МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Розробка застосунку для стеганографії у файли зображень із використанням засобів JavaFX**

Виконав: студент 4 курсу, групи 6.1211-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
Б.С. Собко
(ініціали та прізвище)
Керівник доцент каф. ЕІС та ПЗ, к.ф.-м.н., доцент
Г.П. Коломоєць
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент Директор ТОВ «Дискус»
Р.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 Інженерія програмного забезпечення
(код та назва)

Освітня програма Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри Тетяна КРИТСЬКА

“ 01 ” березня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Собку Богдану Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка застосунку для стеганографії у файли зображень із використанням засобів JavaFX
керівник роботи Коломоєць Геннадій Павлович, к.ф.-м.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 27.12.2024 р. № 2029-с
2. Строк подання студентом кваліфікаційної роботи 02.06.2025
3. Вихідні дані бакалаврської роботи
• комплект нормативних документів ;
• технічне завдання до роботи.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
• огляд та збір літератури стосовно теми кваліфікаційної роботи;
• огляд та аналіз існуючих рішень та аналогів;
• дослідження методів інтервального повторення та їх застосування в мобільних додатках;
• створення програмного продукту та його опис;
• перелік вимог для роботи програми;
• розробка висновків та пропозицій.

4. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
слайдів презентації

Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

Дата видачі завдання 01.03.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Вивчення принципів стеганографії. Дослідження алгоритмів приховування інформації у файлах зображень, аудіо та відеофайлах.	17.12.24 – 16.02.25	
2	Переваги платформи JavaFX при побудові GUI-застосунків. Поняття графу сцени. Основні віджети та лейаути. Дизайн компонентів за допомогою CSS. Обробка подій. Використання FXML та JAVAFX Scene Builder. Формулювання вимог до застосунку для стеганографії у файли зображень.	16.02 – 30.03.25	
3	Проєктування та конструювання застосунку із використанням засобів JavaFX. Напрацювання рекомендацій щодо використання застосунку у навчальних цілях.	31.03 – 26.05.25	
4	Оформлення дипломної роботи	27.05 – 08.06.25	
5	Попередній захист	03.06.2025	
6	Захист дипломної роботи	19.06.2025	

Студент _____ Б.С. Собко
(підпис) (прізвище та ініціали)

Керівник роботи _____ Г.П. Коломоєць
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціа

АНОТАЦІЯ

Сторінок — 74.

Рисунків — 27.

Лістинги — 18.

Таблиць — 1.

Джерел — 20.

Собко Б. С. Розробка застосунку для стеганографії у файли зображень із використанням засобів JavaFX: кваліфікаційна робота бакалавра спеціальності 121 «Програмне забезпечення систем» / наук. керівник Г.П. Коломоєць. Запоріжжя : ЗНУ, 2025. 74 с.

Метою роботи є вивчення алгоритмів стеганографії та розробка JavaFx застосунку, що реалізує приховування користувацької інформації у файлах зображень.

У ході дослідження було проведено теоретичний аналіз сучасних методів стеганографії та обґрунтовано доцільність використання LSB-підходу для вбудовування даних у PNG-зображення. У процесі реалізації застосовано бібліотека JavaFX для створення графічного інтерфейсу. Завдяки FXML, CSS і Scene Builder було досягнуто гнучкості, візуальної привабливості інтерфейсу. Забезпечено обробку типових помилок та реалізовано механізм логування. Результатом стала повнофункціональна десктопна програма для приховання даних, що демонструє високу стабільність та потенціал для подальшого вдосконалення.

Ключові слова: *стеганографія, JavaFX, FXML, Scene Builder, LSB, графічний інтерфейс, приховання даних*.

ABSTRACT

Pages — 74.

Figures — 27.

Listings — 18.

Tables — 1.

References — 20.

Sobko B.S. Development of an application for steganography in image files using JavaFX tools: bachelor's thesis in specialty 121 “ Software Engineering” / scientific supervisor H.P. Kolomoiets. Zaporizhzhia: ZNU, 2025. 74 p.

The aim of the work is to study steganography algorithms and develop a JavaFx application that implements the hiding of user information in image files.

In the course of the study, a theoretical analysis of modern methods of steganography was carried out and the feasibility of using the LSB approach to embed data in PNG images was substantiated. In the process of implementation, the JavaFX library was used to create a graphical interface. Thanks to FXML, CSS and Scene Builder, the flexibility and visual appeal of the interface was achieved. We provided processing of typical errors and implemented a logging mechanism. The result is a fully functional desktop program for hiding data that demonstrates high stability and potential for further improvement.

Keywords: *steganography, JavaFX, FXML, Scene Builder, LSB, graphical interface, data hiding*.

ЗМІСТ

ВСТУП 8

1 ВИКОРИСТАННЯ СТЕГАНОГРАФІЇ ДЛЯ ПРИХОВАНОЇ ПЕРЕДАЧІ ПОВІДОМЛЕНЬ 12

1.1 Аналіз предметної області стеганографії 12

1.2 Класифікація методів стеганографії 13

1.3 Просторові методи 14

1.4 Частотні методи 16

1.5 Морфологічні методи 18

1.6 Стеганографія на основі GAN / машинного навчання 20

1.7 Порівняльна таблиця стеганографічних алгоритмів 22

1.8 Приховування даних у графічних файлах різного типу 23

1.9 Аналіз аналогічних програмних рішень та бібліотек 24

1.10 Постановка завдання (вибір PNG, LSB) 27

2 ВИВЧЕННЯ ЗАСОБІВ ПЛАТФОРМИ JavaFX. 28

2.1 Поняття GUI, базові принципи 28

2.2 JavaFX 29

2.3 Переваги JavaFX 31

2.4 Граф сцени JavaFX 32

2.5 Основні компоненти JavaFX (віджети, контейнери, лейаути) 34

2.6 CSS-стилізація GUI 37

2.7 Обробка подій у JavaFX 38

2.8 Використання FXML та JavaFX Scene Builder 40

2.9 Загальні вимоги до застосунку 41

3 Проектування та конструювання застосунку для стеганографії у файли зображень із використанням засобів JavaFX 44

3.1 Архітектура проекту 44

3.2 Програмна реалізація 46

3.3.1 Реалізація прикладної частини 46

3.3.1.1 Опис класів 48

3.3.2 Реалізація графічної частини 64

3.3 Демонстрація готового застосунку 66

3.4 Висновки до розділу 70

ВИСНОВКИ 71

СПИСОК ВИКОРИСТАних джерел 72

ВСТУП

Актуальність теми

У сучасному світі, де кожна наша дія в мережі залишає цифровий слід, важливість приховування даних виходить на передній план. Методи шифрування, що давно стали стандартом, уже сприймаються як щось буденне: майже кожен користувач знайомий з HTTPS-з'єднанням, мобільні застосунки обмінюються інформацією через усілякі крипто-протоколи, а обмін ключами здійснюється через асиметричні механізми, які дозволяють сторонам безпечно узгодити спільний секрет. Проте простого шифрування часом виявляється недостатньо: якщо за Вами пильно спостерігають, то помітивши сам факт передачі зашифрованого файлу — можна вже привернути небажану увагу. У результаті все більш актуальними стають способи «ховати» інформацію так, щоб навіть не виникало питання про її існування.

Однією з найвідоміших технік приховування є стеганографія — метод, що дає змогу «розчинити» секрет у пікселях зображення або в потоках звуку та відео. Замість того щоб надсилати зашифрований документ, його частинки вбудовують у найменш значущі біти цифрового файлу, і зовні це має вигляд звичайного фото або ролика. Такі прийоми дедалі частіше застосовують політики, військові, журналісти, активісти та бізнес компанії, що прагнуть захистити стратегічні дані від конкурентів. В епоху, коли цифрова реальність стрімко стає «прозорою», багатошарові підходи до захисту і приховування даних перетворюються на запоруку не тільки комерційної або національної безпеки, а й фундаментального права на приватність і анонімність.

Мета дослідження

Метою роботи є вивчення алгоритмів стеганографії та розробка JavaFx застосунку, що реалізує приховування користувацької інформації у файлах зображень.

Завдання дослідження

Для досягнення поставленої мети необхідно вирішити такі завдання:

- ознайомитися з базовими принципами стеганографії та класифікувати основні методи приховування інформації;
- дослідити особливості реалізації стеганографії в зображеннях;
- проаналізувати можливості платформи JavaFX для створення графічного інтерфейсу користувача та оцінити її переваги в навчальних проєктах;
- розробити архітектуру застосунку, що дозволяє вбудовувати приховану інформацію в зображення та зчитувати її;
- створити графічний інтерфейс із використанням JavaFX, FXML та Scene Builder;

Об'єкт дослідження

Об’єктом дослідження є процес прихованого передавання інформації за допомогою цифрових медіафайлів.

Предмет дослідження

Предметом дослідження є розробка JavaFx застосунку для приховування текстової інформації в файлах зображень.

Методи дослідження

У процесі виконання роботи було застосовано такі методи:

- аналіз літературних джерел, наукових публікацій і сучасних програмних рішень у галузі стеганографії;
- використання інструментів JavaFX, включно з CSS-стилізацією, FXML-розміткою та обробкою подій;
- експериментальне моделювання процесу приховування і вилучення повідомлень;
- апробація застосунку на різних зображеннях для перевірки точності й ефективності роботи алгоритмів.

Практичне значення роботи

Застосунок має практичну цінність як зручний інструмент для прихованого передавання даних у зображеннях. Він може використовуватися

як для навчання стеганографії, так і в реальних ситуаціях, де важливо замаскувати сам факт наявності інформації. Інтерфейс на JavaFX забезпечує простоту та легкість використання застосунку у багатьох умовах.

Апробація одержаних результатів

Результати дослідження були представлені на XVIII науково-практичній конференції студентів, PhD, докторантів і молодих вчених Запорізького національного університету «Молода наука–2025» [1].

Глосарій

Стеганографія — метод приховання інформації в цифрових об’єктах так, щоб сама її присутність залишалась непоміченою.

JavaFX — платформа для створення графічних інтерфейсів на мові Java, яка підтримує використання FXML, CSS і Scene Builder.

FXML — формат XML-розмітки, що дозволяє визначати інтерфейс користувача окремо від бізнес-логіки в застосунках JavaFX.

Кодування повідомлення — процес вбудовування прихованих даних у цифровий файл (наприклад, зображення).

Декодування повідомлення — процес зчитування та відновлення інформації, прихованої в цифровому медіа.

Бітова глибина — кількість бітів, що використовується для представлення кольору кожного пікселя, важливий параметр у стеганографії.

GUI (Graphical User Interface) — графічний інтерфейс користувача, що дозволяє керувати застосунком за допомогою візуальних елементів.

Цифровий слід — дані, які залишає користувач під час взаємодії з цифровими речами: перегляд сайтів, листування, транзакції та інші дії, що можуть бути збережені або проаналізовані.

Шифрування — процес перетворення зрозумілих (відкритих) даних у спеціальний закодований вигляд, доступний лише тим, хто володіє відповідним ключем.

Криптопротокол — набір правил, що визначають, як саме дані мають шифруватися, передаватися та перевірятися між сторонами у цифровому середовищі.

Мінімально значущий біт (LSB) — найменш важлива частина байту цифрових даних, яку можна змінити без значної втрати якості, що робить її зручною для вбудовування прихованої інформації.

Багатошаровий захист — стратегія безпеки, яка передбачає використання кількох незалежних методів захисту одночасно, таких як шифрування, стеганографія, автентифікація тощо.

Цифрова анонімність — здатність користувача діяти в інтернеті без розкриття своєї особистості чи місцезнаходження, зберігаючи приватність.

Файл-носій (синоніми: покривне зображення, стегоносій) — це оригінальний файл зображення без прихованих користувацьких даних.

Стегооб'єкт (синонім: стегофайл) — це файл зображення з прихованими користувацькими даними (шифрованими або ні).

1 ВИКОРИСТАННЯ СТЕГАНОГРАФІЇ ДЛЯ ПРИХОВАНОЇ ПЕРЕДАЧІ ПОВІДОМЛЕНЬ

1.1 Аналіз предметної області стеганографії

Стеганографія – це спосіб прихованого написання, при якому секретне повідомлення (може бути як текстом так й різноманітними файлами) вбудовується у звичайні носії (зображення, аудіо, відео, текст і т.д.) таким чином, що сама його наявність у файлі — непомітна для стороннього спостерігача [2]. У випадку цифрових медіа «носієм» може бути будь-який файл (графічний, аудіо, відео), а процес приховування зводиться до маніпуляцій над його даними. Основною задачею при використанні стеганографії є забезпечення невидимості (неможливість візуального чи простого статистичного відокремлення стегооб’єкта від оригіналу). Наприклад, є стегооб'єкт, який неможливо відрізнити від носія стегоінформації, але якщо дуже точно змінити окремі пікселі в ній (якийсь відтінок одного або декількох кольорів), то середньостатистична людина — просто не помітить нічого, водночас, цей же файл можна буде «прочитати» й витягнути необхідну інформацію за допомогою спеціального програмного забезпечення (див. Рис. 1).

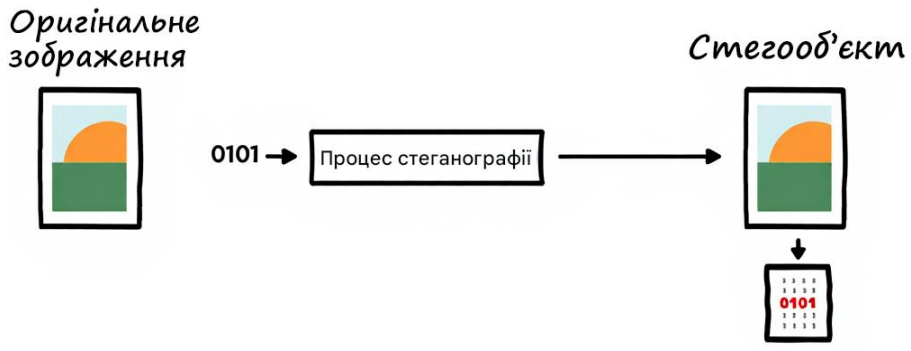


Рисунок *1 — Схема використання стеганографії.*

Стеганографія розглядається поряд із криптографією та водяними знаками як засіб безпеки інформації. При цьому криптографія захищає зміст повідомлення, а водяні знаки – інтегритет (цілісність) та приналежність даних. Стеганографія ж прагне одночасно заховати дані і приховати сам факт їх передачі. Між цими підходами часто спостерігається взаємодія: наприклад, перед вбудовуванням секретне повідомлення може бути зашифроване криптографічним алгоритмом для додаткової безпеки. Загалом, ця область поєднує знання про обробку сигналів, теорію інформації та алгоритми цифрових перетворень [2].

1.2 Класифікація методів стеганографії

Класично методи стеганографії в зображення класифікують на просторові та частотні. А вже більш сучасно, додають ще дві групи: фрактальні та Generative Adversarial Networks (GAN) (див. Рис. 2) [2].

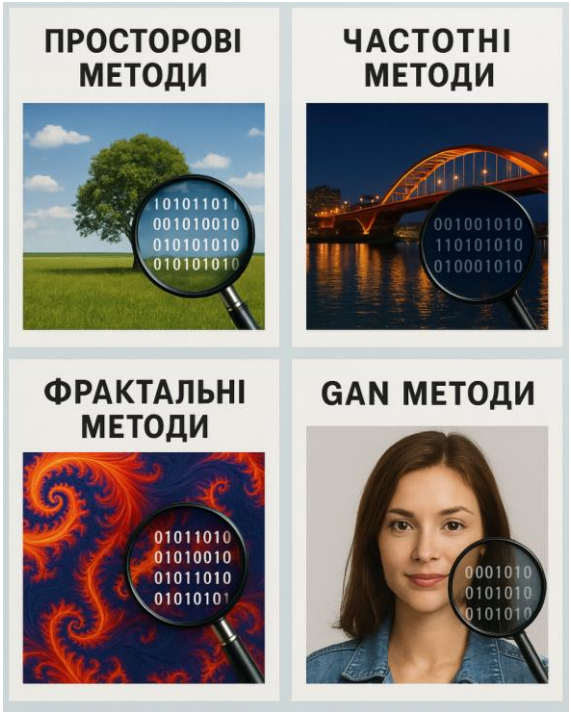


Рисунок *2 — Класифікація видів стенографії за типом носія.*

1.3 Просторові методи

LSB алгоритм

LSB (Least Significant Bit) – це найпростіша і найпоширеніша техніка стеганографії в просторовій області. У цьому методі біти секретного повідомлення просто замінюють наймолодші біти піксельних значень (RGB) покривного зображення. Наприклад, якщо у байті зеленого кольору пікселя молодший біт був 0, а потрібно вбудувати «1», то він змінюється з 0 на 1 (див. Рис. 3). Така операція непомітна для людського ока при помірному обсязі даних, що додаються до носія. Головна перевага LSB – простота впровадження та швидкість роботи. Легко вбудувати й витягти дані, без потреби в перетвореннях. Однак LSB дуже вразливий до аналізу: статистичні методи, такі як тест Чі-квадрат/Chi-square test (статистичний тест, що перевіряє, наскільки розподіл найменш значущих бітів відрізняється від очікуваного випадкового розподілу.) або RS-аналіз (метод, що порівнює зміни у групах пікселів до і після інверсії LSB, щоб виявити приховану інформацію за зміною статистичних властивостей) — швидко виявляють аномалії в розподілі бітів пікселів [5]. Також LSB-сигнал легко зруйнувати – звичайне стиснення JPEG чи фільтрація сильно пошкоджує найменші біти.

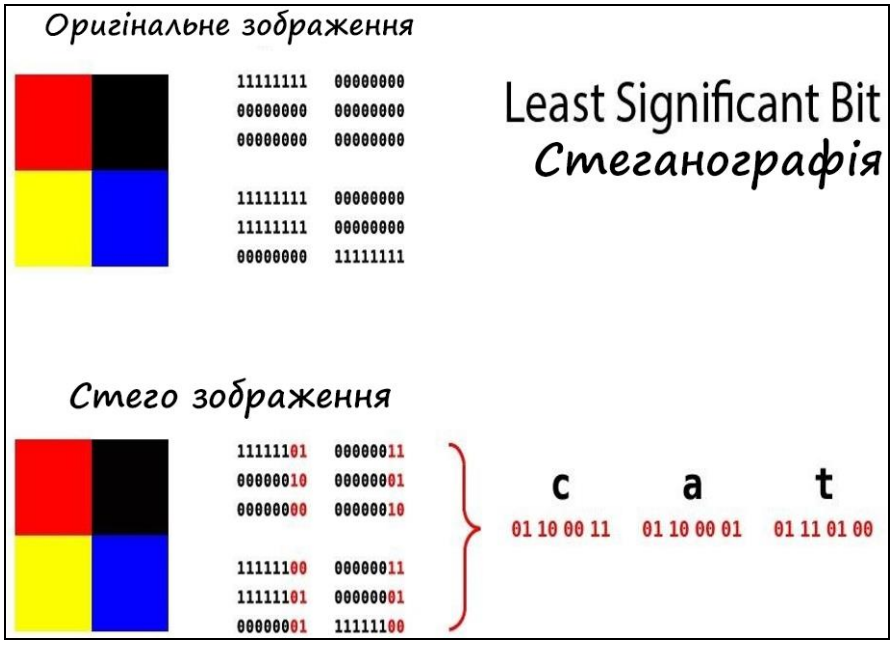


Рисунок *3 — Алгоритм LSB.*

PVD алгоритм

Інший просторовий алгоритм – PVD (Pixel Value Differencing). У PVD-підході аналізується пара суміжних пікселів і змінюється різниця у значеннях кольорових складових (RGB). Ідея така: якщо різниця між двома пікселями велика (наприклад, у ділянках з сильним контрастом чи на краях), то можна безпечно відхилити значення пікселів трохи більше і сховати більше бітів. Якщо ж пікселі дуже схожі (різниця маленька, області без деталізації), то змінювати їх треба обережно, щоб не створити помітні спотворення. Алгоритм PVD групує пікселі попарно, обчислює їхню різницю та на основі неї визначає, скільки бітів вбудувати. Наприклад, якщо початкова різниця дорівнює 5, можна вбудувати 3 біти секретної інформації шляхом корекції цієї різниці, а якщо різниця 1 – то лише 1 біт. Завдяки цьому PVD дозволяє вбудувати більше інформації в текстурні або крайові області, тоді як LSB однаково змінював би і гладкі ділянки (що призводить до помітнішого шуму). При порівнянні двох методів — PVD-ховає більше даних, при збереженні якості стегозображення, бо «уточнює» дані так, щоб максимізувати приховання у вже нестабільних областях, забезпечуючи кращу невидимість і стійкість, ніж просте LSB (див. Рис. 4) [6].

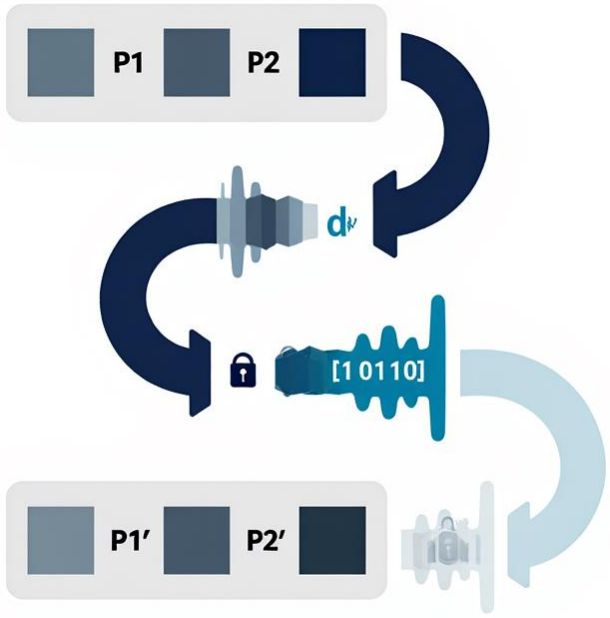


Рисунок *4 — Алгоритм PVD.*

Існують й інші методи просторової домену: перепис стеганографічних бітів у окремі кольорові канали, адаптивні LSB (що враховують текстуру зображення), методи зсуву гістограми та багато інших. Але базовими є LSB та PVD. Загалом, у просторовому домені алгоритми зазвичай є дуже швидкими (минає цикли на перетворення), мають високу ємність (багато можливих біт для запису), але низьку стійкість до обробки зображення і стискання.

1.4 Частотні методи

DCT алгоритм

У частотному домені вбудовування відбувається у перетворені коефіцієнти образу чи сигналу. Найвідоміший приклад – DCT-стеганографія. Цей метод використовує Дискретне Косинусне Перетворення, яке лежить в основі JPEG. Оригінал зображення розбивають на блоки 8×8, перетворюють кожен блок у DCT-коефіцієнти за допомогою дискретного косинусного перетворення, а потім вбудовують секретні біти в їхні молодші біти (див. Рис. 5). На практиці уникають змін у DC-компоненті (коефіцієнті низької частоти) і у дуже малих коефіцієнтах (0 чи 1), щоб не внести очевидних змін. За такого підходу стегозображення виходить стійкішим до подальшого JPEG-стиснення, оскільки дані «розкидані» по DCT-блокам. Водночас DCT-методи зазвичай менш ємні, ніж LSB, і потребують додаткових обчислень (DCT/IDCT) [7].

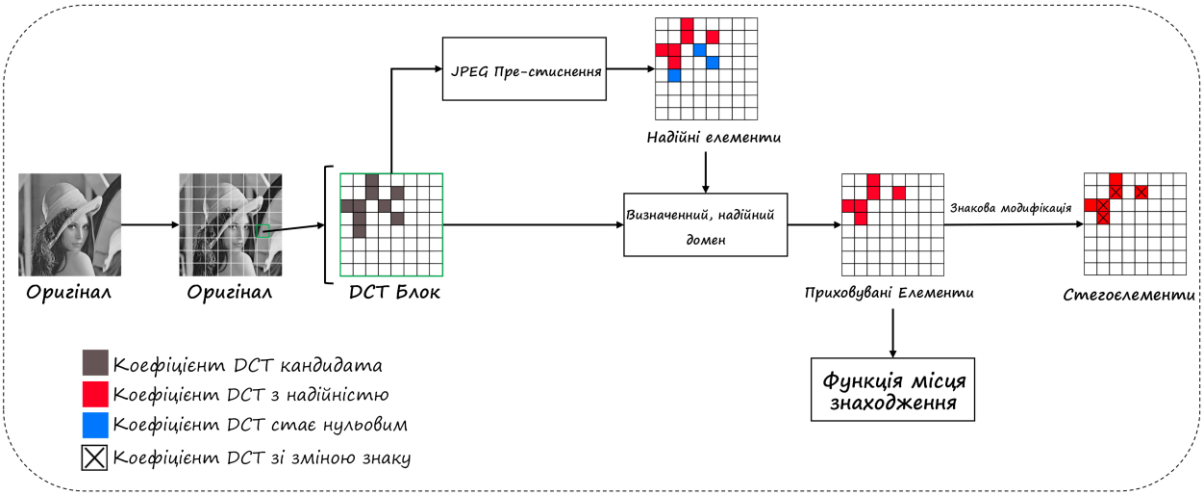


Рисунок *5 — Алгоритм на основі DCT.*

DWT алгоритм

DWT-стеганографія (Discrete Wavelet Transform) спочатку розкладає зображення на субдіапазони за допомогою дискретного вейвлет-перетворення, яке застосовує вейвлет-фільтри для виділення низько- та високочастотних компонент по горизонталі й вертикалі: LL, LH, HL, HH (деталі малого і великого масштабів). Діаграма першого рівня містить чотири субдіапазони: низькочастотний LL та три високочастотні. Оскільки LL-частина містить основні деталі зображення, вбудовування в неї робить стегозображення дуже стійким до багатьох атак (сильна редагування дає змогу відновити точніше), але може помітно спотворювати основний вміст. Тому зазвичай секретні біти впроваджують у середні/високі частоти (LH, HL, HH) – при цьому бачимо, що саме на цих компонентах знаходяться краї та текстури. Оскільки людське око менш чутливе до дрібних змін на краях, вбудовування в високочастотні вейвлет-коефіцієнти створює мало видимого артефакту. DWT-методи часто мають ліпшу стійкість до фільтрацій, оскільки передача по вейвлет-домену є багаторівневою і дозволяє розподіляти повідомлення (крім того, існують комбіновані схеми DWT+DCT). В цілому, DWT-підходи забезпечують високу невидимість, поступаючись LSB за ємністю, але випереджаючи його за стійкістю до компресії [5].

У методах, що базуються на дискретному вейвлет-перетворенні (DWT), зображення розкладається на кілька частотних піддіапазонів. Зокрема:

LH — зміни по вертикалі (горизонтальні краї),

HL — зміни по горизонталі (вертикальні краї),

HH — діагональні краї та текстури.

Саме в ці області (високочастотні субдіапазони) зазвичай вбудовується приховане повідомлення (див. Рис. 6), тому що людське око менш чутливе до дрібних змін у деталях та на краях зображення. Завдяки цьому прихована інформація не помітна візуально.

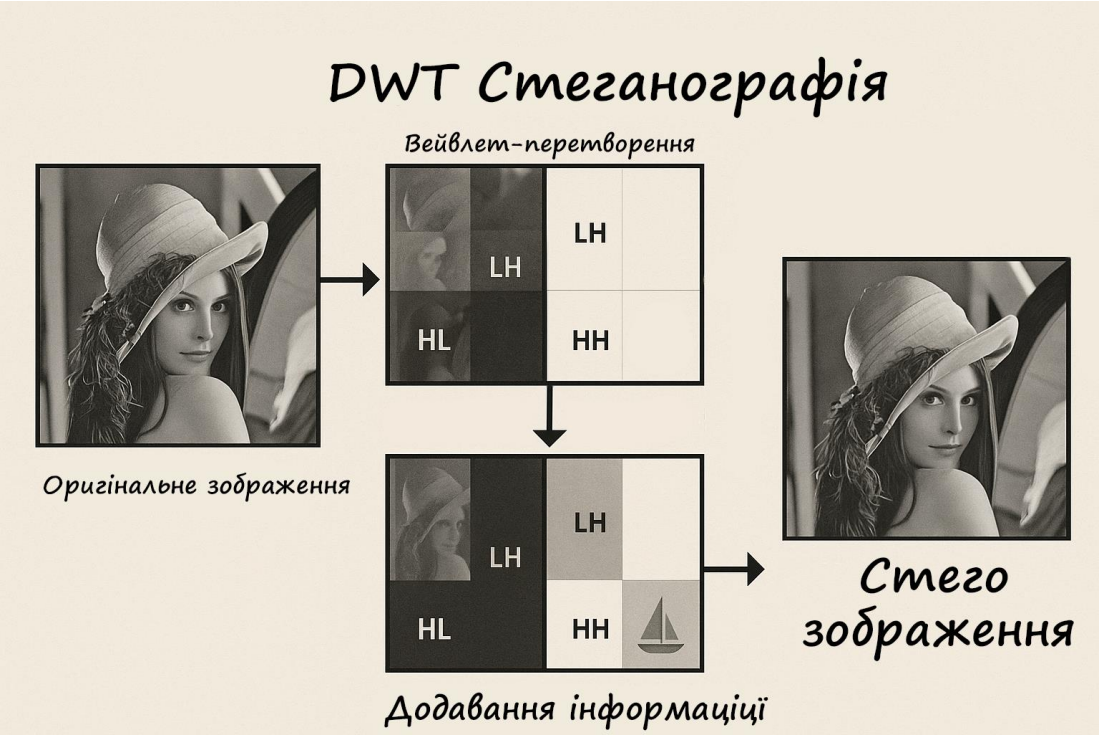


Рисунок 6 — *Області для вбудовування.*

1.5 Морфологічні методи

Крім простих схем, є спеціалізовані підходи на базі фракталів та морфології. Фрактальні методи будують покривні носії за допомогою фрактальних перетворень або генерацій множин Мандельброта–Юлії.

Наприклад, **Alia та Suwais (2020) [**9**]** запропонували фрактальний стеганографічний метод, у якому секретні біти вбудовуються через керування параметрами фрактального коду: вибір типу фракталу (Мандельброт або Юлія), кількість ітерацій та секретний ключ однозначно визначають послідовність координат фрактальної множини, що кодують інформацію. Інший метод поєднує фрактальну компресію секретного зображення та DWT-перетворення покривного: фрактальний код секрету вбудовується у вейвлет-коефіцієнти покривного. Такі схеми зазвичай демонструють високу стійкість (до шуму, нечіткості), але можуть бути обчислювально-дорогими.

Морфологічні стеганографічні алгоритми аналізують структурні властивості зображення. Наприклад, в одному з методів спочатку покривне зображення перетворюють у «морфологічне представлення», яке складається з коефіцієнтів, що відображають морфологічні ознаки (градієнти, форму, текстуру). Потім секретні біти вбудовуються з використанням перестановок та матричного кодування. Завдяки цьому повідомлення розподілено за неявними ознаками, що робить стенографію більш непомітною і стійкою. Дослідження показали, що морфологічні схеми дають дуже малу візуальну деградацію стегозображення порівняно з простим LSB. Загалом, фрактальні і морфологічні методи відносяться до розвинутих доменних підходів – вони складніші, але обіцяють вищі характеристики невидимості та безпеки (див. Рис. 7).

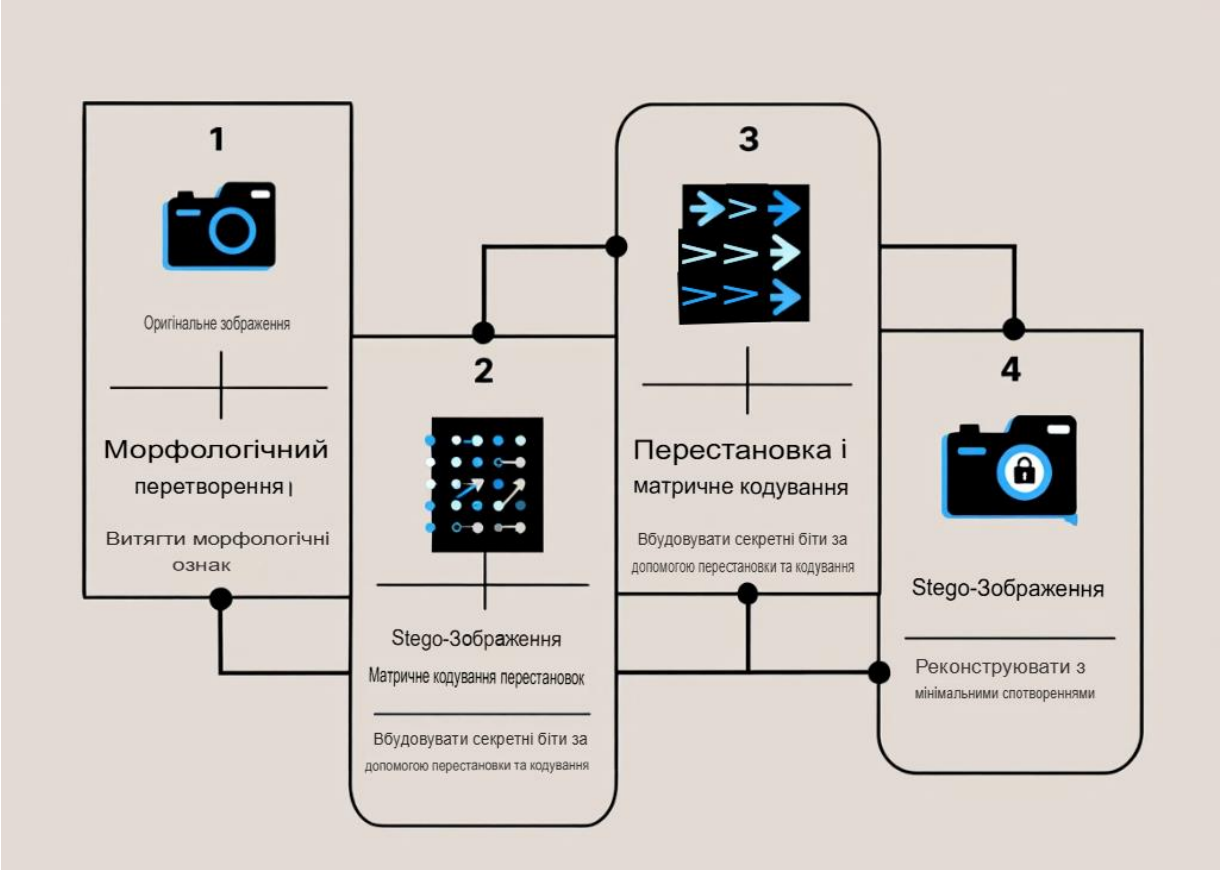


Рисунок *7 — Наглядне зображення циклу стеганографії у зображення методом морфології.*

1.6 Стеганографія на основі GAN / машинного навчання

Сучасні тренди стеганографії пов’язані з глибинним навчанням і нейромережами. У цьому підході генеративні моделі навчаються приховувати дані чи генерувати відповідні стегоносії. Найвідоміша реалізація – SteganoGAN, де використовується архітектура GAN (Generative Adversarial Network) для автоматичного створення стегозображень [10]. У таких системах генератор імітує роботу стеганографа, а дискримінатор намагається розпізнати, чи є в зображенні прихована інформація. В результаті навчання генератор виробляє зображення, що практично не відрізняються від оригінальних і водночас містять дані. Практика показує, що GAN-підходи можуть досягати дуже високої ємності при збереженні якості: наприклад, SteganoGAN

забезпечував 4.4 біта на піксель вбудовування і при цьому «обминав» сучасні методи детекції. Алгоритми стеганографії, побудовані на основі генеративних змагальних мереж (GAN), дозволяють значно покращити **якість стегозображень**. Зокрема, такі методи забезпечують **вищі значення PSNR (пікове відношення сигнал/шум)** та **SSIM (структурна подібність зображень)** у порівнянні з традиційними підходами, як-от вбудовування в LSB або частотну область (DCT). Це означає, що стегозображення виглядають практично ідентично до оригіналу навіть при високому навантаженні даними (див. Рис. 8).

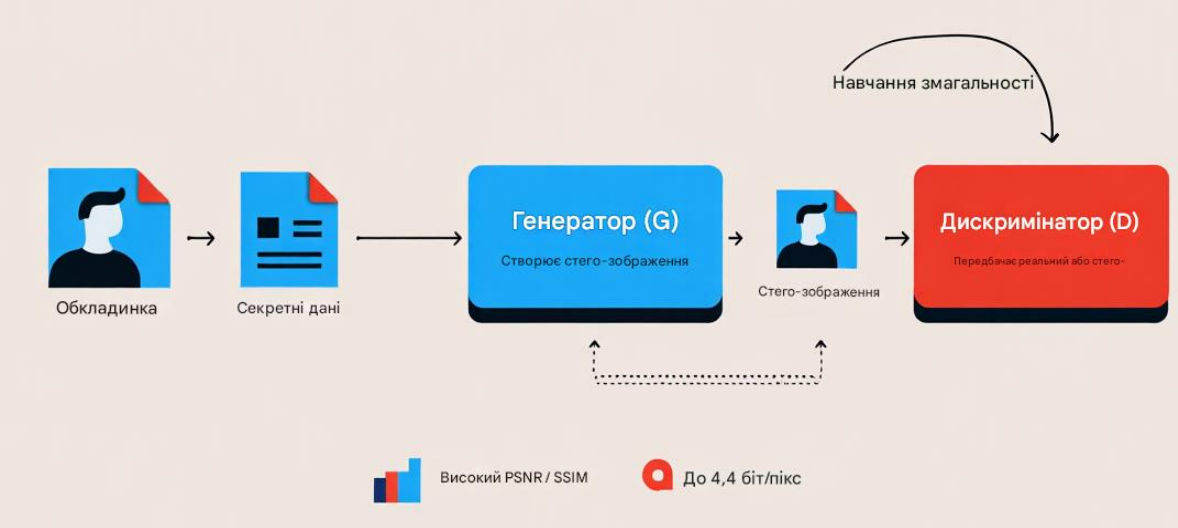


Рисунок *8 — GAN-процес.*

Окремо від GAN, для стеганографії вивчають й інші нейромережеві структури: автоенкодери можуть навчатися кодувати дані в латентний простір зображення (тобто у приховане внутрішнє представлення, яке модель використовує для відновлення зображення), а рекурентні та конволюційні (згорткові) мережі – ховати текст або контрольні знаки. Проте наразі генеративні мережі (GAN) є найпрогресивнішим напрямом: вони дозволяють досягати високої стійкості до стеганалізу та великих обсягів даних. Слід зазначити, що таке вбудовування потребує значних обчислень (навчання

мереж), але забезпечує високу непомітність і динамічну адаптацію під особливості конкретного носія [3].

1.7 Порівняльна таблиця стеганографічних алгоритмів

Незалежно від конкретної техніки, алгоритми стеганографії оцінюють за загальними критеріями: ємність (скільки даних можна сховати), перцептуальна прозорість (відсутність помітних змін у вигляді об'єкта після приховування даних), стійкість (приховані дані мають зберігатися при обробці чи стисненні) і швидкість (обчислювальна ефективність). У цілому, існує компроміс: збільшення ємності чи стійкості — ускладнює збереження прозорості і навпаки. Нижче подано короткий опис особливостей основних методів (результати підведено у табл. 1):

LSB (просторова область): дуже висока ємність і швидкість впровадження; проста реалізація. Однак низька стійкість – будь-яке оброблення зруйнує віддані біти – і вразливість до статистичних атак. У lossless-форматах (BMP, PNG) вбудовані біти зберігаються повністю, але при будь-якому втратному стисненні (JPEG) секрет легко знищується.

PVD (просторова): середня ємність (більша, ніж LSB на контрастних ділянках), непогана невидимість за рахунок адаптивного розміщення; краща стійкість до простого аналізу (очевидно, статистичні тести важче виконати). Але складніша у впровадженні, трохи менша швидкість, ніж LSB.

DCT/DWT (частотна область): помірна ємність; висока стійкість до стиснення (особливо JPEG для DCT або звукові фільтри) і хороша невидимість. Натомість повільніші (потрібні перетворення), а їх дія на ємність обмежена (занадто сильні зміни спотворять результат).

Морфологічні/фрактальні: зазвичай нижча ємність (через складні перетворення), але дуже висока невидимість і відмінна стійкість до аналізу – оскільки зміни розподілені по складних структурних шаблонах. Ці методи повільніші та важчі в реалізації.

GAN/нейромережеві: висока прихована ємність і видима якість (PSNR, SSIM); найбільша стійкість до детекції сучасними методами. Проте повільні в розгортанні (потребують навчання) і потребують багато ресурсів.

Зведене порівняння характеристик різних підходів наведено в Таблиці 1. Воно ілюструє, що немає універсального алгоритму: вибір залежить від пріоритетів (наприклад, швидкість/ємність проти безпеки).

Табл. 1 — *Порівняльна характеристика стеганографічних методів за ємністю, невидимість, стійкістю та швидкістю.*

Метод	**Ємність**	**Невидимість**	**Стійкість**	**Швидкість**
LSB-PNG	Висока	Висока	Низька	Дуже висока
LSB-JPEG	Середня	Середня	Середня	Висока
PVD	Середня	Висока	Середня	Висока
DCT-JPEG	Низька	Висока	Висока	Середня
DWT	Низька	Дуже висока	Дуже висока	Низька
Фрактальні	Низька	Дуже висока	Дуже висока	Дуже низька
GAN	Висока	Дуже висока	Дуже висока	Низька

1.8 Приховування даних у графічних файлах різного типу

Графічні файли – один з найпоширеніших носіїв секретних даних. Формат PNG – популярний безвтратний формат (з підтримкою альфа-каналу) [3], тому він ідеально підходить для простих стеганографічних

методів (напр. LSB). При впровадженні в PNG-жодні зміни не втрачаються через стиснення: заміна LSB у пікселях фізично змінює дані, а при записі в PNG вони зберігаються повністю. Саме тому PNG часто використовують, як «оптимальний» формат для LSB-методів: він має велику ємність (великий обсяг пікселів) та мінімальне викривлення вбудовування.

JPEG — формат зі стисненням із втратами, в якому зображення розбивається на блоки 8×8 пікселів, перетворюється за допомогою дискретного косинусного перетворення (DCT), а отримані коефіцієнти квантуються для зменшення розміру файлу [2]. Такий підхід стійкіший до стиснення JPEG, але менш ємний – надто велика кількість змін призводить до сильних артефактів. Додатково, у JPEG часто використовують методи кодування парності (парних/непарних коефіцієнтів), щоб мінімізувати помилки при зворотному DCT.

Формат BMP (Bitmap) містить необроблені пікселі (часто 24-бітні), тому його легко використовувати для LSB-методів – фактично будь-які модифікації піксельних значень точно зберігаються і ніяк не змінюються стисненням (BMP не використовує стиснення) [3] . Основний недолік BMP – значний об’єм файлу (багато пікселів) та відсутність будь-якої компресії, але це автоматично збільшує обсяг можливої інформації, що буде додаватися стеганографією.

Таким чином, вибір формату впливає на можливості стеганографії. PNG забезпечує збереження вбудованих бітів і високу невидимість; JPEG вимагає роботи з перетвореними коефіцієнтами; BMP дає найвищу ємність та простоту, але потребує великих носіїв. Всі ці аспекти слід враховувати при розробці інструменту.

1.9 Аналіз аналогічних програмних рішень та бібліотек

Існують кілька готових інструментів для стеганографії. Наприклад, OpenStego – це відкрите Java-рішення, яке дозволяє приховувати будь-які файли всередині графічного зображення (режим Data Hiding) і підтримує

водяні знаки (див. Рис. 9) [11]. Програма має підтримку AES-шифрування секретних даних (AES128/AES256) для захисту і може працювати через GUI та командний рядок. OpenStego використовує рандомізоване LSB-вбудовування, що робить цей процес більш стійким до простих статистичних атак.

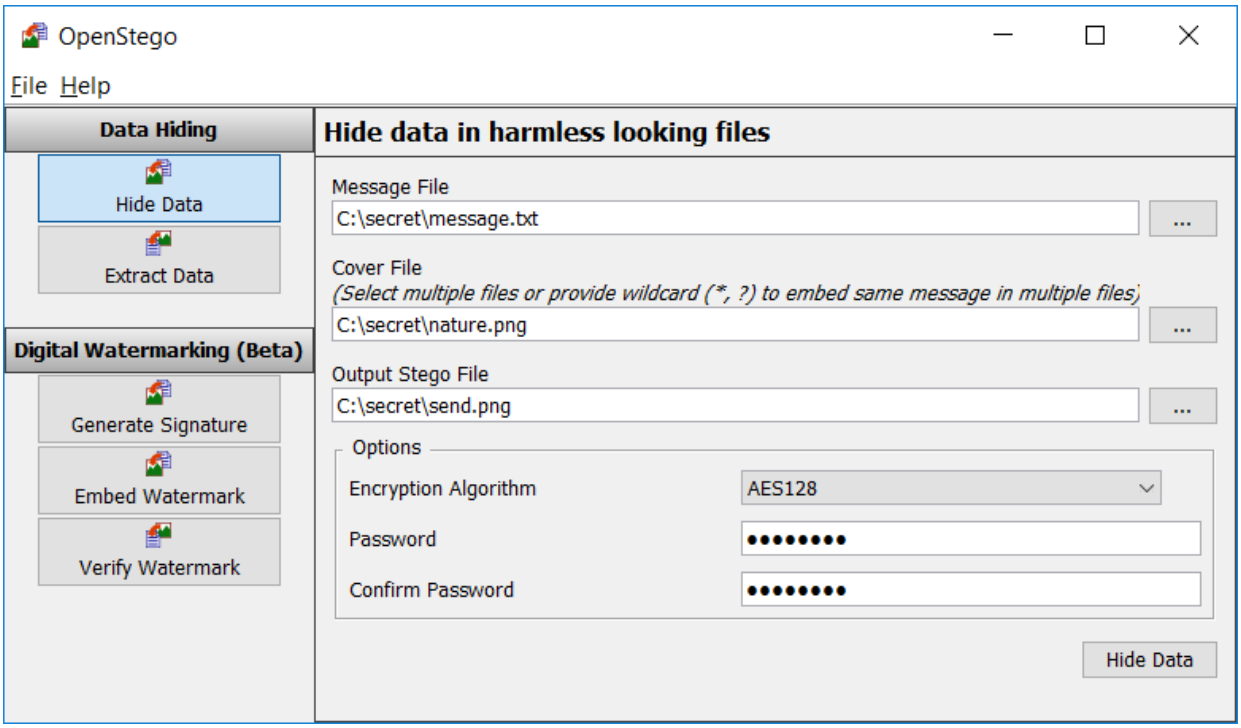


Рисунок *9 — Інтерфейс застосунку OpenStego.*

Ще один приклад – StegIt (JavaFX-додаток, вихідний код на GitHub [12]). StegIt реалізує кодування LSB для 24-бітних PNG, BMP та JPG-файлів; у ньому можна ховати тексти, документи і навіть зображення в інші зображення (див. Рис. 10) [12]. Додатково StegIt стискає повідомлення перед вбудовуванням (щоб підвищити ємність) і застосовує AES-CBC (128-біт) для секретності. У StegIt можна також задати пароль чи хешувати будь-яку картинку як ключ шифру.

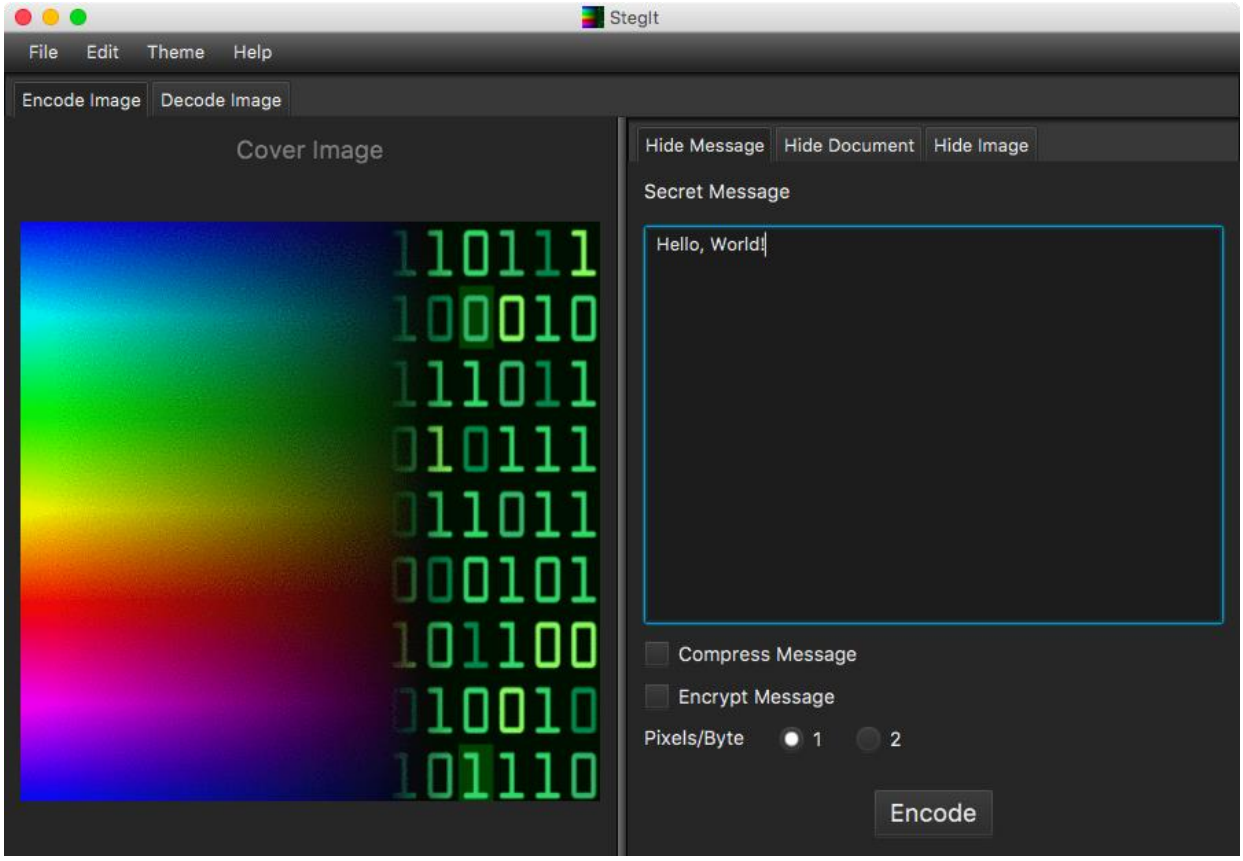


Рисунок *10 — Інтерфейс застосунку StegIt.*

Крім того, існують інші програми: Steghide (командний CLI-інструмент, що працює з JPEG, BMP, WAV, AU та підтримує шифрування), StegoSuite, OutGuess та купа інших. Багато відкритих рішень обмежені певними форматами і досить базові, тож часто розробники писатимуть власний код (напр. на Java з AWT/JavaFX для GUI). Наприклад, вищезазначений GitHub-проект StegIt – це саме саморобна Java-реалізація на LSB.

З практичної точки зору, при розробці власного додатку варто вибирати ті ж інструменти та формати, які підтримують подібні програми. Отже, у роботі за основу взято PNG-формат (через його безвтратність) і метод LSB (через його простоту реалізації та готові приклади в подібних проектах).

1.10 Постановка завдання (вибір PNG, LSB)

Отже, задачі розділу узагальнені:

1. Обрати формат-носія до якого будемо кодувати користувацьку інформацію.
2. Обрати і описати алгоритм для стеганографії.
3. Реалізувати модулі вбудовування та вилучення даних з файлу-носія.
4. Розрахувати максимальний обсяг повідомлення який можливо буде додати до файлу носія.

2 ВИВЧЕННЯ ЗАСОБІВ ПЛАТФОРМИ JAVAFX.

2.1 Поняття GUI, базові принципи

Графічний інтерфейс користувача (GUI) — це прошарок між юзером та програмою, для взаємодії через візуальні елементи: кнопки, меню, поля для вводу, списки, вкладки, рамки і т.д.. На відміну від командного рядка (CLI — Command Line Interface), GUI (Graphical User Interface) забезпечує інтуїтивно зрозумілий спосіб виконання операцій та навігації по функціоналу, що особливо важливо для кінцевого продукту, його конкурентоспроможності й зручного користування (див. Рис. 11) [13].



Рисунок *11 — Наглядна різниця між CLI та GUI.*

У середовищі Java є два основні підходи до створення настільних інтерфейсів: AWT, який використовує вбудовані в операційну систему елементи управління, і Swing, що надає «легковагові» (не створюють окремих вікон на рівні ОС та легші в плані ресурсів) компоненти з однаковим виглядом на різних платформах. Проте з ростом вимог до продуктивності, ефектних

анімацій та гнучкої стилізації класичні бібліотеки виявилися недостатньо зручними для розробки сучасних UI (User Interface).

Щоб задовольнити потреби сучасності було створено платформу JavaFX, яка об'єднала в собі переваги попередників і доповнила їх такими ключовими можливостями, як сцена (граф сцени) для організації вузлів у деревоподібну структуру, підтримка CSS-подібного оформлення, XML-файли FXML для опису інтерфейсу та інтеграцію з візуальним редактором Scene Builder.

2.2 JavaFX

JavaFX — це найновіше покоління Java-бібліотеки для побудови GUI. На відміну від Swing і AWT, JavaFX активно підтримується і розвивається відкритим ком'юніті. JavaFX створювався «з нуля» для сучасних завдань GUI, він володіє низкою сучасних можливостей, які відсутні в Swing та AWT (див. Рис. 12) [15].

Серед ключових технічних особливостей JavaFX:

- Сучасний графічний рушій Prism: JavaFX використовує апаратно-прискорений рендеринг (через OpenGL або Direct3D), що забезпечує вищу продуктивність та плавність анімацій;
- Підтримка CSS-стилізації: Інтерфейс можна стилізувати за допомогою каскадних таблиць стилів (CSS), як у веб-розробці. Це дає можливість легко змінювати вигляд застосунку — наприклад, підлаштувати кольори під потрібний стиль або перемикати теми;
- FXML: JavaFX пропонує декларативний опис інтерфейсу у вигляді XML-файлів (FXML). Це дозволяє розділити структуру UI (FXML) та логіку (Java-код контролерів);
- Архітектура сцени (scene graph): Усі візуальні елементи JavaFX організовано у деревоподібну структуру (граф сцени), де вузли (Nodes) можуть містити дочірні вузли. Такий підхід спрощує

трансформації і анімації (застосування ефекту до контейнера діє на всіх дочірніх елементів);

- Мультимедіа та інтеграція з веб: JavaFX включає вбудовані компоненти для відображення мультимедіа, вбудованого веб-переглядачу WebView та інструменти для розробки багатошарового UI з графікою 3D;
- Автоматичне перерахування властивостей (property binding): JavaFX широко використовує властивості з підтримкою прив'язування (bindings), що спрощує реалізацію реактивного оновлення інтерфейсу при зміні даних.

Внаслідок цього JavaFX є привабливою платформою для нових проєктів GUI: він сучасний, гнучкий, має підтримку зовнішніх інструментів (Scene Builder) і технологій (CSS, FXML) для швидкої розробки високорівневих інтерфейсів.

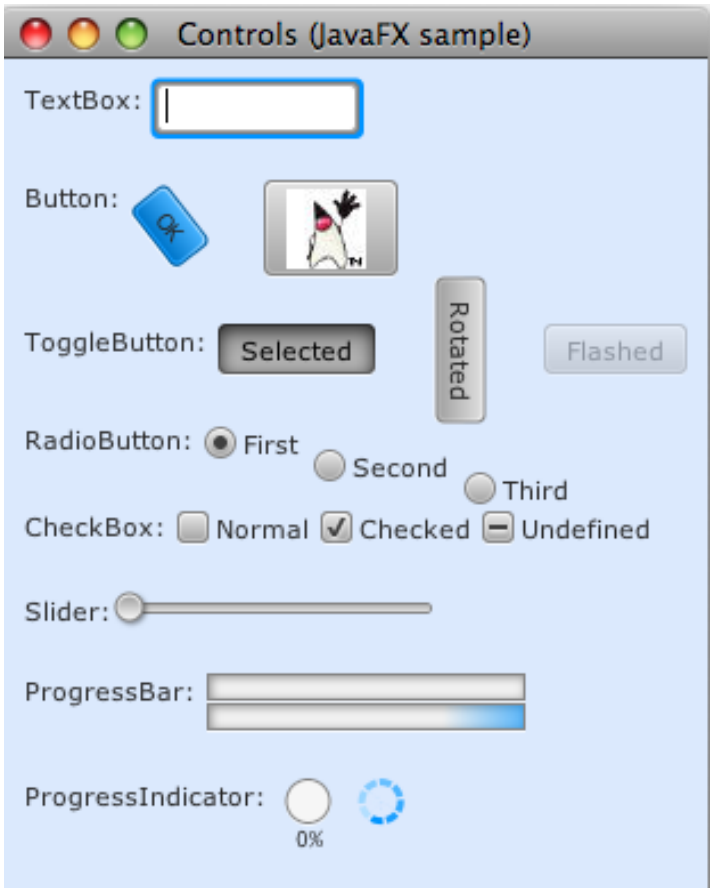


Рисунок *12 — Базовий вигляд інтерфейсу Java FX.*

2.3 Переваги JavaFX

JavaFX пропонує ряд переваг для створення графічних застосунків, особливо порівняно зі Swing/AWT. Найважливіші з них:

- Сучасний підхід до стилізації. JavaFX підтримує стандартні CSS-стилі, що дозволяє відокремити оформлення інтерфейсу від коду. Використання CSS схоже на веб-розробку: можна задавати класи стилів, селектори і властивості (усі властивості, пов’язані зі стилем, починаються з префіксу -fx-). Наприклад, одна стилізація може бути застосована для світлої теми, інша — для темної, і простим завантаженням іншого CSS-файлу весь інтерфейс може змінити колірну схему.
- Графічні ефекти та анімації. Завдяки сценографу та рушію Prism JavaFX підтримує графічні ефекти (тіні, розмиття, заломлення) та апаратно-прискорені анімації. Батьківські вузли в дереві сцени можуть застосовувати трансформації (повороти, масштаб, зсув) до всіх дочірніх елементів разом, що значно спрощує візуальні ефекти. Завдяки цьому графічне представлення оптимізовано на рівні апаратури.
- FXML та Scene Builder. Можливість описувати інтерфейс у XML (FXML) забезпечує чітке розділення «вигляду» та «логіки» програми. Це робить розробку більш гнучкою: наприклад, дизайнери можуть створювати макет у JavaFX Scene Builder (графічному редакторі, що генерує FXML), а програмісти — писати код обробки подій у контролері. Scene Builder інтегрується з IDE (NetBeans, IntelliJ IDEA) та дозволяє «зв’язати» FXML з Java-класом-контролером.
- Багатоплатформеність та нові можливості. JavaFX легко масштабується на різні платформи (Windows, Linux, macOS, а також мобільні платформи через спеціальні рушії). Він підтримує відображення відео, аудіо, а також вбудовує компоненти WebView

(на основі WebKit), що дозволяє створювати багатофункціональні мультимедійні додатки.

- Розвинена система прив’язок (bindings). Завдяки властивостям (properties) і їх прив’язкам розробнику не потрібно вручну оновлювати вигляд при зміні моделі даних. Наприклад, значення тексту в Label може бути прив’язане до властивості моделі, і помінявши цю властивість у коді, інтерфейс оновиться автоматично.

Таким чином, JavaFX поєднує знайому розробникам Java платформу з гнучкістю та візуальними можливостями сучасних веб-технологій. Зокрема, підтримка CSS-стилів та декларативної розмітки інтерфейсу (FXML) дозволяють швидко розроблювати привабливі GUI і легко змінювати їхній зовнішній вигляд без переписування коду. Через всі переваги і позитивні сторони — для розробки застосунку було обрано саме JavaFX.

2.4 Граф сцени JavaFX

JavaFX організовує всі графічні елементи у сценічний граф (scene graph) — ієрархічну структуру, де кожен вузол (Node) має не більше одного батька і може містити нуль або кілька дочірніх вузлів (див. Рис. 13).

Такий підхід (retained mode) схожий на те, як працюють 3D-рушії або векторні графічні бібліотеки: програма описує, що потрібно показати, а сам механізм малювання обходить дерево вузлів і рендерить усе оптимально.

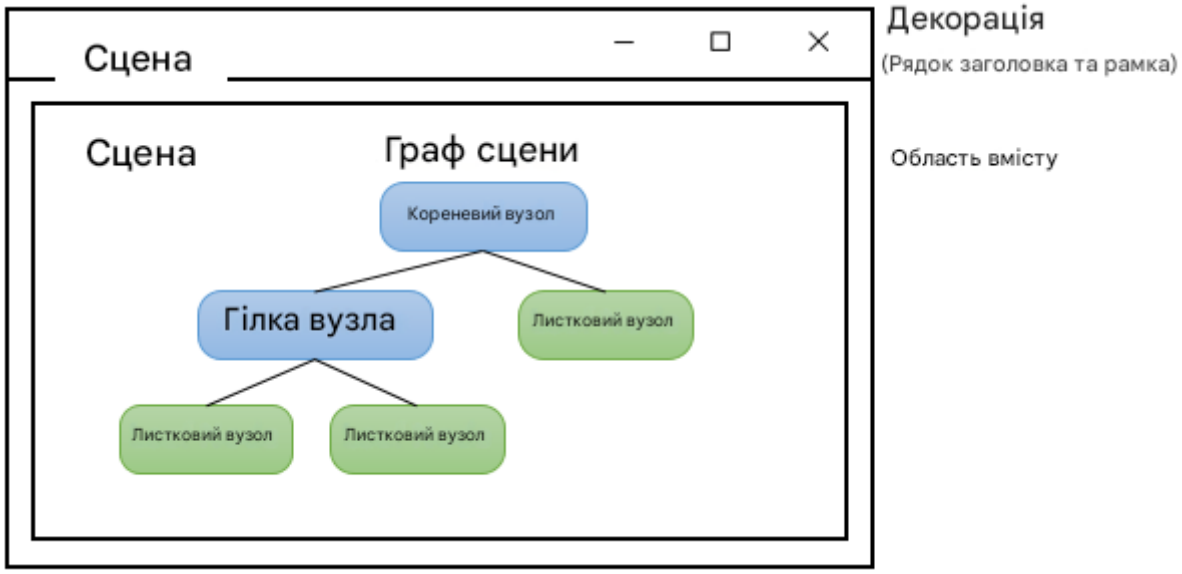


Рисунок *13 — Вміст сцени представлено у вигляді ієрархічного графа вузлів сцени.*

У сценічному графі існують два основні типи вузлів:

Гілкові вузли (branch): мають дочірні елементи. Це різноманітні контейнери і панелі (класи-нащадки Parent, наприклад, Group, Pane, HBox, VBox тощо). Вони відповідають за розташування дочірніх вузлів.

Листові вузли (leaf): не містять дітей. Це, наприклад, прості графічні об’єкти (Rectangle, Text, ImageView, Circle і т.п.) або елементи управління (кнопки, поля введення, мітки). Перший елемент у кожному дереві — кореневий вузол (root), він не має батька. Кожна JavaFX Scene містить один кореневий вузол і, опосередковано — все його піддерево.

Scene Graph — ієрархія вузлів, що репрезентує всі візуальні елементи інтерфейсу. Кожен вузол має ідентифікатор, стиль-клас і область обмеження. За винятком кореневого вузла, усі вузли мають одного батька і можуть мати довільну кількість дітей.

На рівні кореня (Scene) розміщується груповий контейнер Group, який містить кнопку Button (як дочірній вузол). Завдяки такій структурі трансформації та стилі, застосовані до батьківських вузлів, автоматично діють

на дочірні. Наприклад, якщо весь Group повернути або перемістити, всі належні Button та інші вузли змістяться відповідно.

Кожний візуальний елемент JavaFX успадковується від Node (абстрактний клас з пакету javafx.scene) та представлений у вигляді ієрархічної структури-дерева. Елемент може бути контейнером (наприклад, Group, Pane, HBox) або окремим компонентом (Button, Label тощо), при цьому Scene є верхнім контейнером. Через таку ієрархію вплив одного вузла (наприклад, перетворення) автоматично застосовується до його потомків. Цей підхід оптимізує рендеринг та робить можливим апаратне прискорення візуалізації.

2.5 Основні компоненти JavaFX (віджети, контейнери, лейаути)

У JavaFX всі візуальні елементи — це спадкоємці класу Node. Зокрема:

- Контрол-клас (Control) — це клас для побудови елементів взаємодії з користувачем (кнопки, поля введення, списки, таблиці, слайдери). Наприклад: Button, Label, TextField, TextArea, ComboBox, TableView, DatePicker, Canvas (для малювання) тощо. Більшість контролів успадковуються від абстрактного Control, що є підкласом Region, а отже вже має підтримку CSS-стилів та адаптивного розміщення (див. Рис. 14).
- Панель (Pane/Container) — це вузол-контейнер для інших вузлів. Панелі відповідають за компонування своїх дітей. Основні приклади: Pane (вільне позиціонування), Group (без автоматичної перебудови розміщення), HBox/VBox (горизонтальне/вертикальне розташування), BorderPane (розміщення по «південь», «схід», «захід», «центр», "північ"), GridPane (табличне розташування в сітці), StackPane (накладання один на один), тощо (див. Рис. 15).

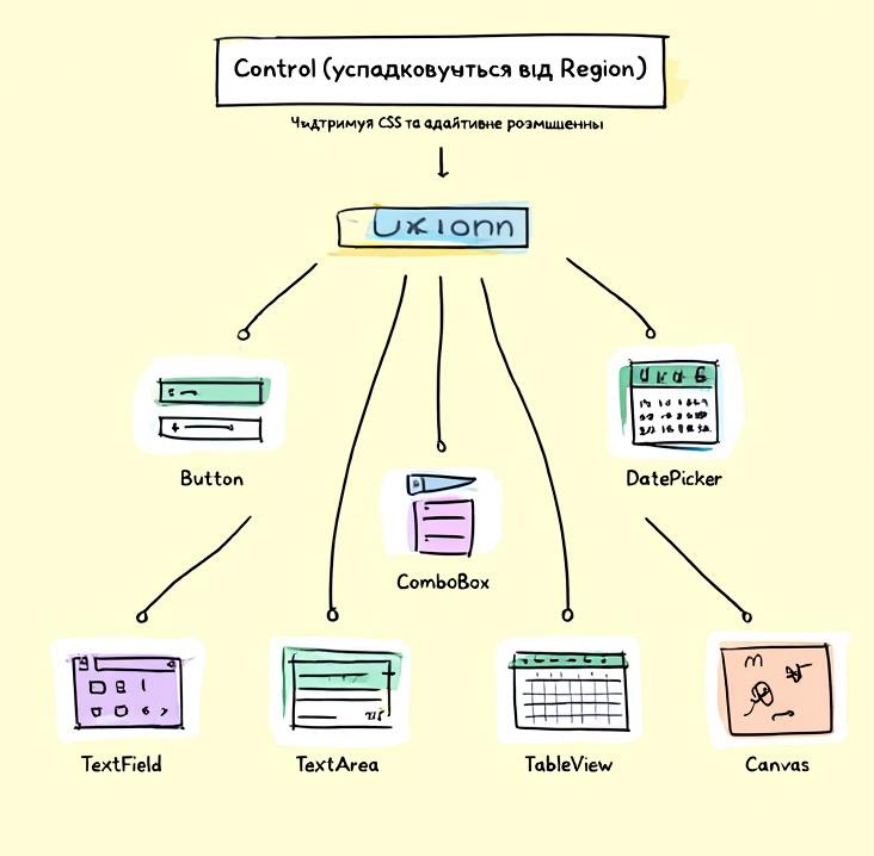


Рисунок *14 — Схема-діаграма успадкувань від класу Control.*

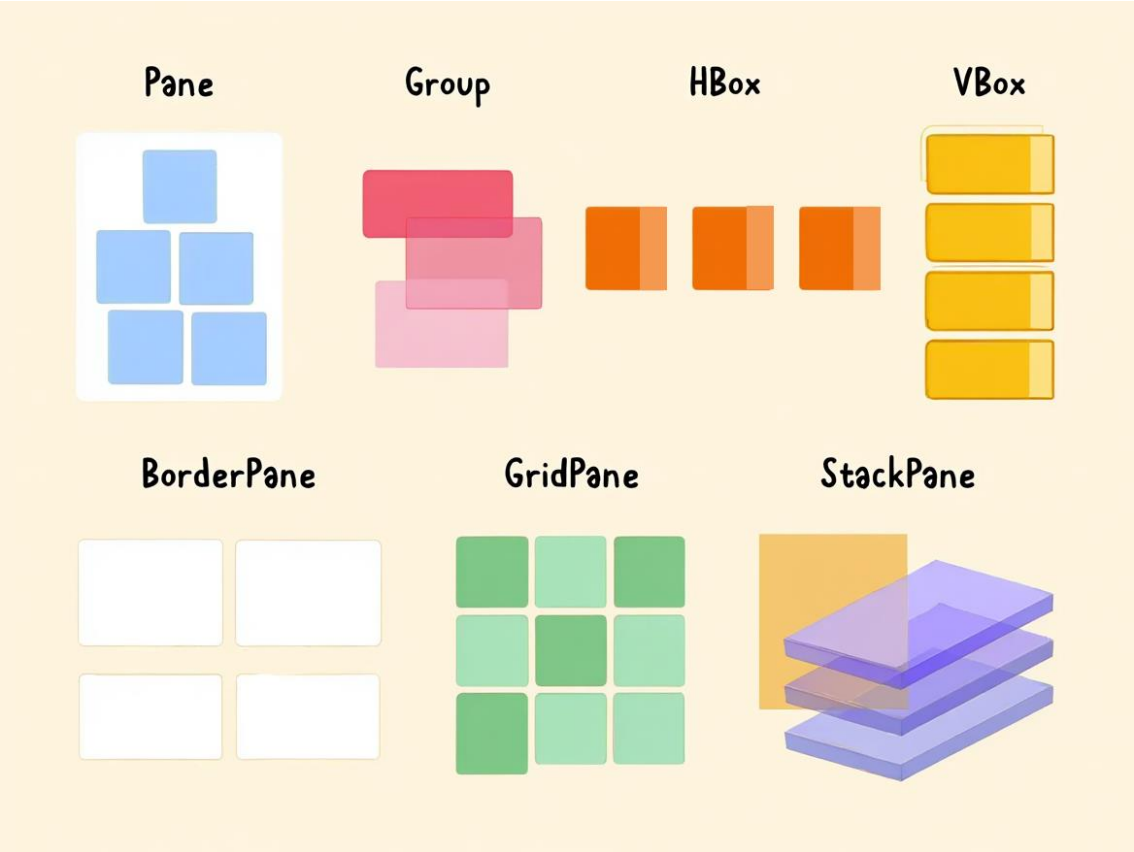


Рисунок *15 — Візуалізація типів панелей-контейнерів.*

- Лейаут (layout) — це зазвичай дуже подібне до панелі в JavaFX. Класичні компоновки (row, column, grid, flow) представлені такими контейнерами: HBox, VBox, GridPane, FlowPane, TilePane. Згідно документації, ці вбудовані лейаути «автоматично реорганізують компоненти при зміні розміру вікна»[16] (див. Рис. 16).

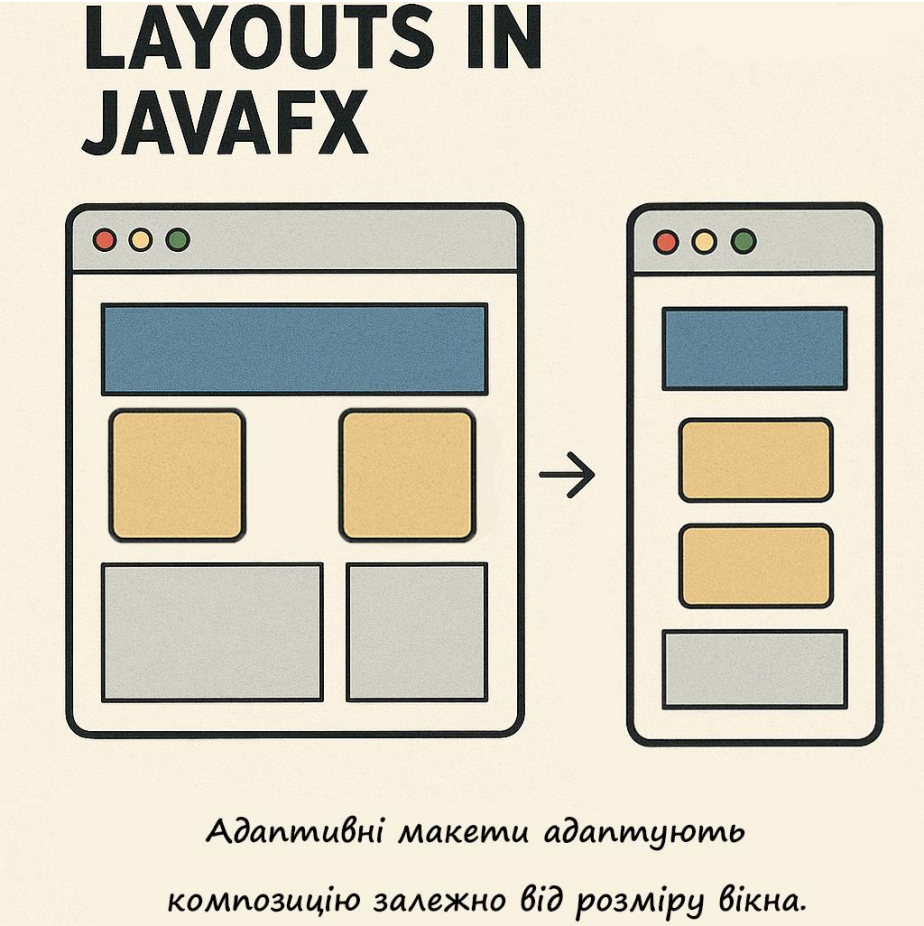


Рисунок *16 — Зміна компонування при змінні розмірів вікна.*

У JavaFX усі візуальні компоненти успадковані від базового класу Node. Серед них контейнери-панелі (Pane, HBox, VBox, тощо) відповідають за розміщення дочірніх вузлів, а елементи керування (Button, Label, TextField тощо) є листовими вузлами сцени. Класи-нащадки Region (зокрема Control) підтримують властивості розтягнення та CSS-стилізації. Є кілька типів

панелей (Group, Pane, Control), кожна з яких визначає свій алгоритм розташування елементів.

Приклад використання: щоб зібрати простий інтерфейс, можна помістити Button і Label в VBox, що автоматично розташовує їх у стовпчик. Якщо вікно буде збільшуватись, VBox перераховуватиме розміри і розташування дочірніх вузлів. JavaFX також надає такі контейнери як ScrollPane (зі смугою прокрутки), SplitPane, Accordion та TabPane. Будь-яку компоненту (наприклад, зображення через ImageView) потрібно додати в панель, щоб вона відображалась. Ці різновиди компонентів і розкладок дозволяють створити гнучку форму GUI з адаптивною реакцією на зміни розміру і змісту вікна.

2.6 CSS-стилізація GUI

Однією з унікальних можливостей JavaFX є вбудована підтримка Cascading Style Sheets (CSS) для оформлення інтерфейсу. Стилі пишуться у звичайних текстових файлах .css з особливими правилами для JavaFX. У стилі необхідно вказувати імена селекторів та властивості з префіксом -fx- (наприклад, -fx-font, -fx-background-color тощо).

Щоб застосувати CSS до програми, досить додати стиль до об’єкту Scene (див. Лістинг 1):

Лістинг 1 — *Приклад додання файлу стилів*.

```
Scene scene = new Scene(root, 800, 600);
scene.getStylesheets().add("styles.css");
```

У прикладі styles.css — це власний файл стилів. Один із прикладів-варіантів коду CSS (див. Лістинг 2):

Лістинг 2 — *Приклад коду CSS файлу*.

```
.custom-button {
    -fx-font: 16px "Serif";
    -fx-padding: 10;
    -fx-background-color: #CCFF99;
```

```
}
```

Цей стиль (.custom-button) можна призначити кнопці через `button.getStyleClass().add("custom-button")`.

За допомогою CSS можна реалізовувати теми інтерфейсу. Замінивши файл-стилю на інший, можна змінити кольорову схему чи шрифтовий стиль усього застосунку без зміни коду. На практиці отримаємо, що два різні CSS-файли (controlsStyles_1.css та controlsStyles_2.css) дають два різні вигляди одного й того ж інтерфейсу. Отже, «скіннінг» UI в JavaFX надзвичайно гнучкий. Крім того, CSS-підхід дозволяє організувати інтерфейсні теми (наприклад, світла та темна тема), перемикаючи різні стилі динамічно. Є навіть готові бібліотеки (наприклад, JMetro, ClarityFX) з популярними сучасними темами для JavaFX. Таким чином, CSS у JavaFX дає розробнику знайомий та потужний інструмент для створення привабливого оформлення та легкої зміни стилів.

2.7 Обробка подій у JavaFX

JavaFX підтримує модель обробки подій, схожу на веб-події («кліки» або натискання клавіш) поширюються по ієрархії вузлів знизу вгору (фазовий підйом події). Коли відбувається подія (наприклад, натиск миші по кнопці), вона спочатку надходить до того вузла (Node), який згенерував її, а потім передається батьківським вузлам, якщо не було «поглинено» (оброблено) раніше.

Щоб отримати та обробити подію, потрібно зареєструвати обробник подій (event handler) для конкретного вузлу. Обробник — це клас, що реалізує інтерфейс EventHandler<T> і його абстрактний метод handle(Event). Наприклад, для кнопки можна встановити обробник натискання так (див. Лістинг 3):

Лістинг *3 — Приклад встановлення обробки події натискання*.

```
button.setOnAction(e -> {
    // Код обробки натиснення.
});
Або універсальніше через addEventHandler():
button.addEventHandler(ActionEvent.ACTION, new
EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
        // Якийсь код.
    }
});
```

Метод addEventHandler(eventType, handler) реєструє обробник події заданого типу на вузлі.

Якщо вузол не отримав подію (тобто в його обробнику виконується e.consume()), подія пройде до батьківського вузла, де може бути інший обробник для тієї самої події. Таким чином можна організувати спільну обробку подій на рівні контейнерів. JavaFX також підтримує фільтри подій (EventFilter), які спрацьовують на фазі захоплення (capturing), до поступки події самому вузлу. Але зазвичай, використовуються прості обробники.

Важливою особливістю є багаторазове реєстрування: один обробник можна прикріпити до кількох вузлів або обробляти кілька типів подій одним методом. Node може мати кілька обробників для різних подій (натискання, наведення, перетягування тощо). JavaFX також надає набір стандартних подій: ActionEvent (натискання кнопок), MouseEvent, KeyEvent, ScrollEvent і інші.

Таким чином, архітектура подій у JavaFX дозволяє гнучко розв’язувати обробку взаємодії користувача: обробники прив’язуються до елементів сцени, подія «піднімається» по сценічному графу, а при необхідності її «поглинають» для уникнення подальшої передачі. Використовуючи відповідні методи (setOnAction, setOnMouseClicked, addEventHandler), можна реалізовувати логіку реакції на будь-які дії користувача.

2.8 Використання FXML та JavaFX Scene Builder

FXML — це спеціальний XML-формат, створений для опису інтерфейсів JavaFX. З його допомогою можна задати ієрархію вузлів, властивості компонентів і прив’язати їх до змінних у контролері (класі, який реалізує поведінку UI). Типовий підхід: у FXML-файлі вказуються елементи (напр. <Button fx:id="okButton" onAction="#handleOk"/>), а в пов’язаному Java-класі-контролері ці елементи позначені анотацією @FXML (напр. @FXML private Button okButton;) і метод handleOk(ActionEvent e) для обробки натискання. Завдяки цьому код зручніше читати і підтримувати, оскільки опис GUI (розташування і налаштування контролів) винесено в XML, а логіка реалізується окремо.

Scene Builder — графічний редактор інтерфейсів (від Oracle), який генерує FXML. Він дозволяє перетягувати на полотно кнопки, тексти, макети тощо, миттєво встановлювати їх властивості та прив’язувати до контролерів (див. Рис. 17) [17].

Серед можливостей Scene Builder:

- Візуальне компонування елементів (drag-and-drop, зміна розміру, вирівнювання).
- Автоматична генерація файлу .fxml, який потім можна відкрити і в IDE для завершення налаштувань.
- Інтеграція з IDE: наприклад, в IntelliJ IDEA можна відразу бачити зміни у Scene Builder.
- Додавання обробників подій (кодова панель в Scene Builder дозволяє призначити методи обробки різним подіям на FXML-елементах).
- Інтернаціоналізація: можливість прив’язувати видимі тексти до ресурсних ключів (Scene Builder підказує, як замінити текст властивості на посилання виду %key для використання в ResourceBundle).

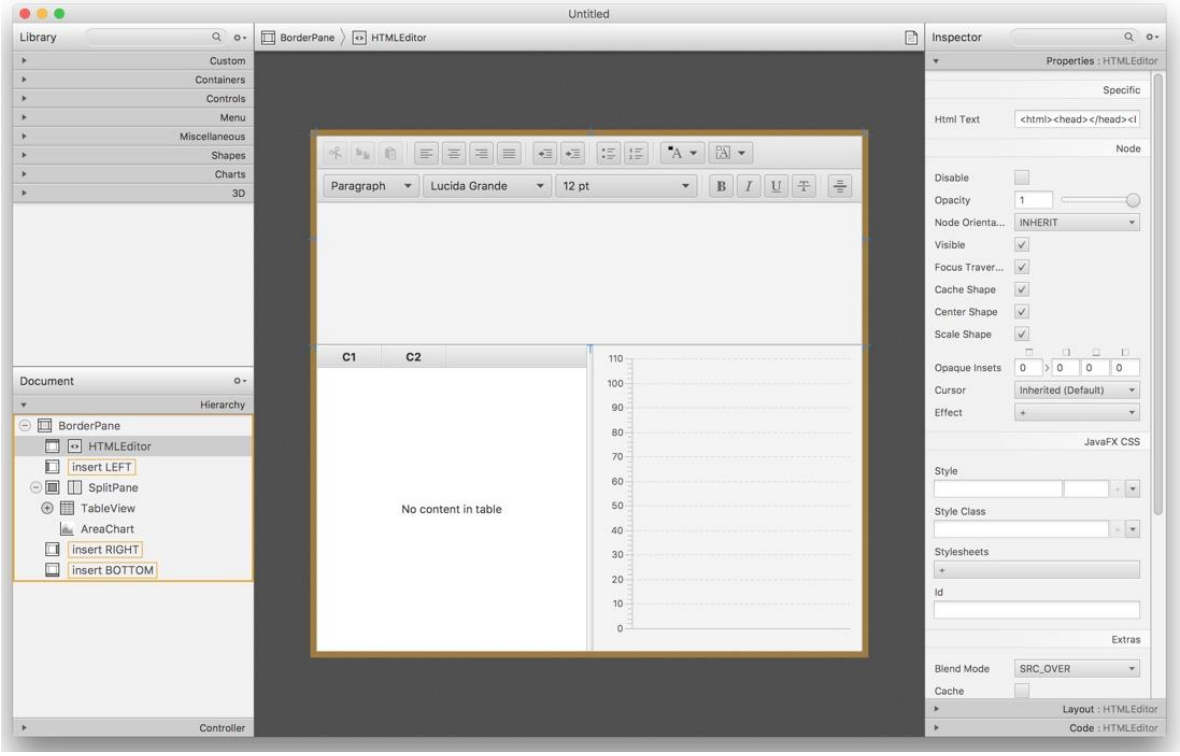


Рисунок *17 — Робоче вікно Scene Builder.*

Тому використання цих інструментів «розв'язує» руки й дозволяє не витрачати забагато часу на написання власноруч макету інтерфейсу, відлагоджуючи потім в пошуках проблем і багів.

2.9 Загальні вимоги до застосунку

Сформулюємо та узагальнимо вимоги до застосунку як функціональні та нефункціональні.

Функціональні вимоги:

- Завантаження та попередній перегляд зображення. Після вибору файлу через FileChooser обране зображення одразу відображається в ImageView. Інтерфейс має бути адаптивним: при зміні розміру вікна ImageView зберігає співвідношення сторін і масштабується відповідно до доступного простору.
- Введення та виведення прихованих даних. Над полем TextArea для вводу секретного повідомлення розміщується підказка про

допустимий обсяг тексту; кнопки Embed/Encode і Extract/Decode запускають вбудовування чи вилучення. Після завершення операції виводиться діалог із підтвердженням успіху або деталями помилки (наприклад, недостатньо пікселів для приховування, неподтримуваний формат файлу).

- Стеганографія за алгоритмом LSB. Виділені модулі encoder та decoder розпізнають PNG truecolor (RGB/RGBA) 8-bit, перетворюють введений текст у бітову послідовність у кодуванні UTF-8 і вбудовують її по найменшому значущому біту кожного пікселя.
- Опціональні розширення функціоналу. Мультимовність інтерфейсу через ResourceBundle (.properties), «світла» й «темна» теми оформлення, автозаповнення імен файлів із історією останніх виборів, анімації чи привітальний екран при запуску, а також екран порівняння: оригінал і стегофайл поруч із метаданими (розміри, обсяг прихованого повідомлення).
- Безпечне завершення роботи (Exit): у верхньому правому куті основного вікна додати кнопку “Вихід”, яка при натисканні коректно зберігає всі тимчасові файли, закриває відкриті ресурси (зображення, потоки), зберігає налаштування користувача та завершує роботу програми без ризику пошкодження даних.

Нефункціональні вимоги:

- UX/UI-дизайн. Інтерфейс має бути інтуїтивно зрозумілим із логічним групуванням контролів: читабельні підписи, уніфіковані відступи й вирівнювання, достатня контрастність.
- Надійність і безпека. Обов’язкова коректна обробка всіх помилкових дій користувача (наприклад, спроба вилучити дані з чистого зображення) зі збереженням цілісності файлу-носія та інформативними повідомленнями.

- Продуктивність. Алгоритми LSB мають працювати з зображеннями PNG саме 8 bit RGB у режимі реального часу або з мінімальними затримками.
- Технічний супровід. Опціонально — модуль логування (запис операцій у файл) для діагностики та налагодження, а також окремий механізм динамічного перемикання мов інтерфейсу в рантаймі через JavaFX ResourceBundle.
- Архітектура MVC (опціонально): застосунок має бути побудований за патерном Model-View-Controller, де Model відповідає за дані та бізнес-логіку, View — за відображення інтерфейсу, Controller — за взаємодію між ними. Це забезпечить чітке розділення обов’язків, спрощене тестування й подальшу підтримку коду.

Підбиваючи підсумки — застосунок має відповідати усім основним вказаним вимогам й може не включати в своїй реалізації додаткових (опціональних) можливостей.

3 ПРОЕКТУВАННЯ ТА КОНСТРУЮВАННЯ ЗАСТОСУНКУ ДЛЯ СТЕГАНОГРАФІЇ У ФАЙЛИ ЗОБРАЖЕНЬ ІЗ ВИКОРИСТАННЯМ ЗАСОБІВ JAVAFX

3.1 Архітектура проекту

Основні архітектурні шари та патерни:

1. MVC (Model-View-Controller) — архітектурний шаблон, що ділить застосунок на три взаємопов'язані, між собою, компоненти: Вид, Модель та Контролер. Це допомагає покращити зручність обробки та масштабованість проекту:
   View: FXML-файли (.fxml) + CSS;
   Controller: класи в пакеті lsb.controllers обробляють події та пов'язують UI з бізнес-логікою;
   Model/Service: SteganographyService інкапсулює алгоритми приховування та вилучення даних.
2. Сервісний шар:
   SteganographyService реалізує алгоритм LSB-стеганографії. Містить деталі роботи із зображеннями, бітовими операціями і метаданими.
3. Утиліти та допоміжні компоненти:
   FileHistoryManager — відповідає за зберігання і завантаження історії операцій вбудовування інформації та її прочитування з файлу-носія у файли: encode_history.txt, decode_history.txt.
   LocalizationService — керує локалізацією рядків інтерфейсу.
   Logger — логує операції та дії користувача у файл lsb_logs.txt.
4. Ресурси проекту:
   Поділенні за типами (views / styles / images / bundles), що спрощує підтримку тем і багатомовність й дає швидкий, сегрегований

доступ. Також використовується CSS для стилізації JavaFX-елементів.

Структура пакетів проекту, з головним файлом має наступний вигляд (див. Рис. 18).

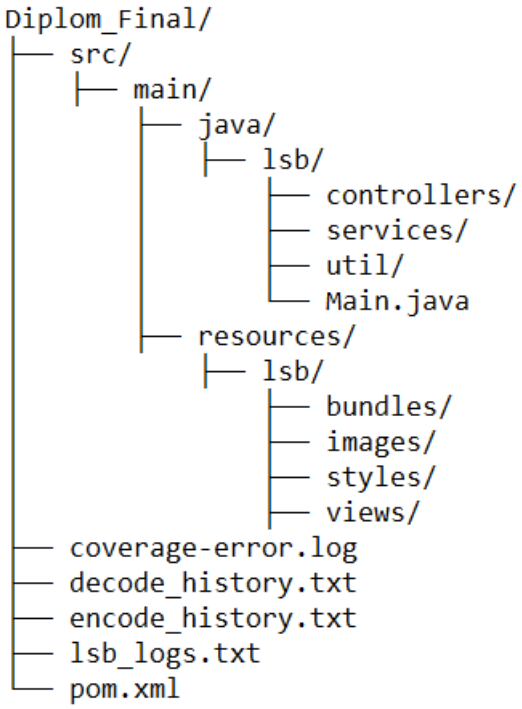


Рисунок *18 — Структура пакетів проекту.*

Систему збірки було обрано Maven [20] (файл pom.xml), через:

- Високий поріг входження: дуже легко налаштовується й практично зі старту можна вже працювати з проектом.
- Стандартизовану структуру проєкту: Maven «примушує» до класичної ієрархії директорій (src/main/java, src/main/resources), що полегшує навігацію і відразу дає зрозумілу картину структури проекту.
- Декларативне управління залежностями: у pom.xml описується лише група, артефакт і версія кожної бібліотеки — Maven сам скачує все з центрального репозиторію. Не потрібно вручну додавати .jar-файли в проєкт.

- Уніфікований життєвий цикл (build lifecycle): Maven має набір заздалегідь визначених фаз (validate → compile → test → package → verify → install → deploy). Плагін-орієнтована архітектура дає змогу «підчепити» генерацію JavaFX-артефактів, створення звітів, перевірку статики коду (Checkstyle, тощо) і багато іншого — просто додаванням потрібного плагіна.
- Конвенція замість конфігурації: більшість завдань (пакетування, копіювання ресурсів, запуск тестів) працює «з коробки» без зайвих налаштувань - достатньо помістити вихідні коди та ресурси в стандартизовані папки.

3.2 Програмна реалізація

Проект має дві основні частини — графічну та прикладну, тому для зручності опису розглянемо їх окремо:

3.3.1 Реалізація прикладної частини

По-перше, при створенні проекту у IntelliJ IDEA потрібно вибрати систему збірки Maven, щоб проект вже створився по патерну й мав структуру й файл (pom.xml) потрібну для запуску та роботи з проектом (див. Рис. 19). Ще важливо обрати версію JDK 21 (не вище), про цю особливість буде далі. Якщо ж немає актуальної версії — потрібно звернутися до офіційного сайту і вже на ньому встановити все необхідне [18].

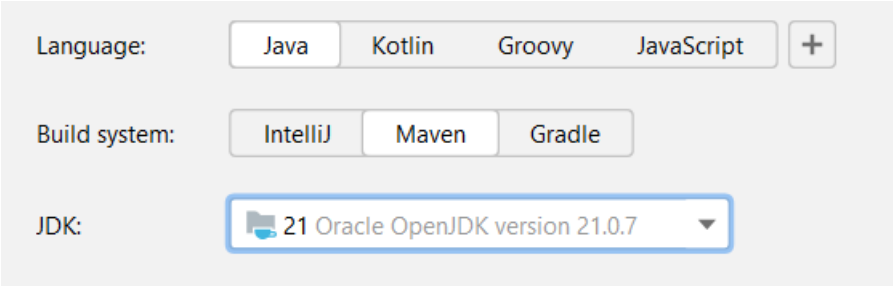


Рисунок *19 — Створення проекту.*

По-друге, якщо буде потреба у «локальному» запуску, через систему IntelliJ IDEA — треба встановити бібліотеку JavaFX [19] версії 21.0.7 локально до комп'ютеру й запам'ятати шлях до неї. Потім обрати "Edit Configurations", натиснути на "+" й заповнити поля на кшталт малюнку (див. Рис. 20):

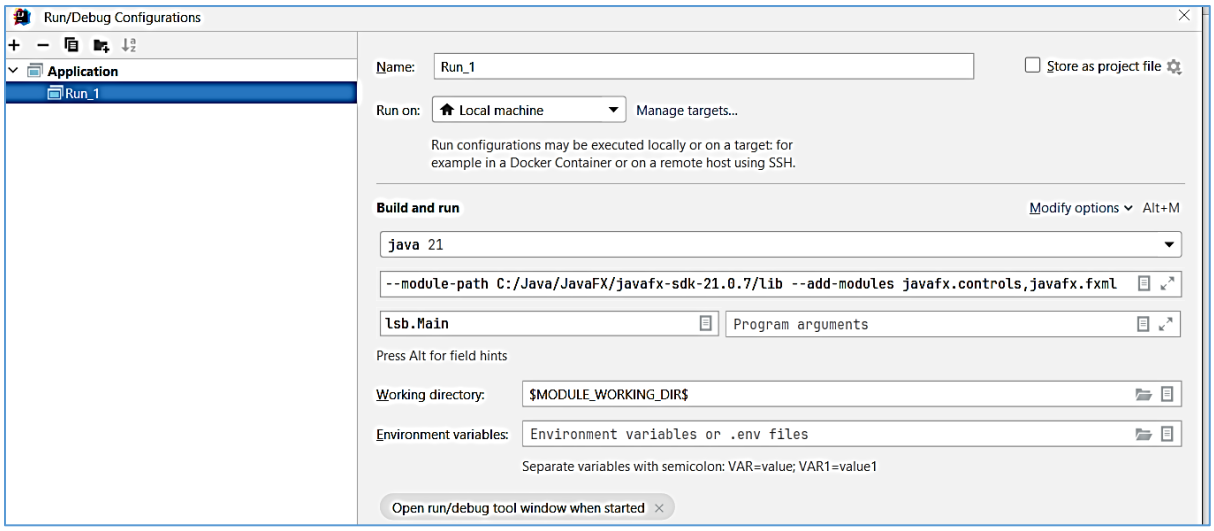


Рисунок *20 — Налаштування запуску проекту*.

Де ім'я можна задати будь-яке, JDK — буде вибрана по замовчуванням, а ось "VM options" потрібно заповнювати власноруч, вказавши точний шлях до бібліотеки JavaFX на комп'ютері (в моєму випадку: --module-path C:/Java/JavaFX/javafx-sdk-21.0.7/lib --add-modules javafx.controls,javafx.fxml), а якщо такого віконця немає — потрібно натиснути на напис «модифікувати опції» й поставити галочку напроти "Add VM options", після чого можна приміняти налаштування й виходити до робочої області IDE.

Також, заодно, можна налаштувати й конфігураційний файл Maven (pom.xml): в загальній інформації о проекті визначаємо через властивості (properties) мову Java версії 21, у залежностях підключаємо такі бібліотеки, як JavaFX — для реалізації графічного інтерфейсу користувача, PNGJ — для обробки зображень у форматі PNG, Metadata Extractor — для роботи з метаданими графічних файлів, у збірці й плагінах потрібно прописати —

компіляцію коду з урахуванням заданої версії Java, запуск JavaFX-застосунку за допомогою javafx-maven-plugin, створення виконуваного JAR-файлу з усіма залежностями через maven-assembly-plugin, та встановити точку входу, як mainClass у плагінах запуску та пакування.

На цьому головні налаштування готові до коректної роботи.

3.3.1.1 Опис класів

Головним класом проекту є **Main.java**:

Пакет і імпорти (див. Лістинг 3)**:**

Лістинг 3 — *Початок Main*.

```
package lsb;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;
import lsb.util.LocalizationService;
import lsb.util.ThemeManager;

import java.util.Objects;
import java.util.prefs.Preferences;
```

1. `package lsb;`

   Вказує, що клас знаходиться в пакеті lsb.

2. Імпорти з JavaFX:
   - `Application, Stage, Scene, Parent` — основні класи для створення JavaFX-додатку.
   - `FXMLLoader` — для завантаження інтерфейсу з FXML-файлів.
   - `Image` — для роботи з іконками вікна.
3. Власні утиліти:
   - `LocalizationService` — сервіс для інтернаціоналізації (вибір мови інтерфейсу).
   - `ThemeManager` — утиліта для застосування темної/світлої теми.

4. Java стандартні:
    - `Objects` — для перевірки на null (метод Objects.requireNonNull()).
    - `Preferences` — API для збереження/завантаження налаштувань користувача (theme, locale).

Клас Main та метод main (див. Лістинг 4):

Лістинг 4 — Початок Main.

```
public static void main(String[] args) {
    launch(args);
}
```

Клас `Main` наслідується від `javafx.application.Application`. У JavaFX для старту програми потрібно успадкувати Application та перевизначити метод `start(Stage)`, тому метод `main(String[] args)` викликає Application.launch(args), який ініціює JavaFX-таймлайн і в результаті викликає `start(Stage)`.

Метод start(Stage primaryStage) (див. Лістинг 5):

Лістинг 5 — Метод start.

```
@Override
public void start(Stage primaryStage) throws Exception {
    // Load user preferences
    Preferences prefs = Preferences.userNodeForPackage(Main.class);
    String theme = prefs.get("theme", "light");
    String locale = prefs.get("locale", "English");

    // Initialize localization service
    LocalizationService.setLocale(locale);

    // Load and configure splash screen
    Parent splashRoot =
FXMLLoader.load(Objects.requireNonNull(getClass().getResource("/lsb/views/spl
ash.fxml")));
    Scene splashScene = new Scene(splashRoot);

splashScene.getStylesheets().add(Objects.requireNonNull(getClass().getResourc
e("/lsb/styles/splash.css")).toExternalForm());
    ThemeManager.applyTheme(splashScene, theme);

    // Configure primary stage
    primaryStage.setTitle("LSB_PNG");
    primaryStage.getIcons().add(new
Image(Objects.requireNonNull(getClass().getResourceAsStream("/lsb/images/icon
.png"))));
```

```
        primaryStage.setScene(splashScene);
        primaryStage.setFullScreen(true);
        primaryStage.show();

        // Schedule transition to main menu after delay
        new java.util.Timer().schedule(
                new java.util.TimerTask() {
                    @Override
                    public void run() {
                        // JavaFX operations must run on UI thread
                        javafx.application.Platform.runLater(() -> {
                            try {
                                // Load main interface
                                Parent mainRoot =
FXMLLoader.load(Objects.requireNonNull(getClass().getResource("/lsb/views/mai
n.fxml")));
                                Scene mainScene = new Scene(mainRoot);

mainScene.getStylesheets().add(Objects.requireNonNull(getClass().getResource(
"/lsb/styles/main.css")).toExternalForm());

                                // Apply theme and switch to main scene
                                ThemeManager.applyTheme(mainScene, theme);
                                primaryStage.setScene(mainScene);
                                primaryStage.setFullScreen(true);
                            } catch (Exception e) {
                                e.printStackTrace();
                            }
                        });
                    }
                },
                2500  // 2.5 second splash screen display duration
        );
    }
```

Деталі по кроках:

1. Preferences:
    - Створюється вузол налаштувань за пакетом класу Main.
    - Зчитуються два ключі:
        - `theme` — тема інтерфейсу (“`light`” за замовчуванням).
        - `locale` — мова інтерфейсу (“`English`” за замовчуванням).
2. LocalizationService:
    - Викликається статичний метод `setLocale(locale)`, який підвантажує відповідні ресурси (файли `.properties`) для інтерфейсу.
3. Splash Screen:

- `FXMLLoader.load(...)` завантажує розмітку `splash.fxml` з ресурсів.
- Створюється нова `Scene`, додається CSS-стиль `splash.css`.
- Через `ThemeManager.applyTheme(...)` на сцену накладається світла чи темна тема.

4. Налаштування вікна:
   - Назва вікна: `LSB_PNG`.
   - Іконка: файл `icon.png` з ресурсів (встановлюється через `getResourceAsStream`).
   - Встановлюється сцена `splash`, режим повноекранного показу і викликається `show()`.
5. Переходи між сценами:
   - Використовується `PauseTransition` на 2500 мс (2.5 с), після чого в `setOnFinished` завантажується головна розмітка `main.fxml` і відповідний CSS `main.css`.
   - Знову викликається `ThemeManager.applyTheme` для збереження користувацької теми.
   - Сцена у `primaryStage` замінюється на основну, без необхідності створювати нове вікно.

За стеганографію у проекті відповідає клас **SteganographyService.java**:

Пакет і імпорти (див. Лістинг 6)**:**

Лістинг 6 — Імпорти стеганографічного класу.

```
package lsb.services;

import java.awt.image.BufferedImage;
import java.io.*;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.*;
import javax.imageio.ImageIO;

import ar.com.hjg.pngj.*;
import ar.com.hjg.pngj.chunks.ChunkCopyBehaviour;
import java.nio.file.Files;
```

1. `package lsb.services;` — клас належить до пакету `lsb.services`.
2. Стандартні імпорти Java
    - `BufferedImage, ImageIO` — для читання/запису растрових зображень у методі декодування.
    - `ByteBuffer, StandardCharsets` — для формування заголовка довжини повідомлення та перетворення рядка в байти.
    - `File, FileInputStream, FileOutputStream, Files` — для роботи з файловою системою та читання байтів тимчасового PNG-файлу.
    - Колекції `List, ArrayList` — для накопичення бітів під час декодування.
3. Бібліотека PNGJ (`ar.com.hjg.pngj`)
    - `PngReader, PngWriter, ImageInfo, ImageLineInt` — для построкового читання й запису пикселів PNG, з можливістю зберегти всі існуючі chunks (включаючи метадані).
    - `ChunkCopyBehaviour.COPY_ALL_SAFE` — вказує писати вивід із збереженням безпечних chunks.

Константи класу (див. Лістинг 7):

Лістинг *7 — Константа-резерв*.

```
private static final int HEADER_SIZE = 32; // Bits reserved for storing
message length (4 bytes)
```

`HEADER_SIZE` = 32 біти (4 байти) — резерв під збереження довжини вбудованого повідомлення.

Метод `encode(String inputPath, String outputPath, String message)`

Основна логіка приховування тексту в зображенні.

1. Підготовка payload (див. Лістинг 8):

   Лістинг *8* — *payload*.

```
byte[] msgBytes = message.getBytes(StandardCharsets.UTF_8);
ByteBuffer headerBuf =
ByteBuffer.allocate(4).putInt(msgBytes.length);
byte[] payload = new byte[4 + msgBytes.length];
System.arraycopy(headerBuf.array(), 0, payload, 0, 4);
        System.arraycopy(msgBytes,      0, payload, 4,
msgBytes.length);
```

- Перетворення повідомлення у масив байтів UTF-8.
- Заголовок довжини (4 байти, big-endian) — містить довжину msgBytes.length.
- Фінальний payload = 4-байтовий header + тіло повідомлення.

2. Перевірка розміру (див. Лістинг 9):

   Лістинг *9* — *Перевірка розміру*.

```
File inputFile = new File(inputPath);
long originalSize = inputFile.length();
// Ініціалізація PngReader
PngReader reader = new PngReader(inputFile);
ImageInfo imgInfo = reader.imgInfo;
// Підтримка лише не-indexed та 8-bit
if (imgInfo.indexed || imgInfo.bitDepth != 8) {
        reader.close();
    throw new IllegalArgumentException(...);
}
// Обчислення capacity: cols × rows × 3 (біти для RGB)
int capacity = imgInfo.cols * imgInfo.rows * 3;
int totalBits = payload.length * 8;
if (totalBits > capacity) {
        reader.close();
    throw new IllegalArgumentException(...);
}
```

- Оригінальний розмір файлу зберігається в originalSize.
- Перевірка формату: не підтримуються indexed-кольори та bitDepth≠8.
- Обчислення ємності: по 1 біту в кожному з трьох кольорових каналів пікселя.
- Валідація достатності ємності для вміщення всіх totalBits.

3. Налаштування PngWriter і копіювання метаданих (див. Лістинг 10):

Лістинг 10 — *Копіювання метаданих*.

```
File tempFile = File.createTempFile("stego", ".png");
PngWriter writer = new PngWriter(tempFile, imgInfo);
writer.copyChunksFrom(reader.getChunksList(),
ChunkCopyBehaviour.COPY_ALL_SAFE);
        writer.setCompLevel(9);
```

- Тимчасовий файл для проміжного запису.
- Копіювання всіх “безпечних” chunks (зберігає коментарі, EXIF тощо).
- Максимальна компресія (рівень 9) для мінімізації розміру IDAT.

4. Вбудовування бітів LSB (див. Лістинг 11):

Лістинг 11 — *Процес LSB*.

```
int bitIndex = 0;
for (int row = 0; row < imgInfo.rows; row++) {
ImageLineInt line = (ImageLineInt) reader.readRow(row);
int[] scanline = line.getScanline();
    for (int col = 0; col < imgInfo.cols && bitIndex < totalBits;
col++) {
int pos = col * imgInfo.channels;
// RED
embedBit(scanline, pos + 0, payload, bitIndex++);
// GREEN
embedBit(scanline, pos + 1, payload, bitIndex++);
// BLUE
embedBit(scanline, pos + 2, payload, bitIndex++);
    }
            writer.writeRow(line);
}
        reader.end();
writer.end();
```

- Проходимо по кожному рядку й пікселю.
- Послідовно в кожний канал (R, G, B) записуємо по одному біту з payload, змінивши LSB:

```
scanline[channelIndex] = (scanline[channelIndex] & 0xFE) | bitValue;
```

- Alpha-канал (якщо є) залишається незмінним, оскільки в ImageInfo.channels він йде останнім та не обробляється.

5. Підгонка розміру та запис фінального файлу (див. Лістинг 12):

Лістинг 12 — *Підведення розміру*.

```
byte[] stegoBytes = Files.readAllBytes(tempFile.toPath());
long stegoSize = stegoBytes.length;
```

```
        try (FileOutputStream os = new FileOutputStream(outputPath)) {
                os.write(stegoBytes);
            if (stegoSize < originalSize) {
        byte[] padding = new byte[(int)(originalSize - stegoSize)];
                os.write(padding);
            }
                    }
                    tempFile.delete();
```

- Читаємо весь вміст тимчасового PNG.
- Записуємо його в outputPath.
- Якщо після стиснення розмір менший за originalSize, додаємо нульбайти padding для точного збігу розміру з оригіналом.
- Видаляємо тимчасовий файл.

Метод `decode(String inputPath)`**:**

Алгоритм витягнення прихованого повідомлення (див. Лістинг 13):

Лістинг 13 — *Decode метод*.

```
BufferedImage image = ImageIO.read(new File(inputPath));
List<Integer> bits = new ArrayList<>();
for (int y = 0; y < image.getHeight(); y++) {
        for (int x = 0; x < image.getWidth(); x++) {
int pixel = image.getRGB(x, y);
        bits.add((pixel >> 16) & 1); // RED LSB
        bits.add((pixel >> 8)  & 1); // GREEN LSB
        bits.add(pixel & 1);         // BLUE LSB
    }
            }
// Зчитуємо 32-бітовий header → довжина повідомлення
int len = 0;
for (int i = 0; i < HEADER_SIZE; i++) {
len = (len << 1) | bits.get(i);
}
// Відновлюємо байти повідомлення
byte[] msgBytes = new byte[len];
for (int i = 0; i < len * 8; i++) {
        if (bits.get(HEADER_SIZE + i) == 1) {
msgBytes[i/8] |= 1 << (7 - (i % 8));
        }
        }
        return new String(msgBytes, StandardCharsets.UTF_8);
```

1. Зчитування всіх пікселів через `ImageIO.read()`.
2. Накопичення LSB каналів R, G, B у список bits.
3. Відновлення довжини повідомлення із перших 32 біт (`HEADER_SIZE`).
4. Побітове формування масиву `msgBytes` із наступних `len * 8` біт.

5. Перетворення на рядок UTF-8.

Метод `calculateMaxChars(String imagePath)`

Швидка оцінка, скільки символів можна заховати (див. Лістинг 14):

Лістинг 14 — *Коротка оцінка*.

```
BufferedImage image = ImageIO.read(new File(imagePath));
int capacity = image.getWidth() * image.getHeight() * 3;
return (capacity - HEADER_SIZE) / 8;
```

Клас контроллер **DecodeController.java** та його шаблон розмітки **decode.fxml**:

DecodeController.java (див. Лістинг 15):
Лістинг 15 — *DecodeController*.

```
package lsb.controllers;

import javafx.animation.*;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.StackPane;
import javafx.scene.Scene;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.util.Duration;
import lsb.services.SteganographyService;
import lsb.util.FileHistoryManager;
import lsb.util.LocalizationService;
import lsb.util.Logger;
import lsb.util.ThemeManager;

import java.io.File;
import java.net.URL;
import java.util.Objects;
import java.util.ResourceBundle;

/**
 * Controller for the decoding interface.
 * Handles:
 *   - PNG file selection for decoding
 *   - Message extraction via steganography
 *   - Animation effects and UI interactions
 *   - Navigation back to main menu
 */
public class DecodeController implements Initializable {
    @FXML private TextField filePathField;    // File path display
    @FXML private Button decodeButton;        // Decode action trigger
    @FXML private Button cancelButton;        // Cancel/back button
    @FXML private ImageView previewImage;     // Image preview area
    @FXML private TextArea messageArea;       // Extracted message display
    @FXML private StackPane rootPane;         // Root container
    @FXML private Label messageText;          // "Message" label
```

```
    @FXML private Button browseText;          // Browse button
    @FXML private Label F_pathText;           // "File path" label

    private File selectedFile;                // Currently selected PNG file
    private final SteganographyService stegoService = new
SteganographyService();

    /**
     * Pre-selects a file for decoding (e.g., from history).
     *
     * @param file Pre-selected PNG file
     */
    public void setSelectedFile(File file) {
        this.selectedFile = file;
        filePathField.setText(file.getAbsolutePath());
        previewImage.setImage(new Image(file.toURI().toString()));
        decodeButton.setDisable(false);  // Enable decode button
    }

    /**
     * Initializes controller after FXML loading.
     * Sets up:
     *   - UI element localization
     *   - Button animations
     *   - Default image placeholder
     *   - Initial button states
     */
    @Override
    public void initialize(URL location, ResourceBundle resources) {
        setupAnimations();  // Configure button hover effects

        // Localize UI elements
        messageText.setText(LocalizationService.getString("message"));
        browseText.setText(LocalizationService.getString("browse"));
        decodeButton.setText(LocalizationService.getString("decode"));
        cancelButton.setText(LocalizationService.getString("cancel"));
        F_pathText.setText(LocalizationService.getString("file.path"));

        // Set default placeholder image
        previewImage.setImage(new
Image(Objects.requireNonNull(getClass().getResourceAsStream("/lsb/images/plac
eholder.png"))));
        decodeButton.setDisable(true);  // Disabled until file selected
    }

    /**
     * Configures button hover animations.
     * Creates scale effects for the decode button:
     *   - Slight enlargement on mouse enter
     *   - Return to normal size on mouse exit
     */
    private void setupAnimations() {
        // Create scale-in effect (enlargement)
        ScaleTransition scaleIn = new ScaleTransition(Duration.millis(200),
decodeButton);
        scaleIn.setToX(1.05);
        scaleIn.setToY(1.05);

        // Create scale-out effect (return to normal)
        ScaleTransition scaleOut = new ScaleTransition(Duration.millis(200),
decodeButton);
        scaleOut.setToX(1.0);
        scaleOut.setToY(1.0);
```

```
        // Apply effects on mouse events
        decodeButton.setOnMouseEntered(e -> {
            if (!decodeButton.isDisable()) {
                scaleOut.stop();
                scaleIn.playFromStart();
            }
        });

        decodeButton.setOnMouseExited(e -> {
            scaleIn.stop();
            scaleOut.playFromStart();
        });
    }

    /**
     * Handles file browsing action.
     * Opens file chooser to select PNG image for decoding.
     * Updates UI with selection and enables decode button.
     */
    @FXML
    private void handleBrowse() {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle(LocalizationService.getString("select.png"));
        fileChooser.getExtensionFilters().add(
                new FileChooser.ExtensionFilter("PNG Images", "*.png"));

        // Show dialog and process selection
        selectedFile =
fileChooser.showOpenDialog(rootPane.getScene().getWindow());
        if (selectedFile != null) {
            filePathField.setText(selectedFile.getAbsolutePath());
            previewImage.setImage(new
Image(selectedFile.toURI().toString()));
            decodeButton.setDisable(false);  // Enable decoding
        }
    }

    /**
     * Performs message extraction from selected image.
     * Steps:
     *  1. Log decoding start
     *  2. Extract message via steganography service
     *  3. Display extracted message with fade animation
     *  4. Add file to decode history
     *  5. Handle errors with user notification
     */
    @FXML
    private void handleDecode() {
        try {
            // Log decoding operation
            Logger.log("DECODE", "Starting decoding: " +
selectedFile.getAbsolutePath());

            // Extract hidden message
            String message =
stegoService.decode(selectedFile.getAbsolutePath());
            messageArea.setText(message);

            // Add to recent files history

FileHistoryManager.addDecodeHistory(selectedFile.getAbsolutePath());
            Logger.log("DECODE", "Decoding successful");
```

```
            // Animate message appearance
            FadeTransition fade = new FadeTransition(Duration.seconds(1),
messageArea);
            fade.setFromValue(0);    // Fully transparent
            fade.setToValue(1);      // Fully opaque
            fade.play();
        } catch (Exception e) {
            Logger.log("ERROR", "Decoding failed: " + e.getMessage());
            showError(LocalizationService.getString("error.decode") + ": " +
e.getMessage());
        }
    }

    /**
     * Returns to main menu.
     * Steps:
     *  1. Log navigation event
     *  2. Load main.fxml interface
     *  3. Apply current theme
     *  4. Switch to fullscreen main scene
     */
    @FXML
    private void handleCancel() {
        try {
            Logger.log("NAVIGATION", "Returning to main menu from decode
page");

            // Load main interface
            FXMLLoader loader = new
FXMLLoader(getClass().getResource("/lsb/views/main.fxml"));
            Parent mainRoot = loader.load();
            Scene mainScene = new Scene(mainRoot);

            // Apply current theme and configure stage
            ThemeManager.applyTheme(mainScene,
ThemeManager.getCurrentTheme());
            Stage stage = (Stage) rootPane.getScene().getWindow();
            stage.setScene(mainScene);
            stage.setFullScreen(true);
        } catch (Exception e) {
            Logger.log("ERROR", "Navigation failed: " + e.getMessage());
            e.printStackTrace();
        }
    }

    /**
     * Displays error alert dialog.
     *
     * @param message Error message to display
     */
    private void showError(String message) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle(LocalizationService.getString("error.title"));
        alert.setHeaderText(null);  // No header
        alert.setContentText(message);
        alert.showAndWait();
    }
}
```

Функціонал:

Цей клас керує логікою роботи інтерфейсу декодування. Основна функціональність:

1. Життєвий цикл інтерфейсу
    - initialize(): Викликається при створенні інтерфейсу
        - Налаштування анімацій кнопок
        - Застосування локалізованих текстів
        - Встановлення заглушки для зображення
        - Блокування кнопки декодування
2. Робота з файлами
    - handleBrowse(): Відкриття діалогового вікна вибору файлу
        - Фільтр для PNG-зображень
        - Оновлення шляху та прев'ю при виборі
    - setSelectedFile(): Програмний вибір файлу
3. Процес декодування
    - handleDecode(): Основний алгоритм роботи
        - Логування початку операції
        - Виклик сервісу стеганографії
        - Відображення результату з анімацією
        - Збереження в історії файлів
        - Обробка помилок з сповіщенням користувача
4. Навігація та додаткові функції
    - handleCancel(): Повернення на головний екран
        - Завантаження main.fxml
        - Застосування поточної теми оформлення
        - Перехід у повноекранний режим
    - showError(): Уніфікований спосіб відображення помилок
5. Поліпшення UX
    - Анімація кнопки декодування:
        - Плавне збільшення на 5% при наведенні

- Повернення до нормального розміру при відведенні
- Анімація появи тексту:
  - Ефект поступового з'явлення (fade-in)
  - Тривалість 1 секунда

decode.fxml (див. Лістинг 16)**:**

Лістинг 16 — *decode.fxml*.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
  Decoding Interface Layout
  =========================
  Defines the UI structure for the message decoding interface.

  Structure:
    - StackPane: Root container with application-wide styling
    └── VBox: Main content container (centered)
        ├── Label: Section title
        ├── GridPane: Form layout with 3 columns
        │   ├── Row 0: File path selection
        │   │   ├── Label: "File path"
        │   │   ├── HBox: Path display and browse button
        │   ├── Row 1: Image preview
        │   │   └── ImageView: PNG preview
        │   └── Row 2: Message output
        │       ├── Label: "Message"
        │       └── TextArea: Decoded message display
        └── HBox: Action buttons
            ├── Button: Decode action
            └── Button: Cancel/back action

  Style Classes:
    - root: Applied to StackPane (root container)
    - content-container: Main content VBox styling
    - section-title: Title text styling
    - form-grid: GridPane styling
    - form-label: Field label styling
    - input-field: TextField styling
    - browse-button: File browse button styling
    - message-area: TextArea styling
    - action-button: Primary action button
    - cancel-button: Secondary/cancel button
-->


<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.layout.StackPane?>
```

```
<?import javafx.scene.layout.VBox?>

<StackPane xmlns="http://javafx.com/javafx/11.0.1"
           xmlns:fx="http://javafx.com/fxml/1"
           fx:controller="lsb.controllers.DecodeController"
           styleClass="root"
           fx:id="rootPane">  <!-- Root container with application-wide
styling -->

    <!-- Main content area - centered with vertical spacing -->
    <VBox alignment="CENTER" spacing="20" styleClass="content-container">

        <!-- Section title - localized through controller -->
        <Label fx:id="titleLabel" styleClass="section-title"/>

        <!-- Form layout with responsive column sizing -->
        <GridPane styleClass="form-grid">
            <columnConstraints>
                <!-- Column 0: Field labels -->
                <ColumnConstraints hgrow="SOMETIMES" minWidth="150"/>
                <!-- Column 1: Input fields (primary content) -->
                <ColumnConstraints hgrow="ALWAYS" minWidth="300"/>
                <!-- Column 2: Spacer/utility -->
                <ColumnConstraints hgrow="SOMETIMES" minWidth="100"/>
            </columnConstraints>
            <rowConstraints>
                <!-- Row 0: File path selection -->
                <RowConstraints minHeight="40"/>
                <!-- Row 1: Image preview -->
                <RowConstraints minHeight="150"/>
                <!-- Row 2: Message output (expands vertically) -->
                <RowConstraints minHeight="150" vgrow="ALWAYS"/>
            </rowConstraints>

            <!-- File Path Selection -->
            <Label fx:id="F_pathText"
                   GridPane.rowIndex="0"
                   GridPane.columnIndex="0"
                   styleClass="form-label"/>  <!-- Localized "File path"
label -->

            <HBox GridPane.rowIndex="0"
                  GridPane.columnIndex="1"
                  spacing="5">
                <!-- Read-only path display -->
                <TextField fx:id="filePathField"
                           styleClass="input-field"
                           HBox.hgrow="ALWAYS"
                           editable="false"/>

                <!-- Browse button - triggers handleBrowse() -->
                <Button fx:id="browseText"
                        onAction="#handleBrowse"
                        styleClass="browse-button"
                        text="Browse"/>  <!-- Placeholder text - localized in
controller -->
            </HBox>

            <!-- Image Preview -->
            <ImageView fx:id="previewImage"
                       fitWidth="150"
                       fitHeight="150"
                       preserveRatio="true"
```

```
                        GridPane.rowIndex="1"
                        GridPane.columnIndex="0"
                        GridPane.columnSpan="3"
                        GridPane.halignment="CENTER"/>  <!-- Centered PNG
preview -->

            <!-- Message Output -->
            <Label fx:id="messageText"
                   text="Message:"
                   GridPane.rowIndex="2"
                   GridPane.columnIndex="0"
                   styleClass="form-label"/>  <!-- Localized "Message" label
-->

            <!-- Non-editable message display area -->
            <TextArea fx:id="messageArea"
                      GridPane.rowIndex="2"
                      GridPane.columnIndex="1"
                      styleClass="message-area"
                      editable="false"
                      GridPane.columnSpan="2"/>  <!-- Spans remaining columns
-->
        </GridPane>

        <!-- Action Buttons -->
        <HBox alignment="CENTER" spacing="20">
            <!-- Decode button - triggers handleDecode() -->
            <Button fx:id="decodeButton"
                    onAction="#handleDecode"
                    styleClass="action-button"
                    text="Decode"/>  <!-- Placeholder text - localized in
controller -->

            <!-- Cancel button - returns to main menu -->
            <Button fx:id="cancelButton"
                    onAction="#handleCancel"
                    styleClass="cancel-button"
                    text="Cancel"/>  <!-- Placeholder text - localized in
controller -->
        </HBox>
    </VBox>
</StackPane>
```

Функціонал:

XML-документ визначає структуру інтерфейсу для функції декодування стеганографічних повідомлень. Основні характеристики:

1. Ієрархічна структура:
    - Кореневий елемент StackPane з класом стилю "root" слугує базовим контейнером
    - Основне вміст розміщене у VBox (вертикальний контейнер) з класом "content-container"
    - Форма організована через GridPane з 3 колонками та 3 рядками

2. Ключові елементи UI:
    - Вибір файлу:
        - Поле шляху (filePathField) лише для читання
        - Кнопка "Огляд" (browseButton) з обробником #handleBrowse
    - Переглядач зображень:
        - ImageView з фіксованими розмірами 150x150px
        - Збереження пропорцій зображення
    - Вивід повідомлення:
        - Багаторядкове поле TextArea (messageArea)
        - Нередагуване, з можливістю прокрутки
    - Кнопки дій:
        - Основна кнопка "Декодувати" (decodeButton)
        - Другорядна кнопка "Скасувати" (cancelButton)
3. Міжнародна підтримка:
    - Усі текстові елементи мають fx:id для подальшої локалізації через контролер

Останні контролери й файли розмітки мають схожі елементи й реалізовані схожим чином.

3.3.2 Реалізація графічної частини

За допомогою Scene Builder вже розроблювався інтерфейс й компонувався згідно якості й чіткості використання.

Для прикладу розглянемо головну сторінку (див. Рис. 21):

На ній бачимо лише «сухий» вигляд, без CSS оформлення. На головному інтерфейсі маємо дві основні кнопки та три другорядні. Основні вміщають в себе: «Закодувати повідомлення» та «Витягнути повідомлення», а другорядні — меню налаштувань, меню «о програмі» та кнопка меню безпечного виключення застосунку. Для основних кнопок використовується

прості надписи, а для другорядних — піктограмки, з ресурсів проекту, через що наочність використання користувачем зростає в рази.

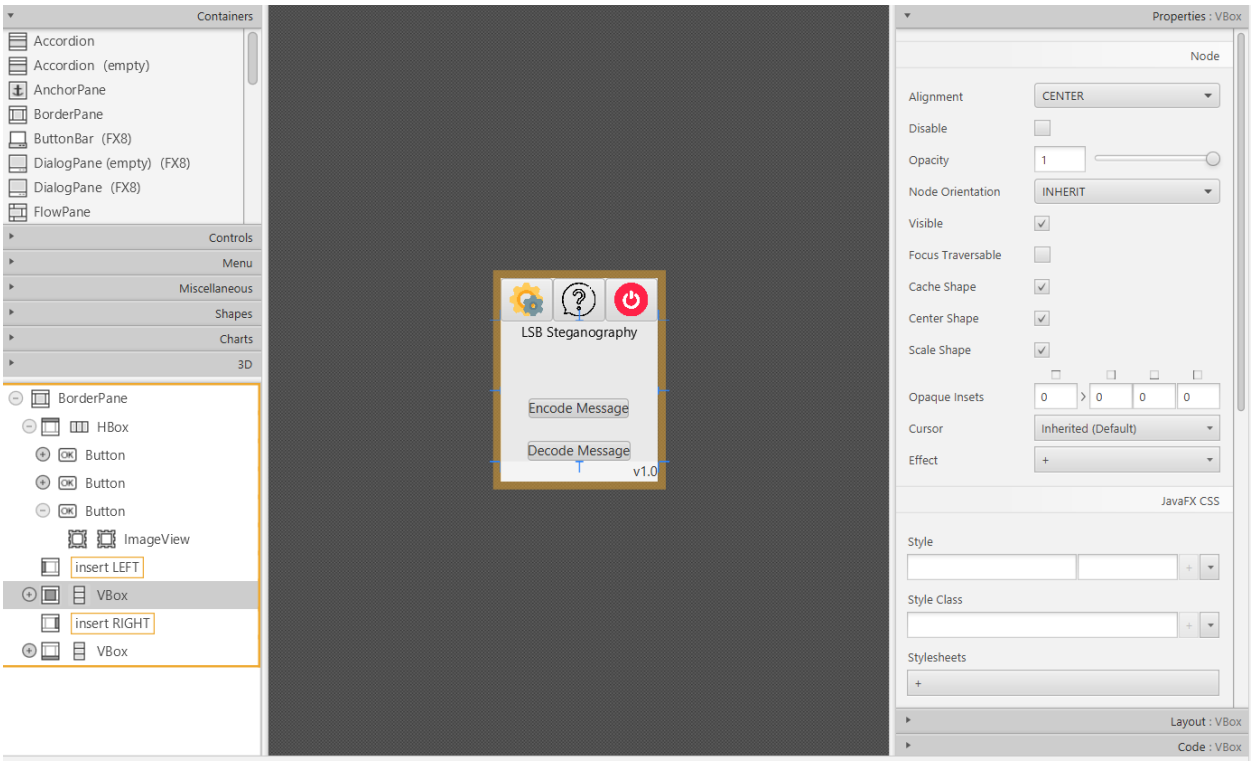


Рисунок *21 — Вигляд головного вікна у Scene Builder.*

Подібним чином розроблено й інші віконця, завдяки Scene Builder це робити значно легше, він одразу генерує код файлу й тільки якщо щось не та (що буває рідко) можна власноруч змінювати код й редагувати безпосередньо файл.

Не мало важливим моментом у створенні сучасного інтерфейсу відіграє роль CSS стилізація. Як можна було побачити раніше, «сухий» вигляд інтерфейсу не дуже задовольняє поставленні задачі. Тому (див. Лістинг 17) як і з веб розробкою інтерфейсу — потрібно описувати стилі і оформлення для кожного елементу інтерфейсу.

Лістинг 17 — *Оформлення інтерфейсу для заголовків тексту.*

```
/* 1. Titles and Text Styles */
.title-text,
.splash-title {
    -fx-font-size: 48px;
```

```
    -fx-font-weight: bold;
    -fx-fill: #FFFFFF; /* Light text for gradient
backgrounds */
    -fx-effect: dropshadow(gaussian, rgba(0, 0, 0, 0.5),
10, 0.5, 0, 1);
}
```

Задаємо:

Великий жирний шрифт розміром 48 px.

Білий колір тексту для читабельності на градієнтному фоні.

М’яку тінь (Gaussian drop shadow) під текстом для об’ємного ефекту.

Таким чином оформлюємо спочатку «світлу тему», а потім вже на її основі, але в протилежному стилі та кольорах — «темну тему» .

3.3 Демонстрація готового застосунку

Після ретельної підготовки проекту, збірки частин до купи, відлагодження проблем та багів, обробки exeptions та апробаціх усіх функцій — програмний продукт набув наступного вигляду:

Під час першої загрузки висвічується заставка з привітальним написом назви програми (див. Рис. 22).

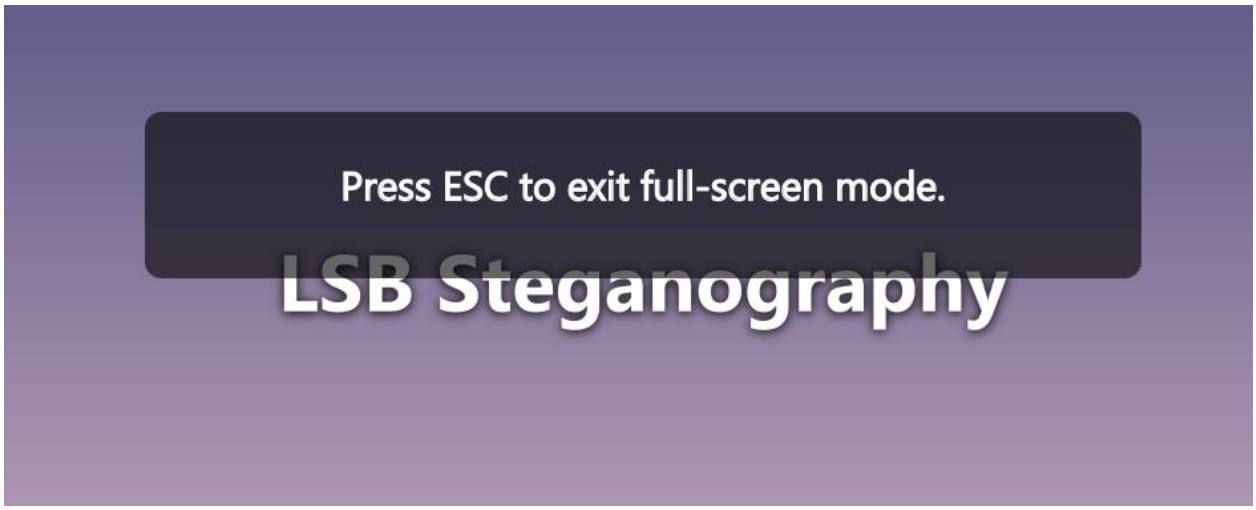


Рисунок *22 — Заставка при включенні застосунку.*

На весь екран, з анімацією і окремим віконцем сповіщення про вихід з повноекранного режиму.

Потім, з’являться основне меню (див. Рис. 23):

Тепер вже, завдяки стилям, картинка інтерфейсу є гарною й зрозумілою. Якщо розтягнути її або стягнути — аномалії не відбудеться й картинка з надписами буде такою же гарною й чіткою.

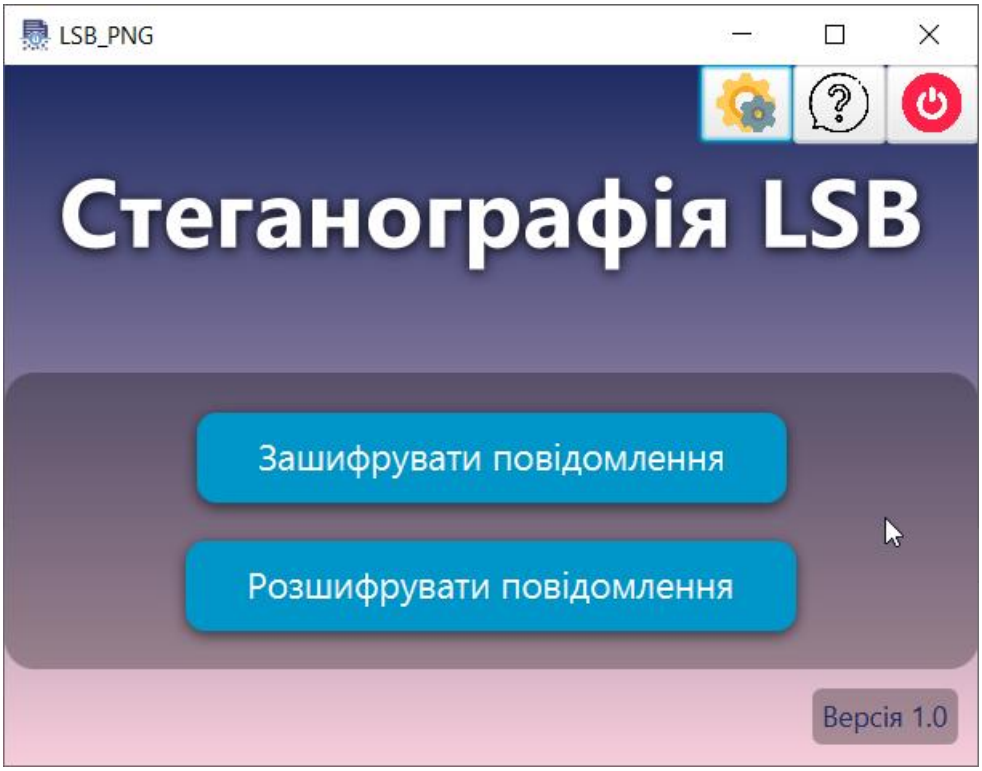


Рисунок *23 — Вигляд головного вікна зі стилями.*

При натисканні другорядної кнопки налаштувань — відкриється таке «підвіконце», що дозволить змінювати тему, мову та переглядати логи програми, або закрити це віконце (також, його можна закрити клікнувши поруч, як в сучасних інтерфейсах, щоб дати користувачеві декілька шляхів закриття контенту). Для демонстрації зміни теми — перемкнемо на темну (див. Рис. 24).

Спостерігаємо переключення теми фону оформлення й шрифтів, щоб зберегти контрастність й читабельність тексту.

При натисканні кнопки «Кодувати повідомлення» — відкривається нове віконце (повнорозмірне) в якому обираємо файл для прикладу й пишемо повідомлення, яке плануємо вбудувати (див. Рис. 25).

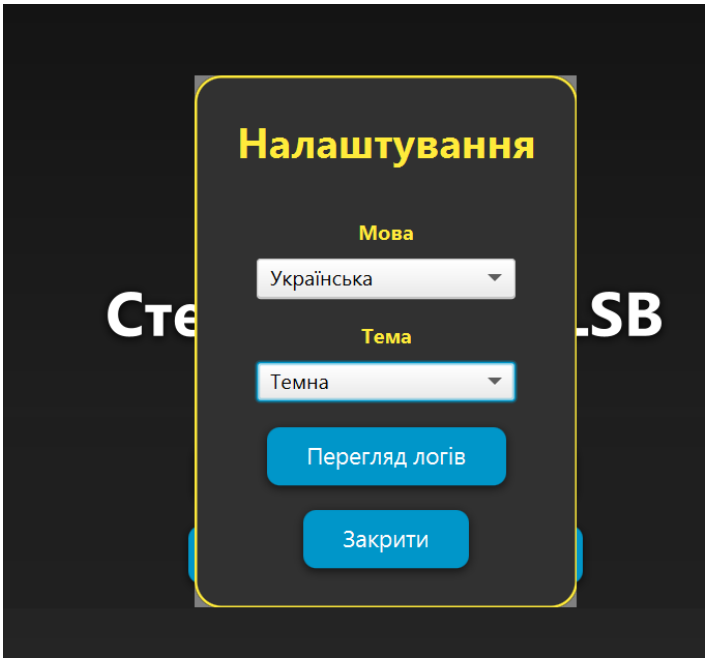


Рисунок *24 — Вигляд темної теми після переключення.*



Рисунок *25 — Вікно вбудовування інформації.*

Натискаємо зашифрувати й відкривається наступне віконце з результатами роботи (див. Рис. 26):



Рисунок *26 — Результат стеганографії.*

На малюнку зображене як оригінальне зображення, так і стего-зображення, для наглядного порівняння на «око» та за розмірами, що, доречі співпадають, завдяки нашому алгоритму й вбудовування складніше відстежити, що і є головною метою стеганографії.

Після натискання кнопки «Повернутись» й вибору розшифрування зображення — відкривається нове вікно, в якому після вибору нашого стегоносія й розшифровки — отримуємо рівно те повідомлення, що й кодували! (див. Рис. 27)



Рисунок *27 — Витягнуте повідомлення з файлу зображення.*

3.4 Висновки до розділу

1. Архітектурна цілісність: Проект використовує MVC з чітким поділом логіки
2. Ефективність LSB-реалізації: Оптимізована робота з PNG через PngJ
3. Юзер-френдлі інтерфейс:
   Анімації та переходи
   Контекстні меню історії
   Динамічна валідація введення
4. Гнучкість налаштувань:
   Можливість зміни мови/теми без перезапуску
5. Стабільність:
   Комплексна обробка помилок у всіх критичних операціях.

ВИСНОВКИ

1. У ході виконання роботи було проведено вивчення принципів стеганографії: досліджено алгоритми приховування інформації в зображеннях й визначення вимог до застосунку.
2. Досліджено можливості та переваги платформи JavaFX для побудови GUI-застосунків: розглянуто поняття графу сцени, основні віджети та лейаути, методи CSS-стилізації, механізми обробки подій, використано FXML та JavaFX Scene Builder.
3. Проєктовано архітектуру застосунку для стеганографії в PNG-зображеннях із використанням JavaFX за MVC патерном із розділенням на модель, представлення та контролери, бізнес-логіку LSB-алгоритму для приховування та витягання повідомлень.
4. Реалізовано додаткові функції та можливості, що визначались як опціональні: декілька мов, різні теми оформлення та логування дій користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Собко Б.С., бакалавріат, Коломоєць Г.П., доцент — науковий керівник. Розробка застосунку для стеганографії у файли зображень із використанням засобів JavaFX. *Молода наука 2025:* зб. науков. праць студентів, аспірантів, докторантів і молодих вчених. Запоріжжя : ЗНУ, 2025. Т.5. 416 – 418 с.
2. К89 Стеганографія : навчальний посібник / О. О. Кузнецов, С. П. Євсеєв, О. Г. Король. Х. : Вид. ХНЕУ, 2011. 232 с. (Укр. мов.) URL: https://repository.hneu.edu.ua/bitstream/123456789/2289/1/Стеганография.pdf (дата звернення: 09.06.2025).
3. Digital watermarking and steganography/Ingemar J. Cox ... [et al.]. p. cm. Includes bibliographical references and index. ISBN 978-0-12-372585-1 (casebound: alk. paper) 1. Computer security. 2. Digital watermarking. 3. Data protection. I. Cox, I. J. (Ingemar J.) QA76.9.A25C68 2008 005.8–dc22 2007040595 ISBN 978-0-12-372585-1 URL: https://www.softouch.on.ca/kb/data/Digital%20Watermarking%20and%20Steganography%202E.pdf (дата звернення: 09.06.2025).
4. Cvejic, Nedeljko, Algorithms for audio watermarking and steganography Department of Electrical and Information Engineering, Information Processing Laboratory, University of Oulu, P.O.Box 4500, FIN-90014 University of Oulu, Finland 2004 Oulu, Finland URL: https://www.mp3-tech.org/programmer/docs/isbn9514273842.pdf (дата звернення: 09.06.2025).
5. Хорошко, В. О. Комп'ютерна стеганографія: навчальний посібник / В. О. Хорошко, Ю. Є. Яремчук, В. В. Карпінець Вінниця: ВНТУ, 2017. – 155 с. 478761 УДК 004.056.5 © ВНТУ, 2017 URL: https://pdf.lib.vntu.edu.ua/books/2018/Xoroshko_Komputer_2017_155.pdf (дата звернення: 09.06.2025).

6. Multimedia Security: Tools, Techniques, and Applications Bhaskar Mondal, Shyam Singh Rajput URL: https://mirknig.su/knigi/seti/685713-multimedia-security-tools-techniques-and-applications.html (дата звернення: 09.06.2025).
7. Вишня В. Б. В 55 Основи інформаційної безпеки : навч. посібник / В. Б. Вишня, О. С. Гавриш, Е. В. Рижков. Дніпро : Дніпроп. держ. ун-т внутріш. справ, 2020. 128 с. ISBN 978-617-7665-08-2 URL: https://er.dduvs.edu.ua/bitstream/123456789/4206/1/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%B8%20%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%BE%D1%97%20%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D0%BA%D0%B8%20%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9%20%D0%BF%D0%BE%D1%81%D1%96%D0%B1%D0%BD%D0%B8%D0%BA%2006.2019%20%283%29.pdf (дата звернення: 09.06.2025).
8. Кузнецов О. О. К89 Стеганографія : навчальний посібник / О. О. Кузнецов, С. П. Євсеєв, О. Г. Король. – Х. : Вид. ХНЕУ, 2011. – 232 с. (Укр. мов.) ISBN УДК 004.67(075.8) ББК 32.973я73 URL: https://repository.hneu.edu.ua/bitstream/123456789/2289/1/%D0%A1%D1%82%D0%B5%D0%B3%D0%B0%D0%BD%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F.pdf (дата звернення: 09.06.2025).
9. The International Arab Journal of Information Technology, Vol. 17, No. 1, January 2020 URL: https://iajit.org/PDF/%20Vol%2017,%20No.%201/16412.pdf (дата звернення: 09.06.2025).
10. SteganoGAN GitHub URL: https://github.com/DAI-Lab/SteganoGAN?tab=readme-ov-file (дата звернення: 09.06.2025).
11. OpenStego Concept URL: https://www.openstego.com/concepts (дата звернення: 09.06.2025).

12. GitHub StegIt URL: https://github.com/feresg/java-steganography (дата звернення: 09.06.2025).
13. Васильев О.М. В19 Програмування мовою Java / О.М. Васильев.- Тернопіль: Навчальна книга Богдан, 2019. 696 с.; іл. ISBN 978-966-5879-7 URL: https://ru.scribd.com/document/658584505/java-ukr-cont (дата звернення: 09.06.2025).
14. The Definitive Guide to Java Swing, Third Edition Copyright © 2005 by John Zukowski All rights reserved. ISBN (pbk): 1-59059-447-9 URL: https://www.cs.buap.mx/~ygalicia/Libros/JavaSwing.pdf (дата звернення: 09.06.2025).
15. JavaFX Documentation Project Published 2022-06-26 version unspecifiedGitHub URL: https://fxdocs.github.io/docs/html5/ (дата звернення: 09.06.2025).
16. JavaFX Layout Controls URL: https://dev.java/learn/javafx/layout/ (дата звернення: 09.06.2025).
17. The Definitive Guide to Modern Java Clients with JavaFX: Cross-Platform Mobile and Cloud Development Stephen Chin Belmont, CA, USA James Weaver Marion, IN, USA Johan Vos Leuven, Belgium ISBN-13 (pbk): 978-1-4842-49 ISBN-13 (electronic): 978-1-4842-4926-0 URL: https://books.google.nl/books?id=Ge69DwAAQBAJ&printsec=copyright&redir_esc=y#v=onepage&q&f=false (дата звернення: 09.06.2025).
18. Java SE 21 Archive Downloads URL: https://www.oracle.com/java/technologies/javase/jdk21-archive-downloads.html (дата звернення: 09.06.2025).
19. Java FX Documentation URL: https://openjfx.io/ (дата звернення: 09.06.2025).
20. Maven Doc. URL: https://maven.apache.org/what-is-maven.html (дата звернення: 09.06.2025).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Собко Богдан Сергійович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення освітньої програми “Програмне забезпечення систем”, адреса електронної пошти se121-21@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Розробка застосунку для стеганографії у файли зображень із використанням засобів JavaFX**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 02.06.2025 _____ Собко Богдан Сергійович
(підпис) (прізвище та ініціали) (студент)

Дата 02.06.2025 _____ Коломоєць Геннадій Павлович
(підпис) (прізвище та ініціали) (керівник)