НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«__» _____ 2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою

«Інженерія програмного забезпечення комп'ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань»

Виконав:
студент 4 курсу, групи ІП-04
Кокошко Ярослав Олегович _____

Керівник:
ас. Ковальчук Олександр Миронович _____

Консультант (нормоконтроль):
доц. Волокита Артем Миколайович _____

Рецензент:
доц. кафедри ІСТ Шимкович Володимир Миколайович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2024 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)
Освітньо-професійна програма
“Інженерія програмного забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
“__” _____2024 р.

_

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента
Кокошко Ярославу Олеговичу

1. Тема проєкту: «Мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань»,
керівник проєкту ас. Ковальчук Олександр Миронович,
затверджені наказом по університету від «27» травня 2024 року № 2112-с.

2. Термін здачі студентом закінченого проєкту: 06 червня 2024 р..

3. Вихідні дані до проєкту: багатопарадигмова компільована мова програмування Swift, інтегроване середовище розробки Xcode, технологія UIKit.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються): огляд існуючих рішень, огляд технологій в розробці системи, опис програмної реалізації мобільного застосунку, огляд розробленого застосунку.

5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень): алгоритм взаємодії користувача з застосунком (принципова схема), діаграма класів (функціональна схема), архітектурна структура застосунку (структурна схема), скриншоти розробленого застосунку.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Волокита А.М.		

7. Дата видачі завдання 05 січня 2024 р..

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	Затвердження теми проєкту	19.02.2024-10.03.2024	
2.	Вивчення та аналіз завдання	11.03.2024-17.03.2024	
3.	Розробка архітектури та загальної структури системи	18.03.2024-24.03.2024	
4.	Розробка структур окремих підсистем	25.03.2024-31.03.2024	
5.	Програмна реалізація системи	01.04.2024-28.04.2024	
6.	Оформлення пояснювальної записки	29.04.2024-02.06.2024	
7.	Захист програмного продукту	05.06.2024	
8.	Передзахист	06.06.2024	
9.	Захист	20.06.2024	

Студент-дипломник Ярослав КОКОШКО

Керівник проєкту Олександр КОВАЛЬЧУК

Анотація

В бакалаврському дипломному проєкті реалізовано мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань. Застосунок взаємодіє з низкою нейронних мереж, розташованих на сервері для аналізу фотографій, який повертає користувачу ймовірність наявності того чи іншого захворювання. За результатами дослідження, система, у випадку високої ймовірності наявності захворювання, може рекомендувати звернутись до лікаря для більш детального аналізу.

Мобільний застосунок розроблений для операційної системи iOS, що дозволяє швидко, використовуючи лише телефон та інтернет, отримати ймовірність наявності захворювання без звернення до лікаря.

Застосунок розроблений з використанням мови програмування Swift, та середовища розробки Xcode.

Ключові слова: мобільний застосунок, діагностика захворювань, нейронні мережі, аналіз фотографій, iOS, Swift, Xcode, штучний інтелект, медичний застосунок.

Annotation

In this project for a bachelor’s degree, a mobile application to support decision-making in the diagnosis of diseases was implemented. The application interacts with a number of neural networks located on a server to analyze photos, which returns the user the probability of having a particular disease. Based on the results of the analysis, the system can recommend consulting a doctor for a more detailed analysis in case of a high probability of the disease.

The mobile application is designed for the iOS operating system, which allows you to quickly get the probability of having a disease without going to the doctor using only your phone and the Internet.

The application was developed using the Swift programming language and the Xcode development environment.

Key words: mobile application, disease diagnostics, neural networks, photo analysis, iOS, Swift, Xcode, artificial intelligence, medical application.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	ІАЛЦ.467100.002 ТЗ	Мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань	3		
			Технічне завдання			
	А4	ІАЛЦ.467100.003 ПЗ	Мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань	62		
			Пояснювальна записка			
	А4	ІАЛЦ.467100.004 Д1	Мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань	1		
			Алгоритм взаємодії користувача з застосунком (принципова схема)			
	А4	ІАЛЦ.467100.005 Д2	Мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань	1		
			Діаграма класів (функціональна схема)			
	А4	ІАЛЦ.467100.006 Д3	Мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань	1		
			Архітектурна структура застосунку (структурна схема)			
	А4	ІАЛЦ.467100.007 Д4	Мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань	17		
			Текст програмного коду			

					ІАЛЦ.467100.001 ОА			
Зм	Лист	№ докум.	Підп.	Дата				
Розробив		Кокошко Я. О.			Мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань Опис альбому			
Перевірив		Ковальчук О. М.						
Реценз.		Шимкович В. М.				КПІ ім. Ігоря Сікорського ФІОТ, ІП-04		
Н. контр.		Волокита А. М.						

Технічне завдання
до дипломного проєкту
на тему «Мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань»

Київ – 2024 р.

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 2
2. ПРИЧИНИ ДЛЯ РОЗРОБКИ 2
3. ЦІЛЬ ТА ПРИЗНАЧЕННЯ РОЗРОБКИ 2
4. ДЖЕРЕЛА РОЗРОБКИ 2
5. ТЕХНІЧНІ ВИМОГИ 3
    - 5.1 ВИМОГИ ДО ПРОДУКТУ, ЩО РОЗРОБЛЯЄТЬСЯ 3
    - 5.2 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 3
    - 5.3 ВИМОГИ ДО АПАРАТНОЇ ЧАСТИНИ 3
6. ЕТАПИ РОЗРОБКИ 4

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Це технічне завдання поширюється на розробку мобільного застосунку для підтримки прийняття рішень при діагностуванні захворювань. Область використання: сприяння користувачам при діагностуванні хвороби шляхом обробки фотографії потенційного захворювання.

2. ПРИЧИНИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського проєкту по освітньо-професійній програмі “Інженерія програмного забезпечення комп’ютерних систем” спеціальності 121 “Інженерія програмного забезпечення”, затверджене кафедрою обчислювальної техніки Національного технічного Університету України “Київський Політехнічний інститут імені Ігоря Сікорського”.

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є розробка мобільного застосунку для підтримки прийняття рішень при діагностуванні захворювань, що допоможе лікарям більш точно призначати діагноз.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література та документація використаних технологій, публікації в інтернеті.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту:

- Простота та доступність у використанні – користувач повинен отримати доступ до обробки фотографії за наявності лише телефону та інтернет-зв’язку.

- Можливість користувачу обрати необхідний вид діагностики.

- Можливість за наслідками надсилання фотографії та її обробки сервером переглянути результати дослідження, зокрема, ймовірність наявності хвороби.

- Можливість переглянути попередні дослідження.

- Можливість ідентифікації користувача шляхом його аутентифікації.

5.2. Вимоги до програмного забезпечення

- Операційна система мобільного телефону: не нижче iOS 16.

5.3. Вимоги до апаратної частини

- Мобільний телефон «iPhone 8» та новіше

- Стабільний зв’язок з Інтернетом

- Вільне місце у пам’яті телефону не менше 5 Мб.

6. ЕТАПИ РОЗРОБКИ

	Дата
6.1 Затвердження теми та вивчення літератури	19.02.2024-10.03.2024
6.2 Складання й узгодження технічного завдання	11.03.2024-17.03.2024
6.3 Розробка архітектури та структури застосунку	11.03.2024-17.03.2024
6.4 Програмна реалізація застосунку	01.04.2024-28.04.2024
6.5 Аналіз застосунку на наявність помилок та їх виправлення	28.04.2024
6.6 Оформлення документації до проєкту	29.04.2024-02.06.2024

Пояснювальна записка
до дипломного проєкту
на тему «Мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань»

Київ – 2024 р.

ЗМІСТ

РОЗДІЛ 1 5

ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 5

1.1. Визначення понять 5

1.2. Огляд існуючих рішень 6

1.2.1. Skinive 6

1.2.2. MIISKIN 7

1.2.3. Google Lens 8

1.2.4. Medgic 9

1.3. Порівняльний аналіз існуючих рішень 10

ВИСНОВОК ДО РОЗДІЛУ 1 11

РОЗДІЛ 2 12

ОГЛЯД ТЕХНОЛОГІЙ В РОЗРОБЦІ СИСТЕМИ 12

2.1. Архітектура 12

2.2 Інтегроване середовище розробки 12

2.3. Мови програмування 14

2.3.1. Swift 15

2.3.2. Інші мови програмування, які використовуються для розробки застосунків під iOS. 16

2.3.3. Підсумок щодо вибору мов програмування 18

2.4. Фреймворки для розробки користувацького інтерфейсу 18

2.4.1. UIKit 19

2.4.2. SwiftUI 21

2.4.3. Підсумок щодо обрання фреймворку (SwiftUI, UIKit) 22

2.5. Інструменти для керування бібліотеками 22

2.5.1. Swift Package Manager 22

2.5.2. Інші інструменти для керування бібліотеками, які є поширеними, проте не використовувались в даному застосунку, та аргументації їх необрання 23
2.5.3. Підсумок стосовно обрання інструменту для управління бібліотеками 24
2.6. Бібліотеки 24
2.6.1. Alamofire 24
2.6.2. SwiftyJSON 25
2.6.3. Підсумок щодо вибору бібліотек 25
ВИСНОВОК ДО РОЗДІЛУ 2 27
РОЗДІЛ 3 29
ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МОБІЛЬНОГО ЗАСТОСУНКУ 29
3.1. Функціональне наповнення мобільного застосунку 29
3.2. Карта екранів мобільного застосунку 30
3.3 Зберігання текстових даних та локалізація 35
3.3. Архітектурний підхід 37
3.4. Аутентифікація 41
3.5. Взаємодія застосунку з сервером 42
3.6. Розгортання та конфігурування мобільного застосунку 46
ВИСНОВОК ДО РОЗДІЛУ 3 48
РОЗДІЛ 4 50
ОГЛЯД РОЗРОБЛЕНОГО ЗАСТОСУНКУ 50
4.1 Аутентифікація користувача 50
4.2 Вибір моделі штучного інтелекту та надсилання заявки 52
4.3 Перегляд вже надісланих заявок 56
ВИСНОВОК ДО РОЗДІЛУ 4 58
ВИСНОВКИ 60

СПИСОК СКОРОЧЕНЬ

1. ОС – операційна система
2. UI – User Interface
3. JSON – JavaScript Object Notation
4. App Store – Application Store
5. API – Application Programming Interface
6. HTTP – Hypertext Transfer Protocol
7. IDE – Integrated Development Environment
8. MedTech – Medical Technology
9. SwiftPM – Swift Package Manager
10. iOS – iPhone Operating System

ВСТУП

У сучасному світі, де технологічний процес є надзвичайно швидким, роль мобільних застосунків у сфері медицини зростає з кожним днем. Високі технології стають важливим інструментом для покращення доступності та якості медичних послуг, а також для сприяння ранній діагностиці та лікування захворювань.

Розроблений мобільний застосунок забезпечує підтримку прийняття рішень при діагностуванні захворювань. Основна ідея застосунку полягає в тому, щоб надати користувачам можливість швидко та ефективно оцінити свій стан здоров’я, зокрема, ймовірність наявності чи відсутності певного захворювання, за допомогою взаємодії з сучасними технологіями обробки зображень та аналізу даних.

Інноваційні методи обробки зображень та алгоритми штучного інтелекту, з якими взаємодіє застосунок, дозволяють відтворювати функціонал деяких аспектів клінічного обстеження. За допомогою камери мобільного пристрою, користувач може зробити фотографії симптомів або патологічних проявів та отримати оцінку ймовірності певного захворювання. Такий підхід може бути корисним як для лікарів у практиці, що шукають допомогу у прийнятті рішень, так і для звичайних людей, які можуть використовувати застосунок для попередньої самодіагностики.

Метою цього дипломного проєкту є створення простого та доступного інструменту, який допоможе в підтримці прийняття рішень при діагностуванні захворювань, зменшуючи час та зусилля, необхідні для отримання медичної допомоги.

Результати цього дослідження можуть мати значний позитивний вплив на якість медичного обслуговування та здоров’я населення. Враховуючи швидкий темп розвитку мобільних технологій, цей проєкт може стати важливим внеском у розвиток медичних застосунків та сприяти покращенню загального стану здоров’я суспільства.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1. Визначення понять

В сучасному світі важко уявити повсякденне життя людини без використання мобільного телефону. Телефонами користується переважна більшість населення як світу, так і нашої країни. Найпоширенішою формою взаємодії користувача з сервісами є використання мобільних застосунків. Мобільний застосунок – це програмне забезпечення, яке розроблене для використання на мобільних пристроях. Вони містять низку переваг у використанні в порівнянні з вебсайтами. Зокрема, мобільний застосунок працює стабільніше і швидше, оскільки переважна більшість інформації зберігається безпосередньо в пам’яті пристрою. Також, їх перевагою є швидкий доступ до галереї та камери пристрою, що дуже важливо в медичних додатках. Також, мобільні додатки можуть надсилати на пристрій користувача сповіщення, наприклад, про те, що заявка оброблена та отримано результат дослідження.

Медична діагностика з використанням систем штучного інтелекту – це процес використання алгоритмів та нейронних моделей для аналізу медичних даних та прийняття рішень щодо ймовірності наявності захворювання. Цей підхід може включати аналіз медичних зображень, обробки медичних даних, попередження та діагностику захворювань.

Важливою, та завершальною складовою додатку є підтримка прийняття рішення при діагностуванні захворювань, яка полягає у використання алгоритмів та моделей для аналізу медичних даних та надання користувачам рекомендацій у процесі прийняття рішень щодо діагнозу. Важливо зауважити, що система самостійно не ставить діагнозу, а лише сприяє користувачу чи лікарю у визначенні діагнозу, шляхом визначення ймовірності наявності хвороби або іншими формами.

Медичні технології (MedTech) також дуже швидкими темпами стають частиною нашого життя. В період, коли всі життєві сфери максимально

диджиталізуються, медична галузь не є винятком. Сьогодні ця галузь значно цифровізувалась, що дозволяє записатись на прийом до лікаря, переглянути результати аналізів, знайти список доступних лікарів тощо не виходячи з дому. Важливим напрямом MedTech є сервіси, які дозволяють встановити ймовірність захворювання по фотографії. Залежно від сфери дослідження, це може бути фотографія рентгену, МРТ, фотографія шкіри тощо. Прикладами MedTech застосунків можуть бути Luna, EyeXam, Helsi, Lilo, Skinive тощо.

Окрім покращення доступу до медичних послуг та діагностики, MedTech також сприяє розвитку персоналізованої медицини. Використання великих даних (Big Data) та штучного інтелекту дозволяє лікарям надавати більш точні рекомендації, враховуючи індивідуальні особливості кожного пацієнта. Наприклад, аналіз змін шкірних об’єктів може допомогти у визначенні ризиків розвитку певних захворювань та підборі оптимальних методів профілактики. Таким чином, MedTech не лише покращує якість медичних послуг, але й сприяє зниженню витрат на охорону здоров’я, запобігаючи розвитку хвороб на ранніх стадіях.

1.2. **Огляд існуючих рішень**

На ринку мобільних застосунків доступні такі подібні продукти:

1.2.1. **Skinive**

Skinive – це мобільний додаток, розроблений для допомоги користувачам у діагностуванні шкірних захворювань. Він використовує штучний інтелект та машинне навчання, щоб аналізувати фотографії шкірних уражень та надавати користувачам можливі діагнози, інформацію про стан та рекомендації щодо лікування [1].

Перевагами Skinive є:

1) Швидкість: результат може надаватись протягом декількох секунд після надсилання фотографії;

2) Точність: завдяки добре навченим моделям ШІ забезпечується висока точність ймовірності діагнозів;
3) Простота використання: користувачу достатньо пройти реєстрацію та прикріпити фотографію, після чого додаток проаналізує її та надасть результат;

Водночас Skinive містить низку недоліків, зокрема:

1) Оплатність: використання додатка є платним, за винятком першого місяця пробного періоду;
2) Відсутність взаємодії з лікарем: система визначає результати дослідження самостійно, без перевірки лікарем. Це збільшує ризик помилкового діагностування;
3) Відсутність відкритого коду: це не дозволяє користувачу бути цілком впевненим у безпеці своїх даних.

Skinive є хорошим рішенням для оцінки ймовірності захворювання, проте містить ряд недоліків, зокрема, оплатність, відсутність взаємодії з лікарем та відсутність відкритого коду.

1.2.2. **MIISKIN**

Додаток, що дозволяє пацієнтам зі шкірними захворюваннями звертатися до певної організації лікарів. Пацієнти надають необхідну вхідну інформацію для відео- або особистих консультацій, або асинхронних консультацій через додаток. Передові технології візуалізації в додатку для пацієнтів, такі як виявлення розмитості, забезпечують високу якість зображень, необхідну для ефективного клінічного огляду лікарями[2].

Додаток має низку переваг, зокрема:

1) Взаємодія з лікарем: застосунок дозволяє надіслати певному лікарю на оцінку ймовірне захворювання та повністю взаємодіяти з лікарем у випадку лікування;
2) Відстеження змін: застосунок дозволяє періодично фотографувати стан шкіри та відстежувати зміни протягом тривалого часу;

3) Анамнез: лікар може налаштувати анкету з низкою запитань кожному окремому пацієнту для отримання необхідної інформації про нього.

Водночас MIISKIN містить низку недоліків, зокрема:

1) Відсутність моделей машинного навчання: діагностика здійснюється виключно лікарями без використання технологій штучного інтелекту;
2) Відсутність українського інтерфейсу: застосунок розрахований на користувачів зі Сполучених Штатів Америки.

Загалом, MIINSKIN може бути корисним інструментом для пацієнтів зі шкірними захворюваннями, які шукають зручний спосіб зв'язатися з лікарем та відстежувати стан своєї шкіри. Однак, важливо зазначити, що додаток не використовує штучний інтелект для діагностики, а також не має українського інтерфейсу.

1.2.3. **Google Lens**

Google Lens – це мобільний додаток, який використовує штучний інтелект та машинне навчання для ідентифікації об'єктів, включаючи шкірні захворювання. Додаток може аналізувати фотографії шкірних уражень та надавати користувачам можливі діагнози, інформацію про стан та рекомендації щодо лікування.

Перевагами цього додатку є:

1) Безоплатність: Google Lens безплатний для скачування та використання;
2) Швидкість: результат надається протягом декількох секунд після надсилання фотографії;
3) Простота у використанні: користувачу потрібно тільки зробити фотографію і натиснути кнопку «Пошук»;
4) Інтеграція з Google пошуком: додаток може надавати додаткову інформацію про потенційне захворювання шляхом взаємодії з пошуком Google;

Але також, Google Lens містить ряд недоліків, зокрема:

1) Низька точність: точність діагнозів Google Lens може бути нижчою, аніж в спеціалізованих аналогах, оскільки Google Lens шукає не тільки шкірні захворювання, але й будь-які об'єкти матеріального світу;

2) Відсутність взаємодії з лікарем: додаток жодним чином не взаємодіє з лікарем;

Загалом, Google Lens може бути корисним інструментом для первинної оцінки ймовірності шкірного захворювання. Проте, містить низку недоліків, оскільки не є спеціалізованим медичним додатком.

1.2.4. **Medgic**

Medgic є мобільним додатком для iOS, призначеним для детекції шкірних захворювань. Застосунок дозволяє пацієнтам фотографувати уражені ділянки шкіри, описувати свої симптоми та отримати результат ймовірності наявності того чи іншого шкірного захворювання. Застосунок повністю самостійно аналізує фотографію без будь-якої участі сторонніх людей за допомогою штучного інтелекту[13].

Medgic є складною системою алгоритмів штучного інтелекту. Ці алгоритми разом дозволяють моделям мобільного застосунку самостійно навчатись на основі доданих користувачами фотографій. Моделі застосунку розвивались протягом багатьох років досліджень та розробок, обробки великої кількості даних тощо. Над розробкою цієї системи працювало багато лікарів та вчених. [14]

Medgic має низку переваг, таких як:

- Опис симптомів: в застосунку можна виділити основні симптоми, які супроводжують потенційне захворювання, що сприяє більш точній діагностиці;
- Постійне навчання: моделі штучного інтелекту постійно навчають на основі фотографій, які додають користувачі;

Водночас Medgic має низку недоліків, зокрема:

- Висока ймовірність неточності: розробники попереджають, що моделі є експериментальними та навченими не до кінця, що значним чином збільшує ймовірність помилки при дослідженні.
- Низький функціонал в порівняння з аналогами: застосунок не пропонує жодної взаємодії з лікарем та має досить обмежений функціонал

Загалом же, Medgic є цікавим застосунком подібної категорії, особливістю якого є постійне навчання, однак його недоліки є досить значними.

1.3. **Порівняльний аналіз існуючих рішень**

Створимо загальну порівняльну таблицю:

Таблиця 1.1 Порівняння різних існуючих додатків для діагностики шкірних захворювань

Критерій	**Skinive**	**MIISKIN**	**Google Lens**	**Medgic**
Використання ШІ у діагностиці	Так	Ні	Так	Так
Взаємодія лікаря та клієнта	Ні	Так	Ні	Ні
Відстеження змін	Ні	Так	Ні	Ні
Анамнез	Так	Так	Ні	Так
Оплата	Платний	Платний	Безплатний	Безплатний
Мова інтерфейсу	Англійська	Англійська	Багатомовний	Англійська

Порівнюючи різні мобільні застосунки для детекції шкірних захворювань, можна виснувати, що кожен з них має як переваги, так і недоліки. Вибір найкращого застосунку залежить від потреб. Якщо потрібна швидка та точна діагностика, хорошим рішенням може бути Skinive. Якщо в пріоритеті взаємодія з лікарем та відстеження стану шкіри слід обрати MIISKIN. При необхідності безплатного та простого у використанні варіанту для первинної оцінки потенційного захворювання – Google Lens. Застосунок Medgic цікавий своєю сучасністю та навчанням. Знову ж таки, кожен із застосунків містить низку недоліків, і може бути не просто обрати кожен із них.

ВИСНОВОК ДО РОЗДІЛУ 1

При обранні найкращої моделі додатка, який описано в дипломному проєкті, було прийнято рішення про використання найкращих практик кожного із перелічених аналогів. Зокрема, дуже важливими критеріями є взаємодія з моделями штучного інтелекту, можливість клієнту взаємодіяти з лікарем, можливість мінімального анамнезу, а також українська мова застосунку. Окрім порівнюваних характеристик, важливим є дизайн, зокрема приємний та елегантний інтерфейс користувача, шрифти та кольори повинні бути чіткими та зрозумілими; технології, зокрема застосунок повинен бути розроблений з використанням найсучасніших технологій, бути оптимізованим для роботи на різних пристроях iOS, бути безпечним та захищеним; функціональним, тобто відповідати потребам користувачів та мати цінність для них, бути простим та інтуїтивно зрозумілим, навігація повинна бути чіткою та логічною, а також бути швидким та надійним.

В майбутньому розроблюваний в межах поточного дипломного проєкту мобільний застосунок значним чином підлягатиме покращенню та розширенню його функціоналу. Зокрема, шляхом покращення взаємодії з лікарем та введення можливості відстеження змін на шкірі тощо.

Аналіз існуючих рішень, дозволяє виснувати, що, хоч і MedTech в галузі детекції захворювань по фотографії значно розвинувся, проте досі не існує комплексного та універсального рішення для належної взаємодії користувача з лікарем при оцінці можливості наявності захворювання з використанням моделей штучного інтелекту.

РОЗДІЛ 2
ОГЛЯД ТЕХНОЛОГІЙ В РОЗРОБЦІ СИСТЕМИ

2.1. Архітектура

Найпоширенішими архітектурними підходами для розробки мобільних застосунків під iPhone є MVC (Model-View-Controller), MVVM (Model-View-ViewModel), Viper (View-Interactor-Presenter-Entity-Router). Кожен з них має низку переваг і недоліків та при виборі підходу слід виходити з потреб кожного окремого застосунку. Для розробки поточного мобільного застосунку було прийнято рішення про використання MVC (Model-View-Controller).

Не вдаючись до деталей, можна зазначити, що особливостями цього архітектурного підходу є поділ на такі його структурні елементи: Model (зберігає дані, логіку та алгоритми), View (відповідає за інтерфейс користувача і відображення даних на екрані смартфона) та Controller (обробляє взаємодію між Model і View, керує циклом View). Основною перевагою цього архітектурного підходу є те, що він легко розуміється і використовується, широко підтримується у стандартних бібліотеках. Більш детально про архітектурні підходи буде описано в наступних розділах.

В межах цього дипломного проєкту особлива увага була зосереджена на View та Controller, оскільки моделі штучного інтелекту розроблені та розташовуються на сервері. Застосунок взаємодіє з моделями штучного інтелекту шляхом надсилання запитів на сервер та обробки результату. Бібліотеки, використані для мережевої взаємодії та обробки JSON-даних описані в пункті 2.5.

2.2 Інтегроване середовище розробки

Для розробки поточного мобільного застосунку буде використовуватись інтегроване середовище розробки Xcode (рис. 2.1).

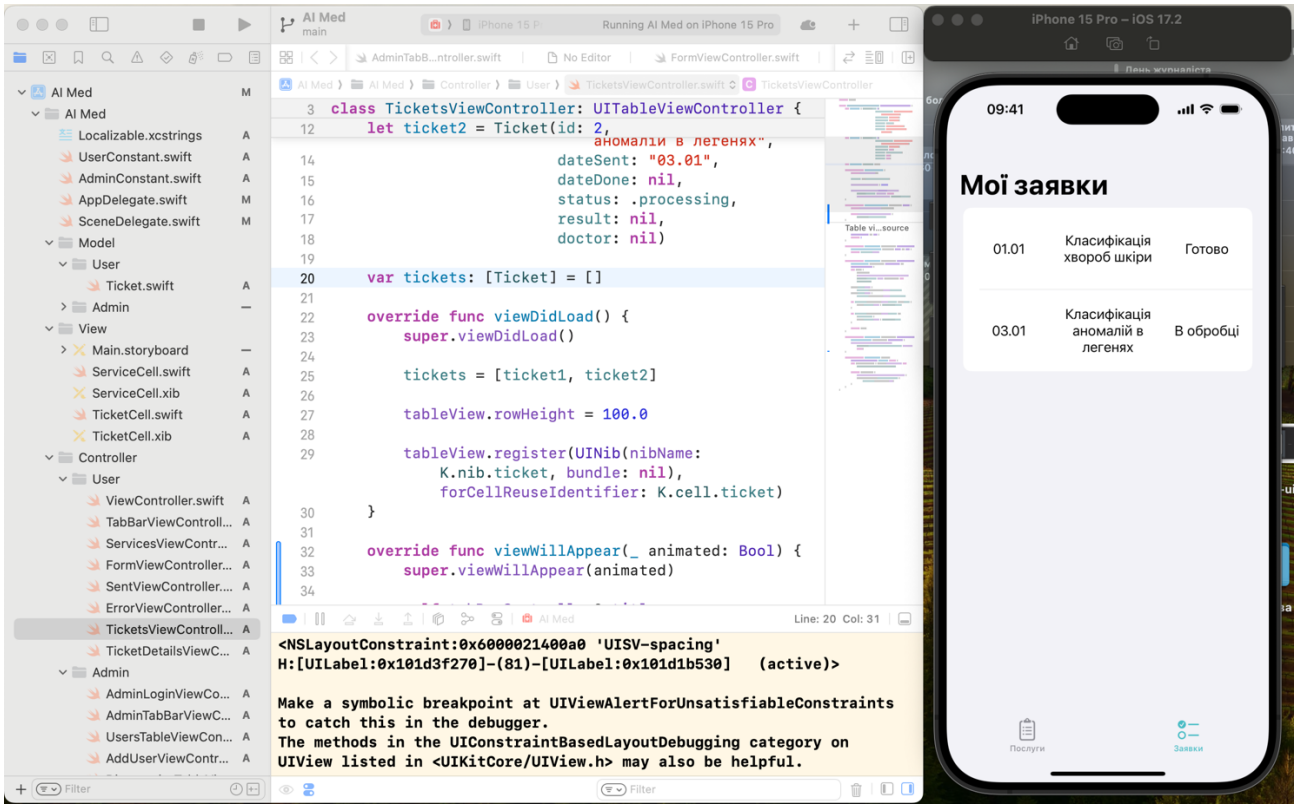


Рисунок 2.1 Інтегроване середовище розробки Xcode

Xcode – це набір інструментів, який використовують розробники для створення додатків, зокрема для iOS. З Xcode можна керувати всім процесом розробки: починаючи створенням самого додатку та завершуючи його оптимізацією, публікацією в App Store та тестуванням.

Xcode є першокласним редактором коду, в який вбудовані низка інструментів для моментального відображення користувацького інтерфейсу в ході розробки [12].

Також, Xcode містить низку інструментів для простого створення прототипів та тестування застосунку. Його функціонал включає використання симулятора для створення прототипів та тестування застосунку в симуляторі. Це значно пришвидшує процес розробки, оскільки нема потреби кожен раз завантажувати оновлений застосунок на фізичний iPhone. Xcode має низку інших вбудованих інструментів, такі як: Swift Package Manager, фреймворк для роботи з машинним навчанням Core ML, інтерактивне середовище для тестування коду Playground,

AutoLayout для створення адаптивних інтерфейсів, що підлаштовані під розміри різних мобільних пристроїв.

Xcode як редактор коду є дуже зручним для розробника з огляду на можливість рефакторингу коду та вбудовані інструменти для поліпшення якості коди. Більше того, він інтегрований з системою контролю версій та гнучкий до змін інтерфейсу, залежно від уподобань розробника.

Акцентуючи увагу на перевагах Xcode можна виділити такі основні блоки:

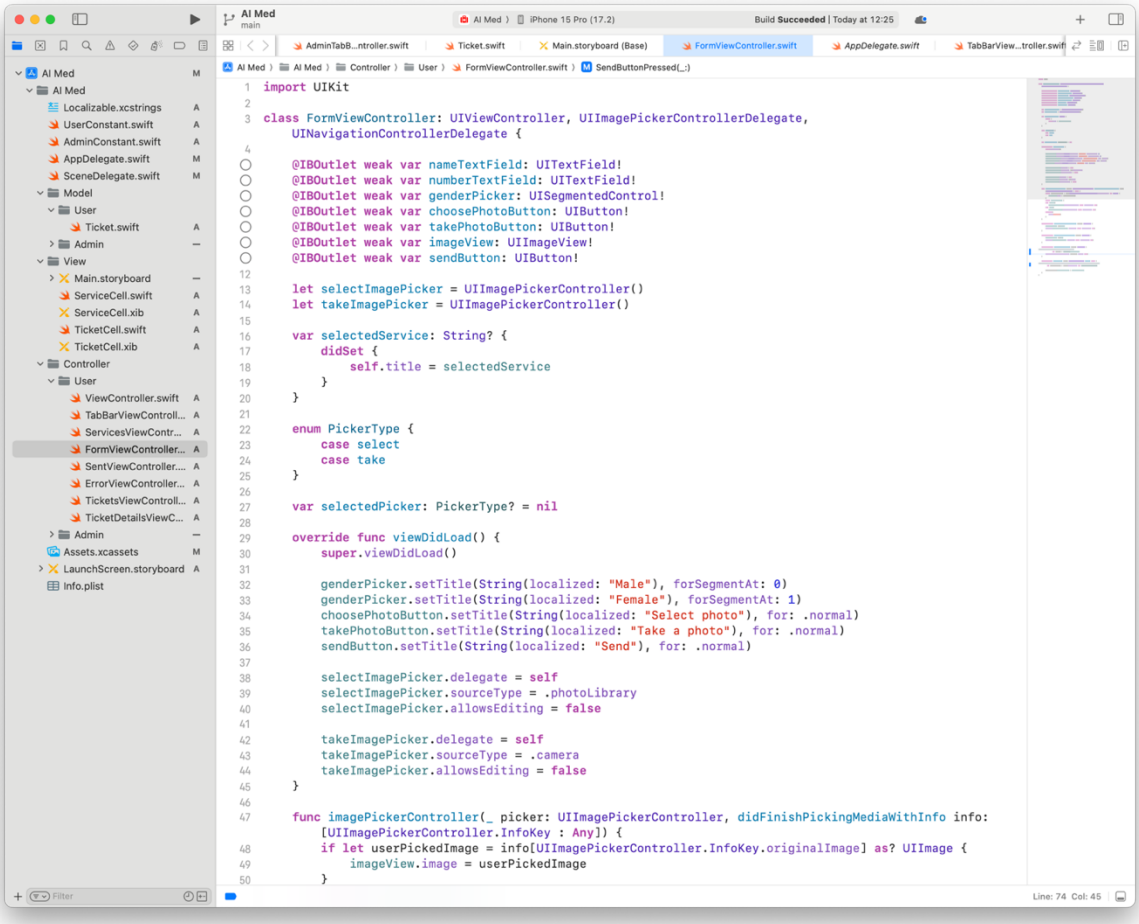
1) Інтеграція з екосистемою Apple: Xcode є офіційним продуктом Apple, що гарантує повну сумісність та оптимізації для всіх пристроїв і операційних систем Apple;
2) Широкий набір функцій: Xcode має всі необхідні інструменти для повного циклу розробки мобільного застосунку: від написання коду до його публікації в мобільному маркеті;
3) Підтримка сучасних технологій: Xcode досить часто оновлюється відповідно до нових сучасних технологій, що з’являються.

2.3. **Мови програмування**

Надалі буде описано мови програмування та технології, що використовувались або могли бути використані під час розробки проєкту, їх переваги та недоліки та обґрунтування обрання тих чи інших мов у порівнянні з аналогічними.

Вибір мови програмування є одним з ключових етапів у розробці поточного мобільного застосунку, оскільки саме від мови залежить ефективність, продуктивність та масштабованість проєкту. Правильний вибір мови програмування може значним чином зменшити час на розробку застосунку, полегшити підтримку коду, та забезпечити можливість інтеграції з іншими системами. Важливим є те, що поточний застосунок може використовуватись в майбутньому іншими командами розробників, а тому вибір сучасної та актуальної мови з можливістю підтримки коду в майбутньому є надважливим питанням.

2.3.1. **Swift**




```swift
import UIKit

class FormViewController: UIViewController, UIImagePickerControllerDelegate,
    UINavigationControllerDelegate {

    @IBOutlet weak var nameTextField: UITextField!
    @IBOutlet weak var numberTextField: UITextField!
    @IBOutlet weak var genderPicker: UISegmentedControl!
    @IBOutlet weak var choosePhotoButton: UIButton!
    @IBOutlet weak var takePhotoButton: UIButton!
    @IBOutlet weak var imageView: UIImageView!
    @IBOutlet weak var sendButton: UIButton!

    let selectImagePicker = UIImagePickerController()
    let takeImagePicker = UIImagePickerController()

    var selectedService: String? {
        didSet {
            self.title = selectedService
        }
    }

    enum PickerType {
        case select
        case take
    }

    var selectedPicker: PickerType? = nil

    override func viewDidLoad() {
        super.viewDidLoad()

        genderPicker.setTitle(String(localized: "Male"), forSegmentAt: 0)
        genderPicker.setTitle(String(localized: "Female"), forSegmentAt: 1)
        choosePhotoButton.setTitle(String(localized: "Select photo"), for: .normal)
        takePhotoButton.setTitle(String(localized: "Take a photo"), for: .normal)
        sendButton.setTitle(String(localized: "Send"), for: .normal)

        selectImagePicker.delegate = self
        selectImagePicker.sourceType = .photoLibrary
        selectImagePicker.allowsEditing = false

        takeImagePicker.delegate = self
        takeImagePicker.sourceType = .camera
        takeImagePicker.allowsEditing = false
    }

    func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info:
        [UIImagePickerController.InfoKey : Any]) {
        if let userPickedImage = info[UIImagePickerController.InfoKey.originalImage] as? UIImage {
            imageView.image = userPickedImage
        }
```

Рисунок 2.2 Приклад програмного коду мови програмування Swift

Swift є мовою програмування, яка створена спеціально для розробки програмного забезпечення під платформи Apple: наприклад iOS, macOS, watchOS, tvOS тощо. Swift - це потужна та інтуїтивно зрозуміла мова програмування для всіх платформ Apple. Легко почати користуватися Swift, зі стислим, але виразним синтаксисом і сучасними функціями. Swift код безпечний за дизайном і виробляє програмне забезпечення, яке працює блискавично [3]. Ця мова програмування має низку переваг, які роблять її привабливою для розробки мобільних додатків:

1) Ефективність та швидкість: Swift розроблялась з акцентом на швидкодію та ефективне використання ресурсів. Час компіляції коду є досить малий, оскільки все сильно оптимізоване. Це дозволяє покращити продуктивність застосунків;

2) Безпека типів: Swift має вбудовані механізми безпеки типів, які допомагають уникнути більшості типових помилок. Це сприяє надійності та безпечності коду;
3) Сучасний синтаксис: Синтаксис мови програмування Swift є сучасним та експресивним, що робить код більш зрозумілим та легким для написання та його подальшого підтримування;
4) Активна спільнота та підтримка: Swift має активну спільноту розробників, які обговорюють проблеми та шляхи їх вирішення у випадку наявності таких.

Мова програмування Swift також має деякі недоліки, зокрема те, що вона була створена відносно нещодавно, а тому має ряд прогалин, а також обмежену підтримку старших версій операційних систем. Однак, з огляду на швидкий розвиток мови та активність спільноти, ці проблеми постійно вирішуються, а мова розвивається.

2.3.2. Інші мови програмування, які використовуються для розробки застосунків під iOS.

Досить поширеним при iOS розробці є використання інших мов програмування, окрім Swift, зокрема Objective-C, React Native та Flutter

Objective-C - високорівнева об'єктноорієнтована мова програмування загального призначення, розроблена у вигляді набору розширень стандартної C [5].

Особливостями цієї мови програмування є те, що вона є повністю об'єктноорієнтованою, тобто підтримує концепції класів, об’єктів, спадкування, поліморфізму та інкапсуляції. Також, ця мова є динамічною, оскільки дозволяє змінювати структуру класів та об’єктів протягом виконання програми, що надає велику гнучкість у розробці. Objective-C є основною мовою програмування для роботи з Cocoa та Cocoa Touch, а також фреймворками розробки програмного забезпечення для macOS та iOS.

Проте, Objective-C має ряд недоліків, зокрема: складний синтаксис, який робить код менш зрозумілим; менша експресивність у порівнянні з Swift, що,

звичайно, призводить до значного збільшення обсягу коду для досягнення тотожних результатів.

React Native є відкритим фреймворком для розробки мобільних додатків, який базується на Java Script та використовує бібліотеку React для створення користувацького інтерфейсу[20]. Одною із найбільших переваг React Native є можливість використання одного коду для розробки додатків під різні платформи, як iOS, так і Android.

Особливостями React Native є: крос-платформеність, тобто один код, який працює як на iOS, так і на Android. Це сприяє економію часу під час розробки, оскільки немає потреби писати код окремо для обох операційних систем; використання Java Script, для розробки додатків, що робить його доступним для багатьох розробників з мінімальним рівнем навичок; ефективне використання компонентів, оскільки використовується компонентний підхід до розробки, що дозволяє розробникам будувати складні інтерфейси шляхом злиття простих компонентів разом; модульність, оскільки React Native дозволяє розробникам легко додавати сторонні модулі та бібліотеки для розширення функціональності додатків.

Але знову ж таки, React Native має низку недоліків, зокрема: відсутня підтримка деяких функцій iOS та Android, а також менша ефективність в порівняння з нативними додатками.

Flutter - це фреймворк з відкритим вихідним кодом від Google для створення красивих, нативно скомпільованих, багатоплатформних додатків з єдиної кодової бази [6].

Flutter використовує мову програмування Dart та надає розробникам можливість створювати користувацький інтерфейс, який працює як на iOS, так і Android.

Особливостями Dart є: крос-платформеність, оскільки він так само як і React Native має спільний код як для iOS, так і для Android операційних систем; швидкість, оскільки код компілюється в машинний код ARM або Intel, та JavaScript,

що сприяє швидкості роботи на будь-яких пристроях; підтримка візуальних ефектів; підтримка командної роботи; декларативний користувацький інтерфейс.

2.3.3. Підсумок щодо вибору мов програмування

Проте, при розробці поточного медичного застосунку вибір мови програмування зупинився на Swift з огляду на низку її переваг.

Мова програмування Swift має низку переваг у порівняння з зазначеними раніше альтернативами. Вона є офіційною мовою програмування для розробки мобільних додатків під iOS, розроблена компанією Apple. Тож, її використання гарантує найбільшу підтримку та оптимізацію для пристроїв Apple, що, безумовно, призведе до кращої продуктивності та ефективності додатка. Також, Swift досить добре інтегрується з іншими інструментами та технологіями від Apple, такими як Xcode, CocoaTouch, UIKit, Core Data тощо. Більше того, Swift має дуже простий та зрозумілий синтаксис, що сприяє більш зручній подальшій підтримці коду та розширенню функціонала додатка.

Отже, хоч і Flutter, Objective-C та React Native можуть мати свої переваги, використання Swift для розробки мобільних застосунків під iOS, Swift вважається найкращим варіантом завдяки його нативності, продуктивності та інтеграції з екосистемою Apple. Тому, при обранні мови програмування для розробки поточного застосунку, було прийняте рішення використовувати Swift.

2.4. Фреймворки для розробки користувацького інтерфейсу

Окрім згаданих раніше мов програмування, у проєкті використовуються фреймворки, для розширення функціональності застосунку.

2.4.1. UIKit

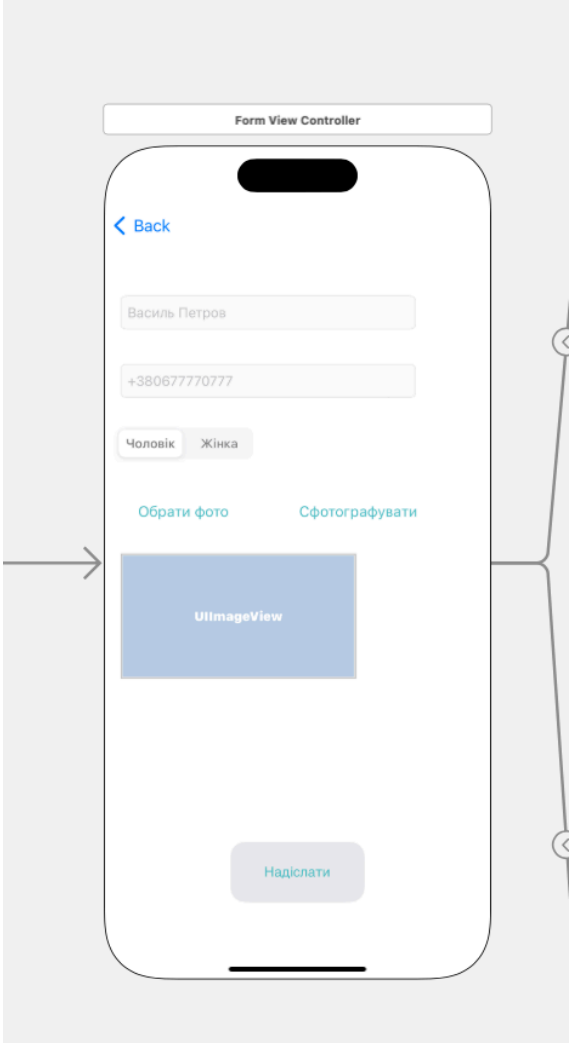


Рисунок 2.3 Зображення розробки користувацького інтерфейсу з використанням UIKit

Однією з основних технологій для розробки мобільних додатків під iOS є UIKit. Ця технологія надає широкий набір інструментів для побудови користувацького інтерфейсу, обробки подій та управління візуальним виглядом додатка, також надає різноманітні функції для створення додатків, включаючи компоненти, які можна використовувати для побудови основної інфраструктури ваших додатків iOS, iPadOS або tvOS. Фреймворк забезпечує архітектуру відображення та рамки для відображення інтерфейсу користувача, інфраструктуру керування подіями для доставки Multi-Touch та інших типів введення до застосунку, а також головний цикл початку для управління взаємодією між додатком та користувачем [4].

Важливими аспектами технології UIKit є такі:

1) UI (інтерфейси користувача): міститься низка класів, які призначені для побудови інтерфейсу користувача. Це включає в себе можливість додавання до інтерфейсу кнопки, тексти, зображення, таблиці тощо. Завдяки цим класам можна створювати складний, але водночас інтуїтивно зрозумілий інтерфейс для додатків;
2) Управління виглядом: можливість управляти відображенням елементів інтерфейсу. Розробники можуть самостійно визначати розміщення елементів, змінювати їх розмір, колір та інші параметри;
3) Обробка подій: UIKit дозволяє обробляти події, наприклад натискання кнопок, ввід тексту, чи додавання фотографії, що є дуже важливим в поточному мобільному застосунку;
4) Анімації: UIKit дозволяє створювати різноманітні анімації та переходи між різними екранами додатка. Це сприяє кращому візуальному ефекту для користувача.

UIKit є ключовою технологією, оскільки саме вона забезпечує розробку та створення всього інтерфейсу застосунку.

2.4.2. SwiftUI

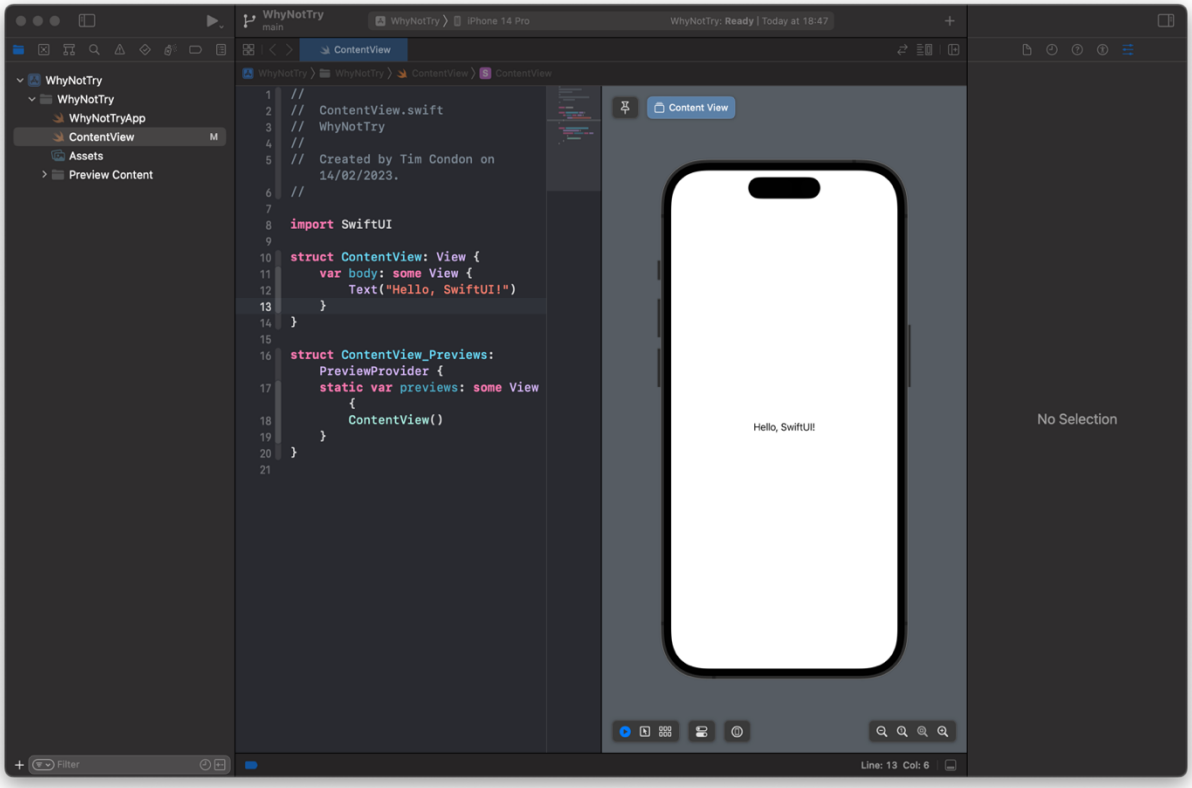


Рисунок 2.4 Зображення процесу розробки застосунку з використанням SwiftUI

SwiftUI надає відображення, елементи керування та структури макетів для декларування користувацького інтерфейсу додатка. Фреймворк надає обробники подій для передачі дотиків, жестів та інших типів введення у додаток. Більше того, він надає засоби для керування потоком даних від моделей додатка до відображень та елементів керування, які бачать користувачі та з якими вони взаємодіють [7].

Особливостями SwiftUI є: компактність та легкість використання, оскільки він надає набір простих та потужних інструментів, які можна комбінувати для створення складних інтерфейсів; універсальність, оскільки він підтримує розробку додатків для різних платформ, таких як iOS, macOS, watchOS, ipadOS тощо; інтеграція з Swift та Xcode, оскільки можна використовувати всі можливості Swift та Xcode для створення додатків у взаємодії з SwiftUI; перегляд в live-режимі, оскільки SwiftUI надає вбудований інструмент перегляду, який дозволяє розробнику бачити, як виглядатиме інтерфейс на різних пристроях та в різних

режимах, навіть під час написання коду. Це сильно прискорює процес розробки та дозволяє одразу виявляти помилки.

2.4.3. Підсумок щодо обрання фреймворку (SwiftUI, UIKit)

Хоча SwiftUI має ряд переваг, при розробці поточного застосунку було прийнято рішення використовувати UIKit.

Таке рішення прийнято з огляду на такі переваги UIKit:

1) Підтримка низки бібліотек та інструментів: UIKit існує значно довше, а відтак має велику кількість документації, ресурсів, бібліотек та інструментів, які підтримуються спільнотою;
2) Підтримка попередніх версій iOS: На відміну від SwiftUI, який підтримує тільки версії iOS новіші за iOS 13, UIKit підтримує старші версії;
3) Стабільність та надійність: UIKit є відомим фреймворком з великою кількістю користувачів. Більшість it-компаній використовують UIKit. Це свідчить про те, що він є більш стабільним та надійним у порівнянні з SwiftUI, який може мати певні недоліки через свою новизну.

Відтак, хоч SwiftUI й має ряд переваг, було прийнято рішення обрати при розробці поточного мобільного застосунку UIKit. Це буде більш доцільним варіантом в нашій ситуації.

2.5. Інструменти для керування бібліотеками

2.5.1. Swift Package Manager

Swift Package Manager (SwiftPM) - це засіб для управління розповсюдженням коду та бібліотеками Swift. Він взаємодіє з системою збірки Swift, щоб спростити процес компіляції, завантаження та з’єднання залежностей. [9]

Було прийнято рішення про її використання з огляду на такі її переваги:

1) Відсутність необхідності додаткового завантаження: SwiftPM вже вбудована в середовищі розробки Xcode;
2) Відкритий код: бібліотека має відкритий програмний код, що сприяє його покращенню з боку спільноти розробників;

3) Створена Apple: ця бібліотека є продуктом безпосередньо компанії Apple, тому є досить оптимізованою та інтегрованою з іншими продуктами компанії;

Отже, Swift Package Manager є найкращим рішення для організації бібліотек в процесі розробки поточного мобільного застосунку.

2.5.2. Інші інструменти для керування бібліотеками, які є поширеними, проте не використовувались в даному застосунку, та аргументації їх необрання

Найбільш поширеними варіантами для управління бібліотеками, окрім Swift Package Manager є Carthage та CocoaPods.

CocoaPods є одним з найвідоміших інструментів для керування залежностями в проєктах, розроблених мовою Swift[18]. Управління бібліотеками в такому випадку здійснюється шляхом створення файлу ‘Podfile’, в якому зазначається перелік бібліотек, які використовуються, версія кожної з них, версія iOS та інші відомості, які у своїй сукупності становлять основу для управління імплементованими бібліотеками. CocoaPods, на відміну від SwiftPM є старшим інструментом, а відтак, має велику спільноту користувачів, багато готових плагінів і розширень. Однак, важливим недоліком CocoaPods є те, що він не імплементований в середовище розробки Xcode.

Carthage є ще одним відомим інструментом для керування залежностями в проєктах на мові Swift[19]. Carthage є більш гнучким у порівнянні з CocoaPods, оскільки дозволяє розробникам контролювати процес інтеграції бібліотек без внесення змін у структуру проєкту Xcode. Однак, це також означає, що процес інтеграції може бути більш складним і вимагати більше ручної роботи. Для простих проєктів або для розробників, які віддають перевагу більшому контролю над процесом, Carthage може бути відмінним вибором. Проте його функціонал є обмеженим порівняно з іншими інструментами, і він може бути менш зручним у використанні для великих проєктів або команд.

2.5.3. Підсумок стосовно обрання інструменту для управління бібліотеками

В ході дослідження відомих інструментів для управління бібліотеками були проаналізовані такі варіанти: SwiftPackageManager, CocoaPods, Carthage.

Врешті-решт, було прийнято рішення про використання SwiftPackageManager з огляду на його переваги, включаючи інтеграцію з Xcode, відкритий код та підтримку з боку Apple. SwiftPM забезпечує оптимальну інтеграцію з іншими інструментами та сервісами Apple, що робить його найбільш підходящим рішенням для ефективного керування бібліотеками в процесі розробки поточного мобільного застосунку.

2.6. **Бібліотеки**

В цьому розділі буде описано бібліотеки, які використовувались при розробці мобільного застосунку. В ньому детально розглянуті дві ключові бібліотеки: Alamofire та SwiftyJSON; а також надано підсумок щодо вибору бібліотек в цілому.

2.6.1. **Alamofire**

Alamofire - це мережева бібліотека з відкритим вихідним кодом, написана на Swift, яка спрощує процес надсилання HTTP-запитів та обробки відповідей. Вона підтримує різні методи HTTP і включає додаткові функції, такі як кодування параметрів і перевірка відповідей. [8]

Ця бібліотека містить низку переваг та причин, чому вона була використана в поточному мобільному застосунку. Перевагами є:

1) Зручність та легкість використання: бібліотека надає простий та інтуїтивно зрозумілий API, який абстрагує розробника від багатьох складнощів щодо здійснення мережевих запитів. Синтаксис бібліотеки є лаконічним, що спрощує інтеграцію для розробників;
2) Абстракція мережі: бібліотека пропонує високорівневу абстракцію для загальних мережевих задач: GET, POST, PUT, обробки JSON та завантаження

файлів. А відтак, у розробників немає необхідності вручну обробляти низькорівневі мережеві задачі;

3) Асинхронність: бібліотека використовує асинхронне програмування, що дозволяє здійснювати мережеві запити без блокування основного потоку.

Отже, Alamofire є досить потужною та широко використовуваною бібліотекою для мережевих запитів у Swift, яка пропоную низку функцій, що спрощують загальні мережеві задачі.

2.6.2. **SwiftyJSON**

SwiftyJSON – це бібліотека для мови Swift, яка допомагає з обробкою даних у форматі JSON. Вона збирає та розбирає дані з JSON-об’єктів, що сприяє більш зручній та безпечній роботі з даними[17]. Було прийнято рішення використовувати цю бібліотеку з огляду на такі її переваги:

1) Спрощення розбору та збору даних: SwiftyJSON дозволяє легко розбирати JSON-дані у Swift-об’єкти, як рядки, числа, масиви, словники, а також збирати у JSON-формат. Завдяки цьому функціоналу немає необхідності парсити JSON-структури вручну.
2) Безпека типів;
3) Простота у використанні;
4) Підтримка Chaining;

А відтак, SwiftyJSON – це потужна та зручна бібліотека для роботи з JSON у мові Swift, яка пропонує багато переваг для розробників.

2.6.3. Підсумок щодо вибору бібліотек

У процесі розробки поточного мобільного застосунку було прийнято рішення використовувати ряд бібліотек. Кожна з обраних бібліотек має своє функціональне спрямування та ряд переваг, що сприяє більш ефективній розробці та підвищенню продуктивності мобільного застосунку.

Бібліотека Alamofire була використана для спрощення процесу взаємодії з мережею, надаючи зручний API та абстрагуючи розробника від низькорівневих деталей мережевої взаємодії, наприклад, HTTP-запити та обробку відповідей.

Бібліотека SwiftyJSON була використана для роботи з даними у форматі JSON, надаючи зручні функції для збору та розбору даних, що значним чином спрощує роботу з форматом JSON у мові програмування Swift.

Всі ці бібліотеки обрані з огляду на їхню простоту використання, потужність та спрощення різних аспектів розробки мобільного застосунку, що дозволяє зосередитись на більш важливих складових елементах розробки застосунку, мінімізуючи зайві технічні складнощі.

ВИСНОВОК ДО РОЗДІЛУ 2

У цьому розділі було розглянуто низку технологій, які будуть використовуватись або могли використовуватись як альтернативні у розробці поточного проєкту, їх позитивні та негативні сторони та причини використання.

Для створення мобільного застосунку серед мов програмування вибір був зосереджений поміж Swift, Objective-C, Dart та JavaScript.

Після проведення аналізу та пошуку кращого рішення було прийнято рішення про використання мови програмування Swift. Дуже важливо, що ця мова є найбільш сучасною та оптимізованою. Ця мова є продуктом компанії Apple та ідеально інтегрована з іншими продуктами компанії. Така інтеграція значно сприяє ефективності розробки, оскільки зв’язаними є мова програмування, інтегроване середовище розробки, можливість тестування програмного коду одразу на мобільних пристроях iPhone тощо. Окрім, інтегрованості мова програмування Swift є дуже оптимізованою та простою, але не менш ефективною та функціональною у порівнянні з іншими мовами програмування, які можуть використовуватись для розробки крос-платформених застосунків.

Для розробки інтерфейсу користувача (UI) вибір був зосереджений між фреймворками SwiftUI та UIKit. Це єдині способи розробки інтерфейсу в середовищі Xcode, які не мають альтернатив. Хоча, вибір між ними є досить складним, оскільки обидва фреймворки мають суттєві переваги, але водночас і недоліки. Сьогодні тенденції, вочевидь, спрямовані до збільшення використання саме SwiftUI, оскільки цей продукт є новішим, та є більш простим та інтуїтивнішим у порівняння з UIKit. Але не дивлячись на них, було прийнято рішення про використання саме UIKit. Це зумовлено низкою переваг, про які зазначалось раніше. Зокрема, важливими перевагами є те, що UIKit існує довше, а тому має значно більше документацію та досвід використання спільнотою розробників. Це сприяє надійності розробленого застосунку. Також, перевагою є підтримка більш старших версій iOS та більшої кількості бібліотек.

Звичайно, в ході розробки мобільного застосунку використовувалась низка бібліотек, серед яких особливу увагу було приділено Alamofire та SwiftyJSON, які сприяють спрощенню розробки та підвищенню продуктивності.

РОЗДІЛ 3
ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МОБІЛЬНОГО ЗАСТОСУНКУ

3.1. Функціональне наповнення мобільного застосунку

В ході розробки поточного мобільного застосунку було реалізовано низку функцій, які може здійснювати користувач задля досягнення необхідних результатів, зокрема, отримання ймовірності наявності того чи іншого захворювання.

Функціонал застосунку розроблений виключно для ролі користувача (пацієнт, лікар).

Функціонал застосунку складається з таких елементів: перегляд наявних моделей ШІ, вибір потрібної моделі ШІ, створення заявки, завантаження фотографії, перегляд створених заявок, видалення заявок, перегляд інформації про заявку. Зобразимо схематично функціонал застосунку, позначивши обов'язкові елементи (include) та додаткові (extends).

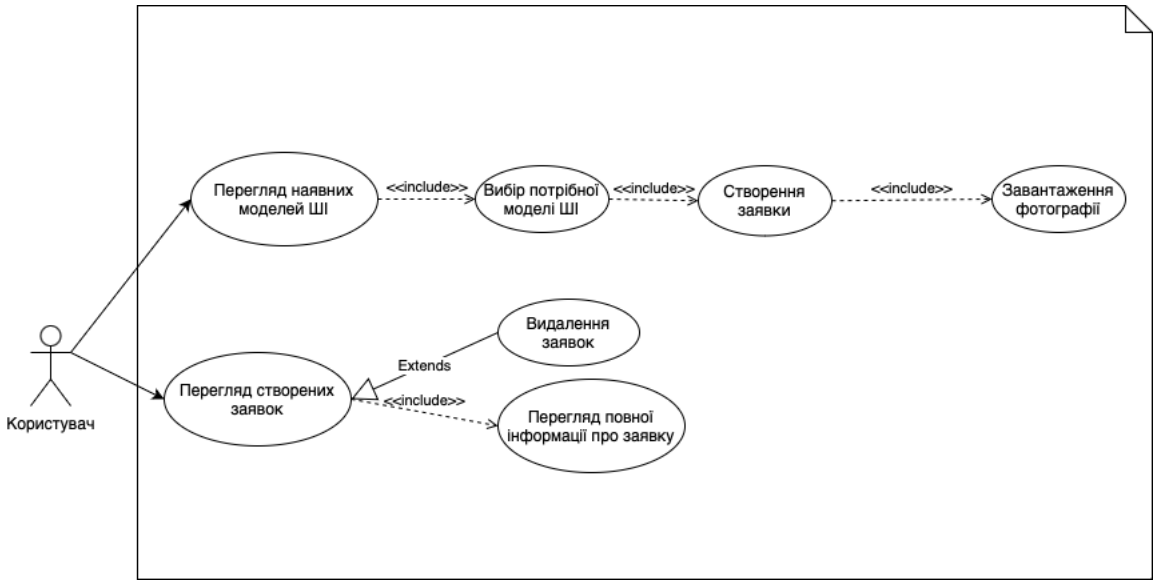


Рисунок 3.1 Функціональне наповнення мобільного застосунку

3.2. **Карта екранів мобільного застосунку**

В ході виконання цього дипломного проєкту було розроблено сім екранів, кожен з яких має своє функціональне призначення: екран авторизації, екран вибору моделі штучного інтелекту, екран надсилання заявки на обробку, екран повідомлення про успішну відправку або помилку, екран з переліком надісланих заявок та екран з деталями кожної окремої заявки.

На рисунку 3.2 зображено розташування екранів та їх взаємозв'язок.

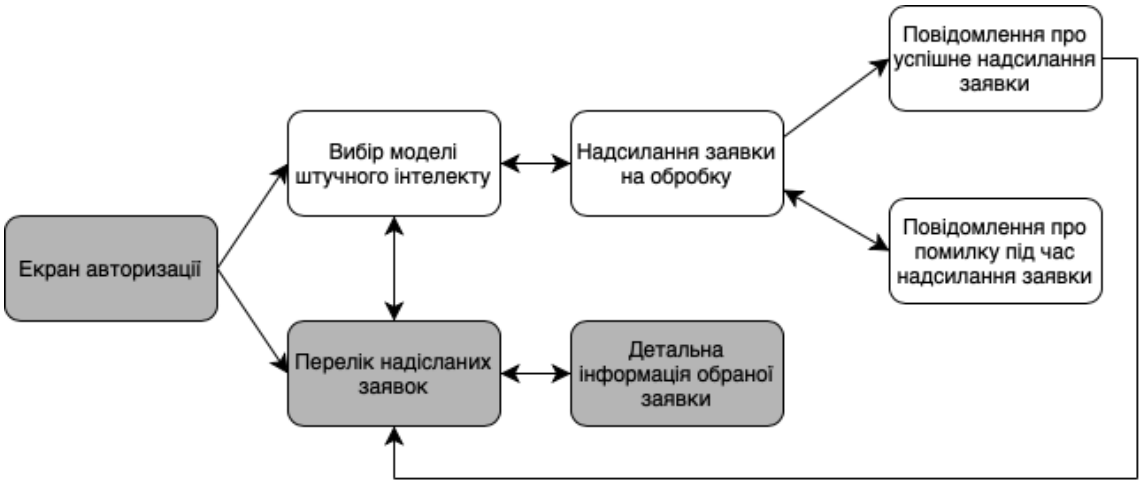


Рисунок 3.2 Діаграма розташування екранів мобільного застосунку

Застосунок розроблено таким чином, щоб забезпечити користувачу зручну та інтуїтивно зрозумілу навігацію між різними екранами. Опишемо функціональне призначення кожного з екранів в таблиці 3.2:

Таблиця 3.2 Функціональне призначення екранів

Назва екрана	**Ідентифікатор в коді**	**Функціональне призначення**
Екран авторизації	ViewController	Перший екран при відкритті застосунку. На цьому екрані користувач вводить свої облікові дані для доступу до системи.
Екран вибору моделі штучного інтелекту	ServicesViewController	Після успішної авторизації користувач потрапляє на екран, де може обрати одну з моделей штучного інтелекту для подальшої роботи.

Продовження таблиці 3.2

Назва екрана	**Ідентифікатор в коді**	**Функціональне призначення**
Екран надсилання заявки на обробку	FormViewController	На цьому екрані користувач заповнює форму заявки, вказуючи необхідні параметри для обробки обраною моделлю ШІ, а також додає фотографію шкірного захворювання. Після заповнення форми користувач може надіслати заявку на обробку.
Екран повідомлення про успішну відправку	SentViewController	Після надсилання заявки користувач, у випадку відсутності помилок, отримує повідомлення про успішну відправку. При цьому йому надається можливість перейти до переліку надісланих заявок для відстеження статусу надісланої заявки.
Екран повідомлення про помилку	ErrorViewController	У випадку ж, якщо при надсиланні заявки виникли помилки (що часто трапляється при взаємодії з сервером через необхідність стабільного інтернет-підключення), то екран повідомлення про помилку повідомить про причину помилки та відобразить рекомендацію надіслати заявку повторно.

Кінець таблиці 3.2

Назва екрана	**Ідентифікатор в коді**	**Функціональне призначення**
Екран з переліком надісланих заявок	TicketsViewController	Цей екран надає користувачу можливість переглядати всі надіслані ним заявки. Кожна з заявок містить коротку інформацію про стан її обробки (готово/в обробці).
Екран з деталями кожної окремої заявки	TicketDetailsViewController	При виборі конкретної заявки з переліку, користувач переходить на екран з детальною інформацією про цю заявку, включаючи всі параметри, статус обробки та результати роботи моделі ШІ.

В проєкті кожен з екранів має свій унікальний ідентифікатор в коді, а також окремий файл, де зберігається код кожного екрану.

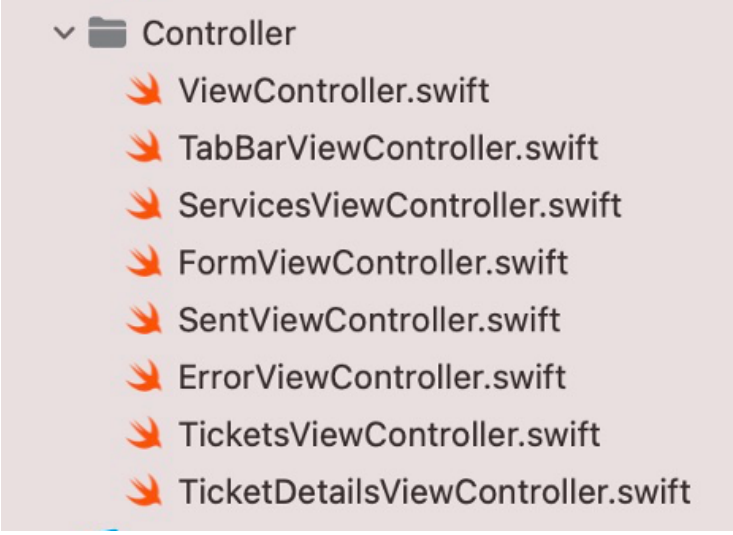


Рисунок 3.3 Перелік файлів для кожного окремого екрану

Кожен файл відповідає за зміст кожного екрану. Такий поділ на окремі файли значно спрощує та робить більш інтуїтивною та зрозумілою роботу з розробки екранів. Можна швидко редагувати необхідний екран.

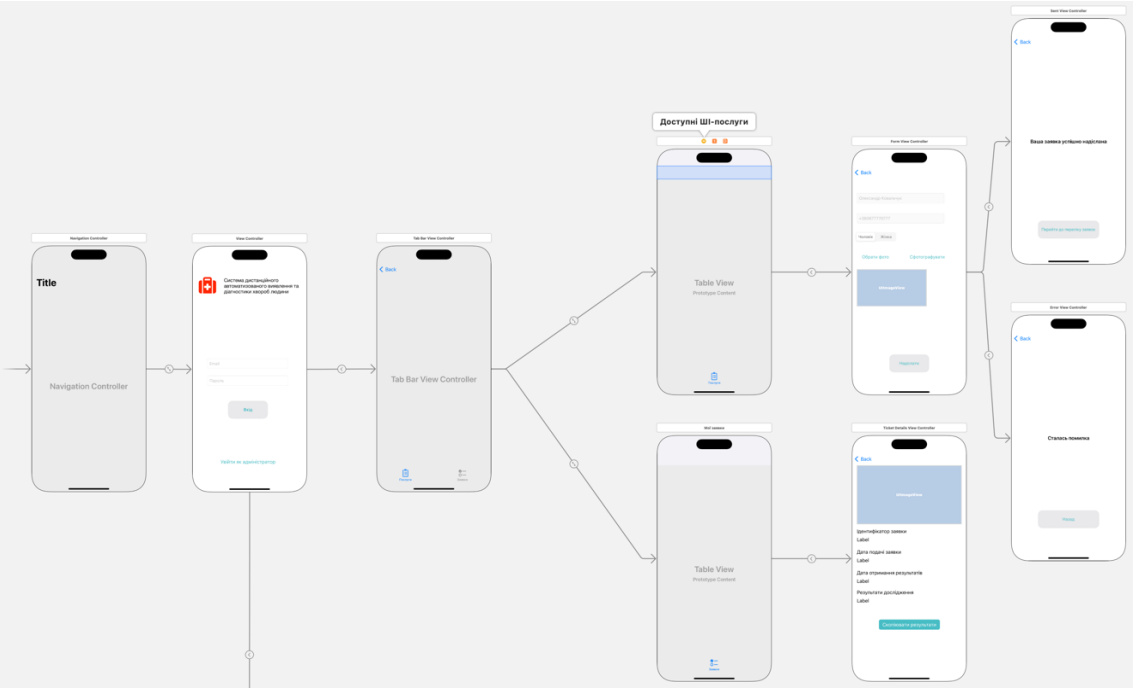


Рис. 3.4 Взаємозв’язок екранів в інтегрованому середовищі розробки Xcode

При цьому ж слід відзначити, що всі екрани є взаємозв’язані між собою, завдяки NavigationController. NavigationController це контролер представлення контейнера, який визначає стекову схему для навігації по ієрархічному вмісту [10]. Цей контролер дозволяє об’єднати всі екрани в один стек, тим самим створивши з них чітку послідовність, та додати кнопку “Back” для повернення на попередній екран. Контролер додається тільки до першого екрану (в поточному застосунку – екран аутентифікації користувача). Оскільки кожен екран є зв'язаний з іншими, то NavigationController поширює свою дію автоматично на всі інші екрани.

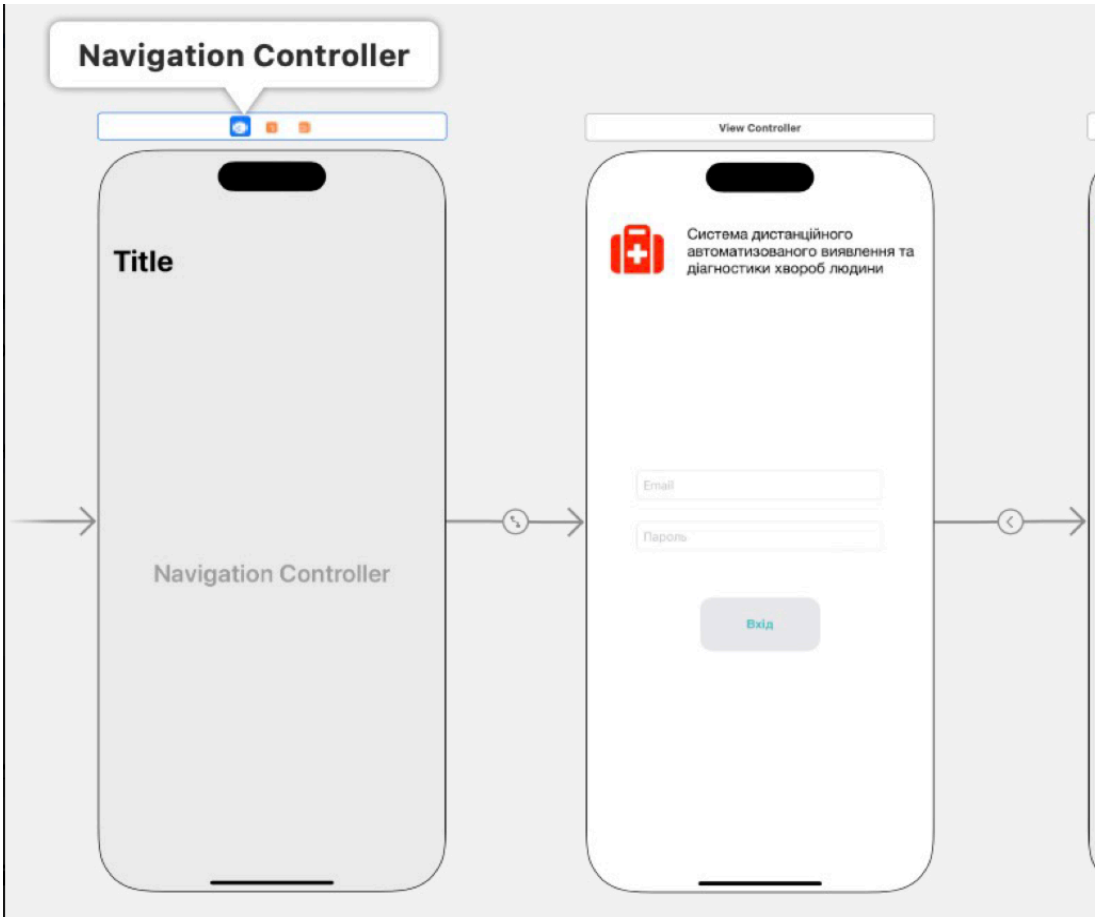


Рисунок 3.5 Імплементація NavigationController для об’єднання всіх екранів в один стек

Також, екрани з переліком доступних моделей штучного інтелекту та переліком надісланих заявок об’єднані технологією tabBar (панелі вкладок). Панелі вкладок використовують елементи панелі для навігації між взаємозаперечними панелями вмісту в одному поданні [11].

Рисунок 3.6 TabBar в мобільному додатку

Завдяки TabBar можна швидко перемикатись між вкладками «доступні ШІ-моделі» та «Мої заявки».

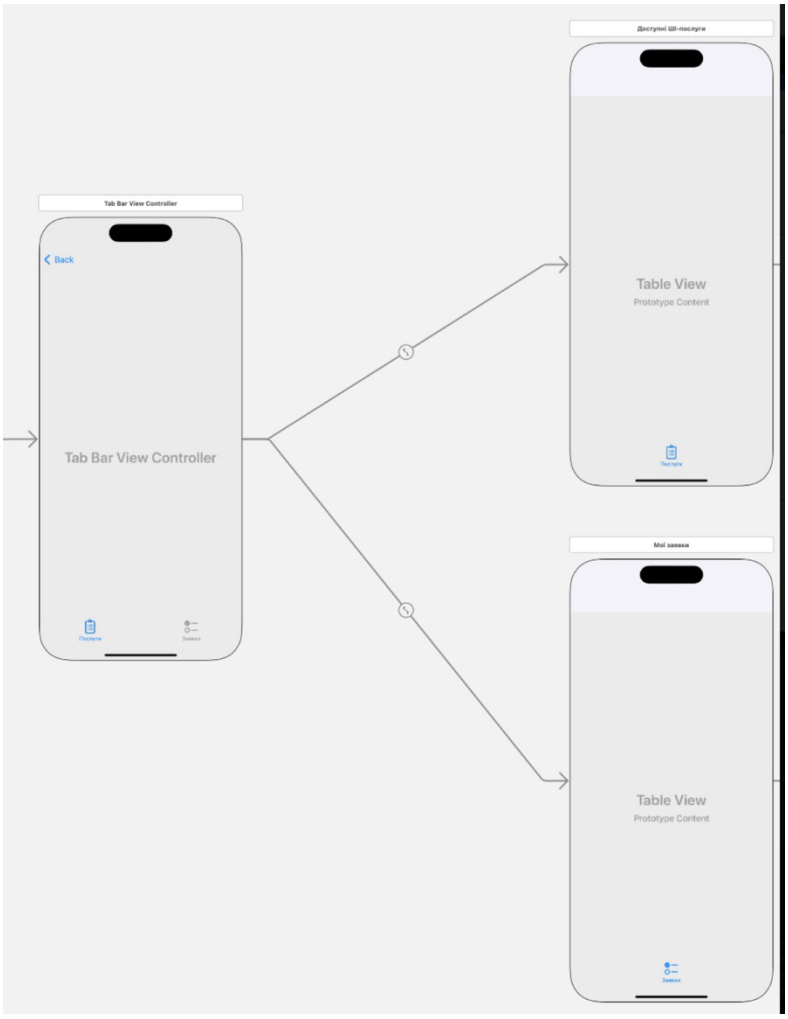


Рисунок 3.7 Ілюстрація панелі вкладок (TabBar) в процесі розробки

Всі ці елементи сприяють зручній та інтуїтивній навігації для користувача у застосунку.

3.3 Зберігання текстових даних та локалізація

Всі текстові поля з файлів з програмним кодом були перенесені в окремий файл UserConstant.swift. Такий підхід має низку переваг, зокрема поліпшує організацію коду, безпеку та підтримку коду.

Розділення коду на логічні блоки робить його більш зрозумілим та зручним для читання. Це дозволяє легше знаходити й модифікувати частини коду, пов’язані з необхідною функціональністю.

Також, перенесення текстових полів у окрему структуру дозволяє краще контролювати доступ до даних і зменшує ймовірність помилок, оскільки

використання текстових полей, які часто повторюються в коді шляхом їх ручного вводу є небезпечним, з огляду на легку можливість помилки. Водночас, якщо поля містяться в окремій структурі слід звернутись до відповідної змінної структури.

Також, такий підхід сприяє зменшенню залежності та зв'язаності різних компонентів коду. Зміни в одній частині коду менше впливатимуть на інші частини коду. Це робить код більш гнучким і зручним для редагування.

На рисунку 3.8 зображено як текстові поля зберігаються в окремому файлі.

AI Med ⟩ AI Med ⟩ UserConstant.swift ⟩ No Selection

```swift
import Foundation

struct K {
    static let appName = "AI-med"
    static let services = [String(localized: "Classification of skin disorders"), String(localized:
        "Detection of pulmonary abnormalities"), String(localized: "Classification of pulmonary
        abnormalities"), String(localized: "Classification of cellular abnormalities"), String(localized:
        "Classification of COVID abnormalities")]
    static let assets = ["SkinDiseases", "LungsAnomalyDetection", "LungsAnomalyClassification",
        "Histology", "CovidAnomaly"]

    struct title {
        static let tabServicesTitle = String(localized: "Services")
        static let tabTicketsTitle = String(localized: "Tickets")
        static let servicesTitle = String(localized: "Available AI-services")
        static let ticketsTitle = String(localized: "My tickets")
    }

    struct segue {
        static let servicesToForm = "ServicesToForm"
        static let ticketsToTicketDetails = "TicketsToTicketDetails"
        static let formToSent = "FormToSent"
        static let formToError = "FormToError"
    }

    struct cell {
        static let service = "serviceCell"
        static let ticket = "ticketCell"
    }

    struct nib {
        static let service = "ServiceCell"
        static let ticket = "TicketCell"
    }

    struct status {
        static let ready = String(localized: "Ready")
        static let wait = String(localized: "Processing")
    }
}
```

Рисунок 3.8 Зберігання текстових полів у файлі UserConstant.swift

Також, дуже важливою складовою інтерфейсу є переклад на англійську мову кожного екрану, для того аби застосунок міг використовувати людьми зі всіх країн.

На рисунку 3.9 зображено як виконано переклад з української мови на англійську в файлі Localizable.xcstrings. Перекладено кожну текстову частину на кожному екрані мобільного застосунку.

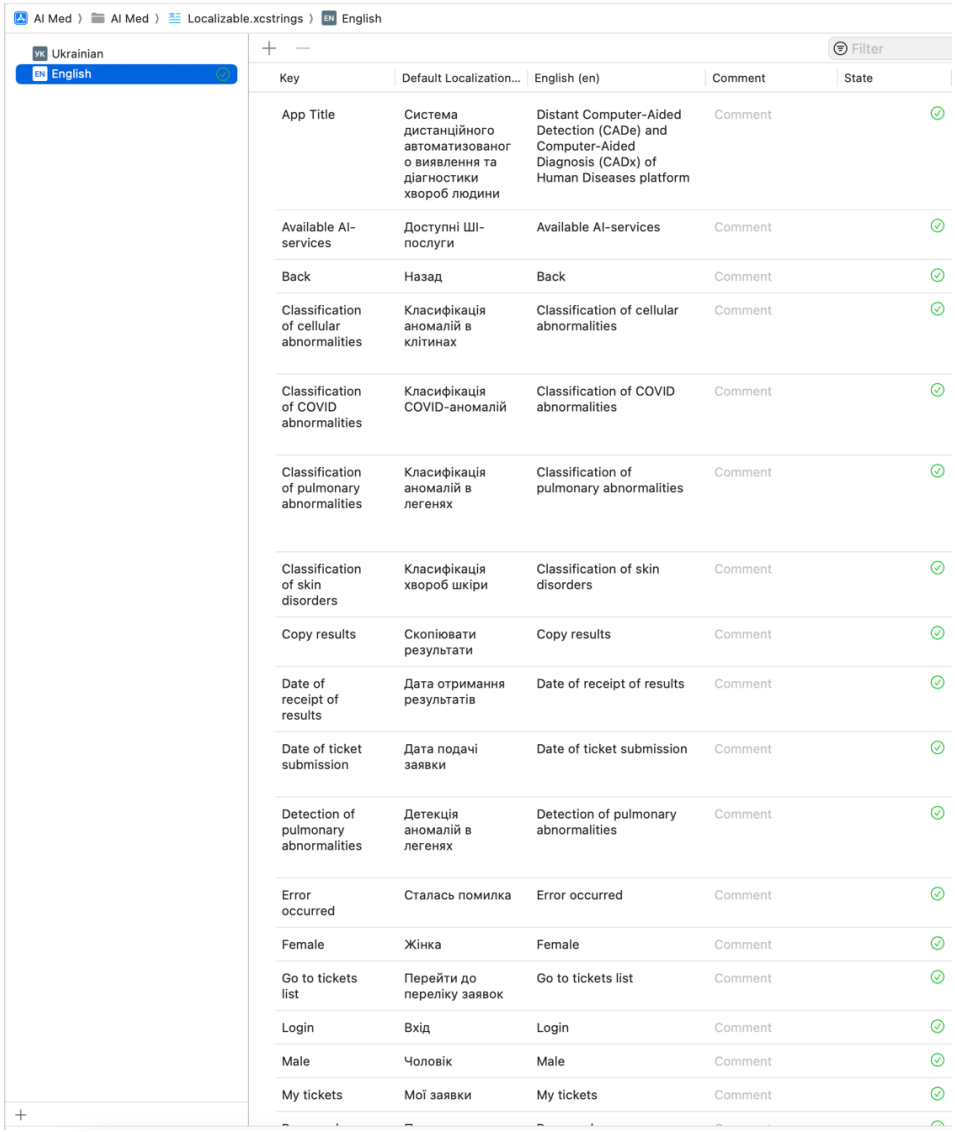


Рисунок 3.9 Приклад файлу, в якому виконаний переклад

3.3. Архітектурний підхід

Для реалізації архітектури поточного мобільного застосунку було обрано підхід MVC, що забезпечує чітке розділення відповідальності між його компонентами. Цей підхід дозволяє структурувати код так, щоб кожен компонент виконував свою специфічну роль, сприяючи кращій підтримці та зрозумілості коду. Надалі буде детальніше описано роль кожного компонента в нашому застосунку.

Компонент Model надає інформацію та реагує на підставі команд контролеру, змінюючи свої властивості[16]. В поточному застосунку, враховуючи що моделі штучного інтелекту розташовані на сервері, Model виконує роль посередника між

сервером та іншими компонентами застосунку. Основні завдання цього компонента включають:

- Зберігання даних: Model зберігає тимчасові дані, отримані від сервера, до моменту коли їх обробить компонент View;
- Взаємодія з API: Model формує запити до сервера, надсилає їх та обробляє отримані відповіді;
- Обробка даних: Model може виконувати базову обробку даних, підготовку до відображення або зберігання.

В нашому випадку, коли застосунок отримує від сервера результати дослідження про ймовірність наявності того чи іншого шкірного захворювання, Model зберігає ці дані у відповідних структурах та забезпечує їхню доступність для View та Controller.

Компонент View відповідає за відображення даних користувачу та взаємодію з ним. В поточному застосунку View реалізовано використовуючи фреймворк UIKit, що забезпечує зручне та інтерактивне середовище інтерфейсу. Основні функції View включають:

- Відображення даних: View відображає дані, отримані від Model, у зручному для користувача форматі;
- Інтерактивність: View обробляє введення користувача, коли він натискає на кнопки, або ж вводить текст тощо;
- Анімації та перехідні ефекти: View також може містити анімації та переходи між різними екранами або станами, забезпечуючи гарніше відображення для користувача.

В цьому застосунку, View відображає 7 екранів, кожен з яких має своє функціональне призначення, та забезпечує взаємодію між ними.

Компонент Controller є посередником між Model та View, обробляючи їхню взаємодію. В поточному застосунку Controller виконує такі функції:

- Обробка логіки взаємодії: Controller обробляє введення користувача, взаємодіючи з Model для отримання або оновлення даних, та передає їх до View для відображення;
- Керування циклом View: Controller відповідає за цикл View, зокрема за ініціалізацію, оновлення та звільнення ресурсів;
- Обробка подій: Controller реагує на події, такі як отримання нових даних від Model, та оновлює View відповідно до змін.

Наприклад, коли користувач переходить на екран з доступними діагностиками, то Controller надсилає запит до Model, та оновлює View відповідно до змін, оскільки в поточному застосунку інформація про доступні діагностики буде отримуватись з сервера.

В поточному застосунку важливою частиною є взаємодія з сервером, де розміщені моделі штучного інтелекту. Для мережевої взаємодії використовуються спеціалізовані бібліотеки, такі як Alamofire для HTTP-запитів та SwiftyJSON для обробки JSON-даних. Ці інструменти забезпечують ефективну та надійну взаємодію між застосунком та сервером. Частина взаємодії з сервером включає такі елементи:

- Формування запитів: Model формує запити до API-сервера, використовуючи Alamofire;
- Обробка відповідей: Отримані від сервера дані у форматі JSON обробляються за допомогою SwiftyJSON та передаються у визначені в Model структури;
- Оновлення View: Після обробки даних Model передає їх до Controller, який оновлює View для відображення нової інформації.

Перевагами архітектури MVC, які покращили розроблюваний застосунок, з огляду на які було прийнято рішення про використання саме цього архітектурного підходу є такі:

- Простота та зрозумілість: MVC є інтуїтивно-зрозумілою архітектурою, що спрощує розробку та підтримку коду;

- Модульність: Розділення на Model, View та Controller дозволяє незалежно розробляти та тестувати кожен з компонентів;
- Широка підтримка: MVC підтримується стандартними бібліотеками iOS, що спрощує інтеграцію з іншими інструментами та технологіями.

Звичайно, як і всі інші архітектурні підходи MVC має певні недоліки, зокрема: великі Controller, оскільки застосунок є великим та має широку взаємодію між View та Model, то Controller може бути занадто великим та складним; залежність компонентів, оскільки якщо неправильно розмежувати відповідальність між компонентами, то це може призвести до тісної залежності між ними, що значним чином знижує гнучкість архітектури.

Загалом же, архітектурний підхід MVC (рисунок 3.10) є одним з найпоширеніших та забезпечує ефективний підхід до розробки iOS застосунків, дозволяючи структурувати код та зосередитись на ключових аспектах взаємодії між користувачем та логікою застосунку[15].

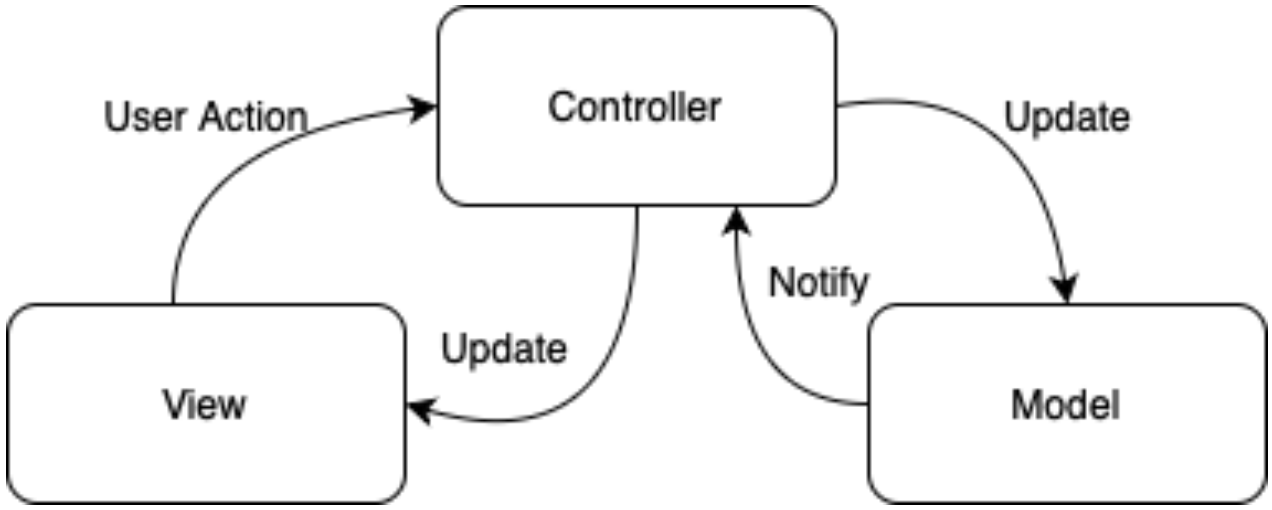


Рисунок 3.10 Зображення взаємодії компонентів архітектурного підходу MVC (Model-View-Controller)

З рисунка 3.11 видно що всі файли розподілені по папкам Model, View, Controller. Таким чином кожна папка відповідає своєму призначенню.

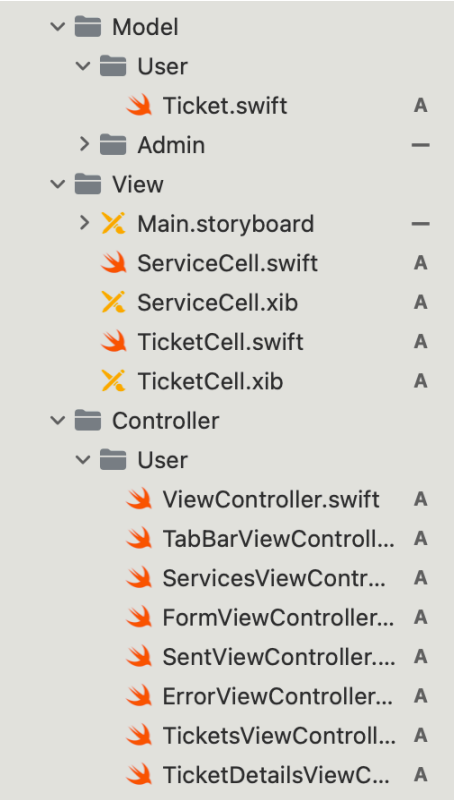


Рисунок 3.11 Розподілення файлів за архітектурним підходом MVC

3.4. Аутентифікація

В поточному мобільному застосунку було реалізовано механізм аутентифікації користувача, що є важливим елементом для забезпечення безпеки та персоналізації роботи з мобільним застосунком. Аутентифікація користувача дозволяє забезпечити доступ до особистих даних та налаштувань, зберігаючи конфіденційність і захищеність інформації.

В цьому застосунку аутентифікація була реалізована через стандартний процес введення електронної пошти та пароля. Після успішної аутентифікації користувач отримує доступ до функціонала, що вимагає авторизації, зокрема надсилання нових та перегляд старих заявок.

Процес аутентифікації складається з таких елементів: введення даних, валідація даних, запит до сервера, обробка відповіді та оновлення інтерфейсу.

На рисунку 3.12 зображено етапи аутентифікації. В першу чергу користувачу надається можливість ввести електрону пошту та пароль, після чого ці дані валідуються (перевірка чи рядки не пусті, перевірка чи було введено вірний формат електронної пошти тощо). Після валідації дані надсилають до сервера, де вони перевіряються. Після перевірки, сервер повертає або дозвіл на вхід, або відмову.

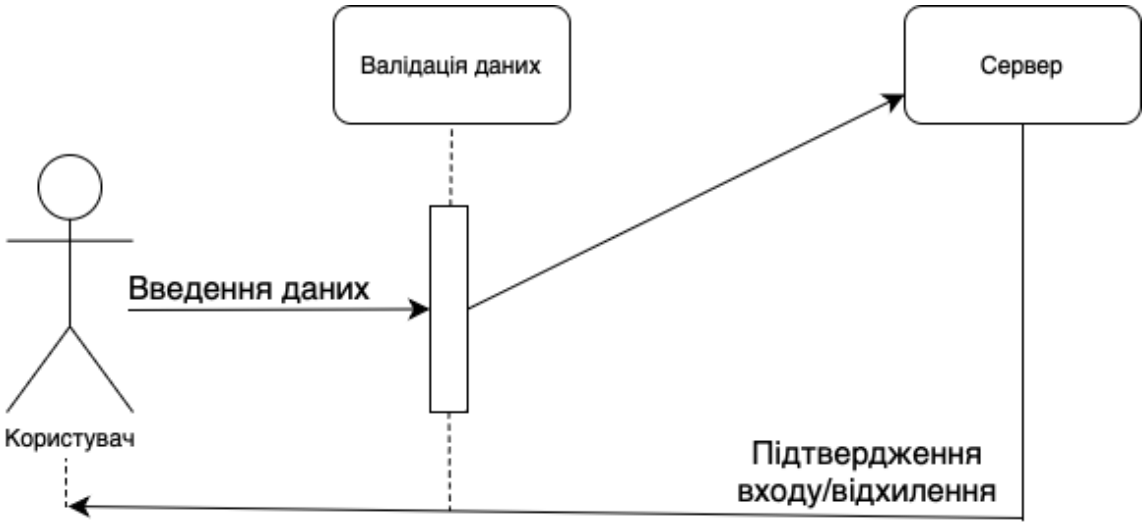


Рисунок 3.12 Схематичне зображення процесу аутентифікації користувача

3.5. Взаємодія застосунку з сервером

Серверний модуль відповідає за обробку запитів від клієнтської частини застосунку. Він забезпечує такі функції:

- Аутентифікація та авторизація користувачів: сервер перевіряє облікові дані користувачів;
- Зберігання та обробка даних: сервер приймає, зберігає та обробляє дані користувачів, включаючи фотографії шкірних уражень;
- Зберігання моделей, та аналіз фотографій з їх допомогою: сервер зберігає моделі штучного інтелекту, які використовуються для аналізу завантажених фотографій;
- Взаємодія з базою даних: сервер забезпечує збереження та отримання необхідних даних з бази даних;

Серверний модуль отримує запити від Model, обробляє їх та повертає відповіді, які включають необхідні дані або результати операцій.

Рисунок 3.13 Схематичне зображення взаємодії застосунку з серверною частиною

На рисунку зображено, який функціонал мобільного застосунку вимагає взаємодії з серверною частиною. Зокрема, при аутентифікації застосунок надсилає на сервер запит з введеними користувачем даними, та отримує від сервера відповідь у вигляді схвалення чи несхвалення авторизації. Також, сервер надсилає до застосунку перелік доступних на поточний момент діагностик.

Сценарії взаємодії з сервером є такими:

1) Аутентифікація користувача: застосунок надсилає запит з введеними користувачем обліковими даними до сервера; сервер перевіряє дані у базі користувачів; сервер надсилає відповідь зі статусом авторизації (успішно/неуспішно);

2) Отримання списку доступних діагностик: застосунок надсилає запит до сервера для отримання актуального списку доступних діагностик; сервер обробляє запит та надсилає відповідь з переліком доступних діагностик;
3) Відправка заявки на діагностику: після обрання діагностики користувач вводить необхідні дані та завантажує фотографію потенційного захворювання; застосунок формує запит і надсилає його на сервер разом із зображенням; сервер приймає запит, обробляє фотографію за допомогою моделей штучного інтелекту та надсилає результати аналізу назад до застосунку;
4) Отримання результатів діагностик: застосунок отримує від сервера результати дослідження фотографії на предмет ймовірності наявності того чи іншого захворювання; сервер також надсилає до застосунку перелік раніше створених та оброблених заявок, щоб користувач міг переглядати історію діагностик;

Model як модуль в поточному застосунку відповідає за управління даними та логікою в застосунку. Його основними функціями при взаємодії з сервером є:

- Формування та відправка запитів до сервера: model готує дані для запитів до сервера та здійснює відправку запиту;
- Обробка отриманих відповідей від сервера: model отримує відповіді від сервера, обробляє їх та передає необхідні дані до інших компонентів застосунку;
- Зберігання локальних даних: model зберігає дані, необхідні для роботи застосунку, які не можуть бути розміщені на сервері;
- Виконання логіки, яка не потребує взаємодії з сервером: model реалізує локальну логіку, наприклад валідацію введених на етапі аутентифікації даних або обробку результатів перед відображенням.

Model взаємодіє з ViewController для надання даних та отримання команд, а також з сервером для виконання запитів.

В коді мобільного застосунку взаємодія з сервером відбувається шляхом створення окремого менеджеру, який відповідає за надсилання запитів. Менеджер в роботі використовує бібліотеки Alamofire та SwiftyJSON для надсилання запитів до сервера, отримання відповідей і обробки даних з нього.

На рисунку 3.14 зображено як менеджер аутентифікації надсилає запити до сервера, шляхом надсилання логіну та паролю. Також, менеджер отримує результати від сервера та обробляє помилки у випадку їх наявності.

```
10      func authenticateUser(username: String, password: String, completion:
            @escaping (Result<String, Error>) -> Void) {
11          let url = "https://your-server-url.com/auth"
12
13          AF.request(url)
14              .authenticate(username: username, password: password)
15              .validate()
16              .responseJSON { response in  'responseJSON(queue:dataPreprocessor:...
17                  switch response.result {
18                  case .success(let value):
19                      let json = JSON(value)
20                      print("JSON: \(json)")
21
22                      if let token = json["token"].string {
23                          completion(.success(token))
24                      } else {
25                          let error = NSError(domain: "", code: 0, userInfo:
                                [NSLocalizedDescriptionKey : "Token not found in
                                response"])
26                          completion(.failure(error))
27                      }
28
29                  case .failure(let error):
30                      completion(.failure(error))
31                  }
32              }
33      }
```

Рисунок 3.14 Менеджер для аутентифікації користувача

Важливим етапом розробки дипломного проєкту є вибір способу взаємодії мобільного застосунку з наявними моделями штучного інтелекту. В ході аналізу та тестувань різних підходів розглядались варіанти розміщення локальних версій моделей штучного інтелекту на пристрої мобільного телефону. Такий підхід має низку переваг, зокрема, можливість використання функціонала застосунку без підключення до інтернету, прискорене отримання результату, з огляду на відсутність необхідності очікування відповіді від сервера, оскільки всі процеси,

пов’язані з аналізом фотографії відбуваються на фізичному пристрої, з якого робиться запит.

Однак, було прийнято рішення про взаємодію з моделями штучного інтелекту виключно шляхом надсилання запитів на сервер, з огляду на такі обставини:

- в планах щодо розробки цієї медичної системи є етап підтвердження лікарем результатів дослідження, що можливо тільки при наявності зв’язку з сервером;
- проведення досліджень через сервер надає можливість вести статистику проведених досліджень та отримувати інші дані, які необхідні для належного функціонування системи та її розвитку;
- використання сервера дозволяє адміністратору модерувати діагностики, моделі та версії моделей шляхом внесення змін до них;
- взаємодія з сервером дозволяє постійно оновлювати моделі штучного інтелекту без їх додаткового завантаження на мобільному пристрої користувача
- розміщення моделей на сервері зменшує завантаження на мобільний застосунок та оптимізує його використання;

Чіткий розподіл функцій між клієнтською та серверною частинами застосунку дозволяє ефективно обробляти дані та забезпечувати користувачів необхідною інформацією в зручному форматі.

3.6. Розгортання та конфігурування мобільного застосунку

Мобільний застосунок працює виключно на мобільних телефонах з операційною системою iOS. В свою чергу, таку операційну систему підтримують мобільні телефони – iPhone. Якщо мобільний застосунок знаходиться в стадії розробки та ще не опублікований в AppStore (магазин застосунків), то зручним варіантом для його розгортання буде його компіляція на ноутбуці/комп’ютері з використанням симулятора iPhone. Це можна зробити використовуючи інтегроване середовище розробки, в якому здійснювалась розробка самого застосунку – Xcode.

Для цього потрібно встановити Xcode на пристрій, відкрити папку з проєктом та запустити симулятор iPhone, на якому відкриється застосунок. Це дозволяє в ході розробки одразу ж бачити внесені зміни та наявно тестувати роботу застосунку та його недоліки. Також, в ході розробки можна запустити мобільний застосунок на фізичному пристрої iPhone, якщо під’єднати його за допомогою кабелю.

Водночас кінцевим етапом розробки публічного застосунку під iOS є його публікація в AppStore. Після завершення всіх етапів в ході розробки цей застосунок буде опублікований та доступним для завантаження в будь-якому куточку планети, що значно спростить процес його розгортання.

ВИСНОВОК ДО РОЗДІЛУ 3

В цьому розділі було вичерпно описано процес розробки мобільного застосунку для підтримки прийняття рішень при діагностуванні захворювань. Застосунок був створений з використанням сучасних технологій та підходів.

Було розглянуто основні функції застосунку, які включають можливість завантаження та обробки фотографій, отримання результатів діагностики на основі аналізу зображень. Додатково реалізовано функції збереження історії попередніх заявок.

Представлено структуру інтерфейсу користувача, що включає основні екрани застосунку, такі як екран авторизація, надсилання заявки, екран з переліком надісланих заявок тощо. Було забезпечено зручність та інтуїтивність навігації між цими екранами.

Також, було описано методи зберігання даних, а також реалізовано підтримку багатомовності, що дозволяє користувачам використовувати застосунок зі зручною для них мовою.

При розробці застосунку було використано архітектурний підхід MVC, який дозволяє ефективно розділити логіку застосунку та його інтерфейс, що сприяє зручності підтримки в майбутньому. Складовими цього підходу є Model, View та Controller, кожен з яких відповідає за певну частину архітектури.

Реалізовано систему аутентифікації користувачів, яка забезпечує захист даних та інформації про користувачів, а також дозволяє ідентифікувати конкретного користувача.

Застосунок активно взаємодіє з сервером, на якому розміщені моделі штучного інтелекту. Взаємодія з сервером відбувається для обробки завантажених зображень та повернення результатів діагностики, аутентифікації користувача тощо. Також було реалізовано обробку можливих помилок під час передачі даних до сервера.

Описано процес налаштування середовища розробки Xcode для компіляції та запуску застосунку на симуляторах мобільних пристроїв.

Таким чином, в цьому розділі було надано вичерпну інформацію про технічні аспекти розробки мобільного застосунку. Використання сучасних технологій забезпечило створення функціонального та надійного інструменту для підтримки прийняття рішень при діагностуванні захворювань.

РОЗДІЛ 4
ОГЛЯД РОЗРОБЛЕНОГО ЗАСТОСУНКУ

В цьому розділі буде зображено результати розробки застосунку. За допомогою скриншотів буде графічно відображено вид кожного екрану.

4.1 Аутентифікація користувача

Користувач вводить логін та пароль, і у випадку успішної аутентифікації переходить на наступний екран з доступними послугами. На етапі аутентифікації спочатку сам застосунок перевіряє чи строки email та пароль не пусті; після цього вони надсилаються на сервер для перевірки на вірність даних.

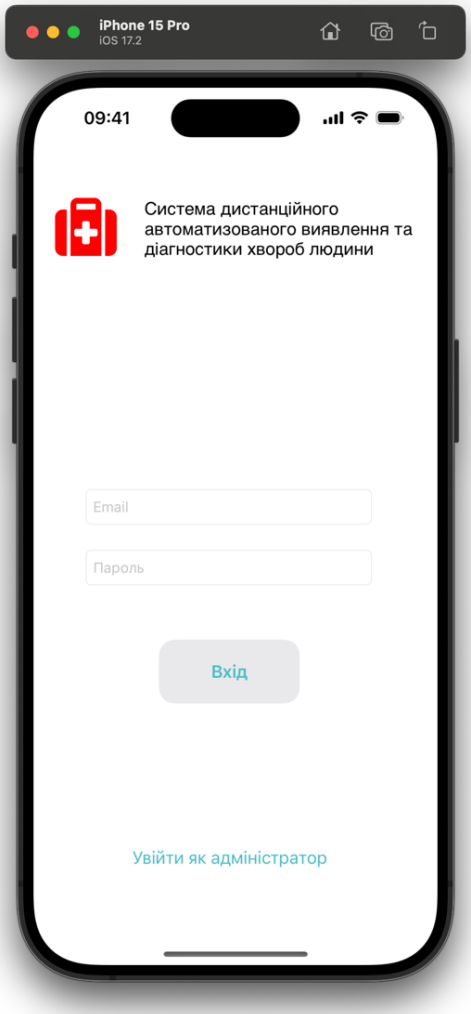


Рисунок 4.1 Екран аутентифікації користувача

Також, важливо зазначити, що всі екрани мобільного застосунку перекладені на англійську мову для зручності його використання іноземними елементами. На рисунку 4.2 зображено приклад англійської локалізації.

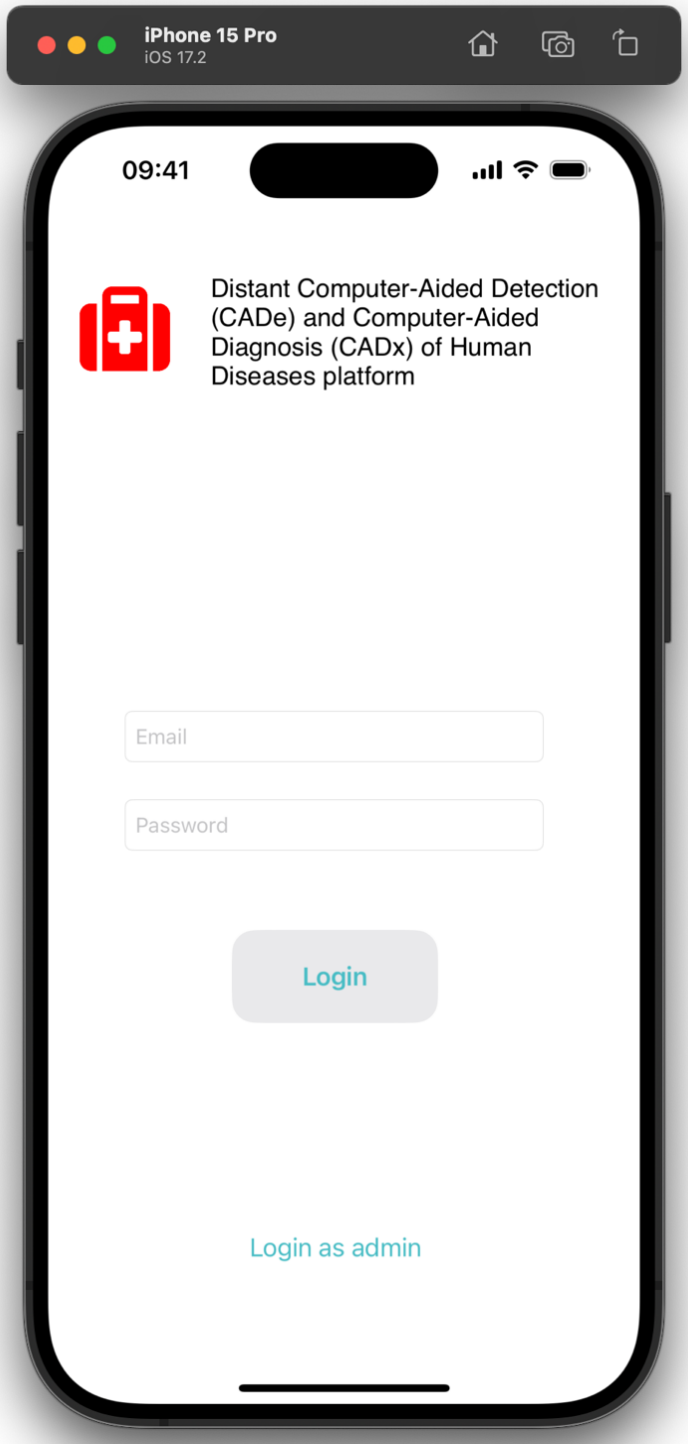


Рисунок 4.2 Екран аутентифікації користувача з англійською локалізацією

4.2 Вибір моделі штучного інтелекту та надсилання заявки

У випадку успішної авторизації користувач потрапляє на екран (рис. 4.3), де може обрати одну з доступних послуг (діагностик), які завантажуються з сервера для заповнення необхідних даних та додавання фото для конкретного виду діагностики. Також, користувач за необхідності може не обирати діагностику, а використовуючи нижню панель перейти на екран з раніше надісланими заявками.

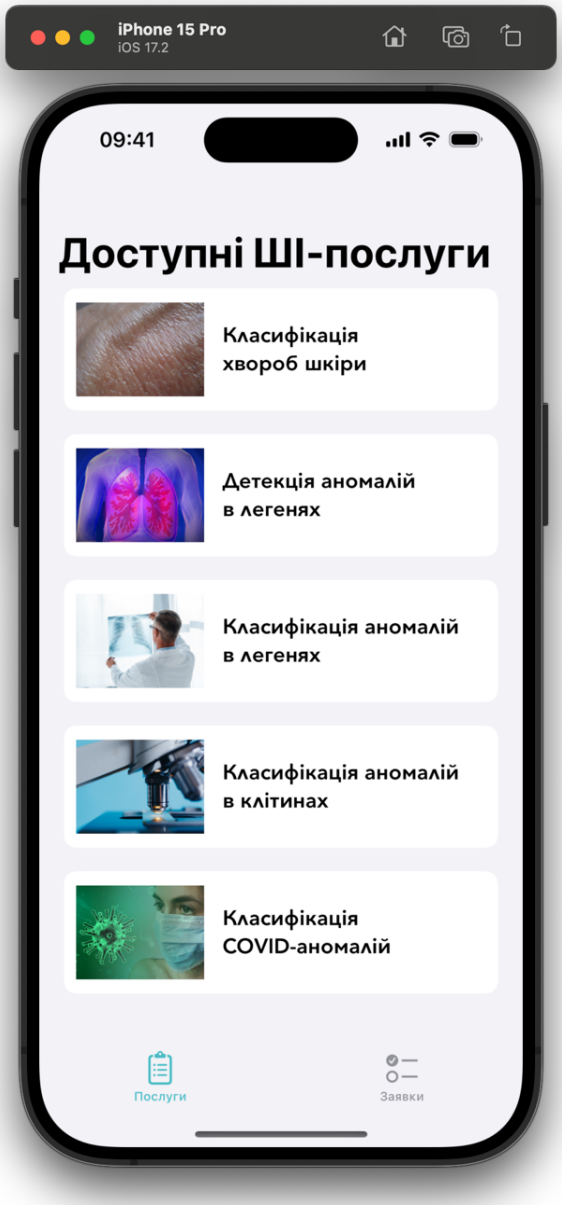


Рисунок 4.3 Екран вибору моделі штучного інтелекту

Після обрання необхідної ШІ-послуги, користувач потрапляє на екран надсилання заявки на обробку (рис. 4.4), на якому вводить необхідні дані для конкретного виду діагностики, а також додає фотографію з галереї, або ж може зробити нову фотографію. Важливо зазначити, що поля для введення імені, номеру телефону та вибору статі, хоч і відображені на екрані, проте їх не можна редагувати. Дані про користувача містяться на сервері й відображаються лише для того, аби користувач був впевнений в їх правильності.

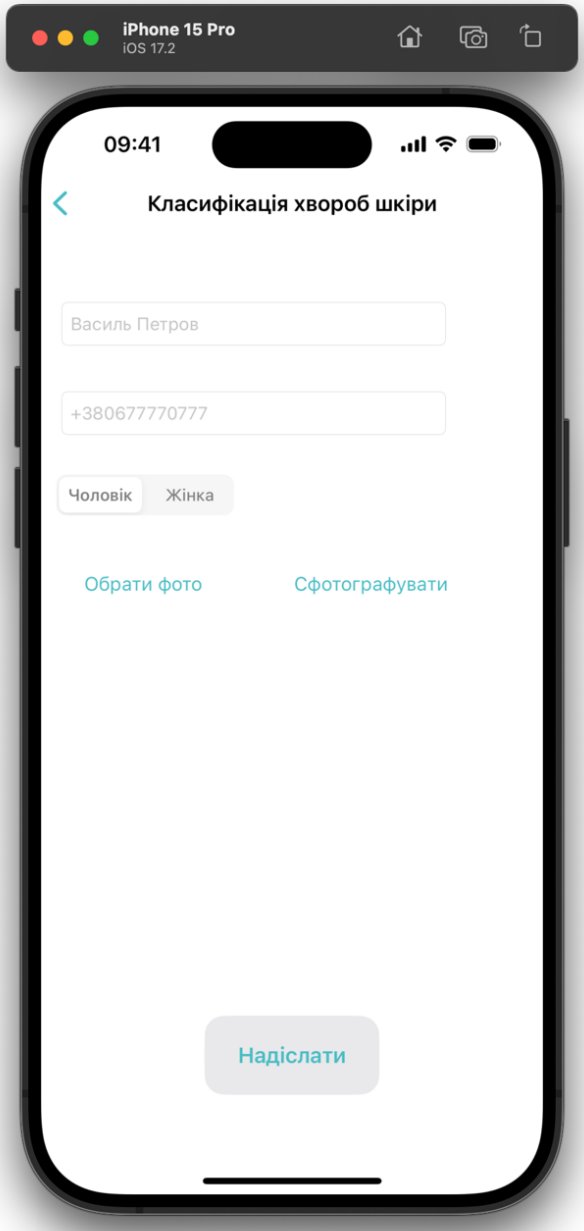


Рисунок 4.4 Екран надсилання заявки на обробку

Після надсилання заявки на обробку, у разі успішної відправки заявки, на екрані відобразиться відповідне повідомлення про успішне надсилання (рисунок 4.5) для того, аби користувач був впевнений, що не виникло помилок в процесі відправки заявки. Також, на екрані повідомлення про успішну відправку заявки є кнопка «Перейти до переліку заявок», яка дозволяє одразу перейти на екран для відстежування статусу заявки, що значно спрощує навігацію користувачу.

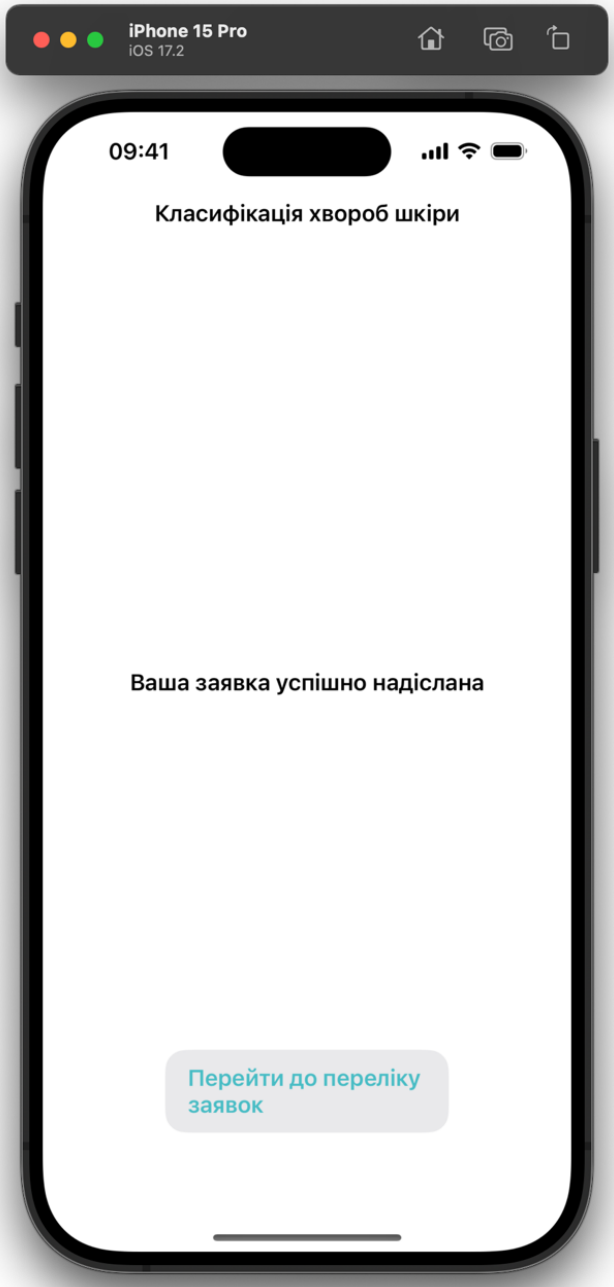


Рисунок 4.5 Екран повідомлення про успішну відправку заявки

У випадку, якщо під час відправки заявки, виникли певні помилки, що доволі часто буває при роботі сервера (такі помилки можуть виникати через нестабільне інтернет-підключення, відсутність підключення, помилки в роботі серверу тощо), тоді користувача перенаправить на екран з повідомленням про помилку (рисунок 4.6). Також, на цьому екрані є кнопка для повернення на попередній екран для повторної спроби надсилання заявки.

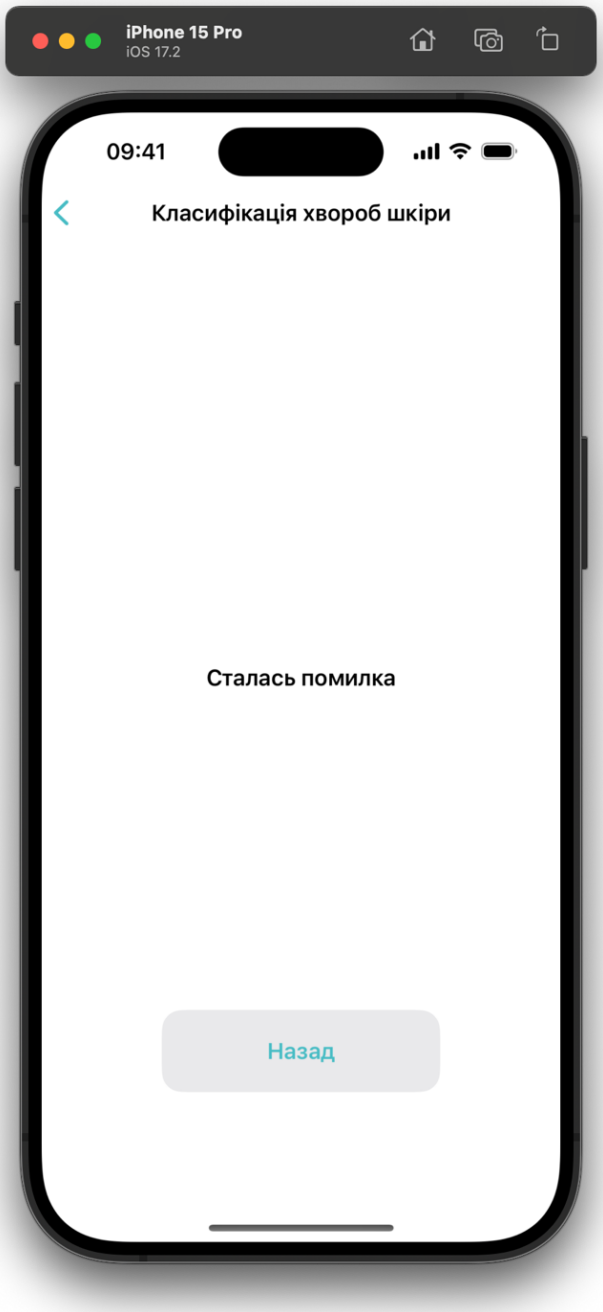


Рисунок 4.6 Екран повідомлення про помилку

4.3 Перегляд вже надісланих заявок

Окремий функціональний блок мобільного застосунку включає перегляд вже надісланих заявок, перегляд статусу їх обробку, а також перегляд деталей кожної окремої заявки. Всі дані про заявки отримуються з сервера. Екран з переліком надісланих заявок (рисунок 4.7) містить дату надсилання заявки, назву діагностики та статус діагностики (готово/в обробці).

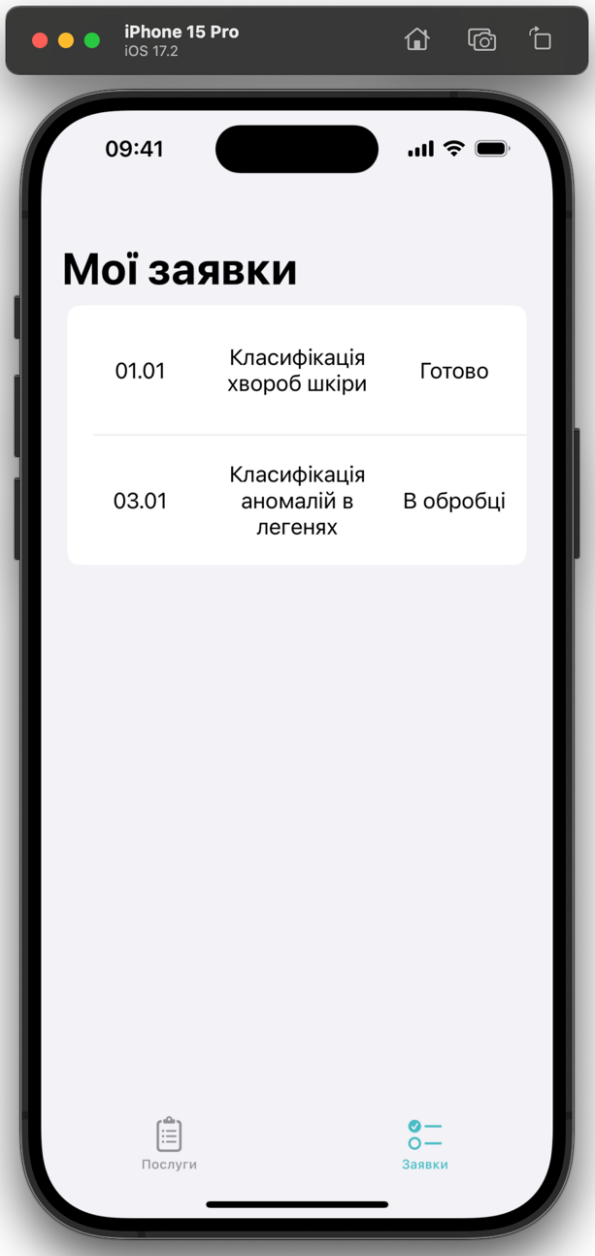


Рисунок 4.7 Екран з переліком надісланих заявок

На екрані з переліком заявок можна обрати будь-яку заявку для перегляду більш детальної інформації про неї. В такому випадку, користувач переходить на екран з деталями кожної окремої заявки (рисунок 4.8), на якому відображені надісланий фотознімок, ідентифікатор заявки, дата подачі заявки, дата отримання результатів та результати дослідження. Також, для користувача доступна кнопка «Скопіювати результати», яка дозволяє скопіювати результат дослідження, тобто ймовірність наявності того чи іншого захворювання на доданому фотознімку.

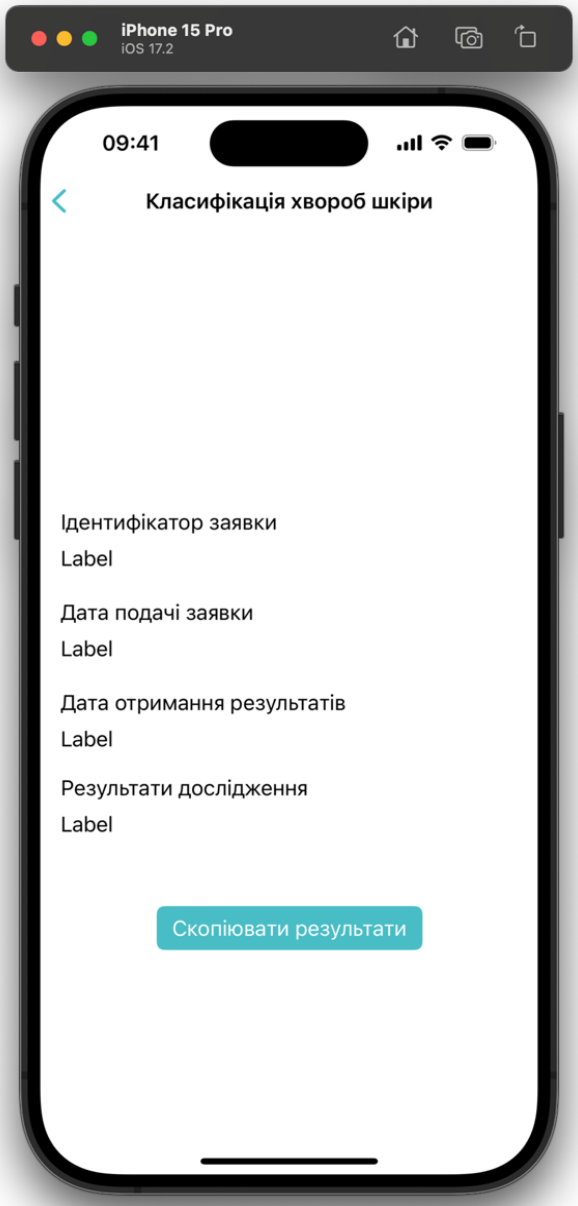


Рисунок 4.8 Екран з деталями кожної окремої заявки

ВИСНОВОК ДО РОЗДІЛУ 4

У 4 розділі було детально розглянуто розроблений мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань. Було розгорнуто застосунок в симуляторі мобільного телефону iPhone 15 Pro (iOS 17.2), який доступний в інтегрованому середовищі розробки Xcode. За допомогою симулятору телефону було зроблено скриншоти основних екранних інтерфейсів, що демонструють функціональні можливості застосунку.

Спочатку було описано процес аутентифікації користувача, що є важливою складовою безпеки й забезпечення персоналізованого використання застосунку. Користувач вводить свої облікові дані для доступу до сервісів застосунку, та в разі успішної аутентифікації, переходить на головний екран з доступними послугами для діагностики.

Далі було розглянуто процес вибору моделі штучного інтелекту для діагностики та надсилання заявки на обробку. Користувач має змогу обрати необхідну діагностику, ввести додаткові дані та додати фотографію для аналізу. Важливим аспектом є те, що поля з особистими даними користувача не підлягають редагування, що забезпечує їх точність та достовірність.

Було описано сценарії успішного та неуспішного надсилання заявок, а також подальші кроки користувача у разі виникнення помилок. Це допомагає забезпечити безперервність процесу діагностики навіть у випадках тимчасових технічних проблем.

Останнім було розглянуто екран для перегляду надісланих заявок. Користувач має змогу бачити список усіх своїх заявок з їхнім статусом та деталями. Цей функціонал дозволяє відстежувати процес обробки кожної заявки та отримувати результати діагностики, що робить застосунок зручним і ефективним для користувачів.

Загалом же, розроблений застосунок забезпечує користувачам зручний та інтуїтивно зрозумілий інтерфейс для взаємодії зі складними процесами діагностики.

ВИСНОВКИ

В межах дипломного проєкту було успішно розроблено мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань.

Основною метою проєкту було створення зручного інструменту, який за допомогою сучасних технологій та взаємодії з моделями штучного інтелекту, здатен забезпечити швидку та точну діагностику, що підвищить якість і доступність медичних послуг.

Реалізація цього застосунку є дуже актуальним питанням, з огляду на зростаючу потребу у швидких та надійних методах діагностики, особливо у віддалених або недостатньо забезпечених медичними ресурсами регіонах.

Розроблений застосунок надає можливість користувачам отримати попередню діагностику на основі взаємодії з сервером, який аналізує зображення, що може стати важливим інструментом для своєчасного виявлення захворювань та прийняття рішень щодо подальшого лікування.

Крім того, застосунок сприяє підвищенню рівня медичної обізнаності серед населення, надаючи доступ до попередньої інформації про стан здоров’я та потенційні захворювання. Це значним чином впливає на профілактику захворювань серед населення, адже користувачі можуть дізнатись про можливу наявність певного захворювання не виходячи з дому.

Розвиток сучасних технологій вимагає від надавачів послуг максимальну цифровізацію послуг для процесуальної економії часу та ресурсу. Таким чином, розроблений мобільний застосунок відповідає сучасним вимогам до медичних інформаційних систем. Важливим є те, що застосунок має значний потенціал для подальшого розвитку та впровадження у практику охорони здоров’я.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Terms & Conditions - Online AI Dermatologist. SKINIVE. URL: https://skinive.com/terms/ (дата звернення: 05.06.2024)
2. Miiskin Patient App. URL: https://miiskin.com/pro/app-patients/ (дата звернення: 05.06.2024).
3. Swift. URL: https://developer.apple.com/swift/ (дата звернення: 05.06.2024).
4. UIKit. URL: https://developer.apple.com/documentation/uikit (дата звернення: 05.06.2024).
5. Objective-C – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Objective-C (дата звернення: 05.06.2024).
6. Flutter - Build apps for any screen. URL: https://flutter.dev (дата звернення: 05.06.2024).
7. SwiftUI Apple Developer Documentation. URL: https://developer.apple.com/documentation/swiftui/ (дата звернення: 05.06.2024).
8. Alamofire & SwiftUI. URL: https://adacn.medium.com/what-is-alamofire-1d41183a0e0a (дата звернення: 05.06.2024).
9. Swift.org. URL: https://www.swift.org/documentation/package-manager/ (дата звернення: 05.06.2024).
10. UINavigationController | Apple Developer Documentation. URL: https://developer.apple.com/documentation/uikit/uinavigationcontroller (дата звернення: 05.06.2024).
11. Tab bars | Apple Developer Documentation. URL: https://developer.apple.com/design/human-interface-guidelines/tab-bars (дата звернення: 05.06.2024).
12. Xcode | Apple Developer Documentation. URL: https://developer.apple.com/documentation/xcode/ (дата звернення: 05.06.2024).

13. Medgic - Scan, Analyze and Detect Skin Problems. URL: https://www.medgic.co/en (дата звернення: 05.06.2024).
14. Medgic - Frequently Asked Questions (FAQ). URL: https://www.medgic.co/en/faq (дата звернення: 05.06.2024).
15. Better Than MVC. URL: https://medium.com/codex/better-than-mvc-537e61928c14 (дата звернення: 06.06.2024).
16. Модель-вид-контролер – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Модель-вид-контролер (дата звернення: 06.06.2024).
17. SwiftyJSON: How To Parse JSON with Swift. URL: https://codewithchris.com/swiftyjson/ (дата звернення: 06.06.2024).
18. CocoaPods.org. URL: https://cocoapods.org (дата звернення: 06.06.2024).
19. GitHub - Carthage/Carthage: A simple, decentralized dependency manager for Cocoa. URL: https://github.com/Carthage/Carthage (дата звернення: 06.06.2024).
20. React Native · Learn once, write anywhere. URL: https://reactnative.dev (дата звернення: 06.06.2024).

ДОДАТОК 1

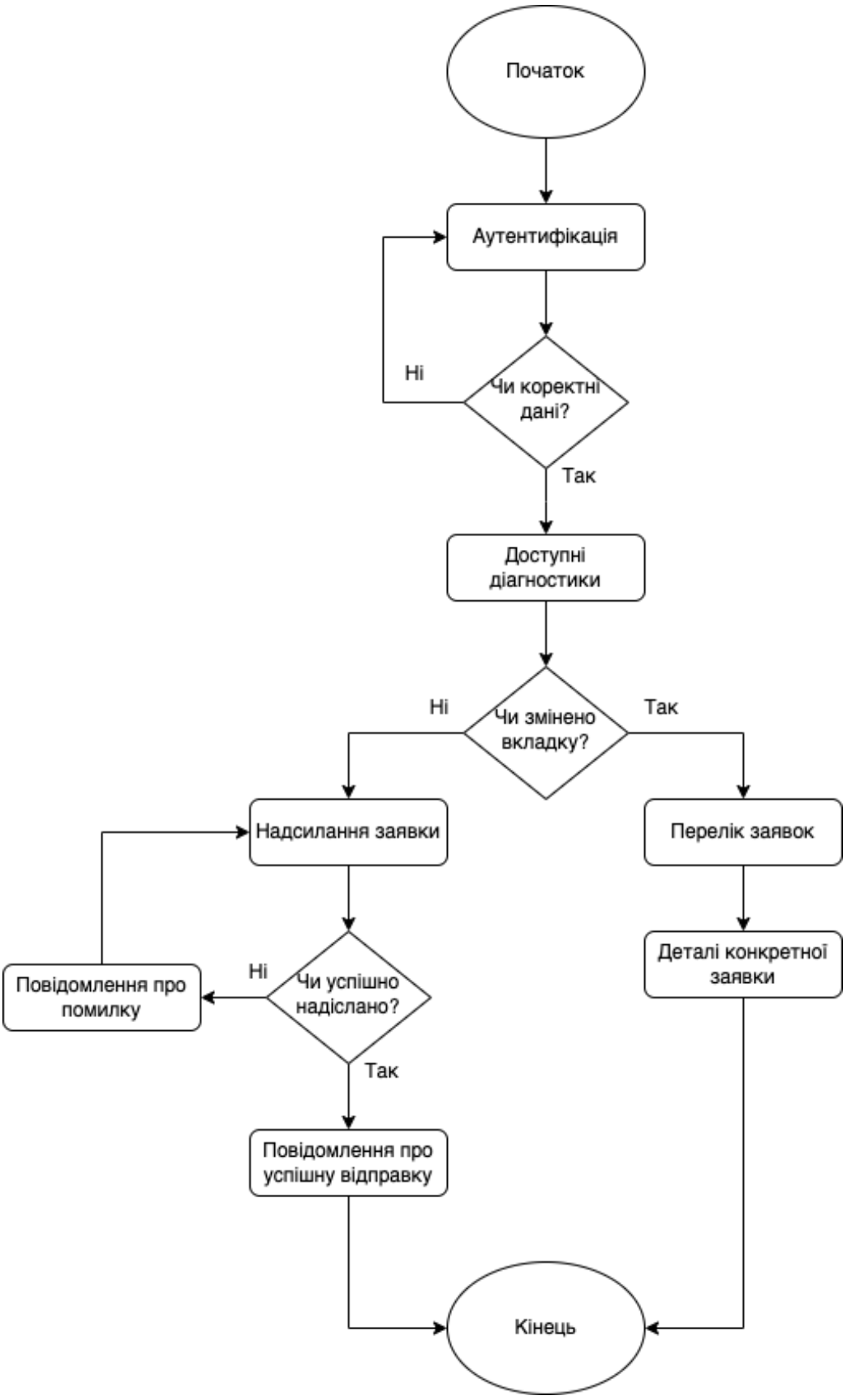
Мобільний застосунок для підтримки прийняття рішень
при діагностуванні захворювань

Алгоритм взаємодії користувача з застосунком (принципова схема)

ІАЛЦ.467100.004 Д1

Аркушів 1

Київ – 2024



					ІАЛЦ.467100.004 Д1		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Кокошко Я.О.			**Мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань** Алгоритм взаємодії користувача з застосунком (принципова схема)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Ковальчук О.М.					
Реценз.		Шимкович В.М.				*КПІ ім. Ігоря Сікорського, ФІОТ, ІП-04*	
Н. Контр.		Волокита А.М.					
Затв.							

ДОДАТОК 2

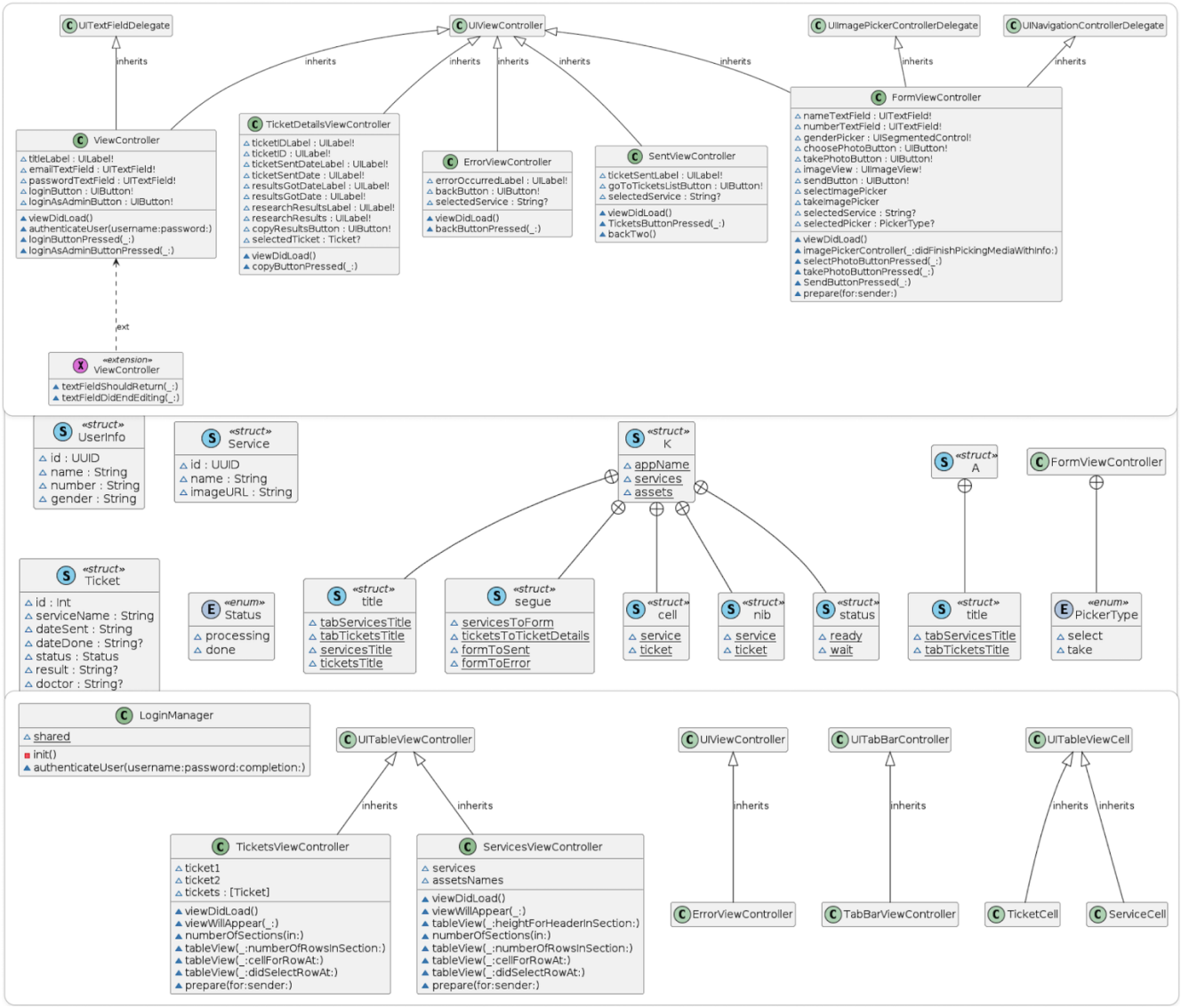
Мобільний застосунок для підтримки прийняття рішень
при діагностуванні захворювань

Діаграма класів (функціональна схема)

ІАЛЦ.467100.004 Д2

Аркушів 1

Київ – 2024



					ІАЛЦ.467100.005 Д2	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Кокошко Я.О.			**Мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань** Діаграма класів (функціональна схема)	Літ. / Аркуш / Аркушів: 1 / 1
Перевірив		Ковальчук О.М.				
Реценз.		Шимкович В.М.				
Н. Контр.		Волокита А.М.				*КПІ ім. Ігоря Сікорського, ФІОТ, ІП-04*
Затв.						

ДОДАТОК 3

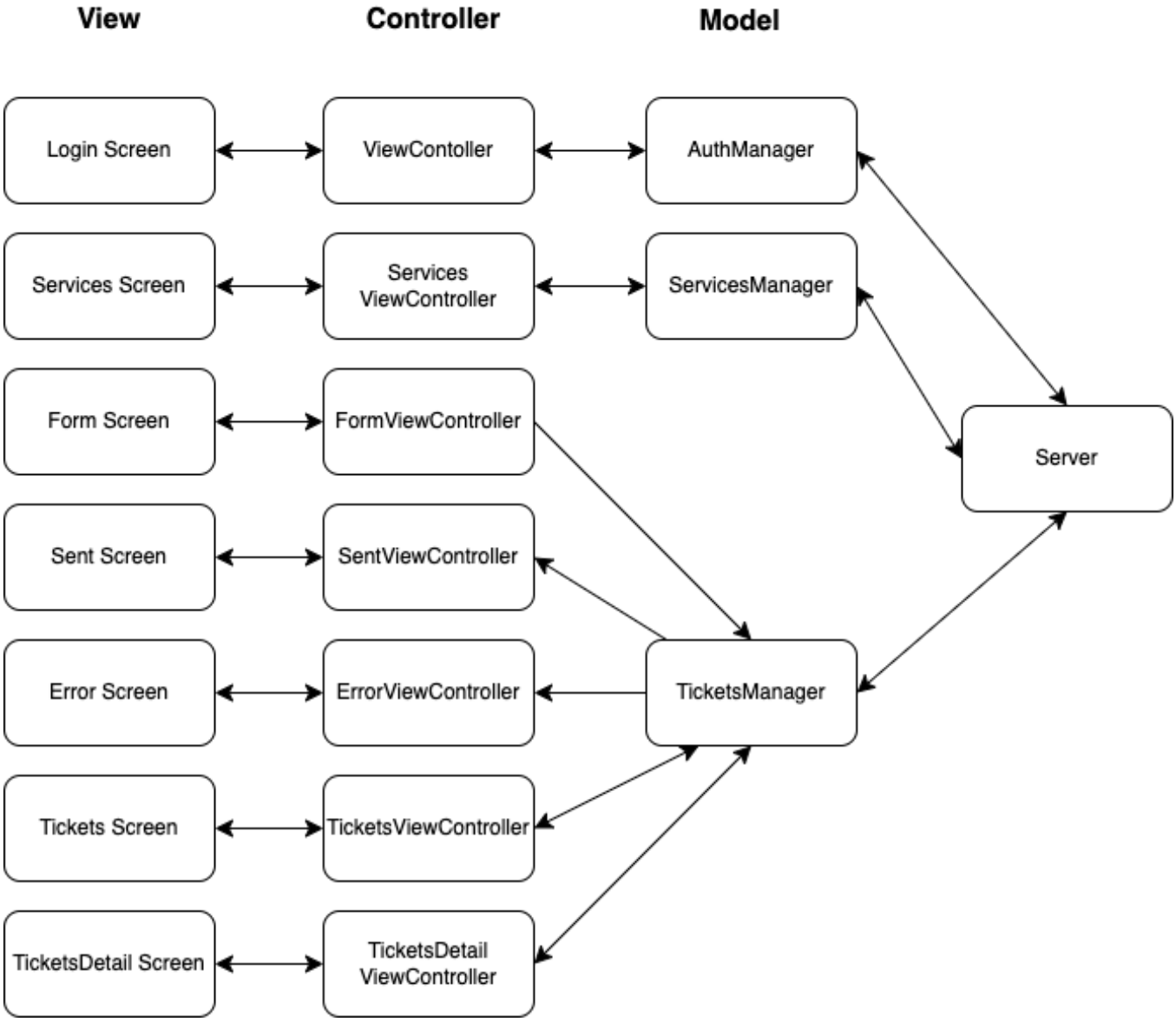
Мобільний застосунок для підтримки прийняття рішень
при діагностуванні захворювань

Архітектурна структура застосунку (структурна схема)

ІАЛЦ.467100.004 Д4

Аркушів 1

Київ – 2024



					ІАЛЦ.467100.006 ДЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Кокошко Я.О.			**Мобільний застосунок для підтримки прийняття рішень при діагностуванні захворювань** Архітектурна структура застосунку (структурна схема)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Ковальчук О.М.					
Реценз.		Шимкович В. М.				*КПІ ім. Ігоря Сікорського, ФІОТ, ІП-04*	
Н. Контр.		Волокита А.М.					
Затв.							

ДОДАТОК 4

Мобільний застосунок для підтримки прийняття рішень
при діагностуванні захворювань

Текст програмного коду

ІАЛЦ.467100.004 Д4

Аркушів 17

Київ – 2024

ViewController.swift

```swift
import UIKit

class ViewController: UIViewController {
    @IBOutlet weak var titleLabel: UILabel!
    @IBOutlet weak var emailTextField: UITextField!
    @IBOutlet weak var passwordTextField: UITextField!
    @IBOutlet weak var loginButton: UIButton!
    @IBOutlet weak var loginAsAdminButton: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

        titleLabel.text = String(localized: "App Title")
        passwordTextField.placeholder = String(localized:
"Password")
        loginButton.setTitle(String(localized: "Login"), for:
.normal)
        loginAsAdminButton.setTitle(String(localized: "Login as
admin"), for: .normal)

        emailTextField.delegate = self
        passwordTextField.delegate = self
    }

    func authenticateUser(username: String, password: String) {
        LoginManager.shared.authenticateUser(username: username,
password: password) { result in
            switch result {
            case .success(let token):
                UserDefaults.standard.setValue(token, forKey:
"Token")
                print("Auth token: \(token)")
                self.performSegue(withIdentifier:
"loginAsUserToTabBar", sender: self)

            case .failure(let error):
                print("Authentication failed: \(error)")
                let alert = UIAlertController(title: "Помилка",
message: "Виникла помилка під час аутентифікації. Спробуйте знову.",
preferredStyle: .alert)
                alert.addAction(UIAlertAction(title:
NSLocalizedString("OK", comment: "Default action"), style: .default,
handler: { _ in
```

```
                    NSLog("The \"OK\" alert occured.")
                }))
                self.present(alert, animated: true, completion: nil)
            }
        }
    }

    @IBAction func loginButtonPressed(_ sender: UIButton) {
        let email = emailTextField.text ?? ""
        let password = passwordTextField.text ?? ""

        authenticateUser(username: email, password: password)
    }

    @IBAction func loginAsAdminButtonPressed(_ sender: UIButton) {
        performSegue(withIdentifier: "loginAsUserToLoginAsAdmin",
sender: self)
    }
}

extension ViewController: UITextFieldDelegate {
    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        self.view.endEditing(true)
        return false
    }

    func textFieldDidEndEditing(_ textField: UITextField) {
        if emailTextField.text != "" && passwordTextField.text != ""
{
            loginButton.isEnabled = true
        }
    }
}
```

TabBarViewController.swift

```swift
import UIKit

class TabBarViewController: UITabBarController {

    override func viewDidLoad() {
        super.viewDidLoad()

        navigationItem.hidesBackButton = true

        tabBar.items![0].title = String(localized: "Services")
        tabBar.items![1].title = String(localized: "Tickets")
    }
}
```

ServicesViewController.swift

```
import UIKit

class ServicesViewController: UITableViewController {
    let services = K.services
    let assetsNames = K.assets

    override func viewDidLoad() {
        super.viewDidLoad()

        tableView.rowHeight = 100.0

        tableView.register(UINib(nibName: K.nib.service, bundle:
nil), forCellReuseIdentifier: K.cell.service)
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        self.tabBarController?.title = K.title.servicesTitle
    }

    // MARK: - Table view data source

    override func tableView(_ tableView: UITableView,
heightForHeaderInSection section: Int) -> CGFloat {
        return 1.0
    }

    override func numberOfSections(in tableView: UITableView) -> Int
{
        return services.count
    }

    override func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
        return 1
    }

    override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
K.cell.service, for: indexPath) as! ServiceCell
        let numberOfSection = indexPath.section

        let cellTitle = services[numberOfSection]
        cell.serviceLabel.text = cellTitle

        let cellImage = UIImage(named: assetsNames[numberOfSection])
```

```
        cell.serviceImage.image = cellImage

        return cell
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt
indexPath: IndexPath) {
        performSegue(withIdentifier: K.segue.servicesToForm, sender:
self)
    }

    override func prepare(for segue: UIStoryboardSegue, sender:
Any?) {
        let destinationVC = segue.destination as! FormViewController

        if let indexPath = tableView.indexPathForSelectedRow {
            destinationVC.selectedService =
services[indexPath.section]
        }
    }
}
```

FormViewController.swift

```
import UIKit

class FormViewController: UIViewController,
UIImagePickerControllerDelegate, UINavigationControllerDelegate {

    @IBOutlet weak var nameTextField: UITextField!
    @IBOutlet weak var numberTextField: UITextField!
    @IBOutlet weak var genderPicker: UISegmentedControl!
    @IBOutlet weak var choosePhotoButton: UIButton!
    @IBOutlet weak var takePhotoButton: UIButton!
    @IBOutlet weak var imageView: UIImageView!
    @IBOutlet weak var sendButton: UIButton!

    let selectImagePicker = UIImagePickerController()
    let takeImagePicker = UIImagePickerController()

    var selectedService: String? {
        didSet {
            self.title = selectedService
        }
    }

    enum PickerType {
        case select
        case take
    }

    var selectedPicker: PickerType? = nil

    override func viewDidLoad() {
        super.viewDidLoad()

        genderPicker.setTitle(String(localized: "Male"),
forSegmentAt: 0)
        genderPicker.setTitle(String(localized: "Female"),
forSegmentAt: 1)
        choosePhotoButton.setTitle(String(localized: "Select
photo"), for: .normal)
        takePhotoButton.setTitle(String(localized: "Take a photo"),
for: .normal)
        sendButton.setTitle(String(localized: "Send"), for: .normal)

        selectImagePicker.delegate = self
        selectImagePicker.sourceType = .photoLibrary
        selectImagePicker.allowsEditing = false

        takeImagePicker.delegate = self
        takeImagePicker.sourceType = .camera
```

```
        takeImagePicker.allowsEditing = false
    }

    func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey
: Any]) {
        if let userPickedImage =
info[UIImagePickerController.InfoKey.originalImage] as? UIImage {
            imageView.image = userPickedImage
        }
        switch selectedPicker {
        case .select:
            selectImagePicker.dismiss(animated: true, completion:
nil)
        case .take:
            takeImagePicker.dismiss(animated: true, completion: nil)
        case nil:
            print("Ya kit")
        }
    }

    @IBAction func selectPhotoButtonPressed(_ sender: UIButton) {
        selectedPicker = .select
        present(selectImagePicker, animated: true, completion: nil)
    }

    @IBAction func takePhotoButtonPressed(_ sender: UIButton) {
        selectedPicker = .take
        present(takeImagePicker, animated: true, completion: nil)
    }

    @IBAction func SendButtonPressed(_ sender: UIButton) {
        let toSent = K.segue.formToSent
        //        let toError = K.segue.formToError
        performSegue(withIdentifier: toSent, sender: self)
    }

    override func prepare(for segue: UIStoryboardSegue, sender:
Any?) {
        let destinationVC = segue.destination as! SentViewController
        //        let destinationVC = segue.destination as!
ErrorViewController

        destinationVC.selectedService = selectedService
    }
}
```

SentViewController.swift

```
import UIKit

class SentViewController: UIViewController {
    @IBOutlet weak var ticketSentLabel: UILabel!
    @IBOutlet weak var goToTicketsListButton: UIButton!

    var selectedService: String? {
        didSet {
            self.title = selectedService
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        ticketSentLabel.text = String(localized: "Your ticket has
been successfully sent")
        goToTicketsListButton.setTitle(String(localized: "Go to
tickets list"), for: .normal)

        navigationItem.hidesBackButton = true
    }

    @IBAction func TicketsButtonPressed(_ sender: UIButton) {
        backTwo()
    }

    func backTwo() {
        let viewControllers: [UIViewController] =
self.navigationController!.viewControllers as [UIViewController]
        let tabBarController = viewControllers[viewControllers.count
- 3] as! TabBarViewController

self.navigationController!.popToViewController(tabBarController,
animated: true)
        tabBarController.selectedIndex = 1
        tabBarController.title = K.title.ticketsTitle
    }
}
```

ErrorViewController.swift

```swift
import UIKit

class ErrorViewController: UIViewController {
    @IBOutlet weak var errorOccurredLabel: UILabel!
    @IBOutlet weak var backButton: UIButton!

    var selectedService: String? {
        didSet {
            self.title = selectedService
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        errorOccurredLabel.text = String(localized: "Error
occurred")
        backButton.setTitle(String(localized: "Back"), for: .normal)
    }

    @IBAction func backButtonPressed(_ sender: UIButton) {
        navigationController?.popViewController(animated: true)
    }
}
```

TicketsViewController.swift

```
import UIKit

class TicketsViewController: UITableViewController {
    let ticket1 = Ticket(id: 1,
                         serviceName: "Класифікація хвороб шкіри",
                         dateSent: "01.01",
                         dateDone: "01.02",
                         status: .done,
                         result: "Все погано",
                         doctor: "Зверніться до лікаря")

    let ticket2 = Ticket(id: 2,
                         serviceName: "Класифікація аномалій в
легенях",
                         dateSent: "03.01",
                         dateDone: nil,
                         status: .processing,
                         result: nil,
                         doctor: nil)

    var tickets: [Ticket] = []

    override func viewDidLoad() {
        super.viewDidLoad()

        tickets = [ticket1, ticket2]

        tableView.rowHeight = 100.0

        tableView.register(UINib(nibName: K.nib.ticket, bundle:
nil), forCellReuseIdentifier: K.cell.ticket)
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        self.tabBarController?.title = K.title.ticketsTitle
    }

    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int
{
        return 1
    }

    override func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
```

```
        return tickets.count
    }

    override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
K.cell.ticket, for: indexPath) as! TicketCell

        cell.date.text = tickets[indexPath.row].dateSent
        cell.name.text = tickets[indexPath.row].serviceName

        if tickets[indexPath.row].status == .done {
            cell.status.text = K.status.ready
        }

        if tickets[indexPath.row].status == .processing {
            cell.status.text = K.status.wait
        }

        return cell
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt
indexPath: IndexPath) {
        performSegue(withIdentifier: K.segue.ticketsToTicketDetails,
sender: self)
    }

    override func prepare(for segue: UIStoryboardSegue, sender:
Any?) {
        let destinationVC = segue.destination as!
TicketDetailsViewController

        if let indexPath = tableView.indexPathForSelectedRow {
            destinationVC.selectedTicket = tickets[indexPath.row]
        }
    }
}
```

TicketsDetailViewController.swift

```
import UIKit

class TicketDetailsViewController: UIViewController {
    @IBOutlet weak var ticketIDLabel: UILabel!
    @IBOutlet weak var ticketID: UILabel!

    @IBOutlet weak var ticketSentDateLabel: UILabel!
    @IBOutlet weak var ticketSentDate: UILabel!

    @IBOutlet weak var resultsGotDateLabel: UILabel!
    @IBOutlet weak var resultsGotDate: UILabel!

    @IBOutlet weak var researchResultsLabel: UILabel!
    @IBOutlet weak var researchResults: UILabel!

    @IBOutlet weak var copyResultsButton: UIButton!

    var selectedTicket: Ticket? {
        didSet {
            self.title = selectedTicket?.serviceName
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        ticketIDLabel.text = String(localized: "Ticket id")
        ticketSentDateLabel.text = String(localized: "Date of ticket
submission")
        resultsGotDateLabel.text = String(localized: "Date of
receipt of results")
        researchResultsLabel.text = String(localized: "Research
results")
        copyResultsButton.setTitle(String(localized: "Copy
results"), for: .normal)
    }

    @IBAction func copyButtonPressed(_ sender: UIButton) {
        let label = researchResults!

        UIPasteboard.general.string = label.text

        // Optionally, provide some visual feedback to the user
        label.transform = CGAffineTransform(scaleX: 0.9, y: 0.9)
        UIView.animate(withDuration: 0.2, delay: 0,
usingSpringWithDamping: 0.5, initialSpringVelocity: 0.5, options:
[], animations: {
            label.transform = .identity
```

```
        }, completion: nil)
    }
}
```

LoginManager.swift

```swift
import Foundation
import Alamofire
import SwiftyJSON

class LoginManager {
    static let shared = LoginManager()

    private init() {}

    func authenticateUser(username: String, password: String,
completion: @escaping (Result<String, Error>) -> Void) {
        let url = "https://your-server-url.com/auth"

        AF.request(url)
            .authenticate(username: username, password: password)
            .validate()
            .responseJSON { response in
                switch response.result {
                case .success(let value):
                    let json = JSON(value)
                    print("JSON: \(json)")

                    if let token = json["token"].string {
                        completion(.success(token))
                    } else {
                        let error = NSError(domain: "", code: 0,
userInfo: [NSLocalizedDescriptionKey : "Token not found in
response"])
                        completion(.failure(error))
                    }

                case .failure(let error):
                    completion(.failure(error))
                }
            }
    }
}
```

UserConstant.swift

```
import Foundation

struct K {
    static let appName = "AI-med"
    static let services = [String(localized: "Classification of skin
disorders"), String(localized: "Detection of pulmonary
abnormalities"), String(localized: "Classification of pulmonary
abnormalities"), String(localized: "Classification of cellular
abnormalities"), String(localized: "Classification of COVID
abnormalities")]
    static let assets = ["SkinDiseases", "LungsAnomalyDetection",
"LungsAnomalyClassification", "Histology", "CovidAnomaly"]

    struct title {
        static let tabServicesTitle = String(localized: "Services")
        static let tabTicketsTitle = String(localized: "Tickets")
        static let servicesTitle = String(localized: "Available AI-
services")
        static let ticketsTitle = String(localized: "My tickets")
    }

    struct segue {
        static let servicesToForm = "ServicesToForm"
        static let ticketsToTicketDetails = "TicketsToTicketDetails"
        static let formToSent = "FormToSent"
        static let formToError = "FormToError"
    }

    struct cell {
        static let service = "serviceCell"
        static let ticket = "ticketCell"
    }

    struct nib {
        static let service = "ServiceCell"
        static let ticket = "TicketCell"
    }

    struct status {
        static let ready = String(localized: "Ready")
        static let wait = String(localized: "Processing")
    }
}
```

ServiceCell.swift

```swift
//
//  ServiceCell.swift
//  AI Med
//
//  Created by Yaroslav on 06.04.2024.
//

import UIKit

class ServiceCell: UITableViewCell {

    @IBOutlet weak var serviceImage: UIImageView!

    @IBOutlet weak var serviceLabel: UILabel!



    override func awakeFromNib() {
        super.awakeFromNib()



    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)

        // Configure the view for the selected state
    }

}
```

TicketCell.swift

```swift
import UIKit

class TicketCell: UITableViewCell {
    @IBOutlet weak var date: UILabel!
    @IBOutlet weak var name: UILabel!
    @IBOutlet weak var status: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()

    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)

    }
}
```