

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 проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
МЕТОДІВ МАШИННОГО НАВЧАННЯ ДЛЯ РОЗПІЗНАВАННЯ
ОПОРНИХ ТОЧОК ЛЮДИНИ
RESEARCH AND SOFTWARE IMPLEMENTATION OF
MACHINE LEARNING METHODS FOR RECOGNITION
OF CONTROL POINTS IN HUMAN BODY

Виконав(ла): студент(ка) 2 курсу, групи КНТ-124м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

БАЛУХТІН М.С.

(ПРИЗВИЩЕ та ініціали)

Керівник ОЛІЙНИК А.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент СРКУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
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)

БАЛУХТІНА Максима Сергійовича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів машинного навчання для розпізнавання опорних точок людини.
Research and Software Implementation of Machine Learning Methods for Recognition of Control Points in Human Body

керівник проєкту (роботи) д.т.н., професор, ОЛІЙНИК Андрій Олександрович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 30 ” вересня 2025 року № 447

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

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Опис предметної області. 2. Вибір та обґрунтування структури системи, яка проєктується. 3. Основні рішення щодо реалізації компонентів системи. 4. Керівництво програміста. 5. Керівництво оператора.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ОЛІЙНИК А.О., професор		
Нормоконтролер	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “ 30 ” вересня 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.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Максим БАЛУХТІН
(Імя ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Андрій ОЛІЙНИК
(Імя ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
97 с., 6 табл., 19 рис., 3 дод., 16 джерел.

МОБІЛЬНЕ ЗАСТОСУВАННЯ, IOS, SWIFT, ГРАФІЧНИЙ
РЕДАКТОР, ОПОРНІ ТОЧКИ, МАШИННЕ НАВЧАННЯ.

Об'єкт дослідження – процес проектування та розробки програмних засобів, що використовують методи машинного навчання.

Предмет дослідження – програмні засоби для визначення опорних точок людини.

Мета роботи – дослідження та реалізація програмного забезпечення для розпізнавання опорних точок людини для зменшення трудомісткості та часу обробки зображень людини.

Матеріали, методи та технічні засоби: CoreML, Vision, Swift 6.0, XCode 16.4, macOS 15.7 Sequoia, MacBook Pro 16” (2024).

Результати. Проведено аналіз сучасних моделей машинного навчання, що застосовуються для визначення ключових точок тіла. Розроблено мобільний графічний редактор, у якому реалізовано автоматичне визначення опорних точок та інструменти їх подальшої обробки.

Наукова новизна полягає у розробленні методу локальної обробки зображень, що поєднує оптимізовану ML-модель визначення опорних точок людини з подальшою графічною трансформацією на мобільних пристроях.

Висновки. Створено мобільний застосунок, який демонструє ефективність використання ML-моделей у задачах обробки зображень та може застосовуватись у різних практичних сценаріях.

Галузь використання – мобільні графічні редактори, системи машинного аналізу зображень, мультимедійні застосунки.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 97 pages, 6 tables, 19 figurines, 3 appendixes, 16 sources.

MOBILE APPLICATION, IOS, SWIFT, GRAPHICS EDITOR, KEYPOINTS, MACHINE LEARNING.

Object of study is the process of designing and developing software systems that incorporate machine learning methods.

Subject of research is a software solution for detecting human body keypoints.

The purpose of the work is to investigate and implement machine learning methods for human body keypoints recognition within a functional software system aimed at reducing the labor intensity and processing time of human image processing.

Materials, methods and tools: CoreML, Vision, Swift 6.0, XCode 16.4, macOS 15.7 Sequoia, MacBook Pro 16” (2024).

Results. An analysis of modern machine learning models for human keypoint detection has been conducted. A mobile graphics editor has been developed with built-in automatic keypoint detection and tools for further processing of the detected points.

The scientific novelty lies in the development of a method for local image processing that combines an optimized ML model for human keypoint detection with subsequent graphical transformation on mobile devices.

Conclusions. A mobile application has been created demonstrating the efficiency of ML models in image processing tasks and enabling their application in various practical scenarios.

Field of use – mobile graphics editors, machine-based image analysis systems, multimedia applications.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Опис предметної області	11
1.1 Аналіз предметної області	11
1.2 Постанова завдання.....	13
1.3 Висновки за розділом 1	13
2 Вибір та обґрунтування структури системи, яка проектується.....	14
2.1 Вибір інструментарію розробки програмного забезпечення	14
2.1.1 Огляд особливостей мови програмування	14
2.1.2 Огляд особливостей середовища розробки.....	16
2.2 Структурна схема системи.....	18
2.3 Опис модулів розроблюваної системи.....	19
2.4 Метод локальної обробки зображень з використанням оптимізованої ML-моделі.....	19
2.4.1 Етап первинного аналізу зображення	20
2.4.2 Етап побудови структурної моделі опорних точок	20
2.4.3 Етап графічної трансформації та інтеграції в редакторі.....	21
2.4.4 Переваги та недоліки запропонованого методу.....	21
2.5 Висновки за розділом 2	22
3 Основні рішення щодо реалізації компонентів системи.....	24
3.1 Алгоритм функціонування розроблюваної системи	24
3.2 Основні рішення щодо реалізації системи	25
3.3 Висновки за розділом 3	27
4 Керівництво програміста.....	28
4.1 Призначення та умови застосування програми	28
4.2 Характеристики програми.....	28
4.3 Вхідні та вихідні дані.....	29
4.4 Звертання до програми	29

	7
4.5 Повідомлення	29
4.6 Висновки за розділом 4	30
5 Керівництво оператора	31
5.1 Призначення та умови виконання програми.....	31
5.2 Виконання програми.....	32
5.3 Висновки за розділом 5	49
Висновки	50
Перелік джерел посилань	52
Додаток А Технічне завдання	54
Додаток Б Текст програми	60
Додаток В Слайди презентації.....	91

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
AR	– Augmented Reality;
CPU	– Central Processing Unit;
GPU	– Graphics Processing Unit;
LLVM	– Low Level Virtual Machine;
ML	– Machine Learning;
UI	– User Interface;
ОС	– операційна система;
ПЗ	– програмне забезпечення;
ПК	– персональний комп'ютер.

ВСТУП

У сучасному світі мобільні пристрої перетворилися на невід'ємний інструмент повсякденного життя. Вони забезпечують доступ до зв'язку, інтернету, банківських послуг, мультимедійних ресурсів, а також виступають платформою для виконання складних професійних завдань. Значна увага сьогодні приділяється мобільним застосункам, що дозволяють не лише взаємодіяти з інформацією, але й ефективно обробляти зображення та відео.

Широке поширення технологій комп'ютерного зору та машинного навчання сприяло появі нових можливостей для автоматизованої обробки візуальних даних. Одним із перспективних напрямів є розпізнавання опорних точок людини (суглоби, кінцівки тощо), що відкриває можливість для створення застосунків у сфері спорту, медицини, анімації та графічного редагування.

Актуальність дослідження зумовлена потребою у мобільних інструментах, які дозволяють швидко та зручно визначати ключові точки людського тіла на зображеннях і надалі використовувати їх для маніпуляцій чи візуалізації. Саме тому важливим завданням є створення мобільного графічного редактора з використанням методів машинного навчання.

Метою роботи є дослідження та програмна реалізація алгоритмів машинного навчання для автоматичного розпізнавання опорних точок людини у графічному редакторі під iOS.

Відповідно до зазначеної мети вирішуються такі завдання:

- уточнити зміст ключових понять у сфері комп'ютерного зору та машинного навчання;
- проаналізувати сучасні методи розпізнавання опорних точок людини;
- визначити архітектурні особливості мобільних графічних редакторів;
- розробити прототип застосунку під iOS.

Об'єкт дослідження – процес використання методів машинного навчання у мобільних застосунках.

Предмет дослідження – мобільний графічний редактор з функцією визначення опорних точок людини.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, мова програмування Swift 6.0, інтегроване середовище розробки програм XCode 16.4, персональний arm-сумісний комп'ютер MacBook Pro 16" (2024), що функціонує під управлінням операційної системи MacOS 15.6 (Sequoia).

Теоретико-методологічну основу дослідження включає методи аналізу та синтезу, систематизації, моделювання та прототипування, що застосовуються при розробці та дослідженні мобільних застосунків.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Сучасний розвиток інформаційних технологій значною мірою зумовлений потребою в автоматизації процесів обробки даних, у тому числі візуальної інформації. Людина щодня взаємодіє з величезними обсягами зображень та відео, що потребує швидких і точних інструментів для їх аналізу. Саме тому комп'ютерний зір та машинне навчання стали одним із найперспективніших напрямів досліджень у галузі штучного інтелекту [1].

Особливе місце серед завдань комп'ютерного зору посідає визначення опорних точок людини. Під цим терміном розуміють координати ключових ділянок тіла (наприклад, очі, плечі, лікті, коліна, стопи), які дають змогу побудувати модель скелета людини [2]. Виявлення таких точок дозволяє проводити аналіз пози, розпізнавати рухи та взаємодії, а також здійснювати їх подальшу інтерпретацію в прикладних задачах [3].

Сфери застосування технологій розпізнавання опорних точок людини охоплюють [4]:

- спорт і фітнес;
- медицина та реабілітація;
- системи безпеки;
- розважальні технології;
- графічна обробка.

Важливо зазначити, що мобільні пристрої поступово стали основною платформою для реалізації таких рішень. Смартфони та планшети оснащені потужними процесорами, камерами та спеціалізованими модулями для прискорення обчислень (GPU, Neural Engine), що дає змогу виконувати складні ML-алгоритми безпосередньо на пристрої [5].

Мобільний застосунок у цьому контексті визначається як програмний продукт, що поєднує інтерфейс для взаємодії з користувачем та реалізує алгоритми аналізу зображень у реальному часі [6]. На відміну від класичних

настільних програм, мобільні застосунки мають бути оптимізовані для обмежених ресурсів, забезпечувати високу швидкість та простоту використання [7].

У сучасній практиці виділяють кілька підходів до створення мобільних застосунків [8]:

- нативні застосунки – створюються мовами програмування, які підтримує конкретна платформа;
- вебзастосунки – реалізуються як сторінки чи вебсервіси, що працюють через браузер, але мають обмежені можливості взаємодії з апаратними ресурсами;
- гібридні застосунки – комбінують елементи вебтехнологій із нативними оболонками.

Нативна розробка вважається найбільш доцільною у випадках, коли потрібна висока швидкість та робота з великими масивами даних. Це особливо актуально для завдань комп'ютерного зору, де від швидкості обчислень залежить зручність і точність використання.

Операційна система iOS забезпечує розробникам доступ до вбудованих інструментів для роботи з ML-моделями [9]:

- CoreML – фреймворк для інтеграції навчальних моделей у застосунки;
- Vision – бібліотека для обробки зображень, включно з розпізнаванням об'єктів та ключових точок на тілі;
- ARKit – інструмент для створення доповненої реальності з використанням виявлених точок тіла.

Таким чином, поєднання потужностей сучасних мобільних пристроїв із можливостями машинного навчання створює умови для реалізації графічного редактора з функцією автоматичного визначення опорних точок людини. Це дозволяє не лише автоматизувати обробку зображень, але й відкрити перспективи для інтерактивних застосувань у різних сферах діяльності.

1.2 Постановка завдання

Метою створення системи є розробка мобільного графічного редактора з функцією автоматичного розпізнавання опорних точок людини на зображеннях.

Система повинна забезпечувати:

- завантаження та відображення фотографій;
- автоматичне визначення ключових точок людського тіла за допомогою ML-моделей;
- можливість візуалізації та редагування виявлених точок;
- застосування графічних інструментів для подальших маніпуляцій;
- збереження результатів у стандартних форматах.

Передача та обробка даних відбувається з використанням вбудованих бібліотек iOS для комп'ютерного зору (Vision, CoreML), які забезпечують роботу моделей локально на пристрої.

1.3 Висновки за розділом 1

У розділі проаналізовано предметну область, визначено основні підходи до створення мобільних застосунків та обґрунтовано вибір нативної розробки під iOS.

Встановлено, що актуальною задачею є створення мобільного графічного редактора з інтегрованими алгоритмами машинного навчання для автоматичного визначення опорних точок людини.

Основними функціями системи визначено:

- завантаження та перегляд зображень;
- автоматичне розпізнавання ключових точок;
- редагування та візуалізація результатів;
- збереження змінених зображень.

2 ВИБІР ТА ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ, ЯКА ПРОЄКТУЄТЬСЯ

2.1 Вибір інструментарію розробки програмного забезпечення

2.1.1 Огляд особливостей мови програмування

Для створення мобільного графічного редактора під iOS з використанням методів машинного навчання було обрано мову Swift 6.0.

Результати аналізу та порівняння можливих мов і технологій наведено у табл. 2.1.

Таблиця 2.1 – Порівняння можливих мов програмування

Характеристика	Objective-C	Swift	SwiftUI
Дизайн	Надбудова над C	Сучасна мова Apple	Декларативний UI-фреймворк від Apple
Парадигма	Відправлення та обробка повідомлень	Пагатопарадигмова	Декларативна
Кількість файлів для одного класу	Два (.h + .m)	Один	Один
Сумісність з ML	Обмежена	Повна	Повна
Читабельність коду	Низька	Висока	Висока
Особливості	Застаріла модель пам'яті, складність підтримки	Підтримка сучасних API	Швидке створення UI

Objective-C – мова програмування, яка історично була основною для iOS, однак у сучасних проєктах використовується переважно для підтримки старого коду [10]. Її недоліками є громіздкий синтаксис та відсутність прямої інтеграції з сучасними ML-інструментами.

Swift – сучасна компільована мова програмування, розроблена Apple для створення застосунків під iOS, macOS та інші платформи [11]. Вона має простий синтаксис, підтримує різні парадигми програмування та забезпечує повну інтеграцію з ML-бібліотеками, такими як CoreML та Vision.

SwiftUI – декларативний UI-фреймворк, представлений Apple у 2019 році [12]. Він значно спрощує процес створення інтерфейсів і добре поєднується з ML-моделями. Однак, через відносну новизну, не всі можливості UIKit ще реалізовані в SwiftUI, тому на практиці в багатьох проєктах (у тому числі даному) використовується саме UIKit. SwiftUI у цьому випадку розглядається лише як перспективний напрям, який може замінити UIKit у майбутньому.

У розробці застосунків під iOS основними мовами програмування, запропонованими Apple, залишаються Objective-C та Swift. Хоча Swift спочатку був побудований на основі Objective-C, він має низку суттєвих переваг над своїм попередником [13]:

- наявність вбудованої системи управління пам'яттю з відстеженням слабких посилань та їх автоматичним звільненням;
- підтримка замикань, що з'явилися в Objective-C значно пізніше;
- переважне використання структур для стандартних типів, що підвищує продуктивність та безпеку;
- можливість реалізувати декілька парадигм програмування (ООП, функціональну, реактивну);
- зручні механізми багатопоточності.

Поряд із цим, сучасним конкурентом у побудові інтерфейсів є SwiftUI – декларативний UI-фреймворк. Попри свою перспективність, Swift має низку переваг і над ним [14]:

- зрілість та стабільність – Swift використовується з 2014 року, тоді як SwiftUI є відносно новим;
- ширша підтримка сторонніх бібліотек та SDK;
- більші можливості для кастомізації інтерфейсу, що важливо для складних застосунків (наприклад, графічних редакторів);
- більший контроль над продуктивністю – Swift дозволяє оптимізувати важкі обчислення та інтеграцію з ML-моделями, тоді як SwiftUI поки має деякі обмеження.

Таким чином, Swift на даному етапі залишається найбільш збалансованим та перевіреним рішенням для реалізації мобільного застосунку з інтегрованими алгоритмами машинного навчання.

2.1.2 Огляд особливостей середовища розробки

Для реалізації мобільного графічного редактора під iOS було обрано офіційне середовище Xcode, яке на сьогодні залишається найповноціннішим інструментом для розробки застосунків в екосистемі Apple.

У табл. 2.2 наведено порівняння Xcode з декількома сучасними альтернативами.

Таблиця 2.2 – Порівняння можливих середовищ розробки

Характеристика	XCode	Visual Studio	VS Code
1	2	3	4
Доступність	Безкоштовно	Умовно безкоштовно	Безкоштовно
Підтримка Apple SDK	Повна	Часткова	Обмежена
Підтримка ML	Вбудований інструментарій	Інтеграція через .NET	Залежить від плагінів

Продовження таблиці 2.2

1	2	3	4
Інструменти розробки	Редактор інтерфейсів, Симулятор, Автопідстановка, Дебагер, Тестування, Контроль версій.	Компілятор, Дебагер, Редактор коду, Тестування.	Редактор коду, Термінал.
Компілятор	Власний LLVM	Roslyn	Зовнішні
Оновлюваність	Разом з Swift та iOS	Часті	Залежно від плагінів

XCode – офіційне інтегроване середовище розробки від компанії Apple для створення програмного забезпечення під macOS, iOS, iPadOS, watchOS та tvOS [15]. Містить повний набір інструментів для розробки, тестування та налагодження застосунків, підтримує Swift, Objective-C і надає доступ до всіх сучасних фреймворків Apple, включно з CoreML, Vision та ARKit.

Visual Studio – середовище розробки, розроблене компанією Microsoft, яке дозволяє створювати застосунки для Android, iOS, macOS і Windows з єдиною кодовою базою [16]. Проте інтеграція з iOS SDK і фреймворками Apple є частковою і вимагає використання додаткових обгортки або симуляторів.

VS Code – легке середовище розробки, яке може використовуватися як допоміжний інструмент для редагування коду Swift. Однак воно не забезпечує повноцінного середовища для побудови інтерфейсу, тестування або відлагодження iOS-застосунків.

XCode має низку суттєвих переваг, що роблять його основним середовищем для створення застосунків під iOS:

- повна інтеграція з iOS SDK та усіма фреймворками Apple;

- вбудовані інструменти тестування, дебагінгу та профілювання продуктивності;
- підтримка CoreML і Vision безпосередньо у межах середовища;
- вбудована система керування версіями;
- часті оновлення, що синхронізуються з новими версіями iOS та Swift;
- велика спільнота розробників і детальна офіційна документація.

Таким чином, для реалізації мобільного графічного редактора з підтримкою методів машинного навчання найбільш оптимальним вибором є XCode, оскільки воно забезпечує максимальну стабільність, сумісність та ефективність розробки.

2.2 Структурна схема системи

Структурна схема системи зазначена на рис. 2.1.

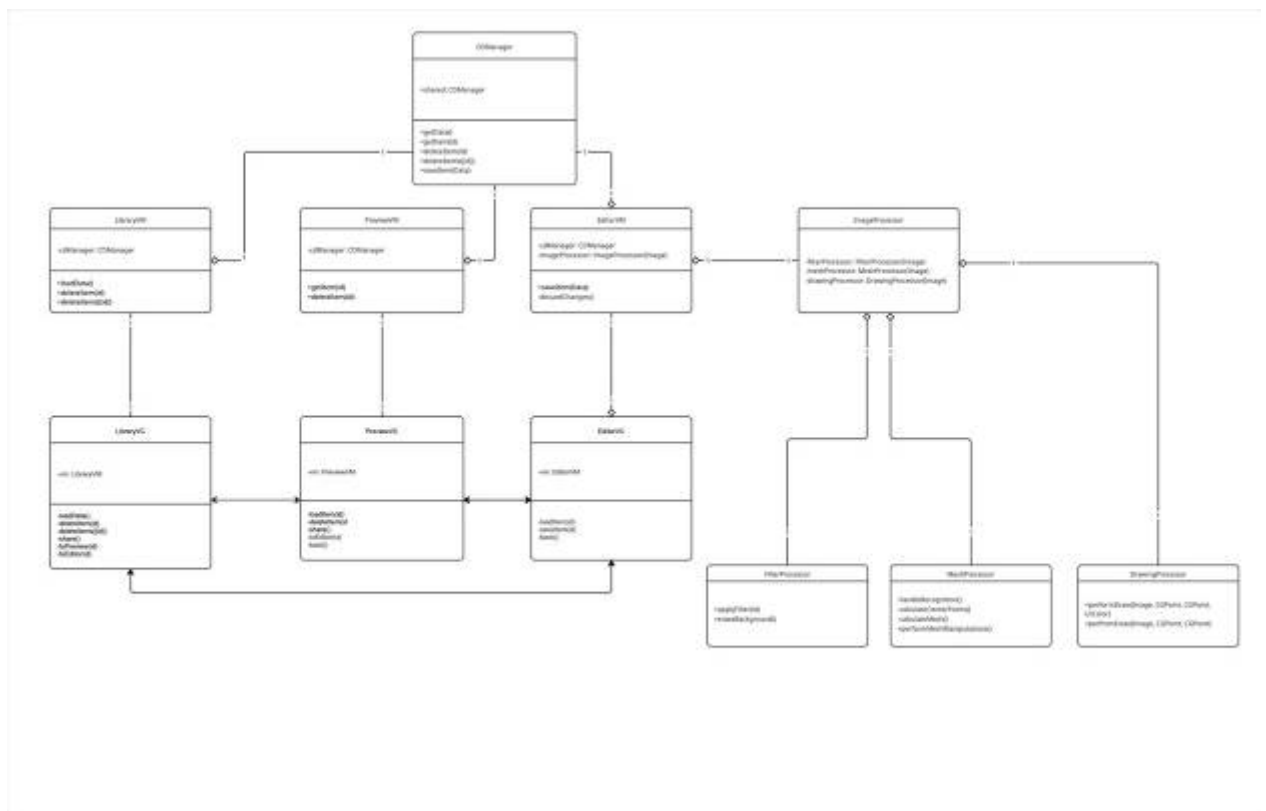


Рисунок 2.1 – Структурна схема системи

2.3 Опис модулів розроблюваної системи

Система складається з наступних розроблених модулів:

- менеджер, який відповідає за спілкування з базою даних;
- процесор, який відповідає за всі маніпуляції із зображенням;
- процесор, який відповідає за накладання фільтрів на зображення;
- процесор, який відповідає за знаходження опорних точок на тілі людини, якщо такі є та за маніпуляції з ними;
- процесор, який відповідає за малювання;
- модуль сторінки бібліотеки, який буде відображати перелік збережених зображень та надає можливість видалення, поширення, доступу до редактора та перегляду;
- модуль сторінки перегляду зображення, який дає можливість видалення, поширення та доступ до редактора;
- модуль сторінки редактора, який надає можливості графічних маніпуляцій над зображенням та можливість зберегти зображення до бібліотеки.

2.4 Метод локальної обробки зображень з використанням оптимізованої ML-моделі

У межах даної роботи запропоновано метод локальної обробки оптимізованої моделі машинного навчання для визначення опорних точок людини з подальшою графічною трансформацією в мобільному середовищі iOS. Цей метод орієнтований на виконання обчислень без використання серверних ресурсів, що забезпечує мінімальні затримки, захищеність даних та можливість інтерактивного редагування зображень у реальному часі.

Метод складається з трьох основних етапів, що реалізують повний цикл аналізу та трансформації зображення.

2.4.1 Етап первинного аналізу зображення

На першому етапі здійснюється попередня обробка вхідного зображення, що включає:

- нормалізацію розміру та кольорового простору;
- масштабування до необхідних розмірів ML-моделі;
- підготовку тензорного представлення для інференсу через CoreML.

Для обчислень використовується оптимізована модель вивлення опорних точок людини, що інтегрована в бібліотеку Vision. Оптимізація включає:

- використання моделі у форматі CoreML Neural Network Specification;
- апаратне прискорення через Apple Neural Engine;
- зниження розрядності ваг, що зменшує затримку інференсу та навантаження на GPU.

Результатом є набір координат у вигляді масиву параметрів X_i , Y_i , що описують положення ключових точок скелета на зображенні.

2.4.2 Етап побудови структурної моделі опорних точок

На другому етапі координати ключових точок використовуються для побудови меш-структури, яка описує топологію тіла.

Така структура містить:

- список ключових точок;
- список зв'язків між ними (суглоби, сегменти тіла);
- параметри жорсткості та обмеження для подальших трансформацій.

Для підвищення точності застосовується алгоритм адаптивної корекції, який:

- усуває можливі розриви між сегментами;
- згладжує різкі аномалії координат;

- застосовує вагові коефіцієнти для важливих точок (плечі, таз, тощо).

Цей етап формує структурне представлення скелета, придатне для маніпуляцій у графічному редакторі.

2.4.3 Етап графічної трансформації та інтеграції в редакторі

На цьому етапі побудована структура використовується для трансформації зображення.

Метод забезпечує:

- локальне деформування вибраних сегментів тіла;
- зміну пози, орієнтації або довжини сегментів;
- накладання корекційних масок та фільтрів;
- інтеграцію результату в робочу область редактора.

Реалізація відбувається за допомогою бібліотеки CoreImage (використовуючи CIColor, CIImage, CIFilter та ін.).

Головною особливістю є те, що всі трансформації виконуються виключно на мобільному пристрої, без використання серверних обчислень, що відражається в роботі додатку як:

- висока продуктивність;
- відносно низьке енергоспоживання;
- повна приватність оброблених зображень;
- висока стабільність.

2.4.4 Переваги та недоліки запропонованого методу

Запропонований метод локальної обробки зображень, що поєднує визначення опорних точок людини та подальшої графічної трансформації, має низку технічних переваг у контексті мобільної розробки:

- локальне виконання всіх обчислень, що забезпечує низьку затримку та високу конфіденційність;

- оптимізація під одну конкретну платформу, що підвищує продуктивність;
- можливість інтерактивного редагування зображень у реальному часі;
- стабільність роботи та відсутність залежностей від мережі;
- узгодженість результатів завдяки використанню внутрішньої меш-структури точок.

Водночас, існують і певні обмеження, зумовлені специфікацією використання локальних ML-моделей та апаратних ресурсів:

- залежність продуктивності від технічних характеристик пристрою;
- зниження точності моделі в складних умовах;
- підвищене навантаження на CPU та GPU;
- обмеження на масштабування моделі;
- потенційно менша точність порівняно з повномасштабними серверними моделями.

2.5 Висновки за розділом 2

В результаті проведеного аналізу та порівняння мов програмування встановлено, що Swift є найбільш доцільною та популярною мовою для розробки мобільних застосунків під iOS. Ця багатопарадигмова компільована мова програмування, розроблена компанією Apple поєднує простоту синтаксису, високу продуктивність та нативну інтеграцію з фреймворками машинного навчання CoreML і Vision.

Середовищем розробки було обрано XCode – офіційне інтегроване середовище, створене Apple для розробки програмного забезпечення під свої платформи. XCode забезпечує повну сумісність із мовою Swift, має зручні інструменти для побудови інтерфейсу, тестування та налагодження програм, а також надає можливість прямої інтеграції моделей машинного навчання.

У межах даної роботи також розроблено метод локальної обробки

зображень, який поєднує оптимізовану ML-модель визначення опорних точок людини з подальшою графічною трансформацією на мобільному пристрої, що дозволило забезпечити інтерактивну роботу редактора без серверних обчислень.

Таким чином, обрані інструменти та методи є оптимальними для реалізації мобільного графічного редактора з функцією розпізнавання опорних точок людини на зображеннях.

В розділі описані основні модулі розроблюваної системи, такі як: «Бібліотека», «Перегляд» та «Редактор».

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Алгоритм функціонування розроблюваної системи

Схема взаємодії модулів системи між собою зазначена на рис. 3.1.

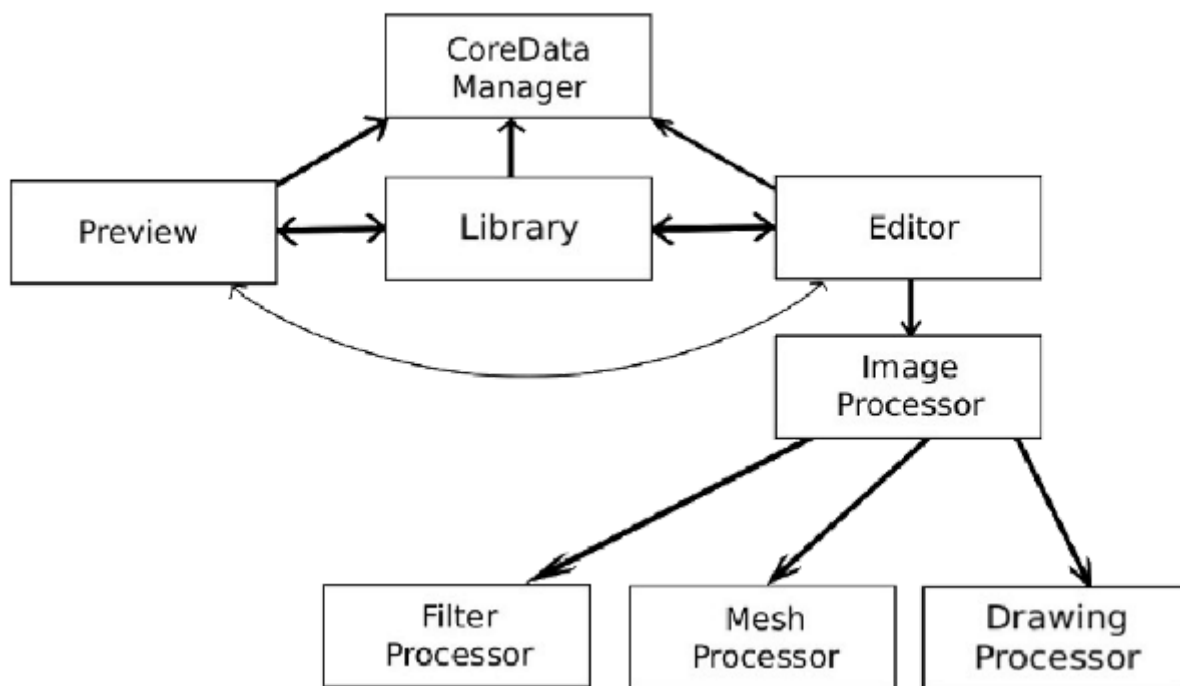


Рисунок 3.1 – Взаємодія модулів системи між собою

Згідно схеми, наведеної вище, алгоритм роботи програмної системи наступний:

- екран бібліотеки, використовуючи менеджер бази даних відображає збережені зображення та надає можливість видалити обрані зображення, а також надає доступу до екранів перегляду та редагування;
- екран перегляду, використовуючи менеджер бази даних відображає обране зображення та надає можливість видалити, а також надає доступу до екрану редагування;
- екран редагування, використовуючи менеджер бази даних надає можливість зберігати зображення та може надає можливості повернутись на екран перегляду або бібліотеки;

– процесори фільтрів, малювання та мешу пов’язані з процесором зображення та надають можливості редагувати фільтри, малюнки та меш відповідно;

– екран редагування пов’язаний з процесором зображення, який відповідає за редагування зображення.

3.2 Основні рішення щодо реалізації системи

У табл. 3.1–3.3 перераховані основні класи та функції модулів, а також їх призначення.

Таблиця 3.1 – Опис модулів менеджерів та процесорів

Назва компоненту	Назва класу або функції	Короткий опис
Managers	class CManager	Реалізує паттерн Singleton та відповідає за взаємодію з базою даних
Managers	class ImageProcessor	Реалізує можливості редагування зображення
	class FilterProcessor	Реалізує редагування фільтрів
	class MeshProcessor	Реалізує пошук опорних точок на тілі людини та взаємодію з меш-таблицями
	class DrawingProcessor	Реалізує можливості малювання на зображенні

Таблиця 3.2 – Опис модулю сторінки бібліотеки

Назва компоненту	Назва класу або функції	Короткий опис
Library Screen	class LibraryVC	Реалізує відображення графічних елементів та взаємодію з ними
	class LibraryVM	Реалізує обробку взаємодії з графічними елементами та зв'язок з менеджером бази даних і зв'язок з різними екранами

Таблиця 3.3 – Опис модулю сторінки перегляду

Назва компоненту	Назва класу або функції	Короткий опис
Preview Screen	class PreviewVC	Реалізує відображення графічних елементів та взаємодію з ними
	class PreviewVM	Реалізує обробку взаємодії з графічними елементами та зв'язок з менеджером бази даних і зв'язок з різними екранами

У табл. 3.4 наведено опис модулю сторінки редагування з описанням класів, з яких він складається.

Таблиця 3.4 – Опис модулю сторінки редагування

Назва компоненту	Назва класу або функції	Короткий опис
Editor Screen	class EditorVC	Реалізує відображення графічних елементів та взаємодію з ними
	class EditorVM	Реалізує обробку взаємодії з графічними елементами та зв'язок з менеджером бази даних, процесором зображення та зв'язок з різними екранами

3.3 Висновки за розділом 3

У результаті виконання дипломної роботи було спроектовано та описано схему взаємодії основних модулів розробленої системи мобільного графічного редактора. На основі проведеного аналізу побудовано архітектурну структуру застосунку, що відображає взаємодію між інтерфейсною частиною, модулем машинного навчання та модулем обробки зображень.

Розроблено ієрархію класів, які реалізують основні функції системи: завантаження зображень, розпізнавання опорних точок людини, відображення результатів та виконання графічних маніпуляцій. У процесі реалізації використано принципи об'єктно-орієнтованого програмування – інкапсуляцію, наслідування та поліморфізм, що забезпечило гнучкість, масштабованість та зручність підтримки програмного продукту.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення та умови застосування програми

Розроблена програмна система призначена для автоматичного розпізнавання опорних точок людини на зображенні з подальшою візуалізацією та редагуванням отриманих результатів у графічному середовищі.

Застосунок створено у середовищі розробки XCode 16.4 із використанням мови програмування Swift 6.0 та архітектурного підходу MVVM, який забезпечує розділення логіки, представлення та даних.

Для роботи програмного продукту необхідно мати мобільний пристрій під керуванням операційної системи iOS 17.0 або новішої, із камерою та доступом до фотогалереї. Програма не потребує постійного підключення до мережі інтернет, оскільки всі обчислення виконуються локально за допомогою вбудованих ML-бібліотек Apple.

4.2 Характеристики програми

Програма реалізує обробку зображень у реальному часі або роботу з уже збереженими фотографіями. Після завантаження зображення та включення режиму редагування меш-таблиць застосунок ініціалізує модель машинного навчання, яка виконує розпізнавання ключових точок людського тіла.

Результати аналізу візуалізуються у вигляді графічних маркерів, що відображаються поверх зображення. Користувач має змогу змінювати положення точок, масштабувати зображення, виділяти окремі області та зберігати результат у пам'яті пристрою.

Програма працює асинхронно, що забезпечує стабільну роботу та високу швидкодію навіть при обробці великих файлів. Дані про результати обробки зберігаються лише локально, без передачі на зовнішні сервери.

4.3 Вхідні та вихідні дані

Вхідними даними є зображення, отримане з фотогалереї користувача або зняте безпосередньо з камери пристрою.

Вихідними даними є:

- відредаговане зображення;
- послідовний перелік виконаних змін.

4.4 Звертання до програми

Для запуску програмної системи необхідно:

- завантажити вихідний код із репозиторію;
- відкрити проєкт у XCode 16.4 або новіше;
- для роботи графічного редактора достатньо запустити на відповідному симуляторі або ж підключити реальний девайс та запустити;
- для роботи ML моделей застосунок потрібно запускати тільки на реальному девайсі.

4.5 Повідомлення

Основні типи повідомлень:

- "Gallery/Camera Access Required" – необхідність надання доступу до галереї або камери;
- "Saved to your gallery" – збереження до галереї відбулось успішно;
- "Image/s deleted" – успішне видалення зображення;
- "Something went wrong" – сталась невідома помилка;
- "Could not find body" – не вдалось знайти тіло.

4.6 Висновки за розділом 4

Керівництво програміста містить такі підрозділи: призначення та умови застосування програми, характеристики програми, звернення до програми, вхідні та вихідні дані, а також повідомлення системи.

Розроблена програмна система призначена для автоматичного розпізнавання опорних точок людини на зображеннях із подальшою візуалізацією та можливістю редагування результатів. У роботі застосовано мову програмування Swift 6.0 та середовище розробки Xcode 16.4, що забезпечує повну сумісність із бібліотеками машинного навчання.

Програмна система працює автономно, без підключення до мережі, виконуючи всі обчислення на пристрої. Вихідними даними є відредаговане зображення та перелік всіх внесених змін для подальшого редагування.

У розділі описано методи завантаження та запуску програмного продукту, а також основні повідомлення, що відображаються під час роботи.

Представлене керівництво може використовуватися як довідковий документ для розробників при подальшій підтримці або розширенні функціональності застосунку.

5 КЕРІВНИЦТВО ОПЕРАТОРА

5.1 Призначення та умови виконання програми

Розроблена програмна система призначена для автоматичного розпізнавання опорних точок людини на зображеннях із подальшою можливістю їхньої візуалізації, редагування та збереження результатів у графічному форматі. Програма дає змогу швидко визначати ключові координати тіла (суглоби, кінцівки тощо) та використовувати ці дані для аналізу, візуальних експериментів або інтерактивних демонстрацій.

Застосунок створено у середовищі програмування XCode 16.4 за допомогою мови програмування Swift 6.0 та архітектурного підходу MVVM, який забезпечує чітке розділення логіки, обробки даних і представлення. Для реалізації розпізнавання використано фреймворки CoreML і Vision, що дозволяють виконувати машинне навчання локально на пристрої без потреби в інтернет-з'єднанні.

Отримані результати розпізнавання можуть бути використані для подальшої обробки та дослідження особливостей положення тіла людини, а також для демонстрації можливостей сучасних методів машинного навчання у прикладних програмних рішеннях. Реалізована програмна система орієнтована на зручність використання та може слугувати основою для подальшого розширення функціоналу, зокрема шляхом інтеграції додаткових алгоритмів аналізу рухів або підтримки інших форматів вхідних даних.

Для функціонування програмної системи необхідно мати мобільний пристрій під керуванням iOS 17.0 або новішої версії, оснащений камерою та доступом до фотогалереї. Підключення до мережі не є обов'язковим, оскільки всі обчислення виконуються автономно, на апаратних потужностях пристрою.

5.2 Виконання програми

При першому запуску застосунку користувач побачить сторінку з вибором одного з варіантів створення нового проєкту (рис. 5.1).

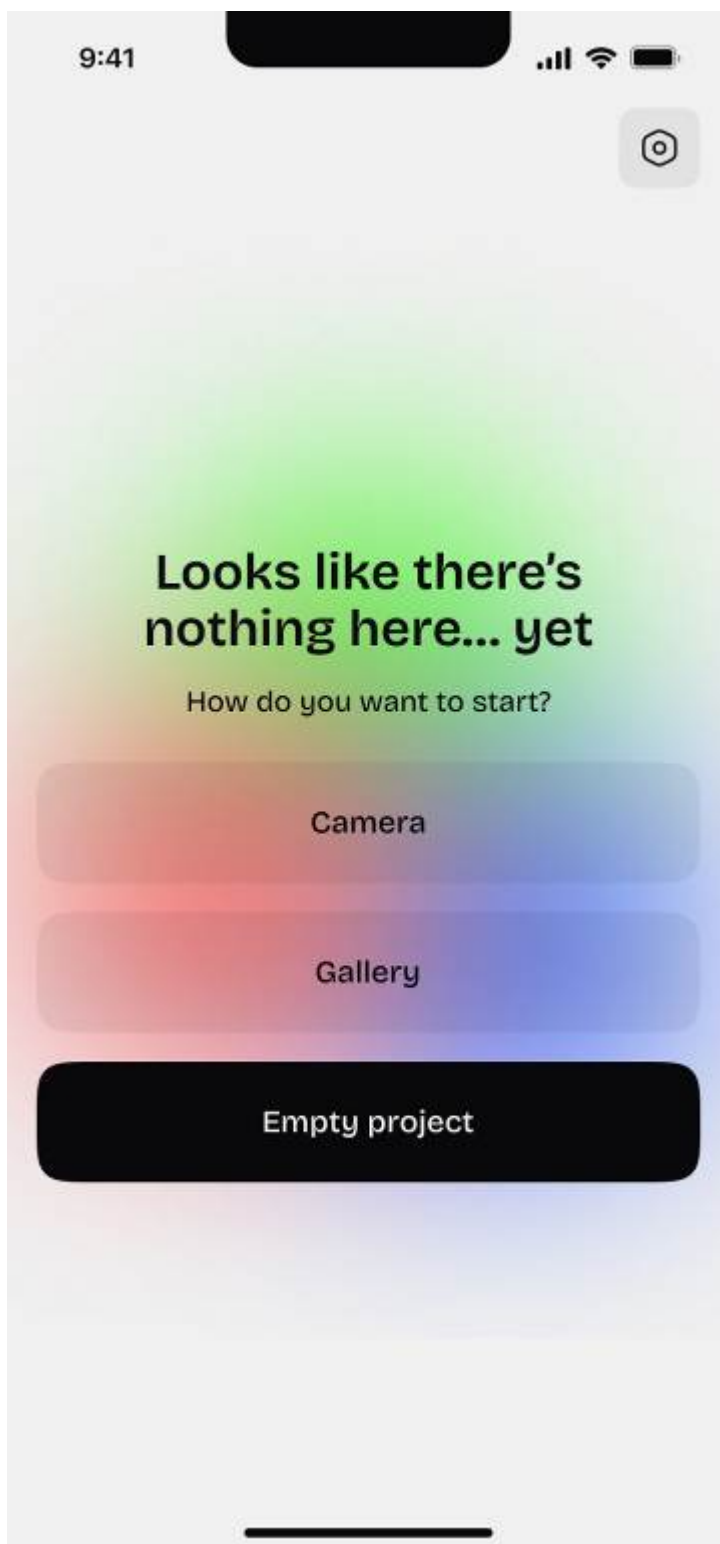


Рисунок 5.1 – Перший запуск застосунку

На цьому екрані користувач бачить, аналогічно першому запуску, варіанти вибору варіантів створення нового проєкта, якщо бібліотека пуста, або перелік бібліотеки (рис 5.2).

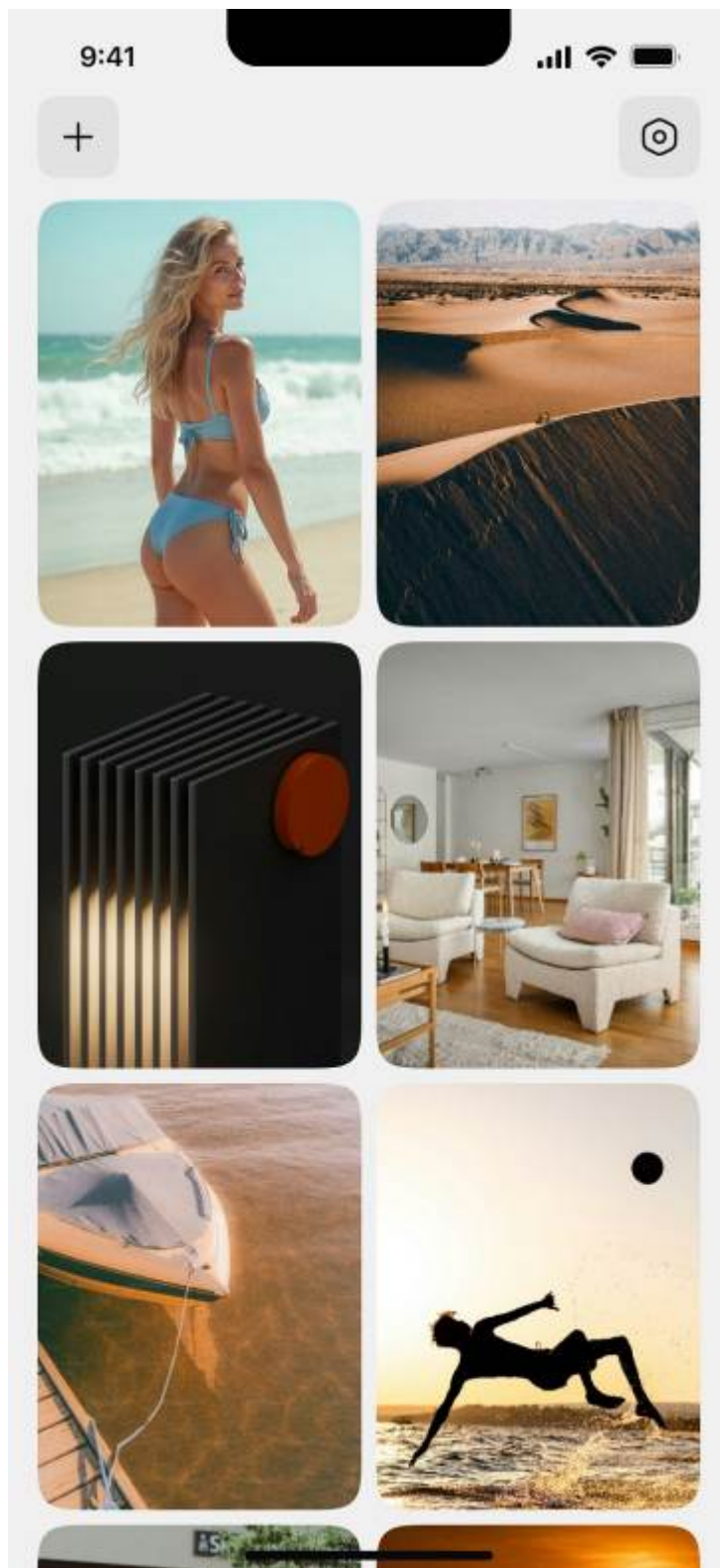


Рисунок 5.2 – Перегляд галереї проєктів

При натисненні на кнопки «Camera», або «Gallery», в разі відсутності доступу до камери, або галереї користувач побачить відповідне повідомлення (рис. 5.3).

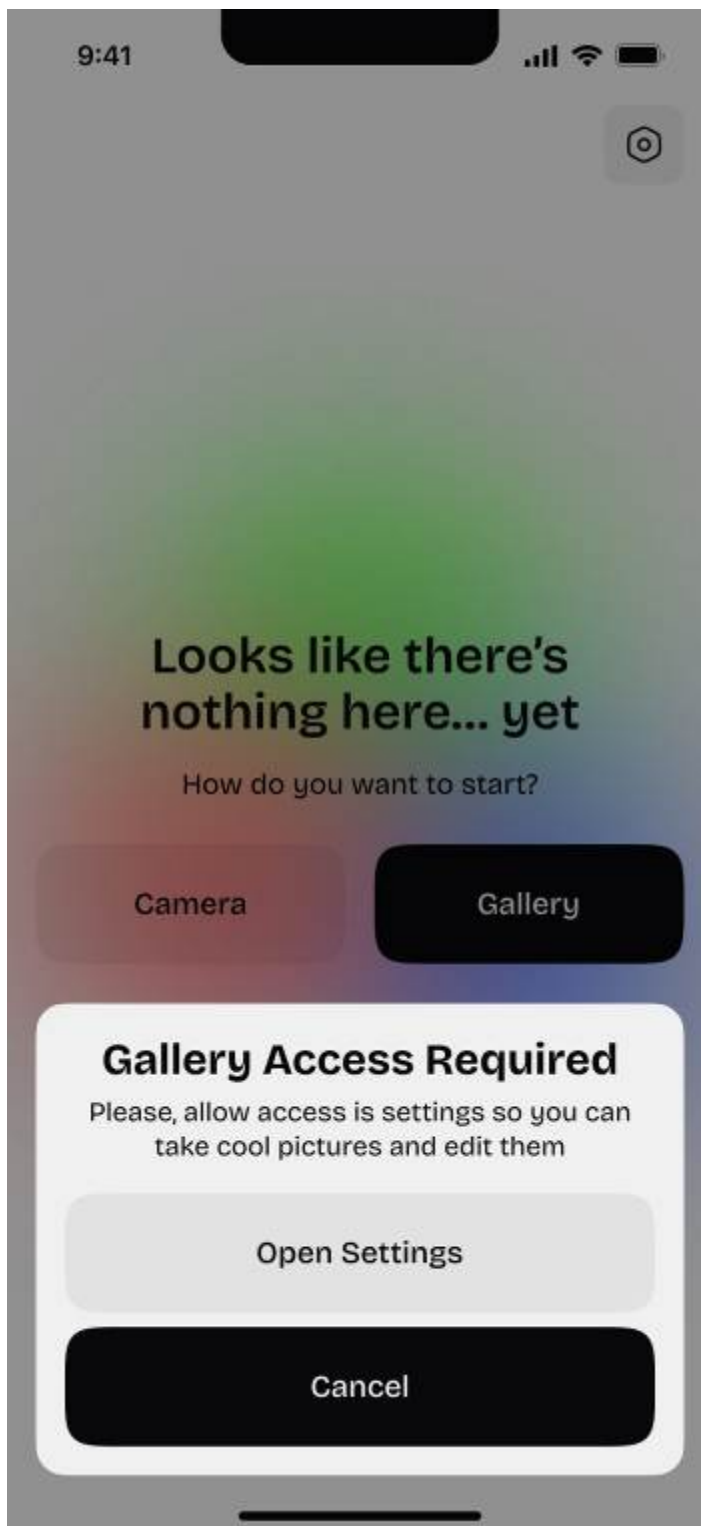


Рисунок 5.3 – Повідомлення про відсутність доступу

При натисканні на кнопку зі значком «+», користувач отримає варіанти створення нового проєкта, аналогічні першому запуску, або пустій сторінці (рис 5.4).

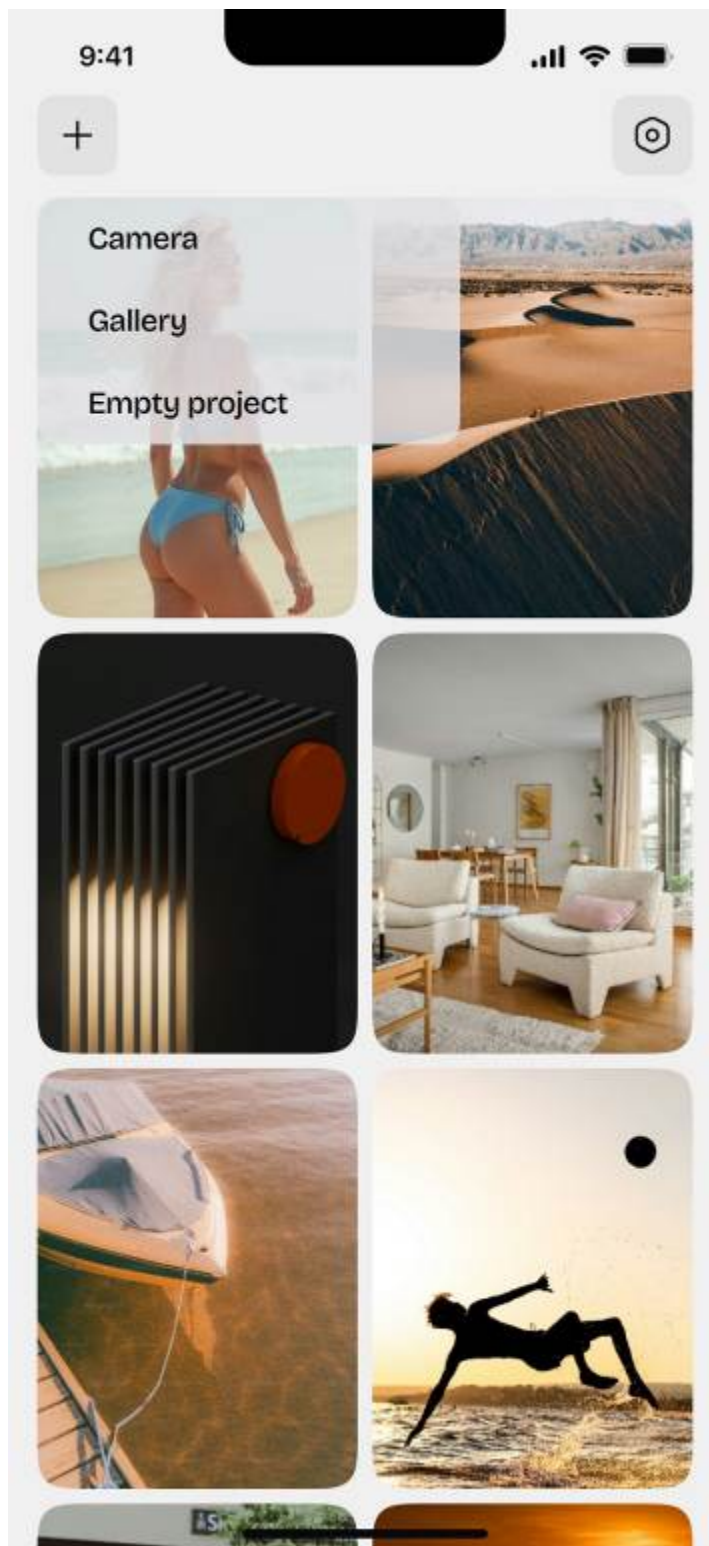


Рисунок 5.4 – Створення нового проєкта

При довгому натисканні на елемент галереї, користувач побачить контекстне меню з можливими варіантами подальшої взаємодії з цим елементом (рис. 5.5).

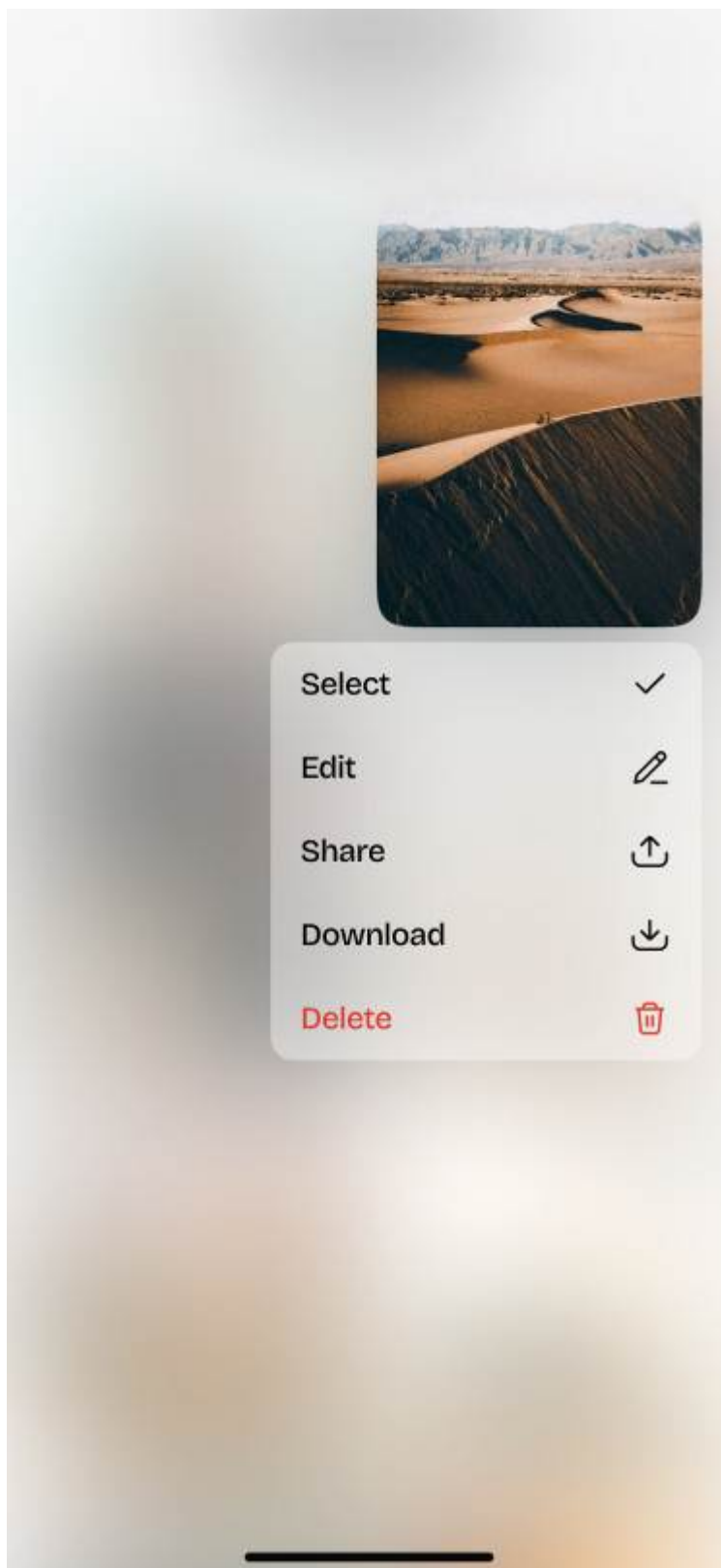


Рисунок 5.5 – Контекстне меню елемента

Якщо натиснути на пункт «Select», обраний елемент буде виділено, що візуально відображається як ободка чорного кольору та знизу з'явиться меню з переліком можливої взаємодії (рис. 5.6).

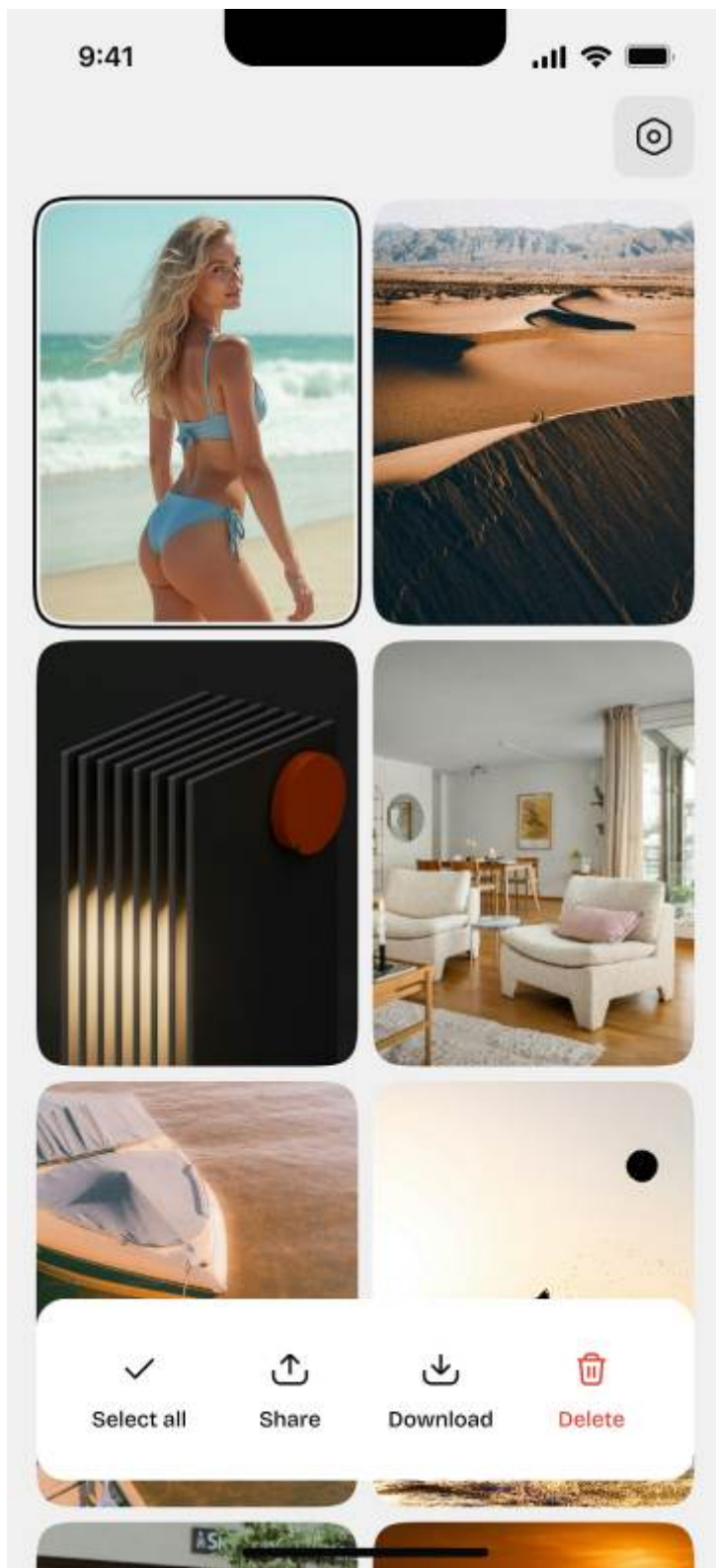


Рисунок 5.6 – Обраний елемент

Після простого натиснення на елемент відкривається сторінка перегляду зображення (рис. 5.7).

На цій сторінки користувач бачить саме зображення та знизу меню з можливими варіантами взаємодії.

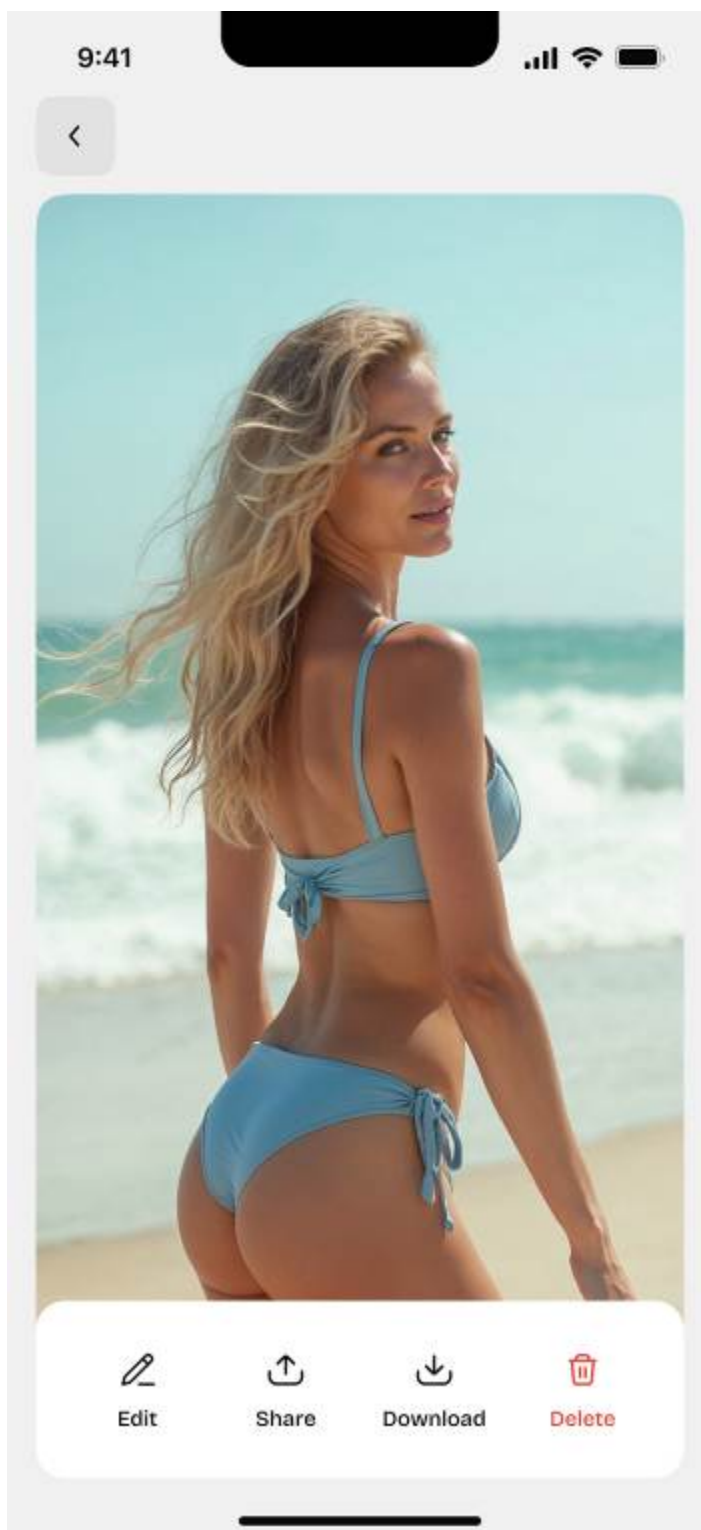


Рисунок 5.7 – Сторінка перегляду проєкта

При натисканні на кнопку «Edit» відкривається сторінка редагування зображення (рис. 5.8).

По центру екрану користувач бачить саме зображення, це також є зоною всіх взаємодій з ним.

Знизу можна побачити меню з можливими варіантами простої обробки, такої як малювання, накладання тексту, фільтри та інше.

Згори знаходяться кнопки для відміни/повтору останньої дії та кнопка «...», по натисканню на яку користувач побачить додаткове контекстне меню, де він зможе знайти можливість включити меню редагування по опорних точках, що автоматично запускає алгоритм пошуку (рис. 5.9).



Рисунок 5.8 – Екран редагування зображення

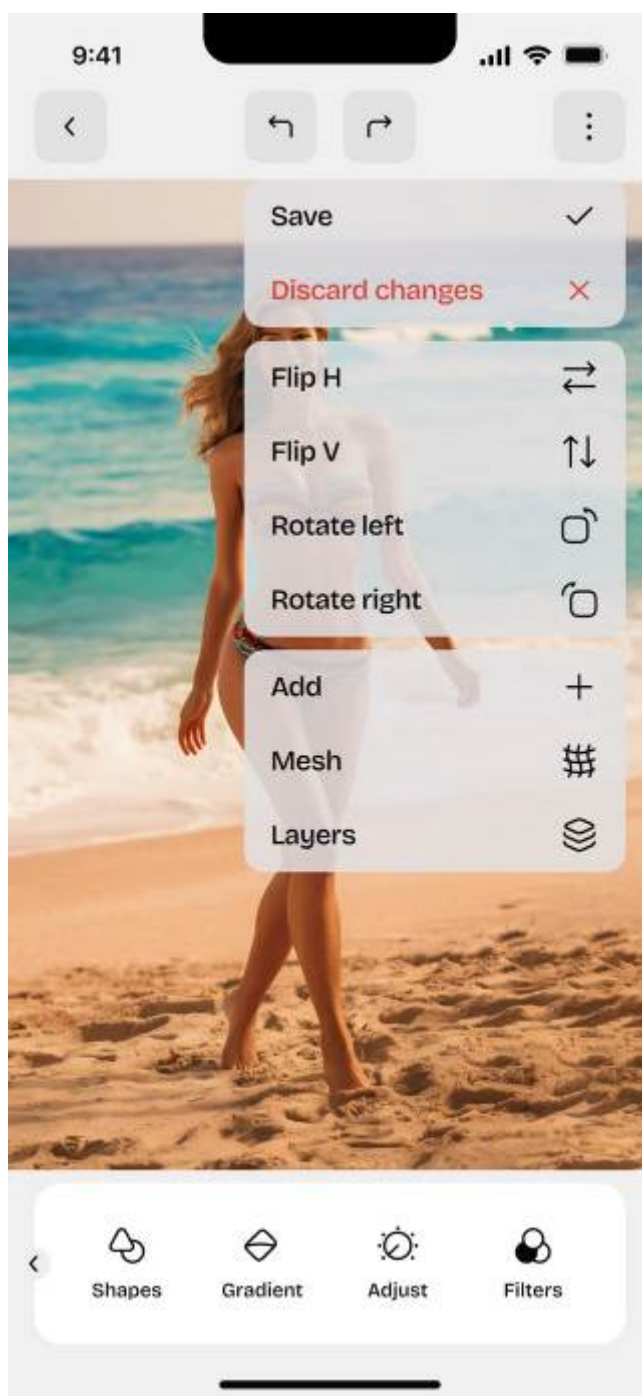


Рисунок 5.9 – Контекстне меню редагування

Після натиснення на пункт меню «Mesh», нижнє меню змінюється на меню редагування меш-таблиць та з'являється кнопка, яка дає можливість включити/виключити перегляд змін (рис. 5.10).

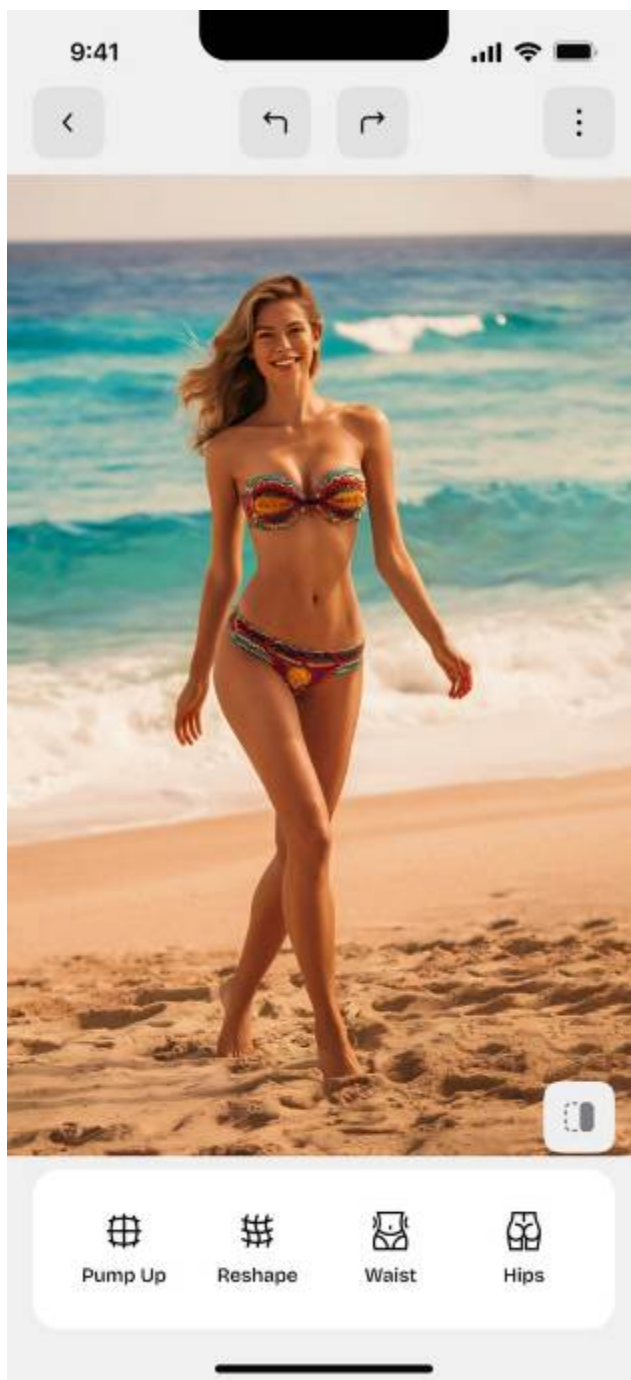


Рисунок 5.10 – Меню редагування меш-таблиць

Після натиснення на пункт меню «Waist», нижнє меню замінюється на меню редагування талії (рис. 5.11).

При цьому, додатково на зображенні, в районі талії людини по знайденим контрольним точкам, з'являються додаткові контрольні елементи з якими користувач може взаємодіяти.

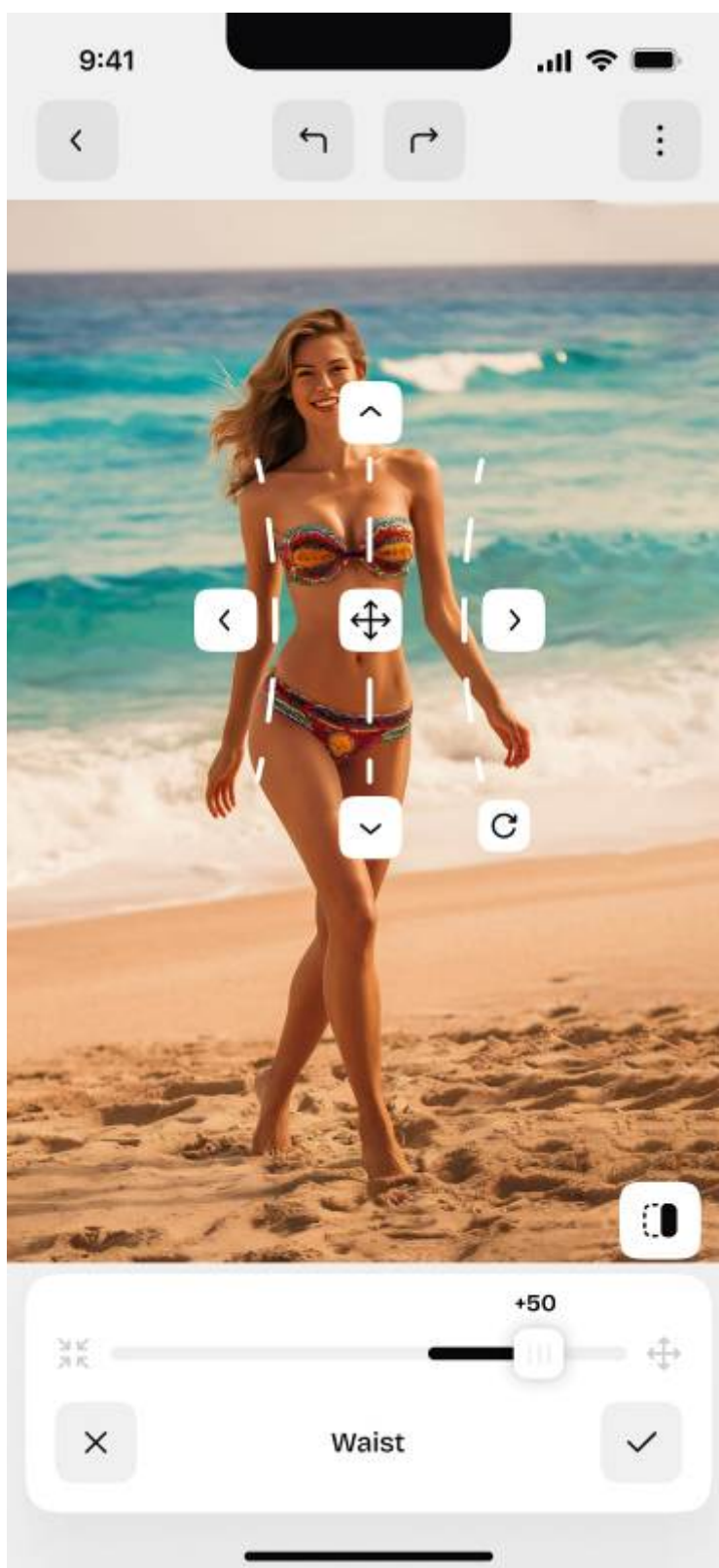


Рисунок 5.11 – Редагування талії людини

В цьому меню двосторонній повзунок зі значеннями від -100 до +100 відповідає за звуження/потовщення в обраній зоні, директивні стрілки відповідають за розтягнення зони, багато направлена стрілка відповідає за

коригування положення зони, а кругла стрілка відповідає за перевертання зони.

При натисненні на пункт меню «Hips» нижнє меню міняється на меню редагування стегон (рис. 5.12).

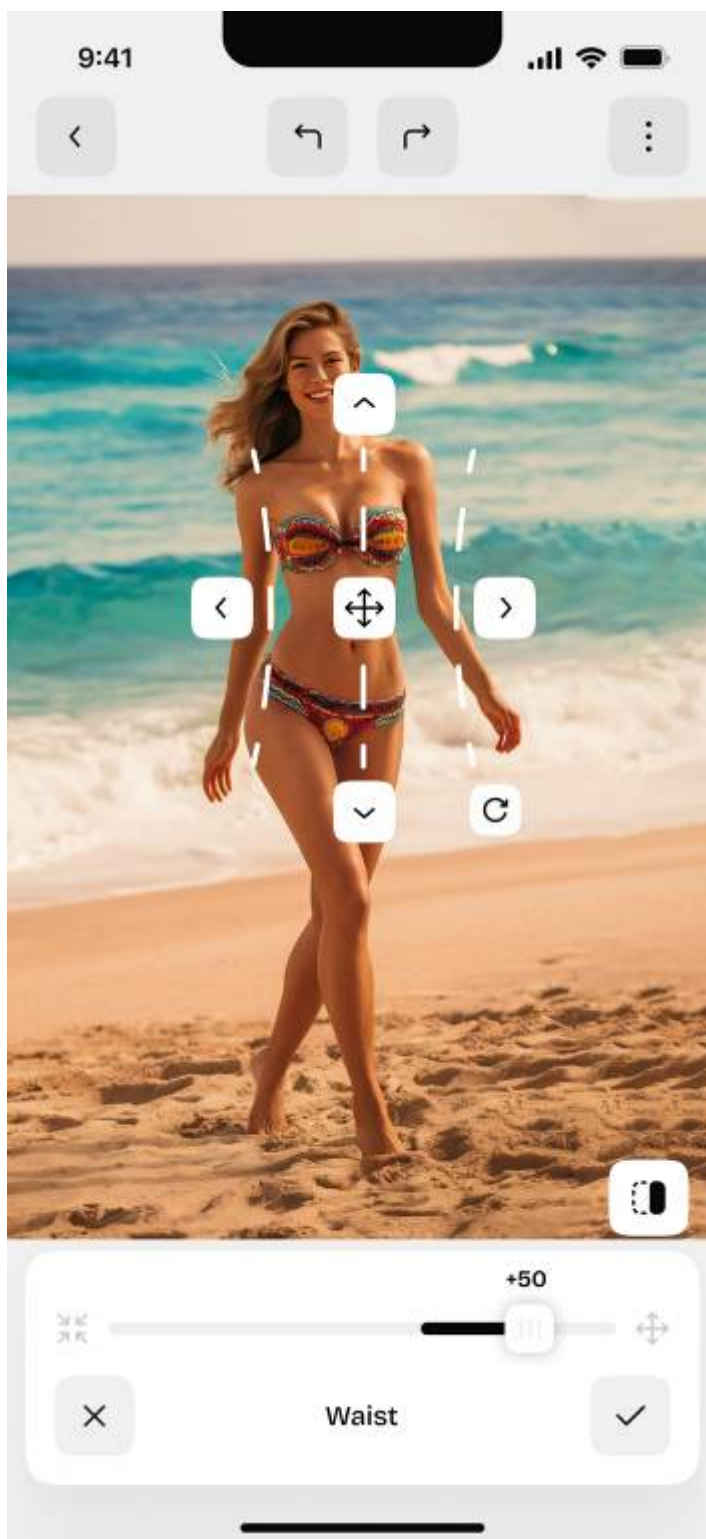


Рисунок 5.12 – Редагування стегон людини

В цьому меню, двосторонній повзунок зі значеннями -100/+100 відповідає за звуження/розтягнення зони стегон, яка була визначена віртуальною моделлю.

При натисненні на пункт меню «Points» нижнє меню міняється на меню редагування опорних точок (рис. 5.13).

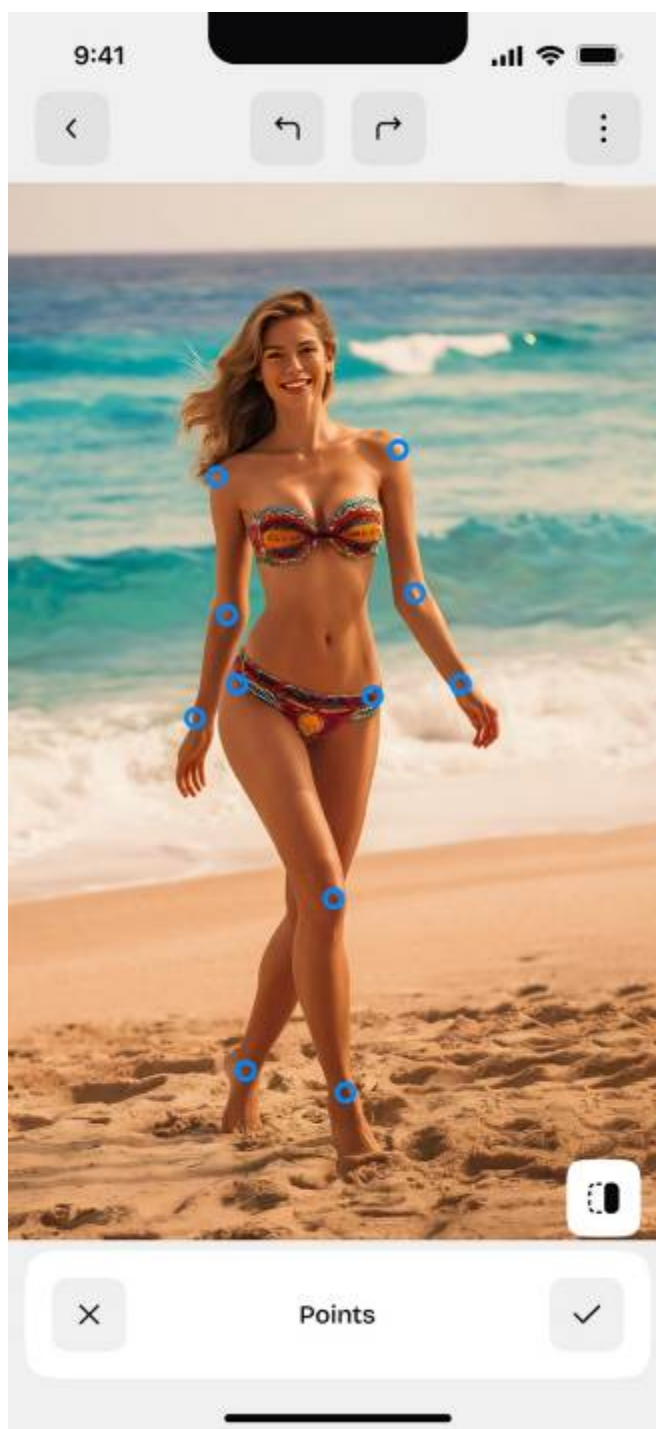


Рисунок 5.13 – Редагування контрольних точок

При цьому, на зображенні, в районі контрольних точок, визначених віртуальною моделлю накладаються маркери з якими користувач може взаємодіяти.

При наявності незбережених змін після натискання на кнопку виходу користувач побачить попередження (рис. 5.14).

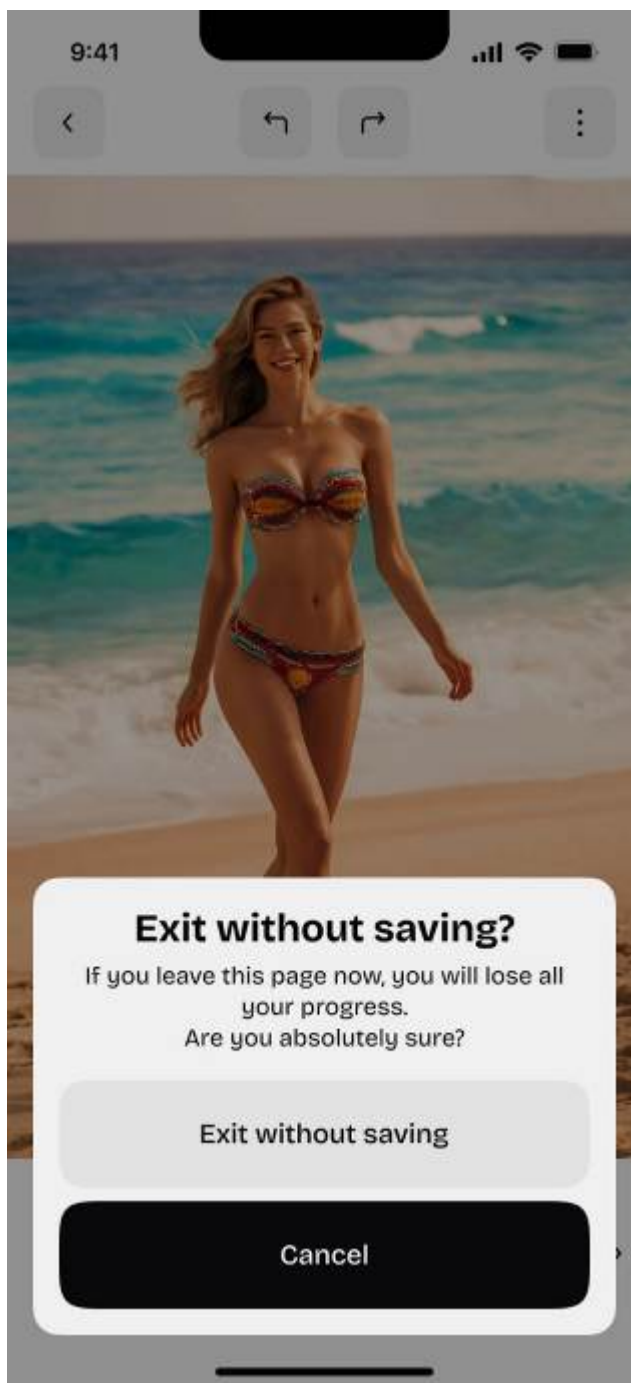


Рисунок 5.14 – Попередження про незбережені зміни

При натисканні в контекстному меню пункту «Save» відкривається

меню збереження поточного проєкта (рис. 5.15).

При натисненні на кнопку «Save to gallery», в перший раз з'явиться системне меню з запрошуванням доступу для збереження в галерею пристрою. В разі, якщо користувач не надасть доступу, та якщо користувач відключить цей доступ в подальшому в налаштуваннях пристрою, з'явиться вікно з пропозицією включити його як при першому запиті на доступ до галереї.

При наданому доступі зображення буде збережено до галереї та локальної бібліотеки проєктів.

При натисненні на кнопку «Save to library» зображення буде збережено в локальну бібліотеку проєктів.

Збережені проєкти в подальшому будуть відображатись на головній сторінці.



Рисунок 5.15 – Сторінка збереження проєкта

При успішному збереженні в обох випадках буде відображено

відповідне повідомлення, та користувача буде перенаправлено на головну сторінку застосунку (рис. 5.16).

При виникненні будь-якої помилки буде відображено відповідне повідомлення (рис. 5.17).

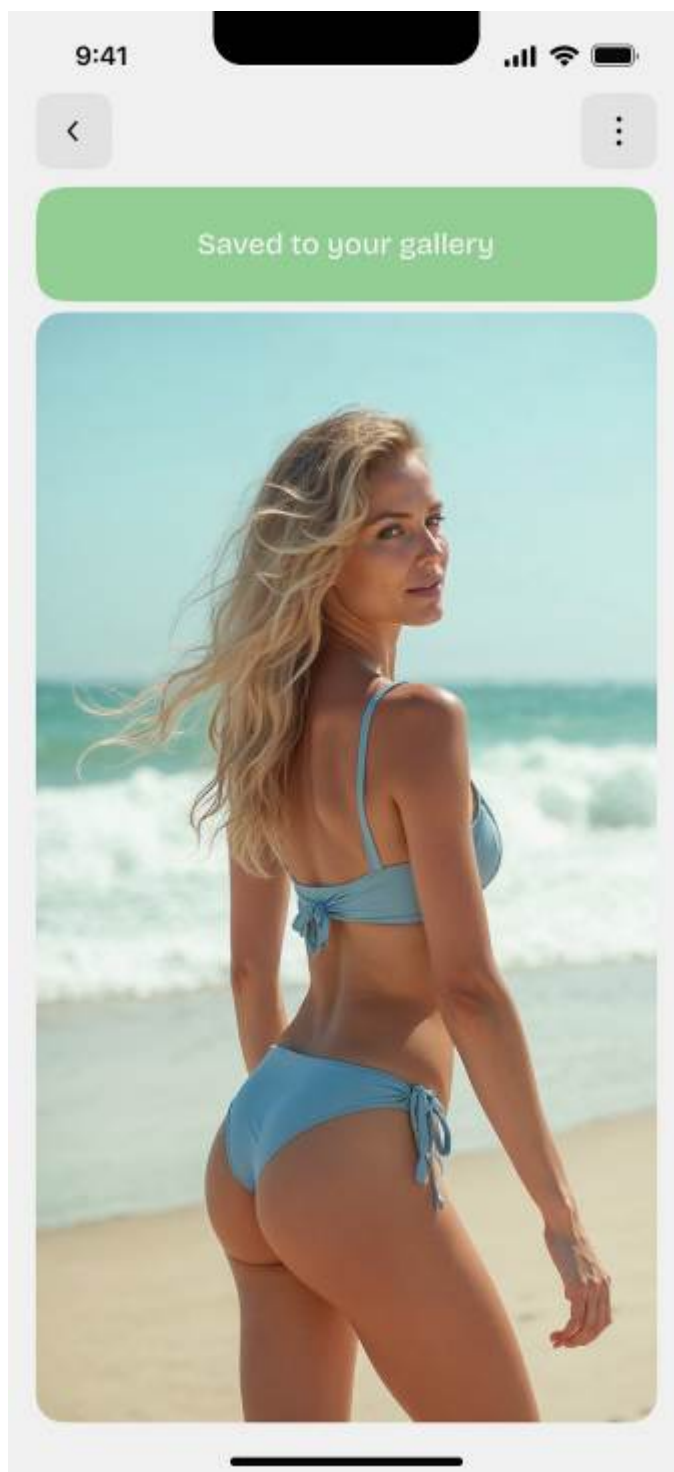


Рисунок 5.16 – Повідомлення про успішне збереження

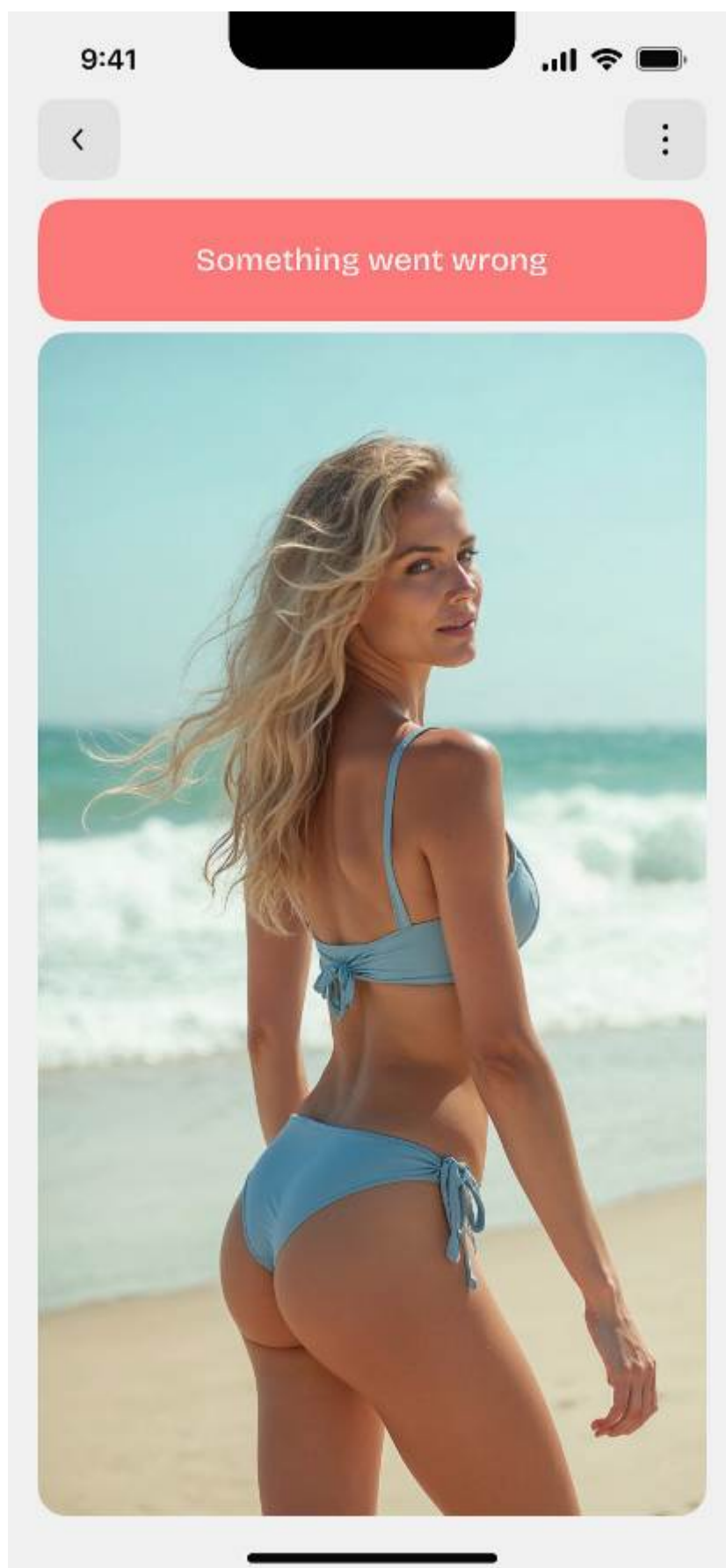


Рисунок 5.17 – Повідомлення про помилку

5.3 Висновки за розділом 5

На наш погляд, використання розробленого мобільного застосунку дозволяє значно підвищити ефективність аналізу та обробки зображень, що містять зображення людини. Завдяки застосуванню методів машинного навчання користувач отримує можливість автоматично визначати опорні точки тіла без потреби у спеціальних знаннях або складних налаштуваннях.

Інтерфейс програми є інтуїтивно зрозумілим та зручним у використанні, що дає змогу швидко виконувати основні операції — завантаження зображення, розпізнавання, візуалізацію та збереження результатів. Такий підхід спрощує роботу з візуальними даними та відкриває можливості для подальшого використання результатів у навчальних, дослідницьких чи творчих цілях.

Розроблений застосунок демонструє перспективність інтеграції технологій машинного навчання у мобільні графічні редактори та може стати основою для створення більш складних систем аналізу й обробки зображень у майбутньому.

ВИСНОВКИ

Під час розробки магістерської дипломної роботи було детально вивчено та проаналізовано предметну область, що стосується застосування методів машинного навчання для розпізнавання опорних точок людини на зображеннях. Розроблено архітектуру та ієрархію класів програмного продукту, яка повністю реалізує поставлену задачу. У процесі створення системи використано основні принципи об'єктно-орієнтованого програмування, що забезпечило масштабованість, надійність та гнучкість коду.

У результаті проведеного дослідження отримано такі висновки:

- Визначено, що мобільний застосунок є ефективною формою реалізації програмного забезпечення, здатного виконувати складні обчислювальні операції на мобільних пристроях без потреби у зовнішніх серверах.

- Проаналізовано основні підходи до створення мобільних застосунків: нативні, веборієнтовані та гібридні. Обґрунтовано вибір нативного підходу як оптимального для забезпечення високої продуктивності та інтеграції з ML-бібліотеками.

- Досліджено сучасні фреймворки для комп'ютерного зору та машинного навчання в екосистемі iOS — CoreML та Vision, які дозволяють виконувати розпізнавання ключових точок людини безпосередньо на пристрої користувача, забезпечуючи високу швидкість і конфіденційність даних.

- Обґрунтовано вибір мови програмування Swift, яка забезпечує простоту синтаксису, надійність, підтримку багатопоточності та повну сумісність із ML-інструментами Apple.

- У якості середовища розробки обрано XCode 16.4 — офіційне IDE від Apple, яке містить повний набір інструментів для створення, налагодження, тестування та оптимізації програмних рішень під iOS.

– Розроблено прототип мобільного графічного редактора, який виконує автоматичне розпізнавання опорних точок людини на зображеннях, відображає результати у вигляді графічних маркерів та надає можливість їх редагування. Програма працює автономно, без необхідності підключення до мережі Інтернет.

– Запропонований застосунок може бути використаний у сферах графічного дизайну, освіти, спорту, медицини та досліджень рухової активності. Реалізований проєкт демонструє практичну придатність технологій машинного навчання для інтеграції у мобільні застосунки з високою продуктивністю та зручністю користування.

– Таким чином, у роботі виконано дослідження, аналіз і програмну реалізацію методів машинного навчання для розпізнавання опорних точок людини. Отримані результати підтвердили ефективність застосування сучасних інструментів Apple у створенні інтелектуальних мобільних застосунків, що поєднують комп’ютерний зір та графічну обробку зображень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Szeliski R. Computer Vision: Algorithms and Applications. 2nd ed. Cham : Springer, 2022. – 947 p.
2. Chen Y. Monocular Human Pose Estimation: A Survey of Deep Learning-based Methods / Y. Chen, Y. Tian, M. He // Computer Vision and Image Understanding. – 2020. Vol. 192. – 23 p.
3. Deep 3D Human Pose Estimation: A Review / [J. Wang, S. Tan, X. Zhen, S. Xu, F. Zheng, Z. He, L. Shao] // Computer Vision and Image Understanding. – 2021. Vol. 210. – 21 p.
4. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications / [Howard A. G., Zhu M., Chen B., et al.]. – [Electronic resource]. – Access mode: <https://arxiv.org/abs/1704.04861>.
5. DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices / [Lane N. D., Bhattacharya S., Georgiev P., et al.] // Proceedings of IPSN. – Vienna, 2016. – P. 1–12.
6. Mobile Application Testing: A Tutorial / [Bai X., Gao J., Tsai W.T., Uehara T.] // Computer. – 2014. – Vol. 47, No. 2. – P. 46–55.
7. Fowler M. Patterns of Enterprise Application Architecture. – Boston : Addison-Wesley, 2003. – 533 p.
8. Silberschatz A. Operating System Concepts. 10th ed. / F. Silberschatz, P. B. Galvin, G. Gagne. – Hoboken : Wiley, 2018. – 976 p.
9. Apple Inc. iOS Human Interface Guidelines [Electronic resource]. – Access mode: <https://developer.apple.com/design/human-interface-guidelines>.
10. Cox B. Object-Oriented Programming: An Evolutionary Approach / B. Cox, A. Novobilski. – Reading : Addison-Wesley, 1991. – 290 p.
11. Apple Inc. The Swift Programming Language [Electronic resource]. – Access mode: <https://docs.swift.org/swift-book/>.
12. Apple Inc. Introducing SwiftUI [Electronic resource]. – Access mode: <https://developer.apple.com/xcode/swiftui/>.

13. Lattner C. LLVM and the Advent of Swift // Communications of the ACM. – Illinois: University of Illinois, 2023. – Vol. 66, No. 5. – P. 44–53.
14. Ghosh S. Mastering SwiftUI. – Birmingham : Packt Publishing, 2021. – 243 p.
15. Apple Inc. Xcode Overview [Electronic resource]. – Access mode: <https://developer.apple.com/xcode/>.
16. Microsoft Corporation. Visual Studio IDE Documentation [Electronic resource]. – Access mode: <https://visualstudio.microsoft.com/>.

ДОДАТОК А
Технічне завдання

Вступ

Програмний продукт є повнофункціональним редактором зображень для мобільних пристроїв та містить модулі для обробки фотографій, роботи з шарами, застосування фільтрів і ефектів, а також інструменти машинного навчання для автоматичного визначення опорних точок людини. Застосунок орієнтований на користувачів, які потребують професійної мобільної платформи для створення та редагування графічних матеріалів.

A.1 Мета розробки і її призначення

Об'єкт дослідження – процес проектування та розробки програмних засобів, що використовують методи машинного навчання.

Предмет дослідження – програмні засоби для визначення опорних точок людини.

Мета роботи – дослідження та реалізація програмного забезпечення для розпізнавання опорних точок людини для зменшення трудомісткості та часу обробки зображень людини.

Наукова новизна полягає у розробленні методу локальної обробки зображень, що поєднує оптимізовану ML-модель визначення опорних точок людини з подальшою графічною трансформацією на мобільних пристроях.

A.2 Стислий зміст основних теоретичних посилок до розробки

Сучасний розвиток інформаційних технологій значною мірою зумовлений потребою в автоматизації процесів обробки даних, зокрема візуальної інформації. Зростання обсягів зображень та відеоданих, з якими щоденно взаємодіє людина, вимагає застосування швидких і точних інструментів аналізу. У зв'язку з цим комп'ютерний зір і машинне навчання стали одним із найбільш перспективних напрямів досліджень у галузі

штучного інтелекту, оскільки дозволяють автоматизувати складні задачі аналізу візуальних даних [1].

Однією з важливих задач комп'ютерного зору є розпізнавання опорних точок людини, під яким розуміють визначення координат ключових частин тіла, таких як очі, плечі, лікті, коліна та стопи. Отримані координати дають змогу побудувати спрощену модель скелета людини та використовувати її для аналізу пози, рухів і взаємодій. Методи визначення опорних точок знаходять широке застосування у різних прикладних сферах, зокрема у спорті, медицині, системах безпеки, розважальних технологіях та графічній обробці зображень [4].

А.3 Підстави для розробки

Підставою для виконання розробки є завдання на дипломний проєкт на тему “Дослідження та програмна реалізація методів машинного навчання для розпізнавання опорних точок людини”, затверджене наказом по НУ «Запорізька політехніка» № 447 від 30 вересня 2025 року.

А.4 Основні вимоги до програми, початковим даним і результату

А.4.1 Вимоги до функціональних характеристик

Вимоги до функціональних характеристик:

- завантаження зображень із пам'яті пристрою та робота з ними у багатошаровому середовищі;
- виконання автоматичного визначення опорних точок людини на зображеннях із використанням моделей машинного навчання;
- візуалізація та редагування скелетної моделі, отриманої після аналізу зображення;
- використання набору інструментів редагування: фільтри, корекція кольору, трансформації, обрізання, маски, кисті тощо;

- застосування додаткових ефектів і режимів змішування шарів;
- збереження відредагованих зображень у стандартних графічних форматах (JPEG, PNG тощо);
- експорт результатів у галерею пристрою або сторонні сервіси;
- можливість роботи з ML-моделями локально без підключення до мережі.

A.4.2 Вимоги до надійності

Програмний продукт має задовольняти такі вимоги:

- забезпечувати стабільну роботу при обробці великих графічних файлів;
- коректно обробляти помилки, пов'язані з відсутністю потрібних ресурсів, недостатністю пам'яті або невірним форматом вхідного зображення;
- забезпечувати гарантію цілісності даних під час збереження та експорту результатів;
- механізми машинного навчання мають функціонувати локально, без залежності від зовнішніх серверів, що підвищує надійність роботи в автономному режимі.

A.4.3 Умови експлуатації

Програмний продукт повинен мати можливість працювати як з інтернет-з'єднанням, так і в автономному режимі.

Інтерфейс має бути оптимізований для використання на сенсорних екранах мобільних пристроїв.

Час відгуку системи при виконанні ключових операцій (завантаження зображення, визначення опорних точок, застосування фільтрів) повинен відповідати вимогам комфортного використання.

A.4.4 Вимоги до складу та параметрів технічних засобів

Для функціонування програмного продукту необхідно мати мобільний пристрій під керуванням iOS 17 або вище.

Доступ до мережі інтернет не є обов'язковим і використовується лише для поширення готових зображень.

A.4.5 Вимоги до маркування та упакування

Програмне забезпечення не потребує додаткових дій при маркуванні та упакуванні.

A.4.6 Вимоги до транспортування та збереження

Програмний продукт зберігається у приватному репозиторії розробника.

Передавання здійснюється виключно в цифровій формі.

A.4.7 Вимоги до програмної документації

Програмна документація повинна містити такі основні розділи:

- керівництво оператора;
- керівництво користувача.

Керівництво оператора повинно містити інформацію щодо:

- технічних засобів та програмного середовища, використаних під час розробки програмного продукту;
- мови програмування та застосованих бібліотек;
- архітектурного підходу, обраного для побудови системи;
- структури застосунку та опису його основних модулів.

Керівництво користувача має містити покрокові інструкції з використання програмного продукту, включно з графічними прикладами інтерфейсу та демонстрацією основних можливостей редагування зображень.

A.4.8 Обмеження на установку і використання програмного продукту

Розроблений програмний продукт потребує інсталяції. Програмний продукт повинен використовуватись на мобільних пристроях під керівництвом iOS, які задовольняють вимогам та параметрам технічних засобів.

ДОДАТОК Б
Текст програми

```

import CoreData
import UIKit

class CoreDataManger {
    static let shared = CoreDataManger()

    private lazy var persistentContainer: NSPersistentContainer = {
        let container = NSPersistentContainer(name: "BFE26")
        container.loadPersistentStores(completionHandler: {
            (storeDescription, error) in
                if let error = error as NSError? {
                    fatalError("Unresolved error \(error), \(error.userInfo)")
                }
            })
        return container
    }()

    private var context: NSManagedObjectContext {
        persistentContainer.viewContext }

    func saveContext () {
        if context.hasChanges {
            do {
                try context.save()
            } catch {
                let nserror = error as NSError
                fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
            }
        }
    }
}

```

```

func create(_ imageData: Data?) {
    let model = Image(context: context)
    model.imageData = imageData
    model.identifier = UUID()
    model.dateModified = Date()
    do {
        try context.save()
    } catch {
        print(error.localizedDescription)
    }
}

func update(_ model: Image, imageData: Data?) {
    model.dateModified = Date()
    model.imageData = imageData
    do {
        try context.save()
    } catch {
        print(error.localizedDescription)
    }
}

func getProjects() -> [Image] {
    do {
        return try context.fetch(Image.fetchRequest())
    } catch {
        print(error.localizedDescription)
        return []
    }
}

```

```
}
```

```
func delete(_ model: Image) {
    do {
        context.delete(model)

        try context.save()
    } catch {
        print(error.localizedDescription)
    }
}
```

```
func deleteAll() {
    do {
        let items = try context.fetch(Image.fetchRequest())
        items.forEach { context.delete($0) }
        try context.save()
    } catch {
        print(error.localizedDescription)
    }
}
```

```
func save(_ imageData: Data?, item: Image?) {
    if let item = item { update(item, imageData: imageData) }
    else { create(imageData) }
}
}
```

```
import UIKit
import Vision
```

```

class EditManager {
    private struct EditingState {
        var image: UIImage
        var drawingMask: UIImage?

        static var none: Self { .init(image: UIImage(), drawingMask:
UIColor.white.image()) }
    }
    private var image: UIImage
    var scrollView = UIScrollView()

    private let meshView = BCMeshTransformView()
    private let imageView = UIImageView()
    private var toolView = UIView()

    private var obs = [VNHumanBodyPoseObservation]()
    private var context: CGContext?

    private var currentState = EditingState.none
    private var undoStates = [EditingState]()
    private var redoStates = [EditingState]()
    var undoRedoEvent: EventHandler2<Bool, Bool>?

    var currentTool: Tool = .none {
        didSet {
            switch currentTool {
            case .none, .eraser:
                applyChanges(nil)
                meshView.isHidden = true

```

```

        default: meshView.isHidden = false
    }
}
}

var drawingEvent: EventHandler<UIImage?>?

init(_ imageData: Data?) {
    image = UIImage()
    if let imageData = imageData, let image = UIImage(data: imageData)
{
        self.image = image
    }
    imageView.contentMode = .scaleAspectFit
    imageView.image = image
    drawingMask = UIColor.white.image()
    currentState = .init(image: image, drawingMask:
UIColor.white.image())
}

func didLoad() {
    guard scrollView.viewWithTag(0xDEADBEEF) == nil else { return }
    meshView.tag = 0xDEADBEEF
    meshView.contentView.addSubview(imageView)
    scrollView.addSubview(meshView)
    [meshView, imageView].forEach { $0.fillSuperview() }
    meshView.isHidden = true

    recognizeImage(image: image)
    undoRedoEvent?(undoStates.isEmpty, redoStates.isEmpty)
}

```

```

func setHiddenTool(_ isHidden: Bool) {
    UIView.transition(with: toolView, duration: 0.15, options:
.curveEaseIn) { [unowned self] in
        toolView.alpha = isHidden ? 0.0 : 1.0
    }
}

```

```

func setComparison(_ isHidden: Bool) {
    setHiddenTool(isHidden)
    UIView.transition(with: meshView, duration: 0.15, options:
.curveEaseIn) { [unowned self] in
        meshView.alpha = isHidden ? 0.0 : 1.0
    }
}

```

```

func performPumpTransform(value: Float, view: ToolControl) {
    guard currentTool == .pumpUp else { return }
    toolView = view

    //    meshView.isHidden = false

```

```

    let converted = view.usableFrame
    let x = converted.midX / scrollView.frame.width
    let y = converted.midY / scrollView.frame.height
    let center = CGPoint(x: x * image.size.width,
        y: y * image.size.height)

```

```

    meshView.meshTransform = BCMutableMeshTransform
        .circularDistortionTransform(centralPoint: center,

```

```

        radius: converted.width / 2.0,
        distortionStrength: -CGFloat(value) / 200,
        in: image.size)
    }

func performHeightTransform(value: Float, view: ToolControl) {
    guard currentTool == .height else { return }
    toolView = view

//    meshView.isHidden = false

    let converted = view.convert(view.usableFrame, to: scrollView)

    let upperLimit = converted.minY / scrollView.frame.height
    let lowerLimit = converted.maxY / scrollView.frame.height

    meshView.meshTransform = BCMutableMeshTransform
        .verticalStretchTransform(topY: upperLimit * image.size.height,
                                bottomY: lowerLimit * image.size.height,
                                stretchFactor: 1.0 + (CGFloat(value) / 1000),
                                in: image.size)
}

func positionOriginWaist(view: WaistResizeView) {
    guard currentTool == .waist else { return }
    toolView = view

    guard let bodyObservation = obs.first else { return }

    let confidenceThreshold: Float = 0.25

```

```

var waistPoints = [VNRecognizedPoint]()

let    waistJoints:    [(VNHumanBodyPoseObservation.JointName,
String)] = [
    (.leftHip, "Left Hip"),
    (.rightHip, "Right Hip"),
    (.root, "Root"),
    (.neck, "Neck")
]

func                                getPoints(for                                joints:
[(VNHumanBodyPoseObservation.JointName, String)]) -> [VNRecognizedPoint]
{
    var points = [VNRecognizedPoint]()
    for (joint, name) in joints {
        do {
            let point = try bodyObservation.recognizedPoint(joint)
            if point.confidence > confidenceThreshold {
                points.append(point)
            }
        } catch {
            print("Error getting \(name) point: \(error)")
        }
    }
    return points
}

waistPoints = getPoints(for: waistJoints)

guard waistPoints.count >= 3 else {

```

```

        print("Insufficient waist points found with sufficient confidence")
        return
    }

    let convertedPoints = convertToImagePoints(waistPoints, image:
image)

    let waistImagePoints = convertedPoints.map { $0.point }

    guard let waistCenter = calculateCenter(of: waistImagePoints) else {
        print("Unable to calculate waist center point")
        return
    }

    view setFrame(using: convertedPoints)
    view.center = waistCenter
}

func performWaistTransform(value: Float, view: ToolControl) {
    guard currentTool == .waist else { return }
    toolView = view

    //    meshView.isHidden = false

    let center = view.center
    let frame = view.usableFrame
    var points = [CGPoint(x: frame.minX, y: frame.midY),
        .init(x: frame.maxX, y: frame.midY),
        .init(x: frame.midX, y: frame.minY),
        .init(x: frame.minX, y: frame.maxY)]
    points = points.map {

```

```

        .init(x: $0.x / scrollView.frame.width * image.size.width,
              y: $0.y / scrollView.frame.height * image.size.height)
    }

```

```

let normalizedSliderValue = CGFloat(value) / 100.0

```

```

let maxScaling: CGFloat = 0.3

```

```

let horizontalScaling = 1.0 + (normalizedSliderValue * maxScaling)

```

```

let scaleTransform = CGAffineTransform(scaleX: horizontalScaling,
y: 1.0)

```

```

points = points.map { point in
    let centered = CGPoint(x: point.x - center.x, y: point.y - center.y)
    let scaled = centered.applying(scaleTransform)
    return CGPoint(x: scaled.x + center.x, y: scaled.y + center.y)
}

```

```

let warpIntensity = normalizedSliderValue * 0.5

```

```

meshView.meshTransform = BCMutableMeshTransform
    .warpTransform(points: points,
                  centralPoint: center,
                  value: warpIntensity,
                  in: image.size,
                  maxDistance: 1.0)
}

```

```

func performHipsTransform(value: Float) {
    guard currentTool == .hips else { return }
    // meshView.isHidden = false
}

```

```

guard let bodyObservation = obs.first else { return }

let confidenceThreshold: Float = 0.25
var leftArmPoints = [VNRecognizedPoint]()
var rightArmPoints = [VNRecognizedPoint]()

let leftArmJoints: [(VNHumanBodyPoseObservation.JointName,
String)] = [
    (.leftHip, "Left Hip"),
    (.leftKnee, "Left Knee"),
    (.root, "Root")
]

let rightArmJoints: [(VNHumanBodyPoseObservation.JointName,
String)] = [
    (.rightHip, "Right Hip"),
    (.rightKnee, "Right Knee"),
    (.root, "Root")
]

func getPoints(for joints:
[(VNHumanBodyPoseObservation.JointName, String)]) -> [VNRecognizedPoint]
{
    var points = [VNRecognizedPoint]()
    for (joint, name) in joints {
        do {
            let point = try bodyObservation.recognizedPoint(joint)
            if point.confidence > confidenceThreshold {
                points.append(point)
            }
        }
    }
}

```

```

    } catch {
        print("Error getting \$(name) point: \$(error)")
    }
}
return points
}

```

```

leftArmPoints = getPoints(for: leftArmJoints)
rightArmPoints = getPoints(for: rightArmJoints)

```

```

guard !leftArmPoints.isEmpty && !rightArmPoints.isEmpty else {
    print("Insufficient arm points found with sufficient confidence")
    return
}

```

```

let leftArmImagePoints = convertToImagePoints(leftArmPoints,
image: image).map { $0.point }
let rightArmImagePoints = convertToImagePoints(rightArmPoints,
image: image).map { $0.point }

```

```

guard let leftArmCenter = calculateCenter(of: leftArmImagePoints),
    let rightArmCenter = calculateCenter(of: rightArmImagePoints)
else {
    print("Unable to calculate arm center points")
    return
}

```

```

let warpIntensity = CGFloat(value) / 500.0

```

```

meshView.meshTransform = BCMutableMeshTransform

```

```

        .warpTransform(leftElementPoints: leftArmImagePoints,
                        leftElementCenter: leftArmCenter,
                        rightElementPoints: rightArmImagePoints,
                        rightElementCenter: rightArmCenter,
                        value: warpIntensity,
                        in: image.size)
    }

    var startPoint = CGPoint.zero
    var drawingMask: UIImage?
    func onTouchesBegan(_ point: CGPoint = .zero) {
        if currentTool == .eraser {
            UIGraphicsBeginImageContext(imageView.frame.size)
            context = UIGraphicsGetCurrentContext()
            drawingMask?.draw(in:          .init(origin:          .zero,          size:
imageView.frame.size))
        } else if currentTool == .reshape {
            //      meshView.isHidden = false
            startPoint = point
        }

    }

    func performTouch(from lhs: CGPoint, to rhs: CGPoint, value: Float) {
        if currentTool == .eraser {
            context?.move(to: lhs)
            context?.addLine(to: rhs)
            context?.setLineCap(.round)
            context?.setLineWidth(CGFloat(value))
            context?.setStrokeColor(UIColor.black.cgColor)
        }
    }

```

```

context?.setBlendMode(.normal)
context?.strokePath()
let image = UIGraphicsGetImageFromCurrentImageContext()

drawingMask = image?.resized(to: self.image.size)

let filter = CIFilter(name: "CIBlendWithMask")
guard let ciImage = CIImage(image: self.image),
    let mask = drawingMask,
    let ciMask = CIImage(image: mask) else { return }

filter?.setValue(ciImage, forKey: kCIInputImageKey)
filter?.setValue(ciMask, forKey: kCIInputMaskImageKey)
guard let outputImage = filter?.outputImage,
    let cgImage = CIContext().createCGImage(outputImage, from:
ciImage.extent)
    else { return }
    let final: UIImage = .init(cgImage:
cgImage).withOrientation(self.image.imageOrientation).resized(to:
self.image.size)
    drawingEvent?(final)
    currentState.image = final
    currentState.drawingMask = drawingMask
} else if currentTool == .reshape {
    meshView.meshTransform = BCMutableMeshTransform
        .warpTransform(from: startPoint,
            to: rhs,
            in: scrollView.bounds.size,
            strength: CGFloat(value) / 125)
}

```

```
}
```

```
func onTouchesEnded(_ completion: EventHandler<Data?>?) {
    if currentTool == .eraser {
        UIGraphicsEndImageContext()
        redoStates = []
        return
    }
}
```

```
guard let glkView = meshView.getGlkView(),
    let image = UIImage.fromGLKView(glkView: glkView) else {
return }

self.image = image
imageView.image = image
currentState.image = image
completion?(image.pngData())
}
```

```
func cancelChangeAndClose() {
    performUndo()

    drawingMask = UIColor.white.image()
    toolView.removeFromSuperview()
    meshView.isHidden = true
    meshView.meshTransform = BCMutableMeshTransform
        .warpTransform(from: .zero,
                       to: .zero,
                       in: image.size,
                       strength: 0)
}
```

```

func applyChanges(_ completion: EventHandler<Data?>?) {
    guard let glkView = meshView.getGlkView(),
        let image = UIImage.fromGLKView(glkView: glkView) else {
return }

    self.image = image
    imageView.image = image
    completion?(image.pngData())

    drawingMask = UIColor.white.image()
    toolView.removeFromSuperview()
    meshView.isHidden = true
    meshView.meshTransform = BCMutableMeshTransform
        .warpTransform(from: .zero,
                        to: .zero,
                        in: image.size,
                        strength: 0)
    guard completion != nil else { return }
    redoStates = []
    undoStates.append(currentState)
    currentState.image = image
    currentState.drawingMask = drawingMask
    undoRedoEvent?(undoStates.isEmpty, redoStates.isEmpty)
}

func performUndo() {
    replace(from: &undoStates, to: &redoStates)
    undoRedoEvent?(undoStates.isEmpty, redoStates.isEmpty)
}

```

```

func performRedo() {
    replace(from: &redoStates, to: &undoStates)
    undoRedoEvent?(undoStates.isEmpty, redoStates.isEmpty)
}

typealias ImagePoint = (string: String, point: CGPoint)

private extension EditManager {
    func recognizeImage(image: UIImage) {
        guard let cgImage = image.cgImage else { return }
        let orientation = CGImagePropertyOrientation(rawValue:
UInt32(image.imageOrientation.rawValue))!
        let handler = VNImageRequestHandler(cgImage: cgImage,
orientation: orientation, options: [:])
        let bodyRequest =
VNDetectHumanBodyPoseRequest(completionHandler: handleBodyRecognize)

        DispatchQueue.main.async {
            try? handler.perform([bodyRequest])
        }
    }

    func handleBodyRecognize(request: VNRequest, error: Error?) {
        guard let results = request.results as?
[VNHumanBodyPoseObservation] else {
            print("Can't find a body in the picture")
            return
        }
        obs = results
        print("Found \(results.count) body(s)")
    }
}

```

```
}
```

```
func convertToImagePoints(_ points: [VNRecognizedPoint], image:
UIImage) -> [ImagePoint] {
    points.map { point in
        (point.identifier.rawValue, CGPoint(x: point.location.x *
image.size.width,
        y: (1 - point.location.y) * image.size.height))
    }
}
```

```
func calculateCenter(of points: [CGPoint]) -> CGPoint? {
    guard !points.isEmpty else {
        return nil
    }
}
```

```
let sum = points.reduce(into: (x: 0.0, y: 0.0)) { result, point in
    result.x += point.x
    result.y += point.y
}
```

```
let count = Double(points.count)
let centerX = sum.x / count
let centerY = sum.y / count
```

```
return CGPoint(x: centerX, y: centerY)
}
```

```
func recognizeddMesh(_ bodyObservation:
VNHumanBodyPoseObservation, value: Float, view: WaistResizeView) {
```

```

let confidenceThreshold: Float = 0.25
var waistPoints = [VNRecognizedPoint]()

let    waistJoints:    [(VNHumanBodyPoseObservation.JointName,
String)] = [
    (.leftHip, "Left Hip"),
    (.rightHip, "Right Hip"),
    (.root, "Root"),
    (.neck, "Neck")
]

func                                getPoints(for                                joints:
[(VNHumanBodyPoseObservation.JointName, String)]) -> [VNRecognizedPoint]
{
    var points = [VNRecognizedPoint]()
    for (joint, name) in joints {
        do {
            let point = try bodyObservation.recognizedPoint(joint)
            if point.confidence > confidenceThreshold {
                points.append(point)
            }
        } catch {
            print("Error getting \(name) point: \(error)")
        }
    }
    return points
}

waistPoints = getPoints(for: waistJoints)

```

```

guard waistPoints.count >= 3 else {
    print("Insufficient waist points found with sufficient confidence")
    return
}

let convertedPoints = convertToImagePoints(waistPoints, image:
image)

var waistImagePoints = convertedPoints.map { $0.point }

guard let waistCenter = calculateCenter(of: waistImagePoints) else {
    print("Unable to calculate waist center point")
    return
}

view setFrame(using: convertedPoints)
view.center = waistCenter

let normalizedSliderValue = CGFloat(value) / 100.0

let maxScaling: CGFloat = 0.3
let horizontalScaling = 1.0 + (normalizedSliderValue * maxScaling)

let scaleTransform = CGAffineTransform(scaleX: horizontalScaling,
y: 1.0)

waistImagePoints = waistImagePoints.map { point in
    let centered = CGPoint(x: point.x - waistCenter.x, y: point.y -
waistCenter.y)
    let scaled = centered.applying(scaleTransform)
    return CGPoint(x: scaled.x + waistCenter.x, y: scaled.y +
waistCenter.y)

```

```
}
```

```
let warpIntensity = normalizedSliderValue * 0.5
```

```
meshView.meshTransform = BCMutableMeshTransform
    .warpTransform(points: waistImagePoints,
                   centralPoint: waistCenter,
                   value: warpIntensity,
                   in: image.size,
                   maxDistance: 1.0)
}
```

```
private func replace(from lhs: inout [EditingState], to rhs: inout
[EditingState]) {
    let last = lhs.removeLast()
    rhs.append(currentState)
    currentState = last
    drawingEvent?(currentState.image)
}
}
```

```
extension UIColor {
    func image(_ size: CGSize = CGSize(width: 1, height: 1)) -> UIImage {
        return UIGraphicsImageRenderer(size: size).image { rendererContext
in
            self.setFill()
            rendererContext.fill(CGRect(origin: .zero, size: size))
        }
    }
}
```

```

import UIKit
import Photos

class ImageManager: NSObject {
    enum PickerType {
        case camera
        case gallery(PHAccessLevel)

        var rawValue: String {
            switch self {
                case .camera: "camera"
                case .gallery: "photos"
            }
        }

        var icon: String {
            switch self {
                case .camera: "im_camera"
                case .gallery: "im_photo"
            }
        }
    }

    enum Event {
        case onImageAction(Data)
        case onError
    }

    enum SaveEvent {
        case onSaved
        case onError
    }
}

```

```
}
```

```
var eventHandler: EventHandler<Event>?
```

```
var savingHandler: EventHandler<SaveEvent>?
```

```
func configurePicker(for type: PickerType) -> UIImagePickerController
```

```
{
```

```
    let picker = UIImagePickerController()
```

```
    switch type {
```

```
        case .camera: picker.sourceType = .camera
```

```
        default: break
```

```
    }
```

```
    picker.delegate = self
```

```
    picker.modalPresentationStyle = .overFullScreen
```

```
    return picker
```

```
}
```

```
@discardableResult
```

```
func saveImage(_ imageData: Data) -> Self {
```

```
    guard let image = UIImage(data: imageData) else { return self }
```

```
    UIImageWriteToSavedPhotosAlbum(image, self,
```

```
#selector(onImageSaved), nil)
```

```
    return self
```

```
}
```

```
func combine(backgroundImage: UIImage?, editedImage: UIImage?) ->
```

```
UIImage? {
```

```
    guard let backgroundImage = backgroundImage,
```

```
        let editedImage = editedImage else { return nil }
```

```
    UIGraphicsBeginImageContext(backgroundImage.size)
```

```

let rect = CGRect(origin: .zero, size: backgroundImage.size)
backgroundImage.draw(in: rect)
editedImage.draw(in: rect)
let image = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()
return image
}

```

```

func formShare(_ image: UIImage, completion:
EventHandler<UIActivityViewController>?, errorHandler: VoidClosure?) {
    let temporaryDirectoryURL =
FileManager.default.temporaryDirectory
    let fileName = "image.jpg"
    let fileURL =
temporaryDirectoryURL.appendingPathComponent(fileName)

    do {
        if let imageData = image.jpegData(compressionQuality: 1.0) {
            try imageData.write(to: fileURL)

            let itemProvider = NSItemProvider(contentsOf: fileURL)

            itemProvider?.suggestedName = fileName

            let activity = UIActivityViewController(activityItems:
[itemProvider as Any],
applicationActivities: nil)

            completion?(activity)
        }
    } catch {

```

```

        print(error.localizedDescription)
        errorHandler?()
    }
}

protocol SharePresentable { }
extension SharePresentable where Self: UIViewController {
    func share(_ imageData: Data?, successHandler: VoidClosure? = nil,
errorHandler: VoidClosure? = nil) {
        let temporaryDirectoryURL =
FileManager.default.temporaryDirectory
        let fileName = "image.jpg"
        let fileURL =
temporaryDirectoryURL.appendingPathComponent(fileName)

        do {
            if let imageData = imageData {
                try imageData.write(to: fileURL)

                let itemProvider = NSItemProvider(contentsOf: fileURL)

                itemProvider?.suggestedName = fileName

                let activity = UIActivityViewController(activityItems:
[itemProvider as Any],
applicationActivities: nil)
                activity.completionWithItemsHandler = { _, completed, _, error
in
                    if error != nil {

```

```

        errorHandler?()
        return
    }
    WindowManager.shared.requestRating()
    successHandler?()
}
present(activity, animated: true)
}
} catch {
    print(error.localizedDescription)
    errorHandler?()
}
}
}

```

```

protocol SaveHandler { }
extension SaveHandler {
    var imageManager: ImageManager { .init() }
    func save(_ imageData: Data?, eventHandler:
EventHandler<ImageManager.SaveEvent>? = nil) {
        guard let imageData = imageData else { return }
        imageManager.saveImage(imageData).savingHandler = eventHandler
    }
}

```

```

extension ImageManager {
    @objc func onImageSaved(_ image: UIImage,
didFinishSavingWithError error: Error?, contextInfo: UnsafeRawPointer) {
        #warning("edit: @objc func onImageSaved(_ image: UIImage,
didFinishSavingWithError error: Error?, contextInfo: UnsafeRawPointer)")
    }
}

```

```

        if error != nil {
            savingHandler?(.onError)
            return
        }
        WindowManager.shared.requestRating()
        savingHandler?(.onSaved)
    }
}

extension ImageManager: UIImagePickerControllerDelegate,
 UINavigationControllerDelegate {
    func imagePickerControllerDidCancel(_ picker:
 UIImagePickerController) {
        #warning("edit: imagePickerControllerDidCancel(_ picker:
 UIImagePickerController)")
        picker.dismiss(animated: true)
    }

    func imagePickerController(_ picker: UIImagePickerController,
                                didFinishPickingMediaWithInfo info:
 [UIImagePickerController.InfoKey : Any]) {
        #warning("""
        edit:
        imagePickerController(_ picker: UIImagePickerController,
                                didFinishPickingMediaWithInfo info:
 [UIImagePickerController.InfoKey : Any])
        """)
        guard let image = info[.originalImage] as? UIImage, let data =
image.jpegData(compressionQuality: 0.5) else { return }
        picker.dismiss(animated: true)
    }
}

```

```

        eventHandler?(.onImageAction(data))
    }
}

class Filter {
    fileprivate var ciImage: CIImage?
    fileprivate var filter: CIFilter?

    fileprivate var type: EffectType
    fileprivate var key: String {
        switch type {
        case .zoom: kCIInputAmountKey
        case .pixel, .glass, .triangle: kCIInputScaleKey
        default: kCIInputRadiusKey
        }
    }
}

var outputImage: CIImage? { filter?.outputImage }

init(ciImage: CIImage?, type: EffectType) {
    self.ciImage = ciImage
    self.type = type

    filter = CIFilter(name: type.blurName)
    setupFilter()
}

func setupFilter() {
    filter?.setValue(ciImage, forKey: kCIInputImageKey)
    setValue(50.0)
}

```

```

    func setValue(_ value: Float) {
        filter?.setValue(type == .glass ? value * 10.0 : value, forKey: key)
    }
}

class MotionFilter: Filter {
    override func setupFilter() {
        super.setupFilter()
        filter?.setValue(type == .motionV ? Float.pi * 1.5 : 0.0, forKey:
kCIInputAngleKey)
    }
}

class CenteredFilter: Filter {
    override func setupFilter() {
        super.setupFilter()
        guard let extent = ciImage?.extent else { return }
        filter?.setValue(CIVector(x: extent.midX, y: extent.midY), forKey:
kCIInputCenterKey)
    }
}

//MARK: - Non nullable
class NonnullableFilter: CenteredFilter {
    fileprivate var value: Float = 0.0

    override var outputImage: CIImage? {
        value == 0.0 ? ciImage : filter?.outputImage
    }

    override func setupFilter() {

```

```
        super.setupFilter()
    }

    override fun setValue(_ value: Float) {
        super.setValue(value)
        self.value = value
    }
}

class GlassFilter: CenteredFilter {
    override fun setupFilter() {
        super.setupFilter()
        guard let img = UIImage(named: "im_glassPattern") else { return }
        filter?.setValue(CIImage(image: img), forKey: "inputTexture")
    }
}
```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»

Кафедра програмних засобів

Дослідження та програмна реалізація методів машинного навчання для розпізнавання опорних точок людини

Виконав:
ст. групи КНТ-124м
Керівник:
д.т.н., професор

Максим БАЛУХТІН

Андрій ОЛІЙНИК

Рисунок В.1 – Слайд № 1

Мета, об'єкт, предмет дослідження

- **Об'єкт дослідження** – процес проектування та розробки програмних засобів, що використовують методи машинного навчання.
- **Предмет дослідження** – програмні засоби для визначення опорних точок людини.
- **Мета роботи** – дослідження та реалізація програмного забезпечення для розпізнавання опорних точок людини для зменшення трудомісткості та часу обробки зображень людини.

2

Рисунок В.1 – Слайд № 2

Огляд предметної області

Характеристика	Серверні алгоритми	Готові бібліотеки	Vision+CoreML
Переваги	Висока точність Масштабованість	Висока якість визначення точок	Швидкість Приватність Енергоефективність
Недопiки	Потреба стабільного інтернет-з'єднання Низька конфіденційність	Складність в адаптації	Обмеження складності моделей

3

Рисунок В.1 – Слайд № 3

Вибір мови програмування

Характеристика	Objective-C	Swift	SwiftUI
Дизайн	Надбудова над C	Сучасна мова Apple	Декларативний UI-фреймворк від Apple
Парадигма	Відправлення та обробка повідомлень	Парадигма	Декларативна
Кількість файлів для одного класу	Два (.h + .m)	Один	Один
Сумісність з ML	Обмежена	Повна	Повна
Читабельність коду	Низька	Висока	Висока
Особливості	Застаріла модель пам'яті, складність підтримки	Підтримка сучасних API	Швидке створення UI

4

Рисунок В.1 – Слайд № 4

Вибір середовища розробки

Характеристика	XCode	Visual Studio	VS Code
Доступність	Безкоштовно	Умовно безкоштовно	Безкоштовно
Підтримка Apple SDK	Повна	Часткова	Обмежена
Підтримка ML	Вбудований інструментарій	Інтеграція через .NET	Залежить від плагінів
Інструменти розробки	Редактор інтерфейсів Симулятор Автопідстановка Дебагер Тестування Контроль версій	Компілятор Дебагер Редактор коду Тестування	Редактор коду Термінал
Компілятор	Власний LLVM	Roslyn	Зовнішні
Оновлюваність	Разом з Swift та iOS	Часті	Залежно від плагінів

5

Рисунок В.1 – Слайд № 5

Наукова новизна роботи



6

Рисунок В.1 – Слайд № 6

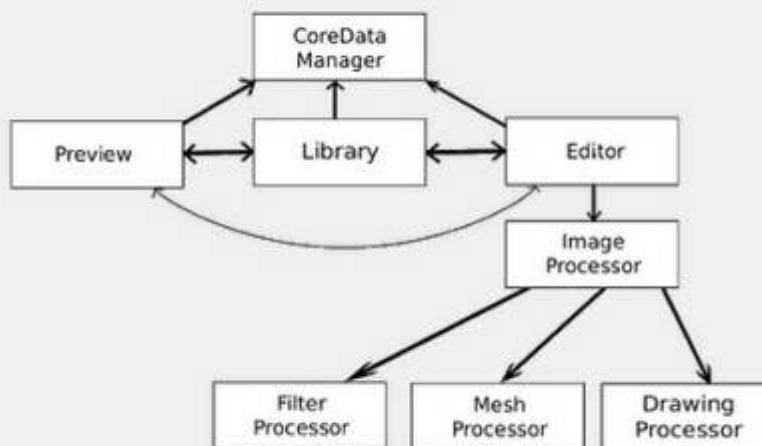
Схема структури програми



7

Рисунок В.1 – Слайд № 7

Схема взаємодії модулів програми



8

Рисунок В.1 – Слайд № 8

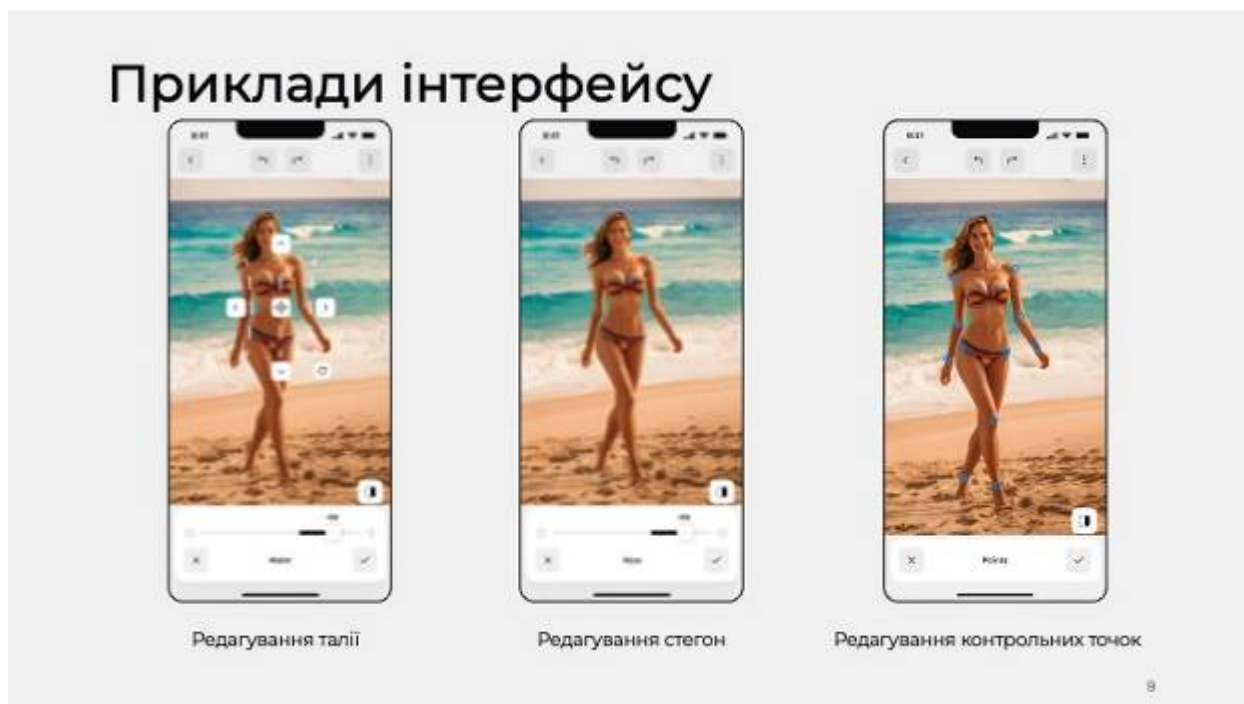


Рисунок В.1 – Слайд № 9

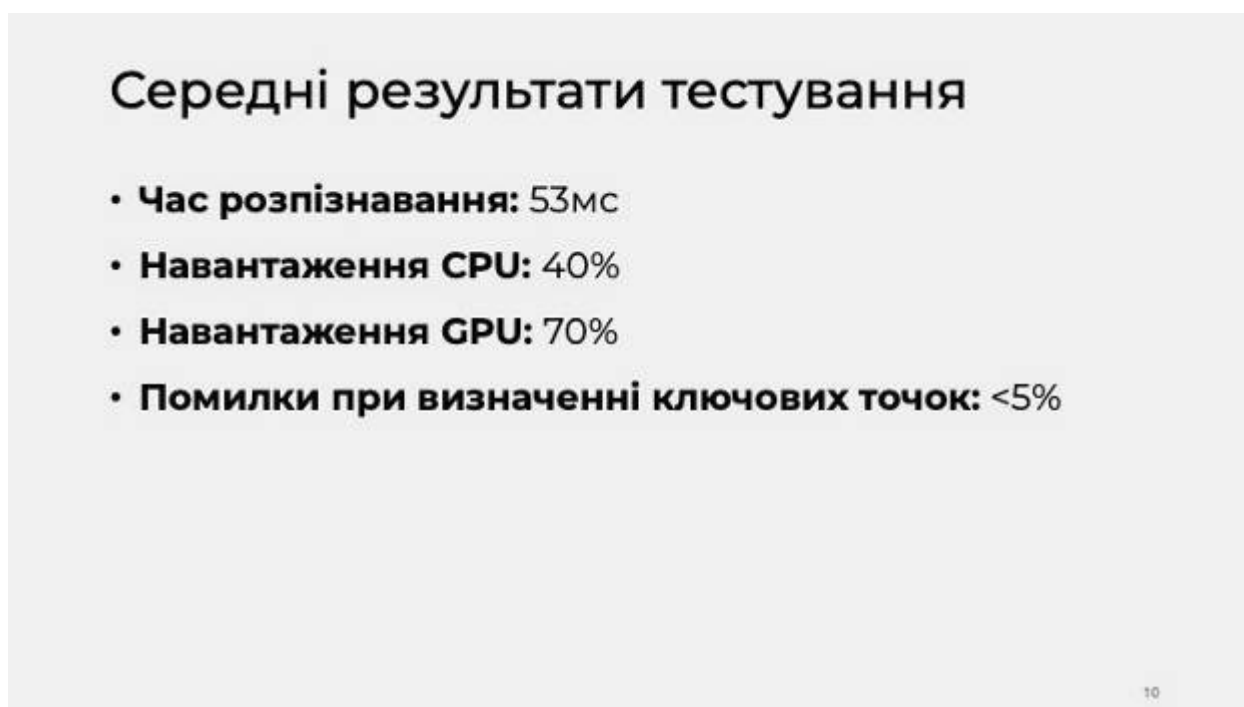


Рисунок В.1 – Слайд № 10

Висновки

У ході роботи досліджено методи машинного навчання для визначення опорних точок людини та реалізовано їх у мобільному графічному редакторі під iOS.

Отримані результати дозволили:

- визначити доцільність використання нативної розробки та інструментів Apple;
- створити архітектуру програмного продукту та реалізувати метод локальної обробки зображень з використанням оптимізованої ML-моделі;
- продемонструвати ефективність локальної обробки.

Рисунок В.1 – Слайд № 11