МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МАРІУПОЛЬСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЕКОНОМІКО-ПРАВОВИЙ ФАКУЛЬТЕТ
КАФЕДРА СИСТЕМНОГО АНАЛІЗУ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Допустити до захисту
В.о. завідувача кафедри
_____ Мнацаканян М.С.
(підпис) *(ПІБ завідувача кафедри)*

«_____»_____20__ р.

РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКА ДЛЯ ОБРОБКИ ТА ПЕРЕГЛЯДУ ГРАФІЧНИХ ЗОБРАЖЕНЬ

Кваліфікаційна робота
здобувача вищої освіти
першого (бакалаврського) рівня
вищої освіти
освітньо-професійної програми
«____Комп'ютерні науки_____»
(назва освітньо-професійної програми)
Кузнецов Віктор Олександрович
(прізвище, ім'я, по батькові здобувача вищої освіти)

Науковий керівник:
_Козловський В.В_____
(прізвище, ініціали, науковий ступінь, вчене звання)

Рецензент:
__Охріменко Т.О., к.т.н., НАУ_
(прізвище, ініціали, науковий ступінь, вчене звання, місце роботи)

Кваліфікаційна робота
захищена з
оцінкою_____
_ Секретар
ЕК_____ «____»
_____20__ р.

Київ -2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МАРІУПОЛЬСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЕКОНОМІКО-ПРАВОВИЙ ФАКУЛЬТЕТ
КАФЕДРА СИСТЕМНОГО АНАЛІЗУ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Рівень вищої освіти *бакалавр*
Шифр та назва спеціальності *122 «Комп'ютерні науки»*
Освітньо-професійна програма *«Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри к.т.н.
(науковий ступінь, вчене звання)

_____ Мнацаканян М.С.
(підпис) *(ПІБ завідувача кафедри)*

«__» _____ 20__ р.

ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Кузнецов Віктор Олександрович

(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка програмного застосунка для обробки та перегляду графічних зображень»

керівник роботи Козловський В.В, к.т.н., доцент,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом Маріупольського державного університету від «___» _____20__ р. №__

2. Строк подання здобувачем роботи.

3. Вихідні дані до роботи (мета, об'єкт, предмет):

Мета: розробка та реалізація програмного забезпечення для обробки зображень з використанням сучасних технологій та алгоритмів.

Розробляємий додаток: програмний застосунок для обробки та перегляду графічних зображень за допомогую мови програмування Python в середовище розробки Visual Studio Code з застосування бібліотке Tkinter та PIL.

Завдання дипломної роботи: Аналіз сучасних методів та алгоритмів обробки графічних зображень на Python. Вибір інструментів та бібліотек для розробки програмного застосунка. Розробка архітектури програмного застосунка. Реалізація основних функціональних можливостей програмного застосунка. Тестування програмного застосунка.

4. Зміст роботи

Розділ 1. Огляд технологій та методів.

Розділ 2. Проектування програми.

Розділ 3. Впровадження додатків.

Розділ 4. Результати та аналіз.

5. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Огляд технологій та методів	21.03.2023	
2.	Проектування програми	04.04.2023	
3.	Впровадження додатків	15.04.2023	
4.	Результати та аналіз	29.04.2023	
5.	Оформлення кваліфікаційної бакалаврської роботи.	15.05.2023	
6.	Розробка графічного матеріалу для презентації роботи.	19.05.2023	

Здобувач _____ _____Кузнецов В.О._____
(підпис) *(прізвище та ініціали)*

Науковий керівник роботи _____ _____Козловський В.В_____
(підпис) *(прізвище та ініціали)*

Анотація

Пояснювальна записка до кваліфікаційної роботи на тему «Розробка програмного застосунка для обробки та перегляду графічних зображень» поєднує в собі наступні розділи: огляд технологій та методів, проектування програми, впровадження додатків та результати та аналіз.

У першому розділі проведено огляд технологій та методів, що використовуються для розробки подібних додатків. Розглянуто різні бібліотеки та фреймворки, доступні для роботи з графічними зображеннями, а також проведено аналіз можливостей та обмежень мови Python у контексті розробки таких додатків.

У другому розділі роботи описано архітектуру програми та її компоненти. Розроблено інтерфейс користувача, який дозволяє зручно взаємодіяти з програмою. Надано детальний опис основних функцій та можливостей розробленої програми.

Третій розділ роботи присвячена впровадженню додатків. Описано основні етапи розробки програми, включаючи реалізацію функцій та компонентів. Проведено тестування та налагодження програми для забезпечення її коректної роботи.

У четвертому розділі роботи наведено результати та аналіз розробленого додатку. Оцінено якість та ефективність програми. Здійснено порівняння з існуючими програмними рішеннями. Також наведено опис можливих напрямів для подальшої розробки та покращення програми.

Дипломна робота містить інформацію, яка може бути корисною для розробників, які працюють над програмами для обробки графічних зображень з використанням мови Python.

Зміст

Вступ

1. Опис проблеми та ціль роботи.

У сучасному світі фотографії стали невід'ємною частиною життя. Ми робимо знімки на мобільних пристроях, фотоапаратах, використовуємо соціальні мережі та месенджери для обміну фотографіями. Однак, не кожна фотографія може бути ідеальною, тому важлива обробка фотографій.

По-перше, обробка фотографій дозволяє покращити якість знімків. За допомогою спеціальних програм та інструментів можна усунути шуми та дефекти на фото, скоригувати яскравість, контрастність та кольорову гаму. Також можна забрати непотрібні об'єкти з фотографії або додати потрібні елементи.

По-друге, обробка фотографій допомагає передати певний настрій чи емоції. Використовуючи різні ефекти та фільтри, можна створити унікальний стиль фотографії. Наприклад, чорно-біла обробка може передавати строгість і класику, а насичені кольори – яскравість та енергію.

По-третє, обробка фотографій є необхідною для професійних фотографів. Вони не тільки повинні знати технічні аспекти зйомки, а й вміти обробляти фотографії для досягнення найкращого результату.

Редагування фотографій дуже важливо для компаній електронної комерції. Якість зображення безпосередньо впливає на думку людей про продукт і кількість продажів. Дослідження підтверджують, що високоякісні зображення перевершують стандартні (або низькоякісні) зображення, а збільшення кількості високоякісних зображень може завоювати довіру споживачів і підвищити коефіцієнт конверсії.[1]

Мета даної дипломної роботи – розробка та реалізація програмного забезпечення для обробки зображень з використанням сучасних технологій та алгоритмів. Кінцева мета роботи полягає у створенні інструменту, який дозволить користувачеві проводити різні операції із зображеннями, такі як редагування, фільтрація, зміна розміру та формату, а також застосування різних ефектів та фільтрів для покращення якості зображення. Результатом роботи має бути проста та інтуїтивно зрозуміла програма, яка буде доступна для широкого кола користувачів та зможе задовольнити їхні потреби в обробці зображень.

Основними функціями цього застосунку будуть:

- Відкриття кількох зображень в одній програмі.
- Збереження зображень.
- Поворот зображення.

- Віддзеркалення зображення.
- Зміна розміру.
- Застосування фільтрів.
- Виділення області.
- Обрізання зображення.

2. Обґрунтування вибору мови Python та середовища розробки.

Мова програмування Python є однією з найпопулярніших мов у світі програмування. Він має безліч переваг, які роблять його придатним для написання програми краще, ніж багато інших мов програмування:

- Простота у використанні: Python має простий і зрозумілий синтаксис, який робить його легко зрозумілим для програмістів-початківців та експертів.
- Кросплатформеність: Python працює на всіх основних операційних системах, включаючи Windows, MacOS, Linux і т.д.
- Велика кількість бібліотек: Python має величезну кількість бібліотек та модулів, що дозволяє швидко та легко реалізовувати різноманітні завдання, включаючи обробку зображень, машинне навчання, аналіз даних тощо.
- Висока продуктивність: Python має високу продуктивність завдяки використанню інтерпретатора CPython, який дозволяє виконувати обчислення швидше, ніж в інших мовах, що інтерпретуються.
- Велика спільнота: Python має величезну та активну спільноту, яка створює нові бібліотеки, вирішує проблеми та обговорює нові можливості, що робить її більш доступною та зручною для роботи.
- Інтеграція з іншими мовами: Python може бути інтегрований з іншими мовами програмування, такими як C/C++, що дозволяє створювати складніші програми.

В цілому, Python підходить для написання програм краще, ніж багато інших мов програмування завдяки своїй простоті, зручності використання,

великій кількості бібліотек, високій продуктивності та великій спільноті розробників.

За своєю суттю Visual Studio Code — це редактор вихідного коду, який ідеально підходить для щоденного використання. VS Code підтримує сотні мов і миттєво підвищує продуктивність завдяки таким функціям, як підсвічування синтаксису, зіставлення дужок, автоматичний відступ, вибір поля, фрагменти коду тощо. Інтуїтивно зрозумілі комбінації клавіш, просте налаштування та зіставлення гарячих клавіш, надані спільнотою, спрощують навігацію по коду.

Visual Studio Code включає вбудовану підтримку завершення коду IntelliSense, глибоке розуміння семантичного коду та навігацію, а також рефакторинг коду.

Отладка часто є єдиною функцією, якої найбільше не вистачає розробникам, коли працюють над більш компактним кодом. Visual Studio Code містить інтерактивний налагоджувач, щоб було можливо покроково переглядати вихідний код, перевіряти змінні, переглядати стек викликів і виконувати команди в консолі.

VS Code також інтегрується з інструментами збирання та для виконання спільних завдань, прискорюючи повсякденні робочі процеси. VS Code підтримує Git, тому може працювати з системою керування версіями, не виходячи з редактора, включаючи перегляд змін, що очікують. [2]

II. Огляд технологій та методів

1. Опис використовуваних бібліотек та фреймворків.

В цій программі для створення GUI, я використовував біблеотеку Tkinter.

Tkinter - це стандартна бібліотека Python, яка дає змогу розробникам створювати графічні інтерфейси для своїх додатків. Tkinter є інструментом для створення графічних користувацьких інтерфейсів (GUI), який є важливою частиною програмування на Python. Бібліотека Tkinter доступна в стандартній бібліотеці Python, тому розробники можуть використовувати її для створення GUI без необхідності встановлення додаткових бібліотек.

Tkinter заснована на бібліотеці Tk, яка була розроблена для мови програмування Tcl. Tkinter надає широкий набір віджетів та елементів керування, які можуть бути використані для створення різних типів користувацьких інтерфейсів. Деякі з цих віджетів включають в себе кнопки, текстові поля, список вибору, прапорці, перемикачі та багато іншого.

Одним із головних компонентів Tkinter є головне вікно, яке є кореневим елементом для всіх інших віджетів і елементів керування. Розробники можуть використовувати це вікно для управління своїми додатками і розміщення віджетів на ньому. Tkinter також надає можливість створення дочірніх вікон і діалогових вікон, які можуть бути використані для створення більш складних користувацьких інтерфейсів.

Tkinter має низку переваг, які роблять її популярним вибором для створення GUI на Python. Однією з цих переваг є її кросплатформеність. Додатки, створені за допомогою Tkinter, працюватимуть на будь-якій платформі, що підтримує Python і Tkinter. Це робить її дуже зручною для розробників, які хочуть створювати додатки, що можуть бути запущені на різних операційних системах.

Ще однією перевагою Tkinter є її інтуїтивно зрозумілий інтерфейс. Бібліотека була розроблена з урахуванням зручності використання, що робить її простою для вивчення і використання навіть для початківців

розробників. Tkinter також має велику документацію і безліч ресурсів, які можуть допомогти розробникам створювати додатки більш ефективно.

Хоча Tkinter надає безліч переваг для створення GUI на Python, вона також має деякі недоліки, які можуть бути важливими для деяких розробників.

Один із недоліків Tkinter полягає в тому, що він має обмежені можливості щодо дизайну та стилізації користувацького інтерфейсу. Хоча Tkinter надає широкий набір віджетів і елементів керування, які можуть бути використані для створення різних типів користувацьких інтерфейсів, можливості зі зміни їхнього зовнішнього вигляду обмежені. Це може обмежувати можливості для створення додатків з більш сучасним і професійним дизайном.

Іншим недоліком Tkinter є його обмежена підтримка багатопоточності. Tkinter не надає механізми для опрацювання багатопоточності, тому розробникам може знадобитися використовувати інші бібліотеки для створення багатопотокових додатків.

Також слід зазначити, що документація Tkinter не завжди є повною і зрозумілою, особливо для нових розробників. Деякі аспекти бібліотеки можуть бути складними для розуміння, і можуть знадобитися додаткові дослідження та експерименти, щоб розібратися в них.

Нарешті, деякі розробники можуть вважати, що Tkinter не має достатньої гнучкості для створення більш складних користувацьких інтерфейсів. У деяких випадках розробники можуть звертатися до інших бібліотек, щоб отримати більшу гнучкість і можливості для створення складніших користувацьких інтерфейсів.

Загалом, незважаючи на деякі недоліки, Tkinter залишається популярним інструментом для створення GUI на Python, особливо для початківців-розробників і для створення простих додатків.

2. Огляд методів обробки графічних зображень

Існує безліч методів обробки графічних зображень, залежно від цілей і завдань, які необхідно вирішити. Деякі з найпоширеніших методів обробки графічних зображень включають в себе:

- Зміна розміру зображення: Цей метод дає змогу змінювати розмір зображення відповідно до необхідних розмірів і співвідношення сторін. Це може бути корисно для створення мініатюр зображень, а також для зменшення розміру зображення для оптимізації зберігання і передачі.
- Кадрування: Цей метод дає змогу обрізати зображення відповідно до певних параметрів. Наприклад, кадрування може використовуватися для видалення небажаних елементів зображення або для створення зображення з певним співвідношенням сторін.
- Фільтри: Фільтри дають змогу змінювати характеристики зображення, такі як яскравість, контрастність, різкість і колірний баланс. Ці методи можуть бути використані для поліпшення якості зображень, а також для створення ефектів, таких як сепія або чорно-біле зображення.
- Розпізнавання об'єктів: Цей метод дає змогу автоматично знаходити певні об'єкти або області на зображенні, наприклад, обличчя людей або автомобілів. Це може бути корисно для автоматичної обробки зображень, таких як сортування зображень за вмістом або автоматичне обрізання зображень відповідно до знайдених об'єктів.
- Розпізнавання тексту: Цей метод дає змогу автоматично розпізнавати текст на зображенні та перетворювати його в текстовий формат. Це може бути корисно для автоматичної обробки сканованих документів або зображень із текстом.
- Сегментація зображень: Цей метод дає змогу розділити зображення на окремі ділянки на основі їхніх властивостей, як-от колір, текстура або яскравість. Це може бути корисно для аналізу та класифікації зображень.

- Обробка за допомогою нейронних мереж: Нейронні мережі можуть бути використані для автоматичної обробки зображень, наприклад, для розпізнавання об'єктів, класифікації зображень або генерації нових зображень на основі навчальних даних.[3]

3. Аналіз можливостей та обмежень мови Python для розробки додатків такого типу.

Python - це одна з найпопулярніших мов програмування, яка широко використовується в багатьох галузях, включно з науковою та інженерною галузями, бізнес-сектором та іншими. Python також є чудовим вибором для обробки та аналізу зображень завдяки своїй багатій бібліотеці, простому і зрозумілому синтаксису і доступності для новачків.

Python надає безліч бібліотек для роботи із зображеннями, таких як Pillow, OpenCV, Scikit-image та інші. Вони надають безліч функцій для обробки зображень, включно зі зміною розміру, різкістю, накладенням фільтрів, коригуванням кольору та багато іншого. Деякі з цих бібліотек, такі як Pillow, мають інтуїтивно зрозумілий інтерфейс, що робить їх дуже доступними для новачків.

Python також має потужний інструментарій для наукових обчислень, який може бути корисним для обробки зображень. Наприклад, бібліотека NumPy дає змогу працювати з багатовимірними масивами, а бібліотека SciPy надає різноманітні функції для наукових та інженерних обчислень, як-от інтерполяція, оптимізація тощо.

Ба більше, Python має простий і зрозумілий синтаксис, що робить його дуже доступним для новачків. Це дає змогу швидко розвивати додатки, а також спрощує роботу з кодом і налагодження. Python також має велику спільноту розробників, які часто діляться своїми знаннями та досвідом в Інтернеті.

Python є кросплатформенною мовою програмування, що робить її дуже гнучкою і зручною для розробки додатків. Python працює на різних

операційних системах, включно з Windows, macOS і Linux, що робить його дуже гнучким і зручним для розробки додатків.

Незважаючи на всі свої переваги, у Python є й обмеження для розробки додатків для обробки та перегляду графічних зображень. Наприклад, Python не є найшвидшою мовою програмування, що може бути недостатньо для роботи з великими об'ємами зображень або для розв'язання завдань у реальному часі.

Крім того, Python має кілька бібліотек для розробки зображень, які можуть бути несумісними з деякими операційними системами або мати обмежені можливості в певних сценаріях.

Також слід враховувати, що Python не є мовою низького рівня, що може бути проблемою для деяких застосунків, які потребують максимальної продуктивності та управління ресурсами. Деякі операції, як-от обробка та перегляд великих обсягів зображень, можуть бути неприйнятними для Python, якщо не використовується оптимізований код або апаратне прискорення.

Ще одним недоліком Python є відносно невелика кількість спеціалізованих бібліотек для обробки конкретних типів зображень, наприклад, медичних зображень або зображень, отриманих з використанням дронів. Деякі бібліотеки можуть бути сильно спеціалізовані і не надавати загальних можливостей для роботи із зображеннями.

Крім того, Python може бути не таким ефективним, як інші мови програмування, якщо застосунок вимагає багатопотоковості або розпаралелювання. Це може означати, що обробка великих обсягів зображень може займати багато часу і ресурсів, що може бути неефективним для розв'язання деяких завдань.

Незважаючи на ці обмеження, Python все ще залишається однією з найкращих мов програмування для обробки та перегляду графічних зображень завдяки своїй багатій бібліотеці, доступності та простому синтаксису. Крім того, Python може бути ефективно використаний для

швидкого розроблення та прототипування додатків, що може бути особливо корисно для розробників-початківців.[4]

III. Проектування програми

1. Опис архітектури програми та її компонентів

Програма має клас ImageEditor, який ініціалізує головне вікно і надає всі необхідні функції для маніпулювання зображеннями. Клас ініціюється створенням об'єкта Tk, який є головним вікном програми.

Клас ImageEditor має наступні методи:

- `init`: Встановлює заголовок та іконку вікна, прив'язує клавішу Escape до методу _close та встановлює протокол виклику методу _close при закритті вікна.

- `run`: Запускає основний цикл роботи вікна та відображає меню і віджети.

- `draw_menu`: Малює рядок меню вікна з різними опціями, такими як відкрити, зберегти, перетворити, перевернути, змінити розмір, фільтрувати та обрізати.

- `draw_widgets`: Малює вкладки зображень і показує відкриті зображення.

- `open_new_images`: Відкриває діалогове вікно для вибору одного або декількох файлів зображень і викликає метод add_new_image для кожного вибраного файлу.

- `add_new_image`: Відкриває вибраний файл зображення за допомогою бібліотеки PIL і створює нову вкладку у вікні з вибраним зображенням. Він також відстежує відкриті зображення у списку opened_images.
- `save_current_image`: Зберігає зміни, внесені до поточного вибраного зображення.

- `save_image_as`: Зберігає поточне вибране зображення з новим іменем або типом файлу.

- `save_all_changes`: Зберегти всі зміни, зроблені у відкритих зображеннях.

- `close_current_image`: Закриває поточне вибране зображення та його вкладку.

- `delete_current_image`: Видаляє поточне вибране зображення та його вкладку.

- `move_current_image`: Переміщує поточне зображення на нове місце.

- `rotate_current_image`: Повертає поточне зображення на заданий кут.

- `flip_current_image`: Перевертає поточне зображення по горизонталі або вертикалі.

- `resize_current_image`: Змінює розмір поточного зображення на заданий відсоток від його початкового розміру.

- `apply_filter_to_current_image`: Застосовує заданий фільтр до поточного зображення.

- `start_area_selection_of_current_image`: Починає нове виділення області на поточному зображенні.
- `stop_area_selection_of_current_image`: Зупиняє виділення області та обрізає виділену область.

Програма використовує віджети:

- `tkinter.ttk.Notebook` для відображення кожного відкритого зображення на окремій вкладці.
- `tkinter.Canvas` для відображення вибраного зображення.

Програма також використовує клас `PIL.ImageTk.PhotoImage` для перетворення відкритих зображень у формат, який можна відобразити у віджеті `tkinter Canvas`.

2. Розробка інтерфейсу користувача.

Під час запуску програми створюється екземпляр класу ImageEditor, який має головне вікно, панель вкладок і список відкритих зображень. Також у конструкторі класу встановлюються початкові значення для змінних, що відповідають за виділену область на зображенні.

```
class ImageEditor:
    def __init__(self):
        self.root = Tk()
        self.image_tabs = Notebook(self.root)
        self.opened_images = []

        self.selection_top_x = 0
        self.selection_top_y = 0
        self.selection_bottom_x = 0
        self.selection_bottom_y = 0

        self.canvas_for_selection = None
        self.selection_rect = None

        self.init()
```

```
def init(self):
    self.root.title("ImageEditor")
    self.root.iconphoto(True, PhotoImage(file="resources/icon.png"))
    self.image_tabs.enable_traversal()

    self.root.bind("<Escape>", self._close)
    self.root.protocol("WM_DELETE_WINDOW", self._close)
```

Метод init() встановлює назву вікна, іконку, налаштування, пов'язані із закриттям вікна додатка та активацією вкладок за допомогою клавіш.

Метод draw_menu() відмальовує меню додатка. Воно містить пункти "File" і "Edit". Пункт "File" містить команди "Open", "Save", "Save as", "Save all", "Close Image", "Delete Image", "Move Image" і "Exit". Пункт "Edit" містить команди "Transform", "Flip", "Resize", "Filter" і "Crop". Деякі з цих команд мають вкладені меню з додатковими командами. Наприклад, меню "Transform" містить команди повороту, меню "Flip" - команди відображення, меню "Resize" - команди зміни розміру, меню "Filter" - команди застосування фільтрів, меню "Crop" - команди виділення області.

Для кожного пункту меню, пов'язаного зі зміною зображення, було створено відповідне меню з підменю, в якому містяться доступні дії. Також були написані методи, що реалізують функціонал кожної дії. Наприклад, для пункту меню "Transform -> Rotate" було створено підменю, що містить команди повороту зображення на 90, 180 і 270 градусів. При виборі однієї з команд викликається метод rotate_current_image з відповідним аргументом.

Для малювання інтерфейсу були використані віджети з бібліотеки tkinter, такі як Notebook, Frame, Button, Canvas та інші. Віджет Notebook використовувався для створення вкладок із зображеннями. Кожна вкладка

містить віджет Frame, на якому розташовані кнопки та Canvas для відображення зображення.

Нижче наведено приклад коду методу rotate_current_image, який повертає поточне зображення на заданий кут:

```
def rotate_current_image(self, angle):
    current_image = self.get_current_image()

    rotated_image = current_image.rotate(angle, expand=True)
    rotated_image_tk = ImageTk.PhotoImage(rotated_image)

    self.set_current_image(rotated_image_tk, rotated_image)
```

Метод отримує поточне зображення, повертає його на заданий кут і створює новий об'єкт ImageTk.PhotoImage з поверненим зображенням. Потім метод викликає метод set_current_image, який замінює поточне зображення на повернуте.

Загалом, програма надає зручний інтерфейс для роботи із зображеннями і дає змогу виконувати основні операції з їхньої зміни.

3. Опис основних функцій та можливостей програми

Основний клас ImageEditor містить низку методів для відкриття, редагування та збереження зображень.

Основні функції програми включають:

- Відкриття зображень: користувач може вибрати одне або кілька зображень із файлової системи та відкрити їх у програмі.
- Редагування зображень: доступні різні інструменти для редагування зображень, включно з поворотом, зміною розміру, віддзеркаленням, фільтрами та виділенням області.

- Збереження змін: користувач може зберегти зміни для окремого зображення, для всіх відкритих зображень або зберегти зображення під новим ім'ям.
- Закриття зображень: користувач може закрити окреме зображення або закрити всі відкриті зображення.
- Видалення та переміщення зображень: користувач може видалити або перемістити зображення у файловій системі, використовуючи відповідні опції в меню.
- Виділення області: користувач може виділити область зображення для подальшої обробки.
- Застосування фільтрів: доступні різні фільтри для обробки зображень, включаючи розмиття, збільшення різкості та інші.

Інтерфейс програми побудовано з використанням віджетів Tkinter, включно з меню, вкладками і полотнами для відображення зображень. Для роботи із зображеннями використовується бібліотека Python Imaging Library (PIL). Код написаний в об'єктно-орієнтованому стилі, використовуючи класи і методи для структурування та організації функціональності програми.

IV. Впровадження додатків

1. Основними моментами розробки цього додатка є:

- Імпорт необхідних модулів і бібліотек, таких як Tkinter, Pillow і shutil.

- Створення класу ImageEditor, який є основним класом додатка і містить усі його функції та методи.

- Ініціалізація кореневого вікна додатка та елемента Notebook, який використовується для зберігання відкритих зображень.

- Створення необхідних змінних, які будуть використовуватися в різних методах класу.

- Створення методів і функцій, які відповідають за відтворення графічного інтерфейсу користувача, обробку подій, відкриття та збереження зображень, а також застосування різних операцій до відкритих зображень.

- Реалізація методів для роботи з виділеною областю на зображенні, що використовуються для обрізки зображень.

- Використання методів і функцій з бібліотек Tkinter і Pillow для відображення та обробки зображень у додатку.

- Оброблення помилок і винятків, які можуть виникати під час роботи додатка.

- Тестування та налагодження додатка для забезпечення його коректної роботи та усунення помилок.

2. Детальний опис реалізації функцій та компонентів

Будь-який код починається з виклику бібліотек. Мій код програми не став винятком, тому під час розробки викликав різні бібліотеки для реалізацій тих функцій, які мені були потрібні. Ось детальне пояснення всіх бібліотек, які я викликав:

“from tkinter import *”: Імпортує всі модулі та функції з бібліотеки tkinter, яка використовується для створення графічних інтерфейсів користувача у Python.

“from tkinter import filedialog as fd”: Імпортує модуль filedialog з tkinter і надає йому псевдонім fd, який можна використовувати для доступу до його функцій.

“from tkinter import messagebox as mb”: Імпортує модуль messagebox з tkinter і надає йому псевдонім mb, який можна використовувати для доступу до його функцій.

“from tkinter.ttk import Notebook”: Імпортує віджет Notebook з модуля tkinter.ttk, який є набором тематичних класів віджетів для tkinter.

“from PIL import Image, ImageTk, ImageOps, ImageFilter”: Імпортує модуль Image з бібліотеки зображень Python (PIL), який використовується для маніпуляцій із зображеннями. Також імпортуються три його підмодулі: ImageTk (для відображення зображень у tkinter), ImageOps (для більш складних операцій над зображеннями) та ImageFilter (для застосування різних фільтрів зображень).

“import os”: Імпортує модуль os, який надає можливість взаємодії з операційною системою.

“import shutil”: Імпортує модуль shutil, який надає можливість працювати з високорівневими файловими операціями, такими як копіювання та видалення файлів.

Після виклику всіх потрібних бібліотек я створюю основний клас, у якому відбуватимуться всі основні функції програми, як-от створення інтерфейсу, редагування інтерфейсу. Після виклику всіх потрібних бібліотек я створюю основний клас, у якому відбуватимуться всі основні функції програми, як-от створення інтерфейсу, редагування інтерфейсу. Клас я назвав так само, як і називається моя програма "ImageEditor".

- def __init__(self):

У цьому класі створюється функція (init). Він являє собою метод-конструктор основного класу програми. За допомогою цього класса створюється вікно програми:

```
    def __init__(self):
        self.root = Tk()
        self.image_tabs = Notebook(self.root)
        self.opened_images = []

        self.selection_top_x = 0
        self.selection_top_y = 0
        self.selection_bottom_x = 0
        self.selection_bottom_y = 0

        self.canvas_for_selection = None
        self.selection_rect = None

        self.init()
```

Кожний рядок цього методу виконує такі дії:

"elf.root = Tk()" - створює екземпляр класу Tk, який буде кореневим вікном для додатка.

"self.image_tabs = Notebook(self.root)" - створює екземпляр класу Notebook з бібліотеки ttk і прив'язує його до кореневого вікна.

"self.opened_images = []" - створює порожній список для відстеження відкритих зображень.

"self.selection_top_x = 0, self.selection_top_y = 0, self.selection_bottom_x = 0, self.selection_bottom_y = 0" - встановлюють початкові координати для рамки виділення на зображенні (0, 0, 0, 0, 0).

"self.canvas_for_selection = None" - створює змінну для зберігання екземпляра класу Canvas, який буде використовуватися для малювання рамки виділення на зображенні.

"self.selection_rect = None" - створює змінну для зберігання id об'єкта прямокутника на Canvas, який буде використовуватися для видалення рамки виділення за необхідності.

"self.init()" - викликає метод init, який ініціалізує користувацький інтерфейс.

- def init(self):

Потім створюю функцію init(self), в якій налаштовую ім’я вікна, так її іконку. Також налаштовую програму, так, щоб вона зачинялась при нажатій клавіши Escape.

```
    def init(self):
        self.root.title("Image Editor")
        self.root.iconphoto(True,PhotoImage(file="resources/icon.png"))
        self.image_tabs.enable_traversal()

        self.root.bind("<Escape>", self._close)
        self.root.protocol("WM_DELETE_WINDOW", self._close)
```

Ось розбивка того, що робить кожен рядок:

“self.root.title("Image Editor")”: Встановлює заголовок головного вікна на "Редактор зображень".

“self.root.iconphoto(True,PhotoImage(file="resources/icon.png"))”: Встановлює іконку головного вікна на файл зображення, розташований за адресою "resources/icon.png".

“self.image_tabs.enable_traversal()”: Вмикає обхід (клавіатурну навігацію) для віджету блокнота, який буде містити зображення.

“self.root.bind("<Escape>", self._close)”: Прив'язує клавішу "Escape" до методу _close, який використовується для закриття програми.

“self.root.protocol("WM_DELETE_WINDOW", self._close)”: Прив'язує подію "WM_DELETE_WINDOW" до методу _close, який використовується для закриття програми, коли користувач натискає кнопку закриття вікна.

- def run(self):

Ця функція запускає головний цикл додатка, який обробляє події, поки користувач не закриє вікно.

```
    def run(self):
        self.draw_menu()
        self.draw_widgets()

        self.root.mainloop()
```

“self.draw_menu()” - викликає метод draw_menu(), який створює верхнє меню додатка.

“self.draw_widgets()” - викликає метод draw_widgets(), який створює віджети додатка на вкладках.

“self.root.mainloop()” - запускає головний цикл оброблення подій додатка й очікує, поки користувач не закриє вікно. Коли це відбувається, головний цикл завершується, і програма закінчує свою роботу.

- def draw_menu(self):

Цей метод створює рядок меню з різними опціями і каскадними підменю, а потім встановлює рядок меню для кореневого вікна.

```
    def draw_menu(self):
        menu_bar = Menu(self.root)

```

```
            file_menu = Menu(menu_bar, tearoff=0)
            file_menu.add_command(label="Open",
command=self.open_new_images)
            file_menu.add_command(label="Save",
command=self.save_current_image)
            file_menu.add_command(label="Save                    as",
command=self.save_image_as)
            file_menu.add_command(label="Save                    all",
command=self.save_all_changes)
            file_menu.add_separator()
            file_menu.add_command(label="Close                  Image",
command=self.close_current_image)
            file_menu.add_separator()
            file_menu.add_command(label="Delete                 Image",
command=self.delete_current_image)
          file_menu.add_command(label="Move                     Image",
command=self.move_current_image)
            file_menu.add_separator()
            file_menu.add_command(label="Exit", command=self._close)
            menu_bar.add_cascade(label="File", menu=file_menu)

            edit_menu = Menu(menu_bar, tearoff=0)
            transform_menu = Menu(edit_menu, tearoff=0)

            rotate_menu = Menu(transform_menu, tearoff=0)
           rotate_menu.add_command(label="Rotate      left      by      90",
command=lambda: self.rotate_current_image(90))
            rotate_menu.add_command(label="Rotate      right      by      90",
command=lambda: self.rotate_current_image(-90))
            rotate_menu.add_command(label="Rotate      left      by      180",
command=lambda: self.rotate_current_image(180))
            rotate_menu.add_command(label="Rotate      right      by      180",
command=lambda: self.rotate_current_image(-180))
```

```
            transform_menu.add_cascade(label="Rotate", menu=rotate_menu)

            flip_menu = Menu(edit_menu, tearoff=0)
                flip_menu.add_command(label="Flip horizontally",
command=lambda: self.flip_current_image("horizontally"))
            flip_menu.add_command(label="Flip vertically", command=lambda:
self.flip_current_image("vertically"))

            resize_menu = Menu(edit_menu, tearoff=0)
            resize_menu.add_command(label="25% of original size",
command=lambda: self.resize_current_image(25))
            resize_menu.add_command(label="50% of original size",
command=lambda: self.resize_current_image(50))
            resize_menu.add_command(label="75% of original size",
command=lambda: self.resize_current_image(75))
            resize_menu.add_command(label="125% of original size",
command=lambda: self.resize_current_image(125))
            resize_menu.add_command(label="150% of original size",
command=lambda: self.resize_current_image(150))
            resize_menu.add_command(label="200% of original size",
command=lambda: self.resize_current_image(200))

            filter_menu = Menu(edit_menu, tearoff=0)
            filter_menu.add_command(label="Blur", command=lambda:
self.apply_filter_to_current_image(ImageFilter.BLUR))
            filter_menu.add_command(label="Sharpen", command=lambda:
self.apply_filter_to_current_image(ImageFilter.SHARPEN))
            filter_menu.add_command(label="Contour", command=lambda:
self.apply_filter_to_current_image(ImageFilter.CONTOUR))
            filter_menu.add_command(label="Detail", command=lambda:
self.apply_filter_to_current_image(ImageFilter.DETAIL))
            filter_menu.add_command(label="Smooth", command=lambda:
self.apply_filter_to_current_image(ImageFilter.SMOOTH))
```

```
        crop_menu = Menu(edit_menu, tearoff=0)
        crop_menu.add_command(label="Start selection",
command=self.start_area_selection_of_current_image)
        crop_menu.add_command(label="Stop selection",
command=self.stop_area_selection_of_current_image)

        edit_menu.add_cascade(label="Transform", menu=transform_menu)
        edit_menu.add_cascade(label="Flip", menu=flip_menu)
        edit_menu.add_cascade(label="Resize", menu=resize_menu)
        edit_menu.add_cascade(label="Filter", menu=filter_menu)
        edit_menu.add_cascade(label="Crop", menu=crop_menu)

        menu_bar.add_cascade(label="Edit", menu=edit_menu)

        self.root.configure(menu=menu_bar)
```

“menu_bar = Menu(self.root)”: Створює новий об'єкт рядка меню для кореневого вікна.

“file_menu = Menu(menu_bar, tearoff=0)”: Створює новий об'єкт меню для файлового меню і прикріпити його до рядка меню. tearoff=0 використовується для запобігання відриву меню.

“file_menu.add_command(...)” і наступні рядки: Додають різні пункти меню до об'єкта меню файлу з відповідними командами, які будуть виконані при натисканні на них.

“menu_bar.add_cascade(label="File", menu=file_menu)”: Додає меню файлів до рядка меню з міткою "Файл".

Ця ж процедура повторюється для меню редагування з різноманітними підменю, включно з меню перетворення, перевертання, зміни розміру, фільтрації та обрізання.

“self.root.configure(menu=menu_bar)”: Встановлює панель меню як меню кореневого вікна.

- def draw_widgets(self):

Цей метод створює вікно графічного інтерфейсу, яке міститиме закладки для кожного відкритого зображення.

```
    def draw_widgets(self):
        self.image_tabs.pack(fill="both", expand=1)
```

Рядок self.image_tabs.pack(fill="both", expand=1) розміщує закладки на всій доступній площі вікна, щоб вони могли займати якомога більше місця. fill вказує, яким чином віджет має заповнювати вільний простір в батьківському вікні, а expand дозволяє розтягнути віджет, якщо доступна площа більша, ніж його поточний розмір.

- def open_new_images(self):

```
    def open_new_images(self):
        image_paths = fd.askopenfilenames(filetypes=(("Images",
"*.jpeg;*.jpg;*.png"), ))
        for image_path in image_paths:
            self.add_new_image(image_path)
```

Цей код визиває метод askopenfilenames() з модуля filedialog для відкриття вікна діалогу вибору файлів. Він приймає аргумент filetypes, який задає фільтри для файлів, які можна вибирати у вікні діалогу. В даному випадку фільтр дозволяє вибирати файли з розширеннями .jpeg, .jpg та .png з назвою "Images".

Після вибору файлів зображень у вікні діалогу, шляхи до цих файлів зберігаються у змінній image_paths, яка містить список шляхів до обраних файлів.

Далі код виконує цикл for, який проходить по кожному шляху image_path у списку image_paths та викликає метод add_new_image() з

аргументом image_path. Це допомагає додати кожне зображення з вибраних файлів до програми.

- def add_new_image(self, path_image):

Це метод add_new_image, який отримує два параметри: self та шлях до зображення.

```
    def add_new_image(self, image_path):
        image = Image.open(image_path)
        image_tk = ImageTk.PhotoImage(image)
        self.opened_images.append([image_path, image])

        image_tab = Frame(self.image_tabs)

        image_panel = Canvas(image_tab, width=image_tk.width(),
height=image_tk.height(), bd=0, highlightthickness=0)
        image_panel.image = image_tk
        image_panel.create_image(0, 0, image=image_tk, anchor="nw")
        image_panel.pack(expand="yes")

        self.image_tabs.add(image_tab, text=os.path.split(image_path)[1])
        self.image_tabs.select(image_tab)
```

“image = Image.open(image_path)”: Цей рядок відкриває файл зображення за адресою image_path за допомогою класу Image з бібліотеки зображень PIL (Python Imaging Library) і присвоює його змінній image.

“image_tk = ImageTk.PhotoImage(image)”: Цей рядок перетворює об'єкт PIL Image image в об'єкт tkinter PhotoImage image_tk. Це перетворення необхідне для відображення зображення у віджеті tkinter.

“self.opened_images.append([image_path, image])”: Цей рядок додає список, що містить шлях до зображення та зображення, до списку opened_images, який відстежує всі зображення, що були відкриті у програмі.

“image_tab = Frame(self.image_tabs)”: Цей рядок створює новий віджет Frame з назвою image_tab, який міститиме панель зображень.

“image_panel = Canvas(image_tab, width=image_tk.width(), height=image_tk.height(), bd=0, highlightthickness=0)”: Цей рядок створює новий віджет Canvas з назвою image_panel у фреймі image_tab. Ширина та висота полотна встановлюються відповідно до розмірів об'єкта image_tk, а bd=0 та highlightthickness=0 прибирають будь-які межі та підсвічування навколо полотна.

“image_panel.image = image_tk”: Цей рядок створює атрибут з назвою image на полотні image_panel і присвоює йому значення об'єкта image_tk. Це необхідно для того, щоб об'єкт image_tk не був зібраний у сміття.

“image_panel.create_image(0, 0, image=image_tk, anchor="nw")”: Цей рядок створює новий об'єкт PhotoImage на полотні image_panel за допомогою методу create_image(). Зображення розміщується у координатах (0, 0) з якорем, встановленим на "nw" (північний захід), що вирівнює верхній лівий кут зображення з координатами (0, 0).

“image_panel.pack(expand="yes")”: Цей рядок пакує віджет Canvas віджету image_panel всередину фрейму image_tab. Аргумент expand="yes" вказує tkinter розгорнути віджет, щоб заповнити будь-який доступний простір.

“self.image_tabs.add(image_tab, text=os.path.split(image_path)[1])”: Цей рядок додає віджет image_tab Frame до віджета image_tabs Notebook з міткою табуляції, яка показує назву файлу зображення. Вираз os.path.split(image_path)[1] витягує частину рядка image_path, що містить назву файлу.

“self.image_tabs.select(image_tab)”: Цей рядок вибирає віджет image_tab Frame у віджеті image_tabs Блокнот, який показує користувачеві панель зображень.

- def get_current_working_data(self):

У цьому коді визначено метод get_current_working_data, який не отримує аргументів.

```
    def get_current_working_data(self):
        """returns current (tab, image, path)
        """
        current_tab = self.image_tabs.select()
        if not current_tab:
            return None, None, None
        tab_number = self.image_tabs.index(current_tab)
        path, image = self.opened_images[tab_number]

        return current_tab, path, image
```

""" returns current (tab, image, path)""": Це документальний рядок, який описує призначення методу.

"current_tab = self.image_tabs.select()": Цей рядок отримує поточну вибрану вкладку в об'єкті self.image_tabs і зберігає її у змінній current_tab.

"if not current_tab": Цей рядок перевіряє, чи current_tab дорівнює None (тобто, наразі не вибрано жодної вкладки). Якщо це так, метод повертає None для всіх трьох значень: табуляції, зображення та шляху.

"tab_number = self.image_tabs.index(current_tab)": Цей рядок отримує індекс поточної вибраної вкладки і зберігає його у змінній tab_number.

"path, image = self.opened_images[tab_number]": Цей рядок отримує шлях та зображення, пов'язані з вибраною вкладкою, зі списку self.opened_images, використовуючи змінну tab_number як індекс. Значення розпаковуються і зберігаються у змінних path та image.

"return current_tab, path, image": Цей рядок повертає кортеж, що містить поточну вибрану вкладку, шлях до неї та її зображення.

- def save_current_image(self):

Цей метод save_current_image() зберігає вибране відображення зображення в переглядачі зображень на диск.

```
    def save_current_image(self):
        current_tab, path, image = self.get_current_working_data()
        if not current_tab:
            return
        tab_number = self.image_tabs.index(current_tab)

        if path[-1] == '*':
            path = path[:-1]
            self.opened_images[tab_number][0] = path
            image.save(path)
            self.image_tabs.add(current_tab, text=os.path.split(path)[1])
```

Ось опис кожного рядка коду:

"current_tab, path, image = self.get_current_working_data()": цей рядок викликає метод get_current_working_data(), щоб отримати шлях та об'єкт зображення поточно вибраного зображення.

"if not current_tab": цей рядок перевіряє, чи вибрано поточну вкладку. Якщо ні, він повертається з методу.

"tab_number = self.image_tabs.index(current_tab)": цей рядок отримує індекс поточної вкладки в списку відкритих зображень.

"if path[-1] == '*'": цей рядок перевіряє, чи шлях закінчується знаком астериск, що означає, що зображення раніше не було збережене.

"path = path[:-1]": цей рядок видаляє знак астериск з кінця шляху.

"self.opened_images[tab_number][0] = path": цей рядок оновлює шлях відкритого зображення новим шляхом.

"image.save(path)": цей рядок зберігає зображення за новим шляхом.

"self.image_tabs.add(current_tab, text=os.path.split(path)[1])": цей рядок оновлює текст вкладки іменем файлу.

- def save_image_as(self):

Цей метод save_image_as() зберігає обране зображення у вікні перегляду зображень на диск.

```
    def save_image_as(self):
        current_tab, path, image = self.get_current_working_data()
        if not current_tab:
            return
        tab_number = self.image_tabs.index(current_tab)

        old_path, old_ext = os.path.splitext(path)
        if '*' in old_ext:
            old_ext = old_ext[:-1]

        new_path = fd.asksaveasfilename(initialdir=old_path, filetypes=(("Images", "*.jpeg;*.jpg;*.png"), ))
        if not new_path:
            return

        new_path, new_ext = os.path.splitext(new_path)
        if not new_ext:
            new_ext = old_ext
        elif old_ext != new_ext:
            mb.showerror("Incorrect extension", f"Got incorrect extension: {new_ext}. Old was: {old_ext}")
            return

        image.save(new_path + new_ext)
        image.close()

        del self.opened_images[tab_number]
        self.image_tabs.forget(current_tab)

        self.add_new_image(new_path + new_ext)
```

Опис кожного рядка коду:

"current_tab, path, image = self.get_current_working_data()": цей рядок викликає метод get_current_working_data(), щоб отримати шлях та об'єкт зображення обраного зображення.

"if not current_tab": цей рядок перевіряє, чи вибрано поточну вкладку. Якщо ні, то він повертається з методу.

"tab_number = self.image_tabs.index(current_tab)": цей рядок отримує індекс поточної вкладки в списку відкритих зображень.

"old_path, old_ext = os.path.splitext(path)": цей рядок отримує шлях та розширення зображення, що було відкрите.

"if '*' in old_ext": цей рядок перевіряє, чи містить розширення зірочку, що вказує на те, що зображення не було збережене раніше.

"old_ext = old_ext[:-1]": цей рядок видаляє символ зірочки з кінця розширення.

"new_path = fd.asksaveasfilename(initialdir=old_path, filetypes=(("Images", ".jpeg;.jpg;*.png"),))": цей рядок запускає вікно діалогу вибору файлу для вибору нового шляху та набору допустимих розширень файлів.

"if not new_path": цей рядок перевіряє, чи був вибраний новий шлях. Якщо ні, то метод повертається.

"new_path, new_ext = os.path.splitext(new_path)": цей рядок отримує шлях та розширення нового зображення.

"if not new_ext": цей рядок перевіряє, чи має нове зображення розширення. Якщо ні, то він встановлює розширення, яке було у попередньому зображенні.

"elif old_ext != new_ext": цей рядок перевіряє, чи розширення нового зображення відрізняється від розширення попереднього зображення.

"image.save(new_path + new_ext)": Цей рядок зберігає зображення в новому шляху з новим розширенням.

"image.close()": Цей рядок закриває старе зображення.

"del self.opened_images[tab_number]": Цей рядок видаляє інформацію про старе зображення зі списку відкритих зображень.

"self.image_tabs.forget(current_tab)": Цей рядок закриває поточну вкладку з зображенням.

"self.add_new_image(new_path + new_ext)": Цей рядок додає нове зображення з новим шляхом до списку відкритих зображень і створює нову вкладку з зображенням в графічному інтерфейсі.

- def save_all_changes(self):

Цей метод save_all_changes() зберігає всі зміни зображень, які є відкритими в програмі.

```
def save_all_changes(self):
        for index, (path, image) in enumerate(self.opened_images):
            if path[-1] != '*':
                continue
            path = path[:-1]
            self.opened_images[index][0] = path
            image.save(path)
            self.image_tabs.tab(index, text=os.path.split(path)[1])
```

Ось опис кожного рядка коду:

"for index, (path, image) in enumerate(self.opened_images)": - цикл, який проходить по кожному відкритому зображенню в програмі, отримуючи індекс зображення, шлях до нього і об'єкт зображення.

"if path[-1] != '*'": - перевіряє, чи в останньому символі шляху до зображення є символ зірочки, яка вказує на те, що зображення не було збережено раніше. Якщо немає, це зображення пропускається і переходить до наступного.

"path = path[:-1]" - видаляє символ зірочки з останнього символу шляху.

"self.opened_images[index][0] = path" - оновлює шлях для зображення в списку відкритих зображень з новим шляхом.

"image.save(path)" - зберігає зображення з оновленим шляхом.

"self.image_tabs.tab(index, text=os.path.split(path)[1])" - оновлює текст вкладки зображення в графічному інтерфейсі програми на ім'я файлу, яке відображається в останній частині шляху до зображення.

- def close_current_image(self):

Цей метод close_current_image() закриває відображуване в даний момент зображення в переглядачі зображень та видаляє його зі списку відкритих зображень.

```
def close_current_image(self):
    current_tab, path, image = self.get_current_working_data()
    if not current_tab:
        return
    index = self.image_tabs.index(current_tab)

    image.close()
    del self.opened_images[index]
    self.image_tabs.forget(current_tab)
```

Ось опис кожного рядка коду:

"current_tab, path, image = self.get_current_working_data()": цей рядок викликає метод get_current_working_data(), щоб отримати шлях до вибраного в даний момент зображення та об'єкт зображення.

"if not current_tab": цей рядок перевіряє, чи вибрана поточна вкладка. Якщо ні, метод повертається.

"index = self.image_tabs.index(current_tab)": цей рядок отримує індекс поточної вкладки в списку відкритих зображень.

"image.close()": цей рядок закриває об'єкт зображення.

"del self.opened_images[index]": цей рядок видаляє зображення зі списку відкритих зображень за вказаним індексом.

"self.image_tabs.forget(current_tab)": цей рядок видаляє поточну вкладку з переглядача зображень.

- def delete_current_image(self):

Цей метод delete_current_image() видаляє обране зображення в переглядачі зображень і знімає його зі списку відкритих зображень.

```
    def delete_current_image(self):
        current_tab, path, image = self.get_current_working_data()
        if not current_tab:
            return
        index = self.image_tabs.index(current_tab)

        if not mb.askokcancel("Delete image", "Are you sure you want to delete image?\nThis operation is unrecoverable!"):
            return

        image.close()
        os.remove(path)

        del self.opened_images[index]
        self.image_tabs.forget(current_tab)
```

Ось опис кожного рядка коду:

"current_tab, path, image = self.get_current_working_data()": цей рядок викликає метод get_current_working_data(), щоб отримати шлях та об'єкт зображення обраного в даний момент зображення.

"if not current_tab": цей рядок перевіряє, чи обрано в даний момент вкладку. Якщо ні, метод повертається.

"index = self.image_tabs.index(current_tab)": цей рядок отримує індекс поточної вкладки в списку відкритих зображень.

"if not mb.askokcancel("Delete image", "Are you sure you want to delete image?\nThis operation is unrecoverable!")": цей рядок показує діалогове вікно, щоб підтвердити видалення зображення. Якщо користувач не підтвердить, метод повертається.

"image.close()": цей рядок закриває об'єкт зображення.

"os.remove(path)": цей рядок видаляє файл зображення за вказаним шляхом.

"del self.opened_images[index]": цей рядок видаляє зображення зі списку відкритих зображень за вказаним індексом.

"self.image_tabs.forget(current_tab)": цей рядок видаляє поточну вкладку з переглядача зображень.

- def move_current_image(self):

Цей метод move_current_image() переміщає вибране в даний момент зображення у вибрану директорію та оновлює шлях до зображення.

```
def move_current_image(self):
    current_tab, old_path, image = self.get_current_working_data()
    if not current_tab:
        return
    index = self.image_tabs.index(current_tab)

    new_dir = fd.askdirectory(initialdir=os.path.dirname(old_path))
    if not new_dir:
        return

    new_path = os.path.join(new_dir, os.path.split(old_path)[1])

    image.close()
    shutil.move(old_path, new_path)

    del self.opened_images[index]
    self.image_tabs.forget(current_tab)
```

```
        self.add_new_image(new_path)
```

Ось опис кожного рядка коду:

"current_tab, old_path, image = self.get_current_working_data()": цей рядок викликає метод get_current_working_data(), щоб отримати поточний шлях та об'єкт зображення вибраного зображення.

"if not current_tab": цей рядок перевіряє, чи вибрано поточну вкладку. Якщо ні, то метод повертається.

"index = self.image_tabs.index(current_tab)": цей рядок отримує індекс поточної вкладки в списку відкритих зображень.

"new_dir = fd.askdirectory(initialdir=os.path.dirname(old_path))": цей рядок відкриває діалогове вікно вибору директорії, щоб вказати нову директорію для переміщення зображення. initialdir - це початкова директорія діалогового вікна, яка встановлюється як батьківська директорія вибраного зображення.

"if not new_dir": цей рядок перевіряє, чи користувач вибрав нову директорію. Якщо ні, метод повертається.

"new_path = os.path.join(new_dir, os.path.split(old_path)[1])": цей рядок створює новий шлях до зображення у вибраній директорії, зберігаючи оригінальне ім'я файлу.

"image.close()": цей рядок закриває об'єкт зображення.

"shutil.move(old_path, new_path)": цей рядок переміщує зображення до нової директорії.

"del self.opened_images[index]": цей рядок видаляє зображення зі списку відкритих зображень за вибраним індексом.

"self.image_tabs.forget(current_tab)": цей рядок видаляє поточну вкладку з переглядача зображень.

"self.add_new_image(new_path)": цей рядок додає нове зображення до списку відкритих зображень та відкриває його в новій вкладці.

- def update_image_inside_app(self, current_tab, image):

Цей код оновлює відображення зображення в програмі, якщо зображення було змінено.

```
    def update_image_inside_app(self, current_tab, image):
        tab_number = self.image_tabs.index(current_tab)
        tab_frame = self.image_tabs.children[current_tab[current_tab.rfind('!'):]]
        canvas = tab_frame.children['!canvas']

        self.opened_images[tab_number][1] = image

        image_tk = ImageTk.PhotoImage(image)

        canvas.delete("all")
        canvas.image = image_tk
        canvas.configure(width=image_tk.width(), height=image_tk.height())
        canvas.create_image(0, 0, image=image_tk, anchor="nw")

        image_path = self.opened_images[tab_number][0]
        if image_path[-1] != '*':
            image_path += '*'
            self.opened_images[tab_number][0] = image_path
            image_name = os.path.split(image_path)[1]
            self.image_tabs.tab(current_tab, text=image_name)
```

Ось опис кожного рядка коду:

"tab_number = self.image_tabs.index(current_tab)" - цей рядок отримує індекс відповідного вкладки зображення в батьківському вікні.

"tab_frame = self.image_tabs.children[current_tab[current_tab.rfind('!'):]]" - цей рядок отримує рамку відповідної вкладки зображення.

"canvas = tab_frame.children['!canvas']" - цей рядок отримує канву відповідної вкладки зображення.

"self.opened_images[tab_number][1] = image" - цей рядок оновлює зображення в списку відкритих зображень.

"image_tk = ImageTk.PhotoImage(image)" - цей рядок створює новий об'єкт зображення для відображення на канві з використанням нового зображення.

"canvas.delete("all")" - цей рядок видаляє всі елементи з канви.

"canvas.image = image_tk" - цей рядок зберігає нове зображення як атрибут image об'єкта канви.

"canvas.configure(width=image_tk.width(), height=image_tk.height())" - цей рядок налаштовує розмір канви на відповідний розмір нового зображення.

"canvas.create_image(0, 0, image=image_tk, anchor="nw")" - цей рядок створює новий елемент зображення на канві з використанням нового зображення.

"image_path = self.opened_images[tab_number][0]" - цей рядок отримує шлях до зображення відповідної вкладки зображення.

"if image_path[-1] != '*'": - цей рядок перевіряє, чи зображення не мітиться як змінене.

"image_path += '*'" - цей рядок позначає зображення як змінене.

"self.opened_images[tab_number][0] = image_path" - цей рядок оновлює шлях до зображення в списку відкритих зображень.

"image_name = os.path.split(image_path)[1]" - цей рядок отримує ім'я файлу зображення з його шляху.

- def rotate_current_image(self, degrees):

Ця функція повертає зображення з обернутим на певний кут (вказаний у градусах) відображенням поточної вкладки в програмі.

```
    def rotate_current_image(self, degrees):
        current_tab, path, image = self.get_current_working_data()
        if not current_tab:
            return
```

```
        image = image.rotate(degrees)
        self.update_image_inside_app(current_tab, image)
```

Отже, кожен рядок коду виконує наступне:

"Параметр degrees" - кут обертання зображення на певну кількість градусів.

"current_tab, path, image = self.get_current_working_data()" - отримуємо поточну вкладку, її шлях та об'єкт зображення.

"if not current_tab: return" - якщо поточна вкладка не знайдена, функція повертає назад без виконання наступних рядків.

"image = image.rotate(degrees)" - повертає нове зображення, яке повернуто на вказаний кут.

"self.update_image_inside_app(current_tab, image)" - оновлює відображення вкладки в програмі з використанням нового обернутого зображення.

- def flip_current_image(self, flip_type):

Це метод, який обертає поточне зображення горизонтально або вертикально.

```
    def flip_current_image(self, flip_type):
        current_tab, path, image = self.get_current_working_data()
        if not current_tab:
            return

        if flip_type == "horizontally":
            image = ImageOps.mirror(image)
        elif flip_type == "vertically":
            image = ImageOps.flip(image)

        self.update_image_inside_app(current_tab, image)
```

"current_tab, path, image = self.get_current_working_data()": Цей рядок викликає інший метод get_current_working_data(), який повертає поточну вкладку, шлях та об'єкти зображення, з якими працює користувач. Ці значення призначаються змінним current_tab, path і image відповідно.

"if not current_tab: return": Якщо користувач не працює з жодним зображенням (немає відкритої вкладки), метод повертається без виконання будь-яких дій.

"if flip_type == "horizontally":": Цей рядок перевіряє, чи користувач хоче обернути зображення горизонтально. Якщо так, виконується наступний рядок, в іншому випадку виконується оператор elif.

"image = ImageOps.mirror(image)": Цей рядок використовує метод ImageOps.mirror() для обертання зображення горизонтально.

"elif flip_type == "vertically":": Цей рядок перевіряє, чи користувач хоче обернути зображення вертикально.

"image = ImageOps.flip(image)": Цей рядок використовує метод ImageOps.flip() для обертання зображення вертикально.

"self.update_image_inside_app(current_tab, image)": Нарешті, цей рядок викликає метод update_image_inside_app() для оновлення зображення, яке відображається на полотні, із новим обернутим зображенням.

- def resize_current_image(self, percents):

Цей метод змінює розмір поточного зображення на задану кількість відсотків.

```
    def resize_current_image(self, percents):
        current_tab, path, image = self.get_current_working_data()
        if not current_tab:
            return

        w, h = image.size
        w = (w * percents) // 100
```

```
        h = (h * percents) // 100

        image = image.resize((w, h), Image.ANTIALIAS)
        self.update_image_inside_app(current_tab, image)
```

"current_tab, path, image = self.get_current_working_data()": Цей рядок коду викликає інший метод get_current_working_data(), який повертає поточну вкладку, шлях та об'єкти зображення, з якими користувач працює. Ці значення призначаються змінним current_tab, path та image відповідно.

"if not current_tab: return": Якщо користувач не працює з жодним зображенням (жодна вкладка не відкрита), метод повертається без виконання будь-яких дій.

"w, h = image.size": Цей рядок коду отримує ширину та висоту зображення.

"w = (w * percents) // 100": Цей рядок коду змінює ширину зображення на відсоток, вказаний у змінній percents.

"h = (h * percents) // 100": Цей рядок коду змінює висоту зображення на відсоток, вказаний у змінній percents.

"image = image.resize((w, h), Image.ANTIALIAS)": Цей рядок коду змінює розмір зображення на вказаний розмір (за допомогою методу resize()) та зберігає зображення в змінній image.

"self.update_image_inside_app(current_tab, image)": Нарешті, цей рядок коду викликає метод update_image_inside_app() для оновлення зображення, відображеного на полотні, з новим зміненим зображенням.

- def apply_filter_to_current_image(self, filter_type):

Цей метод застосовує фільтр до поточного зображення.

```
    def apply_filter_to_current_image(self, filter_type):
        current_tab, path, image = self.get_current_working_data()
        if not current_tab:
            return
```

```
        image = image.filter(filter_type)
        self.update_image_inside_app(current_tab, image)
```

"current_tab, path, image = self.get_current_working_data()": Цей рядок викликає інший метод get_current_working_data(), який повертає поточну вкладку, шлях і об'єкти зображення, з якими працює користувач. Ці значення присвоюються змінним current_tab, path та image відповідно.

"if not current_tab: return": Якщо користувач не працює з жодним зображенням (жодна вкладка не відкрита), метод повертається без виконання будь-яких дій.

"image = image.filter(filter_type)": Цей рядок застосовує фільтр до зображення за допомогою методу filter().

"self.update_image_inside_app(current_tab, image)": Нарешті, цей рядок викликає метод update_image_inside_app() для оновлення зображення, яке відображається на полотні, з новим фільтрованим зображенням.

- def start_area_selection_of_current_image(self):

Цей код містить метод start_area_selection_of_current_image(), який розпочинає виділення області на поточному зображенні. Ось опис кожного рядка коду:

```
    def start_area_selection_of_current_image(self):
        current_tab = self.image_tabs.select()
        if not current_tab:
            return
        tab_frame = self.image_tabs.children[current_tab[current_tab.rfind('!'):]]
        canvas = tab_frame.children['!canvas']

        self.canvas_for_selection = canvas
        self.selection_rect = canvas.create_rectangle(
            self.selection_top_x, self.selection_top_y,
```

```
            self.selection_bottom_x, self.selection_bottom_y,
            dash=(10, 10), fill='', outline="white", width=2
        )

        canvas.bind("<Button-1>", self.get_selection_start_pos)
        canvas.bind("<B1-Motion>", self.update_selection_end_pos)
```

"current_tab = self.image_tabs.select()": Цей рядок отримує поточну вкладку, яку користувач вибрав, і зберігає її у змінну current_tab.

"if not current_tab: return": Якщо користувач не вибрав жодної вкладки, метод повертає нічого не роблячи.

"tab_frame = self.image_tabs.children[current_tab[current_tab.rfind('!'):]]": Цей рядок отримує рамку (frame) вкладки, на якій зображення буде відображатись, і зберігає її у змінну tab_frame. Він знаходить індекс останнього символу рядка current_tab, що містить "!", і використовує його для отримання імені дочірнього віджета.

"canvas = tab_frame.children['!canvas']": Цей рядок отримує об'єкт Canvas, на якому зображення буде відображатись, і зберігає його у змінну canvas.

"self.canvas_for_selection = canvas": Цей рядок зберігає об'єкт Canvas у змінну canvas_for_selection, щоб його можна було звернутися до нього з інших методів.

"self.selection_rect = canvas.create_rectangle(...": Цей рядок створює прямокутник для виділення області. Він використовує координати, які зберігаються у змінних self.selection_top_x, self.selection_top_y, "self.selection_bottom_x" та "self.selection_bottom_y" Використовує пунктирну лінію, щоб позначити прямокутник, та білий колір для його контуру.

- def get_selection_start_pos(self, event):

Цей метод встановлює початкову позицію вибраної області для подальшого вибору на зображенні.

```
    def get_selection_start_pos(self, event):
        self.selection_top_x, self.selection_top_y = event.x, event.y
```

"self.selection_top_x, self.selection_top_y = event.x, event.y" - у цьому рядку коду встановлюється значення координат місця, де користувач клікнув на зображенні, як початкова точка вибору області. event.x та event.y - це координати, що повертаються при кліку на область, на яку наведен курсор миші. Ці координати встановлюються в змінні self.selection_top_x та self.selection_top_y.

- def update_selection_end_pos(self, event):

Цей метод відповідає за оновлення кінцевої позиції області вибору на зображенні, коли користувач переміщує мишу.

```
    def update_selection_end_pos(self, event):
        self.selection_bottom_x, self.selection_bottom_y = event.x, event.y
        if self.canvas_for_selection is not None and self.selection_rect is not None:
            self.canvas_for_selection.coords(
                self.selection_rect,
                self.selection_top_x, self.selection_top_y,
                self.selection_bottom_x, self.selection_bottom_y
            )
```

"self.selection_bottom_x, self.selection_bottom_y = event.x, event.y": цей рядок присвоює значення позиції миші (event.x та event.y) до змінних, що відповідають за кінцеву позицію області вибору (selection_bottom_x та selection_bottom_y).

"if self.canvas_for_selection is not None and self.selection_rect is not None": це умова перевіряє, чи змінні canvas_for_selection та selection_rect не є пустими.

"self.canvas_for_selection.coords": цей рядок коду оновлює координати рамки, що відповідає за виділену область на зображенні, з використанням методу coords() для відповідного canvas. В методі coords() передаються аргументи: ID виділеної рамки (selection_rect), координати точки верхнього лівого кута (selection_top_x та selection_top_y) і координати точки нижнього правого кута (selection_bottom_x та selection_bottom_y) області вибору.

- def stop_area_selection_of_current_image(self):

Цей метод зупиняє вибір області на поточному зображенні та виконує декілька інших операцій.

```
def stop_area_selection_of_current_image(self):
        self.canvas_for_selection.unbind("<Button-1>")
        self.canvas_for_selection.unbind("<B1-Motion>")

        self.canvas_for_selection.delete(self.selection_rect)

        self.crop_current_image()

        self.selection_rect = None
        self.canvas_for_selection = None
        self.selection_top_x, self.selection_top_y = 0, 0
        self.selection_bottom_x, self.selection_bottom_y = 0, 0
```

"self.canvas_for_selection.unbind("<Button-1>")": Цей рядок відв'язує зв'язок між обраним канвасом та обробником подій, який відповідає за обробку подій кліків на кнопку миші.

"self.canvas_for_selection.unbind("<B1-Motion>")": Цей рядок відв'язує зв'язок між обраним канвасом та обробником подій, який відповідає за обробку подій переміщення миші.

"self.canvas_for_selection.delete(self.selection_rect)": Цей рядок видаляє прямокутник вибраної області з канвасу.

"self.crop_current_image()": Цей рядок викликає метод crop_current_image() для обрізки поточного зображення відповідно до вибраної області.

"self.selection_rect = None": Цей рядок встановлює змінну selection_rect в None, оскільки область вибору була знята.

"self.canvas_for_selection = None": Цей рядок встановлює змінну canvas_for_selection в None, оскільки область вибору була знята.

"self.selection_top_x, self.selection_top_y = 0, 0": Цей рядок встановлює змінні selection_top_x та selection_top_y в 0, оскільки область вибору була знята.

"self.selection_bottom_x, self.selection_bottom_y = 0, 0": Цей рядок встановлює змінні selection_bottom_x та selection_bottom_y в 0, оскільки область вибору була знята.

- def crop_current_image(self):

Це метод, що обрізає поточне зображення до прямокутної області виділення, що була визначена користувачем.

```
def crop_current_image(self):
    current_tab, path, image = self.get_current_working_data()
    if not current_tab:
        return

    image = image.crop((
        self.selection_top_x, self.selection_top_y,
        self.selection_bottom_x, self.selection_bottom_y
    ))

    self.update_image_inside_app(current_tab, image)
```

"def crop_current_image(self)": - це визначення методу crop_current_image() з одним аргументом self.

"current_tab, path, image = self.get_current_working_data()" - цей рядок отримує поточну вкладку, шлях та об'єкт зображення, з яким користувач працює, викликаючи метод get_current_working_data().

"if not current_tab: return" - ця умовна конструкція перевіряє, чи існує відкрита вкладка з зображенням. Якщо немає, метод завершується і повертається назад.

"image = image.crop((self.selection_top_x, self.selection_top_y, self.selection_bottom_x, self.selection_bottom_y))" - цей рядок обрізає поточне зображення до області, виділеної користувачем, використовуючи метод crop() об'єкта зображення та передаючи кортеж з координатами області виділення.

"self.update_image_inside_app(current_tab, image)" - цей рядок викликає метод update_image_inside_app(), щоб оновити зображення на вкладці з новим обрізаним зображенням.

- def unsaved_images(self):

Цей метод перевіряє, чи є відкриті зображення, які ще не були збережені. Ось опис кожного рядка коду:

```
    def unsaved_images(self):
        for path, _ in self.opened_images:
            if path[-1] == "*":
                return True
        return False
```

"for path, _ in self.opened_images:": Цей рядок розпочинає цикл, який проходить по всіх відкритих зображеннях.

"if path[-1] == "*":": Цей рядок перевіряє, чи останній символ шляху до зображення дорівнює "*" - це означає, що зображення ще не було збережене.

"return True": Якщо було знайдено незбережене зображення, метод повертає True.

"return False": Якщо незбережених зображень не знайдено, метод повертає False.

- def _close(self, event=None):

```
    def _close(self, event=None):
        if self.unsaved_images():
            if not mb.askyesno("Unsaved changes", "Got unsaved changes! Exit anyway?"):
                return

        self.root.quit()
```

Цей код містить метод _close(), який викликається при закритті додатку. Якщо у додатку є збережені нещодавні зміни в будь-якому відкритому зображенні, метод перевіряє, чи користувач бажає продовжити закриття. Якщо користувач підтверджує бажання продовжити, то метод закриває додаток. Якщо користувач відхиляє закриття, додаток залишається відкритим. Якщо змін немає, додаток закривається без додаткових запитів до користувача. Крім того, метод приймає аргумент event, який за замовчуванням є None.

В кінці програми використовується перевірка:

```
if __name__ == "__main__":
    ImageEditor().run()
```

Ця перевірка використовується для запуску програми лише у випадку, якщо файл з програмою є головним (тобто файл, який запускається) і не імпортується в інший модуль. Це дозволяє запускати програму лише тоді, коли вона запускається як головний файл, і уникнути запуску програми при імпортуванні в інші модулі.

3. Тестування та налагодження програми.

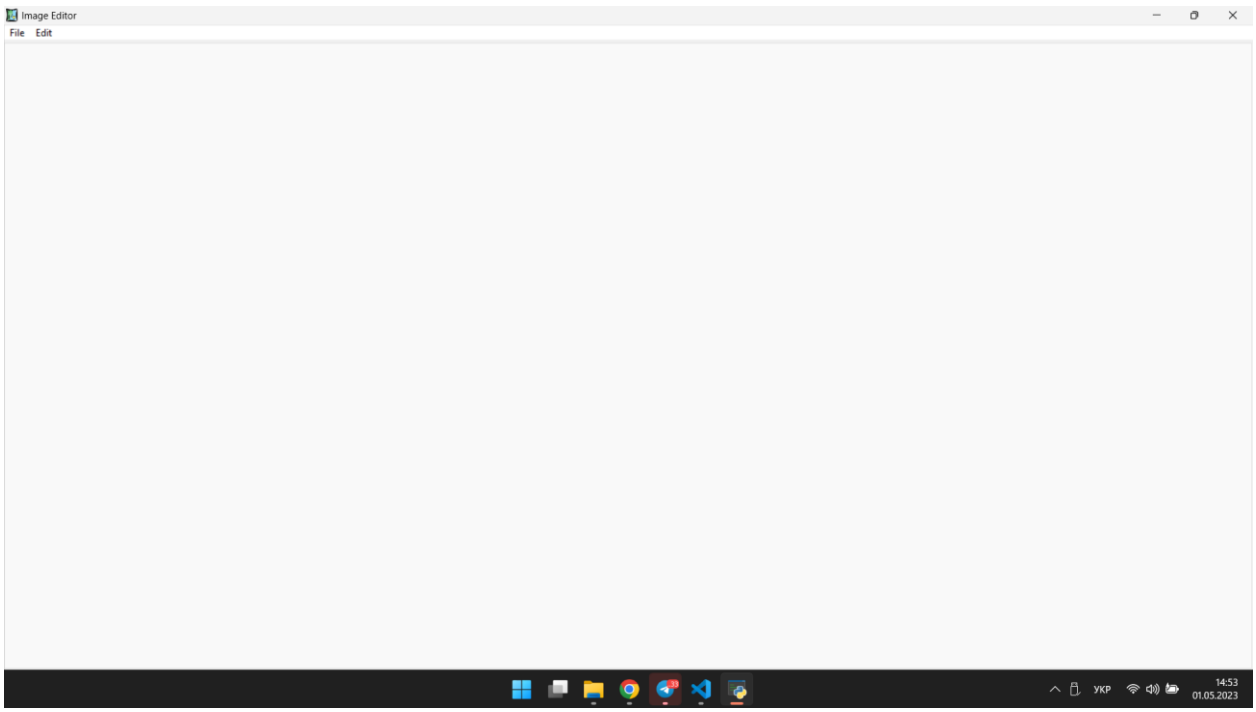
Тестування можна проводити вручну (manual testing) або автоматично (automated testing). Вручну можно просто запустити програму та спробувати виконати різні дії, щоб переконатися, що вона працює як має. Автоматичне тестування зазвичай передбачає написання спеціальних скриптів, які автоматично запускають програму та перевіряють її роботу.

Також, для тестування можна використовувати різні інструменти, наприклад, pytest, unittest, nose, тощо.

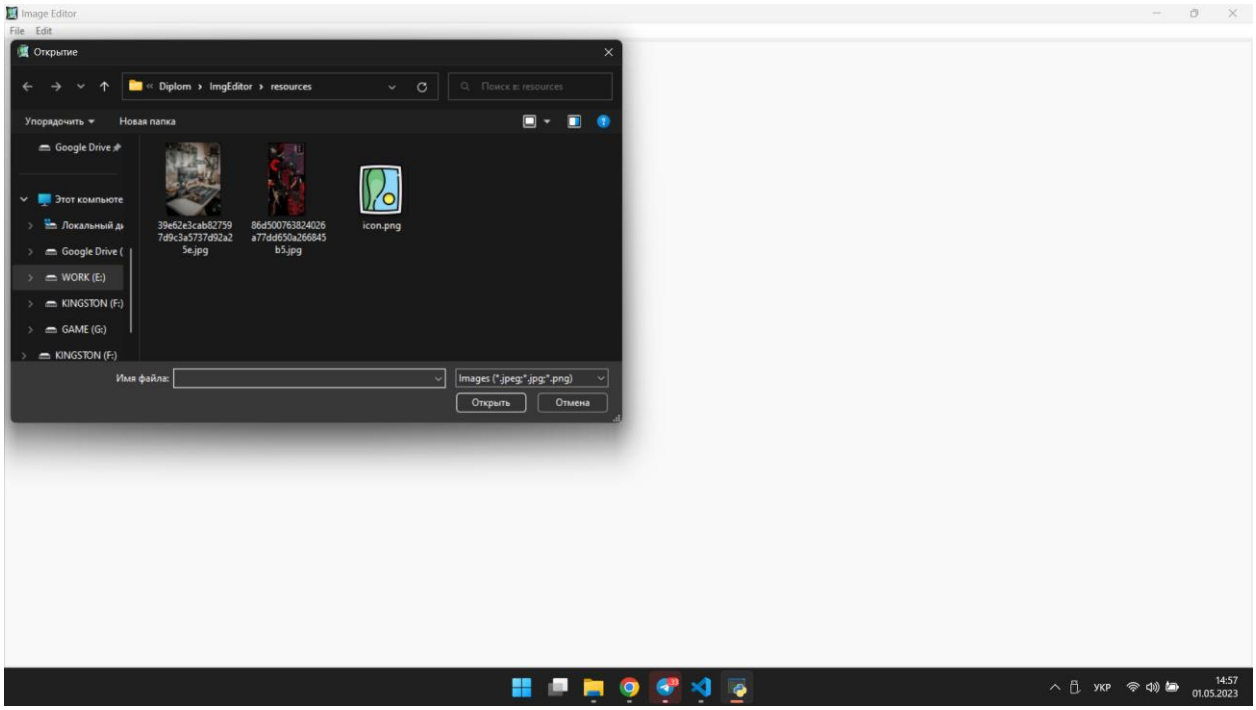
Незалежно від методу тестування, важливо перевіряти не тільки правильність роботи програми, але й її ефективність. Наприклад, якщо програма повільно працює з великими зображеннями, то це може бути проблемою для користувачів, які очікують швидкої роботи. Тому важливо тестувати програму на різних типах зображень та на різних обсягах даних, щоб переконатися, що вона працює швидко та ефективно в усіх випадках.

Для тестування цієї програми я буду проводити ручне тестування. Для цього я розписав подальші кроки тестування:

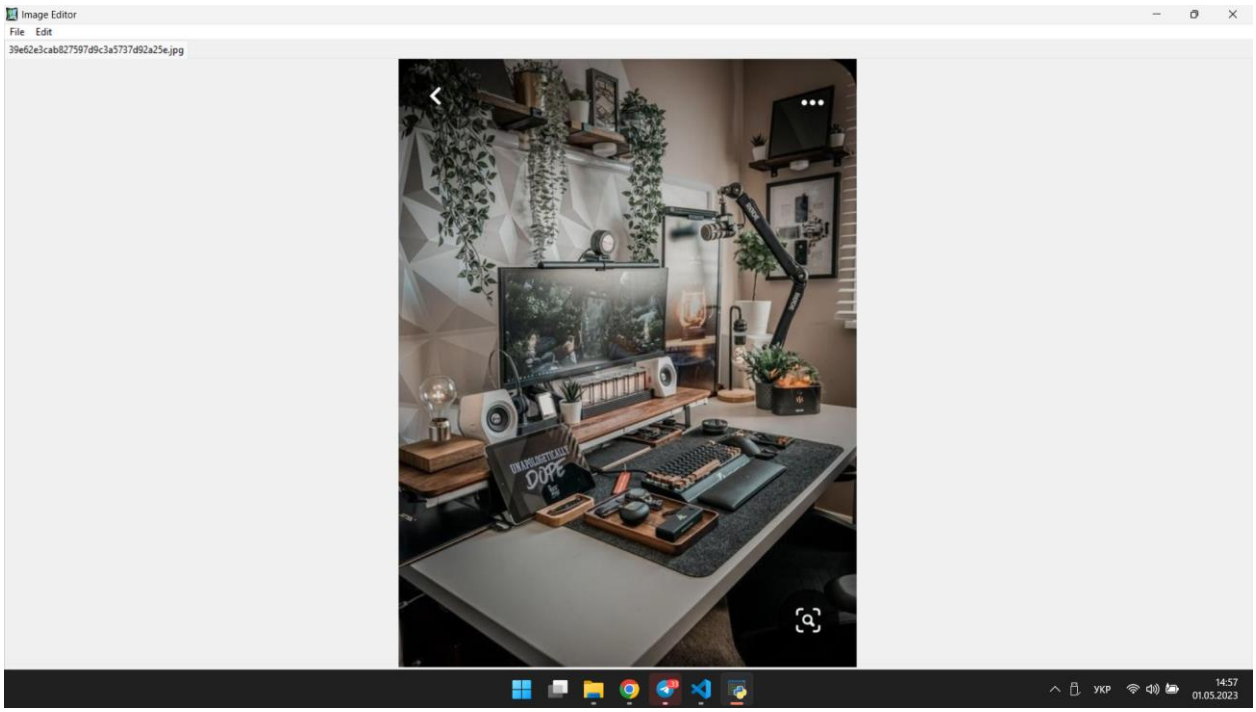
1. Запустити програму.



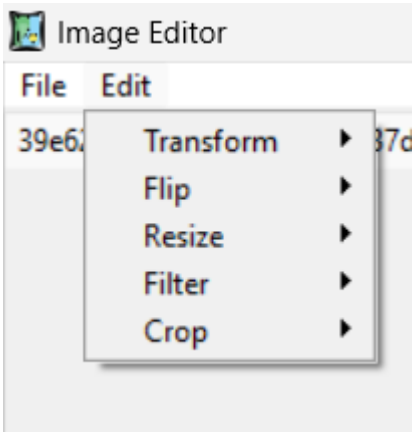
2. Клікнути на пункт меню "File" та обрати "Open".



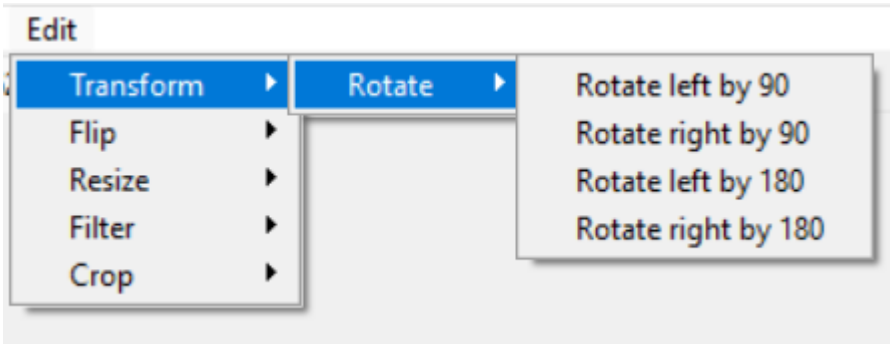
3. Обрати зображення на комп'ютері та клікнути на "Open". Перевірити, що зображення було успішно додано до програми та відображається в новій вкладці.



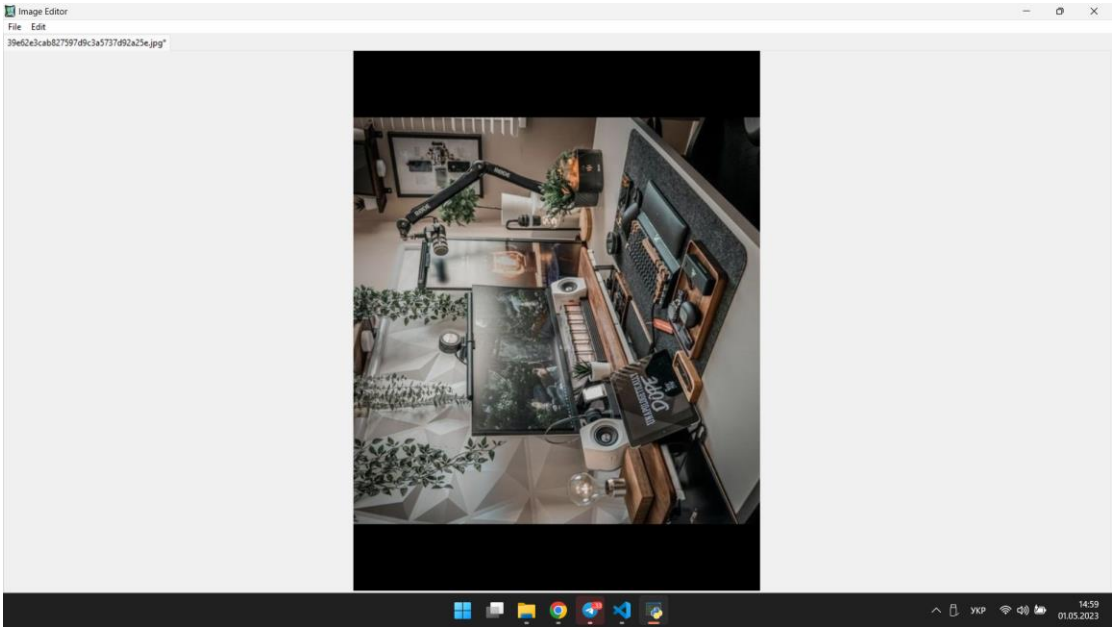
4. Відкрити пункт меню "Edit" та перевірити наявність підпунктів "Transform", "Flip", "Resize", "Filter" та "Crop".



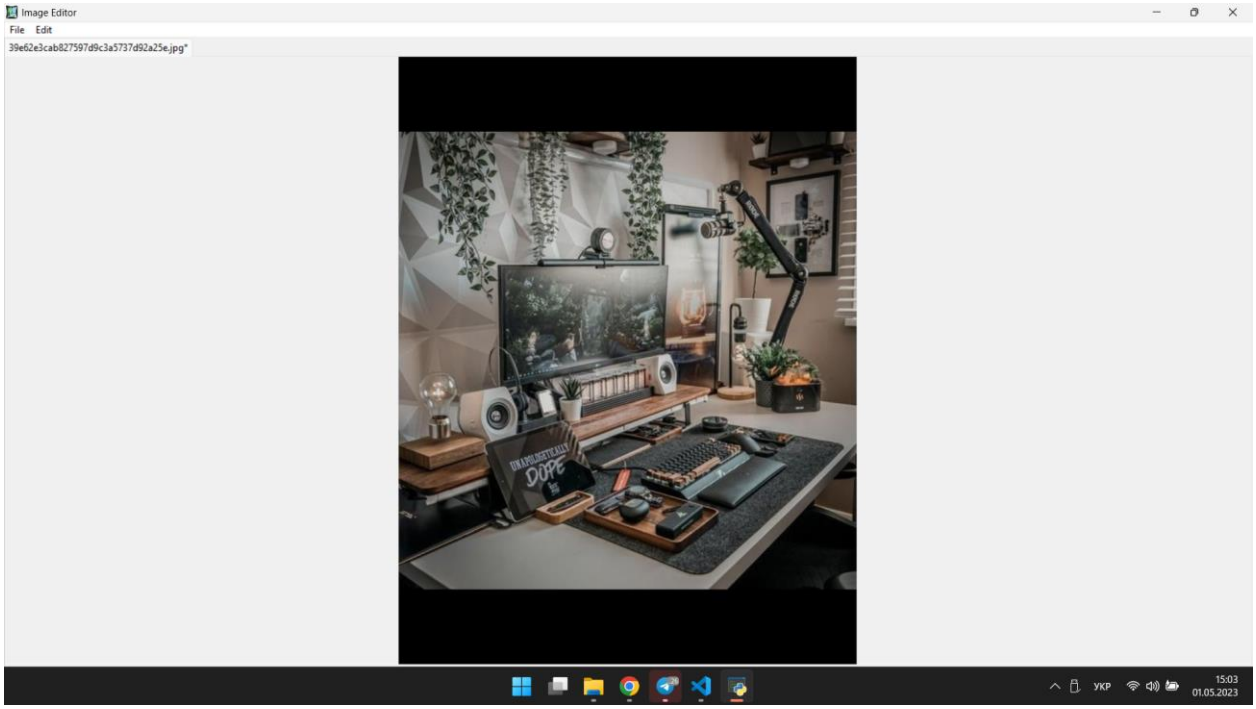
5. Відкрити підпункт "Transform" та перевірити наявність підпункту "Rotate" з підпунктами "Rotate left by 90", "Rotate right by 90", "Rotate left by 180" та "Rotate right by 180".



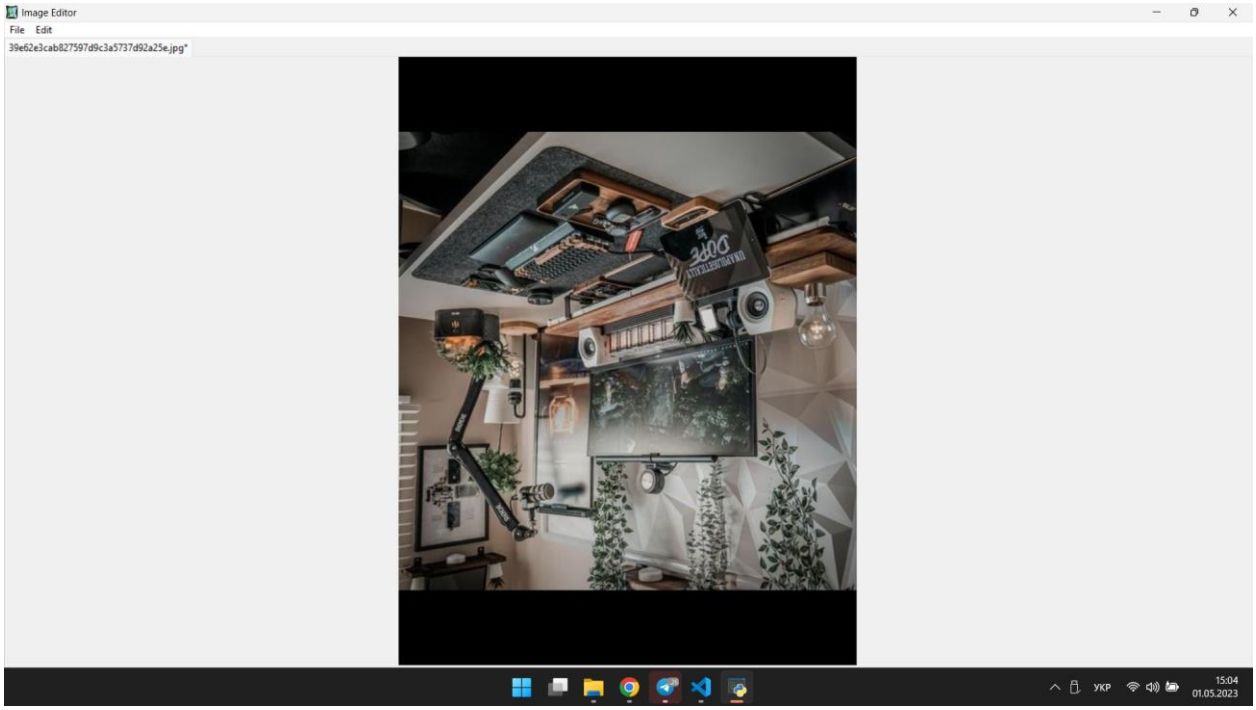
6. Клікнути на підпункт "Rotate left by 90" та перевірити, що зображення було повернуто на 90 градусів вліво.



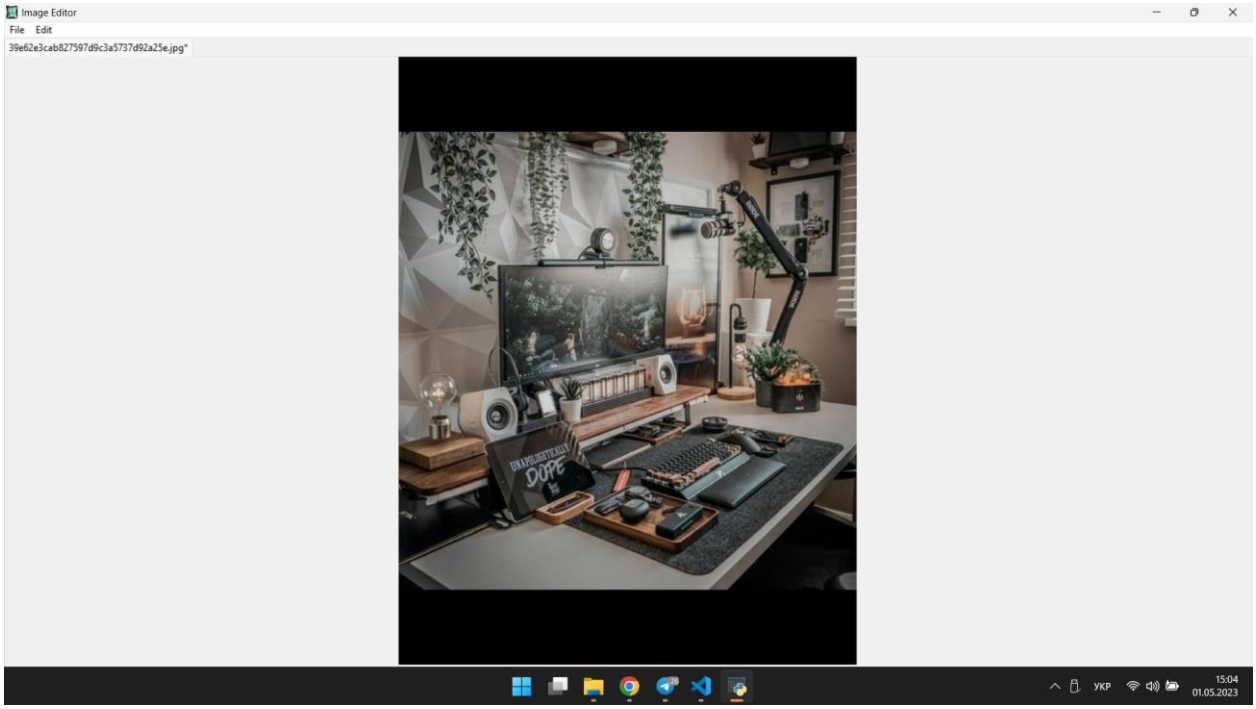
7. Клікнути на підпункт "Rotate right by 90" та перевірити, що зображення було повернуто на 90 градусів вправо.



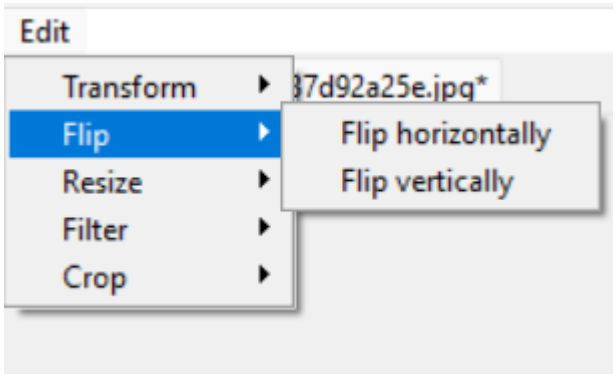
8. Клікнути на підпункт "Rotate left by 180" та перевірити, що зображення було повернуто на 180 градусів вліво.



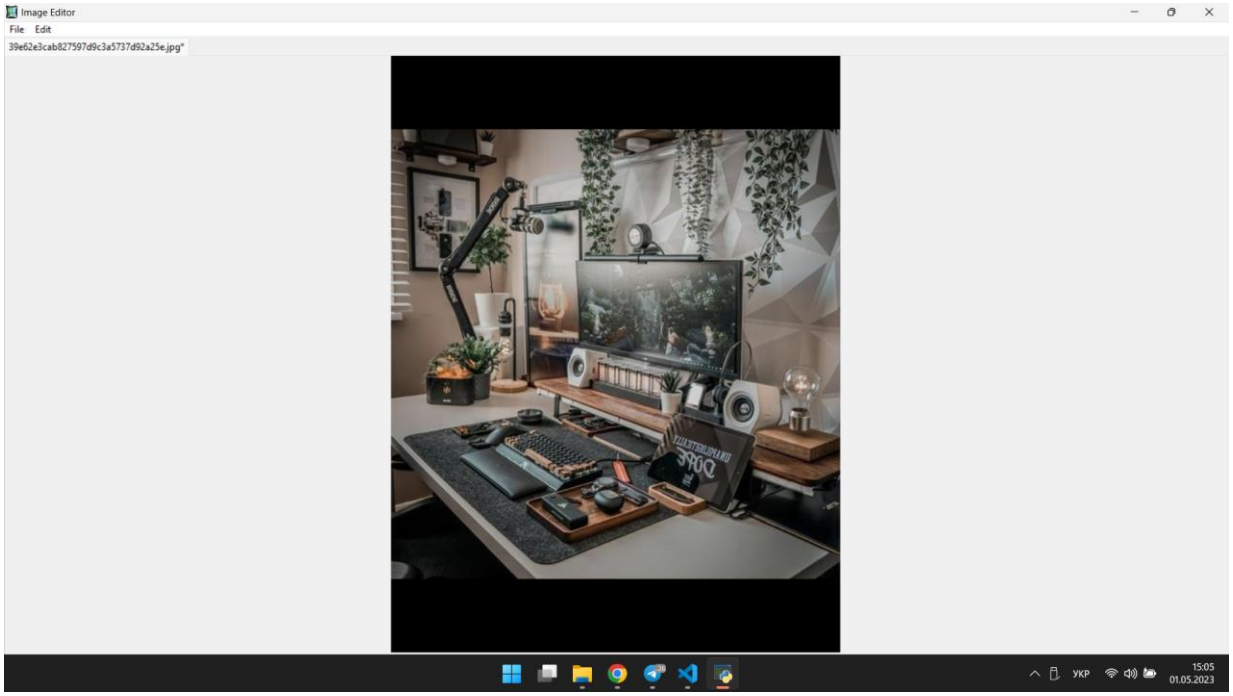
9. Клікнути на підпункт "Rotate right by 180" та перевірити, що зображення було повернуто на 180 градусів вправо.



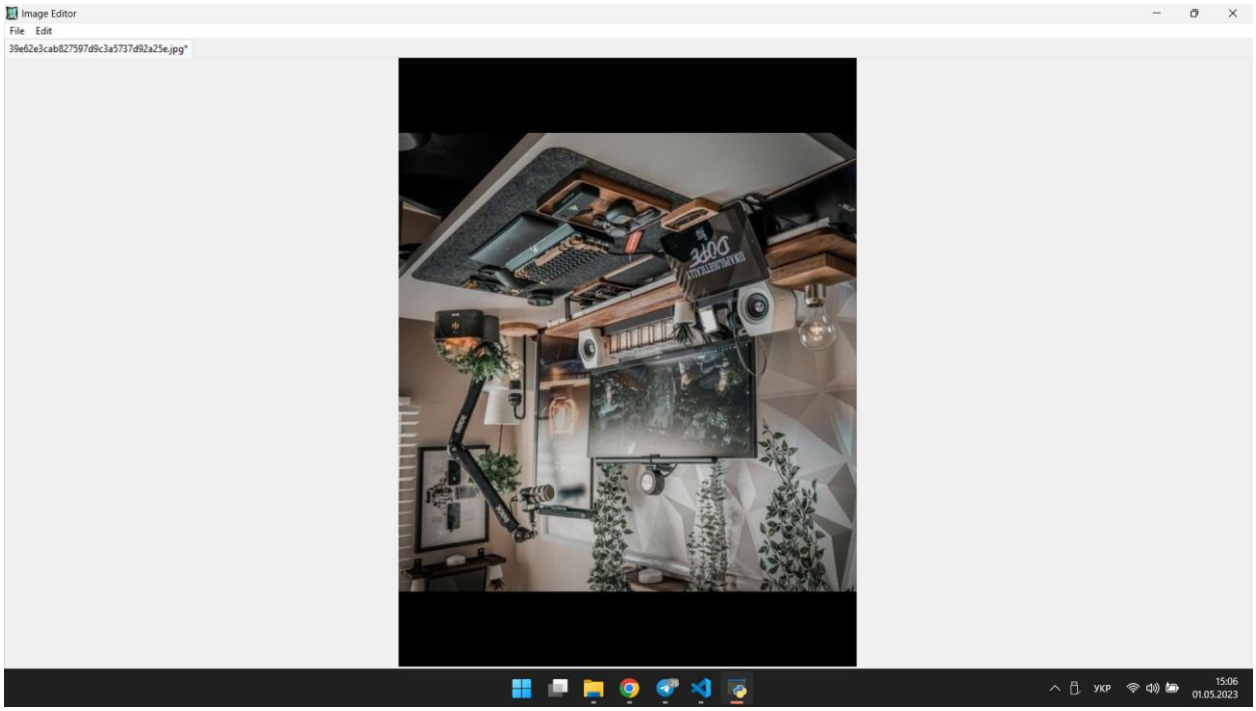
10. Відкрити підпункт "Flip" та перевірити наявність підпунктів "Flip horizontally" та "Flip vertically".



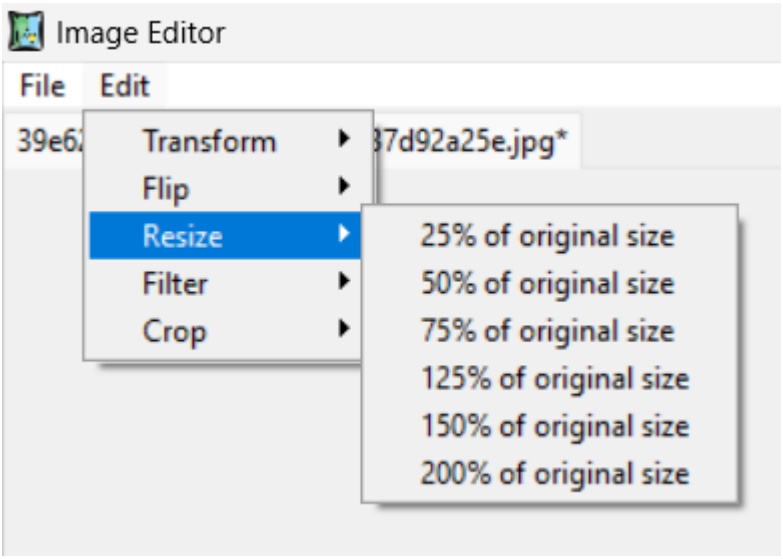
11. Клікнути на підпункт "Flip horizontally" та перевірити, що зображення було відзеркалено горизонтально.



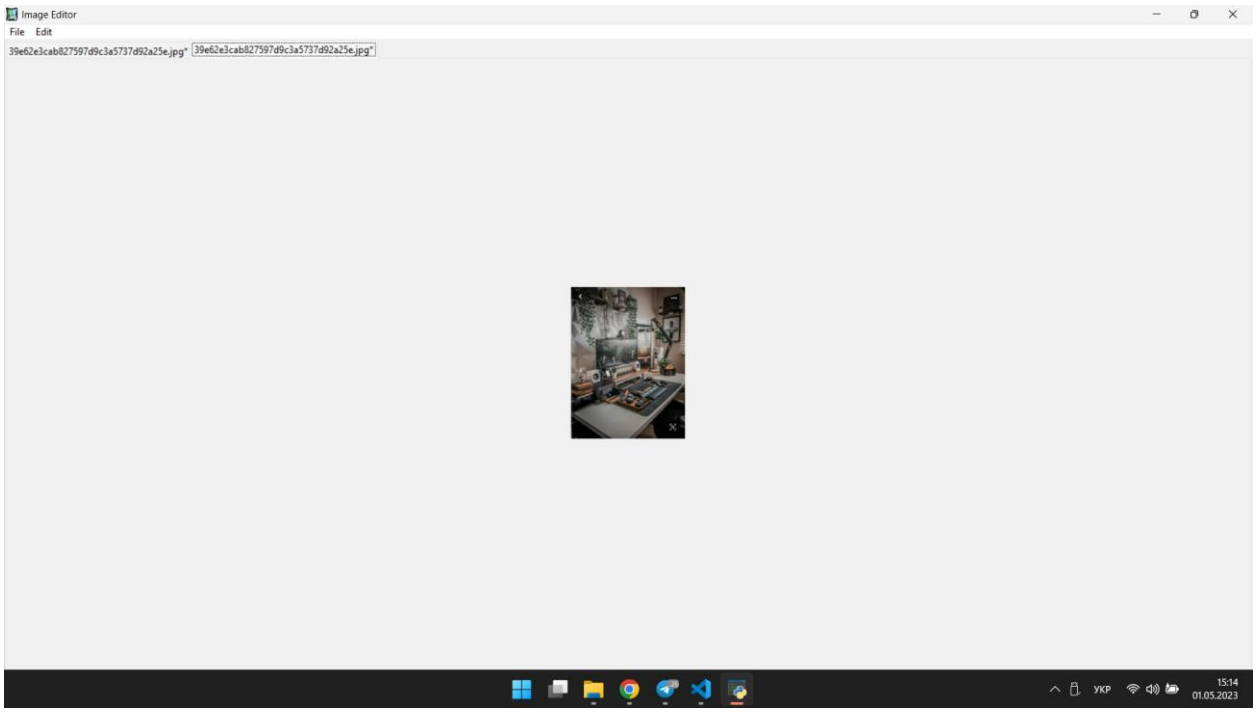
12. Клікнути на підпункт "Flip vertically" та перевірити, що зображення було відзеркалено вертикально.



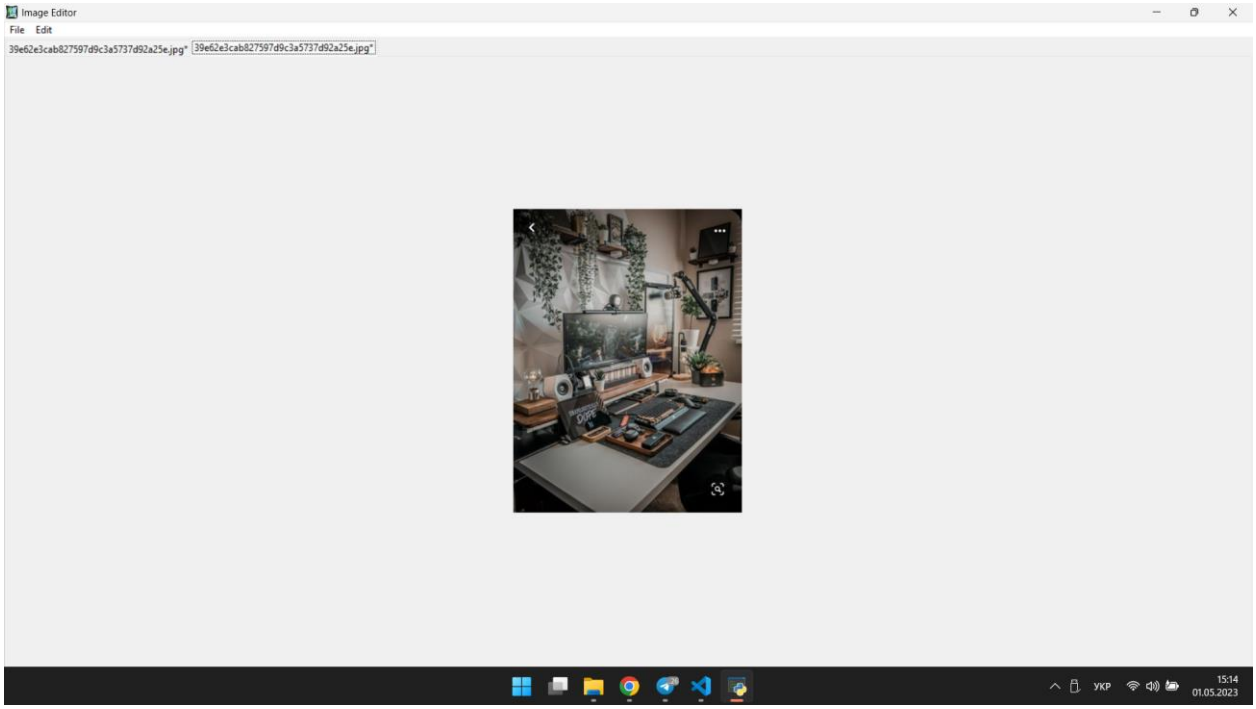
13. Відкрити підпункт "Resize" та перевірити наявність підпунктів "25% of original size", "50% of original size", "75% of original size", "125% of original size", "150% of original size" та "200% of original size".



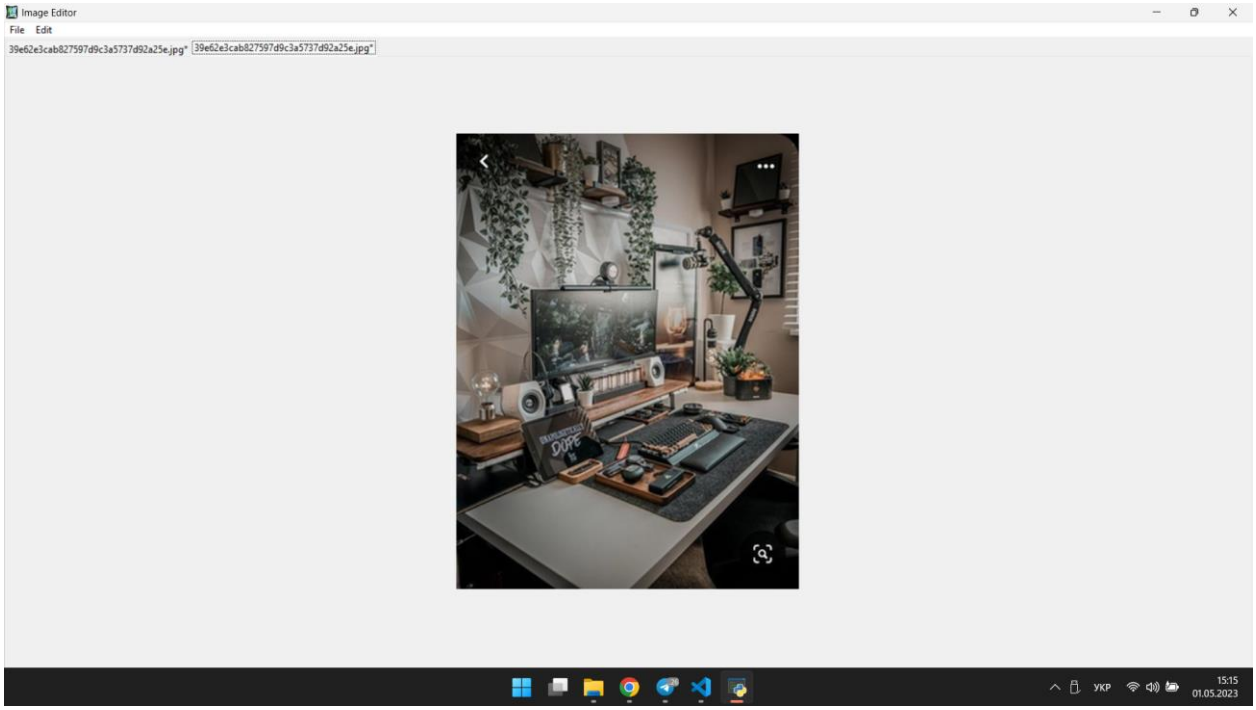
14. Клікнути на підпункт "25% of original size" та перевірити, що зображення було зменшено до 25% від початкового розміру.



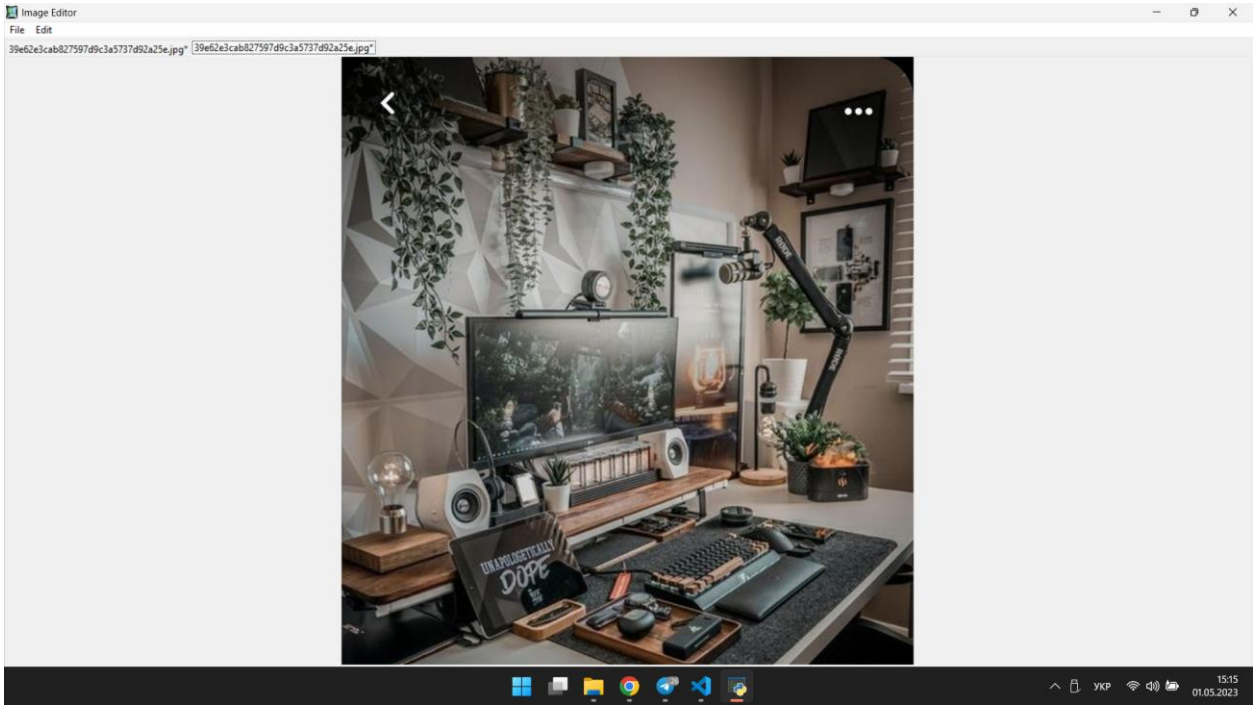
15. Клікнути на підпункт "50% of original size" та перевірити, що зображення було зменшено до 50% від початкового розміру.



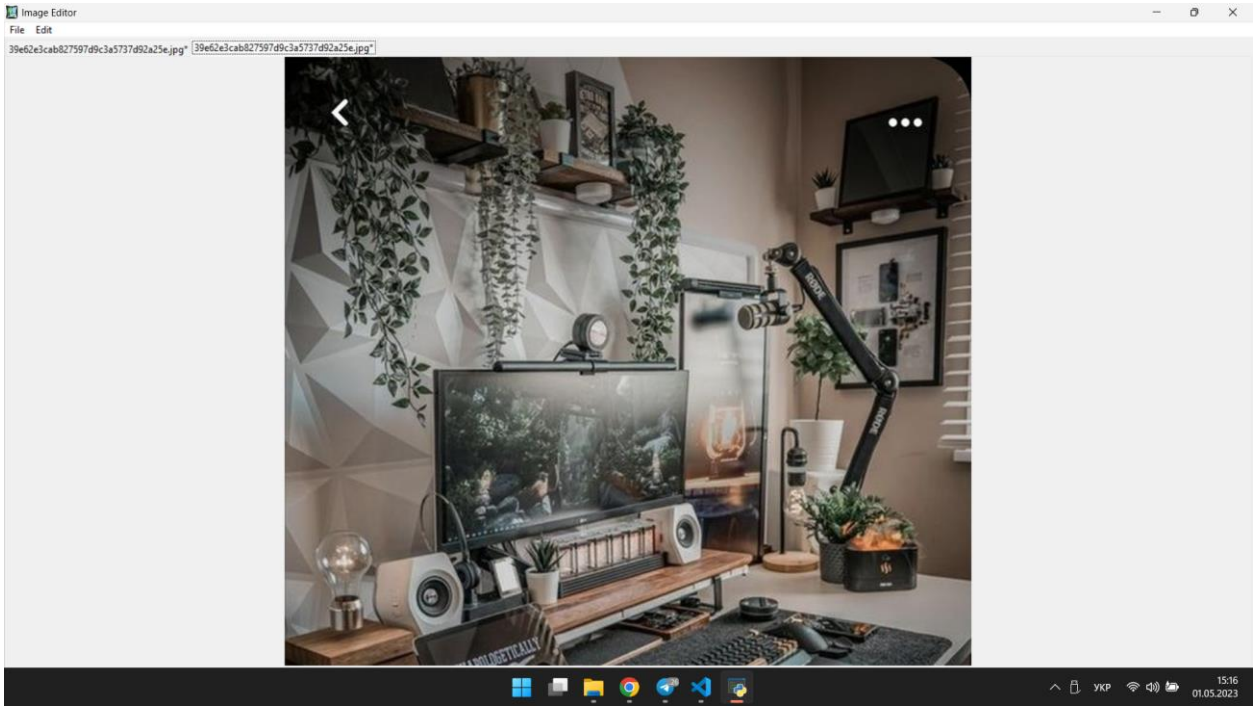
16. Клікнути на підпункт "75% of original size" та перевірити, що зображення було зменшено до 75% від початкового розміру.



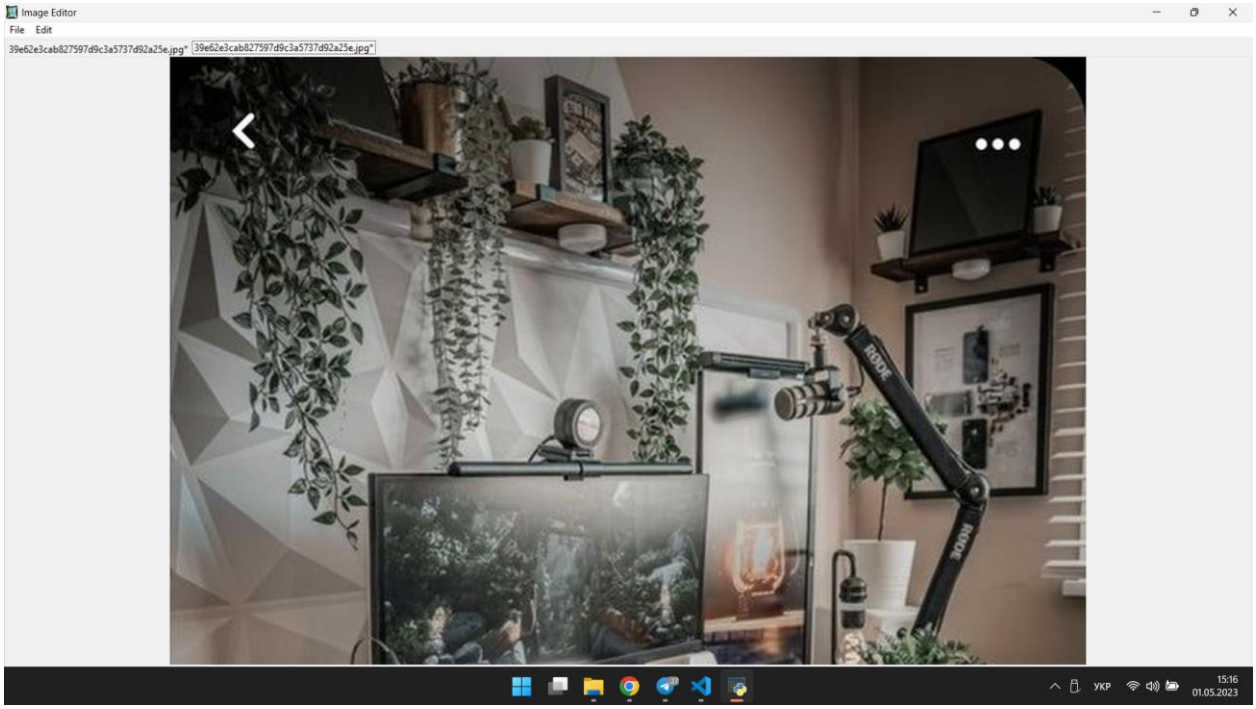
17. Клікнути на підпункт "125% of original size" та перевірити, що зображення було збільшено до 125% від початкового розміру.



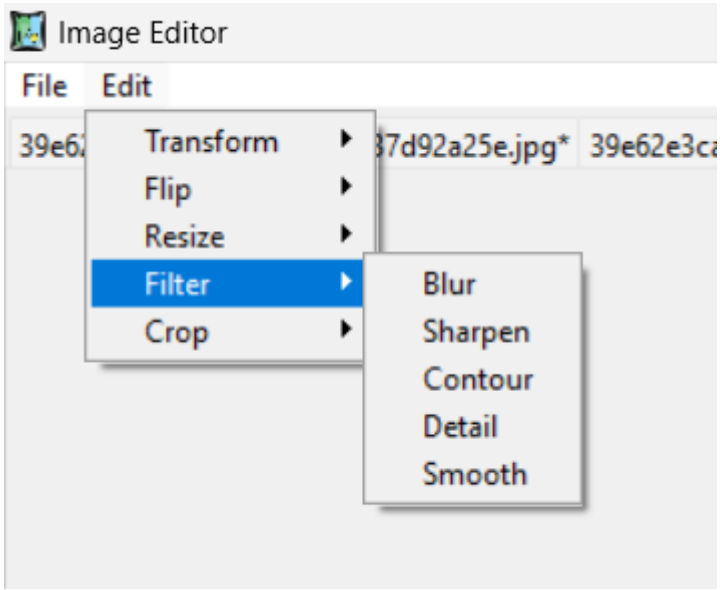
18. Клікнути на підпункт "150% of original size" та перевірити, що зображення було збільшено до 150% від початкового розміру.



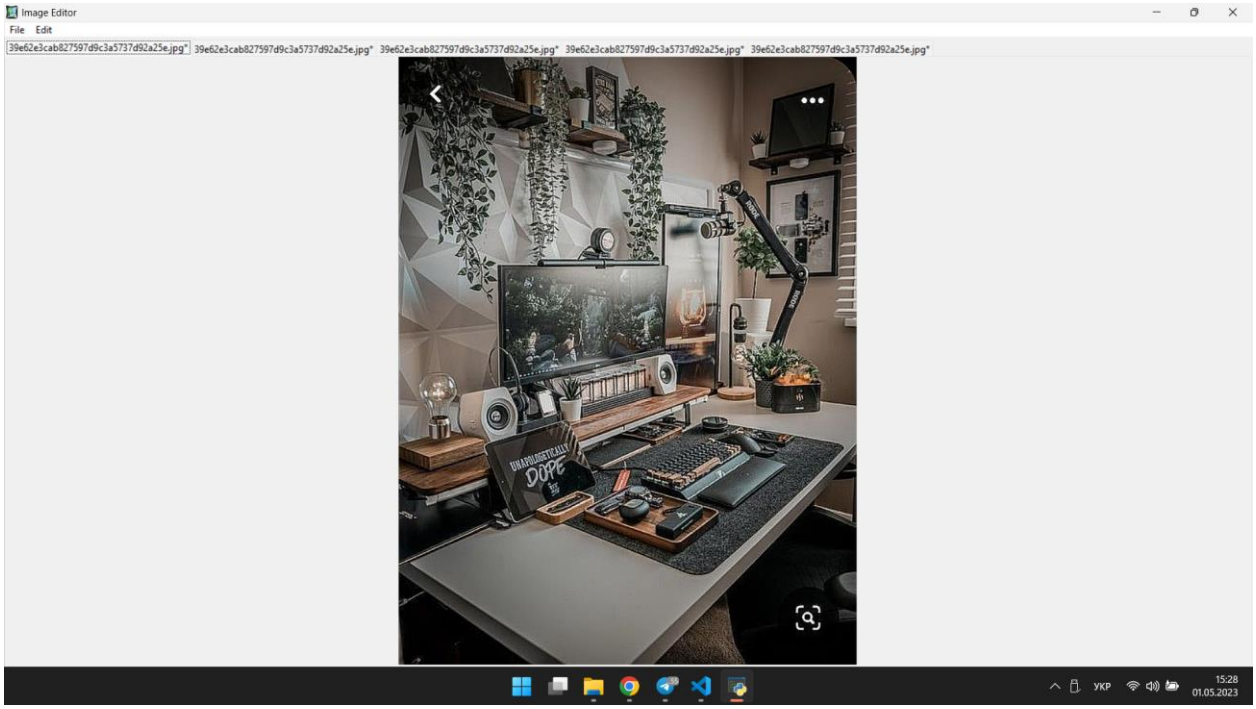
19. Клікнути на підпункт "200% of original size" та перевірити, що зображення було збільшено до 200% від початкового розміру.



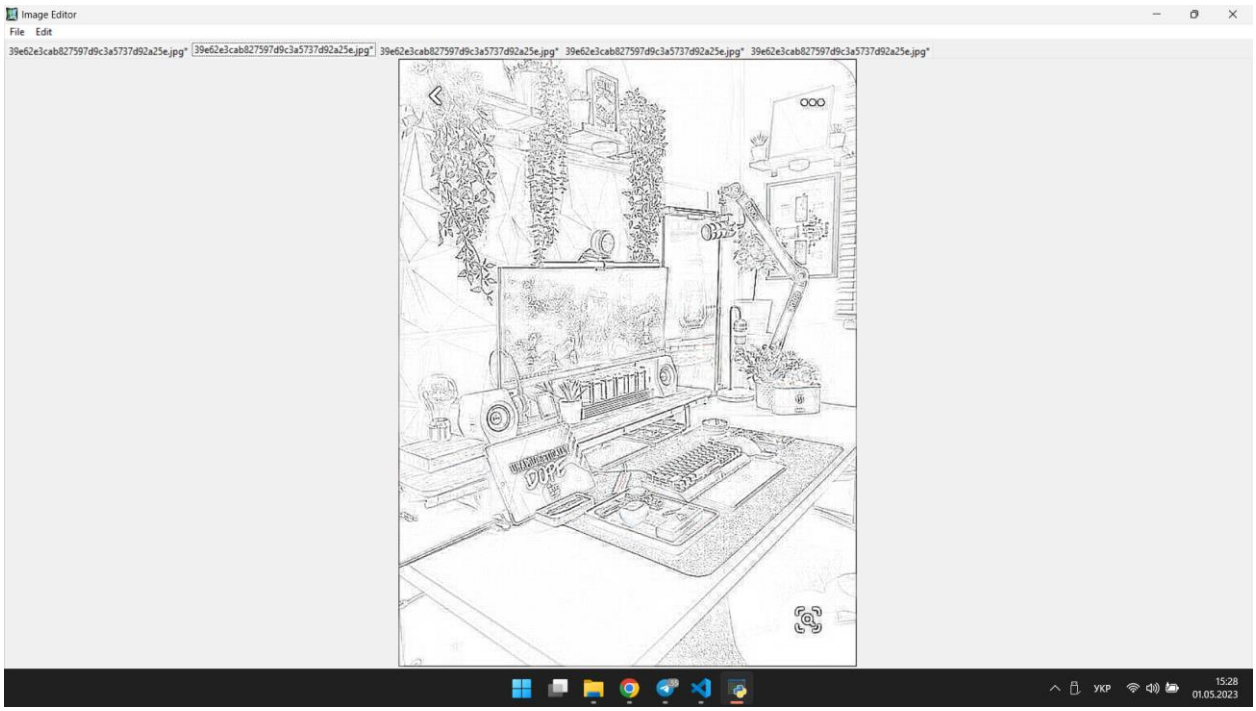
20. Відкрити підпункт "Filter" та перевірити наявність підпунктів "Sharpen", "Contour", "Detail", "Smooth" та "Blur".



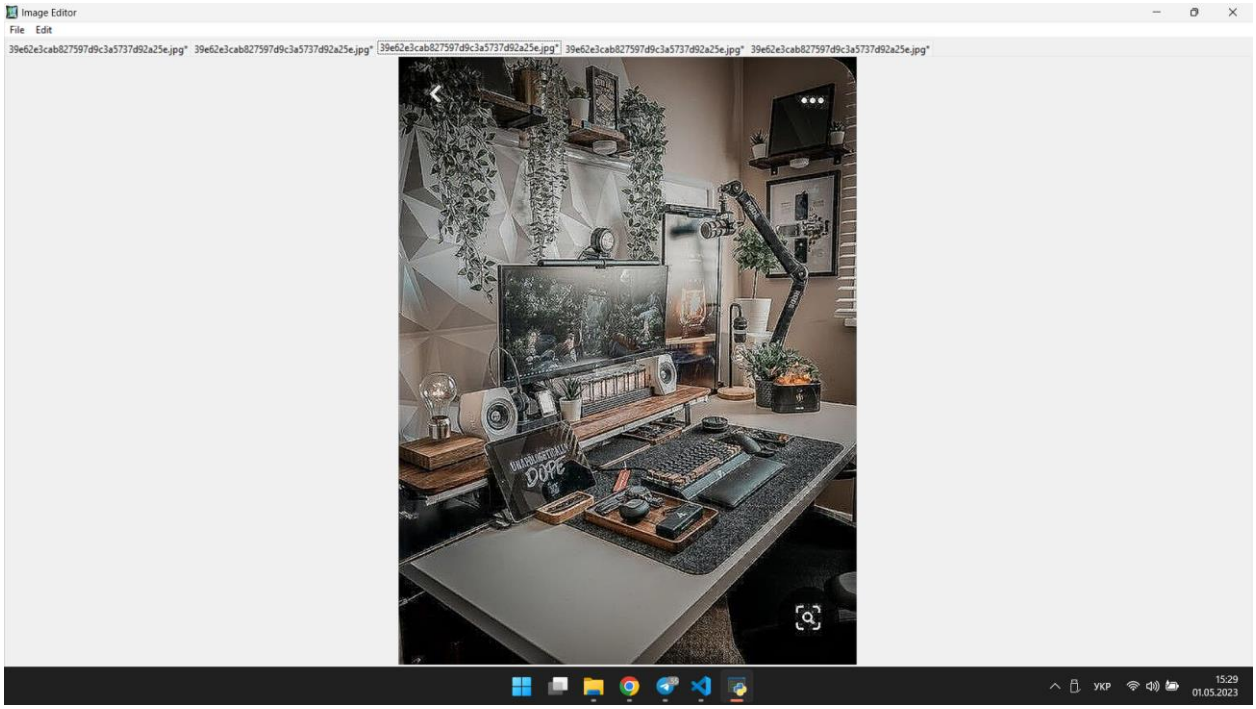
21. Клікнути на підпункт "Sharpen" та перевірити працездатність.



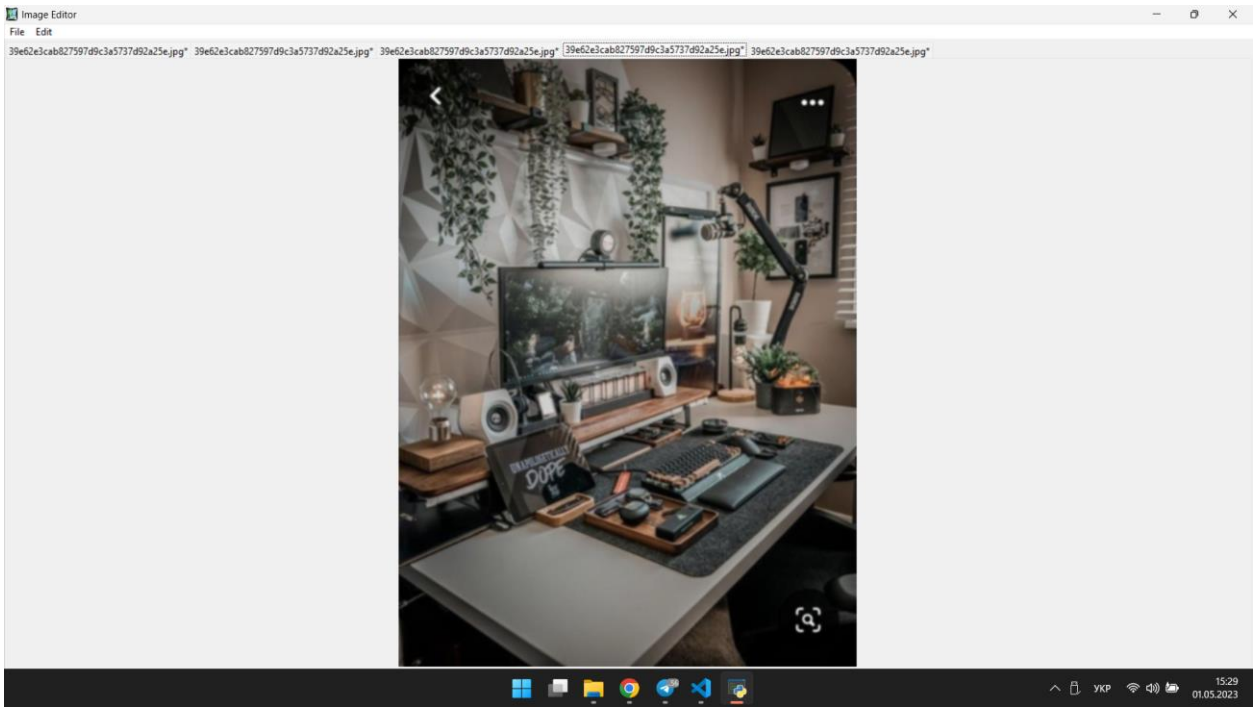
22. Клікнути на підпункт "Contour" та перевірити працездатність.



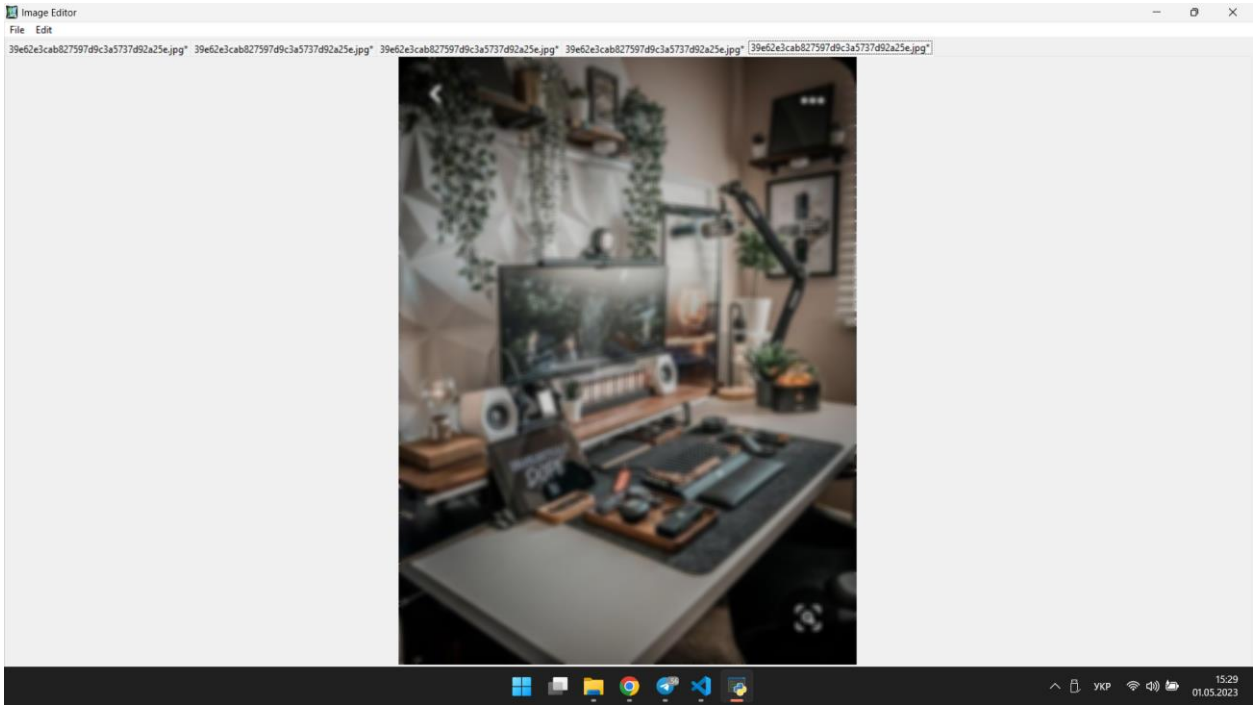
23. Клікнути на підпункт "Detail" та перевірити працездатність.



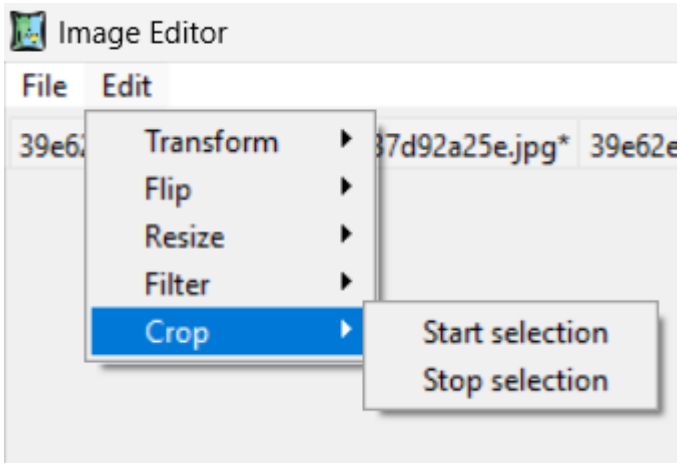
24. Клікнути на підпункт "Smooth" та перевірити працездатність.

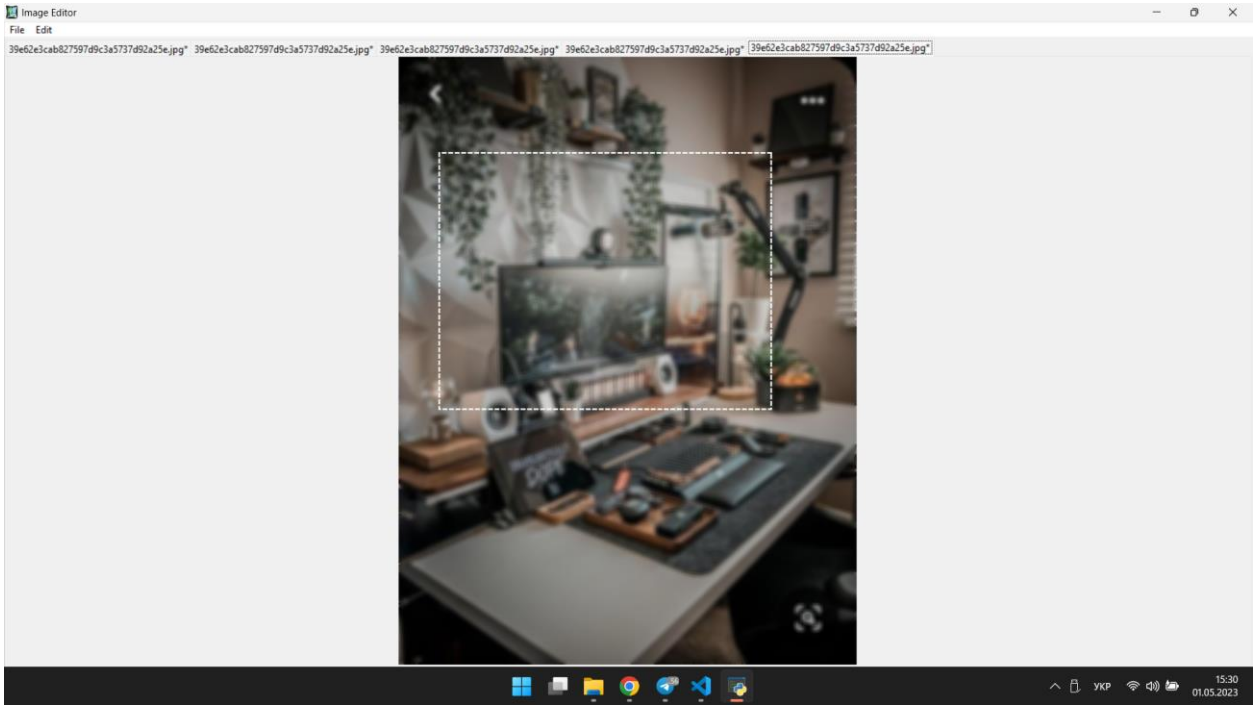


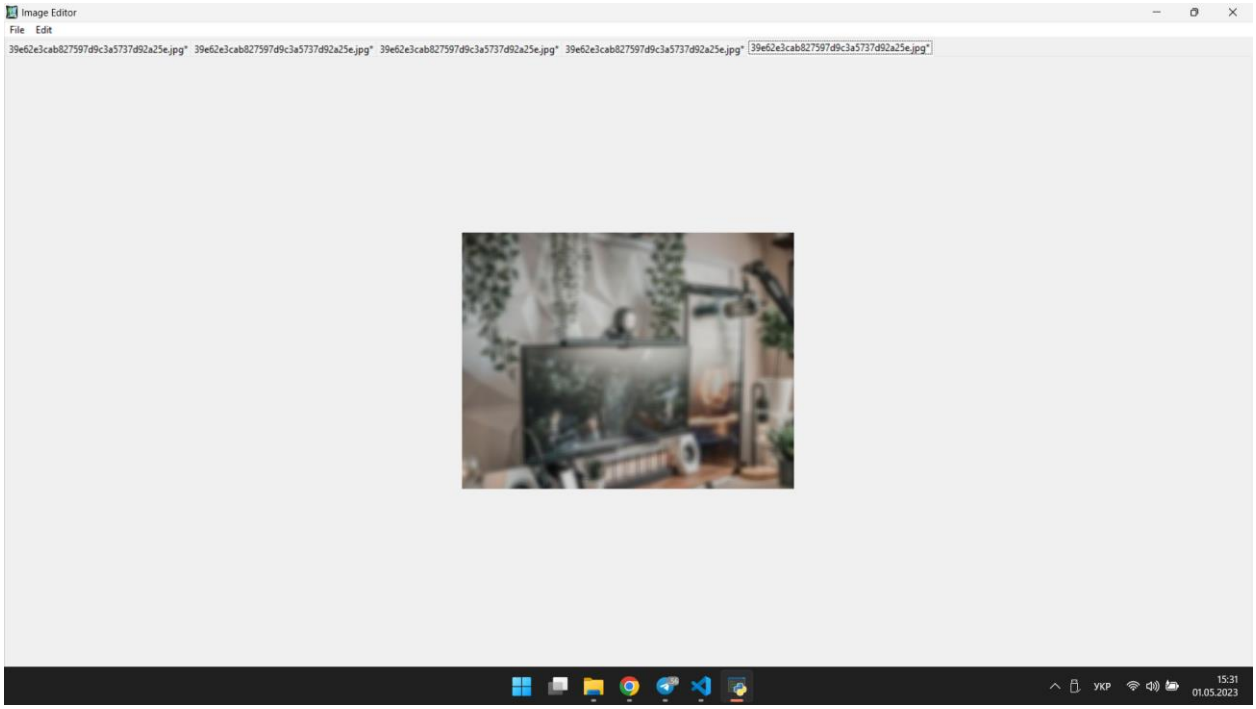
25. Клікнути на підпункт "Blur" та перевірити працездатність.



26. Відкрити підпункт "Crop" та перевірити, що він містить опцію "Start Selection", "Stop Selection". Перевірити працездатність обрізання.







За допомогою данного тестування, ми вияснили, що всі функції даної програми працюють справно.

V. Результати та аналіз

1. Оцінка якості та ефективності розробленого додатку.

Як на мене, додаток має досить зрілу структуру і може бути корисним для простих операцій зі зображеннями, наприклад, обрізка, зміна розміру, розміщення фільтрів, поворот.

Нижче наведено деякі поради щодо покращення якості коду і додавання нового функціоналу:

- Для зменшення забруднення коду можна створити окремі класи для кожної функціональності, яка потребує роботи з зображеннями. Це зменшить складність коду та забезпечить більшу модульність.
- Можна додати можливість відміни дій користувача за допомогою кнопки "Undo".
- Для кращої взаємодії з користувачем можна додати панель інструментів з швидким доступом до часто використовуваних функцій.
- Можна додати можливість зберігати налаштування обробки зображень для подальшого використання.
- Можна додати можливість змінювати якість зображення при збереженні.
- Можна додати функцію автоматичного зміщення та обрізки зображення за допомогою алгоритмів комп'ютерного зору.

Ці ідеї можуть бути використані для розвитку даного додатку та покращення його якості.

2. Порівняння з існуючими програмними рішеннями

Ця програма - простий графічний редактор зі зручним інтерфейсом користувача. Вона дозволяє відкривати, зберігати та редагувати зображення, змінювати їх розмір, повертати, віддзеркалювати та накладати фільтри.

Що стосується порівняння з іншими програмними рішеннями, то, очевидно, ця програма має обмежений функціонал порівняно зі складнішими програмами, такими як Photoshop або GIMP. Однак, вона може бути корисною для простих завдань з редагування зображень, особливо для користувачів, які не мають досвіду в роботі з більш складними програмами.

Також слід відзначити, що програма може бути менш потужною в термінах продуктивності порівняно з іншими програмами.

3. Опис можливих напрямів для подальшої розробки та покращення програми.

Ця програма є досить простою графічною програмою для редагування зображень. Для її подальшого покращення та розробки можна розглянути наступні напрями:

- Додавання нових функцій редагування зображень, таких як зміна контрасту, яскравості, насиченості, видалення шумів та інших. Для цього можна використовувати існуючі бібліотеки, наприклад, OpenCV.
- Реалізація можливості вибору кількох зображень для групового редагування. Для цього можна додати функцію відкриття декількох зображень за один раз, а також можливість використовувати батч-операції для групового застосування фільтрів, ресайзу та інших дій.
- Додавання функції автоматичного вирівнювання зображення. Для цього можна використовувати алгоритми комп'ютерного зору, такі як SIFT або SURF.
- Додавання можливості вибору інших мов за допомогою перекладачів, таких як Google Translate або Microsoft Translator.
- Реалізація додаткових функцій збереження, таких як збереження зображення в форматі PDF або збереження зображення у вигляді відео.
- Додавання можливості додавати текст та інші елементи на зображення.

- Додавання можливості використовувати нейронні мережі для автоматичного розпізнавання облич та інших об'єктів на зображеннях. Для цього можна використовувати фреймворки для глибокого навчання, такі як TensorFlow або PyTorch.

Висновок

1. Узагальнення результатів роботи

У процесі роботи було проведено дослідження та аналіз існуючих програмних рішень, визначено їх переваги та недоліки, що дозволило вибрати найбільш ефективний підхід до розробки власного продукту.

Було розроблено програму зі зрозумілим інтерфейсом, який дозволяє користувачеві виконувати різноманітні операції з графічними зображеннями, такі як зміна розміру, обрізка, поворот, налаштування якості та збереження в різних форматах.

Результатом роботи є функціональний та надійний програмний продукт, який може бути використаний в професійній та особистій сфері для обробки та перегляду графічних зображень.

2. Рекомендації для подальших досліджень у цій галузі.

Для подальших досліджень у галузі розробки програмного забезпечення для обробки та перегляду графічних зображень я розглянув наступні напрямки:

- Розширення можливостей програми: можна розглянути можливість роботи з іншими форматами зображень, використовувати алгоритми компресії зображень, використовувати штучний інтелект для автоматичного розпізнавання об'єктів на зображеннях та їх класифікації.
- Розвиток інтерфейсу користувача: можна розглянути можливість розширення функціональності інтерфейсу, збільшення швидкості роботи програми, збільшення ефективності використання ресурсів комп'ютера.
- Використання глибинного навчання: можна розглянути можливість використання методів глибинного навчання для розпізнавання об'єктів на зображеннях та їх класифікації.
- Розробка мобільного додатку: можна розглянути можливість розробки мобільної версії програмного забезпечення для обробки та перегляду графічних зображень.

Список використаних джерел

1. https://pathedits.com/blogs/tips/what-is-photo-editing#:~:text=Editing%20helps%20you%20get%20the,businesses%2C%20editing%20helps%20cement%20branding.
2. https://code.visualstudio.com/docs/editor/whyvscode
3. https://www.sciencedirect.com/science/article/pii/S2405452620301072
4. https://webandcrafts.com/blog/advantages-and-disadvantages-of-python/