НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«РОЗРОБКА TELEGRAM-БОТУ ДЛЯ ЗАХИСТУ ЗОБРАЖЕНЬ ЦИФРОВИМ ВОДЯНИМ ЗНАКОМ»

Виконав: здобувачка 4 курсу
Рівень бакалавр
Галузь знань 12 «Інформаційні технології»,
спеціальність 125 «Кібербезпека»
Бердникова Марія Олександрівна

Керівник: к.т.н., доцент
Ахмаметьєва Ганна Валеріївна

Рецензент: к.т.н., доцент
Кухаренко Сергій Вікторович

Одеса – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
Факультет **кібербезпеки та інформаційних технологій**
Кафедра **кібербезпеки**
Галузь знань: **12 Інформаційні технології**
Спеціальність: **125 Кібербезпека**
Назва освітньої програми: **Кібербезпека**

ЗАТВЕРДЖУЮ
Завідувач кафедри кібербезпеки
професор С.А. Горбаченко
_____ «____ »_____ 20__ року

З А В Д А Н Н Я
на кваліфікаційну (бакалаврську) роботу
здобувачці Бердниковій Марії Олександрівні

1. Тема роботи: «Розробка Telegram-боту для захисту зображень цифровим водяним знаком»
Керівник роботи: к.т.н, доцент Ахмаметьєва Ганна Валеріївна
затверджені наказом НУ ОЮА від «18»березня 2025 року № 548-6
2. Строк подання здобувачем роботи «19» травня 2025 року
3. Дата видачі завдання «16» вересня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1			
2			
3			
4			
5			
6			
7			
8			

Здобувач _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

\

АНОТАЦІЯ

Кваліфікаційна робота на тему «Розробка telegram-боту для захисту зображень цифровим водяним знаком» на здобуття першого (бакалаврського) рівня вищої освіти за спеціальністю 125 Кібербезпека, освітньою програмою: Кібербезпека.

Містить 18 рисунків, 2 таблиці, 21 літературне джерело за переліком посилань. Робота виконана на 50 сторінках загального тексту і 38 сторінках основного тексту.

Метою роботи є розробка Телеграм-боту для захисту зображень цифровим водяним знаком, а також аналіз методів вбудовування ЦВЗ та засад їх експлуатації.

Об’єктом розробки є Телеграм-бот для впровадження цифрового водяного знаку в просторову область зображення. Під час аналізу предметної області буде проаналізовано основні сучасні методи впровадження цифрових водяних знаків в просторову та частотну області зображення, види атак на ЦВЗ та характеристики стійкості стеганосистем.

Можливий напрям подальших досліджень включає удосконалення функціоналу боту та впровадження нових методів. Продукт має високий рівень реалізації та може бути використаний для подальшої роботи.

ЦИФРОВИЙ ВОДЯНИЙ ЗНАК, TELEGRAM-БОТ, СТЕГАНОГРАФІЯ, ПРОСТОРОВА ОБЛАСТЬ, ЧАСТОТНА ОБЛАСТЬ, СТІЙКІСТЬ

.

ABSTRACT

The qualification work titled "Development of a Telegram Bot for Image Protection Using a Digital Watermark" has been completed to obtain the first (bachelor’s) level of higher education in the specialty 125 "Cybersecurity", educational program: Cybersecurity.

The work contains 18 figures, 2 tables, and 21 literary sources listed in the references. The total length of the work is 50 pages, including 38 pages of main text.

The purpose of this study is to develop a Telegram bot for embedding digital watermarks into images as a means of protection, as well as to analyze modern watermarking techniques and the fundamentals of their application.

The object of development is a Telegram bot designed for embedding digital watermarks into the spatial domain of an image. During the analysis of the subject area, current methods of watermarking in both the spatial and frequency domains, types of attacks on watermarks, and the resistance characteristics of steganographic systems are explored.

A potential direction for further research includes expanding the bot’s functionality and integrating advanced watermarking algorithms. The developed product demonstrates a high level of implementation and may serve as a foundation for future projects.

DIGITAL WATERMARK, TELEGRAM BOT, STEGANOGRAPHY, SPATIAL DOMAIN, FREQUENCY DOMAIN, ROBUSTNESS

ЗМІСТ

ВСТУП

На сьогоднішній день, у зв’язку з тенденцією на цифровізацію більшості сфер діяльності людини, проблема захисту авторських прав на різні типи цифрового контенту має надзвичайно важливе значення. Спеціалісти з зрізних направлень публікують свої наукові статті з унікальними висновками, дослідженнями тощо. Музичні виконавці, художні митці використовують Інтернет для розповсюдження своїх робіт. Без надійного захисту уся ця інформація втрачає свою автентичність та стає абсолютно незахищеною від ймовірності привласнення, спотворення та несанкціонованого використання іншими користувачами. Традиційні криптографічні методи, що використовуються для обмеження доступу до вмісту через його шифрування, мають низку обмежень і не здатні гарантувати повноцінний захист авторських прав. У зв’язку з цим виникає потреба у використанні додаткових технологій, які б забезпечували не лише захист, а й можливість ідентифікації, простежування розповсюдження та перевірки достовірності цифрового контенту. Одним із ефективних інструментів для підтвердження авторства та забезпечення цілісності цифрового контенту є цифрові водяні знаки (ЦВЗ) – технологія, що дозволяє приховано вбудовувати службову інформацію без суттєвого впливу на візуальні характеристики контенту.

У контексті зростання популярності месенджера Telegram, який використовується як засіб комунікації, обміну файлами та створення сервісів через ботів, актуальним є завдання інтеграції механізмів цифрового захисту безпосередньо в його середовище. Розробка Telegram-боту, що забезпечує автоматизоване вбудовування цифрових водяних знаків у зображення, надає можливість не лише захистити візуальний контент, але й створити зручний інструмент для користувачів без потреби у складному програмному забезпеченні.

Метою кваліфікаційної роботи є розробка Телеграм-боту для захисту зображень цифровим водяним знаком.

Задачами кваліфікаційної роботи є:

1) аналіз методів вбудовування ЦВЗ та засад їх експлуатації;

2) розробка практичної реалізації алгоритму ЦВЗ;

3) впровадження розробленого алгоритму в середовище Telegram.

Об’єктом дослідження є процеси захисту мультимедійної інформації за допомогою цифрових водяних знаків.

Предметом дослідження є технології та методи вбудовування цифрових водяних знаків у зображення з метою їх захисту від несанкціонованого використання, а також програмна реалізація цих методів у вигляді Telegram-боту.

Практична значність кваліфікаційної роботи полягає у створенні функціонального Telegram-боту, який реалізує механізм захисту цифрових зображень шляхом вбудовування цифрового водяного знака. Запропоноване рішення не потребує встановлення спеціалізованого програмного забезпечення та є доступним для широкого кола користувачів завдяки інтеграції в популярний месенджер. Розроблений бот дозволяє автоматизувати процес маркування зображень та може бути застосований: для захисту авторських прав у соціальних мережах.

1 ОСНОВНІ ТЕОРЕТИЧНІ АСПЕКТИ ВПРОВАДЖЕННЯ ЦИФРОВИХ ВОДЯНИХ ЗНАКІВ ДЛЯ ЗАХИСТУ ЗОБРАЖЕНЬ

1.1 Поняття цифрового водяного знаку

Впровадження цифрового водяного знаку – це метод мережевої стеганографії, який базується на вбудовуванні інформації безпосередньо у цифровий медіаконтент таким чином, що ця інформація стає невід'ємною частиною даних та виступає предметом ідентифікаційної мітки унікальності контенту.

Сфери застосування цифрових водяних знаків:

- моніторинг ефіру: відстеження трансляції відеоматеріалів для контролю ефірного часу;
- ідентифікація власника: впровадження інформації про авторські права у цифровий контент, що дозволяє ідентифікувати автора та полегшує виявлення порушень;
- відстеження транзакцій: запис історії передачі копій цифрового контенту з унікальним водяним знаком для кожного одержувача;
- контроль копіювання: захист цифрового контенту від незаконного плагіату шляхом вбудовування інформації, яка блокує запис контенту на сумісних пристроях, якщо копія є нелегальною [1].

За методом нанесення інформації на об'єкт цифрові водяні знаки класифікують на дві основні групи: методи просторового домену та методи частотного домену. Частотні методи є більш стійкими до крипто-атак , оскільки при використанні просторового методу ЦВЗ вбудовується в первинну область зображення. Результат такого підходу є вразливим до атак на основі спотворень. Також існують гібридні методи, такі методи поєднують в собі два підходи нанесення для досягнення балансу властивостей.

За просторовим методом ЦВЗ накладається безпосередньо на вихідні дані – наприклад, змінюючи значення пікселів зображення або кольорових компонентів. Алгоритми працюючі за даним методом характеризуються простотою реалізації

та низькою обчислювальною складністю. Основною вразливістю таких алгоритмів є відносно низька стійкість до впливу обробки сигналу.

У частотному домені першочергово здійснюється перетворення носія, після чого ЦВЗ впроваджується шляхом модифікації обраних коефіцієнтів цього перетворення. Нанесення ЦВЗ в трансформовані коефіцієнти підвищує рівень стійкості алгоритму до спотворень. Доволі розповсюдженим є вбудовування ЦВЗ у деталізуючі компоненти DCT (дискретне косинусне перетворення) або DWT (дискретне вейвлет-перетворення) для збереження балансу характеристик, оскільки зміни в низькочастотних складових сильно впливають на якість, а високочастотні компоненти легко знищуються стисненням.

Також ЦВЗ можна класифікувати за їх характеристиками з точки зору сприйняття та стійкості. Розрізняють видимі й невидимі водяні знаки, серед невидимих – робастні, крихкі, а також зворотні відповідно до їх призначення в системі захисту. Видимий водяний знак являє собою помітне зображення накладене на контент. Недоліком таких ЦВЗ є те, що видимі мітки можуть погіршувати візуал контенту і їх можна частково видалити. Невидимі мітки використовуються, коли потрібен захист без візуального втручання – наприклад, для прихованого підтвердження авторства, моніторингу розповсюдження або для перевірки цілісності контенту. Невидимий ЦВЗ можна виявити лише з використанням спеціальних алгоритмів. За рівнем стійкості до модифікацій розрізняють робастні та крихкі типи водяних знаків.

Робастні ЦВЗ мають характеристики які дозволяють витримувати типові спотворення чи обробки сигналу – такі як стиснення, фільтрацію, зміни розміру та форматів. Такий ЦВЗ напряму зв’язаний з носієм, тому його важко видалити без помітного погіршення якості контенту.

Крихкий водяний знак є дуже чутливим до будь-яких змін у носії, найменше редагування або перекодування контенту може призвести до руйнації такого ЦВЗ. Такі ЦВЗ зазвичай використовуються з метою контролю цілісності.

Окрему категорію становлять зворотні водяні знаки, які ще називають інвертованими або відновлюваними.

Після вилучення такого ЦВЗ можна повністю відновити оригінальний немаркований контент без втрати якості. Зворотні водяні знаки можуть комбінуватися з робастними або крихкими підходами відповідно до конкретних вимог щодо захисту інформації. Наприклад, у медичних зображеннях застосування зворотних крихких водяних знаків забезпечує ефективний контроль цілісності даних, водночас гарантуючи можливість повного відновлення оригінального контенту після процедури автентифікації. Такий підхід є особливо актуальним у випадках, коли навіть мінімальні втручання у первинні дані є недопустимими [2].

1.2 Метод заміни найменш значущого біта LSB

Метод LSB є найпопулярнішим методом заміни в просторовій множині зображення. Найменший значущий зберігає в собі найменше інформації. Людина не здатна помітити зміни у цьому біті. В основі методи лежить те, що такі біти пікселей замінюються на біти секретного повідомлення.

LSB-метод працює на рівні окремих бітів цифрового представлення даних. Кожен піксель цифрового зображення можна подати у вигляді послідовності з 8 біт: $b7b6b5b4b3b2b1b0$. Тут $b7$ є найбільш значущим бітом (MSB), а $b0$ – найменш значущим бітом (LSB). Зміна старших бітів (MSB) суттєво впливає на колір або яскравість пікселя, тому навіть незначна їх модифікація може помітно спотворити зображення. Натомість маніпуляція лише найменш значущим бітом $b0$ має мінімальний вплив: підміна LSB практично не викликає видимих змін візуального вигляду зображення [3].

Розглянемо піксель з RGB-значеннями:

- R: 10010101
- G: 00001101
- B: 11001001

Наприклад, необхідно приховати символ ‘A’, ASCII-код якого 65, що в двійковому вигляді дорівнює 01000001. Замінимо найменш значущі біти кожного кольорового каналу на біти цього символу:

- R: 10010100
- G: 00001100
- B: 11001000

Такі зміни не впливають на загальне сприйняття зображення, але дозволяють приховати інформацію, приклад на рисунок 1.1.

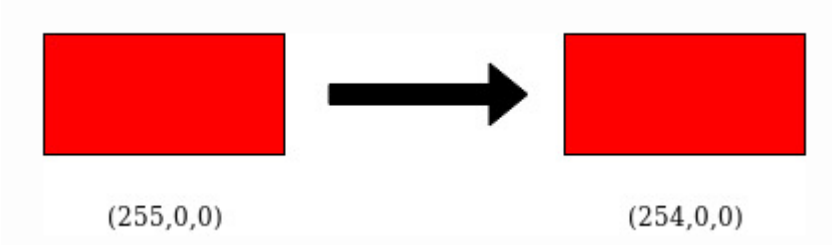

Рисунок 1.1 – Вплив модифікації найменшого значущого біта на колір

Переваги використання такого методу:

- такий алгоритм надзвичайно простий у виконанні: він вимагає лише базових операцій з бітами і не потребує складних обчислень;
- метод дозволяє вбудувати відносно великий обсяг інформації в носій. На кожен піксель зображення можна сховати 1 біт даних, що для звичайного зображення означає тисячі чи навіть мільйони біт;
- непомітність для людського ока – заміна лише найменш значущих бітів пікселів не викликає видимих спотворень зображення;
- метод LSB не збільшує розмір файлу-контейнера.

Недоліки у використанні такого методу:

- вразливість до статистичних атак;
- вразливість до будь-якого виду модифікації;
- легко видаляється;
- відсутність криптографічного захисту [4].

1.3 Метод на основі дискретного косинусного перетворення

Дискретне косинусне перетворення (DCT) – це математичне перетворення, що представляє сигнал (наприклад, зображення) як суму функцій різної частоти. Основна ідея полягає в розділенні зображення на частоти: низькочастотні компоненти містять основну інформацію, високочастотні – деталі та шум. Для цифрових зображень зазвичай використовується двовимірне DCT (2D-DCT), яке застосовується до блоків 8×8 пікселів.

Нехай, f (x, y), матриця зображення в яке необхідно вбудувати водяний знак, ділиться на блоки розміром 8x8. Кожен такий блок піддається дискретному частотному перетворенню, результатом якого буде матриця частотних коефіцієнтів F(u,v).

Пряме DCT – перетворення для зображення розміром M×N, приклад формула 1.1:

$$f(u,v) = c(u)c(v) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} F(x,y) \cos\left[\frac{\pi(2x+1)u}{2M}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right], \quad (1.1)$$

де F(x,y) – значення пікселя у просторі (просторова координата),

f(u,v) – відповідний частотний коефіцієнт (результат перетворення),

c(u),c(v) – нормалізаційні коефіцієнти.

На наступному етапі застосовується спеціальний алгоритм для вибору частотних коефіцієнтів, які будуть модифіковані з метою вбудовування водяного знаку. Відібрані коефіцієнти змінюються згідно з одним із двох базових принципів, що дозволяє сформувати нову матрицю коефіцієнтів частотної області.

Принцип додавання формула 1.2:

$$F' = F + a \cdot W. \quad (1.2)$$

Принцип множення формула 1.3:

$$F' = F \cdot (1 + a \cdot W), \quad (1.3)$$

де F – частотний коефіцієнт до модифікації,

F' – частотний коефіцієнт після вбудовування водяного знаку,

W – інформація водяного знаку ,

a – коефіцієнт сили вбудовування, що визначає інтенсивність впливу водяного знаку на коефіцієнт.

Після модифікації матриці частотних коефіцієнтів F(u,v) до неї застосовується зворотне дискретне косинусне перетворення (IDCT), що дозволяє відновити блок зображення розміром 8×8, який тепер містить вбудований водяний знак. Отриманий блок замінює відповідний блок оригінального зображення, формуючи нову матрицю зображення з вбудованим водяним знаком m(x,y) [5].

Переваги у використанні методу DCT:

- висока стійкість до стиснення з втратами;
- локалізація модифікації завдяки роботі з блоками зображення;
- метод DCT є фундаментальною частиною стандартів стиснення JPEG та MPEG, що спрощує інтеграцію цифрового водяного знаку без порушення існуючих форматів чи процесів обробки зображень і відео;
- модифікація коефіцієнтів у частотній області мінімально впливає на візуальну якість зображення.

Недоліки у використанні методу DCT:

- чутливість до геометричних трансформацій;
- обмежений обсяг вбудованих даних;
- складність реалізації [6].

1.4 Метод на основі дискретного вейвлет-перетворення

Дискретне вейвлет-перетворення (DWT) – це математичне перетворення, яке дозволяє представити цифровий сигнал (зображення) як сукупність компонент на різних масштабах або рівнях роздільної здатності. На відміну від DCT, яке використовує глобальні косинусні функції, DWT базується на локалізованих функціях-вейвлетах, що дозволяє одночасно аналізувати частотну і просторову інформацію.

Спочатку сигнал фільтрується на дві частини: високочастотна містить здебільшого різкі зміни, а низькочастотна – плавні коливання. Далі низькочастотну складову повторно поділяють на нові компоненти. Цей процес можна проводити стільки разів, скільки вимагає конкретна задача. При цьому всі отримані коефіцієнти DWT зберігають інформацію, необхідну для точного відновлення початкового сигналу. Для проведення вейвлет-аналізу можна використовувати різні основи вейвлетів, наприклад: вейвлети Хаара, Добеші, симлети, кофлети та ортогональні вейвлети. Ці методи відрізняються між собою значеннями коефіцієнтів та підходами до розкладання на спектри [7].

В основі методу лежить розкладання зображення на частини різних просторових та частотних областей, схема-приклад перетворення на рисунку 1.2.

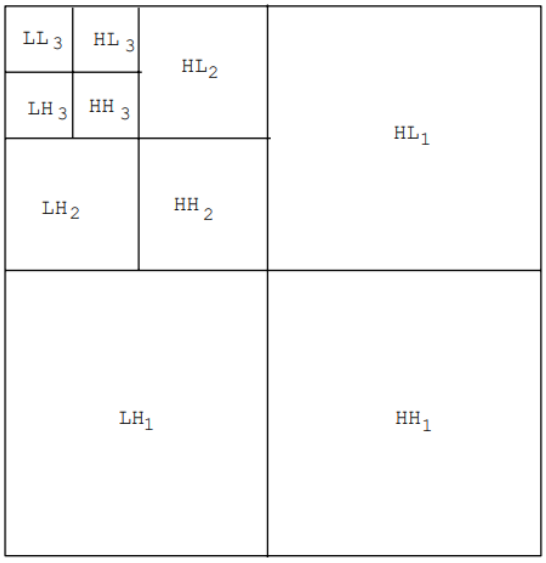

Рисунок 1.2 – Схема вейвлет-перетворення

Після виконання такого перетворення, необхідно проаналізувати інформацію в чотирьох частотних областях: LL – низькочастотній, LH, HL, HH – високочастотними. Чверть коефіцієнтів двовимірного вейвлет-перетворення отримується шляхом послідовного застосування високочастотного фільтра (H): спочатку до рядків, а потім до стовпців оброблюваних даних. Ця група коефіцієнтів формується в правому нижньому кутку схеми та позначається як HH1. Інша чверть коефіцієнтів виникає після того, як інформація проходить спочатку через високочастотний фільтр для рядків, а потім через низькочастотний фільтр для стовпців. Цей блок позначають як LH1 і він розміщується у правому верхньому кутку схеми. Блок коефіцієнтів HL1 у лівому нижньому кутку є результатом застосування низькочастотного фільтра до рядків, а потім високочастотного фільтра до стовпців. Верхній лівий кут схеми складається з менших блоків, які відображають послідовне вейвлет-перетворення з подальшим зменшенням кількості коефіцієнтів. Цей процес триває до отримання одного коефіцієнта в лівому верхньому кутку, до якого застосовується виключно низькочастотний фільтр [8].

Коефіцієнти, що містять високочастотні компоненти, називаються деталізуючими, тоді як коефіцієнти, отримані лише з використанням низькочастотного фільтра (LL) – апроксимуючими.

Для створення ЦВЗ за методом з використанням ДВП-ДКП область (LH, HL, HH) необхідно поділити на блоки розміром 8x8, до кожного такого блоку застосовується ДКП. Формуються дві незалежні псевдовипадкові послідовності: одна призначена для вбудовування біту «0», інша – для вбудовування біту «1». Довжина кожної послідовності дорівнює числу дискретних вейвлет‑коефіцієнтів у блоці. До значень коефіцієнтів ДКП додаються біти псевдовипадкових послідовностей, попередньо помножені на заданий коефіцієнт посилення.

Для використанням методу на основі зміни коефіцієнтів вейвлет-перетворення необхідно так само поділити область (LH, HL, HH) на блоки 8x8. Біти цифрового водяного знаку, підсилені певним коефіцієнтом, формуються на основі усереднених значень вейвлет-коефіцієнтів у відповідному блоці.

Порівняємо результати виконання перетворень першого, другого та третього порядку, приклад наведено на рисунку 1.3.

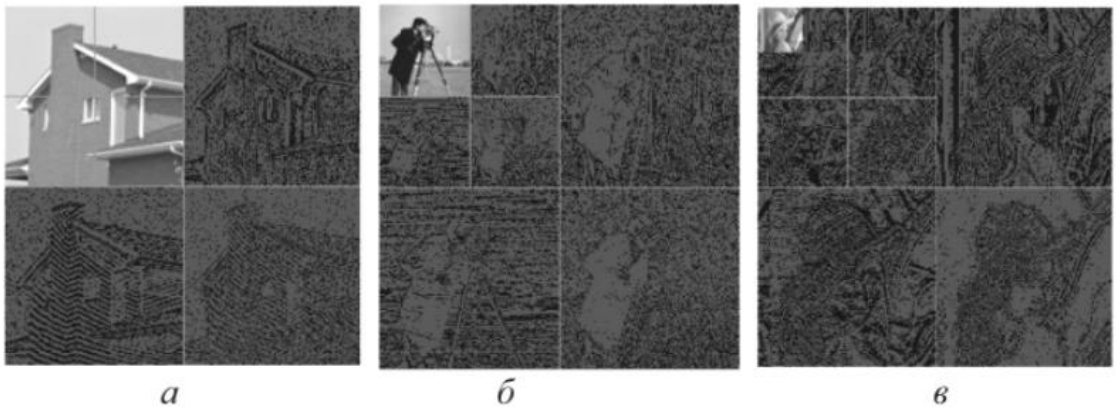

Рисунок 1.3 – Результат застосування вейвлет-перетворень для зображень

Оцінку рівня спотвореності прийнято проводити за трьома показниками: співвідношення сигнал/шум (SNR), якість зображення, нормалізована середня абсолютна різниця NAD: При збільшенні розміру повідомлення параметри погіршуються, адже при збільшенні параметри мають такі характеристики: якість зображення та співвідношення сигналу/шуму зменшуються, а NAD збільшується. Метод ДВП-ДКП дає більші стабільні характеристики якості зображення для трьох вейвлетів, тоді як метод зміни коефіцієнтів погіршує якість зображення [9].

1.5 Оцінка стійкості цифрових водяних знаків

Стеганосистема ЦВЗ має бути побудована таким чином, щоб зменшити вірогідність виникнення помилок, саме тому велике значення має стійкість ЦВЗ. Схема оцінки стійкості ЦВЗ до зовнішнього втручання зображена на рисунку 1.4.

Рисунок 1.4 – Схема оцінки стійкості ЦВЗ

Основні етапи оцінки стійкості ЦВЗ від зовнішнього втручання.

Вбудовування цифрового водяного знаку (ЦВЗ) потребує дотримання однакових початкових умов. Насамперед, ця вимога стосується стеганоконтейнера. Сам водяний знак може мати довільну структуру, при цьому його ідентичність у межах різних алгоритмів стеганографії може не зберігатися. Якщо ЦВЗ формується випадковим чином, це, як правило, забезпечує кращу об’єктивність результатів.

Зовнішній вплив на стеганоконтейнер із вбудованим ЦВЗ може бути будь-яким за характером і охоплювати весь контейнер повністю. Для коректного порівняльного аналізу важливо забезпечити однаковий рівень або діапазон інтенсивності такого впливу.

Витягування ЦВЗ здійснюється згідно з методом його вбудовування. Обсяг вилученого водяного знаку має повністю відповідати обсягу вбудованої інформації. При цьому заборонено використовувати будь-які додаткові засоби для відновлення ЦВЗ, навіть якщо такі засоби передбачені алгоритмом стеганографії.

Стійкість ЦВЗ можна оцінити з використанням різних методів, для прикладу розглянемо метод коефіцієнту помилкових бітів, що застосовується під час оцінки змін бітової послідовності за формулою 1.4:

$$BER(S, S^n) = \frac{\sum p_i}{N}, \tag{1.4}$$

де N – загальна кількість бітів,

Рівень викривлення визначає здатність цифрового водяного знаку, вбудованого в стеганоконтейнер, протистояти різноманітним атакам. Ця здатність залежить від таких чинників, як обраний алгоритм вбудовування, значення коефіцієнта сили вбудовування P, а також характер зовнішнього впливу та інші умови [10].

1.6 Види атак на ЦВЗ

ЦВЗ можуть піддаватися різноманітним атакам, мета яких – знищити, змінити або вилучити приховану інформацію. З точки зору динаміки дій зловмисника, атаки можна класифікувати на навмисні і ненавмисні, що виникають у процесі обробки зображень, на рисунку 1.5 детально класифіковано види атак на ЦВЗ для зображень.

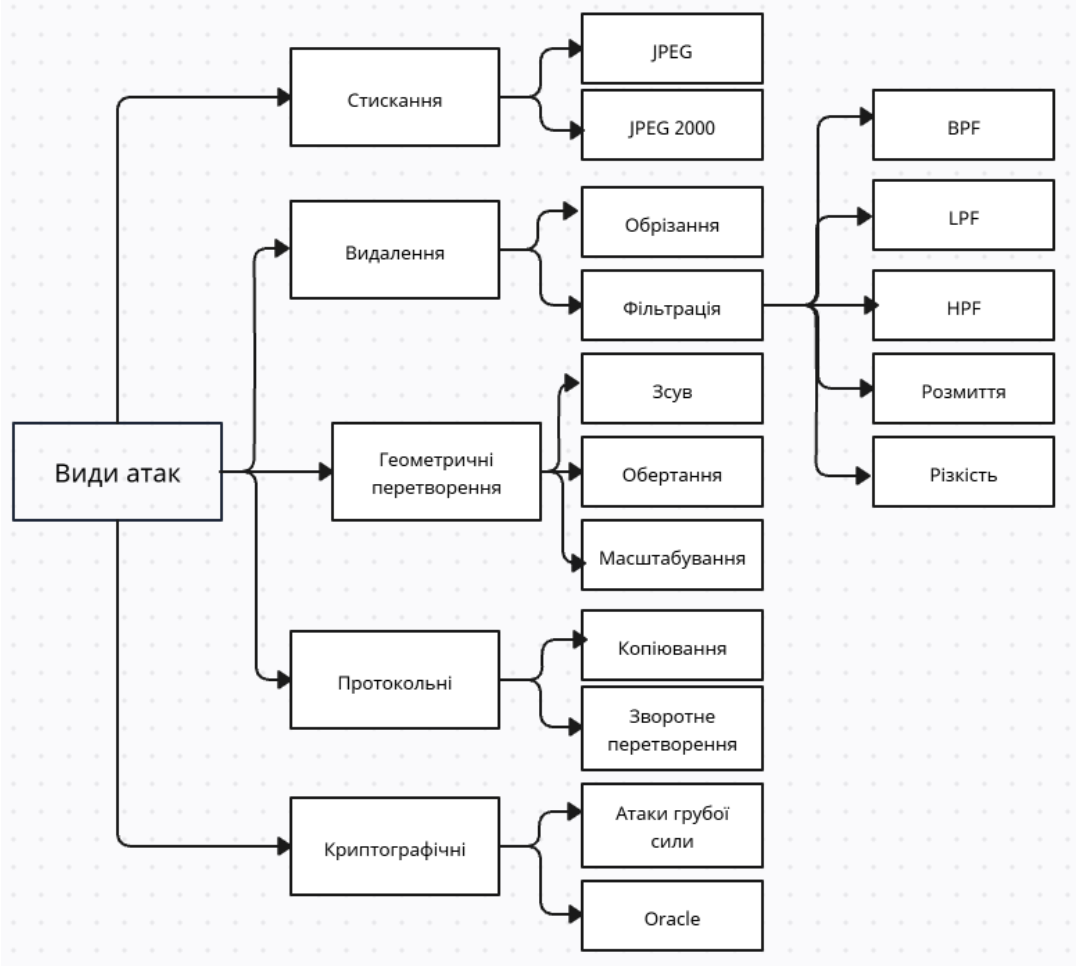

Рисунок 1.5 – Види атак на ЦВЗ

Атаки на видалення (Removal attacks) – це методи атак, метою яких є усунення водяного знака із зображення без спроби зламу алгоритму захисту чи розкриття методу вбудовування. Такі атаки не використовуються з метою визначення методів шифрування або алгоритмів, які були використані при вбудовуванні ЦВЗ. У результаті таких дій зображення з водяним знаком зазнає спотворень, що призводить до втрати сигналу водяного знаку, в результаті чого ЦВЗ не може бути відновлений жодними методами після обробки. До цієї категорії атак належать: додавання шуму, вирівнювання гістограми, розмиття та посилення різкості.

Геометричні атаки відносяться до типу впливів на цифрові водяні знаки, метою яких є не пряме знищення, а деформація просторового розташування знаку, що ускладнює його коректне виявлення. Такі атаки не обов’язково призводять до повної втрати інформації, а лише змінюють її положення або форму. Цей тип атак може вважатися прийнятним у рамках тестування систем захисту, за умови наявності опису параметрів впливу та застосування відповідних механізмів компенсації. Як правило, відповіддю на геометричні атаки є синхронізаційні алгоритми, спрямовані на відновлення початкових координат водяного знака. Проте впровадження таких алгоритмів часто потребує значних обчислювальних ресурсів. До поширених прикладів геометричних атак належать обертання, масштабування, зсув та нахил зображення.

Криптографічні атаки, у свою чергу, спрямовані на компрометацію криптографічних механізмів, що використовуються при вбудовуванні або витягуванні ЦВЗ. Метою таких атак є або повне вилучення вбудованої інформації, або її підміна. Серед типових прикладів можна виокремити атаку повним перебором, яка полягає у систематичному пошуку ключа або параметрів системи, що захищають водяний знак.

Протокольні атаки становлять окрему категорію загроз, що спрямовані не на технічне руйнування або витягування цифрового водяного знака, а на маніпуляцію правовими аспектами володіння інформацією. На відміну від атак,

що спотворюють або знищують водяний знак, протокольні атаки передбачають вбудовування зловмисником власного водяного знака у вже захищені дані. Це призводить до конфліктів автентичності та унеможливлює однозначне встановлення справжнього власника, тим самим підриваючи довіру до самої технології цифрового маркування як засобу захисту авторських прав.

Одним із поширених різновидів цього типу є атака дублювання (duplicate attack). Вона полягає у витягуванні водяного знака з одного об'єкта та подальшому вбудовуванні його в інший, що дозволяє зловмиснику підробити належність нового об’єкта. При цьому знак модифікується з урахуванням властивостей цільового контенту, зберігаючи непомітність і формуючи хибне враження щодо автентичності даних.

Атаки на основі стиснення виникають під час поширення зображень або відео у відкритому доступі. Користувачі можуть здійснювати стиснення або перекодування даних з метою зменшення їхнього обсягу. Застосовуються як безвтратні алгоритми (наприклад, LZW), так і методи з втратами, зокрема JPEG, JPEG 2000 та MPEG. Такі перетворення можуть значно вплинути на структурні характеристики цифрового об’єкта, що, у свою чергу, ставить під загрозу ЦВЗ. Ефективна система цифрового маркування має забезпечувати достатній рівень стійкості до типових процедур стиснення, аби гарантувати можливість витягування знака після таких дій [11].

2 АНАЛІЗ ОСНОВНИХ ЗАСАД РОЗРОБКИ ТЕХНІЧНО-ОРГАНІЗАЦІЙНОГО ПЛАНУ ПРОЕКТУ

2.1 Постановка задачі

В результаті проведення аналізу предметної області та оцінки актуальності даної теми маємо наступні вимоги до продукту:

- програма реалізує вбудову цифрового водяного знаку в просторову область зображення;
- реалізація ініціалізації та можливості запуску боту;
- бот повинен приймати на обробку фото;
- у разі неможливості обробки фото або іншої проблеми на цьому етапі повідомляти про це користувача;
- надати можливість обрати тип водяного знаку у вигляді розгалуження фото/текст;
- для текстового водяного знаку надати користувачу можливість обрати колір тексту, розмір шрифту, видимість на об’єкті, положення на об’єкті;
- для водяного знаку у вигляді фото надати користувачу можливість обрати розмір, положення, видимість;
- надати результат у вигляді фото з водяним знаком за встановленими користувачем характеристиками [12].

В основі методу лежить впровадження видимого цифрового водяного знаку у просторову область зображення, таким чином, щоб ЦВЗ було важко видалити. Для цього будемо використовувати характеристики видимості, розміру та положення ЦВЗ. Впровадження відбувається за допомогою методів програмування без використання окремих математичних формул. Процедура здійснюється на основі властивостей пікселей та позиціонування.

2.2 Структура інтерфейсу

Першочергова ціль створення інтерфейсу для користувача заключається у тому, що він має бути зручним у використанні та надавати широкий функціонал. Саме цим принципом будемо керуватися під час розробки структури загального інтерфейсу.

Алгоритм роботи чат-боту для створення водяних знаків базується на базових алгоритмах роботи будь-якого телеграм-боту. Користувач надсилає команду у вигляді повідомлення /start, починається нова сесія, чат-бот знаходиться у стані в якому він готовий приймати дані. У нашому випадку він приймає лише фото, приклад показано на рисунку 2.1.

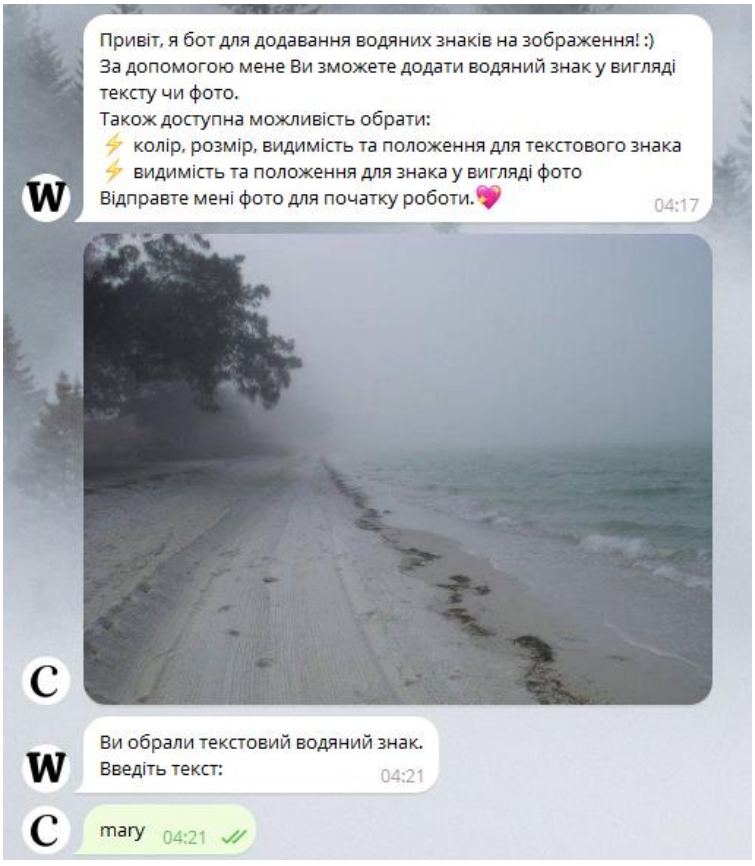

Рисунок 2.1 – Приклад інтерфейсу після відправки зображення

Якщо користувач надсилає текстове повідомлення на такий запит, бот відповідає повідомленням “Надішліть будь ласка фото.”, приклад на рисунку 2.2.

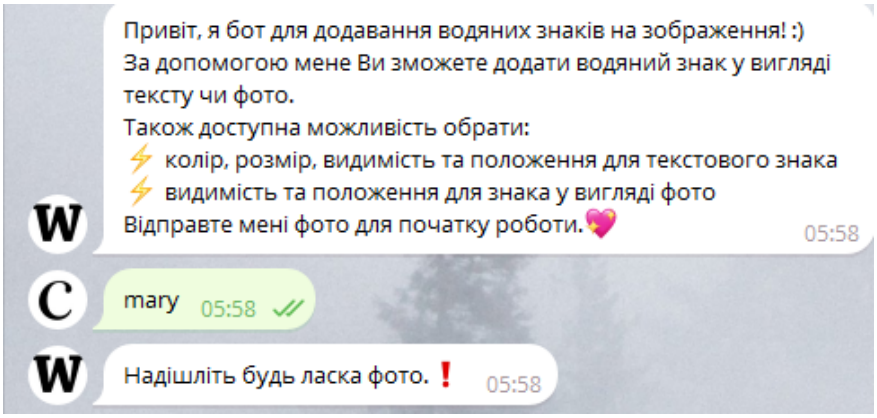

Рисунок 2.2 – Приклад інтерфейсу після відправки тексту на початковий запит

Подальший інтерфейс боту створений з використанням принципу розгалуження. Після того, як користувач надсилає фото, бот надсилає йому повідомлення в якому необхідно обрати тип водяного знаку. Розгалуження використовується також для оптимізації процесу обрання користувачем кольору та місця розташування текстового водяного знаку, місця розташування водяного знаку у вигляді фото, приклад на рисунку 2.3. Це виконано з використанням класу InlineKeyboardButton, InlineKeyboardMarkup бібліотеки python-telegram-bot. Детальний опис програмної складової функціоналу виконано в третьому розділі [13].

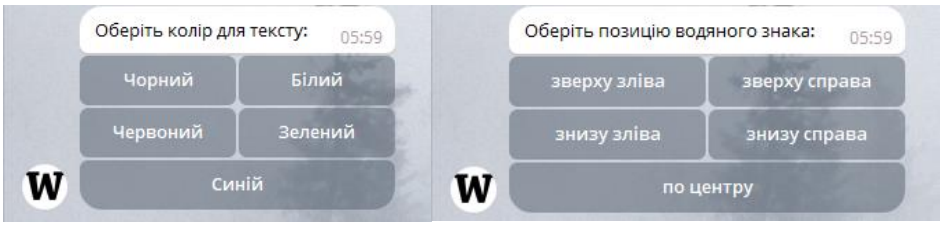

Рисунок 2.3 – Приклад розгалуження з використанням кнопок

Розмір шрифту тексту, фото та значення видимості задається користувачем в цифровому форматі, що є недоліком, адже в такому форматі дуже важко зорієнтуватися. Після виконання всіх процедур користувач отримує результат з підписом про всі обрані ним параметри та автоматичним запрошенням на повторний запит, приклад на рисунку 2.4.

Рисунок 2.4 – Приклад отриманого результату

Це дозволяє внести зміни під час повторного користування, маючи наочне подання характеристик, які вже було застосовано.

2.3 Аналіз засобів розробки програмного забезпечення

Для взаємодії з сервером Telegram було використано BotFather – офіційний бот Telegram, через який здійснюється створення та керування іншими ботами. Через BotFather було зареєстровано нового бота, задано йому ім'я та отримано токен доступу до API Telegram. За допомогою токена була здійснена автентифікація коду.

За основну мову програмування було обрано Python. За основне середовище розробки було обрано програму PyCharm. Це програмне забезпечення має велику кількість переваг серед програм такого типу. PyCharm – це сучасне середовище розробки від компанії JetBrains, зручний функціонал якого максимально полегшує процес написання та відлагодження коду. Програма має інтуїтивно зрозумілий, чистий інтерфейс, що дозволяє швидко орієнтуватися у великих проектах, завдяки потужним засобам автодоповнення та аналізу коду. PyCharm забезпечує інтеграцію з різноманітними фреймворками та технологіями. IDE відмінно підтримує популярні Python‑фреймворки, серед яких Django, Flask, FastAPI та інші, що робить її зручним інструментом для веб‑розробників та спеціалістів, які працюють з REST API або мікросервісною архітектурою. Крім того, PyCharm

адаптована для роботи з великими даними та застосування машинного навчання, завдяки чому вона є універсальним рішенням для широкого спектру проєктів. Однією з ключових переваг PyCharm є її можливість безшовно інтегруватися з провідними системами контролю версій, базами даних, інструментами для тестування, CI/CD-системами та хмарними платформами, що дає можливість виконувати всі необхідні операції безпосередньо з середовища розробки [14].

Для роботи безпосередньо з Telegram Bot API було використано бібліотеку python-telegram-bot. Це бібліотека Python, яка була створена розробниками Telegram. Можна виділити наступні переваги даної бібліотеки:

- відмінно структурована документація;
- з точки зору асинхронного програмування є більш зручною у використанні ніж aiogram;
- можливість використання inline та reply-клавіатури дозволяє виконувати обробку вибору користувача в реальному часі;
- розгорнутий функціонал для роботи з медіа, аудіо, та фото;

Також, під час розробки боту використовувалась бібліотека Pillow, це спеціальна бібліотека Python націлена на обробку зображень. Ця бібліотека широко використовується для: масштабування, візуального редагування, конвертації, змінення кольорової гамми, накладання фільтрів, додавання водяних знаків на зображення [15].

Зовнішня база даних під час розробки боту не використовувалась. Уся необхідна інформація тимчасово зберігається в оперативній пам’яті (словнику context.user_data).

Під час розробки будуть застосовані наступні принципи програмування:

- модульність та розділення відповідальності – код розбитий на невеликі, ізольовані функції, кожна з яких відповідає за свою конкретну задачу;
- об’єктно орієнтований підхід – полягає у використанні класів із сторонніх бібліотек;

– інкапсуляція стану – збереження даних поточного сеансу в словнику context.user_data, що дозволяє передавати стан і параметри між функціями без використання глобальних змінних;

– обробка виключень – використання конструкції try/except є гарантією того, що некоректне введення даних не призведе до завершення роботи бота;

– використання констант для станів розмови – визначає унікальні значення для кожного етапу діалогу [16].

3 РОЗРОБКА ТЕЛЕГРАМ БОТУ ТА ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ ЙОГО РОБОТИ

3.1 Технічний опис розробки функціоналу

Повний код програми наведено у Додатку А. Першочергово виділимо основні функції та задачі, які буде розв’язувати даний програмний продукт. Користувач надсилає боту необхідне зображення та надає задачу додавання водяного знаку. Функціонал боту має надавати можливість додати водяний знак на зображення у двох форматах: текстовому та фото. Після виконання поставленої задачі бот надсилає користувачу вхідне зображення з водяним знаком.

Спочатку імпортуємо необхідні модулі та бібліотеки. Імпортуємо logging модуль для ведення логів стандартної бібліотеки Python, який дає можливість відслідковувати помилки та інформаційні повідомлення під час виконання програми. Імпортуємо об’єкт BytesIO з модуля io, використовується для роботи з бінарними даними у пам’яті. Після встановлення пакету Pillow командою pip install pillow, імпортуємо Image, ImageDraw, ImageFont, за допомогою цих класів буде здійснюватися редагування та збереження зображень, нанесення тексту. Імпортуємо Update, InlineKeyboardButton, InlineKeyboardMarkup з пакету telegram. Імпортуємо ключові компоненти фреймворку python‑telegram‑bot Updater, CommandHandler, MessageHandler, Filters, ConversationHandler, CallbackQueryHandler, CallbackContext з пакету telegram.ext [17].

Створимо функцію main в якій вкажемо токен телеграм-боту. За допомогою параметру use_context=True вмикаємо підхід, який дозволяє зручно передавати дані через об’єкт CallbackContext.

Створимо змінні, за допомогою функції range(11) створимо послідовність чисел, значення яких автоматично присвояться змінним у тому порядку у якому вони вказані. Такі змінні використовуються як ідентифікатор станів у ConversationHandler.

За допомогою функції logging.basicConfig налаштуємо базову конфігурацію системи логування у програмі, демонстрація на рисунку 3.1 [18].

```
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s', level=logging.INFO)
logger = logging.getLogger(__name__)
```

Рисунок 3.1 – Конфігурація системи логування

Створимо інлайн-клавіатуру build_position_keyboard для взаємодії з користувачем за допомогою функцій наведених на рисунку 3.2. Кожна з цих функцій повертає об’єкт типу InlineKeyboardMarkup з заданими кнопками InlineKeyboardButton.

```
def build_position_keyboard():  2 usages
    return InlineKeyboardMarkup([
        [InlineKeyboardButton( text: "зверху зліва", callback_data="зверху зліва"),
         InlineKeyboardButton( text: "зверху справа", callback_data="зверху справа")],
        [InlineKeyboardButton( text: "знизу зліва", callback_data="знизу зліва"),
         InlineKeyboardButton( text: "знизу справа", callback_data="знизу справа")],
        [InlineKeyboardButton( text: "по центру", callback_data="по центру")]
    ])
```

Рисунок 3.2 – Приклад функції для створення клавіатури

Створимо функцію start, яка буде виконувати роль вхідної функції. За допомогою методу reply_text після надсилання користувачем команди /start бот відправить йому вітальне повідомлення, після чого функція повертає константу PHOTO, бот переходить у стан очікування.

Створимо функцію photo_handler, яка буде відповідати за приймання та обробку зображення отриманого від користувача. Функція перевіряє чи містить повідомлення фото,якщо фото надіслано робота продовжується. Функція використовує метод get_file(), цей метод повертає об’єкт файлу доступний для завантаження. За допомогою методу download_as_bytearray() фото завантажується як послідовність байтів. Пакування цієї бінарної послідовності в об’єкт BytesIo дозволяє працювати з зображенням, як з потоком у пам'яті, не зберігаючи його в окремий файл. За допомогою функції Image.open() створюється об’єкт зображення, з використанням методу convert("RGBA") відбувається переведення зображення в режим RGBA. Оброблене зображення зберігається в словнику context.user_data під ключем ‘photo’. Це спеціальний словник об’єкту CallbackContext, який використовується для збереження даних користувача

протягом усієї сесії діалогу. Після надсилання користувачу повідомлення з проханням обрати тип водяного знаку, до якого додається інлайн-клавіатура створена функцією build_watermark_choice_keyboard(), функція повертає константу WATERMARK_CHOICE.

У випадку, якщо користувач надсилає, наприклад, текстове повідомлення, працює функція invalid_photo_handler. Ця функція надсилає повторний запит користувачу і повертає стан PHOTO. Це все прописано у ConversationHandler, демонстрація коду на рисунку 3.3 [19].

```
conv_handler = ConversationHandler(
    entry_points=[CommandHandler( command: "start", start)],
    states={
        PHOTO: [MessageHandler(Filters.photo, photo_handler),
                MessageHandler(~Filters.photo, invalid_photo_handler)],
        WATERMARK_CHOICE: [CallbackQueryHandler(watermark_choice_handler)],
        TEXT: [MessageHandler(Filters.text & ~Filters.command, text_handler)],
        COLOR: [CallbackQueryHandler(color_handler)],
        TEXT_BRIGHTNESS: [MessageHandler(Filters.text & ~Filters.command, text_brightness_handler)],
        WATERMARK_IMAGE: [MessageHandler(Filters.photo, watermark_image_handler)],
        POSITION: [CallbackQueryHandler(position_handler)],
        SIZE: [MessageHandler(Filters.text & ~Filters.command, size_handler)],
        SCALE: [MessageHandler(Filters.text & ~Filters.command, scale_handler)],
        ALPHA: [MessageHandler(Filters.text & ~Filters.command, alpha_handler)],
        RETRY: [CallbackQueryHandler(retry_handler)]
    },
    fallbacks=[CommandHandler( command: "cancel", cancel)]
)
```

Рисунок 3.3 – Код об’єкту ConversationHandler

За обробку даних з даного callback-запиту відповідає функція watermark_choice_handler. Об’єкт query містить дані про callback-запит, ці дані зберігаються в query.data. Це значення має два варіанти "wm_text" або "wm_photo". Якщо користувач обирає “текст” бот записує в context.user_data значення "text" під ключем "watermark_type", така сама процедура відбувається якщо користувач обирає “фото” відповідно. Якщо користувач обирає “текст”, функція повертає константу TEXT, якщо користувач обирає “фото” функція повертає константу WATERMARK_IMAGE.

У випадку, якщо користувач обирає текстовий водяний знак функція text_handler отримує текст із повідомлення і зберігає його в змінну watermark_text. в context.user_data. Після збереження тексту бот відправляє повідомлення с

запитом на вибір кольору, використовується інлайн-клавіатура функції build_color_keyboard(). Функція повертає константу COLOR.

Функція color_handler відповідає за обробку callback-запиту про вибір кольору . Значення обране користувачем зберігається у змінній color_code, за допомогою словника color-mapping функція перетворює отриманий код кольору у числове значення RGB. Отримане значення зберігається в словнику під ключем “text_color”. Функція містить звернення до користувача з проханням ввести значення видимості водяного знаку, на фіналі повертає константу TEXT_BRIGHTNESS.

В функції text_brightness_handler виконується отримання числового значення прозорості тексту водяного знаку. За допомогою функції int відбувається перетворення текстового повідомлення користувача, яке має містити в собі числове значення, в числове значення. Якщо користувач вводить некоректне повідомлення виникає помилка. Після перетворення, відбувається перевірка чи задовольняє число необхідний діапазон. Отримане значення зберігається в словнику під ключем “text_brightness”. Функція надсилає наступний запит, про позицію водяного знаку, з використанням інлайн-клавіатури build_position_keyboard(). Функція повертає константу POSITION.

Функція watermark_image_handler, яка відповідає за приймання фото для водяного знаку у вигляді фото працює за наведеними вище алгоритмами, демонстрація коду на рисунку 3.4.

```
133  def watermark_image_handler(update: Update, context: CallbackContext) -> int:  1 usage
134      if not update.message.photo:
135          update.message.reply_text("Надішліть фото водяного знака.")
136          return WATERMARK_IMAGE
137      photo_file = update.message.photo[-1].get_file()
138      photo_bytes = BytesIO(photo_file.download_as_bytearray())
139      try:
140          wm_image = Image.open(photo_bytes).convert("RGBA")
141      except Exception as e:
142          logger.error( msg: "Помилка: %s",  *args: e)
143          update.message.reply_text("Помилка обробки фото знака. Спробуйте ще раз.")
144          return WATERMARK_IMAGE
145      context.user_data["watermark_photo"] = wm_image
146      update.message.reply_text( text: "Фото водяного знака отримано. Оберіть позицію:", reply_markup=build_position_keyboard())
147      return POSITION
```

Рисунок 3.4 – Код функції watermark_image_handler

Функція position_handler відповідає за вибір параметрів розміру шрифту для текстового водяного знаку та масштабу відносно основного зображення водяного знаку у вигляді фото, застосовується у двох випадках розгалуження логіки за типом водяного знаку. За допомогою query_data присвоюємо дані, які були отримані з інлайн-клавіатури, змінній pos. Ця змінна записується в context_user_data під “postion”. Функція перевіряє, який тип водяного знака було обрано, за допомогою збереженої змінної у context.user_data("watermark_type"). У випадку, якщо обрано текстовий знак бот запитує числове значення, яке буде відповідати розміру шрифту. У цьому випадку функція повертає константу SIZE. Якщо обрано фото знак бот надсилає запит на отримання значення величини фото. Функція повертає константу SCALE.

Функція size_handler підсумовує усі отримані налаштування та виконує необхідні перетворення для отримання результату текстового водяного знаку. Функція отримує текстове повідомлення зі значення, яке має відповідати розміру шрифту, та перетворює його у числове значення за допомогою int(), зберігає його в context.user_data("font_size"). З context.user_data отримуються збережені дані: photo, watermark_text, position, text_color, text_brightness, font_size. Встановлюється кінцеве значення кольору тексту в форматі RGBA, за альфа-канал виступає збережене значення brightness, реалізація:

```
fill_color = (color[0], color[1], color[2], brightness)
```

Обчислюється ширина та висота вхідного зображення. Функція намагається завантажити шифр Arial, якщо завантажити шифр не вдається використовується шифр за замовчуванням.

Виконаємо ініціалізацію об’єкту, який надає можливість виконувати процедуру нанесення тексту на зображення. Використаємо метод textbbox для обчислення координат прямокутника, що охоплює текст. Перший параметр приймає координата з якої планується відображення тексту, другим параметром є watermark_text, третім шрифт. Метод повертає кортеж (x_min, y_min, x_max, y_max): Для отримання ширини тексту віднімаємо координату лівого краю

bbox[0] від координати правого краю bbox[2]. Висоту отримаємо різницею bbox[3] та bbox[1], реалізація коду:

```
draw = ImageDraw.Draw(image)
bbox = draw.textbbox((0, 0),watermark_text, font=font)
text_w = bbox[2] - bbox[0]
text_h = bbox[3] - bbox[1]
```

Обчислимо координати x та y, за якими буде нанесено водяний знак. Задамо константу margin, яка визначає відстань між границями зображення та текстом. Функція перевіряє, яке значення має змінна pos і відповідно до цього обчислює координати, роз'яснення у таблиці 3.1.

Таблиця 3.1 – Обчислення x,y за вказаним положенням.

Позиція	Обчислення
Зверху зліва	x, y = margin, margin
Зверху справа	x, y = img_w - text_w - margin, margin
Знизу зліва	x, y = margin, img_h - text_h - margin
Знизу справа	x, y = img_w - text_w -margin, img_h - text_h – margin
По центру	x, y = (img_w – text_w) // 2, (img_h – text_h) // 2

Після обчислення координат x та y використовуємо метод draw.text ((x, y), watermark_text, font=font, fill=fill_color), Реалізація коду:

```
margin = 10
    if pos in ("зверху зліва", "top_left"):
        x, y = margin, margin
    elif pos in ("зверху справа", "top_right"):
        x, y = img_w - text_w - margin, margin
    elif pos in ("знизу зліва", "bottom_left"):
        x, y = margin, img_h - text_h - margin
    elif pos in ("знизу справа", "bottom_right"):
        x, y = img_w - text_w - margin, img_h - text_h
- margin
    elif pos in ("по центру", "center"):
        x, y = (img_w - text_w) // 2, (img_h - text_h)
// 2
    else:
```

```
            x, y = img_w - text_w - margin, img_h - text_h
- margin
        draw.text((x,   y),   watermark_text,   font=font,
fill=fill_color)
```

Створимо об’єкт BytesIO для збереження зображення. В “watermarked_image.png” встановлюємо властивість bio.name, таким чином задаємо ім’я файлу для цього потоку. Використаємо метод image.save(bio, “PNG”) для збереження готового зображення з водяним знаком у цей буфер. Сформуємо підпис для наочного подання обраних користувачем характеристик. За допомогою методу reply_photo бот надсилає зображення збережене в об’єкті bio з підписом caption. Після цього бот відправляє користувачу повідомлення з запрошенням ще раз скористатися ботом та повертає константу RETRY [19].

Функція scale_handler обробляє введене користувачем значення величини водяного знаку у вигляді фото відносно вхідного зображення. За допомогою int функція перетворю текст у ціле число, це число представляє собою відсоткове значення масштабу. Після збереження бот надсилає запит на отримання значення видимості.

Функція alpha_handler перетворює вказане значення видимості в числове та проводить загальну обробку зображення для нанесення фото водяного знаку за вказаними параметрами. Підтягнуто такі збережені вхідні дані: base_img, wm_img, pos, scale_pct. Отримаємо розмір вхідного зображення з використанням властивості .size об’єкта baze_img - base_w, base_h. Щоб отримату ширину фото водяного знаку помножимо ширину вхідного зображення на значення scale_pct та поділимо це значення на сто .Аналогічно отримуємо ширину та висоту для зображення водяного знаку, після чого обчислюємо пропорційне співвідношення висоти до ширини. Висота водяного знаку обчислюється як добуток ширини і співвідношення. Реалізація коду:

```
base_w, base_h = base_img.size
target_w = int(base_w * scale_pct / 100)
 wm_w, wm_h = wm_img.size
aspect = wm_h / wm_w
target_h = int(target_w * aspect)
```

Масштабуємо зображення за допомогою методу resize, цей метод створює нове зображення, з розмірами заданими кортежем (target_w, target_h). Використовуємо алгоритм Lanczos для позиціонування, що дозволяє зменшувати/збільшувати зображення з мінімальними втратами по якості. За допомогою методу putalpha встановимо значення альфа-каналу (видимості). Встановлюємо стандартний відступ та переходимо до обчислення значення x, y відповідно до значення pos, значення змінних наведені у таблиці 3.2.

Таблиця 3.2 – Обчислення x, y за вказаним положенням

Позиція	Обчислення
Зверху зліва	x, y = margin, margin
Зверху справа	x, y = base_w - target_w - margin, margin
Знизу зліва	x, y = margin, base_h - target_h - margin
Знизу справа	x, y = base_w - target_w - margin, base_h - target_h - margin
По центру	x, y = (base_w - target_w) // 2, (base_h - target_h) // 2

Після обчислення координат використовуємо метод paste для нанесення водяного знаку. Створюємо об’єкт BytesIO для збереження кінцевого зображення. Після відправки результату надсилаємо пропозицію повторного запуску. Функція повертає RETRY [20, 21].

3.2 Перевірка роботи функціоналу програмного продукту

Необхідно перевірити чи робочі усі зазначені функції програми. Після запуску боту через команду /start бот надсилає вітальне повідомлення з проханням надіслати фото. Після надсилання користувачем фото, користувач отримує запит, якого типу водяний знак він бажає додати на фото. Якщо обрати текстовий водяний знак користувач отримує можливість вибору кольору, яскравості, розміру шрифту та позиції. Наведемо приклад порівняння отриманих зображень з водяним знаком. На рисунку 3.5 зображено два результати, в першому випадку

для водяного знаку застосовані наступні характеристики: позиція – по центру, видимість – 255, розмір шрифту – 10, текст – mary, колір – чорний. Не дивлячись на те, що застосований маленький шрифт завдяки 100% видимості водяний знак можна помітити. На другій частині рисунку 3.5 зображено результат до якого застосовані параметри: позиція – по центру, видимість – 10, розмір шрифту – 50, текст – mary, колір – чорний. Можемо зробити висновок, що параметри величини шрифту та видимості тексту функціонують справно. Недоліком є те, у нотатках до результату колір тексту вказаний в форматі RGB.

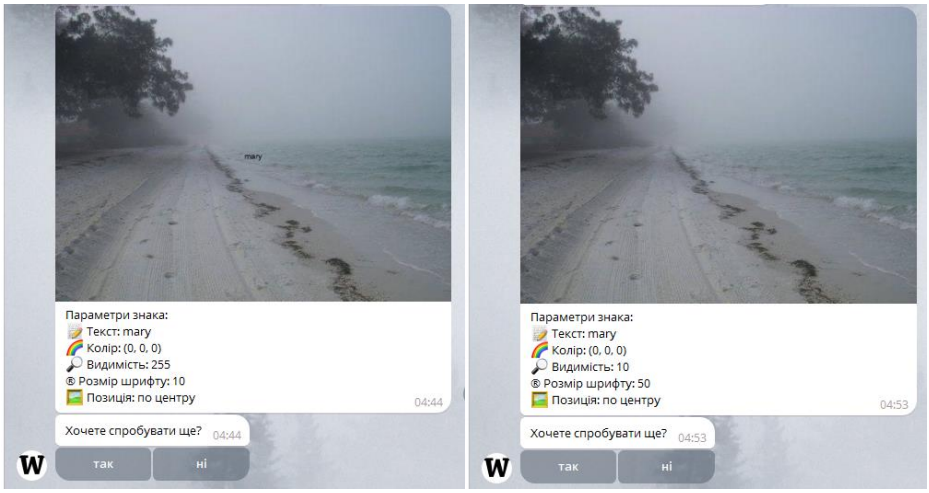

Рисунок 3.5 – Приклад результату обробки зображень ботом

Ми вже переконалися, що функція визначення прозорості працює справно. Переконаємося чи нормально функціонує присвоєння кольорів заданих користувачем. Можемо зробити висновок, що кольори встановлюються правильно, результат перевірки на рисунку 3.6.

Рисунок 3.6 – Приклад результату обробки зображень (колір)

Перевіримо чи працює позиціонування. Можемо зробити висновок, що задані користувачем позиції встановлюються правильно, результат перевірки на рисунку 3.7.

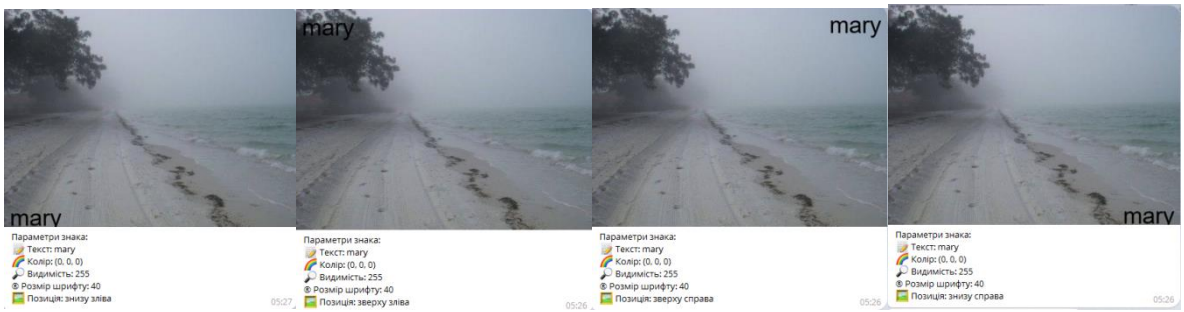

Рисунок 3.7 – Приклад результату обробки зображень (позиціонування)

Перевіримо роботу водяного знаку у вигляді фото. Все працює за описаним алгоритмом після надсилання фото для водяного знаку, бот робить запит на позиціонування. Після вибору користувача просить вказати величину водяного знаку, видимість, результат такої обробки зображено на рисунку 3.8.

Рисунок 3.8 – Результат нанесення водяного знаку у вигляді фото

Перевіримо позиціонування у цій функції, результат перевірки на рисунку 3.9.

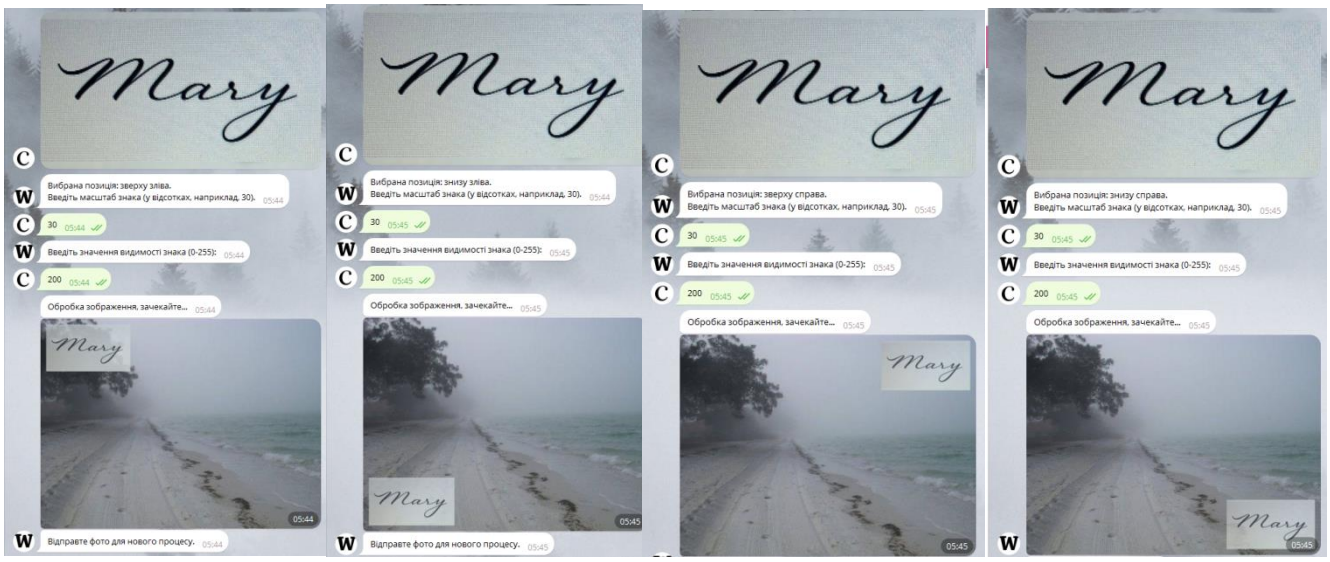

Рисунок 3.9 – Результат перевірки позиціонування

Функціонал фото-водяного знаку працює справно, недоліком є проблеми з масштабуванням під час накладання фото з різними параметрами довжини та висоти.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було проведено збір та аналіз інформації про особливості сучасних методів нанесення цифрових водяних знаків на цифровий контент з метою захисту його унікальності.

У ході дослідження було розглянуто методи нанесення ЦВЗ у просторовій та частотній областях. З цього аналізу можна зробити висновок, що комбінований метод, заснований на дискретному вейвлет-перетворенні (DWT) та дискретному косинусному перетворенні (DCT), на сьогодні вважається одним із найоптимальніших для нанесення цифрових водяних знаків. Його ефективність зумовлена поєднанням переваг обох перетворень: DWT забезпечує багаторівневий аналіз зображення, що дозволяє локалізувати водяний знак у частотній та просторовій областях, тоді як DCT підвищує стійкість до стиснення та шумів завдяки роботі в частотному домені.

Таким чином, цей метод поєднує переваги просторового та частотного аналізу, забезпечуючи високу ефективність. Метод LSB вирізняється простотою реалізації та високою ємністю, однак демонструє низьку стійкість до типових атак, таких як стиснення, фільтрація та геометричні перетворення. Це обмежує його застосування в системах, що вимагають високого рівня захисту інформації.

Під час створення цифрового водяного знака необхідно орієнтуватися на такі ключові параметри, як стійкість до атак, непомітність для людського ока, ємність вбудованої інформації, безпека та можливість відновлення. Успішна реалізація ЦВЗ передбачає досягнення балансу між цими вимогами, що забезпечує ефективний захист цифрового контенту без втрати його якості.

У кваліфікаційній роботі було розроблено простий Telegram-бот для захисту зображення у просторовій області. За допомогою цього бота можна наносити цифровий водяний знак із різним рівнем прозорості, кольору та розміру. Такий ЦВЗ не є стійким до атак стиснення, модифікації, фільтрації та людського втручання. Проте цей бот може бути використаний для публікації контенту в соціальних мережах.

Порівнюючи отриманий цифровий продукт з аналогами, можна зробити висновок, що всі ЦВЗ, отримані за допомогою Telegram-ботів, основаних на цьому підході, не є достатньо стійкими для професійного використання.

Отже, можна зробити висновок, що отриманий результат підходить для аматорського використання. Водночас, для більш серйозних цілей необхідно застосовувати алгоритми нанесення ЦВЗ у частотній області.

ПЕРЕЛІК ПОСИЛАНЬ

1. A Review of Watermarking Algorithms for Digital Image. International Journal of Innovative Research in Computer and Communication Engineering. 2014. Vol. 2, Issue 9. URL: https://surl.li/rwshfg (date of access: 09.04.2025).
2. Яремчук Ю., Карпінець В. Використання цифрових водяних знаків для захисту авторського права в зображеннях. URL: https://ela.kpi.ua/server/api/core/bitstreams/ede32c55-02fa-4032-80ae-c9a3e2d88415/content (дата звернення: 18.04.2025).
3. Aditya Kumar Sahu, Monalisa Sahu. Digital image steganography techniques in spatial domain: A study. International Journal of Pharmacy and Technology. 2017. URL:https://www.researchgate.net/publication/312826532_Digital_image_steganography_techniques_in_spatial_domain_A_study (date of access: 18.04.2025).
4. Least Significant Bit Algorithm in Information Security. URL: https://surl.gd/fkhnxc (date of access: 19.04.2025).
5. Haiming Li, Xiaoyun Guo. Embedding and Extracting Digital Watermark Based on DCT Algorithm. Journal of Computer and Communications. 2018. Vol 6, No 11. URL: https://www.scirp.org/journal/paperinformation?paperid=88826 (date of access: 22.04.2025).
6. Sudhanshi Sharma, Umesh Kumar. Review of Transform Domain Techniques for Image Steganography. URL: https://www.researchgate.net/publication/287210148_Review_of_Transform_Domain_Techniques_for_Image_Steganography (date of access: 25.04.2025).
7. Астраханцев А., Вовк О. О. Аналіз ефективності застосування вейвлетперетворення в стеганографічних системах передавання даних. Харківський національний університет радіоелектроніки. 2015. URL: https://science.lpnu.ua/sites/default/files/journal-paper/2018/jun/12922/3-9-17.pdf (дата звернення: 26.04.2025).

8. Лагун А. Н., Лагун І.І. Використання вейвлет-перетворення для приховування інформації в нерухомих зображеннях. Львівський державний університет безпеки життєдіяльності. 2013. URL: https://ena.lpnu.ua:8443/server/api/core/bitstreams/de54200c-ce46-4732-9419-e58591cf852c/content (дата звернення: 26.04.2025).
9. Tanmay Bhattacharya, Nilanjan Dey, S. R. Bhadra Chaudhur. A Session based Multiple Image Hiding Technique using DWT and DCT. International Journal of Computer Applications. 2012. URL: https://www.researchgate.net/publication/230609314_A_Session_based_Multiple_Image_Hiding_Technique_using_DWT_and_DCT (date of access: 18.04.2025).
10. Гунявий Д.А, Джулій В.М, Чешун В.М. ОЦІНКА СТІЙКОСТІ ЦИФРОВИХ ВОДЯНИХ ЗНАКІВ. Хмельницький національний університет. 2023. URL: https://conf.ztu.edu.ua/wp-content/uploads/2023/06/34.pdf (дата звернення: 23.04.2025).
11. Hamurabi Gamboa Rosales, Chalee Vorakulpipat, Mohammad Ali Nematollahi. Digital Watermarking.Techniques and Trends / ed. by Jacob Benesty Walter Kellermann. 11th ed. Springer, 2017. URL: https://pdf.lib.vntu.edu.ua/books/Springer/2021/2017_Book_DigitalWatermarking.pdf (date of access: 30.04.2025).
12. This telegram bot built with Python and FFmpeg allows you to watermark photos and videos, on the fly. URL: https://dev.to/aahnik/this-bot-built-with-python-and-ffmpeg-allows-you-to-watermark-photos-and-videos-on-the-fly-5af2 (date of access: 13.03.2025).
13. python-telegram-bot Documentation. URL: https://python-telegram-bot.org/ (date of access: 20.03.2025).
14. What is PyCharm? Features, Advantages & Disadvantages. URL: https://www.geeksforgeeks.org/what-is-pycharm-features-advantages-disadvantages/ (date of access: 27.03.2025).
15. Pillow Documentation. URL: https://pillow.readthedocs.io/en/stable/ (date of access: 28.03.2025).

16. ContextTypes. URL: https://docs.python-telegram-bot.org/en/v21.5/telegram.ext.contexttypes.html (date of access: 28.03.2025).
17. Building a Simple Telegram Bot with Buttons Using Python. URL: https://medium.com/@travilabs/building-a-simple-telegram-bot-with-buttons-using-python-0a16c52485c0 (date of access: 29.03.2025).
18. ConversationHandler. URL: https://docs.python-telegram-bot.org/en/v21.5/telegram.ext.conversationhandler.html (date of access: 29.03.2025).
19. logging – Logging facility for Python. URL: https://docs.python.org/3/library/logging.html (date of access: 29.03.2025).
20. ImageDraw Module. URL: https://pillow.readthedocs.io/en/stable/reference/ImageDraw.html (date of access: 29.03.2025).
21. Python PIL putalpha() Method. URL: https://www.geeksforgeeks.org/python-pil-putalpha-method/ (date of access: 28.03.2025).
22. Lanczos Interpolation in Python with 2D images. URL: https://stackoverflow.com/questions/60416856/lanczos-interpolation-in-python-with-2d-images (date of access: 29.03.2025).

Додаток А. Програмний код

```
import logging
from io import BytesIO
from PIL import Image, ImageDraw, ImageFont
from telegram import Update, InlineKeyboardButton,
InlineKeyboardMarkup
from telegram.ext import (Updater, CommandHandler, MessageHandler,
Filters,
                          ConversationHandler, CallbackQueryHandler,
CallbackContext)

PHOTO, WATERMARK_CHOICE, TEXT, COLOR, TEXT_BRIGHTNESS,
WATERMARK_IMAGE, POSITION, SIZE, SCALE, ALPHA, RETRY = range(11)

logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s -
%(message)s', level=logging.INFO)
logger = logging.getLogger(__name__)


def build_watermark_choice_keyboard():
    return InlineKeyboardMarkup([
        [InlineKeyboardButton("текст", callback_data="wm_text"),
         InlineKeyboardButton("фото", callback_data="wm_photo")]
    ])

def build_color_keyboard():
    return InlineKeyboardMarkup([
        [InlineKeyboardButton("Чорний", callback_data="чорний"),
         InlineKeyboardButton("Білий", callback_data="білий")],
        [InlineKeyboardButton("Червоний", callback_data="червоний"),
         InlineKeyboardButton("Зелений", callback_data="зелений")],
        [InlineKeyboardButton("Синій", callback_data="блакитний")]
    ])

def build_position_keyboard():
    return InlineKeyboardMarkup([
        [InlineKeyboardButton("зверху зліва", callback_data="зверху
зліва"),
         InlineKeyboardButton("зверху справа", callback_data="зверху
справа")],
        [InlineKeyboardButton("знизу зліва", callback_data="знизу
зліва"),
         InlineKeyboardButton("знизу справа", callback_data="знизу
справа")],
        [InlineKeyboardButton("по центру", callback_data="по
центру")]
    ])

def build_retry_keyboard():
    return InlineKeyboardMarkup([[InlineKeyboardButton("так",
callback_data="да"),
```

```
                                    InlineKeyboardButton("ні",
callback_data="нет")]])


def start(update: Update, context: CallbackContext) -> int:
    update.message.reply_text(
        "Привіт, я бот для додавання водяних знаків на зображення!
:)\n"
        "За допомогою мене Ви зможете додати водяний знак у вигляді
тексту чи фото.\n"
        "Також доступна можливість обрати:\n"
        "⚡ колір, розмір, видимість та положення для текстового
знака\n"
        "⚡ видимість та положення для знака у вигляді фото\n"
        "Відправте мені фото для початку роботи."
    )
    return PHOTO

def photo_handler(update: Update, context: CallbackContext) -> int:
    if not update.message.photo:
        update.message.reply_text("Надішліть будь ласка фото.!")
        return PHOTO
    photo_file = update.message.photo[-1].get_file()
    photo_bytes = BytesIO(photo_file.download_as_bytearray())
    try:
        image = Image.open(photo_bytes).convert("RGBA")
    except Exception as e:
        logger.error("Не вдалося відкрити зображення: %s", e)
        update.message.reply_text("Помилка при обробці зображення.
Спробуйте ще раз.")
        return PHOTO
    context.user_data['photo'] = image
    update.message.reply_text("Фото додано успішно. Оберіть тип
водяного знака, який хочете додати:",
reply_markup=build_watermark_choice_keyboard())
    return WATERMARK_CHOICE

def invalid_photo_handler(update: Update, context: CallbackContext)
-> int:
    update.message.reply_text("Надішліть будь ласка фото.!")
    return PHOTO

def watermark_choice_handler(update: Update, context:
CallbackContext) -> int:
    query = update.callback_query
    query.answer()
    choice = query.data
    if choice == "wm_text":
        context.user_data["watermark_type"] = "text"
        query.edit_message_text("Ви обрали текстовий водяний
знак.\nВведіть текст:")
        return TEXT
    elif choice == "wm_photo":
```

```
        context.user_data["watermark_type"] = "photo"
        query.edit_message_text("Ви обрали водяний знак у вигляді
фото.\nВідправте фото водяного знаку:")
        return WATERMARK_IMAGE
    else:
        query.edit_message_text("Невідомий вибір, спробуйте ще
раз.")
        return WATERMARK_CHOICE

def text_handler(update: Update, context: CallbackContext) -> int:
    watermark_text = update.message.text
    if not watermark_text:
        update.message.reply_text("Будь ласка, введіть текст.")
        return TEXT
    context.user_data["watermark_text"] = watermark_text
    update.message.reply_text("Оберіть колір для тексту:",
reply_markup=build_color_keyboard())
    return COLOR

def color_handler(update: Update, context: CallbackContext) -> int:
    query = update.callback_query
    query.answer()
    color_code = query.data
    color_mapping = {
        "чорний": (0, 0, 0),
        "білий": (255, 255, 255),
        "червоний": (255, 0, 0),
        "зелений": (0, 255, 0),
        "блакитний": (0, 0, 255)
    }
    context.user_data["text_color"] = color_mapping.get(color_code,
(0, 0, 0))
    query.edit_message_text(
        f"Обрано колір: {color_code}.\nВведіть значення видимості
(0-255, де 255 - максимальна непрозорість)."
    )
    return TEXT_BRIGHTNESS

def text_brightness_handler(update: Update, context:
CallbackContext) -> int:
    try:
        brightness = int(update.message.text)
    except ValueError:
        update.message.reply_text("Введіть коректне числове
значення.")
        return TEXT_BRIGHTNESS
    if not (0 <= brightness <= 255):
        update.message.reply_text("Введіть число від 0 до 255.")
        return TEXT_BRIGHTNESS
    context.user_data["text_brightness"] = brightness
    update.message.reply_text("Оберіть позицію водяного знака:",
reply_markup=build_position_keyboard())
    return POSITION
```

```
def watermark_image_handler(update: Update, context:
CallbackContext) -> int:
    if not update.message.photo:
        update.message.reply_text("Надішліть фото водяного знака.")
        return WATERMARK_IMAGE
    photo_file = update.message.photo[-1].get_file()
    photo_bytes = BytesIO(photo_file.download_as_bytearray())
    try:
        wm_image = Image.open(photo_bytes).convert("RGBA")
    except Exception as e:
        logger.error("Помилка: %s", e)
        update.message.reply_text("Помилка обробки фото знака.
Спробуйте ще раз.")
        return WATERMARK_IMAGE
    context.user_data["watermark_photo"] = wm_image
    update.message.reply_text("Фото водяного знака отримано. Оберіть
позицію:", reply_markup=build_position_keyboard())
    return POSITION


def position_handler(update: Update, context: CallbackContext) ->
int:
    query = update.callback_query
    query.answer()
    pos = query.data
    context.user_data["position"] = pos
    if context.user_data.get("watermark_type") == "text":
        query.edit_message_text(f"Вибрана позиція: {pos}.\nВведіть
розмір шрифту (числове значення).")
        return SIZE
    elif context.user_data.get("watermark_type") == "photo":
        query.edit_message_text(f"Вибрана позиція: {pos}.\nВведіть
масштаб знака (у відсотках, наприклад, 30).")
        return SCALE
    else:
        query.edit_message_text("Невідомий тип знака.")
        return ConversationHandler.END

def size_handler(update: Update, context: CallbackContext) -> int:

    try:
        font_size = int(update.message.text)
    except ValueError:
        update.message.reply_text("Введіть числове значення для
розміру шрифту.")
        return SIZE
    context.user_data["font_size"] = font_size
    update.message.reply_text("Обробка зображення, зачекайте...")


    image = context.user_data["photo"]
    watermark_text = context.user_data["watermark_text"]
```

```
    pos = context.user_data["position"]
    color = context.user_data["text_color"]
    brightness = context.user_data["text_brightness"]
    font_size = context.user_data["font_size"]

    fill_color = (color[0], color[1], color[2], brightness)
    img_w, img_h = image.size

    try:
        font = ImageFont.truetype("arial.ttf", font_size)
    except Exception as e:
        logger.warning("Не вдалося завантажити arial.ttf: %s.
Використовуємо стандартний шрифт.", e)
        font = ImageFont.load_default()

    draw = ImageDraw.Draw(image)
    bbox = draw.textbbox((0, 0), watermark_text, font=font)
    text_w = bbox[2] - bbox[0]
    text_h = bbox[3] - bbox[1]

    margin = 10
    if pos in ("зверху зліва", "top_left"):
        x, y = margin, margin
    elif pos in ("зверху справа", "top_right"):
        x, y = img_w - text_w - margin, margin
    elif pos in ("знизу зліва", "bottom_left"):
        x, y = margin, img_h - text_h - margin
    elif pos in ("знизу справа", "bottom_right"):
        x, y = img_w - text_w - margin, img_h - text_h - margin
    elif pos in ("по центру", "center"):
        x, y = (img_w - text_w) // 2, (img_h - text_h) // 2
    else:
        x, y = img_w - text_w - margin, img_h - text_h - margin

    # Створення окремого прозорого шару для тексту
    txt_layer = Image.new("RGBA", image.size, (255, 255, 255, 0))
    draw_txt = ImageDraw.Draw(txt_layer)
    draw_txt.text((x, y), watermark_text, font=font,
fill=fill_color)

    watermarked = Image.alpha_composite(image.convert("RGBA"),
txt_layer)


    bio = BytesIO()
    bio.name = "watermarked_image.png"
    watermarked.save(bio, "PNG")
    bio.seek(0)

    caption = (
        f"Параметри знака:\n"
        f"□ Текст: {watermark_text}\n"
        f"□ Колір: {color}\n"
```

```
        f"□ Видимість: {brightness}\n"
        f"® Розмір шрифту: {font_size}\n"
        f"□ Позиція: {pos}"
    )
    update.message.reply_photo(photo=bio, caption=caption)
    update.message.reply_text("Хочете спробувати ще?",
reply_markup=build_retry_keyboard())
    return RETRY

def scale_handler(update: Update, context: CallbackContext) -> int:
    try:
        scale_pct = int(update.message.text)
    except ValueError:
        update.message.reply_text("Введіть числове значення для
масштабу.")
        return SCALE
    context.user_data["scale"] = scale_pct
    update.message.reply_text("Введіть значення видимості знака (0-
255):")
    return ALPHA

def alpha_handler(update: Update, context: CallbackContext) -> int:
    try:
        alpha = int(update.message.text)
    except ValueError:
        update.message.reply_text("Введіть числове значення для
видимості.")
        return ALPHA
    context.user_data["alpha"] = alpha
    update.message.reply_text("Обробка зображення, зачекайте...")

    base_img = context.user_data["photo"]
    wm_img = context.user_data["watermark_photo"]
    pos = context.user_data["position"]
    scale_pct = context.user_data["scale"]

    base_w, base_h = base_img.size
    target_w = int(base_w * scale_pct / 100)
    wm_w, wm_h = wm_img.size
    aspect = wm_h / wm_w
    target_h = int(target_w * aspect)

    resized_wm = wm_img.resize((target_w, target_h),
Image.Resampling.LANCZOS)
    resized_wm.putalpha(context.user_data["alpha"])

    margin = 10
    if pos in ("зверху зліва", "top_left"):
        x, y = margin, margin
    elif pos in ("зверху справа", "top_right"):
        x, y = base_w - target_w - margin, margin
    elif pos in ("знизу зліва", "bottom_left"):
        x, y = margin, base_h - target_h - margin
```

```
    elif pos in ("знизу справа", "bottom_right"):
        x, y = base_w - target_w - margin, base_h - target_h -
margin
    elif pos in ("по центру", "center"):
        x, y = (base_w - target_w) // 2, (base_h - target_h) // 2
    else:
        x, y = base_w - target_w - margin, base_h - target_h -
margin

    base_img.paste(resized_wm, (x, y), resized_wm)
    bio = BytesIO()
    bio.name = "watermarked_image.png"
    base_img.save(bio, "PNG")
    bio.seek(0)
    update.message.reply_photo(photo=bio)
    update.message.reply_text("Хочете спробувати ще?",
reply_markup=build_retry_keyboard())
    return RETRY

def retry_handler(update: Update, context: CallbackContext) -> int:
    query = update.callback_query
    query.answer()
    if query.data.lower() == "да":
        query.edit_message_text("Відправте фото для нового
процесу.")
        return PHOTO
    else:
        query.edit_message_text("Дякуємо за використання бота! До
зустрічі!")
        return ConversationHandler.END

def cancel(update: Update, context: CallbackContext) -> int:
    update.message.reply_text("Процес відмінено.")
    return ConversationHandler.END

def main():
    updater =
Updater("7985385533:AAEpuoH15MlopDAJWZxcOlTrjKznFuWFQag",
use_context=True)
    dp = updater.dispatcher

    conv_handler = ConversationHandler(
        entry_points=[CommandHandler("start", start)],
        states={
            PHOTO: [MessageHandler(Filters.photo, photo_handler),
                    MessageHandler(~Filters.photo,
invalid_photo_handler)],
            WATERMARK_CHOICE:
[CallbackQueryHandler(watermark_choice_handler)],
            TEXT: [MessageHandler(Filters.text & ~Filters.command,
text_handler)],
            COLOR: [CallbackQueryHandler(color_handler)],
```

```
            TEXT_BRIGHTNESS: [MessageHandler(Filters.text &
~Filters.command, text_brightness_handler)],
            WATERMARK_IMAGE: [MessageHandler(Filters.photo,
watermark_image_handler)],
            POSITION: [CallbackQueryHandler(position_handler)],
            SIZE: [MessageHandler(Filters.text & ~Filters.command,
size_handler)],
            SCALE: [MessageHandler(Filters.text & ~Filters.command,
scale_handler)],
            ALPHA: [MessageHandler(Filters.text & ~Filters.command,
alpha_handler)],
            RETRY: [CallbackQueryHandler(retry_handler)]
        },
        fallbacks=[CommandHandler("cancel", cancel)]
    )

    dp.add_handler(conv_handler)
    updater.start_polling()
    updater.idle()

if __name__ == "__main__":
    main()
```