Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка системи візуальної детекції об'єктів рослинного походження на основі штучного інтелекту»

Виконав: студент 4 курсу
групи 5ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Жеребнюк М.Р.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Мельник О.В.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Кожем'яко А.В.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
« 21 » березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Жеребнюку Максиму Романовичу

1. Тема роботи – «Розробка системи візуальної детекції об'єктів рослинного походження на основі штучного інтелекту»

Керівник роботи: Мельник Олександр Васильович, к. т. н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студенткою роботи 2 червня 2025 р.

3. Вихідні дані до роботи: тип програми – мобільний застосунок; модель розробки – інкрементна; засоби організації даних програми – Flutter SDK і мову Dart, бібліотеку sqflite; вхідні дані: цифрові зображення рослин у форматах JPEG/PNG, вибір користувачем режиму; вихідні дані: користувацький інтерфейс із картками результатів, вкладка зі збереженими об'єктами; середовище розробки – Android Studio з Flutter SDK, мова програмування – Dart.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз розвитку технологій систем візуальної детекції об'єктів рослинного походження; розробка структури та алгоритмів програмного застосунку; розробка системи візуальної детекції об'єктів рослинного походження; тестування програмного забезпечення; висновки; список використаних джерел;

додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; м
об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практи
цінність одержаних результатів; порівняльний аналіз аналогів, використ
технології; Розробка головного модуля взаємодії із застосунком; Розробка мод
для взаємодії з API; тестування; висновки; апробація результатів роботи
публікації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Мельник О.В., к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз розвитку технологій систем візуальної детекції об'єктів рослинного походження	25.03.25 – 06.04.25	вик
2	Розробка структури та алгоритмів програмного застосунку	07.04.25 – 20.04.25	вик
3	Варіантний аналіз та вибір мови програмування розробки	20.04.25 – 07.05.25	вик
4	Розробка системи візуальної детекції об'єктів рослинного походження	07.05.25 – 20.05.25	вик
5	Тестування програмного забезпечення	20.05.25– 30.05.25	вик
6	Оформлення матеріалів до захисту БКР	25.03.25 – 06.04.25	вик

Студент _____ Жеребнюк М.
(підпис) (прізвище та ініціали

Керівник бакалаврської кваліфікаційної роботи _____ Мельник О.В
(підпис) (прізвище та ініціали

АНОТАЦІЯ

УДК 004:005.9

Жеребнюк М. Р. Розробка системи візуальної детекції об’єктів рослинного походження на основі штучного інтелекту: бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2025. 86 с.

У бакалаврській кваліфікаційній роботі розроблено мобільний застосунок Plant Health для автоматизованої ідентифікації видів рослин і виявлення їхніх захворювань за зображенням із використанням моделей машинного навчання. Застосунок дозволяє робити знімок або завантажувати існуюче зображення, обробляти його на стороні клієнта та отримувати перелік найбільш вірогідних результатів з короткими описами.

Побудовано трирівневу архітектуру: презентаційний шар із екраном завантаження, головним екраном і вкладкою «Мої рослини», логіку з модулями обробки зображень і генерації рекомендацій, а також шар збереження даних у локальній базі SQLite. Реалізовано сортування й фільтрацію результатів за рівнем впевненості, збереження історії ідентифікацій із фотографіями, відображення детальної інформації про вибрану рослину та систему локальних сповіщень із порадами щодо догляду.

Програмне забезпечення розроблено мовою Dart із використанням фреймворка Flutter у середовищі Android Studio. Проведено функціональне та користувацьке тестування, яке підтвердило стабільність роботи застосунку, коректність обробки зображень та відповідність початковим вимогам.

Ключові слова: мобільний застосунок, Flutter, ідентифікація рослин, діагностика хвороб, машинне навчання, SQLite.

ABSTRACT

UDC 004:005.9

Zherebniuk M. R. Development of the Plant Health mobile application for visual plant identification and disease diagnosis using artificial intelligence: bachelor's qualification work in specialty 121 Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2025. 86 p.

In the bachelor’s qualification work, a mobile application called Plant Health was developed for automated identification of plant species and detection of their diseases based on image analysis using machine learning models. The application enables the user to capture a photo or upload an existing image, process it on the client side, and receive a ranked list of the most probable matches with concise descriptions.

A three-layer system architecture was designed, comprising a presentation layer with a splash screen, main interface, and “My Plants” section; a business-logic layer containing modules for image processing and recommendation generation; and a data-persistence layer using a local SQLite database. Functionality was implemented for sorting and filtering results by confidence level, saving identification history with accompanying photographs, displaying detailed plant information, and issuing local notifications with care tips.

The software was developed in Dart using the Flutter framework within Android Studio. Functional and usability testing was carried out, confirming the application’s stability, accuracy of image processing, and fulfillment of the initial project requirements.

Keywords: mobile application, Flutter, plant identification, disease diagnosis, machine learning, SQLite.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Мобільні технології все активніше проникають у різні сфери життя, включаючи аграрний сектор, ботаніку, садівництво. У цьому контексті особливої актуальності набуває застосування комп’ютерного зору для автоматизованого розпізнавання рослин і діагностики їхніх хвороб. Візуальна ідентифікація за допомогою мобільних застосунків дозволяє зменшити залежність від фахівців, забезпечити швидку діагностику і сприяти прийняттю обґрунтованих рішень у догляді за рослинами.

Поширення відкритих API та широке впровадження нейромережевих моделей зробило можливим створення систем, які можуть за лічені секунди аналізувати зображення і надавати точні результати з високою ймовірністю. Проте більшість існуючих мобільних застосунків мають обмеження щодо мовної локалізації, не надають підтримки українською та, що важливо, часто розділяють функціонал ідентифікації та виявлення хвороб у різні додатки.

З іншого боку, простота інтерфейсу, зручність взаємодії та підтримка локальних мов мають вирішальне значення для ефективності та популярності таких рішень. Користувачам важливо не лише отримати наукову назву рослини, але й доступно зрозуміти, що це за об’єкт, чи він безпечний, корисний або уражений шкідниками. Сучасні системи часто перевантажені інформацією, поданою англійською мовою або у складній науковій формі. Вирішення цих недоліків є ключем до підвищення цінності програмного продукту для кінцевого користувача.

У зв’язку з цим доцільною є розробка нового мобільного застосунку для візуальної детекції об’єктів рослинного походження.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Метою роботи є підвищення точності та доступності процесу ідентифікації об'єктів рослинного походження за допомогою мобільного програмного застосунку на основі технологій штучного інтелекту.

Основними задачами дослідження є:

- розробити архітектуру застосунку для детекції рослин і хвороб;
- розробити інтерфейс користувача застосунку;
- розробити алгоритми роботи застосунку;
- розробити функціонал застосунку;
- провести тестування розробленого застосунку.

Об'єкт дослідження – Об’єктом дослідження є процес розробки системи для візуальної детекції об’єктів рослинного походження.

Предмет дослідження – Предметом дослідження є методи та засоби розробки системи для візуальної детекції об’єктів рослинного походження.

Методи дослідження.

Для реалізації поставлених задач використовувалися емпіричні методи розробки мобільних застосунків; аналітичні методи та елементарна алгебра для функціонального тестування; алгоритмічні методи побудови інтерфейсів користувача; а також методи тестування програмних продуктів.

Новизна отриманих результатів:

1. Подальшого розвитку отримав метод візуальної детекції об’єктів рослинного походження на основі комп’ютерного зору, який, на відміну від відомих, дозволяє здійснювати виявлення хвороб.

2. Подальшого розвитку отримав метод аналізу та рекомендацій, який на відміну від відомих, надає персоналізовані рекомендації, залежно від результатів візуальної детекції.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає у тому, що на основі теоретичних положень, розроблено програмні засоби для швидкої візуальної детекції об’єктів рослинного походження та аналізу їхнього стану без потреби у додатковому обладнанні чи спеціалізованих знаннях.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто.

Апробація матеріалів бакалаврської кваліфікаційної роботи. Результати

бакалаврської кваліфікаційної роботи доповідалися на 2-й конференції «Modern Scientific Research: Theoretical and Practical Aspects» (26-28 травня). 2025 р., Рига, Латвія.

Публікації. Тези доповіді «Аналіз сучасних технологій візуальної ідентифікації рослин» опубліковані на 2-й конференції «Modern Scientific Research: Theoretical and Practical Aspects» (26-28 травня). 2025 р., Рига, Латвія.

1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ СИСТЕМ ВІЗУАЛЬНОЇ ДЕТЕКЦІЇ ОБ'ЄКТІВ РОСЛИННОГО ПОХОДЖЕННЯ

1.1 Аналіз стану технологій систем візуальної детекції об'єктів рослинного походження

Технології візуальної ідентифікації об'єктів навколишнього середовища на основі зображень, зокрема цифрових фотографій, стали однією з найперспективніших складових сучасних інтелектуальних систем. Вони знаходять широке застосування в різних галузях людської діяльності – від медицини до автомобілебудування, але особливої актуальності набувають у сфері біологічного моніторингу, сільського господарства та охорони довкілля [1]. Одним із важливих і активно розвиваних напрямів у межах цих технологій є автоматичне розпізнавання об'єктів рослинного походження. Йдеться насамперед про визначення виду рослини або ж виявлення на ній ознак хвороб чи ушкоджень – усе це на основі зображення, отриманого за допомогою звичайної мобільної камери або іншого цифрового сенсора.

Зростання інтересу до подібних систем є абсолютно закономірним і зумовлюється як загальними технологічними трендами, так і конкретними прикладними потребами у різних сферах [2]. Так, в аграрному секторі, де своєчасна діагностика стану культур є критично важливою для збереження врожаю, подібні рішення дозволяють автоматизувати процеси, які раніше потребували участі фахівців і затрат часу. У ботаніці й екології такі інструменти використовуються для каталогізації біорізноманіття, фіксації рідкісних чи інвазивних видів, контролю за станом рослинності в певних регіонах [3]. У повсякденному житті подібні застосунки можуть стати у пригоді як освітній інструмент для школярів, студентів або ж просто допитливих користувачів, а також як практичний засіб догляду за кімнатними чи садовими рослинами.

З технічної точки зору, реалізація візуальної ідентифікації передбачає застосування методів комп'ютерного зору – галузі, що перебуває на стику інформатики, математики та штучного інтелекту [4]. Найпоширенішим на сьогодні підходом до розв'язання задач класифікації об'єктів на зображеннях є використання згорткових

нейронних мереж (Convolutional Neural Networks, CNN) [5]. Такі моделі демонструють високу ефективність у виявленні характерних візуальних патернів: форми, кольору, текстури, структури листя тощо. Зазвичай вони навчаються на заздалегідь підготовлених наборах зображень, які репрезентують різноманітні види рослин, симптоми хвороб, стадії розвитку та інші специфічні характеристики. Навчання таких моделей потребує значних обчислювальних ресурсів, однак після навчання вони можуть бути оптимізовані для роботи навіть на мобільних пристроях[6].

У практичному втіленні ідеї ідентифікації рослин за зображеннями було створено ряд відкритих та комерційних систем. Серед них особливу популярність отримали такі мобільні застосунки, як PlantNet, Plant.id, PictureThis, LeafSnap та інші. Наприклад, LeafSnap, розроблений за участю науковців з Університетського коледжу Лондона, здатен ідентифікувати сотні видів дерев та чагарників за фотографією листя. Ці застосунки працюють за схожим принципом: користувач фотографує рослину, зображення надсилається до сервера, де обробляється за допомогою навченої моделі, а потім повертається список імовірних відповідностей із коротким описом кожного варіанта. Усе це супроводжується інтерактивним зворотним зв’язком, що робить застосунок не лише інструментом, а й платформою навчання.

Окремим напрямом розвитку таких систем є виявлення хвороб рослин, що має винятково важливе значення для сільського господарства. Проєкт PlantVillage Nuru, який підтримується Продовольчою і сільськогосподарською організацією ООН (FAO), є прикладом ефективної синергії технологій і соціальних потреб. Його мета – надати фермерам у країнах, що розвиваються, доступний інструмент для діагностики захворювань культур (зокрема кукурудзи, маніоки, картоплі) у польових умовах без залучення лабораторій. Застосунок на смартфоні порівнює отримане зображення із навченою базою та визначає ймовірність наявності тієї чи іншої хвороби. Такі рішення часто супроводжуються рекомендаціями щодо подальших дій: обробка, ізоляція, консультація.

Разом з тим, попри очевидні переваги, розробники подібних систем стикаються з рядом суттєвих викликів. По-перше, це обмеженість локалізованих даних: для ефективного функціонування моделі необхідно мати якісну вибірку зображень,

відповідну до регіону користувача. Наприклад, для України бракує наборів рослин у національних умовах та описів українською мовою. По-друге, на результати можуть негативно впливати такі фактори, як освітлення, якість камери, фон зображення, наявність кількох об'єктів у кадрі [7]. По-третє, для мобільних додатків важливо забезпечити високу швидкодію та стабільність навіть на пристроях середнього класу, що обмежує розмір і складність моделей [8].

У результаті можна стверджувати, що візуальна ідентифікація об'єктів рослинного походження – це складний міждисциплінарний напрям, який об'єднує досягнення штучного інтелекту, мобільного програмування, агрономії та інформаційних технологій. Розробка адаптивної, гнучкої та доступної системи, яка враховує локальні потреби користувача (зокрема україномовного), є не лише технічно цікавою задачею, а й має велику практичну цінність для розвитку цифрового агросектору та популяризації ботанічних знань.

1.2 Порівняльний аналіз аналогів

На сучасному ринку програмного забезпечення представлено велику кількість мобільних і вебзастосунків, які реалізують функціональність автоматизованої ідентифікації рослин, виявлення ознак хвороб, а також надання супутньої інформації про біологічні об’єкти. Ці рішення активно розвиваються як у форматі комерційних продуктів, так і у вигляді відкритих дослідницьких ініціатив. Їхня популярність зумовлена зростаючим попитом на інтелектуальні системи аналізу середовища, особливо в контексті цифровізації сільського господарства, розвитку біоінформатики та поширення мобільних пристроїв із достатньою обчислювальною потужністю[9].

У межах цього підрозділу буде розглянуто п’ять найбільш поширених та впізнаваних застосунків, які пропонують користувачам функції ідентифікації рослин або діагностики їхніх захворювань на основі аналізу фотографій. До вибірки включено системи, які на момент написання дослідження демонструють стабільну роботу, активну підтримку з боку розробників, наявність великої користувацької бази, а також широкий функціонал

PlantNet – безкоштовний мобільний застосунок, що дозволяє розпізнавати рослини за фотографіями (рис. 1.1). Основна функція – визначення виду рослини на основі зображення листя, квітки або стебла. PlantNet є науковим проєктом, який базується на принципах колективної участі – користувачі можуть надсилати власні фото та підтверджувати результати інших користувачів [11]. Його база даних постійно оновлюється. Застосунок підтримує кілька мов, але українська відсутня. Визначення здійснюється за допомогою власних моделей, точність яких залежить від якості фото. API є, проте обмежене.

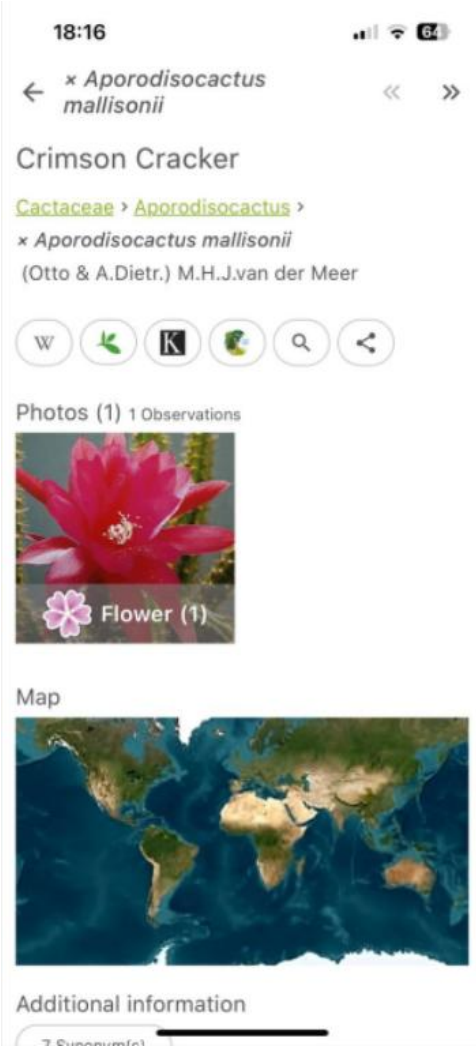

Рисунок 1.1 – Інтефейс PlantNet

LeafSnap – мобільний застосунок, що спеціалізується на визначенні дерев за зображенням листя (рис. 1.2). Розроблений у співпраці з Університетом Коледж Лондона та Університетом Меріленду. LeafSnap забезпечує не лише ідентифікацію,

а й доступ до енциклопедичних відомостей про вид рослини [12]. Його основна перевага – наукова точність, однак набір розпізнаваних видів обмежений, зосереджений здебільшого на флорі Північної Америки та Європи. Українська мова не підтримується, API для публічного доступу не надається.

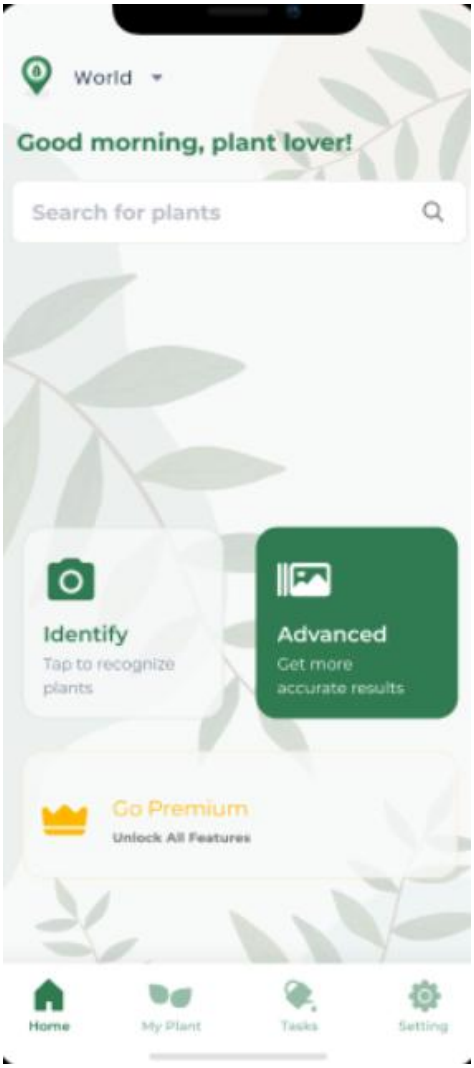

Рисунок 1.2 – Інерфейс LeafSnap

PictureThis – комерційний застосунок, що працює на основі глибокого навчання. Дозволяє не тільки розпізнавати рослини, а й виявляти ознаки хвороб, а також надає поради щодо догляду [13]. Однією з особливостей є інтерфейс із високим рівнем юзабіліті та платна підписка, яка відкриває додаткові можливості, такі як

консультація з ботаніком (рис. 1.3). PictureThis підтримує кілька мов, але українська наразі відсутня. API закрите. Застосунок забезпечує високу точність розпізнавання, однак не дає можливості гнучкого використання в академічних цілях.

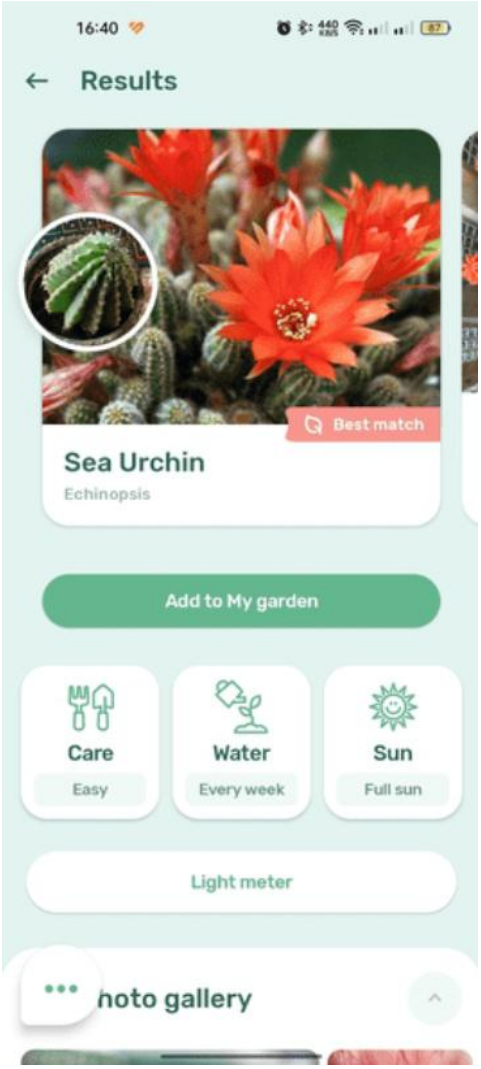

Рисунок 1.3 – Інерфейс PictureThis

PlantVillage Nuru – мобільний застосунок, створений у межах міжнародного проєкту, підтримуваного FAO (рис. 1.4). Призначений для виявлення хвороб сільськогосподарських культур, таких як кукурудза, картопля, томати, маніока тощо [14]. Його особливістю є здатність працювати навіть без доступу до Інтернету – моделі завантажуються локально на пристрій. Платформа орієнтована на фермерів із країн, що розвиваються, зокрема африканських. Українська мова не підтримується. API відсутнє, але застосунок є безкоштовним.

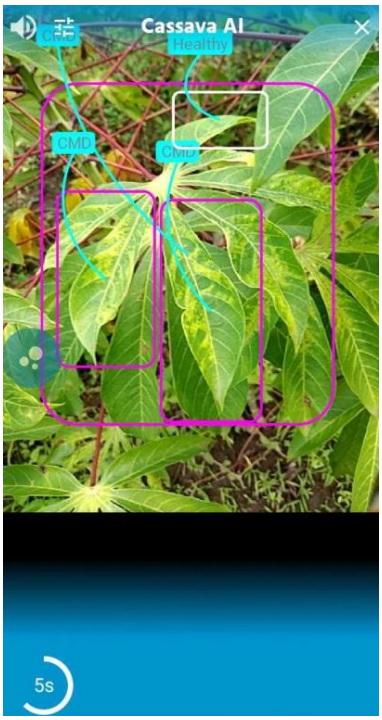

Рисунок 1.4 – Інерфейс PlantVillage Nuru

Flora Incognita – застосунок для автоматичної ідентифікації європейських рослин [15]. Розроблений у Німеччині як частина наукового проєкту (рис. 1.5).

Рисунок 1.5 – Інерфейс Flora Incognita

Має простий інтерфейс і високу точність, підтримує функцію завантаження кількох зображень для покращення результатів.

Працює лише з флорою Центральної та Західної Європи. Наявне часткове перекладення іншими мовами, але українська відсутня. API – закрите. Призначений переважно для науковців, освітніх закладів і екологічних ініціатив. Для порівняння можливостей розглянутих анілогів, було сформновано таблицю 1.1.

Таблиця 1.1 – Порівняння функціональних можливостей аналогів системи

Критерії	**PlantNet**	**LeafSnap**	**PictureThis**	**PlantVillage Nuru**	**Flora Incognita**	**Власна розробка**
Підтримка ідентифікації рослин	+	+	+	-	+	+
Підтримка виявлення хвороб	-	-	+	+	-	+
Наявність української мови	-	-	-	-	-	+
Персоналізовані поради	-	-	-	-	-	+
Загальна оцінка	25%	25%	50%	25%	25%	100%

У результаті аналізу можна зробити висновок, що жоден із розглянутих застосунків не забезпечує повноцінної підтримки української мови, відкритого API, а також одночасної ідентифікації виду рослини й можливих симптомів захворювань. Саме тому створення системи з адаптацією під українського користувача, відкритою архітектурою та інтуїтивно зрозумілим інтерфейсом є актуальним і доцільним.

1.3 Аналіз методів розв’язання задачі

Задача розробки системи для візуальної детекції об’єктів рослинного походження може бути вирішена за допомогою декількох підходів, кожен з яких має свої переваги та обмеження залежно від цілей, ресурсів та умов використання.

Один із поширених методів – побудова централізованої системи, в якій усі компоненти (аналіз зображення, вивід результатів, обробка даних) працюють у межах єдиної логіки. Така модель зручна на початкових етапах, оскільки дозволяє швидко реалізувати базову функціональність, не витрачаючи ресурси на складну інтеграцію [16]. Водночас вона обмежена в масштабуванні: із зростанням кількості функцій або користувачів система стає важкою в обслуговуванні, зміни в одній частині можуть впливати на інші. У таких проєктах навіть оновлення окремого блоку (наприклад, логіки аналізу) потребує втручання в загальну архітектуру, що ускладнює підтримку.

Альтернативою є поділ системи на окремі логічні компоненти, кожен з яких відповідає за свою функцію. Наприклад, модуль аналізу зображення, модуль отримання текстової інформації, модуль обробки результатів та модуль взаємодії з користувачем. Такий підхід дозволяє працювати над кожним блоком незалежно, змінювати або оновлювати його без ризику для загальної системи [17]. Крім того, це забезпечує більшу гнучкість у разі потреби інтеграції нових сервісів або функціоналу. Однак цей варіант потребує чіткого налагодження взаємозв’язків між модулями, визначення форматів обміну даними та послідовності їх обробки.

Ще один метод – делегування частини функціоналу зовнішнім сервісам. У цьому випадку система не виконує розпізнавання самостійно, а передає зображення на аналіз до спеціалізованих платформ. Це дозволяє заощадити час на реалізацію складних алгоритмів, скористатися потужністю вже готових моделей і забезпечити точність результатів. Такий варіант особливо актуальний у навчальних та прототипових проєктах. Водночас є й обмеження: залежність від зовнішніх сервісів, необхідність стабільного інтернет-з’єднання, а також контроль за передачею даних (особливо у випадках із конфіденційною або приватною інформацією).

Додатково важливим аспектом є спосіб взаємодії користувача із системою. Для таких застосунків особливо важливо забезпечити простоту, інтуїтивність та мінімум дій зі сторони користувача – бажано, щоб увесь процес відбувався за декілька кроків: зробити фото, надіслати, переглянути результат. Реалізація зручного та зрозумілого інтерфейсу, який працює швидко та не перевантажує інформацією,

є необхідною умовою ефективного використання такого програмного продукту [18].

Ураховуючи всі розглянуті методи, найбільш доцільним у межах дослідження є варіант з використанням зовнішніх сервісів розпізнавання та побудовою системи з модульною структурою. Це дозволяє зосередитись на головному – побудові логіки взаємодії, інтерфейсу та відображення результатів, з можливістю подальшого розширення та вдосконалення системи.

1.4 Постановка задач дослідження

Провівши аналіз методів розв'язання задачі, дослідивши стан технологій візуальної ідентифікації рослин та аналогічні системи, були сформульовані основні задачі дослідження для реалізації програмного застосунку:

- розробити архітектуру системи для візуальної ідентифікації об'єктів рослинного походження;
- реалізувати модуль взаємодії із застосунком;
- реалізувати модулі взаємодії із API;
- розробити користувацький інтерфейс для відображення результатів;
- провести тестування працездатності системи та якості результатів.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У даному розділі було проведено огляд сучасного стану технологій, пов'язаних із візуальною ідентифікацією рослин, а також розглянуто найбільш відомі аналоги існуючих систем. Порівняльний аналіз показав, що жоден із доступних застосунків не забезпечує одночасної підтримки ідентифікації рослин, виявлення хвороб, локалізації результатів українською мовою та відкритої архітектури для розширення функціональності.

Аналіз методів вирішення задачі дозволив визначити оптимальний підхід для

реалізації системи. Найбільш доцільним є використання зовнішніх сервісів розпізнавання у поєднанні з модульною структурою системи, що забезпечує гнучкість, зручність розробки та можливість масштабування.

У результаті проведеного аналізу було сформульовано конкретні задачі, реалізація яких дозволить створити ефективний інструмент для візуальної ідентифікації об’єктів рослинного походження з інтуїтивно зрозумілим інтерфейсом і підтримкою української мови. Подальші етапи роботи спрямовані на проєктування архітектури, реалізацію модулів системи та їх тестування.

2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ ПРОГРАМНОГО ЗАСТОСУНКУ

2.1 Аналіз вхідних даних системи

У розробленій інформаційній системі, яка орієнтована на ідентифікацію об’єктів рослинного походження та виявлення можливих захворювань, ключову роль відіграють вхідні дані, зокрема візуальна інформація, що подається на аналіз користувачем. Саме вхідні дані формують основу для всіх наступних етапів обробки, розпізнавання, класифікації та інтерпретації результатів. Від їхньої якості, формату, повноти та відповідності технічним вимогам безпосередньо залежить точність і достовірність результатів, які генерує система. Тому належне опрацювання та врахування особливостей цих даних є критично важливим завданням у межах функціонування застосунку.

Центральним видом вхідної інформації в системі виступають зображення рослин – фото, зроблені або завантажені користувачем. Ці зображення подаються на аналіз для виявлення виду рослини або виявлення ознак хвороби. На практиці користувач взаємодіє із застосунком через інтерфейс, де має змогу зробити фото у реальному часі за допомогою камери мобільного пристрою або ж вибрати вже наявне зображення з галереї. Таким чином, сам процес формування вхідного зображення відбувається безпосередньо на стороні користувача і залежить від багатьох зовнішніх факторів, зокрема освітлення, якості камери, фокусування, фону, кута зйомки та інших параметрів, що можуть впливати на результат.

У зв’язку з різноманітністю можливих умов зйомки та варіативністю обладнання користувачів, система повинна бути здатна працювати з широким спектром форматів графічних файлів та підтримувати різні роздільні здатності. Також важливо враховувати, що фотографії можуть містити шум, бути розмитими або частково обрізаними, що вимагає від системи здатності до гнучкої обробки нестандартизованих вхідних даних.

Перед тим як зображення буде передане на зовнішній аналіз, система проводить етап попередньої обробки. Цей етап є обов'язковим і передбачає перетворення отриманого зображення у внутрішній формат, який придатний для подальшої технічної обробки. З метою забезпечення уніфікації та зручності передачі зображень через мережу (наприклад, у тілі HTTP-запиту) зображення конвертується у формат base64. Це текстове подання зображення дозволяє уникнути проблем із кодуванням, зберігає усі піксельні дані без втрат та спрощує передачу до зовнішніх сервісів, які реалізують алгоритми аналізу зображень.

Після завершення попередньої обробки, base64-кодоване зображення надсилається до відповідного стороннього API, що реалізує модель глибокого навчання. У відповідь система отримує структуровану інформацію, яка може містити один або кілька варіантів ідентифікації об'єкта (тобто рослини чи хвороби) з відповідними коефіцієнтами ймовірності. Ці дані в більшості випадків доповнюються додатковими елементами – такими як латинська та народна назва, короткий опис, джерело інформації, рівень впевненості алгоритму тощо.

Окрім цього, під час розробки було передбачено можливість виникнення виняткових ситуацій, пов'язаних із відсутністю вхідних даних, їх пошкодженням або структурною невідповідністю. Наприклад, у випадку, якщо користувач намагається запустити аналіз без попереднього вибору зображення, система генерує відповідне повідомлення з поясненням. Аналогічно, при виникненні помилки на стороні зовнішнього сервісу (наприклад, таймауту, втрати зв'язку або помилки обробки), користувач отримує сповіщення з пропозицією повторити дію. Усе це забезпечує надійність функціонування застосунку та підвищує якість користувацького досвіду.

Загалом, детальний аналіз особливостей вхідних даних демонструє, що система є гнучкою та здатною до адаптації в умовах роботи з неоднорідною, неструктурованою та потенційно неповною інформацією. Це, у свою чергу, створює передумови для надійної і стабільної роботи застосунку в широкому спектрі реальних сценаріїв використання, включно з польовими умовами, де ідеальні умови для зйомки можуть бути недоступними.

2.2 Розробка моделі системи

Для ефективного проєктування та подальшої реалізації програмного забезпечення надзвичайно важливою складовою є побудова моделі, яка відображає логіку взаємодії користувача із системою, а також демонструє основні етапи та процеси, що відбуваються в межах програмного середовища. Така модель слугує не лише інструментом візуалізації, але й дозволяє глибше зрозуміти структуру функціоналу, упорядкувати процеси взаємодії, а також сприяти виявленню потенційних помилок або неузгодженостей ще на етапі розробки. Вона надає змогу дослідити загальну послідовність дій, виявити ключові точки прийняття рішень, визначити залежності між компонентами та сформулювати чітке уявлення про взаємозв'язки, які виникають у процесі використання застосунку [19].

Мобільний застосунок має чітко окреслену функціональну спрямованість і орієнтований на вирішення двох основних задач, що мають практичну значущість у галузі цифрової ботаніки: по-перше, це ідентифікація виду рослини на основі зображення, а по-друге – визначення наявних або потенційних захворювань, що можуть вражати рослину. Обидві ці задачі реалізовані з урахуванням сучасних тенденцій в області застосування штучного інтелекту та машинного навчання, що ґрунтуються на використанні попередньо навчених моделей та зовнішніх API для обробки візуальних даних.

Загальна логіка функціонування системи побудована навколо ключового ресурсу – зображення, що надається користувачем. Весь процес взаємодії починається з того, що користувач, відкривши застосунок, має можливість або зробити нове фото рослини, або обрати наявне зображення з галереї пристрою. Цей крок є початковим, однак критично важливим, адже вся подальша аналітика базується виключно на наданому візуальному матеріалі. Після того як зображення обране, користувач переходить до наступного етапу – вибору однієї з доступних функцій: розпізнавання виду або виявлення хвороби.

Після запуску обраної функції система автоматично виконує низку попередніх перевірок. Зокрема, перевіряється наявність зображення, його валідність, а та-

кож відповідність необхідним параметрам для коректної обробки. Далі проводиться первинна обробка зображення, зокрема його конвертація у формат base64, що є необхідною умовою для передачі даних через API. Отриманий рядок передається до відповідного сервісу – залежно від обраної задачі, це або сервіс для ідентифікації виду, або для визначення захворювань.

У випадку, коли користувач обрав функцію ідентифікації рослини, використовується спеціалізований сервіс, який аналізує передане зображення та повертає список можливих відповідників – тобто назв рослин, які потенційно відповідають зображенню. Отриманий список додатково обробляється застосунком: результати фільтруються, сортуються за рівнем ймовірності та відображаються на екрані у зручному для перегляду вигляді. Кожна рослина виводиться у вигляді картки, що містить назву, зображення та короткий опис, який можна за бажанням розгорнути або приховати.

У разі обрання функції аналізу на наявність хвороб, система передає зображення до іншого сервісу, який спеціалізується на розпізнаванні патологічних змін рослини. У відповідь сервіс надсилає детальну інформацію щодо можливих хвороб, які можуть бути присутніми на рослині. Отримані дані обробляються локально, після чого користувачу демонструється найрелевантніший результат – тобто та хвороба, ймовірність якої є найвищою. Окрім назви, також виводиться опис захворювання, можливі наслідки та рекомендації щодо подальших дій.

У межах інтерфейсу застосунку реалізовано також допоміжні розділи – такі як вікна «Допомога» та «Про програму». Вони не пов’язані безпосередньо з основною функціональною логікою, однак відіграють важливу роль у забезпеченні інформаційної підтримки користувача. У цих розділах можна знайти відповіді на найпоширеніші питання, ознайомитися з принципами роботи застосунку, а також дізнатися основну інформацію про його цілі, авторів та технічні особливості.

Для кращого розуміння логіки функціонування застосунку та взаємодії між його компонентами, доцільно представити відповідну діаграму діяльності. Така діаграма дозволяє не лише узагальнити описані вище процеси, а й подати їх у формалізованому вигляді, що спрощує сприйняття та подальший аналіз (рис. 2.1).

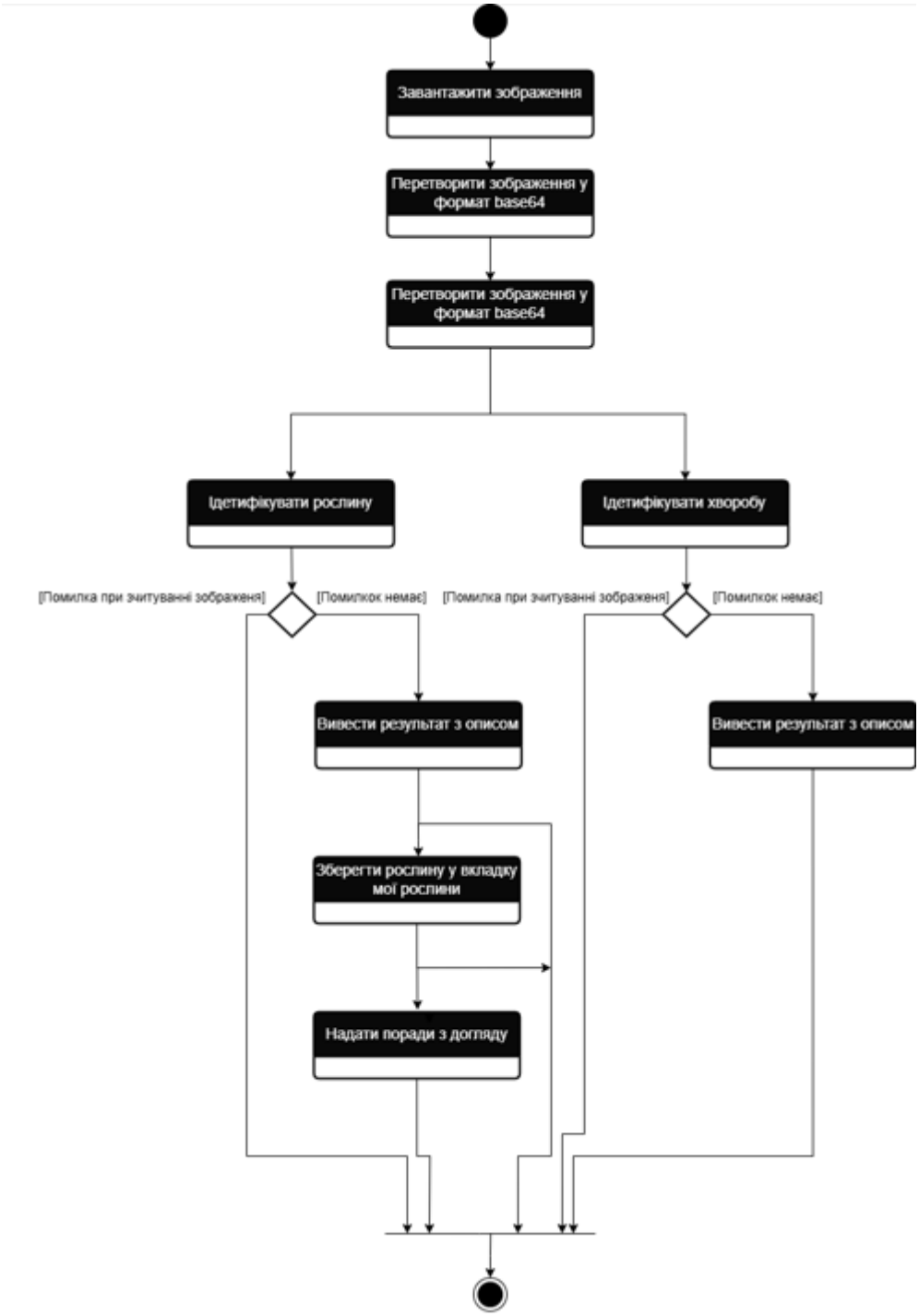

Рисунок 2.1 – Діаграма діяльності системи

Розробка діаграми діяльності була виконана з допомогою draw.io [20].

2.2 Розробка алгоритмів роботи системи

Для розробки програмного застосунку, що реалізує ідентифікацію рослин та виявлення хвороб на основі зображення, було спроєктовано загальний алгоритм,

який описує послідовність взаємодії користувача з інтерфейсом програми, процес обробки зображення та отримання результатів. На основі цього алгоритму реалізовано основну логіку роботи програмного забезпечення.

На початковому етапі взаємодії із застосунком відбувається завантаження графічного інтерфейсу користувача. Застосунок надає користувачеві вітальне вікно, з якого здійснюється перехід до основного функціоналу. Після переходу користувач має змогу обрати фотографію рослини або зробити знімок за допомогою камери. Зображення обробляється в пам’яті пристрою та готується до подальшого аналізу.

Після вибору зображення система надає можливість обрати одну з двох основних функцій: ідентифікація виду рослини або виявлення хвороби. У залежності від обраної дії змінюється логіка подальших обчислень.

У разі запуску ідентифікації виду, зображення автоматично перетворюється у формат base64, після чого відбувається його передача до спеціалізованого API-сервісу. Сервіс повертає список результатів із відповідністю до відомих рослин. Програма виконує сортування результатів за ймовірністю відповідності та перевіряє наявність українського енциклопедичного опису в джерелах (зокрема, Вікіпедії). Якщо такий опис відсутній, застосовується алгоритм автоматичного перекладу з англійської мови. Після цього користувач отримує результат у вигляді картки з назвою, науковим означенням та кнопкою для відображення опису.

У випадку обрання функції виявлення хвороби, програма за аналогічним принципом обробляє зображення, перетворює його у необхідний формат та відправляє запит до відповідного API. У відповідь отримується список можливих захворювань. Система автоматично визначає, чи доступний україномовний опис захворювання, і при необхідності також виконує переклад. У цьому випадку опис виводиться користувачеві автоматично, оскільки інформація про хворобу має бути негайно подана для подальших дій.

Для уявлення про логіку роботи програмного застосунку розроблено відповідний блок-схемний алгоритм (рис. 2.2).

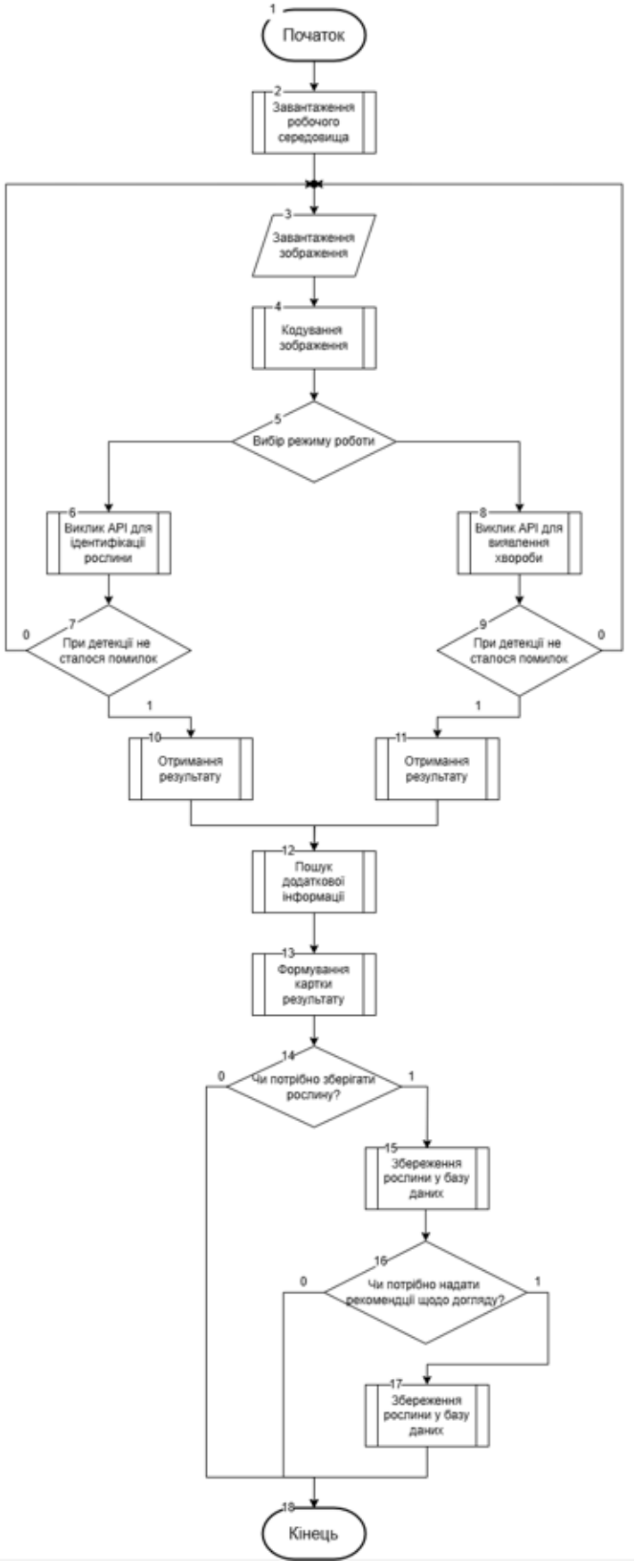

Рисунок 2.2 – Загальний алгоритм роботи програмного застосунку

2.4 Висновки

У другому розділі було здійснено безпосередню розробку програмного застосунку, який виконує ідентифікацію рослин та виявлення їхніх хвороб на основі аналізу зображень. В рамках реалізації було розроблено зручний інтерфейс користувача з використанням фреймворку Flutter, що забезпечує кросплатформенність застосунку та високу швидкодію. Також було побудовано діаграму діяльності, яка відображає загальний алгоритм роботи застосунку, та представлено детальний опис етапів взаємодії користувача із системою.

Крім того, у ході розробки було створено алгоритм основних функцій: застосунку. Це дало змогу структурувати логіку роботи програми та забезпечити коректне функціонування.

3 РОЗРОБКА СИСТЕМИ ВІЗУАЛЬНОЇ ДЕТЕКЦІЇ ОБ'ЄКТІВ РОСЛИННОГО ПОХОДЖЕННЯ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

У процесі створення програмного забезпечення, що реалізує функціональність ідентифікації рослин та виявлення їхніх хвороб на основі аналізу візуальної інформації, було здійснено всебічний аналіз наявних засобів та технологічних рішень, які могли б бути застосовані. Цей етап є важливим для програмного продукту, оскільки від правильності вибору інструментів розробки залежить ефективність, стабільність, масштабованість, а також зручність подальшої підтримки програмного продукту.

Основною метою стало створення мобільного застосунку, який би поєднував у собі сучасний, інтуїтивно зрозумілий інтерфейс. Отже, необхідним стало вибрати технології, які дозволяють розробляти кросплатформні рішення, забезпечуючи при цьому високу продуктивність і зручність розробки.

Під час вибору мови програмування для реалізації мобільного застосунку було розглянуто декілька популярних варіантів, кожен з яких має свої переваги та обмеження. Серед основних кандидатів розглядалися такі мови як Java, Kotlin, Swift, JavaScript (у зв'язці з React Native) та Dart. Усі вони активно використовуються в галузі мобільної розробки, підтримуються великою спільнотою розробників і мають широку базу ресурсів для навчання.

Java – одна з найстаріших і стабільних мов для розробки під Android, широко підтримується інструментами Google [21]. Проте вона вважається дещо «важкою» у плані синтаксису та обсягу шаблонного коду, що ускладнює швидку розробку.

Kotlin – сучасна альтернатива Java, яка офіційно підтримується Google для Android-розробки. Вона забезпечує лаконічніший синтаксис, кращу безпеку типів, а також сумісна з Java, що дозволяє поступово мігрувати старі проєкти[22].

Swift – основна мова для розробки застосунків під iOS, розроблена Apple.

Має сучасний синтаксис, високу продуктивність, але обмежена платформою iOS, тому для створення кросплатформенних рішень не підходить без використання додаткових інструментів (як-от Flutter чи React Native) [23].

JavaScript + React Native – популярний стек для створення кросплатформених застосунків. React Native дозволяє розробляти мобільні інтерфейси на JavaScript, але має певні обмеження у продуктивності й доступі до нативних можливостей платформи [24].

Dart, у поєднанні з Flutter, був обраний як оптимальне рішення. Ця мова була спеціально розроблена Google з урахуванням потреб кросплатформеної розробки. Вона має простий синтаксис, високу продуктивність, асинхронну підтримку та тісну інтеграцію з Flutter SDK [25]. Dart дозволяє досягти максимальної узгодженості логіки застосунку і зовнішнього вигляду незалежно від операційної системи [26]. Для порівняння можливостей розглянутих мов програмування, було сформновано таблицю 3.1.

Таблиця 3.1 – Порівняльна таблиця характеристик мов програмування

Критерій	**Java**	**Kotlin**	**Swift**	**JavaScript (React Native)**	**Dart (Flutter)**
Платформа	Android	Android	iOS	Android / iOS	Android / iOS
Офіційна підтримка	Google	Google	Apple	Meta (спільнота)	Google
Продуктивність	Висока	Висока	Висока	Середня	Висока (майже нативна)
Синтаксис і зручність	Складний, громіздкий	Лаконічний, сучасний	Чистий, строгий	Простий, але з обмеженнями	Простий, сучасний
Швидкість розробки	Низька	Середня	Середня	Висока	Висока (особливо завдяки Hot Reload)
Інтеграція з UI	Через XML	Через Jetpack Compose або XML	Через SwiftUI	Через JSX	Повністю у Flutter, декларативний
Поріг входу для новачка	Високий	Середній	Середній	Низький	Низький
Спільнота і документація	Велика, але стара	Велика, активна	Велика, але лише під iOS	Величезна, з прикладами	Активна, документація від Google

Flutter було обрано як основний фреймворк розробки, оскільки він поєднує в собі можливість створювати високопродуктивні, нативні за відчуттями інтерфейси для Android і iOS, використовуючи одну спільну кодову базу. Це дозволяє істотно зекономити ресурси розробки та уникнути дублювання логіки застосунку. Крім того, Flutter надає широкий набір віджетів, що дозволяють реалізувати будь-які дизайнерські рішення, підтримує гнучке налаштування UI, а також включає інструменти для реалізації анімацій, адаптивного дизайну та роботи з асинхронними запитами.

У ролі основного середовища розробки було обрано Android Studio – офіційно підтримувану компанією Google інтегровану середу розробки, яка є однією з найпоширеніших платформ для мобільної розробки. Android Studio забезпечує розробника усім необхідним: розширеним редактором коду, вбудованим емулятором для тестування застосунків, інструментами для налагодження, профілювання та аналізу продуктивності. Особливо важливо, що Android Studio має повноцінну підтримку розширень (плагінів), зокрема для роботи з Flutter та мовою програмування Dart, що значно спрощує налаштування проєкту та забезпечує безперервність процесу розробки.

Для реалізації функціоналу розпізнавання рослин було обрано сервіс Plant.id API – сучасне програмне рішення, що надає можливість ідентифікувати рослини за фотографією. Даний сервіс відзначається високою точністю, відкритою документацією та простотою інтеграції, оскільки підтримує стандартні REST-запити з передачею зображення у форматі base64. У відповідь користувач отримує список найбільш імовірних варіантів з науковими назвами, описами та рейтингом впевненості моделі.

Для реалізації функціоналу виявлення захворювань рослин було інтегровано Kindwise Crop Health API – спеціалізоване рішення, що дозволяє за наданим зображенням виявити візуальні ознаки різних типових захворювань. API повертає короткі текстові описи симптомів та можливих захворювань, а також дає попередні поради щодо подальших дій. Застосування такого сервісу дозволяє розширити функціональність застосунку до рівня інтелектуального асистента, що здатен

проводити первинну діагностику стану рослини.

Додатково було інтегровано Wikipedia API, який надає можливість автоматично отримувати енциклопедичну інформацію про виявлену рослину або її хворобу. Якщо опис українською мовою недоступний, реалізовано механізм автоматичного перекладу з англомовної версії, що значно підвищує зручність користування додатком для україномовної аудиторії.

Для організації роботи над проєктом, контролю змін у коді, а також спільної роботи було використано систему керування версіями Git у поєднанні з хостингом репозиторіїв GitHub. Це дозволило вести систематичну історію змін, мати доступ до попередніх версій проєкту, організувати резервне копіювання та співпрацю над кодом у випадку розширення команди.

Синергія всіх вищезгаданих інструментів і технологій дозволила реалізувати функціонально багатий, технічно збалансований та зручний для користувача мобільний застосунок. Завдяки використанню Flutter та Android Studio вдалося досягти значної швидкості розробки, зменшити технічні витрати та забезпечити гнучкість проєкту на майбутнє. Інтеграція із зовнішніми API значно підвищила інформативність результатів та надала змогу автоматизувати завдання, які раніше вимагали фахових знань.

Таким чином, вибір саме цих технологій є повністю обґрунтованим, оскільки вони відповідають не лише технічним вимогам поставленого завдання, але й дозволяють розробити застосунок, який є актуальним, сучасним, масштабованим і здатним забезпечити комфортну роботу кінцевому користувачеві у реальних умовах експлуатації.

3.2 Розробка інтерфейсу програмного додатку

Інтерфейс користувача – це ключовий елемент будь-якого сучасного програмного забезпечення, що забезпечує ефективну взаємодію людини з технічною системою. Саме від логіки побудови інтерфейсу, його зручності, доступності та інтуїтивної зрозумілості значною мірою залежить те, наскільки корисним та зручним буде програмний продукт для кінцевого користувача.

У рамках розробки мобільного застосунку «Plant Health» було поставлено завдання створити простий, зручний, візуально привабливий інтерфейс, що забезпечує доступ до основного функціоналу – ідентифікації рослин і виявлення хвороб за їх зображенням. Структура інтерфейсу була спроєктована таким чином, щоб користувач з перших хвилин роботи інтуїтивно розумів, як користуватися додатком, та міг швидко отримати результат. При відкритті програмного додатку, користувача зустрічає екран із назвою системи (рис. 3.1).

Рисунок 3.1 – Вітальний екран застосунку

На цьому рисунку зображено головне вікно, яке бачить користувач після запуску додатку. У центрі розміщено великий зелений логотип у вигляді листка, що символізує екологічність та біологічну спрямованість програми. Нижче розташовано заголовок із привітальним повідомленням та коротким описом можливостей застосунку. Центральним елементом цього екрана є кнопка «Почати роботу», яка ініціює перехід до функціональної частини додатку. Вітальний екран виконує важливу роль першого враження – він задає загальний стиль, формує довіру та дозволяє поступово ввести користувача у контекст роботи з системою.

Рисунок 3.2 демонструє інтерфейс головного функціонального вікна застосунку, який відкривається після старту. На ньому зображено три основні кнопки, розташовані вертикально одна під одною: «Вибрати зображення», «Ідентифікувати рослину», «Ідентифікувати хворобу».

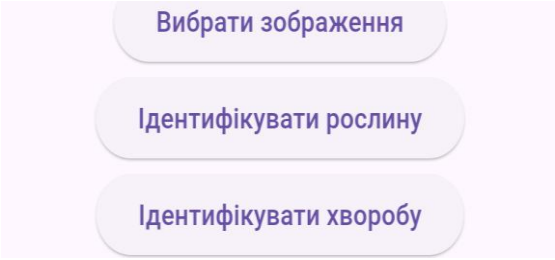

Рисунок 3.2 – Головний екран з кнопками вибору дії

У верхній частині видно заголовок розділу та іконки швидкого доступу до довідки та інформації про програму. У центральній області після вибору користувачем зображення відображається відповідне фото. Така структура забезпечує просту та зрозумілу логіку дій: спочатку вибір фото, потім запуск аналізу. Кожна кнопка виконана у контрастному кольорі, що забезпечує зорову привабливість та видимість навіть на пристроях з невеликим екраном.

На рисунку 3.3 показано, як виглядає екран після виконання операції ідентифікації рослини. Зображення, надане користувачем, проаналізовано, і результат виведено у вигляді картки.

Рисунок 3.3 – Відображення результатів ідентифікації рослини

У верхній частині картки виводиться загальна (поширена) назва рослини з піктограмою, що візуально акцентує увагу. Далі наводиться наукова назва латинською мовою. Особливістю є наявність кнопки «Показати опис», яка дозволяє розгорнути або приховати текстовий опис рослини, що витягується з Вікіпедії. Такий підхід дозволяє уникнути перевантаження екрана та надати користувачу можливість гнучко контролювати обсяг показуваної інформації.

Рисунок 3.4 – Відображення результатів виявлення хвороби

Рисунок 3.4 демонструє вигляд інтерфейсу після натискання кнопки «Ідентифікувати хворобу» і завершення аналізу. Результат представлено у вигляді картки з виведеними даними про можливе захворювання рослини. Заголовок картки містить назву хвороби з піктограмою, далі – наукову назву (за наявності) та розгорнутий опис, що подається автоматично. На відміну від ідентифікації рослини, тут відсутня кнопка «Показати опис», оскільки інформація про хворобу є критично важливою і має бути одразу доступною. Рисунок ілюструє підхід до забезпечення інформативності та швидкого доступу до даних у випадку діагностики захворювання

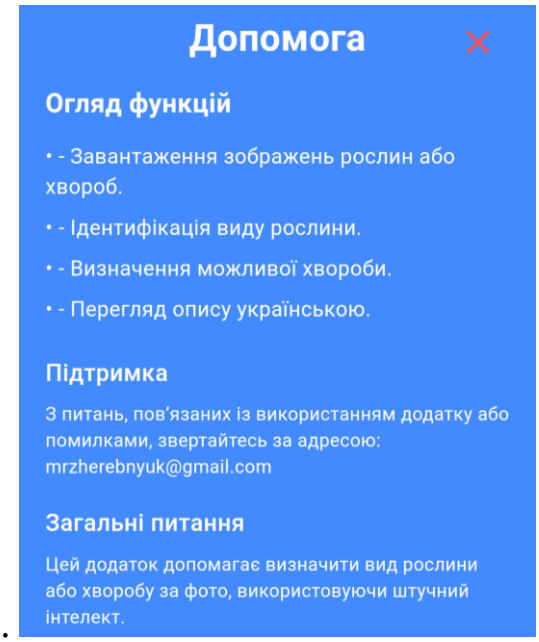

Рисунок 3.5 – Екран «Допомога»

На рисунку 3.5 представлено екран із довідковою інформацією. У верхній частині розміщено заголовок «Допомога» великим шрифтом білого кольору, що чітко контрастує з фоном. Далі розташовано розділи з описами функцій додатку, кожен з яких подається у вигляді маркованого списку. Список містить пункти, що пояснюють, як користуватись додатком: як завантажити фото, як запустити аналіз, де знайти опис тощо. Довідковий блок реалізований у простому та доступному форматі, що дозволяє користувачу самостійно ознайомитися з можливостями програми без додаткових інструкцій. У правому верхньому куті передбачено кнопку для закриття вікна.

Рисунок 3.6 – Екран «Про програму»

На рисунку 3.6 зображено екран із загальною інформацією про застосунок. У центрі знаходиться заголовок «Про програму Plant Health», оформлений великим шрифтом білого кольору. Нижче розміщено опис призначення додатку, імена розробника та групу, до якої він належить, а також дані про ліцензування й версію. Це службовий екран, що виконує інформаційну функцію та підкреслює прозорість розробки. Як і в довідковому вікні, передбачено зручну кнопку виходу у правому верхньому куті. Такий формат подачі дозволяє користувачу дізнатися більше про автора, звернутись у разі потреби або пересвідчитися в надійності джерела.

Розробка графічного інтерфейсу програмного застосунку була здійснена у середовищі розробки Android Studio з використанням фреймворку Flutter та мови програмування Dart, що дозволило створити сучасний, кросплатформний та адаптивний інтерфейс користувача для мобільних пристроїв.

3.3 Розробка головного модуля взаємодії із застосунком

У процесі створення додатку було розроблено модуль, що реалізує основні алгоритми взаємодії користувача із застосунком, керує інтерфейсом, обробляє зображення, викликає зовнішні API, формує запити, обробляє відповіді та керує логікою виводу інформації на екран.

Основною вимогою до реалізації даного модуля було забезпечення зручної, стабільної та інтуїтивно зрозумілої взаємодії між користувачем та системою. Відповідно, реалізація була виконана на базі фреймворку Flutter, що дозволяє створити гнучкий інтерфейс із чіткою логікою переходів між екранами.

При завантаженні застосунку виконується функція runApp(), яка ініціалізує головний клас:

Клас MyApp – це кореневий віджет, який налаштовує тематику, маршрути і початковий екран. Важливо зазначити, що з точки зору архітектури Flutter такий підхід дозволяє гнучко керувати розширенням застосунку у майбутньому – додаючи ноі екрани без втручання в основну логіку.

Клас _PlantIdentifierScreenState (рис 3.7) є станом для екрана ідентифікації

рослин у Flutter додатку. Він належить до станового віджету PlantIdentifierScreen і відповідає за логіку взаємодії користувача з інтерфейсом, обробку зображень, виклики API та відображення результатів.

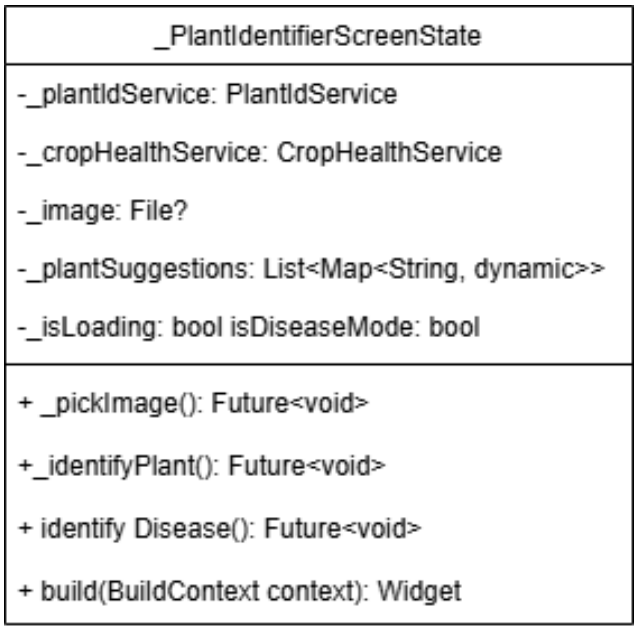

Рисунок 3.7 – Головний клас

Після запуску застосунку користувач переходить до основного екрану розпізнавання. Саме тут реалізується ключова взаємодія із зображенням. Алгоритм побудовано таким чином, щоб зображення завантажувалося максимально просто, без потреби у зовнішньому редагуванні. Це досягається завдяки методу pickImage.

Така реалізація дозволяє не лише завантажити зображення, але й автоматично оновити інтерфейс користувача для подальших дій.

Після завантаження зображення користувач має змогу здійснити ідентифікацію виду рослини. Логіка цього процесу реалізована у вигляді окремого методу. Цей метод формує запит до API сервісу Plant.id, передає зображення у форматі base64, і після отримання результатів здійснює сортування за ймовірністю. Додатково отримується опис з Wikipedia.

У випадку виявлення захворювання на рослині система переходить у режим DiseaseMode, що передбачає іншу логіку. Особливість цього режиму – складніша перевірка даних на наявність опису, а також більша увага до точності результату. Метод identifyDisease() реалізує цей функціонал:

Оскільки захворювання рослин не завжди мають енциклопедичні джерела українською мовою, була реалізована багатоступенева перевірка: чи існує сторінка у Вікіпедії для наукової назви, чи існує для загальної назви, і, за потреби, переклад англійського опису.

У залежності від обраного режиму (визначення виду чи хвороби) користувачу виводиться відповідна інформація у вигляді картки. У випадку рослин – з кнопкою розгортання опису, у випадку хвороб – з автоматичним виводом. Такий підхід дозволяє зменшити час прийняття рішень та зробити взаємодію максимально ефективною.

Таким чином, модуль main.dart виконує роль ядра програмного застосунку. У ньому закладено логіку обробки основних функцій, динамічну зміну інтерфейсу, роботу з мережевими запитами та забезпечення багатомовної підтримки описів.

3.4 Розробка модуля для взаємодії з API

Одним з ключових етапів створення застосунку є розробка модуля, що забезпечує зв’язок із зовнішніми інтелектуальними сервісами для ідентифікації рослин і діагностики їхнього стану. У сучасних умовах обробка зображень у мобільних додатках нерозривно пов’язана з використанням потужних серверних обчислювальних систем, здатних виконувати глибоку аналітику на основі штучного інтелекту. Тому було прийнято рішення реалізувати спеціалізований модуль, який дозволяє надсилати зображення до API сторонніх сервісів та отримувати структуровану відповідь із результатами аналізу.

З технічної точки зору, взаємодія з API була реалізована у вигляді окремих класів, що відповідають принципам інкапсуляції та дозволяють розділити логіку обробки даних і візуальне представлення інформації. Такий підхід суттєво підвищує гнучкість програмного коду, полегшує його розширення і тестування.

Усього було реалізовано два основні класа:

PlantIdService (рис 3.8) – клас, що відповідає за ідентифікацію виду рослини;

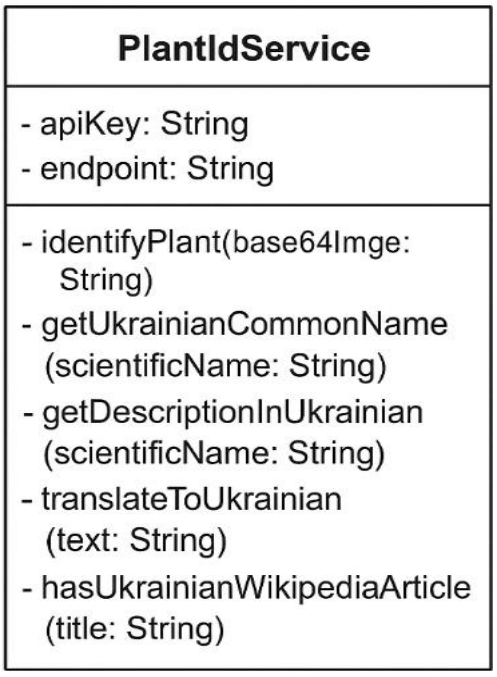

Рисунок 3.8 – клас, що відповідає за ідентифікацію виду рослини

CropHealthService (рис 3.9) – клас, що виконує аналіз стану рослини та виявлення можливих захворювань.

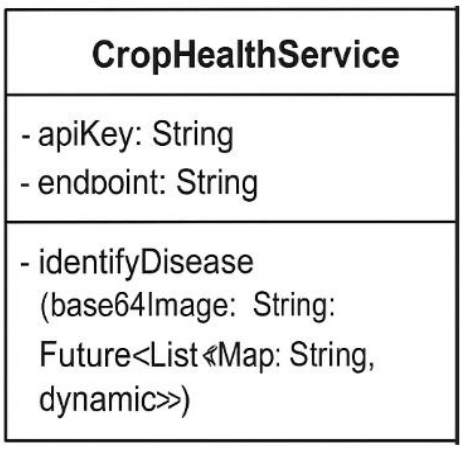

Рисунок 3.9 – клас, що виконує аналіз можливих захворювань

Метою методу identifyPlant() класу PlantIdService є отримання зображення, обробка його у відповідному форматі та передача до API. Цей сервіс дозволяє розпізнати тисячі видів рослин, використовуючи технології комп'ютерного зору та глибокого навчання.

З технічного боку, зображення попередньо перетворюється у формат base64, що забезпечує зручну та універсальну форму для передачі через HTTP-протоколи.

Далі формується POST-запит з відповідними заголовками та вмістом, і відправляється на сервер.

На цьому етапі система очікує відповідь, яка містить список імовірних видів рослин, упорядкованих за рівнем впевненості алгоритму. У разі виявлення – здійснюється спроба локалізації результату: спочатку – пошук української загальноприйнятої назви, далі – перевірка наявності енциклопедичного опису.

Цей фрагмент коду демонструє принцип побудови відповідей на запити користувача. Всі результати перевіряються на достовірність, після чого найбільш релевантний запис передається для відображення в інтерфейсі.

Метод identifyDisease() реалізовано у складі класу CropHealthService. Його завданням є обробка фотографії з ознаками захворювання і подальше отримання діагностичних результатів. Як і у попередньому методі, зображення спочатку кодується у формат base64, а потім надсилається у POST-запиті на API (Рис. 3.13).

Після успішної відповіді сервер повертає список захворювань, кожне з яких містить назву, наукову класифікацію, ймовірність, а також текстовий опис.

Отримані дані систематизуються і передаються у головний інтерфейс. За необхідності, система також може доповнити ці дані перекладеним описом із Wikipedia.

Для підвищення користувацької цінності було реалізовано додаткові методи, що забезпечують отримання описів із Wikipedia, а також переклад із англійської на українську мову. Це забезпечує розширений функціонал без потреби у використанні платних сервісів чи створенні власної бази даних.

Такі методи дозволяють гнучко обробляти ситуації, коли наукова назва не має прямого українського відповідника, або коли інформація виявляється лише англійською мовою.

Таким чином, модуль взаємодії з API є фундаментальним компонентом програмного забезпечення, оскільки саме він забезпечує зв’язок із зовнішніми сервісами, які виконують обробку візуальних даних.

3.5 Розробка бази даних

Під час розробки мобільного застосунку, що виконує важливі функції з ідентифікації рослин та виявлення можливих хвороб на основі зображень, на особливу увагу заслуговує створення надійної системи збереження користувацьких даних. Це пов’язано з тим, що сучасні мобільні рішення повинні не лише виконувати миттєві обчислення або запити, а й забезпечувати збереження отриманої інформації для подальшого використання. Зокрема, у контексті застосунку «Plant Health» надзвичайно важливо, щоби користувач мав змогу зберігати результати ідентифікації рослин, супутні описи, а також відповідні зображення в локальному сховищі. Такий підхід дозволяє значно підвищити зручність використання додатку, адже вся необхідна інформація стає доступною навіть у випадку відсутності підключення до Інтернету.

Іншими словами, реалізація функціоналу офлайн-доступу до збережених даних є однією з критично важливих складових, що визначають загальну ефективність і практичну цінність програмного продукту. Саме тому під час етапу проєктування архітектури застосунку значна частина уваги була зосереджена на виборі оптимального механізму локального збереження даних. Було розглянуто та проаналізовано кілька найбільш популярних бібліотек і підходів, які використовуються у середовищі Flutter для локальної роботи з даними. Кожне з вищезазначених рішень має свої переваги, обмеження та сферу оптимального застосування. Наприклад, Hive і ObjectBox добре підходять для роботи з об’єктно-орієнтованими структурами даних, дозволяючи зберігати об’єкти без необхідності створення складної таблиці. SharedPreferences, своєю чергою, зазвичай використовується для збереження простих налаштувань або невеликих обсягів даних у форматі ключ-значення. Drift пропонує потужну систему генерації запитів, однак у багатьох випадках її функціональність може бути надлишковою.

З огляду на специфіку поставленого завдання, яка передбачає збереження структурованих даних у вигляді таблиць із чітко визначеними полями, виконання SQL-запитів, а також подальшу можливість масштабування та адаптації структури бази, було прийнято обґрунтоване рішення використовувати бібліотеку Sqflite. Ця

бібліотека є офіційним рішенням для роботи з SQLite у Flutter та широко підтримується розробницькою спільнотою, що також є важливим фактором при виборі технології.

Sqflite забезпечує розробнику низькорівневий доступ до бази даних, дозволяє гнучко створювати, редагувати й видаляти таблиці, а також виконувати будь-які SQL-запити, від простих до найскладніших. Завдяки такому підходу можливо реалізувати будь-яку необхідну логіку роботи з даними, забезпечити високий рівень контрольованості та оптимізації з боку розробника. Крім того, важливою перевагою є підтримка централізованого доступу до БД за допомогою допоміжного класу, що реалізує патерн Singleton – класу DatabaseHelper, який інкапсулює всю взаємодію з базою та спрощує повторне використання коду в різних частинах застосунку.

У межах даного застосунку було створено основну таблицю під назвою plants, яка є центральним елементом збереження інформації про всі рослини, додані користувачем до його особистої колекції. Саме ця таблиця зберігає увесь обсяг релевантних даних, що були отримані в результаті виконання функцій розпізнавання або діагностики. На таблиці 3.2 відображено ключові поля, які забезпечують повноцінне відображення інформації:

Таблиця 3.2 – Поля таблиці plants

Назва поля	Тип даних	Опис
id	INTEGER	Унікальний ідентифікатор рослини. Є первинним ключем таблиці.
name	TEXT	Назва рослини, введена або визначена користувачем.
scientific_name	TEXT	Наукова (латинська) назва рослини.
description	TEXT	Короткий опис рослини або хвороби (отриманий з API або вручну).
image_path	TEXT	Шлях до зображення рослини, збереженого у файловій системі.

Для створення таблиці було використано SQL-запит, який виконується під час ініціалізації бази даних.

Така структура забезпечує достатній обсяг для зберігання ключової інформації та, водночас, дозволяє в майбутньому розширювати таблицю за рахунок нових полів без суттєвих змін в архітектурі застосунку.

З метою інкапсуляції логіки доступу до бази даних та зниження зв’язності коду було розроблено окремий клас DatabaseHelper (рис 3.10) , який реалізує патерн Singleton. Це означає, що протягом усього часу виконання програми існує лише один екземпляр класу, який відповідає за взаємодію з базою.

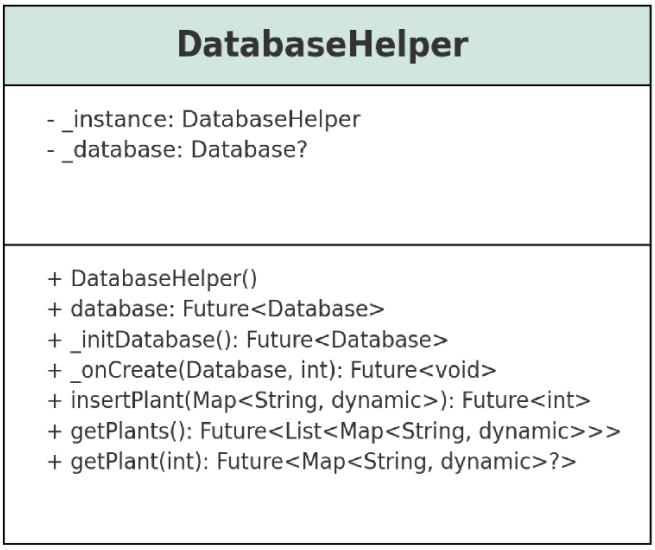

Рисунок 3.10 – клас DatabaseHelper

Функція get database повертає відкриту базу даних або ініціалізує її, якщо вона ще не була створена. У процесі ініціалізації обирається шлях збереження, який відповідає каталогу внутрішніх документів застосунку, де зберігаються всі критично важливі дані:

Далі за допомогою функції openDatabase база відкривається або створюється вперше. Якщо вона створюється з нуля, викликається функція _onCreate, яка ініціалізує таблицю plants.

Для повноцінної взаємодії з базою даних реалізовано низку методів, які дозволяють виконувати типові CRUD-операції (Create, Read, Update, Delete). Зокрема:

1. Додавання нової рослини до бази відбувається за допомогою методу insertPlant, який приймає словник (Map<String, dynamic>) та виконує операцію вставки у відповідну таблицю.

2. Отримання збережених рослин здійснюється методом getPlants, який виконує простий SQL-запит без фільтрації:

3. Пошук рослин за ідентифікатором реалізовано у методі getPlant, що виконує запит з умовою (WHERE):

Ці методи забезпечують мінімально необхідний функціонал для роботи з персональною колекцією рослин користувача. Збереження інформації в локальній базі даних дає низку важливих переваг. По-перше, користувач може отримати доступ до своєї інформації навіть за відсутності підключення до Інтернету, що критично важливо для польових умов. По-друге, збереження зображень локально дозволяє швидше відображати їх у додатку без потреби повторного завантаження. По-третє, структура бази дозволяє в майбутньому реалізувати додаткові функції, зокрема сортування, фільтрацію, позначення улюблених рослин тощо.

Таким чином, розробка та інтеграція бази даних у мобільний застосунок є важливим кроком до підвищення зручності використання, забезпечення автономності функціонування системи та формування персонального архіву користувача. Реалізована архітектура бази є гнучкою, масштабованою та зручною для подальшого розвитку функціоналу застосунку.

3.6 Висновки

У третьому розділі обґрунтовано вибір технологій для розробки програмного забезпечення для ідентифікації рослин і виявлення хвороб. Використано Android Studio та Flutter, що забезпечили кросплатформеність і зручність розробки. API дозволили ефективно ідентифікувати рослини та діагностувати захворювання. Модуль взаємодії з користувачем забезпечив інтуїтивно зрозумілий інтерфейс та стабільну роботу з API. Розробка модулів для обробки зображень і взаємодії з API дозволила створити ефективне і масштабоване програмне забезпечення.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Тестування системи

У процесі створення та впровадження будь-якого програмного забезпечення надзвичайно важливою складовою є етап тестування. Це ключовий етап життєвого циклу програмного продукту, який дозволяє не лише виявити можливі недоліки, помилки або неточності в роботі системи, але й підтвердити відповідність реалізованого функціоналу визначеним вимогам. Тестування виконує роль містка між процесом розробки й подальшою експлуатацією продукту кінцевими користувачами. Саме завдяки тестуванню можна гарантувати, що програмне забезпечення працює стабільно, ефективно, без збоїв і відповідає очікуванням замовника та кінцевих користувачів.

У рамках розробки системи для розпізнавання рослин на основі зображень, тестування відіграє особливо важливу роль. Оскільки програма безпосередньо взаємодіє з візуальним контентом, точність, швидкість і стійкість до помилок стають критично важливими характеристиками. Кожен аспект роботи системи, від моменту завантаження зображення до отримання результату ідентифікації, має бути ретельно перевірений. Важливо також протестувати поведінку системи в умовах нестандартних або стресових ситуацій. Усе це дозволяє сформувати цілісне уявлення про якість, надійність і зручність використання створеного програмного забезпечення.

Перш за все, важливим є тестування завантаження зображень. Завдяки цьому тесту можна перевірити, чи правильно система обробляє зображення різних форматів. Програма повинна підтримувати загальновживані формати зображень, такі як JPEG, PNG, BMP та інші, та коректно обробляти ці файли при їх завантаженні в систему. Однак важливо також перевірити, як система реагує на не підтримувані або пошкоджені файли, адже в такому випадку вона повинна вивести відповідне повідомлення про помилку, не допускаючи збоїв або аварійних ситуацій. На рисунку 4.1 зображено результат завантаження рисунку формат який не підтримується.

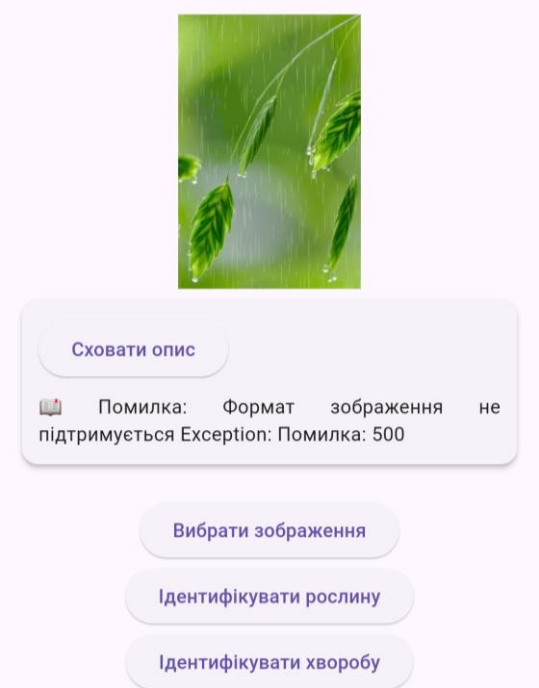

Рисунок 4.1 – Спроба заватажити GIF файл

Крім того, одним із найважливіших та пріоритетних етапів загального процесу тестування програмного забезпечення є перевірка точності ідентифікації рослин, що виконується системою. Цей аспект тестування відіграє ключову роль, оскільки саме від здатності застосунку коректно розпізнавати рослини залежить не лише функціональна успішність, а й загальний рівень довіри з боку кінцевих користувачів до запропонованого рішення.

У межах вказаного тестування перевіряється, наскільки точно і стабільно система здатна розпізнавати різноманітні види рослин, зображених на фото, що мають різну якість, роздільну здатність, рівень освітлення та чіткість. Особлива увага приділяється випадкам, коли користувач надає зображення високої якості, зроблені в хороших умовах – на них застосунок повинен безпомилково визначати вид рослини, з високим рівнем точності та швидкістю обробки.

Однак не менш важливою є ситуація, коли фотографії мають низьку якість, недостатнє освітлення, нечіткий фокус або сторонні об’єкти на задньому плані. Подібні умови відповідають реальним сценаріям використання, коли користувачі не завжди мають змогу зробити ідеальне фото – наприклад, у польових умовах, при недостатньому освітленні або за наявності пошкоджень камери.

Саме тому тестування в умовах «поганої» вхідної інформації дозволяє виявити ступінь стійкості застосунку до складних обставин та перевірити, чи здатен він адаптуватися до неідеальних даних. Навіть у випадках, коли якість зображення не відповідає бажаним критеріям, програма повинна робити спроби ідентифікації, а не припиняти обробку, і, за можливості, надати користувачеві хоча б орієнтовну інформацію про рослину.

Цей тип тестування є надзвичайно важливим з практичної точки зору, адже дозволяє переконатися, що застосунок є придатним до використання в широкому спектрі реальних умов, а не лише в ідеалізованих лабораторних сценаріях. Таким чином, забезпечується більша надійність роботи застосунку, підвищується рівень користувацької задоволеності та зменшується кількість помилок у процесі реальної експлуатації системи. Рисунок 4.2 демонструє можливості ідентифікації рисунку поганої якості

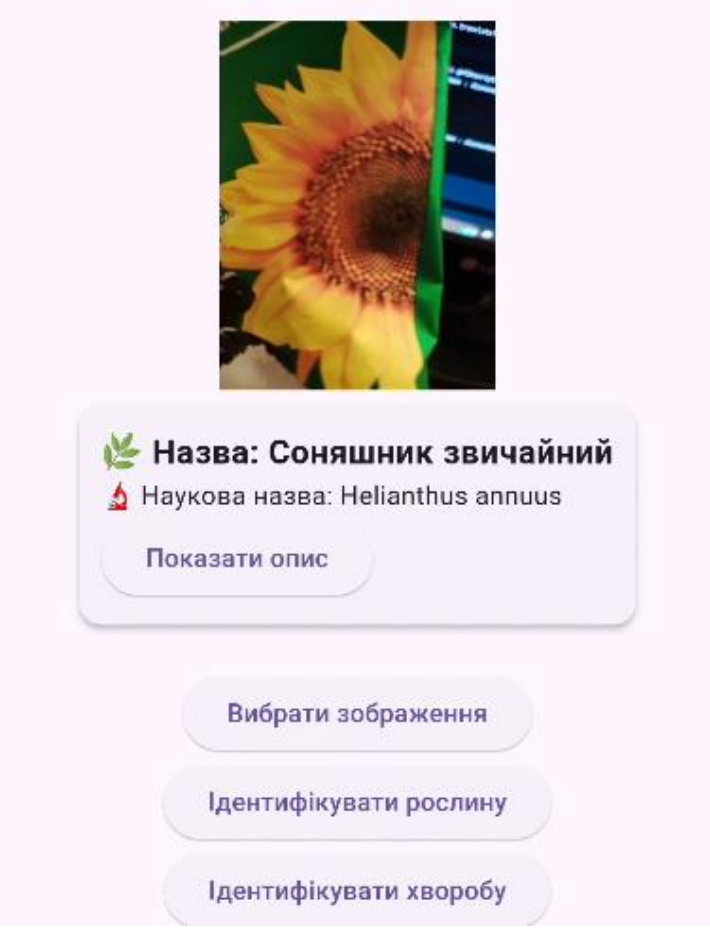

Рисунок 4.2 – Ідентифікація по фото поганої якості

Тестування швидкості обробки дозволяє визначити, наскільки стабільно та швидко працює застосунок у різних умовах, а також виявити можливі «вузькі місця» або затримки, які можуть виникати при роботі з окремими категоріями зображень. Зокрема, важливо здійснити вимірювання часу обробки для зображень, що мають різну роздільну здатність, розмір файлу, ступінь деталізації, а також походження. Це дозволяє сформувати повноцінну картину продуктивності та забезпечити адаптивність алгоритмів під реальні сценарії використання.

У ході проведення тестування слід зафіксувати час, який витрачається системою на обробку зображення на кожному з етапів: завантаження або вибір зображення, його внутрішня обробка (конвертація, перевірка, оптимізація), формування запиту до API, очікування відповіді та, власне, виведення результату користувачу. Таке розділення дозволяє детально проаналізувати, на якому саме етапі виникає основне навантаження або затримка. На рисунку 4.3 зображено швидкість ідентифікації неякісного фото а на рисунку 4.4 якісного.

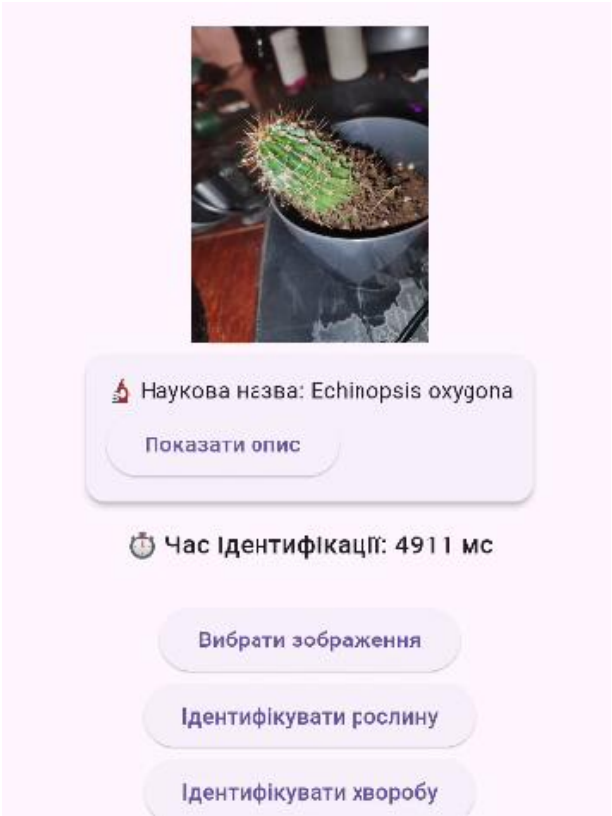

Рисунок 4.3 – Швидкість ідентифікації неякісного фото

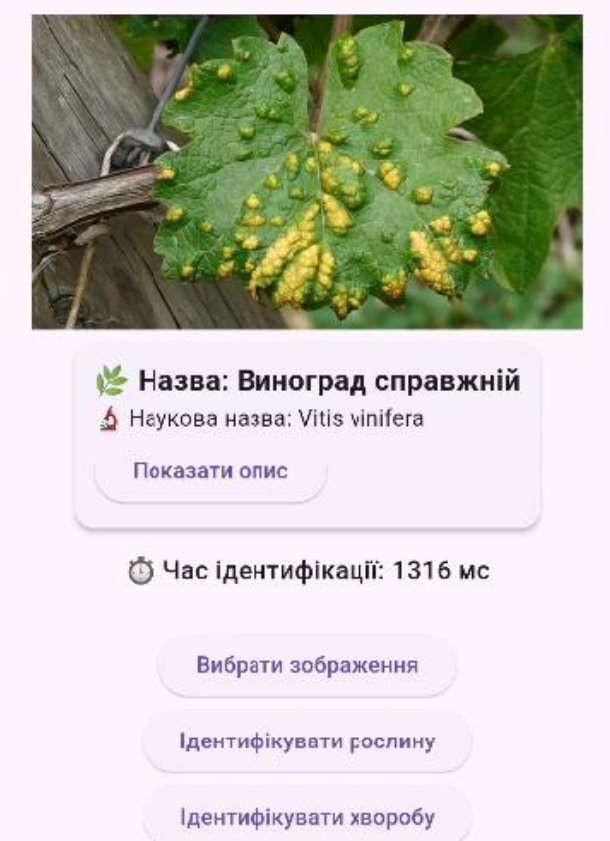

Рисунок 4.4 – Швидкість ідентифікації якісного фото

Одним із важливих етапів перевірки функціональності програмного застосунку є тестування обробки виняткових ситуацій, а саме – перевірка коректності виведення повідомлень про помилки. Цей аспект має принципове значення, оскільки забезпечення зрозумілого та інформативного зворотного зв’язку є запорукою позитивного користувацького досвіду. У процесі взаємодії із застосунком користувач може зіткнутися з різноманітними проблемами – як технічного, так і логічного характеру. Одним з найтиповіших прикладів є ситуація, коли завантажене зображення не відповідає очікуваним критеріям, зокрема не містить чіткого зображення рослини або взагалі не є зображенням об’єкта рослинного походження..

Важливо, щоб у кожному подібному випадку система забезпечувала не лише діагностику помилки, а й передбачала гнучкі сценарії подальших дій: наприклад, пропонувала повторно обрати зображення, надала доступ до прикладів коректних фотографій або навіть автоматично очищала попереднє зображення для

зменшення плутанини. Таким чином, реалізація грамотної обробки помилок у поєднанні з чіткими інструкціями користувачу є не лише технічним, а й UX-рішенням, яке істотно підвищує зручність та ефективність роботи з додатком у реальних умовах експлуатації. Рисунок 4.5 демонструє реакцію застосунку на фото без рослин.

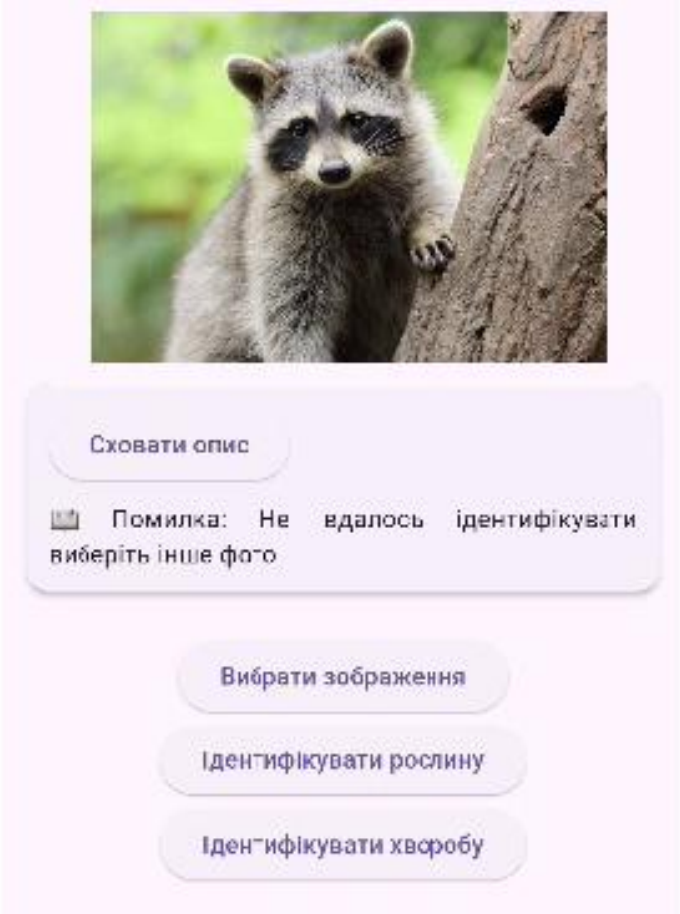

Рисунок 4.5 – Помилка при завантаженні фото без рослини

У процесі тестування особливу увагу необхідно приділити перевірці правильності виведення результатів після ідентифікації рослини. Саме цей етап безпосередньо впливає на користувацький досвід, адже після завантаження зображення користувач очікує не лише швидкої обробки, а й точного та зрозумілого результату. Тому важливо перевірити, чи відповідає виведена інформація реальним даним, які повертає система розпізнавання, та чи подається вона у зручному для користувача форматі. На рисунку 4.6 зображено приклад ідентифікації по фото.

Рисунок 4.6 – Ідентифікація рослини

Під час цього тестування модель має розпізнати рослину та передати назву, можливі альтернативні назви, а також додаткову інформацію – наприклад, короткий опис, ботанічні характеристики, умови росту, наявність токсичних властивостей або використання у побуті чи медицині. Надзвичайно важливо перевірити, чи вся ця інформація відображається коректно: чи відповідають назви зображеній рослині, чи не допускаються дублювання, помилки в тексті або відсутність частини даних, які є критично важливими для користувача. На рисунку 4.7 продемонстровано перевірку коректності результатів ідентифікації.

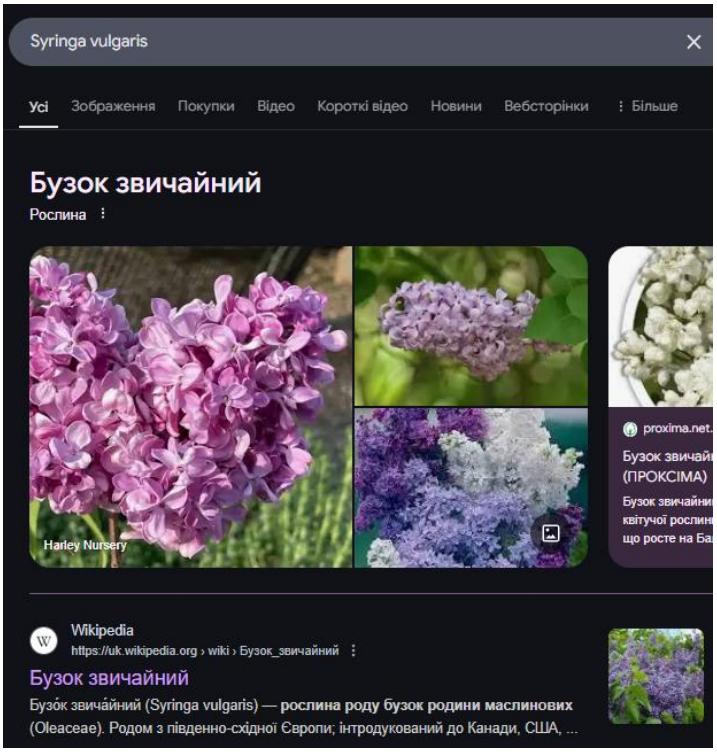

Рисунок 4.7 – Перевірка коректності результатів

Усі ці тести дозволяють забезпечити високу якість програмного продукту, що в свою чергу дозволить користувачам з максимальною точністю та ефективністю здійснювати ідентифікацію рослин. Застосування тестів на різних етапах розробки допомагає виявити та виправити потенційні помилки, забезпечуючи стабільну роботу програми навіть при великих навантаженнях або нестандартних умовах використання.

Крім вищенаведених тестів, окрему категорію становить тестування збереження ідентифікованих рослин у локальну базу даних, адже це одна з ключових функціональних можливостей мобільного застосунку. Після завершення процесу розпізнавання рослини користувач має змогу зберегти результат у вкладку "Мої рослини", що дозволяє формувати персональну колекцію. У цьому контексті надзвичайно важливо забезпечити не лише правильність збереження інформації, але й її цілісність, стабільність доступу та можливість коректного відображення в майбутньому.

Під час тестування перевірялися такі аспекти:

• чи зберігається повний набір даних про рослину (назва, наукова назва, опис, фото);

• чи не виникає дублювання записів;

• чи не відбувається втрата інформації при повторному відкритті додатку;

• чи коректно працює механізм вибірки інформації з бази даних.

На рисунку 4.7 зображено перевірку сторінки «Мої рослини».

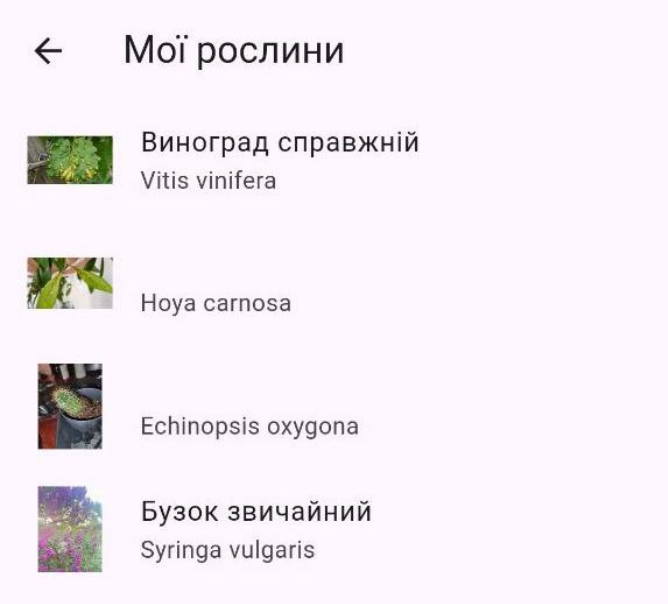

Рисунок 4.8 – Збереження результату ідентифікації у «Мої рослини»

Було протестовано кілька типових сценаріїв використання, включно із збереженням одночасно декількох рослин, переглядом раніше збережених даних, очищенням бази та її повторним наповненням. Особливу увагу було приділено тестуванню стабільності бази даних після нештатного завершення роботи застосунку або раптового перезапуску пристрою. У всіх перевірених випадках інформація залишалася цілісною, а механізм збереження – працездатним. Крім того, перевірялась правильність відображення зображень, шлях до яких зберігається у вигляді локального посилання, що є важливою частиною функціональності. На рисунку 4.9 зображено приклад збереження ідентифікованої рослини.

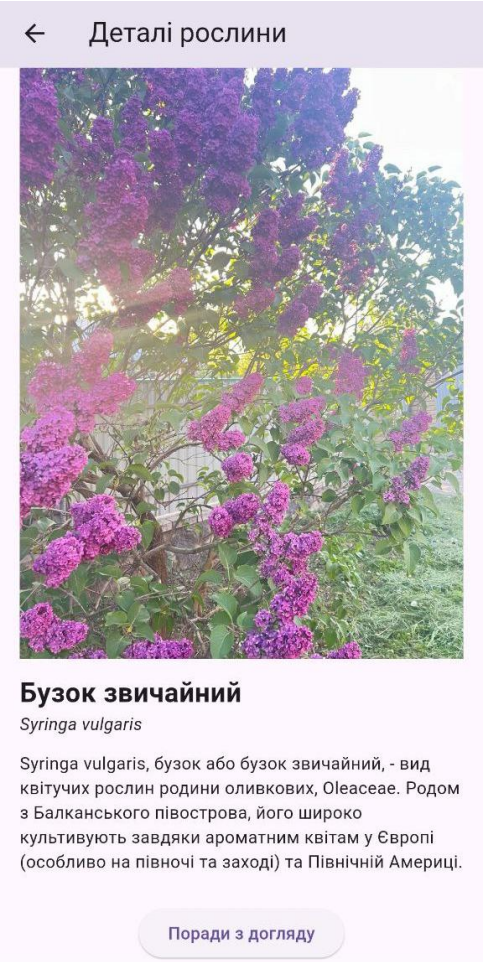

Рисунок 4.9 – Перегляд збереженої рослини з локальної бази

Ще одним важливим напрямом тестування стала перевірка функції генерації рекомендацій по догляду за рослинами. Вона базується на інтеграції з відповідним API, який на основі отриманого виду рослини формує індивідуальні поради для користувача. Ця функція є не лише інформаційною, але й практичною – вона допомагає користувачу приймати обґрунтовані рішення щодо подальших дій: наприклад, як поливати рослину, які добрива використовувати, які температурні умови їй підходять тощо.

У процесі тестування даної функції були перевірені такі аспекти:

- швидкість відповіді API при генерації порад;
- відповідність змісту рекомендацій обраному виду рослини;
- стабільність роботи при відсутності Інтернет-з’єднання (тобто наявність повідомлення про неможливість з’єднання з сервером);

• реакція додатку на некоректну або неповну відповідь API (наприклад, коли сервіс тимчасово недоступний або повертає помилку).

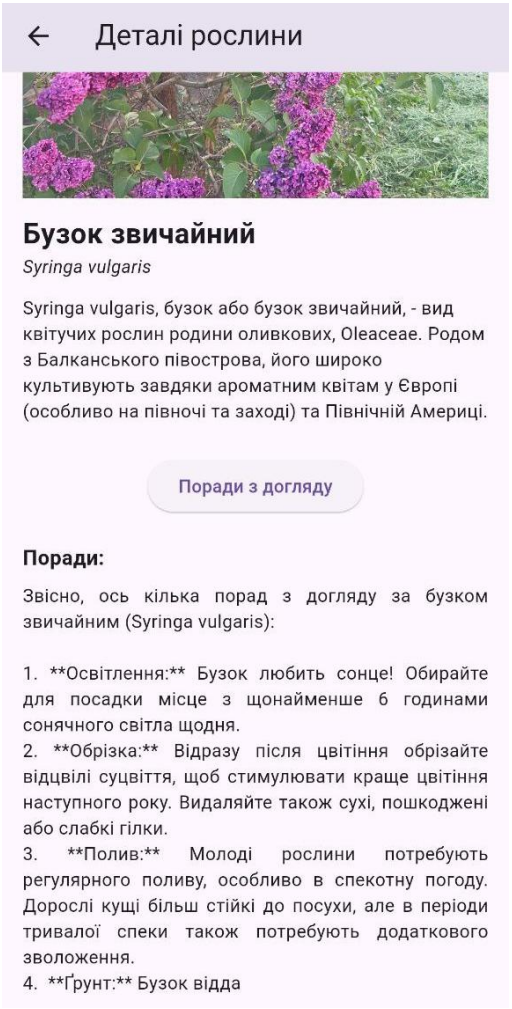

Рисунок 4.10 – Генерація порад для збереженої рослини

Також тестувалась можливість асоціювання рекомендацій із відповідною рослиною у локальній базі. Успішно згенеровані поради можна переглянути на детальному екрані кожної збереженої рослини (рис 4.10), що підвищує зручність користування та сприяє створенню індивідуального “доглядового щоденника”.

У результаті тестування було встановлено, що обидві функції – як збереження рослин, так і отримання рекомендацій по догляду – функціонують коректно, стабільно й без критичних помилок. Це дозволяє забезпечити повноцінний цикл взаємодії користувача з системою: від моменту розпізнавання до формування бази знань, яка доступна у будь-який час і не потребує повторного запиту до серверу.

4.2 Розробка інструкції користувача

Інструкція користувача розроблена з урахуванням особливостей запуску та взаємодії з мобільним застосунком, що працює на платформах Android та iOS. Вона містить відомості про мінімальні та рекомендовані вимоги до пристрою, а також описує основні етапи користування програмою для ідентифікації рослин. У таблиці 4.1 наведено основні технічні характеристики, яких має дотримуватись пристрій користувача для стабільної та ефективної роботи застосунку.

Таблиця 4.1 – Вимоги до апаратного забезпечення мобільного пристрою

Вимоги до конфігурування	Рекомендовано	Мінімально
Операційна система	Android 11+ / iOS 14+	Android 8.0 / iOS 12
Процесор	8-ядерний ARM 2.0 ГГц або Apple A13 Bionic	4-ядерний ARM 1.4 ГГц або Apple A10
Оперативна пам’ять	4 ГБ	2 ГБ
Вільний простір на диску	1.5 ГБ	500 МБ
Камера	12 Мп, автофокус	8 Мп, базова фокусна система
Дисплей	FullHD (1920x1080), 5.5 дюйма або більше	HD (1280x720), 5.0 дюймів

Після встановлення застосунку, користувач потрапляє на вітальний екран, який містить логотип, назву програми та кнопку початку роботи. У першому запуску також може бути доступна коротка інструкція або підказки щодо використання функцій. Головним інтерфейсом взаємодії є екран розпізнавання. Користувачеві надається можливість сфотографувати рослину в реальному часі або завантажити зображення з галереї пристрою. Після цього активується механізм розпізнавання, який реалізовано з використанням стороннього API або власної нейронної мережі. Залежно від якості зображення та складності об’єкта, процес обробки триває від кількох секунд до хвилини.

Після завершення обробки користувач отримує результати ідентифікації у

вигляді ймовірної назви рослини, відсоткової оцінки точності, короткої енциклопедичної інформації, а також альтернативних назв (зокрема народних назв українською мовою). У разі необхідності користувач може перейти до розділу з детальним описом або скопіювати отриману інформацію для подальшого використання.

Інтерфейс програми поділений на кілька вкладок:

- «Головна» – екран із кнопкою початку розпізнавання;
- «Збережене» – список збережених рослин;
- «Про програму» – інформація про авторів та технічну підтримку;
- «Допомога» – короткі поради щодо зйомки та ідентифікації.

Усі дії супроводжуються підказками, візуальними індикаторами або повідомленнями, що дозволяє новим користувачам швидко адаптуватися до використання застосунку. У разі виникнення труднощів передбачено окремий розділ FAQ із відповідями на поширені запитання.Таким чином, інструкція користувача надає чітке уявлення про вимоги до пристрою та алгоритм взаємодії з функціоналом програми. Це дозволяє забезпечити зручність та ефективність використання застосунку для широкого кола користувачів, незалежно від мобільної платформи.

4.3 Висновки

У четвертому розділі було проведено тестування основних функціональних модулів мобільного застосунку для ідентифікації рослин. Особливу увагу було приділено перевірці коректності обробки вхідних зображень, стабільності роботи на різних мобільних платформах, а також точності відображення результатів розпізнавання. Окрім того, було розроблено інструкцію користувача, яка містить детальний опис процесу встановлення та запуску програмного продукту, вимоги до мобільного пристрою, а також покроковий опис основного функціоналу. Інструкція розрахована на широке коло користувачів, включаючи тих, хто не має спеціальної технічної підготовки.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено мобільний застосунок Plant Health для візуальної детекції об’єктів рослинного походження з використанням технологій комп’ютерного зору та зовнішніх інтелектуальних API. Застосунок забезпечує ідентифікацію виду рослини, виявлення можливих хвороб, формування персоналізованих порад щодо догляду, а також збереження інформації у локальній базі даних на мобільному пристрої. Бакалаврську кваліфікаційну роботу реалізовано згідно методичних вказівок [27].

Було проведено огляд сучасного стану технологій систем візуальної ідентифікації рослин, проаналізовано найпоширеніші мобільні рішення (PlantNet, LeafSnap, PictureThis, PlantVillage Nuru, Flora Incognita), визначено їхні обмеження та сформульовано переваги запропонованого рішення. Доведено актуальність теми дослідження, зокрема в контексті україномовної підтримки, доступності інтерфейсу та інтеграції функцій розпізнавання і діагностики в єдиній системі.

У першому розділі роботи виконано порівняльний аналіз аналогічних програмних рішень, охарактеризовано методи візуальної детекції, а також обґрунтовано вибір підходу до реалізації системи з урахуванням архітектурної гнучкості, масштабованості та використання зовнішніх API.

У другому розділі було розроблено загальну модель програмного продукту, описано основні алгоритми роботи, здійснено побудову діаграми діяльності системи, розглянуто етапи попередньої обробки зображень, логіку запитів до API-сервісів та інтерпретацію отриманих результатів. Розглянуто різні сценарії взаємодії користувача із застосунком.

У третьому розділі обґрунтовано вибір середовища розробки Android Studio та фреймворку Flutter для створення кросплатформного мобільного застосунку. Реалізовано інтерфейс користувача, модулі для взаємодії із застосунком, підключення до API Plant.id, Kindwise, Wikipedia, а також модуль локальної бази даних на основі SQLite через бібліотеку Sqflite. Особливу увагу було приділено

збереженню даних користувача, що дозволяє переглядати збережені результати у режимі офлайн.

У четвертому розділі було проведено тестування програмного забезпечення. Перевірено основний функціонал, зокрема завантаження зображень, точність ідентифікації, виявлення хвороб, швидкість обробки запитів, генерацію порад та збереження результатів. Продемонстровано приклади тестових сценаріїв, проаналізовано реакцію системи на нестандартні ситуації. Проведені тести підтвердили стабільність, функціональну завершеність і відповідність додатку поставленим задачам.

Таким чином, бакалаврську кваліфікаційну роботу виконано відповідно до поставленої мети – створено сучасний мобільний застосунок для ідентифікації об’єктів рослинного походження з підтримкою української мови, що забезпечує високу точність, зручність користування та практичну користь для аграріїв, ботаніків, садівників і звичайних користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Review of CNN Applications in Smart Agriculture Using Computer Vision [Електронний ресурс] – Режим доступу до ресурсу: https://www.mdpi.com/1424

2. Artificial Neural Networks in Agriculture [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/S0168169925000444

3. Жеребнюк М.Р. Аналіз сучасних технологій візуальної ідентифікації рослин / Матеріали 2-ї конференції «Modern Scientific Research: Theoretical and Practical Aspects» (26-28 травня). 2025 р., Рига, Латвія – С.80-81.

4. Computer Vision in Smart Agriculture and Precision Farming: Techniques and Applications [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S2589721724000266

5. Deep Learning Techniques for Hyperspectral Image Analysis in Agriculture: A Review [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/abs/

6. Computer Vision Technology in Agricultural Automation — A Review [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencelab.com/ /pii/S2214317319301751

7. Barbedo, J.G.A. "A review on the use of machine learning for plant disease detection using hyperspectral data." *Biosystems Engineering*, 2020. – [Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/ /pii/S1537511020300487

8. Mohanty, S.P., Hughes, D.P., Salathé, M. "Using deep learning for image-based plant disease detection." *Frontiers in Plant Science*, 2016. – [Електронний ресурс] – https://www.frontiersin.org/articles/10.3389/fpls.2016.01419/full

9. TensorFlow Lite [Електронний ресурс] – Режим доступу до ресурсу: https://www.tensorflow.org/lite

10. MobileNet Models for Efficient On-Device Vision [Електронний ресурс] – Режим доступу: https://ai.googleblog.com/2017/06/mobilenets-open-source-models-

11. PlantNet – [Електронний ресурс] – Режим доступу: https://identify.plantnet.org/

12. LeafSnap – [Електронний ресурс] – Режим доступу:https://leafsnap.com/

13. PictureThis – [Електронний ресурс] – Режим доступу: https://www.picturethisai.com/

14. PlantVillage Nuru – [Електронний ресурс] – Режим доступу: https://plantvillage.psu.edu/nuru

15. Flora Incognita – [Електронний ресурс] – Режим доступу: https://floraincognita.com/en/

16. Plant Disease Detection Using Deep Learning by N. R. Choudhury, et al. "Artificial Intelligence in Agriculture" 2020. Springer.

17. Li H. Machine Learning Methods. 2024 Edition. – Springer, 2024. – 412 c..

18. Google Cloud Vision API [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/vision

19. UML Diagrams for Software Design [Електронний ресурс] – Режим доступу до ресурсу: https://www.uml-diagrams.org/

20. draw.io – офіційний сайт [Електронний ресурс] – Режим доступу: https://www.drawio.com/

21. Oracle Java Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/en/java/

22. Kotlin Language Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org/docs/home.html

23. Apple Swift Programming Language Guide [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/swift/resources/

24. React Native – офіційна документація [Електронний ресурс] – Режим доступу до ресурсу: https://reactnative.dev/docs/getting-started

25. Flutter UI Framework [Електронний ресурс] – Режим доступу до ресурсу: https://flutter.dev/docs/development/ui

26. Dart Language Tour – офіційна документація [Електронний ресурс] – Режим доступу до ресурсу: https://dart.dev/guides/language/language-tour

27. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О.Н. Романюк, Г.О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«25» 03 2025 року

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка системи візуальної детекції об'єктів рослинного походження на основі штучного інтелекту»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. Мельник О.В.
«24» 03 2025 року.

Виконав:
студент гр. 5ПІ-21б Жеребнюк М.Р.
«24» 03 2025 року.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка системи візуальної детекції об’єктів рослинного походження на основі штучного інтелекту».

Галузь застосування – мобільні технології.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 18 від 18 лютого 2025 р.

3. Мета та призначення розробки.

Метою роботи є підвищення точності та доступності процесу ідентифікації об'єктів рослинного походження за допомогою мобільного програмного застосунку на основі технологій штучного інтелекту.

4. Вихідні дані для проведення НДР

1. A Review of CNN Applications in Smart Agriculture Using Computer Vision [Електронний ресурс] – Режим доступу до ресурсу: https://www.mdpi.com/1424-8220/25/2/472
2. Mohanty, S.P., Hughes, D.P., Salathé, M. "Using deep learning for image-based plant disease detection." Frontiers in Plant Science, 2016. – [Електронний ресурс] – https://www.frontiersin.org/articles/10.3389/fpls.2016.01419/full
3. TensorFlow Lite [Електронний ресурс] – Режим доступу до ресурсу: https://www.tensorflow.org/lite
4. Li H. Machine Learning Methods. 2024 Edition. – Springer, 2024. – 412 с.
5. Flutter UI Framework [Електронний ресурс] – Режим доступу до ресурсу: https://flutter.dev/docs/development/ui

5. Технічні вимоги

Вхідні дані – зображення рослини, запит на ідентифікацію.

Вихідні дані – назва рослини, можливі хвороби, короткий опис, поради з догляду, збереження результатів у локальну базу даних.

6. Конструктивні вимоги.

Інтерфейс програми повинен відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

–пояснювальна записка до бакалаврської кваліфікаційної роботи;

–технічне завдання;

–лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз розвитку технологій систем візуальної детекціїоб'єктів рослинного походження	25.03.25 – 05.04.25
2	Розробка структури та алгоритмів програмного застосунку	06.04.25 – 18.04.25
3	Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення	19.04.25 – 21.04.25
4	Розробка системи візуальної детекції об'єктів рослинного походження	22.04.25 – 17.05.25
5	Тестування програмного забезпечення	18.05.25 – 25.05.25
6	Підготовка до захисту бакалаврської роботи	26.05.25 – 30.05.25

10. Порядок контролю та прийняття.

Усі етапи бакалаврської роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Додаток Б – Протокол перевірки на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка системи візуальної детекції об'єктів рослинного походження на основі штучного інтелекту

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота

Підрозділ кафедра ПЗ, ФІТКІ, 5ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 8,4 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Мельник О. В., к. т. н., доцент кафедри ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Жеребнюк М.Р.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програми

Лістинг програмного коду мобільного застосунку

Модуль main.dart

```
import 'dart:convert';
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:image_picker/image_picker.dart';
import 'database_helper.dart';
import 'database_service.dart';
import 'plant_id_service.dart';
import 'crop_health_service.dart';
import 'package:flutter/material.dart';
import 'welcome_screen.dart';
import 'help_screen.dart';
import 'about_screen.dart';
import 'my_plants_screen.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Plant Health',
      theme: ThemeData(
        primarySwatch: Colors.green,
        useMaterial3: true,
      ),
      debugShowCheckedModeBanner: false,
      initialRoute: '/',
      routes: {
        '/': (context) => const WelcomeScreen(),
        '/home': (context) => PlantIdentifierScreen(),
        '/help': (context) => const HelpScreen(),
        '/about': (context) => const AboutScreen(),
        '/myplants': (context) => MyPlantsScreen(),
      },
    );
  }
}

class PlantIdentifierScreen extends StatefulWidget {
  @override
  _PlantIdentifierScreenState createState() => _PlantIdentifierScreenState();
}


class _PlantIdentifierScreenState extends State<PlantIdentifierScreen> {
  final PlantIdService _plantIdService = PlantIdService();
  final CropHealthService _cropHealthService = CropHealthService();
  File? _image;
  List<Map<String, dynamic>> _plantSuggestions = [];
  bool _isLoading = false;
  bool _isDiseaseMode = false;


  Future<void> _pickImage() async {
    final picker = ImagePicker();
    final pickedFile = await picker.pickImage(source: ImageSource.gallery);

    if (pickedFile != null) {
      setState(() {
```

```
        _image = File(pickedFile.path);
        _plantSuggestions = [];
      });
    }
  }

  Future<void> _identifyPlant() async {
    if (_image == null) return;

    setState(() {
      _isLoading = true;
      _isDiseaseMode = false;
      _plantSuggestions = [];
    });

    final bytes = _image!.readAsBytesSync();
    final base64Image = base64Encode(bytes);

    try {
      final rawResults = await _plantIdService.identifyPlant(base64Image);
      final results = rawResults.where((item) {
        final prob = item["probability"] ?? 1.0;
        return prob >= 0.8;
      }).toList();

      setState(() {
        _plantSuggestions = results;
      });
    } catch (e) {
      setState(() {
        _plantSuggestions = [
          {
            "name": "",
            "scientific_name": "",
            "description": "Помилка: $e",
            "showDescription": true,
          }
        ];
      });
    } finally {
      setState(() {
        _isLoading = false;
      });
    }
  }

  Future<void> _identifyDisease() async {
    if (_image == null) return;

    setState(() {
      _isLoading = true;
      _isDiseaseMode = true;
      _plantSuggestions = [];
    });

    final bytes = _image!.readAsBytesSync();
    final base64Image = base64Encode(bytes);

    try {
      final results = await _cropHealthService.identifyDisease(base64Image);
      final filteredResults = results.where((item) {
        final prob = item["probability"] ?? 1.0;
        return prob >= 0.71;
      }).toList();
      Map<String, dynamic>? selected;

      for (final item in filteredResults) {
        final diseaseName = item["disease"];
        final scientificName = item["scientific_name"] ?? "";
        final hasUkrWiki = await
```

```
_plantIdService.hasUkrainianWikipediaArticle(scientificName);
        if (!hasUkrWiki && diseaseName.isNotEmpty) {
          final hasCommonNameWiki = await
_plantIdService.hasUkrainianWikipediaArticle(diseaseName);
          if (hasCommonNameWiki) {
            selected = item;
            break;
          }
        } else if (hasUkrWiki) {
          selected = item;
          break;
        }
      }
      selected ??= filteredResults.isNotEmpty ? filteredResults.first : null;
      if (selected != null) {
        final diseaseName = selected["disease"];
        final scientificName = selected["scientific_name"] ?? "";

        String commonName = await _plantIdService.getUkrainianCommonName(scientificName);
        if (commonName.isEmpty && diseaseName.isNotEmpty) {
          commonName = await _plantIdService.getUkrainianCommonName(diseaseName);
          if (commonName.isEmpty) {
            commonName = await _plantIdService.translateToUkrainian(diseaseName);
          }
        }

        final description = await _plantIdService.getDescriptionInUkrainian(
          scientificName.isNotEmpty ? scientificName : diseaseName,
        );

        setState(() {
          _plantSuggestions = [
            {
              "name": commonName.isNotEmpty ? commonName : diseaseName,
              "scientific_name": scientificName,
              "description": description,
              "showDescription": true,
            }
          ];
        });
      } else {
        setState(() {
          _plantSuggestions = [
            {
              "name": "",
              "description": "Не вдалося визначити хворобу.",
              "showDescription": true,
            }
          ];
        });
      }
    } catch (e) {
      setState(() {
        _plantSuggestions = [
          {
            "name": "",
            "description": "Помилка: $e",
            "showDescription": true,
          }
        ];
      });
    } finally {
      setState(() {
        _isLoading = false;
      });
    }
  }
```

```
Future<void> _toggleDescription(int index) async {
  final plant = _plantSuggestions[index];
  final isShown = plant["showDescription"] ?? false;

  if (_isDiseaseMode) return; // не перемикати опис у режимі хвороб

  if (isShown) {
    setState(() {
      _plantSuggestions[index]["showDescription"] = false;
    });
  } else {
    if (plant["description"] == null && plant["scientific_name"] != null) {
      final description = await _plantIdService.getDescriptionInUkrainian(
          plant["scientific_name"]);
      setState(() {
        _plantSuggestions[index]["description"] = description;
        _plantSuggestions[index]["showDescription"] = true;
      });
    } else {
      setState(() {
        _plantSuggestions[index]["showDescription"] = true;
      });
    }
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text("Ідентифікація рослин"),
      actions: [
        IconButton(
          icon: Icon(Icons.help_outline),
          tooltip: 'Довідка',
          onPressed: () {
            Navigator.pushNamed(context, '/help');
          },
        ),
        IconButton(
          icon: Icon(Icons.info_outline),
          tooltip: 'Про програму',
          onPressed: () {
            Navigator.pushNamed(context, '/about');
          },

        ),
        IconButton(
          icon: Icon(Icons.bookmark),
          tooltip: 'Мої рослини',
          onPressed: () {
            Navigator.pushNamed(context, '/myplants');
          },
        ),
      ],
    ),

    body: Center(
      child: SingleChildScrollView(
        padding: EdgeInsets.all(16.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            if (_image != null)
              Image.file(
                _image!,
                height: 200,
              ),
            if (_plantSuggestions.isNotEmpty)
              ..._plantSuggestions.asMap().entries.map((entry) {
```

```
final i = entry.key;
final plant = entry.value;
final showDesc = plant["showDescription"] ?? false;

return Card(
  margin: EdgeInsets.symmetric(vertical: 8),
  elevation: 3,
  child: Padding(
    padding: const EdgeInsets.all(12.0),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        if (plant['name'] != null &&
            plant['name'].toString().isNotEmpty)
          Text(
            _isDiseaseMode
                ? "□ Хвороба: ${plant['name']}"
                : "🌿 Назва: ${plant['name']}",
            style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
          ),
        if (_isDiseaseMode &&
            plant['scientific_name'] != null &&
            plant['scientific_name'].toString().isNotEmpty)
          Text(
            "🔬 Наукова назва: ${plant['scientific_name']}",
            style: TextStyle(fontStyle: FontStyle.italic),
          ),

        if (!_isDiseaseMode &&
            plant['scientific_name'] != null &&
            plant['scientific_name'].toString().isNotEmpty)
          Text("🔬 Наукова назва: ${plant['scientific_name']}"),
        if (!_isDiseaseMode)
          ElevatedButton(
            onPressed: () => _toggleDescription(i),
            child: Text(showDesc
                ? "Сховати опис"
                : "Показати опис"),
          ),
        if (showDesc && plant["description"] != null)
          Padding(
            padding: const EdgeInsets.only(top: 8.0),
            child: Text(
              "📖 ${plant['description']}",
              textAlign: TextAlign.justify,
            ),
          ),
        ElevatedButton(
          onPressed: () async {
            if (_plantSuggestions.isNotEmpty && _image != null) {
              final plant = _plantSuggestions.first;
              final db = DatabaseHelper();
              await db.insertPlant({
                'name': plant['name'],
                'scientific_name': plant['scientific_name'],
                'description': plant['description'],
                'image_path': _image!.path,
              });
              ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(content: Text("Рослину збережено до списку!")),
              );
            }
          },
          child: Text("Зберегти до моїх рослин"),
        ),

      ],
    ),
  ),
```

```
                );
              }),
            if (_isLoading) CircularProgressIndicator(),
            SizedBox(height: 16),
            ElevatedButton(
              onPressed: _pickImage,
              child: Text("Вибрати зображення"),
            ),
            ElevatedButton(
              onPressed: _identifyPlant,
              child: Text("Ідентифікувати рослину"),
            ),
            ElevatedButton(
              onPressed: _identifyDisease,
              child: Text("Ідентифікувати хворобу"),
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

Модуль plant_id_service.dart

```
import 'dart:convert';
import 'package:http/http.dart' as http;

class PlantIdService {
  final String apiKey = "ZhV6DM5WIMtiWXLFrOrrLtDPKgEewSYdyuL51NK35RGfezB83c";
  final String endpoint = "https://api.plant.id/v2/identify";

  Future<List<Map<String, dynamic>>> identifyPlant(String base64Image) async {
    final response = await http.post(
      Uri.parse(endpoint),
      headers: {
        'Content-Type': 'application/json',
        'Api-Key': apiKey,
      },
      body: jsonEncode({
        "images": [base64Image],
        "organs": ["leaf"],
      }),
    );

    if (response.statusCode != 200) {
      throw Exception("Помилка: ${response.statusCode}");
    }

    final data = jsonDecode(response.body);
    final suggestions = data["suggestions"] ?? [];

    suggestions.sort((a, b) =>
        (b["probability"] as num).compareTo(a["probability"] as num));
    for (var suggestion in suggestions) {
      final scientificName = suggestion["plant_details"]?["scientific_name"];
      if (scientificName == null) continue;
      final commonName = await getUkrainianCommonName(scientificName);
      final hasWiki = await _hasWikipediaPage(scientificName);
      if (!hasWiki) continue;

      return [
        {
          "name": commonName,
          "scientific_name": scientificName,
          "description": null,
        }
      ];
    }
    return [
```

```
      {
        "name": "",
        "scientific_name": "",
        "description": null,
      }
    ];
  }

  Future<String> getUkrainianCommonName(String scientificName) async {
    try {
      final url =

"https://uk.wikipedia.org/w/api.php?action=query&format=json&titles=$scientificName&redirec
ts=1";

      final response = await http.get(Uri.parse(url));
      final data = jsonDecode(response.body);

      final pages = data["query"]?["pages"];
      if (pages == null || pages.isEmpty) return "";

      final page = pages.values.first;
      final title = page["title"];

      // Повертаємо лише якщо назва сторінки не дорівнює науковій
      if (title != null && title != scientificName) {
        return title;
      }
    } catch (_) {}

    return "";
  }

  Future<String> getDescriptionInUkrainian(String scientificName) async {
    if (scientificName.trim().isEmpty || scientificName == "Невідомо") {
      return "Опис недоступний.";
    }

    try {
      final url =

"https://uk.wikipedia.org/w/api.php?action=query&format=json&prop=extracts&titles=$scientif
icName&exintro=1&explaintext=1";

      var response = await http.get(Uri.parse(url));
      var data = jsonDecode(response.body);
      var pages = data["query"]?["pages"] ?? {};
      if (pages.isEmpty) return "Опис не знайдено.";

      var page = pages.values.first;
      final ukText = page["extract"] ?? "";

      if (ukText.isNotEmpty) return ukText;

      // Якщо немає українською, беремо англійською і перекладаємо
      final enUrl =

"https://en.wikipedia.org/w/api.php?action=query&format=json&prop=extracts&titles=$scientif
icName&exintro=1&explaintext=1";
      response = await http.get(Uri.parse(enUrl));
      data = jsonDecode(response.body);
      pages = data["query"]?["pages"] ?? {};
      if (pages.isEmpty) return "Опис не знайдено.";

      page = pages.values.first;
      final enText = page["extract"] ?? "";
      if (enText.isEmpty) return "Опис не знайдено.";

      return await _translateText(enText, "uk");
    } catch (_) {
```

```
      return "Не вдалося завантажити опис.";
    }
  }

  Future<bool> _hasWikipediaPage(String scientificName) async {
    final langs = ['uk', 'en'];
    for (var lang in langs) {
      final url =

"https://$lang.wikipedia.org/w/api.php?action=query&format=json&titles=$scientificName";
      final response = await http.get(Uri.parse(url));
      if (response.statusCode == 200) {
        final data = jsonDecode(response.body);
        final pages = data["query"]?["pages"] ?? {};
        final page = pages.values.first;
        if (page["missing"] == null) return true;
      }
    }
    return false;
  }

  Future<String> _translateText(String text, String targetLang) async {
    // Обрізаємо текст до 500 символів
    final truncatedText = text.length > 500 ? text.substring(0, 500) : text;

    final String url =

"https://api.mymemory.translated.net/get?q=${Uri.encodeComponent(truncatedText)}&langpair=e
n|$targetLang";
    final response = await http.get(Uri.parse(url));
    if (response.statusCode == 200) {
      final data = jsonDecode(response.body);
      return data["responseData"]["translatedText"] ?? text;
    }
    return text;
  }

  Future<String> translateToUkrainian(String text) async {
    return await _translateText(text, "uk");
  }
  Future<bool> hasUkrainianWikipediaArticle(String title) async {
    if (title.trim().isEmpty) return false;

    final url =
        "https://uk.wikipedia.org/w/api.php?action=query&format=json&titles=$title";
    final response = await http.get(Uri.parse(url));
    if (response.statusCode != 200) return false;

    final data = jsonDecode(response.body);
    final pages = data["query"]?["pages"];
    if (pages == null || pages.isEmpty) return false;

    final page = pages.values.first;
    return page["missing"] == null;
  }

}
```

Модуль crop_health_service.dart

```
import 'dart:convert';
import 'package:http/http.dart' as http;

class CropHealthService {
  final String apiKey = 'TXITxqfrBpPJOhFjMfTWAmNmnS31q1edUPXCFzQW90Dw61hzmp'; // Замініть
```

```
на свій API-ключ
  final String endpoint = 'https://crop.kindwise.com/api/v1/identification';

  Future<List<Map<String, dynamic>>> identifyDisease(String base64Image) async {
    final response = await http.post(
      Uri.parse(endpoint),
      headers: {
        'Content-Type': 'application/json',
        'Api-Key': apiKey,
      },
      body: jsonEncode({
        "images": [base64Image],
      }),
    );

    if (response.statusCode != 200 && response.statusCode != 201) {
      throw Exception("Помилка: ${response.statusCode}");
    }


    final data = jsonDecode(response.body);
    final results = data["result"]?["disease"]?["suggestions"] ?? [];
    print(jsonEncode(results));

    return results.map<Map<String, dynamic>>((item) {
      return {
        "disease": item["name"] ?? "",
        "scientific_name": item["scientific_name"] ?? "",
        "confidence": item["probability"] ?? 0.0,
        "description": item["details"]?["description"] ?? "",
      };
    }).toList();
  }
}
```

Модуль database_service.dart

```
import 'dart:io';
import 'package:path/path.dart';
import 'package:path_provider/path_provider.dart';
import 'package:sqflite/sqflite.dart';

class DatabaseHelper {
  static final DatabaseHelper _instance = DatabaseHelper._internal();
  factory DatabaseHelper() => _instance;
  static Database? _database;

  DatabaseHelper._internal();

  Future<Database> get database async {
    if (_database != null) return _database!;
    _database = await _initDatabase();
    return _database!;
  }

  Future<Database> _initDatabase() async {
    Directory documentsDirectory = await getApplicationDocumentsDirectory();
    String path = join(documentsDirectory.path, "plants.db");
    return await openDatabase(
      path,
      version: 1,
      onCreate: _onCreate,
    );
  }

  Future _onCreate(Database db, int version) async {
    await db.execute('''
      CREATE TABLE plants (
        id INTEGER PRIMARY KEY,
        name TEXT,
```

```
        scientific_name TEXT,
        description TEXT,
        image_path TEXT
      )
    ''');
  }

  Future<int> insertPlant(Map<String, dynamic> plant) async {
    final db = await database;
    return await db.insert("plants", plant);
  }

  Future<List<Map<String, dynamic>>> getPlants() async {
    final db = await database;
    return await db.query("plants");
  }

  Future<Map<String, dynamic>?> getPlant(int id) async {
    final db = await database;
    final res = await db.query("plants", where: "id = ?", whereArgs: [id]);
    return res.isNotEmpty ? res.first : null;
  }
}
```

Модуль crop_health_service.dart

```
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';
import 'dart:async';

class DatabaseService {
  static final DatabaseService _instance = DatabaseService._internal();
  factory DatabaseService() => _instance;
  static Database? _database;

  DatabaseService._internal();

  Future<Database> get database async {
    if (_database != null) return _database!;
    _database = await _initDb();
    return _database!;
  }

  Future<Database> _initDb() async {
    final path = join(await getDatabasesPath(), 'plants.db');
    return openDatabase(
      path,
      onCreate: (db, version) {
        return db.execute('''
          CREATE TABLE plants(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT,
            scientific_name TEXT,
            description TEXT
          )
        ''');
      },
      version: 1,
    );
  }

  Future<void> insertPlant(Map<String, dynamic> plant) async {
    final db = await database;
    await db.insert('plants', plant, conflictAlgorithm: ConflictAlgorithm.replace);
  }

  Future<List<Map<String, dynamic>>> getPlants() async {
    final db = await database;
    return db.query('plants');
```

```
  }

  Future<void> deletePlant(int id) async {
    final db = await database;
    await db.delete('plants', where: 'id = ?', whereArgs: [id]);
  }
}
```

Модуль gemini_service.dart

```
import 'dart:convert';
import 'package:http/http.dart' as http;

class GeminiService {
  // Ваш Google API Key (має починатися на "AIza...")
  static const String _apiKey = 'AIzaSyBJLRtFbjz_WVulohdbTJmtI-wxNSh54Y0';

  // REST-ендпоінт для free-tier Generative Language API
  static final String _endpoint =
      'https://generativelanguage.googleapis.com'
      '/v1beta/models/gemini-2.0-flash:generateContent'
      '?key=$_apiKey';

  /// Згенерує поради з догляду за рослиною [plantName] українською.
  Future<String> generateCareTips(String plantName) async {
    final prompt = '''
Будь ласка, дай 3–5 коротких, чітких порад з догляду за рослиною "$plantName" українською
мовою.
''';

    final response = await http.post(
      Uri.parse(_endpoint),
      headers: {'Content-Type': 'application/json'},
      body: jsonEncode({
        'contents': [
          {
            'role': 'user',
            'parts': [
              {'text': prompt}
            ]
          }
        ],
        'generationConfig': {'maxOutputTokens': 200},
      }),
    );

    if (response.statusCode != 200) {
      throw Exception('Gemini API помилка: ${response.statusCode}\n${response.body}');
    }

    final data = jsonDecode(response.body);
    final candidates = data['candidates'] as List<dynamic>?;

    if (candidates == null || candidates.isEmpty) {
      return 'Не вдалося згенерувати поради.';
    }

    final contentObj = candidates[0]['content'];

    // Якщо content — просто рядок
    if (contentObj is String) {
      return contentObj.trim();
    }

    // Якщо content — Map
    if (contentObj is Map<String, dynamic>) {
      // Варіант 1: коли є 'parts' - список фрагментів
      if (contentObj.containsKey('parts')) {
        final parts = contentObj['parts'] as List<dynamic>;
```

```
        return parts
            .map((p) => (p as Map<String, dynamic>)['text'] as String? ?? '')
            .join()
            .trim();
      }
      // Варіант 2: коли є безпосередньо 'text'
      if (contentObj.containsKey('text')) {
        return (contentObj['text'] as String).trim();
      }
    }

    // Fallback — просто виведемо всю структуру
    return contentObj.toString().trim();
  }


}
```

Модуль my_plants_screen.dart

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'database_helper.dart';
import 'plant_details_screen.dart';

class MyPlantsScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Мої рослини")),
      body: FutureBuilder(
        future: DatabaseHelper().getPlants(),
        builder: (context, snapshot) {
          if (snapshot.connectionState != ConnectionState.done) {
            return Center(child: CircularProgressIndicator());
          }
          final plants = snapshot.data as List<Map<String, dynamic>>;
          if (plants.isEmpty) {
            return Center(child: Text("Немає збережених рослин"));
          }
          return ListView.builder(
            itemCount: plants.length,
            itemBuilder: (context, index) {
              final plant = plants[index];
              return ListTile(
                leading: Image.file(File(plant['image_path']), width: 50, height: 50),
                title: Text(plant['name']),
                subtitle: Text(plant['scientific_name']),
                onTap: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (_) => PlantDetailsScreen(plantId: plant['id']),
                    ),
                  );
                },
              );
            },
          );
        },
      ),
    );
  }
}
```

Модуль plant_details_screen.dart

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'gemini_service.dart';
import '../database_helper.dart'; // ваш сервіс для SQLite

class PlantDetailsScreen extends StatefulWidget {
  final int plantId;
  const PlantDetailsScreen({Key? key, required this.plantId}) : super(key: key);

  @override
  State<PlantDetailsScreen> createState() => _PlantDetailsScreenState();
}

class _PlantDetailsScreenState extends State<PlantDetailsScreen> {
  String? _careTips;
  bool _loadingTips = false;
  final GeminiService _gemini = GeminiService();

  Future<void> _loadCareTips(String scientificName) async {
    setState(() => _loadingTips = true);
    try {
      final tips = await _gemini.generateCareTips(scientificName);
      setState(() => _careTips = tips);
      // Якщо хочете зберегти поради в БД:
      // await DatabaseHelper.instance.updatePlant(
      //   widget.plantId,
      //   {'care_tips': tips},
      // );
    } catch (e) {
      setState(() => _careTips = 'Помилка генерації порад: $e');
    } finally {
      setState(() => _loadingTips = false);
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Деталі рослини")),
      body: FutureBuilder<Map<String, dynamic>?>(
        future: DatabaseHelper().getPlant(widget.plantId),
        builder: (context, snapshot) {
          if (snapshot.connectionState != ConnectionState.done) {
            return Center(child: CircularProgressIndicator());
          }
          if (!snapshot.hasData || snapshot.data == null) {
            return Center(child: Text("Рослину не знайдено"));
          }

          final plant = snapshot.data!;
          final imagePath = plant['image_path'] as String?;
          final name = plant['name'] as String? ?? 'Невідома';
          final scientific = plant['scientific_name'] as String? ?? '';
          final description = plant['description'] as String? ?? '';

          return SingleChildScrollView(
            padding: EdgeInsets.all(16),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                if (imagePath != null && File(imagePath).existsSync())
                  Image.file(File(imagePath))
                else
                  SizedBox(
                    height: 200,
                    child: Center(
                      child: Icon(Icons.image_not_supported, size: 60),
                    ),
```

```
                ),
                SizedBox(height: 12),
                Text(name,
                    style:
                    TextStyle(fontSize: 22, fontWeight: FontWeight.bold)),
                Text(scientific,
                    style: TextStyle(fontStyle: FontStyle.italic)),
                SizedBox(height: 12),
                Text(description),
                SizedBox(height: 24),
                Center(
                  child: ElevatedButton(
                    onPressed:
                    _loadingTips ? null : () => _loadCareTips(scientific),
                    child: Text(_loadingTips
                        ? 'Генерація...'
                        : 'Поради з догляду'),
                  ),
                ),
                if (_careTips != null) ...[
                  SizedBox(height: 20),
                  Text("Поради:",
                      style:
                      TextStyle(fontSize: 16, fontWeight: FontWeight.bold)),
                  SizedBox(height: 8),
                  Text(_careTips!, textAlign: TextAlign.justify),
                ],
              ],
            ),
          );
        },
      ),
    );
  }
}
```

Модуль welcome_screen.dart

```
import 'package:flutter/material.dart';

class WelcomeScreen extends StatelessWidget {
  const WelcomeScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.green.shade50,
      body: SafeArea(
        child: Center(
          child: Padding(
            padding: const EdgeInsets.symmetric(horizontal: 24.0),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Icon(Icons.eco, size: 80, color: Colors.green.shade700),
                const SizedBox(height: 20),
                const Text(
                  'Ласкаво просимо до Plant Health!',
                  style: TextStyle(
                    fontSize: 26,
                    fontWeight: FontWeight.bold,
                    color: Colors.green,
                  ),
                  textAlign: TextAlign.center,
                ),
```

```
                const SizedBox(height: 20),
                const Text(
                  'Розпізнавайте рослини та визначайте хвороби за допомогою фото.',
                  style: TextStyle(fontSize: 16),
                  textAlign: TextAlign.center,
                ),
                const SizedBox(height: 40),
                ElevatedButton(
                  style: ElevatedButton.styleFrom(
                    backgroundColor: Colors.green,
                    padding: const EdgeInsets.symmetric(horizontal: 32, vertical: 14),
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(12),
                    ),
                  ),
                  onPressed: () {
                    Navigator.pushReplacementNamed(context, '/home');
                  },
                  child: const Text(
                    'Почати роботу',
                    style: TextStyle(fontSize: 18, color: Colors.white),
                  ),
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }
}
```

Модуль about_screen.dart

```
import 'package:flutter/material.dart';

class AboutScreen extends StatelessWidget {
  const AboutScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.blueAccent,
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.all(20),
          child: Stack(
            children: [
              SingleChildScrollView(
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Center(
                      child: Text(
                        'Про програму\nPlant Health',
                        textAlign: TextAlign.center,
                        style: TextStyle(
                          fontSize: 26,
                          fontWeight: FontWeight.bold,
                          color: Colors.white,
                        ),
                      ),
                    ),
                    const SizedBox(height: 16),
                    Text(
                      'Plant Health — це мобільний додаток для розпізнавання рослин і виявлення хвороб за допомогою штучного інтелекту та фотоаналізу.',
                      style: TextStyle(color: Colors.white, fontSize: 16),
                    ),
```

```
                    const SizedBox(height: 12),
                    Text(
                      'Розробник: Жеребнюк Максим\nГрупа: 5ПІ-21б',
                      style: TextStyle(color: Colors.white, fontSize: 16),
                    ),
                    const SizedBox(height: 12),
                    Text(
                      'Додаток поширюється безкоштовно і має відкритий вихідний код.',
                      style: TextStyle(color: Colors.white, fontSize: 16),
                    ),
                    const SizedBox(height: 20),
                    Text(
                      'Версія: v.1.0.0',
                      style: TextStyle(color: Colors.white70, fontSize: 14),
                    ),
                  ],
                ),
              ),
              Positioned(
                top: 0,
                right: 0,
                child: IconButton(
                  icon: Icon(Icons.close, color: Colors.redAccent, size: 30),
                  onPressed: () => Navigator.pop(context),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Модуль help_screen.dart

```
import 'package:flutter/material.dart';

class HelpScreen extends StatelessWidget {
  const HelpScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.blueAccent,
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.all(20),
          child: Stack(
            children: [
              SingleChildScrollView(
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Center(
                      child: Text(
                        'Допомога',
                        style: TextStyle(
                          fontSize: 28,
                          fontWeight: FontWeight.bold,
                          color: Colors.white,
                        ),
                      ),
                    ),
                    const SizedBox(height: 16),
                    Text(
                      'Огляд функцій',
                      style: TextStyle(
                        fontSize: 20,
```

```
                        fontWeight: FontWeight.w600,
                        color: Colors.white,
                      ),
                    ),
                    const SizedBox(height: 10),
                    _bullet('- Завантаження зображень рослин або хвороб.'),
                    _bullet('- Ідентифікація виду рослини.'),
                    _bullet('- Визначення можливої хвороби.'),
                    _bullet('- Перегляд опису українською.'),
                    const SizedBox(height: 20),
                    Text(
                      'Підтримка',
                      style: TextStyle(
                        fontSize: 18,
                        fontWeight: FontWeight.w500,
                        color: Colors.white,
                      ),
                    ),
                    const SizedBox(height: 8),
                    Text(
                      'З питань, пов’язаних із використанням додатку або помилками, звер-
тайтесь за адресою: mrzherebnyuk@gmail.com',
                      style: TextStyle(color: Colors.white),
                    ),
                    const SizedBox(height: 20),
                    Text(
                      'Загальні питання',
                      style: TextStyle(
                        fontSize: 18,
                        fontWeight: FontWeight.w500,
                        color: Colors.white,
                      ),
                    ),
                    const SizedBox(height: 8),
                    Text(
                      'Цей додаток допомагає визначити вид рослини або хворобу за фото, ви-
користовуючи штучний інтелект.',
                      style: TextStyle(color: Colors.white),
                    ),
                  ],
                ),
              ),
              Positioned(
                top: 0,
                right: 0,
                child: IconButton(
                  icon: Icon(Icons.close, color: Colors.redAccent, size: 30),
                  onPressed: () => Navigator.pop(context),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }

  Widget _bullet(String text) {
    return Padding(
      padding: const EdgeInsets.symmetric(vertical: 4),
      child: Text(
        '• $text',
        style: TextStyle(color: Colors.white, fontSize: 16),
      ),
    );
  }
}
```

Додаток Г – Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА СИСТЕМИ ВІЗУАЛЬНОЇ ДЕТЕКЦІЇ ОБ’ЄКТІВ РОСЛИННОГО ПОХОДЖЕННЯ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

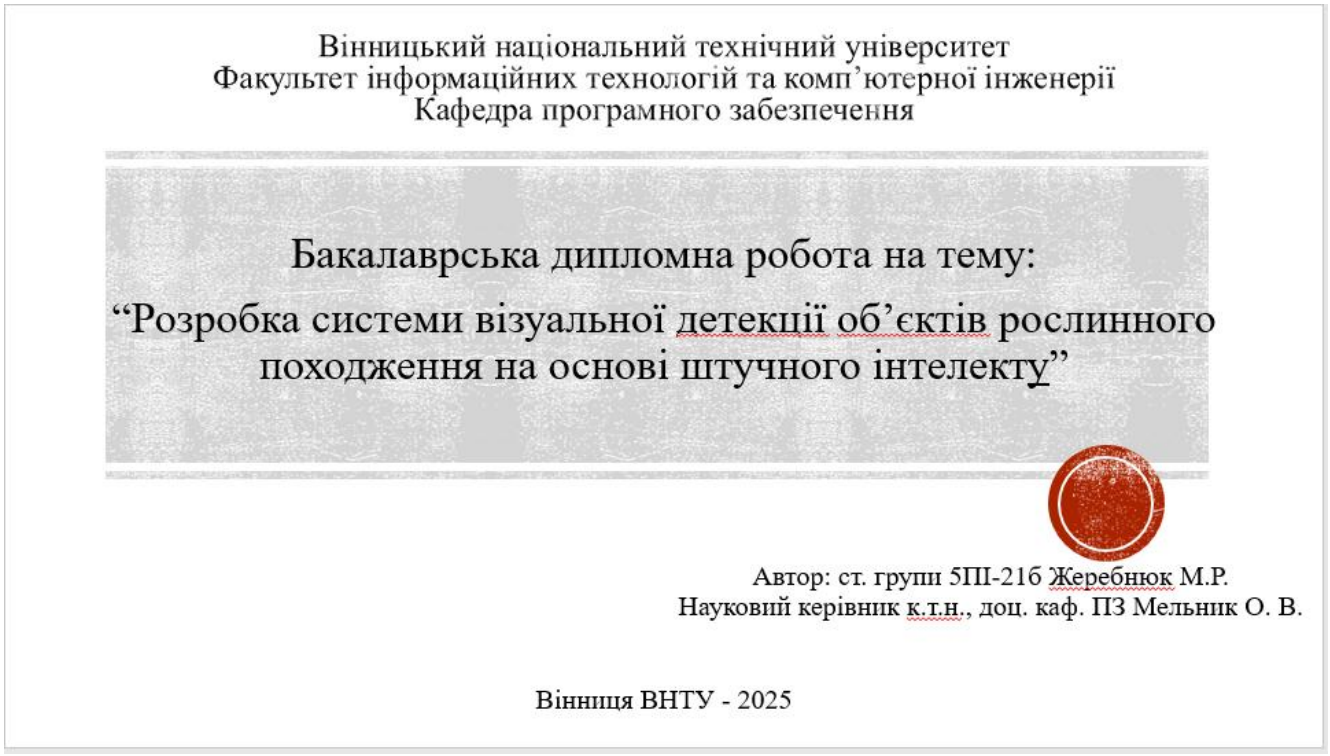

Рисунок Г.1 – Титульний слайд

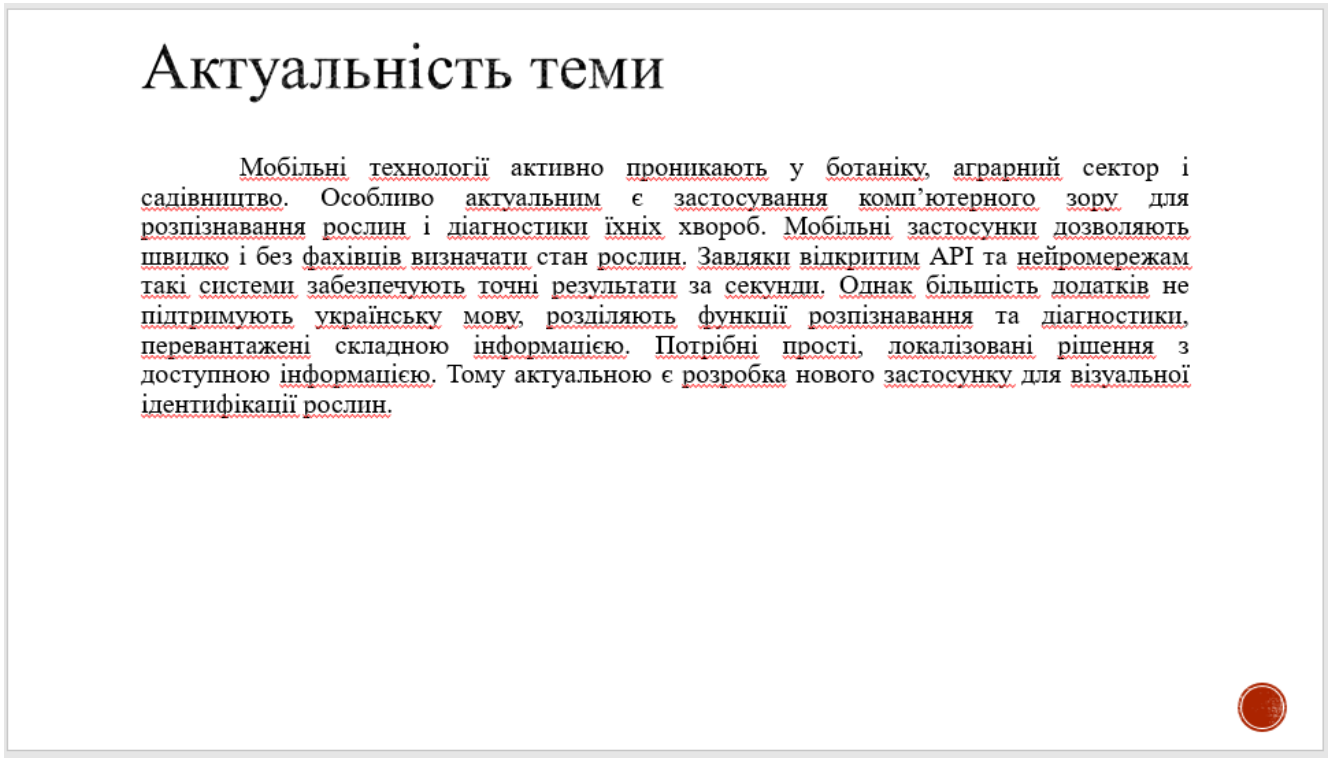

Рисунок Г.2 – Актуальність теми

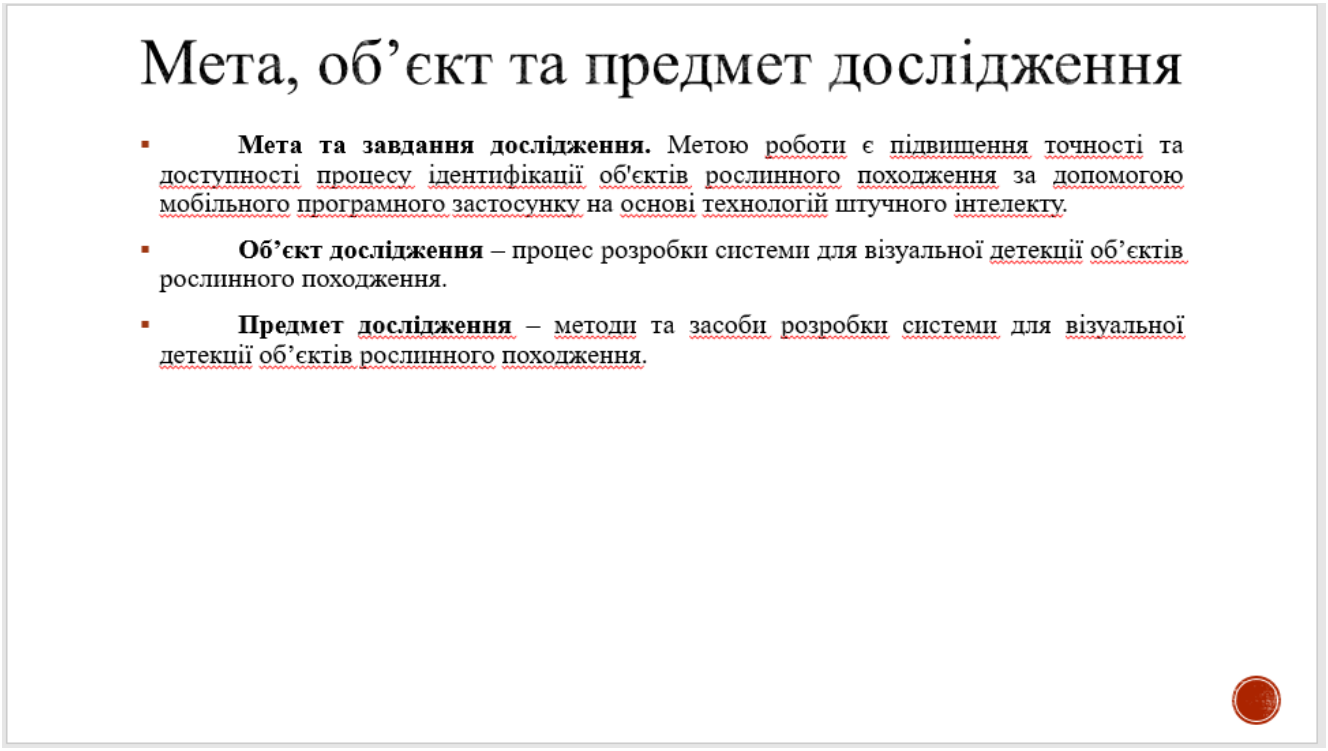

Рисунок Г.3 – Мета, об’єкт та предмет дослідження

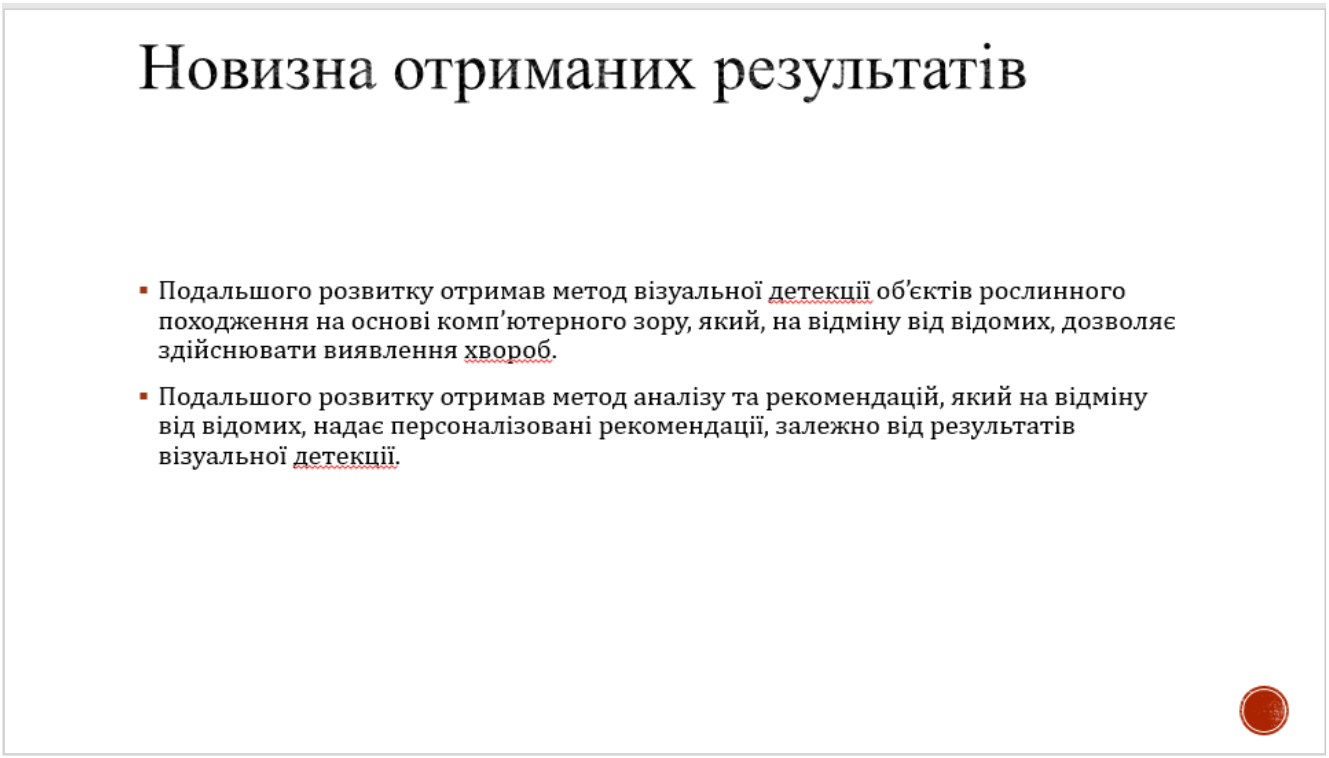

Рисунок Г.4 – Новизна отриманих результатів

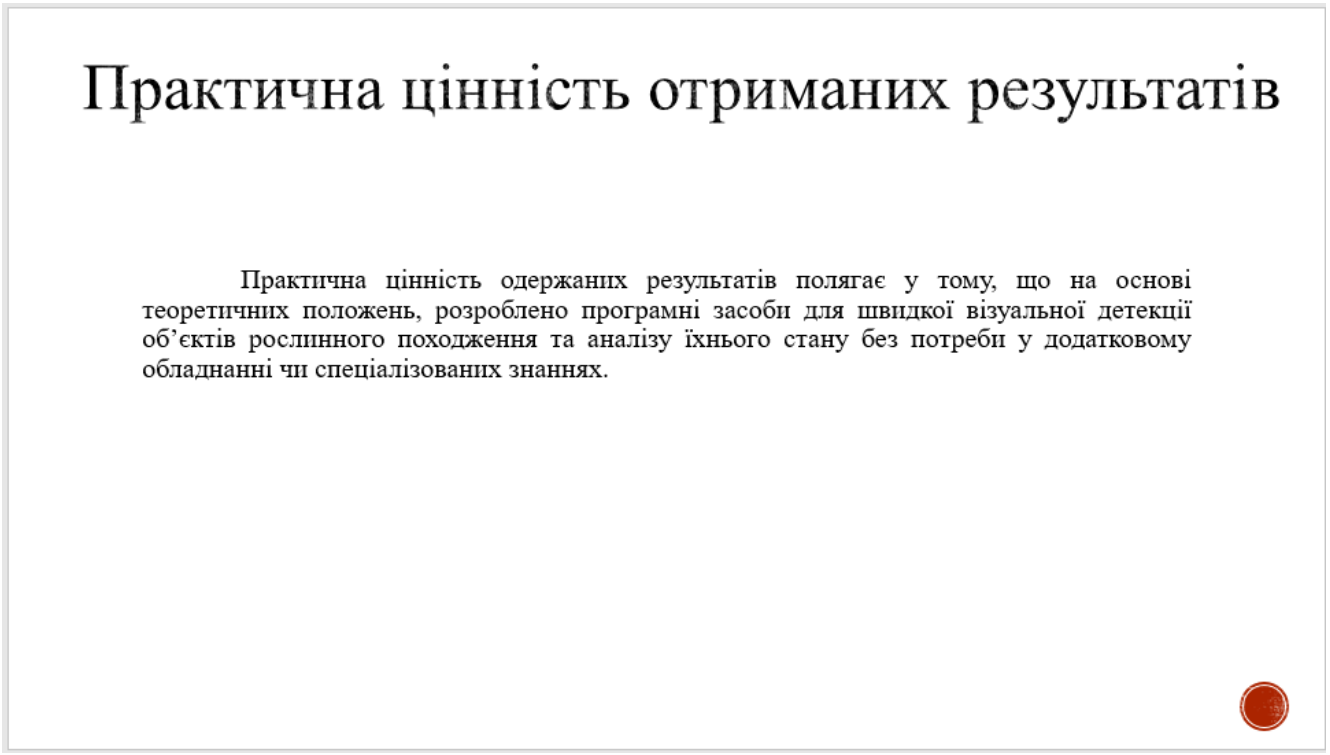

Рисунок Г.5 – Практична цінність отриманих результатів

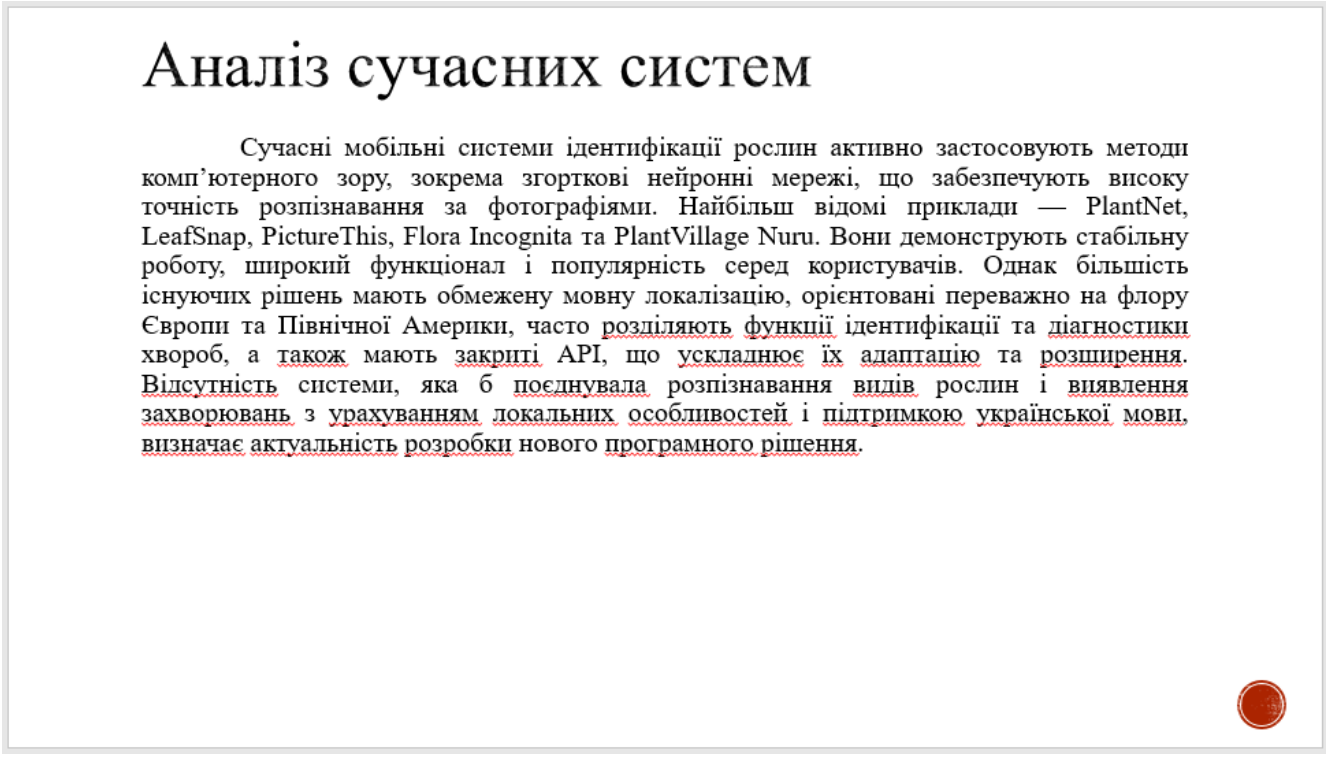

Рисунок Г.6 – Аналіз сучасних систем

Структура мобільної системи

Система має три рівні: інтерфейс з екранами вибору, обробки та збереження результатів; логіку з модулями для аналізу зображень, запитів до API Plant.id та Kindwise, перекладу й сортування даних; та локальну базу SQLite для збереження історії. Застосунок розроблено на Flutter з використанням Dart, що забезпечує кросплатформенність та високу продуктивність. Для кращого розуміння структури мобільної системи було розроблено діаграму діяльності.

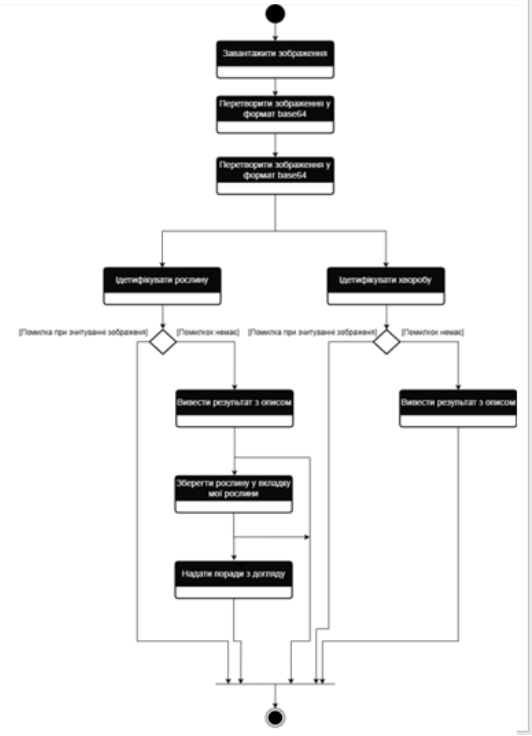

Рисунок Г.7 – Структура мобільної системи

Розробка графічного інтерфейсу

Інтерфейс забезпечує просту роботу: вибір зображення, ідентифікація виду або хвороби, відображення результатів у вигляді карток із назвами та описами.

Основні екрани застосунку

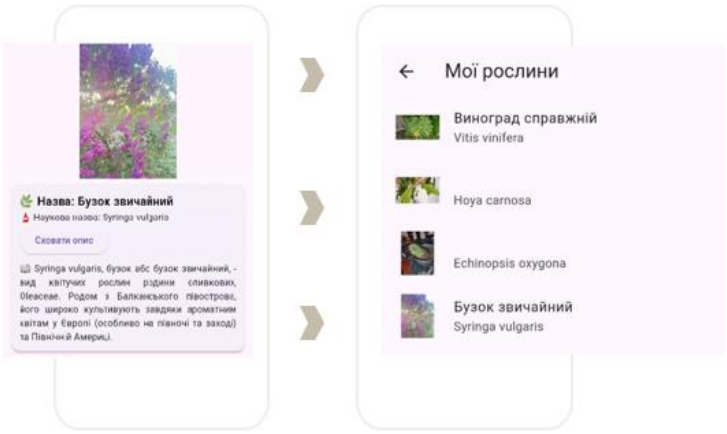

Рисунок Г.8 – Розробка графічного інтерфейсу

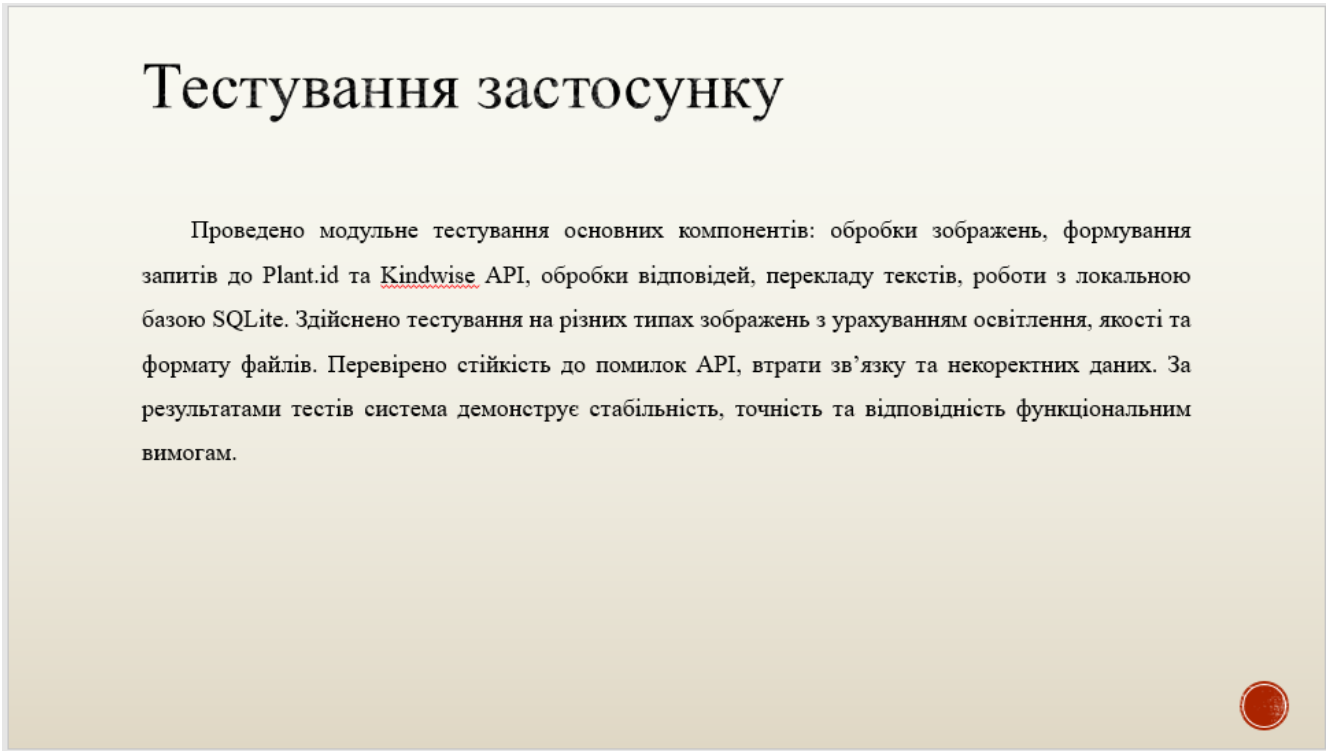

Рисунок Г.9 – Тестування застосунку

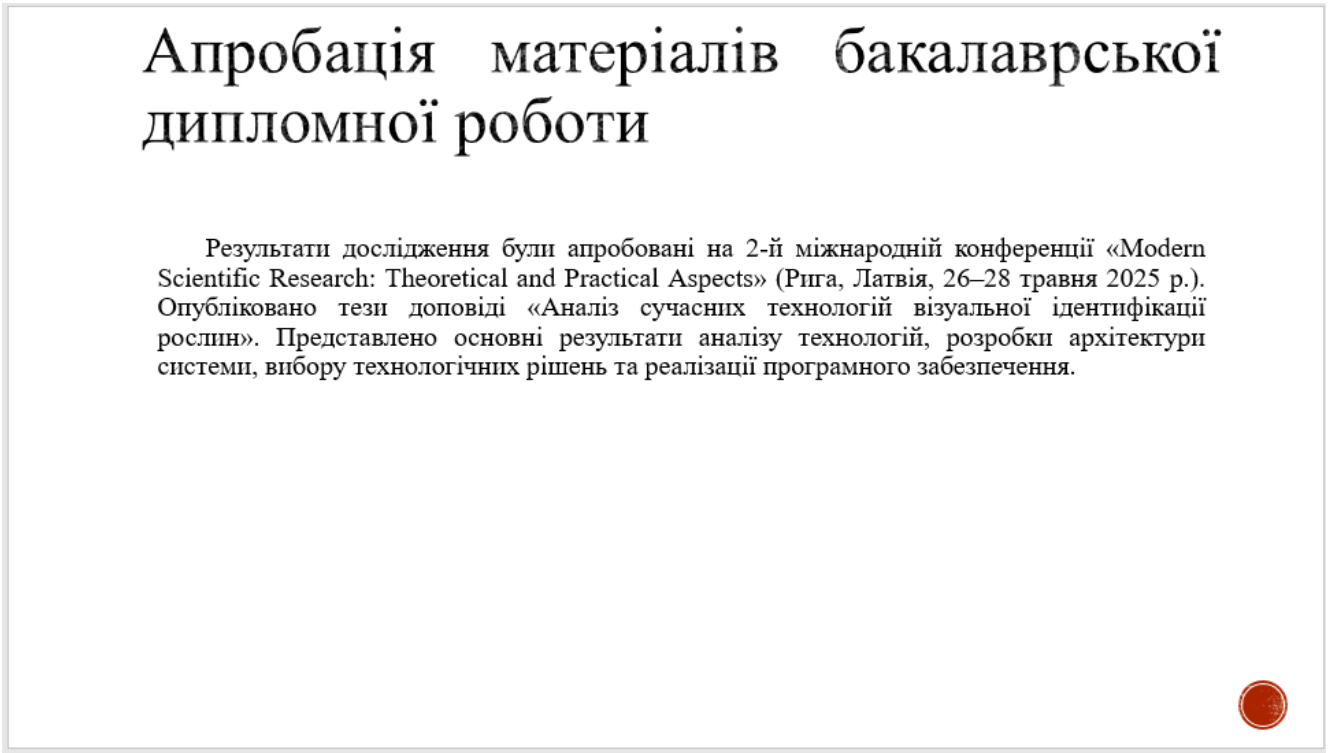

Рисунок Г.10 – Апробація матеріалів бакалаврської кваліфікаційної дипломної роботи

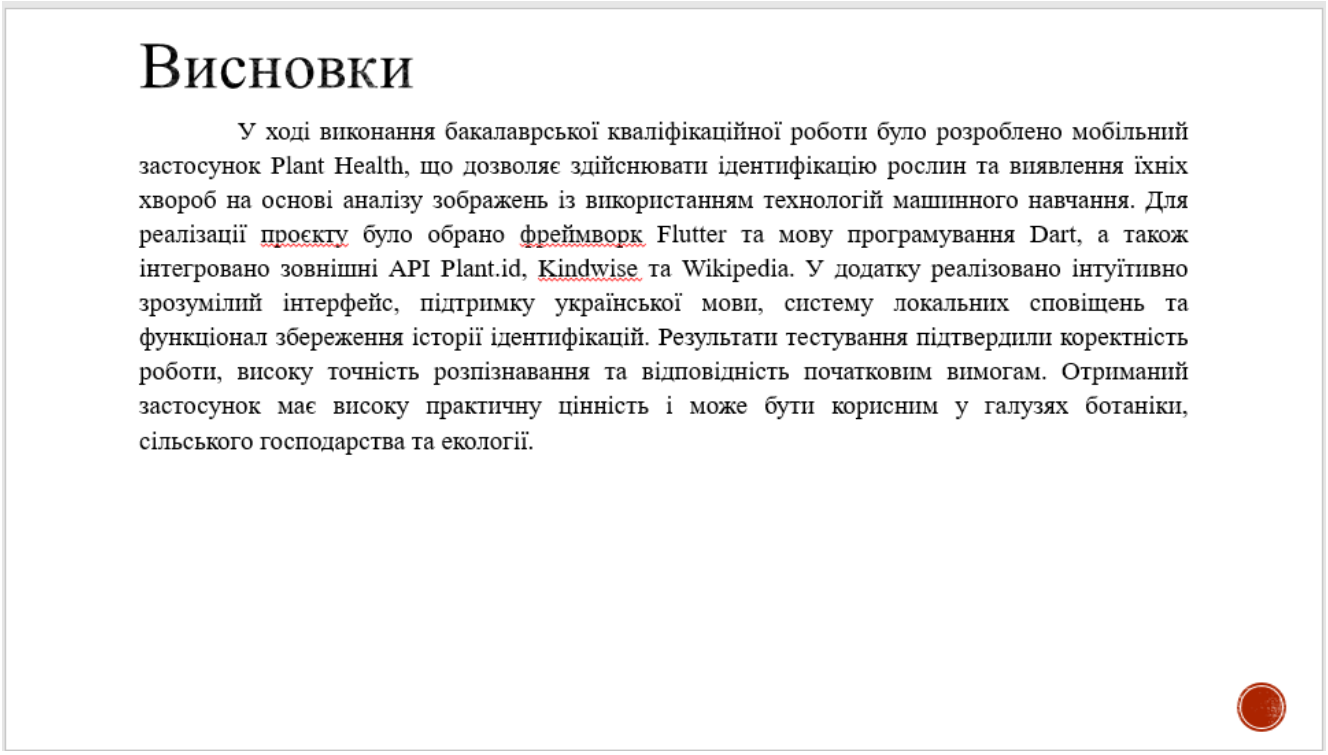

Рисунок Г.11 – Висновки

Рисунок Г.12 – Дякую за увагу