

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕКОНСТРУКЦІЇ ОБЛИЧЧЯ З
НЕПОВНИХ АБО ПОШКОДЖЕНИХ ЗОБРАЖЕНЬ
SOFTWARE FOR RECONSTRUCTING FACES FROM PARTIAL OR
DAMAGED IMAGES

Виконав(ла): студент(ка) 4 курсу, групи КНТ-131

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КРУКОВСЬКИЙ Є.С

(ПРИЗВИЩЕ та ініціали)

Керівник СЕРДЮК С.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ТІМЕНКО К.І.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КРУКОВСЬКОГО Євгенія Сергійовича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення для реконструкції обличчя з неповних або пошкоджених зображень Software for Reconstructing Faces from Partial or Damaged Images

керівник проєкту (роботи) к.т.н., доцент, СЕРДЮК Сергій Микитович,
 (науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 02 червня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СЕРДЮК С.М., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю, ст. викладач		

7. Дата видачі завдання «14» квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Євгеній КРУКОВСЬКИЙ
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Сергій СЕРДЮК
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 101 с., 11 табл., 29 рис., 3 дод., 15 джерел.

ВІДНОВЛЕННЯ ОБЛИЧ НА ПОШКОДЖЕНИХ ЗОБРАЖЕННЯХ, КОМП'ЮТЕРНІ ПРОГРАМИ, ШТУЧНИЙ ІНТЕЛЕКТ, GANS, ВІДНОВЛЕННЯ ДЕТАЛЕЙ, ЗОБРАЖЕННЯ, НЕЙРОННІ МЕРЕЖІ.

Об'єкт дослідження – процес відновлення пошкоджених зображень облич.

Предмет дослідження – методи відновлення деталей облич на пошкоджених фотографіях.

Мета роботи – створення програмної системи для автоматичного відновлення обличчя на пошкоджених зображеннях з використанням передових технологій штучного інтелекту.

Результати. Створено програмний додаток, що використовує нейронні мережі для відновлення пошкоджених частин обличчя на фотографіях. Програма дозволяє відновлювати як дрібні дефекти, так і великі пошкодження, створюючи максимально реалістичні результати.

Висновки. Розроблено програмну систему для відновлення облич на пошкоджених зображеннях, яка використовує найсучасніші методи штучного інтелекту.

Галузь використання – використання в цифровій обробці зображень, медицині для відновлення облич після травм, у криміналістиці для відновлення пошкоджених доказів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 101 pages, 11 tables, 29 figures, 3 appendixes, 15 sources.

FACIAL RESTORATION IN DAMAGED IMAGES, COMPUTER PROGRAMS, ARTIFICIAL INTELLIGENCE, GANS, DETAIL RESTORATION, IMAGES, NEURAL NETWORKS.

The object of the study is software tools for restoring damaged facial images.

The subject of the study is methods for restoring facial details in damaged photographs.

The purpose of the work is to create a software system for automatic face restoration in damaged images using advanced artificial intelligence technologies.

Results. A software application has been created that uses neural networks to restore damaged parts of the face in photographs. The program allows you to restore both minor defects and major damage, creating the most realistic results.

Conclusions. A software system for restoring faces in damaged images has been developed, which uses the most modern artificial intelligence methods.

Field of application – use in digital image processing, medicine for restoring faces after injuries, and in forensics for restoring damaged evidence.

ЗМІСТ

Перелік скорочень та умовних познач.....	9
Вступ.....	10
1 Аналіз предметної області.....	11
1.1 Огляд відомих методів.....	11
1.1.1 Реставрація за допомогою традиційних алгоритмів обробки зображень	11
1.1.2 Використання традиційних методів машинного навчання	12
1.1.3 Використання глибокого навчання та генеративних моделей.....	13
1.2 Аналіз аналогів	14
1.2.1 Застосунок «VanceAI Image Enhancer».....	15
1.2.2 Сервіс «RestorePhotos.io»	15
1.2.3 Сервіс «Remini».....	16
1.3 Постановка завдання роботи.....	19
1.4 Висновки за розділом 1	19
2 Вибір та обґрунтування структури системи, яка проєктується.....	20
2.1 Вибір мови програмування	20
2.1.1 Огляд C+	20
2.1.2 Огляд Java	21
2.1.3 Огляд Python.....	22
2.1.4 Огляд C#.....	23
2.1.5 Огляд JavaScript.....	24
2.1.6 Огляд Ruby.....	25
2.2 Порівняльний аналіз мов програмування.....	26
2.3 Вибір середовища розробки.....	28
2.3.1 Огляд PyCharm	28
2.3.2 Огляд Eclipse.....	29
2.3.3 Огляд Visual Studio	30
2.4 Порівняльний аналіз середовищ розробки.....	31

2.5 Розробка структурної схеми програми	32
2.6 Висновки за розділом 2	33
3 Основні рішення з реалізації компонентів системи	35
3.1 Розробка архітектури системи	35
3.2 Опис модулів розроблюваної системи.....	36
3.3 Створення моделі для генерації.....	41
3.3.1 Алгоритм навчання	43
3.3.2 Результати навчання	48
3.4 Рішення щодо розробки інтерфейсу	52
3.5 Обробка виняткових витуацій	53
3.6 Висновки за розділом 3	55
4 Керівництво програміста.....	56
4.1 Призначення та умови застосування програми	56
4.1.1 Призначення програми	56
4.1.2 Функції програми.....	56
4.1.3 Умови застосування програми.....	56
4.2 Характеристика програми	57
4.2.1 Структура програми.....	57
4.3 Звертання до програми	58
4.4 Вхідні та вихідні данні.....	58
4.4.1 Вхідні дані.....	58
4.4.2 Вихідні дані	59
4.5 Повідомлення програмісту.....	59
4.6 Висновки за роздіом 4	59
5 Керівництво оператора.....	60
5.1 Призначення програми	60
5.2 Умови виконання програми	60
5.2.1 Вимоги програми	60
5.2.2 Вмоги до користувача.....	60
5.3 Виконання програми.....	60

5.3.1 Запуск програми	60
5.3.2 Робота з програмою	61
5.4 Повідомлення оператору	65
5.5 Висновки за розділом 5	67
6 Тестування і результати.....	68
6.1 Опис методики проведення тестування.....	68
6.2 Результат використання розробленого програмного забезпечення.....	69
6.3 Висновки за розділом 6	72
Висновки	73
Перелік джерел посилання	75
Додаток А Технічне завдання	77
Додаток Б Текст програми	86
Додаток В Слайди презентації.....	95

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
ML	– Machine Learning;
IDE	– Integrated Development Environment,
DL	– Deep Learning;
GAN	– Generative Adversarial Network;
SVM	– Support Vector Machine;
ПЗ	– програмне забезпечення.

ВСТУП

Відновлення обличчя на пошкоджених зображеннях відіграє важливу роль у збереженні візуальної інформації, естетики та ідентичності особи. Такі технології мають велике значення як у побутовій сфері — для реставрації старих сімейних фотографій, так і у професійних галузях, таких як криміналістика, судова експертиза чи медицина. Пошкодження зображень можуть бути спричинені фізичними дефектами, віковими змінами носіїв або цифровими артефактами, що значно ускладнює ідентифікацію або збереження цінної візуальної інформації.

У сучасному цифровому середовищі технології на основі штучного інтелекту стали ефективним інструментом для відновлення зображень. Алгоритми дозволяють автоматично реконструювати втрачені або пошкоджені частини обличчя, базуючись на контексті та характерних особливостях. Це значно спрощує процес реставрації та робить його доступним навіть для користувачів без спеціальної підготовки.

Попит на програмні рішення для автоматичного відновлення зображень постійно зростає. Такі системи дозволяють швидко та якісно відновлювати пошкоджені фото, зберігати пам'ятки минулого та підвищувати якість зображень у професійних архівах. Застосування таких рішень особливо актуальне в архівній справі, цифровій реставрації, візуальній ідентифікації осіб та навіть у створенні фото- і відеоматеріалів у медіа.

Таким чином, впровадження інтелектуальних інструментів відновлення облич на пошкоджених зображеннях є не лише технологічним досягненням, а й соціально значущим кроком до збереження візуальної історії та підвищення якості цифрового контенту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Для того, щоб програма для реставрації облич була ефективною, необхідно мати високоточні алгоритми для відновлення пошкоджених частин зображень, зокрема облич. Від якості роботи програмного забезпечення залежить точність і швидкість відновлення, а також здатність зберігати природний вигляд обличчя.

Предметною областю є комп'ютерний зір і обробка зображень, особливо відновлення зображень, зосереджене на відновленні пошкоджених чи неповних даних на обличчях. Це включає аналіз зображень, визначення часткових пошкоджень та використання сучасних нейронних мереж для відновлення втрачених частин.

1.1 Огляд відомих методів

Завдання реставрації обличчя є важливим у багатьох сферах, таких як медицина (відновлення зображень постраждалих осіб), криміналістика (розпізнавання осіб на пошкоджених фотографіях), а також в індустрії розваг (фільми, відеоігри).

Існує кілька методів виконання завдання реставрації обличчя з пошкоджених зображень.

1.1.1 Реставрація за допомогою традиційних алгоритмів обробки зображень

Традиційні методи реставрації зображень базуються на застосуванні класичних алгоритмічних підходів, зокрема інтерполяції, фільтрації та контекстного заповнення, які реалізуються за допомогою аналітичних або емпіричних математичних моделей.

Одним із найбільш поширених методів є використання медіанного фільтра, принцип якого полягає в заміні значення окремого пікселя на медіанне значення його локального оточення. Такий підхід демонструє високу ефективність у задачах усунення імпульсного шуму або точкових дефектів зображення.

Переваги:

- простота реалізації та налаштування;
- можливість використання на застарілих апаратних системах.

Недоліки:

- низька ефективність при відновленні зображень з великими пошкодженими ділянками;
- висока ймовірність появи артефактів та неприродного вигляду зображення.

1.1.2 Використання традиційних методів машинного навчання

Методи, засновані на класичному машинному навчанні, можуть використовуватися для аналізу і відновлення пошкоджених ділянок на зображенні шляхом класифікації частин обличчя, що підлягають відновленню, та використанням окремих моделей для кожної частини.

Одним з таких методів є метод опорних векторів (SVM), який застосовується для класифікації пікселів або фрагментів зображення як пошкоджених або непошкоджених. Це дозволяє точніше визначити області, які потребують реконструкції. Також можуть використовуватись регресійні моделі, як-от лінійна чи поліноміальна регресія, для передбачення відсутніх значень пікселів у зображенні на основі навколишнього контексту, особливо в умовах часткових пошкоджень або спотворень.

Переваги:

- можуть обробляти зображення з меншими пошкодженнями;
- використання існуючих моделей для обличчя.

Недоліки:

- технічно складніше, ніж традиційні методи;
- потрібні значні ресурси для навчання моделей.

1.1.3 Використання глибокого навчання та генеративних моделей

Глибоке навчання, особливо генеративні моделі, такі як GAN (Generative Adversarial Networks), є найсучаснішим підходом до реставрації пошкоджених зображень. Моделі, натреновані на великих базах даних зображень облич, можуть ефективно відновлювати пошкоджені частини зображень.

Переваги:

- висока точність та природність відновлених зображень;
- можливість відновлювати великі ділянки зображень, включаючи відсутні частини.

Недоліки:

- потребує великих обчислювальних ресурсів для навчання та виконання;
- необхідність великої кількості даних для тренування.

Вище було наведено відомі методи відновлення зображень. Вони мають свої переваги і недоліки. В табл. 1.1 наведено порівняння цих методів

Таблиця 1.1 – Порівняння методів відновлень

Критерії порівняння	Традиційні методи	Методи машинного навчання	Глибоке навчання (GAN)
Ефективність	Середня	Висока	Дуже висока
Зручність використання	Висока	Середня	Низька

Продовження таблиці 1.1

Критерії порівняння	Традиційні методи	Методи машинного навчання	Глибоке навчання (GAN)
Надійність і точність	Низька при великих пошкодженнях	Середня	Висока
Вартість розробки	Низька	Середня	Висока
Складність реалізації	Низька	Висока	Висока
Масштабованість	Низька	Висока	Висока
Потреба в великих даних	Немає	Помірна	Висока
Потрібність в обчислювальних потужностях	Низька	Середня	Висока

1.2 Аналіз аналогів

Для реалізації ефективного програмного забезпечення для реставрації обличчя з пошкоджених зображень доцільно обрати методи на основі глибокого навчання. Це дозволить отримати високу якість відновлених зображень та забезпечить можливість обробки складних випадків пошкоджень. Використання генеративних моделей, таких як GAN, дозволить створювати природні, реалістичні зображення, що є важливим для задач, пов'язаних з обличчями.

Цей метод має значний потенціал для використання у криміналістиці, медицині та в цифрових мистецтвах, де важлива висока точність і відновлення пошкоджених зображень.

1.2.1 Застосунок «VanceAI Image Enhancer»

«VanceAI Image Enhancer» — це онлайн-сервіс для автоматичного покращення фотографій (рис. 1.1), який використовує технології глибокого навчання, зокрема GAN, для відновлення, масштабування та очищення зображень [1]. Сервіс дозволяє покращити портрети, відновити розмиті фотографії, а також видалити шуми та артефакти, зберігаючи високу якість деталей [1].

Основні переваги програмного забезпечення:

- покращення зображень без втрати якості завдяки GAN;
- автоматичне усунення розмиття та шумів;
- підтримка покращення портретів і ландшафтів;
- простий веб-інтерфейс, не потребує встановлення.

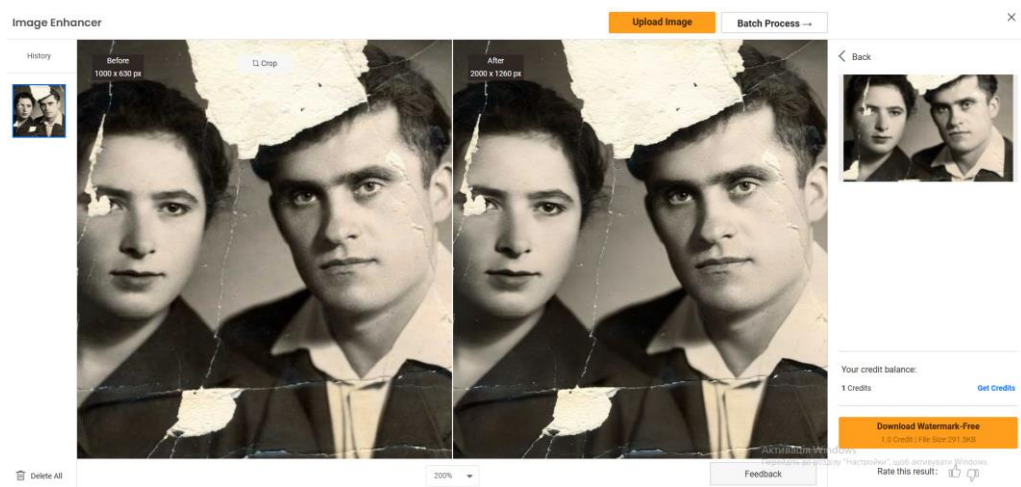


Рисунок 1.1 – Застосунок «VanceAI Image Enhancer»

1.2.2 Сервіс «RestorePhotos.io»

«RestorePhotos.io» — це онлайн-сервіс для відновлення зображень (рис. 1.2), що використовує сучасні моделі штучного інтелекту GFPGAN та CodeFormer для реставрації старих, пошкоджених або розмитих фотографій

[2]. Сервіс спеціалізується на відновленні облич, відтворюючи втрачені деталі та підвищуючи загальну якість зображень, що особливо корисно для архівних або сімейних фото [2].

Основні переваги програмного забезпечення:

- автоматичне відновлення облич на основі AI;
- підтримка сучасних моделей GFPGAN і CodeFormer;
- зручний веб-інтерфейс без необхідності встановлення;
- швидка обробка та завантаження результатів.

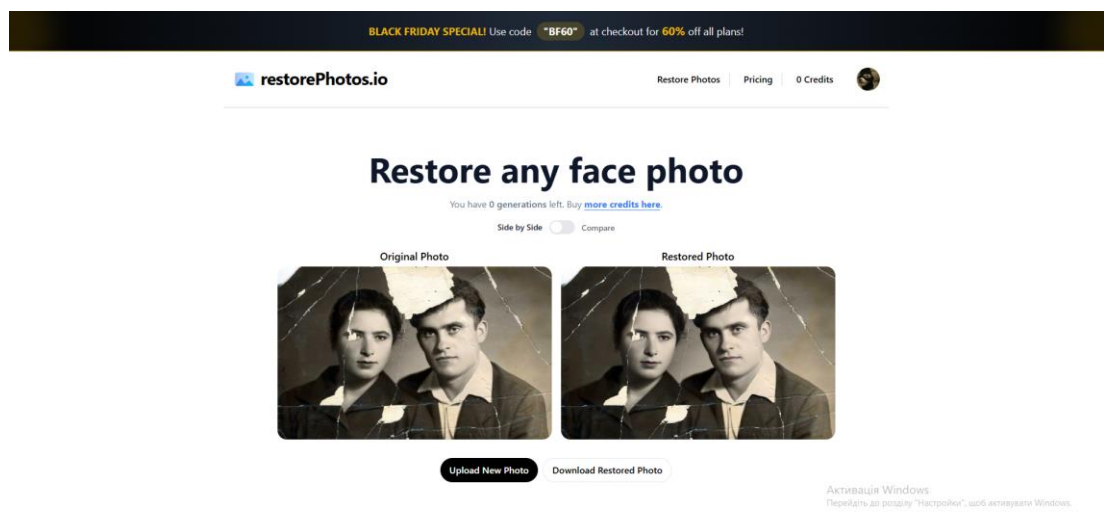


Рисунок 1.2 – Застосунок «RestorePhotos.io»

1.2.3 Сервіс «Remini»

«Remini» — це онлайн-сервіс для редагування зображень (рис. 1.3), що використовує штучний інтелект для покращення якості фотографій, зокрема старих, розмитих або пошкоджених зображень [3]. Додаток особливо ефективний для відновлення облич на знімках низької якості, забезпечуючи деталізацію та чіткість завдяки глибокому навчанню нейронних мереж [3].

Основні переваги програмного забезпечення:

- автоматичне покращення зображень за допомогою AI;
- висока якість відновлення портретів;
- можливість порівняння «до» та «після»;

— простий і зручний мобільний інтерфейс.

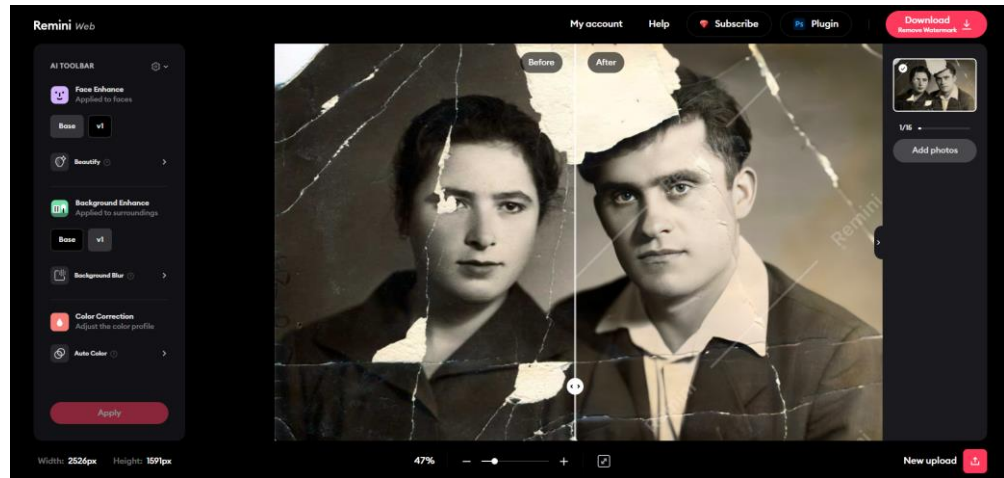


Рисунок 1.3 – Застосунок «Remini»

Вище було розглянуто найбільш популярні на сьогоднішній день програмні засоби для покращення якості зображень та їх відновлення. Кожен із них має свої сильні сторони, проте існують і певні недоліки. Наприклад, деякі сервіси не підтримують обробку зображень офлайн, не забезпечують зручного редагування результатів користувачем або не дозволяють працювати з декількома зображеннями одночасно.

Разом з тим, більшість із проаналізованих програм забезпечують базовий функціонал, такий як автоматичне відновлення зображень за допомогою штучного інтелекту, збереження результату, а також покращення якості портретів.

У межах даної роботи буде реалізовано застосунок, який міститиме низку переваг у порівнянні з існуючими аналогами. Зокрема, планується реалізувати функції автоматичного відновлення зображень, можливість локальної обробки без підключення до Інтернету та підвищену точність розпізнавання деталей облич.

У таблиці 1.2 представлено порівняння розглянутих програмних засобів та розробленого застосунку.

Таблиця 1.2 — Порівняння застосунків

Можливості	VanceAI Image Enhancer	RestorePhotos.io	Remini	Власний застосунок
Потужні інструменти ретуші	—	—	—	+
Автоматичне відновлення зображень	+	+	+	+
Підтримка штучного інтелекту	+	+	+	+
Потрібне встановлення	—	—	—	+
Онлайн-доступ	+	+	+	—
Простота використання	+	+	+	+
Можливість обробки портретів	+	+	+	+
Професійний рівень редагування	—	—	—	+
Безкоштовний доступ	± (частково)	+	—	+

У результаті проведеного аналізу сучасних програмних засобів для відновлення зображень встановлено, що попри широке застосування штучного інтелекту й автоматизованих методів обробки, більшість з них мають функціональні обмеження. Зокрема, відсутність офлайн-доступу,

обмеженість в редагуванні результатів знижують їхню універсальність та зручність у практичному використанні.

1.3 Постановка завдання роботи

Застосунок повинен забезпечувати виконання нижче перерахованих функцій:

- відновлення пошкоджених частин обличчя на зображеннях;
- підтримка високої точності відновлення для різних типів пошкоджень;
- відновлення кольору та контрасту зображення;
- можливість збереження відновлених зображень у високій якості;
- можливість подальшого редагування результату відновлення;
- інтеграція алгоритмів глибокого навчання для точного розпізнавання та відновлення образів.

1.4 Висновки за розділом 1

У першому розділі проведено ґрунтовний аналіз предметної області, реставрації обличчя на пошкоджених зображеннях. Було виявлено, що найефективнішими методами для цієї задачі є підходи, засновані на глибокому навчанні, зокрема генеративні моделі GAN. Вони забезпечують найвищу якість відновлення, особливо у випадках складних пошкоджень, де традиційні методи або класичне машинне навчання виявляються недостатніми.

На основі порівняльного аналізу існуючих програмних рішень було встановлено, що хоча вони демонструють хорошу якість відновлення, мають прості інтерфейси та підтримку AI, однак мають і недоліки, переважно працюють онлайн та потребують підключення до серверів.

2 ВИБІР ТА ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ, ЯКА ПРОЄКТУЄТЬСЯ

2.1 Вибір мови програмування

Вибір мови програмування для створення програмного забезпечення є важливим етапом у процесі його розробки. Для реалізації застосунків, що працюють на персональних комп'ютерах з операційною системою Windows, широко використовуються такі мови, як C++, Java, Python та C# [4]. Кожна з них характеризується власними перевагами, можливостями й обмеженнями, які необхідно враховувати при розробці.

Аналогічно, під час проєктування серверної частини системи постає потреба у виборі відповідної мови програмування. До найпоширеніших мов для реалізації серверної логіки належать JavaScript, C#, Ruby і Python[4]. Ці мови дають змогу створювати надійні, масштабовані та ефективні серверні рішення, відповідні до вимог конкретного проєкту.

2.1.1 Огляд C+

C++ - мова програмування яка відома своїм високим рівнем контролю над системними ресурсами та управлінням пам'яттю, що робить її найкращим вибором для створення програмного забезпечення, яке вимагає ефективності та продуктивності [5].

Вона також підтримує об'єктно-орієнтоване програмування, яке дозволяє організовувати код більш модульним та багаторазовим способом [5]. За допомогою C++ ви можете безпосередньо маніпулювати пам'яттю та отримувати доступ до обладнання, забезпечуючи безпрецедентний контроль над системними ресурсами [5].

Це дозволяє створювати програми, які безперебійно працюють у Windows, macOS та Linux [5]. Вона забезпечує низькорівневе маніпулювання

пам'яттю, що робить його ідеальним для створення програмного забезпечення системного рівня та ресурсомістких програм [5].

Крім того, C++ може бути складним та більш схильним до помилок управління пам'яттю, що робить критично важливим написання чистого та ефективного коду [5]. Крута крива навчання мови може створювати труднощі, але продуктивність та контроль, які вона пропонує, не мають собі рівних у багатьох сценаріях [5].

Переваги [5]:

- висока продуктивність та ефективність;
- міжплатформна сумісність;
- низькорівневе маніпулювання пам'яттю.

Недоліки [5]:

- крута крива навчання;
- потенціал для помилок управління пам'яттю.

2.1.2 Огляд Java

Java - це об'єктно-орієнтована мова програмування з відкритим вихідним кодом, розроблена Oracle. Java була розроблена в 1995 році групою інженерів з Sun Microsystems на чолі з Джеймсом Гослінгом. З моменту свого випуску Java пройшла кілька основних оновлень, поступово вдосконалюючи функціональність та продуктивність. Останньою стабільною версією є Java 17, випущена у 2021 році.

Гасло Java «пиши один раз, запускай будь-де» робить її ідеальним вибором для розробки комп'ютерних додатків, оскільки вона дозволяє створювати кросплатформне програмне забезпечення, яке може працювати на будь-якій операційній системі з JVM [5]. Її надійність, функції безпеки та сильна підтримка спільноти роблять її надійним варіантом для створення додатків, які можуть масштабуватися та адаптуватися до змінних технологічних ландшафтів [5].

Крім того, її суворий об'єктно-орієнтований підхід сприяє повторному використанню коду та зручності обслуговування [5]. Хоча додатки Java можуть бути більш ресурсоемними порівняно з нативними додатками, продуктивність мови значно покращилася за ці роки [5].

Переваги [5]:

- незалежність від платформи;
- сильна підтримка спільноти;
- широкі бібліотеки та фреймворки.

Проблеми [5]:

- використання пам'яті;
- проблеми з продуктивністю в певних сценаріях.

2.1.3 Огляд Python

Python - це потужна та проста у вивченні мова програмування з відкритим вихідним кодом [6]. Хоча вона використовується у багатьох галузях, Python активно застосовується для розробки веб-додатків, особливо на бекенді[6].

Python був створений Гвідо ван Россумом у 1991 році [6]. Мова отримала свою назву від популярної серії британських комедій "Монті Пайтон". З моменту свого створення Python пройшов через декілька значних версій, останньою з яких є Python 3, випущена у 2008 році [6].

Використання на бекенді У веб-розробці Python використовується в основному для створення бекенд-логіки додатків, таких як обробка запитів HTTP, взаємодія з базами даних, авторизація користувачів, маршрутизація URL-адрес, обробка файлів та інші серверні завдання [6].

Переваги [6]:

- простий та зрозумілий синтаксис, що прискорює розробку;
- багатий набір вбудованих бібліотек та модулів;
- кросплатформеність та підтримка різних операційних систем;

- динамічна типізація та автоматичне керування пам'яттю;
- велика та активна спільнота розробників.

Недоліки [6]:

- відносно повільна швидкість виконання в порівнянні з компільованими мовами;
- обмежені можливості для паралельного програмування;
- труднощі з інтеграцією в низькорівневі системи;
- проблеми з глобальною інтерпретацією імен (gil).

2.1.4 Огляд C#

C# - потужна мова програмування для створення настільних програм Windows за допомогою .NET Framework [5]. Вона пропонує багатий набір функцій для розробки графічних інтерфейсів користувача, обробки операцій введення/виведення та доступу до системних ресурсів [5]. Вона має сучасний та зрозумілий синтаксис, що спрощує написання, читання та підтримку коду[5].

C# підтримує швидку розробку програм завдяки своєму широкому набору інструментів та бібліотек [5]. Однак програми C# переважно обмежені платформою Windows, що може бути обмеженням для деяких проектів [5].

Переваги [5]:

- сучасний синтаксис;
- інтеграція з visual studio;
- швидка розробка програм.

Недоліки [5]:

- обмеженість платформою windows;
- залежність від .net framework.

2.1.5 Огляд JavaScript

JavaScript є однією з найпопулярніших мов програмування для веб-розробки [6]. Спочатку розроблена як клієнтська мова сценаріїв для додавання інтерактивності до веб-сторінок, JavaScript зараз широко використовується як на фронтенді (браузерна частина), так і на бекенді (серверна частина) [6].

JavaScript був створений Бренданом Ейхом у 1995 році в компанії Netscape і спочатку мав назву Mocha, пізніше перейменовану в LiveScript, а згодом - в JavaScript [6]. Мова базувалась на концепціях мови Self та синтаксисі C. З часом JavaScript став стандартизованим під назвою ECMAScript, який регулярно оновлюється з додаванням нових функцій та покращень [6].

На фронтенді JavaScript використовується для створення інтерактивних веб-інтерфейсів, обробки подій, анімацій, валідації форм та взаємодії з користувачем [6]. На бекенді, завдяки таким платформам, як Node.js, JavaScript може виконуватись на сервері для створення веб-серверів, API та роботи з базами даних [6].

Переваги [6]:

- простий у вивченні та використанні;
- підтримується всіма сучасними браузерами;
- велика спільнота та багатий вибір бібліотек та фреймворків;
- асинхронне програмування та подійно-орієнтована модель;
- крос-платформеність завдяки node.js.

Недоліки [6]:

- проблеми з безпекою та потенційні вразливості;
- відсутність строгої типізації (хоча це частково вирішено в typescript);
- проблеми з продуктивністю на стороні клієнта для ресурсноємких операцій.

2.1.6 Огляд Ruby

Ruby - це інтерпретована мова програмування з відкритим вихідним кодом, орієнтована на простоту і продуктивність[6]. Вона широко використовується для веб-розробки, зокрема завдяки фреймворку Ruby on Rails[6].

Ruby була створена Юкіхіро Мацуо (також відомим як Matz) у 1995 році [6]. Мова була розроблена з акцентом на продуктивність та забезпечення задоволення від програмування [6]. З моменту свого випуску Ruby постійно оновлювалася, і наразі останньою стабільною версією є Ruby 3.0, випущена у 2020 році [6].

Ruby в основному використовується для розробки бекенду веб-додатків [6]. Вона дозволяє створювати серверну логіку, обробляти HTTP-запити, взаємодіяти з базами даних, реалізовувати автентифікацію та авторизацію користувачів, а також виконувати інші завдання на стороні сервера [6].

Переваги [6]:

- простий та зрозумілий синтаксис, орієнтований на читабельність;
- динамічна типізація та автоматичне керування пам'яттю;
- потужна концепція об'єктно-орієнтованого програмування;
- ефективна робота з текстом та рядками.

Недоліки [6]:

- повільна швидкість виконання в порівнянні з компільованими мовами;
- обмежена підтримка паралельного програмування;
- менша популярність і спільнота в порівнянні з іншими мовами, такими як python або javascript;
- проблеми з горизонтальним масштабуванням веб-додатків на ruby.

2.2 Порівняльний аналіз мов програмування

На основі вищезазначеної інформації було проведено порівняльний аналіз мов програмування для розробки застосунків, що працюють на персональних комп'ютерах з операційною системою Windows, результат якого наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння мов програмування

Критерій	C++	Java	Python	C#
Продуктивність	+	+—	—	+
Підтримка GUI	+	+	+	+
Простота синтаксису	—	+	+	+
Бібліотеки сторонніх розробників	+	+	+	+
Інтеграція з Windows API	+	—	—	+
Швидкість розробки	—	—	+	+
Машинне навчання / AI	—	+	+	+
Підтримка багатопоточності	+	+	+	+

За результатами порівняльної таблиці 2.1 можна зробити висновок, що мова програмування C# разом із фреймворком .NET (WinForms або WPF) є найкращим варіантом для розробки настільних застосунків під Windows.

C# мова програмування з високим рівнем інтеграції у Windows-середовище. Завдяки повній підтримці з боку середовища розробки Visual Studio та інструментів для створення графічного інтерфейсу користувача (WinForms, WPF), ця мова дозволяє ефективно розробляти функціональні, зручні та стабільні застосунки.

Технології WinForms і WPF забезпечують можливість гнучкого дизайну інтерфейсів, а також швидкої реалізації подій і логіки програми. Розробка на C# дозволяє легко взаємодіяти з Windows API та іншими

системними ресурсами, що є важливою перевагою при створенні професійних десктопних застосунків.

Отже, для розробки ПК-застосунку було обрано мову програмування C# з використанням фреймворку .NET та технологію WPF, як найбільш доцільний варіант, що поєднує у собі зручність, продуктивність та повну сумісність із Windows.

Під час аналізу мов програмування для розробки серверної частини програми було проведено порівняння, результати якого наведено в таблиці 2.2.

Таблиця 2.2 – Порівняння мов програмування

Критерій	JavaScript	C#	Ruby	Python
Продуктивність	+	+—	—	+
Простота синтаксису	—	+	+	+
Продуктивність сервера	+	+—	—	+
Популярність у backend-розробці	+	+	—	+
Підтримка обробки зображень	—	—	—	+
Підтримка машинного навчання	—	—	—	+
Екосистема для API	+	+	+	+
Підтримка багатопоточності	+	+	+	+

За результатами порівняльної таблиці 2.2 видно, що Python має найбільше переваг серед інших мов в контексті веброзробки та підтримки роботи з зображеннями. З використанням фреймворку Flask, Python надає простий та ефективний спосіб створення серверної частини. Крім того, завдяки бібліотекам PyTorch та PIL, Python є потужним інструментом для розробки систем машиного начання та генеації зображень. Отже, для розробки серверної частини програми було обрано мову 29 програмування Python з використанням Flask, PyTorch та PIL.

2.3 Вибір середовища розробки

При розробці застосунку під Windows та серверної частини вибір правильного середовища розробки відіграє важливу роль. Існує кілька популярних середовищ, які надають розробникам необхідні інструменти та можливості для створення потужних програмних продуктів. Серед таких середовищ можна виділити PyCharm, Eclipse та Visual Studio [7].

2.3.1 Огляд PyCharm

PyCharm виділяється як широко використовуване інтегроване середовище розробки (IDE), адаптоване для програмування на Python [8]. Розроблене JetBrains, воно охоплює широкий спектр можливостей, що підвищують ефективність та задоволення від розробки на Python [8]. PyCharm це надійне та насичене функціями IDE, яке багато розробників Python використовують [8]. PyCharm пропонує інтелектуальне автодоповнення коду, яке пропонує фрагменти коду, назви змінних та назви функцій під час введення тексту, заощаджуючи ваш час та зменшуючи кількість помилок [8].

Переваги [8]:

- містить інструменти аналізу коду;
- надійна екосистема;
- підтримка спільноти;
- чудова документація;
- кросплатформність.

Недоліки [8]:

- ресурсоемність. PyCharm може не працювати безперебійно на старих або менш потужних комп'ютерах;
- велике використання дискового простору. PyCharm споживає значну кількість дискового простору.

2.3.2 Огляд Eclipse

Eclipse IDE — це багатомовне середовище розробки програмного забезпечення з відкритим вихідним кодом, яке в основному використовується для розробки Java-додатків [9]. Воно включає базовий робочий простір та систему плагінів для налаштування [9]. Воно також підтримує інші мови програмування, такі як C, C++, Python, Perl, PHP тощо, через плагіни [9]. Eclipse IDE пропонує розширені інструменти для написання коду, налагодження, тестування та розгортання, включаючи редактор коду з підсвічуванням синтаксису та функціями автозаповнення [9]. Воно підтримує системи контролю версій, такі як Git, та спільну роботу. Його висока налаштовуваність та розширюваність роблять його популярним вибором серед програмістів у всьому світі [9]. Доступні спеціальні акції для покращення вашого досвіду кодування з Eclipse IDE та подібним програмним забезпеченням [9].

Переваги [9]:

- доступ до конфігурацій та інтеграції API;
- відкритий вихідний код та безкоштовний;
- потужний інструмент для Java, C та Python;
- компактний та легкий;
- підтримка плагінів.

Недоліки [9]:

- високе споживання пам'яті;
- відсутність активного сервера;
- складний процес налагодження;
- обмежена підтримка мов, відмінних від Java;
- застарілий інтерфейс користувача.

2.3.3 Огляд Visual Studio

Visual Studio — це потужний інструмент для розробників, який можна використовувати для завершення всього циклу розробки в одному місці [10]. Це комплексне інтегроване середовище розробки (IDE), яке можна використовувати для написання, редагування, налагодження та створення коду [10]. Visual Studio містить компілятори, інструменти автодоповнення коду, систему керування версіями, розширення та багато інших функцій для покращення кожного етапу процесу розробки програмного забезпечення [10].

Завдяки різноманітності функцій та мов, що підтримуються у Visual Studio, ви можете перейти від написання своєї першої програми «Hello World» до розробки та розгортання програм [10]. Наприклад, створювати, налагоджувати та тестувати програми .NET та C++, редагувати сторінки ASP.NET у режимі веб-дизайнера, розробляти кросплатформні мобільні та настільні програми за допомогою .NET або створювати адаптивні веб-інтерфейси на C# [10].

Переваги [11]:

- допомога з кодуванням;
- платформа тестування додатків у Visual Studio допомагає розробникам з налагодженням;
- дозволяє розширені налаштування за допомогою доповнень та розширень, доступних на VS Marketplace;
- потужний функціонал можна використовувати для кількох мов, таких як C++, Python, TypeScript, SQL тощо.

Недоліки [11]:

- ресурсоемність: Visual Studio — це важке IDE;.
- крута крива навчання;
- обмежена кросплатформна розробка.

2.4 Порівняльний аналіз середовищ розробки

На основі вищезазначеної інформації було проведено порівняння середовищ розробки, результат якого наведено в таблиці 2.3.

Таблиця 2.3 – Порівняння середовищ розробки

Критерій	PyCharm	Eclipse	Visual Studio
Основне призначення	Python	Java	C#
Підтримка Python	+	—	+
Підтримка C#	+	—	+
Підтримка інших мов	+	+	+
Автодоповнення	+	+	+
Refactoring	+	—	+
Інспекція коду	+	—	+
Налагодження (debugging)	+	+	+
Великі вимоги до ресурсів	+	+	+
Простий користувацький інтерфейс	+	—	+
Кросплатформеність	+	+	+

За результатами порівняльної таблиці 2.3 можна зробити висновки, що середовище розробки PyCharm є найкращим варіантом для розробки серверної частини на Python. PyCharm надає великі можливості для розробки на Python. Він має вбудовано інтеграцію `pip`, що дозволяє встановлювати бібліотеки не виходячи з середовища. Також доступна вбудована підтримка Flask, що значно полегшує розробку саме серверної частини програми.

Для розробки ком'ютерного додатку на C# найкращим варіантом буде Visual Studio. C# — це основна мова для платформи .NET, яку розробляє сама корпорація Microsoft, і Visual Studio є її "рідним" середовищем розробки. Visual Studio надає розробнику повноцінну підтримку останніх версій мови C# і .NET SDK, включаючи розширення функціоналу за допомогою NuGet-пакетів.

2.5 Розробка структурної схеми програми

Розроблюване програмне забезпечення складається з двох компонентів: коп'ютерний застосунок та серверна частина. Структура цих компонентів та їхній зв'язок зображено на рисунку 2.1.

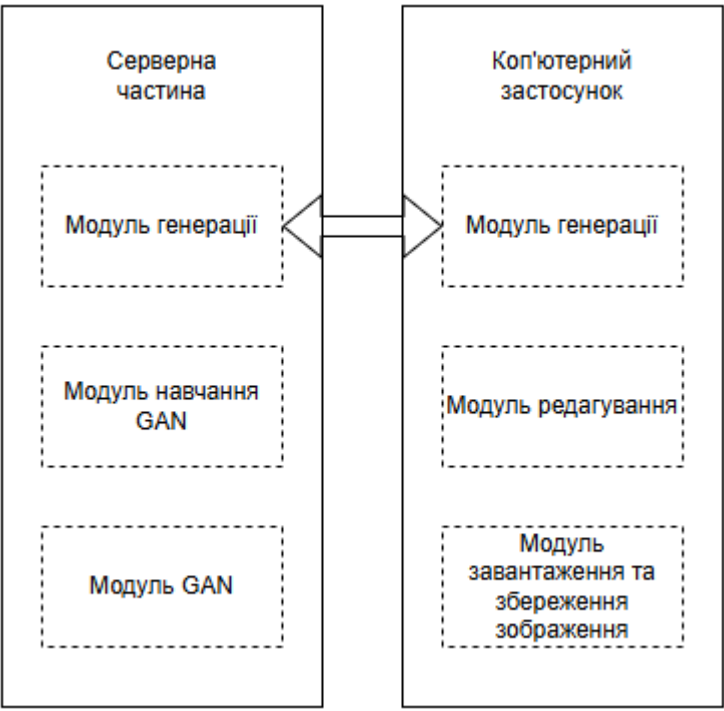


Рисунок 2.1 – Структурна схема

Опис модулів з яких складається програмне забезпечення, з визначенням їхніх основних функцій наведено в таблицях 2.4-2.5.

Таблиця 2.4 – Опис модулів серверної частини

Модуль	Опис
Модуль GAN	Модуль призначений для реалізації архітектури нейроної мережи GAN.

Продовження таблиці 2.4

Модуль	Опис
Модуль навчання GAN	Модуль призначений для тренування нейронної мережі GAN, з глибоким налаштуванням тренування.
Модуль генерації	Модуль призначений для генерації втраченої частини зображення за допомоги попередньо натренованої мережі GAN. Також модуль отримує зображення для генерації та відправляє згенероване зображення назад користувачу.

Таблиця 2.5 – Опис модулів коп'ютерного застосунку

Модуль	Опис
Модуль завантаження та збереження	Модуль призначений для завантаження зображення в додаток, та збереження згенерованого зображення.
Модуль редагування	Модуль призначений для виділення частин зображення які потрібно згенерувати.
Модуль генерації	Модуль відправляє редаговане зображення до серверу та отримує згенероване зображення з серверу.

2.6 Висновки за розділом 2

У другому розділі було розглянуто вибір інструментів для розробки програм та визначено та описано структуру програмного забезпечення. Було проведено порівняння та аналіз мов програмування та було встановлено, що для розробки коп'ютерної частини програми буде використано мову програмування C# з фреймворком .NET, а для розробки серверної частини – мову програмування Python з використанням Flask, PyTorch та PIL.

Було проведено порівняння та аналіз середовищ розробки, в ході яких було встановлено, що найкращім вибором для розробки на C# буде Visual Studio, тому що воно є “рідним” середовищем для цієї мови програмування. Для розробки на мові програмування Python найкращім варіантом буде

використання PyCharm, тому що воно має вбудовані інструменти для роботи з Python такі як рір та вбудовану інтеграцію Flask.

3 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Розробка архітектури системи

Шаблони проектування програмного забезпечення – ефективні способи вирішення задач проектування програмного забезпечення [12]. Шаблон не є закінченим зразком, який можна безпосередньо транслювати в програмний код [12]. Об'єктно-орієнтований шаблон найчастіше є зразком вирішення проблеми і відображає відношення між класами та об'єктами, без вказівки на те, як буде зрештою реалізоване це відношення [12].

Model-View-Controller (MVC) – це архітектурний шаблон, який розділяє застосунок на три основні логічні компоненти: Model, View та Controller [13].

Controller(Контролер) – це компонент, який забезпечує взаємозв'язок між представленнями та моделлю, тому він діє як посередник[13]. Контролеру не потрібно турбуватися про обробку логіки даних, він просто вказує моделі, що робити[13]. Він обробляє всю бізнес-логіку та вхідні запити, маніпулює даними за допомогою компонента модель та взаємодіє з представленням для візуалізації кінцевого результату[13].

View(Представлення) – компонент, який використовується для всієї логіки інтерфейсу користувача застосунку[13]. Він генерує інтерфейс користувача[13]. View створюються на основі даних, зібраних компонентом моделі, але ці дані не надходять безпосередньо, а через контролер. Він взаємодіє лише з контролером[13].

Model(Модель) – компонент, який відповідає всій логіці, пов'язаній з даними, з якою працює користувач[13]. Це може представляти як дані, що передаються між компонентами View та Controller, так і будь-які інші дані, пов'язані з бізнес-логікою[13]. Він може додавати або отримувати дані з бази даних[13]. Він відповідає на запити контролера, оскільки контролер не може самостійно взаємодіяти з базою даних[13]. Модель взаємодіє з базою даних і повертає необхідні дані контролеру[13].

Цей розподіл робить зменшення залежності між компонентами та розподіл відповідальності. Також розподіл допомагає писати більш чистий та структурований код, що покращує розуміння та орієнтування в програмі.

Схему архітектури застосунку з використанням шаблону проєктування наведено на рисунку 3.1.

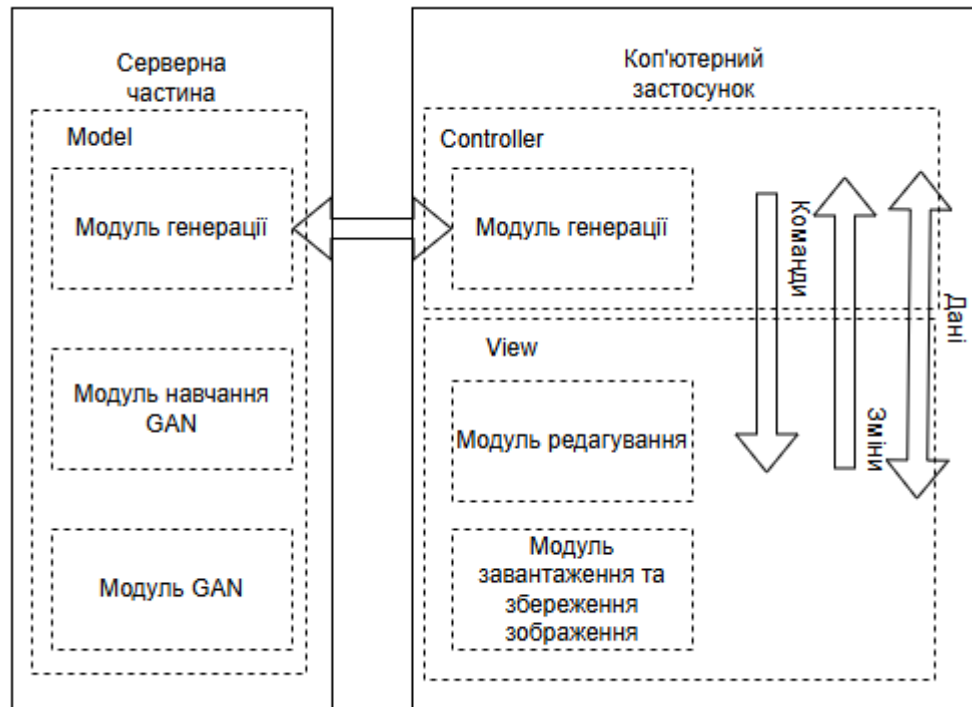


Рисунок 3.1 – Архітектура системи з використанням шаблону

Архітектура системи розроблялась з врахуванням, що система буде розвиватися далі і неможливо знати наперед, що необхідно для системи. Дизайн системи може змінюватись в ході подальшого розвинення програми.

3.2 Опис модулів розроблюваної системи

Архітектура системи містить три функціональні частини: View, Controller та Model. Модулі що містяться в View, описані в таблиці 3.1.

Таблиця 3.1 – Опис модулів що містяться в View

Модуль	Структра	Функції
Модуль редагування	ImageProcessor.cs	Збереження вихідного зображення з намальованою областю.
Модуль завантаження та збереження	ImageWork.cs	Завантаження зображення з локального сховища; Збереження згенерованого зображення до локального сховища.

Клас ImageProcessor призначений для обробки зображень у застосунку. Його головне завдання – комбінування базового зображення та графічного шару у єдине зображення, яке можна зберегти або відправити на сервер.

Клас ImageProcessor містить такі методи:

- `public static RenderCombinedImage(Image baseImage, Canvas overlayCanvas)` – створює нове зображення шляхом «малювання» базового зображення та накладеного шару на один об'єкт типу `RenderTargetBitmap`.

Як параметри функція приймає `Image baseImage` – елемент інтерфейсу WPF, що містить базове зображення та `Canvas overlayCanvas` – прозора область, на якій користувач може малювати поверх базового зображення.

Клас ImageWork призначений для роботи із файлами зображень у застосунку. Його головне завдання, завантаження зображення з диска та збереження обробленого зображення у файл. Цей клас інкапсулює функціональність взаємодії з файловою системою через діалогові вікна Windows.

Клас ImageWork містить такі методи:

- `public static BitmapImage LoadImageFromFile()` – відкриває діалогове вікно для вибору зображення з локальної файлової системи та завантажує його як об'єкт `BitmapImage`;
- `public static void SaveImageToFile(BitmapSource source)` – зберігає передане зображення `BitmapSource` у файл формату PNG, використовуючи діалогове вікно збереження.

Модулі що містяться в Controller, описані в таблиці 3.2.

Таблиця 3.2 – Опис модулів що містяться в Controller

Модуль	Структура	Функції
Модуль генерування	ServerClient.cs	Завантаження зображення на сервер для обробки. Обробка відповідей від сервера

Клас `ServerClient` призначений для взаємодії із серверною частиною застосунку. Його головна мета, відправлення зображення на сервер для обробки та отримання у відповідь нового зображення. Клас інкапсулює функціонал HTTP-запитів і конвертації зображень.

Клас `ServerClient` містить такі методи:

– `public static async Task<byte[]?> SendImageAsync(byte[] imageData)` – відправляє зображення у вигляді масиву байтів на сервер за заданою адресою і очікує відповідь.

– `public static BitmapImage ConvertToBitmapImage(byte[] imageData)` – перетворює отримане від сервера зображення у вигляді байтів у формат `BitmapImage`, який можна відобразити в інтерфейсі WPF.

Алгоритм роботи методу `public static async Task<byte[]?> SendImageAsync(byte[] imageData)` представлено далі.

Крок 1. Ініціалізація HTTP-клієнта.

`using (HttpClient client = new HttpClient())` – Створюється об'єкт `HttpClient` для здійснення HTTP-запитів.

Крок 2. Формування тіла запиту.

`using (var content = new MultipartFormDataContent())` – Об'єкт `MultipartFormDataContent` використовується для передачі файлів у форматі `multipart/form-data`.

Крок 3. Інкапсуляція зображення.

`var byteContent = new ByteArrayContent(imageData);`

```

byteContent.Headers.ContentType = new
MediaTypeHeaderValue("image/png");
content.Add(byteContent, "file", "drawn_image.png");

```

ByteArrayContent інкапсулює байтовий масив зображення. Заголовок Content-Type вказує, що передається PNG-файл. Зображення додається до тіла запиту під іменем file з умовною назвою drawn_image.png.

Крок 4. Виконання HTTP POST-запиту.

```

var response = await client.PostAsync(serverUrl,
content); - Здійснюється асинхронна передача запиту на сервер за URL-
адресою serverUrl.

```

Крок 5. Обробка відповіді.

```

if (response.IsSuccessStatusCode)
{
    return await
response.Content.ReadAsByteArrayAsync();
}

```

У разі успішного коду відповіді зчитується тіло відповіді та повертається у вигляді масиву байтів.

Крок 6. Обробка винятків.

```

catch (Exception ex)
{
    Console.WriteLine("Помилка під час надсилання: " +
ex.Message);
}

```

У разі виникнення винятку виводиться повідомлення про помилку. Метод повертає null.

Алгоритм роботи методу `public static BitmapImage ConvertToBitmapImage(byte[] imageData)` представлено далі.

Крок 1. Вхідні дані.

Метод приймає один параметр, масив байтів `imageData`, який містить зображення у закодованому вигляді.

Крок 2. Створення потоку пам'яті.

За допомогою `MemoryStream` створюється потік на основі цього масиву байтів.

```
using (MemoryStream ms = new MemoryStream(imageData)) –
```

Потік дозволяє читати зображення так, ніби воно зберігається у файлі.

Крок 3. Створення об'єкта зображення.

`BitmapImage bitmap = new BitmapImage();` - Створюється новий об'єкт `BitmapImage`, який буде заповнений даними з потоку.

Крок 4. Ініціалізація `BitmapImage`.

`bitmap.BeginInit();` - Починається процес ініціалізації.

`bitmap.StreamSource = ms;` - Призначається джерело потоку.

`bitmap.CacheOption = BitmapCacheOption.OnLoad;` -

Встановлюється параметр кешування `OnLoad`, що означає завантаження всіх даних одразу з потоку (і дозволяє закрити потік після ініціалізації).

`bitmap.EndInit();` - Завершується ініціалізація.

Крок 5. Повернення результату.

`return bitmap;` - Готовий `BitmapImage` повертається з функції.

Модулі що містяться в `Model` всі розміщені на серверній частині програми, вони описані в таблиці 3.3.

Таблиця 3.3 – Опис модулів що містяться в `Controller`

Модуль	Структура	Функції
Модуль генерування	<code>GANGenerate.py</code>	Створення бінарної маски на основі вхідного зображення; Генерація втраченої частини зображення на основі вхідного зображення та бінарної маски; Отримання із застосунку запиту з вхідним зображенням.

Продовження таблиці 3.3

Модуль	Структура	Функції
Модуль навчання GAN	GANTrain.py	Тренування моделі GAN з використанням датасету та створенням випадкових бінарних масок; Створення чекпоінтів для подальшої можливості навчання.
Модуль GAN	GAN.py	Функція створення випадкових бінарних масок. Реалізація нейронної мережі GAN.

3.3 Створення моделі для генерації

Для реалізації в програмному забезпеченні генерації виділеної частини зображення було створено GAN – генеративно-змагальну нейронну мережу.

Генеративні змагальні мережі (GAN – Generative Adversarial Nets) – це засоби для генеративного моделювання з використанням методів глибокого навчання, таких як в глибоких згорткових нейронних мережах [14]. GAN є генеративною моделлю, яка навчається з використанням двох нейронних мереж [14]. Одна модель називається моделлю «генератор» або «генеративна мережа», яка навчається генерувати нові вірогідні зразки [14]. Інша модель називається «дискримінатором» або «дискримінаційною мережею» і вчиться диференціювати генеровані приклади з реальними прикладами [14]. Вони навчаються одночасно [14]. Генеративна мережа породжує підробку, а дискримінаційна нейромережа визначає наскільки генерований зразок реалістичний, порівнюючи його з реальним зразком [14]. Генеративна і дискримінаційна нейромережі поведуть себе як противники в деякій грі, звідси й назва змагальні нейромережі [14]. Схема, як працює нейронна мережа GAN зображено на рисунку 3.2.

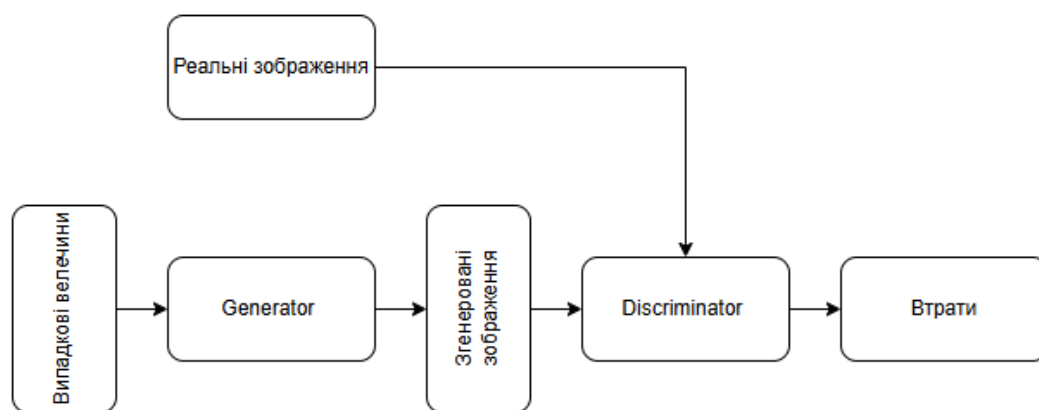


Рисунок 3.2 – Схема GAN

Модель було реалізовано з використанням фреймворку PyTorch, що забезпечує гнучкість при налаштуванні архітектури та ефективне використання GPU. Для попередньої обробки зображень використано бібліотеки NumPy та Pillow. Модель GAN, реалізована у файлі GAN.py.

Функція `random_smooth_mask` реалізує процедуру створення випадкових згладжених масок, що використовуються для моделювання частково пошкоджених зображень. Маски генеруються шляхом малювання випадкових ламаних кривих фіксованої довжини і ширини пензля з обмеженням на центральну частину зображення. Це дозволяє моделювати більш реалістичні сценарії пошкодження даних. Маска перетворюється у тензор PyTorch для подальшої обробки в нейронній мережі.

Клас `SingleFolderMaskedDataset` є спеціалізованим датасетом, що завантажує зображення з заданої директорії, генерує до кожного з них маску, а також створює частково пошкоджене зображення шляхом накладення цієї маски. Таким чином, кожен елемент вибірки складається з трьох компонентів: пошкодженого зображення, відповідної маски та оригінального зображення.

Клас `Generator` реалізує згорткову нейронну мережу типу `encoder-decoder`. Вхідним до генератора є зображення, доповнене маскою, тобто тензор з 4 каналами (RGB+маска). Енкодер виконує зменшення розмірності простору ознак за допомогою згорток, тоді як декодер здійснює реконструкцію

повнорозмірного зображення методом транспонованих згорток. На виході генератора використовується функція активації $\tanh$, що нормалізує вихідні значення до інтервалу $[-1,1]$.

Клас Discriminator реалізує згорткову мережу для класифікації зображень як справжніх чи згенерованих. Мережа складається з кількох згорткових шарів з поступовим зменшенням розмірності, завершуючись шаром, який видає ймовірність справжності. На виході формується скалярна оцінка, яка відображає впевненість дискримінатора у «реальності» зображення.

3.3.1 Алгоритм навчання

Для початку роботи з GAN мережею потрібно з'ясувати що саме вона повинна генерувати. Для точної генерації області обличчя, яку потрібно відновити, потрібно зробити датасет з зображень обличчя людини одноклової роздільної здатності. Прикладом такого датасету є CelebA-HQ [15], який містить зображення відомих людей з різною роздільною здатністю. Вміст датасету зображено на рисунку 3.3.

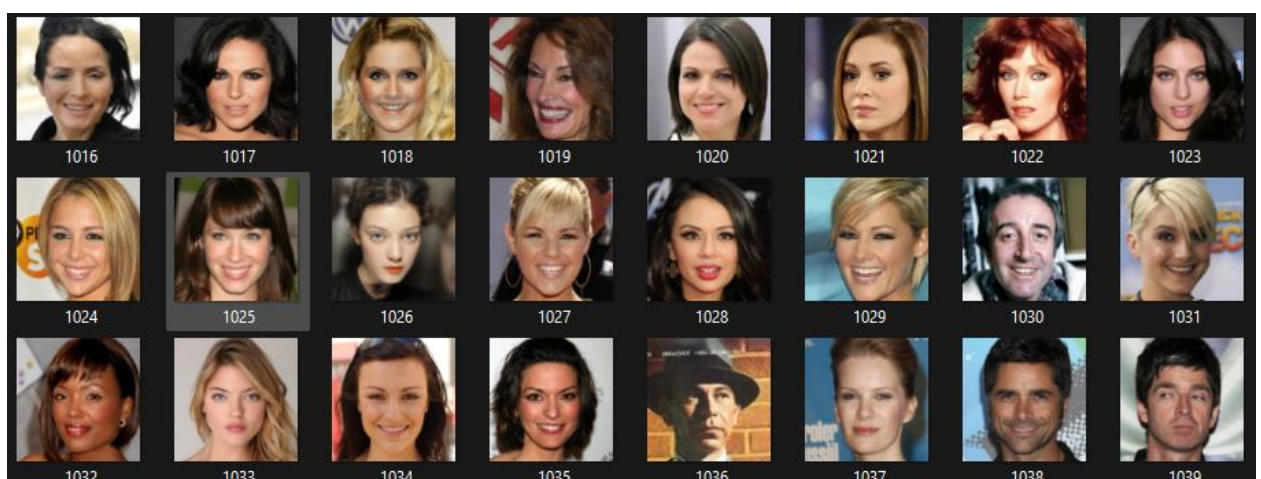


Рисунок 3.3 – Вміст датасету

Після завантаження та попередньої обробки зображень, модель проходить фазу навчання, в якій реалізується класичний підхід змагальних генеративних мереж GAN. У якості вхідних даних генератор отримує зображення з випадково згенерованими масками, що визначають області, які потрібно відновити. Маски створюються за допомогою криволінійних траєкторій різної товщини з обмеженням області нанесення, що імітує часткову втрату або перекриття інформації на зображенні.

На кожній ітерації навчання виконується двоетапна оптимізація: спочатку оновлюються ваги дискримінатора, потім генератора. Дискримінатор навчається розрізняти між справжніми та згенерованими зображеннями. Генератор, у свою чергу, намагається обдурити дискримінатор, генеруючи реалістичні заповнення маскованих областей.

Функція втрат генератора складається з двох компонентів:

- змагальна втрата (adversarial loss), яка стимулює генератор створювати правдоподібні зображення, що дискримінатор сприймає як реальні;
- реконструкційна втрата (reconstruction loss), яка вимірює L1-відхилення між згенерованими і справжніми пікселями в межах замаскованої області, забезпечуючи збереження контекстної узгодженості.

Модель тренується протягом фіксованої кількості епох. Після кожної епохи результати генератора візуалізуються для фіксованого набору зразків, що дозволяє відслідковувати якість інпейнтингу на різних стадіях тренування. Періодично зберігаються контрольні точки, що дає змогу відновити навчання з попереднього стану у разі переривання. Модель навчання GAN реалізована в GANTrain.py.

Для кращого навчання реалізовано функції для збереження та завантаження чекпоінтів:

- `save_checkpoint` – зберігає поточні стани генератора (netG) та дискримінатора (netD), а також оптимізаторів у файл, що дозволяє відновити навчання з тієї ж точки у випадку переривання процесу;

– `load_checkpoint` – завантажує вказаний чекпоінт, відновлюючи ваги моделей та параметри оптимізаторів, повертаючи номер останньої збереженої епохи.

Для підготовки вибірки, застосовується клас `SingleFolderMaskedDataset`, який автоматично генерує маски випадкового розміру (`mask_size=16`) для моделювання задачі.

```
dataset = SingleFolderMaskedDataset(dataroot,
transform, mask_size=16)

dataloader = DataLoader(dataset, batch_size=64,
shuffle=True, num_workers=2)
```

Попередня обробка зображень включає масштабування, обрізання по центру, перетворення до тензора та нормалізацію.

```
transform = transforms.Compose([
    transforms.Resize(64),
    transforms.CenterCrop(64),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5,
0.5))
])
```

Для ініціалізації моделі, генератор (`netG`) та дискримінатор (`netD`) передаються на GPU. Для оптимізації застосовується метод Adam з типовими гіперпараметрами.

```
netG = Generator().to(device)
netD = Discriminator().to(device)
optimizerG = optim.Adam(netG.parameters(), lr=0.0002,
betas=(0.5, 0.999))
optimizerD = optim.Adam(netD.parameters(), lr=0.0002,
betas=(0.5, 0.999))
```

Основний цикл навчання реалізовано як ітеративний процес, який складається з двох основних етапів на кожній ітерації: оновлення

дискримінатора та оновлення генератора. Цей процес повторюється для кожної міні-вибірки у кожній епісі.

```
for epoch in range(start_epoch, num_epochs):
    for i, (masked_imgs, masks, real_imgs) in
enumerate(dataloader):
```

На кожній ітерації завантажуються наступні дані:

`masked_imgs` – зображення з випадково маскованими ділянками.

`masks` – бінарні маски, що визначають область відновлення.

`real_imgs` – оригінальні незмінені зображення.

На кожній ітерації відбувається підготовка до навчання – перенесення даних на обране обчислювальне середовище та обнулення градієнтів обох мереж.

```
masked_imgs, masks, real_imgs = masked_imgs.to(device),
masks.to(device), real_imgs.to(device)
```

```
optimizerD.zero_grad()
```

```
optimizerG.zero_grad()
```

Навчання дискримінатора починіється з оцінки справжніх зображень.

```
output_real = netD(real_imgs)
```

```
label_real = torch.ones_like(output_real)
```

```
loss_real = nn.BCELoss()(output_real, label_real)
```

`output_real` – передбачення дискримінатора для справжніх зображень.

`label_real` – мітка класу 1 (реальне зображення).

`loss_real` – функція втрат Binary Cross Entropy (BCE) для справжніх зображень.

Для оцінки згенерованих зображень генератор створює зображення на основі входу `masked_imgs + masks`. Оцінка дискримінатора здійснюється без оновлення градієнтів генератора.

```
gen_input = torch.cat((masked_imgs, masks), dim=1)
```

```
fake_imgs = netG(gen_input).detach()
```

```
output_fake = netD(fake_imgs)
```

```
label_fake = torch.zeros_like(output_fake)
loss_fake = nn.BCELoss()(output_fake, label_fake)
```

Сумарна втрата дискримінатора забезпечує оновлення ваг дискримінатора таким чином, щоб він краще розрізняв справжні та фальшиві приклади.

```
loss_D = loss_real + loss_fake
loss_D.backward()
optimizerD.step()
```

На другому етапі ваги генератора оновлюються з метою обману дискримінатора та точного відновлення маскованих областей.

```
gen_input = torch.cat((masked_imgs, masks), dim=1)
fake_imgs = netG(gen_input)
```

Адвесаріальна втрата – генератор намагається змусити дискримінатор визнати згенеровані зображення як справжні (мітка = 1).

```
output = netD(fake_imgs)
loss_G_adv = nn.BCELoss()(output, label_real)
```

Реконструкційна втрата обчислюється лише на пікселях, що покриті маскою, тобто лише на ділянці, яку слід відновити.

```
loss_G_rec = nn.L1Loss()(fake_imgs * masks, real_imgs *
masks)
```

Комбінована функція втрат поєднує BCE для змагання з дискримінатором та L1 для забезпечення точності відновлення.

```
loss_G = loss_G_adv + loss_G_rec
loss_G.backward()
optimizerG.step()
```

Після проходження всіх епох навчання результати зберігаються у файлі формату .pth, який потім використовується для генерації зображення без попереднього навчання.

3.3.2 Результати навчання

Після 100 епох навчання на наборі даних який складається з тридцяти тисяч зображень обличч відомих людей, модель досягла стабільної генерації, що зберігає анатомічну структуру обличчя. Графік навчання зображено на рисунку 3.4 На рисунку 3.5 зображено реальні зображення з накладеною маскою, бінарні маски для відновлення та результати відновлення при 1 епохсі навчання.

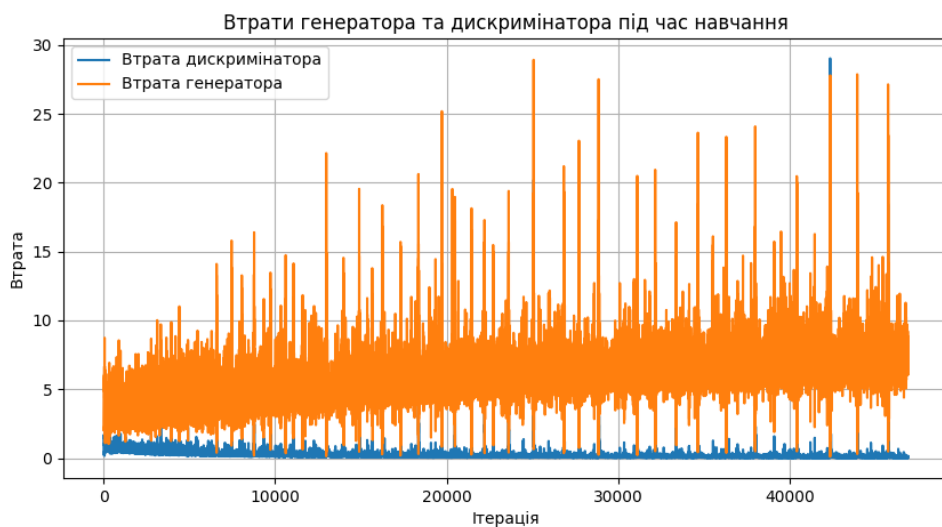


Рисунок 3.4 – Графік навчання



Рисунок 3.5 – Результат генерації на 1 епісі

На рисунку 3.6 зображено реальні зображення з накладеною маскою, бінарні маски для відновлення та результати відновлення при 50 епохах навчання.



Рисунок 3.6 – Результат генерації на 50 епохах

На рисунку 3.7 зображено реальні зображення з накладеною маскою, бінарні маски для відновлення та результати відновлення при 100 епохах навчання.



Рисунок 3.7 – Результат генерації на 100 епохах

3.4 Рішення щодо розробки інтерфейсу

Інтерфейс користувача розроблений з урахуванням принципів простоти, зручності та інтуїтивності. Застосунок складається з одного вікна.

Застосунок реалізовано у вигляді одновіконного інтерфейсу, що спрощує взаємодію та виключає необхідність перемикання між вкладками чи формами. Головне вікно (рис. 3.8) є центральним елементом усієї взаємодії користувача із застосунком і містить стандартні компоненти WPF:

- Buttons – реалізують доступ до основних функцій програми;
- Image – слугує для виведення завантаженого або обробленого зображення без спотворень масштабу;
- Canvas – прозорий шар, розташований над зображенням, на якому реалізована можливість малювання. Це дає змогу вручну позначати області зображення перед відправкою на серверну обробку;
- Slider – дозволяє гнучко регулювати розмір інструменту малювання, що забезпечує точніший контроль над створенням маски.

Використання стандартних елементів WPF дозволяє досягти високої адаптивності інтерфейсу до різних розмірів екранів, а також забезпечує можливість подальшого розширення функціоналу без значного перегляду архітектури застосунку.

Завантажити зображення Очистити малюнок Розмір кисті: 2 Згенерувати Зберегти зображення



Рисунок 3.8 – Головне вікно застосунку

3.5 Обробка виняткових витуацій

Для підвищення надійності та стабільності застосунку реалізовано механізм обробки виняткових ситуацій, які можуть виникати під час роботи. Обробка помилок здійснюється за допомогою стандартних засобів мови програмування C#, конструкцій try-catch для перехоплення винятків та компоненту MessageBox для інформування користувача про характер проблеми.

У застосунку реалізована обробка найпоширеніших виняткових ситуацій:

- помилки під час взаємодії з серверною частиною;
- помилки під час завантаження та збереження зображення.

У кожному з випадків користувачу виводиться інформативне повідомлення за допомогою елемента MessageBox.Show(), яке містить короткий опис проблеми. Такий підхід дозволяє не лише запобігти аварійному завершенню роботи програми, а й забезпечити зручність у виявленні та усуненні помилок. Приклад повідомлення про неможливість збереження обробленого зображення, надано на рисунку 3.9.

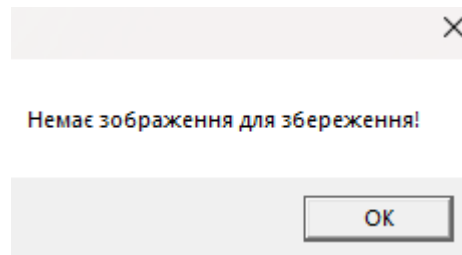


Рисунок 3.9 – Повідомлення про відсутність обробленого зображення

Крім того, в застосунку реалізовано базову валідацію даних, що дозволяє запобігти некоректному введенню та зменшує ймовірність виникнення виняткових ситуацій на ранніх етапах взаємодії з користувачем.

У функціоналі завантаження зображення реалізовано базову валідацію даних шляхом фільтрації дозволених типів файлів у діалоговому вікні OpenFileDialog. Це забезпечує захист програми від спроб завантаження файлів недопустимих або непідтримуваних форматів, що могли б призвести до помилок обробки чи порушення роботи застосунку.

У параметрі Filter визначено перелік допустимих розширень:

```
dlg.Filter = "Image Files|*.jpg;*.png;*.bmp";
```

Це означає, що користувач під час вибору файлу в діалоговому вікні зможе бачити та вибрати лише ті файли, які мають розширення .jpg, .png або .bmp.

Крім цього, функція додатково перевіряє, чи справді користувач обрав файл (`dlg.ShowDialog() == true`). У випадку скасування вибору повертається значення `null`, що дозволяє коректно обробити ситуацію відсутності зображення у викликах цієї функції.

Таким чином, уже на етапі вибору файлу реалізовано попередню перевірку допустимості формату вхідних даних.

3.6 Висновки за розділом 3

У третьому розділі було розглянуто основні архітектурні рішення та структуру компонентів програмного забезпечення. В якості базового шаблону проєктування було обрано архітектуру Model-View-Controller (MVC), що дозволило чітко розділити логіку взаємодії з користувачем, обробку даних та роботу із серверною частиною.

Було детально описано реалізовані модулі кожного з компонентів MVC:

- у View реалізовано модулі для завантаження, збереження та редагування зображень;
- у Controller – функціонал для надсилання зображень на сервер і обробки відповідей;
- у Model – серверна частина з реалізацією та навчанням генеративно-змагальної нейронної мережі (GAN) для відновлення зображень.

Було створено та навчано генеративно-змагальної нейронної мережі (GAN), яка виконує генерацію втрачених частин зображень. Навчання GAN здійснювалось на попередньо підготовленому датасеті зображень облич.

Також розроблено зручний та інтуїтивно зрозумілий інтерфейс користувача з можливістю малювання на зображенні та керування обробкою. Передбачено обробку виняткових ситуацій, що підвищує надійність застосунку.

4 КРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення та умови застосування програми

4.1.1 Призначення програми

Призначенням програми є генерація пошкодженої чи відсутньої частини обличчя людини.

4.1.2 Функції програми

Програма виконує наступні функції:

- завантаження зображення до застосунку;
- створення білої маски для генерування;
- відправка зображення на сервер;
- генерація частини зображення перекритого маскою;
- збереження зображення на локальному сховищі.

4.1.3 Умови застосування програми

Мінімальні вимоги до програмних та апаратних засобів для нормального функціонування застосунку наступні:

- 64-bit Windows 10 / 11;
- двоядерний процесор або вище частотою 1,44 ГГц;
- 50 Мб вільного місця на диску;
- 4 Гб оперативної пам'яті і більше;
- доступ до інтернету.

Мінімальні вимоги до програмних та апаратних засобів для нормального функціонування серверної частини програми наступні:

- Python 3.9, Flask 3.1.0, PyTorch 2.7.0, NumPy 2.2.5, Pillow 11.2.1, Matplotlib 3.10.3;
- 64-bit Windows 10 / 11;

- двоядерний процесор або вище частотою 1,44 ГГц;
- 5 Гб вільного місця на диску;
- 4 Гб оперативної пам'яті і більше;
- доступ до інтернету.

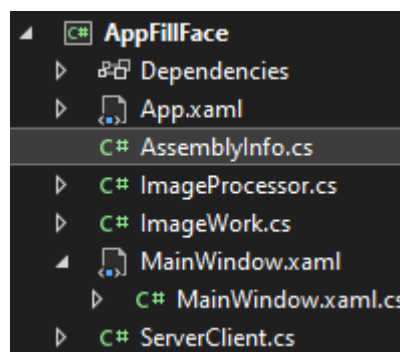
4.2 Характеристика програми

Застосунок реалізовано за допомогою фреймворку .NET мови програмування C# в середовищі розробки Microsoft Visual Studio.

Серверну частину реалізовано за допомогою мови програмування Python з використанням бібліотек Flask, PyTorch, NumPy, Pillow, Matplotlib, в середовищі розробки PyTorch.

4.2.1 Структура програми

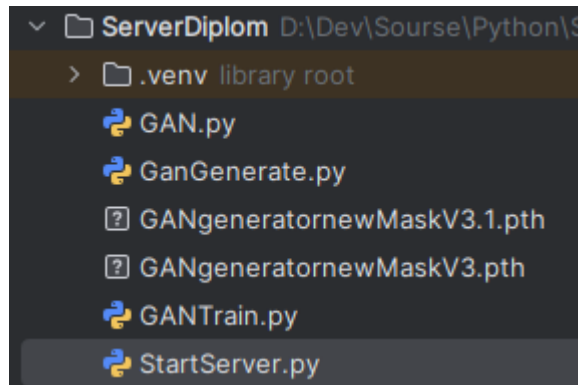
Структура застосунку складається з 4 класів та 1 файлу XAML (eXtensible Application Markup Language), з якими пов'язано 1 клас. Структуру файлів мобільного застосунку зображено на рисунку 4.1.



Рисунко 4.1 – Структура файлів застосунку

Структура серверної частини програми складається з 4 файлів .py, 2 з яких відповідають за створення моделі для генерації зображення, а інші — за опрацювання HTTP-запитів та генерацію. 2 файли .pth відповідають за

налаштування моделі генерації зображення. Структуру файлів серверної частини програми зображено на рис. 4.2.



Рисунко 4.2 – Структура файлів сервера

4.3 Звертання до програми

Перед запуском програми потрібно упевнитися, що пристрій відповідає системним вимогам встановленої програми. Також потрібно перевірити наявність усіх файлів застосунку. Для коректної роботи HTTP-запитів потрібно впевнитися, що середовище розробки містить необхідні компоненти для роботи з Інтернетом.

Для запуску проєкту застосунку в Microsoft Visual Studio потрібно відкрити файл AppFillFace.sln.

Для запуску серверу в застосунку PyCharm потрібно відкрити файл ServerStart.py

Вихідними файлами застосунку та серверної частини є AppFillFace.exe та StartServer.py, GAN.py, GANTrain.py та GANGenerate.py.

4.4 Вхідні та вихідні данні

4.4.1 Вхідні дані

Вхідними даними програми є зображення частину якого користувач хоче згенерувати.

4.4.2 Вихідні дані

Вихідними даними є зображення яке було згенеровано.

4.5 Повідомлення програмісту

Програма не виводить жодних повідомлень для програміста.

4.6 Висновки за роздіом 4

У цьому розділі було детально описано призначення, функціональні можливості та умови застосування створеного програмного забезпечення, призначеного для генерації пошкоджених або відсутніх частин обличчя на зображеннях. Розглянуто мінімальні системні вимоги як для застосунку, так і для серверної частини, реалізованої за допомогою бібліотек Python.

У розділі також описано порядок запуску програми, а також формат вхідних та вихідних даних. Вхідними даними є зображення з пошкодженою або відсутньою ділянкою обличчя, а вихідними — згенероване зображення з відновленою ділянкою.

5 КЕРІВНИЦВО ОПЕРАТОРА

5.1 Призначення програми

Призначенням програми є генерація пошкодженої чи відсутньої частини обличчя людини

5.2 Умови виконання програми

5.2.1 Вимоги програми

Мінімальні вимоги до програмних та апаратних засобів для нормального функціонування застосунку описані в пункті 4.1.3.

5.2.2 Вмоги до користувача

Основною вимогою до користувача є володіння основами роботи з пристроєм.

5.3 Виконання програми

5.3.1 Запуск програми

Для запуску програми потрібно запустити на виконання файл AppFillFace.exe. Після запуску відкриється головне вікно програми (рис. 5.1).

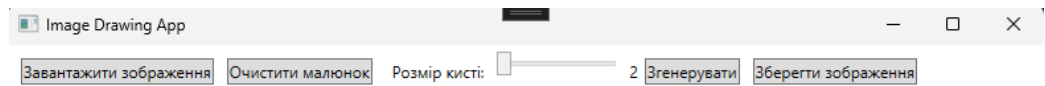


Рисунок 5.1 – Головне вікно

5.3.2 Робота з програмою

Після запуску програми в головному вікні можна завантажити зображення з частиною для генерації натиснувши на кнопку “Завантажити зображення”. Після натискання кнопки відкривається діалогове вікно з можливістю вибору зображення (рис. 5.2). Для вибору файлу достатньо 2 рази клікнути по ньому правою кнопкою миші.

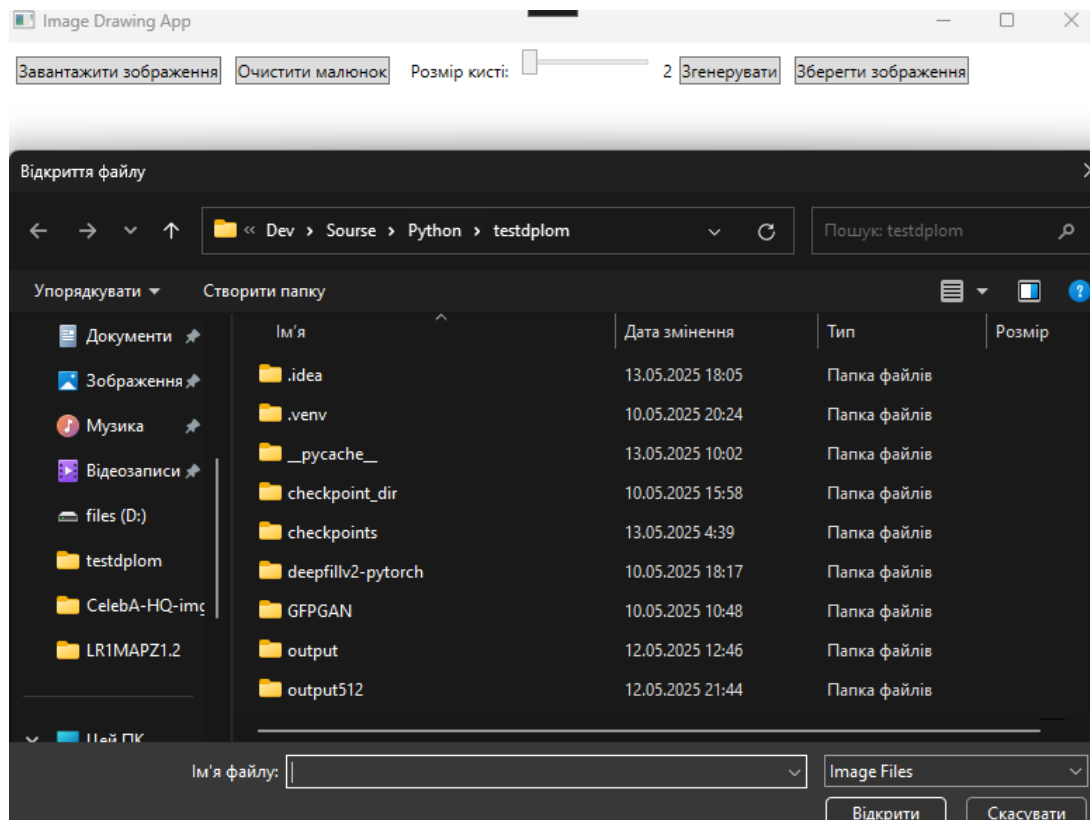


Рисунок 5.2 – Вибір зображення для завантаження

Після завантаження зображення воно з'явиться у лівій частині вікна (рис. 5.3). Для виділення області яку треба перегенерувати треба обрати розмір кисті та замалювати область на зображенні (рис. 5.4). Якщо область була не коректно замальована, можна натиснути кнопку “Очистити малюнок” для очищення області.

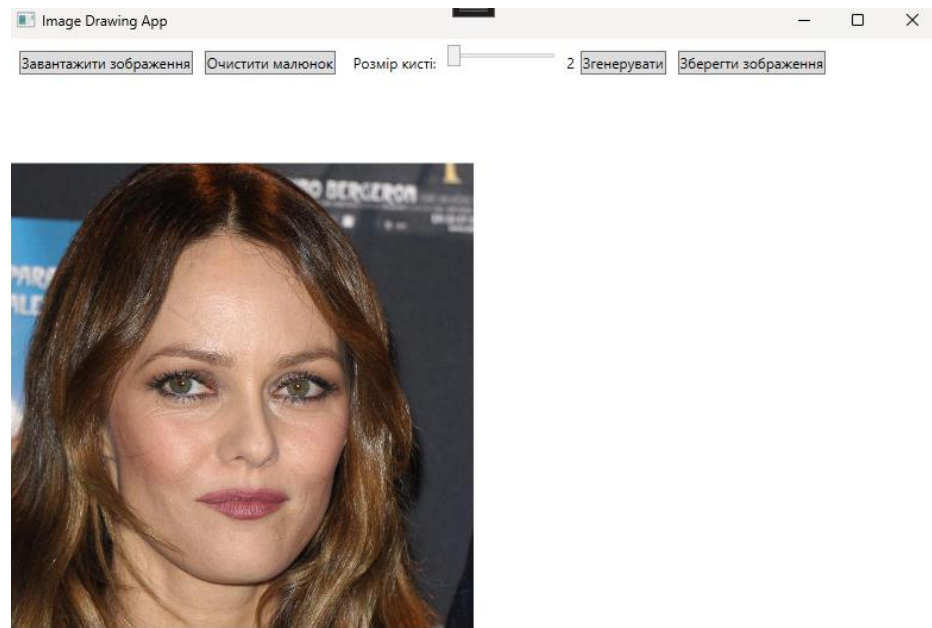


Рисунок 5.3 – Завантаження зображення в застосунок

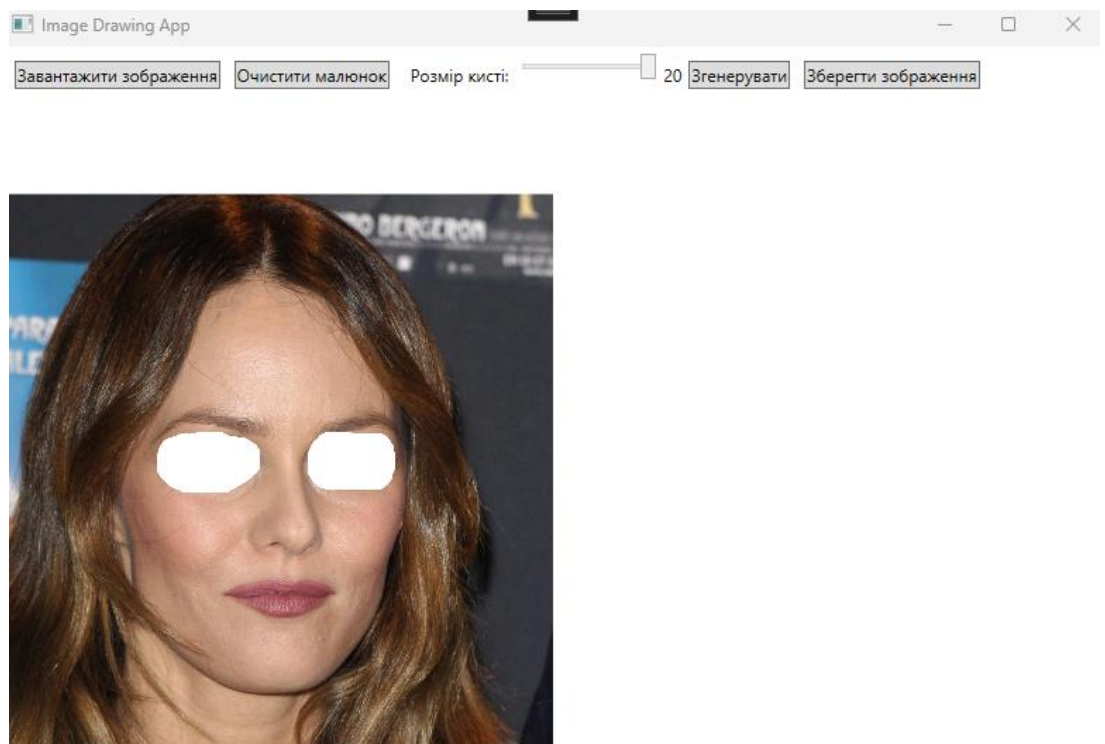


Рисунок 5.4 – Завантаження зображення з областю генерації

Після того як потрібна область генерації була замальована, потрібно написати кнопку “Згенерувати”. Результат генерації замальованої області з’явиться в правій частині вікна (рис. 5.5).

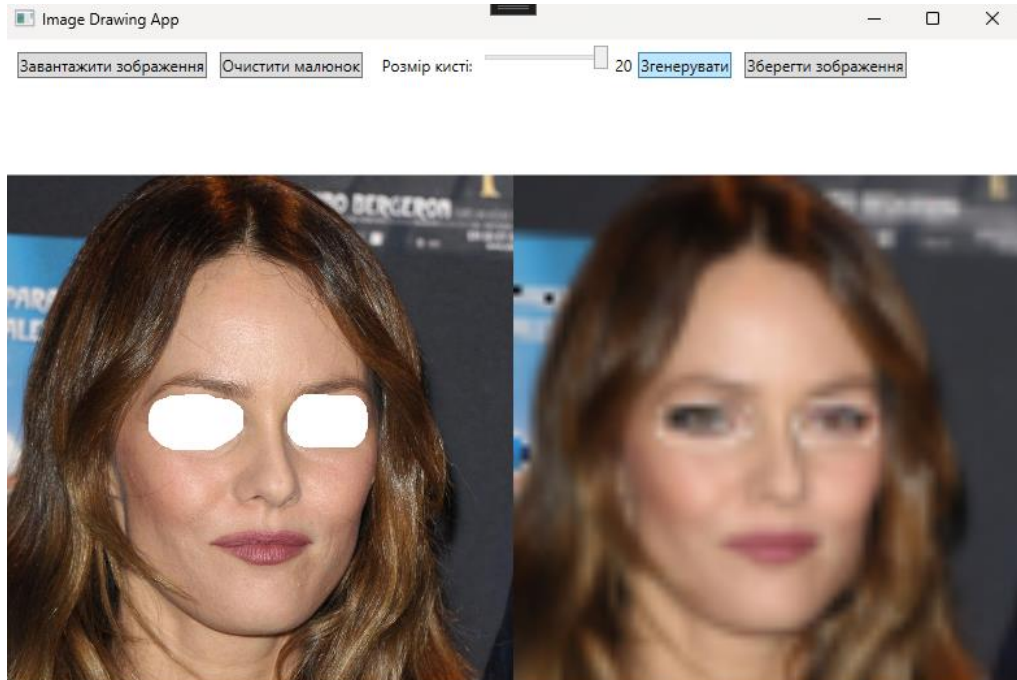


Рисунок 5.5 – Результат генерації

Якщо зображення задовольняє, то його можна зберегти на локальному сховищі натиснувши кнопку “Зберегти зображення”. Після натискання кнопки з’являється діалогове вікно з вибором директорії та назви для збереження файлу (рис. 5.6).

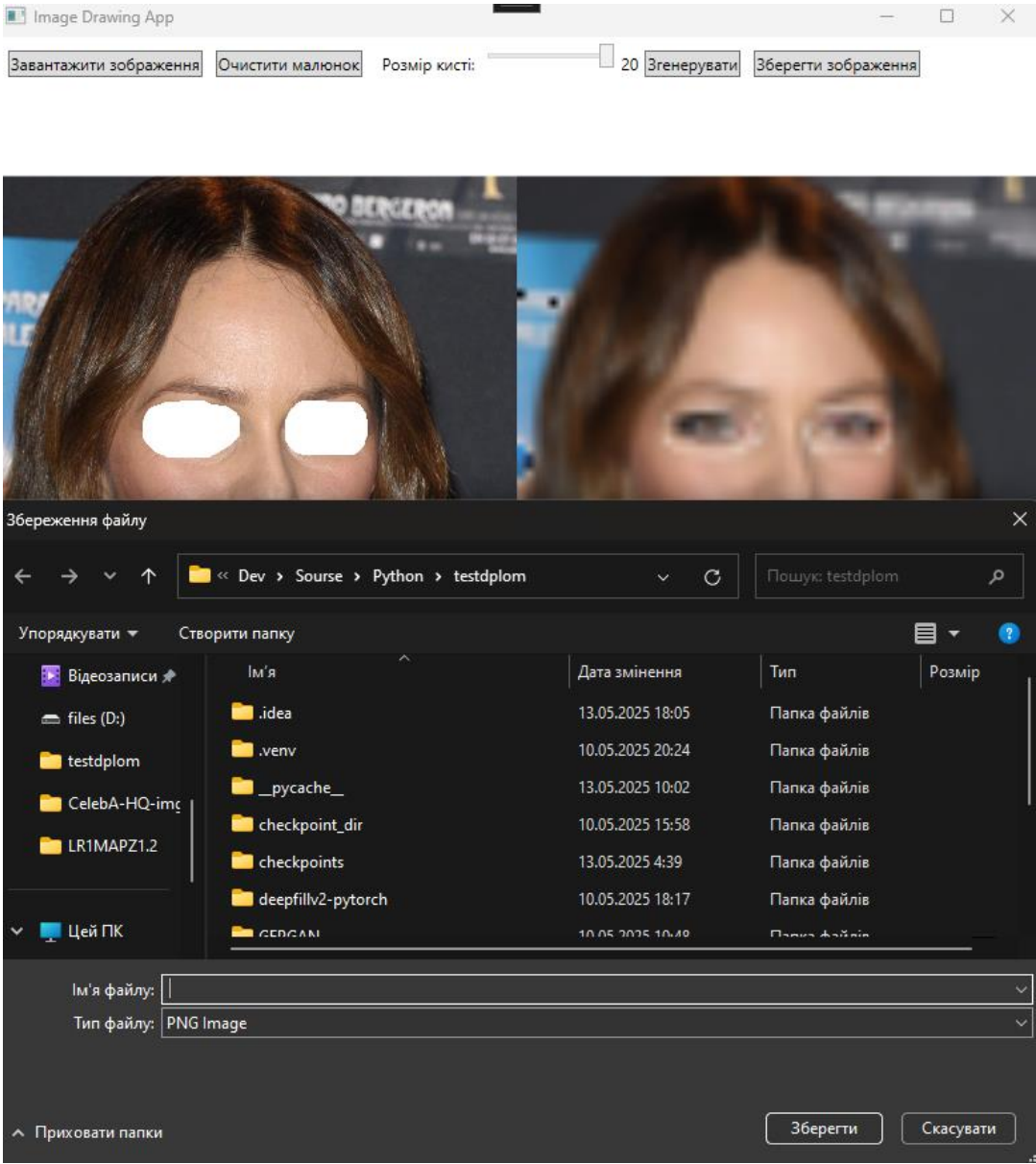


Рисунок 5.6 – Збереження результату генерації

5.4 Повідомлення оператору

Під час взаємодії з програмою оператор може побачити наведені повідомлення.

“Не вдалося згенерувати зображення” – повідомлення виникає коли в застосунок не завантажено зображення для генерування (рис. 5.7). Оператору варто завантажити зображення перед тим, як відправляти його на обробку.

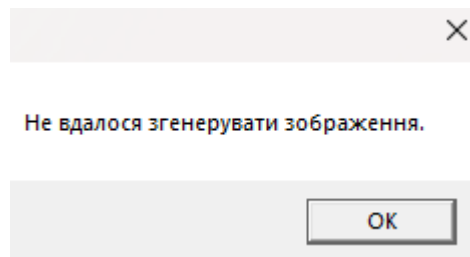


Рисунок 5.7 – Повідомлення про не завантажене зображення

“Немає зображення для збереження” – повідомлення виникає коли в застосуноку не має зображення яке було оброблено(рис. 5.8). Оператору варто зробити генерування зображення, після чого вже намагатися його зберегти.

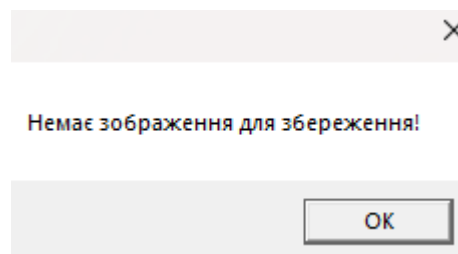


Рисунок 5.8 – Повідомлення про відсутність обробленого зображення

“Не вдалося отримати оброблене зображення” – повідомлення виникає коли застосунок не може з’єднатися з сервером(рис. 5.9). Оператору варто спробувати пізніше, коли будуть відновлені сервера.

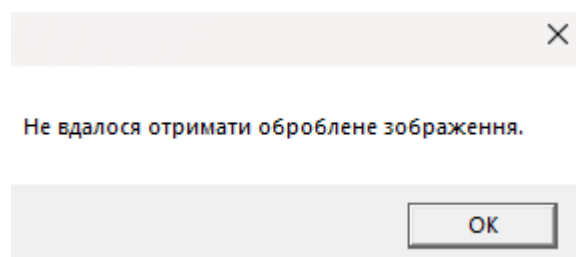


Рисунок 5.9 – Повідомлення про відсутність з’єднання

“Зображення збережено!” – повідомлення виникає коли оператор успішно зберігає оброблене зображення на локальному сховищі(рис. 5.10).

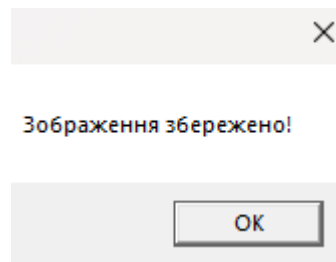


Рисунок 5.10 – Повідомлення про вдале збереження

5.5 Висновки за розділом 5

У цьому розділі було детально описано призначення, умови виконання та порядок роботи з програмою з точки зору кінцевого користувача (оператора). Програма забезпечує інтуїтивно зрозумілий інтерфейс, що дозволяє користувачеві з мінімальними навичками роботи з комп'ютером швидко освоїти її використання.

Були наведені покрокові інструкції запуску та використання основного функціоналу: завантаження зображення, створення області для генерації, процес генерації та збереження результату. Окрему увагу приділено повідомленням, які може отримати оператор під час взаємодії з програмою, що дозволяє своєчасно реагувати на можливі помилки чи недоліки у використанні.

6 ТЕСТУВАННЯ І РЕЗУЛЬТАТИ

6.1 Опис методики проведення тестування

Тестування виконувалось за таким сценарієм:

- запуск застосунку;
- завантаження зображення в додаток;
- малювання області генерації;
- очищення області генерації;
- генерування зображення;
- збереження згенерованого зображення.

Під час тестування було перевірено основні функціональні можливості застосунку, правильність додавання, відображення зображень і т. ін.

Тестування виконувалось на ПК з наступними характеристиками:

- операційна система: Windows 11 Pro 64-bit;
- процесор: Intel Core i5-1135G7, 4 ядра, 2.4 ГГц (Turbo до 4.2 ГГц);
- оперативна пам'ять: 8 ГБ DDR4;
- накопичувач: SSD 256 ГБ;
- графічна система: Intel Iris Xe Graphics;
- інтернет-з'єднання: стабільне широкосмугове підключення (не менше 10 Мбіт/с);
- Програмне забезпечення: .NET Desktop Runtime 6.0, Python 3.9, Flask 3.1.0 , PyTorch 2.7.0 ,NumPy 2.2.5 ,Pillow 11.2.1, Matplotlib 3.10.3.

Мета використання цієї платформи – перевірити стабільність роботи застосунку в умовах, близьких до середньостатистичних для користувача, з використанням як клієнтської, так і серверної частини програми.

6.2 Результат використання розробленого програмного забезпечення

Для тестування розробленого програмного забезпечення на ПК було встановлено застосунок. Після запуску програми в головному вікні для завантаження зображення для генерації було натисно на кнопку “Завантажити зображення”. Після натискання кнопки та вибору файлу в діалоговому вікні зображення з’явилося в лівій частині вікна(рис. 6.1).

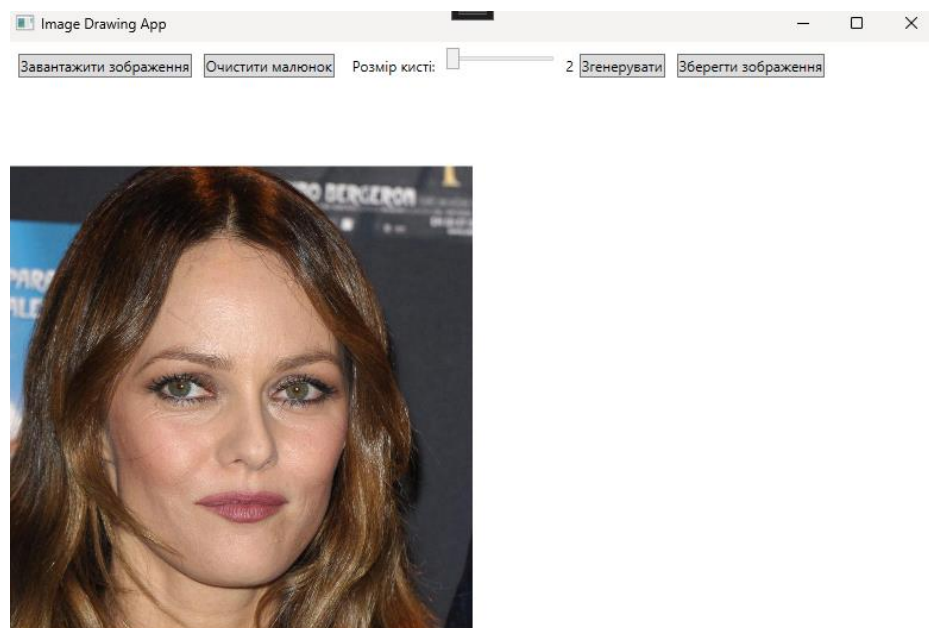


Рисунок 6.1 – Завантаження зображення

Для виділення області яку треба регенерувати було обрано розмір кисті 20 та замальовано область на зображенні(рис. 6.2).

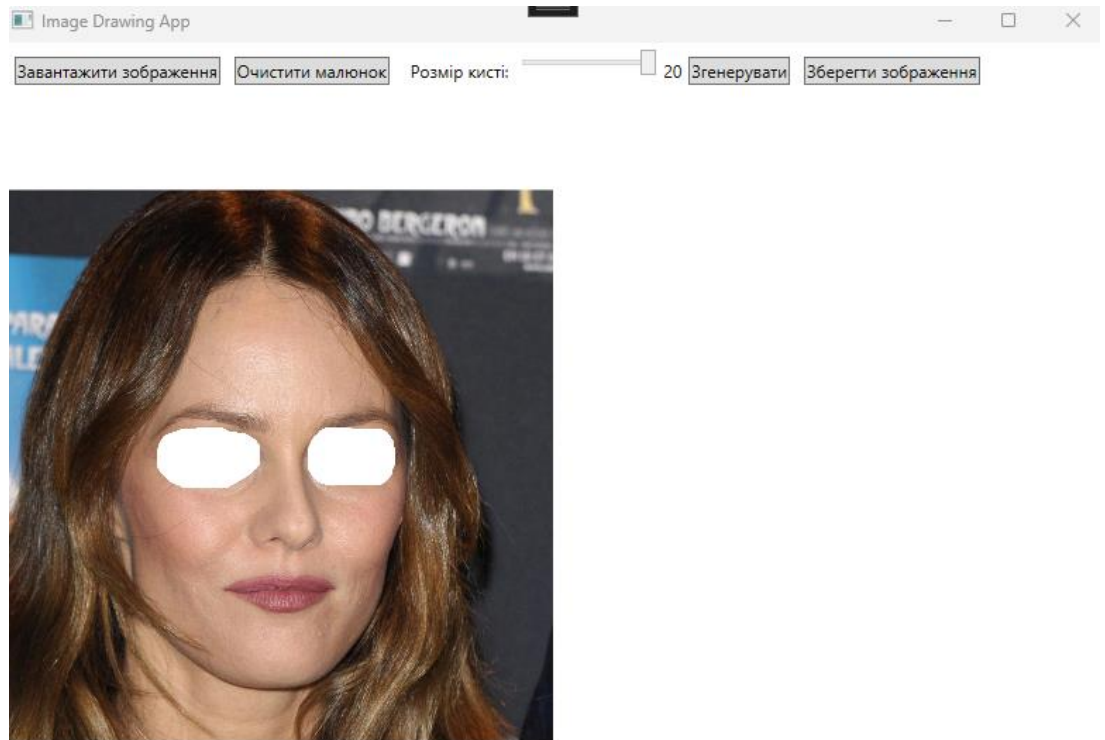


Рисунок 6.2 – Зображення з областю для генерації

Для того щоб очистити зображення від виділеної області, було натиснуто кнопку “Очистити”. Після натискання зображення в лівій частині було повернуто в початковий стан (рис. 6.3).

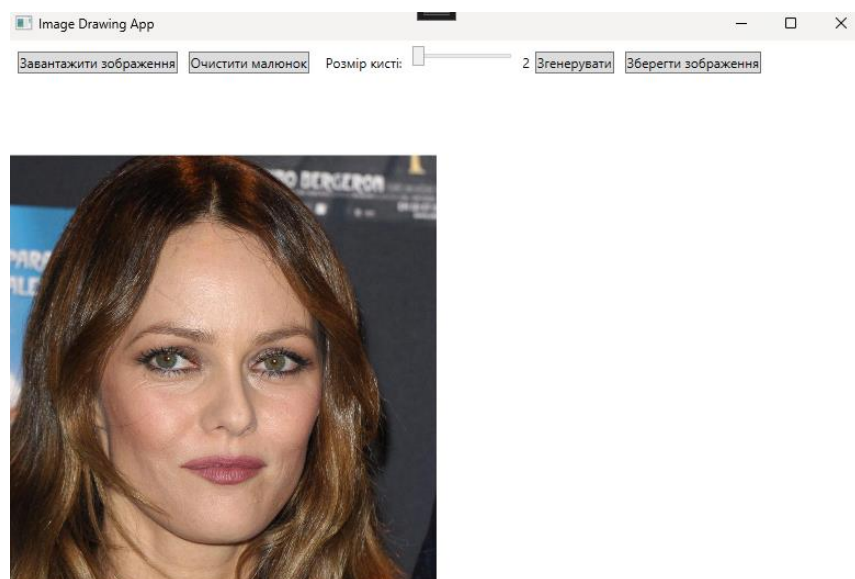


Рисунок 6.3 – Очищене зображення

Після того як потрібна область генерації була замальована, було написано кнопку “Згенерувати”. Результат генерації в замальованій області з’явився в правій частині вікна (рис. 6.4).

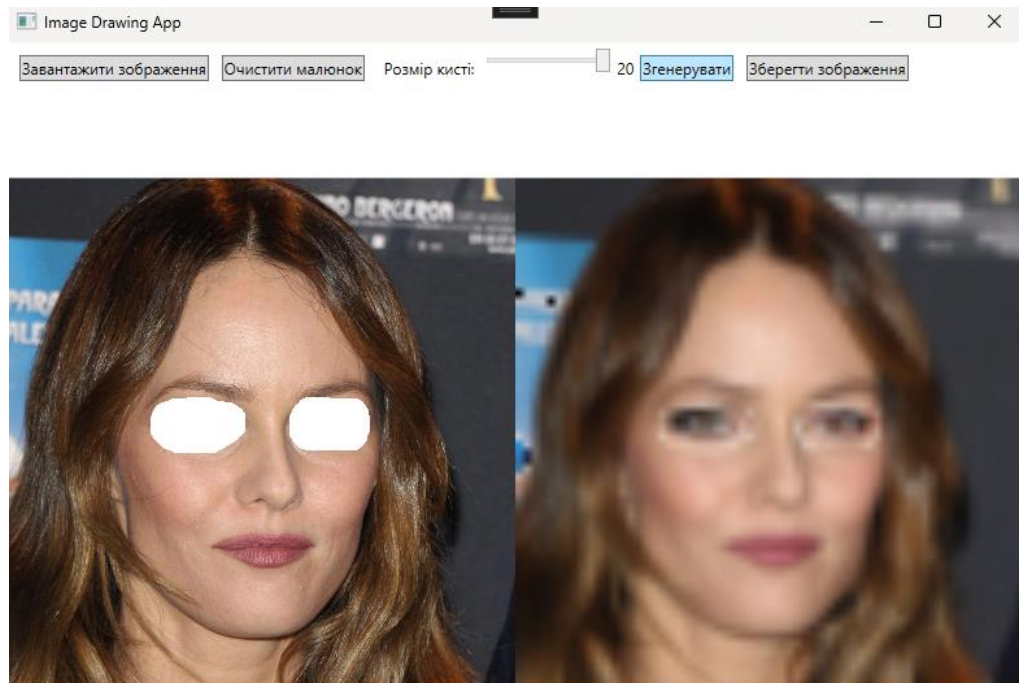


Рисунок 6.4 – Результат генерації

Для збереження згенерованого зображення було натиснуто кнопку “Зберегти зображення”. Після натискання кнопки в діалоговому вікні було обрано місце на локальному сховищі та test1 як назву файлу зображення(рис. 6.5).

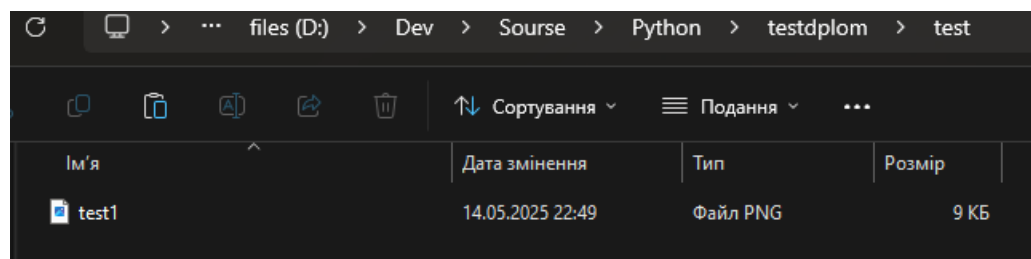


Рисунок 6.5 – Результат збереження в локальному сховищі

Під час тестування застосунку були виконані різні сценарії, включаючи перевірку функціональності, стабільності, взаємодії з серверною частиною, коректного зберігання та обробки даних. Результати тестування було зведено в таблицю 6.1.

Таблиця 6.1 — Результати тестування

Перевірка	Результат
Запуск застосунку	Пройдено
Завантаження зображення в додаток	Пройдено
Малювання області генерації	Пройдено
Очищення області генерації	Пройдено
Генерування зображення	Пройдено
Збереження згенерованого зображення	Пройдено

6.3 Висновки за розділом 6

У результаті проведеного тестування було підтверджено працездатність основних функціональних можливостей розробленого програмного забезпечення. Всі ключові сценарії, зокрема запуск застосунку, завантаження зображення, виділення області генерації, її очищення, виконання генерації, а також збереження обробленого зображення – були успішно виконані.

Застосунок стабільно працює на заданій платформі тестування без критичних збоїв чи помилок. Інтерфейс програми є інтуїтивно зрозумілим, а взаємодія із серверною частиною відбувається коректно. Всі повідомлення оператору відображаються відповідно до очікувань у випадках відсутності вхідних даних або проблем із підключенням.

ВИСНОВКИ

Під час виконання роботи було проведено аналіз предметної області, обґрунтовано актуальність та доцільність розробки програмного забезпечення для відновлення обличчя на зображеннях.

У першому розділі було розглянуто існуючі підходи до завдання відновлення обличчя. Проведено аналіз існуючих аналогічних рішень, визначено їхні сильні сторони, обмеження, а також сформовано перелік функціональних можливостей, які доцільно реалізувати у розроблюваному ПЗ.

У другому розділі було здійснено аналіз і вибір інструментальних засобів розробки. Було обґрунтовано вибір мови програмування Python та використання бібліотек PyTorch, NumPy, Pillow, Matplotlib, а також фреймворку Flask для реалізації серверної частини. Клієнтську частину було реалізовано на C# з використанням WPF, що забезпечило зручний графічний інтерфейс. Проведено аналіз середовищ розробки, в результаті чого було обрано Visual Studio як основне середовище для створення клієнтського застосунку, а PyCharm для створення серверної частини.

У третьому розділі описано архітектуру програмного забезпечення та принципи взаємодії між клієнтською та серверною частинами. Детально розглянуто роботу з моделлю генерації, що відповідає за відновлення зображення на основі маски. Також було реалізовано механізм створення маски для області генерації на клієнті, що дозволяє користувачу самостійно задавати пошкоджену частину зображення. Наведено реалізацію інтерфейсу для взаємодії з сервером, а також розглянуто обробку виключних ситуацій та забезпечення стійкості до некоректних даних.

Розроблене програмне забезпечення реалізує всі функціональні можливості, передбачені на етапі проєктування, зокрема: завантаження зображення, створення маски області генерації, очищення маски, генерацію нового зображення, відображення результату та його збереження.

Тестування програми довело її працездатність і стабільність в умовах, близьких до середньостатистичних для користувача. Основні функції працюють коректно, результат генерації має задовільну якість, а взаємодія між клієнтською та серверною частиною є надійною.

У подальшому програмне забезпечення може бути вдосконалене шляхом навчання моделі генерування з більшою кількістю епох навчання, більшим датасетом облич людей з різним розміром зображення та вдосконаленням системи GAN для генерації зображення більшої роздільної здатності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. VanceAI Image Enhancer [Electronic resource]. – Access mode: <https://vanceai.com/image-enhancer/>.
2. RestorePhotos.io – Free AI Tool to Restore Old Photos [Electronic resource]. – Access mode: <https://www.restorephotos.io/>
3. Remini – AI Photo Enhancer [Electronic resource]. – Access mode: <https://remini.ai/>.
4. Language Rating 2025: огляд популярності мов програмування в Україні [Електрон. ресурс]. – Режим доступу: <https://dou.ua/lenta/articles/language-rating-2025/>.
5. Best Coding Languages for Desktop Applications – порівняльний аналіз мов програмування для створення десктопних застосунків [Electronic resource]. – Access mode: <https://bestcodinglanguage.com/for-desktop-applications/>.
6. Мови програмування для веброзробки та супутні технології – аналітичний матеріал щодо популярних мов і технологій для фронтенду і бекенду [Електрон. ресурс]. – Режим доступу: <https://drukarnia.com.ua/articles/movi-programuvannya-dlya-vebrozrobki-ta-suputni-tekhnologiyi-wzeAr#heading-3-976>.
7. The Top 10 IDEs for Programmers – рейтинг і порівняння найкращих середовищ розробки для [Electronic resource]. – Access mode: <https://medium.com/codex/the-top-10-ides-for-programmers-a-comprehensive-guide-to-choosing-the-best-ide-for-your-needs-c72e97c34591>.
8. What is PyCharm? Features, Advantages & Disadvantages – огляд функцій, переваг та недоліків середовища PyCharm для [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/what-is-pycharm-features-advantages-disadvantages/>.
9. Eclipse IDE Reviews – аналіз можливостей середовища Eclipse IDE, включаючи підтримку розширень та інструментів командної розробки для

[Electronic resource]. – Access mode: <https://www.joinsecret.com/eclipse-ide/reviews#:~:text=Eclipse%20IDE%20offers%20extensive%20tools>.

10. Get started with Visual Studio IDE – офіційна документація Microsoft щодо початку роботи з Visual Studio IDE [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022>.

11. Visual Studio vs Visual Studio Code – порівняння двох середовищ розробки від Microsoft з точки зору функціональності, гнучкості та продуктивності [Electronic resource]. – Access mode: <https://www.bairesdev.com/blog/visual-studio-vs-visual-studio-code/>.

12. Патерни програмування – шаблони проектування та їх використання в розробці ПЗ [Електрон. ресурс]. – Режим доступу: <http://www.znannya.org/?view=patterns>.

13. MVC Framework – вступ до архітектури Model-View-Controller [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/mvc-framework-introduction/>.

14. Генеративні змагальні мережі (GAN) [Електронний ресурс]. – Режим доступу: <https://hackyourmom.com/kibervijna/generatyvni-zmagalni-merezhi-gan/>.

15. Набір даних CelebA-HQ – високоякісні зображення облич для навчання моделей комп'ютерного зору [Електрон. ресурс]. – Режим доступу: <https://www.kaggle.com/datasets/j53t3r/celebahq/data>.

ДОДАТОК А
Технічне завдання

Вступ

Застосунок призначений для відновлення пошкоджених, частково закритих або спотворених облич на фотографіях. Основними сферами застосування програми є криміналістика та медицина.

A.1 Підстави для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмне забезпечення для реконструкції обличчя з неповних або пошкоджених зображень», затверджене наказом Національного університету «Запорізька політехніка» № 209 від 28 квітня 2025 р.

A.2 Призначення розробки

Програма буде використовуватись для відновлення частини обличчя людини.

A.2.1 Функціональне призначення

Програма має надавати можливості завантаження зображення до застосунку з локального сховища, виділення області відновлення обличчя, генерації виділеної частини обличчя та зберігання готового результату на локальному сховищі.

A.2.2 Експлуатаційне призначення

Програма має експлуатуватися користувачами на ПК, які задовольняють мінімальні вимоги до програмних і апаратних засобів для нормального функціонування застосунку.

A.3 Вимоги до програми або програмного продукту

A.3.1 Вимоги до функціональних характеристик

A.3.1.1 Користувач

До класу «Користувач» відносяться фахівці та кінцеві користувачі, що взаємодіють із програмним забезпеченням з метою завантаження вихідних зображень, запуску процесу реконструкції, отримання результатів та/або проведення аналізу. Це можуть бути:

- співробітники правоохоронних органів та криміналісти;
- судмедексперти;
- дослідники та фахівці у галузі комп'ютерного зору та біометрії;
- художники-реконструктори та 3D-дизайнери;
- інженери, оператори та тестувальники.

A.3.1.2 Вимоги до складу виконуваних функцій

Застосунок має забезпечувати виконання наступних функцій:

- завантаження зображення в застосунок;
- малювання області генерації;
- очищення області генерації;
- генерування зображення;
- збереження згенерованого зображення.

A.3.1.3 Вимоги до організації початкових даних

Як вхідні дані програма має приймати зображення частину якого треба згенерувати.

A.3.1.4 Вимоги до організації вихідних даних

Вихідними даними програми має бути зображення з згенерованою частиною.

A.3.1.5 Вимоги до тимчасових характеристик

Вимоги до тимчасових характеристик програми не пред'являються.

A.3.2 Вимоги до надійності

Імовірність безвідмовної роботи системи повинна бути не менше 99%.

A.3.2.1 Вимоги до забезпечення надійного функціонування програми

Надійне (стійке) функціонування програми має бути забезпечене виконанням сукупності організаційно-технічних заходів, перелік яких наведено нижче:

- організацією безперебійного живлення технічних засобів;
- використанням ліцензійного програмного забезпечення;
- регулярним виконанням вимог стандартів захисту інформації;
- виконання перевірок на наявність вірусів.

A.3.2.2 Збійні ситуації

Програмою мають оброблятися наступні збійні ситуації:

- помилки під час взаємодії з серверною частиною;
- помилки під час завантаження та збереження зображення.

A.3.2.3 Час відновлення після відмови

Вимоги до часу відновлення після відмови не пред'являються.

A.3.3 Умови експлуатації

A.3.3.1 Кліматичні умови експлуатації

Кліматичні умови експлуатації, за яких повинні забезпечуватися вказані характеристики, повинні задовольняти вимоги, які пред'являються до технічних засобів у частині умов їх експлуатації.

A.3.3.2 Вимоги до видів обслуговування

Програма не вимагає проведення яких-небудь видів обслуговування.

A.3.3.3 Вимоги до чисельності та кваліфікації персоналу

Вимоги до чисельності та кваліфікації персоналу не пред'являються.

A.3.4 Вимоги до складу та параметрів технічних засобів

Мінімальні вимоги до програмних та апаратних засобів для нормального функціонування застосунку наступні:

- 64-bit Windows 10 / 11;
- двоядерний процесор або вище частотою 1,44 ГГц;
- 50 Мб вільного місця на диску;
- 4 Гб оперативної пам'яті і більше;
- доступ до інтернету.

Мінімальні вимоги до програмних та апаратних засобів для нормального функціонування серверної частини програми наступні:

- Python 3.9, Flask 3.1.0, PyTorch 2.7.0, NumPy 2.2.5, Pillow 11.2.1, Matplotlib 3.10.3;
- 64-bit Windows 10 / 11;
- двоядерний процесор або вище частотою 1,44 ГГц;
- 5 Гб вільного місця на диску;
- 4 Гб оперативної пам'яті і більше;
- доступ до інтернету.

А.3.5 Вимоги до інформаційної та програмної сумісності

Програма повинна бути написана на мові програмування C#. Для її створення має бути використане інтегроване середовище розробки Microsoft Visual Studio. При визначенні архітектури програмного забезпечення має застосовуватися шаблон проектування MVC. Для створення графічного інтерфейсу має бути використано фреймворк .NET.

Застосунок орієнтований на роботу в ОС Windows, тому для коректної роботи програми необхідне стабільне функціонування ОС.

Серверна частина програми повинна бути написана на мові програмування Python з використанням бібліотек Flask, PyTorch, NumPy, Pillow, Matplotlib.

Для її створення має бути використане інтегроване середовище розробки PyCharm. Серверна частина програми орієнтована на роботу в будь-якій операційній системі, яка підтримує Python та бібліотеки Flask, PyTorch, NumPy, Pillow, Matplotlib.

A.3.6 Вимоги до маркування та пакування

Застосунок передається у вигляді EXE файлу. Вимоги до маркування відсутні.

A.3.7 Вимоги до транспортування та збереження

Вимоги до транспортування та збереження відсутні.

A.3.8 Спеціальні вимоги

Програма повинна забезпечувати взаємодію з користувачем за допомогою графічного інтерфейсу, зображеного на рисунку А.1

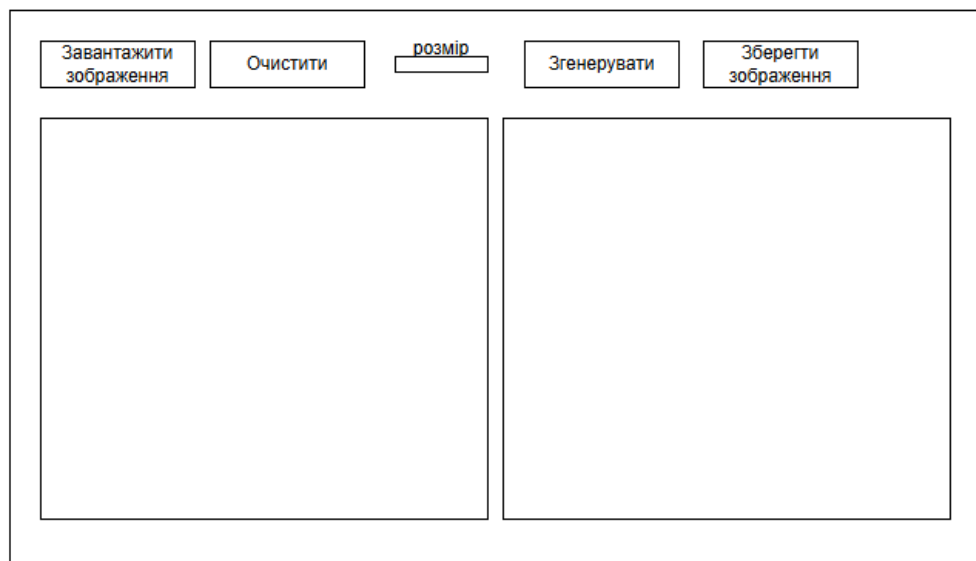


Рисунок А.1 – Головне вікно

A.4 Вимоги до програмної документації

Склад програмної документації повинен включати:

- технічне завдання;
- керівництво програміста;

- керівництво оператора;
- опис програми;
- програма і методика випробувань.

A.5 Техніко-економічні показники

Економічна ефективність від використання програми забезпечується за рахунок скорочення витрат часу на ведення паперового журналу, в якому 109 вручну заносяться назви ліків, дати придбання та споживання, для будь-якого користувача незалежно від його розташування та часу доби. Також ефективність досягається завдяки уникненню придбання зайвих ліків.

A.6 Стадії та етапи розробки

Розробка повинна бути проведена в 3 стадії:

- технічне завдання;
- технічний і робочий проекти;
- впровадження.

На стадії «Технічне завдання» повинен бути виконаний етап розробки, узгодження і затвердження технічного завдання.

На стадії «Технічний і робочий проекти» повинні бути виконані:

- розробка програми;
- розробка програмної документації;
- тестування програми.

На стадії «Впровадження» повинен бути виконаний етап підготовки та передачі програми.

Зміст робіт за етапами. На етапі розробки технічного завдання повинні бути виконані перераховані нижче роботи:

- постановка задачі;
- визначення та уточнення вимог до технічних засобів;

- визначення вимог до програми;
- визначення стадій, етапів і термінів розробки програми і документації до неї;
- узгодження і затвердження технічного завдання.

На етапі розробки програми повинна бути виконана робота з програмування (кодування) і налагодження програми.

На етапі розробки програмної документації розробляються відповідні документи.

На етапі тестування програми повинні бути виконані перераховані нижче види робіт:

- розробка, узгодження та затвердження порядку і методики випробувань;
- проведення приймально-здавальних випробувань;
- документації за результатами випробувань.

На етапі підготовки та передачі програми повинна бути виконана робота з підготовки та передачі програми і програмної документації в експлуатацію.

Виконавець повинен розробити застосунок, надати вихідні коди та документацію до розробленої програми не пізніше 26.06.2025.

A.7 Порядок контролю та приймання

Приймально-здавальні випробування програми повинні проводитися відповідно до розробленого виконавцем і узгодженого із замовником документа «Програма і методика випробувань».

Хід проведення приймально-здавальних випробувань замовник і виконавець документують в протоколі випробувань.

На підставі протоколу випробувань виконавець спільно із замовником підписують акт приймання-здачі програми в експлуатацію.

ДОДАТОК Б
Текст програми

Б.1 Текст файлу StartServer.py

```

from flask import Flask, request, send_file
import io
from PIL import Image, ImageOps
from GanGenerate import test_single_image

app = Flask(__name__)

@app.route('/upload', methods=['POST'])
def upload():
    file = request.files['file']
    image = Image.open(file.stream).convert("RGB")

    print(f"Отримане зображення має розмір: {image.size}")
    processed_image = test_single_image(image)

    img_io = io.BytesIO()
    processed_image.save(img_io, 'PNG')
    img_io.seek(0)

    return send_file(img_io, mimetype='image/png')

if __name__ == '__main__':
    app.run(port=5000)

```

Б.2 Текст файлу GAN.py

```

import cv2
import os
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from PIL import Image
import numpy as np

def random_smooth_mask(size, max_strokes=5, max_length=60, max_brush_width=20):

    h, w = size
    mask = np.zeros((h, w), dtype=np.uint8)

    for _ in range(np.random.randint(1, max_strokes + 1)):
        num_points = np.random.randint(5, 10)
        brush_width = np.random.randint(10, max_brush_width)

        # Центр малювання – не по всьому зображенню
        margin = min(h, w) // 4
        points = np.stack([
            np.random.randint(margin, w - margin, size=num_points),
            np.random.randint(margin, h - margin, size=num_points)
        ], axis=1).astype(np.int32)

        curve = cv2.approxPolyDP(points, epsilon=1.0, closed=False)
        curve = np.array([curve], dtype=np.int32)

        cv2.polylines(mask, curve, isClosed=False, color=255, thickness=brush_width,
            lineType=cv2.LINE_8)

    mask = (mask > 127).astype(np.uint8)
    mask_tensor = torch.from_numpy(mask).float().unsqueeze(0) # (1, H, W)

```

```

    return mask_tensor

class SingleFolderMaskedDataset(Dataset):
    def __init__(self, root, transform=None, mask_size=32):
        self.root = root
        self.transform = transform
        self.mask_size = mask_size
        self.image_files = [os.path.join(root, f) for f in os.listdir(root) if
f.lower().endswith(('.png', '.jpg', '.jpeg'))]

    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, index):
        image_path = self.image_files[index]
        image = Image.open(image_path).convert("RGB")
        if self.transform:
            image = self.transform(image)

        mask = random_smooth_mask((image.shape[1], image.shape[2]))

        masked_image = image.clone()
        masked_image = masked_image * (1 - mask) + mask

        return masked_image, mask, image

class Generator(nn.Module):
    def __init__(self, in_channels=4, out_channels=3):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(in_channels, 64, 4, 2, 1), # 64 → 32
            nn.ReLU(),
            nn.Conv2d(64, 128, 4, 2, 1), # 32 → 16
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.Conv2d(128, 256, 4, 2, 1), # 16 → 8
            nn.BatchNorm2d(256),
            nn.ReLU()
        )

        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(256, 128, 4, 2, 1), # 8 → 16
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.ConvTranspose2d(128, 64, 4, 2, 1), # 16 → 32
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.ConvTranspose2d(64, out_channels, 4, 2, 1), # 32 → 64
            nn.Tanh()
        )

    def forward(self, x):
        return self.decoder(self.encoder(x))

class Discriminator(nn.Module):
    def __init__(self, nc=3):
        super().__init__()
        self.model = nn.Sequential(
            nn.Conv2d(nc, 64, 4, 2, 1), # 64 → 32
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 128, 4, 2, 1), # 32 → 16
            nn.BatchNorm2d(128),

```

```

        nn.LeakyReLU(0.2, inplace=True),
        nn.Conv2d(128, 256, 4, 2, 1), # 16 → 8
        nn.BatchNorm2d(256),
        nn.LeakyReLU(0.2, inplace=True),
        nn.Conv2d(256, 1, 8), # 8 → 1
        nn.Sigmoid()
    )

    def forward(self, x):
        return self.model(x).view(-1)

```

Б.3 Текст файлу GANTrain.py

```

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.utils as vutils
import os
from torch.utils.data import DataLoader
from GAN import Generator, Discriminator, SingleFolderMaskedDataset

def save_checkpoint(epoch, netG, netD, optimizerG, optimizerD,
                    checkpoint_dir='checkpoints'):
    os.makedirs(checkpoint_dir, exist_ok=True)
    checkpoint_path = os.path.join(checkpoint_dir, f"checkpoint_epoch_{epoch}.pth")

    torch.save({
        'epoch': epoch,
        'netG_state_dict': netG.state_dict(),
        'netD_state_dict': netD.state_dict(),
        'optimizerG_state_dict': optimizerG.state_dict(),
        'optimizerD_state_dict': optimizerD.state_dict(),
    }, checkpoint_path)

    print(f"Чекпоінт збережено: {checkpoint_path}")

def load_checkpoint(checkpoint_path, netG, netD, optimizerG, optimizerD):
    checkpoint = torch.load(checkpoint_path)
    epoch = checkpoint['epoch']

    netG.load_state_dict(checkpoint['netG_state_dict'])
    netD.load_state_dict(checkpoint['netD_state_dict'])
    optimizerG.load_state_dict(checkpoint['optimizerG_state_dict'])
    optimizerD.load_state_dict(checkpoint['optimizerD_state_dict'])

    print(f"Чекпоінт завантажено з: {checkpoint_path}, епоха: {epoch}")
    return epoch

def GANtrain():
    dataroot = r'D:/Dev/Source/Python/testdpom/deepfillv2-
pytorch/data/celeb/CelebA-HQ-img'
    transform = transforms.Compose([
        transforms.Resize(64),
        transforms.CenterCrop(64),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    dataset = SingleFolderMaskedDataset(dataroot, transform, mask_size=16)

```

```

dataloader = DataLoader(dataset, batch_size=64, shuffle=True, num_workers=2)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
netG = Generator().to(device)
netD = Discriminator().to(device)

optimizerG = optim.Adam(netG.parameters(), lr=0.0002, betas=(0.5, 0.999))
optimizerD = optim.Adam(netD.parameters(), lr=0.0002, betas=(0.5, 0.999))

checkpoint_path = 'checkpoints/checkpoint_epoch_last.pth'
if os.path.exists(checkpoint_path):
    epoch = load_checkpoint(checkpoint_path, netG, netD, optimizerG, optimizerD)
else:
    epoch = 0

num_epochs = 100

fixed_samples = next(iter(dataloader))
fixed_masked_imgs, fixed_masks, fixed_real_imgs = fixed_samples
fixed_masked_imgs = fixed_masked_imgs[:64].to(device)
fixed_masks = fixed_masks[:64].to(device)

for epoch in range(epoch, num_epochs):
    for i, (masked_imgs, masks, real_imgs) in enumerate(dataloader):
        masked_imgs, masks, real_imgs = masked_imgs.to(device),
        masks.to(device), real_imgs.to(device)

        netD.zero_grad()
        output_real = netD(real_imgs)
        label_real = torch.ones_like(output_real, device=device)
        loss_real = nn.BCELoss()(output_real, label_real)

        gen_input = torch.cat((masked_imgs, masks), dim=1)
        fake_imgs = netG(gen_input)
        output_fake = netD(fake_imgs.detach())
        label_fake = torch.zeros_like(output_fake, device=device)
        loss_fake = nn.BCELoss()(output_fake, label_fake)

        loss_D = loss_real + loss_fake
        loss_D.backward()
        optimizerD.step()

        netG.zero_grad()
        output = netD(fake_imgs)
        loss_G_adv = nn.BCELoss()(output, label_real)
        loss_G_rec = nn.L1Loss()(fake_imgs * masks, real_imgs * masks)
        loss_G = loss_G_adv + loss_G_rec
        loss_G.backward()
        optimizerG.step()

        if i % 50 == 0:
            print(f"[{epoch}/{num_epochs}][{i}/{len(dataloader)}] Loss_D:
{loss_D.item():.4f}, Loss_G: {loss_G.item():.4f}")

    if epoch % 10 == 0:
        save_checkpoint(epoch, netG, netD, optimizerG, optimizerD)

    with torch.no_grad():
        gen_input = torch.cat((fixed_masked_imgs, fixed_masks), dim=1)
        fake_imgs = netG(gen_input).detach().cpu()
        fake_imgs = (fake_imgs + 1) / 2
        masked_imgs = (fixed_masked_imgs.cpu() + 1) / 2
        masks_rgb = fixed_masks.cpu().repeat(1, 3, 1, 1)
        full_grid = torch.cat((masked_imgs, masks_rgb, fake_imgs), dim=0)
        vutils.save_image(full_grid,

```

```

f"outputnogaus/fulloutput_epoch_{epoch}.png", nrow=8)

    vutils.save_image(fake_imgs, f"outputnogaus/output_epoch_{epoch}.png",
nrow=8, normalize=True)
    inpainted_only = fake_imgs * fixed_masks
    inpainted_imgs = fake_imgs * fixed_masks + masked_imgs * (1 - fixed_masks)

    vutils.save_image(inpainted_imgs,
f"outputnogaus/inpainted_imgs_epoch_{epoch}.png", nrow=8)
    vutils.save_image(inpainted_only,
f"outputnogaus/inpainted_only_epoch_{epoch}.png", nrow=8)

    torch.save(netG.state_dict(), "GANgeneratornewMaskV3.1.pth")
    print("Навчання завершено!")

if __name__ == '__main__':
    GANtrain()

```

Б.4 Текст файлу GanGenerate.py

```

import torch
import numpy as np
import torchvision.utils as vutils
from PIL import Image
from torchvision import transforms
from GAN import Generator # твій генератор

def create_binary_mask(image, threshold=200):

    image_np = np.array(image)

    mask = np.all(image_np >= threshold, axis=-1)
    mask = mask.astype(np.uint8)

    mask_image = Image.fromarray(mask * 255)
    return mask_image

def test_single_image(image, generator_path="GANgeneratornewMaskV3.1.pth",
output_path="test_single_output.png"):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    netG = Generator().to(device)
    netG.load_state_dict(torch.load(generator_path, map_location=device))
    netG.eval()

    transform = transforms.Compose([
        transforms.Resize(64),
        transforms.CenterCrop(64),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    image_tensor = transform(image).unsqueeze(0).to(device)

    mask_image = create_binary_mask(image)
    mask_image = mask_image.resize((64, 64))
    mask = np.array(mask_image)
    mask = (mask > 127).astype(np.float32)
    mask = torch.tensor(mask).unsqueeze(0).unsqueeze(0).to(device)

```

```

mask_rgb = mask.repeat(1, 3, 1, 1)
masked_image = image_tensor * (1 - mask_rgb) + mask_rgb

gen_input = torch.cat((masked_image, mask), dim=1) # -> (1, 4, H, W)

with torch.no_grad():
    fake_image = netG(gen_input)

fake_image = (fake_image + 1) / 2
masked_image = (masked_image + 1) / 2
inpainted_result = fake_image * mask + masked_image * (1 - mask)

result_image = vutils.make_grid(inpainted_result, nrow=1).cpu().permute(1, 2,
0).numpy()
result_image = (result_image * 255).astype(np.uint8) # Масштабуємо до діапазону
[0, 255]
result_image = Image.fromarray(result_image)

return result_image

```

Б.5 Текст файлу ServerClient.cs

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Media.Imaging;

namespace AppFillFace
{
    internal class ServerClient
    {
        private static readonly string serverUrl = "http://localhost:5000/upload";

        public static async Task<byte[]?> SendImageAsync(byte[] imageData)
        {
            try
            {
                using (HttpClient client = new HttpClient())
                using (var content = new MultipartFormDataContent())
                {
                    var byteContent = new ByteArrayContent(imageData);
                    byteContent.Headers.ContentType = new
System.Net.Http.Headers.MediaTypeHeaderValue("image/png");
                    content.Add(byteContent, "file", "drawn_image.png");

                    var response = await client.PostAsync(serverUrl, content);

                    if (response.IsSuccessStatusCode)
                    {
                        return await response.Content.ReadAsByteArrayAsync();
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine("Помилка під час надсилання: " + ex.Message);
            }
        }
    }
}

```

```

        return null;
    }

    public static BitmapImage ConvertToBitmapImage(byte[] imageData)
    {
        using (MemoryStream ms = new MemoryStream(imageData))
        {
            BitmapImage bitmap = new BitmapImage();
            bitmap.BeginInit();
            bitmap.StreamSource = ms;
            bitmap.CacheOption = BitmapCacheOption.OnLoad;
            bitmap.EndInit();
            return bitmap;
        }
    }
}

```

Б.6 Текст файла ImageWork.cs

```

using Microsoft.Win32;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Media.Imaging;
using System.Windows;

namespace AppFillFace
{
    internal class ImageWork
    {
        public static BitmapImage LoadImageFromFile()
        {
            OpenFileDialog dlg = new OpenFileDialog();
            dlg.Filter = "Image Files|*.jpg;*.png;*.bmp";

            if (dlg.ShowDialog() == true)
            {
                return new BitmapImage(new System.Uri(dlg.FileName));
            }

            return null;
        }

        public static void SaveImageToFile(BitmapSource source)
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog
            {
                Filter = "PNG Image|*.png"
            };

            if (saveFileDialog.ShowDialog() == true)
            {
                PngBitmapEncoder encoder = new PngBitmapEncoder();
                encoder.Frames.Add(BitmapFrame.Create(source));

                using (var fileStream = new FileStream(saveFileDialog.FileName,
                    FileMode.Create))

```

```

        {
            encoder.Save(fileStream);
        }

        MessageBox.Show("Зображення збережено!");
    }
}
}
}

```

Б.7 Текст файлу ImageProcessor.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Controls;
using System.Windows.Media.Imaging;
using System.Windows.Media;
using System.Windows;

namespace AppFillFace
{
    internal class ImageProcessor
    {
        public static RenderTargetBitmap RenderCombinedImage(Image baseImage, Canvas
overlayCanvas)
        {
            if (baseImage.Source is BitmapSource baseBitmap)
            {
                int width = baseBitmap.PixelWidth;
                int height = baseBitmap.PixelHeight;

                RenderTargetBitmap rtb = new RenderTargetBitmap(
                    width, height, 96, 96, PixelFormats.Pbgra32);

                DrawingVisual dv = new DrawingVisual();
                using (DrawingContext ctx = dv.RenderOpen())
                {
                    VisualBrush vb = new VisualBrush(baseImage)
                    {
                        Stretch = Stretch.Fill
                    };
                    ctx.DrawRectangle(vb, null, new Rect(0, 0, width, height));

                    VisualBrush overlay = new VisualBrush(overlayCanvas)
                    {
                        Stretch = Stretch.Fill
                    };
                    ctx.DrawRectangle(overlay, null, new Rect(0, 0, width, height));
                }

                rtb.Render(dv);
                return rtb;
            }

            return null;
        }
    }
}

```

ДОДАТОК В
Слайди презентації

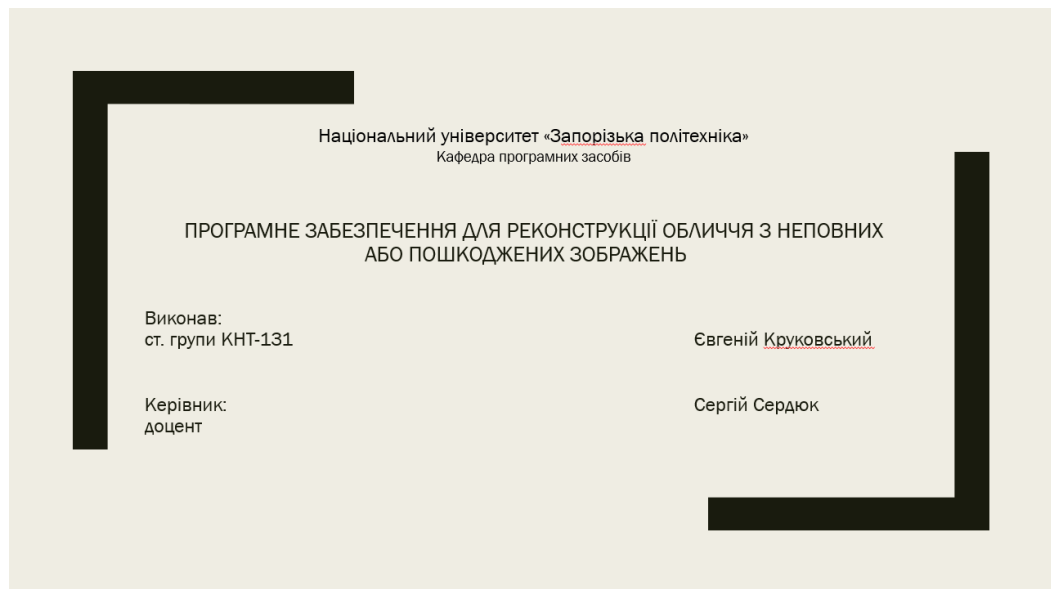


Рисунок В.1 — Слайд № 1

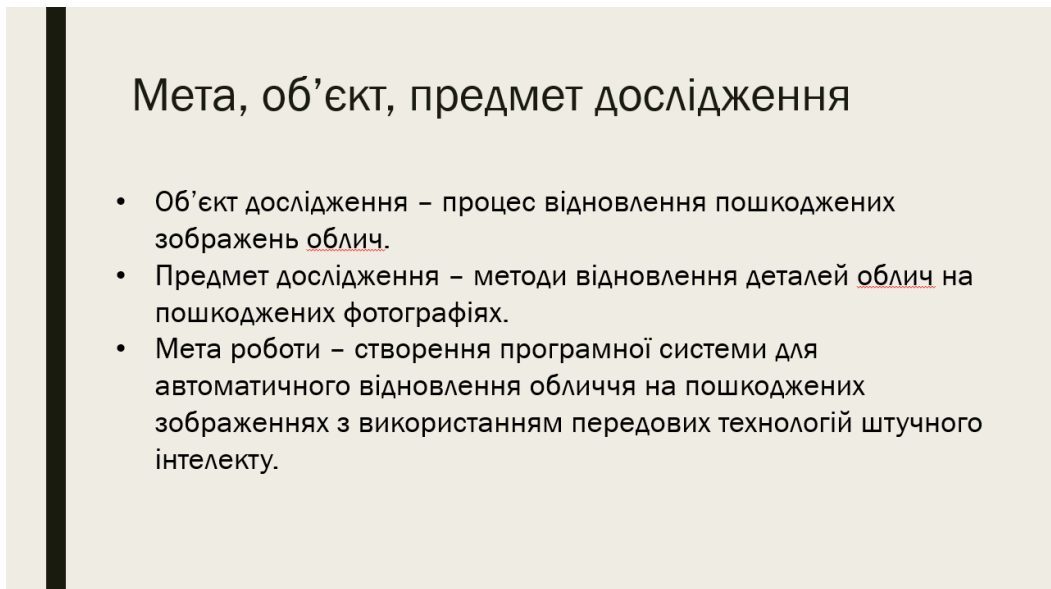


Рисунок В.2 — Слайд № 2

Задачі, що вирішуються

- Проаналізувати сучасні підходи до відновлення зображень, зокрема методи реконструкції облич, що базуються на нейронних мережах та алгоритмах глибокого навчання.
- Дослідити архітектури нейронних мереж, що застосовуються для завдань відновлення облич.
- Реалізувати програмну систему, яка дозволяє здійснювати відновлення зображень облич.

Рисунок В.3 — Слайд № 3

Аналіз наявних аналогів

Можливості	VanceAI Image Enhancer	RestorePhotos.io	Remini	Власний застосунок
Потужні інструменти ретуші	-	-	-	+
Автоматичне відновлення зображень	+	+	+	+
Підтримка штучного інтелекту	+	+	+	+
Потрібне встановлення	-	-	-	+
Онлайн-доступ	+	+	+	-
Простота використання	+	+	+	+
Можливість обробки портретів	+	+	+	+
Професійний рівень редагування	-	-	-	+
Безкоштовний доступ	± (частково)	+	-	+

Рисунок В.4 — Слайд № 4

Вибір мови програмування					Для серверної частини				
Для комп'ютерного застосунку					Для серверної частини				
Критерій	C++	Java	Python	C#	Критерій	JavaScript	C#	Ruby	Python
Продуктивність	+	+-	-	+	Продуктивність	+	+-	-	+
Підтримка GUI	+	+	+	+	Простота синтаксису	-	+	+	+
Простота синтаксису	-	+	+	+	Продуктивність сервера	+	+-	-	+
Бібліотеки сторонніх розробників	+	+	+	+	Популярність у backend-розробці	+	+	-	+
Інтеграція з Windows API	+	-	-	+	Підтримка обробки зображень	-	-	-	+
Швидкість розробки	-	-	+	+	Підтримка машинного навчання	-	-	-	+
Машинне навчання / AI	-	+	+	+	Екосистема для API	+	+	+	+
Підтримка багатопоточності	+	+	+	+	Підтримка багатопоточності	+	+	+	+

Рисунок В.5 — Слайд № 5

Критерій	PyCharm	Eclipse	Visual Studio
Основне призначення	Python	Java	C#
Підтримка Python	+	-	+
Підтримка C#	+	-	+
Підтримка інших мов	+	+	+
Автодоповнення	+	+	+
Refactoring	+	-	+
Інспекція коду	+	-	+
Налагодження (debugging)	+	+	+
Великі вимоги до ресурсів	+	+	+
Простий користувацький інтерфейс	+	-	+
Кросплатформеність	+	+	+

Рисунок В.6 — Слайд № 6

Схема структури програми

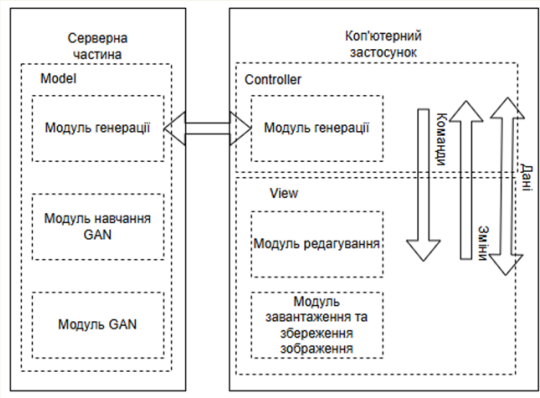


Рисунок В.7 — Слайд № 7

Модулі програми

Модуль	Структура	Функції
Модуль редагування	ImageProcessor.cs	Збереження відножного зображення з намальованою областю.
Модуль завантаження та збереження	ImageWork.cs	Завантаження зображення з локального сховища; Збереження згенерованого зображення до локального сховища.
Модуль генерування	ServerClient.cs	Завантаження зображення на сервер для обробки. Обробка відповідей від сервера
Модуль генерування	GANGenerate.py	Створення бінарної маски на основі відножного зображення; Генерація втраченої частини зображення на основі відножного зображення та бінарної маски; Отримання із застосунку запиту з віддним зображенням.
Модуль навчання GAN	GANTrain.py	Тренування моделі GAN з використанням даних та створенням випадкових бінарних масок; Створення чекпоінтів для подальшої можливості навчання.
Модуль GAN	GAN.py	Функція створення випадкових бінарних масок Реалізація нейронної мережі GAN.

Рисунок В.8 — Слайд № 8

Нейрона мережа GAN

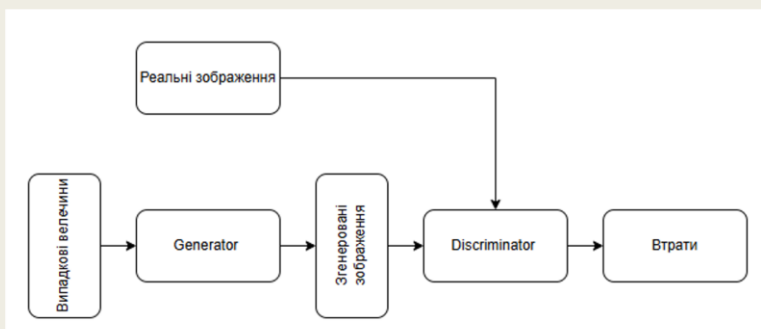


Рисунок В.9 — Слайд № 9

Результати навчання мережи GAN

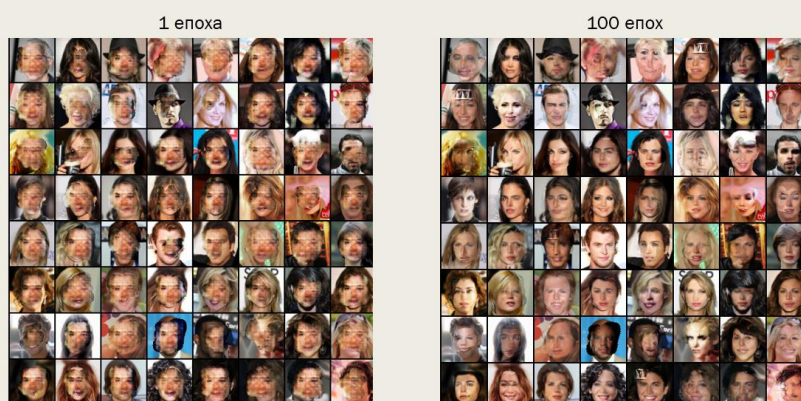


Рисунок В.10 — Слайд № 10

Інтерфейс

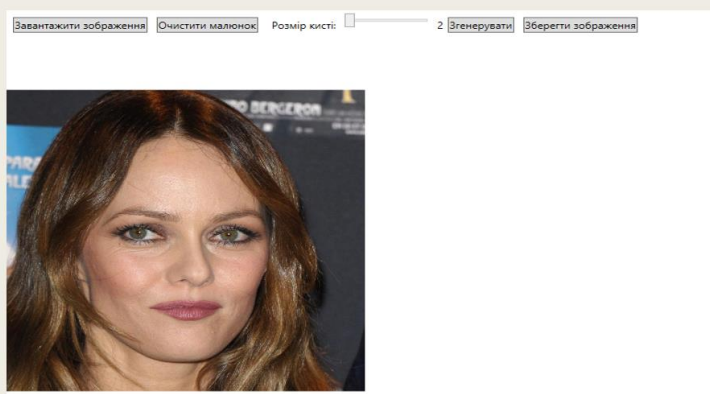


Рисунок В.11 — Слайд № 11

Висновки

- В створеному програмному забезпеченні було реалізовано всі заплановані функції, зокрема: завантаження зображення, створення маски області генерації, очищення маски, генерацію нового зображення, відображення результату та його збереження.
- Тестування програми довело її працездатність і стабільність в умовах, близьких до середньостатистичних для користувача. Основні функції працюють коректно, результат генерації має задовільну якість, а взаємодія між клієнтською та серверною частиною є надійною.
- У подальшому програмне забезпечення може бути вдосконалене шляхом навчання моделі генерування з більшою кількістю епох навчання, більшим дата сетом облич людей з різним розміром зображення та вдосконаленням системи GAN для генерації зображення більшої роздільної здатності.

Рисунок В.12— Слайд № 12