Форма № 24

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)
магістр
(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ВИЯВЛЕННЯ МЕЛАНОМИ ЗА ЗОБРАЖЕННЯМ ШКІРИ
RESEARCH AND SOFTWARE IMPLEMENTATION OF A MOBILE APPLICATION FOR DETECTING MELANOMA FROM SKIN IMAGES

Виконав(ла): студент(ка) 2 курсу, групи КНТ-214м
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

ЙОЖИКОВ А.А.
(ПРІЗВИЩЕ та ініціали)
Керівник ФЕДОРОНЧАК Т.В.
(ПРІЗВИЩЕ та ініціали)
Рецензент БАБЕНКО Н.В.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

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

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(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)

ЙОЖИКОВА Анатолія Андрійовича
(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація мобільного застосунку для виявлення меланоми за зображенням шкіри. Research and Software Implementation of a Mobile Application for Detecting Melanoma from Skin Images
керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові)
затверджені наказом закладу вищої освіти від “ 30 ” вересня 2025 року № 447
2. Строк подання студентом проєкту (роботи) 01 грудня 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. Розробка архітектури програми. 4. Основні рішення щодо реалізації компонентів системи. 5. Експлуатація, тестування та експериментальне дослідження програми.
5. Перелік графічного матеріалу (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
1-5 Основна частина	ФЕДОРОНЧАК Т.В., доцент		
Нормоконтроль	БЄЛОВА А.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	4-5 тижні	Розділ 2
	алгоритмів вирішення задачі.		
4	Вибір мови програмування та інших технологій	6 тиждень	Розділ 3
	розробки.		
5	Розробка архітектури програми.	6 тиждень	Розділ 3
6	Розробка програми.	7-8 тижні	Розділ 4
7	Тестування та експериментальне дослідження	9 тиждень	Розділ 5
	програмного забезпечення.		
8	Оформлення пояснювальної записки та документів	10-11 тижні	Додатки
	до неї.		
9	Нормоконтроль та рецензування.	12 тиждень	
10	Захист роботи.	12 тиждень	

Студент(ка) _____ Анатолій ЙОЖИКОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Тетяна ФЕДОРОНЧАК
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 93 с., 11 табл., 48 рис., 3 дод., 21 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, МЕЛАНОМА, FLUTTER, DART, PYTHON, MOBILENET.

Об'єкт дослідження – автоматизована діагностика меланоми на основі аналізу зображень шкіри за допомогою методів штучного інтелекту.

Предмет дослідження – мобільні застосунки для визначення та аналізу зображення шкіри з метою виявлення меланоми.

Мета роботи – створення мобільного застосунку для автоматизованого аналізу зображень шкіри, що дозволить виявляти меланому.

Матеріали, методи та технічні засоби: мови програмування Dart та Python, платформи Flutter та Flask, бібліотека TensorFlow, сервіс Firestore, шаблон проєктування Bloc, персональний комп’ютер з процесором Intel Core i5 під управлінням операційної системи Windows 10.

Результати. Створено мобільний застосунок, який дозволяє за зображеннями шкіри проводити автоматичний аналіз з метою виявлення ознак меланоми.

Висновки. Розроблено мобільний застосунок для автоматичного виявлення меланоми за зображенням шкіри, що може сприяти ранньому виявленню захворювання та підвищити ефективність первинного самоконтролю.

Галузь використання – медичними установами для швидкої попередньої перевірки на наявність ознак меланоми на основі аналізу зображень шкіри пацієнтів. Користувачами для самоконтролю стану шкіри в домашніх умовах.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 93 pages, 11 tables, 48 figures, 3 appendixes, 21 sources.

MOBILE APPLICATION, MELANOMA, FLUTTER, DART, PYTHON, MOBILENET.

The object of the study is automated melanoma diagnostics based on skin image analysis using artificial intelligence methods.

The subject of the study is mobile applications for determining and analyzing skin images to detect melanoma.

The purpose of the work is to create a mobile application for automated analysis of skin images, which will allow detecting melanoma.

Materials, methods and technical means: Dart and Python programming languages, Flutter and Flask platforms, TensorFlow library, Firestore service, Bloc design template, personal computer with an Intel Core i5 processor running the Windows 10 operating system.

Results. A mobile application has been created that allows for automatic analysis of skin images to detect signs of melanoma.

Conclusions. A mobile application has been developed for automatic detection of melanoma from skin images, which can contribute to early detection of the disease and increase the effectiveness of primary self-monitoring.

Field of use – by medical institutions for rapid preliminary screening for signs of melanoma based on analysis of patient skin images. By users for self-monitoring of skin condition at home.

ЗМІСТ

С.
Перелік скорочень та умовних познак 9
Вступ 10
1 Аналіз проблеми та постановка завдань роботи 11
1.1 Аналіз предметної області 11
1.2 Огляд методів штучного інтелекту 12
1.2.1 SVM 12
1.2.2 k-NN 13
1.2.3 CNN 13
1.3 Огляд існуючих аналогів 15
1.3.1 SkinVision 15
1.3.2 SkinScreener 16
1.3.3 SkiniveMD 17
1.4 Висновки до розділу 18
2 Матеріали і методи 19
2.1 Огляд архітектур згорткових нейронних мереж 19
2.1.1 VGG16 19
2.1.2 ResNet50 19
2.1.3 MobileNet 20
2.2 Модифікація моделі 21
2.3 Аналіз датасету для навчання та тестування моделі 22
2.4 Висновки до розділу 23
3 Розробка архітектури програми 24
3.1 Вибір мови програмування та програмні засоби для створення мобільного застосунку 24
3.2 Вибір технологій для розробки серверної частини та нейромереж 25
3.2.1 Вибір мови програмування 25
3.2.2 Вибір інструменту для створення серверної частини 26
3.2.3 Вибір бібліотеки для побудови нейромережі 26
3.3 Формування функціональних вимог до програмного продукту 27

3.4 Обґрунтування вибору бази даних та сервісу для побудови 28
3.5 Архітектура розроблювального програмного продукту 30
3.6 Висновки до розділу 32
4 Основні рішення щодо реалізації компонентів системи 33
4.1 Схема функціонування програмного продукту 33
4.2 Структура бази даних 34
4.3 Опис організації серверної частини та моделі нейромережі 35
4.4 Опис організації програмного продукту 37
4.4.1 Модуль data та його класи 37
4.4.2 Модуль view та його класи 41
4.4.3 Модуль core та його класи 43
4.4.4 Модуль bloc та його класи 44
4.5 Опис алгоритмів роботи програмного продукту 51
4.5.1 Алгоритм вибору зображення з галереї 51
4.5.2 Алгоритм створення зображення за допомогою камери 52
4.5.3 Алгоритм виконання аналізу зображення шкіри для виявлення меланоми 52
4.5.4 Алгоритм перевірки якості моделі 52
4.5.5 Алгоритм попередньої обробки даних 54
4.5.6 Алгоритм формування моделі 54
4.5.7 Алгоритм навчання та збереження моделі 55
4.6 Висновки до розділу 56
5 Експлуатація, тестування та експериментальне дослідження програми 57
5.1 Призначення програмного продукту та вимоги до виконання 57
5.2 Експлуатація програмного продукту 57
5.3 Проведення тестування програмного продукту та аналіз отриманих результатів 65
5.4 Висновки до розділу 66
Висновки 67
Перелік джерел посилання 68
Додаток А Технічне завдання 71
Додаток Б Текст програми 74

Додаток В Слайди презентації 84

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

Bloc – Business logic component;

CNN – Convolutional Neural Network;

JSON – JavaScript Object Notation;

k-NN – k-Nearest Neighbors;

ML – Machine Learning;

NoSQL – Not only SQL;

PDF – Portable Document Format;

RAM – Random Access Memory;

ReLU – Rectified Linear Unit;

RGB – Red, Green, Blue;

SVM – Support Vector Machine;

VGG16 – Visual Geometry Group 16;

БД – База даних;

Гб – Гігабайт;

Мб – Мегабайт;

ПІБ – Прізвище, Ім’я, По батькові.

ВСТУП

Сьогодні меланома є одним із найсерйозніших видів раку шкіри, і її небезпека виходить далеко за межі поверхні шкіри. Хоча меланома може починатися як маленька, здавалося б, нешкідлива родимка, вона може швидко стати небезпечною для життя, якщо її не виявити та не лікувати на ранній стадії [1].

Згідно з оцінками Американського онкологічного товариства щодо меланоми в США на 2025 рік, було діагностовано близько 104 960 нових випадків меланоми, де близько 60550 у чоловіків та 44410 у жінок. Також очікується, що від меланоми помре близько 8 430 людей, де 5470 чоловіків та 2960 жінок [2].

Тому у зв’язку з цим проблема раннього виявлення меланоми набуває критичної важливості як у медичній практиці, так і в сфері цифрових технологій. Застосування методів штучного інтелекту у медичній діагностиці дозволяє покращити точність виявлення цієї хвороби на основі зображень шкіри. Проте не кожен має змогу регулярно проходити огляд в медичних закладах, а самостійне розпізнавання цього виду раку шкіри є складним завданням для більшості людей.

Зважаючи на це, дана робота присвячена створенню мобільного застосунку, який надасть користувачам можливість оперативно отримати попередню оцінку наявності ознак меланоми, що сприятиме ранньому зверненню до фахівця та покращить перспективи успішного лікування.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ РОБОТИ

1.1 Аналіз предметної області

Пігментні утворення шкіри умовно поділяють на доброякісні та злоякісні. До доброякісних зазвичай відносяться звичайні родимки, які є безпечними та не становлять загрози життю пацієнта. До злоякісних належить меланома, яка при відсутності своєчасної діагностики та лікування може становити серйозну загрозу життю пацієнта.

Меланома – це рак, який зазвичай починається в клітинах шкіри, відомих як меланоцити, що виробляють пігмент шкіри. Порівняно з більш поширеними видами раку шкіри, які починаються в плоскоклітинному або базальному шарі, меланома частіше поширюється на інші частини тіла. Зазвичай більшість меланом виглядають як плоскі або злегка підняті плями темного кольору на шкірі, які часто піддавалися впливу ультрафіолетового випромінювання, наприклад, на шкірі голови та обличчя, руках, спині та ногах (хоча вони можуть виникати і на ділянках, які ніколи не піддавалися впливу сонця). У меншій кількості випадків новоутворення може виглядати як темний або червоний горбок і проростати в шкіру, що може ускладнити його виявлення [3].

Тому на сьогодні спостерігається активний розвиток програмних засобів для полегшення проведення діагностики стану шкіри та виявлення меланоми. Ці засоби зазвичай використовують сучасні методи штучного інтелекту для виявлення та класифікації підозрілих утворень шкіри, де класифікація найчастіше здійснюється у вигляді бінарної задачі – віднесенням зображення утворення шкіри до доброякісних або злоякісних. Особливої популярності набувають мобільні застосунки, які дають змогу кожному користувачеві легко зробити фото підозрілих родимок та отримати оцінку наявності ознак даної хвороби. Також дають змогу отримувати персоналізовані рекомендації, що дає змогу контролювати стан шкіри та вчасно звертатися до фахівця за потреби.

1.2 Огляд методів штучного інтелекту

Існує багато методів штучного інтелекту, які застосовуються для класифікації зображень, серед яких найчастіше застосовують метод опорних векторів (SVM), метод найближчих сусідів (k-NN) та згорткові нейронні мережі (CNN).

1.2.1 SVM

Метод опорних векторів (SVM) – це метод штучного інтелекту, який використовується для задач класифікації [4].

У задачах класифікації зображень вхідні дані подаються у вигляді матриць піксельних значень, проте для ефективної класифікації за допомогою цього методу необхідно попереднє визначення інформативних ознак. До таких ознак можуть належати: колірні характеристики, контури об’єктів, текстурні властивості зображення тощо.

Основна ідея SVM полягає у знаходженні гіперплощини, яка найкраще розділяє точки даних на різні класи. У класифікації SVM працює шляхом знаходження оптимальної гіперплощини, яка максимізує запас, тобто відстань між нею та найближчими точками даних з кожного класу, відомими як опорні вектори. Вона ефективно діє як межа прийняття рішень, дозволяючи SVM класифікувати нові точки даних в один із заздалегідь визначених класів на основі того, на якій її стороні вони знаходяться [4].

Проте він добре підходить для задач класифікації з невеликим або середнім обсягом даних, коли ознаки чітко розділяють класи, бо на великих наборах даних цей метод може бути обчислювально витратним і менш ефективним.

1.2.2 k-NN

Метод найближчих сусідів (k-NN) – це метод машинного навчання, який зазвичай використовується для класифікації. Він працює шляхом знаходження k-найближчих точок даних (сусідів) до заданого вхідного зразка та визначає клас екземпляра на основі більшості серед цих сусідів. Цей метод є простим у реалізації, однак його ефективність може залежити від вибору метрики відстані (зазвичай евклідової) та кількості сусідів k. Як і з методом опорних векторів, метод найближчих сусідів зазвичай застосовується після попереднього вилучення ознак, таких як значення кольору пікселів, текстурні властивості зображення тощо. Крім того ефективність цього методу знижується на великих наборах даних через необхідність обчислення відстані до кожного зразка в навчальному наборі даних, що призводить до обчислювальних витрат [5].

1.2.3 CNN

Згорткові нейронні мережі (CNN) – це метод машинного навчання, який призначений для розпізнавання закономірностей у даних за допомогою ієрархічного навчання. CNN особливо добре підходять для обробки даних зображень завдяки своїй унікальній архітектурі, яка складається з кількох ключових шарів:

– згортковий шар, який застосовує набір фільтрів до вхідного зображення, створюючи набір карт ознак. Кожен фільтр виявляє певну ознаку, таку як краї, текстури або візерунки;

– шар активації для введення нелінійності в модель, що дозволяє вивчати складніші закономірності;

– шар об’єднання, який виконує зниження частоти дискретизації (наприклад, максимальне об'єднання або усереднення об'єднання), щоб

зменшити просторові розміри карт ознак, зберігаючи важливу інформацію та зменшуючи обчислювальне навантаження;

– повнозв’язний шар, який бере характеристики з попередніх шарів і використовують їх для отримання кінцевого результату;

– шар відсіву, який використовується для запобігання перенавчання деактивуючи частину нейронів під час навчання, забезпечуючи хороше узагальнення мережі на нових даних [6].

Таким чином згорткові нейронні мережі обробляють вхідні зображення через серію згорткових, активаційних та об'єднувальних шарів для вилучення ієрархічних ознак. Спочатку виявляються низькорівневі ознаки, такі як ребра, а в міру просування даних глибше в мережу розпізнаються ознаки вищого рівня, такі як форми та об'єкти. Таке ієрархічне вилучення ознак робить CNN дуже ефективними для завдань, пов'язаних із зображеннями [6].

І на відміну від методів опорних векторів та найближчих сусідів, згорткові нейронні мережі не потребують ручного вилучення ознак, оскільки вони здатні навчатися цьому автоматично, що забезпечує високу ефективність при великих та складних наборах даних.

Порівняльний аналіз трьох зазначених методів штучного інтелекту представлено в табл. 1.1.

Таблиця 1.1 – Порівняльний аналіз методів штучного інтелекту

Метод	Ручне вилучення ознак	Ефективність на великих наборах даних	Точність на складних зображеннях	Здатність до узагальнення
SVM	Потрібно	Середня	Середня	Середня
k-NN	Потрібно	Низька	Низька	Низька
CNN	Не потрібно	Висока	Висока	Висока

Виходячи з результатів порівняння можна сказати, що для класифікації зображення шкіри з метою виявлення меланоми оптимальним вибором є згорткові нейронні мережі. Це пов’язано з їхньою здатністю до автоматичного

вилучення ознак, високою ефективністю при обробці великих обсягів даних, кращою точністю класифікації складних зображень та високою здатністю до узагальнення.

1.3 Огляд існуючих аналогів

1.3.1 SkinVision

SkinVision – це схвалена дерматологами служба, яка допомагає оцінити плями на шкірі та родимки на наявність найпоширеніших типів раку шкіри, включаючи меланому [7]. Застосунок дозволяє користувачам створювати свої облікові записи, аналізувати зображення з галереї або камери, переглядати історії сканувань, отримувати рекомендації та інше. На рис. 1.1 наведено основний вигляд застосунку.

Переваги:

– інтуїтивно зрозумілий інтерфейс;
– надання оцінки стану шкіри з рекомендаціями;
– доступний в Україні.

Недоліки:

– результати аналізу залежать від якості зображення;
– відсутній експорт результатів аналізу у форматі PDF;
– деякі сервіси доступні за передплатою.

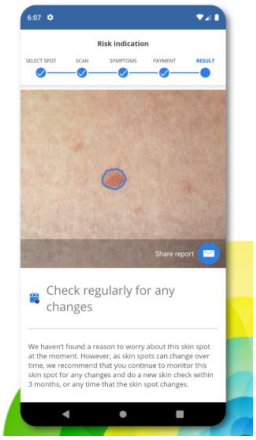


Рисунок 1.1 – Застосунок «SkinVision» [7]

1.3.2 SkinScreener

SkinScreener – це зручний застосунок, який використовує штучний інтелект для швидкої та легкої оцінки ризику меланоми при ураженнях шкіри (родимки, плями на шкірі) [8]. Цей застосунок дозволяє користувачам створювати свої облікові записи, авторизувати через пошту та пароль або через Google акаунт, редагувати дані про себе, здійснювати аналіз шкіри для оцінки потенційного ризику на основі завантажених зображень з галереї або зроблених через камеру, отримувати базові рекомендації щодо подальших дій та переглядати історії сканувань. Також SkinScreener дозволяє користувачам експортувати результати у форматі PDF та звертатися за допомогою через модуль зворотного зв'язку. На рис. 1.2 наведено основний вигляд застосунку.

Переваги:

– інтуїтивний та стильний інтерфейс;
– можливість експортувати результати у форматі PDF;
– отримувати допомогу через модуль зворотного зв'язку;
– можливість отримувати оцінку ризику меланоми та базові рекомендації щодо подальших дій.

Недоліки:

– недоступний в Україні.

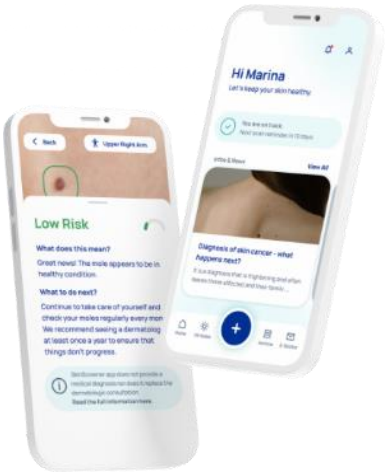


Рисунок 1.2 – Застосунок «SkinScreener» [8]

1.3.3 SkiniveMD

SkiniveMD – це мобільний застосунок для оцінки стану здоров'я шкіри та відстеження змін шкіри з використанням штучного інтелекту [9]. Застосунок дозволяє автоматично аналізувати зображення шкіри, визначати рівень ризику, зберігати та експортувати історію сканувань у форматі PDF. На рис. 1.3 наведено основний вигляд застосунку.

Переваги:

– інтуїтивний інтерфейс;

– можливість зберігати та експортувати історію сканувать у форматі PDF.

Недоліки:

– недоступний в Україні;

– результати аналізу залежать від якості зображення;

– деякі функціонал доступний за передплатою.

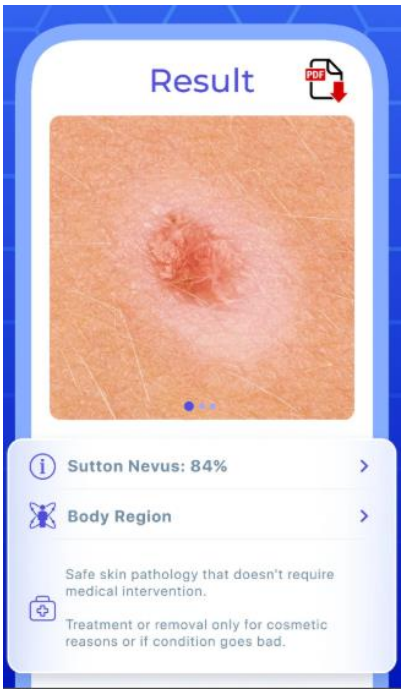


Рисунок 1.3 – Застосунок «SkiniveMD» [9]

Порівняльний аналіз застосунків представлено в табл. 1.2.

Таблиця 1.2 – Порівняльний аналіз застосунків

Параметр порівняння	SkinVision	SkinScreener	SkiniveMD
Доступний в Україні	+	-	-
Зручність використання	+	+	+
Відсутність платної підписки	-	+	-
Нечутливість до якості зображення	-	+	-
Можливість експортувати результати в PDF	-	+	+
Можливість отримання оцінки ризику меланоми та рекомендації	+	+	+

Виходячи з результатів аналізу існуючих аналогів, завданням даної роботи є створення мобільного застосунку для автоматизованої оцінки ризику меланоми на основі аналізу зображень шкіри. Відповідно до цього було розроблено технічне завдання, що міститься в додатку А.

1.4 Висновки до розділу

У даному розділі було проведено аналіз предметної області, існуючих аналогів та методів штучного інтелекту, їхніх переваг та недоліків. В результаті було обрано згорткові нейронні мережі для класифікації зображення шкіри з метою виявлення меланоми та розроблено технічне завдання.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Огляд архітектур згорткових нейронних мереж

У підрозділі 1.2 для класифікації зображення шкіри з метою оцінки ризику меланоми було обрано згорткові нейронні мережі (CNN). У цьому підрозділі буде виконано огляд основних архітектур CNN (VGG16, ResNet50 та MobileNet) та проведено їхнє порівняння для обґрунтування вибору базової моделі для подальшої модифікації.

2.1.1 VGG16

VGG16 – це архітектура згорткової нейронної мережі, яка була розроблена Visual Geometry Group Оксфордського університету. Вперше вона була представлена у 2014 році, і з того часу вона стала однією з найпопулярніших архітектур глибокого навчання для класифікації зображень. Вона складається з 16 шарів, включаючи 13 згорткових шарів і 3 повнозв'язні шари. Кожен згортковий шар має розмір фільтра 3x3 і крок 1 піксель. Шари об’єднання використовуються для зменшення дискретизації карт функцій і зменшення просторового розміру вхідних даних, а повнозв’язні шари використовуються для класифікації зображень на основі виділених ознак [10].

Однією з переваг VGG16 є її простота та чітка структура. Проте ця модель містить велику кількість параметрів (близько 138 мільйонів), що може призвести до великих обчислювальних витрат і ускладнює її застосування.

2.1.2 ResNet50

ResNet50 – це архітектура згорткової нейронної мережі, розроблена Microsoft Research у 2015 році. Вона складається з 50 шарів, включаючи 48 згорткових шарів, 1 шар максимального об’єднання (MaxPool) та 1 шар глобального усередненого об’єднання (Global Average Pooling) [11].

Ключовою особливістю ResNet50 є використання залишкових зв'язків, також відомих як скорочені зв'язки. Це прості шляхи, які дозволяють моделі пропускати деякі кроки в процесі навчання. Іншими словами, замість того, щоб змушувати модель передавати інформацію через кожен окремий шар, ці скорочені зв'язки дозволяють їй переносити важливі деталі безпосередньо. Це робить навчання швидшим і надійнішим [12].

Вона містить близько 25,6 мільйонів параметрів, що значно менше, ніж у VGG16. Однак через складнішу структуру та наявності залишкових зв’язків це може ускладнити реалізацію та збільшити обчислювальні витрати.

2.1.3 MobileNet

MobileNet – це архітектура згорткової нейронної мережі від Google, оптимізована для завдань класифікації зображень [13].

Вона складається з 28 основних шарів, включаючи згорткові шари з розділенням по глибині (depthwise separable convolutions), які чередуються з точковими згортками (pointwise convolutions) та шарами об’єднання. Її основною метою є швидка та точна класифікація зображень на пристроях з обмеженими ресурсами. Тому ключовою інновацією, яка відрізняє MobileNet від VGG16 та ResNet50, є використання згорток, розділених по глибині, що значно зменшує розмір моделі та обчислювальні витрати [14].

Також модель містить приблизно 4,2 мільйонів параметрів, що значно менше у порівнянні з VGG16 та ResNet50. Проте вона поступається їм в точності у складних задачах класифікації, де потрібна висока здатність до узагальнення.

Порівняльний аналіз цих архітектур представлено в табл. 2.1. Виходячи з результатів порівняння, можна сказати, що MobileNet є оптимальним вибором в якості базової моделі для подальших модифікацій через свою компактність, низькі обчислювальні витрати та високу придатність для мобільних пристроїв.

Таблиця 2.1 – Порівняльний аналіз архітектур

Параметр порівняння	VGG16	ResNet50	MobileNet
Обсяг параметрів	138 млн	25,6 млн	4,2 млн
Кількість шарів	16	50	28
Обчислювальні витрати	Високі	Високі	Низькі
Придатність для мобільних пристроїв	Низька	Середня	Висока
Рівень точності на ImageNet	71,5 %	76,3 %	70,6 %

2.2 Модифікація моделі

Для підвищення ефективності класифікації та адаптації до особливостей завдання було прийнято рішення модифікувати архітектуру MobileNet.

На рис. 2.1 представлено архітектуру модифікованої моделі.

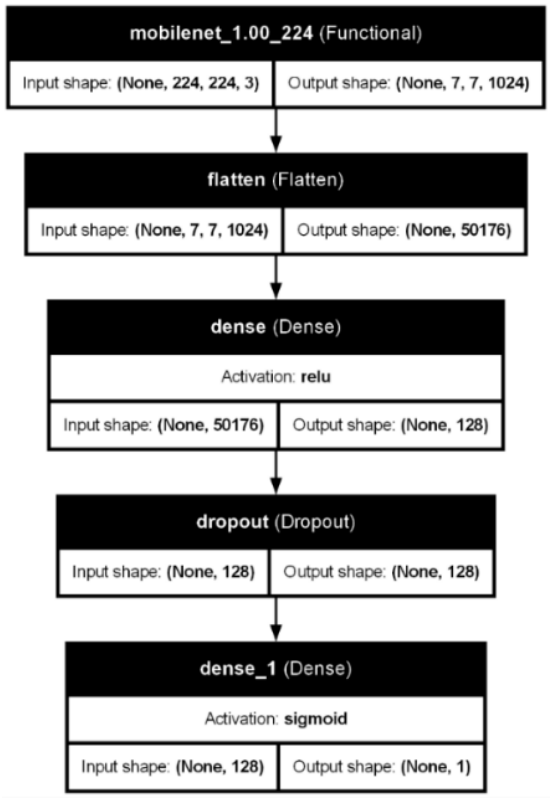


Рисунок 2.1 – Архітектура модифікованої моделі

Модель включає вхідний шар, який приймає зображення шкіри у форматі 224×224 з трьома каналами RGB. Потім використовується базова модель MobileNet, що виконує автоматичне виділення високорівневих ознак із вхідних зображень. Після цього отримані ознаки проходять через шар Flatten, який перетворює їх у вектор для подальшої обробки. Далі використовується повнозв'язний шар з 128 нейронами та функцією активації ReLU, який формує узагальнене представлення ознак. Для зменшення ризику перенавчання моделі застосовується шар Dropout з коефіцієнтом 0,5. Завершується модель вихідним повнозв'язним шаром з одним нейроном та функцією активації sigmoid, який видає результат бінарної класифікації зображення.

2.3 Аналіз датасету для навчання та тестування моделі

Для навчання та тестування моделі буде використовуватися датасет «Melanoma Cancer Image Dataset» з сайту Kaggle [15]. Цей набір даних містить 13900 зображень, поділених на дві основні категорії: «Benign» (доброякісні) та «Malignant» (злоякісні). Крім цього, цей набір даних вже розподілений на тренувальну та тестову вибірку. Приклади зображень з датасету представлено на рис. 2.2.

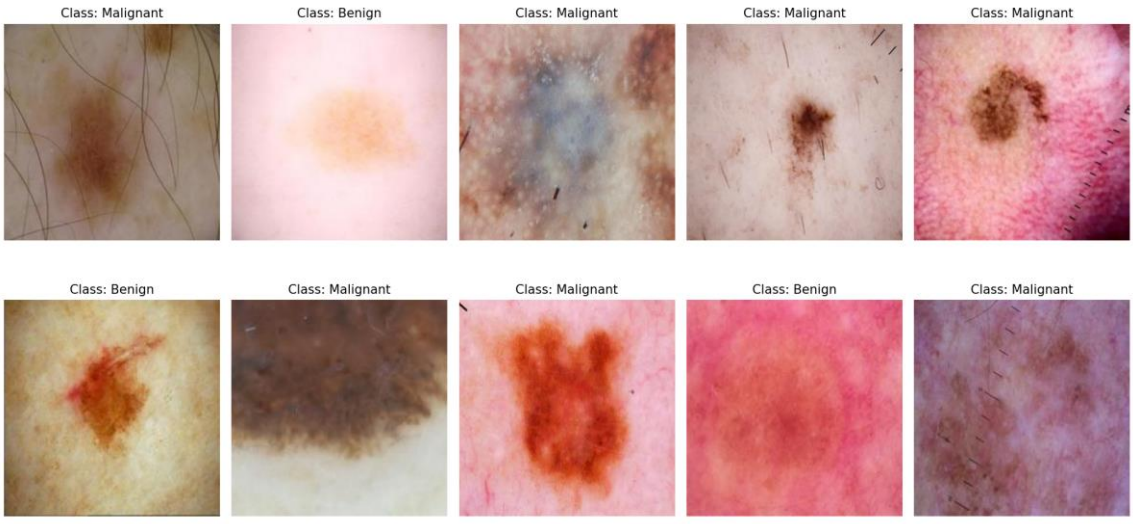


Рисунок 2.2 – Приклади зображень з датасету

2.4 Висновки до розділу

У даному розділі було виконано огляд основних архітектур CNN (VGG16, ResNet50 та MobileNet) та проведено їхнє порівняння для вибору базової моделі. В результаті базовою моделлю було обрано MobileNet та на її основі розроблено модифіковану архітектуру, адаптовану під бінарну класифікацію зображення шкіри з метою виявлення меланоми. Також для навчання та тестування моделі було вирішено використовувати датасет «Melanoma Cancer Image Dataset» з сайту Kaggle.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

3.1 Вибір мови програмування та програмні засоби для створення мобільного застосунку

Dart – це сучасна об'єктно-орієнтована мова програмування, розроблена компанією Google та призначена для створення мобільних, веб та інших застосунків. [16].

Ця мова програмування відзначається своєю ефективністю, що дозволяє програмам працювати безперебійно та швидко реагувати. Також Dart оптимізований для швидкого виконання та може ефективно обробляти складні завдання, що робить його гарним вибором для програм, які потребують високої продуктивності. Мова має простий синтаксис і структуру, що робить її доступною для початківців. Її прямолінійність сприяє швидшим циклам розробки та легшому супроводженню коду. Крім цього, Dart пропонує гнучкість у варіантах розгортання. Як зазначалося раніше, його можна використовувати для створення застосунків для різних платформ, включаючи мобільні, веб та інші. Ця гнучкість дозволяє розробникам орієнтуватися на широкий спектр пристроїв та охоплювати ширшу аудиторію своїми застосунками [17].

У табл. 3.1 представлено порівняльний аналіз мови Dart та інших мов програмування для розробки мобільного застосунку.

Таблиця 3.1 – Порівняльний аналіз мов програмування

Параметр порівняння	Dart	Swift	Kotlin
Простота розробки	Висока	Середня	Висока
Активність спільноти	Висока	Висока	Середня
Стійкість та захищеність	Висока	Середня	Середня
Швидкодія	Висока	Низька	Середня

Відповідно, для розробки мобільного застосунку було обрано платформу Flutter, яка є основним інструментом розробки для мови Dart.

3.2 Вибір технологій для розробки серверної частини та нейромереж

3.2.1 Вибір мови програмування

Для розробки серверної частини та нейромереж обрана мова Python.

Python – це високорівнева інтерпретована мова програмування, відома своїм чітким синтаксисом та читабельністю. Створена Гвідо ван Россумом та вперше випущена в 1991 році. Мова орієнтована на простоту розробки та зниження вартості обслуговування програм, що робить її улюбленою серед розробників, які цінують чистий та ефективний код [18].

Однією з основних причин, чому Python використовується для машинного навчання та побудови бекенду, є велика кількість бібліотек та фреймворків, які спрощують процес кодування та заощаджують час розробки. Працюючи з Python, розробнику не потрібно витрачати багато часу на безпосереднє написання коду: завдяки простому синтаксису та високій читабельності він може зосередитися на вирішенні більш складних проблем, пов’язаних з машинним навчанням або бекендом [19].

У табл. 3.2 наведено порівняльний аналіз мови Python та інших мов програмування для створення бекенду та роботи з машинним навчанням.

Таблиця 3.2 – Поріняльний аналіз мови Python та інших мов програмування для створення бекенду та роботи з машинним навчанням

Параметр порівняння	C#	Java	Python
1	2	3	4
Масштабованість	Висока	Висока	Середня
Простота коду	Середня	Середня	Висока

Продовження таблиці 3.2

1	2	3	4
Спільнота та документація	Середня	Середня	Висока
Популярність в машинному навчанні та бекенді	Середня	Середня	Висока
Ефективність виконання	Висока	Висока	Низька

3.2.2 Вибір інструменту для створення серверної частини

У табл. 3.3 подано порівняльний аналіз двох найпоширеніших інструментів для розробки серверної частини на Python для взаємодії з моделлю нейромереж: Flask та Django.

Таблиця 3.3 – Порівняльний аналіз інструментів для розробки бекенду з можливістю взаємодії з моделлю нейромереж

Параметр критерію	Flask	Django
Здатність до масштабування	-	+
Зручність налаштування	+	-
Простота інтеграції з бібліотеками ML	+	-
Швидкість прототипування	+	-

Таким чином, принято рішення обрати Flask для реалізації бекенду на Python через його зручність налаштування, простоту інтеграції з бібліотеками машинного навчання.

3.2.3 Вибір бібліотеки для побудови нейромережі

У табл. 3.4 представлено порівняння найвідоміших бібліотек для побудови нейронних мереж на Python: TensorFlow, MXNet та PyTorch.

Таблиця 3.4 – Огляд найвідоміших бібліотек для побудови нейронних мереж

Параметр критерію	MXNet	TensorFlow	PyTorch
Зручність використання	Низька	Висока	Середня
Швидкість виконання	Середня	Висока	Висока
Популярність та підтримка	Низька	Висока	Висока
Екосистема та інструментарій	Середня	Висока	Середня

Відповідно, було вирішено обрати TensorFlow для побудови нейронних мереж через його високу зручність використання, швидкість виконання, попурністю та підтримку спільноти, а також розвинену екосистему інструментів.

3.3 Формування функціональних вимог до програмного продукту

Функціональні вимоги програмного продукту:

– система повинна надати можливість користувачу проходити процедуру реєстрації та авторизації;

– користувач має змогу переглядати та редагувати інформацію про себе;

– система повинна надати можливість користувачу вийти з облікового запису;

– система повинна дозволяти користувачу змінювати тему та мову застосунку;

– користувачу надається можливість виконати аналіз шкіри для виявлення ризику меланоми за допомогою зображення;

– система повинна забезпечувати користувачу можливість обрати зображення з галереї або зробити зображення за допомогою камери;

– користувачу надається можливість переглядати власні аналізи;

– користувач має змогу переглядати повну інформацію про результати аналізу;

– користувач має змогу видаляти конкретний аналіз;

– система повинна забезпечити можливість користувачу зберігати результати аналізу у форматі PDF.

З урахуванням наведених функціональних вимог було розроблено діаграму варіантів використання. Діаграма варіантів використання зображена на рис. 3.1.

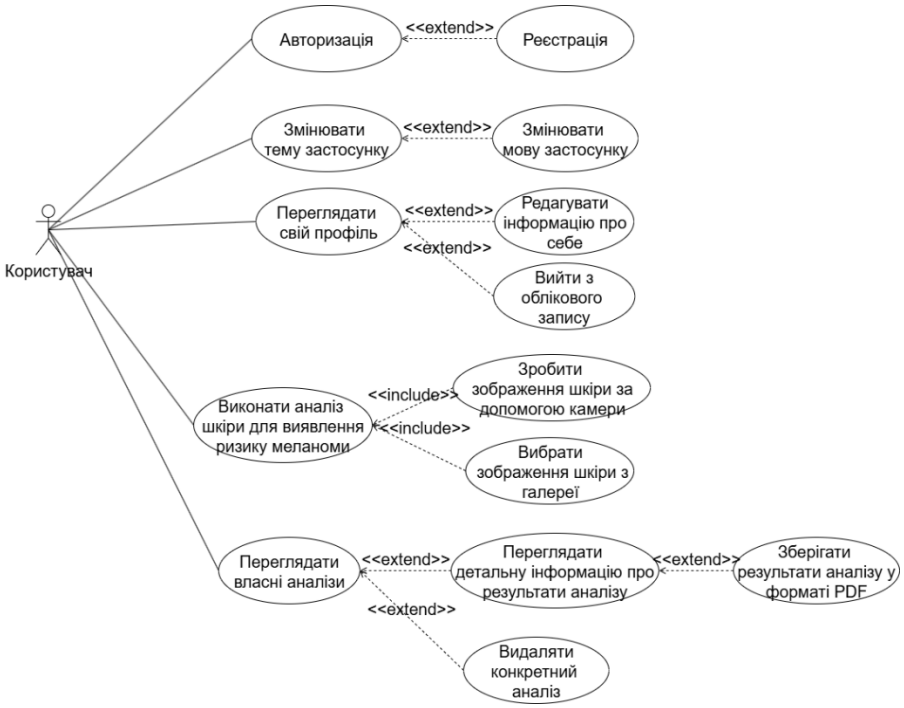


Рисунок 3.1 – Діаграма варіантів використання

3.4 Обґрунтування вибору бази даних та сервісу для побудови

Попри різноманіття типів БД, найпопулярнішими є реляційні та нереляційні.

Реляційна база даних – це тип бази даних, яка зберігає та впорядковує дані структурованим чином. У реляційній базі даних дані організовані в одну або кілька таблиць, кожна з яких має унікальне ім'я та набір стовпців. Кожен

рядок у таблиці представляє окремий запис, а стовпці представляють різні атрибути або характеристики цього запису. Однією з ключових особливостей реляційної бази даних є те, що вона дозволяє встановлювати зв'язки між таблицями. Це означає, що дані можна пов'язувати між собою на основі спільних значень, що дозволяє виконувати складніші запити та аналіз [20].

Нереляційна база даних (NoSQL) – це тип бази даних, яка не спирається на традиційну табличну структуру рядків і стовпців, що зустрічається в реляційних базах даних. Натомість вона використовує гнучкі моделі даних, такі як пари ключ-значення, документи, графіки та сховища з широкими стовпцями. Ця гнучкість дозволяє нереляційним базам даних ефективно керувати неструктурованими, напівструктурованими та структурованими даними [21].

У табл. 3.3 подано порівняльний аналіз цих двох типів бази даних.

Таблиця 3.3 – Порівняльний аналіз типів бази даних

Параметр порівняння	Реляційна БД	Нереляційна БД
Гнучкість структури даних	-	+
Надійність та цілісність даних	+	-
Підтримка різних типів даних	-	+
Здатність до масштабування	-	+

Виходячи з результатів порівняння реляційної та нереляційної бази даних, принято рішення обрати нереляційну базу даних через її гнучкість структури даних, підтримку різних типів даних та здатності до масштабування. Для побудови NoSQL баз даних можна використати платформу Firebase, яка забезпечує широкий спектр інструментів для розробки застосунків різного призначення.

У Firebase міститься два сервіси для роботи з NoSQL базами даних: Firestore та Realtime Database. Їхнє порівняння представлено в табл. 3.4.

Таким чином, прийнято рішення обрати Firestore для реалізації NoSQL бази даних через її гнучку організацію даних, масштабованість та підтримку складних запитів.

Таблиця 3.4 – Порівняння основних можливостей Firestore та Realtime Database

Характеристика	Firestore	Realtime Database
Організація даних	Дані організовані у форматі документів та колекцій	Дані організовані у вигляді JSON дерев
Можливості масштабування	Підтримує гнучке та автоматичне масштабування	Обмежене масштабування
Можливості виконання запитів	Підтримка складних запитів зі складними критеріями фільтрації та сортування	Можливі лише прості запити з базовим сортуванням та фільтрацією

3.5 Архітектура розроблювального програмного продукту

Архітектура цього застосунку буде реалізована з використанням шаблону проєктування Bloc та включатиме такі складові: Data, View та Bloc.

На рис. 3.2 представлено архітектуру застосунку, створену з використанням шаблону проєктування Bloc.

Компонент Data буде відповідати за управління даними та їхню обробку. Він складатиметься з двох основних частин: DataSource та Repository. DataSource забезпечує прямий доступ до різних джерел даних, таких як: API, БД та інші. Також він здійснює безпосереднє отримання даних із цих джерел у їх необробленому вигляді. У свою чергу, Repository забезпечує обробку та відправку цих даних до Bloc, абстрагуючи його від конкретного джерела даних.

Компонент View буде відповідати за візуальну частину застосунку та взаємодію з користувачем. Цей компонент буде займатися відображенням даних, отриманих від Bloc, а також перетворенням дії користувача (натискання

на кнопку, введення тексту тощо) на конкретні події, які передаються до Bloc для подальшої обробки.

Компонент Bloc буде виступати в ролі посередника між View та Data, та відповідати за бізнес-логіку застосунку. Окрім цього, він оброблює події користувача, отриманих від View, взаємодіє з даними через Repository і повертає їх у вигляді станів для відображення. Головною відмінністю Bloc від інших підходів полягає в тому, що Bloc немає прямих посилань на View. Замість цього працює зв'язок «один до багатьох», де один Bloc може передавати дані кільком View одночасно. Тому View потрібно самостійно підписатися на оновлення даних через спеціальні механізми. При цьому Bloc не знає і не контролює, які саме View на нього підписані. Він просто відправляє оновлення всім підписаним компонентам View.

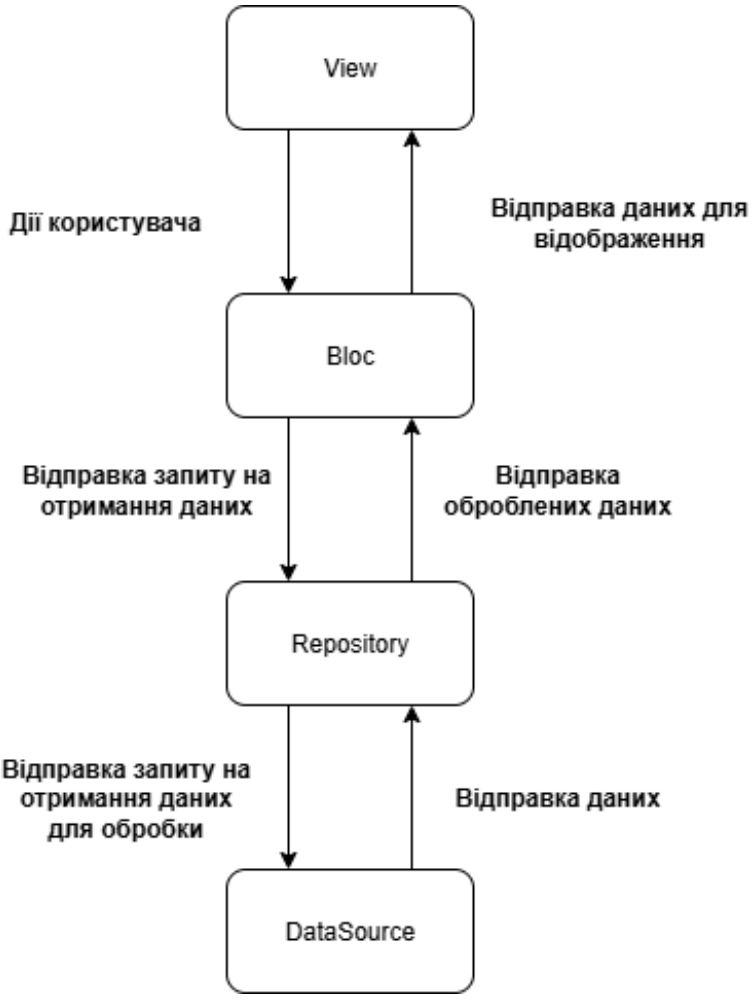


Рисунок 3.2 – Архітектура застосунку на основі шаблона проєктування Bloc

3.6 Висновки до розділу

У даному розділі проведено огляд мов програмування та програмних засобів для створення мобільного застосунку, серверної частини та нейромережі. В результаті аналізу обрано мову програмування Dart та платформу Flutter для розробки програмного забезпечення, а для побудови серверу та нейромережі обрано мову Python. Також у якості інструменту для побудови бекенду обрано Flask, а для нейронних мереж обрано TensorFlow. Для організації збереження даних буде використовуватися база даних NoSQL із використанням сервісу Firestore.

Крім того, було визначено функціональні вимоги та архітектуру програмного продукту.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

4.1 Схема функціонування програмного продукту

На рис. 4.1 подано функціональну схему програмного забезпечення.

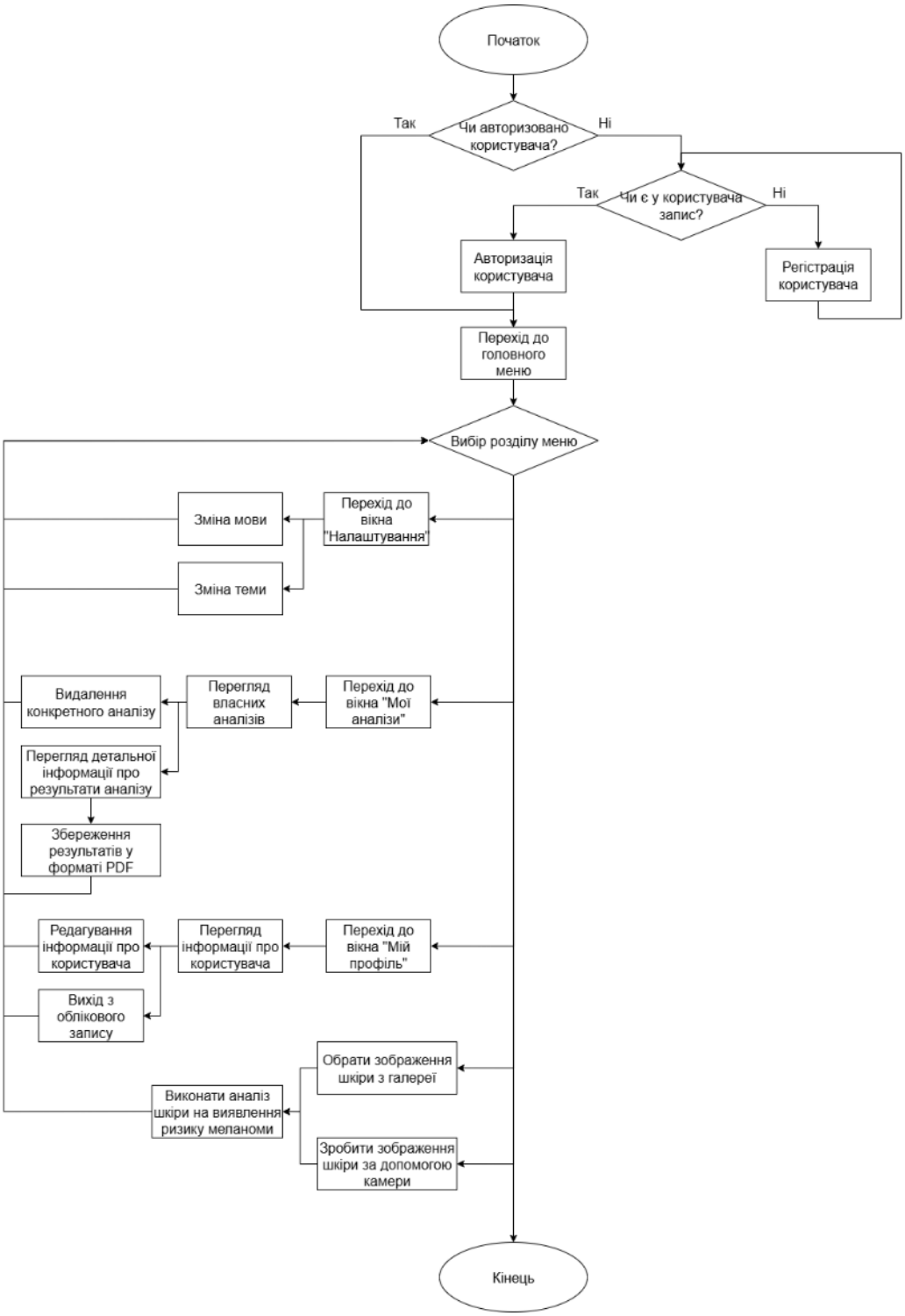


Рисунок 4.1 – Функціональна схема програмного забезпечення

4.2 Структура бази даних

У підрозділі 3.4 було прийнято рішення використовувати інструмент Firestore для створення нереляційної бази даних. До складу БД входять такі сутності: analysis та users.

Сутність «users» містить дані про користувачів системи: ідентифікаційний номер, прізвище, ім’я та по батькові, електронну адресу та пароль. Опис полів сутності міститься в табл. 4.1.

Таблиця 4.1 – Поля сутності «users»

Атрибут	Тип атрибута
id	String
realName	String
mailAddress	String
userPassword	String

Сутність «analysis» містить дані про результати аналізу: ідентифікаційний номер, ідентифікатор користувача, зображення шкіри, назва класу, показник ймовірності приналежності до цього класу та рекомендації щодо подальших дій. Опис полів сутності міститься в табл. 4.2.

Таблиця 4.2 – Поля сутності «analysis»

Атрибут	Тип атрибута
id	String
userId	String
imageUrl	String
className	String
confidence	double
recomendation	String

На рис. 4.2 представлено структуру бази даних з відображенням зв’язків між цими сутностями.

Сутності «users» та «analysis» пов’язані відношенням «один до багатьох», оскільки один користувач може мати кілька аналізів, тоді як кожен аналіз прив’язаний до конкретного користувача.

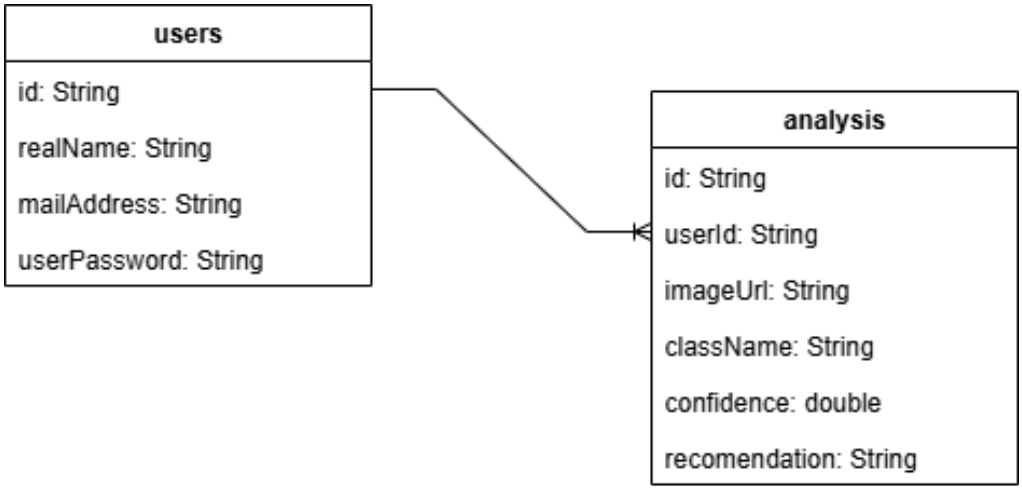


Рисунок 4.2 – Структура бази даних з відображенням зв’язків сутностей

4.3 Опис організації серверної частини та моделі нейромережі

У підрозділі 3.2 було прийнято рішення використовувати мову програмування Python, інструменти Flask та TensorFlow для розробки серверу та моделі. На рис. 4.3 представлено організацію файлів проєкту з використанням цих технологій.

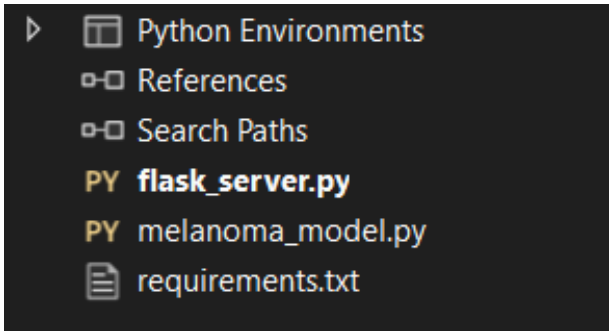


Рисунок 4.3 – Організація файлів серверної частини та моделі нейромережі

Файл flask_server.py можна вважати точкою входу в програму. Його основним компонентом є клас FlaskServer, який складається з таких методів:

– isModelExist(), який виконує перевірку наявності створенної моделі. У разі відсутності виконується процес її навчання через метод trainModel() класу MelanomaModel;

– runServer(port), який запускає сервер на вказаному порті;

– setupRoute(), який конфігурує маршрут «/predict» для обробки POST-запиту за допомогою методу predictMelanoma();

– predictMelanoma(), який виконує обробку вхідного POST-запиту маршруту «/predict» для виявлення ризику меланоми, отримуючи з тіла запиту зображення шкіри та викликаючи метод predict(image_path) класу MelanomaModel. Після чого повертає результати у вигляді JSON об’єкта.

Основним компонентом файлу melanoma_model.py є клас MelanomaModel, який відповідає за логіку роботи з моделлю нейромережі, включаючи створення, навчання, перевірку якості, використання для отримання оцінки ризику меланоми та інше. Клас складається з таких методів:

– createDataGenerators(train_dir, test_dir), який забезпечує попередню обробку даних;

– buildModel(), який ініціює побудову моделі та повертає її для навчання в методі trainModel();

– trainAndSaveModel(), який виконує процес навчання та збереження моделі;

– predict(image_path), який забезпечує отримання оцінки наяності ознак меланоми на основі зображення шкіри та повертає діагностичний клас (Benign або Malignant), показник ймовірності принадлежності до цього класу та рекомендації щодо подальших дій. Показник ймовірності обчислюється ймовірність обраного класу у відсотках, використовуючи для цього метод round(class_prob * 100, 2), а текст рекомендацій був сформований вручну і програмно обирається залежно від обраного класу;

– checkModelQuality(test_generator, model), який виконує перевірку якості моделі на тестових даних та відображає матрицю помилок.

4.4 Опис організації програмного продукту

У підрозділі 3.5 було вирішено використовувати шаблон проєктування Bloc для розробки застосунку. На рис. 4.4 представлено організацію проєкту відповідно до вибраної архітектури.

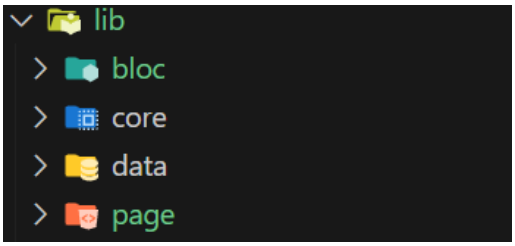


Рисунок 4.4 – Організація проєкту на основі шаблону проєктування Bloc

4.4.1 Модуль data та його класи

Пакет data складається з класів, що забезпечують управління та обробку даних (рис. 4.5).

Клас UserModel представляє модель сутності «users».

Клас AnalysisModel представляє модель сутності «analysis».

Клас CameraPicker забезпечує отримання зображень з галереї або камери та складається з таких методів:

– Future<File> pickImageFileFromGallery(), який здійснює отримання зображення з галереї та повертає його файл;

– Future<File> pickImageFileFromCamera(), який здійснює створення та отримання зображення за допомогою камери та повертає його файл.

Клас LocalizationSharedPreference реалізує роботу з локальним збереженням та завантаженням мови застосунку. Клас складається з таких методів:

– Future<void> saveLanguageCode(String languageCode), який виконує збереження налаштування мови застосунку локально;

– Future<String> loadLanguageCode(), який здійснює завантаження мови застосунку з локального сховища.

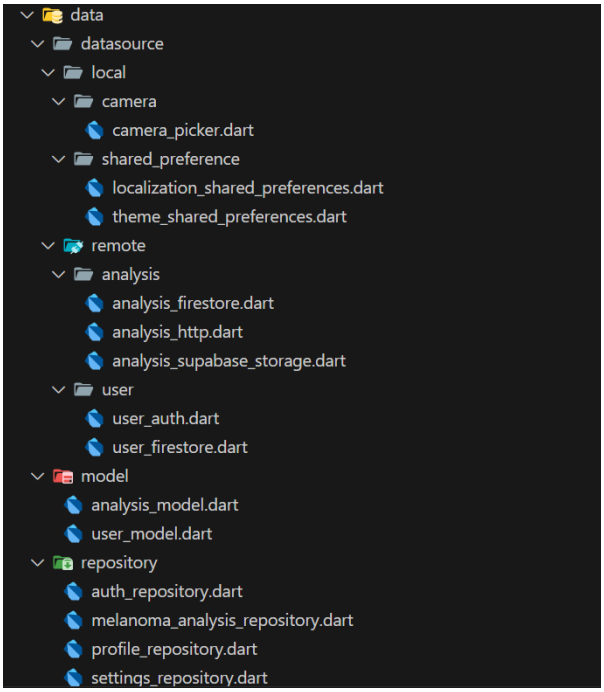


Рисунок 4.5 – Модуль data

Клас ThemeSharedPreferences реалізує роботу з локальним збереженням та завантаженням теми застосунку. Клас складається з таких методів:

– Future<void> saveTheme(bool isDarkMode), який зберігає налаштування теми застосунку локально;

– Future<bool> loadTheme(), який реалізує завантаження теми з локального сховища.

Клас UserFirestore здійснює роботу зі збереженням та отриманням даних про користувача з Firestore. Клас складається з таких методів:

– Future<void> saveUser(UserModel userModel), який зберігає дані про користувача в Firestore;

– Future<UserModel?> getUserModelById(String id), який реалізує отримання даних про користувача з Firestore по його ідентифікаційному номеру.

Клас UserAuth здійснює роботу зі регістрацією, авторизацією та виходом користувача з облікового запису. Клас складається з таких методів:

– Future<void> signOut(), який забезпечує вихід з облікового запису;

– Future<String> signUpWithEmailAndPassword(String email, String password), який реалізує регістрацію користувача;

– Future<void> signInWithEmailAndPassword(String email, String password), який реалізує авторизацію користувача.

Клас AnalysisFirestore забезпечує збереження, видалення та отримання списку аналізів користувача з Firestore. Клас складається з таких методів:

– Future<void> saveAnalysis(AnalysisModel analysisModel), який здійснює збереження даних про аналіз користувача в Firestore;

– Future<void> deleteAnalysisFromFirestore(String analysisId), який реалізує видалення аналізу з Firestore по його ідентифікаційному номеру;

– Future<List<AnalysisModel>> getUsersAnalysis(String userId), який забезпечення отримання списку аналізів користувача з Firestore по його ID.

Клас AnalysisSupabaseStorage реалізує роботу зі збереженням зображення шкіри віддалено та отриманням посилання на зображення. Клас складається з методу Future<String> saveAnalysisImage(File imageFile), який зберігає файл зображення у віддаленому сховищі та повертає посилання на нього.

Клас AnalysisHttp здійснює взаємодію зі сервером для аналізу зображення з метою визначення наявності ознак меланоми. Клас складається з методу Future<Map<String, dynamic>> predictMelanoma(String imagePath), який відправляє зображення на сервер, отримує результати аналізу у вигляді JSON об’єкта та повертає його.

Клас ProfileRepository виконує взаємодію з такими DataSource як: UserAuth та UserFirestore. Також забезпечує обробку даних та відправку цих даних до ProfileBloc. Клас складається з таких методів:

– Future<void> updateUserProfile(UserModel userModel), який оновлює дані про користувача використовуючи для цього метод saveUser(UserModel userModel) класу UserFirestore;

– Future<void> signOut(), який забезпечує вихід з облікового запису через метод signOut() класу UserAuth.

Клас SettingsRepository забезпечує взаємодію з такими DataSource як: ThemeSharedPreferences та LocalizationSharedPreferences. Також забезпечує обробку даних та відправку цих даних до SettingsBloc. Клас складається з таких методів:

– Future<bool> loadThemeMode(), який реалізує завантаження теми застосунку через метод loadTheme() класу ThemeSharedPreferences;

– Future<void> saveThemeMode(bool isDarkMode), який здійснює збереження теми застосунку локально через метод saveTheme(bool isDarkMode) класу ThemeSharedPreferences;

– Future<String> loadLanguageCode(), який забезпечує завантаження мови через метод loadLanguageCode() класу LocalizationSharedPreferences;

– Future<void> saveLanguageCode(String languageCode), який виконує збереження мови локально через метод saveLanguageCode(String languageCode) класу LocalizationSharedPreferences.

Клас AuthRepository реалізує взаємодію з такими DataSource як: UserAuth та UserFirestore. Також забезпечує обробку даних та відправку цих даних до LoginBloc, RegisterBloc та AppBloc. Клас складається з таких методів:

– Future<void> signUpUser(String email, String password, String fullName), який реалізує процес регістрації користувача через методи signUpWithEmailAndPassword(String email, String password) класу UserAuth та saveUser(UserModel userModel) класу UserFirestore;

– Future<void> signInUser(String email, String password), який реалізує процес авторизації користувача через метод signInWithEmailAndPassword(String email, String password) класу UserAuth.

Клас MelanomaAnalysisRepository реалізує взаємодію з такими DataSource як: CameraPicker, AnalysisHttp, AnalysisSupabaseStorage, AnalysisFirestore та UserAuth. Клас складається з таких методів:

– Future<void> pickImageFromGallery(), який реалізує аналіз зображення шкіри, обраного з галереї, для виявлення ризику меланоми та збереження результатів через методи pickImageFileFromGallery() класу CameraPicker, saveAnalysisImage(File imageFile) класу AnalysisSupabaseStorage, predictMelanoma(String imagePath) класу AnalysisHttp та saveAnalysis(AnalysisModel analysisModel) класу AnalysisFirestore;

– Future<void> pickImageFromCamera(), який також реалізує аналіз зображення шкіри, але отриманого за допомогою камери. Відрізняється лише тим, що замість методу pickImageFileFromGallery() використовується метод pickImageFromCamera();

– Future<void> deleteAnalysis(AnalysisModel analysisModel), який забезпечує видалення конкретного аналізу користувача через метод deleteAnalysisFromFirestore(String analysisId) класу AnalysisFirestore;

– Future<List<AnalysisModel>> getUsersAnalysis(), який здійснює отримання списку всіх аналізів користувача через метод getUsersAnalysis(String userId) класу AnalysisFirestore.

4.4.2 Модуль view та його класи

Пакет view складається з класів, що реалізують візуальну частину застосунку та взаємодію з користувачем (рис. 4.6).

Клас HomePage реалізує екран основного меню застосунку та є підкласом StatelessWidget.

Клас LoginPage реалізує екран авторизації та є підкласом StatelessWidget.

Клас ProfilePage формує екран профілю користувача та є підкласом StatelessWidget.

Клас RegisterPage створює екран регістрації та є підкласом StatelessWidget.

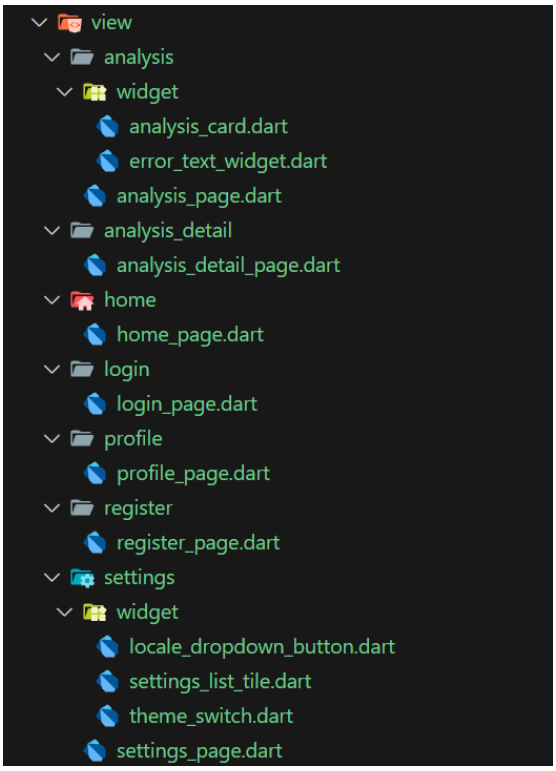


Рисунок 4.6 – Модуль view

Клас AnalysisDetailPage втілює екран перегляду повної інформації про результати конкретного аналізу та є підкласом StatelessWidget.

Клас AnalysisPage втілює екран власних аналізів користувача та є підкласом StatelessWidget.

Клас AnalysisCard забезпечує відображення аналізу у форматі картки та застосовується в класі AnalysisPage.

Клас ErrorTextWidget здійснює відображення тексту помилки у випадку, коли користувач не проводив жодних аналізів та застосовується в класі AnalysisPage.

Клас SettingsPage формує екран налаштування застосунку та є підкласом StatelessWidget.

Клас ThemeSwitch забезпечує керування зміною теми застосунку та застосовується в класі SettingsPage.

Клас LocaleDropdownButton реалізує управління зміною мови та задіюється в класі SettingsPage.

Клас SettingsListTile здійснює відображення елемента списку налаштувань та використовується в класі SettingsPage.

4.4.3 Модуль core та його класи

Пакет core складається з класів, які використовуються в усьому застосунку (рис. 4.7).

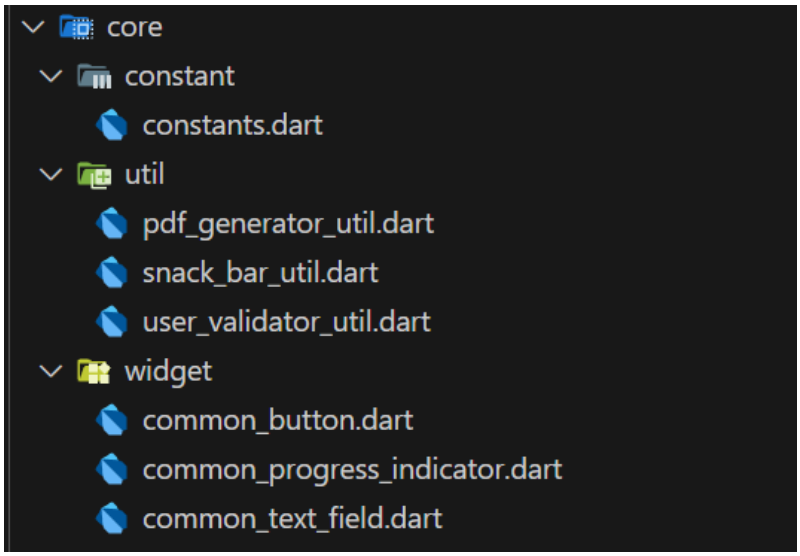


Рисунок 4.7 – Модуль core

Клас Constants зберігає константи для конфігурації застосунку.

Клас CommonButton реалізує стандартну кнопку для багаторазового використання в усьому застосунку.

Клас CommonProgressIndicator містить реалізацію універсального індикатора прогресу для багаторазового використання.

Клас CommonTextField забезпечує реалізацію універсального текстового поля з настроюваними параметрами.

Клас PdfGeneratorUtil здійснює експорт результатів конкретного аналізу у формат PDF. Клас складається з методу static Future<void> generateAnalysisPdf(AnalysisModel model), який реалізує цю функціональність.

Клас SnackBarUtil забезпечує відображення спливаючих повідомлень. Клас складається з методу static void showSnackBar(BuildContext context, String message, Icon icon), який здійснює цю функціональність.

Клас UserValidatorUtil здійснює перевірку коректності введених користувачем даних під час авторизації, реєстрації або редагуванню профілю. Клас складається з таких методів:

– static String? validateEmail(String email), який перевіряє, чи електронна адреса не порожня та закінчується на @gmail.com, та повертає сповіщення про некорретне введення у разі невідповідності;

– static String? validateFullName(String fullName), який перевіряє, щоб прізвище, ім’я та по-батькові не було порожнім та не містило спеціальних символів, та повертає сповіщення про некорретне введення у разі невідповідності;

– static String? validatePassword(String password), який перевіряє, щоб пароль був не менше 8 символів та мав хоча б одну велику літер, та повертає сповіщення про некорретне введення у разі невідповідності.

4.4.4 Модуль bloc та його класи

Пакет bloc складається з класів, що реалізують бізнес-логіку, обробляють події користувача та формують відповідні стани (рис. 4.8).

Клас SettingsBloc реалізує бізнес-логіку управління налаштування застосунку та забезпечує обробку подій зміни теми та мови. Клас складається з таких методів:

– Future<void> _initializeSettings(SettingsInitializeEvent event, Emitter<SettingsState> emit), який завантажує збережені налаштування теми та

мови з локального сховища використовуючи методи loadThemeMode() та loadLanguageCode() класу SettingsRepository;

– Future<void> _changeTheme(SettingsThemeChangeEvent event, Emitter<SettingsState> emit), який обробляє зміну теми застосунку, зберігає нове налаштування використовуючи метод saveThemeMode(bool isDarkMode) класу SettingsRepository та оновлює поточний стан для відображення;

– Future<void> _changeLanguage(SettingsLanguageChangeEvent event, Emitter<SettingsState> emit), який обробляє зміну мови, зберігає нове значення використовуючи метод saveLanguageCode(String languageCode) класу SettingsRepository та оновлює стан для відображення.

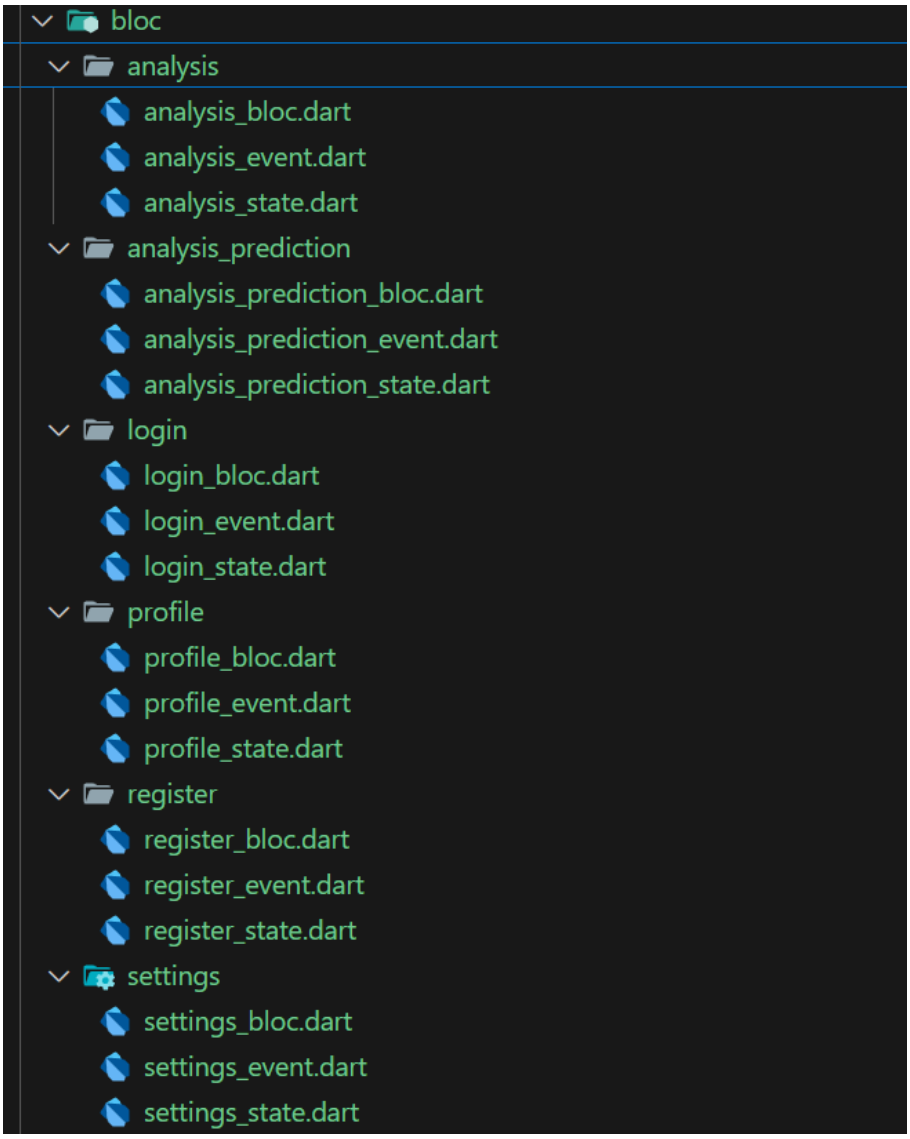


Рисунок 4.8 – Модуль bloc

Клас SettingsState містить дані про поточні налаштування застосунку та забезпечує їх передачу до компонентів View.

Файл settings_event.dart складається з класів SettingsInitializeEvent, SettingsThemeChangeEvent та SettingsLanguageChangeEvent.

Клас SettingsInitializeEvent представляє подію ініціалізації збережених налаштувань при запуску застосунку та оброблюється методом _initializeSettings(SettingsInitializeEvent event, Emitter<SettingsState> emit).

Клас SettingsThemeChangeEvent представляє подію зміни теми застосунку та оброблюється методом _changeTheme(SettingsThemeChangeEvent event, Emitter<SettingsState> emit).

Клас SettingsLanguageChangeEvent представляє подію зміни мови та оброблюється методом _changeLanguage(SettingsLanguageChangeEvent event, Emitter<SettingsState> emit).

Клас RegisterBloc реалізує бізнес-логіку процесу реєстрації користувача, забезпечує валідацію введених даних та взаємодію з AuthRepository. Клас складається з таких методів:

– void _setRegisterFullName(RegisterFullNameChangeEvent event, Emitter<RegisterState> emit), який оброблює зміну ПІБ під час реєстрації, виконує валідацію використовуючи метод validateFullName(String fullName) класу UserValidatorUtil та оновлює стан;

– void _setRegisterEmail(RegisterEmailChangeEvent event, Emitter<RegisterState> emit), який оброблює зміну електронної адреси під час реєстрації, виконує валідацію використовуючи метод validateEmail(String email) класу UserValidatorUtil та оновлює стан для відображення;

– void _setRegisterPassword(RegisterPasswordChangeEvent event, Emitter<RegisterState> emit), який оброблює зміну паролю під час реєстрації, виконує виконує валідацію використовуючи метод validatePassword(String password) класу UserValidatorUtil та оновлює стан для відображення;

– Future<void> _signUpUser(RegisterSubmitEvent event, Emitter<RegisterState> emit), який виконує процес реєстрації користувача

викликаючи метод signUpUser(String email, String password, String fullName) класу AuthRepository.

Клас RegisterState містить дані про поточний стан процесу реєстрації та забезпечує їх передачу до компонентів View.

Файл register_event.dart складається з класів RegisterFullNameChangeEvent, RegisterEmailChangeEvent, RegisterPasswordChangeEvent та RegisterSubmitEvent.

Клас RegisterFullNameChangeEvent представляє подію зміни ПІБ під час реєстрації та оброблюється методом _setRegisterFullName(RegisterFullNameChangeEvent event, Emitter<RegisterState> emit).

Клас RegisterEmailChangeEvent представляє подію зміни електронної адреси під час реєстрації та оброблюється методом _setRegisterEmail(RegisterEmailChangeEvent event, Emitter<RegisterState> emit).

Клас RegisterPasswordChangeEvent представляє подію зміни пароллю при реєстрації користувача та оброблюється методом _setRegisterPassword(RegisterPasswordChangeEvent event, Emitter<RegisterState> emit).

Клас RegisterSubmitEvent представляє подію підтвердження форми реєстрації користувача та оброблюється методом _signUpUser(RegisterSubmitEvent event, Emitter<RegisterState> emit).

Клас ProfileBloc реалізує бізнес-логіку управління профілю користувача, забезпечує валідацію даних та взаємодію з ProfileRepository. Клас складається з таких методів:

– void _setProfileFullName(ProfileFullNameChangeEvent event, Emitter<ProfileState> emit), який оброблює зміну ПІБ під час редагування профілю, виконує перевірку на правильність введення використовуючи метод validateFullName(String fullName) класу UserValidatorUtil та оновлює поточний стан;

– Future<void> _signOut(ProfileLogOutEvent event, Emitter<ProfileState> emit), який оброблює подію виходу з облікового запису викликаючи метод signOut() класу ProfileRepository;

– Future<void> _updateProfile(ProfileSubmitEvent event, Emitter<ProfileState> emit), який оброблює подію оновлення профілю користувача викликаючи метод updateUserProfile(UserModel userModel) класу ProfileRepository.

Клас ProfileState містить дані про поточний стан профілю користувача та забезпечує їх передачу до View.

Файл profile_event.dart складається з класів ProfileFullNameChangeEve nt, ProfileLogOutEvent та ProfileSubmitEvent.

Клас ProfileFullNameChangeEvent представляє подію зміни ПІБ під час редагуванню профілю та оброблюється методом _setProfileFullName(ProfileFullNameChangeEvent event, Emitter<ProfileState> emit).

Клас ProfileLogOutEvent представляє подію виходу з облікового запису та оброблюється методом _signOut(ProfileLogOutEvent event, Emitter<ProfileState> emit).

Клас ProfileSubmitEvent представляє подію оновлення профілю користувача та оброблюється методом _updateProfile(ProfileSubmitEvent event, Emitter<ProfileState> emit).

Клас LoginBloc забезпечує реалізацію бізнес-логіки процесу авторизації користувача, забезпечує валідацію даних та взаємодію з AuthRepository. Клас складається з таких методів:

– void _setLoginEmail(LoginEmailChangeEvent event, Emitter<LoginState> emit), який оброблює зміну електронної пошти під час авторизації, виконує валідацію даних викликаючи метод validateEmail(String email) класу UserValidatorUtil та оновлює поточний стан;

– void _setLoginPassword(LoginPasswordChangeEvent event, Emitter<LoginState> emit), який оброблює зміну паролю під час авторизації,

виконує перевірку на правильність введення викликаючи метод validatePassword(String password) класу UserValidatorUtil та оновлює стан;

– Future<void> _signInUser(LoginSubmitEvent event, Emitter<LoginState> emit), який виконує процес авторизації користувача викликаючи метод signInUser(String email, String password) класу AuthRepository.

Клас LoginState містить дані про поточний стан процесу авторизації та забезпечує їх передачу до View.

Файл login_event.dart складається з класів LoginEmailChangeEvent, LoginPasswordChangeEvent та LoginSubmitEvent.

Клас LoginEmailChangeEvent представляє подію зміни електронної пошти при авторизації та оброблюється методом _setLoginEmail(LoginEmailChangeEvent event, Emitter<LoginState> emit).

Клас LoginPasswordChangeEvent представляє подію зміни паролю при авторизації користувача та оброблюється методом _setLoginPassword(LoginPasswordChangeEvent event, Emitter<LoginState> emit).

Клас LoginSubmitEvent представляє подію підтвердження форми авторизації та оброблюється методом _signInUser(LoginSubmitEvent event, Emitter<LoginState> emit).

Клас AnalysisPredictionBloc реалізує бізнес-логіку процесу аналізу зображення для виявлення меланоми та забезпечує взаємодію з MelanomaAnalysisRepository. Клас складається з таких методів:

– Future<void> _pickUpImageFromGallery(PickImageFromGalleryEvent event, Emitter<AnalysisPredictionState> emit), який оброблює подію вибору зображення з галереї для аналізу викликаючи метод pickImageFromGallery() класу MelanomaAnalysisRepository та оновлює поточний стан;

– Future<void> _pickUpImageFromCamera(PickImageFromCameraEvent event, Emitter<AnalysisPredictionState> emit), який оброблює подію створення та вибору зображення для аналізу за допомогою камери викликаючи

метод pickImageFromCamera() класу MelanomaAnalysisRepository та оновлює стан.

Клас AnalysisPredictionState містить дані про поточний стан процесу аналізу зображення для виявлення ризику меланоми та забезчує їх передачу до компонентів View.

Файл analysis_prediction_event.dart складається з класів PickImageFromGalleryEvent та PickImageFromCameraEvent.

Клас PickImageFromGalleryEvent представляє подію вибору зображення з галереї для аналізу та оброблюється методом _pickUpImageFromGallery(PickImageFromGalleryEvent event, Emitter<AnalysisPredictionState> emit).

Клас PickImageFromCameraEvent представляє подію створення та вибору зображення для аналізу за допомогою камери, та оброблюється методом _pickUpImageFromCamera(PickImageFromCameraEvent event, Emitter<AnalysisPredictionState> emit).

Клас AnalysisBloc реалізує бізнес-логіку керуванням проведених користувачем аналізів, забезпечує взаємодію з MelanomaAnalysisRepository. Клас складається з таких методів:

– Future<void> _fetchData(AnalysisFetchEvent event, Emitter<AnalysisState> emit), який оброблює подію завантаження списку аналізів користувача використовуючи метод getAllUsersAnalysis() класу MelanomaAnalysisRepository та оновлює поточний стан;

– Future<void> _deleteAnalysis(AnalysisDeleteEvent event, Emitter<AnalysisState> emit), який оброблює подію видалення конкретного аналізу викликаючи метод deleteAnalysis(AnalysisModel analysisModel) класу MelanomaAnalysisRepository та оновлює стан;

– Future<void> _saveAnalysisInPDF(AnalysisSaveInPDFEvent event, Emitter<AnalysisState> emit), який оброблює подію збереження результатів аналізу у формат PDF використовуючи метод generateAnalysisPdf(AnalysisModel model) класу PdfGeneratorUtil.

Клас AnalysisState містить дані про поточний стан історії аналізів меланоми та забезпечує їх передачу до View.

Файл analysis_event.dart складається з класів AnalysisFetchEvent, AnalysisDeleteEvent та AnalysisSaveInPDFEvent.

Клас AnalysisFetchEvent представляє подію завантаження списку аналізів користувача та оброблюється методом _fetchData(AnalysisFetchEvent event, Emitter<AnalysisState> emit).

Клас AnalysisDeleteEvent представляє подію видалення конкретного аналізу та оброблюється методом _deleteAnalysis(AnalysisDeleteEvent event, Emitter <AnalysisState> emit).

Клас AnalysisSaveInPDFEvent представляє подію збереження результатів аналізу у формат PDF та оброблюється методом _saveAnalysisInPDF(AnalysisSaveInPDFEvent event, Emitter<AnalysisState> emit).

4.5 Опис алгоритмів роботи програмного продукту

4.5.1 Алгоритм вибору зображення з галереї

Крок 1. Користувач натискає на кнопку вибору зображення з галереї, генеруючи подію PickImageFromGalleryEvent.

Крок 2. AnalysisPredictionBloc отримує цю подію та викликає метод _pickUpImageFromGallery(), який передає запит до методу pickImageFromGallery() класу MelanomaAnalysisRepository для відкриття галереї.

Крок 3. Користувач обирає потрібне зображення з галереї.

Крок 4. Обране зображення зберігається у віддаленому сховищі, викликаючи метод saveAnalysisImage() класу AnalysisSupabaseStorage та отримується його посилання для подальшого аналізу.

4.5.2 Алгоритм створення зображення за допомогою камери

Крок 1. Користувач натискає на кнопку створення зображення за допомогою фото, генеруючи подію PickImageFromCameraEvent.

Крок 2. AnalysisPredictionBloc отримує цю подію та викликає метод _pickUpImageFromCamera(), який передає запит до методу pickImageFromCamera() класу MelanomaAnalysisRepository для відкриття камери.

Крок 3. Користувач створює зображення за допомогою камери.

Крок 4. Створене зображення зберігається у віддаленому сховищі, викликаючи метод saveAnalysisImage() класу AnalysisSupabaseStorage та отримується її посилання для подальшого аналізу.

4.5.3 Алгоритм виконання аналізу зображення шкіри для виявлення меланоми

Крок 1. Користувач обирає один із двух способів: завантажити зображення з галереї (підрозділ 4.5.1) або створити фото за допомогою камери (підрозділ 4.5.2).

Крок 2. Обране зображення передається через метод predictMelanoma() класу AnalysisHttp для аналізу.

Крок 3. Отримані результати аналізу зберігаються в Firestore викликаючи метод saveAnalysis() класу AnalysisFirestore.

Крок 4. Оновлюється стан AnalysisPredictionState та відображається користувачу сповіщення щодо успішного виконання аналізу.

4.5.4 Алгоритм перевірки якості моделі

Крок 1. Завантаження навченої моделі для перевірки її якості.

Крок 2. Виконується передбачення на тестовому наборі даних через метод predict() моделі, який повертає масив ймовірностей для кожного зображення.

Крок 3. Отримані ймовірності перетворюються в мітки класів.

Крок 4. Витягуються справжні мітки класів з тестових даних для подальшого порівняння з передбаченими результатами.

Крок 5. Обчислюється точність моделі з використанням методу accuracy_score(true_classes, predicted_classes), який приймає справжні та передбачені мітки класів.

Крок 6. Обчислюється чутливість та специфічність моделі за допомогою методів recall_score(true_classes, predicted_classes) та recall_score(true_classes, predicted_classes, pos_label=0), що визначають частку правильно класифікованих позитивних та негативних випадків.

Крок 7. Побудова матриці невідповідностей з використанням методу confusion_matrix(), яка порівнює справжні та передбачені мітки класів.

Крок 8. Відображення матриці невідповідностей, точності, чутливості та специфічності моделі на тестових даних (рис. 4.9).

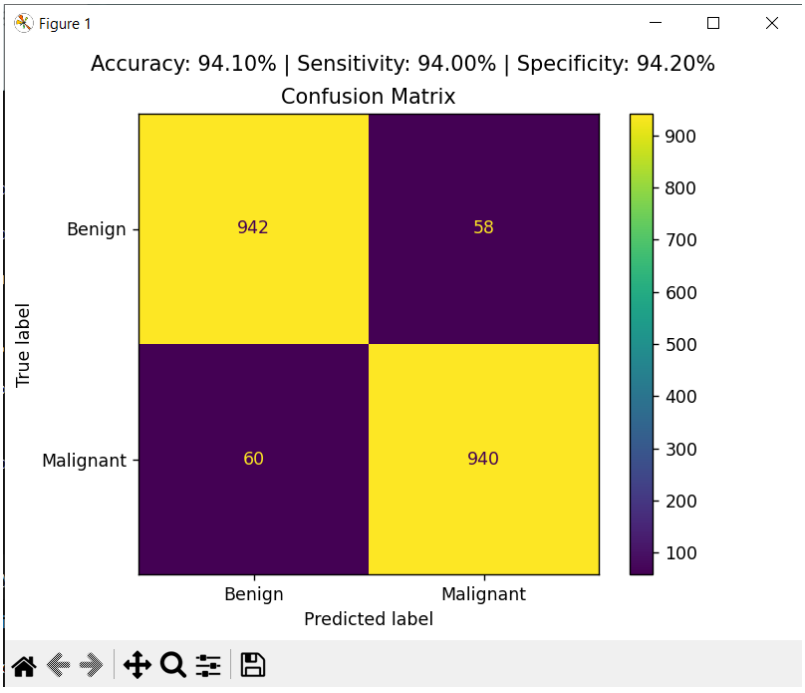


Рисунок 4.9 – Матриця невідповідностей побудованої моделі

Виходячи з рисунку, можна сказати, що модель показала хороші результати. Вона досягла точності 94,10% і правильно класифікувала 942 екземплярів «Benign» та 940 екземплярів «Malignant», але помилково класифікувала 60 екземпляри як «Malignant» та 58 екземпляри як «Benign». Крім того, модель демонструє високу чутливість 94% та специфічність 94,20%, що свідчить про її здатність ефективно розрізняти «Benign» та «Malignant» з оптимальною збалансованістю.

4.5.5 Алгоритм попередньої обробки даних

Крок 1. Завантажаються зображення з навчальної та тестової вибірки даних.

Крок 2. Виконується нормалізація пікселів зображень шляхом масштабування їх значень до діапазону від 0 до 1.

Крок 3. Змінюється розмір всіх зображень до 224x224 пікселів, щоб забезпечити єдиний формат вхідних даних.

Крок 4. На основі підготовлених зображень формуються генератори для навчальних та тестових даних, які автоматично подають дані компактними блоками по 32 зображення з відповідними мітками класів у необхідному форматі. Це дозволяє підвищити ефективність навчання моделі та оптимізувати використання пам’яті.

Крок 5. Сформовані генератори даних повертаються та використовуються для навчання та тестування моделі відповідно.

4.5.6 Алгоритм формування моделі

Крок 1. Завантажується базова модель MobileNet з попередньо навченими вагами ImageNet та встановленням вхідної форми зображення 224×224 з трьома каналами RGB.

Крок 2. Здійснюється заморожування всіх шарів базової моделі шляхом встановлення параметра trainable у значення False для запобігання оновлення їх ваг під час навчання.

Крок 3. Створюється послідовна архітектура моделі з використанням класу Sequential, що включає базову модель MobileNet для автоматичного вилучення високорівневих ознак із вхідних зображень.

Крок 4. Додається шар Flatten для перетворення багатовимірних ознак, отриманих від базової моделі, у одновимірний вектор.

Крок 5. Додається повнозв'язний шар з 128 нейронами та функцією активації ReLU, який формує узагальнене представлення ознак.

Крок 6. Застосовується шар Dropout з коефіцієнтом 0,5 для зменшення ризику перенавчання моделі.

Крок 7. Додається вихідний повнозв'язний шар з одним нейроном та функцією активації «sigmoid» для бінарної класифікації.

Крок 8. Сформована модель повертається для подальшого використання.

4.5.7 Алгоритм навчання та збереження моделі

Крок 1. Виконується попередня обробка даних (підрозділ 4.5.13).

Крок 2. Виконується побудова модифікації моделі MobileNet (підрозділ 4.5.14).

Крок 3. Здійснюється компіляція моделі із застосуванням оптимізатора Adam з коефіцієнтом швидкості навчання 0,0001, функцією втрат «binary_crossentropy» та метрикою оцінювання точності «accuracy».

Крок 4. Проводиться навчання моделі з використанням генератора навчальних даних протягом 5 навчальних епох через метод fit().

Крок 5. Здійснюється збереження тренованої моделі.

На рис. 4.10 зображено процес навчання моделі, як можна побачити, навчання прошло успішно з досягненням точності 94,95% при мінімальних втрат.

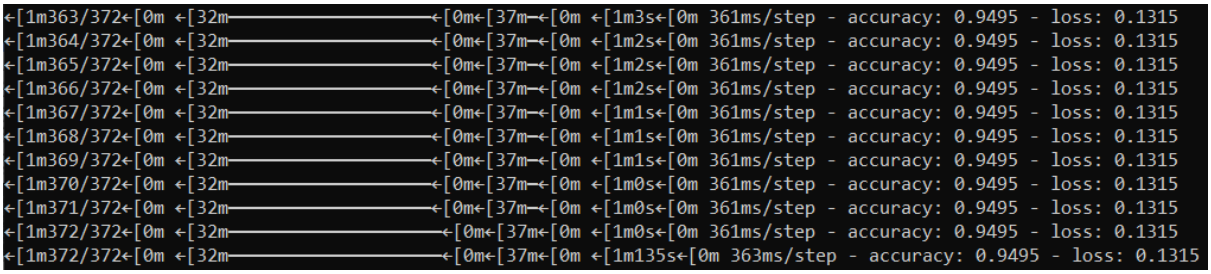


Рисунок 4.10 – Навчання моделі

4.6 Висновки до розділу

У даному розділі було побудовано схему функціонування програмного продукту. Детально описано реалізовані класи програмного забезпечення, серверної частини та моделі нейромережі. Представлено структуру бази даних і зв’язки між сутностями, а також описано алгоритми роботи програми.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Призначення програмного продукту та вимоги до виконання

Призначення застосунку полягає у тому, щоб користувачі могли оперативно отримувати попередню оцінку ризику меланоми, переглядати результати та зберігати їх у форматі PDF тощо.

Мінімальні вимоги до апаратного забезпечення для роботи застосунку такі:

– смартфон на базі операційної системи Android або IOS версії 8.0 або вищої;

– мінімум 200 Мб вільної пам’яті на пристрої;

– наявність підключення до Інтернету.

Розгортання сервера для взаємодії з моделлю потребує таких вимог:

– комп’ютер з процесором Intel Core i5 та операційною системою Windows 10;

– щонайменше 8 Гб RAM та 600 Мб доступного простору на жорсткому диску;

– встановлена мова програмування Python починаючи з версії 3.10.

5.2 Експлуатація програмного продукту

Для локального запуску серверу потрібно відкрити папку «FlaskWebServer» та знайти файл «flask_server.py». Потім через консоль команд потрібно ввести «python flask_server.py», що призведе до запуску сервера за адресою http://192.167.1.5:5000.

Для запуску застосунку потрібно вибрати іконку програми «Melanoma Cancer App». Якщо користувач раніше пройшов авторизацію, відкривається основне меню застосунку (рис. 5.1). У протилежному випадку з’являється екран авторизації (рис. 5.2).

У разі відсутності облікового запису користувач повинен зареєструватися. Щоб зареєструватися в системі, користувач має натиснути на напис «Sign Up». Далі з’являється екран регістрації, де необхідно ввести ПІБ, електронну пошту та пароль.

Рисунок 5.1 – Основне меню програми

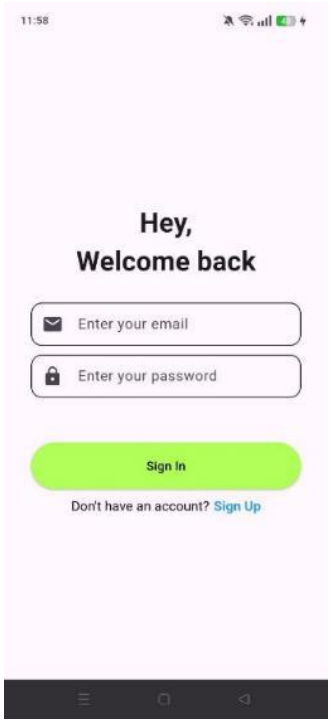


Рисунок 5.2 – Екран авторизації

На рис. 5.3 представлено екран регістрації користувача.

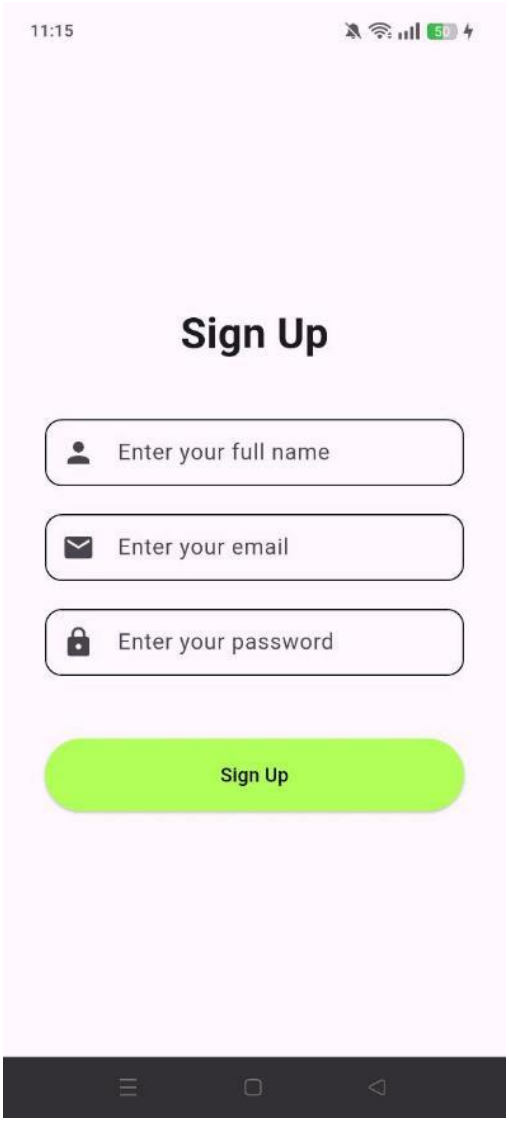


Рисунок 5.3 – Екран регістрації

Якщо під час введення ПІБ, електронної пошти чи пароля були допущені помилки, відображається відповідне сповіщення (рис. 5.4). Якщо всі дані введені вірно, після натискання «Sign Up» з’являється екран авторизації.

Щоб увійти в систему, користувачеві необхідно ввести електронну пошту та пароль. У випадку, коли електронна пошта чи пароль введено неправильно, відображається відповідне сповіщення (рис. 5.5). Коли дані введено вірно і натиснуто «Sign In», користувач потрапляє до основного меню застосунку.

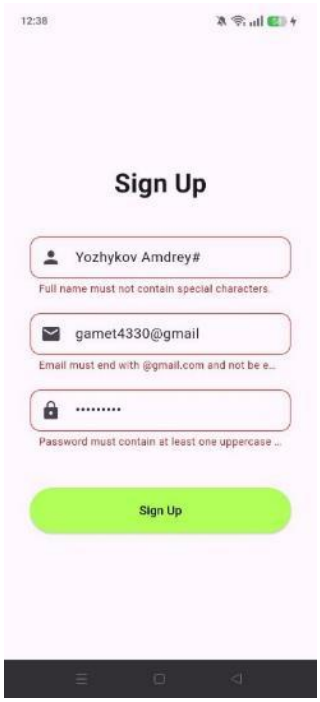


Рисунок 5.4 – Сповіщення щодо помилок при реєстрації

Рисунок 5.5 – Сповіщення щодо помилок під час входу до системи

Після переходу користувача до основного меню застосунку, він має можливість переглянути або оновити інформацію про себе, а також вийти з облікового запису. Для цього потрібно в меню навігації вибрати іконку користувача, після чого відкриється екран профілю (рис. 5.6).

Якщо під час оновлення даних було допущено помилку, відображається відповідне сповіщення (рис. 5.7). Якщо дані введено вірно та натиснуто «Save», то вони успішно зберігаються. Щоб вийти з облікового запису, користувачу необхідно натиснути «Log Out», що призведе до переходу на екран авторизації.

Рисунок 5.6 – Екран профілю

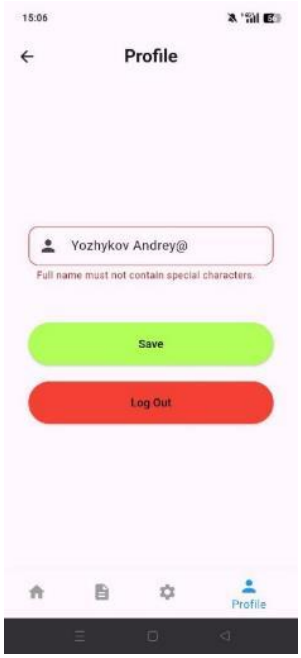


Рисунок 5.7 – Сповіщення щодо помилки при редагуванні даних користувача

Якщо користувач бажає зміни мову або тему застосунку, то потрібно в меню навігації вибрати іконку шестірні. Після цього відкривається екран налаштування, де користувач може вибрати мову або тему (рис. 5.8).

Рисунок 5.8 – Екран налаштування

Щоб виконати аналіз шкіри для визначення ризику меланоми, користувачу необхідно обрати один із двох способів: завантажити зображення з галереї або зробити фото за допомогою камери. Для прикладу буде використано зображення з галереї. Щоб обрати зображення з галереї, потрібно натиснути «Pick up image from gallery» та вибрати конкретне зображення шкіри

для аналізу. Після цього відображається сповіщення щодо успішного виконання аналізу (рис. 5.9).

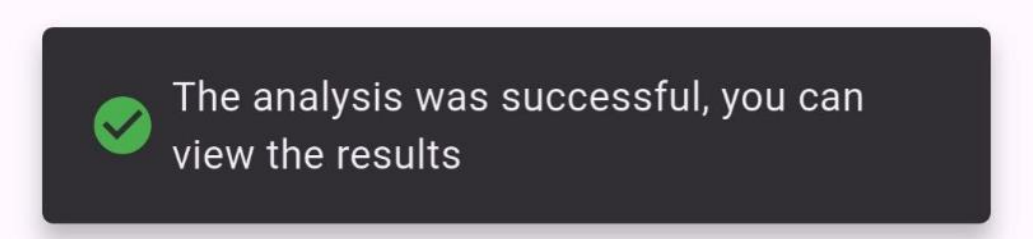


Рисунок 5.9 – Сповіщення щодо успішного виконання аналізу

Для перегляду власних аналізів користувач повинен обрати іконку файлу в меню навігації, де буде відображено перелік його аналізів (рис. 5.10). Для отримання докладної інформації про результати аналізу користувач має обрати конкретний аналіз, що призведе до відображення екрану з повним описом результатів (рис. 5.11).

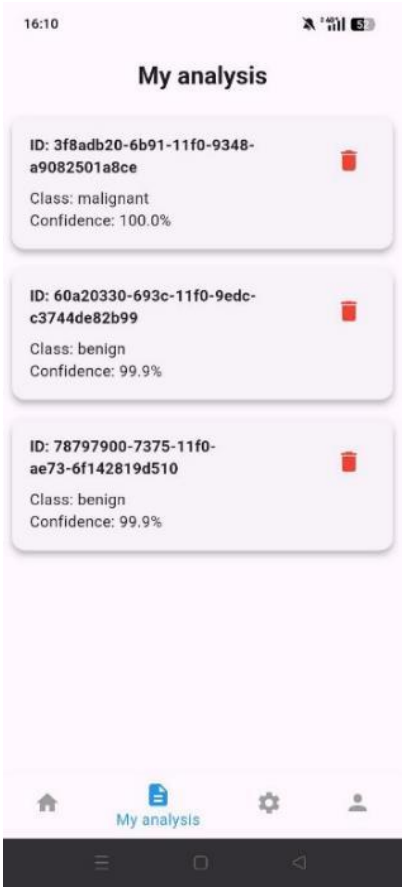


Рисунок 5.10 – Екран перегляду аналізів користувача

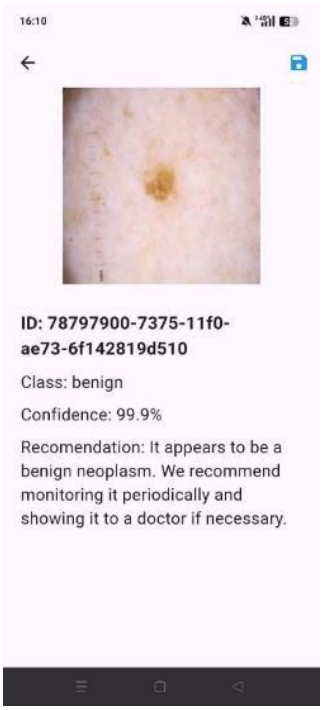


Рисунок 5.11 – Екран перегляду повної інформації про результати аналізу

Окрім цього, користувач може завантажити результати аналізу у форматі PDF (рис. 5.12). При цьому він має опцію видалення конкретного аналізу шляхом вибору відповідної іконки у вигляді кошика.

Рисунок 5.12 – Результати аналізу у форматі PDF

5.3 Проведення тестування програмного продукту та аналіз отриманих результатів

Для оцінки якості застосунку було виконано тестування верстки та функціональне тестування. Для обох етапів тестування були створені чеклісти, що містили інформацію про використані пристрої, список проведених перевірок та результати. На рис. 5.13 та 5.14 представлені чеклісти, що використовувалися під час перевірки верстки та функціональної частини застосунку. Усі тести були успішно пройдені і жоден із пунктів не позначений як «Не пройдено».

Перевірка	Oppo Reno 13	Xiaomi Mi 8 Lite
Перевірити коректне відображення сторінок згідно файлів дизайну	Пройдено	Пройдено
Перевірити наявність головного меню	Пройдено	Пройдено
Перевірити стиль та коректне відображення шрифтів тексту та сам текст	Пройдено	Пройдено
Перевірити орфографію контенту застосунку	Пройдено	Пройдено
Перевірити коректне відображення кнопок, блоків меню та ін.	Пройдено	Пройдено

Рисунок 5.13 – Чекліст, що використовувався під час перевірки верстки

Перевірка	Oppo Reno 13	Xiaomi Mi 8 Lite
Перевірити реєстрацію в систему	Пройдено	Пройдено
Перевірити авторизацію в систему	Пройдено	Пройдено
Перевірити редагування профілю	Пройдено	Пройдено
Перевірити валідацію полів	Пройдено	Пройдено
Перевірити роботу збереження результатів аналізу у формат PDF	Пройдено	Пройдено
Перевірити роботу зміни мови застосунку	Пройдено	Пройдено
Перевірити роботу зміни теми застосунку	Пройдено	Пройдено
Перевірити роботу вибору зображення з галереї	Пройдено	Пройдено
Перевірити роботу створення та вибору зображення за допомогою камери	Пройдено	Пройдено
Перевірити відображення валідаційних повідомлень:		
Некореткний логін	Пройдено	Пройдено
Некоректна пошта	Пройдено	Пройдено
Відхилення доступу до галереї	Пройдено	Пройдено
Пароль, який містить менше 8 символів	Пройдено	Пройдено
Некоректне ПІБ	Пройдено	Пройдено
Відхилення доступу до камери	Пройдено	Пройдено

Рисунок 5.14 – Чекліст, що використовувався під час перевірки функціональної частини застосунку

5.4 Висновки до розділу

У даному розділі було наведено призначення програмного продукту та вимоги до виконання. Також було представлено опис експлуатації програмного продукту та проведено його тестування, а саме: верстки та функціональної частини програми. У результаті всі тести були успішно пройдені і жоден із пунктів не позначено як «Не пройдено».

ВИСНОВКИ

При виконанні дипломної роботи було створено мобільний застосунок для автоматичної оцінки наявності ознак меланоми за зображенням шкіри.

Було розроблено технічне завдання, проаналізовано предметну область та порівняно існуючі аналоги й методи штучного інтелекту з урахуванням їхніх переваг і недоліків.

Було здійснено огляд основних архітектур CNN (VGG16, ResNet50 та MobileNet) та проведено їхнє порівняння для вибору базової моделі. В результаті базовою моделлю було вирішено використовувати MobileNet, на основі якої розроблено модифіковану архітектуру, адаптовану під бінарну класифікацію зображення шкіри з метою виявлення меланоми. Для навчання та тестування моделі було застосовано датасет «Melanoma Cancer Image Dataset» з сайту Kaggle.

Для розробки програмного продукту було визначено мову програмування Dart, платформу Flutter та шаблон проєктування Bloc, а для побудови серверу та нейромережі обрано мову Python із використанням інструментів Flask та TensorFlow. Для організації збереження даних було застосовано базу даних NoSQL із використанням сервісу Firestore.

Було організовано структуру бази даних, описано алгоритми роботи програми та здійснено розробку ключових класів і методів.

Було перевірено коректність роботи верстки та функціональної частини застосунку. У результаті всі тести були успішно пройдені.

Розроблене програмне забезпечення є практично корисним, адже дозволяє користувачам зручно та своєчасно отримувати попередню оцінку наявності ознак меланоми, що сприятиме ранньому зверненню до фахівця та підвищить шанси на успішне лікування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Why is melanoma dangerous [Electronic resource]. – Access mode: https://www.essuntials.com/blogs/journal/why-melanoma-is-dangerous-what-you-need-to-know.

2. Key statistics for melanoma skin cancer [Electronic resource]. – Access mode: https://www.cancer.org/cancer/types/melanoma-skin-cancer/about/key-statistics.html.

3. Ted Alcorn. How to avoid one of the deadliest forms of skin cancer [Electronic resource]. – Access mode: https://www.nytimes.com/article/melanoma-skin-cancer-symptoms-risk.html.

4. Support Vector Machine Algorithm [Electronic resource]. – Access mode: https://www.geeksforgeeks.org/machine-learning/support-vector-machine-algorithm/.

5. K-Nearest Neighbor (KNN) algorithm [Electronic resource]. – Access mode: https://www.geeksforgeeks.org/machine-learning/k-nearest-neighbours/.

6. What is CNN for image processing? [Electronic resource]. – Access mode: https://intelgic.com/what-is-cnn-for-image-processing.

7. SkinVision [Electronic resource]. – Access mode: https://play.google.com/store/apps/details?id=com.rubytribe.skinvision.ac&hl=uk.

8. SkinScreener [Electronic resource]. – Access mode: https://skinscreener.com/language/en/hautkrebs/.

9. Skinive MD [Electronic resource]. – Access mode: https://play.google.com/store/apps/details?id=com.skinive.SkiniveApplication&hl=uk.

10. Muhammad Abdullah Arif. Understanding VGG16: A Powerful Deep Learning Model for Image Recognition [Electronic resource]. – Access mode: https://smuhabdullah.medium.com/understanding-vgg16-a-powerful-deep-learning-model-for-image-recognition-d40b074fd01c.

11. Nitish Kundu. Exploring ResNet50: An In-Depth Look at the Model Architecture and Code Implementation [Electronic resource]. – Access mode: https://medium.com/@nitishkundu1993/exploring-resnet50-an-in-depth-look-at-the-model-architecture-and-code-implementation-d8d8fa67e46f.

12. Abirami Vina. What is ResNet-50 and what is its relevance in computer vision? [Electronic resource]. – Access mode: https://www.ultralytics.com/blog/what-is-resnet-50-and-what-is-its-relevance-in-computer-vision.

13. Abhijeet Pujara. Image classification with MobileNet [Electronic resource]. – Access mode: https://builtin.com/machine-learning/mobilenet.

14. Kushagra Pandya. Understanding MobileNet: Lightweight Deep Learning for Mobile Devices [Electronic resource]. – Access mode: https://medium.com/@p.kushagra22/understanding-mobilenet-lightweight-deep-learning-for-mobile-devices-db171803d8c8?sk=98329f9bf8096f157726445589172f0f.

15. Melanoma Cancer Image Dataset [Electronic resource]. – Access mode: https://www.kaggle.com/datasets/bhaveshmittal/melanoma-cancer-dataset/data.

16. Introduction to the Dart Programming Language: A Beginner’s Guide [Electronic resource]. – Access mode: https://etwinworkshop.medium.com/introduction-to-the-dart-programming-language-a-beginners-guide-30b4fb6d6221.

17. Akshay Badkar. What is Dart Programming? Everything you need to get started [Electronic resource]. – Access mode: https://www.simplilearn.com/what-is-dart-programming-article.

18. Francesco Saviano. Introduction to Python: What it is and why you should use it? [Electronic resource]. – Access mode: https://medium.com/@francesco.saviano87/introduction-to-python-what-it-is-and-why-you-should-use-it-24cf9e606fc6.

19. Jakub Protasiewicz. Python AI: Why is Python so good for machine learning? [Electronic resource]. – Access mode: https://www.netguru.com/blog/python-machine-learning.

20. Bentil Shadrack. Why are relational databases useful [Electronic resource]. – Access mode: https://dev.to/documatic/why-are-relational-databases-useful-307p.

21. Non-Relational databases and their types [Electronic resource]. – Access mode: https://www.geeksforgeeks.org/dbms/non-relational-databases-and-their-types/.

ДОДАТОК А

Технічне завдання

А.1 Підстави до розробки

Виконання дипломної кваліфікаційної роботи здійснюється за темою «Дослідження та програмна реалізація мобільного застосунку для виявлення меланоми за зображенням шкіри» відповідно до наказу №447 від 30 вересня 2025 р. за Національним університетом «Запорізька політехніка».

А.2 Призначення розробки

Призначення застосунку полягає у тому, щоб користувачі могли оперативно отримати попередню оцінку ризику меланоми, переглядати результати та зберігати їх у форматі PDF тощо.

А.3 Вимоги до програмного продукту

А.3.1 Вимоги до функціональних характеристик

Програмний продукт має забезпечувати виконання таких функціональних характеристик:

- реєстрація облікового запису та авторизація в систему;
- перегляд та редагування інформації про користувача;
- вихід з облікового запису;
- зміна теми та мови застосунку;
- виконання аналізу шкіри для виявлення меланоми за допомогою зображення, отриманого з галереї або камери;
- перегляд власних аналізів;
- видалення конкретного аналізу;
- перегляд повної інформації про результати аналізу;
- збереження результатів у форматі PDF.

А.3.2 Вимоги до надійності

Застосунок має ідентифікувати ситуації некоректного введення даних користувачем під час редагування профілю, авторизації та реєстрації, відображаючи відповідні сповіщення щодо помилок. При цьому застосунок під час виконання аналізу шкіри повинен інформувати користувача про вдале проведення аналізу у вигляді сповіщення.

А.3.3 Вимоги до складу та параметрів технічних засобів

Мінімальні вимоги до апаратного забезпечення для роботи застосунку такі:

- смартфон на базі операційної системи Android або IOS версії 8.0 або вищої;
- мінімум 200 Мб вільної пам’яті на пристрої;
- наявність підключення до Інтернету.

А.4 Порядок контролю та приймання

Дипломна робота має узгоджуватися з вимогами, визначеними у пункті А.3, після чого обов’язкового перевіряється керівником. Процедура її прийняття здійснюється відповідно до вимогам, встановлених для робіт магістерського рівня.

ДОДАТОК Б

Текст програми

Б.1 Текст файлу flask_server.py

```
import os
from flask import Flask, jsonify, request
from melanoma_model import MelanomaModel
import uuid

class FlaskServer:
    def __init__(self, model_path="MobileNet.h5"):
        self.app = Flask(__name__)
        self.model_path = model_path
        self.model = MelanomaModel()
        self.setupRoute()
        self.isModelExist()

    def setupRoute(self):
        @self.app.route('/predict', methods = ['POST'])
        def predictMelanoma():
            file = request.files['image']
            print(file.filename)
            filename = f"{uuid.uuid4()}.jpg"
            filepath                                                    =
os.path.join('C:\\Users\\Acer\\Desktop\\MelanomaTest', filename)
            file.save(filepath)
            class_label,       confidence,       recomendation       =
self.model.predict(self.model_path, filepath)
            return  jsonify({"class_label":  class_label,  "confidence":
confidence, "recomendation": recomendation})

    def isModelExist(self):
        if not os.path.exists(self.model_path):
            print("Model didn't find...")
            self.model.trainModel()
        else:
            print("Download a existed model")

    def runServer(self):
        self.app.run(host='0.0.0.0', port=5000)


if __name__ == '__main__':
    server = FlaskServer()
```

```
server.runServer()
```

Б.2 Текст файлу melanoma_model.py

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras import optimizers
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

class MelanomaModel:
    def __init__(self):
        self.class_indices = None

    def createDataGenerators(self, train_dir, test_dir):
        train_datagen = ImageDataGenerator(rescale=1.0/255)
        test_datagen = ImageDataGenerator(rescale=1.0/255)
        train_generator = train_datagen.flow_from_directory(train_dir,
target_size=(224, 224), batch_size = 32,  class_mode='binary')
        test_generator   =   test_datagen.flow_from_directory(test_dir,
target_size=(224, 224), batch_size = 32,  class_mode='binary', shuffle=False)
        return train_generator, test_generator

    def buildModel(self):
        base_model  =  MobileNet(weights='imagenet',  include_top=False,
input_shape=(224, 224, 3))
        for layer in base_model.layers:
            layer.trainable = False
        model = Sequential([
            base_model,
            Flatten(),
            Dense(128, activation='relu'),
            Dropout(0.5),
            Dense(1, activation='sigmoid')
        ])
```

```
            return model

        def trainModel(self):
            train_dir                                                    =
'C:\\Users\\Acer\\source\\repos\\Dyplom_master\\Dyplom_master\\Melanoma\\trai
n'
            test_dir                                                     =
'C:\\Users\\Acer\\source\\repos\\Dyplom_master\\Dyplom_master\\Melanoma\\test
'
            train_generator, test_generator = self.createDataGenerators(
                train_dir,
                test_dir
            )
            model = self.buildModel()

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
loss='binary_crossentropy', metrics=['accuracy'])
            callbacks = [
                EarlyStopping(monitor='loss',                    patience=3,
restore_best_weights=True),
                ModelCheckpoint('melanoma_classifier.keras',
save_best_only=True)
            ]
            history      =      model.fit(train_generator,      epochs=5,
callbacks=callbacks)
            plot_model(model,  to_file='MobileNet.png',  show_shapes=True,
show_layer_names=True, show_layer_activations=True)
            model.save('MobileNet.h5')
            self.checkModelQuality(test_generator, model)

        def checkModelQuality(self, test_generator, model):
            predictions = model.predict(test_generator)
            predicted_classes          =          (predictions          >
0.5).astype("int32").flatten()
            true_classes = test_generator.classes
            conf_matrix = confusion_matrix(true_classes, predicted_classes)
            accuracy = accuracy_score(true_classes, predicted_classes)
            ConfusionMatrixDisplay(conf_matrix,
display_labels=test_generator.class_indices).plot()
            plt.title(f'Confusion Matrix')
            plt.show()

        def predict(self, model_path, image_path):
```

```
            model = tf.keras.models.load_model(model_path)
            processed_img = self.preprocessImage(image_path)
            class_prob = float(model.predict(processed_img)[0][0])
            class_label = ""
            recomendation = ""
            confidence = 0.0
            if class_prob > 0.5:
                class_label = "malignant"
                confidence = round(class_prob * 100, 2)
                recomendation = "There are signs of malignancy. Urgently
consult a doctor for accurate diagnosis and treatment."
            else:
                class_label = "benign"
                confidence = round((1 - class_prob) * 100, 2)
                recomendation = "It appears to be a benign neoplasm. We
recommend monitoring it periodically and showing it to a doctor if necessary."
            return class_label, confidence, recomendation


        def preprocessImage(self, image_path):
            img = tf.keras.preprocessing.image.load_img(
                image_path,
                target_size=(224, 224)
            )
            img_array  =  tf.keras.preprocessing.image.img_to_array(img)  /
255.0
            img_array = np.expand_dims(img_array, axis=0)
            return img_array
```

Б.3 Текст файлу user_model.dart

```
class UserModel {
  final String id;
  final String fullName;
  final String email;
  final String password;

  UserModel({
    required this.id,
    required this.fullName,
    required this.email,
    required this.password,
```

```
  });

  UserModel copyWith({
    String? id,
    String? fullName,
    String? email,
    String? password,
  }) {
    return UserModel(
      id: id ?? this.id,
      fullName: fullName ?? this.fullName,
      email: email ?? this.email,
      password: password ?? this.password,
    );
  }

  Map<String, dynamic> toFirestore() {
    return <String, dynamic>{
      'id': id,
      'fullName': fullName,
      'email': email,
      'password': password,
    };
  }

  factory UserModel.fromFirestore(
    DocumentSnapshot<Map<String, dynamic>> snapshot,
    SnapshotOptions? options,
  ) {
    final data = snapshot.data();
    return UserModel(
      id: data?['id'] as String,
      fullName: data?['fullName'] as String,
      email: data?['email'] as String,
      password: data?['password'] as String,
    );
  }
}
```

Б.4 Текст файлу analysis_model.dart

```
class AnalysisModel {
  final String id;
  final String userId;
  final String className;
  final double confidence;
  final String imageUrl;
  final String recomendation;

  AnalysisModel({
    required this.id,
    required this.userId,
    required this.className,
    required this.confidence,
    required this.imageUrl,
    required this.recomendation,
  });

  AnalysisModel copyWith({
    String? id,
    String? userId,
    String? className,
    double? confidence,
    String? imageUrl,
    String? recomendation,
  }) {
    return AnalysisModel(
      id: id ?? this.id,
      userId: userId ?? this.userId,
      className: className ?? this.className,
      confidence: confidence ?? this.confidence,
      imageUrl: imageUrl ?? this.imageUrl,
      recomendation: recomendation ?? this.recomendation,
    );
  }

  Map<String, dynamic> toFirestore() {
    return <String, dynamic>{
      'id': id,
      'userId': userId,
      'className': className,
```

```
      'confidence': confidence,
      'imageUrl': imageUrl,
      'recomendation': recomendation,
    };
  }

  factory AnalysisModel.fromFirestore(
    DocumentSnapshot<Map<String, dynamic>> snapshot,
    SnapshotOptions? options,
  ) {
    final data = snapshot.data();
    return AnalysisModel(
      id: data?['id'] as String,
      userId: data?['userId'] as String,
      className: data?['className'] as String,
      confidence: data?['confidence'] as double,
      imageUrl: data?['imageUrl'] as String,
      recomendation: data?['recomendation'] as String,
    );
  }
}
```

Б.5 Текст файлу common_progress_indicator.dart

```
class CommonProgressIndicator extends StatelessWidget {
  final double? scale;

  const CommonProgressIndicator({
    super.key,
    this.scale,
  });

  @override
  Widget build(BuildContext context) {
    return Transform.scale(
      scale: scale,
      child: const CircularProgressIndicator(
        color: Colors.blue,
      ),
    );
  }
}
```

Б.6 Текст файлу common_text_field.dart

```
class CommonTextField extends StatelessWidget {
  final TextEditingController? controller;
  final IconData prefixIcon;
  final String hintText;
  final String? errorText;
  final bool obscureText;
  final bool readOnly;
  final Function(String)? onChanged;
  final Function()? onTap;
  final TextInputType? keyboardType;

  const CommonTextField({
    super.key,
    required this.prefixIcon,
    required this.hintText,
    this.errorText,
    this.onChanged,
    this.controller,
    this.onTap,
    this.obscureText = false,
    this.readOnly = false,
    this.keyboardType,
  });

  @override
  Widget build(BuildContext context) {
    return TextField(
      controller: controller,
      onChanged: onChanged,
      onTap: onTap,
      readOnly: readOnly,
      obscureText: obscureText,
      keyboardType: keyboardType,
      decoration: InputDecoration(
        enabledBorder: OutlineInputBorder(
          borderRadius: BorderRadius.circular(12),
        ),
        focusedBorder: OutlineInputBorder(
          borderSide: const BorderSide(color: Colors.blue),
          borderRadius: BorderRadius.circular(12),
```

```
        ),
        focusedErrorBorder: OutlineInputBorder(
          borderSide: const BorderSide(color: Colors.red),
          borderRadius: BorderRadius.circular(12),
        ),
        errorBorder: OutlineInputBorder(
          borderSide: const BorderSide(color: Colors.red),
          borderRadius: BorderRadius.circular(12),
        ),
        contentPadding: const EdgeInsets.all(10),
        errorText: errorText,
        prefixIcon: Icon(prefixIcon),
        hintText: hintText,
      ),
    );
  }
}
```

ДОДАТОК В

Слайди презентації

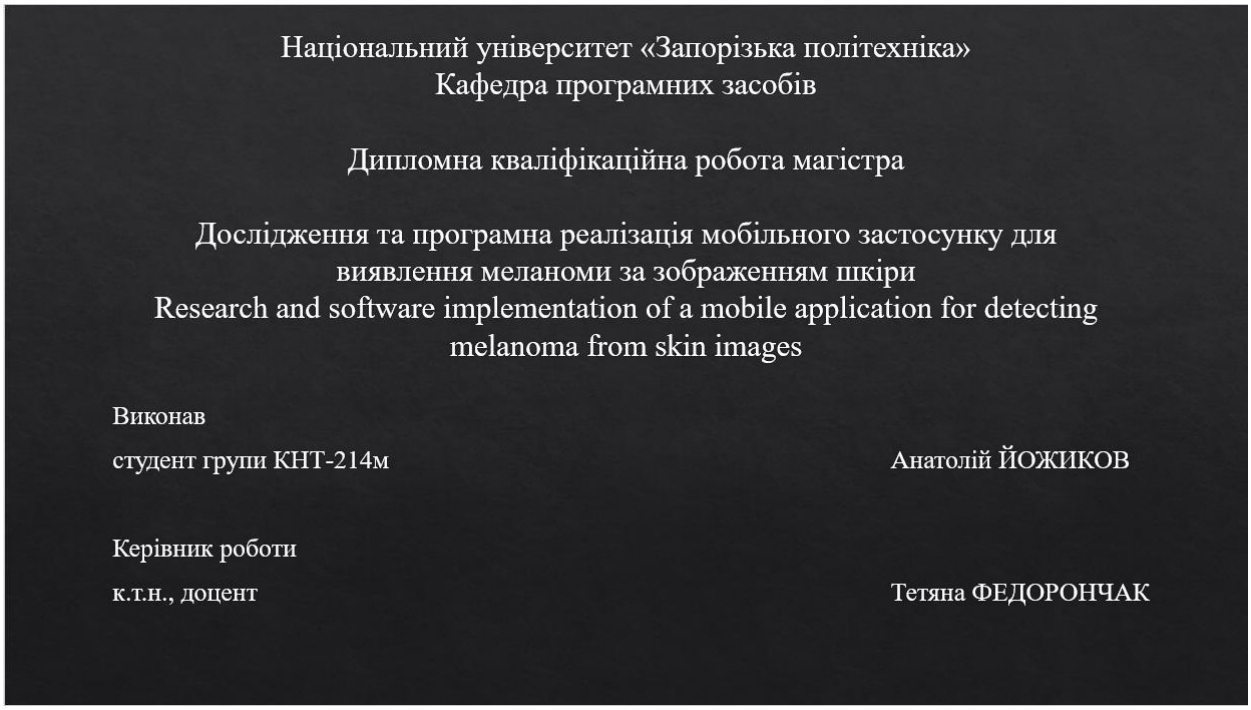


Рисунок В.1 – Слайд 1

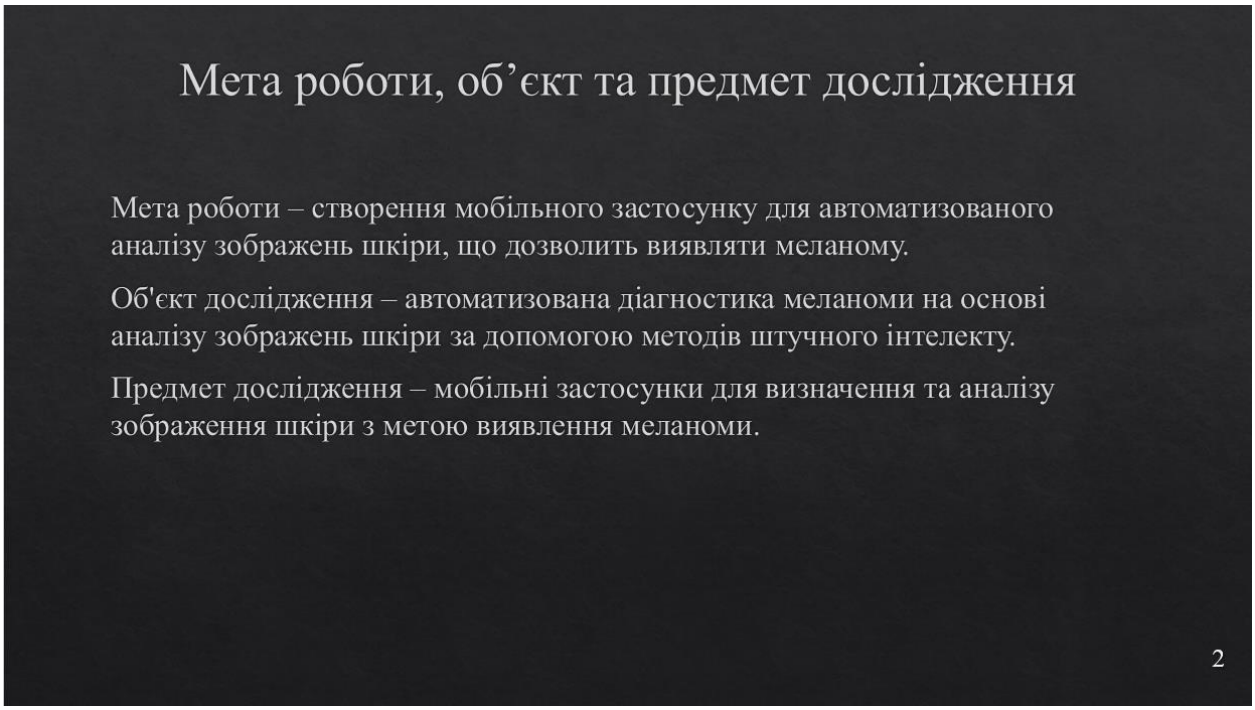


Рисунок В.2 – Слайд 2

Порівняння методів штучного інтелекту

Метод	Ручне вилучення ознак	Ефективність на великих наборах даних	Точність на складних зображеннях	Здатність до узагальнення
SVM	Потрібно	Середня	Середня	Середня
k-NN	Потрібно	Низька	Низька	Низька
CNN	Не потрібно	Висока	Висока	Висока

3

Рисунок В.3 – Слайд 3

Порівняння існуючих аналогів

Параметр порівняння	SkinVision	SkinScreener	SkiniveMD
Доступний в Україні	+	-	-
Зручність використання	+	+	+
Відсутність платної підписки	-	+	-
Нечутливість до якості зображення	-	+	-
Можливість експортувати результати в PDF	-	+	+
Можливість отримання оцінки ризику меланоми та рекомендації	+	+	+

4

Рисунок В.4 – Слайд 4

Архітектура модифікованої моделі

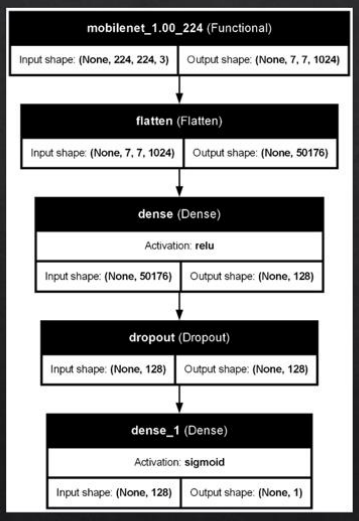


Рисунок 1 – Архітектура модифікованої моделі

5

Рисунок В.5 – Слайд 5

Вибір мови програмування для розробки застосунку

Параметр порівняння	Dart	Swift	Kotlin
Простота розробки	Висока	Середня	Висока
Активність спільноти	Висока	Висока	Середня
Стійкість та захищеність	Висока	Середня	Середня
Швидкодія	Висока	Низька	Середня

6

Рисунок В.6 – Слайд 6

Вибір мови програмування для серверу та нейромережі

Параметр порівняння	C#	Java	Python
Популярність в машинному навчанні та бекенді	Середня	Середня	Висока
Ефективність виконання	Висока	Висока	Низька
Масштабованість	Висока	Висока	Середня
Простота коду	Середня	Середня	Висока
Спільнота та документація	Середня	Середня	Висока

7

Рисунок В.7 – Слайд 7

Вибір сервісу для побудови бази даних

Характеристика	Firestore	Realtime Database
Організація даних	Дані організовані у форматі документів та колекцій	Дані організовані у вигляді JSON дерев
Можливості масштабування	Підтримує гнучке та автоматичне масштабування	Обмежене масштабування
Можливості виконання запитів	Підтримка складних запитів зі складними критеріями фільтрації та сортування	Можливі лише прості запити з базовим сортуванням та фільтрацією

8

Рисунок В.8 – Слайд 8

Рисунок В.9 – Слайд 9

Рисунок В.10 – Слайд 10

Рисунок В.11 – Слайд 11

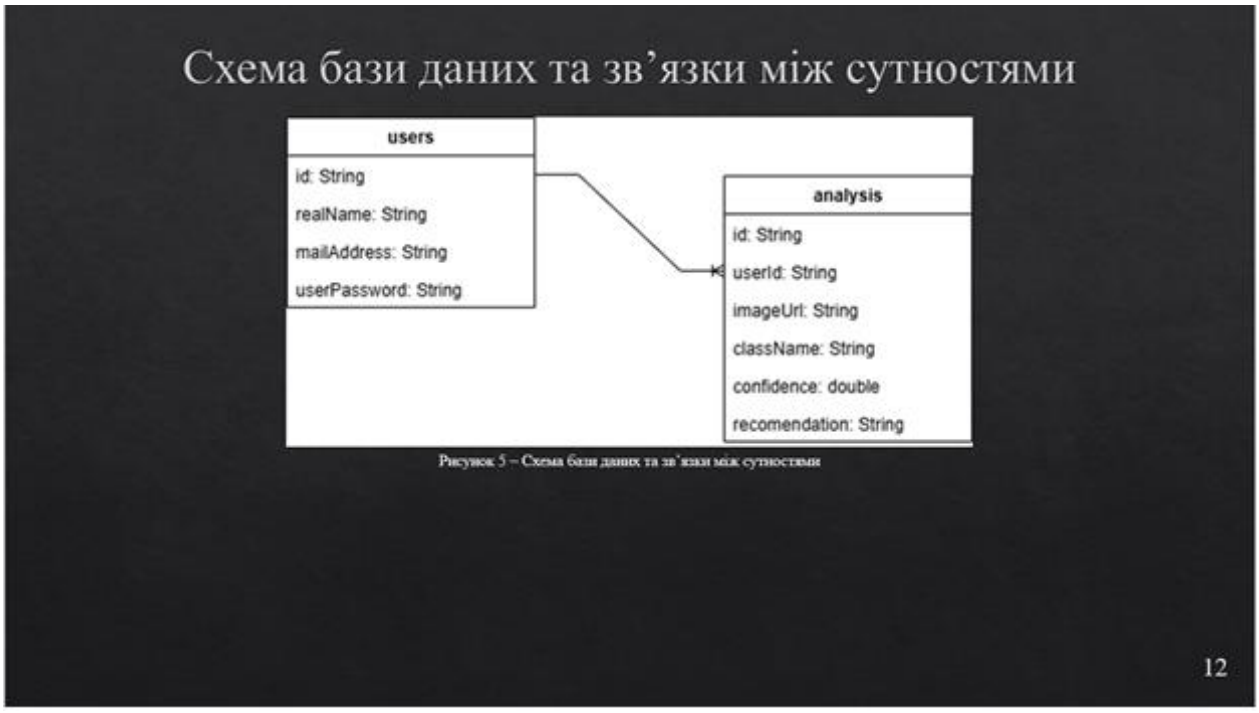


Рисунок В.12 – Слайд 12

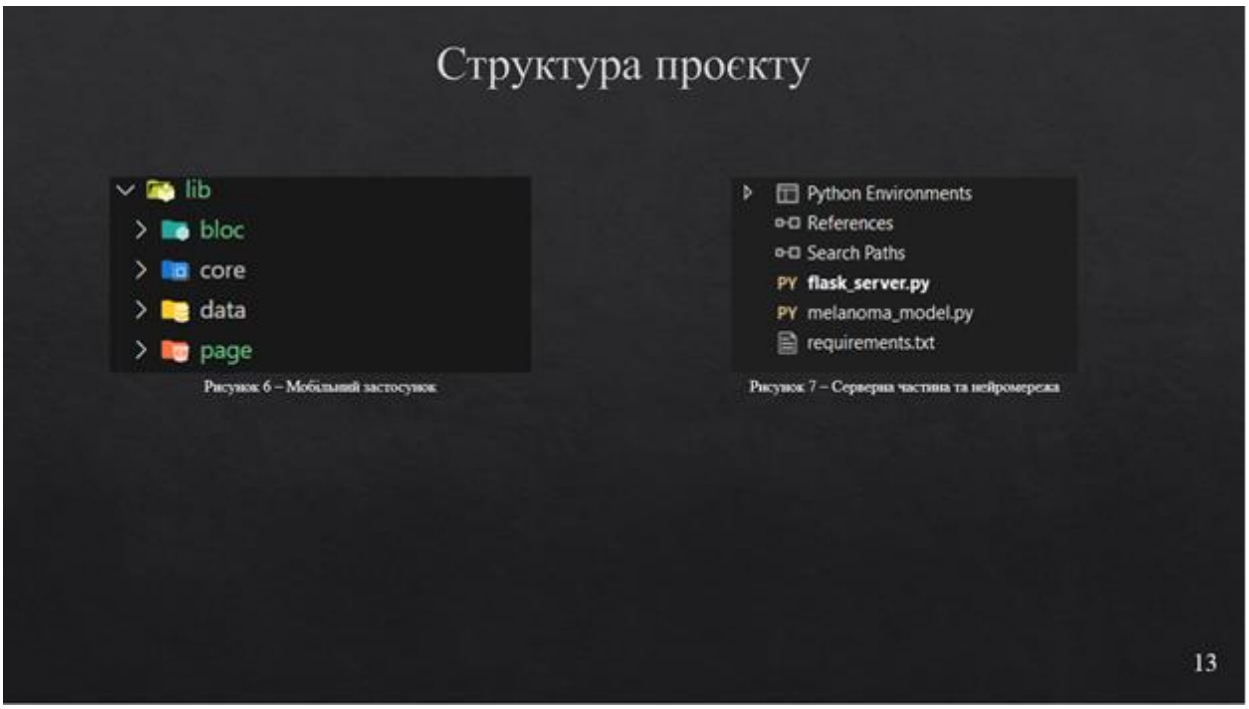


Рисунок 6 – Мобільний застосунок

Рисунок 7 – Серверна частина та нейромережа

Рисунок В.13 – Слайд 13

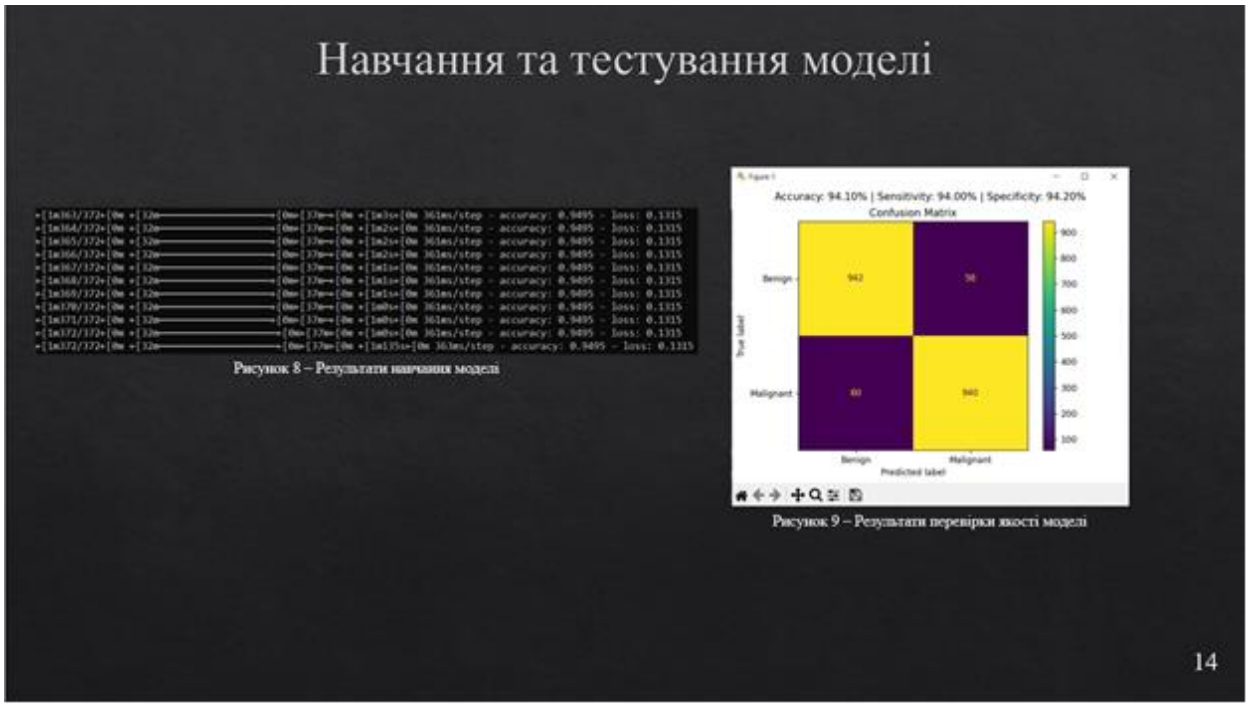


Рисунок 8 – Результати навчання моделі

Рисунок 9 – Результати перевірки якості моделі

Рисунок В.14 – Слайд 14

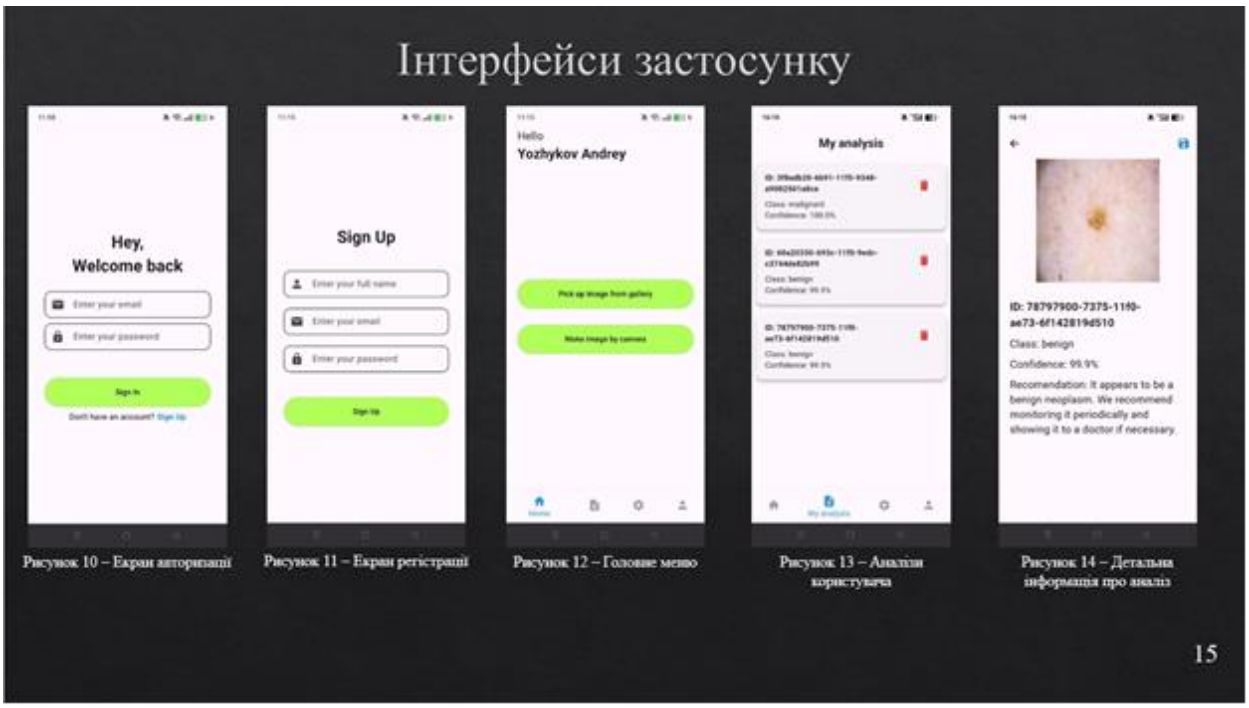


Рисунок 10 – Екран авторизації

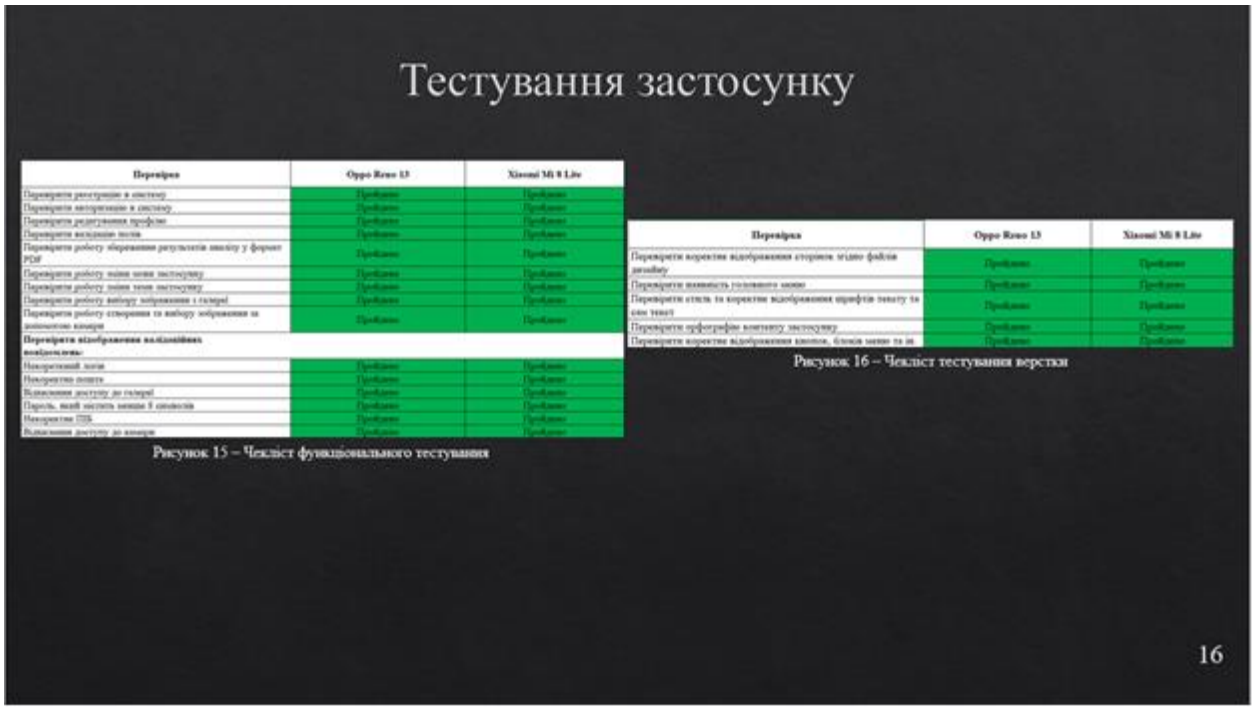
Рисунок 11 – Екран реєстрації

Рисунок 12 – Головне меню

Рисунок 13 – Аналізи користувача

Рисунок 14 – Детальна інформація про аналіз

Рисунок В.15 – Слайд 15



Перевірка	Oppo Reno 13	Xiaomi Mi 8 Lite
Перевірити реєстрацію в системі	Пройдено	Пройдено
Перевірити авторизацію в системі	Пройдено	Пройдено
Перевірити редагування профілю	Пройдено	Пройдено
Перевірити вкладання полів	Пройдено	Пройдено
Перевірити роботу збереження результатів аналізу у форматі PDF	Пройдено	Пройдено
Перевірити роботу нової теми застосунку	Пройдено	Пройдено
Перевірити роботу зміни теми застосунку	Пройдено	Пройдено
Перевірити роботу вибору зображення з галереї	Пройдено	Пройдено
Перевірити роботу створення та вибору зображення за допомогою камери	Пройдено	Пройдено
Перевірити відображення валідаційних повідомлень:		
Некоректний логін	Пройдено	Пройдено
Некоректна пошта	Пройдено	Пройдено
Відмовлення доступу до галереї	Пройдено	Пройдено
Пароль, який містить менше 8 символів	Пройдено	Пройдено
Некоректне ПІБ	Пройдено	Пройдено
Відмовлення доступу до камери	Пройдено	Пройдено

Рисунок 15 – Чеклист функціонального тестування

Перевірка	Oppo Reno 13	Xiaomi Mi 8 Lite
Перевірити коректне відображення сторінок згідно файлів дизайну	Пройдено	Пройдено
Перевірити наявність головного меню	Пройдено	Пройдено
Перевірити стиль та коректне відображення шрифтів тексту та його тексту	Пройдено	Пройдено
Перевірити орфографію контенту застосунку	Пройдено	Пройдено
Перевірити коректне відображення кнопок, блоків меню та ін	Пройдено	Пройдено

Рисунок 16 – Чеклист тестування верстки

Рисунок В.16 – Слайд 16

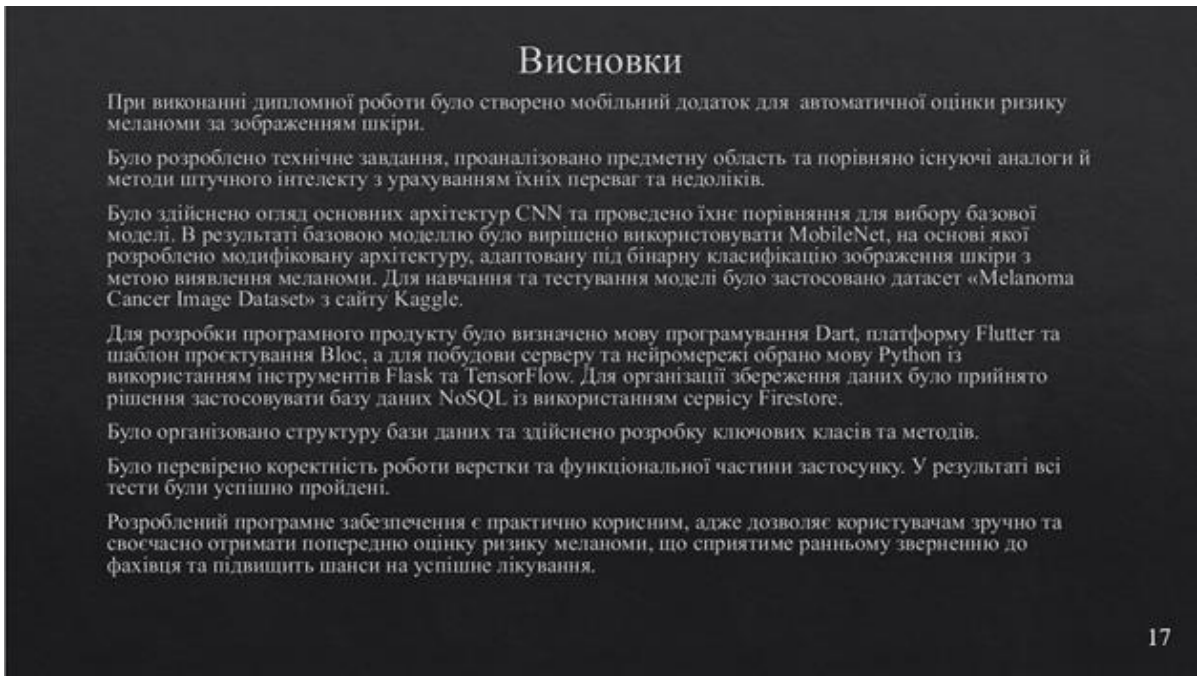


Рисунок В.17 – Слайд 17