Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп'ютерних наук

Спеціальність 125 «Кібербезпека»

Освітня програма «Безпека інформаційних та комунікаційних систем»

«Допущено до захисту»

В.о. зав. кафедрою БІСТ

Ольга МЕЛКОЗЬОРОВА

_____

« » 2023 р.

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Телеграм-бот, що використовує стеганографічні алгоритми для приховування інформації в цифрових контейнерах»

оцінка « »

Голова ЕК

Олександр ЛЕМЕШКО _____

Керівник к.т.н, доцент Громико І.О.

Рецензент

д.т.н, професор Краснобаєв В.А.

Виконавець: студент групи КБ-61

Коршенко Владислав Сергійович

Харків – 2023

РЕФЕРАТ

Дипломна робота другого (магістерського) рівня вищої містить 111 сторінок, 45 рисунків, 3 таблиці. Перелік посилань нараховує 11 найменувань.

Актуальність роботи. Актуальність теми зумовлена наявністю потреби в розробці та реалізації нових закритих каналів зв'язку для комунікації.

Метою роботи. Метою роботи є розробка закритого каналу зв'язку на основі платформи для миттєвого обміну повідомленнями Телеграм з застосуванням стеганографічних алгоритмів для забезпечення конфіденційності, цілісності та доступності інформаційних повідомлень.

Результати роботи та їх новизна. Результатом роботи є програмний продукт, що дозволяє приховувати повідомлення у цифрових зображення, що, на відміну від криптографічних методів, скриває сам факт передачі повідомлення. ПЗ має покращений стегано-алгоритм, що бореться із результатами таких геометричних атак, як поворот, віддзеркалення зображення та відсікання граничних областей.

Рекомендації щодо використання результатів роботи. Створений програмний продукт, та алгоритми, за якими він працює, можуть бути використані для обміну повідомленнями у випадках, коли є необхідність скрити сам факт передачі повідомлення.

Значущість роботи та висновки. Для України є важливим розвиток цифрових технологій, в тому числі технологій комунікації як для громадян, так і для спеціальних служб, де фактор збереження конфіденційності, цілісності та доступності є ключовим.

Припущення про можливі напрямки розвитку. Стеганографічний алгоритм, що реалізований у даному програмному продукті є покращеною версією стеганографічного алгоритму Куттера-Джордана-Боссена. На його основі можна створювати вдосконалені алгоритми, щоб протистояти ширшому спектру атак.

Ключові слова: ТЕЛЕГРАМ-БОТ, СТЕГАНОГРАФІЧНІ АЛГОРИТМИ, ПРИХОВУВАННЯ ІНФОРМАЦІЇ, ЦИФРОВІ ЗОБРАЖЕННЯ, БЕЗПЕЧНА КОМУНІКАЦІЯ, КОНФІДЕНЦІЙНІСТЬ.

ABSTRACT

The thesis of the second (master's) level of higher education contains 89 pages, 25 figures, 1 table. The list of references includes 10 items.

Relevance of the study. The relevance of the study is determined by the necessity of the development and implementation of new secure communication channels.

The purpose of the study. The purpose of the study is to develop a secure communication channel based on the Telegram instant messaging platform using steganography algorithms to ensure the confidentiality, integrity and availability of information messages.

Results and novelty. The result of the work is a software product that allows you to hide messages in digital images, which, unlike cryptographic methods, hides the very fact of message transmission. The software has an improved steganography algorithm that combats the results of such geometric attacks as rotation, image reflection, and cutting off of boundary regions.

Recommendations for using the results of the work. The created software product and the algorithms by which it works can be used for messaging in cases where there is a need to hide the fact of message transmission.

Significance of the work and conclusions. The development of digital technologies, including communication technologies for both citizens and special services, is important for Ukraine, where the factor of confidentiality, integrity and accessibility is key.

Assumptions about possible directions of development. The steganography algorithm implemented in this software product is an improved version of the Kutter-Jordan-Bossen steganography algorithm. On its basis, it is possible to create improved algorithms that will resist a wider range of attacks.

Keywords: TELEGRAM BOT, STEGANOGRAPHY ALGORITHMS, INFORMATION HIDING, DIGITAL IMAGES, SECURE COMMUNICATION, PRIVACY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ	-	програмне забезпечення,
Телеграм	-	кросплатформенна система миттєвого обміну повідомленнями з функціями обміну текстовими, голосовими та відео-повідомленнями, а також стікерами, фотографіями та файлами багатьох форматів,
Телеграм-бот	-	окремий обліковий запис у Telegram, який самостійно відповідає на повідомлення користувачів,
Апдейт	-	від англ. update – оновлений пакет даних,
IP	-	унікальний числовий ідентифікатор пристрою в комп'ютерній мережі,
LSB	-	least significant bit,
AES	-	advanced encryption standard,
PBKDF2	-	password-based key derivation function,
SHA256	-	secure hash algorithm,
F5	-	назва стеганографічного алгоритму,
MTProto	-	криптографічний протокол, який використовується в системі обміну повідомленнями Telegram,
IP	-	internet protocol,
API	-	application programming interface,
Unicode	-	стандарт кодування символів.

ВСТУП

Кібербезпека є однією з найважливіших сфер сучасного світу, яка стає все більш актуальною в контексті військових операцій. Вона необхідна не лише для боротьби з хакерами та кіберзлочинцями, але й для досягнення перемоги в збройних конфліктах. З відкритих програм телебачення прозвучало, що і Збройні Сили України (ЗСУ), і ворожі сили використовують однотипні прилади зв'язку, що створені на заводах західної Європи. Станом на даний момент, деякі «польові» радіостанції для ЗСУ і, одночасно, держава-агресор вільно придбає через фірми-прокладки. Це вказує на те, що обидві сторони можуть прослуховувати один одного, що створює серйозні потенційні загрози для безпеки інформації з обмеженим доступом.

Саме тому поява закритих (захищених) каналів зв'язку, до яких не мають доступу вороги, стає важливим завданням. Створення таких каналів зв'язку є актуальною проблемою, яка вимагає розробки нових технологій та підходів. Один із потенційних методів вирішення цієї проблеми полягає в застосуванні стеганографічних алгоритмів для приховування інформації в цифрових зображеннях.

Метою даної магістерської роботи є розробка та створення телеграм-боту, який використовує стеганографічні алгоритми з метою приховування інформації в цифрових зображеннях. Основний сенс роботи полягає у створенні ефективного програмного застосунку, що забезпечує конфіденційність передачі інформації та є зручним у користуванні. Результати цього дослідження можуть мати важливе значення для забезпечення безпеки комунікаційних систем, а також для підвищення їх ефективності в умовах сучасних кіберзагроз.

1 ФОРМУЛЮВАННЯ ВИМОГ ДО СТВОРЮВАНОГО ПЗ

У процесі розробки ПЗ були встановлені основні критерії для вибору програмних рішень. Ці критерії враховують вимоги до безпеки та простоти користування.

Першим і найважливішим критерієм є конфіденційність передачі інформації. Приховування даних в зображеннях повинно забезпечувати високий рівень безпеки та недоступність для сторонніх осіб.

Другим критерієм є анонімність користувачів. Застосунок повинен забезпечувати можливість взаємодії між користувачами без необхідності розголошувати їх особисту інформацію.

Третім критерієм є крипостійкість. Алгоритми, що використовуються для обміну повідомленнями, мають бути стійкими до розшифрування та атак з боку противника.

Нарешті, останнім критерієм була простота користування. Розроблений програмний застосунок повинен бути легким у використанні, навіть для недосвідчених користувачів, щоб забезпечити зручну і ефективну комунікацію.

2 АНАЛІЗ ТА ДОСЛІДЖЕННЯ МЕТОДІВ ЗАХИСТУ ПЕРСОНАЛЬНИХ ДАНИХ

У процесі аналізу існуючих програмних рішень для стеганографічного приховування інформації в зображеннях, з метою організації закритих каналів зв'язку, було виявлено ряд програм, що вирішують цю проблему, але з обмеженою ефективністю та функціональністю.

Нижче наведена порівняльна таблиця найпопулярніших стеганографічних застосунків (Таблиця 2.1):

Таблиця 2.1 – Порівняльна таблиця найпопулярніших стеганографічних застосунків

Назва застосунку	Платформа	Стеганографічні алгоритми	Додаткові алгоритми безпеки
StegOnline	Веб	LSB	Відсутні
Steganography Online	Веб	LSB	Відсутні
Steganography Online Codec by PELock	Веб	LSB	AES, PBKDF2
ImageSteganography	Веб\Консоль	LSB	SHA256
Hide’N’Send	Windows	F5, LSB	AES, SHA512
Hallucinate	Windows	LSB	Відсутні

Як можна побачити, усі перелічені застосунки використовують стеганографічний алгоритм LSB, що є першим недоліком. Цей алгоритм не є стійким до стиснення зображення. Ще однією проблемою при використанні цього алгоритму є його низька стійкість до атак з метою виявлення повідомлення або його знищення.

Також недоліком є сама платформа на якій працюють описані застосунки. Веб-браузер чи Windows – ці платформи не мають вбудованих засобів передачі повідомлень між користувачами, що призводить до потреби інсталяції застосунків для комунікації, що в свою чергу подовжує ланцюжок кроків, які необхідно виконати для передачі повідомлення.

Таким чином, в результаті пошуку та аналізу готових застосунків, таких, які б відповідали встановленим вимогам, виявлено не було.

3 ОБРАНА ПЛАТФОРМА ДЛЯ РЕАЛІЗАЦІЇ

З урахуванням вищезазначених критеріїв, було прийнято рішення про використання месенджеру Телеграм, як платформи для розробки боту. Цей вибір був обґрунтований низкою факторів, що позитивно впливають на захищеність даних, анонімність користувачів та зручність користування.

По-перше, Телеграм використовує криптографічний протокол MTProto для забезпечення конфіденційності передачі даних.

Крім того, месенджер має відкритий вихідний код API (прикладного програмного інтерфейсу), що дозволяє розробникам створювати безпечні програмні рішення для комунікації.

По-друге, практика показала, що більшість військових використовують смартфони для обміну тактичною та особистою інформацією, зокрема у месенджері Телеграм. Враховуючи цей факт, створення боту на цій самій платформі усуває необхідність завантаження стороннього програмного забезпечення та час, що зазвичай потрібен для його освоєння.

Важливо зазначити, що Телеграм не виклав у відкритий доступ вихідний код серверної частини месенджеру, що в свою чергу виключає можливість об’єктивно оцінити рівень захищеності даних користувачів. Але цей факт не є підставою для відмови від використання цього месенджеру, як основи для створення закритого каналу зв’язку. Потенційну проблему витоку даних можна вирішити засобами стеганографії, що і є темою даної науково-дослідницької роботи.

Кросплатформеність Телеграму також є однією із переваг обраного месенджеру. Він є доступним на платформі Windows, Android, IOS, Linux та Web (користування через інтернет-браузер).

4 ТЕЛЕГРАМ БОТ

Роботи або боти – це спеціальні аккаунти (облікові записи) в Телеграм, які можуть автоматично обробляти і відправляти повідомлення. Боти можуть виконувати практично будь-які завдання, які може робити кожен користувач акаунту Телеграм з онлайн-сервісами.

В контексті обраної предметної області потрібно виконувати такі операції, як прийом усіх можливих видів повідомлень: текстові, цифрові зображення, відео, аудіо, стікери; обробка прийнятих повідомлень, відправка текстових відповідей, цифрових зображень.

Саме тому автоматична обробка запитів користувачів є єдиним оптимальним варіантом.

5 СТЕК ТЕХНОЛОГІЙ, ЩО БУЛИ ВИКОРИСТАНІ

Програмна реалізація була виконана мовою програмування Java з використанням фреймворку Spring. Ця комбінація дозволяє ефективно створювати додатки з мікросервісною архітектурою, де великий застосунок розбивається на невеликі, незалежні та автономні компоненти, які називаються мікросервісами. Кожен мікросервіс виконує конкретну функціональність та має власну базу даних, комунікацію та інші ресурси. Обрана архітектура дозволить в майбутніх версіях ПЗ винести стеганографічні алгоритми в окремий мікросервіс. Це може бути корисним при потребі в розміщені стеганографічного шифратора та дешифратора на окремому сервері в цілях створення додаткового шару безпеки.

Для взаємодії ПЗ і месенджеру Телеграм був використаний TelegramBotsAPI – прикладний програмний інтерфейс, що створений розробниками самого месенджеру. Цей інтерфейс містить програмні методи, що дозволяють налагодити прийом даних від користувача, а саме вибір режиму роботи ПЗ, текст для приховування, зображення-контейнер, та відправку зворотних даних, таких, як список доступних дій, зображення-контейнер із прихованим текстом, повідомлення про помилку.

Для розгортання ПЗ на сервері була використана система контейнеризації Docker, що дозволяє суттєво спростити сам процес розгортання.

Важливою перевагою створеного програмного застосунку є те, що процеси приховування і вилучення повідомлення відбуваються на одному сервері, де розміщений бот, а не є окремим додатком, що потрібно завантажувати на смартфон. Таким чином, алгоритм приховування інформації доступний лише на цьому сервері, що унеможливлює використання реверс-інжинірингу для аналізу вихідного коду застосунку, тому що користувач не має доступу до вихідного коду.

В процесі роботи користувача із ПЗ, дані рухаються наступним ланцюжком:

1) Від користувача на сервер Телеграм;
2) Від серверу Телеграм на сервер на якому працює ПЗ;
3) Від серверу на якому працює ПЗ на сервер Телеграм;

4) Від серверу Телеграм до користувача;

В момент виконання кожного кроку, дані користувача захищені протоколом MTProto [1], а отже передаються у зашифрованому вигляді.

Додатковим шаром безпеки є те, що передача даних відбувається не відразу на сервер ПЗ, а через проміжний сервер самого месенджеру Телеграм. Це заважає потенційним зловмисникам дізнатись IP адресу серверу, на якому працює стеганографічний шифратор та дешифратор у складі ПЗ.

6 ОПИС ВНУТРІШНЬОЇ СТРУКТУРИ СТВОРЕНОГО ПЗ

Програмний застосунок на 99.8% (Рисунок 6.1) реалізовно засобами високорівневої мови програмування Java з використанням методології ООП – об’єктно орієнтованого програмування.

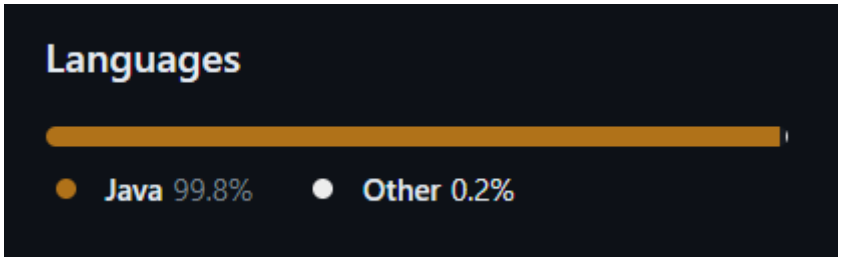

Рисунок 6.1 – Аналітичний звіт із сайту Github.com із описом використаних у проекті мов програмування.

Відповідно до обраної методології та мови програмування були реалізовані системні класи із усім необхідним функціоналом задля забезпечення обміну повідомленнями із користувачем, приховування та вилучення текствоих даних із цифрових зображеннь.

Поняття, які найчастіше використовуються при описі програмного коду:

Клас [2] – спеціальна конструкція, що використовується для групування логічно пов'язаних полів та методів.

Метод [2] – частина програмного коду, що виконує певну фунцію та є частиною класу.

Об’єкт [2] – екземпляр класу.

Далі наведено детальний опис основної частини створених класів, способи їх взаємодії та загальна роль у проекті:

SteganoBotApplication – клас, з якого починається виконання програми. Містить головний метод main(), що запускає виконання програми методом SpringApplication.run(). Діаграма класу показана на Рисунку 6.2.

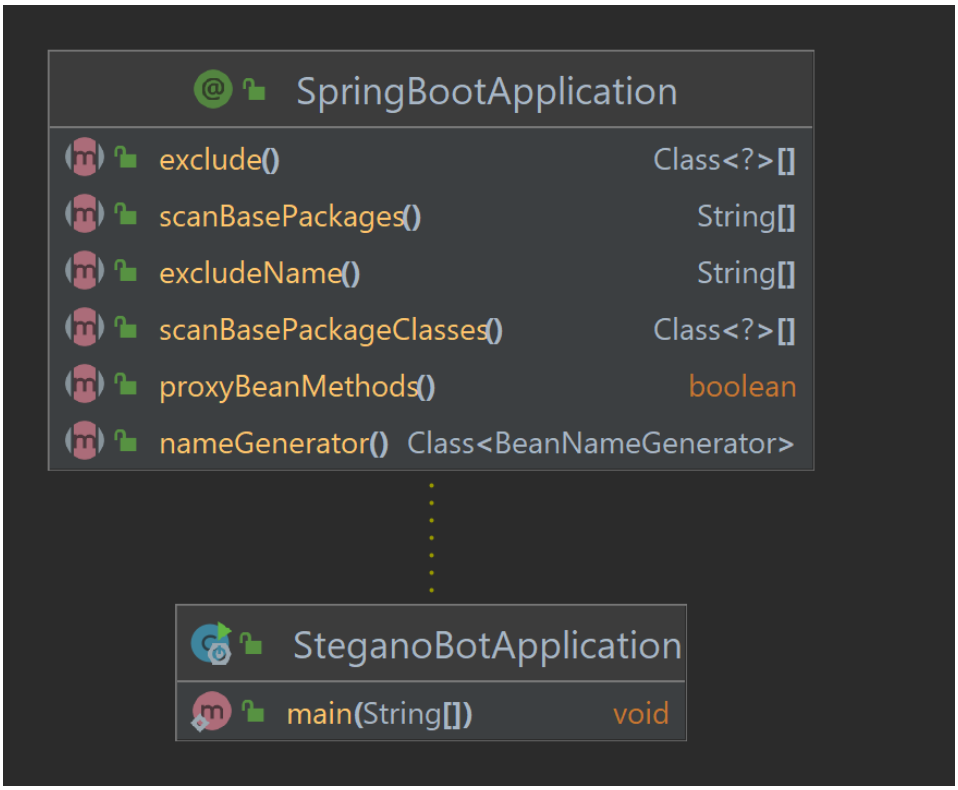

Рисунок 6.2 – Діаграма класу SteganoBotApplication та його зв’язків із іншими сутностями

MySteganoBot – клас, що наслідує [2] клас TelegramWebhookBot прикладного програмного інтерфейсу від розробників Telegram і реалізує [2] методи для обміну утилітарною інформацією (ім’я боту, шлях боту та токен боту) та прийому пакетів даних із повідомленням – апдейтів.

Кожен раз, коли користувач відсилає повідомлення боту, метод onWebhookUpdateReceived() отримує об’єкт апдейту, що містить як утилітарну інформацію (айді чату, айді користувача та інше), так і власне повідомлення користувача.

Далі, обробка повідомлення, а саме відправлення відповіді та клавіатури із опціями меню, інструкцій та підказок користувачу, приховування, вилучення повідомлення починається у методах sendMessage(), sendTip(), encodeIfNeeded(), decodeIfNeeded() відповідно.

Важливо зазначити, що в конструкторі [2] класу відбувається ініціалізація об’єкту класу TelegramFacade, методи якого використовуються у вказаних вище методах.

Метод sendTipImage() використовується для відправлення зображення-підказки (Рисунок 6.3) для користувача, що показує приклад зображення з яким алгоритм буде працювати корректно та приклад зображення, які не слід використовувати.

Рисунок 6.3 – Зображення-підказка для вибора коректного зображення

Метод sendLongMessage(), відповідно до його назви, використовується для відправлення довгих повідомлень – повідомлень, довжина яких перевищує 4096 символів. Обмеження у 4096 символів є внутрішнім обмеженням на довживну повідомлення в Телеграм.

Метод sendImageAsDocument() використовується для відправлення зображення без стиснення – як документ. Такий спосіб передачі зображення важливий, щоб запобігти втраті прихованого повідомлення в ітоговому зображенні при стиненні.

Діаграма класу показана на Рисунку 6.4.

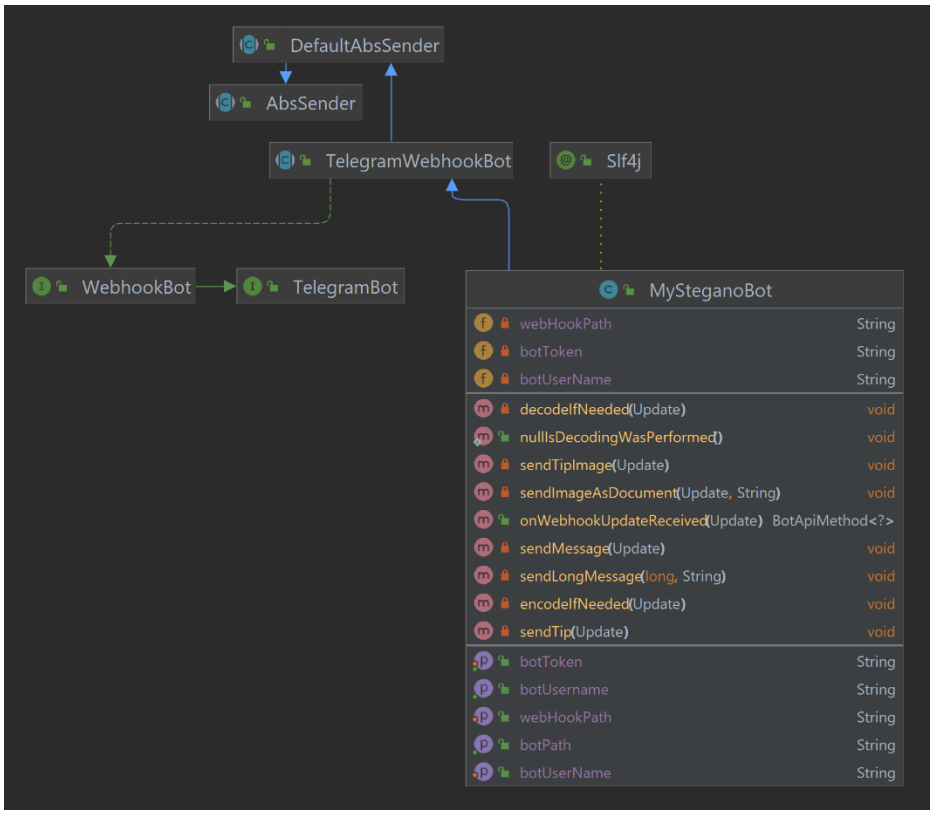

Рисунок 6.4 – Діаграма класу MySteganoBot та його зв’язків із іншими сутностями

TelegramFacade – клас, що виконує роль посередника між класом MySteganoBot та класами BotStateContext, UserDataCache, ініціалізація яких відбувається у конструкторі, та має декілька утилітарних методів.

Метод handleUpdate() використовується, щоб перевірити апдейт на наявність тексту повідомлення, записати в лог результат перевірки і передати керування методу handleInputMessage().

Метод handleInputMessage() отримує поточний стан бота із можливих BotState та передає керування відповідному обробнику повідомлень.

Метод handleTip() виконує дії, аналогічні попередньо описаному методу, для підказок користувача.

Методи isFilesReadyToEncode(), isFilesReadyToDecode() повертають статус готовності файлів до приховування чи вилучення тексту відповідно.

Діаграма класу показана на Рисунку 6.5

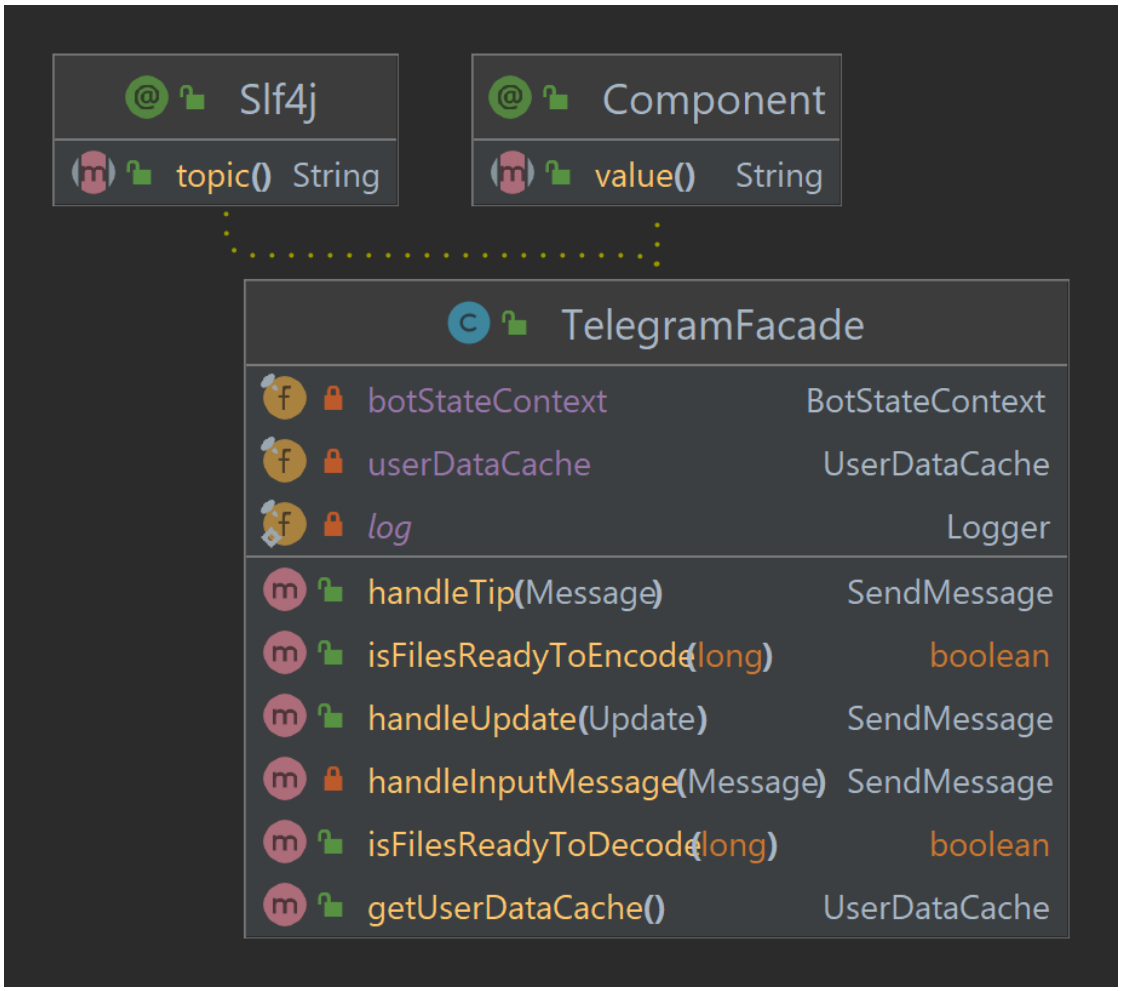

Рисунок 6.5 - Діаграма класу TelegramFacade та його зв’язків із іншими сутностями

BotState – перелічення(enum) [2], що містить список усіх можливих станів Телеграм-боту.

Система станів в створеному ПЗ являє собою кінцевий автомат (finit state machine).

Кінцевий автомат (finit state machine) [3] - математична абстракція, модель дискретного пристрою, що має один вхід, один вихід і в кожен момент часу перебуває в одному стані з множини можливих.

На Рисунку 6.6 наведена діаграма станів створеного ПЗ.

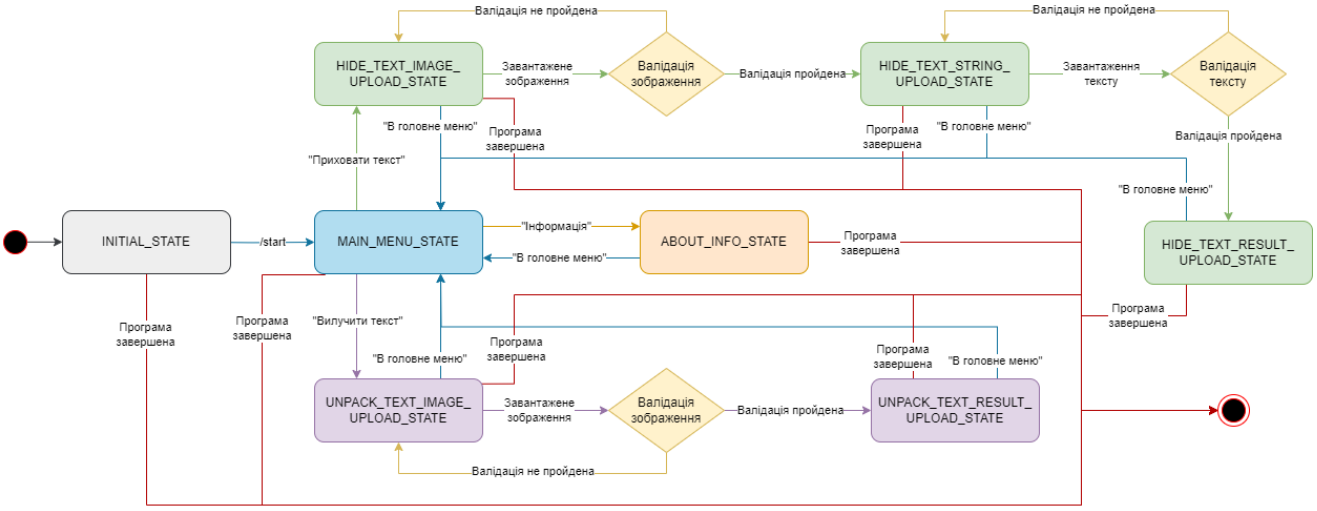

Рисунок 6.6 – Діаграма станів створеного ПЗ

INITIAL_STATE – початковий стан, в якому знаходиться ПЗ відразу після запуску. Для переходу в стан головного меню необхідно ввести команду «/start».

MAIN_MENU_STATE – стан головного меню, в якому користувач отримує підказку «Скористайтесь головним меню» і три кнопки меню з опціями «Приховати текст», «Вилучити текст», «Інформація», при виборі однієї з яких відбувається зміна стану ПЗ відповідно до вибраної.

ABOUT_INFO_STATE – стан інформаційного меню, де користувач отримує основну інформацію та рекомендаціх по користуванню ботом та зображення-підказку (Рисунок 6.3). Єдина доступна опція в даному стані – «В головне меню», що поверне користувача в головне меню.

HIDE_TEXT_IMAGE_UPLOAD_STATE – перший із ланюжка станів приховування тексту. На даному етапі відбувається завантаження зображення від користувача, валідація цього зображення (формат, розмір) і, у випадку успішної валідації, перехід до наступного стану. Також можливий перехід у головне меню.

HIDE_TEXT_STRING_UPLOAD_STATE – другий із ланцюжка станів приховування тексту. На даному етапі відбувається завантаження тексту від користувача, валідація цього тексту (довжина) і, у випадку успішної валідації, перехід до наступного стану. Також можливий перехід у головне меню.

HIDE_TEXT_RESULT_UPLOAD_STATE – фінальний із ланцюжка станів приховування тексту. На даному етапі користувач отримує повідомлення про те, що дані завантажені успішно та обробляються. Результат роботи алгоритму, що

приховує дані у зображенні відправляється користувачу. Єдина доступна опція в даному стані – «В головне меню», що поверне користувача в головне меню.

UNPACK_TEXT_IMAGE_UPLOAD_STATE – перший із двох станів, що відповідають за вилучення тексту із зображення. Після завантаження зображення користувачем, відбувається валідація даного зображення (формат, розмір) і, у випадку і, у випадку успішної валідації, перехід до наступного стану. Перехід у головне меню також можливий.

UNPACK_TEXT_RESULT_UPLOAD_STATE – стан, в якому відбувається сповіщення користувача про результати роботи алгоритму. Користувач може отримати повідомлення про відсутність повідомлення в зображення, про те, що воно було випадково або навмисно пошкоджено, що призвело до повної або часткової втрати повідомлення, і власне повідомлення, що вдалось вилучити. Можливий перехід у головне меню.

На Рисунку 6.7 показана діаграма перелічення BotState.

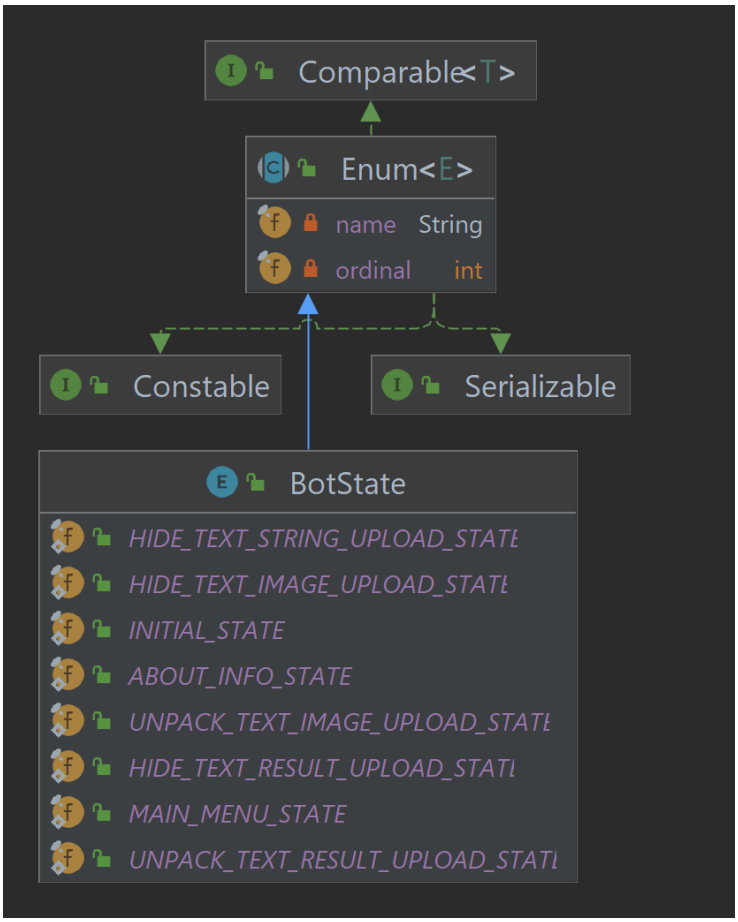

Рисунок 6.7 – Діаграма перелічення BotState та його зв’язків із іншими сутностями

MenuOptions – перелічення (enum) [2], що описує усі можливі опції меню. Для опису кожної опції використовується 4 значення: назва опції, назва текстової локалізації опції, назва стану, на який буде замінено поточний стан, та список станів, де доступна опція. Опис усіх опцій наведено у Таблиці 6.1.

Таблиця 6.1 – Опис усіх доступних опцій створеного ПЗ

Назва опції	Назва текстової локалізації	Текстове представлення	Стан, яким буде замінено поточний стан	Список станів, у яких дана опція буде доступна
1	2	3	4	5
START_OPTION	option.initial-state-valid-option	/start	MAIN_MENU_STATE	INITIAL_STATE
BACK_TO_ MAIN_ MENU_OPTION	option.back-to-main-menu-valid-option	В головне меню	MAIN_MENU_STATE	HIDE_TEXT_IMAGE_ UPLOAD_STATE, HIDE_TEXT_STRING_ UPLOAD_STATE, HIDE_TEXT_RESULT_ UPLOAD_STATE, UNPACK_TEXT_IMAGE_ UPLOAD_STATE, UNPACK_TEXT_ RESULT_ UPLOAD_STATE, ABOUT_INFO_STATE
HIDE_TEXT_ OPTION	option.main-menu-state-hide-text-option	Приховати текст	HIDE_TEXT_IMAGE_ UPLOAD_STATE	MAIN_MENU_STATE
UNPACK_ TEXT_ OPTION	option.main-menu-state-unpack-text-option	Вилучити текст	UNPACK_TEXT_ IMAGE_UPLOAD_ STATE	MAIN_MENU_STATE
ABOUT_INFO_ OPTION	option.main-menu-state-about-info-option	Інформація	ABOUT_INFO_STATE	MAIN_MENU_STATE

Обрана структура опису опцій має за собою ціль логічно поєднати в одному об’єкті усі необхідні дані для відображення опцій та обробки запитів користувача.

Розглянемо використання описаної структури на прикладі опції START_OPTION. Коли користувач відправляє повідомлення боту, відбувається перевірка поточного стану та опцій, що доступні у цьому стані. START_OPTION доступна, як можна побачити з Таблиці 6.1, лише в початковому стані - INITIAL_STATE. Це означає, що телеграм-бот буде реагувати на текстове представлення цієї опції – “/start” – лише у початковому стані. Результатом вибору цієї опцію буде зміна стану на той, що записано у четвертій колонці відповідного рядка Таблиці 6.1 - MAIN_MENU_STATE. Назва тектової локалізації в Таблиці 6.1 – це унікальний ідентифікатор відповідного значення із файлу мовної локалізації, що в даному випадку – файл з українською локалізацією.

На Рисунку 6.8 показана діаграма переліченння MenuOptions.

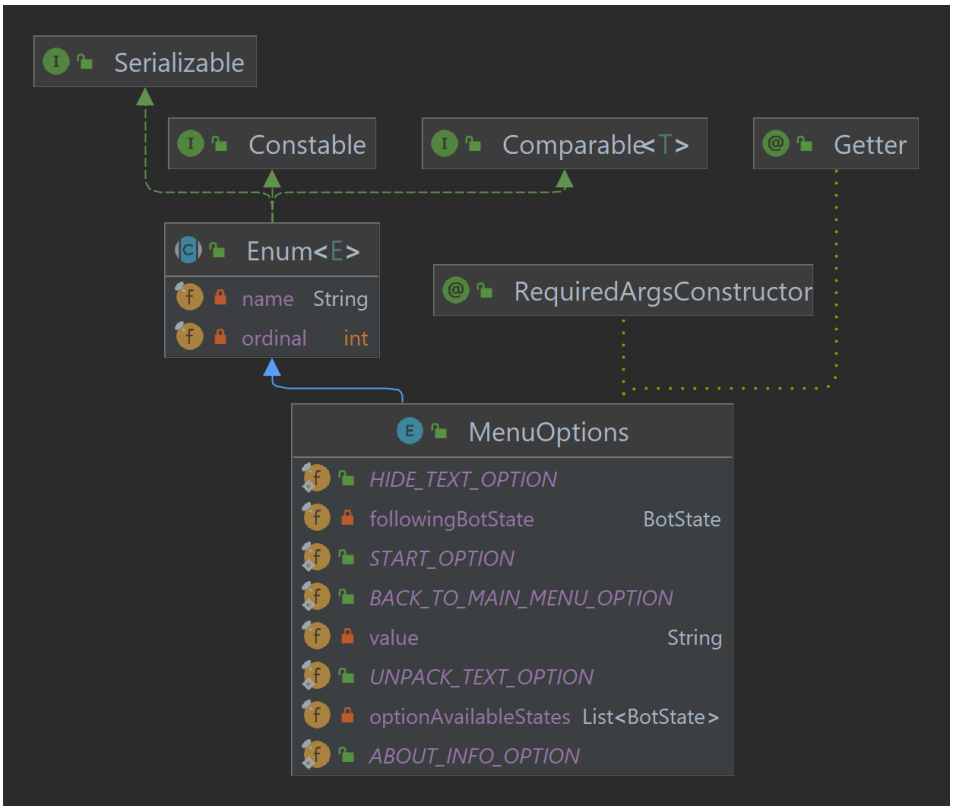

Рисунок 6.8 – Діаграма переліченння та його зв’язків із іншими сутностями

BotStateContext – клас, що виконує роль посередника між станами боту та обробниками станів.

У конструкторі класу відбувається маппінг – прив’язка – усіх обробників до відповідних їм станів.

Також у класі реалізовані два методи, що передають керування обробникам повідомлень (Message) і підказок (Tip).

На Рисунку 6.9 показана діаграма класу BotStateContext.

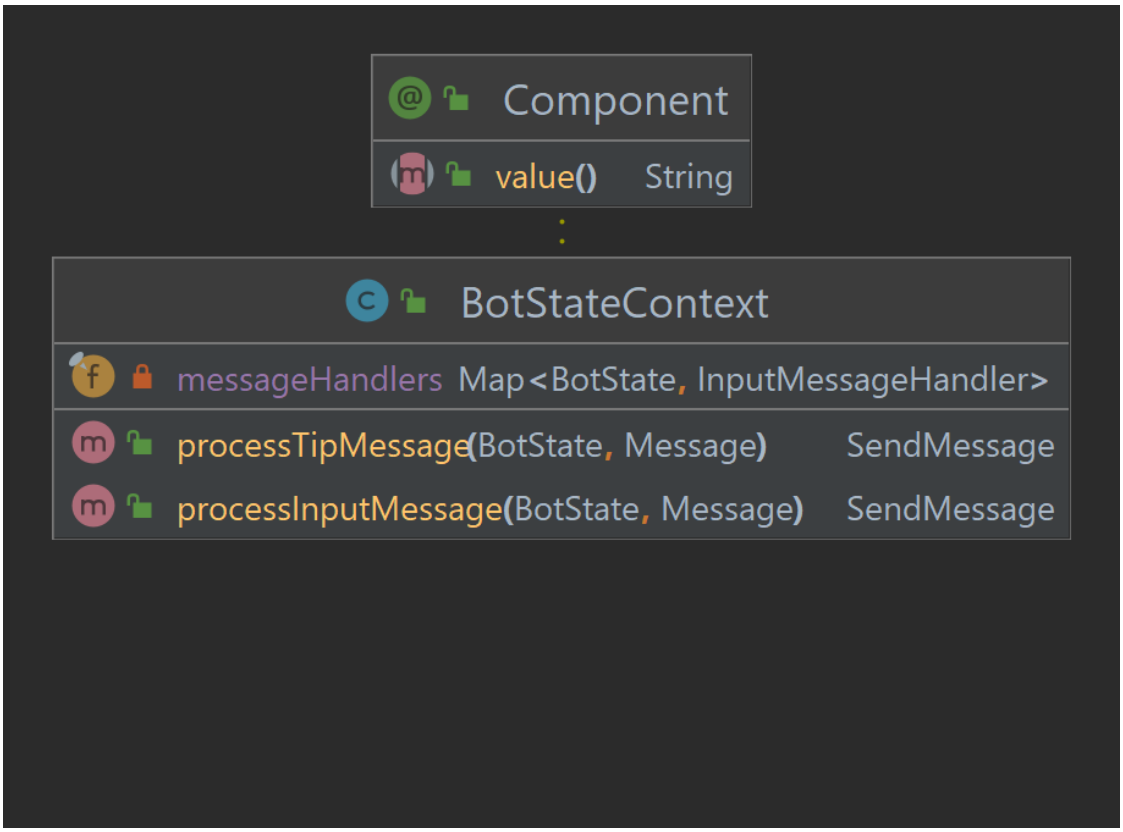

Рисунок 6.9 - Діаграма класу BotStateContext та його зв’язків із іншими сутностями

StateHandler – клас-предок для класів обробників, що реалізує базовий фунціонал обробки повідомлень та зберігає об’єки класів UserDataCache, ReplyMessageService та LocaleMessageService.

CheckMessageForRightOption() – основний метод класу, що обробляє повідомлення користувача, перевіряючи його на наявність опцій меню, і передає керування обробнику відповідно до стану боту.

LogReplyMessage() – метод, що записує в журнал подій відповідь, що була надіслана користувачу.

LogBotStateChange() – метод, що записує в журнал подій зміну стану боту.

InitialHandler – клас, що успадковє клас StateHandler і реалізує інтерфейс InputMessageHandler. Цей клас є одним із класів обробників для відповідних станів боту. Це означає, що кожен з відповідних класів має однаковий набір базових методів для обробки повідомлення користувача.

До базових методів відносяться:

Handle() – метод, інтерфейсу InputMessageHandler, що передає керування методу processUsersInput(). Реалізовано таким чином, щоб мати можливість за необхідності змінювати алгоритм обробки повідомлення, не змінюючи сигнатуру початкового методу.

GetHandlerName() – метод, інтерфейсу InputMessageHandler, що повертає назву стану із переліченння BotState, який обробляє даний клас.

GetStateTip() – метод, інтерфейсу InputMessageHandler, що передає керування методу generateTip(). Реалізовано таким чином, щоб мати можливість за необхідності змінювати алгоритм генерації підказки користувачу, не змінюючи сигнатуру початкового методу.

ProcessUsersInput() – метод, що реалізовує обробку повідомлення користувача, відповідно до особливостей поточного стану боту.

GenerateTip() – метод, що генерую підказку для користувача, відповідно до поточного стану боту.

На Рисунку 6.10 показана діаграма класу InitialHandler.

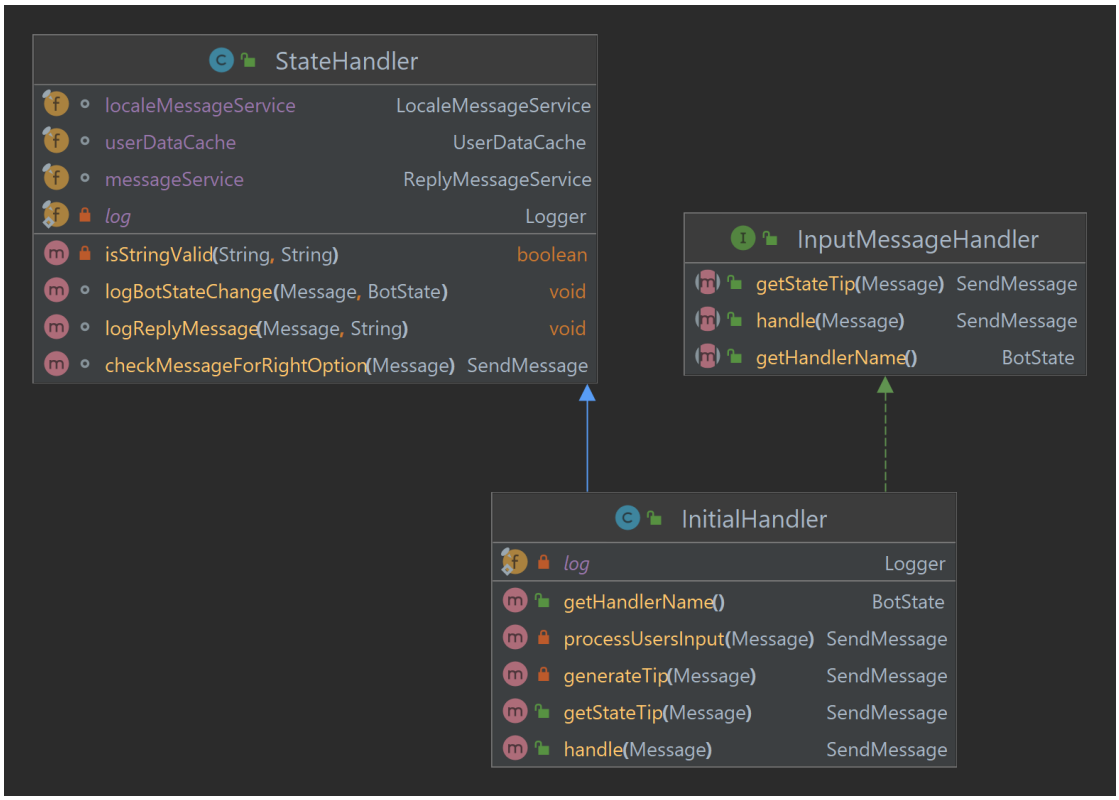

Рисунок 6.10 - Діаграма класу InitialHandler та його зв’язків із іншими сутностями

MainMenuHandler, AboutInfoStateHandler, HideTextStringUploadHandler, HideTextResultUploadHandler, UnpackTextStringUploadHandler, UnpackTextResultUploadHandler – класи, що успадковують клас StateHandler і реалізують інтерфейс InputMessageHandler. Окрім базового набору методів обробки повідомлень, мають додатковий метод generateKeyBoard() для генерації клавіатури кнопок з опціями, доступними у відповідному стані боту.

HideTextImageUploadHandler та UnpackTextImageUploadHandler - класи, що успадковують клас StateHandler і реалізують інтерфейс InputMessageHandler. Окрім базового набору методів обробки повідомлень та методу generateKeyBoard(), мають додатковий метод processUserImage() для обробки зображення, що завантажує користувач.

На Рисунку 6.11 показана повна діаграма класів обробників станів боту.

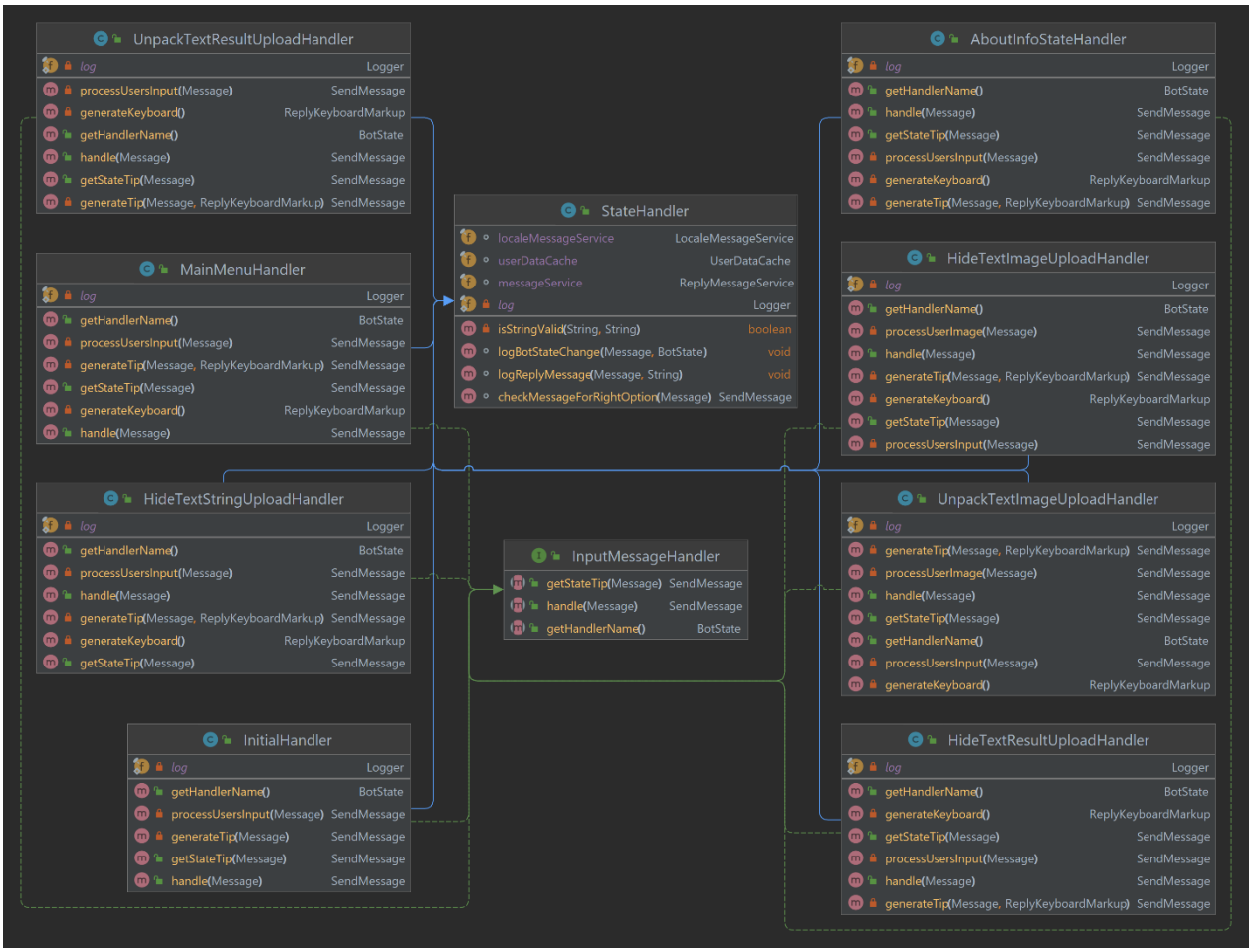

Рисунок 6.11 - Діаграма класів обробників станів боту та їх зв’язків із іншими сутностями

LocalMessageService – клас, що відповідає за завантаження повідомлень відповідно до обраної мови локалізації. В даній версії ПЗ доступна тільки українська локалізація.

GetMessage() – метод, що повертає текстове значення із файла локалізації відповідно до унікального ідентифікатора.

На Рисунку 6.12 показана діаграма класу LocalMessageService.

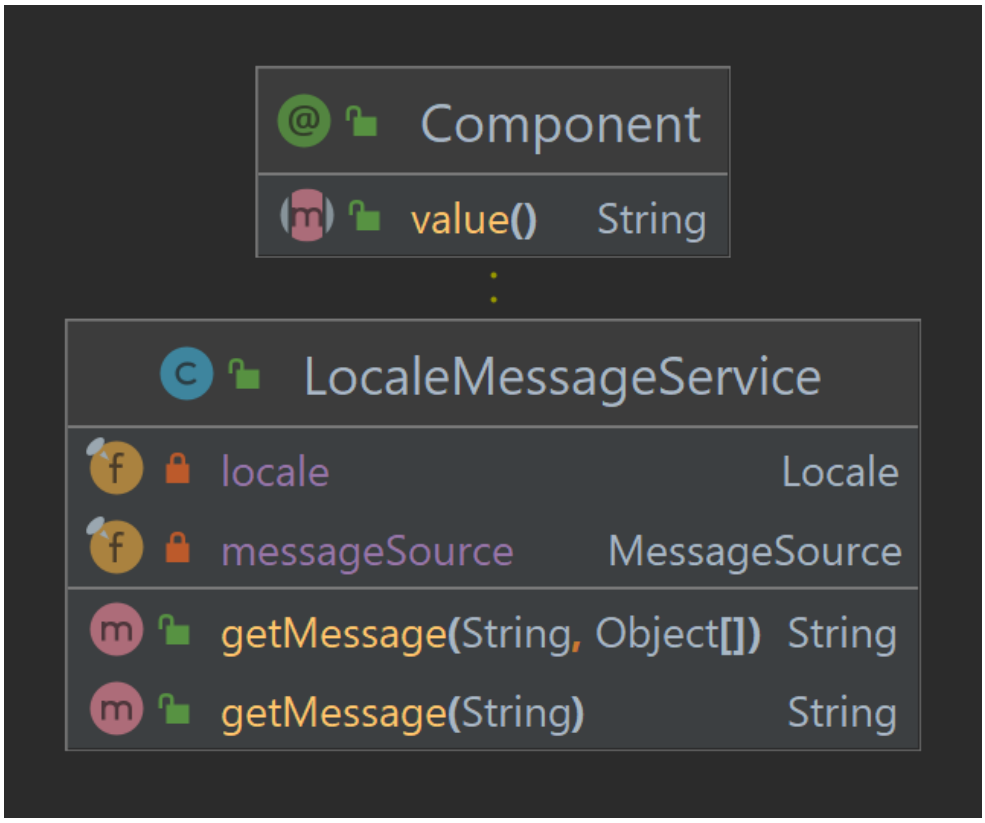

Рисунок 6.12 - Діаграма класу LocaleMessageService та його зв’язків із іншими сутностями

ReplyMessageService – клас-обгортка для класу LocaleMessageService, що використовується для створення повідомлень типу SendMessage.

GetReplyMessage() – метод, що повертає об’єкт класу SendMessage із chat_id, та повідомленням отриманим із файлу локалі.

На Рисунку 6.13 показана діаграма класу ReplyMessageService.

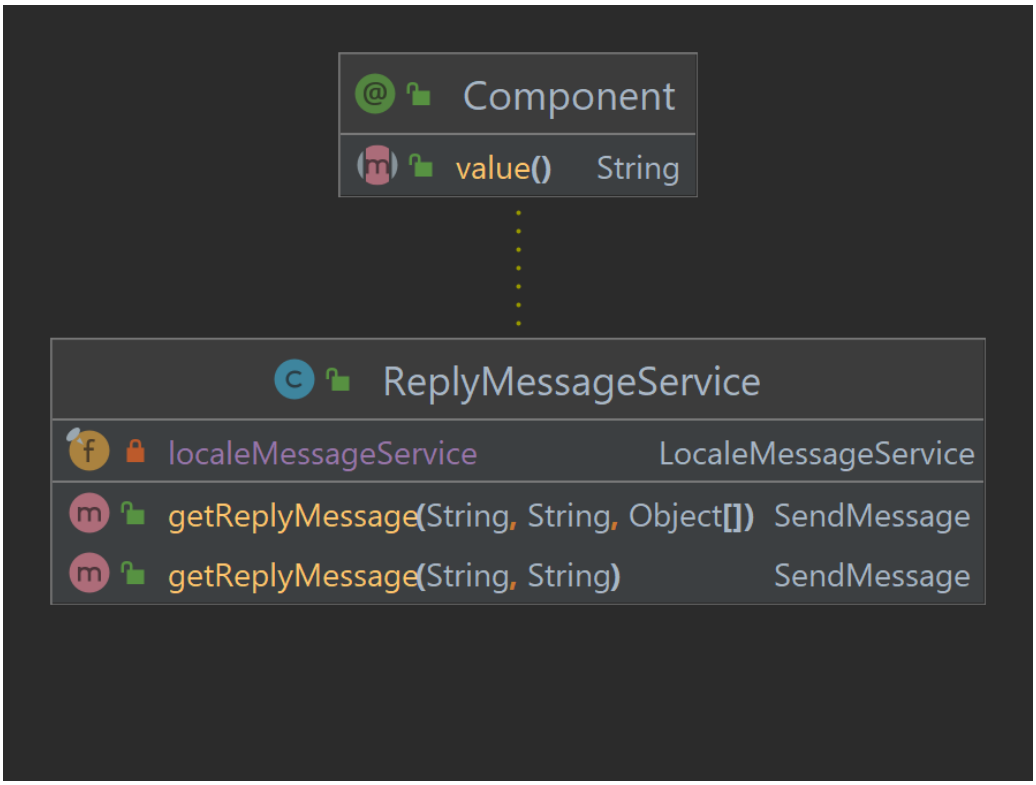

Рисунок 6.13 – Діаграма класу ReplyMessageService та його зв’язків із іншими сутностями

ReplyKeyboardMarkupBuilder – утилітарний клас, що використовується для побудови клавіатури з опціями меню.

Build() – метод, що реалізує процес побудови клавіатури з опціями меню, що відповідають поточному стану боту.

На Рисунку 6.14

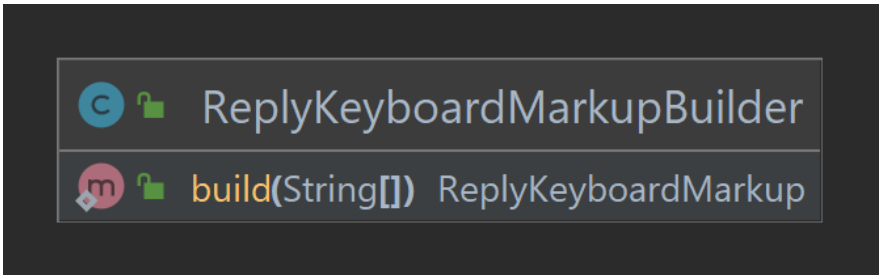

Рисунок 6.14 – Діаграма класу ReplyKeyboardMarkupBuilder

UserDataCache – клас, що забезпечує зберігання, встановленя та отримання поточного стану кожного користувача боту.

SetUserCurrentBotState() – метод, що використовується для зміни поточного стану боту для користувача.

GetUserCurrentBotState() – метод, що повертає значення поточного стану боту для користувача.

На Рисунку 6.15 наведена діаграму класу UserDataCache.

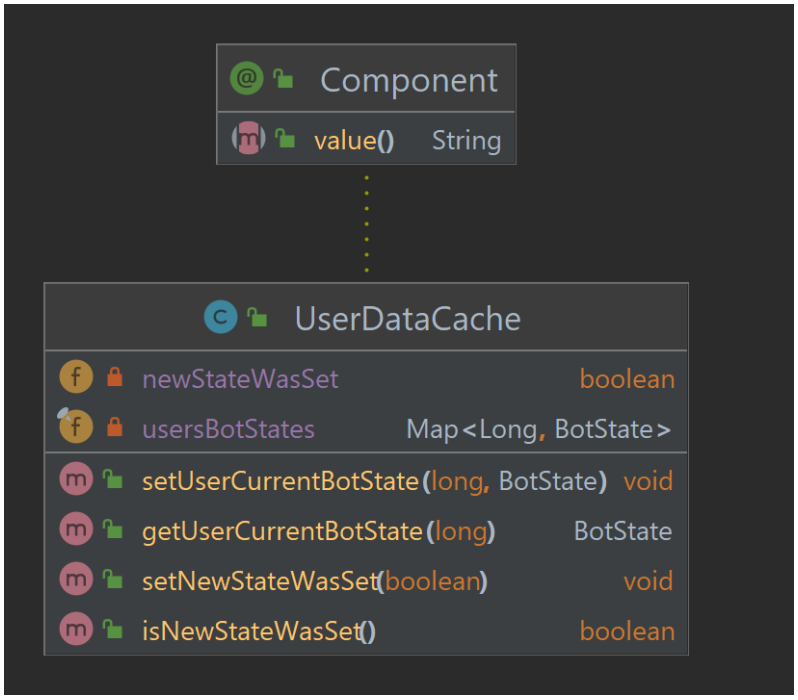

Рисунок 6.15 - Діаграма класу UserDataCache та його зв’язків із іншими сутностями

FilesService – клас, що реалізує усі необхідні функції для роботи з файлами, а саме запису, читання, перевірки за певними критеріями, видалення.

DownloadImage() – метод, що виконує завантаження зображення, що відправив користувач, та сберігає його в спеціальній директорії для подальшого використання.

SaveUserString() – метод, що копіює текст, що буде приховано в зображенні, та зберігає його в текстовому файлі в спеціальній директорії для подальшого використання.

DeleteUserCache() – метод, що видаляє усі файли зображень та текстові файли з повідомленням, що відносяться до певного користувача. Метод викликається

кожен раз, коли алгоритм по приховуванню або вилученню даних завершує свою роботу.

LogFileDeletingStatus() – метод, що записує в журнал подій статус видалення файлів.

IsFileExtensionValid() – метод, що перевіряє чи валідне розширення файлу. Для даної версії ПЗ, валідним розширенням є «.PNG».

HasUserCacheForEncoding(), hasUserCacheForDecoding() – методи, що виконують перевірку наявності необхідних файлів для приховування або вилучення повідомлення відповідно.

На Рисунку 6.16 наведена діаграму класу FilesService.

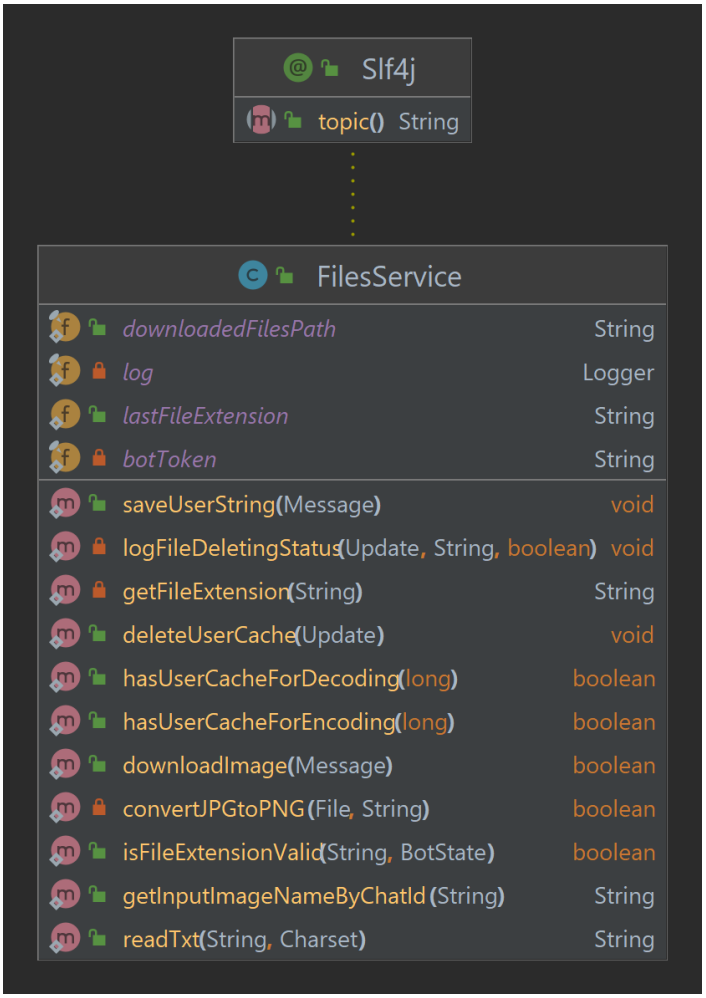

Рисунок 6.16 - Діаграма класу FilesService та його зв’язків із іншими сутностями

Отсанні три класи, що розглянуті далі працюють в коплексі, щоб реалізувати роботу стеганографічного шифратора та дешифратора.

KJBHandler – клас, що виконує роль попереднього обробника запитів на приховування та вилучення повідомлень та передає підготовлені дані класу SpiralKutterJordanBossen.

Encode() – метод, що виконує попередню обробку запиту на приховування повідомлення, виконує виклик методу encode() класу SpiralKutterJordanBossen та забезпечує збереження зображення результату у відповідній директорії.

Decode() – метод, що виконує попередню обробку запиту на вилучення повідомлення, виконує виклик методу decode() та повертає результат.

На Рисунку 6.17 наведена діаграму класу KJBHandler.

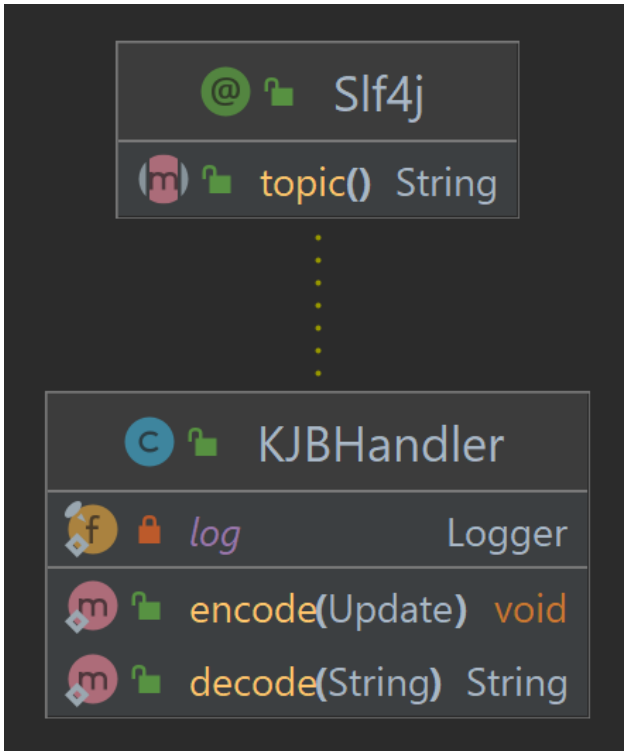

Рисунок 6.17 - Діаграма класу KJBHandler та його зв’язків із іншими сутностями

PixelCoordinates – клас, що використовується для зберігання координат пікселя в двохвимірній площині. Має методи для запису та отримання координат пікселя.

На Рисунку 6.18 наведена діаграму класу PixelCoordinates.

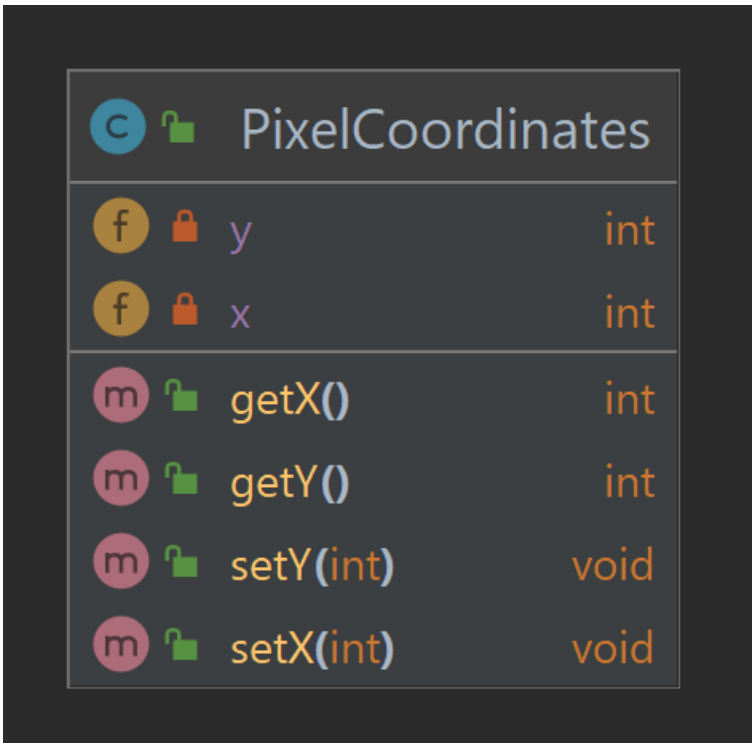

Рисунок 6.18 – Діаграма класу PixelCoordinates

SpiralKutterJordanBossen – клас, в якому реалізовані усі необхідні методи для стеганографічного приховування та вилучення даних із зображення.

Encode() – метод, що запускає процес приховування повідомлення в цифровому зображенні та повертає результат.

Decode() – метод, що запускає процес вилучення повідомлення із цифрового зображення та повертає результат.

На Рисунку 6.19 показана діаграма класу SpiralKutterJordanBossen.

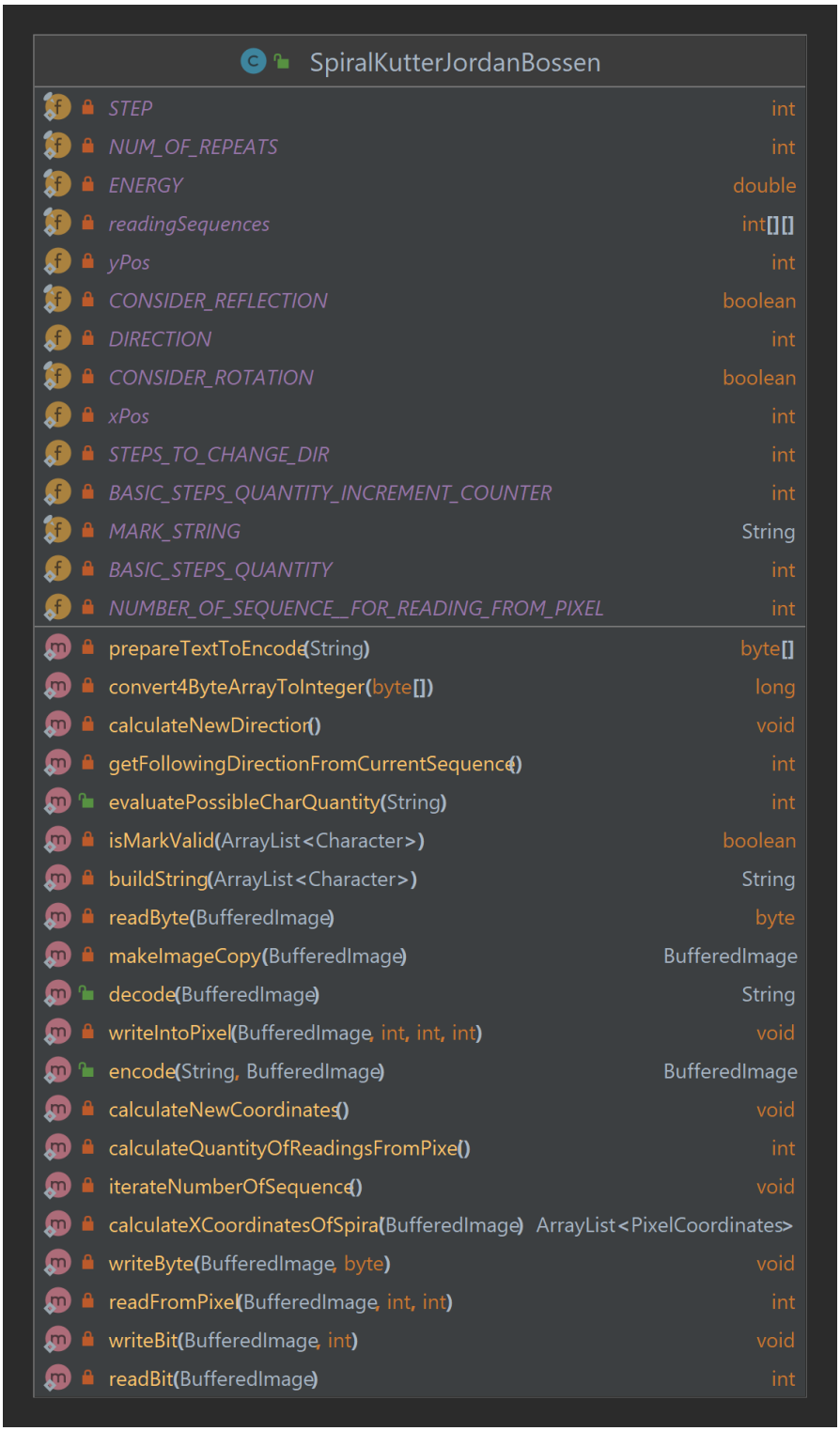

Рисунок 6.19 – Діаграма класу SpiralKutterJordanBossen

Інші методи цього класу виконують різні специфічні задачі, щоб забезпечити коректне вбудовування або вилучення текстового повідомлення із цифрового зображення.

7 СТЕГАНОГРАФІЧНИЙ АЛГОРИТМ

У даній версії ПЗ за основу був використаний стеганографічний алгоритм Куттера-Джордана-Боссена, також відомий як Метод хрестів.

Метод Куттера-Джордана-Боссена - стеганографічний метод, що здійснює вбудовування інформації в зображення. Метод був представлений Мартіном Куттером, Фредеріком Джорданом і Френком Боссеном у журналі Journal of Electronic Imaging у квітні 1998 року [4].

Метод Куттера-Джордана-Боссена належить до класу алгоритмів, що здійснюють приховування даних у просторовій області. В алгоритмах цього класу впровадження даних виконується завдяки зміні яскравості або колірних компонент пікселя. У цьому методі окремі біти даних багаторазово впроваджуються в зображення шляхом зміни значення синього каналу в пікселі. Ця зміна пропорційна до яскравості компоненти пікселя і може набувати як позитивних, так і негативних значень залежно від значення біта даних, що вбудовується [4].

Основними властивостями, якими має володіти цифрове зображення із прихованими даними, - це невиразність для людського ока і стійкість до різних спотворень і змін зображення. Метод Куттера-Джордана-Боссена задовольняє першій вимозі за рахунок вбудовування бітів даних саме в синій канал пікселя, так як людське око найменш чутливе саме до цього кольору [5]. Стійкість до спотворень зображення забезпечується шляхом багаторазового вбудовування бітів даних у різних частинах вихідного зображення.

Задля покращення стійкості до таких видів атак, як обрізання країв зображення та поворот зображення на кількість градусів, кратну 90, та віддзеркалення зображення, був розроблений покращений алгоритм вбудовування інформації в зображення.

Ключовими змінами є вставка спеціальної мітки на початку повідомлення, вставка значення довжини повідомлення після мітки, вибір початкової точки для вбудовування у центрі зображення та сама схема вбудовування по квадратній спіралі.

Спеціальна мітка, що вставляється на початку повідомлення складається із 9 символів. У даній версії алгоритму використовується послідовність «ABCDEFGHI». *У* процесі вилучення даних, спочатку відбувається пошук цієї мітки для визначення координат початку повідомлення та на основі положення цієї мітки визначається, чи було зображення повернуто або віддзеркалено, що дозволяє обрати правильну схему зчитування даних.

Наявність саме 9 символів у мітці зумовлена прагненням мінімізувати вірогідність помилкового детектування мітки в процесі її пошуку в зображенні. Помилкове детектування можливе, через особливість обраного алгоритму зчитування даних. А саме через те, що навіть при відсутності даних, записаних у зображенні, в результаті зчитування алгоритм розраховує і повертає певний набор біт даних, що потім конвертуються у символьні значення відповідно до таблиці Unicode.

Таким чином, можна розрахувати вірогідність помилкового детектування. Кожен символ із обраної послідовності конвертується у послідовність з восьми біт. Наприклад, символ «A» спочатку конвертується у число 65, а потім у бінарну послідовність 01000001 яка і буде вбудована в зображення 15 (для даної версії алгоритму) разів.

Позначимо ймовірність отримання одного символу (0 або 1) як P. Оскільки кожен символ незалежний і може бути 0 або 1, P дорівнює 0.5. Тепер використаємо формулу для ймовірності незалежних подій:

$$\text{P(послідовності)} = \text{P}^{(\text{кіл-ть символів})}$$

, що для даного випадку буде:

$$\text{P(послідовності)} = 0{,}5^{1080} \approx 8.2718 * 10^{-326}$$

З наявних розрахунків можна побачити, що вірогідність помилкового детектування мітки вкрай мала.

Вставка значення довжини повідомлення після мітки дає змогу алгоритму визначити момент зупинки процесу вилучення даних та визначити, чи було повідомлення пошкоджене, і попередити користувача про це.

Вибір початкової точки запису даних у центрі зображення та запис біт даних за схемою квадратної спіралі зумовлені тим, що центральна частина зображення зазвичай містить основне інформаційне навантаження через те, що найважливіші об’єкти зосереджені ближче до центру зображення, а отже, вірогідність навмисної зміни саме центральної частини зображення користувачем найменша, що, в свою чергу, робить цю частину зображення більш пріоритетною для приховування даних у ній.

8 АЛГОРИТМ ПРИХОВУВАННЯ ТЕКСТУ В ЗОБРАЖЕНІ

Передумовами для запуску алгоритму приховування текстової інформації в зображенні є наявність завантаженої відповідної інформації і зображення у відповідну директорію в корні каталогу програмного забезпечення. Цей крок виконується шляхом взаємодії користувача із програмним інтерфейсом телеграм-боту, який в свою чергу і завантажує необхідні файли у відповідну директорію.

Виконання описаних умов призводить до запуску алгоритму приховування даних шляхом виклику статичного методу encode() і передачі тексту та зображення як аргументів.

Крок 1: Підготовка тексту до приховування.

На даному кроці відбувається перетворення текстових даних у масив байтів, де кожен елемент масиву є числом, що означає унікальний номер відповідного символу із таблиці Unicode. Попередньо, на початок повідомлення додається мітка – послідовність із 9 символів, що для даної версії алгоритму є «ABCDEFGHI». Після мітки вставляється довжина самого повідомлення, під що відводиться 4 байти.

Крок 2: Визначення початкових координат.

Так як метод Куттера-Джордана-Боссена використовує просторову область для приховування даних, необхідно визначити координати початкового пікселя в який буде вбудована інформація.

У даній версії алгоритму початковим є центральний піксель зображення, координати якого можна знайти як:

$$(x_{\text{поч.}} = \frac{imageWidth}{2}, y_{\text{поч.}} = \frac{imageHeight}{2})$$

де

$x_{\text{поч}}$ – значення координати x початкового пікселя,

$y_{\text{поч}}$ – значення координати y початкового пікселя,

imageWidth – ширина зображення,
imageHeight – висота зображення.

Крок 3: Перевірка розміру зображення.

Останнім підготовчим кроком перед початком вбудовування інформації в зображення є перевірка даного зображення на наявність достатньої кількості пікселів для приховування інформації. Для цього спочатку виконується розрахунок необхідної кількості пікселів для приховування повідомлення за формулою:

$$n_{\text{необх.}} = msgLength * 8 * numOfRepeats$$

де
msgLength – кількість символів повідомлення із міткою та довжиною власне повідомлення,
numOfRepeats – кількість повторів для вбудовування кожного біта інформації.

Множення на 8 необхідно, тому що кожен символ повідомлення спочатку перетворюється на його байтове представлення і далі у відповідне бітове представлення. Це означає, що для запису одного символу потрібно 8 біт.

Також важливо зазначити, що зберігання символів кирилиці вимагає 2 байт замість 1. Ця особливість враховується при підрахунку за наведеними формулами у створеному ПЗ.

Кількість повторів для вбудовування кожного біта – це значення, що відповідає за те, скільки разів один і той самий біт буде вбудовано в зображення. В даній версії алгоритму це значення дорівнює 15.

Значення, з яким буде порівнюватись необхідна кількість пікселів для приховування даних розраховується за формулою:

$$n_{\text{наявн.}} = \left(\frac{\left(\frac{\min(imgWidth, imgHeight)}{4} \right)^2}{8 * numOfRepeats} \right)$$

де

min() – функція, що повертає найменше з двох значень,

imgWidth – ширина зображення (в пікселях),

imgHeight – висота зображення (в пікселях),

numOfRepeats – кількість повторів для вбудовування кожного біта інформації.

Ділення на 4 виконується, щоб врахувати відстань між пікселями, в які буде приховано інформацію.

Ділення на 8 * numOfRepeats виконується, щоб врахувати кількість пікселів, необхідну для приховування 1 байту інформації.

В результаті виконуємо просте порівняння $n_{\text{необх.}}$ та $n_{\text{наявн.}}$.

Якщо $n_{\text{необх.}}$ більше ніж $n_{\text{наявн.}}$ – цифрове зображення занадто мале, щоб вмістити в себе повідомлення користувача. Телеграм-бот повідомить про це користувача і запропонує ввести інше повідомлення.

Якщо $n_{\text{необх.}}$ менше або дорівнює $n_{\text{наявн.}}$ – виконується перехід до наступного кроку.

Крок 4: Вбудовування даних.

Як було визначено раніше, вбудовування починається із пікселя, що знаходиться у центрі зображення і має координати ($x_{\text{поч}}$,$y_{\text{поч}}$).

Схема вбудовування даних – квадратна спіраль – передбачає систематичну зміну напрямку вибору координат наступного пікселя для вбудовування даних.

Базовою схемою була обрана спіраль з напрямком обертання проти годинникової стрілки з такою послідовністю зміни напрямків: праворуч, вгору, ліворуч, вниз. Задля зручності, цим напрямкам були поставлені у відповідність числові значення від 0 до 3 відповідно (праворуч – 0, вгору – 1, ліворуч – 2, вниз - 3). Візуальне представлення обраної схеми показано на Рисунку 7.1.

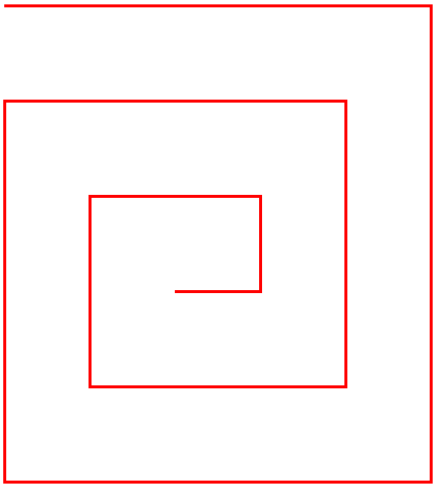

Рисунок 8.1 – Візуальне представлення базової схеми спіралі

Отже послідовність зміни напрямків для базової схеми спіралі має вигляд «0123».

Крок між пікселями дорівнює 4.

Для визначення координат наступного пікселя алгоритм буде враховувати поточний напрямок руху та декілька внутрішніх лічильників, що дозволить вбудувати дані за описаною схемою.

В результаті роботи алгоритму буде створено зображення із прихованим повідомленням, яке далі буде відправлено користувачу.

9 АЛГОРИТМ БЕЗПОСЕРЕДНЬОГО ВБУДОВУВАННЯ ДАНИХ У ПІКСЕЛЬ

Під «безпосереднім вбудовуванням даних» мається на увазі процес зміни компонент кольору певного пікселя зображення певним чином для створення можливості подальшого вилучення із нього біт даних початкового повідомлення.

Безпосереднє вбудовування даних відбувається в декілька кроків.

Спочатку отримується черговий символ у байтовому представлені із масиву символів повідомлення.

Далі для кожного біта отриманого байту спочатку обирається координата пікселя для вбудовування інформації, як описано у попередньому розділі.

Після чого викликається функція запису даних у піксель і в якості параметрів передається саме зображення, координати обраного пікселя, бітове значення і енергія.

Параметр енергії відповідає за те, наскільки сильно буде змінена кольорова компонента обраного пікселя. У даній версії алгоритму ця змінна має значення 0.25. Таке значення встановлено шляхом експериментального підбору і забезпечує достатню стійкість даних при мінімальній помітності змін.

У тілі функції відбувається розрахунок яскравості обраного пікселя за загальноприйнятою формулою [6]:

$$pixelBrightness = 0.29890 * red + 0.58662 * green + 0.11448 * blue$$

де red, green та blue – значення інтенсивності відповідних кольорів моделі RGB – червоного, зеленого та синього.

Для приховування даних буде використовуватись синя кольорова компонента. Такий вибір зумовлений низькою чутливістю людського ока до зміни саме синьої компоненти кольору.

Значення, яким буде замінено початкове значення синьої компоненти кольору обраного пікселя розраховується за формулою:

$$modifiedBlueComponent = blue + energy * pixelBrightness$$

, якщо значення чергового біту для приховування дорівнює 1.

Якщо значення чергового біту для приховування дорівнює 0, формула приймає вигляд:

$$modifiedBlueComponent = blue - energy * pixelBrightness$$

де

blue - початкове значення синьої компоненти кольору обраного пікселя,

energy – константне значення енергії, що передане аргументом функції,

pixelBrightness – значення яскравості обраного пікселя.

Отримане нове значення синьої компоненти кольору обраного пікселя перевіряється на входження до проміжку $[0,255]$ і, якщо воно виходить за межі, обрізається до найближчої границі вказаного проміжку:

- до 0, якщо modifiedBlueComponent менше за 0
- до 255, якщо modifiedBlueComponent більше за 255

Фінальним кроком є присвоєння значення modifiedBlueComponent синій компоненті обраного пікселя, що реалізовано за допомогою відповідної функції.

10 АЛГОРИТМ ВИЛУЧЕННЯ ТЕКСТУ ІЗ ЗОБРАЖЕННЯ

Передумовами для запуску алгоритму вилучення текстової інформації із зображення є наявність завантаженого відповідного зображення у відповідну директорію в корні каталогу програмного забезпечення. Цей крок виконується шляхом взаємодії користувача із програмним інтерфейсом телеграм-боту, який в свою чергу і завантажує необхідний файл у відповідну директорію.

Виконання описаних умов призводить до запуску алгоритму вилучення даних шляхом виклику статичного методу decode() і передачі зображення як аргументу.

Крок 1: Визначення початкових координат і послідовності вилучення даних.

Пошук мітки, що була додана на початок повідомлення перед його вбудовуванням в зображення, є основною задачею на цьому етапі роботи алгоритму.

Пошук мітки реалізовано шляхом послідовного перебору усіх пікселів зображення за схемою базової спіралі із кроком 1, починаючи із центрального пікселя за таким алгоритмом:

1) Отримати координати наступного пікселя.
2) Отримати схему зміни напрямків із масиву.
3) Виконати послідовне зчитування 9 символів за отриманою схемою.
4) Порівняти отриману послідовність символів із символами мітки («ABCDEFGHI»).
5) Якщо отримана послідовність співпадає із послідовністю символів мітки, продовжити вилучення повідомлення з поточних координат за поточною схемою зміни напрямків.
6) Інакше, перейти до кроку 2 цього алгоритму.
7) Після повного перебору масиву схем зміни напрямків, перейти до кроку 1 цього алгоритму .
8) У випадку досягнення краю зображення - зупинити алгоритм та повідомити користувача про відсутність повідомлення у даному зображені або можливу утрату повідомлення через пошкодження структури зображення при передачі.

На Рисунку 9.1 наведена блок-схема описаного алгоритму.

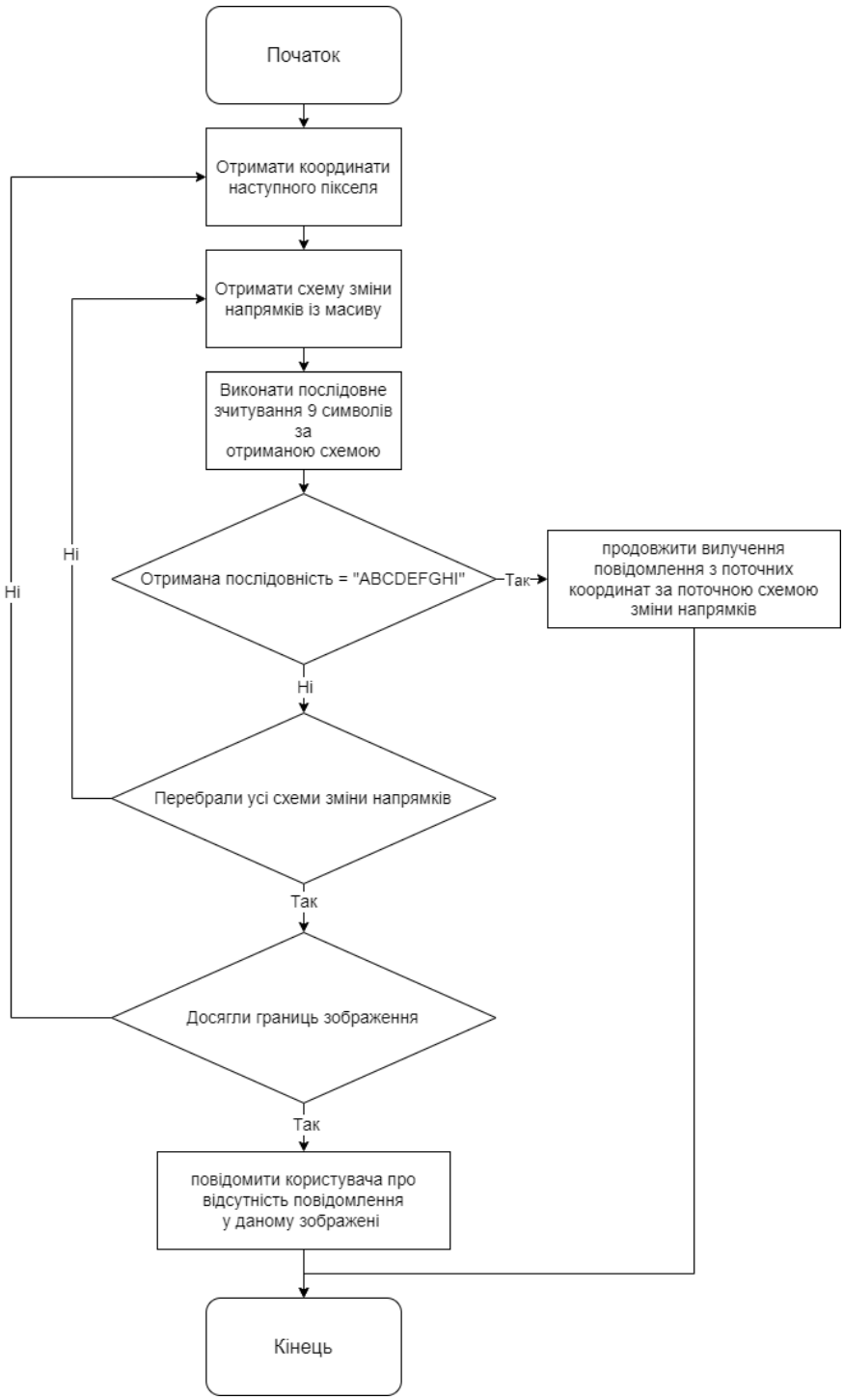

Рисунок 10.1 – Блок-схема алгоритму пошуку мітки в зображенні

Масив схем зміни напрямків містить схеми зміни напрямків, що були визначені завчасно для боротьби з такими атаками, як поворот зображення на 90 градусів будь-яку кількість разів та віддзеркалення зображення як по горизонталі,

так і по вертикалі, або комбінації усіх описаних атак. Таблиця 9.1 надає детальний опис атак та схем зміни напрямків, що боряться із наслідками цих атак.

Таблиця 10.1 – Опис атак на зображення та схем зміни напрямків, що боряться з описаними атаками:

№	Опис атаки		Схема зміни напрямків	Напрямок обертання спіралі	Приклад спіралі		
	Віддзеркалення	Кут повороту зображення					
0	Без віддзеркалення	0	0123	Проти годинникової стрілки	5	4	3
					6	1	2
					7	8	9
1	Віддзеркалено по горизонталі	0	2103	За годинниковою стрілкою	3	4	5
					2	1	6
					9	8	7
2	Віддзеркалено по горизонталі і вертикалі	0	2301	Проти годинникової стрілки	9	8	7
					2	1	6
					3	4	5
3	Віддзеркалено по вертикалі	0	0321	За годинниковою стрілкою	7	8	9
					6	1	2
					5	4	3
4	Без віддзеркалення	90 градусів за годинниковою стрілкою	3012	Проти годинникової стрілки	7	6	5
					8	1	4
					9	2	3
5	Віддзеркалено по горизонталі	90 градусів за годинниковою стрілкою	3210	За годинниковою стрілкою	5	6	7
					4	1	8
					3	2	9
6	Віддзеркалено по горизонталі і вертикалі	90 градусів за годинниковою стрілкою	1230	Проти годинникової стрілки	3	2	9
					4	1	8
					5	6	7
7	Віддзеркалено по вертикалі	90 градусів за годинниковою стрілкою	1032	За годинниковою стрілкою	9	2	3
					8	1	4
					7	6	5

У наведеній таблиці відсутні описи атак із поворотом зображення більше ніж на 90 градусів (180, 270, 360 і т.д) через те, що експериментальним шляхом було з’ясовано, що 8 описаних схем зміни напрямків також успішно боряться із наслідками повороту зображення на будь-яку іншу кількість градусів, кратну 90.

Крок 2: Визначення довжини прихованого повідомлення.

У випадку успішного детектування мітки, відбувається зчитування наступних чотирьох байтів повідомлення задля визначення довжини саме прихованого повідомлення користувача.

Крок 3: Вилучення даних.

Вилучення даних відбувається за допомогою функцій readByte(), readBit() та readFromPixel().

У тілі функції readByte() формується байт інформації з розрахованих бітів даних. Формування байт відбувається ітеративно:

- виконати побітовий зсув вліво на один розряд;
- встановити в молодший розряд новий розрахований біт.

Розрахунок біт даних реалізовано у функції readBit(). У тілі функції відбувається послідовний ітеративний перебір пікселів за схемою зміни напрямків, що була визначена на першому кроці, з інтервалом у 4 пікселі.

Для вилучення одного біта відбувається 15 ітерацій під час яких буде викликатись функція readFromPixel() і буде отримана проміжна сума. Далі розраховується середнє арифметичне шляхом ділення проміжної суми на кількість доданків.

Отримане значення являє собою розрахункове значення, яке потрібно оцінити.

Оцінка відбувається за наступними правилами.

- Якщо розрахункове значення більше за 0.5, це означає, що більше половини отриманих біт були одиницями, а отже вважаємо, що було приховано одиницю.
- Якщо розрахункове значення менше або дорівнює 0.5, це означає, що більше половини отриманих біт були нулями, а отже вважаємо, що було приховано нуль.

11 АЛГОРИТМ БЕСПОСЕРЕДНЬО ВИЛУЧЕННЯ ДАНИХ ІЗ ПІКСЕЛЯ

Процес вилучення даних із пікселя реалізований у функції readFromPixel(), що приймає зображення та координати обраного пікселя як аргументи.

Спочатку розраховується середнє значення синіх компонент кольору пікселів, що розташовані «хрестом» відносно обраного пікселя (Рисунок 11.1).

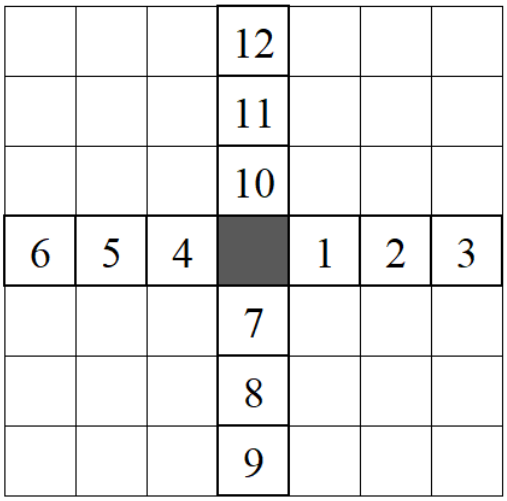

Рисунок 11.1 – Пікселі, що розташовані «хрестом» відносно обраного пікселя

На даному малюнку ці пікселі підписані номерами від 1 до 12.

Формула, що використовується в розрахунку:

$$estimate = \frac{\sum_{i=1}^{3} blue_{x+i,y} + \sum_{i=1}^{3} blue_{x-i,y} + \sum_{i=1}^{3} blue_{x,y+i} + \sum_{i=1}^{3} blue_{x,y-i}}{12}$$

де $blue_{x,y}$ - значення синьої компоненти кольору пікселя з координатами X та Y.

Після чого виконується порівняння значення синьої компоненти обраного пікселя - $blue$ із розрахованим значенням $estimate$.

Якщо значення $blue$ більше ніж значення $estimate$, прихований біт дорівнює одиниці.

Інакше - прихований біт дорівнює нулю.

12 ТЕСТУВАННЯ СТІЙКОСТІ СТВОРЕНОГО АЛГОРИТМУ ДО НАЙПОШИРЕНІШИХ СТЕГАНОГРАФІЧНИХ АТАК

Потенційний зловмисник, що буде спостерігати за обміном інформацією між користувачами, що користуються створеним програмним застосунком, може використовувати різноманітні техніки та програмні застосунки задля проведення стегоаналітичних атак з метою детектування самого факту наявності повідомлення в зображенні, знищення інформації, що була прихована, та її вилучення.

Прикладами описаних видів атак є [7, 8, 9]:

- Суб'єктивна – зловмисник намагається визначити «на око», чи є в зображенні приховане повідомлення
- Атака на основі обраного порожнього контейнера – зловмисник має змогу змусити користувача обрати певне зображення. Після чого зловмисник порівнює початкове зображення, що не містить прихованих даних, та зображення із прихованими даними, що дає змогу виявити стеганографічний канал передачі даних
- Геометричні перетворення – зловмисник намагається знищити повідомлення, що міститься в зображення, шляхом зменшення зображення за рахунок відрізання граничних областей, повороту чи віддзеркалення зображення.
- Вилучення повідомлення на основі відомих стеганографічних алгоритмів – зловмисник намагається вилучити та прочитати приховане повідомлення шляхом використання відомих стеганографічних алгоритмів.

Тестування створеного алгоритму на стійкість до описаних атак проводилось на прикладі обраного цифрового зображення (Рисунок 12.1), в яке було приховано повідомлення «Ще не вмерла України, ні слава, ні воля!». В результаті було отримано зображення, що містить дане повідомлення (Рисунок 12.2).

Рисунок 12.1 – Обране цифрове зображення

Рисунок 12.2 – Цифрове зображення із прихованим повідомленням «Ще не вмерла України, ні слава, ні воля!»

Процес приховування тексту в обраному зображенні за допомогою створеного ПЗ показано на Рисунку 12.3.

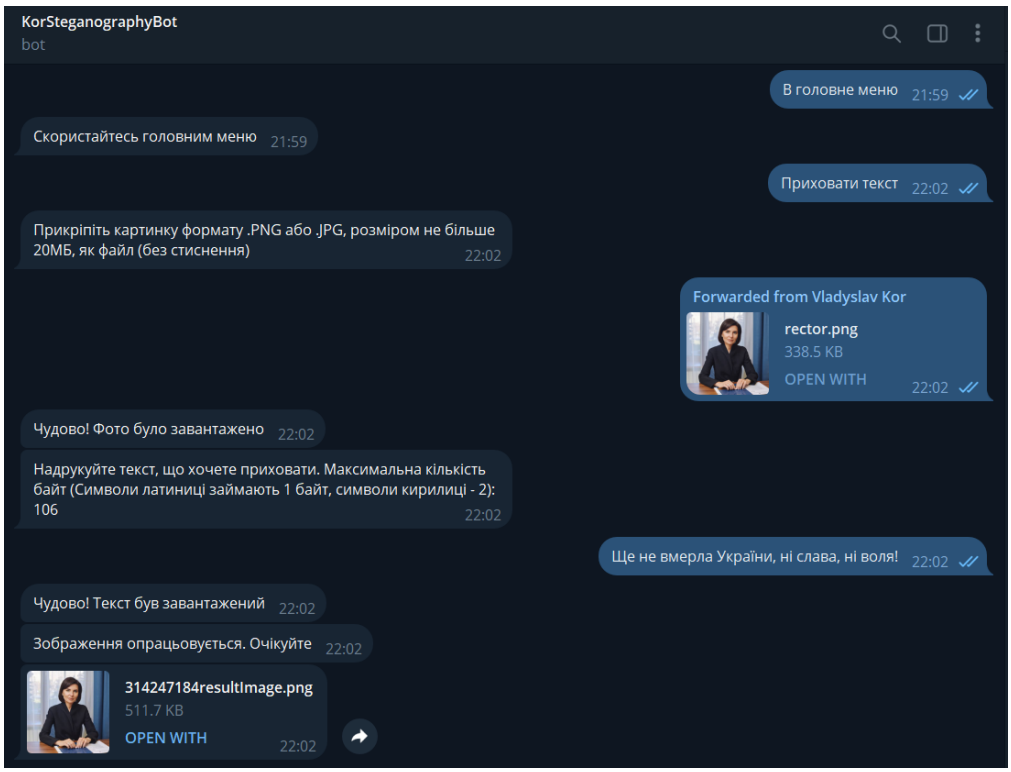

Рисунок 12.3 – Процес приховування тексту в зображенні за допомогою створеного ПЗ

Після отримання зображення із прихованим текстом, є доцільним провести вилучення задля перевірки коректності роботи алгоритму (Рисунок 12.4).

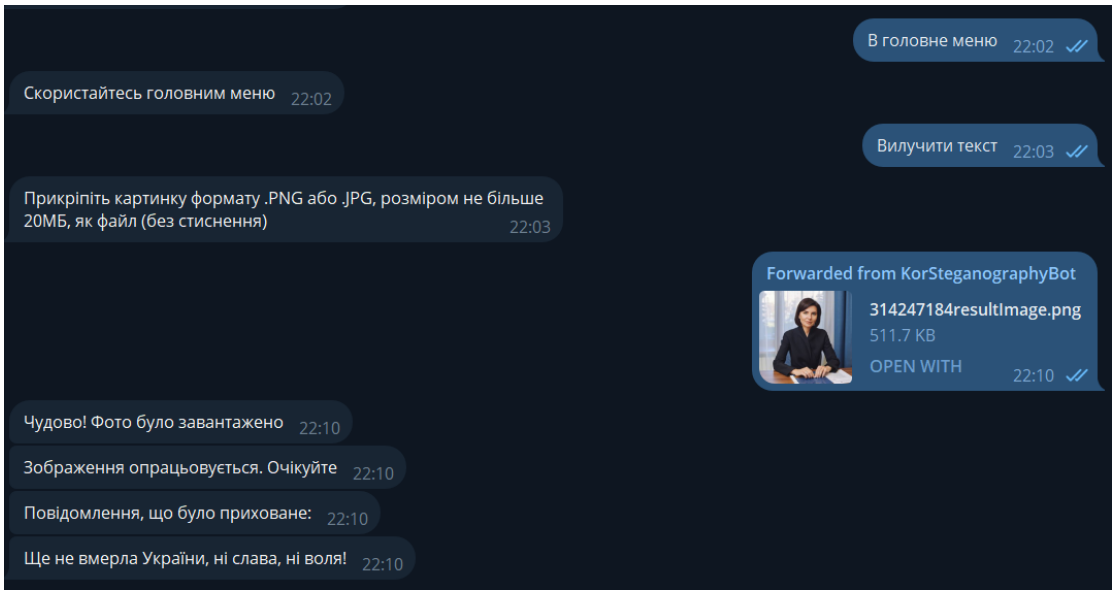

Рисунок 12.4 – Результат вилучення тексту із зображення

Суб’єктивна атака.

Для реалізації цієї атаки необхідно виконати візуальний огляд зображення з прихованим повідомленням на наявність візуальних артефактів, таких як пікселі іншого кольору або відтінку.

Наприклад, на Рисунок 12.5 наведено приклад зображення в яке було вбудовано повідомлення зі значенням змінної енергії рівним 1, що у 4 рази більше ніж підібране стандартне значення, що призводить до утворення видимої сітки пікселів.

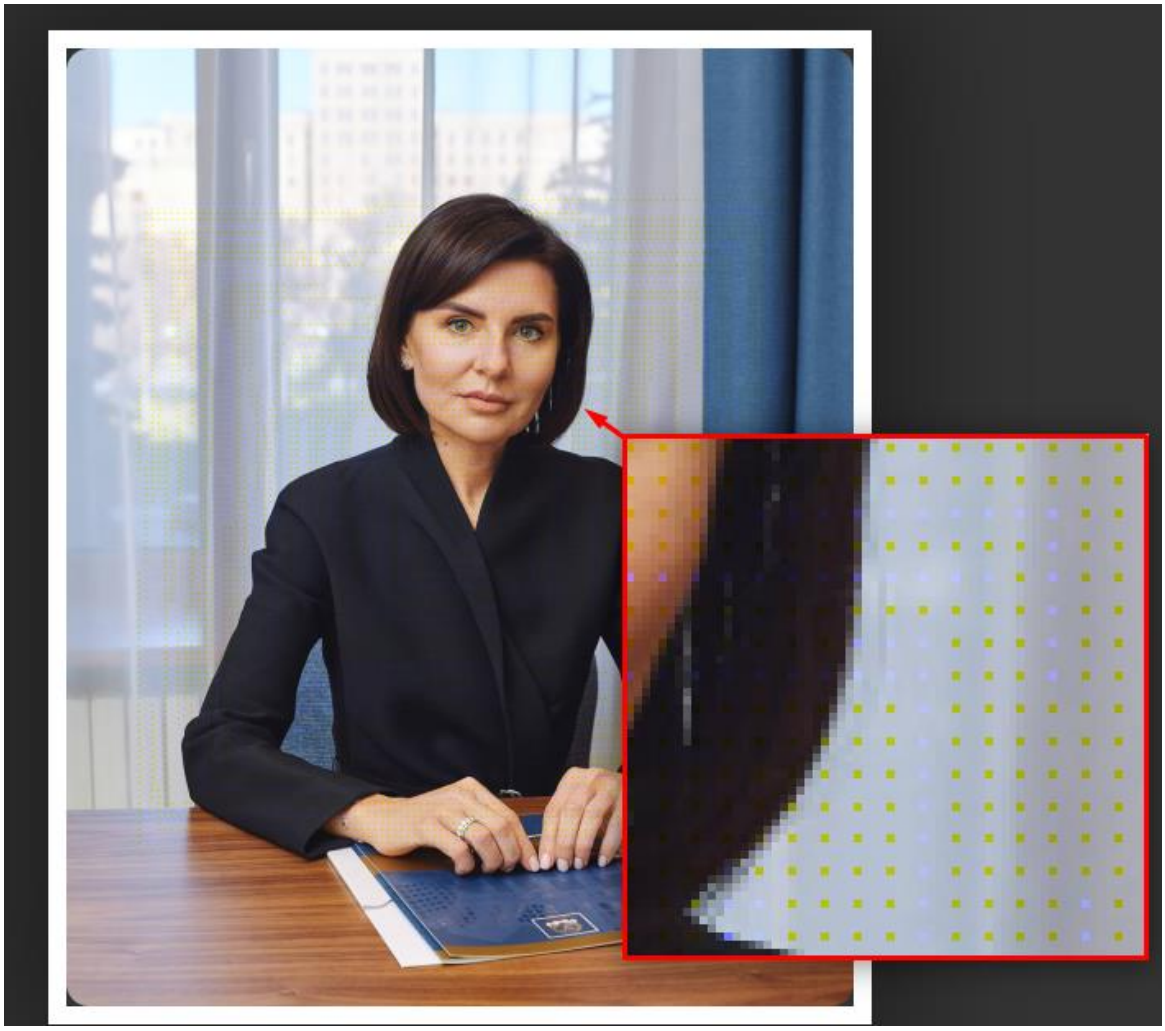

Рисунок 12.5 – Приклад зображення, що утворено в результаті роботи алгоритму із неправильно підібраними параметрами змінних, що призводить до утворення видимих артефактів

На відміну від зображення, наведеного на Рисунок 12.5, зображення на Рисунок 12.6, що є результатом роботи створеного ПЗ, не має (або майже не має – в залежності від кольорів зображення) видимих артефактів, що викривають наявність прихованого повідомлення.

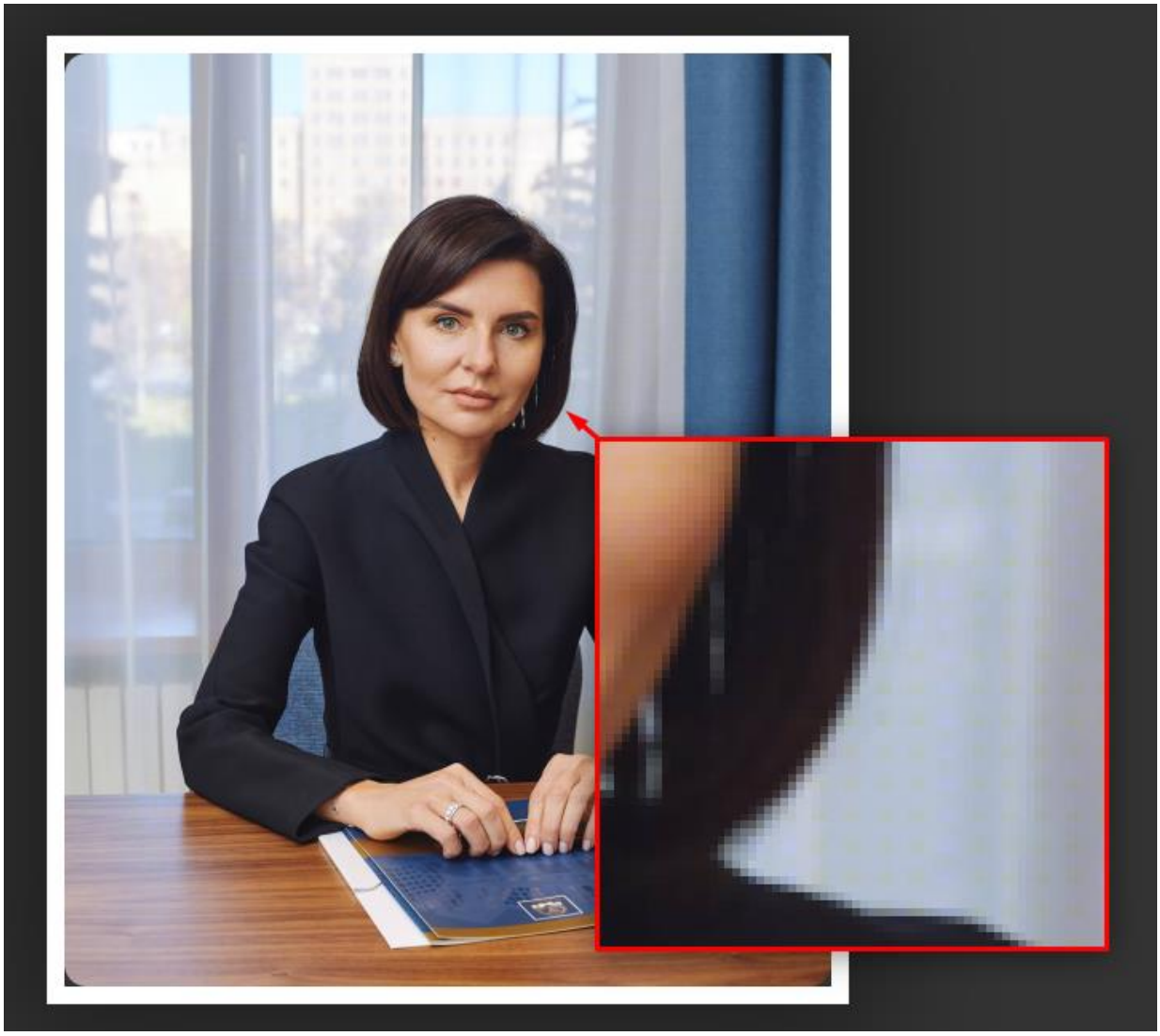

Рисунок 12.6 - Приклад зображення, що утворено в результаті роботи алгоритму із правильно підібраними параметрами змінних.

Як можна помітити при порівняння наведених зображень, візуальний огляд зображення не дає можливості виявляти наявність прихованого повідомлення у зображенні.

Атака на основі обраного порожнього контейнера.

Зімітуємо ситуацію, коли зловмисник отримав обидва зображення: початкове зображення і зображення із прихованим повідомленням.

Далі виконується порівняння цих зображень «на око» (Рисунок 12.7) та за допомогою спеціалізованого ПЗ (Рисунок 12.8).

Рисунок 12.7 – Порівняння початкового зображення (№1) та зображення із прихованим повідомленням (№2) «на око»

Рисунок 12.8 - Порівняння початкового зображення (№1) та зображення із прихованим повідомленням (№2) за допомогою спеціалізованого ПЗ

Зображення, що показано на Рисунок 12.8, отримано в результаті використання спеціалізованого ПЗ для криміналістики цифрових зображень. Режим роботи – «Аналіз рівня помилок»(Error Level Analysis) [10]. У даному режимі роботи ПЗ висвітлює області зображення, що мають різні ступені стиснення, що може вказувати на редагування зображення.

На Рисунок 12.8 на другому зображення явно видно сітку з пікселів висвітлених синім кольором. Таким чином зловмисник може явно побачити, що зображення зазнало змін, а отже може містити приховане повідомлення. В цьому випадку можна констатувати, що зловмисник має можливість детектувати сам факт наявність прихованого повідомлення.

Геометричні перетворення.

У випадку детектування самого факту наявності повідомлення, зловмисник може провести геометричні атаки на зображення з метою зміни внутрішньої структури розташування пікселів, що потенційно може призвести до втрати прихованого повідомлення. Далі наведено приклад зображень з прихованим текстом після проведення атак та результат спроби вилучення даних за допомогою ПЗ. Атаки, що були реалізовані: поворот зображення на кількість градусів, кратну 90 (Рисунок 12.9, Рисунок 12.10), відсікання граничних областей (Рисунок 12.11, Рисунок 12.12), віддзеркалення як по вертикалі, так і по горизонталі (Рисунок 12.13, Рисунок 12.14), комбінація усіх описаних атак (Рисунок 12.15, Рисунок 12.16).

Рисунок 12.9 – Зображення із прихованим повідомленням, що було повернуто на 90 градусів

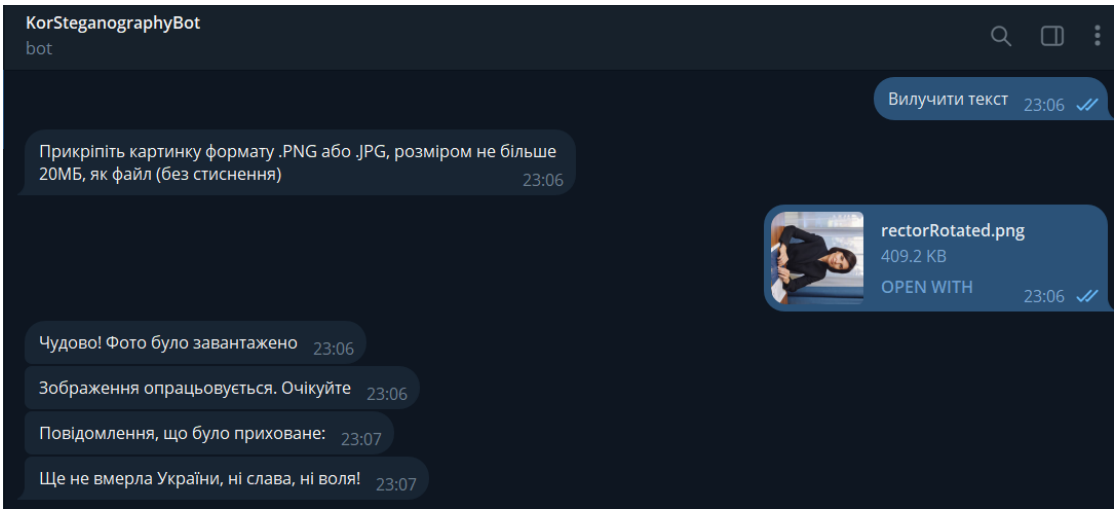

Рисунок 12.10 – Результат спроби вилучення даних із зображення, що було повернуто на 90 градусів

Рисунок 12.11 - Зображення із прихованим повідомленням, що було обрізано знизу на 10 пікселів та справа на 14 пікселів

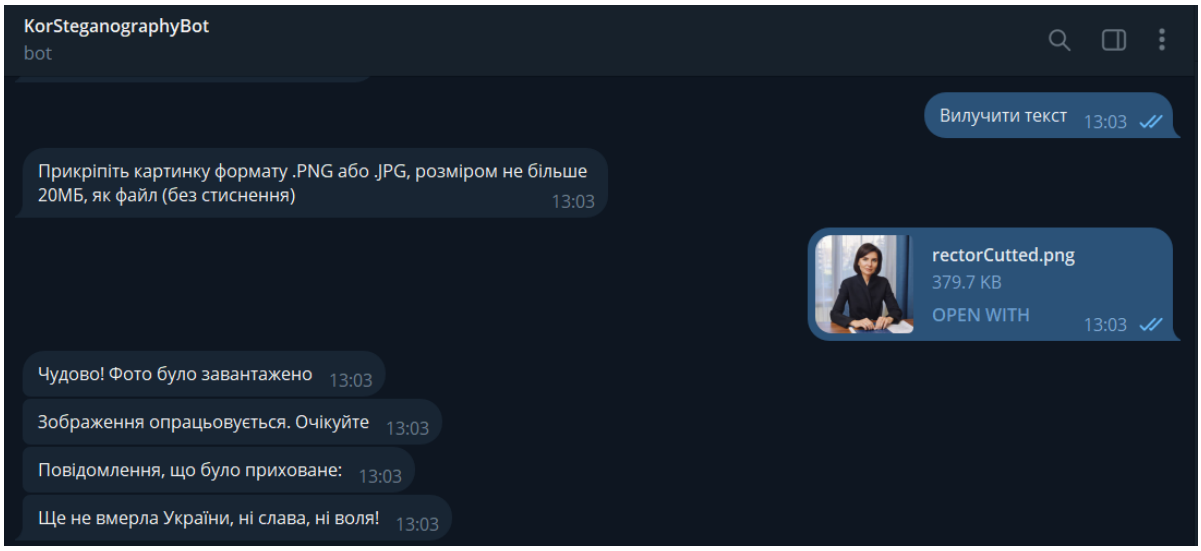

Рисунок 12.12 - Результат спроби вилучення даних із зображення, що було обрізано знизу на 10 пікселів та справа на 14 пікселів

Рисунок 12.13 - Зображення із прихованим повідомленням, що було віддзеркалено по горизонталі і вертикалі

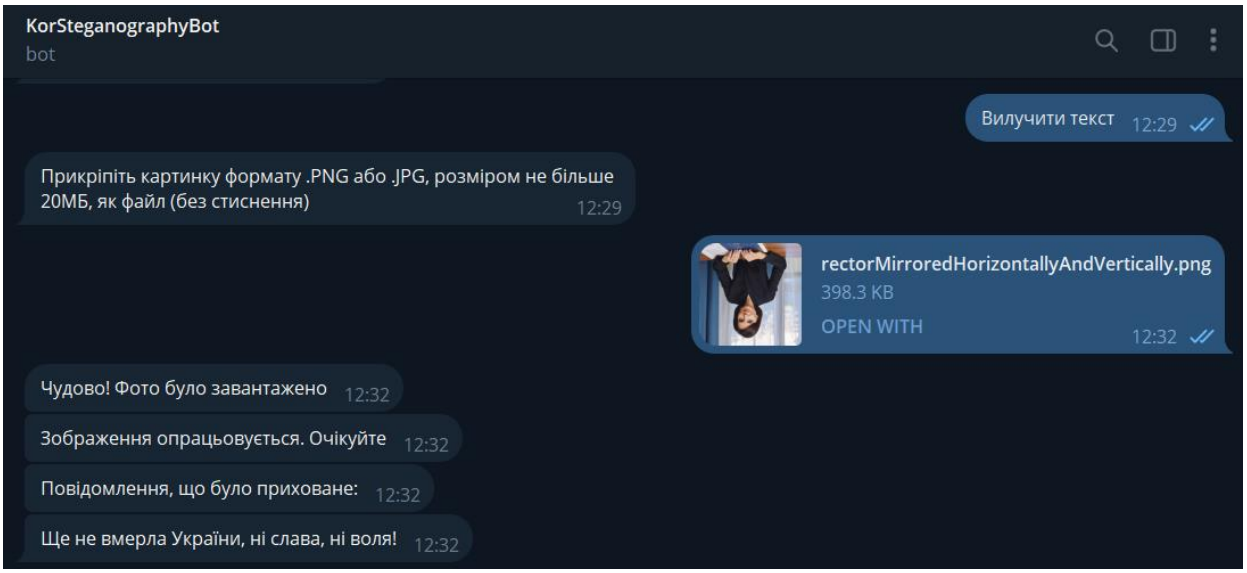

Рисунок 12.14 - Результат спроби вилучення даних із зображення, що було віддзеркалено по горизонталі і вертикалі

Рисунок 12.15 - Зображення із прихованим повідомленням, що було віддзеркалено по горизонталі і вертикалі, повернуто на 90 градусів і обрізано знизу на 10 пікселів та справа на 14 пікселів

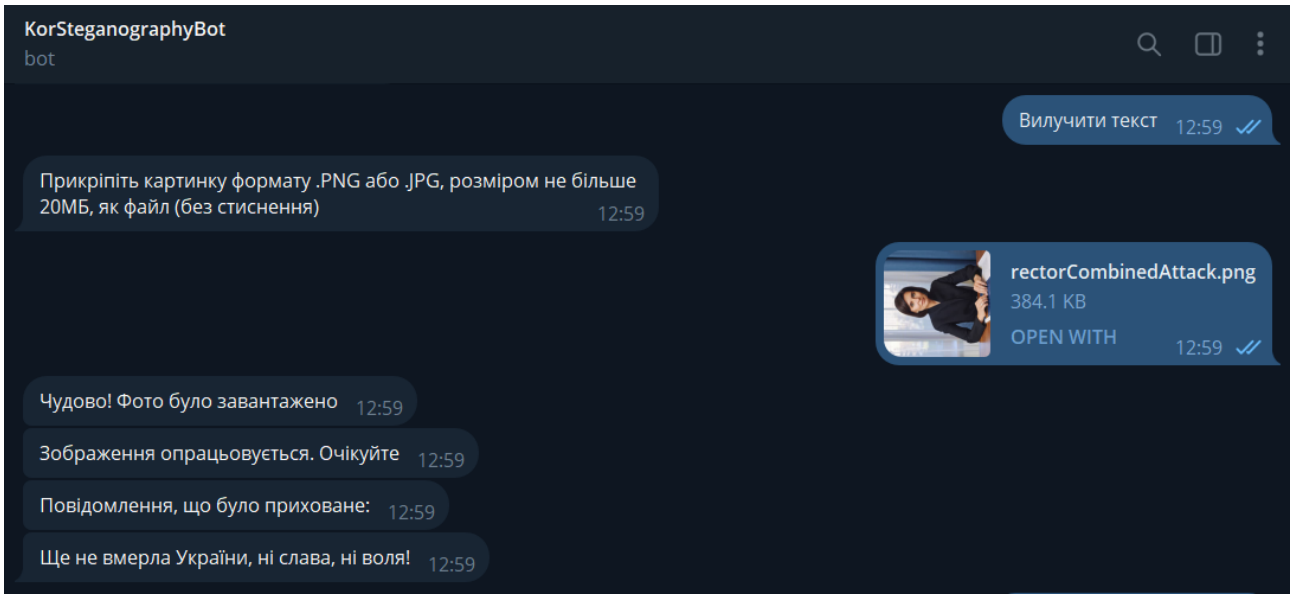

Рисунок 12.16 - Результат спроби вилучення даних із зображення, що було віддзеркалено по горизонталі і вертикалі, повернуто на 90 градусів і обрізано знизу на 10 пікселів та справа на 14 пікселів

З наведених результатів тестів можна побачити, що алгоритм, створений для даного ПЗ здатен ефективно боротися із наслідками описаних геометричних атак, як поодиноких, так і їх комбінацій.

Вилучення повідомлення на основі відомих стеганографічних алгоритмів.

У разі детектування самого факту наявності прихованого повідомлення, зловмисник може спробувати вилучити це повідомлення використовуючи відомі стеганографічні алгоритми[11, 12]. У випадку із створеним ПЗ, така можливість наближається до нуля через те, що створена схема порядку запису даних не була реалізована раніше ніким, а отже не є загальновідомою.

13 АНАЛІЗ ПЕРЕВАГ ТА НЕДОЛІКІВ СТВОРЕНОГО ПЗ

У результаті роботи над науковим проектом, дослідження джерел та проведення практичних тестів, були виявлені переваги та недоліки створеного програмного продукту.

До переваг можна віднести:

1) Обрана платформа миттєвого обміну повідомленнями – Телеграм – дозволяє знизити необхідний мінімальний рівень технічних навичок та знань для користування створеним ПЗ.
2) Покращена версія алгоритму Куттера-Джордана-Боссена дозволяє боротися з такими видами геометричних атак, як поворот, віддзеркалення та відсікання країв зображення.
3) Алгоритми взаємодії користувача із ПЗ, задля приховування або вилучення даних, потребують виконання мінімально можливої необхідної кількості кроків для отримання результату та забезпечені детальними інструкціями та підказками на кожному кроці.

До несуттєвих недоліків можна віднести:

1) Великий шанс втрати інформації при конвертації зображення із стисненням чи до геометричної атаки стиснення зображення.
2) Неможливість проаналізувати серверну частину коду месенджеру Телеграм, через її відсутність у відкритому доступі, а отже існує потенційний ризик витоку інформації.
3) Обмежена пропускна здатність, через особливості роботи алгоритму, а саме через використання областей 7 на 7 пікселів для приховування одного біту даних та через багатократне приховування кожного біту даних з метою збільшення стійкості до спотворення зображення.

Описані недоліки зумовлені тим, що обраний стеганографічний алгоритм використовує просторову область зображення. В свою чергу, описані переваги показують, що мета роботи досягнута і ПЗ готове до використання.

14 РЕЗУЛЬТАТИ

Результати роботи над проектом під назвою "Телеграм-бот, що використовує стеганографічні алгоритми для приховування інформації в цифрових зображеннях" дозволили створити функціональний телеграм-бот. Його основна мета досягнута і полягає в можливості приховування текстової інформації в зображеннях з подальшим їх використанням для безпечної комунікації в месенджері Телеграм.

Користувачеві надається зручний інтерфейс, який дозволяє приховати повідомлення всього за кілька простих кроків. Після запуску бота, користувач може обрати опцію "Приховати текст" з головного меню, відправити обране зображення, що стане контейнером для повідомлення, та відправити саме повідомлення. Результатом є зображення з прихованим текстом, яке можна негайно переслати або відправити в потрібний чат у месенджері.

Процес вилучення прихованого повідомлення також відбувається у тому ж боті. Для цього користувач повинен запустити бота, обрати опцію "Вилучити текст" з головного меню і надіслати зображення, що містить приховану інформацію. Результатом буде отримання прихованого повідомлення.

15 ІНСТРУКЦІЯ КОРИСТУВАЧА

Як було визначено у вимогах до програмного застосунку на етапі розробки, простота користування є однією із ключових вимог. Ця мета було досягнута шляхом розробки зручного візуального інтерфейсу для взаємодії користувача та ПЗ, що містить мінімальну необхідну кількість кроків для отримання результату.

Алгоритм приховання повідомлення:

1) Обрати пункт головного меню «Приховати текст»;
2) Прикріпити зображення, як файл (Без стиснення);
3) Відправити повідомлення, що має бути приховано;
4) Отримати зображення із прихованим текстом;
5) Перевірити результат, вилучивши повідомлення;

Алгоритм вилучення повідомлення:

1) Обрати пункт головного меню «Вилучити текст»;
2) Прикріпити зображення, як файл (Без стиснення);

Отримати приховане повідомлення;

16 ПРИКЛАД ВИКОРИСТАННЯ

Користувач Боб вирішив передати Алісі таємне повідомлення. Боб використає KorSteganographyBot.

Крок 1:

Боб та Аліса запустили месенджер Телеграм та ввели в поле пошуку назву бота – KorStenographyBot (Рисунок 16.1).

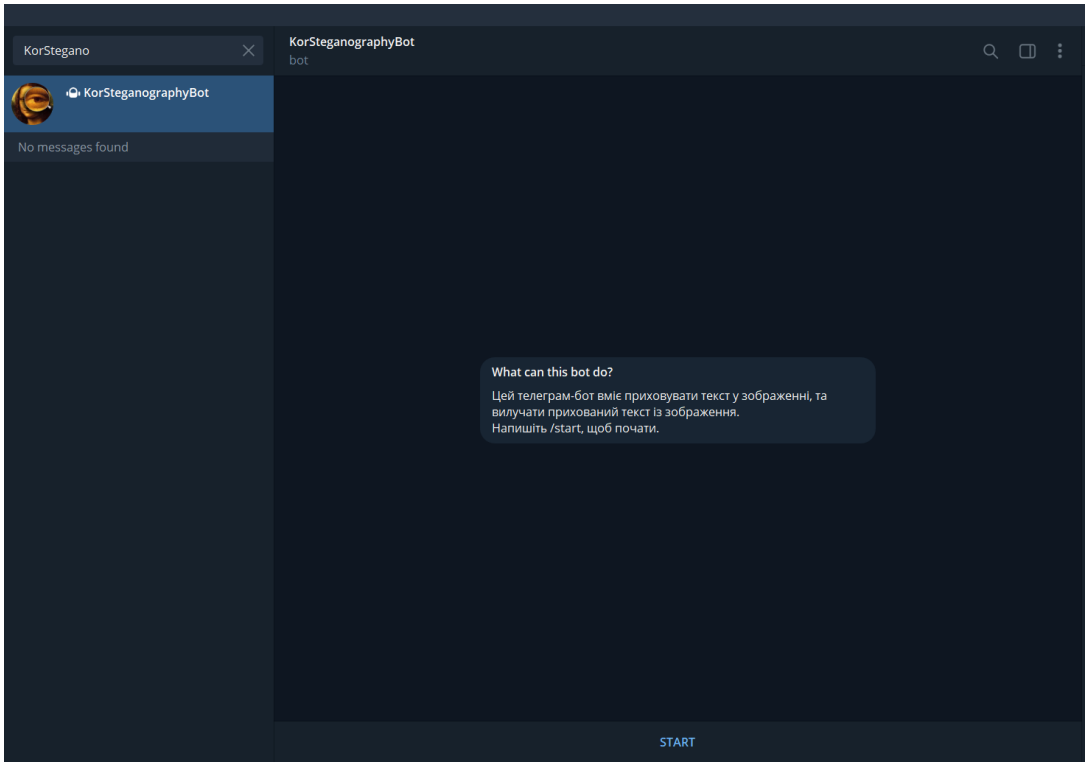

Рисунок 16.1 – Пошук бота за ім’ям.

Після того, як Боб та Аліса знайшли бота та перейшли із ним в чат, вони натиснули на кнопку або відправили команду «/start» для початку роботи і побачили перед собою головне меню (Рисунок 16.2).

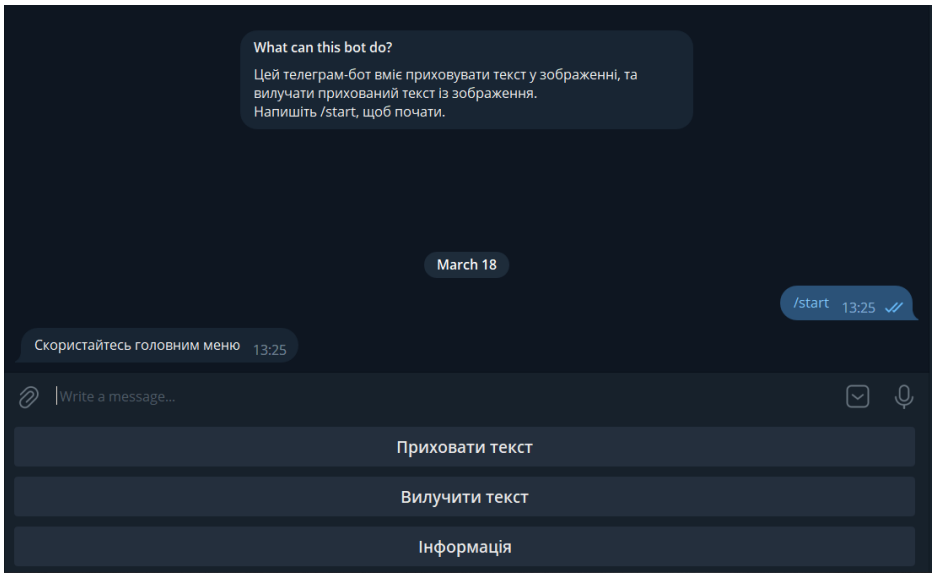

Рисунок 16.2. – Головне меню

Боб користується ботом перший раз, тому він обирає пункт меню «Інформація», щоб ознайомитись із інформацією про бота (Рисунок 16.3).

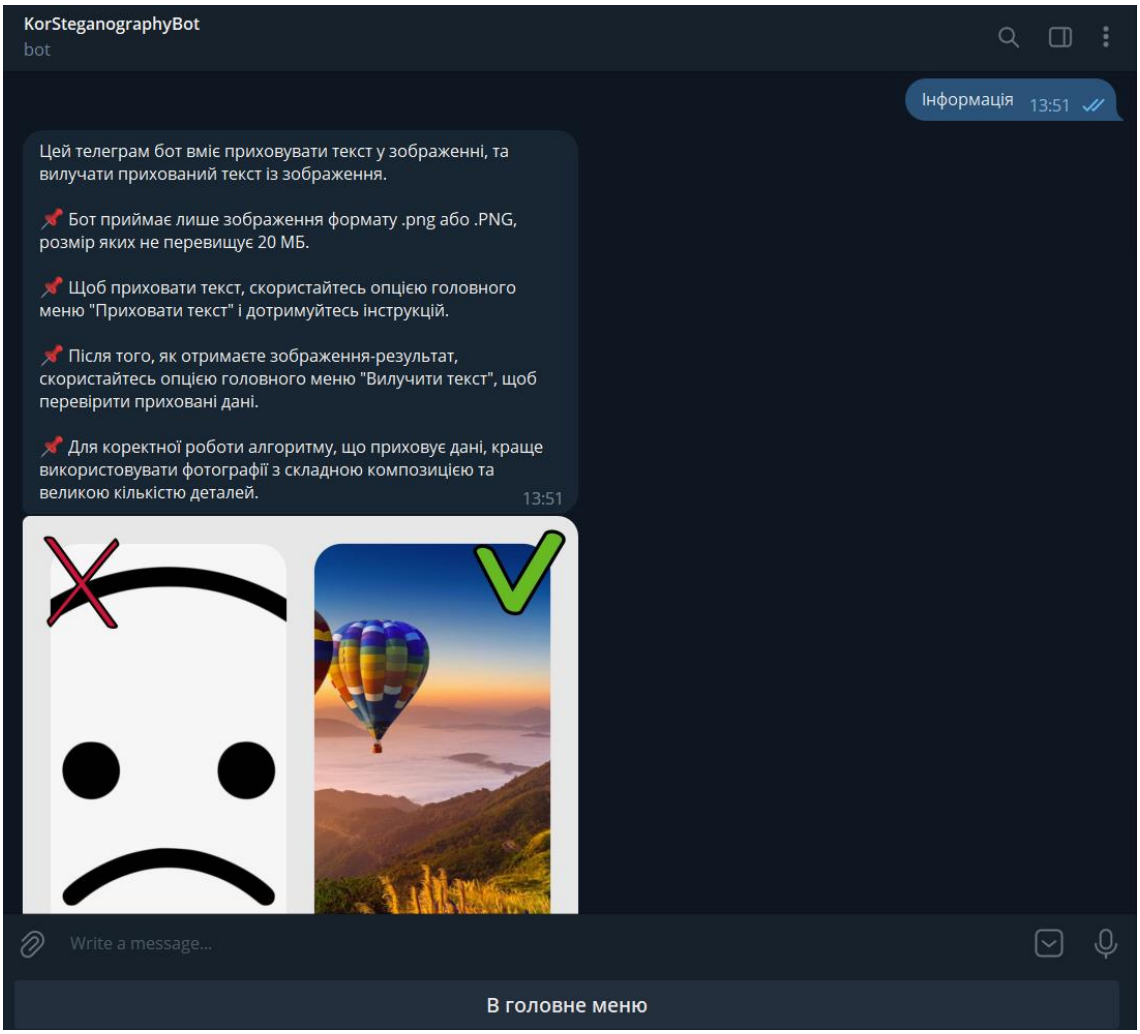

Рисунок 16.3 – Пункт меню «Інформація»

Щоб приховати текст у зображенні, Боб переходить в головне меню і обирає пункт «Приховати текст» (Рисунок 16.4).

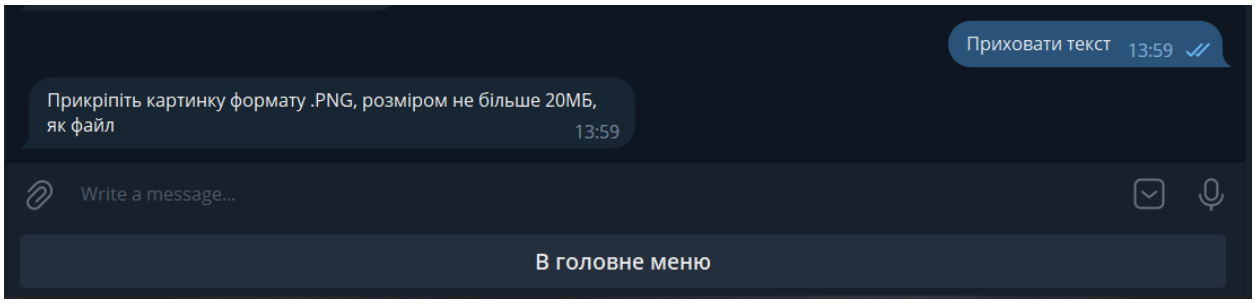

Рисунок 16.4 – Приховання тексту. Крок 1

Боб обрав зображення та прикріплює це зображення без стиснення (Рисунок 16.5.).

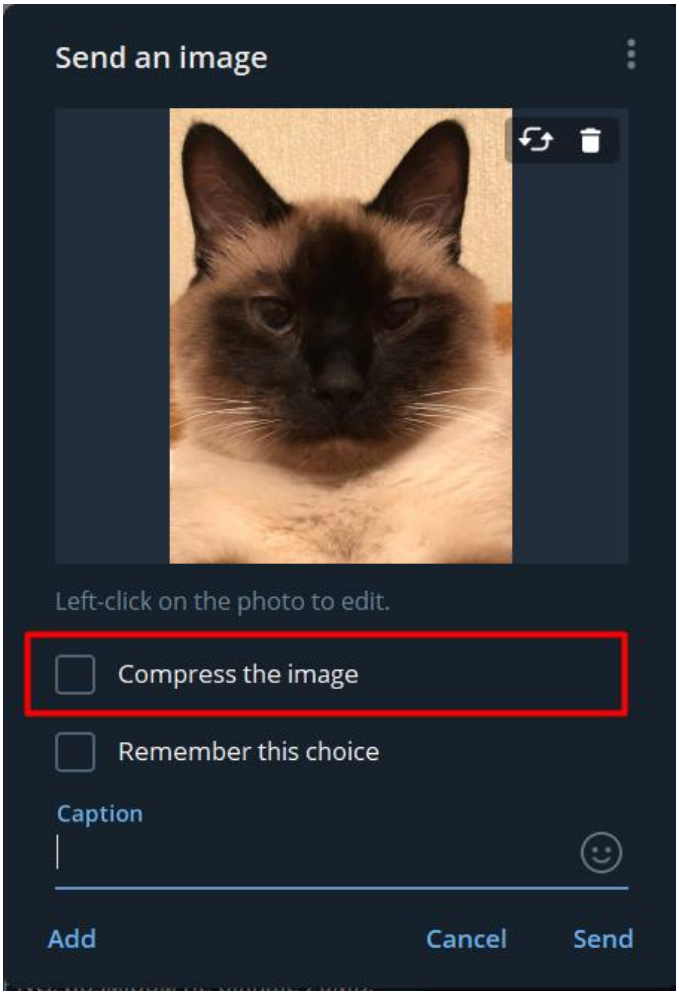

Рисунок 16.5. – Прикріплення зображення без стиснення

Зображення було завантажене. Далі необхідно відправити повідомлення, що буде приховано (Рисунок 16.6).

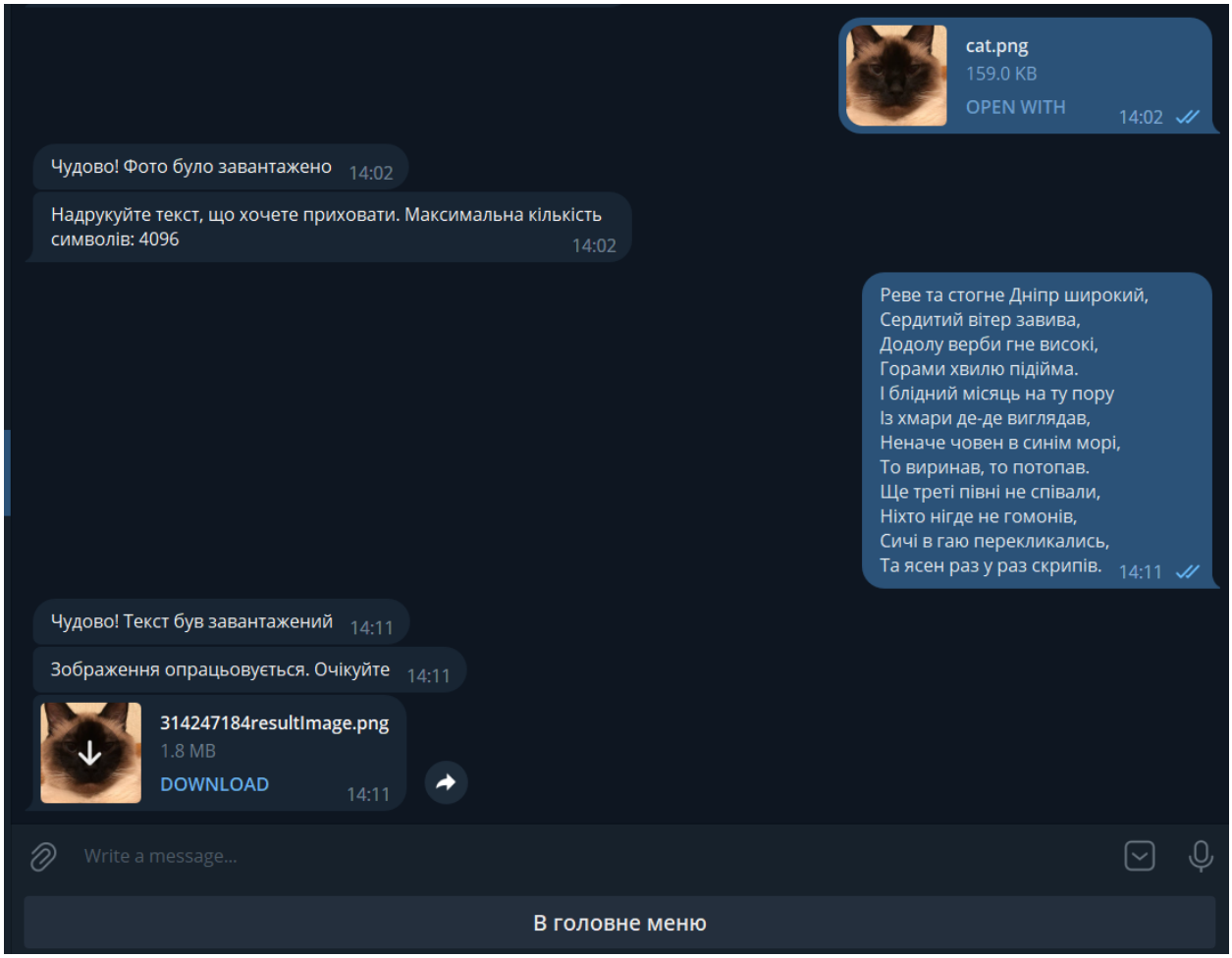

Рисунок 16.6 – Завантажено фото та текст. Отримано результат.

Щоб переконатись у правильності роботи алгоритму, Боб скористався пунктом головного меню «Вилучити текст» і прикріпив картинку, що отримав на попередньому кроці (Рисунок 16.7).

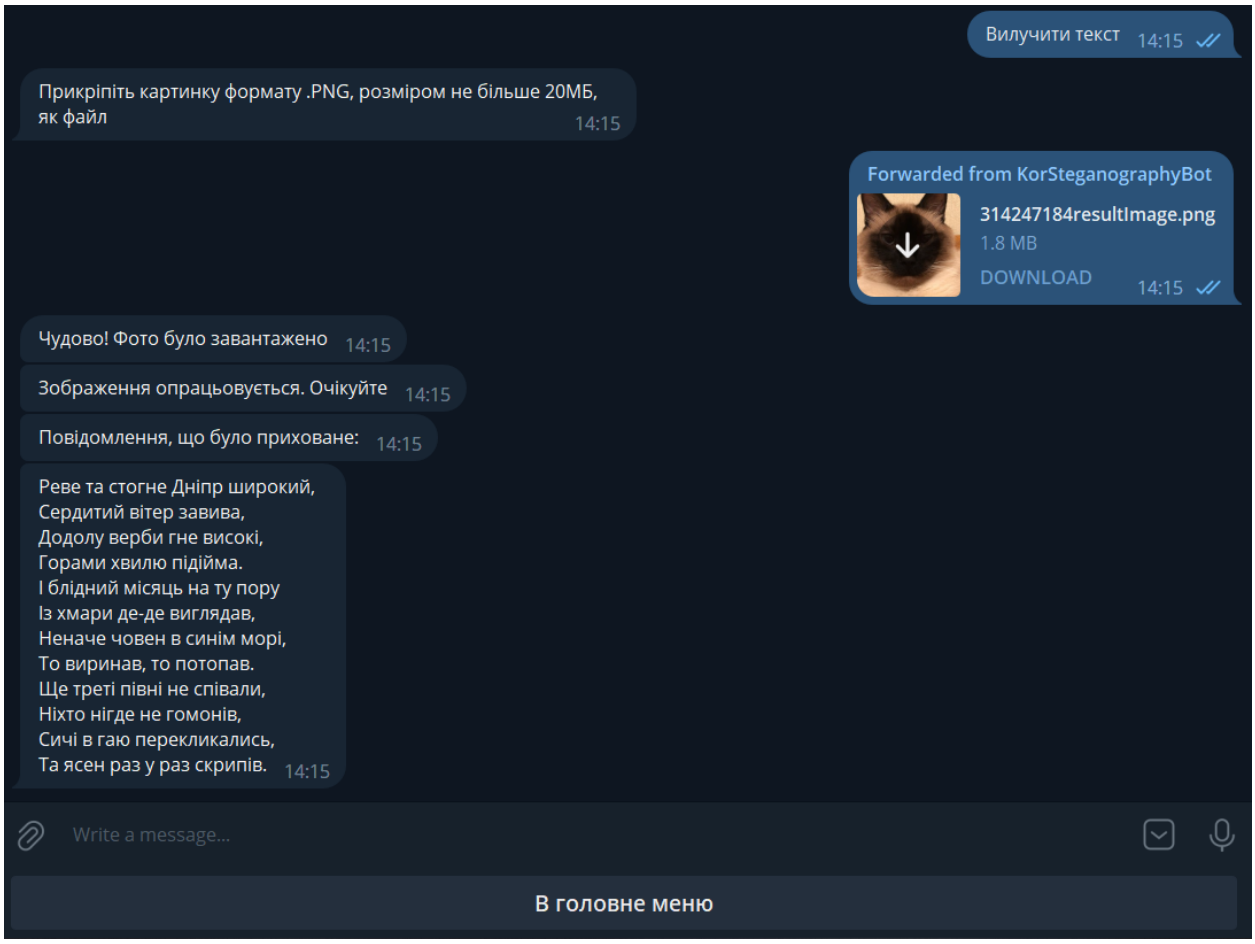

Рисунок 16.7 – Вилучення повідомлення із зображення

Коли Боб переконався, що повідомлення було приховано правильно, він відправляю це зображення Алісі.

Аліса, користуючись пунктом меню «Вилучити текст», отримає приховане повідомлення.

Для стороннього спостерігача спілкування Аліси та Боба виглядало як звичайний обмін зображеннями.

ВИСНОВКИ

В результаті роботи над науковим проектом «Телеграм-бот, що використовує стеганографічні алгоритми для приховування інформації в цифрових контейнерах» було:

- Сформульовано вимоги до створюваного ПЗ
- Проаналізовано існуючі рішення
- Обрано платформу для реалізації
- Визначено стек технологій
- Розроблено покращену версію стеганографічного алгоритму Куттера-Джордана-Боссена
- Проведено тестування створеного алгоритму на стійкість до різних видів атак
- Розроблено програмний застосунок із інтерфейсом користувача на базі месенджеру Телеграм
- Проведено тестування та виправлення помилок програмного застосунку
- Створено інструкцію користувача

Розроблений програмний застосунок забезпечує створення закритих каналів зв'язку з високим рівнем захисту даних та зручною комунікацією для військових та інших користувачів. Використання стеганографічних алгоритмів та месенджеру Телеграм створює надійну основу для приховування інформації в цифрових зображеннях і забезпечує надійну комунікацію, значно зменшуючи ризику виявлення та перехоплення повідомлень.

Це відкриває нові можливості для забезпечення безпеки та приватності комунікації, а також створює підґрунтя для подальших досліджень у галузі стеганографії та кібербезпеки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Official Telegram MTProto Mobile Protocol documentation [Електронний ресурс] – URL: https://core.telegram.org/mtproto (date of access: 20.11.2023)
2. Naughton P. Java: The Complete Reference (Complete Reference Series) / Patrick Naughton, Herbert Schildt. – [Б. м.] : Mcgraw-Hill Osborne Media, 1996. – 886 с.
3. Introduction to automata theory, languages and computation [Електронний ресурс] // Mathematics and Computers in Simulation. – 1981. – Т. 23, № 2. – С. 215. – Режим доступу: https://doi.org/10.1016/0378-4754(81)90068-9 (дата звернення: 20.11.2023)
4. Jordan F. Digital watermarking of color images using amplitude modulation [Electronic resource] / Fre´de´ric Jordan // Journal of Electronic Imaging. – 1998. – Vol. 7, no. 2. – P. 326. – Mode of access: https://doi.org/10.1117/1.482648 (date of access: 20.11.2023)
5. Kostyukov V. E. Стеганографические системы. Критерии и методическое обеспечение [Електронний ресурс] / V. E. Kostyukov, A. P. Martynov, D. B. Nikolaev ; ред. V. G. Gribunin. – Саров : Россйс. Федеральный ядерный центр - Всероссийс. науч.-исследовательс. институт экспериментальной физики, 2016. – Режим доступу: https://doi.org/10.53403/9785951503176 (дата звернення: 20.11.2023)
6. ITU BT.601. Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios [Електронний ресурс]. – Вид. офіц. – [Б. м. : б. в.]. – Режим доступу: https://www.itu.int/rec/R-REC-BT.601/ (дата звернення: 20.11.2023)
7. Грибунин В.Г., Оков И.Н., Туринцев И.В. Цифровая стеганография. — М.: Солон-Пресс, 2002.
8. Основи комп'ютерної стеганографії : Навчальний посібник для студентів і аспірантів. Хорошко В.О. та ін.— Вінниця: ВДТУ, 2003
9. Быков С.Ф. Мотуз О. В. Основы стегоанализа "Защита информации Конфидент" N 3 2000г

10. Wang, W.; Dong, J.; Tan, T. (October 2010). "Tampered Region Localization of Digital Color Images". Digital Watermarking: 9th International Workshop, IWDW 2010. Seoul, Korea: Springer. pp. 120–133. ISBN 9783642184048.
11. Suri J. S. Handbook of Image-Based Security Techniques / Jasjit S. Suri, Shivendra Shivani, Suneeta Agarwal. – [Б. м.] : Taylor & Francis Group, 2018. – 412 с.
12. Yahya A. Steganography Techniques [Електронний ресурс] / Abid Yahya // Steganography Techniques for Digital Images. – Cham, 2018. – Режим доступу: https://doi.org/10.1007/978-3-319-78597-4_2 (дата звернення: 20.11.2023)

ДОДАТОК А

Сертифікат про участь у міжнародній конференції “V International Scientific and Practical Conference. Education and science of today: THEORETICAL AND EMPIRICAL SCIENTIFIC RESEARCH: CONCEPTS AND TRENDS”, що відбулась 23 червня 2023 року у Оксфорді (Англія), і публікацію наукової статті “ТЕЛЕГРАМ-БОТ, ЩО ВИКОРИСТОВУЄ СТЕГАНОГРАФІЧНІ АЛГОРИТМИ ДЛЯ ПРИХОВУВАННЯ ІНФОРМАЦІЇ В ЦИФРОВИХ ЗОБРАЖЕННЯХ”.

Посилання на DOI: https://doi.org/10.36074/logos-14.10.2022

ДОДАТОК Б

Сертифікат про участь у «I міжнародній конференції ТЕОРІЯ МОДЕРНІЗАЦІЇ В КОНТЕКСТІ СУЧАСНОЇ СВІТОВОЇ НАУКИ», що відбулась 23 червня 2023 року у Полтаві (Україна), і публікацію наукової статті “ ТЕЛЕГРАМ-БОТ, ЩО ВИКОРИСТОВУЄ СТЕГАНОГРАФІЧНІ АЛГОРИТМИ ДЛЯ ПРИХОВУВАННЯ ІНФОРМАЦІЇ В ЦИФРОВИХ ЗОБРАЖЕННЯХ ”.

Посилання на DOI: https://doi.org/10.36074/mcnd-23.06.2023

ДОДАТОК В

ТЕКСТ ПРОГРАМИ

Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп’ютерних наук

Спеціальність 125 «Кібербезпека»

Освітня програма «Безпека інформаційних та комунікаційних систем»

ЗАТВЕРДЖУЮ
Керівник курсового проекту,
посада
к.т.н., доцент Громико І.О.

«Телеграм-бот, що використовує стеганографічні алгоритми для приховування інформації в цифрових контейнерах»

ТЕКСТ ПРОГРАМИ

ЛИСТ ЗАТВЕРДЖЕННЯ

ГЮІК.507200.006 – 01 12 01 – ЛЗ

РОЗРОБНИК:
ст. гр. КБ-61 Коршенко В.С.

2023

```java
package com.komin.steganobot.appconfig;

import com.komin.steganobot.MySteganoBot;
import com.komin.steganobot.botapi.TelegramFacade;
import lombok.Getter;
import lombok.Setter;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.MessageSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.support.ReloadableResourceBundleMessageSource;
import org.telegram.telegrambots.bots.DefaultBotOptions;

@Setter
@Getter
@Configuration
@ConfigurationProperties(prefix = "telegrambot")
public class BotConfig {

    private String webHookPath;
    private String botUserName;
    private String botToken;

    @Bean
    public MySteganoBot mySteganoBot(TelegramFacade telegramFacade) {
        MySteganoBot mySteganoBot = new MySteganoBot(new DefaultBotOptions(),
telegramFacade);
        mySteganoBot.setBotUserName(botUserName);
        mySteganoBot.setBotToken(botToken);
        mySteganoBot.setWebHookPath(webHookPath);
        return mySteganoBot;
    }

    @Bean
    public MessageSource messageSource() {
        ReloadableResourceBundleMessageSource messageSource
                = new ReloadableResourceBundleMessageSource();
        messageSource.setBasename("classpath:messages");
        messageSource.setDefaultEncoding("UTF-8");
        return messageSource;
    }
}
```

Рисунок В.1 - Код класу BotConfig

```java
package com.komin.steganobot;

import com.komin.steganobot.botapi.BotState;
import com.komin.steganobot.botapi.TelegramFacade;
import com.komin.steganobot.files_service.FilesService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.util.ResourceUtils;
import org.telegram.telegrambots.bots.DefaultBotOptions;
import org.telegram.telegrambots.bots.TelegramWebhookBot;
import org.telegram.telegrambots.meta.api.methods.BotApiMethod;
import org.telegram.telegrambots.meta.api.methods.send.SendDocument;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.methods.send.SendPhoto;
import org.telegram.telegrambots.meta.api.objects.InputFile;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;
import steganoAlgorythms.KJBHandler;

import java.io.File;
import java.io.FileNotFoundException;
```

```
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.Files;
import java.nio.file.StandardCopyOption;

@Slf4j
public class MySteganoBot extends TelegramWebhookBot {

    private String webHookPath;
    private String botUserName;
    private String botToken;
    private static boolean isDecodingWasPerformed = false;

    private final TelegramFacade telegramFacade;

    public MySteganoBot(DefaultBotOptions botOptions, TelegramFacade
telegramFacade) {
        super(botOptions);
        this.telegramFacade = telegramFacade;
    }

    @Override
    public String getBotToken() {
        return botToken;
    }

    @Override
    public String getBotUsername() {
        return botUserName;
    }

    @Override
    public String getBotPath() {
        return webHookPath;
    }

    @Override
    public BotApiMethod<?> onWebhookUpdateReceived(Update update) {
        if (update != null) {
            sendMessage(update);
            sendTip(update);
            encodeIfNeeded(update);
            decodeIfNeeded(update);
        }
        return null;
    }

    public static void nullIsDecodingWasPerformed() {
        System.out.println("IS DECODING WAS PERFORMED = " +
isDecodingWasPerformed);
        isDecodingWasPerformed = false;
    }

    private void sendMessage(Update update) {
        SendMessage replyMessageToUser = telegramFacade.handleUpdate(update);
        if (replyMessageToUser == null) {
            return;
        }
        try {
            execute(replyMessageToUser);
        } catch (TelegramApiException e) {
            e.printStackTrace();
        }
    }

    private void sendTip(Update update) {
```

```
        SendMessage tipMessageToUser =
telegramFacade.handleTip(update.getMessage());
        if (tipMessageToUser == null) {
            return;
        }
        try {
            execute(tipMessageToUser);
            sendTipImage(update);
        } catch (TelegramApiException e) {
            e.printStackTrace();
        }
    }

    private void encodeIfNeeded(Update update) {
        if (update.getMessage() != null) {
            long chadId = update.getMessage().getChatId();

            if (telegramFacade.isFilesReadyToEncode(chadId)) {
                log.info("[{}] Files for User: {} are ready to be encoded",
                        update.getMessage().getChatId(),
                        update.getMessage().getFrom().getUserName());
                try {
                    KJBHandler.encode(update);
                    sendImageAsDocument(update, "");
                    FilesService.deleteUserCache(update);
                } catch (IOException e) {
                    e.printStackTrace();
                    //TODO CATCH
                }
            }
        }
    }

    private void decodeIfNeeded(Update update) {
        if (update.getMessage() != null && !isDecodingWasPerformed) {
            long chatId = update.getMessage().getChatId();
            if (telegramFacade.isFilesReadyToDecode(chatId)) {
                log.info("[{}] Files for User: {} are ready to be decoded",
                        update.getMessage().getChatId(),
                        update.getMessage().getFrom().getUserName());
                try {
                    String decodedText =
KJBHandler.decode(String.valueOf(chatId));
                    log.info("[{}] File inputImage for User: {} was decoded
successfully",
                            update.getMessage().getChatId(),
                            update.getMessage().getFrom().getUserName());

                    sendLongMessage(chatId, decodedText);

                    log.info("[{}] Reply for User: {}, with decoded text",
                            update.getMessage().getChatId(),
                            update.getMessage().getFrom().getUserName());

                    FilesService.deleteUserCache(update);
                } catch (Exception e) {
                    e.printStackTrace();
                    try {
                        execute(new SendMessage(String.valueOf(chatId),
                                "Цей контейнер не містить приховане
повідомлення"));
                    } catch (TelegramApiException ex) {
                        e.printStackTrace();
                        //TODO CATCH
                    }
```

```
                    log.info("[{}] Reply for User: {}, with text: {}",
                            update.getMessage().getChatId(),
                            update.getMessage().getFrom().getUserName(),
                            "Цей контейнер не містить приховане повідомлення");

                }
            }
            isDecodingWasPerformed = true;
        }
    }

    private void sendTipImage(Update update) {
        long chatID = update.getMessage().getChatId();
        if
(telegramFacade.getUserDataCache().getUserCurrentBotState(chatID).equals(BotState.
ABOUT_INFO_STATE)) {
            try {
                InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("example.jpg");
                File imagePath = new File("example.png");
                assert inputStream != null;
                Files.copy(inputStream, imagePath.toPath(),
StandardCopyOption.REPLACE_EXISTING);
                InputFile image = new InputFile(imagePath);
                SendPhoto sendPhoto = new SendPhoto();
                sendPhoto.setPhoto(image);
                sendPhoto.setChatId(chatID);
                execute(sendPhoto);
            } catch (TelegramApiException | IOException e) {
                e.printStackTrace();
            }
        }
    }

    private void sendLongMessage(long chatId, String text) throws
TelegramApiException {
        System.out.println("SendLongMessage was called");
        if (text.startsWith("container_was_damaged_error")) {
            if (!isDecodingWasPerformed) {
                execute(new SendMessage(String.valueOf(chatId),
                        "Контейнер був випадково або навмисно пошкоджений, що
призвело до повної втрати інформації."));
            }
        } else {
            if (text.startsWith("message_was_extracted_partly_error")) {
                execute(new SendMessage(String.valueOf(chatId), "Контейнер був
випадково або навмисно пошкоджений, що призвело до часткової втрати
інформації."));
                execute(new SendMessage(String.valueOf(chatId), "Частина
повідомлення, що було приховане:"));
                text = text.substring(34);
            } else {
                execute(new SendMessage(String.valueOf(chatId), "Повідомлення, що
було приховане:"));
            }
            while (text.length() > 4096) {
                execute(new SendMessage(String.valueOf(chatId), text.substring(0,
4095)));
                text = text.substring(4096);
            }
            execute(new SendMessage(String.valueOf(chatId), text));
        }
    }

    private void sendImageAsDocument(Update update, String caption) {
        long chatId = update.getMessage().getChatId();
```

```
        try {
            InputFile image = new
InputFile(ResourceUtils.getFile(FilesService.downloadedFilesPath
                    + chatId + "resultImage" + FilesService.lastFileExtension));
            SendDocument sendDocument = new SendDocument();
            sendDocument.setDocument(image);
            sendDocument.setChatId(chatId);
            sendDocument.setCaption(caption);
            execute(sendDocument);
        } catch (TelegramApiException | FileNotFoundException e) {
            e.printStackTrace();
            //TODO CATCH
        }
        log.info("[{}] File resultImage for User: {} was sent successfully",
                update.getMessage().getChatId(),
                update.getMessage().getFrom().getUserName());
    }

    public void setWebHookPath(String webHookPath) {
        this.webHookPath = webHookPath;
    }

    public void setBotUserName(String botUserName) {
        this.botUserName = botUserName;
    }

    public void setBotToken(String botToken) {
        this.botToken = botToken;
    }
}
```

Рисунок В.2 - Код класу MySteganoBot

```
package com.komin.steganobot;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SteganoBotApplication {

    public static void main(String[] args) {
        SpringApplication.run(SteganoBotApplication.class, args);
    }

}
```

Рисунок В.3 - Код класу SteganoBotApplication

```
package steganoAlgorythms;

import com.komin.steganobot.files_service.FilesService;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;

import javax.imageio.ImageIO;

public class SpiralKutterJordanBossen {
```

```
    /*
     * Every bit is encoded for NUM_OF_REPEATS times
     * in order to improve noise stability.
     */
    private final static int NUM_OF_REPEATS = 15;
    private final static int STEP = 4;
    private final static double ENERGY = 0.25;

    /*Settings of the mark of the beginning of the coordinates*/
    private final static String MARK_STRING = "ABCDEFGHI";
    private static int STEPS_TO_CHANGE_DIR;
    private static int DIRECTION;
    private static int BASIC_STEPS_QUANTITY;
    private static int BASIC_STEPS_QUANTITY_INCREMENT_COUNTER;

    /*Settings of the mode of the mark searching algorithm*/
    private static final boolean CONSIDER_REFLECTION = true;
    private static final boolean CONSIDER_ROTATION = true;

    private static int NUMBER_OF_SEQUENCE__FOR_READING_FROM_PIXEL = 0;

    private static final int[][] readingSequences =
            {{0, 1, 2, 3}, {2, 1, 0, 3}, {2, 3, 0, 1}, {0, 3, 2, 1}, {3, 0, 1, 2},
{3, 2, 1, 0}, {1, 2, 3, 0}, {1, 0, 3, 2}};

    /* The current position of an image point which has information being written
into it */
    private static int xPos;
    private static int yPos;

    public static BufferedImage encode(String text, BufferedImage bitmap) {

        byte[] message = prepareTextToEncode(text);

        xPos = bitmap.getWidth() / 2;
        yPos = bitmap.getHeight() / 2;
        STEPS_TO_CHANGE_DIR = 1;
        DIRECTION =
readingSequences[NUMBER_OF_SEQUENCE__FOR_READING_FROM_PIXEL][0];
        BASIC_STEPS_QUANTITY = 1;
        BASIC_STEPS_QUANTITY_INCREMENT_COUNTER = 1;
        /* Creating a result image */
        BufferedImage result = makeImageCopy(bitmap);

        /* Encoding */
        for (byte b : message) {
            writeByte(result, b);
        }

        return result;
    }

    private static byte[] prepareTextToEncode(String text) {

        /* Converting text to byte array */
        byte[] msgBytes = text.getBytes();

        byte[] msgLength = new byte[4];
        msgLength[3] = (byte) (msgBytes.length & 0xff);
        msgLength[2] = (byte) ((msgBytes.length >> 8) & 0xff);
        msgLength[1] = (byte) ((msgBytes.length >> 16) & 0xff);
        msgLength[0] = (byte) ((msgBytes.length >> 24) & 0xff);

        /* Preparing information to insert */
        byte[] message = new byte[MARK_STRING.length() + 4 + msgBytes.length];
```

```
        System.arraycopy(MARK_STRING.getBytes(), 0, message, 0,
MARK_STRING.length());
        System.arraycopy(msgLength, 0, message, MARK_STRING.length(),
msgLength.length);
        System.arraycopy(msgBytes, 0, message, MARK_STRING.length() +
msgLength.length, msgBytes.length);

        return message;
    }

    public static String decode(BufferedImage bitmap) {
        ArrayList<Character> messageCharacters = new ArrayList<>();
        try {
            ArrayList<PixelCoordinates> coordinates =
calculateXCoordinatesOfSpiral(bitmap);
            char followingCharacter;
            for (PixelCoordinates pixel : coordinates) {

                /* Decoding */
                for (int readingStep = 0; readingStep <
calculateQuantityOfReadingsFromPixel(); readingStep++) {
                    xPos = pixel.getX();
                    yPos = pixel.getY();
                    STEPS_TO_CHANGE_DIR = 1;
                    DIRECTION =
readingSequences[NUMBER_OF_SEQUENCE__FOR_READING_FROM_PIXEL][0];
                    BASIC_STEPS_QUANTITY = 1;
                    BASIC_STEPS_QUANTITY_INCREMENT_COUNTER = 1;
                    messageCharacters = new ArrayList<>();
                    try {
                        while (messageCharacters.size() < 9) {
                            followingCharacter = (char) readByte(bitmap);
                            messageCharacters.add(followingCharacter);
                        }
                    } catch (IndexOutOfBoundsException e) {
                        return "container_was_damaged_error";
                    }
                    if (isMarkValid(messageCharacters)) {
                        byte[] msgLength = new byte[4];
                        for (int i = 0; i < 4; i++) {
                            msgLength[i] = readByte(bitmap);
                        }
                        long value = convert4ByteArrayToInteger(msgLength);
                        messageCharacters = new ArrayList<>();
                        for (int i = 0; i < value; i++) {
                            followingCharacter = (char) readByte(bitmap);
                            messageCharacters.add(followingCharacter);
                        }

                        return buildString(messageCharacters);
                    }
                    iterateNumberOfSequence();
                }
            }
        } catch (IndexOutOfBoundsException e) {
            StringBuilder result = new
StringBuilder(buildString(messageCharacters));
            result.insert(0,"message_was_extracted_partly_error");
            return result.toString();

        }
        return "";
    }

    private static void iterateNumberOfSequence() {
        if (CONSIDER_REFLECTION && !CONSIDER_ROTATION) {
```

```
            NUMBER_OF_SEQUENCE__FOR_READING_FROM_PIXEL =
(NUMBER_OF_SEQUENCE__FOR_READING_FROM_PIXEL + 1) % 4;
        } else if (!CONSIDER_REFLECTION && CONSIDER_ROTATION) {
            NUMBER_OF_SEQUENCE__FOR_READING_FROM_PIXEL =
(NUMBER_OF_SEQUENCE__FOR_READING_FROM_PIXEL + 2) % 8;
        } else if (CONSIDER_REFLECTION && CONSIDER_ROTATION) {
            NUMBER_OF_SEQUENCE__FOR_READING_FROM_PIXEL =
(NUMBER_OF_SEQUENCE__FOR_READING_FROM_PIXEL + 1) % 8;
        } else {
            NUMBER_OF_SEQUENCE__FOR_READING_FROM_PIXEL = 0;
        }
    }

    private static int calculateQuantityOfReadingsFromPixel() {
        if (CONSIDER_REFLECTION ^ CONSIDER_ROTATION) {
            return 4;
        } else if (CONSIDER_REFLECTION && CONSIDER_ROTATION) {
            return 8;
        }
        return 1;
    }

    private static long convert4ByteArrayToInteger(byte[] msgLength) {
        long value = msgLength[3] & 0xFF;
        value |= ((msgLength[2] << 8) & 0xFF00);
        value |= ((msgLength[1] << 16) & 0xFF0000);
        value |= (((msgLength[0] & 0x7F) << 24) & 0xFF000000);
        return value;
    }

    private static boolean isMarkValid(ArrayList<Character> characters) {
        int occurrencesCounter = 0;
        for (Character ch : MARK_STRING.toCharArray()) {
            if (characters.contains(ch)) {
                occurrencesCounter++;
            }
        }

        return occurrencesCounter > 7;
    }

    private static String buildString(ArrayList<Character> messageCharacters) {
        byte[] result = new byte[messageCharacters.size()];
        for (int i = 0; i < messageCharacters.size(); i++) {
            result[i] = (byte) messageCharacters.get(i).charValue();
        }

        return new String(result, StandardCharsets.UTF_8);
    }

    private static ArrayList<PixelCoordinates>
calculateXCoordinatesOfSpiral(BufferedImage bitmap) {
        ArrayList<PixelCoordinates> result = new ArrayList<>();
        int X = bitmap.getWidth() / 2;
        int Y = bitmap.getHeight() / 2;
        int stepsToChangeDir = 1;
        int direction = 0;
        int basicStepsQuantity = 1;

        while (true) {
            result.add(new PixelCoordinates(X, Y));
            switch (direction) {
                case 0: {
                    X += 1;
                    break;
                }
```

```
                case 1: {
                    Y -= 1;
                    break;
                }
                case 2: {
                    X -= 1;
                    break;
                }
                default: {
                    Y += 1;
                    break;
                }
            }
            stepsToChangeDir--;
            if (stepsToChangeDir < 1) {
                direction = (direction + 1) % 4;
                stepsToChangeDir = basicStepsQuantity;
                if (direction % 2 != 0) {
                    basicStepsQuantity++;
                    if ((basicStepsQuantity > bitmap.getWidth()) ||
(basicStepsQuantity > bitmap.getHeight())) {
                        break;
                    }
                }
            }
        }

        return result;
    }

    private static void writeByte(BufferedImage img, byte byteVal) {

        /* Loop through 8 bits of byteVal byte */
        for (int j = 7; j >= 0; j--) {
            int bitVal = (byteVal >>> j) & 1;
            writeBit(img, bitVal);
        }
    }

    private static byte readByte(BufferedImage img) {

        byte byteVal = 0;

        /* Getting a byte from 8 bits */
        for (int i = 0; i < 8; i++) {
            /* Left shift founded bits and add a bit to the right */
            byteVal = (byte) ((byteVal << 1) | (readBit(img) & 1));
        }

        return byteVal;
    }

    private static void writeBit(BufferedImage img, int bit) {
        for (int i1 = 0; i1 < NUM_OF_REPEATS; i1++) {
            writeIntoPixel(img, xPos, yPos, bit);
            calculateNewCoordinates();
            calculateNewDirection();
        }
    }

    private static void calculateNewCoordinates() {
        switch (DIRECTION) {
            case 0: {
                xPos += STEP;
                break;
            }
```

```
            case 1: {
                yPos -= STEP;
                break;
            }
            case 2: {
                xPos -= STEP;
                break;
            }
            default: {
                yPos += STEP;
                break;
            }
        }
    }

    private static void calculateNewDirection() {
        STEPS_TO_CHANGE_DIR--;
        if (STEPS_TO_CHANGE_DIR < 1) {
            BASIC_STEPS_QUANTITY_INCREMENT_COUNTER--;
            DIRECTION = getFollowingDirectionFromCurrentSequence();
            STEPS_TO_CHANGE_DIR = BASIC_STEPS_QUANTITY;
            if (BASIC_STEPS_QUANTITY_INCREMENT_COUNTER < 1) {
                BASIC_STEPS_QUANTITY++;
                BASIC_STEPS_QUANTITY_INCREMENT_COUNTER = 2;
            }
        }
    }

    private static int getFollowingDirectionFromCurrentSequence() {
        int followingElementIndex = 0;
        for (int i = 0; i < 4; i++) {
            int currentElement =
readingSequences[NUMBER_OF_SEQUENCE__FOR_READING_FROM_PIXEL][i];
            if (currentElement == DIRECTION) {
                followingElementIndex = (i + 1) % 4;
            }
        }
        return
readingSequences[NUMBER_OF_SEQUENCE__FOR_READING_FROM_PIXEL][followingElementIndex
];
    }

    private static int readBit(BufferedImage img) {
        /* Probabilistic estimate of an information bit */
        float bitEstimate = 0;

        for (int i1 = 0; i1 < NUM_OF_REPEATS; i1++) {
            bitEstimate += readFromPixel(img, xPos, yPos);
            calculateNewCoordinates();
            calculateNewDirection();
        }
        bitEstimate /= NUM_OF_REPEATS;

        /* if more than half of NUM_OF_REPEATS read bits were 1s, so consider 1
was encoded */
        if (bitEstimate > 0.5) {
            return 1;
        } else {
            return 0;
        }
    }

    private static void writeIntoPixel(BufferedImage image, int x, int y, int bit)
{

        Color pixel = new Color(image.getRGB(x, y));
```

```
        int red = pixel.getRed();
        int green = pixel.getGreen();
        int blue = pixel.getBlue();

        int pixelBrightness = (int) (0.29890 * red + 0.58662 * green + 0.11448 *
blue);

        /* Variable blue component */
        int modifiedBlueComponent;
        if (bit > 0) {

            modifiedBlueComponent = (int) (blue + ENERGY * pixelBrightness);

        } else {

            modifiedBlueComponent = (int) (blue - ENERGY * pixelBrightness);
        }

        if (modifiedBlueComponent > 255) {

            modifiedBlueComponent = 255;
        }

        if (modifiedBlueComponent < 0) {

            modifiedBlueComponent = 0;
        }

        Color pixelModified = new Color(red, green, modifiedBlueComponent);
        image.setRGB(x, y, pixelModified.getRGB());

    }

    private static int readFromPixel(BufferedImage image, int x, int y) {

        /* Summing up all the blue components of surrounding points */
        int estimate = 0;

        for (int i1 = 1; i1 <= 3; i1++) {

            Color pixel = new Color(image.getRGB(x + i1, y));
            estimate += pixel.getBlue();
        }

        for (int i1 = 1; i1 <= 3; i1++) {

            Color pixel = new Color(image.getRGB(x - i1, y));
            estimate += pixel.getBlue();
        }

        for (int i1 = 1; i1 <= 3; i1++) {

            Color pixel = new Color(image.getRGB(x, y + i1));
            estimate += pixel.getBlue();
        }

        for (int i1 = 1; i1 <= 3; i1++) {

            Color pixel = new Color(image.getRGB(x, y - i1));
            estimate += pixel.getBlue();
        }

        /* Average */
        estimate /= 12;

        Color pixel = new Color(image.getRGB(x, y));
```

```
        int blue = pixel.getBlue();

        if (blue > estimate) {

            return 1;

        } else {

            return 0;
        }
    }

    private static BufferedImage makeImageCopy(BufferedImage imageToCopy) {
        BufferedImage result = new BufferedImage(imageToCopy.getWidth(),
imageToCopy.getHeight(), imageToCopy.getType());
        Graphics g = result.getGraphics();
        g.drawImage(imageToCopy, 0, 0, null);
        return result;
    }

    public static int evaluatePossibleCharQuantity(String chatID) {
        String fileName = FilesService.getInputImageNameByChatId(chatID);
        File initFile = new File(fileName);
        int result = 0;
        try {
            BufferedImage initImage = ImageIO.read(initFile);
            result = (int) ((Math.pow(Math.min(initImage.getWidth(),
initImage.getHeight()) / STEP, 2.0) / (8 * NUM_OF_REPEATS)) - 14);
        } catch (IOException e) {
            e.printStackTrace();
        }

        return Math.min(result, 4096);
    }
}
```

Рисунок В.4 - Код класу SpiralKutterJordanBossen

```
package steganoAlgorythms;

import com.komin.steganobot.files_service.FilesService;
import lombok.extern.slf4j.Slf4j;
import org.telegram.telegrambots.meta.api.objects.Update;

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

import javax.imageio.ImageIO;

@Slf4j
public class KJBHandler {

    public static void encode(Update update) throws IOException {
        String chatId = update.getMessage().getChatId().toString();
        String filename = FilesService.downloadedFilesPath
                + chatId + "inputImage" + FilesService.lastFileExtension;
        File initFile = new File(filename);

        BufferedImage initImage = ImageIO.read(initFile);

        String messageToEncode =
FilesService.readTxt(FilesService.downloadedFilesPath
                + chatId + "inputText.txt", StandardCharsets.UTF_8);
```

```
        BufferedImage newImage = null;
        try {
            newImage = SpiralKutterJordanBossen.encode(messageToEncode,
initImage);
        } catch (Exception e) {
            e.printStackTrace();
        }

        File dir = new File(FilesService.downloadedFilesPath);
        if (dir.exists() && dir.isDirectory() && dir.canWrite()) {
            File finalImage = new File(dir + "/" + chatId + "resultImage.png");
            FilesService.lastFileExtension = ".png";
            ImageIO.write(newImage, "png", finalImage);
        } else {
            throw new IOException("Invalid !");
        }

        log.info("[{}] File with encoded text for User: {}, was created
successfully",
                update.getMessage().getChatId(),
                update.getMessage().getFrom().getUserName());
    }

    public static String decode(String chatId) throws Exception {
        String filePath = FilesService.downloadedFilesPath + chatId + "inputImage"
+ FilesService.lastFileExtension;
        File outFile = new File(filePath);

        BufferedImage image = ImageIO.read(outFile);
        if (image == null) {
            //TODO Create custom exception
            throw new IOException();
        }

        return SpiralKutterJordanBossen.decode(image);
    }
}
```

Рисунок В.5 - Код класу KJBHandler

```
package steganoAlgorythms;

public class PixelCoordinates {

    private int x;
    private int y;

    public PixelCoordinates(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }
```

```
    public void setY(int y) {
        this.y = y;
    }
}
```

Рисунок В.6 - Код класу PixelCoordinates

```
package com.komin.steganobot.service;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.MessageSource;
import org.springframework.stereotype.Component;

import java.util.Locale;

@Component
public class LocaleMessageService {

    private final Locale locale;
    private final MessageSource messageSource;

    public LocaleMessageService(@Value("ua-UA") String localeTag, MessageSource
messageSource) {
        this.messageSource = messageSource;
        this.locale = Locale.forLanguageTag(localeTag);
    }

    public String getMessage(String message) {
        return messageSource.getMessage(message, null, locale);
    }

    public String getMessage(String message, Object... args) {
        return messageSource.getMessage(message, args, locale);
    }
}
```

Рисунок В.7 - Код класу LocalMessageService

```
package com.komin.steganobot.service;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;

@Component
public class ReplyMessageService {

    private final LocaleMessageService localeMessageService;

    public ReplyMessageService(LocaleMessageService localeMessageService) {
        this.localeMessageService = localeMessageService;
    }

    public SendMessage getReplyMessage(String chat_id, String replyMessage) {
        return new SendMessage(chat_id,
localeMessageService.getMessage(replyMessage));
    }

    public SendMessage getReplyMessage(String chat_id, String replyMessage,
Object... args) {
        return new SendMessage(chat_id,
localeMessageService.getMessage(replyMessage, args));
    }
}
```

Рисунок В.8 - Код класу ReplyMessageService

```java
package com.komin.steganobot.files_service;

import com.komin.steganobot.botapi.BotState;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.io.FileUtils;
import org.json.JSONObject;
import org.telegram.telegrambots.meta.api.objects.Message;
import org.telegram.telegrambots.meta.api.objects.Update;

import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

import javax.imageio.ImageIO;

@Slf4j
public class FilesService {

    public static String lastFileExtension;

    //TODO
    public static final String downloadedFilesPath = "src/downloaded_files/";

    //TODO Get token from properties
    private static final String botToken = "xxxxxxxxxxxxxxxxxx";

    public static boolean downloadImage(Message inputMessage) throws IOException {
        String fileID;
        if (inputMessage.hasDocument()) {
            fileID = inputMessage.getDocument().getFileId();
        } else {
            fileID = inputMessage.getPhoto().get(0).getFileId();
        }

        String chatID = inputMessage.getChatId().toString();
        String userName = inputMessage.getFrom().getUserName();

        URL url = new URL("https://api.telegram.org/bot" + botToken +
"/getFile?file_id=" + fileID);
        BufferedReader br = new BufferedReader(new
InputStreamReader(url.openStream()));
        String getFileResponse = br.readLine();

        JSONObject jResult = new JSONObject(getFileResponse);
        boolean isOk = jResult.getBoolean("ok");
        if (!isOk) {
            return false;
        }

        JSONObject path = jResult.getJSONObject("result");
        String filePath = path.getString("file_path");
        int fileSize = path.getInt("file_size");
        if (fileSize > 2e7) {
```

```
            return false;
        }
        File localFile = new File(downloadedFilesPath + chatID + "inputImage" +
getFileExtension(filePath));
        InputStream is = new URL("https://api.telegram.org/file/bot" + botToken +
"/" + filePath)
                .openStream();
        FileUtils.copyInputStreamToFile(is, localFile);
        br.close();
        is.close();

        if (ImageIO.read(localFile) == null) {
            throw new IOException();
        }

        log.info("[{}] File {} from User: {} was downloaded successfully",
                chatID,
                "inputImage",
                userName);

        return true;

    }

    private static boolean convertJPGtoPNG(File localFile, String chatID) {
        FileInputStream inputStream = null;
        boolean result;
        try {
            inputStream = new FileInputStream(localFile.getPath());
            FileOutputStream outputStream = new
FileOutputStream(downloadedFilesPath + chatID + "inputImage.png");

            // reads input image from file
            BufferedImage inputImage = ImageIO.read(inputStream);

            // writes to the output image in specified format
            result = ImageIO.write(inputImage, "PNG", outputStream);

            // needs to close the streams
            outputStream.close();
            inputStream.close();

            localFile.delete();

        } catch (IOException e) {
            return false;
        }
        lastFileExtension = ".png";
        return result;
    }

    public static void saveUserString(Message inputMessage) {
        String text = inputMessage.getText();
        String chatId = inputMessage.getChatId().toString();
        try {
            FileUtils.writeStringToFile(new File(downloadedFilesPath + chatId +
"inputText.txt"),
                    text,
                    StandardCharsets.UTF_8);

            log.info("[{}] Text from User: {} was saved as inputText.txt",
                    chatId,
                    inputMessage.getFrom().getUserName());
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
    }

    public static String readTxt(String path, Charset encoding) {
        byte[] encoded = new byte[0];
        try {
            encoded = Files.readAllBytes(Paths.get(path));
        } catch (IOException e) {
            e.printStackTrace();
        }
        return new String(encoded, encoding);
    }

    private static String getFileExtension(String filePath) {
        StringBuilder result = new StringBuilder();
        int charIdx = filePath.length() - 1;
        while (filePath.charAt(charIdx) != '.') {
            result.append(filePath.charAt(charIdx));
            charIdx--;
        }
        result.append('.');
        result.reverse();
        lastFileExtension = result.toString();
        return result.toString();
    }

    public static String getInputImageNameByChatId(String chatID) {
        File directoryPath = new File(downloadedFilesPath);
        List<File> files =
List.of(Objects.requireNonNull(directoryPath.listFiles()));

        return files.stream()
                    .filter(file -> file.getName()
                                        .startsWith(chatID + "inputImage"))
                    .findFirst().get().toString().replaceAll("\\\\", "/");
    }

    public static void deleteUserCache(Update update) {
        String chatId = update.getMessage().getChatId().toString();
        log.info("[{}] Cache deleting for User: {}, was STARTED",
                chatId,
                update.getMessage().getFrom().getUserName());

        File file = new File(downloadedFilesPath
                + chatId + "resultImage" + lastFileExtension);
        logFileDeletingStatus(update, "resultImage", file.delete());

        file = new File(downloadedFilesPath
                + chatId + "inputImage" + lastFileExtension);
        logFileDeletingStatus(update, "inputImage", file.delete());

        file = new File(downloadedFilesPath
                + chatId + "inputText.txt");
        logFileDeletingStatus(update, "inputText.txt", file.delete());

        log.info("[{}] Cache deleting for User: {}, was ENDED",
                chatId,
                update.getMessage().getFrom().getUserName());

    }

    private static void logFileDeletingStatus(Update update, String fileName,
boolean status) {
        log.info("[{}] File {} from User: {}, was deleted: {}",
                update.getMessage().getChatId(),
                fileName,
                update.getMessage().getFrom().getUserName(),
```

```
                status);
    }

    public static boolean hasUserCacheForEncoding(long chatID) {
        File directoryPath = new File(downloadedFilesPath);
        List<File> files =
List.of(Objects.requireNonNull(directoryPath.listFiles()));

        return files.stream()
                    .filter(file ->
file.getName().startsWith(String.valueOf(chatID)))
                    .toList().size() == 2;
    }

    public static boolean hasUserCacheForDecoding(long chatID) {
        File directoryPath = new File(downloadedFilesPath);
        List<File> files =
List.of(Objects.requireNonNull(directoryPath.listFiles()));
        return files.stream()
                    .filter(file ->
file.getName().startsWith(String.valueOf(chatID)))
                    .toList().size() == 1;
    }

    public static boolean isFileExtensionValid(String fileName, BotState botState)
{
        //Add necessary extensions here
        ArrayList<String> validExtensions = null;
        switch (botState) {
            case HIDE_TEXT_IMAGE_UPLOAD_STATE: {
                validExtensions = new ArrayList<>(
                        List.of(".png")
                );
                break;
            }
            case UNPACK_TEXT_IMAGE_UPLOAD_STATE: {
                validExtensions = new ArrayList<>(
                        List.of(".png")
                );
                break;
            }
        }

        return validExtensions.stream().anyMatch(extension ->
fileName.toLowerCase().endsWith(extension));
    }
}
```

Рисунок В.9 - Код класу FilesService

```
package com.komin.steganobot.controller;

import com.komin.steganobot.MySteganoBot;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.telegram.telegrambots.meta.api.methods.BotApiMethod;
import org.telegram.telegrambots.meta.api.objects.Update;

@RestController
public class WebHookController {

    private final MySteganoBot mySteganoBot;
```

```
    public WebHookController(MySteganoBot mySteganoBot) {
        this.mySteganoBot = mySteganoBot;
    }

    @RequestMapping(value = "/", method = RequestMethod.POST)
    public BotApiMethod<?> onUpdateReceived(@RequestBody Update update) {
        return mySteganoBot.onWebhookUpdateReceived(update);
    }
}
```

Рисунок В.10 - Код класу WebHookController

```
package com.komin.steganobot.cache;

import com.komin.steganobot.MySteganoBot;
import com.komin.steganobot.botapi.BotState;
import org.springframework.stereotype.Component;

import java.util.HashMap;
import java.util.Map;

@Component
public class UserDataCache {

    // Add communication with DB here
    private final Map<Long, BotState> usersBotStates = new HashMap<>();
    private boolean newStateWasSet = false;

    public void setUserCurrentBotState(long userId, BotState botState) {
        usersBotStates.put(userId, botState);
        setNewStateWasSet(true);
        MySteganoBot.nullIsDecodingWasPerformed();
    }

    public BotState getUserCurrentBotState(long userId) {
        BotState botState = usersBotStates.get(userId);
        if (botState == null) {
            botState = BotState.INITIAL_STATE;
        }
        return botState;
    }

    public boolean isNewStateWasSet() {
        return newStateWasSet;
    }

    public void setNewStateWasSet(boolean newStateWasSet) {
        this.newStateWasSet = newStateWasSet;
    }
}
```

Рисунок В.11 - Код класу UserDataCache

```
package com.komin.steganobot.builder;

import
org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardButton;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;

import java.util.ArrayList;
import java.util.List;
```

```
public class ReplyKeyboardMarkupBuilder {

    public static ReplyKeyboardMarkup build(String... buttonNames) {
        final ReplyKeyboardMarkup replyKeyboardMarkup = new ReplyKeyboardMarkup();
        replyKeyboardMarkup.setSelective(true);
        replyKeyboardMarkup.setResizeKeyboard(true);
        replyKeyboardMarkup.setOneTimeKeyboard(false);

        List<KeyboardRow> keyboard = new ArrayList<>();

        for (String buttonName : buttonNames) {
            KeyboardRow row = new KeyboardRow();
            row.add(new KeyboardButton(buttonName));
            keyboard.add(row);
        }
        replyKeyboardMarkup.setKeyboard(keyboard);
        return replyKeyboardMarkup;
    }
}
```

Рисунок В.12 - Код класу ReplyKeyboardMarkupBuilder

```
package com.komin.steganobot.botapi;

public enum BotState {
    INITIAL_STATE,
    MAIN_MENU_STATE,
    HIDE_TEXT_IMAGE_UPLOAD_STATE,
    HIDE_TEXT_STRING_UPLOAD_STATE,
    HIDE_TEXT_RESULT_UPLOAD_STATE,
    UNPACK_TEXT_IMAGE_UPLOAD_STATE,
    UNPACK_TEXT_RESULT_UPLOAD_STATE,
    ABOUT_INFO_STATE
}
```

Рисунок В.13 - Код класу BotState

```
package com.komin.steganobot.botapi;

import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Component
public class BotStateContext {

    private final Map<BotState, InputMessageHandler> messageHandlers = new
HashMap<>();

    public BotStateContext(List<InputMessageHandler> messageHandlers) {
        messageHandlers.forEach(handler ->
this.messageHandlers.put(handler.getHandlerName(), handler));
    }

    public SendMessage processInputMessage(BotState currentState, Message message)
{
        return messageHandlers.get(currentState).handle(message);
    }
```

```
    public SendMessage processTipMessage(BotState currentState, Message message) {
        return messageHandlers.get(currentState).getStateTip(message);
    }
}
```

Рисунок В.14 - Код класу BotStateContext

```
package com.komin.steganobot.botapi;

import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;

public interface InputMessageHandler {

    SendMessage handle(Message message);

    BotState getHandlerName();

    SendMessage getStateTip(Message message);
}
```

Рисунок В.15 - Код класу InputMessageHandler

```
package com.komin.steganobot.botapi;

import com.komin.steganobot.cache.UserDataCache;
import com.komin.steganobot.files_service.FilesService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
import org.telegram.telegrambots.meta.api.objects.Update;

@Component
@Slf4j
public class TelegramFacade {

    private final BotStateContext botStateContext;
    private final UserDataCache userDataCache;

    public TelegramFacade(BotStateContext botStateContext, UserDataCache
userDataCache) {
        this.botStateContext = botStateContext;
        this.userDataCache = userDataCache;
    }

    public SendMessage handleUpdate(Update update) {
        SendMessage replyMessage = null;

        Message message = update.getMessage();
        if (message != null) {

            if
(userDataCache.getUserCurrentBotState(update.getMessage().getChatId())
                                .equals(BotState.HIDE_TEXT_STRING_UPLOAD_STATE)) {
                log.info("[{}] New message from User: {}, with text to hide,
hasPhoto: {}, hasDocument: {}",
                        message.getChatId(),
                        message.getFrom().getUserName(),
                        message.hasPhoto(),
                        message.hasDocument());
            } else {
                log.info("[{}] New message from User: {}, with text: {}, hasPhoto:
{}, hasDocument: {}",
```

```
                        message.getChatId(),
                        message.getFrom().getUserName(),
                        message.getText(),
                        message.hasPhoto(),
                        message.hasDocument());
            }

            replyMessage = handleInputMessage(message);
        }
        return replyMessage;
    }

    private SendMessage handleInputMessage(Message message) {
        long userId = message.getFrom().getId();
        BotState botState;
        SendMessage replyMessage;

        botState = userDataCache.getUserCurrentBotState(userId);
        replyMessage = botStateContext.processInputMessage(botState, message);
        return replyMessage;
    }

    public SendMessage handleTip(Message message) {
        if (userDataCache.isNewStateWasSet()) {
            userDataCache.setNewStateWasSet(false);
            return botStateContext.processTipMessage(

userDataCache.getUserCurrentBotState(message.getFrom().getId()), message);
        }
        return null;
    }

    public boolean isFilesReadyToEncode(long chatID) {
        return
userDataCache.getUserCurrentBotState(chatID).equals(BotState.HIDE_TEXT_RESULT_UPLO
AD_STATE)
                && FilesService.hasUserCacheForEncoding(chatID);
    }

    public boolean isFilesReadyToDecode(long chatID) {
        return userDataCache.getUserCurrentBotState(chatID)
                            .equals(BotState.UNPACK_TEXT_RESULT_UPLOAD_STATE)
                && FilesService.hasUserCacheForDecoding(chatID);
    }

    public UserDataCache getUserDataCache() {
        return this.userDataCache;
    }
}
```

Рисунок В.16 - Код класу TelegramFacade

```
package com.komin.steganobot.botapi.options;

import com.komin.steganobot.botapi.BotState;
import lombok.Getter;
import lombok.RequiredArgsConstructor;

import java.util.ArrayList;
import java.util.List;

import static com.komin.steganobot.botapi.BotState.ABOUT_INFO_STATE;
import static com.komin.steganobot.botapi.BotState.HIDE_TEXT_IMAGE_UPLOAD_STATE;
import static com.komin.steganobot.botapi.BotState.HIDE_TEXT_RESULT_UPLOAD_STATE;
import static com.komin.steganobot.botapi.BotState.HIDE_TEXT_STRING_UPLOAD_STATE;
```

```
import static com.komin.steganobot.botapi.BotState.INITIAL_STATE;
import static com.komin.steganobot.botapi.BotState.MAIN_MENU_STATE;
import static com.komin.steganobot.botapi.BotState.UNPACK_TEXT_IMAGE_UPLOAD_STATE;
import static
com.komin.steganobot.botapi.BotState.UNPACK_TEXT_RESULT_UPLOAD_STATE;

@Getter
@RequiredArgsConstructor
public enum MenuOptions {

    START_OPTION("option.initial-state-valid-option", MAIN_MENU_STATE, new
ArrayList<>(
            List.of(INITIAL_STATE))),

    BACK_TO_MAIN_MENU_OPTION("option.back-to-main-menu-valid-option",
MAIN_MENU_STATE, new ArrayList<>(
            List.of(HIDE_TEXT_IMAGE_UPLOAD_STATE,
                    HIDE_TEXT_STRING_UPLOAD_STATE,
                    HIDE_TEXT_RESULT_UPLOAD_STATE,
                    UNPACK_TEXT_IMAGE_UPLOAD_STATE,
                    UNPACK_TEXT_RESULT_UPLOAD_STATE,
                    ABOUT_INFO_STATE))),

    HIDE_TEXT_OPTION("option.main-menu-state-hide-text-option",
HIDE_TEXT_IMAGE_UPLOAD_STATE, new ArrayList<>(
            List.of(MAIN_MENU_STATE))),

    UNPACK_TEXT_OPTION("option.main-menu-state-unpack-text-option",
UNPACK_TEXT_IMAGE_UPLOAD_STATE, new ArrayList<>(
            List.of(MAIN_MENU_STATE))),

    ABOUT_INFO_OPTION("option.main-menu-state-about-info-option",
ABOUT_INFO_STATE, new ArrayList<>(
            List.of(MAIN_MENU_STATE)));

    private final String value;
    private final BotState followingBotState;
    private final List<BotState> optionAvailableStates;
}
```

Рисунок В.17 - Код класу MenuOptions

```
package com.komin.steganobot.botapi.handlers;

import com.komin.steganobot.botapi.BotState;
import com.komin.steganobot.botapi.InputMessageHandler;
import com.komin.steganobot.builder.ReplyKeyboardMarkupBuilder;
import com.komin.steganobot.cache.UserDataCache;
import com.komin.steganobot.service.LocaleMessageService;
import com.komin.steganobot.service.ReplyMessageService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;

@Slf4j
@Component
public class AboutInfoStateHandler extends StateHandler implements
InputMessageHandler {

    public AboutInfoStateHandler(UserDataCache userDataCache, ReplyMessageService
messageService, LocaleMessageService localeMessageService) {
```

```
        super(userDataCache, messageService, localeMessageService);
    }

    @Override
    public SendMessage handle(Message message) {
        return processUsersInput(message);
    }

    @Override
    public BotState getHandlerName() {
        return BotState.ABOUT_INFO_STATE;
    }

    @Override
    public SendMessage getStateTip(Message message) {
        return generateTip(message, generateKeyboard());
    }

    private SendMessage processUsersInput(Message inputMessage) {
        return checkMessageForRightOption(inputMessage);
    }

    private SendMessage generateTip(Message inputMessage, ReplyKeyboardMarkup
replyKeyboardMarkup) {
        long chat_id = inputMessage.getChatId();
        SendMessage replyTip = new SendMessage(String.valueOf(chat_id),
localeMessageService.getMessage("tip.about-info-state"));
        if (replyKeyboardMarkup != null) {
            replyTip.enableMarkdown(true);
            replyTip.setReplyMarkup(replyKeyboardMarkup);
        }
        return replyTip;
    }

    private ReplyKeyboardMarkup generateKeyboard() {
        String backToMainMenu = localeMessageService.getMessage("option.back-to-
main-menu-valid-option");

        return ReplyKeyboardMarkupBuilder.build(backToMainMenu);
    }
}
```

Рисунок В.18 - Код класу AboutInfoStateHandler

```
package com.komin.steganobot.botapi.handlers;

import com.komin.steganobot.botapi.BotState;
import com.komin.steganobot.botapi.InputMessageHandler;
import com.komin.steganobot.builder.ReplyKeyboardMarkupBuilder;
import com.komin.steganobot.cache.UserDataCache;
import com.komin.steganobot.files_service.FilesService;
import com.komin.steganobot.service.LocaleMessageService;
import com.komin.steganobot.service.ReplyMessageService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;

import java.io.IOException;

@Slf4j
@Component
public class HideTextImageUploadHandler extends StateHandler implements
```

```
InputMessageHandler {

    public HideTextImageUploadHandler(UserDataCache userDataCache,
ReplyMessageService messageService, LocaleMessageService localeMessageService) {
        super(userDataCache, messageService, localeMessageService);
    }

    @Override
    public SendMessage handle(Message message) {
        return processUsersInput(message);
    }

    @Override
    public BotState getHandlerName() {
        return BotState.HIDE_TEXT_IMAGE_UPLOAD_STATE;
    }

    @Override
    public SendMessage getStateTip(Message message) {
        return generateTip(message, generateKeyboard());
    }

    private SendMessage processUsersInput(Message inputMessage) {
        if (inputMessage.hasDocument()) {
            final String fileName = inputMessage.getDocument().getFileName();
            if (FilesService.isFileExtensionValid(fileName,
BotState.HIDE_TEXT_IMAGE_UPLOAD_STATE)) {
                return processUserImage(inputMessage);
            } else {
                logReplyMessage(inputMessage, "reply.wrong-file-extension-error-
message");
                long chatID = inputMessage.getChatId();
                return messageService
                        .getReplyMessage(String.valueOf(chatID), "reply.wrong-
file-extension-error-message");
            }
        } else if (inputMessage.hasPhoto()) {
            logReplyMessage(inputMessage, "reply.upload-photo-as-file-error-
message");
            long chatID = inputMessage.getChatId();
            return messageService
                    .getReplyMessage(String.valueOf(chatID), "reply.upload-photo-
as-file-error-message");
        }
        return checkMessageForRightOption(inputMessage);
    }

    private SendMessage processUserImage(Message inputMessage) {

        long userID = inputMessage.getFrom().getId();
        long chatID = inputMessage.getChatId();
        try {
            if (!FilesService.downloadImage(inputMessage)) {
                logReplyMessage(inputMessage, "reply.photo-size-too-big-error-
message");
                return messageService
                        .getReplyMessage(String.valueOf(chatID), "reply.photo-
size-too-big-error-message");
            }
        } catch (IOException e) {
            //e.printStackTrace();
            logReplyMessage(inputMessage, "reply.photo-downloading-error-
message");
            return messageService
                    .getReplyMessage(String.valueOf(chatID), "reply.photo-
downloading-error-message");
```

```
        }

        userDataCache.setUserCurrentBotState(userID,
BotState.HIDE_TEXT_STRING_UPLOAD_STATE);
        logBotStateChange(inputMessage, BotState.HIDE_TEXT_STRING_UPLOAD_STATE);

        logReplyMessage(inputMessage, "reply.photo-was-uploaded-successfully-
message");
        return messageService
                .getReplyMessage(String.valueOf(chatID), "reply.photo-was-
uploaded-successfully-message");

    }

    private SendMessage generateTip(Message inputMessage, ReplyKeyboardMarkup
replyKeyboardMarkup) {
        long chat_id = inputMessage.getChatId();
        SendMessage replyTip = new SendMessage(String.valueOf(chat_id),
localeMessageService.getMessage("tip.hide-text-image-upload-state"));
        if (replyKeyboardMarkup != null) {
            replyTip.enableMarkdown(true);
            replyTip.setReplyMarkup(replyKeyboardMarkup);
        }
        return replyTip;
    }

    private ReplyKeyboardMarkup generateKeyboard() {
        String backToMainMenu = localeMessageService.getMessage("option.back-to-
main-menu-valid-option");

        return ReplyKeyboardMarkupBuilder.build(backToMainMenu);
    }

}
```

Рисунок В.19 - Код класу HideTextImageUploadHandler

```
package com.komin.steganobot.botapi.handlers;

import com.komin.steganobot.botapi.BotState;
import com.komin.steganobot.botapi.InputMessageHandler;
import com.komin.steganobot.builder.ReplyKeyboardMarkupBuilder;
import com.komin.steganobot.cache.UserDataCache;
import com.komin.steganobot.service.LocaleMessageService;
import com.komin.steganobot.service.ReplyMessageService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;

@Slf4j
@Component
public class HideTextResultUploadHandler extends StateHandler implements
InputMessageHandler {

    public HideTextResultUploadHandler(UserDataCache userDataCache,
ReplyMessageService messageService, LocaleMessageService localeMessageService) {
        super(userDataCache, messageService, localeMessageService);
    }

    @Override
    public SendMessage handle(Message message) {
        return processUsersInput(message);
```

```
    }

    @Override
    public BotState getHandlerName() {
        return BotState.HIDE_TEXT_RESULT_UPLOAD_STATE;
    }

    @Override
    public SendMessage getStateTip(Message message) {
        return generateTip(message, generateKeyboard());
    }

    private SendMessage processUsersInput(Message inputMessage) {
        return checkMessageForRightOption(inputMessage);
    }

    private SendMessage generateTip(Message inputMessage, ReplyKeyboardMarkup
replyKeyboardMarkup) {
        String chatID = inputMessage.getChatId().toString();
        SendMessage replyTip = new SendMessage(chatID,
localeMessageService.getMessage("tip.hide-text-result-upload-state"));
        if (replyKeyboardMarkup != null) {
            replyTip.enableMarkdown(true);
            replyTip.setReplyMarkup(replyKeyboardMarkup);
        }
        return replyTip;
    }

    private ReplyKeyboardMarkup generateKeyboard() {
        String backToMainMenu = localeMessageService.getMessage("option.back-to-
main-menu-valid-option");
        return ReplyKeyboardMarkupBuilder.build(backToMainMenu);
    }
}
```

Рисунок В.20 - Код класу HideTextResultUploadHandler

```
package com.komin.steganobot.botapi.handlers;

import com.komin.steganobot.botapi.BotState;
import com.komin.steganobot.botapi.InputMessageHandler;
import com.komin.steganobot.builder.ReplyKeyboardMarkupBuilder;
import com.komin.steganobot.cache.UserDataCache;
import com.komin.steganobot.service.LocaleMessageService;
import com.komin.steganobot.service.ReplyMessageService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;
import steganoAlgorythms.SpiralKutterJordanBossen;

@Slf4j
@Component
public class HideTextStringUploadHandler extends StateHandler implements
InputMessageHandler {

    public HideTextStringUploadHandler(UserDataCache userDataCache,
ReplyMessageService messageService, LocaleMessageService localeMessageService) {
        super(userDataCache, messageService, localeMessageService);
    }

    @Override
    public SendMessage handle(Message message) {
```

```
        return processUsersInput(message);
    }

    @Override
    public BotState getHandlerName() {
        return BotState.HIDE_TEXT_STRING_UPLOAD_STATE;
    }

    @Override
    public SendMessage getStateTip(Message message) {
        return generateTip(message, generateKeyboard());
    }

    private SendMessage processUsersInput(Message inputMessage) {
        int msgLength = inputMessage.getText().getBytes().length;
        int possibleCharQuantity =
SpiralKutterJordanBossen.evaluatePossibleCharQuantity(inputMessage.getChatId().toS
tring()) - 1;
        if (msgLength > possibleCharQuantity) {
            logReplyMessage(inputMessage, "reply.text-has-too-many-chars-error-
message");
            long chatID = inputMessage.getChatId();
            return messageService
                    .getReplyMessage(String.valueOf(chatID), "reply.text-has-too-
many-chars-error-message");
        }
        return checkMessageForRightOption(inputMessage);
    }

    private SendMessage generateTip(Message inputMessage, ReplyKeyboardMarkup
replyKeyboardMarkup) {
        String chat_id = inputMessage.getChatId().toString();
        SendMessage replyTip = new SendMessage(chat_id,
                localeMessageService.getMessage("tip.hide-text-string-upload-
state")
                        + " " +
SpiralKutterJordanBossen.evaluatePossibleCharQuantity(chat_id));
        if (replyKeyboardMarkup != null) {
            replyTip.enableMarkdown(true);
            replyTip.setReplyMarkup(replyKeyboardMarkup);
        }
        return replyTip;
    }

    private ReplyKeyboardMarkup generateKeyboard() {
        String backToMainMenu = localeMessageService.getMessage("option.back-to-
main-menu-valid-option");

        return ReplyKeyboardMarkupBuilder.build(backToMainMenu);
    }
}
```

Рисунок В.21 - Код класу HideTextStringUploadHandler

```
package com.komin.steganobot.botapi.handlers;

import com.komin.steganobot.botapi.BotState;
import com.komin.steganobot.botapi.InputMessageHandler;
import com.komin.steganobot.cache.UserDataCache;
import com.komin.steganobot.service.LocaleMessageService;
import com.komin.steganobot.service.ReplyMessageService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
```

```
@Slf4j
@Component
public class InitialHandler extends StateHandler implements InputMessageHandler {

    public InitialHandler(UserDataCache userDataCache, ReplyMessageService
messageService, LocaleMessageService localeMessageService) {
        super(userDataCache, messageService, localeMessageService);
    }

    @Override
    public SendMessage handle(Message message) {
        return processUsersInput(message);
    }

    @Override
    public BotState getHandlerName() {
        return BotState.INITIAL_STATE;
    }

    @Override
    public SendMessage getStateTip(Message message) {
        return generateTip(message);
    }

    private SendMessage processUsersInput(Message inputMessage) {
        return checkMessageForRightOption(inputMessage);
    }

    private SendMessage generateTip(Message inputMessage) {
        long chat_id = inputMessage.getChatId();
        return new SendMessage(String.valueOf(chat_id),
localeMessageService.getMessage("tip.initial-state"));
    }
}
```

Рисунок В.22 - Код класу InitialHandler

```
package com.komin.steganobot.botapi.handlers;

import com.komin.steganobot.botapi.BotState;
import com.komin.steganobot.botapi.InputMessageHandler;
import com.komin.steganobot.builder.ReplyKeyboardMarkupBuilder;
import com.komin.steganobot.cache.UserDataCache;
import com.komin.steganobot.service.LocaleMessageService;
import com.komin.steganobot.service.ReplyMessageService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;

@Slf4j
@Component

public class MainMenuHandler extends StateHandler implements InputMessageHandler {

    public MainMenuHandler(UserDataCache userDataCache, ReplyMessageService
messageService, LocaleMessageService localeMessageService) {
        super(userDataCache, messageService, localeMessageService);
    }

    @Override
    public SendMessage handle(Message message) {
```

```
        return processUsersInput(message);
    }

    @Override
    public BotState getHandlerName() {
        return BotState.MAIN_MENU_STATE;
    }

    @Override
    public SendMessage getStateTip(Message message) {
        return generateTip(message, generateKeyboard());
    }

    private SendMessage processUsersInput(Message inputMessage) {
        return checkMessageForRightOption(inputMessage);
    }

    private SendMessage generateTip(Message inputMessage, ReplyKeyboardMarkup
replyKeyboardMarkup) {
        long chat_id = inputMessage.getChatId();
        SendMessage replyTip = new SendMessage(String.valueOf(chat_id),
                localeMessageService.getMessage("tip.main-menu-state"));
        if (replyKeyboardMarkup != null) {
            replyTip.enableMarkdown(true);
            replyTip.setReplyMarkup(replyKeyboardMarkup);
        }
        return replyTip;
    }

    private ReplyKeyboardMarkup generateKeyboard() {
        String hideTextOption = localeMessageService.getMessage("option.main-menu-
state-hide-text-option");
        String unpackTextOption = localeMessageService.getMessage("option.main-
menu-state-unpack-text-option");
        String aboutInfoOption = localeMessageService.getMessage("option.main-
menu-state-about-info-option");

        return ReplyKeyboardMarkupBuilder.build(hideTextOption, unpackTextOption,
aboutInfoOption);
    }
}
```

Рисунок В.23 - Код класу MainMenuHandler

```
package com.komin.steganobot.botapi.handlers;

import com.komin.steganobot.botapi.BotState;
import com.komin.steganobot.botapi.options.MenuOptions;
import com.komin.steganobot.cache.UserDataCache;
import com.komin.steganobot.files_service.FilesService;
import com.komin.steganobot.service.LocaleMessageService;
import com.komin.steganobot.service.ReplyMessageService;
import lombok.extern.slf4j.Slf4j;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
import steganoAlgorythms.SpiralKutterJordanBossen;

import java.util.Objects;
import java.util.Optional;
import java.util.stream.Stream;

@Slf4j
public class StateHandler {

    final UserDataCache userDataCache;
```

```
    final ReplyMessageService messageService;
    final LocaleMessageService localeMessageService;

    public StateHandler(UserDataCache userDataCache, ReplyMessageService
messageService, LocaleMessageService localeMessageService) {
        this.userDataCache = userDataCache;
        this.messageService = messageService;
        this.localeMessageService = localeMessageService;
    }

    SendMessage checkMessageForRightOption(Message inputMessage) {
        String chatID = String.valueOf(inputMessage.getChatId());
        long userID = inputMessage.getFrom().getId();
        BotState currentBotState = userDataCache.getUserCurrentBotState(userID);

        Optional<MenuOptions> stateMenuOptionOptional =
                Stream.of(MenuOptions.values())
                        .filter(option ->
Objects.equals(localeMessageService.getMessage(option.getValue()),
                                inputMessage.getText()))
                        .filter(option ->
option.getOptionAvailableStates().contains(currentBotState))
                        .findFirst();

        if (stateMenuOptionOptional.isEmpty()) {
            if
(userDataCache.getUserCurrentBotState(userID).equals(BotState.INITIAL_STATE)) {
                logReplyMessage(inputMessage, "tip.initial-state");
                return messageService.getReplyMessage(chatID, "tip.initial-
state");
            }

            if
(userDataCache.getUserCurrentBotState(userID).equals(BotState.HIDE_TEXT_IMAGE_UPLO
AD_STATE)
                    ||
userDataCache.getUserCurrentBotState(userID).equals(BotState.UNPACK_TEXT_IMAGE_UPL
OAD_STATE)) {

                logReplyMessage(inputMessage, "tip.hide-text-image-upload-state");
                return messageService.getReplyMessage(chatID, "tip.hide-text-
image-upload-state");
            }

            if
(userDataCache.getUserCurrentBotState(userID).equals(BotState.HIDE_TEXT_STRING_UPL
OAD_STATE)) {
                if (isStringValid(inputMessage.getText(), chatID)) {
                    FilesService.saveUserString(inputMessage);

                    userDataCache.setUserCurrentBotState(userID,
BotState.HIDE_TEXT_RESULT_UPLOAD_STATE);
                    logBotStateChange(inputMessage,
BotState.HIDE_TEXT_RESULT_UPLOAD_STATE);
                    logReplyMessage(inputMessage, "reply.text-was-uploaded-
message");
                    return messageService.getReplyMessage(chatID, "reply.text-was-
uploaded-message");
                } else {
                    logReplyMessage(inputMessage, "reply.invalid-text-error-
message");
                    return messageService
                            .getReplyMessage(chatID, "reply.invalid-text-error-
message");
                }
            }
```

```
            logReplyMessage(inputMessage, "reply.no-such-option-error-message");
            return messageService
                    .getReplyMessage(chatID, "reply.no-such-option-error-
message");
        }

        MenuOptions chosenOption = stateMenuOptionOptional.get();
        userDataCache.setUserCurrentBotState(userID,
chosenOption.getFollowingBotState());
        logBotStateChange(inputMessage, chosenOption.getFollowingBotState());

        return null;
    }

    void logReplyMessage(Message inputMessage, String reply) {
        log.info("[{}] Reply for User: {}, with text: {}",
                inputMessage.getChatId(),
                inputMessage.getFrom().getUserName(),
                reply);
    }

    void logBotStateChange(Message inputMessage, BotState botState) {
        log.info("[{}] BotState for User: {}, was changed to: {}",
                inputMessage.getChatId(),
                inputMessage.getFrom().getUserName(),
                botState);
    }

    private boolean isStringValid(String text, String chatId) {
        if (text == null) {
            return false;
        }
        return text.length() <=
SpiralKutterJordanBossen.evaluatePossibleCharQuantity(chatId);
    }
}
```

Рисунок В.24 - Код класу StateHandler

```
package com.komin.steganobot.botapi.handlers;

import com.komin.steganobot.botapi.BotState;
import com.komin.steganobot.botapi.InputMessageHandler;
import com.komin.steganobot.builder.ReplyKeyboardMarkupBuilder;
import com.komin.steganobot.cache.UserDataCache;
import com.komin.steganobot.files_service.FilesService;
import com.komin.steganobot.service.LocaleMessageService;
import com.komin.steganobot.service.ReplyMessageService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;

import java.io.IOException;

@Slf4j
@Component
public class UnpackTextImageUploadHandler extends StateHandler implements
InputMessageHandler {

    public UnpackTextImageUploadHandler(UserDataCache userDataCache,
ReplyMessageService messageService, LocaleMessageService localeMessageService) {
```

```
        super(userDataCache, messageService, localeMessageService);
    }

    @Override
    public SendMessage handle(Message message) {
        return processUsersInput(message);
    }

    @Override
    public BotState getHandlerName() {
        return BotState.UNPACK_TEXT_IMAGE_UPLOAD_STATE;
    }

    @Override
    public SendMessage getStateTip(Message message) {
        return generateTip(message, generateKeyboard());
    }

    private SendMessage processUsersInput(Message inputMessage) {
        if (inputMessage.hasDocument()) {
            final String fileName = inputMessage.getDocument().getFileName();
            if (FilesService.isFileExtensionValid(fileName,
BotState.UNPACK_TEXT_IMAGE_UPLOAD_STATE)) {
                return processUserImage(inputMessage);
            } else {
                logReplyMessage(inputMessage, "reply.wrong-file-extension-error-
message");
                long chatID = inputMessage.getChatId();
                return messageService
                        .getReplyMessage(String.valueOf(chatID), "reply.wrong-
file-extension-error-message");
            }
        } else if (inputMessage.hasPhoto()) {
            logReplyMessage(inputMessage, "reply.upload-photo-as-file-error-
message");
            long chatID = inputMessage.getChatId();
            return messageService
                    .getReplyMessage(String.valueOf(chatID), "reply.upload-photo-
as-file-error-message");
        }
        return checkMessageForRightOption(inputMessage);
    }

    private SendMessage processUserImage(Message inputMessage) {
        long userID = inputMessage.getFrom().getId();
        long chatID = inputMessage.getChatId();
        try {
            if (!FilesService.downloadImage(inputMessage)) {
                logReplyMessage(inputMessage, "reply.photo-size-too-big-error-
message");
                return messageService
                        .getReplyMessage(String.valueOf(chatID), "reply.photo-
size-too-big-error-message");
            }
        } catch (IOException e) {
            //e.printStackTrace();
            logReplyMessage(inputMessage, "reply.photo-downloading-error-
message");
            return messageService
                    .getReplyMessage(String.valueOf(chatID), "reply.photo-
downloading-error-message");
        }

        userDataCache.setUserCurrentBotState(userID,
BotState.UNPACK_TEXT_RESULT_UPLOAD_STATE);
        logBotStateChange(inputMessage, BotState.UNPACK_TEXT_RESULT_UPLOAD_STATE);
```

```
        logReplyMessage(inputMessage, "reply.photo-was-uploaded-successfully-
message");
        return messageService
                .getReplyMessage(String.valueOf(chatID), "reply.photo-was-
uploaded-successfully-message");

    }

    private SendMessage generateTip(Message inputMessage, ReplyKeyboardMarkup
replyKeyboardMarkup) {
        long chat_id = inputMessage.getChatId();
        SendMessage replyTip = new SendMessage(String.valueOf(chat_id),
localeMessageService.getMessage("tip.hide-text-image-upload-state"));
        if (replyKeyboardMarkup != null) {
            replyTip.enableMarkdown(true);
            replyTip.setReplyMarkup(replyKeyboardMarkup);
        }
        return replyTip;
    }

    private ReplyKeyboardMarkup generateKeyboard() {
        String backToMainMenu = localeMessageService.getMessage("option.back-to-
main-menu-valid-option");

        return ReplyKeyboardMarkupBuilder.build(backToMainMenu);
    }


}
```

Рисунок В.25 - Код класу UnpackTextImageUploadHandler

```
package com.komin.steganobot.botapi.handlers;

import com.komin.steganobot.botapi.BotState;
import com.komin.steganobot.botapi.InputMessageHandler;
import com.komin.steganobot.builder.ReplyKeyboardMarkupBuilder;
import com.komin.steganobot.cache.UserDataCache;
import com.komin.steganobot.service.LocaleMessageService;
import com.komin.steganobot.service.ReplyMessageService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;

@Slf4j
@Component
public class UnpackTextResultUploadHandler extends StateHandler implements
InputMessageHandler {

    public UnpackTextResultUploadHandler(UserDataCache userDataCache,
ReplyMessageService messageService, LocaleMessageService localeMessageService) {
        super(userDataCache, messageService, localeMessageService);
    }

    @Override
    public SendMessage handle(Message message) {
        return processUsersInput(message);
    }

    @Override
    public BotState getHandlerName() {
```

```
        return BotState.UNPACK_TEXT_RESULT_UPLOAD_STATE;
    }

    @Override
    public SendMessage getStateTip(Message message) {
        return generateTip(message, generateKeyboard());
    }

    private SendMessage processUsersInput(Message inputMessage) {
        return checkMessageForRightOption(inputMessage);
    }

    private SendMessage generateTip(Message inputMessage, ReplyKeyboardMarkup
replyKeyboardMarkup) {
        long chat_id = inputMessage.getChatId();
        SendMessage replyTip = new SendMessage(String.valueOf(chat_id),
localeMessageService.getMessage("tip.hide-text-result-upload-state"));
        if (replyKeyboardMarkup != null) {
            replyTip.enableMarkdown(true);
            replyTip.setReplyMarkup(replyKeyboardMarkup);
        }
        return replyTip;
    }

    private ReplyKeyboardMarkup generateKeyboard() {
        String backToMainMenu = localeMessageService.getMessage("option.back-to-
main-menu-valid-option");

        return ReplyKeyboardMarkupBuilder.build(backToMainMenu);
    }
}
```

Рисунок В.26 - Код класу UnpackTextResultUploadHandler

```
reply.no-such-option-error-message = Такої опції не існує! Скористайтесь головним
меню
reply.upload-photo-as-file-error-message = Помилка! Прикріпіть картинку, як файл
(без стиснення)
reply.wrong-file-extension-error-message = Помилка! Такий формат файлу не
підходить
reply.photo-was-uploaded-successfully-message = Чудово! Фото було завантажено
reply.photo-size-too-big-error-message = Помилка! Размір фото перевищує 20 МБ
reply.photo-do-not-contains-text-message = Фото не містить прихованої інформації
reply.photo-downloading-error-message = Помилка! Не вдалось завантажити файл.
Перевірте коректність даних та спробуйте ще раз
reply.text-has-too-many-chars-error-message = Забагато символів!
reply.invalid-text-error-message = Помилка! Перевірте текст та спробуйте ще раз
reply.text-was-uploaded-message = Чудово! Текст був завантажений




option.back-to-main-menu-valid-option = В головне меню
option.initial-state-valid-option = /start
option.main-menu-state-hide-text-option = Приховати текст
option.main-menu-state-unpack-text-option = Вилучити текст
option.main-menu-state-about-info-option = Інформація

tip.initial-state = Надрукуйте "/start", щоб почати роботу
tip.main-menu-state = Скористайтесь головним меню
tip.hide-text-image-upload-state = Прикріпіть картинку формату .PNG розміром не
більше 20МБ, як файл (без стиснення)
tip.hide-text-string-upload-state = Надрукуйте текст, що хочете приховати.
Максимальна кількість байт (Символи латиниці займають 1 байт, символи кирилиці -
2):
```

```
tip.hide-text-result-upload-state = Зображення опрацьовується. Очікуйте
tip.about-info-state = Цей телеграм бот вміє приховувати текст у зображенні, та вилучати прихований текст із зображення.\n\n📌 Бот приймає лише зображення формату .png або .PNG, розмір яких не перевищує 20 МБ.\n\n📌 Щоб приховати текст, скористайтесь опцією головного меню "Приховати текст" і дотримуйтесь інструкцій.\n\n📌 Після того, як отримаєте зображення-результат, скористайтесь опцією головного меню "Вилучити текст", щоб перевірити приховані дані.\n\n📌 Для коректної роботи алгоритму, що приховує дані, краще використовувати фотографії зі складною композицією та великою кількістю деталей.
```

Рисунок В.27 - Код файлу локалізації української мови