

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО
ЗАСТОСУНКУ ДЛЯ МОНІТОРИНГУ ТА ОЦІНКИ СТАНУ ЗДОРОВ'Я НІГТІВ
RESEARCH AND SOFTWARE IMPLEMENTATION OF A MOBILE APPLICATION
FOR MONITORING AND ASSESSING THE CONDITION OF NAILS

Виконав: студент(ка) 2 курсу, групи КНТ-123м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація) Інженерія
програмного забезпечення

ГВОЗДИЛКО Д.О.

(ПРИЗВИЩЕ та ініціали)

Керівник ФЕДОРОНЧАК Т.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

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

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

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 20 24 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТА(КИ)

ГВОЗДИЛКО Данила Олександровича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проекту (роботи) Дослідження та програмна реалізація мобільного застосунку для моніторингу та оцінки стану здоров'я нігтів. Research and Software Implementation of a Mobile Application for Monitoring and Assessing the Condition of Nails.

керівник проекту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна,
 (науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “7” жовтня 2024 року № 398

2. Строк подання студентом проекту (роботи) 2 грудня 2024 року

3. Вихідні дані до проекту (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. Проектування програмного забезпечення. 4. Основні рішення щодо реалізації програмного забезпечення. 5. Експлуатація, тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проекту (роботи)

Розділ	ПРИЗВИЩЕ, 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 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Дослідження проблеми	3–5 тижні	Розділ 2
4	Розробка архітектури програми	6 тиждень	Розділ 3
5	Розробка програми	7–8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програми	9–10 тижні	Розділ 5
7	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	11 тиждень	Додатки
8	Захист роботи	12 тиждень	

Студент (ка)

_____ Данило ГВОЗДИЛКО
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проекту (роботи)

_____ Тетяна ФЕДОРОНЧАК
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
135с., 28 рис., 7 табл., 3 дод., 21 джерел.

ШТУЧНИЙ ІНТЕЛЕКТ, МОНІТОРИНГ, СТАН ЗДОРОВ'Я, НІГТИ,
МОБІЛЬНИЙ ЗАСТОСУНОК.

Об'єкт дослідження – процес збору та аналізу даних про стан здоров'я нігтів за допомогою мобільних пристроїв, що охоплює методи отримання, обробки та інтерпретації даних для раннього виявлення захворювань нігтів та загального моніторингу здоров'я.

Предмет дослідження – програмне забезпечення для мобільного застосунку, яке відстежує та аналізує стан нігтів користувача.

Мета роботи – розробка програмного забезпечення для моніторингу стану нігтів з використанням мобільних пристроїв, з можливістю збереження історії запитів для відстеження змін у стані нігтів користувачів.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, середовище розробки Android Studio, мова програмування Kotlin, фреймворки Jetpack Compose для Android та бібліотеки для обробки зображень.

Результати. Розроблено мобільний застосунок для моніторингу стану нігтів, який аналізує зображення нігтів, визначає можливі ознаки проблем зі здоров'ям нігтів та надає рекомендації щодо догляду.

Висновки. Розроблене програмне забезпечення забезпечує моніторинг стану нігтів, надає користувачам доступ до історії запитів для відстеження змін і може бути використане для профілактики захворювань нігтів.

Галузь використання – сфера здоров'я та догляду за нігтями.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 135 pages, 28 figures, 7 tables, 3 appendices, 21 references.

ARTIFICIAL INTELLIGENCE, MONITORING, HEALTH CONDITIONS, NAILS, MOBILE APPLICATION.

The study aims to collect and analyze data about nail health using mobile devices. This encompasses methods of obtaining, processing, and interpreting data for the early detection of nail diseases and overall health monitoring.

Subject of study – software for a mobile application that monitors and analyzes the condition of the user's nails.

Purpose of the work – to develop software for monitoring nail health using mobile devices, with the capability to save query history for tracking changes in nail condition over time.

Materials, methods, and technical tools: object-oriented programming, the Android Studio development environment, the Kotlin programming language, the Jetpack Compose frameworks for Android, and image processing libraries.

Results. A mobile application has been developed for monitoring nail health, which analyzes nail images, identifies potential signs of nail health issues, and provides care recommendations.

Conclusions. The developed software enables nail condition monitoring, provides users with access to query history for tracking changes, and can be utilized for the prevention of nail diseases.

Field of application – health and nail care.

ЗМІСТ

С.

Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз проблеми та огляд існуючих методів аналізу стану нігтів.....	9
1.1 Огляд існуючих застосунків для оцінки стану здоров'я нігтів.....	9
1.2 Методи аналізу стану нігтів.....	11
1.3 Порівняння аналогів програм для моніторингу стану нігтів.....	12
1.4 Висновки за розділом.....	17
2 Матеріали і методи.....	18
2.1 Збір та обробка даних про стан нігтів.....	18
2.2 Алгоритми обробки зображень для аналізу стану нігтів.....	21
2.3 Використання фреймворка Gemini для моніторингу та оцінки стану здоров'я нігтів.....	24
2.4 Висновки за розділом 2.....	25
3 Проєктування програмного забезпечення.....	26
3.1 Вибір інструментів та середовищ розробки.....	26
3.2 Моделювання програмного забезпечення на основі функціональних вимог... 33	
3.3 Проєктування структури бази даних для зберігання інформації про стан нігтів.....	35
3.4 Висновки за розділом 3.....	36
4 Основні рішення щодо реалізації програмного забезпечення.....	38
4.1 Порядок роботи застосунку для моніторингу стану нігтів.....	38
4.2 Технічна реалізація обробки зображень та взаємодії з штучним інтелектом... 41	
4.3 Висновки за розділом 4.....	41
5 Експлуатація, тестування та експериментальне дослідження програми.....	43
5.1 Опис роботи застосунку.....	43
5.2 Тестування точності аналізу стану нігтів.....	49
5.3 Висновки за розділом 5.....	53
Висновки.....	55
Перелік джерел посилання.....	56
Додаток А Технічне завдання.....	58
Додаток Б Текст програми.....	62
Додаток В Слайди презентації.....	126

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
AI	– Artificial Intelligence;
SDK	– Software Development Kit;
ML	– Machine Learning;
OCR	– Optical Character Recognition;
CNN	– Convolutional Neural Networks.

ВСТУП

Сучасні технології активно проникають у багато сфер повсякденного життя, змінюючи підхід до турботи про здоров'я та естетичний вигляд людини. Однією з найважливіших сфер є догляд за нігтями. Це пов'язано з тим, що стан нігтів може бути індикатором загального стану здоров'я. Своєчасне виявлення проблем з нігтями може запобігти розвитку серйозних захворювань. Водночас догляд за нігтями вимагає спеціальних знань та інструментів і не завжди доступний широкій аудиторії.

З огляду на зростаючу популярність мобільних застосунків для здоров'я, існує потреба у створенні застосунків, які можуть не лише контролювати стан нігтів, але й надавати оцінку здоров'я нігтів та рекомендації. Такий підхід може значно спростити процес діагностики та догляду за нігтями для пересічного користувача.

Метою даної роботи є розробка мобільного застосунку для моніторингу та оцінки здоров'я нігтів, який дозволить користувачам отримати візуальну оцінку стану своїх нігтів, отримати рекомендації щодо покращення їх здоров'я та відслідковувати зміни з часом.

Зі зростанням інтересу до персоналізованого моніторингу здоров'я розробка мобільних застосунків для аналізу здоров'я нігтів є актуальним питанням. Такі інструменти корисні як для широкого кола користувачів, так і для професіоналів, що займаються доглядом за нігтями.

Тому важливість цього дослідження полягає у створенні інноваційних рішень, які допоможуть підвищити якість догляду за нігтями, зробивши процес моніторингу здоров'я доступним і зручним.

1 АНАЛІЗ ПРОБЛЕМИ ТА ОГЛЯД ІСНУЮЧИХ МЕТОДІВ АНАЛІЗУ СТАНУ НІГТІВ

1.1 Огляд існуючих застосунків для оцінки стану здоров'я нігтів

Стрімкий розвиток цифрових технологій у сучасному світі призвів до створення інноваційних рішень для моніторингу здоров'я, зокрема спеціалізованих мобільних застосунків, які допомагають користувачам стежити за своїм фізичним станом. Одним з найцікавіших напрямків є моніторинг та оцінка здоров'я нігтів, які є важливим показником загального стану здоров'я. Оскільки нігті можуть сигналізувати про наявність певних захворювань і дефіциту вітамінів в організмі, застосунки для моніторингу нігтів мають великий потенціал.

Застосунки для моніторингу здоров'я нігтів використовують можливості сучасних мобільних пристроїв, таких як камери смартфонів та алгоритми обробки зображень, для аналізу стану нігтів. Вони дозволяють користувачам проводити самооцінку, надавати поради щодо догляду за нігтями, а в деяких випадках - інформацію про можливі проблеми зі здоров'ям.

Технологія, що лежить в основі таких застосунків, полягає у використанні комп'ютерного зору та AI. Зазвичай програми просять користувачів сфотографувати свої нігті, а потім алгоритми аналізують зміни в кольорі, структурі, формі та товщині нігтя на зображенні. Ці параметри можуть вказувати на проблеми зі здоров'ям, такі як дефіцит заліза, грибкові інфекції або інші захворювання.

Застосунки для моніторингу нігтів мають кілька важливих функцій. По-перше, вони відстежують стан нігтів і допомагають вчасно виявити можливі проблеми. Це особливо корисно для людей, які хочуть запобігти проблемам зі здоров'ям шляхом регулярних самоперевірок. По-друге, такі застосунки можуть надавати рекомендації по догляду за нігтями на основі виявлених змін. Наприклад, якщо застосунок виявляє зміни, які можуть свідчити про авітаміноз, він може запропонувати звернутися до фахівця або покращити раціон харчування.

Багато застосунків також інтегрують функцію реєстрації змін, яка дозволяє користувачам відстежувати динаміку нігтів протягом певного періоду часу. Таким чином можна виявити навіть незначні зміни, які можуть свідчити про розвиток проблем зі здоров'ям.

Важливими є також технічні аспекти таких застосунків. Обробка зображень та аналіз нігтів повинні бути точними, щоб надати користувачеві правильні результати. Більшість сучасних застосунків використовують алгоритми машинного навчання для точного виявлення змін у нігтях. застосунки також повинні мати інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам легко фотографувати свої нігті і без труднощів отримувати результати аналізу.

З точки зору енергоспоживання, програми для аналізу нігтів зазвичай не повинні працювати безперервно у фоновому режимі, що зменшує навантаження на батарею пристрою. Обробка зображень відбувається лише тоді, коли програма дійсно використовується, що робить її енергоефективною.

На ринку вже є кілька таких застосунків, що пропонують різні функціональні можливості для аналізу нігтів. Деякі з них зосереджені виключно на діагностиці, тоді як інші пропонують комплексні рішення, такі як поради по догляду за нігтями та огляд загального стану здоров'я. Той факт, що деякі застосунки є платними, а інші - безкоштовними, дозволяє користувачам обирати варіант, який їм найбільше підходить, залежно від їхніх особливостей та бюджету.

Таким чином, мобільні застосунки для моніторингу та оцінки здоров'я нігтів є перспективним інструментом, який допоможе користувачам самостійно стежити за станом своїх нігтів, отримувати рекомендації по догляду та запобігати проблемам зі здоров'ям. Такі застосунки можуть підвищити обізнаність про власне здоров'я та забезпечити своєчасне реагування на можливі зміни.

1.2 Методи аналізу стану нігтів

Аналіз стану нігтів у мобільних застосунках базується на оцінці візуальних характеристик, що можуть сигналізувати про наявність певних проблем зі здоров'ям. Основними параметрами, які підлягають аналізу, є:

- колір. Зміни кольору нігтя можуть вказувати на проблеми з кровообігом, анемію або грибкові інфекції. Блідість чи жовтуватий відтінок часто свідчать про анемію або інші системні захворювання;
- текстура. Нерівності або зміни текстури нігтя можуть бути ознакою травм, дефіциту вітамінів або мікроелементів, таких як залізо чи цинк;
- форма. Вигнута або змінена форма нігтів може свідчити про серйозні захворювання, включаючи серцево-судинні проблеми або хронічні хвороби легенів;
- товщина. Товщі нігті можуть бути пов'язані з псоріазом або грибковими інфекціями.

Для аналізу цих параметрів використовуються різні математичні та обчислювальні методи:

- алгоритми обробки зображень. Використовуються для виділення контурів нігтя, аналізу його кольору та виявлення змін у текстурі. Обробка зображень дозволяє точно оцінити візуальні зміни нігтя з часом;
- методи машинного навчання. Для автоматичного розпізнавання патологій застосовуються нейронні мережі, які здатні навчатися на великій кількості зображень здорових та пошкоджених нігтів. Це дозволяє застосункам пропонувати точну діагностику, аналізуючи структуру, колір та форму нігтів;
- аналіз текстур. Цей метод базується на статистичному аналізі пікселів зображення для оцінки грубості або однорідності текстури нігтів. Різні патології нігтів можуть проявлятися як зміни в текстурі, що добре виявляються за допомогою алгоритмів текстурного аналізу;

- фрактальний аналіз. Застосовується для оцінки складних візерунків на поверхні нігтя, оскільки він дозволяє визначити нерегулярні зміни форми або структури, які можуть бути симптомом патології;

- методи класифікації. Алгоритми, такі як дерева рішень або методи опорних векторів (SVM), використовуються для класифікації зображень нігтів на різні категорії стану: здорові, пошкоджені, з патологіями;

- аналіз геометричних характеристик. Включає вимірювання співвідношень між довжиною, шириною та кривизною нігтя. Ці параметри можуть змінюватися при таких станах, як коїлоніхія (ложкоподібні нігті) або "барабанні палички", що можуть свідчити про проблеми з легенями або серцем.

1.3 Порівняння аналогів програм для моніторингу стану нігтів

У цьому розділі буде проведено детальний аналіз популярних застосунків, які пропонують різні функції для моніторингу здоров'я нігтів. Порівняння таких застосунків дозволить виявити їхні сильні та слабкі сторони, а також визначити, які з них найкраще відповідають потребам користувачів. Розглядаючи ці застосунки, ми зосередимося на їхніх основних функціях, цільовій аудиторії, можливостях аналізу стану нігтів та інтеграції з медичними показниками, що допоможе зрозуміти їхнє місце в сучасному цифровому середовищі догляду за здоров'ям.

1.3.1 Застосунок Nail Keeper

Застосунок Nail Keeper пропонує можливість відстежувати стан нігтів за допомогою регулярного фотографування. Головний екран застосунку наведений на рис. 1.1. Основна мета застосунку — допомогти користувачам моніторити зміни в зовнішньому вигляді нігтів з часом. Це дозволяє користувачам документувати будь-які зміни кольору, текстури або форми нігтів, що можуть свідчити про можливі проблеми зі здоров'ям. На жаль, застосунок не пропонує

автоматизованого аналізу або діагностики хвороб за допомогою фото, а лише сприяє профілактичному контролю стану нігтів. Застосунок доступний в Google Play і орієнтований на самостійне спостереження за здоров'ям нігтів [1].



Рисунок 1.1 – Головний екран застосунку Nail Keeper

1.3.2 Застосунок Healthy Nails

Healthy Nails є застосунком, який можна знайти на Amazon, і він пропонує освітню інформацію про правильний догляд за нігтями. Головний екран застосунку наведений на рис. 1.2. Користувачі можуть отримати поради щодо підтримки здорових нігтів, дізнатися про різні захворювання нігтів та методи їх лікування. Проте застосунок не пропонує діагностичних інструментів або можливості аналізу фотографій нігтів. Основна його функція — це надання базових порад щодо підтримки гарного стану нігтів [2].



Рисунок 1.2 – Головний екран застосунку Healthy Nails

1.3.3 Застосунок Frenchies Modern Nail Care

Цей застосунок відрізняється тим, що орієнтований на управління послугами салону. Головний екран застосунку наведений на рис. 1.3. Користувачі можуть за допомогою застосунку записатися на процедури в мережі салонів Frenchies Modern Nail Care, переглядати послуги та керувати своїми візитами. Хоча він пов'язаний з доглядом за нігтями, застосунок не має функцій для моніторингу здоров'я або автоматизованого аналізу стану нігтів. Застосунок фокусується на сервісному обслуговуванні клієнтів, а не на діагностиці [3].

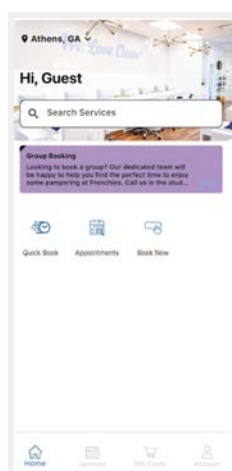


Рисунок 1.3 – Головний екран застосунку Frenchies Modern Nail Care

1.3.4 Застосунок Anemia Detection App

Цей застосунок створений для виявлення анемії через аналіз нігтьових лож за допомогою фотографій. Один з головних екранів застосунку наведено на рис. 1.4. Використовуючи штучний інтелект, застосунок може визначати рівень гемоглобіну у крові, що допомагає виявляти анемію. Оскільки один із проявів анемії — це зміни в кольорі нігтів (блідість, ламкість), застосунок допомагає оцінити стан здоров'я користувача на основі зображень нігтів. Хоча він більше сфокусований на виявленні анемії, його функціонал показує можливості фотоаналізу для оцінки здоров'я за станом нігтів [4].

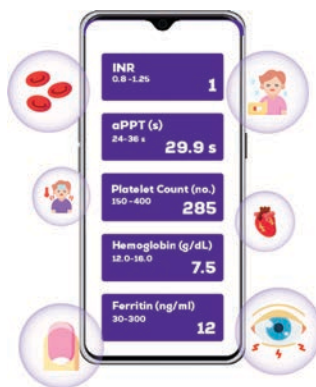


Рисунок 1.4 - Головний екран застосунку Anemia Detection App

1.3.5 Порівняльний аналіз

У табл. 1.1 проводиться порівняння застосунків для моніторингу та догляду за нігтями, підкреслюючи їх відмінності за ключовими функціями. Ці застосунки пропонують різні можливості для аналізу стану нігтів та їх здоров'я, що дозволяє користувачам обрати найбільш підходящий варіант відповідно до своїх потреб. У таблиці представлено основні характеристики кожного застосунку, що допоможе в оцінці їхньої ефективності та корисності. Здійснюючи порівняння, ми можемо зрозуміти, як технології можуть полегшити догляд за здоров'ям нігтів.

Таблиця 1.1 – Порівняння застосунків для відстеження здоров'я нігтів

Критерій	Nail Keeper	Healthy Nails	Frenchies Modern Nail Care	Anemia Detection App
Основна функція	Моніторинг змін нігтів через регулярні фото	Інформація про догляд за нігтями та профілактику захворювань	Керування візитами до салонів манікюру та вибір послуг	Аналіз нігтів для виявлення анемії через штучний інтелект
Можливість аналізу стану здоров'я	Відсутня автоматична діагностика, можливість моніторингу змін	Відсутня, застосунок зосереджений на інформації про догляд за нігтями	Відсутня, зосереджується на обслуговуванні в салонах, а не на здоров'ї	Аналіз здоров'я через фото нігтів, лише для виявлення анемії
Цільова аудиторія	Люди, які хочуть самостійно відстежувати зміни в нігтях	Користувачі, які шукають поради по догляду за нігтями	Користувачі, які бажають отримувати догляд у салонах	Люди, які перевіряють нігті для виявлення анемії
Фотографування нігтів	Відстеження змін, без автоматичного аналізу	Відсутнє	Відсутнє	Аналіз фото нігтів з метою діагностики анемії
Інтеграція з медичними показниками	Відсутня	Відсутня	Відсутня	Орієнтований на діагностику анемії через показники
Додаткові функції	Тільки базовий моніторинг стану нігтів	Освітні статті та поради з догляду	Бронювання послуг у салонах та управління профілем	Аналіз для визначення анемії, не охоплює інші захворювання
Тип застосунку	Самостійне відстеження та моніторинг	Освітній застосунок	Застосунок для запису до салонів краси	Застосунок для діагностики на основі зображень

1.4 Висновки за розділом

Отже, аналіз існуючих мобільних застосунків для моніторингу та оцінки здоров'я нігтів свідчить про різноманітність можливостей, які пропонують сучасні технології. Кожен з розглянутих застосунків має свої унікальні особливості та задовольняє різні потреби користувачів, починаючи від простого моніторингу і закінчуючи складним аналізом для виявлення можливих захворювань.

Такі інноваційні рішення значно підвищують обізнаність про власне здоров'я та дозволяють вчасно реагувати на зміни у стані нігтів. Проте варто зазначити, що деякі застосунки все ще потребують доопрацювання з точки зору функціональності та точності діагностичних інструментів. Тому, обираючи відповідний застосунок, користувачі повинні враховувати не лише його основні функції, а й свої особисті потреби у сфері охорони здоров'я.

2 МАТЕРІАЛИ І МЕТОДИ

У сучасному суспільстві здоров'я нігтів є важливим показником загального стану здоров'я організму, а моніторинг змін у нігтях може забезпечити раннє виявлення різних захворювань і певних авітамінозів. Мобільні застосунки для моніторингу здоров'я нігтів можуть надати користувачам можливість регулярно оцінювати стан своїх нігтів, отримувати поради щодо догляду за ними та попереджати про потенційні проблеми. Дослідження в цій галузі зосереджені на використанні сучасних технологій, таких як аналіз зображень, штучний інтелект та автоматизовані системи обробки зображень, щоб надати користувачам точні та корисні інструменти для моніторингу здоров'я нігтів. У цьому розділі описано матеріали та методи, що використовуються для розробки таких мобільних застосунків, зокрема алгоритми обробки зображень та збору даних про стан нігтів.

2.1 Збір та обробка даних про стан нігтів

2.1.1 Фотографування нігтів за допомогою камери

Фотографування нігтів - один з ключових етапів збору даних для мобільних застосунків, які відстежують стан нігтів. Цей процес створює високоякісні зображення, які обробляються за допомогою алгоритмів комп'ютерного зору для виявлення потенційних проблем зі здоров'ям [7]. Нижче ми розглянемо ключові аспекти цього процесу:

а) якість зображення. Важливо, щоб камера смартфона робила чіткі та детальні фотографії. Це дозволить алгоритмам точно аналізувати стан нігтів. Фактори, що впливають на якість зображення:

1) роздільна здатність камери. Більшість сучасних смартфонів мають камери з високою роздільною здатністю, що дозволяє зробити детальні знімки поверхні нігтів;

2) освітлення. Для отримання чіткого зображення важливо використовувати достатнє природне або штучне освітлення;

3) фокусування та стабілізація. Камера повинна фокусуватися безпосередньо на нігтях, щоб уникнути розмиття. У деяких смартфонах вбудовані функції стабілізації зображення для уникнення тремтіння рук під час зйомки [8] - [9];

б) процедура фотографування. Щоб забезпечити максимальну точність даних, користувачеві можуть надаватися інструкції щодо правильного фотографування:

1) кут зйомки. Нігті повинні бути зняті під прямим кутом до камери, щоб уникнути спотворень;

2) чистота нігтів. Перед фотографуванням користувачеві рекомендується очистити нігті від лаку чи бруду, оскільки це може заважати точному аналізу;

3) позиція пальців. Застосунок може використовувати наочні інструкції або рамки на екрані, щоб допомогти користувачу правильно розмістити нігті в кадрі [10];

в) безпека даних. Для захисту конфіденційності користувачів всі зображення мають бути зашифровані, зберігатися у безпечних базах даних і використовуватися виключно для медичних або аналітичних цілей [11] - [12].

2.1.2. Google ML Kit для розпізнавання об'єктів на фото

Google ML Kit — це набір інструментів (SDK) від Google, призначений для розробників мобільних застосунків, що дозволяє використовувати можливості машинного навчання (ML) на мобільних пристроях без необхідності постійного підключення до Інтернету. Це спрощує інтеграцію складних моделей машинного навчання в мобільні застосунки для виконання різних завдань, таких як обробка зображень, тексту, мови та розпізнавання об'єктів.

Основні можливості Google ML Kit:

- розпізнавання тексту. Цей модуль дозволяє витягувати текст із зображень, що може бути корисним для автоматизації введення даних або для інтерпретації рукописних записів;
- розпізнавання об'єктів і маркування зображень. Це дозволяє визначати об'єкти на зображеннях, зокрема людей, тварин, транспортні засоби та багато іншого. Водночас, маркування зображень може автоматично ідентифікувати і класифікувати об'єкти на зображенні, що є корисним для застосунків, які потребують розпізнавання сцен або контекстів;
- розпізнавання обличчя. API для розпізнавання обличчя може ідентифікувати обличчя на зображенні або відео в реальному часі. Це може використовуватися для безпеки або анімованих аватарів;
- сегментація зображень. Ця функція дозволяє відокремлювати передній план зображення від фону, що корисно для застосунків, які потребують акцентування певних частин зображення, наприклад, рук або нігтів;
- розпізнавання мовлення та переведення тексту. Дозволяє перетворювати мовлення в текст або перекладати текст із різних мов в реальному часі.

Google ML Kit розроблений для спрощення розробки мобільних застосунків із вбудованими можливостями машинного навчання, що працюють на пристрої. Це дозволяє створювати більш інтерактивні та розумні застосунки, такі як програми для обробки зображень, виявлення об'єктів, розпізнавання тексту або мови. Завдяки цим функціям розробники можуть підвищити ефективність та користувацький досвід застосунків, забезпечуючи швидку обробку даних без необхідності у підключенні до хмарних сервісів. ML Kit можна використовувати в різних галузях: від охорони здоров'я до розваг, включаючи автоматизацію, покращення безпеки, та багато інших завдань, що потребують аналізу зображень чи тексту [5].

2.2 Алгоритми обробки зображень для аналізу стану нігтів

Алгоритми обробки зображень для аналізу стану нігтів можуть варіюватися залежно від конкретних цілей (наприклад, виявлення захворювань, оцінка стану здоров'я, визначення кольору або текстури тощо). Основні алгоритми та підходи, які можуть бути використані для цієї мети, наведені у табл. 2.1 [7-10].

Таблиця 2.1 – Алгоритми обробки зображень

Алгоритм	Призначення	Методи	Опис
Сегментація	Виділення об'єктів на зображеннях	Нейронні мережі	Використання моделей для сегментації об'єктів на зображеннях, таких як нігті або шкіра
Глибоке навчання	Інтеграція даних з різних джерел	Трансформерні моделі	Використання складних моделей для аналізу та генерації тексту та зображень
Мультимодальні моделі	Обробка тексту та зображень одночасно	Моделі для обробки тексту та зображень	Інтеграція інформації з тексту та зображень для більш глибокого контексту
Метод граничного детектування (Edge Detection)	Виявлення контурів нігтів на зображенні	Оператор Кенні, оператор Собеля	Використовується для визначення контурів нігтя, допомагає виявляти дефекти. Оператори Кенні і Собеля аналізують зміни інтенсивності пікселів

Кінець таблиці 2.1

Алгоритм	Призначення	Методи	Опис
Алгоритми класифікації на основі CNN	Оцінка стану нігтів, виявлення грибкових інфекцій або пошкоджень	Згорткові нейронні мережі (CNN)	CNN добре підходять для аналізу медичних зображень, зокрема для виявлення захворювань, таких як грибкові інфекції або структурні пошкодження нігтів
Моделі кластеризації K-Means	Аналіз кольорових аномалій нігтів	K-Means	Використовується для сегментації зображення на кольорові сегменти, що дозволяє виявляти зміни кольору або плями на нігтях, які можуть свідчити про хворобу

Google ML Kit є потужним набором інструментів для мобільних розробників, який надає можливість інтеграції функцій машинного навчання у мобільні застосунки. У ML Kit реалізовано низку алгоритмів, що забезпечують різноманітні можливості обробки зображень. Одним з ключових компонентів є функція оптичного розпізнавання символів (OCR), яка використовує алгоритми для виявлення та розпізнавання тексту на зображеннях. Крім того, ML Kit підтримує сегментацію зображень за допомогою глибоких нейронних мереж, що дозволяє виділяти об'єкти, такі як нігті або шкіра. Виявлення облич є ще однією важливою функцією, що дозволяє розробникам реалізовувати різноманітні

рішення для аналізу зображень, використовуючи передові алгоритми комп'ютерного зору [5].

Gemini представляє собою новий фреймворк штучного інтелекту, який фокусується на інтеграції різноманітних моделей машинного навчання для генерації контенту та обробки даних. Цей фреймворк використовує складні трансформерні моделі, що дозволяють виконувати завдання глибокого навчання з високою точністю. Однією з ключових особливостей Gemini є його здатність обробляти текст і зображення одночасно, що реалізується за рахунок використання мультимодальних моделей. Це забезпечує більш глибоке розуміння контексту та покращує результати аналізу. Алгоритми, реалізовані в Gemini, дають змогу розробникам створювати інтуїтивні та багатофункціональні рішення, що відповідають вимогам сучасних користувачів у різних сферах застосування [6].

OCR використовується для виявлення і розпізнавання тексту на зображеннях. Google ML Kit реалізує OCR для сканування документів і зображень, що містять текст, і може ідентифікувати різні шрифти та стилі написання. Це може бути корисним для збору інформації з медичних карток або етикеток.

Глибоке навчання застосовується для складних завдань класифікації та виявлення об'єктів у зображеннях. У Google ML Kit можна використовувати попередньо навчені моделі глибокого навчання для аналізу зображень, що дозволяє вирішувати завдання, пов'язані з виявленням і класифікацією медичних станів, таких як грибкові інфекції.

Алгоритми класифікації на основі CNN використовуються для виявлення та класифікації об'єктів на зображеннях, зокрема для медичних застосувань.

2.3 Використання фреймворка Gemini для моніторингу та оцінки стану здоров'я нігтів

Gemini — це інноваційний фреймворк штучного інтелекту, розроблений компанією Google, що інтегрує різноманітні моделі машинного навчання для генерації контенту та обробки даних. Gemini спроектовано для забезпечення високої точності та гнучкості при виконанні завдань, пов'язаних з обробкою тексту та зображень. Однією з основних особливостей Gemini є його здатність працювати з мультимодальними даними, що дозволяє здійснювати комплексний аналіз інформації з різних джерел. Це особливо важливо для проектів, що потребують синергії між текстовими та візуальними даними, таких як моніторинг здоров'я нігтів.

Gemini може використовувати CNN для класифікації стану нігтів, зокрема для виявлення дефектів або грибкових інфекцій, що вимагає обробки та аналізу візуальної інформації.

У контексті дослідження та програмної реалізації мобільного застосунку для моніторингу та оцінки стану здоров'я нігтів, Gemini може бути використаний для реалізації кількох ключових функцій. По-перше, фреймворк дозволяє проводити глибокий аналіз зображень нігтів, використовуючи алгоритми комп'ютерного зору для виявлення аномалій, змін кольору або текстури, що можуть свідчити про проблеми зі здоров'ям. Зокрема, Gemini може бути застосований для реалізації моделей глибокого навчання, які можуть класифікувати стан нігтів (здорові, уражені, з ознаками захворювань) на основі зображень, зроблених користувачами.

По-друге, Gemini може інтегрувати текстову інформацію, наприклад, дані про симптоми, рекомендації лікарів або освітні матеріали про догляд за нігтями. Це дозволяє створити комплексний інтерфейс, де користувачі отримують не лише візуальний аналіз своїх нігтів, а й корисну інформацію для поліпшення їхнього здоров'я. Мультимодальні можливості Gemini сприяють покращенню взаємодії

користувачів із застосунком, адже на основі візуальних та текстових даних можна створювати персоналізовані рекомендації та графіки прогресу [6].

2.4 Висновки за розділом 2

У цьому розділі було представлено комплексний огляд матеріалів та методів, що використовуються для розробки мобільного застосунку для моніторингу та оцінки стану здоров'я нігтів. Зокрема, акцентовано увагу на важливості збору та обробки даних, що забезпечує можливість виявлення змін у стані нігтів, які можуть свідчити про різні захворювання або недоліки вітамінів.

Фотографування нігтів, як ключовий етап, підкреслює необхідність високої якості зображень, адже саме вони слугують основою для подальшого аналізу. Використання Google ML Kit забезпечує точність у розпізнаванні об'єктів, сегментації та аналізу тексту, що є невід'ємною частиною сучасних мобільних застосунків. Завдяки цим технологіям розробники можуть впроваджувати інтуїтивно зрозумілі інтерфейси, що сприяють підвищенню користувацького досвіду.

Фреймворк Gemini, що поєднує в собі алгоритми машинного навчання та мультимодальні моделі, відкриває нові можливості для комплексного аналізу даних, дозволяючи інтегрувати текстову та візуальну інформацію. Це забезпечує глибше розуміння стану здоров'я нігтів і допомагає користувачам отримувати персоналізовані рекомендації.

Таким чином, використання сучасних технологій, таких як обробка зображень, штучний інтелект та автоматизовані системи, не лише покращує точність діагностики, а й робить мобільні застосунки для моніторингу здоров'я нігтів більш доступними і корисними для широкого кола користувачів. У подальших розділах дослідження буде представлено практичну реалізацію описаних методів і технологій.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір інструментів та середовищ розробки

При розробці мобільного застосунку для відстеження стану нігтів вибір інструментів відіграє важливу роль у забезпеченні ефективності та якості розробки. Для проекту було обрано офіційне середовище розробки Android Studio, оскільки воно надає повний набір інструментів для створення, тестування та оптимізації застосунків для Android.

Android Studio-провідна платформа, рекомендована Google для розробки застосунків для Android, що пропонує розробникам широкий спектр функцій, від підтримки новітніх мов програмування, таких як Kotlin і Java, до вбудованих емуляторів для тестування застосунків на різних пристроях.

Головним аргументом на користь Android Studio було її зручність, інтеграція з Android SDK і прекрасна можливість роботи з призначеним для користувача інтерфейсом і управління проектами. За допомогою цієї IDE можна не тільки прискорити процес розробки, але і забезпечити стабільність і продуктивність кінцевого продукту.

У табл. 3.1 проводиться порівняння середовищ розробки застосунку для моніторингу та догляду за нігтями, підкреслюючи їх відмінності за ключовими критеріями [14].

Таблиця 3.1 – Порівняння середовищ розробки

Критерій	Android Studio	Eclipse	IntelliJ IDEA	Xamarin
Функціональність	Повний набір інструментів для розробки, тестування та налагодження. Вбудований емулятор	Широкий набір плагінів для різних задач	Потужні засоби для написання коду та рефакторингу. Підтримка різних мов програмування	Можливість створення застосунків для Android, iOS та Windows з єдиної бази коду

Кінець таблиці 3.1

Критерій	Android Studio	Eclipse	IntelliJ IDEA	Xamarin
Гнучкість та масштабованість	Підтримка великих проектів. Активна підтримка спільноти та часті оновлення	Висока гнучкість завдяки модульності	Придатна для мультиплатформових проектів	Підходить для розробників, які працюють у екосистемі Microsoft
Інтеграція з Android SDK	Глибока інтеграція з Android SDK. Підтримка Jetpack	Потребує додаткового налаштування для роботи з Android	Менш тісна інтеграція з Android SDK	Нативні функції реалізовані через бібліотеки
Зручність використання	Зручний інтерфейс. Швидке створення UI завдяки інструментам, таким як Layout Editor або Jetpack Compose	Інтерфейс менш зручний порівняно з Android Studio	Інтуїтивно зрозумілий інтерфейс, але менш оптимізований для Android	Високий поріг входу для новачків, які не знайомі з C#

Для реалізації застосунку для моніторингу та догляду за нігтями було обрано мову Kotlin, яка офіційно підтримується Google і рекомендована для Android-розробки. Ця мова володіє лаконічним синтаксисом, високим ступенем захисту від помилок, підтримкою сучасних технологій та повною сумісністю з Java.

Java є традиційною мовою для розробки застосунків для Android і вже давно є основним стандартом. Однак його синтаксис більш багатословніший, і додатковий код доводиться писати вручну для захисту від помилок, пов'язаних зі значеннями null [21]. Однак Java все ще популярна, особливо серед розробників, які вже знайомі з цією мовою.

У табл. 3.2 проводиться порівняння між написанням UI за допомогою View та за допомогою Jetpack Compose за ключовими критеріями [20].

Таблиця 3.2 – Порівняння методів написання UI.

Критерій	Використання View та Layout	Використання Jetpack Compose
Оголошення UI	Потрібно створити XML-файли для визначення інтерфейсу	Інтерфейс описується прямо у коді Kotlin, використовуючи функції
Код та структура	Суворі структура з поділом між логікою та UI (XML та Activity/Fragment)	UI та логіка в одному файлі, що робить код компактним та читабельним
Реактивність	Для оновлення UI часто використовуються ViewModel та LiveData або StateFlow	Compose має вбудовану реактивну природу, що полегшує оновлення UI
Гнучкість у змінах UI	Для зміни UI потрібне оновлення XML та перезапуск Activity/Fragment	UI автоматично оновлюється через декларативний підхід у реальному часі.
Простота навчання	Для новачків простіше починати з XML та стандартних View	Менше шаблонів і документації, оскільки Compose є відносно новим, але воно простіше для людей, вже знайомих з Kotlin

У процесі розробки Android-застосунків управління залежностями (Dependency Injection, DI) є важливою частиною архітектури. На сьогоднішній день існує кілька популярних інструментів для впровадження залежностей, серед яких можна виділити Hilt, Dagger 2 та Koin. Кожен із цих фреймворків має свої особливості, переваги та недоліки, і вибір між ними залежить від конкретних вимог проекту. Після ретельного аналізу було обрано Hilt для Android-застосунку для проекту моніторингу та догляду за нігтями, оскільки він надає зручні засоби

інтеграції з Android, автоматичне керування залежностями та тісний зв'язок із Android Jetpack. У наступному абзаці буде докладно порівняно та описано різницю між Hilt, Dagger 2 і Koin, для пояснення вибору саме Hilt.

Hilt є фреймворком, який інтегрований безпосередньо з Android та Jetpack, що робить його ідеальним для розробки Android-застосунків. На відміну від Dagger 2, який потребує додаткового налаштування для роботи з Android, Hilt використовує анотації, такі як `@InstallIn`, для інтеграції з компонентами Android, що спрощує налаштування та впровадження залежностей. Важливо відзначити, що Hilt автоматично генерує код, мінімізуючи потребу в ручному налаштуванні та надаючи зручні механізми для роботи із залежностями на різних рівнях програми.

Dagger 2, у свою чергу, є універсальним інструментом, який може бути використаний не тільки для Android, але і для інших платформ. Він надає більшу гнучкість та контроль над створенням залежностей, але для цього розробник повинен вручну налаштовувати компоненти та модулі. Це потребує більшої кількості коду та складнішої конфігурації, що може стати проблемою при масштабуванні проекту. Однак Dagger 2 генерує код на етапі компіляції, що дозволяє уникнути проблем з продуктивністю, пов'язаних з використанням рефлексії, і є більш ефективним при роботі з великими та складними програмами.

Koin, у свою чергу, є більш легковажним рішенням для впровадження залежностей, орієнтованим на мову Kotlin. Він не вимагає анотацій та генерування коду, а використовує декларативний підхід, що спрощує налаштування та робить його зручним для невеликих проектів. Однак, через те, що Koin використовує рефлексію для роботи з залежностями, це може призвести до деякого зниження продуктивності, особливо у великих застосунках. Крім того, Koin не має такої гнучкості та потужності, як Dagger 2, і може бути менш підходящим для більш складних сценаріїв [17].

На закінчення можна сказати, що вибір між Hilt, Dagger 2 і Koin залежить від конкретних потреб проекту. Hilt є відмінним вибором для Android-розробників завдяки своїй інтеграції з Android Jetpack, простоті налаштування та високій продуктивності. Dagger 2 є більш гнучким та потужним інструментом, що

підходить для більш складних проєктів, але потребує більше зусиль при налаштуванні. Koін ідеально підходить для невеликих проєктів, завдяки своїй простоті і мінімальному налаштуванню, але може бути менш продуктивним для великих застосунків. Оскільки Hilt ідеально відповідає вимогам Android-розробки та значно спрощує управління залежностями, роблячи процес розробки більш швидким та зручним, було обрано саме цей фреймворк.

У контексті розробки Android-програми для відстеження стану здоров'я нігтів, використання Git дозволяє легко керувати версією коду, гарантувати, що зміни будуть внесені до проєкту без ризику втрати даних. Git також забезпечує гнучкість в інтеграції з іншими інструментами та платформами, такими як GitHub, для зберігання коду та управління завданнями. Крім того, використання Git дозволяє документувати всі зміни, проводити код-ревью і зливати різні гілки розробки, що робить процес роботи над проєктом більш структурованим і прозорим. У разі виникнення помилок або необхідності повернення до попередньої версії Git надає ефективні механізми для відкату та відновлення попередніх станів проєкту [18]. Таким чином, вибір Git був зроблений через його універсальність, простоту у використанні, гнучкість, широке поширення в індустрії та автоматизації процесів розробки.

Room був обраний для роботи з базою даних у проєкті з кількох причин. По-перше, це офіційна бібліотека від Google, яка інтегрується з іншими сучасними бібліотеками Android, що робить її чудовим вибором для програм, побудованих на Android Jetpack. По-друге, Room дозволяє значно спростити роботу з базою даних, приховуючи багато складнощів роботи з низькорівневим SQL, що підвищує продуктивність розробки та знижує ймовірність помилок [19]. Крім того, Room є високорівневим рішенням, яке автоматизує багато процесів, наприклад створення таблиць, міграцію та обробку запитів, що скорочує кількість рукописного коду. Це особливо важливо для програми, де потрібне швидке та безпечне зберігання даних про стан здоров'я нігтів.

Використання ViewModel у поєднанні з Room забезпечує зручне керування даними та реактивне оновлення UI, що ідеально підходить для програми з даними,

що динамічно змінюються. Room також надає вбудовану підтримку асинхронних операцій, що дозволяє уникнути блокування UI-потoku, покращуючи користувацький досвід. Таким чином, вибір Room був зумовлений його тісною інтеграцією з Android, простотою використання, високою продуктивністю та підтримкою сучасних архітектурних підходів. Ця бібліотека дозволяє ефективно та безпечно працювати з даними у локальній базі, що є основою для надійного та масштабованого застосунка для відстеження стану здоров'я нігтів.

При розробці мобільних застосунків важливо вибрати відповідну архітектуру, яка відповідатиме вимогам проекту та забезпечить зручну підтримку, розширюваність та тестованість. В Android-розробці існує кілька популярних архітектурних патернів, серед яких MVC (Model-View-Controller), MVVM (Model-View-ViewModel) та MVI (Model-View-Intent). Кожен з них має свої особливості та може бути використаний залежно від складності програми та вимог команди розробників. Для розробки програми для відстеження стану здоров'я нігтів було обрано архітектурний паттерн MVVM. Цей вибір зумовлений низкою факторів, таких як поділ логіки та представлення, підтримка реактивних даних, простота тестування та масштабованості [13]. У табл. 3.3 буде представлений детальний розбір кожного з трьох архітектурних патернів з акцентом на їх переваги та недоліки, а також пояснення чому MVVM виявився найкращим вибором для даної програми.

Таблиця 3.3 – Розбір архітектурних патернів

Критерій	MVVM	MVI	MVC
Основна ідея	Поділ логіки подання та даних через ViewModel	Реактивне оновлення UI на основі дій користувача	Поділ на Модель, Вид та Контролер для керування станом

Кінець таблиці 3.3

Критерій	MVVM	MVI	MVC
Як обробляється UI	LiveData, StateFlow: автоматичне оновлення UI, коли дані змінюються	View оновлюється у відповідь на Intent та Model	Контролер управляє змінами UI вручну, взаємодіючи з моделлю та видом
Управління станом	Стан зберігається у ViewModel і автоматично оновлюється у UI	Стан у Model змінюється на основі дій Intent	Контролер керує станом та оновлює вигляд вручну
Складність реалізації	Середня. Потрібно розуміння життєвого циклу ViewModel та підходів роботи зі станом	Висока. Потребує глибокої інтеграції реактивних підходів та стану	Низька для простих проєктів, але може стати громіздким у складніших
Затребуваність	Програми з розділенням бізнес-логіки та UI	Програми з реактивним UI, де дані змінюються у відповідь на дії	Для простих програм або програм з обмеженою складністю

3.2 Моделювання програмного забезпечення на основі функціональних вимог

Моделювання програмного забезпечення є ключовим етапом у процесі розробки, що дозволяє на основі функціональних вимог визначити структуру та поведінку системи. Цей процес допомагає розробникам заздалегідь виявити можливі проблеми та спроектувати програму таким чином, щоб вона максимально відповідала потребам користувачів. У цьому розділі наведено моделювання

програмного забезпечення для програми, яка аналізує стан нігтів за допомогою технологій машинного навчання та штучного інтелекту.

3.2.1 Функціональні вимоги

Програма надає користувачам можливість зробити фотографію нігтя, яка оброблятиметься з використанням ML Kit для аналізу вмісту зображення. Якщо система підтверджує, що на зображенні є ніготь, фотографія зберігається у локальній базі даних, щоб користувач міг пізніше переглянути результати на екрані історії (History Screen). Крім того, після обробки зображення користувач може запросити додаткову допомогу Gemini AI через функціонал чату. Натиснувши на відповідну кнопку, користувач переходить у чат, де може отримати рекомендації та уточнення стану нігтя. Весь діалог із штучним інтелектом також зберігається у локальній базі даних, надаючи користувачеві доступ до історії повідомлень та рекомендацій. Такий підхід робить програму зручним інструментом для моніторингу здоров'я нігтів та отримання консультацій, інтегруючи сучасні технології машинного навчання та AI.

3.2.2 Структура даних та основних компонентів

Для ефективної роботи програми та забезпечення зручності користувача потрібна грамотно організована структура даних. Вона включає обробку фотографій, аналіз зображень, взаємодію зі штучним інтелектом та збереження всіх даних у базі для подальшого доступу. У табл. 3.4 представлена структура даних, що використовується в застосунку, з описом ключових сутностей, їх полів та призначення.

Таблиця 3.4 – Структура даних

Сутність	Поля	Опис
Фото(PhotoEntity)	id,photo,date	Містить дані про зроблене фото та час його створення
Повідомлення(Message)	text,isUser	Містить дані про текст повідомлення та хто його відправив користувач або штучний інтелект
Повідомлення(MessageEntity)	id,text,isUser	Логіка як у Message,але використовується для збереження до бази даних

Для забезпечення функціональності застосування та взаємодії між різними компонентами були виділені основні компоненти, кожен з яких відповідає за виконання певних завдань. Ці компоненти забезпечують роботу з камерою, аналіз зображень, взаємодію зі штучним інтелектом та управління базою даних. Список компонентів з їх функціями та коротким описом їхньої ролі в системі наведено у табл. 3.5.

Таблиця 3.5 – Основні компоненти

Компонент	Функції	Опис
CameraRepository	takePhoto,detectNail	Відповідає за взаємодію з камерою, створення файлу фото та передачу даних про зображення
DatabaseRepository	inserMessage,getAllMessages	Відповідає за взаємодію з базою даних чату,відправку та отримання повідомлень

Продовження таблиці 3.5

Компонент	Функції	Опис
GeminyRepository	getAnswerWithPhoto,getAnswerWithoutPhoto	Відповідає за взаємодію зі штучним інтелектом,відправку запиту з фото або без, та отримання відповіді
NetworkCheckerRepository	isConnectionAvailable	Відповідає за перевірку чи є у користувача інтернет приєднання або ні
PhotoRepository	savePhotoToRoom,getAllPhotos	Відповідає за взаємодію з базою даних,збереження фото у базу даних та відправку користувачу
StorageRepository	setLightBack,isLightBack	Відповідає за взаємодію з UI,зміна кольорів які використовуються у застосунку

3.3 Проєктування структури бази даних для зберігання інформації про стан нігтів

У цьому розділі буде описано структуру бази даних, яка використовується для зберігання інформації про стан нігтів у застосунку. Для реалізації функціоналу зберігання повідомлень чату та фотографій, зроблених користувачами, розроблені відповідні сутності та репозиторії, які взаємодіють із локальною базою даних. Кожна сутність є таблицею в базі даних, де зберігаються різні дані: повідомлення, надіслані користувачами або системою, а також фотографії, зроблені для аналізу стану нігтів. Репозиторії забезпечують методи для збереження та вилучення цих даних, що дозволяє ефективно керувати інформацією у застосунку.

У програмі використовується локальна база даних для зберігання ключових даних, таких як фотографії, зроблені користувачами, та повідомлення, надіслані в

чат з AI. Для ефективного управління даними було створено дві основні таблиці – для зберігання фотографій та повідомлень. Ці таблиці забезпечують можливість зберігати, видобувати та обробляти дані, необхідні для функціонування програми.

Перша таблиця – PhotoEntity – призначена для зберігання інформації про фотографії. Кожна фотографія, зроблена користувачем, зберігається з унікальним ідентифікатором (id), шляхом до файлу зображення (фото) і датою її зйомки (date). Ця таблиця використовується для зберігання всіх фотографій, які аналізуються за допомогою ML Kit і передаються Gemini AI для отримання рекомендацій. Завдяки цій таблиці користувачі можуть переглядати зроблені фотографії в історії.

Друга таблиця — MessageEntity — зберігає повідомлення, надіслані в чат зі штучним інтелектом. важлива для збереження історії чатів і дозволяє користувачу повернутися до раніше отриманих рекомендацій та уточнень щодо станом нігтів.

Для роботи з цими таблицями було створено два репозиторії — DatabaseRepository та PhotoRepository. DatabaseRepository забезпечує методи для вставки та вилучення повідомлень з бази даних, а PhotoRepository керує збереженням фотографій та їх вилученням. Обидва репозиторії взаємодіють з Room Database, що забезпечує надійне та ефективне зберігання даних на пристрої.

Таким чином, база даних та таблиці у ній відіграють ключову роль у зберіганні інформації, необхідної для аналізу стану нігтів та надання рекомендацій користувачам. Це рішення дозволяє забезпечити зручний доступ до даних і гарантує, що всі зміни, пов'язані з фотографіями та повідомленнями, зберігаються для подальшого використання у програмі.

3.4 Висновки за розділом 3

У цьому розділі було розглянуто моделювання програмного забезпечення для мобільного застосунку, яке дозволяє користувачам аналізувати стан своїх нігтів за допомогою технологій машинного навчання та штучного інтелекту. Ми описали функціональні вимоги та детально розглянули основні процеси, такі як

фотографування нігтів, їх аналіз, збереження результатів у базі даних та взаємодія з AI для отримання рекомендацій.

Моделювання бази даних показало, що для зберігання важливої інформації, такої як фотографії та повідомлення чату, використовується локальна база даних з таблицями PhotoEntity та MessageEntity. Це дозволяє ефективно зберігати і витягувати дані, що забезпечує можливість перегляду історії фотографій і чату.

В результаті, було продемонстровано, як система забезпечує зручність використання для кінцевого користувача, інтегруючи сучасні технології штучного інтелекту для моніторингу здоров'я нігтів і надаючи можливість отримувати корисні рекомендації. Використання локальної бази даних гарантує, що всі дані будуть доступні навіть без інтернет-з'єднання, забезпечуючи швидкий доступ до них в будь-який час.

Завдяки правильно спроектованій базі даних і чітко визначеним функціональним вимогам, застосунок забезпечує високий рівень зручності та ефективності для користувачів, дозволяючи легко відстежувати стан нігтів і отримувати необхідну допомогу від штучного інтелекту.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розділ буде присвячений основним рішенням щодо реалізації програмного забезпечення, що стосуються функціональності застосунку для моніторингу стану нігтів. У цьому розділі буде розглянуто порядок роботи застосунку, який включає етапи взаємодії користувача з програмою, а також технічні аспекти реалізації аналітичних функцій, що забезпечують обробку зображень, збереження даних і взаємодію з штучним інтелектом для надання рекомендацій. Окрім того, буде підсумовано ключові рішення, що були прийняті в процесі розробки, зокрема, вибір технологій і підходів для досягнення високої ефективності та зручності використання застосунку.

4.1 Порядок роботи застосунку для моніторингу стану нігтів

Порядок роботи застосунку для моніторингу стану нігтів визначає ключові етапи взаємодії користувача з застосунком, від початкової фотографії до отримання результатів аналізу та рекомендацій. У цьому підрозділі буде описано, як користувач може за допомогою програми здійснювати фотографування своїх нігтів, обробляти зображення за допомогою машинного навчання та штучного інтелекту, а також отримувати консультації через чат з AI. Крім того, детально розглядатимуться процеси збереження фото та чату в базі даних, що дозволяє користувачеві переглядати історію своїх дій і результатів аналізу в будь-який зручний час.

Порядок роботи застосунку для моніторингу стану нігтів полягає в декількох етапах, які дозволяють користувачеві ефективно здійснювати аналіз стану своїх нігтів і отримувати допомогу від штучного інтелекту.

Першим етапом є фотографування нігтя. Користувач відкриває головний екран застосунку, на якому є кнопка для переходу на екран з діагностуванням нігтя. На екрані діагностування є кнопка «Зробити фото». Натискаючи на її,

користувач активує камеру і робить знімок нігтя. Камера надає можливість отримати чітке зображення, яке буде використано для подальшого аналізу. Завдяки цьому кроку програма отримує необхідне зображення для виявлення на ньому нігтя.

Після того, як фотографія зроблена, вона передається до аналізу зображення. Це відбувається через ViewModel, яка звертається до репозиторію фото для збереження отриманого зображення в локальній базі даних. Одночасно із збереженням, система проводить аналіз фото за допомогою ML Kit, щоб визначити, чи є на зображенні ніготь. Якщо ніготь виявлено, фотографія зберігається, і користувач може переглянути результат у майбутньому на екрані «Історія». Якщо ж на фото не виявлено нігтя, програма відображає повідомлення про це.

Наступний етап — перегляд результатів. У разі успішного виявлення нігтя, фотографія додається до бази даних і стає доступною для перегляду на екрані «Історія». Якщо зображення не містить нігтя, на екрані з'являється відповідне повідомлення, яке інформує користувача про результат аналізу.

Завершальним етапом є збереження всіх даних в локальній базі даних. Всі фотографії, а також чати з AI, зберігаються в базі, що дозволяє користувачеві зберігати історію своїх дій і результатів. Це створює можливість для моніторингу стану нігтів протягом часу, а також надає доступ до всіх отриманих порад і рекомендацій.

Таким чином, порядок роботи застосунку забезпечує простоту в отриманні результатів аналізу стану нігтів і надає можливість користувачам отримувати консультації від AI для підтримки здоров'я своїх нігтів.

4.2 Технічна реалізація обробки зображень та взаємодії з штучним інтелектом

Для реалізації обробки зображень та взаємодії з штучним інтелектом у застосунку використовується кілька ключових функцій, кожна з яких відповідає за окремі аспекти роботи з фото та обробки даних.

По-перше, процес отримання відповіді без використання фотографії включає створення екземпляра моделі `GenerativeModel`. Ця модель, з використанням API-ключа та конфігурації генерації, створює відповідь на основі запитання та заданого тексту. Цей процес не включає роботу з зображеннями, а тільки текстову взаємодію з користувачем.

Для випадків, коли потрібно обробити зображення, використовується метод `getAnswerWithPhoto`. Тут відбуваються кілька ключових етапів:

- модель `GenerativeModel` запускається з використанням більш підходящої версії для обробки фото — `gemini-1.5-flash`;
- зображення проходить стиснення за допомогою функції `compressBitmap`, що дозволяє зменшити розмір картинки та пришвидшити обробку;
- за допомогою `content` передається як зображення, так і текст (запитання та додатковий текст), після чого модель генерує відповідь на основі комбінації зображення та тексту.

Важливою частиною процесу є функція `takePhoto`, яка використовує можливості камери для захоплення фото. Цей метод створює файл із зображенням, обробляє його та зберігає в галереї пристрою. При успішному знятті фотографії повертається `Bitmap` та шлях до файлу. Якщо камера не може захопити зображення, відбувається обробка помилки.

Для обробки зображень та розпізнавання об'єктів на фото використовується метод `detectNail`. У ньому застосовується бібліотека `ImageLabeling` для аналізу зображення та пошуку об'єктів, що відповідають мітці "nail" (ніготь). Якщо розпізнано об'єкт з достатньою впевненістю (більше 80%), функція повертає результат, що підтверджує наявність нігтя на зображенні.

Ці методи дозволяють не тільки працювати з зображеннями, але й інтегрувати їх у взаємодію з AI-системою, що відкриває можливості для аналітики та надання користувачу відповідей на основі фотографії. Взаємодія з Gemini API дозволяє отримати більш персоналізовані та точні результати, при цьому стиснення зображень та використання AI для їх аналізу роблять систему швидкою та ефективною.

4.3 Висновки за розділом 4

У розділі було розглянуто основні рішення щодо реалізації програмного забезпечення для моніторингу стану нігтів, а також технічні аспекти обробки зображень і взаємодії з штучним інтелектом.

Порядок роботи застосунку забезпечує зручний процес взаємодії з користувачем, починаючи з фотографування нігтя і закінчуючи збереженням результатів у локальній базі даних. Користувач може зробити знімок, отримати аналіз стану нігтя та зберігати історію своїх дій. Це дозволяє легко відслідковувати зміни в стані нігтів і отримувати рекомендації від штучного інтелекту.

Технічна реалізація обробки зображень включає різні методи для обробки фото, зокрема стиснення зображень перед передачею в систему для швидкої обробки. Використання бібліотеки ImageLabeling дозволяє ефективно виявляти нігті на зображеннях, а методи інтеграції з AI забезпечують точні та персоналізовані результати.

Інтеграція з штучним інтелектом через модель GenerativeModel і Gemini API дозволяє швидко реагувати на запити користувачів як з текстовими запитами, так і з фото. Це забезпечує можливість надання рекомендацій, що базуються на зображеннях, а також ведення чату з AI для отримання консультацій.

Збереження даних і історії є важливою частиною роботи застосунку. Всі отримані фотографії та чати з AI зберігаються в базі даних, що дозволяє

користувачам переглядати історію своїх дій і результатів аналізу в будь-який час, забезпечуючи постійний моніторинг стану нігтів.

Загалом, вжиті рішення щодо організації роботи застосунку і технологій обробки зображень забезпечують високу ефективність, зручність і персоналізацію досвіду користувача, дозволяючи отримати точні та оперативні результати на основі зображень нігтів.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

У цьому розділі буде розглянуто, як застосунок для моніторингу стану нігтів працює в реальних умовах, які методи тестування використовувалися для оцінки точності аналізу, а також проведено експериментальні дослідження для визначення ефективності програми в різних сценаріях. Завдяки цим етапам буде можливість оцінити стабільність і точність роботи програми, а також виявити можливі недоліки та запропонувати шляхи для подальшого вдосконалення.

5.1 Опис роботи застосунку

Застосунок для моніторингу стану нігтів був розроблений для забезпечення зручного та ефективного способу аналізу здоров'я нігтів користувачів за допомогою фотографій, обробки зображень за допомогою штучного інтелекту та надання рекомендацій. Робота застосунку складається з кількох етапів, що дозволяють користувачеві здійснювати аналіз стану нігтів, отримувати консультації та зберігати історію своїх даних для подальшого використання.

При першому запуску застосунку користувачеві буде запропоновано надати доступ до камери та до збереження фото на пристрої. Це необхідно для того, щоб програма могла здійснювати фотографування нігтів та зберігати отримані зображення в локальній базі даних для подальшого аналізу. Також варто зазначити, що API Gemini AI може бути недоступним в деяких регіонах через обмеження доступу. У таких випадках використання VPN дозволяє вирішити цю проблему та забезпечити доступ до сервісу, що дозволяє користувачам продовжувати отримувати консультації та аналіз від штучного інтелекту без обмежень, незалежно від їхнього географічного розташування, рис 5.1 та рис 5.2.

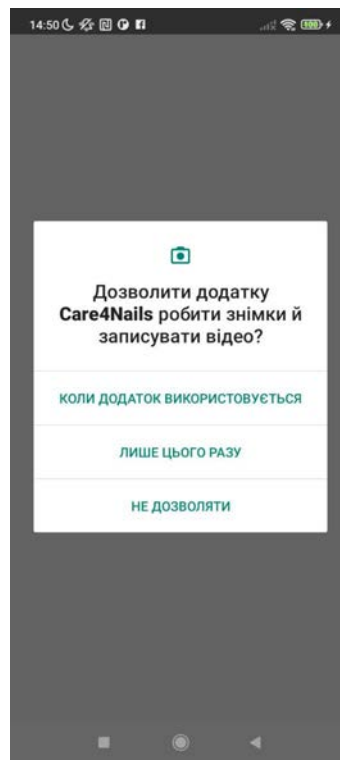


Рисунок 5.1 – Скріншот застосунку з дозволом доступу до даних



Рисунок 5.2 – Скріншот застосунку з повідомленням про недоступність у регіоні

Користувач відкриває головний екран застосунку, на якому розташовано чотири кнопки для переходу на різні екрани, рис 5.3. Натискаючи на кнопку "Діагностування", користувач потрапить на екран діагностування, рис 5.4. Тут є кнопка "Зробити фото", яка активує камеру смартфона для захоплення зображення нігтя. Зображення повинно бути чітким, щоб система могла ефективно здійснити його аналіз. На рис 5.4 показано також як не потрібно робити фото. На рис 5.5 показано навпаки, як правильно робити фото.

На екрані також присутній CheckBox з текстом "Допомога від AI", якщо його залишити неактивним, з'являється кнопка зберегти. Натиснувши на цю кнопку, користувач збереже фото і повернеться на головний екран. Якщо CheckBox зробити активним, з'являється кнопка "Діагностувати". Натиснувши її, користувач потрапить на екран чату зі штучним інтелектом, який зробить аналіз за фото, рис 5.6.

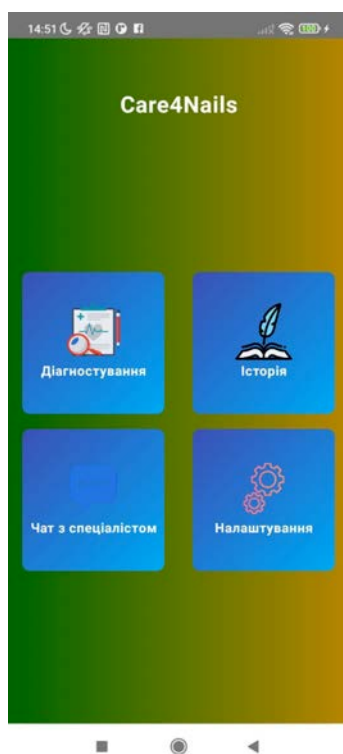


Рисунок 5.3 – Скріншот головного екрану застосунку



Рисунок 5.4 – Скріншот екрану діагностування з неправильним фото

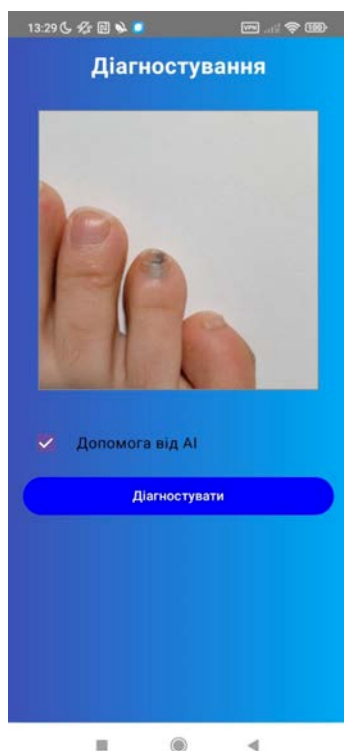


Рисунок 5.5 – Скріншот правильного прикладу фото

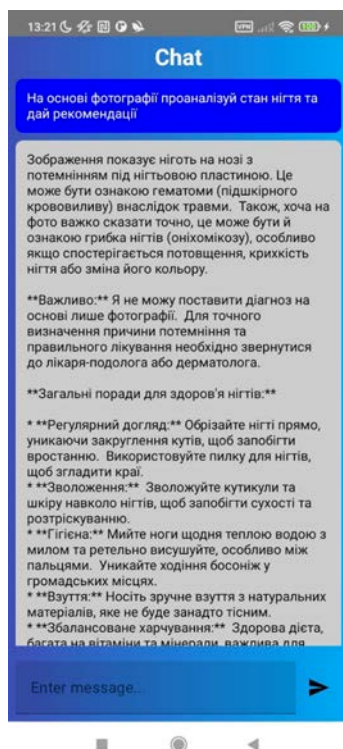


Рисунок 5.6 – Скріншот екрану чату з аналізом від штучного інтелекту

Натиснувши кнопку "Чат з спеціалістом" у головному меню ,користувач також потрапить на екран чату зі штучним інтелектом, тільки буде викликатись інший запрос до Gemini(без фото).

Натиснувши кнопку "Історія" у головному меню, користувач потрапить на екран із фото нігтів, які були збережені, рис 5.7.

І нарешті якщо користувач натисне кнопку "Налаштування" у головному меню, потрапить на екран, на котрому він може змінити кольорову палітру у застосунку, рис 5.8.

На рис 5.9 відображено результат зміни кольорової палітри.

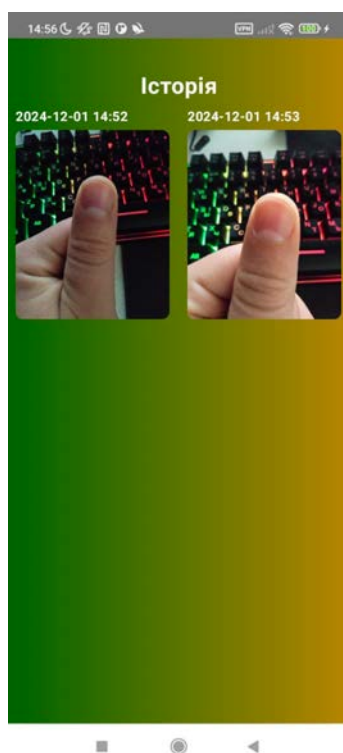


Рисунок 5.7 – Скріншот екрану історія



Рисунок 5.8 – Скріншот екрану налаштування

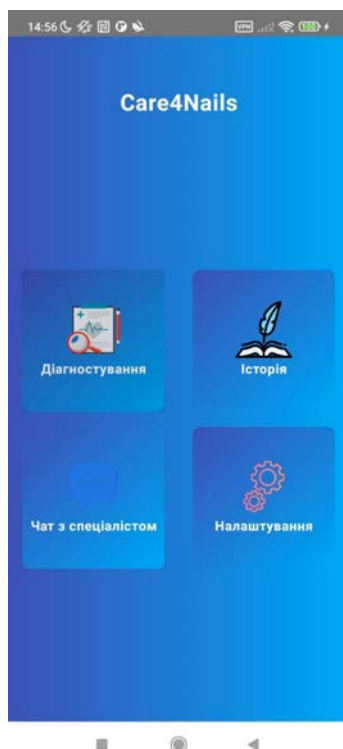


Рисунок 5.8 – Скріншот екрану головного меню після зміни кольорової палітри

5.2 Тестування точності аналізу стану нігтів

Тестування точності аналізу стану нігтів є важливою частиною процесу розробки застосунку, оскільки воно дозволяє визначити ефективність використання технологій обробки зображень та штучного інтелекту для правильного виявлення стану нігтів. Для того щоб забезпечити високу точність, система повинна точно розпізнавати ніготь на фотографіях і правильно класифікувати його стан. Тестування дозволяє виявити можливі помилки, покращити алгоритми та забезпечити користувачам надійні результати.

Процес тестування точності аналізу стану нігтів включає кілька важливих етапів. Спочатку було зібрано набір тестових зображень нігтів різного стану, знятих за різних умов освітлення та з різних кутів. Це дало змогу перевірити, як система справляється з різноманітними ситуаціями, коли фотографії можуть бути не зовсім ідеальними.

Для оцінки точності розпізнавання використовувався алгоритм, заснований на бібліотеці ImageLabeling, який допомагає виявити на зображенні ніготь. Оцінка точності проводилась шляхом порівняння результатів аналізу з еталонними даними — тобто, визначалося, чи правильно система виявляє наявність чи відсутність нігтя. На рис 5.9 та 5.10, можемо побачити що ImageLabeling добре справляється із розпізнаванням нігтя при поганій якості та розмитості фото.

Наступним фактором перевірки захвату нігтя було освітлення фото. На рис 5.11 відображено результат захвату нігтя при не дуже доброму освітленні.

Останнім фактором було дальність захвату нігтя на фото. На рис 5.12 відображено результат захвату нігтя на середній дальності.

Точність відповідей, наданих моделлю Gemini, була також важливим аспектом тестування застосунку. Модель активно використовувалася для оцінки стану нігтів на основі наданих зображень, що дозволяло перевірити її здатність правильно класифікувати різні стани нігтів, рис 5.13.

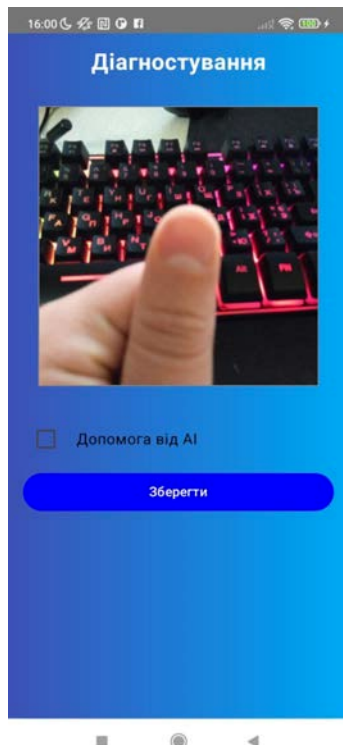


Рисунок 5.9 – Скріншот результату захвату нігтя при не дуже добрій якості фото

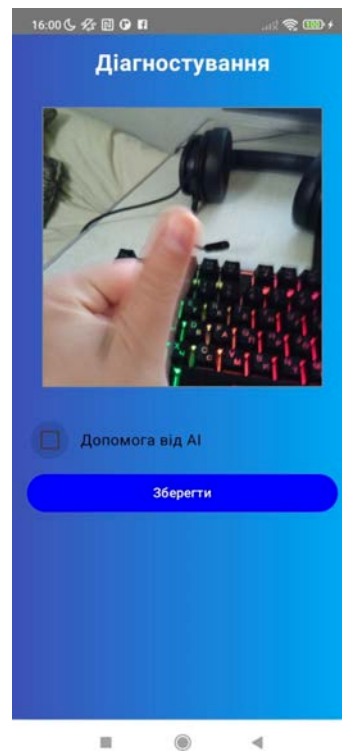


Рисунок 5.10 – Скріншот результату захвату нігтя при поганій якості фото

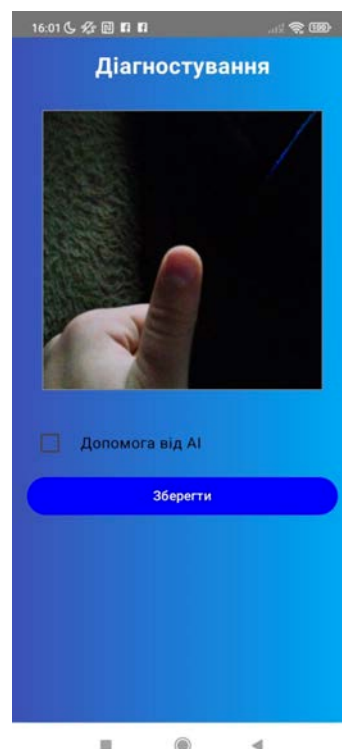


Рисунок 5.11 – Скріншот результату захвату нігтя при не дуже доброму освітленні

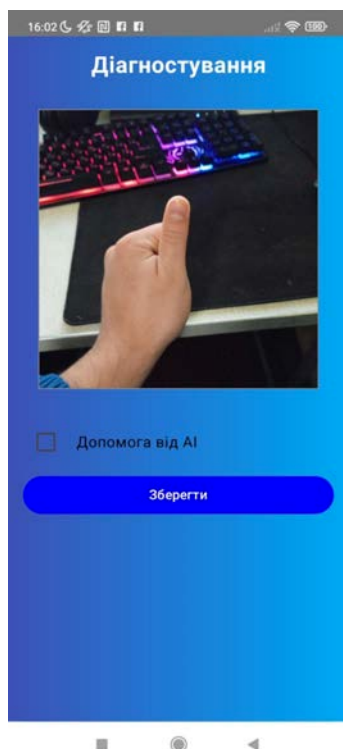


Рисунок 5.12 – Скріншот результату захвату нігтя на середній дальності

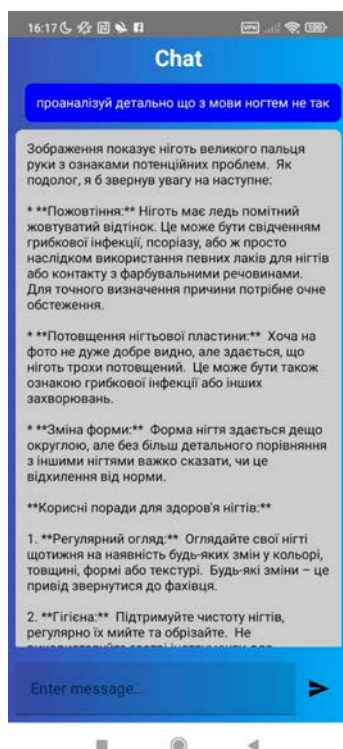


Рисунок 5.13 – Скріншот результату детального аналізу нігтя

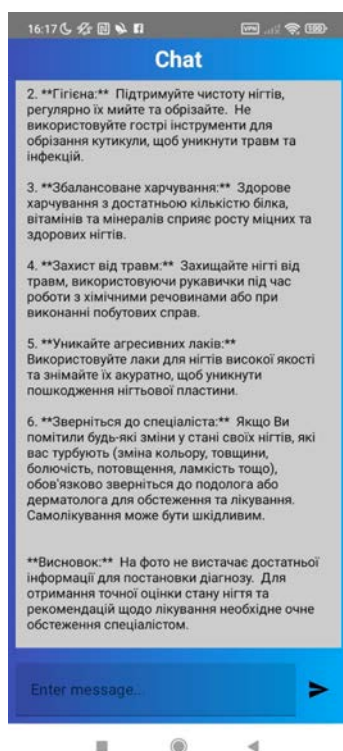


Рисунок 5.13, аркуш 2

Модель Gemini досить добре справляється з аналізом фотографій стану нігтів. Вона продемонструвала хорошу здатність до точного визначення стану нігтів навіть при зображеннях низької якості або поганому освітленні. Модель успішно класифікує основні типи пошкоджень або захворювань нігтів, що дозволяє використовувати її для базового аналізу.

Особливо варто відзначити, що Gemini є безкоштовною моделлю, що робить її доступною для широкого кола користувачів. Незважаючи на це, вона показала досить високу ефективність і точність у своїй роботі, навіть у складних умовах. Це робить Gemini привабливим вибором для тих, хто шукає безкоштовне рішення для аналізу зображень без значних втрат в якості результатів.

5.3 Висновки за розділом 5

В результаті тестування точності аналізу стану нігтів було виявлено, що застосунок, що використовує модель ImageLabeling, здатен ефективно працювати

з різними умовами зображень. Система добре справляється з низькою якістю зображень, розмитістю та недостатнім освітленням, що є важливим фактором для користувачів, які можуть зробити фотографії в менш ніж ідеальних умовах. Завдяки використанню набору тестових зображень з різних ситуацій, вдалося перевірити здатність системи точно виявляти наявність чи відсутність нігтя, а також класифікувати його стан.

Модель Gemini, яка також активно використовувалась для аналізу зображень, продемонструвала хорошу точність у класифікації стану нігтів. Вона змогла успішно визначати основні типи пошкоджень та захворювань навіть при поганому освітленні або низькій якості фотографій. Це свідчить про її ефективність у різноманітних умовах, що дозволяє користувачам отримувати надійні результати аналізу зображень.

ВИСНОВКИ

У результаті виконаної дипломної роботи було розроблено застосунок для аналізу стану нігтів, що використовує сучасні технології обробки зображень та штучного інтелекту для точного визначення різних станів нігтів на фотографіях. Застосунок успішно справляється з розпізнаванням нігтів навіть при низькій якості зображень, поганому освітленні та різних кутах зйомки, що є важливим фактором для користувачів, які роблять фото в умовах, що не завжди є ідеальними. Тестування точності аналізу показало, що система здатна ефективно виявляти основні пошкодження та захворювання нігтів, класифікуючи їх відповідно до наданих зображень. Особливу увагу було приділено роботі моделі Gemini, яка продемонструвала високу точність у визначенні стану нігтів при різних умовах. Окрім того, важливою перевагою є те, що Gemini є безкоштовною моделлю, що робить її доступною для широкого кола користувачів, не знижуючи при цьому точність результатів. Завдяки поєднанню передових технологій і доступних інструментів, розроблений застосунок може стати корисним інструментом для користувачів, які хочуть швидко і точно оцінити стан своїх нігтів. В цілому, робота підтвердила ефективність використання технологій обробки зображень та штучного інтелекту для аналізу зображень у медицині та догляді за здоров'ям. Враховуючи позитивні результати тестування, застосунок має перспективи для подальшого розвитку та вдосконалення, включаючи підтримку нових типів зображень та розширення функціональних можливостей.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. NailKeeper - Stop Biting Nails [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.nailkeeper>.
2. Healthy Nails [Electronic resource]. – Access mode: <https://www.amazon.com/Shlok-Consulatants-Healthy-Nails/dp/B00MHRKG1C>.
3. Frenchies Modern Nail Care [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.zenoti.frenchiesmodern>.
4. Application to Detect Anemia [Electronic resource]. – Access mode: <https://www.reanfoundation.org/anemia-detection-application/>.
5. Google. Google ML Kit: An easy way to add machine learning to your app. [Electronic resource]. – Access mode: <https://developers.google.com/ml-kit>
6. Google. Gemini. [Electronic resource]. – Access mode: <https://blog.google/products/gemini/>.
7. Zeng Z. Nail health monitoring using mobile applications / Z. Zeng, J. Wang // Journal of Digital Health. – 2023. – T. 2, № 4. – С. 235-240.
8. Ghaleb F. Image processing techniques for nail health assessment / F. Ghaleb, M. Kamal // International Journal of Medical Informatics. – 2021. – T. 151. – С. 104-476.
9. Zhang H. Advances in computer vision for nail disease detection / H. Zhang, Z. Chen, Y. Liu // Computers in Biology and Medicine. – 2022. – T. 141. – С. 105-154.
10. Kaur R. A review of image processing techniques for health monitoring applications / R. Kaur, M. Jain // Journal of Biomedical Informatics. – 2021. – T. 118. – С. 103-806.
11. Goodfellow I. Deep Learning / I. Goodfellow, Y. Bengio, A. Courville. – MIT Press, 2016. – С. 51.
12. Rajalakshmi P. Machine learning for nail health assessment: A survey / P. Rajalakshmi, A. Vidhya // Healthcare Technology Letters. – 2022. – T. 9, № 4. – С. 132-138.

13. GeeksforGeeks. Difference between MVC, MVP, and MVVM architecture pattern in Android [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/difference-between-mvc-mvp-and-mvvm-architecture-pattern-in-android/>.

14. Android Developer. Android Studio vs Eclipse: Know which is the best for Android app development [Electronic resource]. – Access mode: <https://www.androiddeveloper.co.in/blog/android-studio-vs-eclipse-know-which-is-the-best-for-android-app-development/>.

15. Dopebase. Android Studio vs IntelliJ IDEA: Which is better for Kotlin development [Electronic resource]. – Access mode: <https://dopebase.com/android-studio-intellij-idea-better-kotlin-development>.

16. Medium. Xamarin vs Android Studio: A comparison guide with pros and cons [Electronic resource]. – Access mode: <https://medium.com/nerd-for-tech/xamarin-vs-android-studio-a-comparison-guide-with-pros-and-cons-dab30cb11008>.

17. Medium. Dagger Hilt vs Koin [Electronic resource]. – Access mode: <https://medium.com/@aruncse2k20/dagger-hilt-vs-koin-e17af98427da>.

18. GeeksforGeeks. What is Git? [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/what-is-git/>.

19. Kraithatha K. Room Database in Android [Electronic resource] / K. Kraithatha. – Access mode: <https://medium.com/@kathankraithatha/room-database-in-android-294442467bc8>.

20. OpenReplay. XML vs Jetpack Compose: A Comparison [Electronic resource]. – Access mode: <https://blog.openreplay.com/xml-vs-jetpack-compose--a-comparison/>.

21. freeCodeCamp. Kotlin vs Java: What's the Difference? [Electronic resource]. – Access mode: <https://www.freecodecamp.org/news/kotlin-vs-java-whats-the-difference/>.

ДОДАТОК А
Технічне завдання

Вступ

У роботі досліджується сфера розробки програмного забезпечення, яка стосується впровадження методів аналізу стану нігтів за допомогою мобільного застосунку, що використовує технології машинного навчання та штучного інтелекту. Основна увага приділяється створенню програмного продукту, який забезпечує автоматизоване виявлення нігтів на фотографіях та аналіз їхнього стану, спрямований на підвищення точності й зручності моніторингу здоров'я користувачів.

A.1 Підстава для розробки

Підставою для виконання роботи є завдання на дипломний проєкт за темою «Дослідження та програмна реалізація аналітичного застосунку для моніторингу та оцінки стану нігтів», затверджене наказом Національного університету «Запорізька політехніка», протокол № 398 від 7 жовтня 2024 року.

A.2 Призначення розробки

Програмне забезпечення Care4Nails призначене для моніторингу стану нігтів користувачів, зокрема для аналізу їхнього здоров'я, виявлення можливих проблем та відображення рекомендацій. Застосунок використовує сучасні технології машинного навчання для обробки та аналізу зображень нігтів, отриманих через камеру мобільного пристрою. Результати аналізу зберігаються в базі даних для подальшого відстеження та порівняння змін. Користувачі можуть отримувати підказки щодо поліпшення стану нігтів.

А.3 Основні вимоги до програмного забезпечення, що розробляється

А.3.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати наступні функції:

- аналіз стану нігтів користувача на основі зображень, отриманих через камеру мобільного пристрою;
- збереження результатів аналізу в базі даних для подальшого відстеження динаміки;
- надання рекомендацій щодо покращення стану нігтів;
- можливість налаштування кольорової гами застосунку;
- підтримка чату зі штучним інтелектом та збереження діалогу у базу даних.

А.3.2 Вимоги до надійності

Програма повинна забезпечувати:

- високу точність аналізу стану нігтів на основі наданих зображень;
- обробку виключних ситуацій, таких як помилки під час отримання відповіді від Gemini через відсутність інтернет з'єднання або недоступність у регіоні;
- стабільну та швидку роботу на різних мобільних пристроях з операційною системою Android.

А.3.3 Умови експлуатації

Програмне забезпечення розраховане для використання на мобільних пристроях з операційною системою Android версії 10 та новіших. Для завантаження та оновлення застосунку користувачам необхідно мати доступ до

інтернету. Для отримання відповіді можливо необхідно мати доступ використання VPN сервісів.

A.3.4 Вимоги до складу і параметрів технічних засобів

Для коректної роботи застосунку на мобільних пристроях необхідні такі мінімальні вимоги:

- операційна система Android версії 10 або новішої;
- процесор із тактовою частотою щонайменше 1.8 ГГц;
- щонайменше 2 ГБ оперативної пам'яті.

A.3.5 Вимоги до програмної документації

Система повинна бути забезпечена технічною документацією, включаючи інструкції з експлуатації для користувача, а також технічною документацією щодо подальшого обслуговування та подальших оновлень програмного забезпечення, як описано у відповідних розділах документації.

A.4 Порядок контролю та приймання

Терміни виконання дипломної кваліфікаційної роботи магістра визначені у завданні, включаючи дати подання роботи, календарний план із зазначенням етапів, результатів та строків виконання. Захист роботи проводиться згідно з графіком, затвердженим заздалегідь, та відповідно до розкладу засідань екзаменаційної комісії.

ДОДАТОК Б
Текст програми

DatabaseRepositoryImpl.kt

```
package com.diplom.care4nails.data.ai
```

```
import com.diplom.care4nails.domain.model.MessageEntity
```

```
import com.diplom.care4nails.domain.repository.DatabaseRepository
```

```
import javax.inject.Inject
```

```
class DatabaseRepositoryImpl @Inject constructor(private val dao: MessageDao) :
```

```
DatabaseRepository {
```

```
    override suspend fun getAllMessages(): List<MessageEntity> {
        return dao.getAllMessages()
    }
}
```

```
    override suspend fun insertMessage(message: MessageEntity) {
        dao.insertMessage(message)
    }
}
```

```
}
```

GeminiRepositoryImpl.kt

```
package com.diplom.care4nails.data.ai
```

```
import android.graphics.Bitmap
```

```
import com.diplom.care4nails.domain.repository.GeminiRepository
```

```
import com.diplom.care4nails.util.Constants
```

```
import com.diplom.care4nails.util.compressBitmap
```

```
import com.diplom.care4nails.util.encodeImageToBase64
```

```
import com.google.ai.client.generativeai.GenerativeModel
```

```
import com.google.ai.client.generativeai.type.content
```

```
import com.google.ai.client.generativeai.type.generationConfig
```

```
import javax.inject.Inject
```

```
class GeminiRepositoryImpl @Inject constructor() : GeminiRepository {
```

```
    override suspend fun getAnswerWithoutPhoto(question: String, prompt: String): String {
        val generativeModel = GenerativeModel(
            modelName = "gemini-pro",
            apiKey = Constants.API_KEY,
            generationConfig = generationConfig {
                temperature = 0.70f
            }
        )
    }
}
```

```

    )
    val result = generativeModel.generateContent("$prompt $question")
    return result.text.toString()
}

override suspend fun getAnswerWithPhoto(
    question: String,
    prompt: String,
    image: Bitmap
): String {

    val generativeModel = GenerativeModel(
        modelName = "gemini-1.5-flash",
        apiKey = Constants.API_KEY,
        generationConfig = generationConfig {
            temperature = 0.70f
        }
    )

    val compressedBitmap = compressBitmap(image, 80)

    val content = content {
        image(compressedBitmap)
        text("$prompt $question")
    }

    val result = generativeModel.generateContent(content)
    return result.text.toString()
}
}

```

MessageDao.kt

```
package com.diplom.care4nails.data.ai
```

```
import androidx.room.Dao
```

```
import androidx.room.Insert
```

```
import androidx.room.OnConflictStrategy
```

```
import androidx.room.Query
```

```
import com.diplom.care4nails.domain.model.MessageEntity
```

```

@Dao
interface MessageDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertMessage(message: MessageEntity)

    @Query("SELECT * FROM messages")
    suspend fun getAllMessages(): List<MessageEntity>
}

NetworkCheckerRepositoryImpl.kt
package com.diplom.care4nails.data

import android.content.Context
import android.net.ConnectivityManager
import android.net.NetworkCapabilities
import com.diplom.care4nails.domain.repository.NetworkCheckerRepository
import javax.inject.Inject

class NetworkCheckerRepositoryImpl @Inject constructor(private val context: Context) :
    NetworkCheckerRepository {

    override fun isConnectionAvailable(): Boolean {
        val connectivityManager =
            context.getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
        val network = connectivityManager.activeNetwork
        val networkCapabilities = connectivityManager.getNetworkCapabilities(network)
        return
        networkCapabilities?.hasCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET)
            ?: false
    }
}

StorageRepositoryImpl.kt
package com.diplom.care4nails.data

import android.content.Context
import android.content.SharedPreferences

```

```

import com.diplom.care4nails.domain.repository.StorageRepository

class StorageRepositoryImpl(private val context: Context) : StorageRepository {

    private val sharedPreferences = context.getSharedPreferences(PREFS_NAME,
Context.MODE_PRIVATE)

    override fun setLightBack(isLight: Boolean) {
        val editor: SharedPreferences.Editor = sharedPreferences.edit()
        editor.putBoolean(EXTRA_DATA, isLight)
        editor.apply()
    }

    override fun isLightBack(): Boolean {
        return sharedPreferences.getBoolean(EXTRA_DATA, false)
    }

    companion object {
        const val EXTRA_DATA = "ExtraData"
        const val PREFS_NAME = "prefs"
    }
}

```

ContextModule.kt

```
package com.diplom.care4nails.di
```

```

import android.content.Context
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import javax.inject.Singleton

```

```
@Module
```

```
@InstallIn(SingletonComponent::class)
```

```
class ContextModule {
```

```
    @Singleton
```

```
    @Provides
```

```
    fun provideContext(@ApplicationContext appContext: Context): Context {
        return appContext
    }
}

```

```

    }

}

DatabaseModule.kt
package com.diplom.care4nails.di

import android.content.Context
import androidx.room.Room
import com.diplom.care4nails.data.ai.DatabaseRepositoryImpl
import com.diplom.care4nails.data.ai.MessageDao
import com.diplom.care4nails.data.diagnostic.PhotoDao
import com.diplom.care4nails.data.diagnostic.PhotoDatabase
import com.diplom.care4nails.data.diagnostic.PhotoRepositoryImpl
import com.diplom.care4nails.domain.repository.DatabaseRepository
import com.diplom.care4nails.domain.repository.PhotoRepository
import com.diplom.care4nails.util.Constants
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import it.fungokiller.develop.di.Database

@Module
@InstallIn(SingletonComponent::class)
class DatabaseModule {

    @Provides
    fun getDatabase(@ApplicationContext applicationContext: Context): PhotoDatabase {
        return Room.databaseBuilder(
            applicationContext,
            PhotoDatabase::class.java,
            Constants.DATABASE_NAME
        )
            .build()
    }

    @Provides
    fun getPhotoDao(database: PhotoDatabase): PhotoDao {

```

```

        return database.photoDao()
    }

    @Provides
    fun getGeminiDao(database: PhotoDatabase): MessageDao {
        return database.geminiDao()
    }

    @Provides
    @Database
    fun provideDatabasePhotoRepository(impl: PhotoRepositoryImpl): PhotoRepository {
        return impl
    }

    @Provides
    @Database
    fun provideDatabaseGeminiRepository(impl: DatabaseRepositoryImpl): DatabaseRepository
{
    return impl
}
}

```

DataModule.kt

```

package com.diplom.care4nails.di

import android.content.Context
import com.diplom.care4nails.data.StorageRepositoryImpl
import com.diplom.care4nails.data.ai.DatabaseRepositoryImpl
import com.diplom.care4nails.data.ai.GeminiRepositoryImpl
import com.diplom.care4nails.data.diagnostic.CameraRepositoryImpl
import com.diplom.care4nails.data.diagnostic.PhotoRepositoryImpl
import com.diplom.care4nails.domain.repository.CameraRepository
import com.diplom.care4nails.domain.repository.DatabaseRepository
import com.diplom.care4nails.domain.repository.GeminiRepository
import com.diplom.care4nails.domain.repository.PhotoRepository
import com.diplom.care4nails.domain.repository.StorageRepository
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent

```

```

import javax.inject.Singleton

@Module
@InstallIn(SingletonComponent::class)
class DataModule {
    @Provides
    @Singleton
    fun provideCameraRepository(@ApplicationContext context: Context): CameraRepository {
        return CameraRepositoryImpl(context)
    }

    @Provides
    @Singleton
    fun provideDatabasePhotoRepository(impl: PhotoRepositoryImpl): PhotoRepository {
        return impl
    }

    @Provides
    fun providePreferences(@ApplicationContext context: Context): StorageRepository {
        return StorageRepositoryImpl(context)
    }

    @Provides
    @Singleton
    fun provideGemini(impl: GeminiRepositoryImpl): GeminiRepository {
        return impl
    }

    @Provides
    @Singleton
    fun provideDB(impl: DatabaseRepositoryImpl): DatabaseRepository {
        return impl
    }
}

NetworkModule.kt
package com.diplom.care4nails.di

```

```

import android.content.Context
import com.diplom.care4nails.data.NetworkCheckerRepositoryImpl
import com.diplom.care4nails.domain.repository.NetworkCheckerRepository
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import javax.inject.Singleton

```

```

@Module
@InstallIn(SingletonComponent::class)
class NetworkModule {
    @Provides
    @Singleton
    fun provideNetworkRepository(@ApplicationContext context: Context):
NetworkCheckerRepository {
        return NetworkCheckerRepositoryImpl(context)
    }
}

```

Message.kt

```
package com.diplom.care4nails.domain.model
```

```
data class Message(val text: String, val isUser: Boolean)
```

MessageEntity.kt

```
package com.diplom.care4nails.domain.model
```

```
import androidx.room.Entity
```

```
import androidx.room.PrimaryKey
```

```
@Entity(tableName = "messages")
```

```
data class MessageEntity(
```

```
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
```

```
    val text: String,
```

```
    val isUser: Boolean
```

```
)
```

PhotoEntity.kt

```
package com.diplom.care4nails.domain.model
```

```
import androidx.room.Entity
```

```
import androidx.room.PrimaryKey
```

```
@Entity(tableName = "photos")
```

```
data class PhotoEntity(
```

```
    @PrimaryKey(autoGenerate = true)
```

```
    val id: Int = 0,
```

```
    val photo: String,
```

```
    val date:String
```

```
)
```

```
CameraRepository.kt
```

```
package com.diplom.care4nails.domain.repository
```

```
import android.graphics.Bitmap
```

```
import androidx.camera.core.ImageCapture
```

```
import com.google.android.gms.tasks.Task
```

```
interface CameraRepository {
```

```
    fun takePhoto(
```

```
        imageCapture: ImageCapture,
```

```
        onPhotoTaken: (Bitmap, String) -> Unit,
```

```
        onError: (String) -> Unit
```

```
)
```

```
    fun detectNail(bitmap: Bitmap): Task<Boolean>
```

```
}
```

```
DatabaseRepository.kt
```

```
package com.diplom.care4nails.domain.repository
```

```
import com.diplom.care4nails.domain.model.MessageEntity
```

```
interface DatabaseRepository {
```

```
    suspend fun insertMessage(message: MessageEntity)
```

```

        suspend fun getAllMessages(): List<MessageEntity>

    }

GeminiRepository.kt
package com.diplom.care4nails.domain.repository

import android.graphics.Bitmap

interface GeminiRepository {

    suspend fun getAnswerWithoutPhoto(question: String, prompt: String): String

    suspend fun getAnswerWithPhoto(question: String, prompt: String, image: Bitmap):
String

}

NetworkCheckerRepository.kt
package com.diplom.care4nails.domain.repository

interface NetworkCheckerRepository {
    fun isConnectionAvailable(): Boolean
}

PhotoRepository.kt
package com.diplom.care4nails.domain.repository

import com.diplom.care4nails.domain.model.PhotoEntity

interface PhotoRepository {

    suspend fun savePhotoToRoom(photoEntity: PhotoEntity)

    suspend fun getAllPhotos(): List<PhotoEntity>

}

StorageRepository.kt
package com.diplom.care4nails.domain.repository

```

```
interface StorageRepository {
```

```
    fun setLightBack(isLight: Boolean)
```

```
    fun isLightBack(): Boolean
```

```
}
```

```
NavigationHost.kt
```

```
package com.diplom.care4nails.navigation
```

```
import androidx.compose.runtime.Composable
```

```
import androidx.navigation.NavHostController
```

```
import androidx.navigation.NavType
```

```
import androidx.navigation.compose.NavHost
```

```
import androidx.navigation.compose.composable
```

```
import androidx.navigation.navArgument
```

```
import com.diplom.care4nails.ui.screens.chat.ChatScreen
```

```
import com.diplom.care4nails.ui.screens.diagnostic.DiagnosticScreen
```

```
import com.diplom.care4nails.ui.screens.history.HistoryScreen
```

```
import com.diplom.care4nails.ui.screens.home.HomeScreen
```

```
import com.diplom.care4nails.ui.screens.loading.LoadingScreen
```

```
import com.diplom.care4nails.ui.screens.nonnetwork.NoNetworkScreen
```

```
import com.diplom.care4nails.ui.screens.settings.SettingsScreen
```

```
@Composable
```

```
fun NavigationHost(navController: NavHostController) {
```

```
    NavHost(navController = navController, startDestination =
ScreenRoutes.LoadingScreen.route) {
```

```
        composable(route = ScreenRoutes.LoadingScreen.route) {
```

```
            LoadingScreen(navController)
```

```
        }
```

```
        composable(route = ScreenRoutes.HomeScreen.route) {
```

```
            HomeScreen(navController)
```

```
        }
```

```

        composable(route = ScreenRoutes.AddInfoScreen.route) {
            DiagnosticScreen(navController)
        }

        composable(route = ScreenRoutes.NoNetworkScreen.route) {
            NoNetworkScreen(navController)
        }

        composable(route = ScreenRoutes.HistoryScreen.route) {
            HistoryScreen(navController)
        }

        composable(
            route = "${ScreenRoutes.ChatScreen.route}/{problem}",
            arguments = listOf(
                navArgument("problem") {
                    type = NavType.StringType
                }
            )
        ) { backStackEntry ->
            val problem = backStackEntry.arguments?.getString("problem") ?: ""
            ChatScreen(problem = problem, navController = navController)
        }

        composable(route = ScreenRoutes.SettingsScreen.route) {
            SettingsScreen(navController)
        }
    }
}

```

ScreenRoutes.kt

```
package com.diplom.care4nails.navigation
```

```

sealed class ScreenRoutes(val route: String) {

    data object LoadingScreen : ScreenRoutes(LOADING_SCREEN)
    data object HomeScreen : ScreenRoutes(HOME_SCREEN)
    data object HistoryScreen : ScreenRoutes(HISTORY_SCREEN)
    data object SettingsScreen : ScreenRoutes(SETTINGS_SCREEN)
}

```

```

data object AddInfoScreen : ScreenRoutes(ADD_INFO_SCREEN)
data object ChatScreen : ScreenRoutes(CHAT_SCREEN)
data object NoNetworkScreen : ScreenRoutes(NO_NETWORK_SCREEN)

companion object {
    private const val LOADING_SCREEN = "LOADING_SCREEN"
    private const val HOME_SCREEN = "Home_SCREEN"
    private const val HISTORY_SCREEN = "History_SCREEN"
    private const val SETTINGS_SCREEN = "SETTINGS_SCREEN"
    private const val ADD_INFO_SCREEN = "ADD_INFO_SCREEN"
    private const val CHAT_SCREEN = "CHAT_SCREEN"
    private const val NO_NETWORK_SCREEN = "NO_NETWORK_SCREEN"
}
}

```

ChatScreen.kt

```

package com.diplom.care4nails.ui.screens.chat

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.Icon
import androidx.compose.material.IconButton
import androidx.compose.material.Text
import androidx.compose.material.TextField
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Send
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf

```

```

import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavController
import androidx.navigation.compose.rememberNavController
import com.diplom.care4nails.data.NetworkCheckerRepositoryImpl
import com.diplom.care4nails.domain.model.Message
import com.diplom.care4nails.navigation.ScreenRoutes
import com.diplom.care4nails.ui.theme.Burgundy
import com.diplom.care4nails.ui.theme.DARK_BLUE
import com.diplom.care4nails.ui.theme.Green
import com.diplom.care4nails.ui.theme.Light_BLUE
import com.diplom.care4nails.util.Constants
import com.diplom.care4nails.util.getBitmapFromPath
import kotlinx.coroutines.delay

@Composable
fun ChatScreen(
    navController: NavController,
    problem: String,
    viewModel: ChatViewModel = hiltViewModel()
) {
    val isLight = viewModel.getIsLightBack()
    val colors = if (isLight) {
        listOf(DARK_BLUE, Light_BLUE)
    } else {
        listOf(Green, Burgundy)
    }

    val listOfImages by viewModel.imagesListState.collectAsState(listOf())

```

```

val context = LocalContext.current

var messageText by remember { mutableStateOf("") }

LaunchedEffect(Unit) {
    if (problem != "no_problem") {
        val network =
            NetworkCheckerRepositoryImpl(context).isConnectionAvailable()
        if (network) {
            viewModel.getAllPhotos()
            delay(2000)
            val imagePath = listOfImages[listOfImages.size - 1].photo
            val bitmap = getBitmapFromPath(imagePath)

            if (bitmap != null) {
                viewModel.sendMessageWithPhoto(
                    context,
                    problem,
                    Constants.PROMPT_WITH_PHOTO,
                    bitmap
                )
            }
        } else {
            navController.navigate(ScreenRoutes.NoNetworkScreen.route)
        }
    }
}

Box(
    modifier = Modifier
        .fillMaxSize()
        .background(
            brush = Brush.horizontalGradient(
                colors = colors
            )
        )
) {
    Column(
        modifier = Modifier.fillMaxSize()
    )
}

```

```

) {

    Text(
        text = "Chat",
        fontSize = 24.sp,
        fontWeight = FontWeight.Bold,
        color = Color.White,
        textAlign = TextAlign.Center,
        modifier = Modifier
            .fillMaxWidth()
            .padding(top = 8.dp)
    )

    LazyColumn(
        modifier = Modifier
            .weight(1f)
            .padding(8.dp)
    ) {
        items(viewModel.messages) { message ->
            MessageItem(message = message)
        }
    }

    Row(
        modifier = Modifier
            .fillMaxWidth()
            .padding(8.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        TextField(
            value = messageText,
            onValueChange = { messageText = it },
            placeholder = { Text("Enter message...") },
            modifier = Modifier.weight(1f)
        )
        IconButton(
            onClick = {
                if (messageText.isNotBlank()) {
                    val network =

```

```
NetworkCheckerRepositoryImpl(context).isConnectionAvailable()
```

```

        if (network) {
            viewModel.sendMessage(
                context,
                messageText,
                Constants.PROMPT_WITHOUT_PHOTO
            )
            messageText = ""
        } else {
            navController.navigate(ScreenRoutes.NoNetworkScreen.route)
        }
    }
}
) {
    Icon(imageVector = Icons.Default.Send, contentDescription = "Send
Message")
}
}
}
}

}

@Composable
fun MessageItem(message: Message) {
    Box(
        modifier = Modifier
            .fillMaxWidth()
            .padding(vertical = 4.dp),
        contentAlignment = if (message.isUser) Alignment.CenterEnd else
Alignment.CenterStart
    ) {
        Text(
            text = message.text,
            modifier = Modifier
                .background(
                    if (message.isUser) Color.Blue else Color.LightGray,
                    shape = RoundedCornerShape(8.dp)
                )
            .padding(12.dp),
            color = if (message.isUser) Color.White else Color.Black

```

```

        )
    }
}

@Preview
@Composable
fun PreviewLoading() {
    ChatScreen(rememberNavController(), "no problem")
}

```

ChatViewModel.kt

```

package com.diplom.care4nails.ui.screens.chat

import android.content.Context
import android.graphics.Bitmap
import android.util.Log
import android.widget.Toast
import androidx.compose.runtime.mutableStateListOf
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.diplom.care4nails.domain.model.Message
import com.diplom.care4nails.domain.model.MessageEntity
import com.diplom.care4nails.domain.model.PhotoEntity
import com.diplom.care4nails.domain.repository.DatabaseRepository
import com.diplom.care4nails.domain.repository.GeminiRepository
import com.diplom.care4nails.domain.repository.PhotoRepository
import com.diplom.care4nails.domain.repository.StorageRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

import javax.inject.Inject

@HiltViewModel
class ChatViewModel @Inject constructor(
    private val gptRepository: GeminiRepository,
    private val databaseRepository: DatabaseRepository,
    private val photoRepository: PhotoRepository,

```

```

        private val storageRepository: StorageRepository
    ) : ViewModel() {

        private val _messages = mutableStateListOf<Message>()
        val messages: List<Message> = _messages

        private val _imagesListState = MutableStateFlow<List<PhotoEntity>>>(listOf())
        val imagesListState: StateFlow<List<PhotoEntity>>> = _imagesListState

        init {
            viewModelScope.launch {
                val savedMessages = databaseRepository.getAllMessages()
                _messages.addAll(savedMessages.map { Message(it.text, it.isUser) })
            }
        }

        fun getAllPhotos() {
            viewModelScope.launch {
                val all = photoRepository.getAllPhotos()
                Log.v("123123", "$all")
                _imagesListState.value = all
            }
        }

        fun getIsLightBack(): Boolean {
            return storageRepository.isLightBack()
        }

        fun sendMessage(context: Context, text: String, prompt: String) {
            _messages.add(Message(text, isUser = true))
            saveMessageToDatabase(MessageEntity(text = text, isUser = true))

            viewModelScope.launch(Dispatchers.IO) {
                try {
                    val result = gptRepository.getAnswerWithoutPhoto(text, prompt)
                    _messages.add(Message(result, isUser = false))
                    saveMessageToDatabase(MessageEntity(text = result, isUser = false))
                } catch (e: Exception) {
                    Log.e("LoadingViewModel", "Error fetching advice", e)
                    val errorMessage =

```

```

        if (e.message?.contains("User location is not supported for the API
use.") == true) {
            "API не підтримується в вашому регіоні. Спробуйте VPN."
        } else {
            "Щось пішло не так..."
        }
        withContext(Dispatchers.Main) {
            Toast.makeText(context, errorMessage, Toast.LENGTH_LONG).show()
        }
    }
}

fun sendMessageWithPhoto(context: Context, text: String, prompt: String, image:
Bitmap) {
    _messages.add(Message(text, isUser = true))
    saveMessageToDatabase(MessageEntity(text = text, isUser = true))

    viewModelScope.launch(Dispatchers.IO) {
        try {
            val result = gptRepository.getAnswerWithPhoto(text, prompt, image)
            Log.v("123123", "RESULT:$result")
            _messages.add(Message(result, isUser = false))
            saveMessageToDatabase(MessageEntity(text = result, isUser = false))
        } catch (e: Exception) {
            Log.e("LoadingViewModel", "Error fetching advice", e)
            val errorMessage =
                if (e.message?.contains("User location is not supported for the API
use.") == true) {
                    "API не підтримується в вашому регіоні. Спробуйте VPN."
                } else {
                    "Щось пішло не так..."
                }
            withContext(Dispatchers.Main) {
                Toast.makeText(context, errorMessage, Toast.LENGTH_LONG).show()
            }
        }
    }
}

private fun saveMessageToDatabase(message: MessageEntity) {

```

```

        viewModelScope.launch {
            databaseRepository.insertMessage(message)
        }
    }
}

```

CameraPreview.kt

```
package com.diplom.care4nails.ui.screens.diagnostic
```

```

import android.content.Context
import android.graphics.Bitmap
import android.util.Log
import androidx.camera.core.CameraSelector
import androidx.camera.core.ImageCapture
import androidx.camera.core.Preview
import androidx.camera.lifecycle.ProcessCameraProvider
import androidx.camera.view.PreviewView
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clipToBounds
import androidx.compose.ui.draw.shadow
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalLifecycleOwner
import androidx.compose.ui.res.stringResource

```

```

import androidx.compose.ui.unit.dp
import androidx.compose.ui.viewinterop.AndroidView
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.LifecycleOwner
import com.diplom.care4nails.MainActivity
import com.diplom.care4nails.R
import com.diplom.care4nails.util.Constants.CAMERAX_PERMISSIONS

@Composable
fun CameraPreview(
    onImageCaptured: (Bitmap, String) -> Unit,
    onError: (String) -> Unit,
    modifier: Modifier = Modifier,
    viewModel: DiagnosticViewModel = hiltViewModel()
) {
    val context = LocalContext.current
    val activity = LocalContext.current as MainActivity

    val lifecycleOwner = LocalLifecycleOwner.current
    val cameraProviderFuture = remember { ProcessCameraProvider.getInstance(context) }
    var imageCapture by remember { mutableStateOf<ImageCapture?>(null) }

    Box(modifier = Modifier.fillMaxSize()) {
        AndroidView(
            factory = { ctx ->
                PreviewView(ctx).apply {
                    this.scaleType = PreviewView.ScaleType.FILL_CENTER
                    this.implementationMode = PreviewView.ImplementationMode.COMPATIBLE
                }
            },
            update = { previewView ->
                startCamera(context, lifecycleOwner, previewView) { capture ->
                    imageCapture = capture
                }
            },
            modifier = modifier
                .fillMaxWidth()
                .aspectRatio(1f)
                .clipToBounds()
        )
    }
}

```

```

        .border(2.dp, Color.Cyan, RoundedCornerShape(20.dp))
        .shadow(1.dp, RoundedCornerShape(20.dp))
    )
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .align(Alignment.BottomCenter),
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.Center
    ) {
        Button(
            onClick = {
                imageCapture?.let {
                    if (!activity.hasRequiredPermissions(activity)) {
                        ActivityCompat.requestPermissions(
                            activity, CAMERAX_PERMISSIONS, 0
                        )
                    } else {
                        viewModel.takePhoto(it, onImageCaptured, onError)
                    }
                } ?: run {
                    onError("ImageCapture не инициализировано")
                }
            },
            colors = ButtonDefaults.buttonColors(
                containerColor = Color.Blue,
                contentColor = Color.Black,
                disabledContentColor = Color.Black,
                disabledContainerColor = Color.Blue
            ),
        ) {
            Text(text = stringResource(id = R.string.take_photo), color = Color.White)
        }
    }
}

private fun startCamera(
    context: Context,
    lifecycleOwner: LifecycleOwner,

```

```

        previewView: PreviewView,
        onImageCaptureReady: (ImageCapture) -> Unit
    ) {
        val cameraProviderFuture = ProcessCameraProvider.getInstance(context)
        cameraProviderFuture.addListener({
            val cameraProvider = cameraProviderFuture.get()

            val preview = Preview.Builder()
                .build()
                .also {
                    it.setSurfaceProvider(previewView.surfaceProvider)
                }

            val imageCapture = ImageCapture.Builder()
                .setTargetRotation(previewView.display.rotation)
                .build()

            onImageCaptureReady(imageCapture)

            val cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA

            try {
                cameraProvider.unbindAll()
                cameraProvider.bindToLifecycle(
                    lifecycleOwner, cameraSelector, preview, imageCapture
                )
            } catch (e: Exception) {
                Log.e("CameraPreview", "Camera binding failed", e)
            }
        }, ContextCompat.getMainExecutor(context))
    }
}

```

DiagnosticScreen.kt

```
package com.diplom.care4nails.ui.screens.diagnostic
```

```

import android.graphics.Bitmap
import android.widget.Toast
import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row

```

```

import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.heightIn
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Checkbox
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavHostController
import androidx.navigation.compose.rememberNavController
import com.diplom.care4nails.R
import com.diplom.care4nails.navigation.ScreenRoutes
import com.diplom.care4nails.ui.theme.Burgundy
import com.diplom.care4nails.ui.theme.DARK_BLUE

```

```

import com.diplom.care4nails.ui.theme.Green
import com.diplom.care4nails.ui.theme.Light_BLUE
import com.skydoves.landscapist.glide.GlideImage

@Composable
fun DiagnosticScreen(
    navController: NavHostController,
    viewModel: DiagnosticViewModel = hiltViewModel()
) {
    val context = LocalContext.current
    val isLight = viewModel.getIsLightBack()
    val colors = if (isLight) {
        listOf(DARK_BLUE, Light_BLUE)
    } else {
        listOf(Green, Burgundy)
    }

    val listOfImages by viewModel.imagesListState.collectAsState(listOf())

    var description by remember { mutableStateOf("") }
    var isChecked by remember { mutableStateOf(false) }
    var detectedImage by remember { mutableStateOf<Bitmap?>(null) }
    var isNailDetected by remember { mutableStateOf(false) }

    val isFormValid = description.isNotBlank() && isChecked

    val cameraContent = remember {
        mutableStateOf<@Composable () -> Unit>({})
    }

    LaunchedEffect(Unit) {
        cameraContent.value = {
            CameraPreview(
                onImageCaptured = { photo, url ->
                    viewModel.detectNail(photo).addOnSuccessListener { isDetected ->
                        if (isDetected) {
                            viewModel.savePhoto(url)
                            detectedImage = photo
                            isNailDetected = true
                        } else {

```

```

Toast.makeText(context, R.string.not_detected,
Toast.LENGTH_LONG)

        .show()
    }
}
},
onError = { error ->
    Toast.makeText(context, error, Toast.LENGTH_LONG).show()
},
modifier = Modifier
    .fillMaxWidth()
    .aspectRatio(1f)
)
}
}

Column(
    modifier = Modifier
        .fillMaxSize()
        .background(
            brush = Brush.horizontalGradient(
                colors = colors
            )
        )
) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(horizontal = 16.dp)
            .verticalScroll(rememberScrollState())
            .weight(0.9f)
    ) {
        Text(
            text = stringResource(id = R.string.diagnostic),
            color = Color.White,
            fontSize = 24.sp,
            fontWeight = FontWeight.Bold,
            textAlign = TextAlign.Center,
            modifier = Modifier
                .fillMaxWidth()
                .fillMaxHeight(0.1f)
        )
    }
}
}

```

```

        .align(Alignment.CenterHorizontally)
        .padding(16.dp)
    )

    if (!isNailDetected) {
        cameraContent.value()
    } else {
        viewModel.getAllPhotos()
        if (listOfImages.isNotEmpty()) {
            GlideImage(
                imageModel = { listOfImages[listOfImages.size - 1].photo },
                modifier = Modifier
                    .fillMaxWidth()
                    .aspectRatio(1f)
                    .padding(16.dp)
                    .border(1.dp, Color.Gray)
            )
        } else {
            viewModel.getAllPhotos()
            Column(
                modifier = Modifier
                    .fillMaxWidth()
                    .height(50.dp)
            ) {
                CircularProgressIndicator()
            }
        }
    }
}

Spacer(modifier = Modifier.height(16.dp))

Row(
    verticalAlignment = Alignment.CenterVertically,
    modifier = Modifier.fillMaxWidth()
) {
    Checkbox(
        checked = isChecked,
        onCheckedChange = { isChecked = it }
    )
    Text(

```

```

text = stringResource(id =
R.string.i_need_a_specialist_to_contact_me),
        Modifier.padding(start = 8.dp)
    )
}

Spacer(modifier = Modifier.height(8.dp))

if (isChecked) {
    TextField(
        value = description,
        onChange = { description = it },
        label = { Text(stringResource(id = R.string.describe_case)) },
        modifier = Modifier
            .fillMaxWidth()
            .heightIn(min = 80.dp)
    )
    Spacer(modifier = Modifier.height(18.dp))
}

Button(
    onClick = {
        if (detectedImage != null && isChecked) {
            val problem = description
            val route = "${ScreenRoutes.ChatScreen.route}/$problem"
            navController.navigate(route)
        } else {
            if (detectedImage != null) {
                navController.navigate(ScreenRoutes.HomeScreen.route)
            } else {
                Toast.makeText(
                    context,
                    "Please take a photo of your nail",
                    Toast.LENGTH_LONG
                ).show()
            }
        }
    },
    enabled = if (isChecked) isFormValid else detectedImage != null,
    colors = ButtonDefaults.buttonColors(
        containerColor = Color.Blue,

```

```

        contentColor = Color.Black,
        disabledContentColor = Color.Black,
        disabledContainerColor = Color.Blue
    ),
    modifier = Modifier.fillMaxWidth()
) {
    Text(
        text = if (isChecked) stringResource(id = R.string.submit) else
stringResource(
            id = R.string.save
        ),
        color = Color.White
    )
}
}
}
}
}

```

```

@Preview
@Composable
fun PreviewLoading() {
    DiagnosticScreen(rememberNavController())
}

```

DiagnosticViewModel.kt

```
package com.diplom.care4nails.ui.screens.diagnostic
```

```

import android.graphics.Bitmap
import androidx.camera.core.ImageCapture
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.diplom.care4nails.domain.model.PhotoEntity
import com.diplom.care4nails.domain.repository.CameraRepository
import com.diplom.care4nails.domain.repository.PhotoRepository
import com.diplom.care4nails.domain.repository.StorageRepository
import com.google.android.gms.tasks.Task
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch
import java.text.SimpleDateFormat

```

```

import java.util.Date
import java.util.Locale
import javax.inject.Inject

@HiltViewModel
class DiagnosticViewModel @Inject constructor(
    private val cameraRepository: CameraRepository,
    private val photoRepository: PhotoRepository,
    private val storageRepository: StorageRepository
) : ViewModel() {

    private val _imagesListState = MutableStateFlow<List<PhotoEntity>>>(listOf())
    val imagesListState: StateFlow<List<PhotoEntity>>> = _imagesListState

    fun takePhoto(
        imageCapture: ImageCapture,
        onPhotoTaken: (Bitmap, String) -> Unit,
        onError: (String) -> Unit
    ) {
        cameraRepository.takePhoto(imageCapture, onPhotoTaken, onError)
    }

    fun getAllPhotos() {
        viewModelScope.launch {
            _imagesListState.value = photoRepository.getAllPhotos()
        }
    }

    fun detectNail(bitmap: Bitmap): Task<Boolean> {
        return cameraRepository.detectNail(bitmap)
    }

    fun getIsLightBack(): Boolean {
        return storageRepository.isLightBack()
    }

    fun savePhoto(image: String) {
        viewModelScope.launch {

```

```

        val photo = PhotoEntity(
            photo = image,
            date = SimpleDateFormat("yyyy-MM-dd HH:mm", Locale.getDefault()).format(
                Date()
            ).toString()
        )
        photoRepository.savePhotoToRoom(photo)
    }
}

```

HistoryScreen.kt

```

package com.diplom.care4nails.ui.screens.history

import android.util.Log
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.grid.GridCells
import androidx.compose.foundation.lazy.grid.LazyVerticalGrid
import androidx.compose.foundation.lazy.grid.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.stringResource

```

```

import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavController
import androidx.navigation.compose.rememberNavController
import com.diplom.care4nails.R
import com.diplom.care4nails.domain.model.PhotoEntity
import com.diplom.care4nails.ui.theme.Burgundy
import com.diplom.care4nails.ui.theme.DARK_BLUE
import com.diplom.care4nails.ui.theme.Green
import com.diplom.care4nails.ui.theme.Light_BLUE
import com.skydoves.landscapist.glide.GlideImage

@Composable
fun HistoryScreen(navController: NavController, viewModel: HistoryViewModel =
hiltViewModel()) {

    val listOfImages by viewModel.imagesListState.collectAsState(listOf())
    viewModel.getAllPhotos()

    val isLight = viewModel.getIsLightBack()
    val colors = if (isLight) {
        listOf(DARK_BLUE, Light_BLUE)
    } else {
        listOf(Green, Burgundy)
    }

    LaunchedEffect(listOfImages) {
        Log.d("dsadsa", listOfImages.toString())
    }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .background(
                brush = Brush.horizontalGradient(
                    colors = colors
                )
            )
    )

```

```

        )
    ) {
        Box(Modifier.fillMaxWidth(), contentAlignment = Alignment.TopCenter) {
            Text(
                text = stringResource(id = R.string.history),
                color = Color.White,
                fontSize = 24.sp,
                textAlign = TextAlign.Center,
                fontWeight = FontWeight.Bold, modifier = Modifier
                    .fillMaxWidth()
                    .padding(top = 36.dp)
            )
        }

        Box(modifier = Modifier.weight(0.9f)) {
            LazyVerticalGrid(
                columns = GridCells.Adaptive(minSize = 128.dp),
                verticalArrangement = Arrangement.spacedBy(3.dp),
                horizontalArrangement = Arrangement.spacedBy(3.dp)
            ) {
                items(listOfImages) { photoEntity ->
                    PhotoItem(photoEntity, navController)
                }
            }
        }
    }
}

@Composable
fun PhotoItem(photoEntity: PhotoEntity, navController: NavController) {
    Column(modifier = Modifier.padding(8.dp)) {
        Text(
            text = photoEntity.date,
            fontWeight = FontWeight.Bold,
            style = MaterialTheme.typography.body2,
            color = Color.White,
            modifier = Modifier.padding(bottom = 4.dp)
        )

        GlideImage(

```

```

        imageModel = { photoEntity.photo },
        modifier = Modifier
            .fillMaxWidth()
            .height(200.dp)
            .clip(RoundedCornerShape(8.dp))
            .background(Color.LightGray)
    )
}
}

```

```

@Preview
@Composable
fun PreviewLoading() {
    HistoryScreen(rememberNavController())
}

```

HistoryViewModel.kt

```
package com.diplom.care4nails.ui.screens.history
```

```

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.diplom.care4nails.domain.model.PhotoEntity
import com.diplom.care4nails.domain.repository.PhotoRepository
import com.diplom.care4nails.domain.repository.StorageRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

```

```
@HiltViewModel
```

```

class HistoryViewModel @Inject constructor(
    private val photoRepository: PhotoRepository,
    private val storageRepository: StorageRepository
) : ViewModel() {

    private val _imagesListState = MutableStateFlow<List<PhotoEntity>>>(listOf())
    val imagesListState: StateFlow<List<PhotoEntity>>> = _imagesListState

```

```

    fun getAllPhotos() {
        viewModelScope.launch {
            _imagesListState.value = photoRepository.getAllPhotos()
        }
    }

    fun getIsLightBack(): Boolean {
        return storageRepository.isLightBack()
    }
}

```

HomeScreen.kt

```
package com.diplom.care4nails.ui.screens.home
```

```

import androidx.activity.compose.BackHandler
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign

```

```

import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavHostController
import androidx.navigation.compose.rememberNavController
import com.diplom.care4nails.R
import com.diplom.care4nails.navigation.ScreenRoutes
import com.diplom.care4nails.ui.theme.Burgundy
import com.diplom.care4nails.ui.theme.DARK_BLUE
import com.diplom.care4nails.ui.theme.Green
import com.diplom.care4nails.ui.theme.Light_BLUE

@Composable
fun HomeScreen(navController: NavHostController, viewModel: HomeViewModel =
hiltViewModel()) {

    val isLight = viewModel.getIsLightBack()
    val colors = if (isLight) {
        listOf(DARK_BLUE, Light_BLUE)
    } else {
        listOf(Green, Burgundy)
    }
    BackHandler {}

    Box(
        modifier = Modifier
            .fillMaxSize()
            .background(
                brush = Brush.horizontalGradient(
                    colors = colors
                )
            )
    ) {

        Column(
            horizontalAlignment = Alignment.CenterHorizontally,
            modifier = Modifier
                .fillMaxSize()
        ) {
            Spacer(modifier = Modifier.height(20.dp))

```

```

Box(Modifier.fillMaxWidth(), contentAlignment = Alignment.TopCenter) {
    Text(
        text = stringResource(id = R.string.app_name),
        color = Color.White,
        fontSize = 24.sp,
        textAlign = TextAlign.Center,
        fontWeight = FontWeight.Bold, modifier = Modifier
            .fillMaxWidth()
            .padding(top = 36.dp)
    )
}
Box(
    modifier = Modifier
        .fillMaxWidth()
        .fillMaxHeight()
) {
    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Row(
            Modifier.fillMaxWidth(),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.SpaceAround
        ) {
            MenuButton(
                icon = R.drawable.diagnostic,
                text = R.string.diagnostic
            ) {
                navController.navigate(ScreenRoutes.AddInfoScreen.route)
            }

            MenuButton(icon = R.drawable.history, text = R.string.history) {
                navController.navigate(ScreenRoutes.HistoryScreen.route)
            }
        }

        Spacer(modifier = Modifier.height(16.dp))

        Row(

```

```
@Composable
fun MenuButton(icon: Int, text: Int, onClick: () -> Unit) {
    Column(
        modifier = Modifier
            .background(
                brush = Brush.linearGradient(
                    listOf(
                        DARK_BLUE,
                        Light_BLUE
                    )
                ),
                shape = RoundedCornerShape(8.dp)
            )
        .size(150.dp)
        .clickable { onClick() },
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    )
}
```

```

    ) {
        Image(
            painter = painterResource(id = icon),
            contentDescription = icon.toString(),
            modifier = Modifier.size(60.dp)
        )
        Text(
            text = stringResource(id = text),
            textAlign = TextAlign.Center,
            color = Color.White,
            fontSize = 14.sp,
            fontWeight = FontWeight.Bold
        )
    }
}

```

```

@Composable
@Preview
fun previewHome() {
    HomeScreen(navController = rememberNavController())
}

```

HomeViewModel.kt

```
package com.diplom.care4nails.ui.screens.home
```

```

import androidx.lifecycle.ViewModel
import com.diplom.care4nails.domain.repository.StorageRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import javax.inject.Inject

```

```
@HiltViewModel
```

```

class HomeViewModel @Inject constructor(private val storageRepository: StorageRepository)
:
    ViewModel() {

    fun getIsLightBack(): Boolean {
        return storageRepository.isLightBack()
    }

}

```

LoadingScreen.kt

```
package com.diplom.care4nails.ui.screens.loading
```

```
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavController
import androidx.navigation.compose.rememberNavController
import com.airbnb.lottie.compose.LottieAnimation
import com.airbnb.lottie.compose.LottieCompositionSpec
import com.airbnb.lottie.compose.LottieConstants
import com.airbnb.lottie.compose.animateLottieCompositionAsState
import com.airbnb.lottie.compose.rememberLottieComposition
import com.diplom.care4nails.R
import com.diplom.care4nails.data.NetworkCheckerRepositoryImpl
import com.diplom.care4nails.navigation.ScreenRoutes
import kotlinx.coroutines.delay
```

```
@Composable
```

```

fun LoadingScreen(navController: NavController, viewModel: LoadingViewModel =
hiltViewModel()) {
    val composition by
rememberLottieComposition(LottieCompositionSpec.RawRes(R.raw.loading))
    val progress by animateLottieCompositionAsState(
        composition,
        iterations = LottieConstants.IterateForever
    )
    val context = LocalContext.current
    val advice = viewModel.advice.collectAsState().value

    LaunchedEffect(Unit) {
        val network = NetworkCheckerRepositoryImpl(context).isConnectionAvailable()
        if (network) {
            viewModel.getHelp(context)
            delay(5000)
            navController.navigate(ScreenRoutes.HomeScreen.route)
        } else {
            delay(5000)
            navController.navigate(ScreenRoutes.HomeScreen.route)
        }
    }

    Box(
        modifier = Modifier
            .fillMaxSize()
            .background(
                brush = Brush.horizontalGradient(
                    colors = listOf(
                        Color(0xFFB8860B),
                        Color(0xFF006400)
                    )
                )
            )
    ) {
        Column(
            modifier = Modifier.fillMaxSize(),
            verticalArrangement = Arrangement.SpaceBetween,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {

```

```

Text(
    text = stringResource(id = R.string.app_name),
    color = Color.White,
    fontSize = 24.sp,
    textAlign = TextAlign.Center,
    fontWeight = FontWeight.Bold,
    modifier = Modifier
        .fillMaxWidth()
        .padding(top = 36.dp)
)

if (advice.isNotEmpty()) {
    Text(text = advice, color = Color.LightGray, fontSize = 20.sp)
} else {
    Text(
        text = "Loading advice...",
        color = Color.LightGray,
        fontSize = 20.sp
    )
}

LottieAnimation(
    composition = composition,
    progress = { progress },
    modifier = Modifier
        .padding(bottom = 32.dp)
        .size(200.dp)
)
}
}
}
}

```

```

@Preview
@Composable
fun PreviewLoading() {
    LoadingScreen(rememberNavController())
}

```

```

LoadingViewModel.kt
package com.diplom.care4nails.ui.screens.loading

```

```

import android.content.Context
import android.util.Log
import android.widget.Toast
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.diplom.care4nails.domain.repository.GeminiRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import javax.inject.Inject

@HiltViewModel
class LoadingViewModel @Inject constructor(private val geminiRepository: GeminiRepository)
:
    ViewModel() {

        private val _advice = MutableStateFlow<String>("")
        val advice: StateFlow<String> = _advice

        fun getHelp(context: Context) {
            viewModelScope.launch {
                try {
                    val advice = geminiRepository.getAnswerWithoutPhoto(
                        "Give me advice on nail care, just one sentence,in ukrainian language",
                        "Answer like a doctor"
                    )
                    _advice.emit(advice)
                } catch (e: Exception) {
                    Log.e("LoadingViewModel", "Error fetching advice", e)
                    val errorMessage =
                        if (e.message?.contains("User location is not supported for the API
use.") == true) {
                            "API не підтримується в вашому регіоні. Спробуйте VPN."
                        } else {
                            "Щось пішло не так..."
                        }
                }
            }
        }
    }

```

```
withContext(Dispatchers.Main) {
    Toast.makeText(context, errorMessage, Toast.LENGTH_LONG).show()
}
}
}
}
```

NoNetworkScreen.kt

```
package com.diplom.care4nails.ui.screens.nonetwork
```

```
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavController
import androidx.navigation.compose.rememberNavController
import com.airbnb.lottie.compose.LottieAnimation
import com.airbnb.lottie.compose.LottieCompositionSpec
```

```

import com.airbnb.lottie.compose.LottieConstants
import com.airbnb.lottie.compose.animateLottieCompositionAsState
import com.airbnb.lottie.compose.rememberLottieComposition
import com.diplom.care4nails.R
import com.diplom.care4nails.data.NetworkCheckerRepositoryImpl
import com.diplom.care4nails.ui.theme.Burgundy
import com.diplom.care4nails.ui.theme.DARK_BLUE
import com.diplom.care4nails.ui.theme.Green
import com.diplom.care4nails.ui.theme.Light_BLUE

@Composable
fun NoNetworkScreen(navController: NavController, viewModel: NoNetworkViewModel =
hiltViewModel()) {

    val isLight = viewModel.getIsLightBack()
    val colors = if (isLight) {
        listOf(DARK_BLUE, Light_BLUE)
    } else {
        listOf(Green, Burgundy)
    }

    val context = LocalContext.current

    Box(
        modifier = Modifier
            .fillMaxSize()
            .background(
                brush = Brush.verticalGradient(
                    colors = colors
                )
            )
    ) {
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(16.dp),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.SpaceBetween
        ) {

            val composition by
rememberLottieComposition(LottieCompositionSpec.RawRes(R.raw.no_network_screen))

```

```

        val progress by animateLottieCompositionAsState(
            composition = composition,
            iterations = LottieConstants.IterateForever
        )

        LottieAnimation(
            composition = composition,
            progress = progress,
            modifier = Modifier
                .size(200.dp)
        )

        Text(
            text = "Немає інтернет з'єднання...",
            fontSize = 24.sp,
            fontWeight = FontWeight.Bold,
            color = Color.White,
            textAlign = TextAlign.Center
        )

        Image(
            painter = painterResource(id = R.drawable.refresh),
            contentDescription = "Refresh Image Button",
            modifier = Modifier
                .size(100.dp)
                .clickable {

                    val network =
NetworkCheckerRepositoryImpl(context).isConnectionAvailable()
                    if (network) {
                        navController.navigateUp()
                    }
                }
        )
    }
}

@Preview
@Composable
fun PreviewNoNetwork() {

```

```

        NoNetworkScreen(navController = rememberNavController())
    }

```

NoNetworkViewModel.kt

```
package com.diplom.care4nails.ui.screens.nonetwork
```

```

import androidx.lifecycle.ViewModel
import com.diplom.care4nails.domain.repository.StorageRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import javax.inject.Inject

```

```
@HiltViewModel
```

```
class NoNetworkViewModel @Inject constructor(private val storageRepository:
StorageRepository) :
```

```
    ViewModel() {
```

```

        fun getIsLightBack(): Boolean {
            return storageRepository.isLightBack()
        }

```

```
    }
```

SettingsScreen.kt

```
package com.diplom.care4nails.ui.screens.settings
```

```

import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment

```

```

import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavController
import androidx.navigation.compose.rememberNavController
import com.diplom.care4nails.R
import com.diplom.care4nails.navigation.ScreenRoutes
import com.diplom.care4nails.ui.theme.Burgundy
import com.diplom.care4nails.ui.theme.DARK_BLUE
import com.diplom.care4nails.ui.theme.Green
import com.diplom.care4nails.ui.theme.Light_BLUE

@Composable
fun SettingsScreen(
    navController: NavController,
    settingsViewModel: SettingsViewModel = hiltViewModel()
) {
    val isLight = settingsViewModel.getIsLightBack()
    val colors = if (isLight) {
        listOf(DARK_BLUE, Light_BLUE)
    } else {
        listOf(Green, Burgundy)
    }

    Box(
        modifier = Modifier
            .fillMaxSize()
            .background(
                brush = Brush.horizontalGradient(
                    colors = colors
                )
            )
    ) {

```

```

Column(
    horizontalAlignment = Alignment.CenterHorizontally,
    modifier = Modifier
        .fillMaxSize()
) {
    Spacer(modifier = Modifier.height(20.dp))

    Box(Modifier.fillMaxWidth(), contentAlignment = Alignment.TopCenter) {
        Text(
            text = stringResource(id = R.string.settings),
            color = Color.White,
            fontSize = 24.sp,
            textAlign = TextAlign.Center,
            fontWeight = FontWeight.Bold,
            modifier = Modifier
                .fillMaxWidth()
                .padding(top = 36.dp)
        )
    }
}

Box(
    modifier = Modifier
        .fillMaxWidth()
        .fillMaxHeight()
) {
    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Row(
            horizontalArrangement = Arrangement.spacedBy(16.dp),
            verticalAlignment = Alignment.CenterVertically,
            modifier = Modifier
                .fillMaxWidth()
                .padding(horizontal = 32.dp)
        ) {
            Image(
                painter = painterResource(id = R.drawable.dark_back),
                contentDescription = "First Image",
            )
        }
    }
}

```



```

@HiltViewModel
class SettingsViewModel @Inject constructor(private val storageRepository:
StorageRepository) :
    ViewModel() {

    fun setLightBack(isLight: Boolean) {
        storageRepository.setLightBack(isLight)
    }

    fun getIsLightBack(): Boolean {
        return storageRepository.isLightBack()
    }

}

```

Color.kt

```
package com.diplom.care4nails.ui.theme
```

```
import androidx.compose.ui.graphics.Color
```

```
val Purple80 = Color(0xFFD0BCFF)
```

```
val PurpleGrey80 = Color(0xFFCCC2DC)
```

```
val Pink80 = Color(0xFFE8B8C8)
```

```
val Purple40 = Color(0xFF6650a4)
```

```
val PurpleGrey40 = Color(0xFF625b71)
```

```
val Pink40 = Color(0xFF7D5260)
```

```
val Green = Color(0xFF006400)
```

```
val Burgundy = Color(0xFFB8860B)
```

```
val DARK_BLUE = Color(0xFF3F51B5)
```

```
val Light_BLUE = Color(0xFF03A9F4)
```

Theme.kt

```
package com.diplom.care4nails.ui.theme
```

```
import android.os.Build
```

```
import androidx.compose.foundation.isSystemInDarkTheme
```

```
import androidx.compose.material3.MaterialTheme
```

```
import androidx.compose.material3.darkColorScheme
```

```

import androidx.compose.material3.dynamicDarkColorScheme
import androidx.compose.material3.dynamicLightColorScheme
import androidx.compose.material3.lightColorScheme
import androidx.compose.runtime.Composable
import androidx.compose.ui.platform.LocalContext

private val DarkColorScheme = darkColorScheme(
    primary = Purple80,
    secondary = PurpleGrey80,
    tertiary = Pink80
)

private val LightColorScheme = lightColorScheme(
    primary = Purple40,
    secondary = PurpleGrey40,
    tertiary = Pink40

    /* Other default colors to override
    background = Color(0xFFFFFBFE),
    surface = Color(0xFFFFFBFE),
    onPrimary = Color.White,
    onSecondary = Color.White,
    onTertiary = Color.White,
    onBackground = Color(0xFF1C1B1F),
    onSurface = Color(0xFF1C1B1F),
    */
)

@Composable
fun Care4NailsTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    // Dynamic color is available on Android 12+
    dynamicColor: Boolean = true,
    content: @Composable () -> Unit
) {
    val colorScheme = when {
        dynamicColor && Build.VERSION.SDK_INT >= Build.VERSION_CODES.S -> {
            val context = LocalContext.current
            if (darkTheme) dynamicDarkColorScheme(context) else
dynamicLightColorScheme(context)
        }
    }

```

```

        darkTheme -> DarkColorScheme
        else -> LightColorScheme
    }

    MaterialTheme(
        colorScheme = colorScheme,
        typography = Typography,
        content = content
    )
}

Type.kt
package com.diplom.care4nails.ui.theme

import androidx.compose.material3.Typography
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp

// Set of Material typography styles to start with
val Typography = Typography(
    bodyLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp,
        lineHeight = 24.sp,
        letterSpacing = 0.5.sp
    )
    /* Other default text styles to override
    titleLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 22.sp,
        lineHeight = 28.sp,
        letterSpacing = 0.sp
    ),
    labelSmall = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Medium,

```

```

        fontSize = 11.sp,
        lineHeight = 16.sp,
        letterSpacing = 0.5.sp
    )
    */
)

```

Compressor.kt

```
package com.diplom.care4nails.util
```

```
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import java.io.ByteArrayOutputStream
```

```
fun compressBitmap(bitmap: Bitmap, quality: Int): Bitmap {
    val outputStream = ByteArrayOutputStream()
    bitmap.compress(Bitmap.CompressFormat.JPEG, quality, outputStream)
    val byteArray = outputStream.toByteArray()
    return BitmapFactory.decodeByteArray(byteArray, 0, byteArray.size)
}

```

Constants.kt

```
package com.diplom.care4nails.util
```

```
import android.Manifest
```

```
object Constants {
```

```

    const val DATABASE_NAME = "PHOTO_DATABASE"
    const val PROMPT_WITHOUT_PHOTO =
        "Уяви що ти подолог. Дай корисну пораду для здоров'я нігтів. Дай відповідь українською мовою"
    const val PROMPT_WITH_PHOTO =
        "Уяви що ти подолог. Дай корисну пораду для здоров'я нігтів. Дай відповідь українською мовою."

```

```

const val API_KEY =
    "AIzaSyC4nH0_uCv43noy_BW3clGA2gcHfSwyGEQ"

```

```
val CAMERAX_PERMISSIONS = arrayOf(
```

```

        Manifest.permission.CAMERA,
        Manifest.permission.WRITE_EXTERNAL_STORAGE
    )
}

```

Converter.kt

```
package com.diplom.care4nails.util
```

```
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import java.io.File
```

```
fun getBitmapFromPath(imagePath: String): Bitmap? {
    val file = File(imagePath)
    return if (file.exists()) {
        BitmapFactory.decodeFile(imagePath)
    } else {
        null
    }
}

```

App.kt

```
package com.diplom.care4nails
```

```
import android.app.Application
import dagger.hilt.android.HiltAndroidApp
```

```
@HiltAndroidApp
class App : Application()
```

MainActivity.kt

```
package com.diplom.care4nails
```

```
import android.content.Context
import android.content.pm.PackageManager
import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.core.app.ActivityCompat
```

```

import androidx.core.content.ContextCompat
import androidx.navigation.compose.rememberNavController
import com.diplom.care4nails.navigation.NavigationHost
import com.diplom.care4nails.ui.theme.Care4NailsTheme
import com.diplom.care4nails.util.Constants.CAMERAX_PERMISSIONS
import dagger.hilt.android.AndroidEntryPoint

```

```
@AndroidEntryPoint
```

```

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        requestPermissions()
        setContent {
            Care4NailsTheme {
                val navController = rememberNavController()
                NavigationHost(navController)
            }
        }
    }
}

```

```

private fun requestPermissions() {
    if (!hasRequiredPermissions(this@MainActivity)) {
        Log.d("adsas", "permisison")
        ActivityCompat.requestPermissions(
            this@MainActivity, CAMERAX_PERMISSIONS, 0
        )
    }
}

```

```

fun hasRequiredPermissions(
    context: Context,
): Boolean {
    return CAMERAX_PERMISSIONS.all {
        ContextCompat.checkSelfPermission(
            context,
            it
        ) == PackageManager.PERMISSION_GRANTED
    }
}

```

```
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-feature
        android:name="android.hardware.camera"
        android:required="false" />

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:name=".App"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Care4Nails"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.Care4Nails">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

```
</manifest>
```

```
CameraRepositoryImpl.kt
```

```
package com.diplom.care4nails.data.diagnostic
```

```
import android.Manifest
```

```
import android.content.ContentValues
```

```
import android.content.Context
```

```
import android.content.pm.PackageManager
```

```
import android.graphics.Bitmap
```

```
import android.graphics.BitmapFactory
```

```
import android.provider.MediaStore
```

```
import android.util.Log
```

```
import androidx.camera.core.ImageCapture
```

```
import androidx.camera.core.ImageCaptureException
```

```
import androidx.core.content.ContextCompat
```

```
import com.diplom.care4nails.domain.repository.CameraRepository
```

```
import com.google.android.gms.tasks.Task
```

```
import com.google.android.gms.tasks.TaskCompletionSource
```

```
import com.google.mlkit.vision.common.InputImage
```

```
import com.google.mlkit.vision.label.ImageLabeling
```

```
import com.google.mlkit.vision.label.defaults.ImageLabelerOptions
```

```
import java.io.File
```

```
import java.io.OutputStream
```

```
import javax.inject.Inject
```

```
class CameraRepositoryImpl @Inject constructor(private val context: Context) :  
CameraRepository {
```

```
    override fun takePhoto(  
        imageCapture: ImageCapture,  
        onPhotoTaken: (Bitmap, String) -> Unit,  
        onError: (String) -> Unit  
    ) {
```

```
        val photoFile = createFile(context.applicationContext.filesDir, "CameraXApp",  
".jpg")
```

```
        val outputOptions = ImageCapture.OutputFileOptions.Builder(photoFile).build()  
        imageCapture.takePicture(  
            outputOptions,
```

```

ContextCompat.getMainExecutor(context),
object : ImageCapture.OnImageSavedCallback {
    override fun onError(exc: ImageCaptureException) {
        Log.e("CameraXApp", "Photo capture failed: ${exc.message}", exc)
    }

    override fun onImageSaved(output: ImageCapture.OutputFileResults) {
        var bitmap = BitmapFactory.decodeFile(photoFile.absolutePath)
        if (ContextCompat.checkSelfPermission(
            context,
            Manifest.permission.WRITE_EXTERNAL_STORAGE
        ) == PackageManager.PERMISSION_GRANTED
        ) {
            onPhotoTaken(bitmap, photoFile.absolutePath)
            saveImageToGallery(bitmap)
        }
    }
})
}

private fun saveImageToGallery(bitmap: Bitmap) {
    val contentValues = ContentValues().apply {
        put(
            MediaStore.MediaColumns.DISPLAY_NAME,
            "CameraXApp_${System.currentTimeMillis()}.jpg"
        )
        put(MediaStore.MediaColumns.MIME_TYPE, "image/jpeg")
        put(MediaStore.MediaColumns.RELATIVE_PATH, "Pictures/CameraXApp")
    }

    val resolver = context.contentResolver
    val uri = resolver.insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
contentValues)

    if (uri != null) {
        var outputStream: OutputStream? = null
        try {
            outputStream = resolver.openOutputStream(uri)
            if (outputStream != null) {
                bitmap.compress(Bitmap.CompressFormat.JPEG, 100, outputStream)
            }
        }
    }
}

```

```

    } catch (e: Exception) {
        Log.e("CameraXApp", "Error saving image to gallery: ${e.message}", e)
    } finally {
        outputStream?.close()
    }
}

private fun createFile(baseFolder: File, format: String, extension: String) =
    File(baseFolder, "${System.currentTimeMillis()}${format}${extension}")

override fun detectNail(bitmap: Bitmap): Task<Boolean> {
    val image = InputImage.fromBitmap(bitmap, 0)
    val labeler = ImageLabeling.getClient(ImageLabelerOptions.DEFAULT_OPTIONS)

    val taskCompletionSource = TaskCompletionSource<Boolean>()

    labeler.process(image)
        .addOnSuccessListener { labels ->
            var nailDetected = false
            for (label in labels) {
                val text = label.text
                val confidence = label.confidence
                if (text.equals("nail", ignoreCase = true) && confidence > 0.8) {
                    nailDetected = true
                    break
                }
            }
            taskCompletionSource.setResult(nailDetected)
        }
        .addOnFailureListener { exception ->
            Log.e("detectNail", "Labeling failed", exception)
            taskCompletionSource.setResult(false)
        }

    return taskCompletionSource.task
}
}

```

```

package com.diplom.care4nails.data.diagnostic

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.Query
import com.diplom.care4nails.domain.model.PhotoEntity

@Dao
interface PhotoDao {
    @Insert
    suspend fun addPhoto(photo: PhotoEntity)

    @Query("SELECT * FROM photos")
    suspend fun getAllPhoto(): List<PhotoEntity>
}

PhotoDatabase.kt
package com.diplom.care4nails.data.diagnostic

import androidx.room.Database
import androidx.room.RoomDatabase
import com.diplom.care4nails.data.ai.MessageDao
import com.diplom.care4nails.domain.model.MessageEntity
import com.diplom.care4nails.domain.model.PhotoEntity

@Database(
    entities = [PhotoEntity::class, MessageEntity::class],
    version = 1, exportSchema = true
)
abstract class PhotoDatabase : RoomDatabase() {

    abstract fun photoDao(): PhotoDao

    abstract fun geminiDao(): MessageDao

}

```

```

PhotoRepositoryImpl.kt
package com.diplom.care4nails.data.diagnostic

```

```

import com.diplom.care4nails.domain.model.PhotoEntity
import com.diplom.care4nails.domain.repository.PhotoRepository
import javax.inject.Inject

class PhotoRepositoryImpl @Inject constructor(private val dao: PhotoDao) : PhotoRepository
{

    override suspend fun savePhotoToRoom(photo: PhotoEntity) {
        dao.addPhoto(photo)
    }

    override suspend fun getAllPhotos(): List<PhotoEntity> {
        return dao.getAllPhoto()
    }

}

```

strings.xml

```

<resources>
    <string name="app_name">Care4Nails</string>
    <string name="take_photo">Зробити фото</string>
    <string name="not_detected">Нігтя не виявлено</string>
    <string name="diagnostic">Діагностування</string>
    <string name="describe_case">Опишіть проблему</string>
    <string name="submit">Відправити</string>
    <string name="save">Зберегти</string>
    <string name="i_need_a_specialist_to_contact_me">Допомога від AI</string>
    <string name="settings">Налаштування</string>
    <string name="history">Історія</string>
    <string name="general_info">Загальна інформація</string>
    <string name="chat">Чат з спеціалістом</string>

</resources>

```

ДОДАТОК В
Слайди презентації

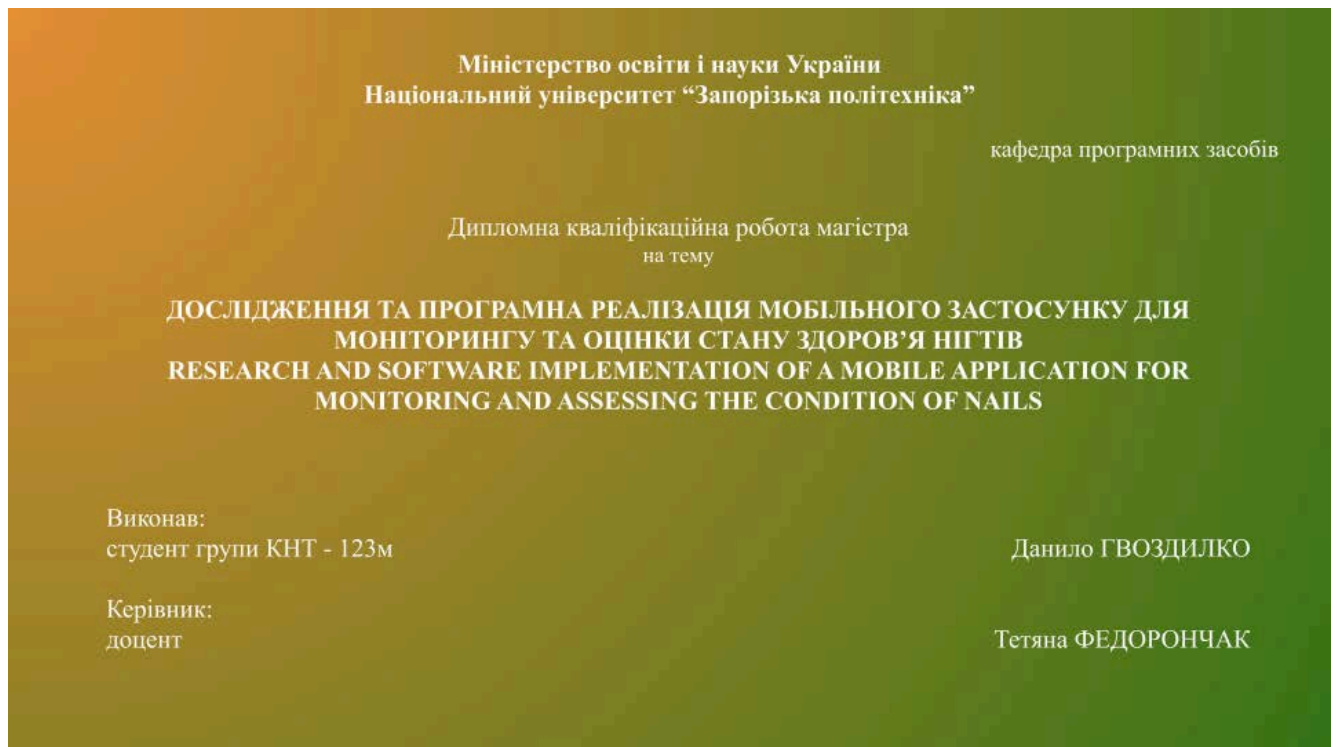


Рисунок В.1 – Слайд 1

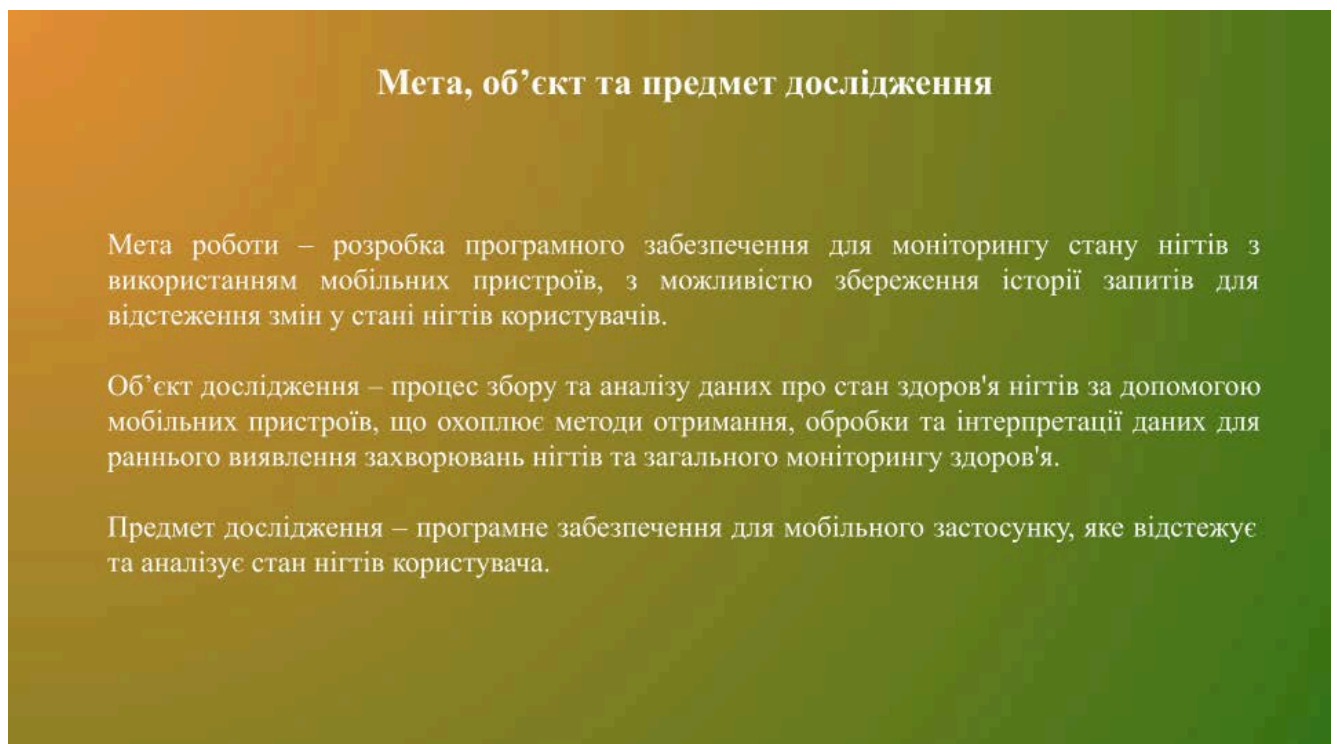


Рисунок В.2 - Слайд 2

Завдання дослідження

- проаналізувати існуючі додатки для моніторингу стану здоров'я шкіри, волосся та нігтів, а також визначити основні підходи до оцінки їхнього стану;
- розробити методи аналізу стану нігтів за допомогою зображень, отриманих із камери мобільного пристрою;
- виконати проєктування програмного забезпечення для моніторингу та оцінки стану нігтів із використанням методів комп'ютерного зору та машинного навчання;
- реалізувати мобільний застосунок для моніторингу стану нігтів, що включає інструменти аналізу, оцінки та рекомендацій;
- провести експериментальне дослідження точності алгоритмів аналізу стану нігтів, ефективності мобільного застосунку та його впливу на покращення здоров'я користувачів.

Рисунок В.3 – Слайд 3

Аналіз предметної області

Критерій	Nail Keeper	Healthy Nails	Frenchies Modern Nail Care	Anemia Detection App
Основна функція	Моніторинг змін нігтів через регулярні фото	Інформація про догляд та профілактику захворювань	Керування візитами до салонів та вибір послуг	Аналіз нігтів для виявлення анемії через штучний інтелект
Можливість аналізу стану здоров'я	Відсутня діагностика, можливість моніторингу змін	Відсутня, лише інформація про догляд за нігтями	Відсутня, обслуговування в салонах	Аналіз здоров'я через фото нігтів, лише для виявлення анемії
Цільова аудиторія	Люди, які хочуть самостійно відстежувати зміни	Користувачі, які шукають поради по догляду за нігтями	Користувачі, які бажають отримувати догляд у салонах	Люди, які перевіряють нігті для виявлення анемії
Фотографування нігтів	Відстеження змін, без автоматичного аналізу	Відсутнє	Відсутнє	Аналіз фото нігтів з метою діагностики анемії
Інтеграція з медичними показниками	Відсутня	Відсутня	Відсутня	Орієнтований на діагностику анемії через показники
Додаткові функції	Тільки базовий моніторинг стану нігтів	Освітні статті та поради з догляду	Бронювання послуг у салонах та управління профілем	Аналіз для визначення анемії, не охоплює інші захворювання
Тип застосунку	Самостійне відстеження та моніторинг	Освітній застосунок	Застосунок для запису до салонів краси	Застосунок для діагностики на основі зображень

Рисунок В.4 – Слайд 4

Етапи вирішення поставленої задачі

- використання камер смартфона для збору даних про стан нігтів;
- розробка застосунку для автоматичного аналізу стану нігтів;
- зберігання та аналіз даних. Організація бази даних для збереження зображень, результатів аналізу та історії стану нігтів користувача;
- створення зручного інтерфейсу для взаємодії з користувачем, що дозволяє легко завантажувати фотографії, отримувати результати аналізу і отримувати поради;
- перевірка роботи застосунку на різних пристроях і в різних умовах освітлення для забезпечення коректного аналізу зображень.

Рисунок В.5 – Слайд 5

Порівняння методів написання UI

Алгоритм	Призначення	Методи
Сегментація	Виділення об'єктів на зображеннях	Нейронні мережі
Глибоке навчання	Інтеграція даних з різних джерел	Трансформерні моделі
Мультимодальні моделі	Обробка тексту та зображень одночасно	Моделі для обробки тексту та зображень
Метод граничного детектування (Edge Detection)	Виявлення контурів нігтів на зображенні	Оператор Кенні, оператор Собеля
Алгоритми класифікації на основі CNN	Оцінка стану нігтів, виявлення грибкових інфекцій або пошкоджень	Згорткові нейронні мережі (CNN)
Моделі кластеризації K-Means	Аналіз кольорових аномалій нігтів	K-Means

Рисунок В.6 – Слайд 6

Алгоритми обробки зображень

Критерій	Використання View та Layout	Використання Jetpack Compose
Оголошення UI	Потрібно створити XML-файли для визначення інтерфейсу	Інтерфейс описується прямо у коді Kotlin, використовуючи функції
Код та структура	Суворі структура з поділом між логікою та UI (XML та Activity/Fragment)	UI та логіка в одному файлі, що робить код компактним та читабельним
Реактивність	Для оновлення UI часто використовуються ViewModel та LiveData або StateFlow	Compose має вбудовану реактивну природу, що полегшує оновлення UI
Гнучкість у змінах UI	Для зміни UI потрібне оновлення XML та перезапуск Activity/Fragment	UI автоматично оновлюється через декларативний підхід у реальному часі.
Простота навчання	Для новачків простіше починати з XML та стандартних View	Менше шаблонів і документації, оскільки Compose є відносно новим, але воно простіше для людей, вже знайомих з Kotlin

Рисунок В.7 – Слайд 7

Використані Технології

Мова програмування: Kotlin
 Фреймворк: Jetpack Compose, Gemini
 ML Інструменти: Google ML Kit
 Серовище розробки: Android Studio

Gemini — це інноваційний фреймворк штучного інтелекту, розроблений компанією Google, що інтегрує різноманітні моделі машинного навчання для генерації контенту та обробки даних.

Google ML Kit — це набір інструментів (SDK) від Google, призначений для розробників мобільних застосунків, що дозволяє використовувати можливості машинного навчання (ML) на мобільних пристроях без необхідності постійного підключення до Інтернету.

Android Studio-провідна платформа, рекомендована Google для розробки застосунків для Android, що пропонує розробникам широкий спектр функцій, від підтримки новітніх мов програмування, таких як Kotlin і Java, до вбудованих емуляторів для тестування застосунків на різних пристроях.

Рисунок В.8 – Слайд 8

Структура даних

Сутність	Поля	Опис
Фото(PhotoEntity)	id,photo,date	Містить дані про зроблене фото та час його створення
Повідомлення(Message)	text,isUser	Містить дані про текст повідомлення та хто його відправив користувач або штучний інтелект
Повідомлення(MessageEntity)	id,text,isUser	Логіка як у Message,але використовується для збереження до бази даних

Рисунок В.9 – Слайд 9

Основні компоненти

Компонент	Функції	Опис
CameraRepository	takePhoto,detectNail	Відповідає за взаємодію з камерою, створення файлу фото та передачу даних про зображення
DatabaseRepository	inserMessage.getAllMessages	Відповідає за взаємодію з базою даних чату,відправку та отримання повідомлень
GeminyRepository	getAnswerWithPhoto,getAnswerWithoutPhoto	Відповідає за взаємодію зі штучним інтелектом,відправку запиту з фото або без, та отримання відповіді
NetworkCheckerRepository	isConnectionAvailable	Відповідає за перевірку чи є у користувача інтернет приєднання або ні
PhotoRepository	savePhotoToRoom,getAllPhotos	Відповідає за взаємодію з базою даних, збереження фото у базу даних та відправку користувачу
StorageRepository	setLightBack,isLightBack	Відповідає за взаємодію з UI,змінна кольорів які використовуються у застосунку

Рисунок В.10 – Слайд 10

Технічна реалізація обробки зображень та взаємодії з штучним інтелектом

Для випадків, коли потрібно обробити зображення, використовується метод `getAnswerWithPhoto`.

Тут відбуваються кілька ключових етапів:

- модель `GenerativeModel` запускається з використанням більш підходящої версії для обробки фото — `gemini-1.5-flash`;
- зображення проходить стиснення за допомогою функції `compressBitmap`, що дозволяє зменшити розмір картинки та пришвидшити обробку;
- за допомогою `content` передається як зображення, так і текст (запитання та додатковий текст), після чого модель генерує відповідь на основі комбінації зображення та тексту.

Рисунок В.11 – Слайд 11

Технічна реалізація обробки зображень та взаємодії з штучним інтелектом

Важливою частиною процесу є функція `takePhoto`, яка використовує можливості камери для захоплення фото. Цей метод створює файл із зображенням, обробляє його та зберігає в галереї пристрою. При успішному знятті фотографії повертається `Bitmap` та шлях до файлу. Якщо камера не може захопити зображення, відбувається обробка помилки.

Для обробки зображень та розпізнавання об'єктів на фото використовується метод `detectNail`. У ньому застосовується бібліотека `ImageLabeling` для аналізу зображення та пошуку об'єктів, що відповідають мітці "nail" (ніготь). Якщо розпізнано об'єкт з достатньою впевненістю (більше 80%), функція повертає результат, що підтверджує наявність нігтя на зображенні.

Рисунок В.12 – Слайд 12

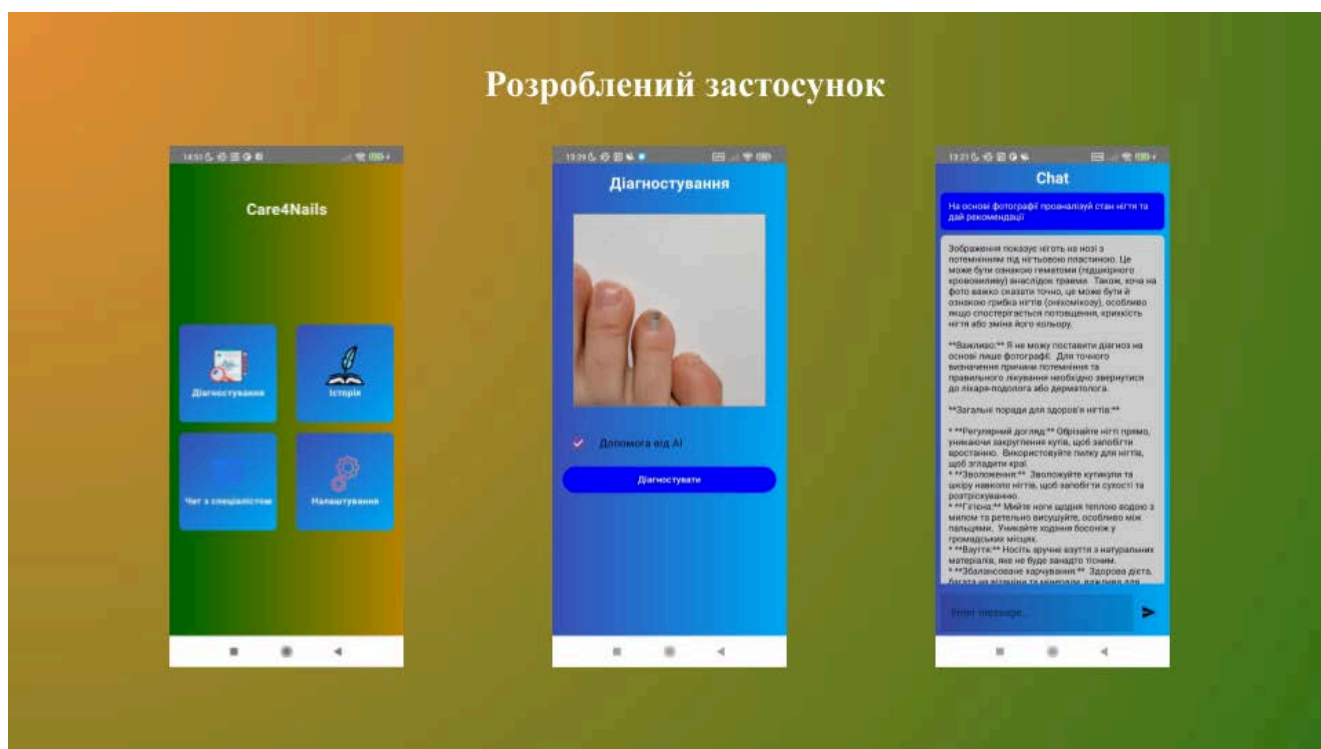


Рисунок В.13 – Слайд 13

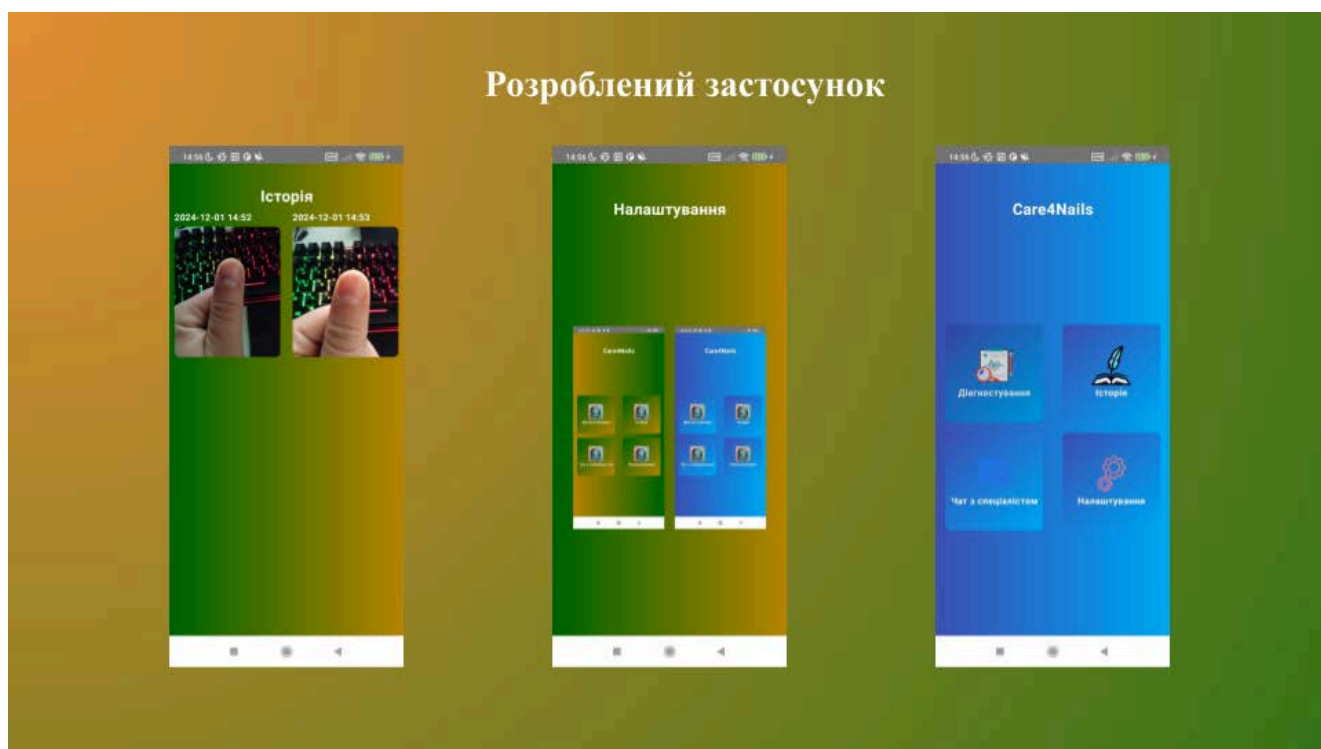


Рисунок В.14 – Слайд 14

Висновки

У результаті виконаної дипломної роботи було розроблено застосунок для аналізу стану нігтів, що використовує сучасні технології обробки зображень та штучного інтелекту для точного визначення різних станів нігтів на фотографіях. Застосунок успішно справляється з розпізнаванням нігтів навіть при низькій якості зображень, поганому освітленні та різних кутах зйомки, що є важливим фактором для користувачів, які роблять фото в умовах, що не завжди є ідеальними. Тестування точності аналізу показало, що система здатна ефективно виявляти основні пошкодження та захворювання нігтів, класифікуючи їх відповідно до наданих зображень. Особливу увагу було приділено роботі моделі Gemini, яка продемонструвала високу точність у визначенні стану нігтів при різних умовах. Окрім того, важливою перевагою є те, що Gemini є безкоштовною моделлю, що робить її доступною для широкого кола користувачів, не знижуючи при цьому точність результатів. Завдяки поєднанню передових технологій і доступних інструментів, розроблений застосунок може стати корисним інструментом для користувачів, які хочуть швидко і точно оцінити стан своїх нігтів. В цілому, робота підтвердила ефективність використання технологій обробки зображень та штучного інтелекту для аналізу зображень у медицині та догляді за здоров'ям. Враховуючи позитивні результати тестування, застосунок має перспективи для подальшого розвитку та вдосконалення, включаючи підтримку нових типів зображень та розширення функціональних можливостей.

Рисунок В.15 – Слайд 15