | 'Adaptive';
```

```
export interface EncodeResult {
  outputPath: string;
  method: StegoMethod;
  psnr: number;
  mse: number;
  bitsUsed: number;
  capacity: number;
  duration: number;
}
```

```
export interface DecodeResult {
  message: string;
  method: StegoMethod;
  duration: number;
}
```

```
export interface CapacityInfo {
  lsb: number;
  lsbm: number;
  adaptive: number;
  width: number;
  height: number;
  totalPixels: number;
}
```

```
export interface MethodTestResult {
  capacity: number;
  psnr: number;
  time: number;
}
```

// маркери методів (перші 2 біти)

```
const METHOD_MARKERS: Record<StegoMethod, string> = {
  LSB: '00',
  LSBM: '01',
  Adaptive: '10',
};
```

// Data Whitening: вирівнює розподіл бітів payload для стійкості до χ^2 -аналізу

// text -> utf8 -> deflate -> xor scramble

// формат: [32 біти довжина][N*8 біт скрамбл дані]

```
const WHITENING_KEY = 'StegoEngine_DataWhitening_v2';
```

// PRNG на основі HMAC-SHA256 в режимі лічильника

```
function generatePRNGStream(length: number): Buffer {
```

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Концидайло А.М.				87
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const chunks: Buffer[] = [];
let totalLen = 0;
let counter = 0;
while (totalLen < length) {
  const hmac = crypto.createHmac('sha256', WHITENING_KEY);
  const counterBuf = Buffer.alloc(4);
  counterBuf.writeUInt32BE(counter);
  hmac.update(counterBuf);
  chunks.push(hmac.digest());
  totalLen += 32;
  counter++;
}
return Buffer.concat(chunks).subarray(0, length);
}

// xor скрамбл (самоінверсна операція - один і той же код для encode/decode)
function xorScramble(data: Buffer): Buffer {
  const stream = generatePRNGStream(data.length);
  const result = Buffer.alloc(data.length);
  for (let i = 0; i < data.length; i++) {
    result[i] = data[i] ^ stream[i];
  }
  return result;
}

// текст -> бінарний пейлоад з компресією і скрамблом
function textToPayload(text: string): string {
  const utf8 = Buffer.from(text, 'utf-8');
  const compressed = zlib.deflateSync(utf8, { level: 9 });
  const whitened = xorScramble(compressed);

  // 32-bit length prefix + payload bits
  const lengthBits = whitened.length.toString(2).padStart(32, '0');
  let payloadBits = "";
  for (let i = 0; i < whitened.length; i++) {
    payloadBits += whitened[i].toString(2).padStart(8, '0');
  }

  return lengthBits + payloadBits;
}

// зворотна операція: бінарний payload -> текст
function payloadToText(binary: string): string {
  // Read 32-bit length
  const lengthBits = binary.substring(0, 32);
  const payloadLength = parseInt(lengthBits, 2);

  if (payloadLength <= 0 || payloadLength > 10_000_000) {
    throw new Error("No hidden message found (invalid payload)");
  }

  // Read payload bytes
  const buf = Buffer.alloc(payloadLength);
  for (let i = 0; i < payloadLength; i++) {
    const byteBits = binary.substring(32 + i * 8, 32 + i * 8 + 8);
    if (byteBits.length < 8) throw new Error("Incomplete payload data");
    buf[i] = parseInt(byteBits, 2);
  }

  // Un-scramble and decompress
  const unscrambled = xorScramble(buf);
  try {
    const decompressed = zlib.inflateSync(unscrambled);
  }

```

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидайло А.М.				88
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return decompressed.toString('utf-8');
  } catch {
    throw new Error('No hidden message found (decompression failed)');
  }
}

function calculateMSE(original: Jimp, modified: Jimp): number {
  const w = original.bitmap.width;
  const h = original.bitmap.height;
  let sum = 0;
  let count = 0;

  for (let y = 0; y < h; y++) {
    for (let x = 0; x < w; x++) {
      const origPixel = Jimp.intToRGBA(original.getPixelColor(x, y));
      const modPixel = Jimp.intToRGBA(modified.getPixelColor(x, y));
      sum += Math.pow(origPixel.r - modPixel.r, 2);
      sum += Math.pow(origPixel.g - modPixel.g, 2);
      sum += Math.pow(origPixel.b - modPixel.b, 2);
      count += 3;
    }
  }

  return count > 0 ? sum / count : 0;
}

function calculatePSNR(mse: number): number {
  if (mse === 0) return Infinity;
  return 10 * Math.log10((255 * 255) / mse);
}

// Sobel детектор країв
function computeEdgeMap(image: Jimp): number[][] {
  const w = image.bitmap.width;
  const h = image.bitmap.height;
  const gray: number[][] = [];

  // Convert to grayscale
  for (let y = 0; y < h; y++) {
    gray[y] = [];
    for (let x = 0; x < w; x++) {
      const p = Jimp.intToRGBA(image.getPixelColor(x, y));
      // Mask out LSBs so edge map is identical on original & encoded images
      gray[y][x] = 0.299 * (p.r & 0xFE) + 0.587 * (p.g & 0xFE) + 0.114 * (p.b & 0xFE);
    }
  }

  const edgeMap: number[][] = [];
  for (let y = 0; y < h; y++) {
    edgeMap[y] = [];
    for (let x = 0; x < w; x++) {
      if (y === 0 || y === h - 1 || x === 0 || x === w - 1) {
        edgeMap[y][x] = 0;
        continue;
      }
      // Sobel kernels
      const gx =
        -gray[y - 1][x - 1] + gray[y - 1][x + 1] +
        -2 * gray[y][x - 1] + 2 * gray[y][x + 1] +
        -gray[y + 1][x - 1] + gray[y + 1][x + 1];

      const gy =
        -gray[y - 1][x - 1] - 2 * gray[y - 1][x] - gray[y - 1][x + 1] +

```

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				89
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        gray[y + 1][x - 1] + 2 * gray[y + 1][x] + gray[y + 1][x + 1];

        edgeMap[y][x] = Math.sqrt(gx * gx + gy * gy);
    }
}

return edgeMap;
}

// розрахунок місткості

export function calculateCapacity(width: number, height: number): CapacityInfo {
    const totalPixels = width * height;
    // Each pixel has 3 channels (RGB), each can store 1 bit
    const totalBits = totalPixels * 3;
    // Overhead: 2 bits method marker + 32 bits length prefix = 34 bits
    const overhead = 34;
    const availableBytes = Math.floor((totalBits - overhead) / 8);
    const compressionFactor = 1.0;

    const lsbCapacity = Math.floor(availableBytes / compressionFactor);
    const lsbmCapacity = Math.floor(availableBytes / compressionFactor);
    // Adaptive uses ~60% of pixels (only edge areas)
    const adaptiveBytes = Math.floor((totalBits * 0.6 - overhead) / 8);
    const adaptiveCapacity = Math.floor(adaptiveBytes / compressionFactor);

    return {
        lsb: Math.max(0, lsbCapacity),
        lsbm: Math.max(0, lsbmCapacity),
        adaptive: Math.max(0, adaptiveCapacity),
        width,
        height,
        totalPixels,
    };
}

// --- LSB encode ---

async function encodeLSB(image: Jimp, message: string): Promise<Jimp> {
    const result = image.clone();
    const binary = METHOD_MARKERS.LSB + textToPayload(message);
    const w = result.bitmap.width;
    const h = result.bitmap.height;

    let bitIndex = 0;

    for (let y = 0; y < h && bitIndex < binary.length; y++) {
        for (let x = 0; x < w && bitIndex < binary.length; x++) {
            const pixel = Jimp.intToRGBA(result.getPixelColor(x, y));
            const channels: ('r' | 'g' | 'b')[] = ['r', 'g', 'b'];

            for (const ch of channels) {
                if (bitIndex >= binary.length) break;
                const bit = parseInt(binary[bitIndex], 2);
                pixel[ch] = (pixel[ch] & 0xFE) | bit;
                bitIndex++;
            }

            const color = Jimp.rgbaToInt(pixel.r, pixel.g, pixel.b, pixel.a);
            result.setPixelColor(color, x, y);
        }
    }
}

```

		Пилипчук С.А.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидайло А.М.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    if (bitIndex < binary.length) {
        throw new Error('Image too small for this message');
    }

    return result;
}

// --- LSB decode ---

async function decodeLSB(image: Jimp): Promise<string> {
    const w = image.bitmap.width;
    const h = image.bitmap.height;
    let binary = "";
    let payloadLength = -1;

    for (let y = 0; y < h; y++) {
        for (let x = 0; x < w; x++) {
            const pixel = Jimp.intToRGBA(image.getPixelColor(x, y));
            binary += (pixel.r & 1).toString();
            binary += (pixel.g & 1).toString();
            binary += (pixel.b & 1).toString();

            // After 34 bits (2 marker + 32 length), parse the payload length
            if (payloadLength < 0 && binary.length >= 34) {
                payloadLength = parseInt(binary.substring(2, 34), 2);
                if (payloadLength <= 0 || payloadLength > 10_000_000) {
                    throw new Error('No hidden message found');
                }
            }

            // Check if we have all needed bits
            if (payloadLength > 0 && binary.length >= 34 + payloadLength * 8) {
                return payloadToText(binary.substring(2));
            }
        }
    }

    if (payloadLength > 0 && binary.length >= 34 + payloadLength * 8) {
        return payloadToText(binary.substring(2));
    }
    throw new Error('No hidden message found');
}

// LSBM: зміна значення пікселя на  $\pm 1$  замість прямої заміни LSB

async function encodeLSBM(image: Jimp, message: string): Promise<Jimp> {
    const result = image.clone();
    const binary = METHOD_MARKERS.LSBM + textToPayload(message);
    const w = result.bitmap.width;
    const h = result.bitmap.height;

    let bitIndex = 0;

    for (let y = 0; y < h && bitIndex < binary.length; y++) {
        for (let x = 0; x < w && bitIndex < binary.length; x++) {
            const pixel = Jimp.intToRGBA(result.getPixelColor(x, y));
            const channels: ('r' | 'g' | 'b')[] = ['r', 'g', 'b'];

            for (const ch of channels) {
                if (bitIndex >= binary.length) break;
                const bit = parseInt(binary[bitIndex], 2);
                const currentLSB = pixel[ch] & 1;

```

		Пилипчук С.А.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидайло А.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		91

```

    if (currentLSB !== bit) {
      // LSBM: randomly add or subtract 1 instead of direct replacement
      if (pixel[ch] === 0) {
        pixel[ch] = 1;
      } else if (pixel[ch] === 255) {
        pixel[ch] = 254;
      } else {
        pixel[ch] += Math.random() > 0.5 ? 1 : -1;
      }
    }
    bitIndex++;
  }

  const color = Jimp.rgbToInt(pixel.r, pixel.g, pixel.b, pixel.a);
  result.setPixelColor(color, x, y);
}

if (bitIndex < binary.length) {
  throw new Error('Image too small for this message');
}

return result;
}

// Адаптивний метод - вбудовуємо біти в областях з високою текстурою (edge areas)

async function encodeAdaptive(image: Jimp, message: string): Promise<Jimp> {
  const result = image.clone();
  const binary = METHOD_MARKERS.Adaptive + textToPayload(message);
  const w = result.bitmap.width;
  const h = result.bitmap.height;
  const edgeMap = computeEdgeMap(result);

  // Sort pixel positions by edge magnitude (descending) — embed in complex areas first
  const positions: { x: number; y: number; edge: number; idx: number }[] = [];
  let posIdx = 0;
  for (let y = 1; y < h - 1; y++) {
    for (let x = 1; x < w - 1; x++) {
      positions.push({ x, y, edge: edgeMap[y][x], idx: posIdx++ });
    }
  }
  // Stable sort: edge desc, then raster order asc (deterministic tiebreaker)
  positions.sort((a, b) => b.edge !== a.edge ? b.edge - a.edge : a.idx - b.idx);

  // Threshold: only use pixels with edge value > median
  const threshold = positions.length > 0
    ? positions[Math.floor(positions.length * 0.4)].edge
    : 0;

  let bitIndex = 0;

  for (const pos of positions) {
    if (bitIndex >= binary.length) break;
    if (pos.edge < threshold) continue;

    const pixel = Jimp.intToRGBA(result.getPixelColor(pos.x, pos.y));
    const channels: ('r' | 'g' | 'b')[] = ['r', 'g', 'b'];

    for (const ch of channels) {
      if (bitIndex >= binary.length) break;
      const bit = parseInt(binary[bitIndex], 2);
      pixel[ch] = (pixel[ch] & 0xFE) | bit;
    }
  }

```

		Пилипчук С.А.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидаіло А.М.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        bitIndex++;
    }

    const color = Jimp.rgbToHsl(pixel.r, pixel.g, pixel.b, pixel.a);
    result.setPixelColor(color, pos.x, pos.y);
}

if (bitIndex < binary.length) {
    throw new Error('Image too small for this message with Adaptive method');
}

return result;
}

// adaptive decode (той самий edge map що і в encode)

async function decodeAdaptive(image: Jimp): Promise<string> {
    const w = image.bitmap.width;
    const h = image.bitmap.height;
    const edgeMap = computeEdgeMap(image);

    const positions: { x: number; y: number; edge: number; idx: number }[] = [];
    let posIdx = 0;
    for (let y = 1; y < h - 1; y++) {
        for (let x = 1; x < w - 1; x++) {
            positions.push({ x, y, edge: edgeMap[y][x], idx: posIdx++ });
        }
    }
    positions.sort((a, b) => b.edge !== a.edge ? b.edge - a.edge : a.idx - b.idx);

    const threshold = positions.length > 0
        ? positions[Math.floor(positions.length * 0.4)].edge
        : 0;

    let binary = "";
    let payloadLength = -1;

    for (const pos of positions) {
        if (pos.edge < threshold) continue;

        const pixel = Jimp.intToRGBA(image.getPixelColor(pos.x, pos.y));
        binary += (pixel.r & 1).toString();
        binary += (pixel.g & 1).toString();
        binary += (pixel.b & 1).toString();

        if (payloadLength < 0 && binary.length >= 34) {
            payloadLength = parseInt(binary.substring(2, 34), 2);
            if (payloadLength <= 0 || payloadLength > 10_000_000) {
                throw new Error('No hidden message found');
            }
        }

        if (payloadLength > 0 && binary.length >= 34 + payloadLength * 8) {
            return payloadToText(binary.substring(2));
        }
    }

    if (payloadLength > 0 && binary.length >= 34 + payloadLength * 8) {
        return payloadToText(binary.substring(2));
    }
    throw new Error('No hidden message found');
}

```

		Пилипчук С.А.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидайло А.М.				93
Змн.	Арк.	№ докум.	Підпис	Дата		

```

// детект методу по перших 2 бітах

export async function detectMethod(imagePath: string): Promise<StegoMethod | null> {
  const image = await Jimp.read(imagePath);
  const pixel = Jimp.intToRGBA(image.getPixelColor(0, 0));

  const bit1 = (pixel.r & 1).toString();
  const bit2 = (pixel.g & 1).toString();
  const marker = bit1 + bit2;

  for (const [method, code] of Object.entries(METHOD_MARKERS)) {
    if (marker === code) return method as StegoMethod;
  }
  return null;
}

export async function encode(
  imagePath: string,
  message: string,
  method: StegoMethod,
  outputDir: string,
  outputFilename: string
): Promise<EncodeResult> {
  const startTime = Date.now();
  const image = await Jimp.read(imagePath);
  console.log(`[encode] method=${method}, image=${image.bitmap.width}x${image.bitmap.height}`);
  const original = image.clone();

  let encoded: Jimp;

  switch (method) {
    case 'LSB':
      encoded = await encodeLSB(image, message);
      break;
    case 'LSBM':
      encoded = await encodeLSBM(image, message);
      break;
    case 'Adaptive':
      encoded = await encodeAdaptive(image, message);
      break;
    default:
      throw new Error(`Unknown method: ${method}`);
  }

  const outputPath = path.join(outputDir, outputFilename);
  await encoded.writeAsync(outputPath);

  const mse = calculateMSE(original, encoded);
  const psnr = calculatePSNR(mse);
  const capacity = calculateCapacity(image.bitmap.width, image.bitmap.height);
  const duration = Date.now() - startTime;

  return {
    outputPath,
    method,
    psnr: Math.round(psnr * 100) / 100,
    mse: Math.round(mse * 1000) / 1000,
    bitsUsed: textToPayload(message).length + 2,
    capacity: capacity[method.toLowerCase() as 'lsb' | 'lsbm' | 'adaptive'],
    duration,
  };
}

```

		Пилипчук С.А.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидайло А.М.				94
Змн.	Арк.	№ докум.	Підпис	Дата		

```

export async function decode(
  imagePath: string,
  method?: StegoMethod
): Promise<DecodeResult> {
  const startTime = Date.now();
  const image = await Jimp.read(imagePath);

  // Auto-detect method if not specified
  const detectedMethod = method || await detectMethod(imagePath);
  if (!detectedMethod) {
    throw new Error('Could not detect steganography method');
  }

  let message: string;

  switch (detectedMethod) {
    case 'LSB':
    case 'LSBM': // LSBM decode is same as LSB (we read LSBs)
      message = await decodeLSB(image);
      break;
    case 'Adaptive':
      message = await decodeAdaptive(image);
      break;
    default:
      throw new Error(`Unknown method: ${detectedMethod}`);
  }

  return {
    message,
    method: detectedMethod,
    duration: Date.now() - startTime,
  };
}

export async function testAllMethods(
  imagePath: string,
  testMessage: string,
  outputDir: string
): Promise<Record<string, MethodTestResult>> {
  const methods: StegoMethod[] = ['LSB', 'LSBM', 'Adaptive'];
  const results: Record<string, MethodTestResult> = {};

  const image = await Jimp.read(imagePath);
  const cap = calculateCapacity(image.bitmap.width, image.bitmap.height);

  for (const method of methods) {
    try {
      const filename = `test_${method.toLowerCase()}_${Date.now()}.png`;
      const result: any = await encode(imagePath, testMessage, method, outputDir, filename);
      results[method.toLowerCase()] = {
        capacity: cap[method.toLowerCase() as 'lsb' | 'lsbm' | 'adaptive'],
        psnr: result.psnr,
        time: result.duration,
      };
    } catch {
      results[method.toLowerCase()] = {
        capacity: cap[method.toLowerCase() as 'lsb' | 'lsbm' | 'adaptive'],
        psnr: 0,
        time: 0,
      };
    }
  }
}

```

		Пилипчук С.А.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидайло А.М.				95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

return results;
}

export async function getImageInfo(imagePath: string) {
  const image = await Jimp.read(imagePath);
  const capacity = calculateCapacity(image.bitmap.width, image.bitmap.height);

  return {
    mime: image.getMIME(),
    ...capacity,
  };
}

// карта різниць між оригіналом і закодованим

export async function generatePixelDiffMap(
  originalPath: string,
  encodedPath: string,
  outputPath: string,
  amplification: number = 50
): Promise<{ diffPath: string; changedPixels: number; totalPixels: number; changePercent: number }> {
  const original = await Jimp.read(originalPath);
  const encoded = await Jimp.read(encodedPath);

  const w = Math.min(original.bitmap.width, encoded.bitmap.width);
  const h = Math.min(original.bitmap.height, encoded.bitmap.height);
  const diffImage = new Jimp(w, h);
  let changedPixels = 0;
  const totalPixels = w * h;

  for (let y = 0; y < h; y++) {
    for (let x = 0; x < w; x++) {
      const origIdx = (y * original.bitmap.width + x) * 4;
      const encIdx = (y * encoded.bitmap.width + x) * 4;

      const dr = Math.abs(original.bitmap.data[origIdx] - encoded.bitmap.data[encIdx]);
      const dg = Math.abs(original.bitmap.data[origIdx + 1] - encoded.bitmap.data[encIdx + 1]);
      const db = Math.abs(original.bitmap.data[origIdx + 2] - encoded.bitmap.data[encIdx + 2]);

      const diffIdx = (y * w + x) * 4;
      if (dr > 0 || dg > 0 || db > 0) {
        changedPixels++;
        // Amplify differences for visibility — map to heat-color
        const intensity = Math.min(255, (dr + dg + db) * amplification);
        diffImage.bitmap.data[diffIdx] = intensity; // R
        diffImage.bitmap.data[diffIdx + 1] = Math.max(0, 255 - intensity * 2); // G
        diffImage.bitmap.data[diffIdx + 2] = 0; // B
        diffImage.bitmap.data[diffIdx + 3] = 255; // A
      } else {
        // No change — dark pixel
        diffImage.bitmap.data[diffIdx] = 0;
        diffImage.bitmap.data[diffIdx + 1] = 0;
        diffImage.bitmap.data[diffIdx + 2] = 0;
        diffImage.bitmap.data[diffIdx + 3] = 255;
      }
    }
  }

  await diffImage.writeAsync(outputPath);

  return {
    diffPath: outputPath,
    changedPixels,
  }
}

```

		Пилипчук С.А.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидайло А.М.				96
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    totalPixels,
    changePercent: parseFloat(((changedPixels / totalPixels) * 100).toFixed(4)),
  };
}

// RGB гістограми

export interface HistogramData {
  r: number[];
  g: number[];
  b: number[];
}

export async function generateHistogram(imagePath: string): Promise<HistogramData> {
  const image = await Jimp.read(imagePath);
  const r = new Array(256).fill(0);
  const g = new Array(256).fill(0);
  const b = new Array(256).fill(0);

  for (let y = 0; y < image.bitmap.height; y++) {
    for (let x = 0; x < image.bitmap.width; x++) {
      const idx = (y * image.bitmap.width + x) * 4;
      r[image.bitmap.data[idx]]++;
      g[image.bitmap.data[idx + 1]]++;
      b[image.bitmap.data[idx + 2]]++;
    }
  }

  return { r, g, b };
}

export async function compareHistograms(
  originalPath: string,
  encodedPath: string
): Promise<{ original: HistogramData; encoded: HistogramData; difference: HistogramData }> {
  const original = await generateHistogram(originalPath);
  const encoded = await generateHistogram(encodedPath);

  const difference: HistogramData = {
    r: original.r.map((v, i) => encoded.r[i] - v),
    g: original.g.map((v, i) => encoded.g[i] - v),
    b: original.b.map((v, i) => encoded.b[i] - v),
  };

  return { original, encoded, difference };
}

// chi-square стеганаліз (Westfeld & Pfitzmann)

export interface ChiSquareResult {
  /** Overall embedding probability 0..1 */
  probability: number;
  chiSquare: number;
  degreesOfFreedom: number;
  /** Per-channel results */
  channels: {
    r: ChannelChiResult;
    g: ChannelChiResult;
    b: ChannelChiResult;
  };
  curve: { position: number; probability: number }[];
  risk: 'clean' | 'low' | 'medium' | 'high';
  verdict: string;
}

```

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				97
Змн.	Арк.	№ докум.	Підпис	Дата		

```

}

interface ChannelChiResult {
  chiSquare: number;
  probability: number;
  pairsAnalyzed: number;
}

// ln(Gamma) для розрахунку p-value
function lnGamma(z: number): number {
  const c = [
    76.18009172947146, -86.50532032941677, 24.01409824083091,
    -1.231739572450155, 0.001208650973866179, -0.000005395239384953,
  ];
  let x = z;
  let y = z;
  let tmp = x + 5.5;
  tmp -= (x + 0.5) * Math.log(tmp);
  let ser = 1.000000000190015;
  for (let j = 0; j < 6; j++) {
    y++;
    ser += c[j] / y;
  }
  return -tmp + Math.log(2.5066282746310005 * ser / x);
}

// Q(a,x) - upper incomplete gamma function
// Lentz continued fraction (Numerical Recipes)
function gammaQ(a: number, x: number): number {
  if (x <= 0) return 1;
  if (x < a + 1) {
    const gln = lnGamma(a);
    let ap = a;
    let sum = 1.0 / a;
    let del = sum;
    for (let n = 0; n < 300; n++) {
      ap += 1;
      del *= x / ap;
      sum += del;
      if (Math.abs(del) < Math.abs(sum) * 3e-14) break;
    }
    const P = sum * Math.exp(-x + a * Math.log(x) - gln);
    return Math.max(0, Math.min(1, 1 - P));
  } else {
    const gln = lnGamma(a);
    const FPMIN = 1e-30;
    let b = x + 1 - a;
    let c = 1 / FPMIN;
    let d = 1 / b;
    let h = d;
    for (let i = 1; i < 300; i++) {
      const an = -i * (i - a);
      b += 2;
      d = an * d + b;
      if (Math.abs(d) < FPMIN) d = FPMIN;
      c = b + an / c;
      if (Math.abs(c) < FPMIN) c = FPMIN;
      d = 1 / d;
      const del = d * c;
      h *= del;
      if (Math.abs(del - 1) < 3e-14) break;
    }
    const Q = Math.exp(-x + a * Math.log(x) - gln) * h;
  }
}

```

		Пилипчук С.А.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидайло А.М.				98
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return Math.max(0, Math.min(1, Q));
  }
}

// chi-square p-value
function chiSquarePValue(chiSq: number, df: number): number {
  if (df <= 0) return 0;
  return gammaQ(df / 2, chiSq / 2);
}

function analyzeChannel(values: number[]): ChannelChiResult {
  const hist = new Array(256).fill(0);
  for (const v of values) hist[v]++;

  // Pairs of Values (PoVs): (0,1), (2,3), (4,5), ... (254,255)
  let chiSq = 0;
  let df = 0;

  for (let i = 0; i < 256; i += 2) {
    const h_i = hist[i];
    const h_i1 = hist[i + 1];
    const expected = (h_i + h_i1) / 2;
    if (expected > 0) {
      chiSq += ((h_i - expected) ** 2) / expected;
      chiSq += ((h_i1 - expected) ** 2) / expected;
      df++;
    }
  }

  return {
    chiSquare: chiSq,
    probability: chiSquarePValue(chiSq, df),
    pairsAnalyzed: df,
  };
}

/**
 * Run Chi-square steganalysis on an image.
 * Analyses Pairs of Values (PoVs) in each RGB channel.
 */
export async function chiSquareAnalysis(imagePath: string): Promise<ChiSquareResult> {
  const image = await Jimp.read(imagePath);
  const w = image.bitmap.width;
  const h = image.bitmap.height;
  const data = image.bitmap.data;
  const totalPixels = w * h;

  // Collect per-channel values
  const rVals: number[] = [];
  const gVals: number[] = [];
  const bVals: number[] = [];

  for (let i = 0; i < totalPixels; i++) {
    rVals.push(data[i * 4]);
    gVals.push(data[i * 4 + 1]);
    bVals.push(data[i * 4 + 2]);
  }

  const rResult = analyzeChannel(rVals);
  const gResult = analyzeChannel(gVals);
  const bResult = analyzeChannel(bVals);

  const overallProb = Math.max(rResult.probability, gResult.probability, bResult.probability);

```

		Пилипчук С.А.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидайло А.М.				99
Змн.	Арк.	№ докум.	Підпис	Дата		

```

// Sliding window curve
const curve: { position: number; probability: number }[] = [];
const windowSize = Math.max(1000, Math.floor(totalPixels / 20));
const step = Math.max(1, Math.floor(windowSize / 2));

for (let start = 0; start + windowSize <= totalPixels; start += step) {
  const wR: number[] = [];
  const wG: number[] = [];
  const wB: number[] = [];
  for (let i = start; i < start + windowSize; i++) {
    wR.push(data[i * 4]);
    wG.push(data[i * 4 + 1]);
    wB.push(data[i * 4 + 2]);
  }
  const wr = analyzeChannel(wR);
  const wg = analyzeChannel(wG);
  const wb = analyzeChannel(wB);
  const windowProb = Math.max(wr.probability, wg.probability, wb.probability);
  curve.push({
    position: Math.round((start + windowSize / 2) / totalPixels * 100),
    probability: windowProb,
  });
}

const windowMax = curve.length > 0 ? Math.max(...curve.map(c => c.probability)) : 0;
const probability = Math.max(overallProb, windowMax);

let risk: ChiSquareResult['risk'];
let verdict: string;

if (probability > 0.9) {
  risk = 'high';
  verdict = 'Виявлено з високою ймовірністю приховане повідомлення (LSB-вбудовування). Розподіл PoV-пар суттєво вирівняний.';
} else if (probability > 0.5) {
  risk = 'medium';
  verdict = 'Є ознаки можливого стеганографічного вбудовування. Рекомендується додатковий аналіз.';
} else if (probability > 0.2) {
  risk = 'low';
  verdict = 'Невеликі відхилення у розподілі PoV-пар. Ймовірність вбудовування низька.';
} else {
  risk = 'clean';
  verdict = 'Зображення чисте. Розподіл пар значень природний, ознак LSB-стеганографії не виявлено.';
}

return {
  probability,
  chiSquare: Math.max(rResult.chiSquare, gResult.chiSquare, bResult.chiSquare),
  degreesOfFreedom: rResult.pairsAnalyzed,
  channels: { r: rResult, g: gResult, b: bResult },
  curve,
  risk,
  verdict,
};
}

// "тупе" вбудовування для демонстрації вразливості
export async function embedWeakNaive(imagePath: string, outputPath: string): Promise<{ outputPath: string; bitsEmbedded: number }> {
  const image = await Jimp.read(imagePath);
  const w = image.bitmap.width;
  const h = image.bitmap.height;

```

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидайло А.М.				100
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const capacityBits = w * h * 3;

const totalBits = Math.floor(capacityBits * 0.5)
let binaryString = ""
for (let i = 0; i < totalBits; i++) {
  binaryString += Math.random() > 0.5 ? '1' : '0'
}

let bitIndex = 0

for (let y = 0; y < h; y++) {
  for (let x = 0; x < w; x++) {
    if (bitIndex >= binaryString.length) break

    const pixel = Jimp.intToRGBA(image.getPixelColor(x, y))

    if (bitIndex < binaryString.length) {
      const bit = parseInt(binaryString[bitIndex])
      pixel.r = (pixel.r & 0xFE) | bit
      bitIndex++
    }
    if (bitIndex < binaryString.length) {
      const bit = parseInt(binaryString[bitIndex])
      pixel.g = (pixel.g & 0xFE) | bit
      bitIndex++
    }
    if (bitIndex < binaryString.length) {
      const bit = parseInt(binaryString[bitIndex])
      pixel.b = (pixel.b & 0xFE) | bit
      bitIndex++
    }

    image.setPixelColor(Jimp.rgbaToInt(pixel.r, pixel.g, pixel.b, pixel.a), x, y)
  }
}

await image.writeAsync(outputPath)
return { outputPath, bitsEmbedded: bitIndex }
}

```

A.2. Модуль шифрування (backend/src/engine/encryption.ts)

```

import crypto from 'crypto';

const ALGORITHM = 'aes-256-cbc';
const KEY_LENGTH = 32;
const IV_LENGTH = 16;
const SALT_LENGTH = 16;
const ITERATIONS = 100000;

export function encryptMessage(message: string, password: string): string {
  const salt = crypto.randomBytes(SALT_LENGTH);
  const key = crypto.pbkdf2Sync(password, salt, ITERATIONS, KEY_LENGTH, 'sha512');
  const iv = crypto.randomBytes(IV_LENGTH);

  const cipher = crypto.createCipheriv(ALGORITHM, key, iv);
  let encrypted = cipher.update(message, 'utf8', 'hex');
  encrypted += cipher.final('hex');

  // Format: salt:iv:encrypted
  return `${salt.toString('hex')}:${iv.toString('hex')}:${encrypted}`;
}

```

		Пилипчук С.А.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидайло А.М.				101
Змн.	Арк.	№ докум.	Підпис	Дата		

```

export function decryptMessage(encryptedData: string, password: string): string {
  const parts = encryptedData.split(':');
  if (parts.length !== 3) {
    throw new Error('Invalid encrypted data format');
  }

  const salt = Buffer.from(parts[0], 'hex');
  const iv = Buffer.from(parts[1], 'hex');
  const encrypted = parts[2];

  const key = crypto.pbkdf2Sync(password, salt, ITERATIONS, KEY_LENGTH, 'sha512');
  const decipher = crypto.createDecipheriv(ALGORITHM, key, iv);

  let decrypted = decipher.update(encrypted, 'hex', 'utf8');
  decrypted += decipher.final('utf8');

  return decrypted;
}

```

A.3. REST API кодування (backend/src/routes/encode.routes.ts)

```

import { Router, Response } from 'express';
import path from 'path';
import fs from 'fs';
import { auth, AuthRequest } from '../middleware/auth';
import { upload } from '../middleware/upload';
import { encodeLimiter } from '../middleware/rateLimiter';
import { encode, getImageInfo } from '../engine/steganography';
import { encryptMessage } from '../engine/encryption';
import Message from '../models/Message';
import History from '../models/History';
import SystemLog from '../models/SystemLog';
import { logger } from '../config/logger';

const router = Router();
const uploadDir = process.env.UPLOAD_DIR || './uploads';

// GET IMAGE INFO (capacity, etc)
router.post('/analyze', auth, upload.single('image'), async (req: AuthRequest, res: Response) => {
  try {
    if (!req.file) {
      return res.status(400).json({ error: 'Зображення не завантажено' });
    }
    const info = await getImageInfo(req.file.path);
    return res.json(info);
  } catch (error: any) {
    logger.error('Analyze error:', error);
    return res.status(500).json({ error: error.message || 'Помилка аналізу' });
  }
});

// ENCODE MESSAGE
router.post(
  '/',
  auth,
  encodeLimiter,
  upload.single('image'),
  async (req: AuthRequest, res: Response) => {
    const startTime = Date.now();
    try {
      if (!req.file) {

```

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				102
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return res.status(400).json({ error: 'Зображення не завантажено' });
}

const {
  message,
  method = 'LSB',
  encrypt: shouldEncrypt,
  password,
  title = 'Без назви',
  category = 'personal',
  expiresInDays,
  selfDestruct,
  maxViews,
} = req.body;

if (!message || message.length === 0) {
  return res.status(400).json({ error: 'Повідомлення не може бути порожнім' });
}
if (message.length > 5000) {
  return res.status(400).json({ error: 'Повідомлення занадто довге (макс 5000 символів)' });
}
if (![ 'LSB', 'LSBM', 'Adaptive' ].includes(method)) {
  return res.status(400).json({ error: 'Невідомий метод' });
}

let finalMessage = message;
const isEncrypted = shouldEncrypt === 'true' || shouldEncrypt === true;

if (isEncrypted) {
  if (!password) {
    return res.status(400).json({ error: 'Пароль обов\'язковий для шифрування' });
  }
  finalMessage = encryptMessage(message, password);
}

const outputFilename = `encoded_${Date.now()}.png`;
const encodedDir = path.join(uploadDir, 'encoded');

if (!fs.existsSync(encodedDir)) {
  fs.mkdirSync(encodedDir, { recursive: true });
}

const result = await encode(req.file.path, finalMessage, method, encodedDir, outputFilename);

// Save message to DB
const expiresAt = expiresInDays
  ? new Date(Date.now() + parseInt(expiresInDays) * 24 * 60 * 60 * 1000)
  : undefined;

const savedMessage = await Message.create({
  userId: req.user!._id,
  title,
  category,
  originalImagePath: req.file.path,
  encodedImagePath: result.outputPath,
  messagePreview: message.substring(0, 50),
  messageLength: message.length,
  method,
  encrypted: isEncrypted,
  psnr: result.psnr,
  mse: result.mse,
  expiresAt,
  isPublic: false,

```

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				103
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    selfDestruct: selfDestruct === 'true' || selfDestruct === true,
    maxViews: maxViews ? parseInt(maxViews) : 0,
  });

  // Save to history
  await History.create({
    userId: req.user!._id,
    action: 'encode',
    method,
    fileSize: req.file.size,
    duration: Date.now() - startTime,
    success: true,
    details: `Encoded ${message.length} chars with ${method}`,
  });

  await SystemLog.create({
    level: 'info',
    message: `Encoding: ${method}, ${message.length} chars`,
    userId: req.user!._id,
    action: 'encode',
  });

  return res.json({
    success: true,
    messageId: savedMessage._id,
    encodedImageUrl: `/uploads/encoded/${outputFilename}`,
    metrics: {
      psnr: result.psnr,
      mse: result.mse,
      bitsUsed: result.bitsUsed,
      capacity: result.capacity,
      duration: result.duration,
    },
    method,
    encrypted: isEncrypted,
  });
} catch (error: any) {
  // Log failure
  await History.create({
    userId: req.user!._id,
    action: 'encode',
    method: req.body?.method || 'LSB',
    fileSize: req.file?.size || 0,
    duration: Date.now() - startTime,
    success: false,
    details: error.message,
  });

  logger.error('Encode error:', error);
  return res.status(500).json({ error: error.message || 'Помилка кодування' });
}
}
);

export default router;

```

A.4. REST API декодування (backend/src/routes/decode.routes.ts)

```

import { Router, Response } from 'express';
import { auth, AuthRequest } from '../middleware/auth';
import { upload } from '../middleware/upload';
import { decode, detectMethod } from '../engine/steganography';
import { decryptMessage } from '../engine/encryption';

```

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Концидайло А.М.				104
Змн.	Арк.	№ докум.	Підпис	Дата		

```

import History from '../models/History';
import { logger } from '../config/logger';

const router = Router();

router.post('/', auth, upload.single('image'), async (req: AuthRequest, res: Response) => {
  const startTime = Date.now();
  try {
    if (!req.file) {
      return res.status(400).json({ error: 'Зображення не завантажено' });
    }

    const { method, password, encrypted } = req.body;

    // Detect method if not specified
    const detectedMethod = method || (await detectMethod(req.file.path));

    const result = await decode(req.file.path, detectedMethod || undefined);

    let finalMessage = result.message;
    const isEncrypted = encrypted === 'true' || encrypted === true;

    if (isEncrypted) {
      if (!password) {
        return res.status(400).json({ error: 'Пароль обов'язковий для розшифрування' });
      }
      try {
        finalMessage = decryptMessage(result.message, password);
      } catch {
        return res.status(400).json({ error: 'Невірний пароль або пошкоджені дані' });
      }
    }

    // Save to history
    await History.create({
      userId: req.user!._id,
      action: 'decode',
      method: result.method,
      fileSize: req.file.size,
      duration: Date.now() - startTime,
      success: true,
      details: `Decoded ${finalMessage.length} chars with ${result.method}`,
    });

    return res.json({
      success: true,
      message: finalMessage,
      method: result.method,
      duration: result.duration,
      encrypted: isEncrypted,
    });
  } catch (error: any) {
    await History.create({
      userId: req.user!._id,
      action: 'decode',
      method: req.body?.method || 'LSB',
      fileSize: req.file?.size || 0,
      duration: Date.now() - startTime,
      success: false,
      details: error.message,
    }).catch(() => {});

    logger.error('Decode error:', error);
  }
}

```

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				105
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return res.status(500).json({ error: error.message || 'Помилка декодування' });
  }
});

// DETECT METHOD
router.post('/detect', auth, upload.single('image'), async (req: AuthRequest, res: Response) => {
  try {
    if (!req.file) {
      return res.status(400).json({ error: 'Зображення не завантажено' });
    }
    const method = await detectMethod(req.file.path);
    return res.json({ method: method || 'unknown' });
  } catch (error: any) {
    return res.status(500).json({ error: error.message });
  }
});

export default router;

```

A.5. REST API стегоаналізу (backend/src/routes/steganalysis.routes.ts)

```

import { Router, Response } from 'express';
import fs from 'fs';
import path from 'path';
import { auth, AuthRequest } from '../middleware/auth';
import { upload } from '../middleware/upload';
import { chiSquareAnalysis, embedWeakNaive } from '../engine/steganography';
import { logger } from '../config/logger';
import History from '../models/History';

const router = Router();

// хі-квадрат стегоаналіз
router.post(
  '/chi-square',
  auth,
  upload.single('image'),
  async (req: AuthRequest, res: Response) => {
    try {
      if (!req.file) {
        return res.status(400).json({ error: 'Зображення не завантажено' });
      }

      const start = Date.now();
      const result = await chiSquareAnalysis(req.file.path);
      const duration = Date.now() - start;

      // зберігаємо в історію
      try {
        await History.create({
          userId: req.user!._id,
          action: 'analyze',
          method: 'Chi-square',
          details: `Стегоаналіз: ймовірність ${((result.probability * 100).toFixed(1))}%, рівень ризику: ${result.risk}`,
        });
      } catch (_) {
        // non-critical
      }

      // очищення файлу після аналізу
      try { fs.unlinkSync(req.file.path); } catch (_) {}
    }
  }
);

```

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				106
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    res.json({
      ...result,
      duration,
      imageInfo: {
        filename: req.file.originalname,
        size: req.file.size,
      },
    });
  } catch (error: any) {
    logger.error('Chi-square analysis error:', error);
    if (req.file) {
      try { fs.unlinkSync(req.file.path); } catch (_) {}
    }
    res.status(500).json({ error: error.message || 'Помилка стегоаналізу' });
  }
}
);

// симуляція слабкої атаки - наївний LSB без whitening
router.post(
  '/weak-embed',
  auth,
  upload.single('image'),
  async (req: AuthRequest, res: Response) => {
    try {
      if (!req.file) {
        return res.status(400).json({ error: 'Зображення не завантажено' });
      }

      const outputDir = path.join(__dirname, '../uploads')
      if (!fs.existsSync(outputDir)) fs.mkdirSync(outputDir, { recursive: true })
      const outputFilename = `weak_${Date.now()}.png`
      const outputPath = path.join(outputDir, outputFilename)

      const result = await embedWeakNaive(req.file.path, outputPath)

      try { fs.unlinkSync(req.file.path) } catch (_) {}

      // віддаємо файл і чистимо потім
      res.download(result.outputPath, 'weak_stego.png', (err) => {
        try { fs.unlinkSync(result.outputPath) } catch (_) {}
        if (err) logger.error('weak-embed download error', err)
      })
    } catch (error: any) {
      logger.error('Weak embed error:', error)
      if (req.file) {
        try { fs.unlinkSync(req.file.path) } catch (_) {}
      }
      res.status(500).json({ error: error.message || 'Помилка генерації' })
    }
  }
);

export default router;

```

A.6. Middleware аутентифікації (backend/src/middleware/auth.ts)

```

import { Request, Response, NextFunction } from 'express';
import jwt from 'jsonwebtoken';
import User, { IUser } from '../models/User';

```

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				107
Змн.	Арк.	№ докум.	Підпис	Дата		

```

export interface AuthRequest extends Request {
  user?: IUser;
  userId?: string;
}

export const auth = async (req: AuthRequest, res: Response, next: NextFunction): Promise<void> => {
  try {
    const header = req.headers.authorization;
    if (!header || !header.startsWith('Bearer ')) {
      res.status(401).json({ error: 'Доступ заборонено. Токен не надано.' });
      return;
    }

    const token = header.split(' ')[1];
    const decoded = jwt.verify(token, process.env.JWT_SECRET || 'secret') as { userId: string };

    const user = await User.findById(decoded.userId).select('-passwordHash');
    if (!user) {
      res.status(401).json({ error: 'Користувача не знайдено' });
      return;
    }
    if (user.isBlocked) {
      res.status(403).json({ error: 'Акаунт заблоковано' });
      return;
    }

    req.user = user;
    req.userId = user._id.toString();
    next();
  } catch (error) {
    res.status(401).json({ error: 'Невалідний токен' });
  }
};

export const adminOnly = (req: AuthRequest, res: Response, next: NextFunction): void => {
  if (!req.user || req.user.role !== 'admin') {
    res.status(403).json({ error: 'Доступ лише для адміністраторів' });
    return;
  }
  next();
};

```

		Пилипчук С.А.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Концидайло А.М.				108
Змн.	Арк.	№ докум.	Підпис	Дата		