Egyptian E-Learning University

Faculty of Computers & Information Technology

Planty Care

By

Farah Mohamed Ahmed	2100686
Kholoud Mohsen Hussein	2100882
Mariam Hany ElSayed	2101011
Hager Abdelkader Saleh	2102131
George Hany Milad	2100724
Eslam Ayman Shehata	2101854
Bishoy Ezzat Hanna	2100584

Supervised by

Dr. Mahmoud Bassiouni

Assistant

Eng. Toka Ashraf

[Fayoum]-2025

Abstract

Diagnosing plant diseases early is crucial for healthy crop growth, yet it's often a slow and difficult process—especially for small farmers without expert knowledge. Planty Care is a smart mobile application designed to make this task easier and faster by using deep learning to detect plant diseases from leaf images and offer treatment suggestions.

The app allows users to simply take a photo of a diseased leaf. Using a Convolutional Neural Network (CNN) model trained on a dataset of labelled plant images, the system identifies the disease and provides a brief description along with recommended treatment—all through an active internet connection. In addition to diagnosis, the app also suggests suitable crops based on environmental conditions and recommends fertilizers to improve plant health and yield.

The model performed well in tests, showing high accuracy in recognizing common plant diseases. One of the key outcomes of the project is a smooth and user-friendly mobile interface that connects users to accurate, AI-powered diagnosis and practical agricultural advice in real time.

By combining artificial intelligence, image processing, and agricultural insight, Planty Care offers a modern, accessible tool to support farmers in making informed decisions and improving crop outcomes

Acknowledgments

We would like to express our sincere gratitude to our esteemed supervisor, Dr. Mahmoud Bassiouni, for his exceptional guidance, continuous encouragement, and unwavering support throughout the course of this project. His valuable insights and inspiring direction have greatly shaped our academic journey and prepared us to transition confidently and competently from university life to the professional world.

We also extend our heartfelt thanks to the College of Information Technology for their ongoing support and for providing all the necessary resources and environment that contributed to the success of this project.

We would like to express deep appreciation to our dedicated teaching assistant, who served as a true mentor, guiding us with clarity and playing a significant role in the successful completion of the first phase of our graduation project.

Finally, we thank Almighty God, our families, our team members, and everyone who offered support and encouragement throughout this

stage—whether from within academia or beyond. Every word of support and contribution, no matter how small, has made a meaningful difference in achieving this accomplishment.

TIME PLAN

Time plan				
	Duration	start	finish	Rescouce Name
Graduation Project	246	05/10/2024	11/06/2025	
Search	8	05/10/2024	13/10/2024	All Team Members
Business Requirement	23			
Tools installaion	1	14/10/2024	15/10/2024	George
fields study	60	16/10/2024	16/11/2024	All Team Members
System analysis	21			
Use Case	21	17/11/2024	08/12/2024	Farah
Use Case Senario	21	17/11/2024	08/12/2024	Mariam
DFD	21	17/11/2024	08/12/2024	Eslam
Design	30			
UI(Figma)	30	17/11/2024	17/12/2024	George
Sequence	30	17/11/2024	15/12/2024	George , Hager
Class digram	30	17/11/2024	15/12/2024	Bishoy
System Architecturet	12	09/05/2025	21/05/2025	Hager
Programming	163			
Frontend implementation	42	18/12/2024	01/02/2025	Bishoy
Backend implementation	90	09/02/2025	09/05/2025	Bishoy
AI implementation	121	18/12/2024	19/04/2025	Eslam , Mariam, Farah
AI API Integration	20	20/04/2025	09/05/2025	Kholod , Hager
Testing	10			
Frontend Testing	10	09/05/2025	19/05/2025	George , Bishoy
Backend Testing	10	09/05/2025	19/05/2025	Kholod , Hager,Bishoy
AI Testing	10	09/05/2025	19/05/2025	Eslam , Mariam, Farah
Ducmentation	198	27/12/2024	08/06/2025	Farah
Meetings (review)	3	09/06/2025	10/06/2025	All Team Members

LIST OF FIGURES

Contents

Chapter 1

Introduction

1.1 Introduction

Agriculture has always been the backbone of food production, but farmers constantly face challenges that threaten their crops. One of the biggest concerns is plant diseases, which can spread quickly and lead to major losses if not identified and treated in time. In the past, diagnosing these diseases relied heavily on personal experience or consulting agricultural experts—a process that was often time-consuming and, at times, inaccurate.

Today, technology is transforming the way farmers protect their crops. A smart plant disease detection system is designed to help farmers and agricultural professionals detect diseases simply by analysing images of plant leaves. This system not only identifies the type of plant but also provides a brief description and recommended treatments, offering a fast and reliable solution that saves time and resources.

What makes this system particularly valuable is its accessibility. Farmers no longer need to wait for expert advice or risk misdiagnosing a problem. By using artificial intelligence, the system can quickly scan and analyse a leaf, detect potential diseases, and suggest effective solutions—all within seconds.

In addition to disease detection, the system includes a crop recommendation feature that suggests the most suitable crops based on

current environmental conditions. It also offers fertilizer recommendations tailored to the soil and plant needs, helping optimize plant health and yield.

Beyond just disease detection, such technology has the potential to revolutionize the agricultural industry. It enables better decision-making,

reduces crop losses, and supports sustainable farming practices by ensuring that plants receive the right treatment at the right time.

As we explore this system further, we will look at how it works, the challenges it helps overcome, and the broader impact of technology in modern agriculture.

1.2 Background and motivation.

Agriculture is a vital sector worldwide, providing food security and livelihoods for billions of people. However, farmers face numerous challenges, among which plant diseases pose a significant threat to crop productivity and quality. Traditional disease diagnosis methods heavily rely on manual expertise and direct inspection, processes that are often time-consuming, costly, and prone to human error. Therefore, there is an urgent need for accurate, fast, and accessible solutions for all farmers, especially those in remote or resource-limited areas.

Rapid advancements in artificial intelligence and image processing technologies open new opportunities to address these challenges. Deep learning models, particularly Convolutional Neural Networks (CNNs), have demonstrated remarkable success in disease diagnosis from images. By leveraging these technologies, it is possible to automate plant disease recognition through leaf images, enabling faster and more accurate diagnosis.

The motivation for this project stems from the urgent need to support farmers in effectively managing crop health. The project aims to develop a smart system that not only detects diseases but also provides appropriate treatment recommendations. Furthermore, by integrating a crop and fertilizer recommendation system based on environmental conditions, the system seeks

to promote sustainable farming practices and improve overall crop productivity.

This study aims to bridge the gap between advanced artificial intelligence technologies and practical agricultural applications, empowering farmers with tools that reduce losses, save time, and enhance crop health.

1.3 Importance of the problem being addressed

Plant diseases pose a significant threat to global agriculture, causing major crop losses and economic damage every year. These diseases can spread rapidly across fields and regions, reducing both the quantity and quality of harvests. For many farmers, especially smallholders in developing countries, such losses directly impact their income and food security.

Traditional methods of diagnosing plant diseases rely heavily on manual expertise and direct inspection, which are often inaccessible to many farmers due to lack of resources, remote locations, or limited agricultural extension services. This leads to delays in diagnosis and treatment, allowing diseases to worsen and spread.

Addressing this problem with modern technology is crucial to improving agricultural productivity and sustainability. In this context, a smart Android application has been developed to enable farmers to quickly and accurately diagnose plant diseases by capturing and analyzing images of plant leaves using artificial intelligence techniques. This app provides an easy and accessible way to receive timely diagnosis and treatment recommendations, helping reduce crop losses and lower costs.

Moreover, integrating crop and fertilizer recommendations supports sustainable farming practices by optimizing resource use and promoting

healthy crop growth. Overall, solving this problem contributes to food security, farmer income stability, and environmental protection

1.4 Problem Statement

1.4.1 Clear Definition of the Problem:

Plant diseases are a major threat to agricultural productivity worldwide. Farmers often face difficulties in accurately and promptly identifying these diseases due to limited access to expert knowledge and diagnostic tools. This results in delayed or incorrect treatment, which exacerbates crop damage and leads to significant economic losses.

1.4.2 Justification for Solving the Problem

Solving this problem is crucial because early and accurate disease detection directly impacts food security, farmer income, and sustainable agriculture. Providing an accessible, fast, and reliable diagnostic solution empowers farmers to take timely action, reduce crop losses, and optimize the use of resources. Integrating crop and

fertilizer recommendations further supports sustainable farming, promoting healthier crops and improving yields. Thus, addressing this issue contributes to economic stability, environmental protection, and improved agricultural productivity.

1.5 Objectives

1.5.1 Main Objective:

To develop an AI-powered mobile application, Planty Care, that helps farmers and gardeners accurately detect plant diseases from leaf images and provide effective treatment and agricultural recommendations to improve crop health and productivity.

1.5.2 Specific Objectives:

1. Time Optimization:
    - Enable instant disease detection by analysing photos of plant leaves using AI-driven image processing.

- Provide fast and reliable diagnoses to prevent disease spread through timely intervention.

2. Automation and Error Reduction:

   - Utilize a comprehensive plant disease database for accurate symptom comparison and diagnosis.
   - Minimize human errors and reduce misdiagnosis to recommend the most suitable treatments.

3. Conflict Resolution in Diagnosis:

   - Address symptom similarities among different diseases by offering multiple possible diagnoses with confidence scores.
   - Assist users in selecting appropriate treatments and provide preventive care tips

1.6 Brief Overview of the Proposed Solution

The proposed solution is an AI-powered Android application called Planty Care designed to assist farmers and gardeners in quickly diagnosing plant

diseases through images of affected leaves. The app utilizes advanced Convolutional Neural Networks (CNN) to analyse the images and accurately identify the disease type. Once diagnosed, it provides users with a brief description of the disease and recommends effective treatment options.

Beyond disease detection, the app also offers crop recommendations and fertilizer suggestions tailored to environmental conditions, helping optimize plant health and yield. The user-friendly interface allows farmers to capture and upload leaf photos easily, receiving near-instant results that facilitate timely intervention.

This solution aims to overcome traditional challenges by providing a fast, accessible, and reliable diagnostic tool that minimizes reliance on expert consultations, reduces misdiagnosis, and promotes sustainable agricultural practices through integrated recommendations.

1.7 Software process model

The Agile Model stands as the counterpart to the Waterfall Model in software development. It adopts an iterative and flexible approach where the project is divided into smaller cycles, referred to as sprints or iterations. Each sprint encompasses the phases of planning, designing, developing, and testing, allowing for incremental progress and frequent delivery of functional components. Unlike the sequential structure of the Waterfall Model, Agile embraces adaptability by enabling changes to requirements at any stage of development. This model emphasizes continuous collaboration among stakeholders, the development team, and customers, ensuring that the software evolves in alignment with the users' needs. The Agile methodology is characterized by iterative progress, flexibility in handling evolving requirements, and quick responses to changes, making it particularly effective in dynamic and fast-paced environments. Examples of Agile methodologies include Scrum, which focuses on short, time-boxed sprints; Kanban, which visualizes tasks on a board to enhance workflow; and Extreme Programming (XP), which promotes rapid releases with frequent testing and feedback.

The Agile Model stands in contrast to the rigid, linear progression of the Waterfall Model, offering a dynamic framework for software development that prioritizes collaboration, adaptability, and continuous delivery.

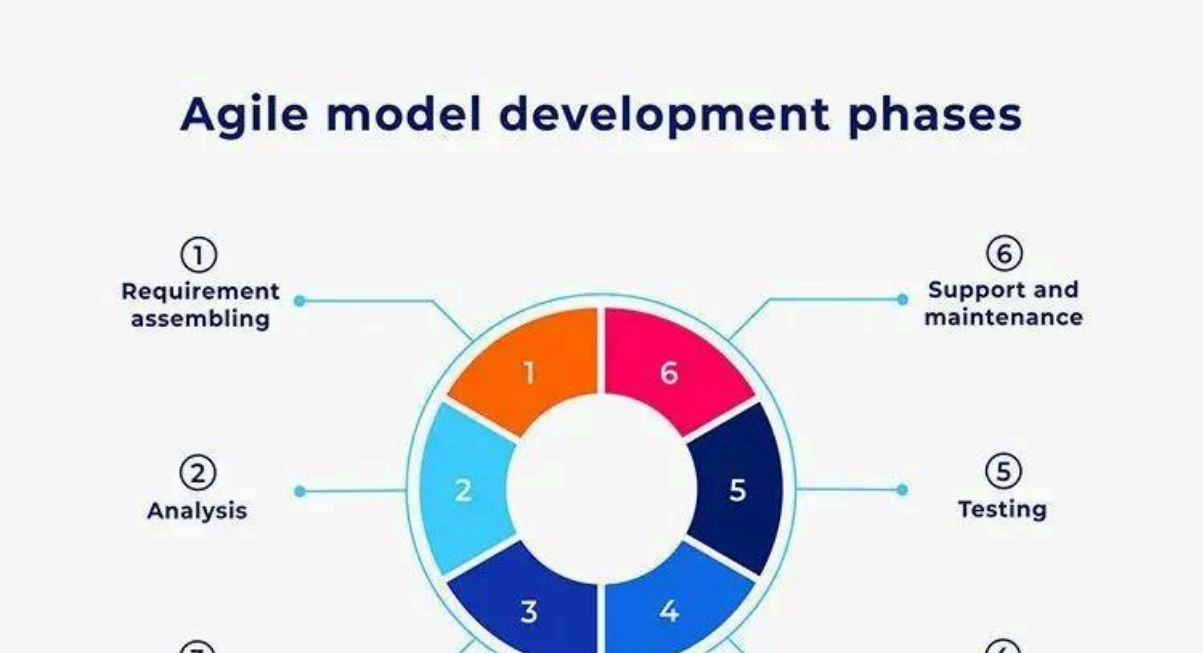


Figure 1.1 (Agile model)

Chapter 2

Literature Review / Related Work

2.1 Summary of Existing Research and Technologies

In recent years, several artificial intelligence and deep learning-based models have been developed to accurately and efficiently detect and diagnose plant diseases. Below is a summary of three key studies that are directly relevant to our project concept

2.1.1 Plant Disease Detection Using Deep Learning Models (Konduru et al., 2023)

The research by Konduru et al. (2023) highlights the use of deep learning models, such as MobileNet, for plant disease detection. Applications inspired by this paper often rely on CNN-based models to detect diseases in plants by analyzing images of leaves. These apps are designed to assist farmers and plant enthusiasts in diagnosing plant health issues and providing treatment suggestions based on image input.

While not an app itself, the findings of this paper are implemented in various existing applications that use MobileNet or similar models to detect diseases like powdery mildew, rust, and other common plant ailments. The high accuracy of these models (MobileNet achieved a training accuracy of 99.69%) makes these apps highly reliable for plant disease detection.

2.1.2 Deep Learning for Plant Disease Detection (Chowdhury et al., 2024)

Chowdhury et al. (2024) explore the use of models like InceptionV3 and EfficientNetB0 for plant disease detection. Apps inspired by this study typically use advanced deep learning models to identify diseases in crops. These models often focus on achieving high accuracy in recognizing a wide variety of plant diseases, making them suitable for real-time use by farmers.

Similar apps, based on research findings, often incorporate large datasets of plant images to detect diseases like bacterial wilt, blight, and more. Apps using EfficientNetB0, for example, have been shown to achieve impressive accuracy levels around 99.56%. These applications offer users an interface to upload images of plant leaves and receive disease diagnoses within seconds.

2.1.3 A Systematic Review of Machine Learning and Deep Learning Techniques for Plant Disease Detection (Islam et al., 2024)

Islam et al. (2024) provide an in-depth review of machine learning and deep learning techniques used in plant disease detection. While this paper itself is a review and does not directly present a new app, it highlights the potential of using models like CNN, VGG, and ResNet for plant disease classification. Many existing apps, such as Plantix and PlantSnap, are built upon these very techniques.

These apps often use a combination of machine learning models and databases to classify plant diseases, and they rely on user-uploaded photos for analysis. By using CNN and similar models, these apps can deliver accurate disease diagnoses, helping users to identify issues in their plants and take appropriate actions.

2.2 Gaps in Current Solutions

Despite the significant progress in plant disease detection using deep learning, existing solutions still present several limitations that *Planty Care* aims to overcome:

1. Limited Scope of Diagnosis

   Most current models focus solely on disease detection and classification, without offering practical treatment advice or contextual recommendations.

   Our solution provides both diagnosis and actionable treatment suggestions.

2. Lack of Fertilizer and Crop Recommendations

   Many applications ignore environmental factors and do not assist farmers with choosing suitable crops or fertilizers.

   Planty Care includes crop recommendation based on environmental input and soil type, as well as fertilizer suggestions.

3. No Mobile-First Lightweight Design

   Some models (e.g., large CNN architectures) are computationally heavy and unsuitable for real-time mobile applications.

   We designed our model to be lightweight and optimized for Android smartphones, ensuring fast, offline-friendly performance.

2.3 Requirement Analysis

Requirement analysis plays a crucial role in laying the groundwork for the entire software development lifecycle. It directly impacts the design, implementation, testing, and deployment stages. Conducting thorough and accurate requirement analysis helps minimize risks, avoid uncontrolled changes in project scope, and ensures that the final product meets both stakeholder expectations and real-world needs—ultimately contributing to the project's success.

2.3.1 Gathering Requirements

The first step in the requirement analysis process is gathering data from various stakeholders, including agricultural experts, farmers, and developers, to understand their needs and expectations for the application. This stage involves collecting information regarding:

- The types of diseases and plants the application should cover.

- The specific soil conditions that need to be supported for crop recommendations.
- The features and usability preferences of the end-users (farmers).
- Any constraints regarding system performance, security, and scalability.

This information will guide the development of the functional and non-functional requirements, ensuring that the application meets the real-world needs of its users.

2.3.2 Project Goals

The project aims to provide a comprehensive solution for plant disease detection, crop recommendation, and fertilizer suggestion. The primary goals are:

- Plant Disease Detection: To develop an AI-driven system that accurately identifies diseases based on leaf images.
- Crop Recommendations: To create a model that suggests the most suitable crops for a given soil type and environment.
- Fertilizer Recommendations: To develop a system that recommends the best fertilizers based on crop and soil analysis.
- User-Friendliness: To ensure the application is intuitive and accessible to farmers with varying levels of technical expertise.

- Real-Time Decision Making: To allow farmers to make immediate, informed decisions about disease management, crop selection, and fertilization.

2.3.3 Functional Requirements

The functional requirements outline the core features and functionality that the application must support. These include:

1. **Plant Disease Detection**:
    - The system must be able to process leaf images and classify them according to the disease.
    - It should provide treatment suggestions based on the detected disease.
2. **Crop Selection**:

- The system must recommend the most appropriate crops based on soil data and environmental factors.

3. **Fertilizer Recommendation**:
    - The system should suggest fertilizers suitable for specific crops and soil types, based on scientific data and research.
4. **User Interface (UI)**:
    - The UI should allow farmers to upload images, enter soil data, and view recommendations.
    - The UI should be easy to navigate and offer clear, actionable insights.
5. **Data Storage**:
    - The system should store historical data on disease detection, crop recommendations, and fertilizer usage for future analysis.

2.3.4 Non-Functional Requirements

These requirements focus on the performance, security, and reliability of the system. Key non-functional requirements include:

1. **Performance**:
    - The application should provide quick responses (preferably within seconds) for disease detection and recommendations.
2. **Scalability**:
    - The application should handle large amounts of data (such as multiple images and soil datasets) without performance degradation.
3. **Reliability**:
    - The system must be robust and error-free, ensuring continuous operation with minimal downtime.
4. **Security**:
    - Data security and privacy must be ensured, especially for sensitive agricultural data.
    - The application should include proper user authentication and data encryption.
5. **Usability**:
    - The app should be simple to use for non-technical users (farmers) with minimal training required.
    - It should be mobile-friendly and accessible on different devices.

2.3.5 User Requirements

User requirements focus on the needs of the end-users, specifically the farmers who will interact with the application. These requirements include:

1. **Ease of Use**:
    - The application should have a simple, intuitive interface, where users can upload images, enter soil information, and receive recommendations with minimal effort.
2. **Accurate Results**:
    - Users expect accurate disease detection and reliable crop/fertilizer recommendations to improve their productivity and decision-making.
3. **Localization**:
    - The application should support different languages to cater to a diverse audience of farmers in various regions.

2.4 Summary

Overall Problems of Existing Systems

- Limited Accuracy in Certain Conditions
- Narrow Scope of Disease Detection
- Dependence on High-Quality Images
- Lack of Personalization
- Limited Integration with Other Agricultural Tools
- User Experience Challenges

Overall Solution Approach

- Improving Disease Detection Accuracy
- Expanding Disease Detection Scope
- Personalizing Recommendations Based on Local Conditions
- Improving User Experience

Chapter 3

Proposed system

3.1 Approach used to solve the problem

To address the challenges farmers face in diagnosing plant diseases accurately and efficiently, we developed Planty Care, an AI-powered Android application that leverages deep learning and image processing techniques. Our approach started with building a well-structured dataset of plant leaf images from publicly available sources, including both healthy and diseased samples. The data was pre-processed through normalization, resizing, and augmentation techniques to improve model robustness and prevent overfitting.

We designed a Convolutional Neural Network (CNN) using PyTorch, tailored to classify diseases based on leaf patterns. The model was trained and evaluated using training, validation, and test splits, achieving high accuracy across all sets (Training Accuracy: 99.34%, Validation Accuracy: 99.04%, Test Accuracy: 96.30%). Once the model was finalized, it was exported and integrated into the mobile application using appropriate tools to ensure lightweight and efficient performance on Android devices.

The app allows users to take a picture of a plant leaf, which is then analysed by the AI model to detect the disease. Along with the diagnosis, the app provides a brief description of the disease and actionable treatment recommendations. In addition, Planty Care includes two extra modules:

- A crop recommendation system based on environmental and soil data.
- A fertilizer recommendation feature to enhance plant growth and sustainability.

This approach offers a real-time, accurate, and accessible tool that empowers farmers to manage plant health with confidence and minimal reliance on external agricultural support.

3.2 System architecture

The most essential and time-consuming aspect of a project is App design. In this phase, the information gathered before is used to complete the logical design of the information App. This includes designing user interfaces, databases, and outputs in collaboration with users to satisfy their information needs. The technical or implementation component of the App development project was the emphasis of information App design. The App analysis phase of an App development project is carried out independently. The requirements from the requirement analysis phase are transformed into technological solutions by the App designer. Software architecture, database design, and interface design are all factors in App design.

3.2.1 Use Case

A Use Case diagram helps identify the main components and actions that make up a App. In this diagram, the main components are known as "actors," and the actions are referred to as "use cases." It illustrates how actors interact with each use case. The Use Case diagram focuses on the functional aspects of a App, specifically capturing the business processes it performs. Additionally, these diagrams define the requirements of the modelled App, serving as a basis for creating test scripts for the App being modelled.

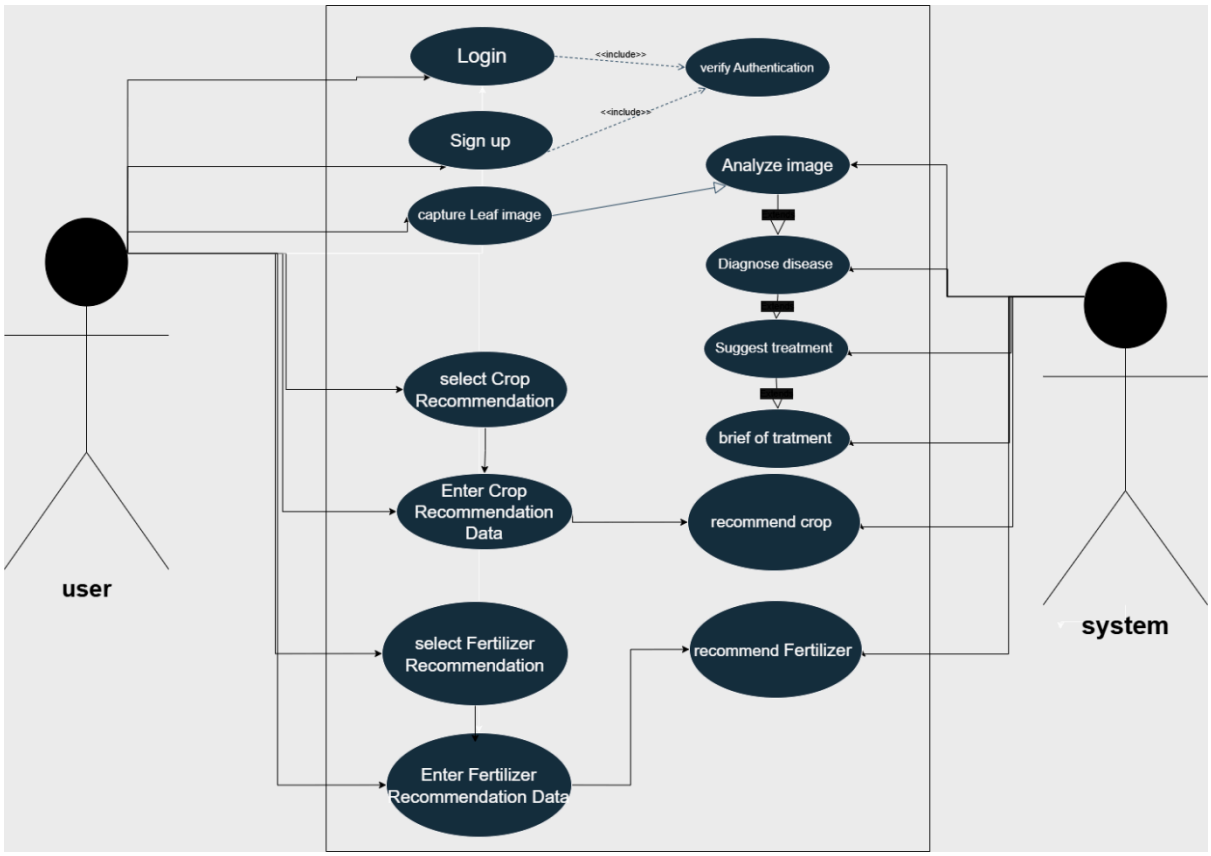


Figure 3.1 (use case diagram)

3.2.2 Use Case Scenario

Actors :

User

Application

System

Use Case Specification: User Login

Use Case ID	**UC-01**
Use Case Name	**User Login**
Actors	**User, Application**
Description	**Allows users to log in to access the app's features.**
Preconditions	**- The user must have a registered account.** **- The application must be connected to the authentication database.**
Normal Flow	**1. The user opens the app.** **2. The application displays the login screen.** **3. The user enters their email and password.** **4. The application verifies the credentials against stored user data** **5. If correct, the application redirects the user to the home page.**

Alternative Flow	**- A1: If the user enters incorrect credentials, the system displays an error message.** **- A2: If the user forgets the password, they can request a password reset.**
Postconditions	**- The user gains access to the application upon successful login.**
Non-Functional Requirements	**- Security: User credentials must be encrypted using industry-standard encryption.** **- Performance: The login process should not exceed 2 seconds.** **- Availability: The authentication service must be available 99.9% of the time.** **- Usability: The login screen must be user-friendly and support biometric authentication (if applicable).** **- Scalability: The application must handle up to 10,000 concurrent login requests efficiently.**

Use Case Specification: Diagnose Plant Disease and Provide Treatment

Use Case ID	UC-02
Use Case Name	Diagnose Plant Disease and Provide Treatment
Actors	User, Application, System (CNN Model)
Description	The user uploads an image of a plant, and the system diagnoses the disease and provides treatment recommendations.
Preconditions	The user must have a stable internet connection and a clear image of the plant. The application must be able to communicate with the CNN model.
Normal Flow	1. The user opens the application and selects "Diagnose Plant Disease". 2. The application prompts the user to upload an image. 3. The user uploads an image of the plant. 4. The application sends the image to the CNN model for analysis. 5. The CNN model processes the image and identifies the disease. 6. The application displays the diagnosis, including disease name. 7. The application retrieves treatment recommendations from the system's database.

	8. The application suggests preventive measures to avoid future infections. 9. The user reviews the diagnosis and treatment. 10. The use case ends when the user saves or exits the results.
Alternative Flow	**A1**: If the uploaded image is unclear, the application prompts the user to upload a clearer image. **A2**: If the disease cannot be identified, the application suggests sending the image for manual expert review.
Postconditions	The user receives a disease diagnosis with recommended treatment. The system logs the diagnosis data for future reference.
Non-Functional Requirements	**Accuracy:** The CNN model must have a minimum accuracy of 85% in disease detection. **Performance:** The system should process and return the diagnosis within 5 seconds. **Availability:** The disease diagnosis feature must be operational at least 99% of the time. **Usability:** The application should provide a simple and intuitive UI for image uploading and result display. **Security:** User-uploaded images must be stored securely and deleted after processing to maintain privacy. **Scalability:** The system should handle up to 1,000 diagnoses per hour without performance degradation.

Use Case Specification: Crop Recommendation Based on Soil and Weather Conditions

Use Case ID	**UC-03**
Use Case Name	**Crop Recommendation Based on Soil and Weather Conditions**
Actors	**User, Application, System (Recommendation Engine)**
Description	**The system recommends the best crop based on soil nutrients and weather conditions provided by the user.**
Preconditions	**The user must enter soil and weather parameters correctly. The application must be able to process the input data and communicate with the Recommendation Engine.**
Normal Flow	**1. The user opens the application.** **2. The user selects "Best Crop Recommendation".** **3. The application prompts the user to enter soil and weather**

	parameters (e.g., nitrogen, phosphorus, potassium levels, temperature, humidity, pH, and rainfall). 4. The user manually inputs the required values. 5. The application sends the input data to the Recommendation Engine for processing. 6. The Recommendation Engine analyzes the data and determines the best crop for the given conditions. 7. The application displays the single most suitable crop recommendation. 8. The user reviews the recommendation and may save it for future reference. 9. The use case ends when the user exits or proceeds with additional actions.
Alternative Flow	A1: If the user enters incomplete or invalid data, the application prompts an error message and asks for correct input. A2: If no suitable crop is found, the application notifies the user and suggests improving soil conditions.
Postconditions	The system provides a single recommended crop based on the input data, and the recommendation is logged for future reference.
Non-Functional Requirements	Accuracy: The recommendation model must have at least 90% accuracy in suggesting crops. Performance: The system should generate a recommendation within 3 seconds. Availability: The crop recommendation feature must be available at least 99% of the time. Usability: The application should provide clear input fields and validation checks to assist users in entering correct data.

	Security: User-submitted soil and weather data should be encrypted and stored securely. **Scalability: The system should support up to 5,000 crop recommendations per hour without performance issues.**

3.2.3 Data Flow Diagram

A Data Flow Diagram (DFD) is a visualization tool that illustrates the information flow within a system, mapping out processes, data stores, data flows, and external entities, and depicting their interactions. Widely employed in system analysis and design, DFDs facilitate a comprehensive understanding and documentation of information flow dynamics within a given system.

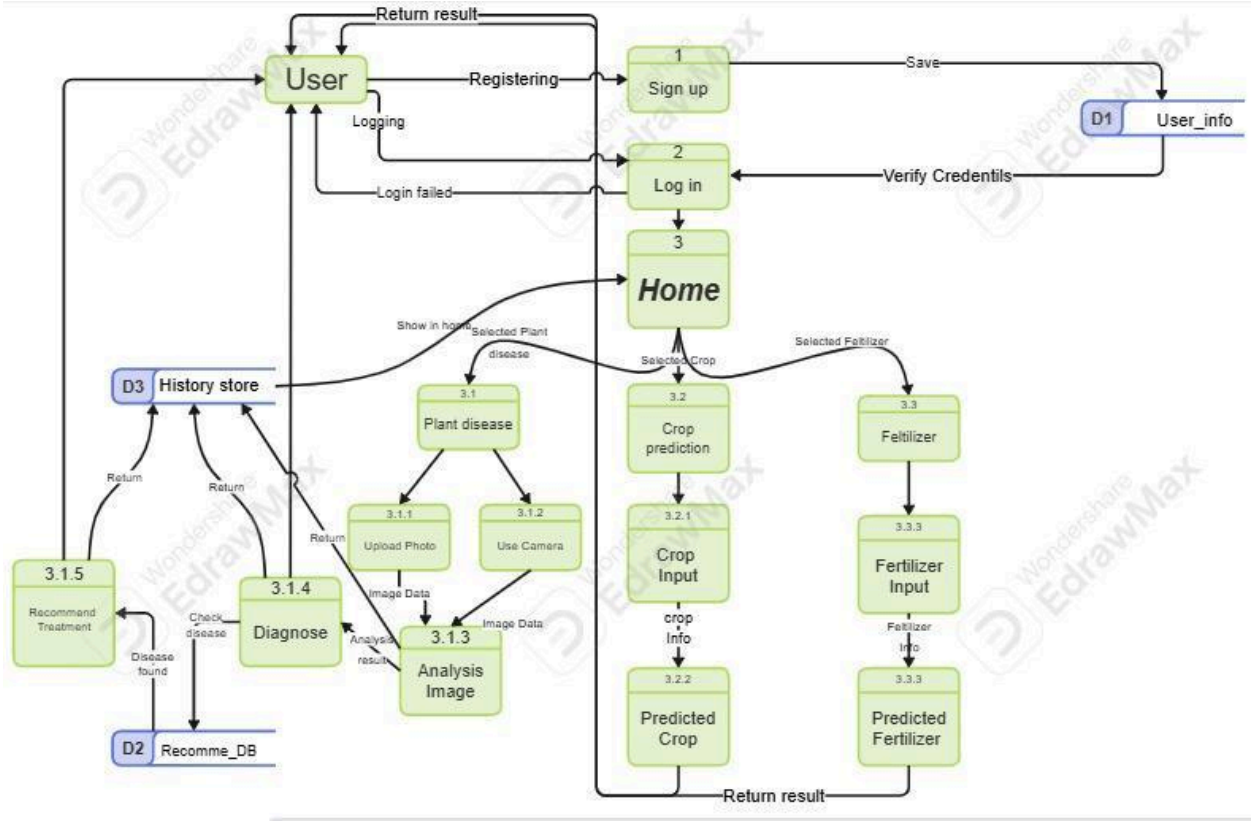


Figure 3.2 (DFD diagram)

3.2.3 Class Diagram

Class diagrams play a crucial role in object-oriented analysis and design. They visually represent the classes in a system, showcasing

their relationships, such as inheritance, aggregation, and association, along with the activities and attributes of these classes. These diagrams serve multiple purposes, ranging from conceptual/domain modeling to detailed design modeling, offering a comprehensive view of the system's structure and functionality.

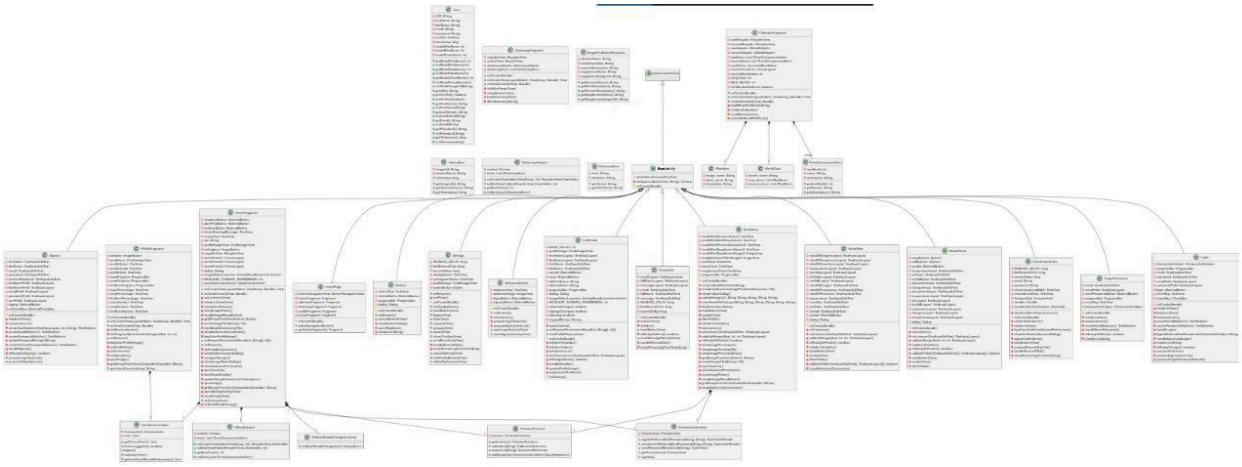


Figure 3.3 (class diagram)

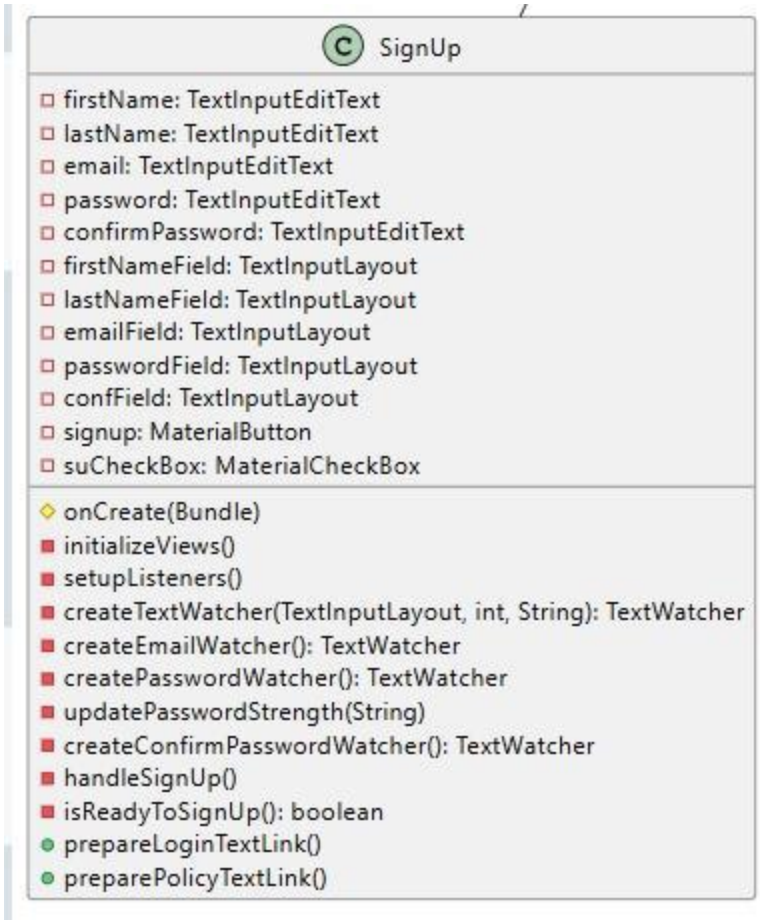


Figure 3.4 (SignUp class diagram)

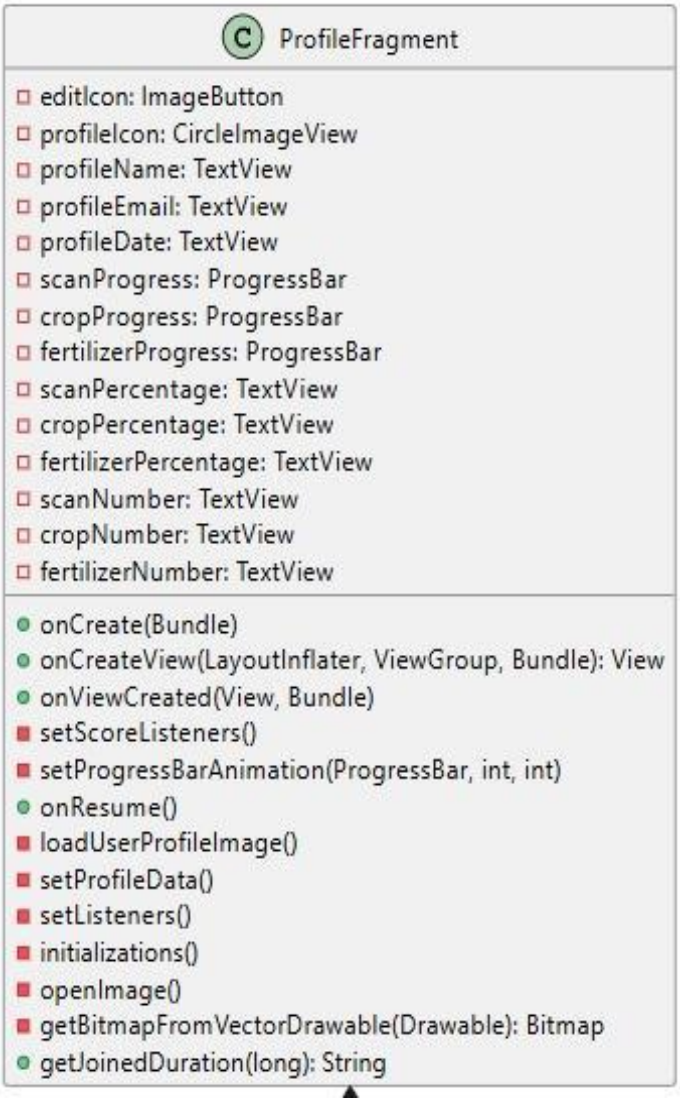


Figure 3.5 (Profile Fragment class diagram)

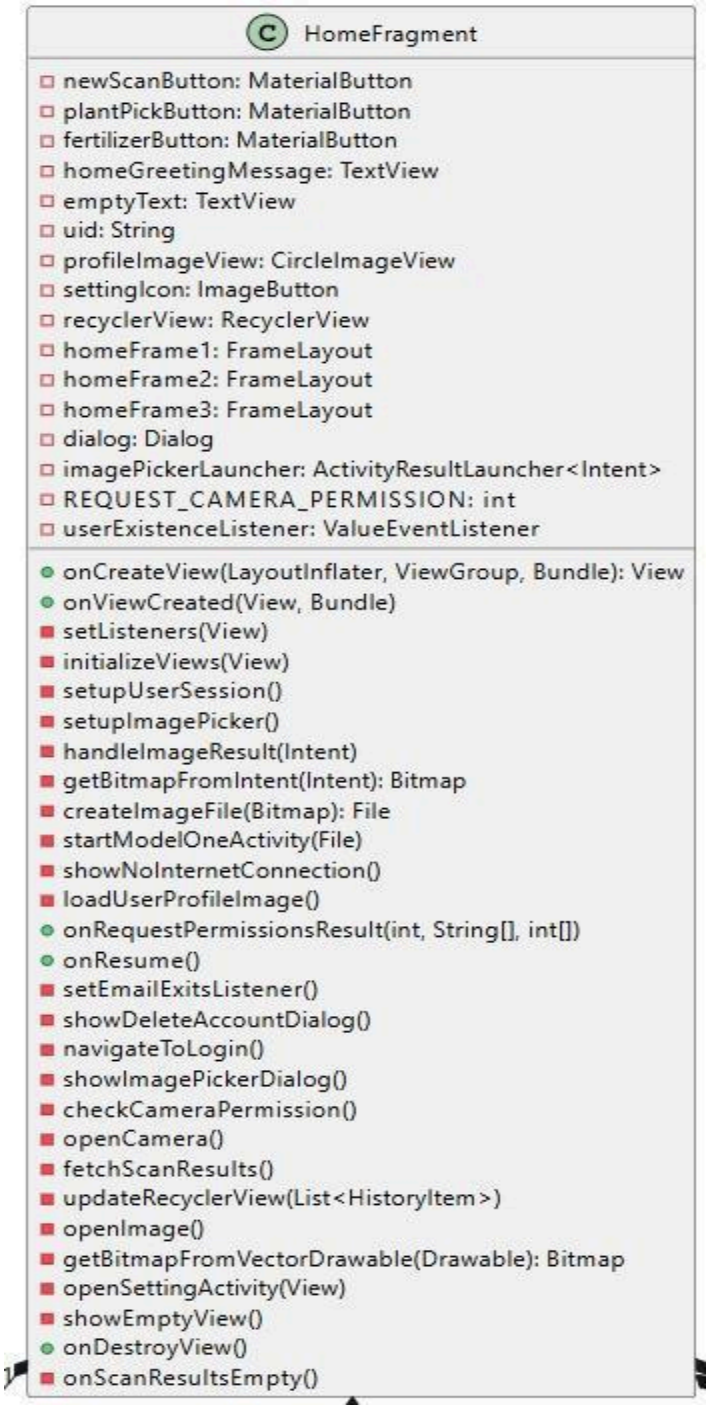


Figure 3.6 (Home Fragment class diagram)

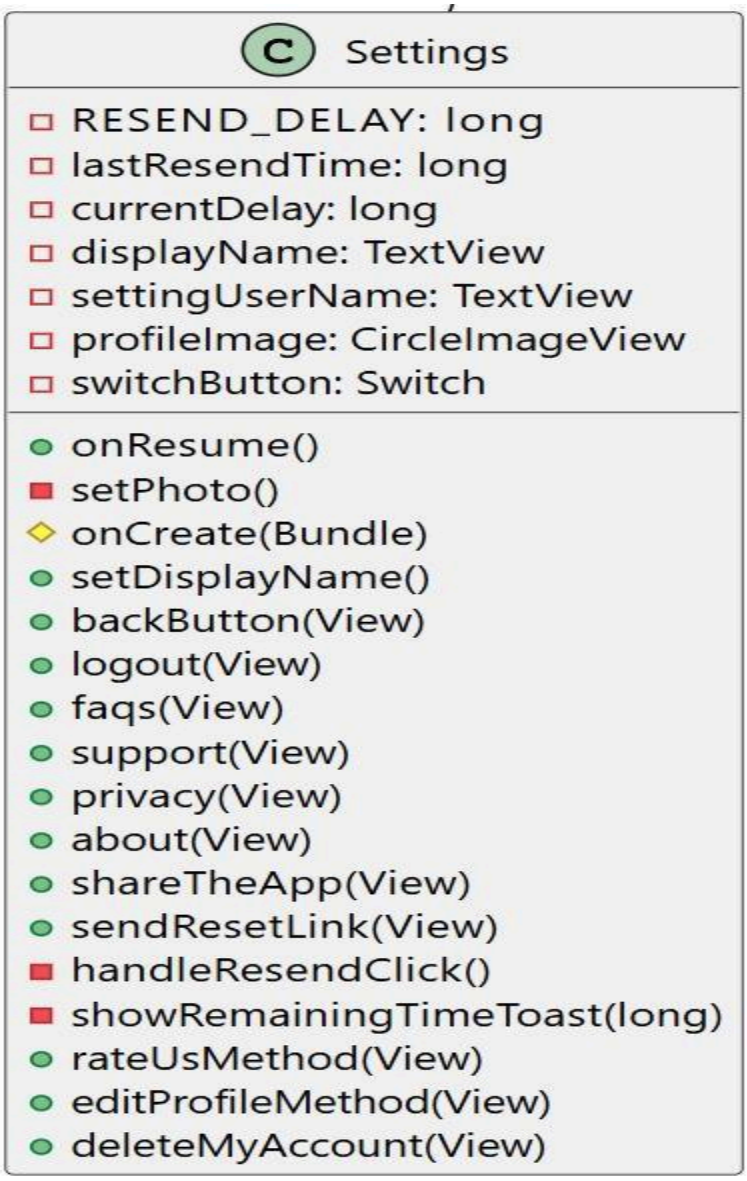


Figure 3.7 (Settings class diagram)

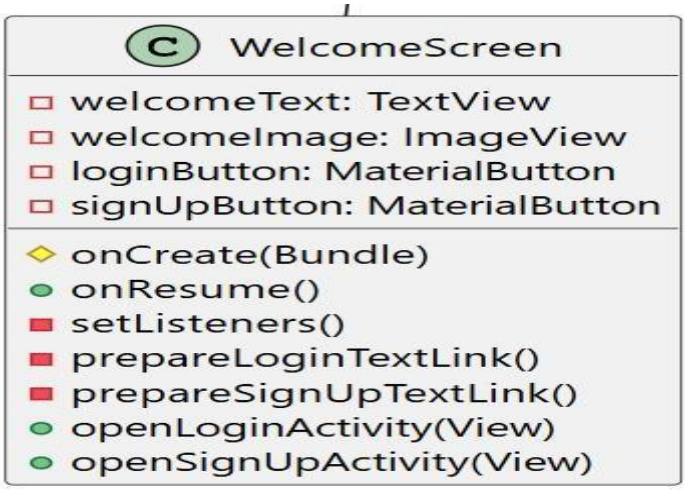


Figure 3.8 (Welcome screen class diagram)

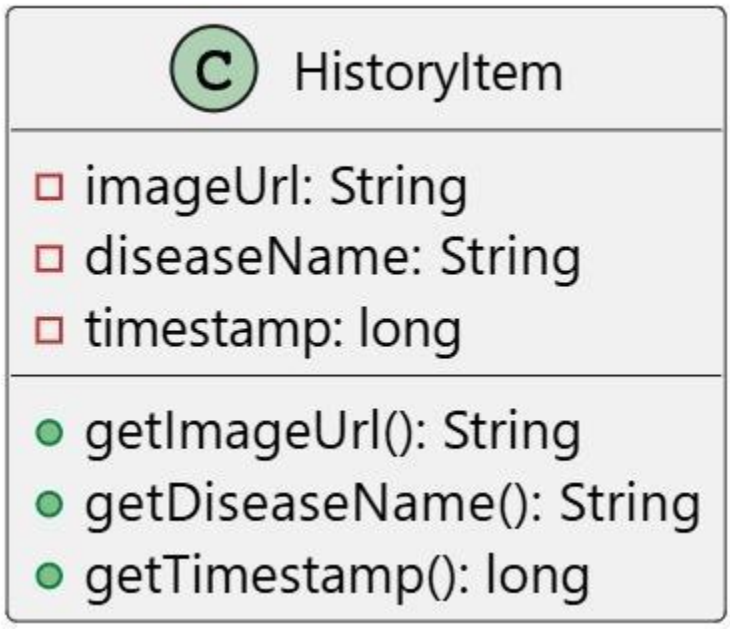


Figure 3.9 (History Item class diagram)

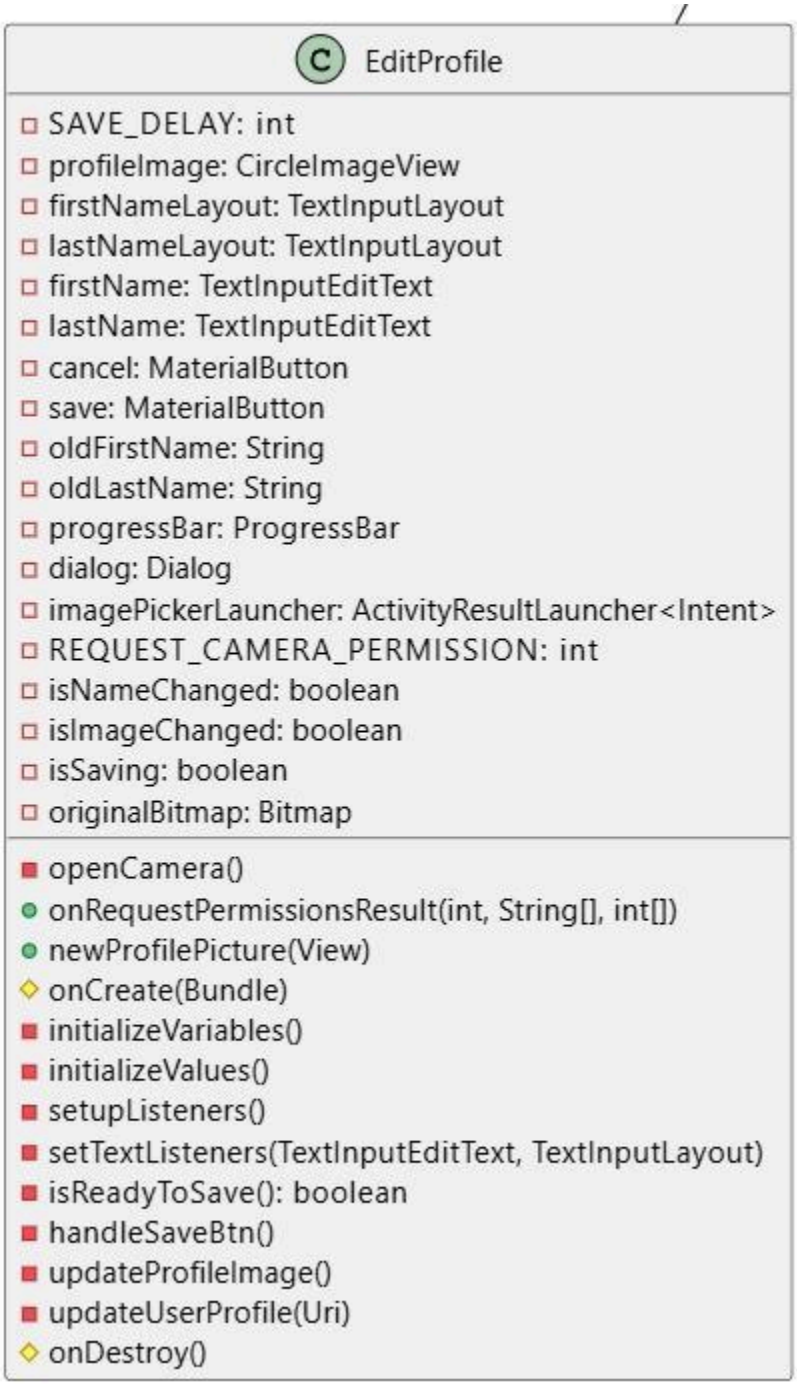

EditProfile
SAVE_DELAY: int
profileImage: CircleImageView
firstNameLayout: TextInputLayout
lastNameLayout: TextInputLayout
firstName: TextInputEditText
lastName: TextInputEditText
cancel: MaterialButton
save: MaterialButton
oldFirstName: String
oldLastName: String
progressBar: ProgressBar
dialog: Dialog
imagePickerLauncher: ActivityResultLauncher<Intent>
REQUEST_CAMERA_PERMISSION: int
isNameChanged: boolean
isImageChanged: boolean
isSaving: boolean
originalBitmap: Bitmap
openCamera()
onRequestPermissionsResult(int, String[], int[])
newProfilePicture(View)
onCreate(Bundle)
initializeVariables()
initializeValues()
setupListeners()
setTextListeners(TextInputEditText, TextInputLayout)
isReadyToSave(): boolean
handleSaveBtn()
updateProfileImage()
updateUserProfile(Uri)
onDestroy()

Figure 3.10 (Edit Profile class diagram)

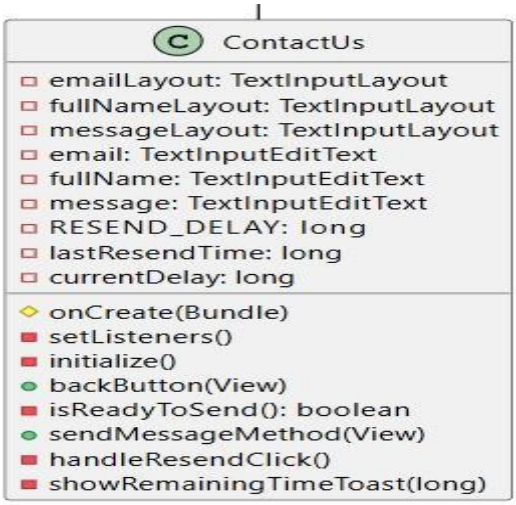


Figure 3.11 (Contact Us class diagram)

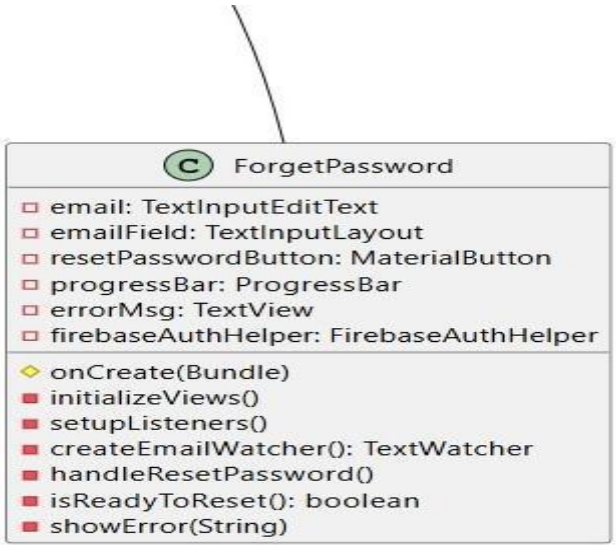


Figure 3.12 (Forget Password class diagram)

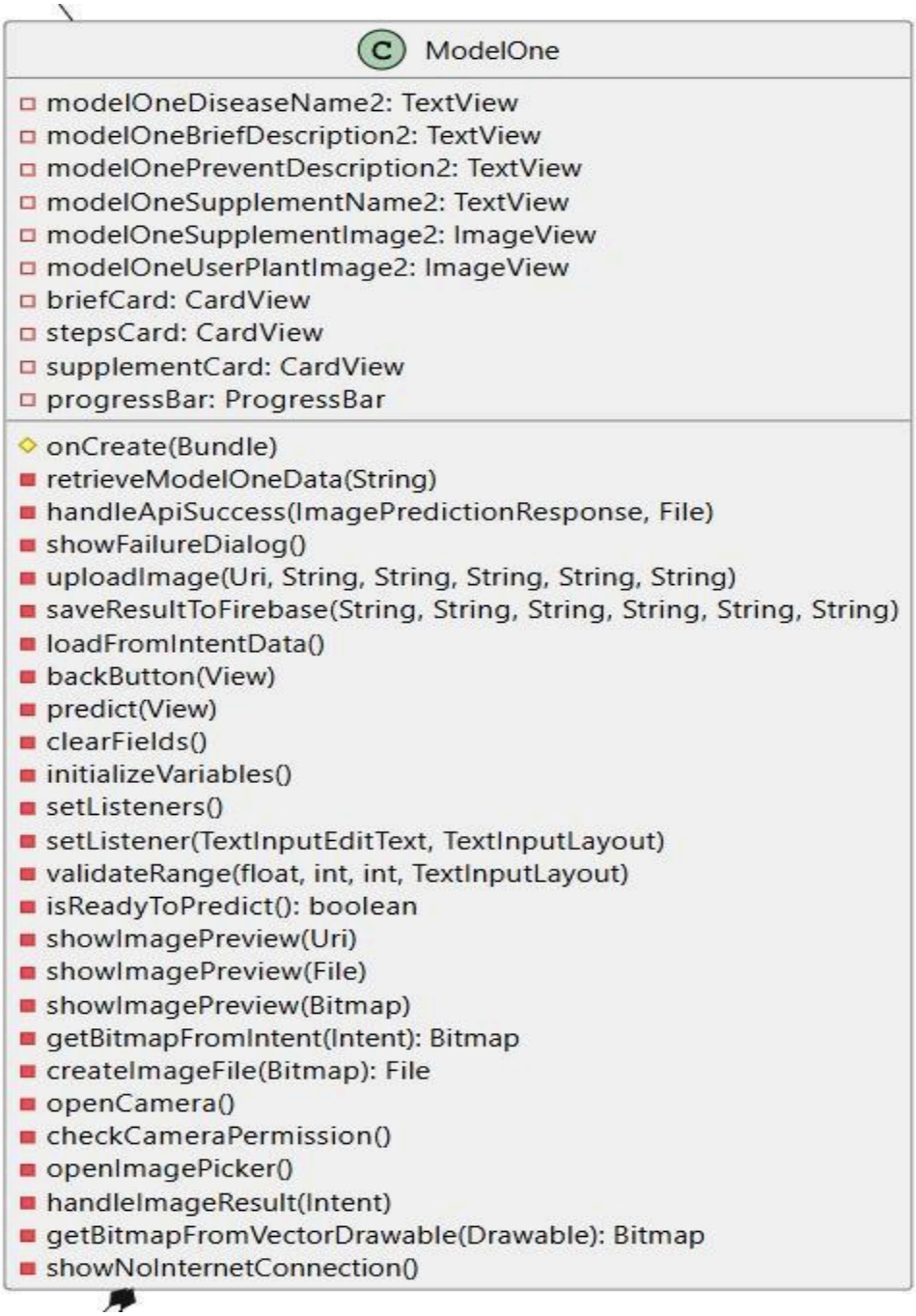


Figure 3.13 (Plant disease model class diagram)

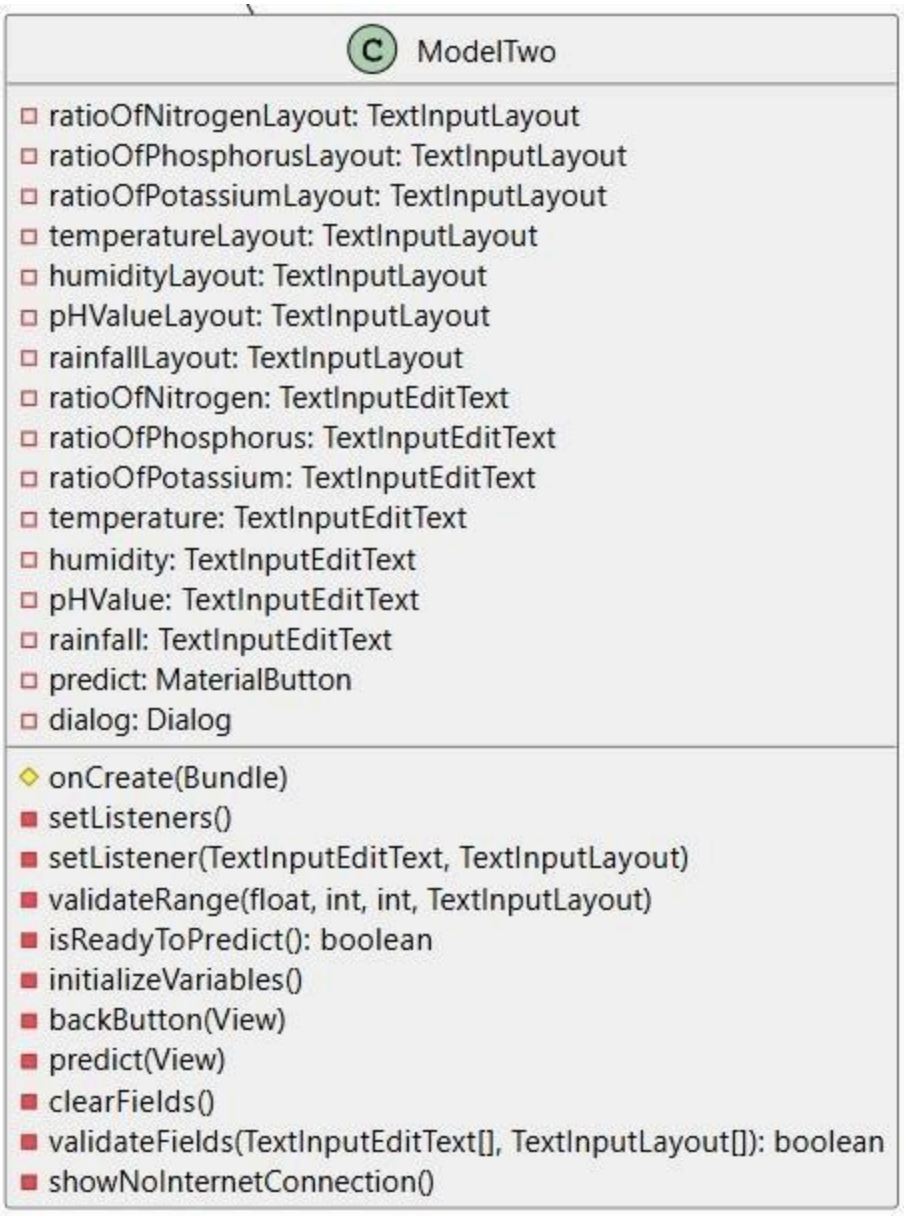


Figure 3.14 (Crop prediction model class diagram)

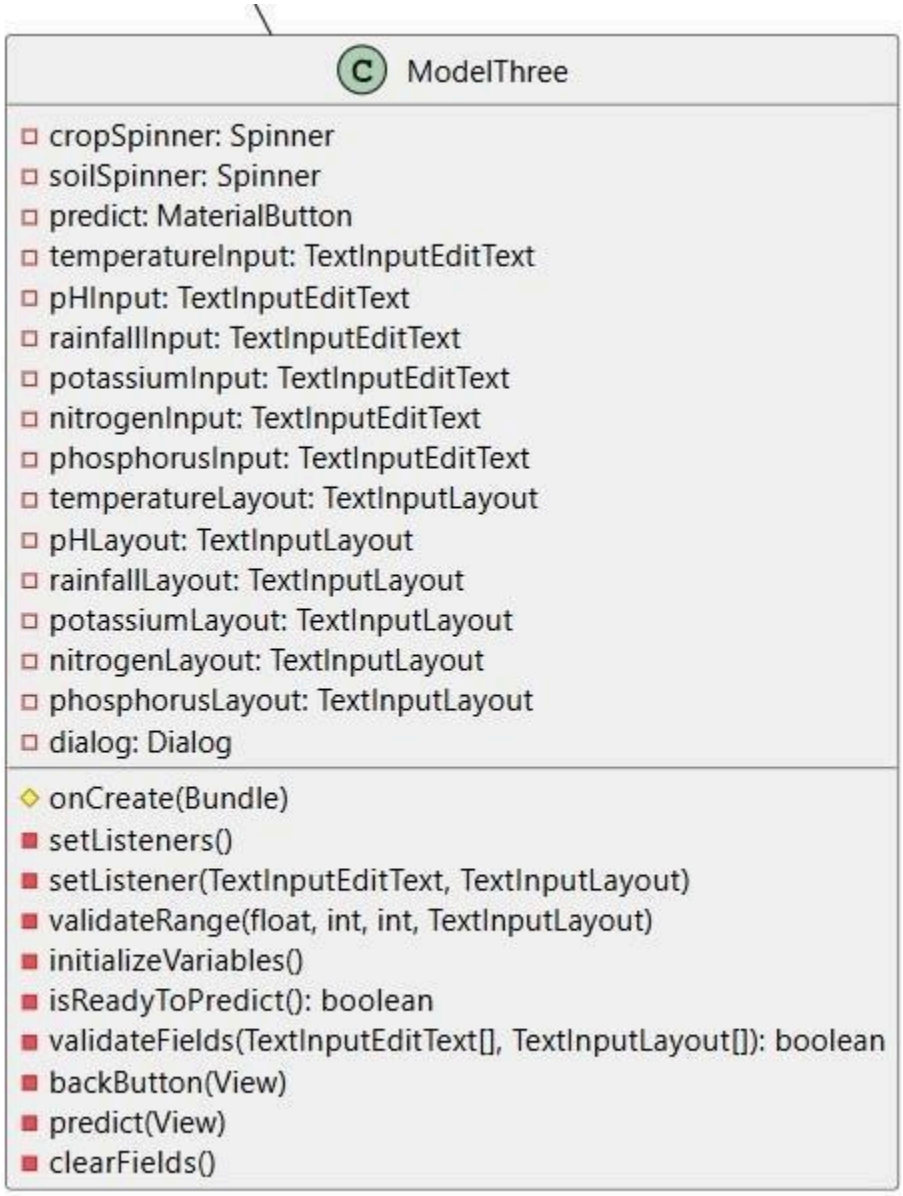


Figure 3.15 (Fertilizer prediction model class diagram)

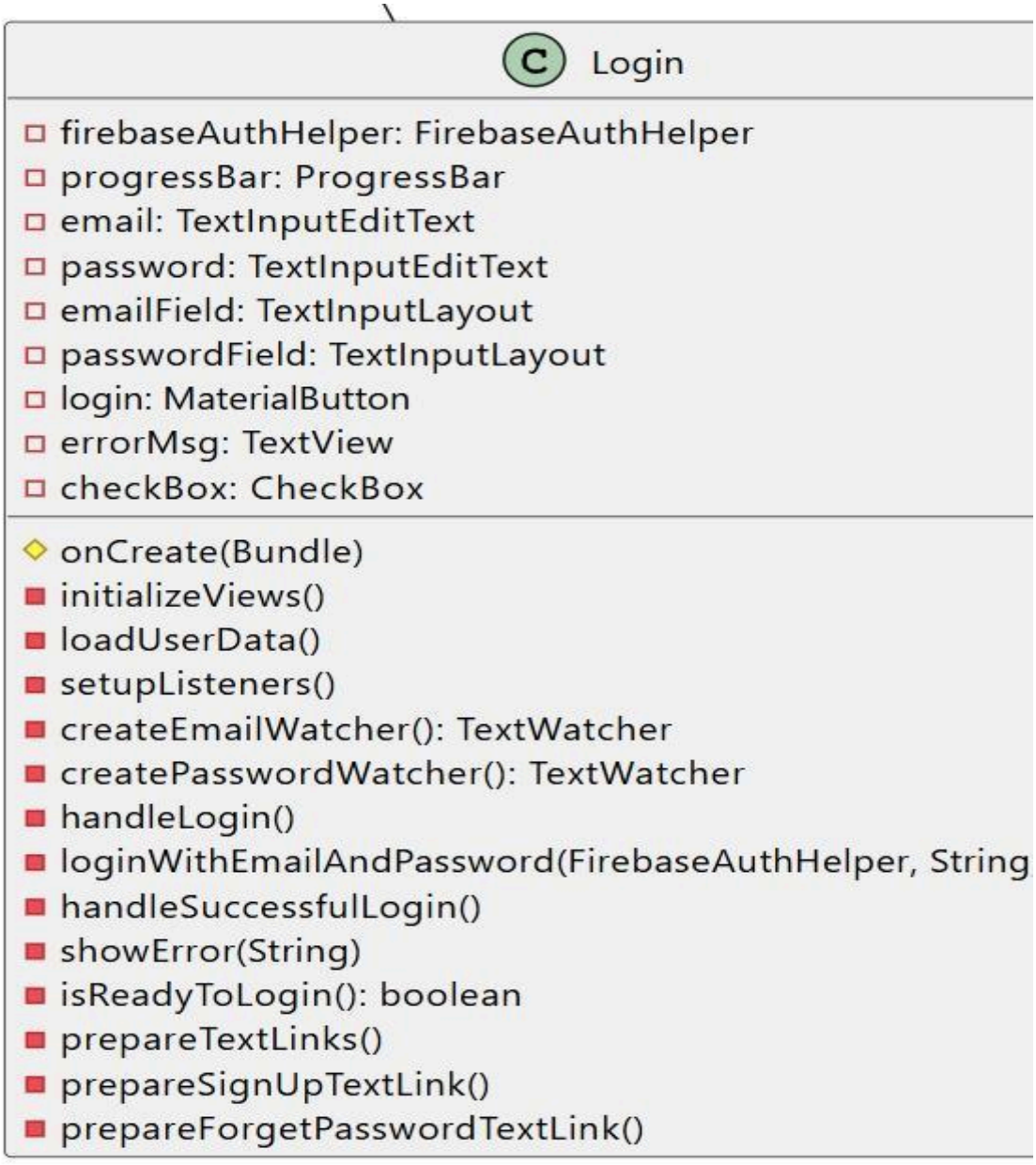


Figure 3.16 (Login class diagram)

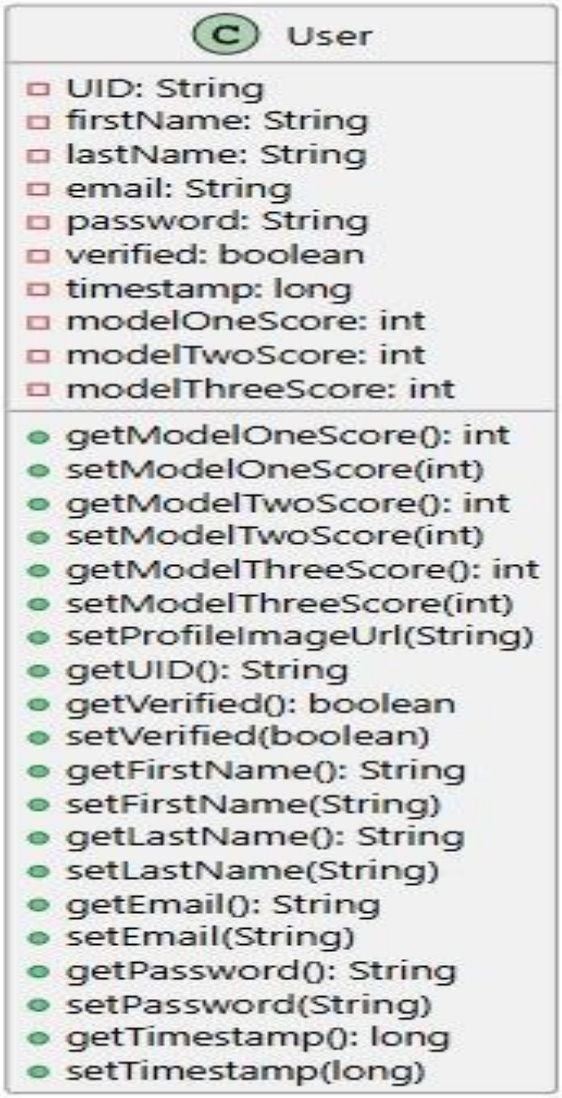


Figure 3.17 (User class diagram)

Figure 3.18 (Fire base AUTH Helper class diagram)

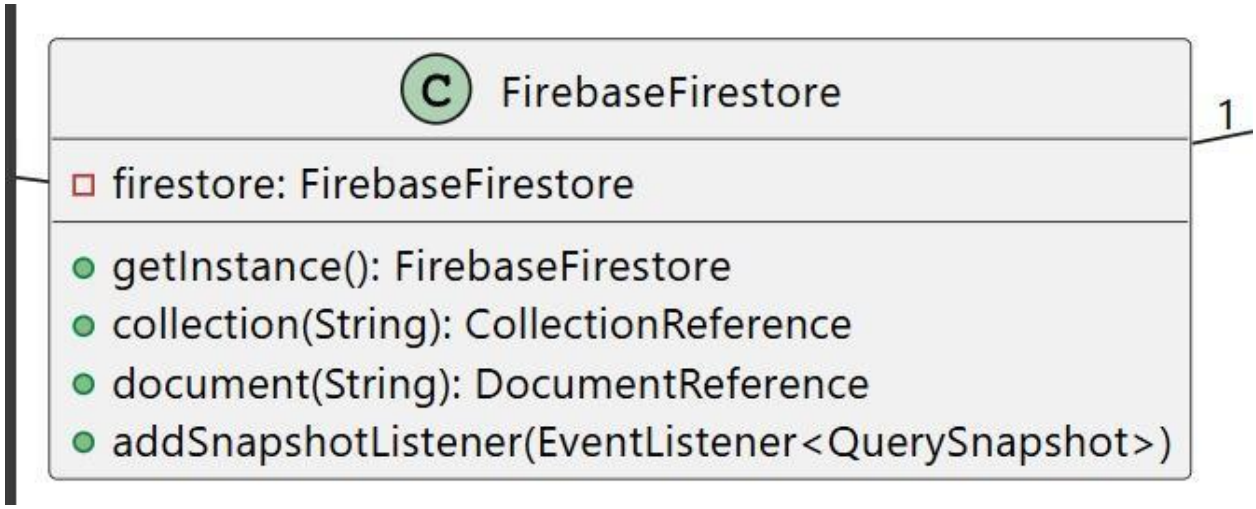


Figure 3.19 (Fire base Fire Store class diagram)

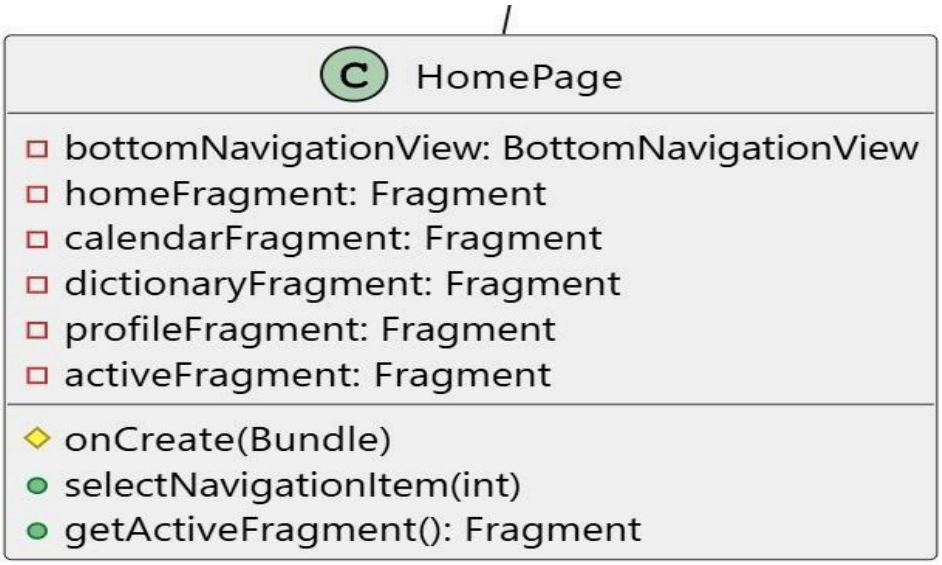


Figure 3.20 (Home Page class diagram)

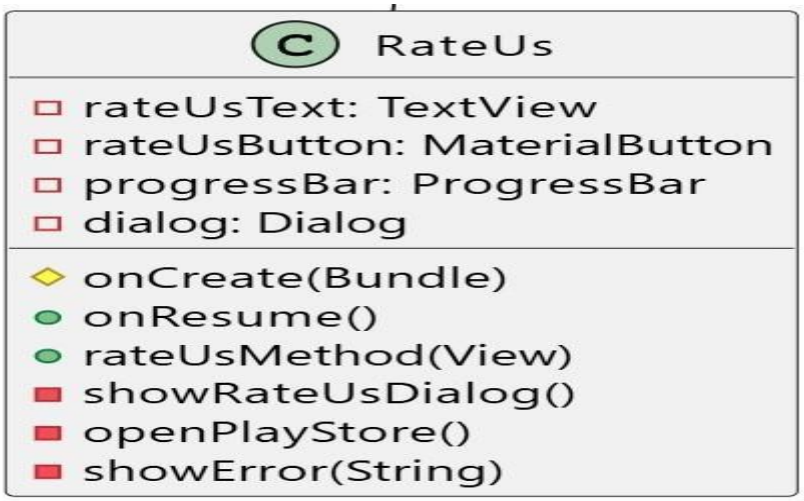


Figure 3.21 (Rate Us class diagram)

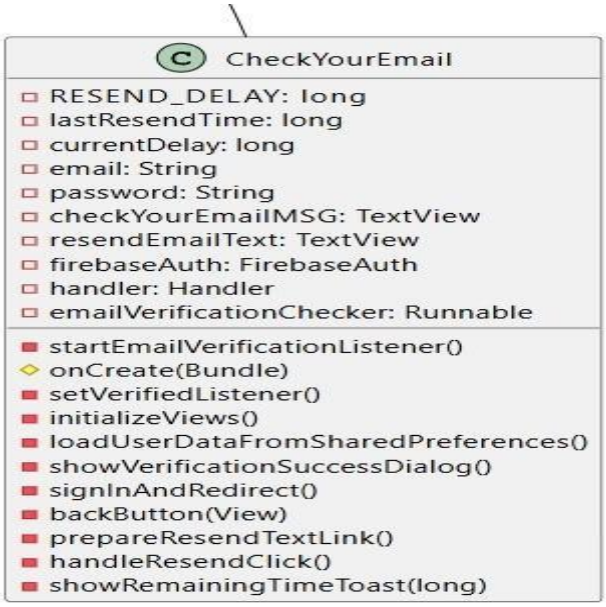


Figure 3.22 (Check Your Email class diagram)

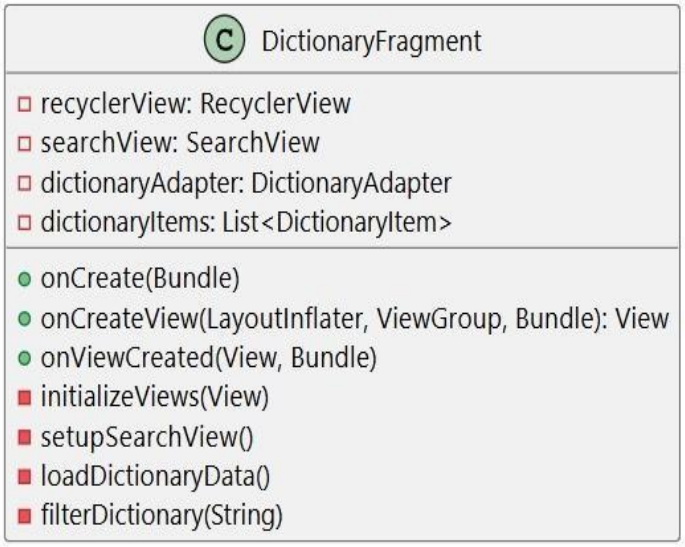


Figure 3.23 (Dictionary Fragment class diagram)

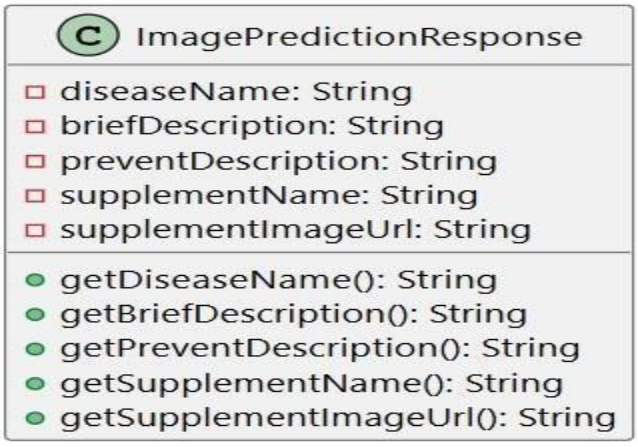


Figure 3.24 (Image prediction Response class diagram)

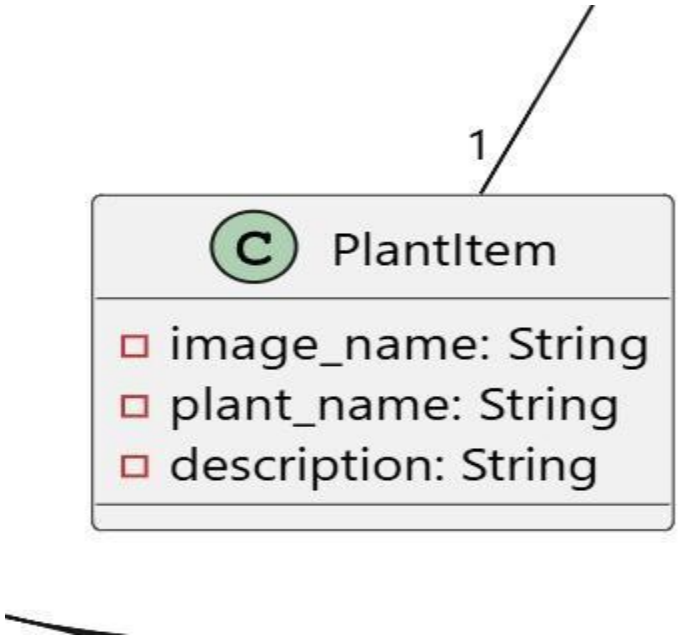


Figure 3.25 (Plant Item class diagram)

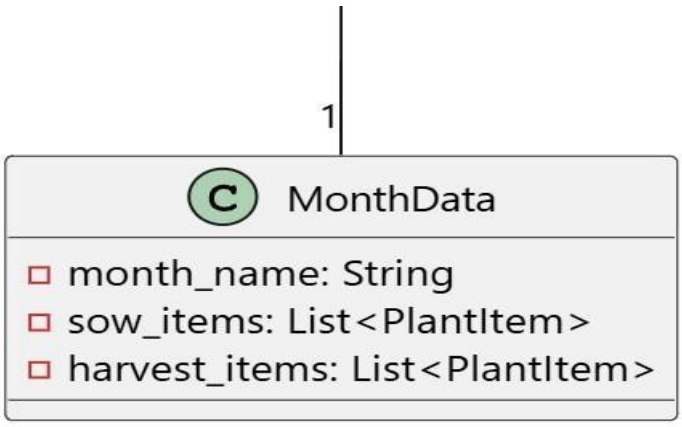


Figure 3.26 (Month Date class diagram)

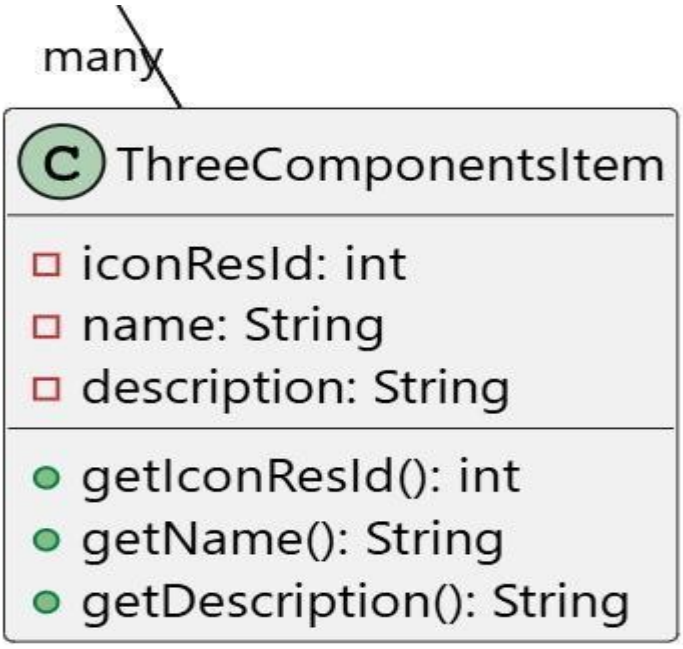


Figure 3.27 (Three Components Item class diagram)

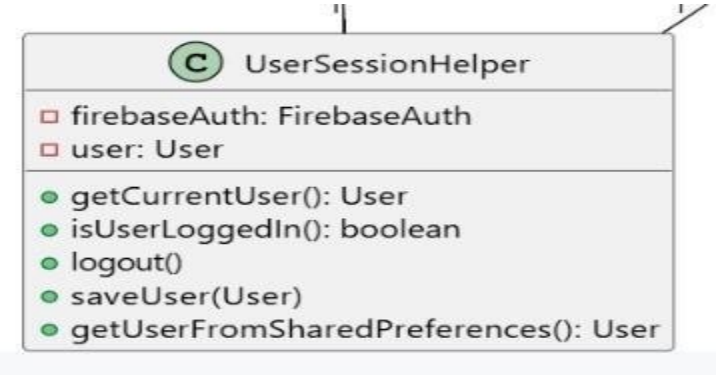


Figure 3.28 (User Session Helper class diagram)

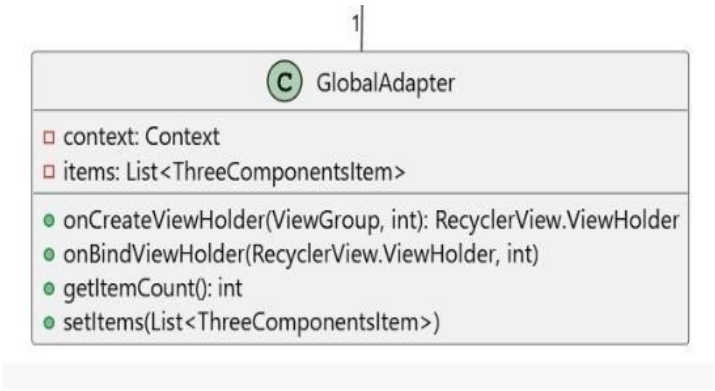


Figure 3.29 (Global Adapter class diagram)

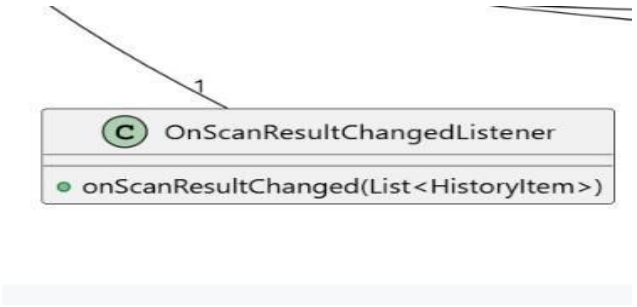


Figure 3.30 (On Scan Result Changed Listener class diagram)

3.2.4 Sequence Diagram

The Sequence Diagram illustrates the interaction between the user and the system across three separate functions: plant disease detection, crop recommendation, and fertilizer recommendation. In the disease detection system, the user uploads a plant image, and the system analyzes it using a CNN model to identify the plant type and disease, then returns the result along with a suitable treatment recommendation. In the crop

recommendation system, the user inputs soil data such as pH, moisture, and temperature, and the system suggests the most suitable crops based on the analysis. In the fertilizer recommendation system, the user provides information about the soil and the crop, and the system analyzes the nutrient levels to recommend the appropriate fertilizer. This diagram shows the sequence of steps in each system from data input to result output.

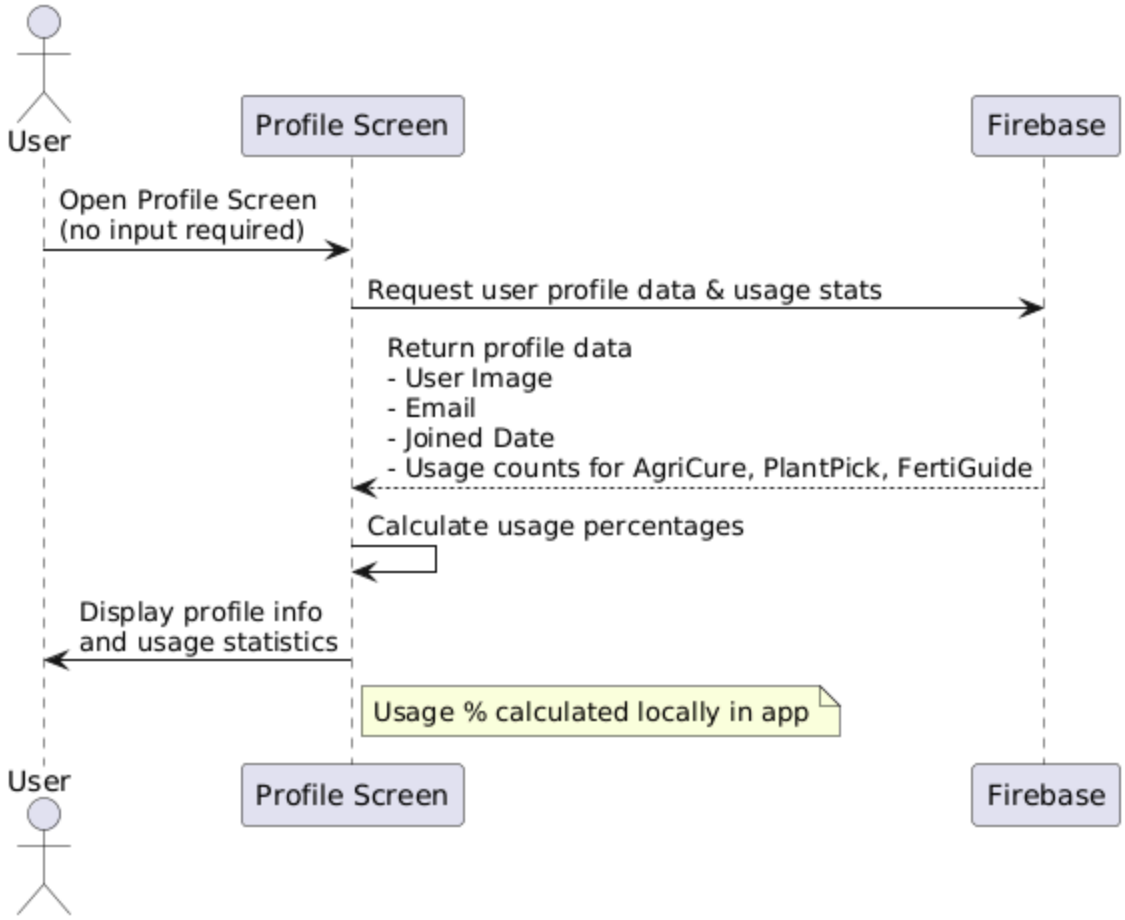


Figure 3.31 (profile Screen sequence diagram)

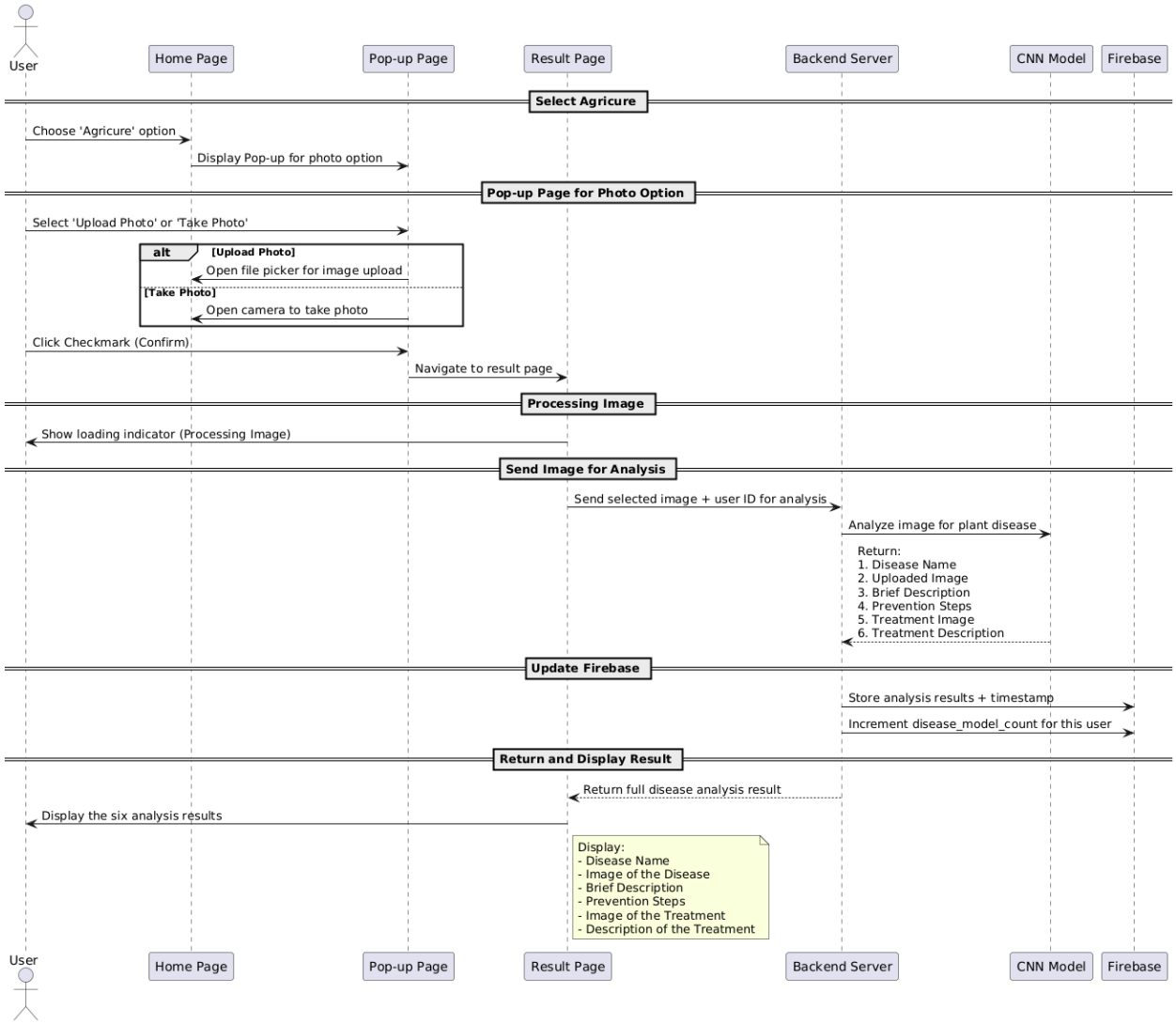


Figure 3.32 (Plant disease detection sequence diagram)

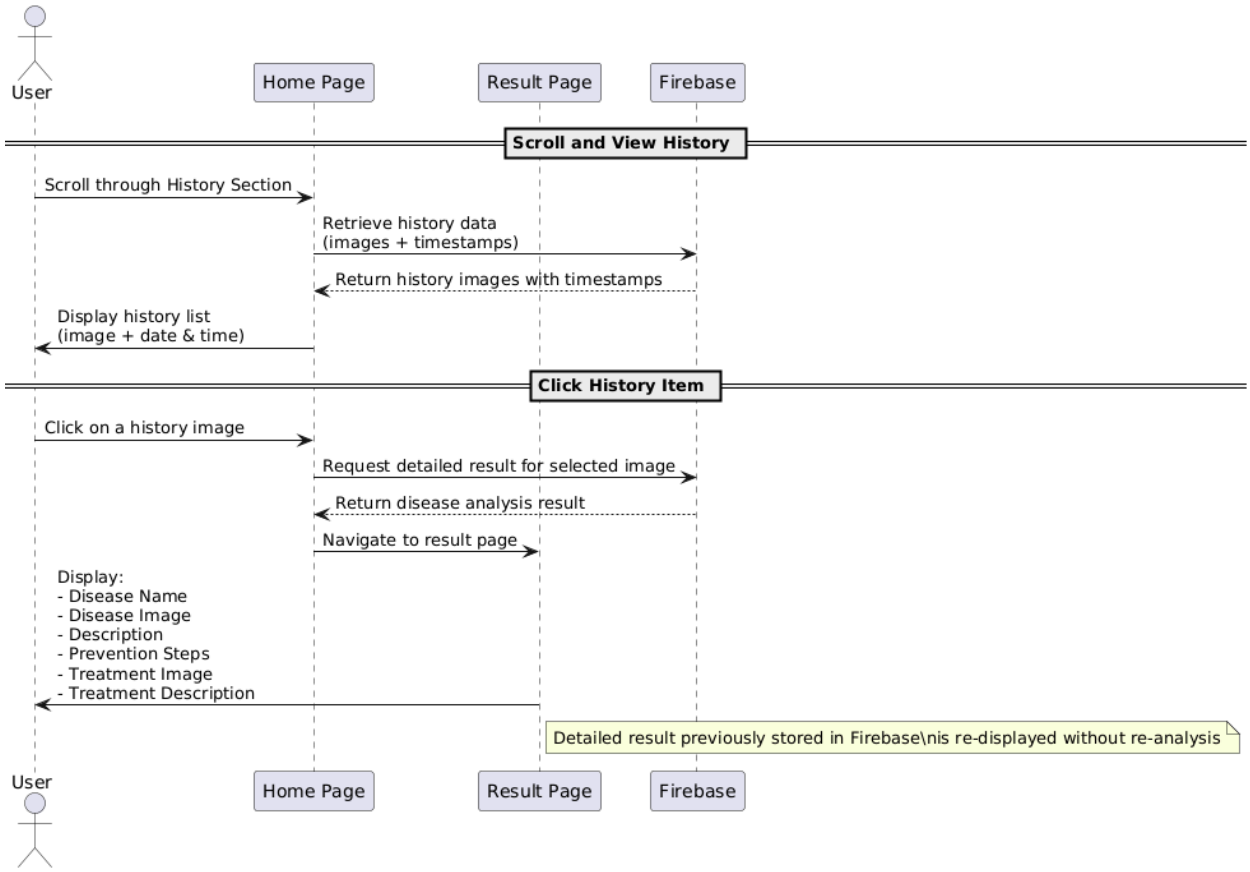


Figure 3.33 (History sequence diagram)

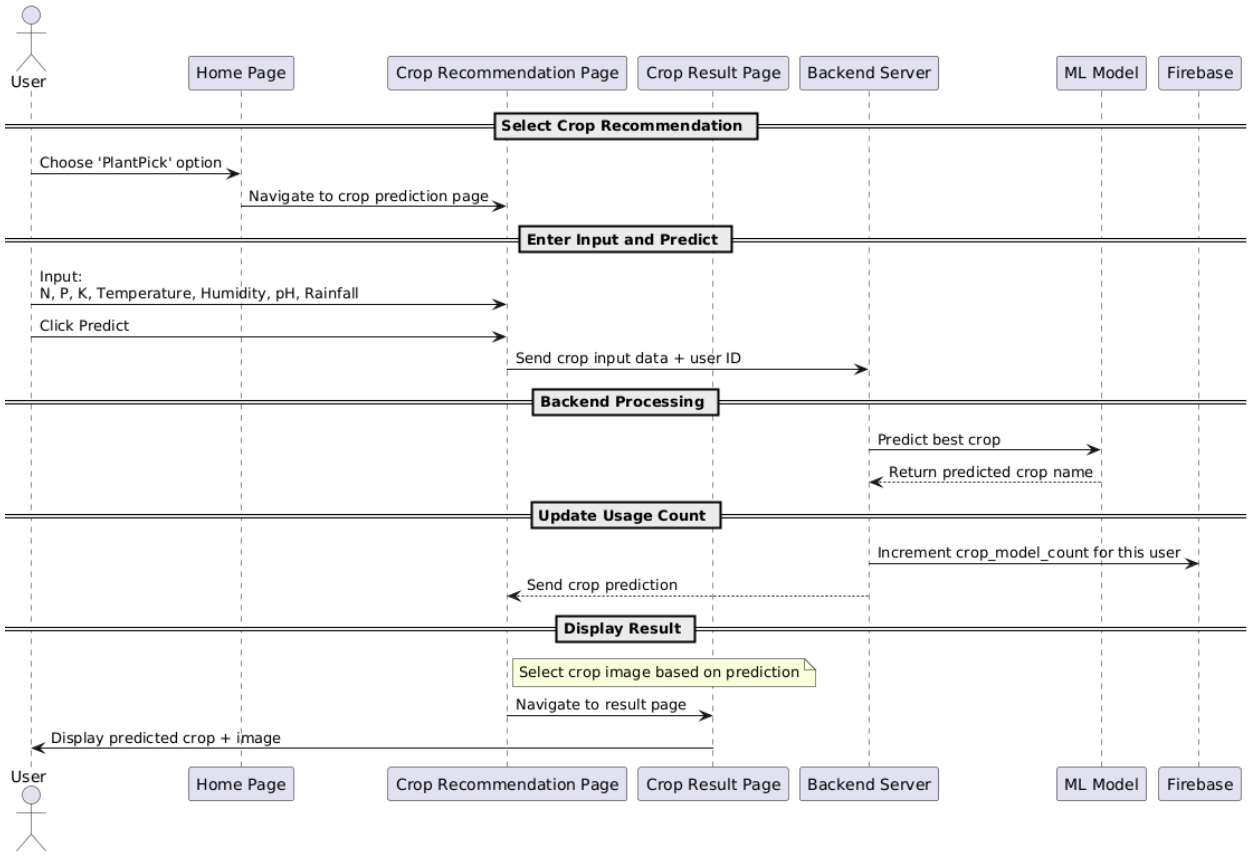


Figure 3.34 (Crop prediction sequence diagram)

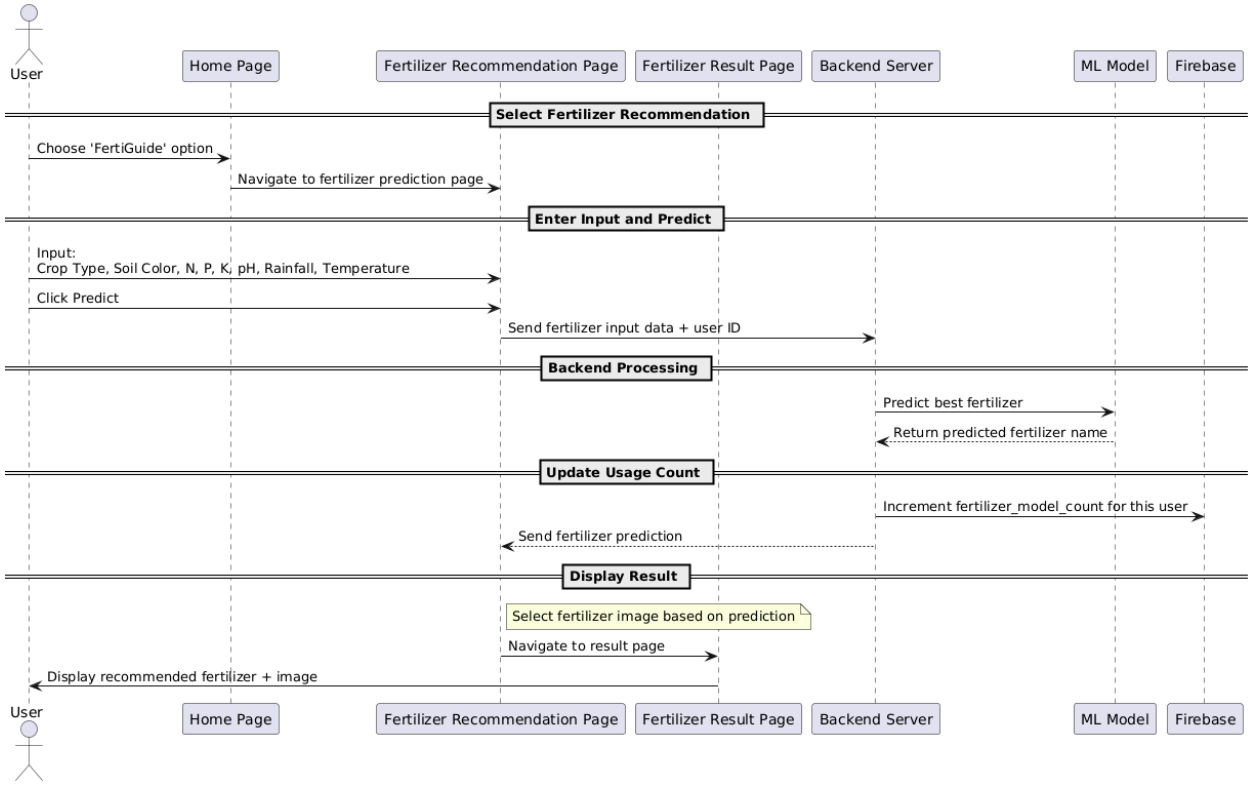


Figure 3.35 (Fertilizer prediction sequence diagram)

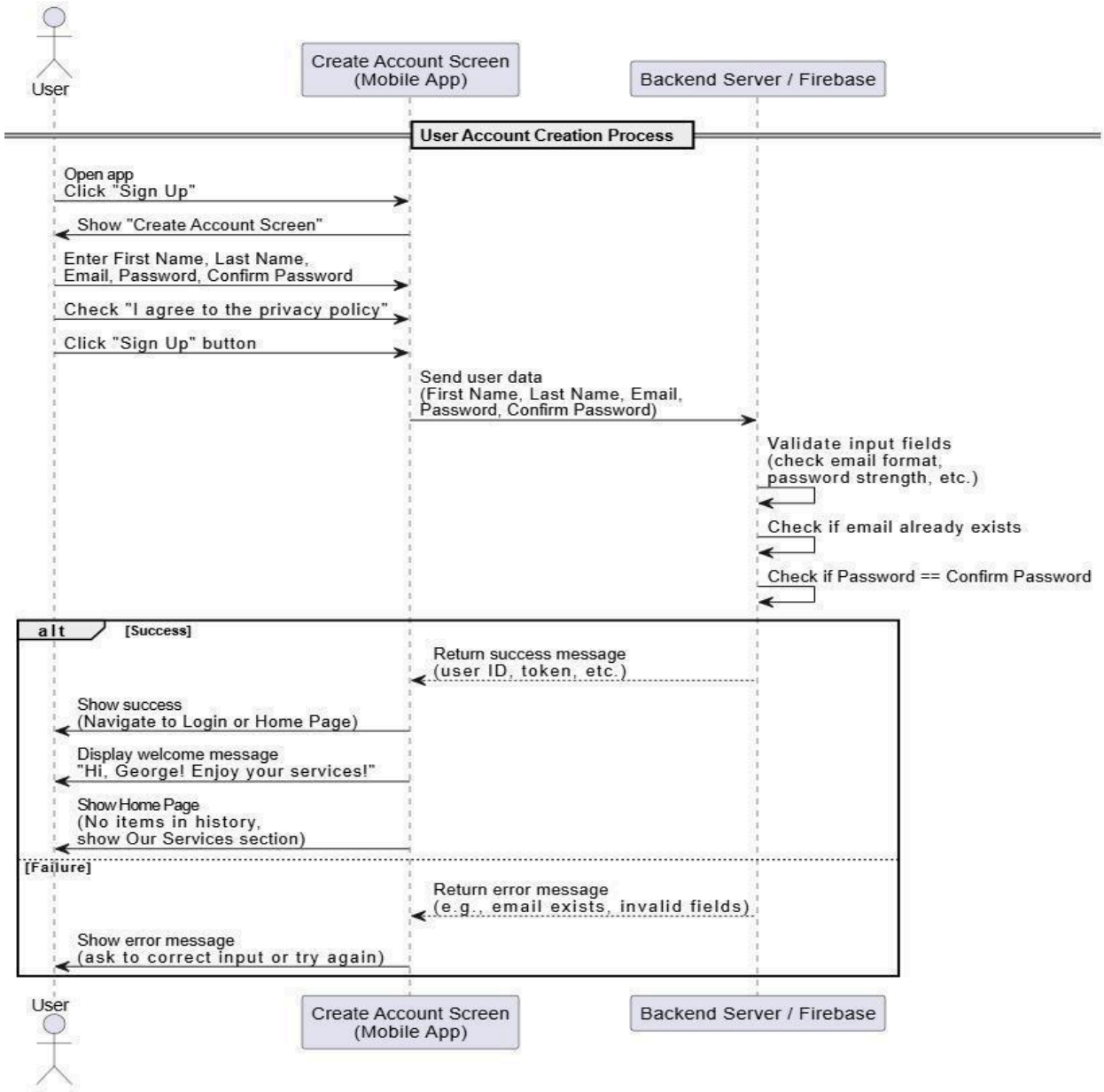


Figure 3.36 (Sign Up sequence diagram)

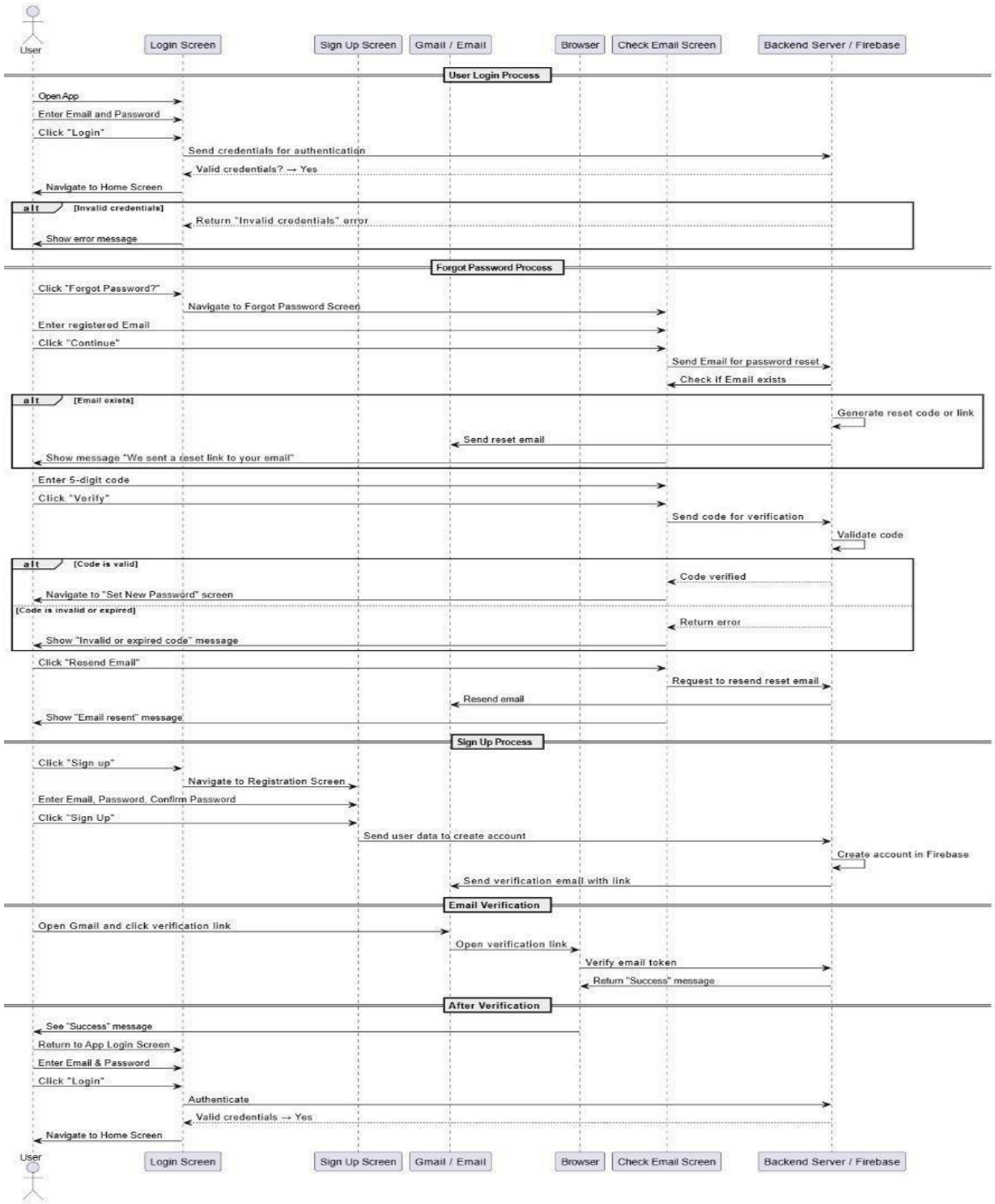


Figure 3.37 (Login sequence diagram)

3.3 Algorithms, Frameworks, and Tools Used

The development of the Planty Care application relied on a combination of machine learning and deep learning algorithms, supported by modern tools and platforms that enabled efficient and flexible implementation.

- **Plant Disease Detection – ResNet50 (CNN):**

  We used the ResNet50 architecture, a deep convolutional neural network known for its residual learning capabilities, to diagnose plant diseases with high precision. This model analyzes leaf images and achieved a test accuracy of 96.30%, ensuring reliable classification of various diseases.

- **Crop Recommendation – Random Forest:**

  To recommend the most suitable crop based on environmental factors such as temperature, humidity, and soil type, we implemented the Random Forest algorithm. It is robust against overfitting and provides strong predictive performance for multi-feature datasets.

- **Fertilizer Recommendation – XGBoost:**

  For suggesting the appropriate type of fertilizer, we employed the XGBoost algorithm, a high-performance gradient boosting framework known for its speed and accuracy. It analyses crop type, soil nutrients, and other variables to give optimal fertilizer suggestions.

All machine learning models were trained and evaluated using Google Colab, a cloud-based platform ideal for AI experimentation. Once trained, these models were integrated into a mobile application developed with Android Studio, which served as the primary environment for frontend and

backend development. The user interface (UI/UX) was designed using Figma, allowing for a clean, user-friendly experience and effective design collaboration.

This integrated use of AI algorithms and development tools resulted in a comprehensive, intelligent system that supports farmers with instant, data-driven agricultural decisions.

Chapter 4

Implementation

System Architecture:

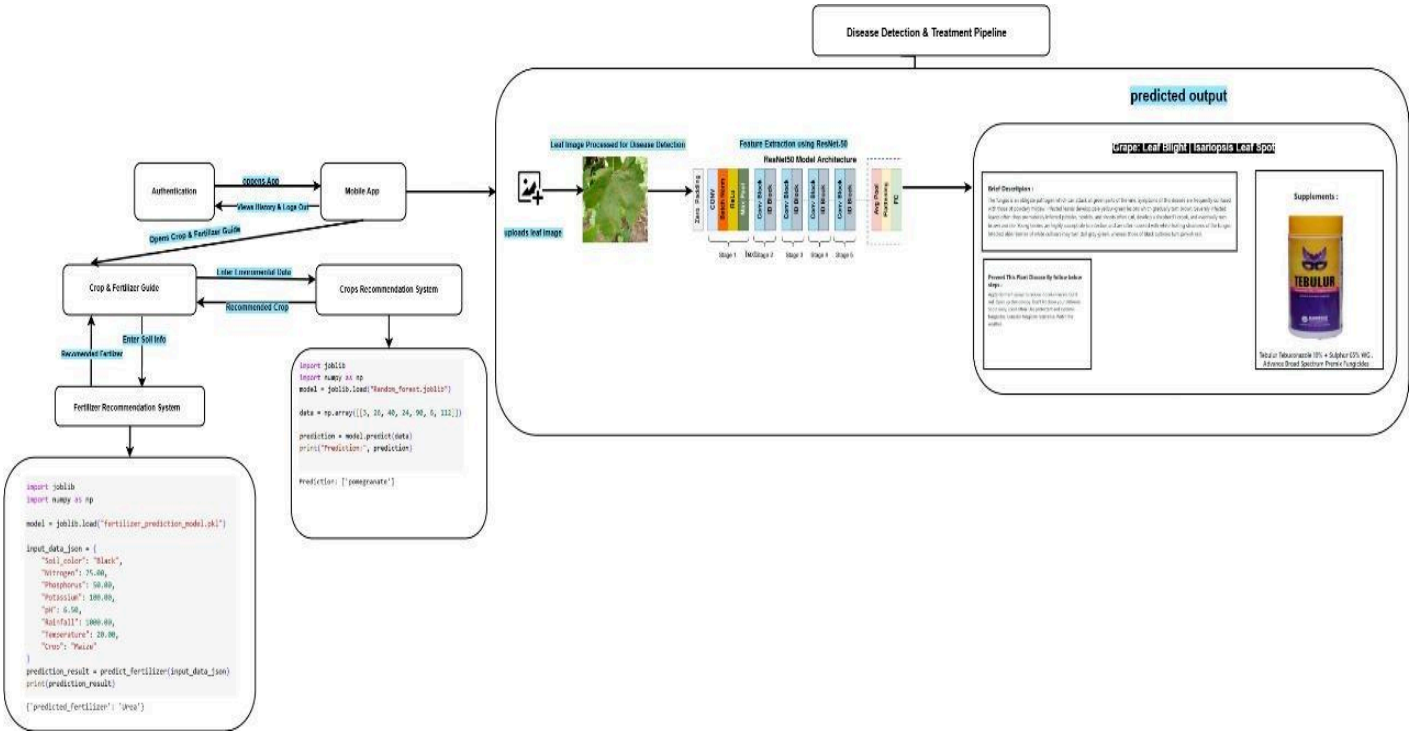


Figure 4.1 (System Architecture diagram)

4.1 Technologies, tools, and programming languages used

The development of the Planty Care application involved a combination of technologies, tools, and programming languages tailored to meet both AI processing needs and mobile application deployment. The selection was made to ensure efficiency, scalability, and smooth user experience.

4.1.1 Technologies and Tools

- Google Colab:

  Used to develop, train, and test the machine learning and deep learning models in a cloud-based environment with GPU acceleration.

- Android Studio:

  The primary integrated development environment (IDE) used to build and test the Android mobile application

- Figma**:**

  Utilized for user interface (UI/UX) design to ensure the app is clean, user-friendly, and accessible.

- Firebase:
  - Firebase Authentication – For user login and registration.
  - Firebase Realtime Database – To store user history and prediction results.
  - Firebase Cloud Storage – To store uploaded or captured plant images
- Image Handling Tools:
  Integrated Android camera and gallery access for image input.

4.1.2 Frameworks and Libraries:

- PyTorch:

  Chosen for training and developing the deep learning model (ResNet50) for plant disease detection due to its flexibility and dynamic computational graph support.
- scikit-learn:

  Used for implementing the Random Forest algorithm in the crop recommendation system.
- XGBoost:

  Applied in the fertilizer recommendation model for high-performance gradient boosting.
- Pandas:

  Used for data manipulation and preprocessing, including handling missing values, feature engineering, and preparing datasets for training
- Matplotlib:

  Employed for visualizing model performance (such as accuracy/loss graphs) and exploratory data analysis.

4.1.3 Programming Languages:

- Python:

  Used to build and train the AI models and implement data processing pipelines.

- Java:

  Used in developing the Android application and integrating AI features into the mobile environment.

- XML:

  Used to design and structure the User Interface (UI) components of the Android application. XML files defined layouts, buttons, text fields, and overall screen arrangements to ensure a responsive and user-friendly interface.

4.2 Datasets Used in the System Implementation.

We have 3 models with 5 datasets we divide with model:

4.2.1 Disease detection model:

in this model we use 3 datasets: 1- new plant disease dataset, 2- disease information, 3- supplement information.

4.2.1.1 new plant disease dataset

4.2.1.1.1 Dataset Definition:

The New Plant Diseases Dataset is used in this project to build a deep learning-based system for the automated diagnosis of plant diseases through image classification. The dataset was obtained from Kaggle and contains approximately 87,725 high-quality RGB images of plant

leaves in .jpg format. These images cover a wide range of crops and include both diseased and healthy samples, organized into 38 distinct classes.

The dataset is structured into three main directories: train, valid, and test, each containing subfolders for every class. The training set contains around 72,000 images, the validation set around 8,800 images, and the test set around 6,900 images. Each class corresponds to a specific plant disease or a healthy condition, with examples such as Apple Scab, Tomato Leaf Mold, Potato Late Blight, and Corn Common Rust

4.2.1.1.2 Dataset Properties:

The system will classify plant diseases by generating a probability score for each of the disease classes. The image of the plant leaf, containing the visible symptoms, is resized to 224x224 pixels and is passed as input to the *ResNet-50* Convolutional Neural Network (CNN). The network generates a list of SoftMax scores for each class representing a specific plant disease or healthy condition. On the output screen, the disease or healthy state with the highest probability score is displayed as the predicted result. This research proposes a solution for plant disease classification using *ResNet-50* based feature extraction in real-time, leveraging the power of deep learning algorithms for accurate and efficient plant health monitoring.


Apple scabe

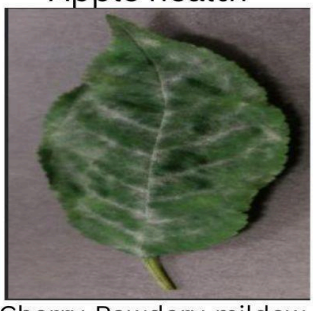

Apple black rot


cedar apple rust


Apple health

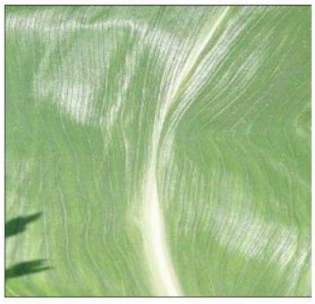

Blueberry healthy

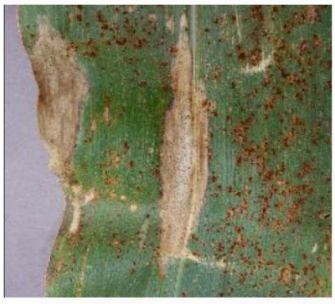

cherry health


Cherry_Powdery_mildew


Corn_Cercospora_leaf_spot


Corn_Common_rust_


Corn_(maize) healthy


Corn Northern Leaf Blight


Grape Black rot

Figure 4.2 (leaves disease images 1)

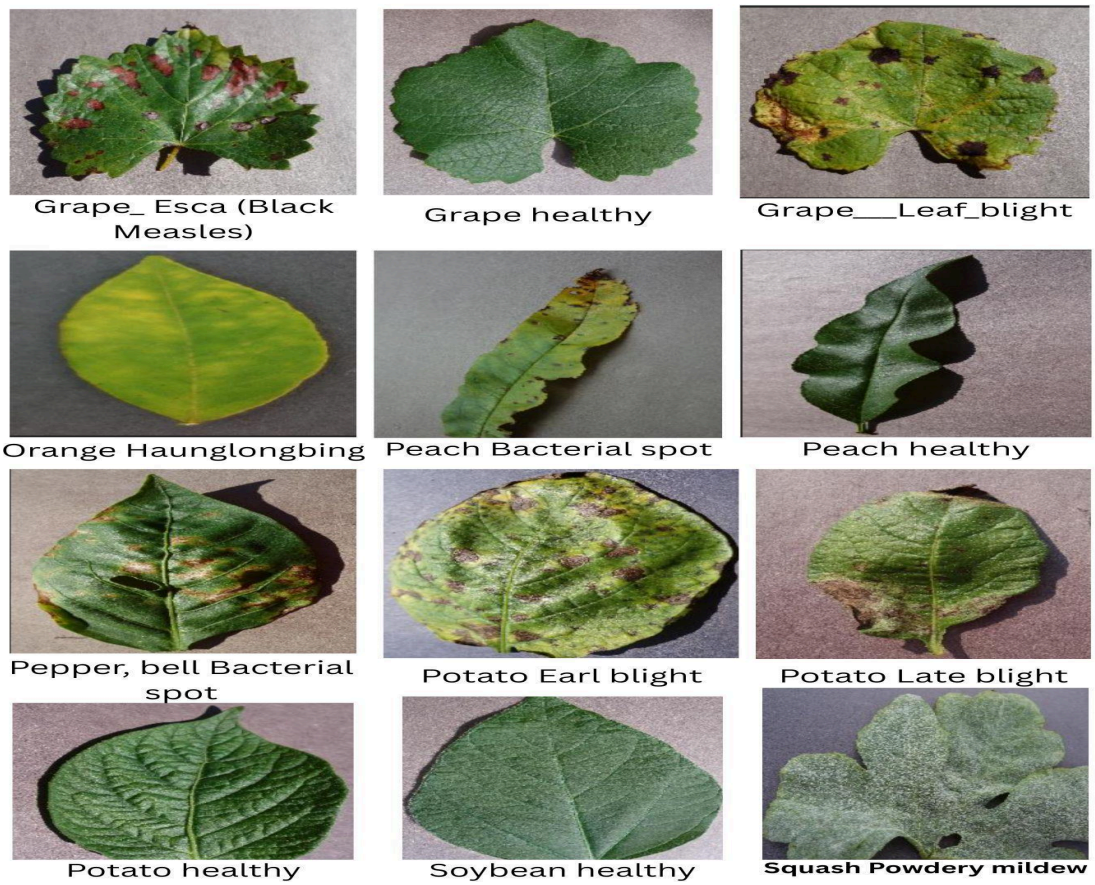


Figure 4.3 (leaves disease images 2)

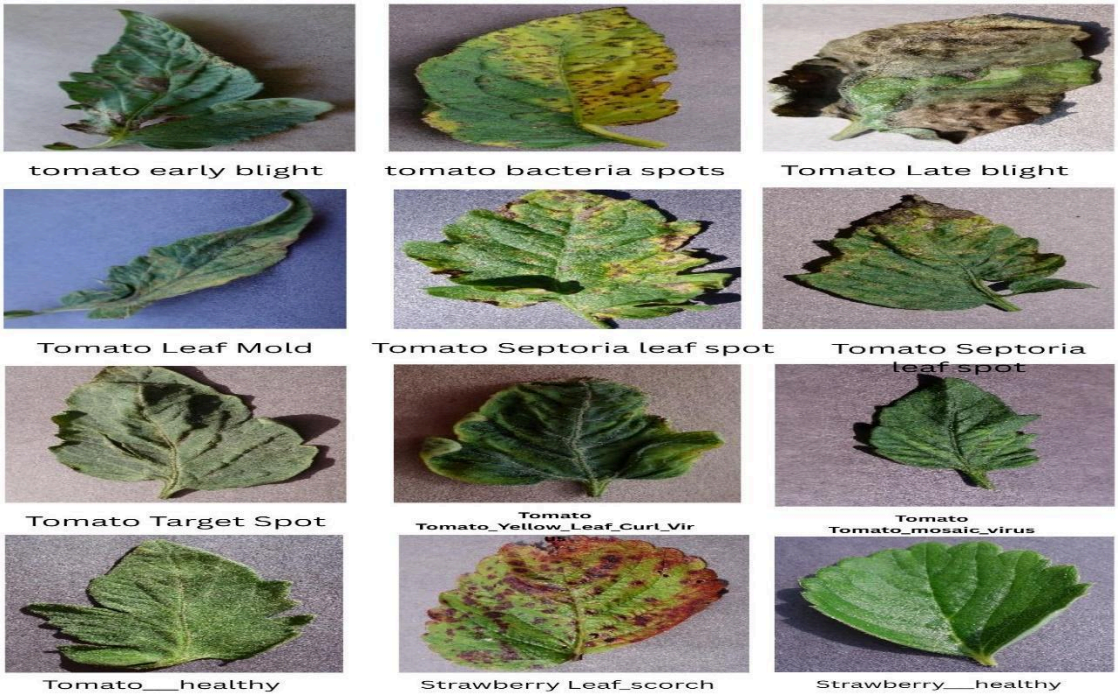


Figure 4.4 (leaves disease images 3)

4.2.1.2 Disease information dataset:

4.2.1.2.1 Dataset definition:

The Plant Disease Information Dataset is a supplementary dataset that provides detailed textual descriptions for 39 distinct plant conditions, including both diseases and healthy states. Each entry includes the disease name, a short description, possible preventive or corrective steps, and an illustrative image URL. This dataset is designed to enrich the plant disease classification system by providing meaningful feedback and suggestions to users once a disease is detected. The integration of this data allows the model not only to predict the disease but also to display informative guidance, making the system more user-friendly and educational for farmers and agricultural specialists

4.2.1.2.2 Dataset properties:

The dataset provides essential textual information for each plant disease to support the prediction and recommendation process. Each record in the dataset includes three key attributes: the disease name, a description of how it affects the plant, and possible treatment or preventive steps. These attributes include:

- **disease name**: The name of the identified plant disease.
- **description**: A summary of symptoms and the potential impact of the disease on plant health.
- **Possible Steps**: Recommended actions that can be taken to manage or prevent the disease.

This dataset serves as an informative layer that complements the predictions generated by the image-based plant disease detection

model. While the model identifies the most likely disease, this dataset adds value by providing practical and reliable guidance that supports agricultural decision-making and directs the user toward appropriate actions for each case

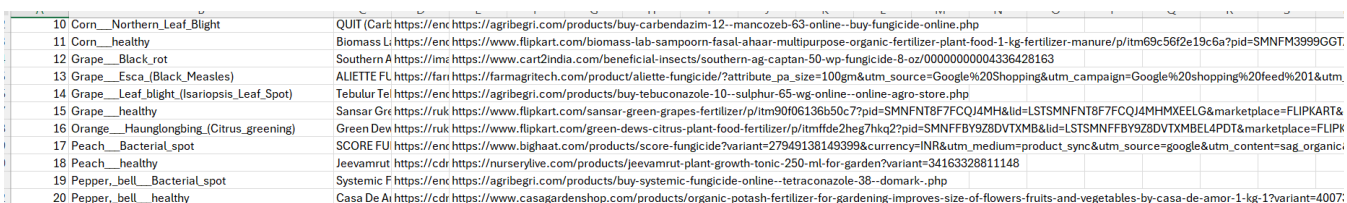
Index	disease_name	description
0	Apple : Scab	Apple scab is the most common disease of apple and crabapple trees in Minnesota.
1	Apple : Black Rot	Leaf symptoms first occur early in the spring when the leaves are unfolding.
2	Apple : Cedar rust	Cedar apple rust (Gymnosporangium juniperi-virginianae) is a fungal disease that requires juniper plants to complete its complicated two year life-cycle. Spores overwinter as a reddish-brown gall on young twigs of various juniper species. In early spring, during wet weather, these galls swell
3	Apple : Healthy	As with most fruit, apples produce best when grown in full sun, which means six or more hours of direct summer Sun daily. The best exposure for apples is a north side of a house, tree line, or rise rather than the south. Apple trees need well-drained soil, but should be able to retain some moistu
4	Blackground Without Leaves	There is no leaf in the given image
5	Blueberry : Healthy	Blueberries Are Low in Calories But High in Nutrients: ...
6	Cherry: Powdery Mildew	Initial symptoms, often occurring 7 to 10 days after the onset of the first irrigation, are light roughly-circular, powdery looking patches on young, susceptible leaves (newly unfolded, and light green expanding leaves). Older leaves develop an age-related (ontogenic) resistance to powdery mil
7	Cherry : Healthy	There is no difference in care between sour and sweet cherries.
8	Corn : Cercospora Leaf Spot | Gray Leaf Spo	Gray leaf spot on corn, caused by the fungus Cercospora zeae-maydis, is a perennial and economically damaging disease in the United States. Since the mid-1990s, the disease has increased in importance in Indiana, and now is the one of the most important foliar diseases of corn in the
9	Corn : Common Rust	Although a few rust pustules can always be found in corn fields throughout the growing season, symptoms generally do not appear until after tasseling. These can be easily recognized and distinguished from other diseases by the development of dark, reddish-brown pustules (uredinia) sca
10	Corn : Northern Leaf Blight	Northern corn leaf blight (NCLB) is caused by the fungus Setosphaeria turcica. Symptoms usually appear first on the lower leaves. Leaf lesions are long (1 to 6 inches) and elliptical, gray-green at first but then turn pale gray or tan. Under moist conditions, dark gray spores are produced, usua
11	Corn : Healthy	Corn plants prefer daytime temperatures of 75 to 80 degrees F and 65 to 70 degrees F during the night. The soil should be kept consistently moist, but not soggy and they only need fertilizer every 6 months. Although dracaena can take low light conditions, they do best when placed in bright
12	Grape : Black Rot	Grape black rot is a fungal disease caused by an ascomycetous fungus, Guignardia bidwellii, that attacks grape vines during hot and humid weather. *Grape black rot originated in eastern North America, but now occours in portions of Europe, South America, and Asia. It can cause complete
13	Grape : Esca | Black Measles	Grapevine measles, also called esca, black measles or Spanish measles, has long plagued grape growers with its cryptic expression of symptoms and, for a long time, a lack of identifiable causal organism(s). The name "measles" refers to the superficial spots found on the fruit (Fig. 1). During
14	Grape : Leaf Blight | Isariopsis Leaf Spot	The fungus is an obligate pathogen which can attack all green parts of the vine.
15	Grape : Healthy	Apply water only to the root zone. Avoid getting grape foliage wet as this can encourage many grape diseases. Reduce watering young vines in the fall to encourage the plant to harden-off its canes to prepare for winter. Older vines seldom need any watering unless on sandy or other very w
16	Orange : Haunglongbing | Citrus Greening	Citrus greening disease is a disease of citrus caused by a vector-transmitted pathogen. HLB is distinguished by the common symptoms of yellowing of the veins and adjacent tissues; followed by splotchy mottling of the entire leaf, premature defoliation, dieback of twigs, decay of feeder root
17	Peach : Bacterial Spot	Bacterial spot is an important disease of peaches, nectarines, apricots, and plums caused by Xanthomonas campestris pv. pruni. Symptoms of this disease include fruit spots, leaf spots, and twig cankers. Fruit symptoms include pitting, cracking, gumming, and water-soaked tissue, which ca
18	Peach : Healthy	Peach trees grow in USDA Zones 5 to 8 or 9, and they're available in dwarf or standard sizes, making them a great choice for backyard orchards. Unlike most ornamentals, peach trees need regular pruning, fertilizing, and spraying to stay healthy and productive. Keep the ground around you
19	Pepper bell : Bacterial Spot	Leaf spots that appear on the lower surface of older leaves as small, pimples and on the upper leaf surface as small water-soaked spots are a symptom of bacterial spot. This is an important pepper disease in Maryland. It also occasionally attacks tomatoes. Eventually, the spots develop gra
20	Pepper bell : Healthy	Keep bell peppers well-watered, but never leave soil soggy. Water to moisten soil about 6 inches deep, then let it dry slightly. Watering is especially important during fruit set, when tiny peppers take the place of blossoms, and as the bells mature. Consistent moisture helps keep peppers firm a
21	Potato : Early Blight	In most production areas, early blight occurs annually to some degree. The severity of early blight is dependent upon the frequency of foliar wetness from rain, dew, or irrigation; the nutritional status of the foliage; and cultivar susceptibility. The first symptoms of early blight appear as small, cir
22	Potato : Late Blight	The primary host is potato, but P. infestans also can infect other solanaceous plants, including tomatoes, petunias and hairy nightshade, that can act as source of inoculum to potato. The first symptoms of late blight in the field are small, light to dark green, circular to irregular-shaped water-s
23	Potato : Healthy	Many potatoes need consistent moisture levels to develop, so water regularly on a daily basis when tubers establish. When you the water the potato plants it should reach to the ground level of about 8-10 inches. During hot or warm seasons water the potato plants in huge amounts twice a d
24	Raspberry : Healthy	Water one inch per week from spring until after harvest. Regular watering is better than infrequent deep soaking. Keep your raspberry bushes tidy by digging up any "suckers" or canes that grow well away from the rows; if you don't dig them up, they'll draw nutrients away and you'll have less t
25	Soybean : Healthy	Keep planting beds evenly moist until soybeans have pushed through the soil. Water regularly during flowering and pod formation. Avoid overhead watering which can cause flowers and pods to fall off. Mulch when the soil warms to greater than 60˚F (16˚C) to conserve soil moisture.
26	Squash : Powdery Mildew	White powdery spots can form on both upper and lower leaf surfaces, and quickly expand into large blotches.
27	Strawberry : Leaf Scorch	Leaf scorch symptoms are very similar to the early stages of common (Mycosphaerella) leaf spot, with irregular dark purple spots being scattered over the upper leaf surface. As the spots enlarge, they begin to look like drops of tar, and are actually the accumulations of black fruiting bodies (a
28	Strawberry : Healthy	Find them a sunny spot because they love and need lots of light. Ipreler a spot where they receive the morning sun.
29	Tomato : Bacterial Spot	Bacterial spot of tomato is a potentially devastating disease that, in severe cases, can lead to unmarketable fruit and even plant death. Bacterial spot can occur wherever tomatoes are grown, but is found most frequently in warm, wet climates, as well as in greenhouses. The disease is ofter
30	Tomato : Early Blight	Early blight is one of the most common tomato diseases, occuring nearly every season wherever tomatoes are grown.
31	Tomato : Late Blight	Late blight is caused by the oomycete Phytophthora infestans. Oomycetes are fungus-like organisms also called water molds, but they are not true fungi.
32	Tomato : Leaf Mold	Tomato leaf mold is a fungal disease that can develop when there are extended periods of leaf wetness and the relative humidity is high (greater than 85 percent). Due to this moisture requirement, the disease is seen primarily in hoophouses and greenhouses. Tomato leaf mold can develop
33	Tomato : Septoria Leaf Spot	Septoria leaf spot is caused by a fungus, Septoria lycopersici. It is one of the most destructive diseases of tomato foliage and is particularly severe in areas where wet, humid weather persists for extended periods. Septoria leaf spot usually appears on the lower leaves after the first fruit sets. S
34	Tomato : Spider Mites | Two-Spotted Spider	The two-spotted spider mite is the most common mite species that attacks vegetable and fruit crops in New England. Spider mites can occur in tomato, eggplant, potato, vine crops such as melons, cucumbers, and other crops. Two-spotted spider mites are one of the most important pests o
35	Tomato : Target Spot	The disease starts on the older leaves and spreads upwards. The first signs are irregular-shaped spots (less than 1mm) with a yellow margin. Some of the spots enlarge up to 10 mm and show characteristics rings, hence the name of "target spot". Spread to all leaflets and to other leaves is
36	Tomato : Yellow Leaf Curl Virus	Tomato yellow leaf curl virus is a species in the genus Begomovirus and family Geminiviridae. Tomato yellow leaf curl virus (TYLCV) infection induces severe symptoms on tomato plants and causes serious yield losses worldwide. TYLCV is persistently transmitted by the sweetpotato whitefly,
37	Tomato : Mosaic Virus	Tomato mosaic virus (ToMV) and Tobacco mosaic virus (TMV) are hard to distinguish.
38	Tomato : Healthy	Fertilize one week before as well as on the day of planting. They especially love phosphorous, which promotes the formation of blossoms and the fruits or vegetables that grow from them. Avoid high nitrogen when your tomato plants have blossoms as it promotes vine growth rather than fruit g

Figure 4.5 (Disease information dataset)

4.2.1.2 Supplement information dataset:

4.2.1.2.1 Dataset Definition:

The Supplement Recommendation Dataset is a supporting dataset used to enhance the treatment recommendation functionality of the Planty Care application. It contains curated information for various plant diseases, including the disease name, a recommended supplement or product name, an image URL of the supplement, and a purchase link. This dataset is specifically designed to provide users with actionable solutions after a disease has been identified by the system. By linking disease predictions to real treatment products, the system improves its practical value, enabling users—especially farmers—to take immediate steps toward recovery and prevention. The integration of this dataset bridges the gap between diagnosis and treatment, offering a more complete and user-oriented experience.



10	Corn___Northern_Leaf_Blight	QUIT (Carb	https://agribegri.com/products/buy-carbendazim-12--mancozeb-63-online--buy-fungicide-online.php
11	Corn___healthy	Biomass Li	https://www.flipkart.com/biomass-lab-sampoorn-fasal-ahaar-multipurpose-organic-fertilizer-plant-food-1-kg-fertilizer-manure/p/itm69c56f2e19c6a?pid=SMNFM3999GGT
12	Grape___Black_rot	Southern A	https://www.cart2india.com/beneficial-insects/southern-ag-captan-50-wp-fungicide-8-oz/00000000004336428163
13	Grape___Esca_(Black_Measles)	ALIETTE FL	https://farmagritech.com/product/aliette-fungicide/?attribute_pa_size=100gm&utm_source=Google%20Shopping&utm_campaign=Google%20shopping%20feed%201&utm
14	Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	Tebulur Te	https://agribegri.com/products/buy-tebuconazole-10--sulphur-65-wg-online--online-agro-store.php
15	Grape___healthy	Sansar Gre	https://www.flipkart.com/sansar-green-grapes-fertilizer/p/itm90f06136b50c7?pid=SMNFNT8F7FCQJ4MH&lid=LSTSMNFNT8F7FCQJ4MHMXEELG&marketplace=FLIPKART&
16	Orange___Haunglongbing_(Citrus_greening)	Green Dew	https://www.flipkart.com/green-dews-citrus-plant-food-fertilizer/p/itmffde2heg7hkq2?pid=SMNFFBY9Z8DVTXMB&lid=LSTSMNFFBY9Z8DVTXMBEL4PDT&marketplace=FLIPK
17	Peach___Bacterial_spot	SCORE FU	https://agribegri.com/products/score-fungicide?variant=27949138149399&currency=INR&utm_medium=product_sync&utm_source=google&utm_content=sag_organic
18	Peach___healthy	Jeevamrut	https://nurserylive.com/products/jeevamrut-plant-growth-tonic-250-ml-for-garden?variant=34163328811148
19	Pepper,_bell___Bacterial_spot	Systemic F	https://agribegri.com/products/buy-systemic-fungicide-online--tetraconazole-38--domark-.php
20	Pepper,_bell___healthy	Casa De Ar	https://www.casagardenshop.com/products/organic-potash-fertilizer-for-gardening-improves-size-of-flowers-fruits-and-vegetables-by-casa-de-amor-1-kg-1?variant=4007

Figure 4.6 (Supplement information dataset)

4.2.2 Crop recommendation:

4.2.2.1 Dataset Definition:

Our model's dataset is the Crop Recommendation dataset, which was designed to recommend the most suitable crop based on soil nutrients and weather conditions. The dataset comprises 2,200 instances, each containing seven numerical features representing environmental and soil parameters. The aim is to categorize each instance into one of several crop types depending on the provided inputs. The features include the content of Nitrogen, Phosphorous, and Potassium in the soil (in kg/ha), as well as the temperature (in Celsius), humidity (in percentage), pH value of the soil, and rainfall (in mm). The crops are labelled according to the best match for the given conditions. The dataset supports multi-class classification with crops such as rice, maize, cotton, and more. The training set typically includes the full dataset for model development and evaluation, with a suggested split of 80% for training and 20% for testing. The data varies naturally in weather

and soil conditions, making it suitable for building robust and generalizable agricultural recommendation systems.

4.2.2.2 Dataset Properties:

The model generates probability scores for each crop class, where each data instance is classified based on the highest predicted probability. The features related to soil and climate—such as Nitrogen, Phosphorus, and Potassium levels, temperature, humidity, soil pH, and rainfall—are input into machine learning algorithms that determine the most suitable crop under those conditions.

Several classification algorithms were tested, including Logistic Regression, Decision Tree, Random Forest, and XGBoost. Among these, the Random Forest algorithm demonstrated the highest performance in terms of accuracy.

This research proposes a system capable of predicting and recommending the most appropriate crop based on environmental and agricultural inputs using machine learning techniques. The system integrated into our application to provide real-time recommendations for farmers or agricultural specialists.

N	P	K	temperature	humidity	ph	rainfall	label
90	42	43	20.87974371	82.00274423	6.502985292	202.9355362	rice
85	58	41	21.77046169	80.31964408	7.038096361	226.6555374	rice
60	55	44	23.00445915	82.3207629	7.840207144	263.9642476	rice
74	35	40	26.49109635	80.15836264	6.980400905	242.8640342	rice
78	42	42	20.13017482	81.60487287	7.628472891	262.7173405	rice
69	37	42	23.05804872	83.37011772	7.073453503	251.0549998	rice
69	55	38	22.70883798	82.63941394	5.70080568	271.3248604	rice
94	53	40	20.27774362	82.89408619	5.718627178	241.9741949	rice
89	54	38	24.51588066	83.5352163	6.685346424	230.4462359	rice
68	58	38	23.22397386	83.03322691	6.336253525	221.2091958	rice
91	53	40	26.52723513	81.41753846	5.386167788	264.6148697	rice
90	46	42	23.97898217	81.45061596	7.50283396	250.0832336	rice
78	58	44	26.80079604	80.88684822	5.108681786	284.4364567	rice
93	56	36	24.01497622	82.05687182	6.98435366	185.2773389	rice
94	50	37	25.66585205	80.66385045	6.94801983	209.5869708	rice
60	48	39	24.28209415	80.30025587	7.042299069	231.0863347	rice
85	38	41	21.58711777	82.7883708	6.249050656	276.6552459	rice
91	35	39	23.79391957	80.41817957	6.970859754	206.2611855	rice
77	38	36	21.8652524	80.1923008	5.953933276	224.5550169	rice
88	35	40	23.57943626	83.58760316	5.85393208	291.2986618	rice
89	45	36	21.32504158	80.47476396	6.442475375	185.4974732	rice
76	40	43	25.15745531	83.11713476	5.070175667	231.3843163	rice
67	59	41	21.94766735	80.97384195	6.012632591	213.3560921	rice
83	41	43	21.0525355	82.67839517	6.254028451	233.1075816	rice
98	47	37	23.48381344	81.33265073	7.375482851	224.0581164	rice

Figure 4.7 (Crop recommendation dataset)

4.2.3 Fertilizer recommendation model:

4.2.3.1 Dataset definition:

This research utilizes a Crop and Fertilizer Recommendation dataset designed to support agricultural decision-making by recommending the most suitable type of fertilizer based on soil properties and environmental conditions. The dataset contains 4,513 records collected from various districts, with each record describing the condition of the soil, the suitable crop, and the corresponding recommended fertilizer.

The dataset includes the following features: District Name, Soil Color, Nitrogen (N), Phosphorus (P), Potassium (K) levels, pH value, Rainfall, Temperature, Recommended Crop, Recommended Fertilizer, and an educational reference link. The diversity of environmental and geographic conditions represented in the dataset makes it highly suitable for training machine learning models that can generalize well to different farming scenarios.

4.2.3.2 Dataset properties:

The XGBoost (Extreme Gradient Boosting) algorithm is employed to train a classification model that predicts the most appropriate fertilizer based on the given soil and climate features. This problem is approached as a multi-class classification task, where the model outputs a set of probability scores corresponding to different fertilizer types.

Initially, the data is pre-processed by encoding categorical variables (such as soil color and district name) into numerical values and then fed into the model. The XGBoost model produces a probability distribution across all possible fertilizer classes, and the class with the highest probability is selected as the final recommendation.

XGBoost's strength lies in its ability to handle imbalanced data and its integrated regularization features, which help reduce model bias and improve prediction accuracy. This makes it highly effective for smart agricultural applications like our application that require accurate and timely recommendations for fertilizer usage tailored to specific soil and crop conditions.

	A	B	C	D	E	F	G	H	I	J	K
1	District_Name	Soil_color	Nitrogen	Phosphorus	Potassium	pH	Rainfall	Temperature	Crop	Fertilizer	Link
2	Kolhapur	Black	75	50	100	6.5	1000	20	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
3	Kolhapur	Black	80	50	100	6.5	1000	20	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
4	Kolhapur	Black	85	50	100	6.5	1000	20	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
5	Kolhapur	Black	90	50	100	6.5	1000	20	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
6	Kolhapur	Black	95	50	100	6.5	1000	20	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
7	Kolhapur	Black	100	50	100	6.5	1000	20	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
8	Kolhapur	Black	75	55	105	7	1100	25	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
9	Kolhapur	Black	80	55	105	7	1100	25	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
10	Kolhapur	Black	85	55	105	7	1100	25	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
11	Kolhapur	Black	90	55	105	7	1100	25	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
12	Kolhapur	Black	95	55	105	7	1100	25	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
13	Kolhapur	Black	100	55	105	7	1100	25	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
14	Kolhapur	Black	75	60	110	7.5	1200	30	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
15	Kolhapur	Black	80	60	110	7.5	1200	30	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
16	Kolhapur	Black	85	60	110	7.5	1200	30	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
17	Kolhapur	Black	90	60	110	7.5	1200	30	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
18	Kolhapur	Black	95	60	110	7.5	1200	30	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
19	Kolhapur	Black	100	60	110	7.5	1200	30	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
20	Kolhapur	Black	75	50	115	6.5	1300	35	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
21	Kolhapur	Black	80	50	115	6.5	1300	35	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
22	Kolhapur	Black	85	50	115	6.5	1300	35	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
23	Kolhapur	Black	90	50	115	6.5	1300	35	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
24	Kolhapur	Black	95	50	115	6.5	1300	35	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
25	Kolhapur	Black	100	50	115	6.5	1300	35	Sugarcane	Urea	https://youtu.be/2t5Am0xLTOo
26	Kolhapur	Black	75	55	100	7	1400	20	Sugarcane	DAP	https://youtu.be/2t5Am0xLTOo

Figure 4.8 (Fertilizer recommendation dataset)

4.3 Key Components/Modules of the System

4.3.1 Models:

We have 3 models in our application,

The Planty Care system is composed of several integrated modules, each responsible for a specific functionality. These modules work together to provide a complete and intelligent plant care experience:

1. **Image Acquisition Module**
    - Captures images of plant leaves using the camera on the user's mobile device through the Android application
    - Ensures the image quality is sufficient for accurate classification.
2. **Image Preprocessing Module**
    - Performs essential transformations using torchvision.transforms to prepare images for model input**.**
    - Operations include:
        - Resizing to 128x128 pixels: transforms.Resize((128, 128))
        - Conversion to tensors: transforms.ToTensor()
        - Normalization using ImageNet statistics:

```
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Figure 4.9 (Image Preprocessing code)

3. **Plant Disease Detection Module**

- Utilizes a pretrained ResNet50 model implemented in PyTorch to classify the disease from the processed leaf image.
- The CNN model extracts deep features from the image and outputs a probability distribution across 38 disease classes.
- Trained on a labeled dataset with an 80:20 train-validation split.
- The model is trained and evaluated using custom scripts, including a training loop that performs forward passes, computes loss, performs backpropagation, and updates weights using the Adam optimizer.
- Libraries Used:

    - NumPy: A core library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

    - matplotlib: A plotting library for creating static, animated, and interactive visualizations in Python. It helps visualize model performance, including loss curves and accuracy, as well as the final results of disease classification.

    - PyTorch: The backbone of the model, PyTorch provides the necessary tools to define, train, and evaluate neural networks. Learningdes automatic differentiation, which is critical for training deep

learning models, and its modularity allows for easy experimentation with various model architectures.

- Training Parameters:
  - **Number of classes**: 38
  - **Batch size (train)**: 32
  - **Batch size (validation)**: 60
  - **Learning rate**: 0.001
  - **Number of epochs**: 20
- **Train accuracy**: 99.33%
- **Validation accuracy**: 99.42%

```
# Load pre-trained ResNet50 model with fine-tuning
model = models.resnet50(pretrained=True)
model.fc = nn.Linear(model.fc.in_features, num_classes)

# Define device (GPU if available, otherwise CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.AdamW(model.parameters(), lr=0.001)

# Variables for tracking performance
train_accuracies = []
val_accuracies = []

best_val_acc = 0.0
start_epoch = 0
epochs = 20


# Training loop with periodic print statements to prevent session timeout
for epoch in range(start_epoch, epochs):
    model.train()  # Set model to training mode
    running_loss = 0.0
    correct_train = 0
    total_train = 0

    for batch_idx, (inputs, labels) in enumerate(train_loader):
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()  # Zero the parameter gradients
        outputs = model(inputs)  # Forward pass
        loss = criterion(outputs, labels)  # Compute loss
        loss.backward()  # Backpropagation
        optimizer.step()  # Update model parameters

        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1) # Get predicted class
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

        # Print every 50 batches to prevent session timeout
        if batch_idx % 50 == 0:
            print(f"[Train] Epoch {epoch+1}, Batch {batch_idx}: Loss = {loss.item():.4f}")
```

Figure 4.10 (train code for plant disease detection model)

```
    # Calculate training accuracy
    train_acc = 100 * correct_train / total_train
    train_accuracies.append(train_acc)

    # Validation phase with batch-wise logging
    model.eval()  # Set model to evaluation mode
    correct_val = 0
    total_val = 0

    with torch.no_grad():  # Disable gradient computation for efficiency
        for batch_idx, (inputs, labels) in enumerate(val_loader):
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            _, predicted = torch.max(outputs, 1)
            total_val += labels.size(0)
            correct_val += (predicted == labels).sum().item()

            # Print every 10 batches during validation
            if batch_idx % 10 == 0:
                print(f"[Validation] Epoch {epoch+1}, Batch {batch_idx}: Loss = {loss.item():.4f}")

    # Calculate validation accuracy
    val_acc = 100 * correct_val / total_val
    val_accuracies.append(val_acc)

    # Print epoch results
    print(f"Epoch {epoch+1}, Train Accuracy: {train_acc:.2f}%, Validation Accuracy: {val_acc:.2f}%")

    # Save best model based on validation accuracy
    if val_acc > best_val_acc:
        best_val_acc = val_acc
        checkpoint_path = f"/content/best_model_checkpoint_epoch_{epoch + 1}.pth"
        torch.save({
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'epoch': epoch + 1,
        }, checkpoint_path)
        print(f"✅ Best model checkpoint saved at {checkpoint_path}")

    print('-' * 50)

# Download the last saved model
files.download(checkpoint_path)
```

Figure 4.11 (validation code for plant disease detection model)

Reproducibility:

• To ensure the reproducibility of our experiments and consistent results across multiple runs, we manually set a random seed. We initialized the random seed across different libraries and environments using the

```
# Set random seeds for reproducibility
seed = 42
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
np.random.seed(seed)
random.seed(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

Figure 4.12 (Reproducibility code)

```
# Download the last saved model
files.download(checkpoint_path)


# Plot training and validation accuracy
plt.plot(range(start_epoch, epochs), train_accuracies, label='Train Accuracy')
plt.plot(range(start_epoch, epochs), val_accuracies, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Train and Validation Accuracy')
plt.legend()
plt.show()
```

Figure 4.13 (save plant disease detection model)

```
[Validation] Epoch 17, Batch 250: Loss = 0.0009
[Validation] Epoch 17, Batch 260: Loss = 0.0709
[Validation] Epoch 17, Batch 270: Loss = 0.0007
[Validation] Epoch 17, Batch 280: Loss = 0.0023
Epoch 17, Train Accuracy: 99.33%, Validation Accuracy: 99.42%
✅ Best model checkpoint saved at /content/best_model_checkpoint_epoch_17.pth
--------------------------------------------------
```

Figure 4.14 (Train and Validation accuracy)

4. Recommendation Module

- After identifying the disease, this module uses a supplementary dataset (supplement_info.csv) to fetch relevant details such as:
  - Disease name and description
  - Preventive and corrective steps
  - Image references and explanations
- Provides this information to the user in the mobile application.

```python
def ALL_info(outputs):
    index = np.argmax(outputs, axis=1)
    index = index.item()
    Name = info["disease_name"][index]
    description = info["description"][index]
    prev = info["Possible Steps"][index]
    supplement_name = sup['supplement name'][index]
    supplement_image_url = sup['supplement image'][index]
    return Name, description, prev, supplement_name, supplement_image_url
```

Figure 4.13 (Supplement and description output code)

5. Crop Prediction Model

This machine learning model assists farmers in selecting the most suitable crop based on key soil and environmental parameters. It helps optimize agricultural output by aligning crop choice with soil health and climatic conditions.

- Model Used: Random Forest Classifier
- Train accuracy: 99.93 %
- Validation accuracy : 99.15 %
- Reason for Choice: Random Forest was selected for its high accuracy and ability to handle non-linear relationships and noisy data
- Input features:
    - **N (Nitrogen)**: Essential for vegetative growth.
    - **P (Phosphorus)**: Enhances root and flower development.

- **K (Potassium)**: Supports metabolism and stress resistance.
- **Temperature (°C)**: Affects growth and metabolic activity.
- **Humidity (%)**: Influences transpiration and disease spread.
- **pH**: Determines nutrient availability in soil.
- **Rainfall (mm)**: Affects water availability for crops.

- **Output**: The most suitable crop type for the given conditions.

```
[ ]  features = df[['N', 'P','K','temperature', 'humidity', 'ph', 'rainfall']]
     target = df['label']

     labels = df['label']
```

Figure 4.14 (features and target)

```
[ ]
    Xtrain, Xval, Ytrain, Yval = train_test_split(features,target,test_size = 0.3,random_state =42, shuffle = True)

    Xval, Xtest, Yval, Ytest = train_test_split(Xval,Yval,test_size = 0.1, shuffle = False)
```

Figure 4.15 (split data)

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

random_forest = RandomForestClassifier(random_state=2, n_jobs=-1)

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10]
```

Figure 4.16 (Algorithm Code)

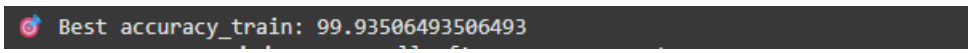


Figure 4.17 (train accuracy)

	precision	recall	f1-score	support
apple	1.00	1.00	1.00	66
banana	1.00	1.00	1.00	74
blackgram	1.00	1.00	1.00	74
chickpea	1.00	1.00	1.00	66
coconut	1.00	1.00	1.00	67
coffee	1.00	1.00	1.00	70
cotton	1.00	1.00	1.00	72
grapes	1.00	1.00	1.00	77
jute	0.99	1.00	0.99	66
kidneybeans	1.00	1.00	1.00	64
lentil	1.00	1.00	1.00	78
maize	1.00	1.00	1.00	74
mango	1.00	1.00	1.00	68
mothbeans	1.00	1.00	1.00	66
mungbean	1.00	1.00	1.00	70
muskmelon	1.00	1.00	1.00	76
orange	1.00	1.00	1.00	75
papaya	1.00	1.00	1.00	63
pigeonpeas	1.00	1.00	1.00	63
pomegranate	1.00	1.00	1.00	62
rice	1.00	0.99	0.99	72
watermelon	1.00	1.00	1.00	77
accuracy			1.00	1540
macro avg	1.00	1.00	1.00	1540
weighted avg	1.00	1.00	1.00	1540

Figure 4.18 (classification report for train)

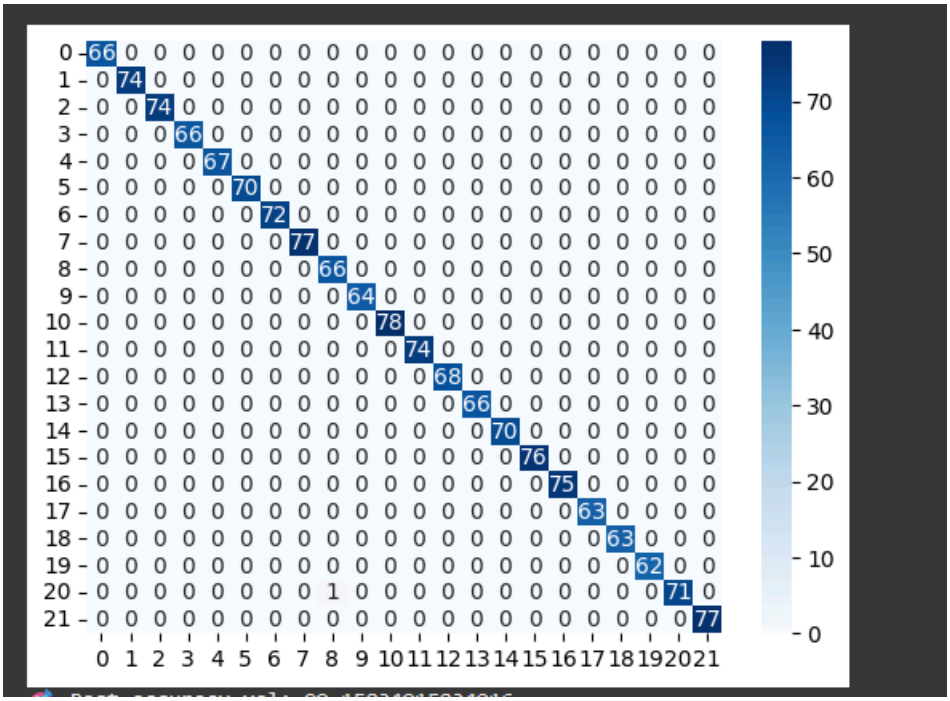


Figure 4.19 (confusion matrix for train)

```
Best accuracy_val: 99.15824915824916
```

Figure 4.20 (Validation accuracy)

	precision	recall	f1-score	support
apple	1.00	1.00	1.00	30
banana	1.00	1.00	1.00	23
blackgram	1.00	1.00	1.00	24
chickpea	1.00	1.00	1.00	32
coconut	1.00	1.00	1.00	29
coffee	1.00	1.00	1.00	27
cotton	1.00	1.00	1.00	23
grapes	1.00	1.00	1.00	20
jute	0.86	1.00	0.93	31
kidneybeans	1.00	1.00	1.00	29
lentil	1.00	1.00	1.00	17
maize	1.00	1.00	1.00	24
mango	1.00	1.00	1.00	28
mothbeans	1.00	1.00	1.00	30
mungbean	1.00	1.00	1.00	29
muskmelon	1.00	1.00	1.00	23
orange	1.00	1.00	1.00	22
papaya	1.00	1.00	1.00	33
pigeonpeas	1.00	1.00	1.00	36
pomegranate	1.00	1.00	1.00	35
rice	1.00	0.81	0.90	27
watermelon	1.00	1.00	1.00	22
accuracy			0.99	594
macro avg	0.99	0.99	0.99	594
weighted avg	0.99	0.99	0.99	594

Figure 4.21 (Classification report for validation)

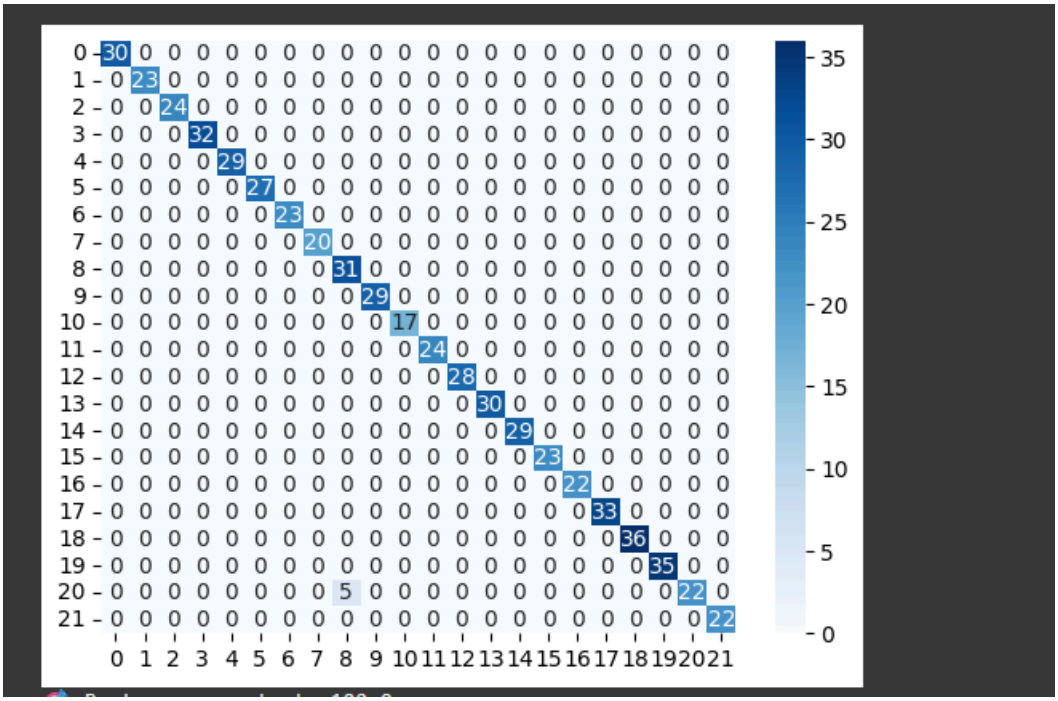


Figure 4.22 (Confusion matrix for validation)

6. Fertilizer Recommendation Model

The Fertilizer Prediction model is developed to recommend the optimal fertilizer for a given crop and soil condition. It helps in improving crop yield, sustaining soil health, and reducing excess chemical use.

- Algorithm Used: XGBoost Classifier
- Accuracy:
  - **Training Accuracy**: 99.99%
  - **Testing Accuracy**: 99.48%
- Why XGBoost?
  - It is efficient, handles missing data well, and provides high performance even on small datasets.
- Input Features:
  - **Soil Color**: Indicates organic content and soil type.
  - **Nitrogen (N)**: Critical for leaf and plant growth.
  - **Phosphorus (P)**: Supports roots and flowering.
  - **Potassium (K)**: Enhances crop quality and disease resistance.
  - **pH Level**: Influences nutrient absorption.
  - **Rainfall**: Affects nutrient availability and soil moisture.

- **Temperature**: Impacts nutrient uptake and plant processes.
- **Crop Type**: Helps tailor fertilizer to crop-specific needs.

- Feature Engineering:

  To enhance model accuracy and allow the model to capture nutrient dynamics more effectively, we introduced several new features derived from the core nutrients:

  - Total_NPK: Combined value of N, P, and K

```
df["Total_NPK"] = df["Nitrogen"] + df["Phosphorus"] + df["Potassium"]
```

Figure 4.23 (Total_NPK code)

- Ratios:

```
df["N_P_ratio"] = df["Nitrogen"] / (df["Phosphorus"] + 1)
df["N_K_ratio"] = df["Nitrogen"] / (df["Potassium"] + 1)
df["P_K_ratio"] = df["Phosphorus"] / (df["Potassium"] + 1)
```

Figure 4.24 (Ratios code)

- Differences:

```
df["N_minus_P"] = df["Nitrogen"] - df["Phosphorus"]
df["P_minus_K"] = df["Phosphorus"] - df["Potassium"]
```

Figure 4.25 (Differences code)

- Crop-Specific Deviations:

```
crop_group = df.groupby("Crop")[["Nitrogen", "Phosphorus", "Potassium"]].transform("mean")
```

Figure 4.26 (Crop-Specific Deviations code)

- Then added:

```
df["N_diff_crop"] = df["Nitrogen"] - crop_group["Nitrogen"]
df["P_diff_crop"] = df["Phosphorus"] - crop_group["Phosphorus"]
df["K_diff_crop"] = df["Potassium"] - crop_group["Potassium"]
```

Figure 4.27 (Features independent for crop group)

These engineered features helped the model better understand nutrient balance and deficiencies, significantly improving prediction reliability.

- Handling Imbalanced Data:

  The original dataset had class imbalance in fertilizer labels. To solve this, we applied SMOTE (Synthetic Minority Over-sampling Technique)

```
[ ] # Defining features (X) and target (y)
    X = df.drop(columns=["Fertilizer","District_Name","Link"])
    y = df["Fertilizer"]

[ ] from imblearn.over_sampling import SMOTE
    smote = SMOTE()
    X_resampled, y_resampled = smote.fit_resample(X, y)

[ ] # Split data into training and testing
    X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

Figure 4.28 (SMOTE and split Data)

SMOTE helped improve model generalization by ensuring the model does not become biased toward dominant classes.

- Output: The model outputs the recommended fertilizer type that best suits the given crop and soil conditions.

```
from xgboost import XGBClassifier

model = XGBClassifier(n_estimators=300, learning_rate=0.5, max_depth=5, random_state=42)
model.fit(X_train, y_train)
```

Figure 4.29 (Algorithm code)

```
y_train_pred_encoded = model.predict(X_train)
y_train_pred = le.inverse_transform(y_train_pred_encoded)
y_train_actual = le.inverse_transform(y_train)

train_accuracy = accuracy_score(y_train_actual, y_train_pred)
print(f"✅ Training Accuracy: {train_accuracy:.2%}")
print("\n📊 Classification Report (Training):\n")
print(classification_report(y_train_actual, y_train_pred))


plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_train_actual, y_train_pred), annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted Fertilizer")
plt.ylabel("Actual Fertilizer")
plt.title("Confusion Matrix - Training Data")
plt.show()
```

Figure 4.30 (Train code)

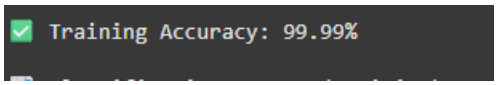


Figure 4.31 (Train Accuracy)

Classification Report (Training):

	precision	recall	f1-score	support
10:10:10 NPK	1.00	1.00	1.00	1104
10:26:26 NPK	1.00	1.00	1.00	1081
12:32:16 NPK	1.00	1.00	1.00	1080
13:32:26 NPK	1.00	1.00	1.00	1082
18:46:00 NPK	1.00	1.00	1.00	1083
19:19:19 NPK	1.00	1.00	1.00	1093
20:20:20 NPK	1.00	1.00	1.00	1077
50:26:26 NPK	1.00	1.00	1.00	1105
Ammonium Sulphate	1.00	1.00	1.00	1090
Chilated Micronutrient	1.00	1.00	1.00	1101
DAP	1.00	1.00	1.00	1119
Ferrous Sulphate	1.00	1.00	1.00	1099
Hydrated Lime	1.00	1.00	1.00	1085
MOP	1.00	1.00	1.00	1085
Magnesium Sulphate	1.00	1.00	1.00	1076
SSP	1.00	1.00	1.00	1096
Sulphur	1.00	1.00	1.00	1103
Urea	1.00	1.00	1.00	1075
White Potash	1.00	1.00	1.00	1098
accuracy			1.00	20732
macro avg	1.00	1.00	1.00	20732
weighted avg	1.00	1.00	1.00	20732

Figure 4.33 (Train Classification report)

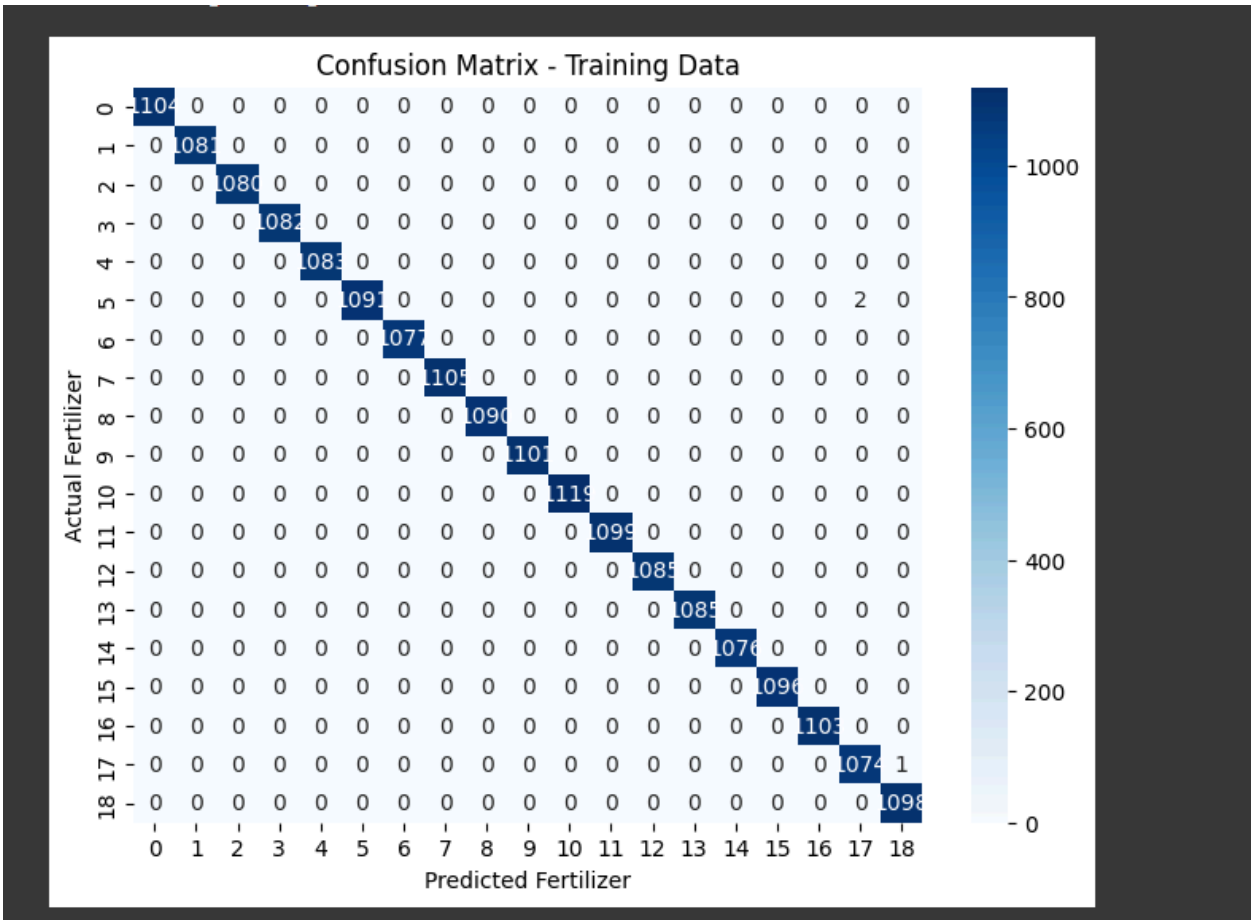


Figure 4.34 (Train confusion matrix)

4.3.2 App:

1. Login Module:

This component will be covering Login process to manage user Login and role-based access control.

Firebase:

Here a registered user will enter his email and password to be checked if user is already registered his data which is restored in firebase from the registration process.

```
private void loginWithEmailAndPassword(FirebaseAuthHelper firebaseHelper, String mail, String pass) {
    firebaseHelper.signIn(mail, pass, new FirebaseAuthHelper.AuthCallback() {
        @Override
        public void onSuccess(FirebaseUser user) {
            if (user.isEmailVerified()) {
                UserSessionHelper userSessionHelper = UserSessionHelper.getInstance( context: Login.this);
                userSessionHelper.saveUser(user, checkBox.isChecked());
                new Handler().postDelayed(Login.this::handleSuccessfulLogin, delayMillis: 2000);
            } else {
                showError("Please verify your email address!");
            }
        }

        @Override
        public void onFailure(String errorMessage) {
            runOnUiThread(() -> {
                progressBar.setVisibility(View.GONE);
                login.setEnabled(true);
                showError("Invalid Email or Password!");
            });
        }
    });
}
```

Figure 4.35 (Login check information)

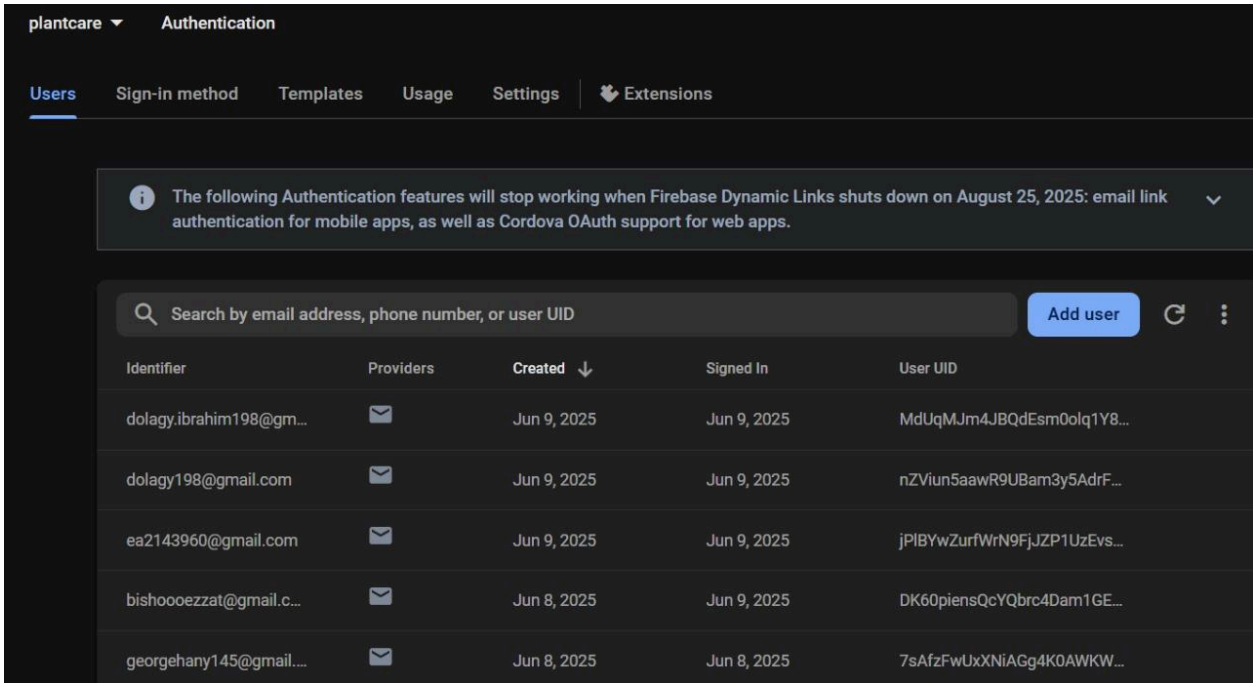


Figure 4.36 (Login in firebase)

Frontend: we create a form and interface for login using Android

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="htt
    <TextView
        android:ellipsize="marquee"
        android:maxLines="2"
        android:paddingStart="20dp"
        android:paddingTop="8dp"
        android:paddingEnd="5dp"
        android:paddingBottom="8dp"
        android:text="Incorrect email or password, Try again."
        android:textColor="@color/red"
        android:textDirection="locale"
        android:textSize="14sp"
        android:visibility="invisible"
        app:drawableStartCompat="@drawable/warning"
        app:layout_constraintBottom_toTopOf="@+id/horizontal25"
        app:layout_constraintEnd_toStartOf="@+id/vertical90"
        app:layout_constraintStart_toStartOf="@+id/vertical10"
        app:layout_constraintTop_toTopOf="@+id/horizontal25" />

    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/loginEmailField"
        style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        android:layout_width="0dp"
```

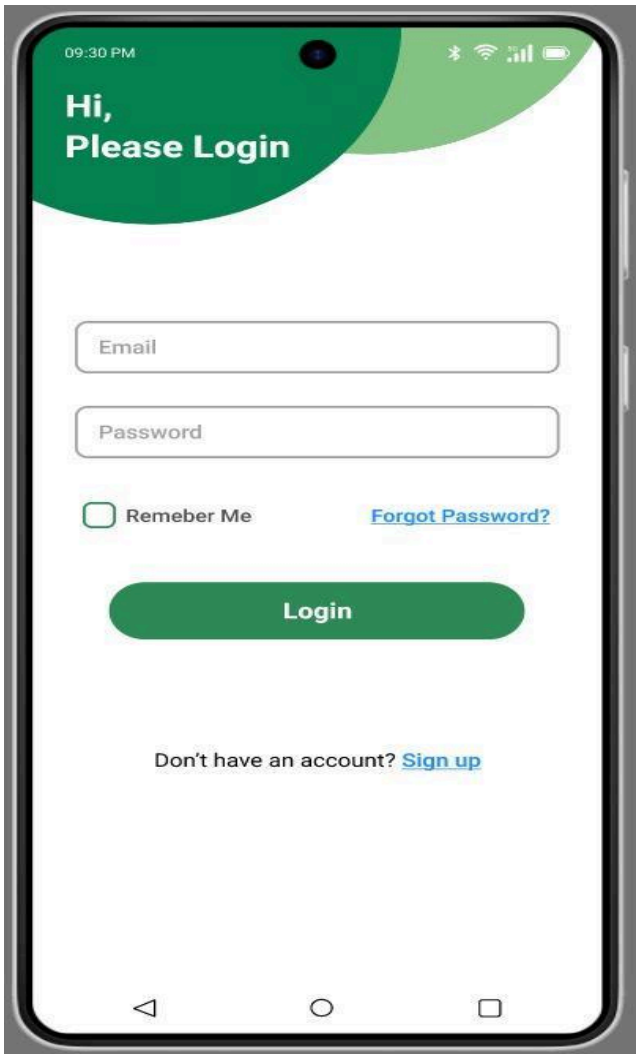


Figure 4.37 (Login for implementation) **Figure 4.38 (Login for interface)**

Backend: Here the session will start when user try to login

```
private boolean isReadyToLogin() {  1 usage
    String emailInput = email.getText().toString().trim();
    String passwordInput = password.getText().toString();

    boolean isValid = true;

    if (emailInput.isEmpty()) {
        emailField.setErrorEnabled(true);
        emailField.setError("Required!");
        isValid = false;
    } else if (!Patterns.EMAIL_ADDRESS.matcher(emailInput).matches()) {
        emailField.setErrorEnabled(true);
        emailField.setError("Invalid email format!");
        isValid = false;
    } else {
        emailField.setError(null);
        emailField.setErrorEnabled(false);
    }

    if (passwordInput.isEmpty()) {
        passwordField.setErrorEnabled(true);
        passwordField.setError("Required!");
```

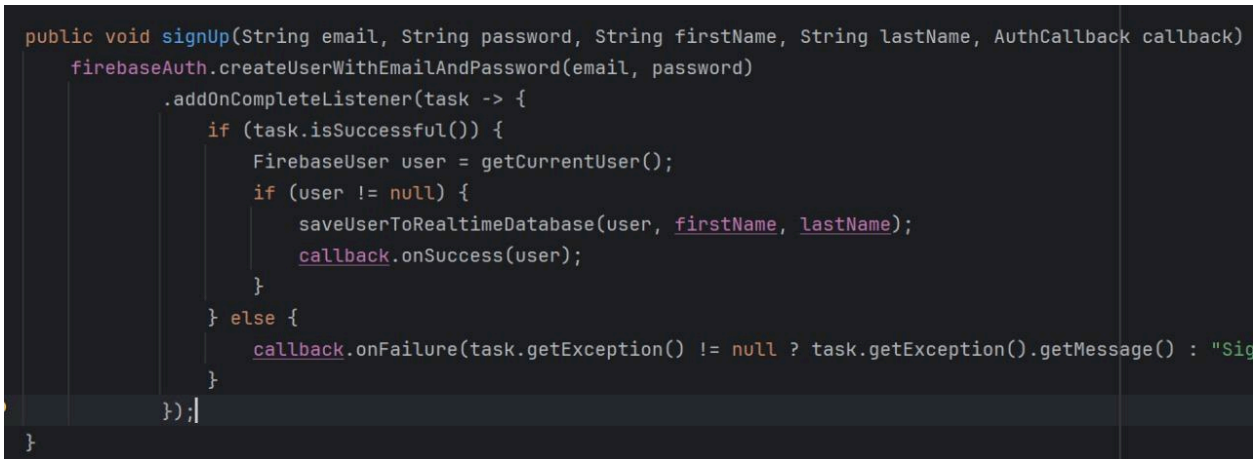
Figure 4.39 (backend for login)

2. Sign up Module:

This component will be covering signup process to manage user Signup and role-based access control

Firebase:

During sign-up, the user provides their first name, last name, email, and password. This information is securely stored using Firebase Authentication for future login and user identification.



```
public void signUp(String email, String password, String firstName, String lastName, AuthCallback callback)
    firebaseAuth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener(task -> {
                if (task.isSuccessful()) {
                    FirebaseUser user = getCurrentUser();
                    if (user != null) {
                        saveUserToRealtimeDatabase(user, firstName, lastName);
                        callback.onSuccess(user);
                    }
                } else {
                    callback.onFailure(task.getException() != null ? task.getException().getMessage() : "Sig
                }
            });
}
```

Figure 4.40 (signup check information)

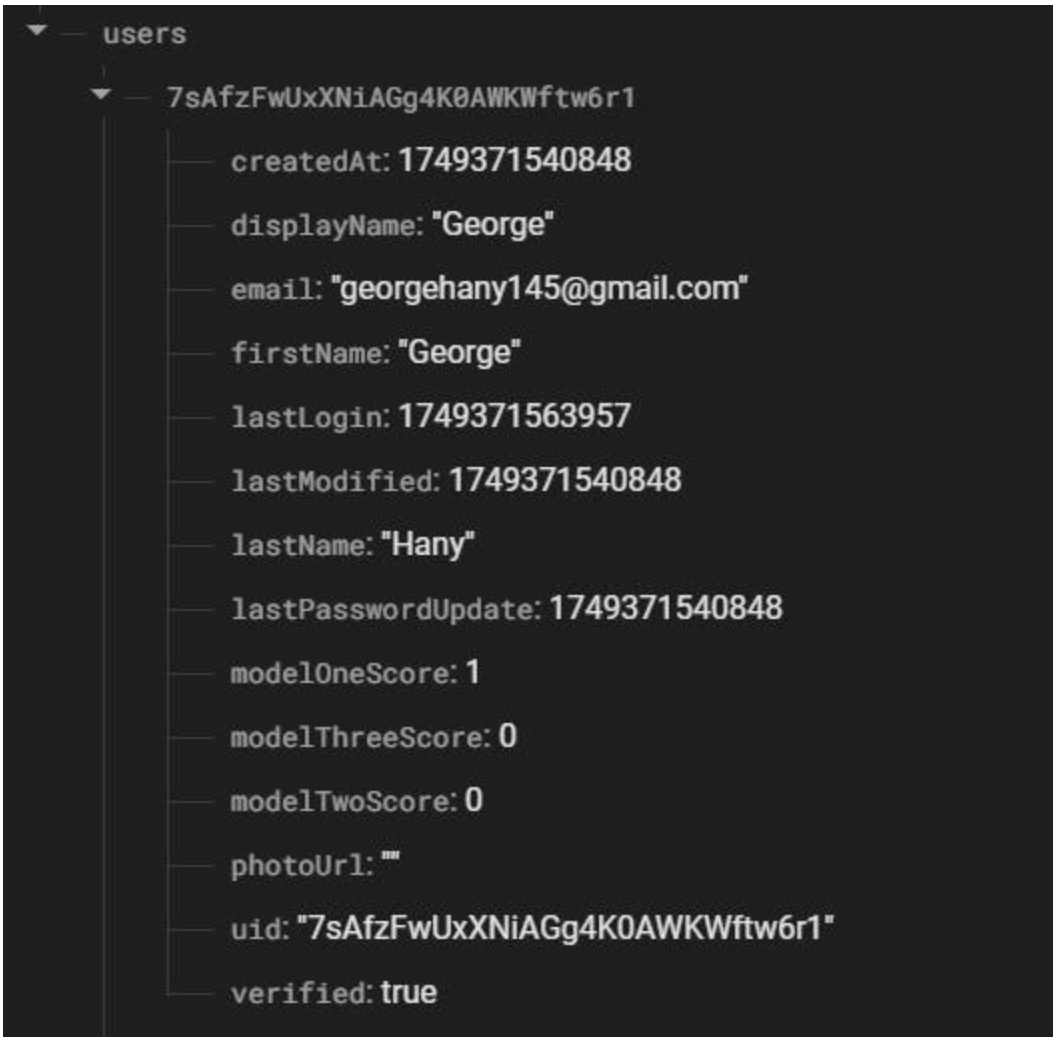


Figure 4.41 (Signup in firebase)

Frontend: we create a form and interface for Signup using Android

```
<androidx.constraintlayout.widget.Guideline
    android:id="@+id/horizontal25"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    app:layout_constraintGuide_percent="0.25" />

<ImageView
    android:id="@+id/imageView"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginTop="-10dp"
    android:paddingBottom="30dp"
    android:scaleType="centerCrop"
    app:layout_constraintBottom_toTopOf="@+id/horizontal25"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/top_background1" />
```

Figure 4.42 (signup for implementation)

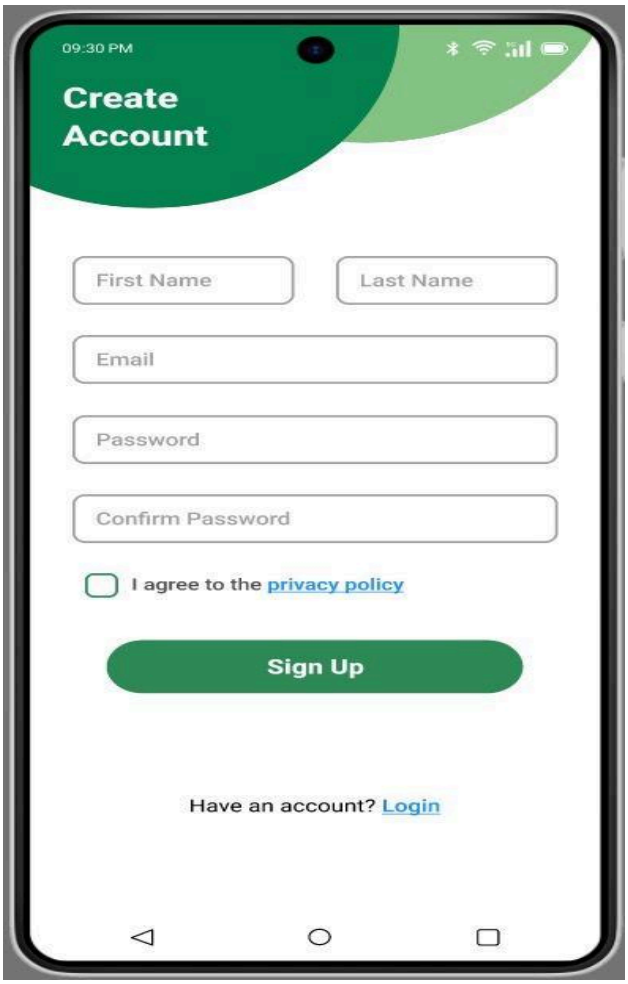


Figure 4.43 (Signup for interface)

Backend: Here the session will start when user try to create account.

```
private boolean isReadyToSignUp() {  1 usage
    boolean isValid = true;
    String fName = firstName.getText().toString().trim();
    String lName = lastName.getText().toString().trim();
    String emailInput = email.getText().toString();
    String passwordInput = password.getText().toString();
    String confirmPasswordInput = confirmPassword.getText().toString();

    if (fName.isEmpty()) {
        firstNameField.setErrorEnabled(true);
        firstNameField.setError("Required!");
        isValid = false;
    }
    if (lName.isEmpty()) {
        lastNameField.setErrorEnabled(true);
        lastNameField.setError("Required!");
        isValid = false;
    }
    if (emailInput.isEmpty()) {
        emailField.setErrorEnabled(true);
        emailField.setError("Required!");
        isValid = false;
```

Figure 4.44 (backend for Signup)

3. Plant disease detection module:

This component will be covering detection process to manage user experience and role-based access control

Firebase:

When a user uploads a plant image for disease detection, the image is stored securely in Firebase Cloud Storage

```
private void saveResultToFirebase(String imageUri, String diseaseName, String description,  1 usage
                                  String steps, String supplementName, String supplementImage) {
    FirebaseAuthHelper authHelper = FirebaseAuthHelper.getInstance();
    FirebaseFirestore db = FirebaseFirestore.getInstance();

    Log.d( tag: "Firestore",  msg: "Saving result to Firebase...");
    Map<String, Object> result = new HashMap<>();
    result.put( k: "userId", authHelper.getCurrentUser().getUid());
    result.put( k: "imageUrl", imageUri);
    result.put( k: "diseaseName", diseaseName);
    result.put( k: "description", description);
    result.put( k: "steps", steps);
    result.put( k: "supplementName", supplementName);
    result.put( k: "supplementImage", supplementImage);
    result.put( k: "timestamp", System.currentTimeMillis());

    db.collection( collectionPath: "first_model_scans") CollectionReference
            .add(result) Task<DocumentReference>
            .addOnSuccessListener((documentReference) -> {
                Log.d( tag: "Firestore",  msg: "Document added: " + documentReference.getId());
                progressBar.setVisibility(View.GONE);
            })
            .addOnFailureListener(e ->
```

Figure 4.45 (Save result to firebase)

```
ApiService apiService = ApiClient.getApiService();
RequestBody requestFile = RequestBody.create(MediaType.parse( string: "image/*"), imageFile);
MultipartBody.Part body = MultipartBody.Part.createFormData( name: "image", imageFile.getName(), requestFile);

Call<ImagePredictionResponse> call = apiService.predictWithImage(body);
call.enqueue(new Callback<>() {
    @Override
    public void onResponse(Call<ImagePredictionResponse> call, Response<ImagePredictionResponse> response) {
        Log.e( tag: "ModelOne",  msg: "API Error: " + response.body());
        if (response.isSuccessful() && response.body() != null) {
            handleApiSuccess(response.body(), imageFile);
            FirebaseHelper  firebaseHelper = new FirebaseHelper();
            firebaseHelper.increaseModelOneCount(UserSessionHelper.getInstance( context: ModelOne.this).getUser().getUID(),
            UserSessionHelper.getInstance( context: ModelOne.this).increaseModelOneScore();
        } else {
            showFailureDialog();
        }
    }

    @Override
    public void onFailure(Call<ImagePredictionResponse> call, Throwable t) {
        Log.e( tag: "ModelOne",  msg: "API Error: " + t.getMessage());
```

Figure 4.46 (response in firebase)

```
private void uploadImage(Uri imageUri, String diseaseName, String description,  1 usage
                         String steps, String supplementName, String supplementImage) {

    FirebaseStorageHelper.uploadPlantImage(imageUri, new FirebaseStorageHelper.OnImageUploadListener() {
        @Override  2 usages
        public void onSuccess(Uri downloadUri) {
            Log.d( tag: "FirebaseStorage",  msg: "Image uploaded successfully: " + downloadUri.toString());
            setAnimations();
            saveResultToFirebase(downloadUri.toString(), diseaseName, description, steps, supplementName, supplementImage);
        }

        @Override  5 usages
        public void onFailure(String errorMessage) {
            Log.e( tag: "FirebaseStorage",  msg: "Upload failed: " + errorMessage);
            Toast.makeText( context: ModelOne.this,  text: "Upload failed: " + errorMessage, Toast.LENGTH_SHORT).show();
        }
    });
}
```

Figure 4.47 (success or failure in firebase)

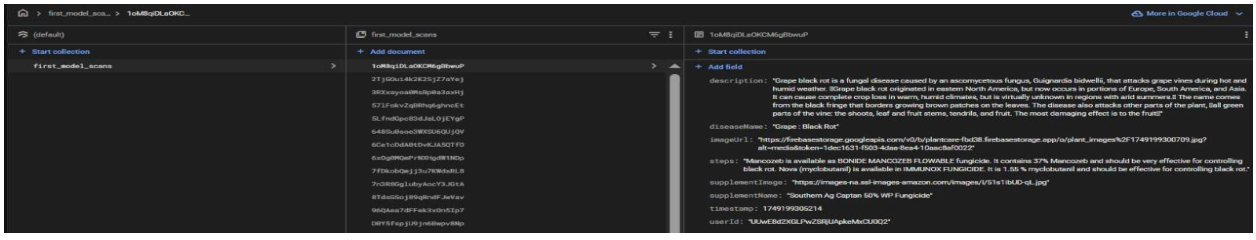


Figure 4.48 (Check in firebase)

Frontend: we create a form and interface for plant disease detection model using Android

```
<TextView
    android:id="@+id/modelOneBriefTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginBottom="8dp"
    android:ellipsize="marquee"
    android:fontFamily="sans-serif-medium"
    android:maxLines="1"
    android:text="Brief Description"
    android:textColor="@color/colorPrimary"
    android:textIsSelectable="true"
    android:textSize="18sp" />

<TextView
    android:id="@+id/modelOneBriefDescription"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

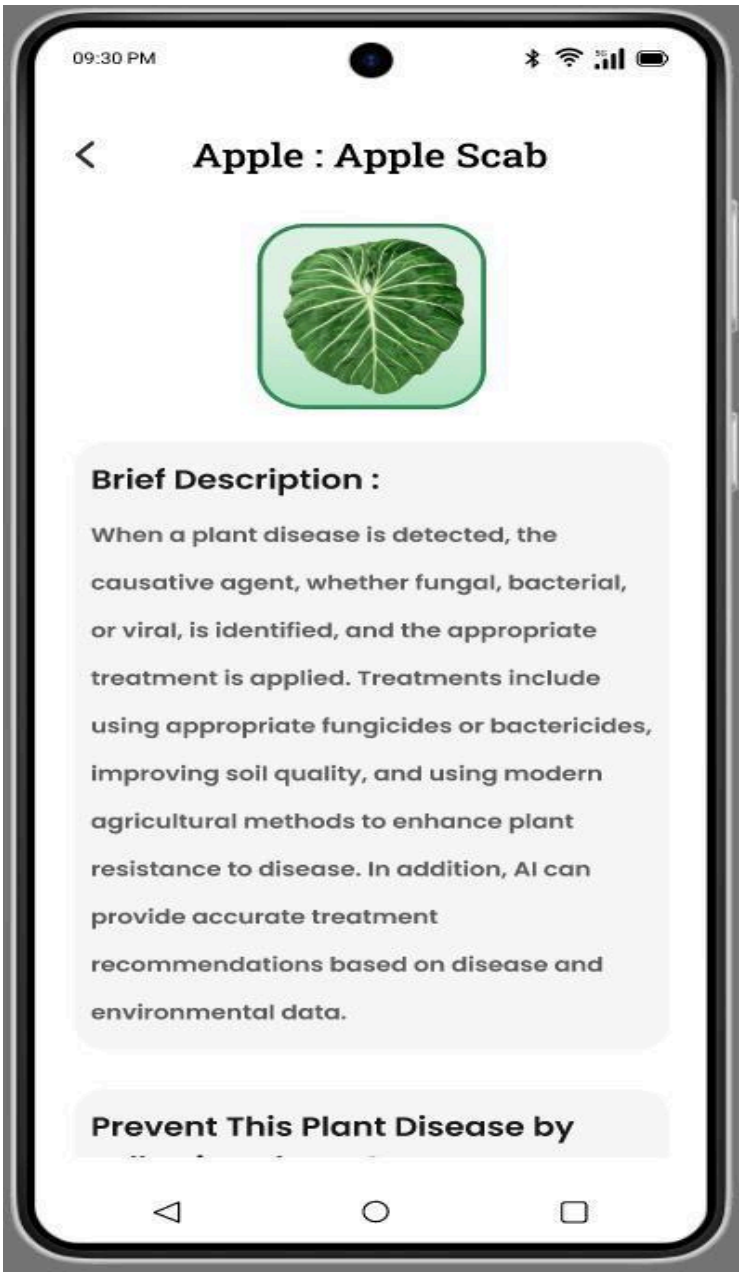


Figure 4.49 (plant disease detection for implementation)

Figure 4.50 (plant disease detection for interface)

Backend: Here the session will start when user try to use plant disease detection model.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable( $this$enableEdgeToEdge: this);
    setContentView(R.layout.model_one_result);

    initialize();
    String imagePath = getIntent().getStringExtra( name: "imagePath");
    retrieveModelOneData(imagePath);
}

public void initialize() {  1 usage
    modelOneBriefDescription2 = findViewById(R.id.modelOneBriefDescription);
    modelOneDiseaseName2 = findViewById(R.id.modelOneDiseaseName);
    modelOnePreventDescription2 = findViewById(R.id.modelOnePreventDescription);
    modelOneSupplementName2 = findViewById(R.id.modelOneSupplementName);
    modelOneSupplementImage2 = findViewById(R.id.modelOneSupplementImage);
    modelOneUserPlantImage2 = findViewById(R.id.modelOneUserPlantImage);
    progressBar = findViewById(R.id.modelOneProgressBar);
    briefCard = findViewById(R.id.briefCard);
    stepsCard = findViewById(R.id.stepsCard);
    supplementCard = findViewById(R.id.supplementCard);
}
```

Figure 4.51 (backend for plant disease detection model)

4. **Crop prediction module:**

This component will be covering prediction process to manage user experience and role-based access control

Firebase:

When a user inputs soil and environmental data (such as N, P, K, pH, temperature, humidity, and rainfall) to receive a crop recommendation, this input along with the predicted crop type is increase crop counter in the Firebase Realtime Database**.**

```
public void increaseModelTwoCount(String userId, DataCallback<Boolean> callback) {  1 usage
    DatabaseReference userModelRef = usersRef.child(userId).child( pathString: "modelTwoScore");
    userModelRef.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override  2 usages
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            long currentValue = 0;
            if (snapshot.exists()) {
                try {
                    currentValue = snapshot.getValue(Long.class);
                } catch (Exception e) {
                    Log.e(TAG,  msg: " :فشل في تحويل القيمة" + e.getMessage());
                }
            }
            long newValue = currentValue + 1;
            userModelRef.setValue(newValue)
                    .addOnSuccessListener(aVoid -> callback.onDataReceived(true))
                    .addOnFailureListener(e -> {
                        Log.e(TAG,  msg: " :فشل في تحديث القيمة" + e.getMessage());
                        callback.onDataReceived(false);
                    });
        }
```

Figure 4.52 (Check Firebase in crop prediction model)

Frontend: we create a form and interface for Crop prediction model using Android

```
<ImageButton
    android:id="@+id/btn_back2"
    android:onClick="backButton"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:scaleType="centerInside"
    android:src="@drawable/left"
    android:background="@null"
    app:layout_constraintBottom_toTopOf="@+id/guideline41"
    app:layout_constraintEnd_toStartOf="@+id/guideline42"
    app:layout_constraintStart_toStartOf="@+id/guideline39"
    app:layout_constraintTop_toTopOf="@+id/guideline32" />

<androidx.constraintlayout.widget.Guideline
    android:id="@+id/guideline42"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintGuide_percent="0.2" />
```

Figure 4.53 (Crop prediction for implementation)

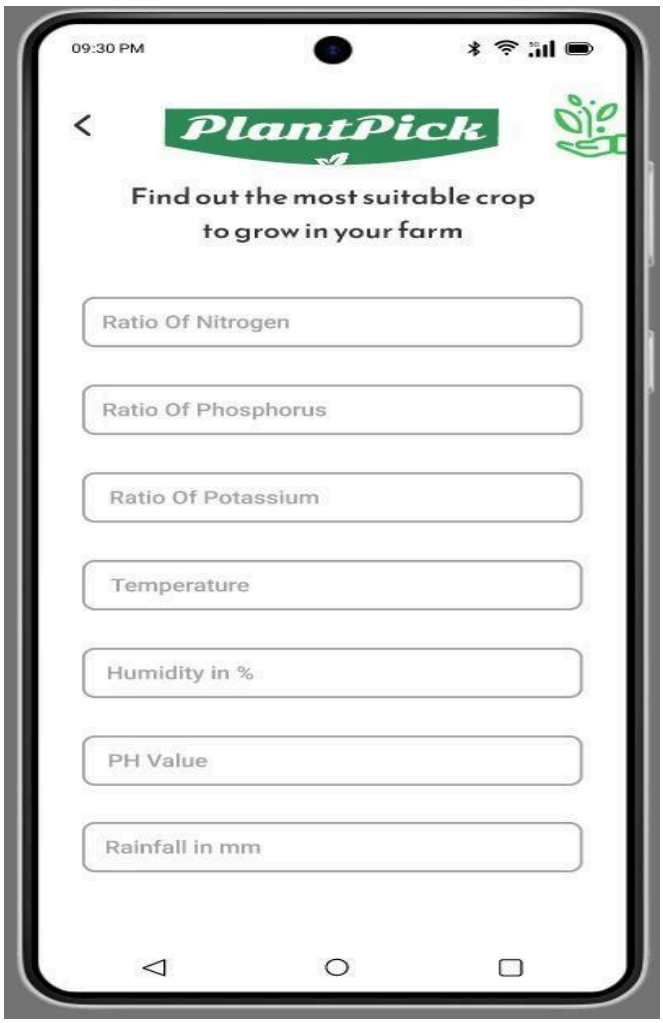


Figure 4.54 (Crop prediction for interface)

Backend: Here the session will start when user try to use Crop prediction model.

```
public void increaseModelTwoCount(String userId, DataCallback<Boolean> callback) {  1 usage
    DatabaseReference userModelRef = usersRef.child(userId).child( pathString: "modelTwoScore");
    userModelRef.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override  2 usages
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            long currentValue = 0;
            if (snapshot.exists()) {
                try {
                    currentValue = snapshot.getValue(Long.class);
                } catch (Exception e) {
                    Log.e(TAG,  msg: " :فشل في تحويل القيمة" + e.getMessage());
                }
            }
            long newValue = currentValue + 1;
            userModelRef.setValue(newValue)
                    .addOnSuccessListener(aVoid -> callback.onDataReceived(true))
                    .addOnFailureListener(e -> {
                        Log.e(TAG,  msg: " :فشل في تحديث القيمة" + e.getMessage());
                        callback.onDataReceived(false);
                    });
        }
```

Figure 4.55 (Backend for Crop prediction model)

5. Fertilizer prediction module:

This component will be covering prediction process to manage user experience and role-based access control

Firebase:

When a user inputs soil and crop-related data (such as N, P, K levels, pH, temperature, rainfall, soil color, and crop type) to receive a fertilizer recommendation, this input along with the predicted fertilizer type is used to increase the fertilizer counter in the Firebase Realtime Database.

```
public void increaseModelThreeCount(String userId, DataCallback<Boolean> callback) {  1 usage
    DatabaseReference userModelRef = usersRef.child(userId).child( pathString: "modelThreeScore");
    userModelRef.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override  2 usages
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            long currentValue = 0;
            if (snapshot.exists()) {
                try {
                    currentValue = snapshot.getValue(Long.class);
                } catch (Exception e) {
                    Log.e(TAG,  msg: " :فشل في تحويل القيمة" + e.getMessage());
                }
            }
            long newValue = currentValue + 1;
            userModelRef.setValue(newValue)
                    .addOnSuccessListener(aVoid -> callback.onDataReceived(true))
                    .addOnFailureListener(e -> {
                        Log.e(TAG,  msg: " :فشل في تحديث القيمة" + e.getMessage());
                        callback.onDataReceived(false);
                    });
        }
```

Figure 4.56 (Check Firebase in fertilizer prediction model)

Frontend: we create a form and interface for fertilizer prediction model using Android.

```
<ImageView
    android:id="@+id/imageView10"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:scaleType="centerInside"
    app:layout_constraintBottom_toTopOf="@+id/guideline41"
    app:layout_constraintEnd_toStartOf="@+id/guideline40"
    app:layout_constraintStart_toStartOf="@+id/guideline43"
    app:layout_constraintTop_toTopOf="@+id/guideline32"
    app:srcCompat="@drawable/top_right_corner" />

<ScrollView
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:padding="10dp"
    android:scrollbars="none"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toStartOf="@+id/guideline40"
    app:layout_constraintStart_toStartOf="@+id/guideline39"
    app:layout_constraintTop_toBottomOf="@+id/textView14">

    <LinearLayout
```

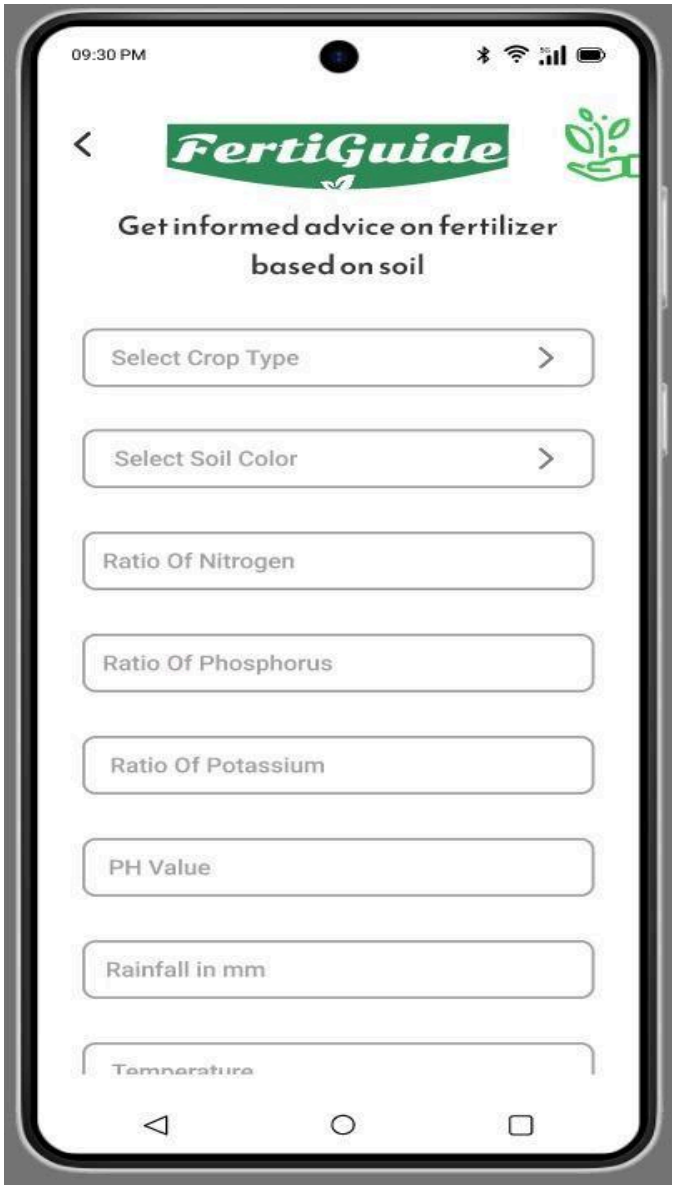


Figure 4.57 (fertilizer prediction for implementation)

Figure 4.58 (fertilizer prediction for interface)

Backend: Here the session will start when user try to use fertilizer prediction model.

```
private boolean isReadyToPredict() {  1 usage
    return validateFields(
            new TextInputEditText[]{nitrogenInput, phosphorusInput, potassiumInput, pHInput, rainfallInput,
            new TextInputLayout[]{nitrogenLayout, phosphorusLayout, potassiumLayout, pHLayout, rainfallLayou
    );
}

public static boolean validateFields(TextInputEditText[] inputs, TextInputLayout[] layouts) {  1 usage
    boolean isValid = true;
    for (int i = 0; i < inputs.length; i++) {
        if (inputs[i].getText().toString().isEmpty() || inputs[i].getText().toString().equals("-")) {
            layouts[i].setErrorEnabled(true);
            layouts[i].setError("Required!");
            isValid = false;
        } else {
            layouts[i].setErrorEnabled(false);
            layouts[i].setError(null);
        }
    }
    return isValid;
}
```

Figure 4.59 (Backend for Fertilizer prediction model)

4.4 Challenges Faced and How We Were Resolved

Throughout the development of the system, several challenges were encountered across data processing, model training, integration, and deployment. Below is a summary of key challenges and the strategies used to overcome them:

1. Image Preprocessing and Normalization
    - **Challenge**: Raw images had varying sizes and color distributions, affecting model performance.
    - **Resolution**:
        - Implemented a consistent preprocessing pipeline using `torchvision.transforms`, including resizing, tensor conversion, and normalization using ImageNet statistics.

2. **Model Overfitting**
    - **Challenge**: Initial models showed high training accuracy but low validation performance.
    - **Resolution :**

1. Introduced dropout layers and data augmentation.
2. Tuned hyperparameters such as learning rate and batch size
3. Applied early stopping and cross-validation strategies.

3. Integration of Multiple Models

- Challenge: Combining multiple prediction models (plant disease, crop, fertilizer) within a single system caused complexity in input/output handling
- Resolution:
    1. Designed modular code architecture with well-defined APIs for each model.
    2. Ensured standardized input formats and consistent preprocessing pipelines across modules.

4. Lack of Domain-Specific Data for Fertilizer Recommendation

- Challenge: Difficulty in finding clean, labeled datasets for fertilizer recommendations.
- Resolution:

1. Collected data from Kaggle and performed manual cleaning.
2. Engineered additional features to improve model performance.

Chapter 5

Testing & Evaluation

5.1 Testing strategies

To ensure the reliability, accuracy, and performance of the application and its machine learning models, the following testing strategies were applied:

- Unit Testing:
  - Focused on individual components such as the ML model functions, data preprocessing steps, and API responses. Each function was tested separately to verify its correctness and handle edge cases.
- Integration Testing:
  - Ensured the interaction between modules (ML models and Android app) works smoothly. This included testing image uploads, API responses, and how predictions are fetched and displayed in the app.
- User Testing:
  - Conducted with target users to evaluate the usability and clarity of the app. Feedback was collected regarding navigation, layout, responsiveness, and accuracy of displayed results. Adjustments were made accordingly to improve user experience.

5.2 Performance metrics

5.2.1 Accuracy:

Each machine learning model (Plant Disease Detection, Crop Recommendation, Fertilizer Recommendation) was tested using unseen data to verify accuracy, precision, and robustness before deployment.

1- Plant disease detection model:

- **Test Accuracy:** 96.30%

```
Test Accuracy: 96.30%
Average Test Loss: 0.0956

Classification Report:
                                        precision    recall  f1-score   support

                     Apple___Apple_scab       1.00      1.00      1.00         3
                 Apple___Cedar_apple_rust       1.00      1.00      1.00         4
               Background_without_leaves       0.90      1.00      0.95         9
                      Corn___Common_rust       1.00      1.00      1.00         3
             Grape___Esca_(Black_Measles)       1.00      1.00      1.00         4
           Pepper,_bell___Bacterial_spot       1.00      1.00      1.00         3
                  Potato___Early_blight       1.00      1.00      1.00         5
                      Potato___healthy       1.00      1.00      1.00         2
                Strawberry___Leaf_scorch       1.00      1.00      1.00         3
                    Strawberry___healthy       1.00      1.00      1.00         3
                   Tomato___Early_blight       1.00      0.60      0.75         5
                    Tomato___Late_blight       0.00      0.00      0.00         0
Tomato___Tomato_Yellow_Leaf_Curl_Virus       1.00      1.00      1.00         6
                      Tomato___healthy       1.00      1.00      1.00         4

                              accuracy                           0.96        54
                             macro avg       0.92      0.90      0.91        54
                          weighted avg       0.98      0.96      0.97        54
```

Figure 5.1 (Test accuracy and classification report for Plant disease detection model)

2- Crop recommendation system:

- **Test Accuracy:** 100%

Best accuracy_test: 100.0

	precision	recall	f1-score	support
apple	1.00	1.00	1.00	4
banana	1.00	1.00	1.00	3
blackgram	1.00	1.00	1.00	2
chickpea	1.00	1.00	1.00	2
coconut	1.00	1.00	1.00	4
coffee	1.00	1.00	1.00	3
cotton	1.00	1.00	1.00	5
grapes	1.00	1.00	1.00	3
jute	1.00	1.00	1.00	3
kidneybeans	1.00	1.00	1.00	7
lentil	1.00	1.00	1.00	5
maize	1.00	1.00	1.00	2
mango	1.00	1.00	1.00	4
mothbeans	1.00	1.00	1.00	4
mungbean	1.00	1.00	1.00	1
muskmelon	1.00	1.00	1.00	1
orange	1.00	1.00	1.00	3
papaya	1.00	1.00	1.00	4
pigeonpeas	1.00	1.00	1.00	1
pomegranate	1.00	1.00	1.00	3
rice	1.00	1.00	1.00	1
watermelon	1.00	1.00	1.00	1
accuracy			1.00	66
macro avg	1.00	1.00	1.00	66
weighted avg	1.00	1.00	1.00	66

Figure 5.2 (Test accuracy and classification report for Crop recommendation system)

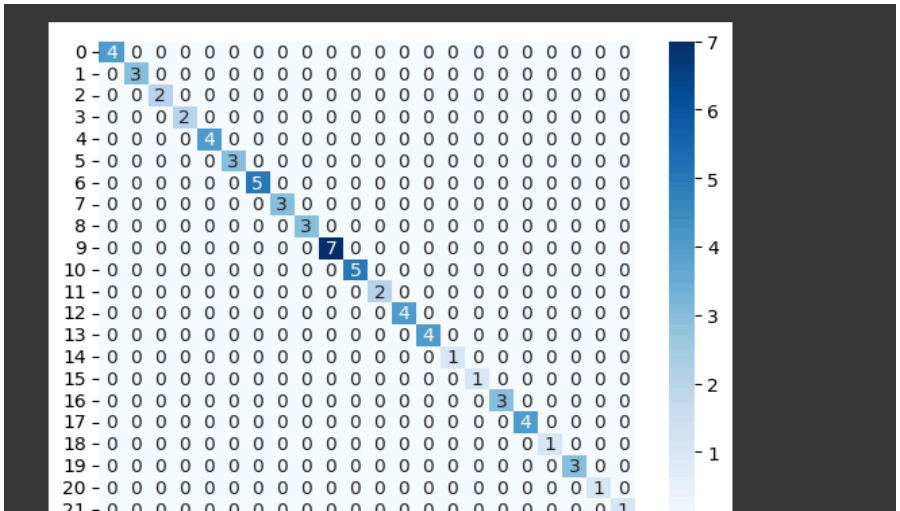


Figure 5.3 (confusion matrix for Crop recommendation system)

3- Fertilizer recommendations system:

- Test Accuracy: 99.48%

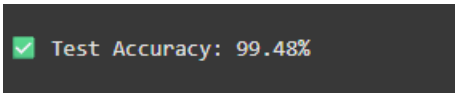


Figure 5.4 (Test accuracy for fertilizer model)

Classification Report (Test):

	precision	recall	f1-score	support
10:10:10 NPK	1.00	1.00	1.00	260
10:26:26 NPK	1.00	1.00	1.00	283
12:32:16 NPK	1.00	1.00	1.00	284
13:32:26 NPK	1.00	1.00	1.00	282
18:46:00 NPK	1.00	1.00	1.00	281
19:19:19 NPK	0.99	0.98	0.98	271
20:20:20 NPK	1.00	1.00	1.00	287
50:26:26 NPK	1.00	1.00	1.00	259
Ammonium Sulphate	0.99	1.00	1.00	274
Chilated Micronutrient	1.00	1.00	1.00	263
DAP	1.00	0.98	0.99	245
Ferrous Sulphate	1.00	0.99	0.99	265
Hydrated Lime	1.00	1.00	1.00	279
MOP	0.98	0.99	0.99	279
Magnesium Sulphate	1.00	1.00	1.00	288
SSP	0.99	0.99	0.99	268
Sulphur	1.00	1.00	1.00	261
Urea	0.98	0.98	0.98	289
White Potash	0.99	1.00	0.99	266
accuracy			0.99	5184
macro avg	0.99	0.99	0.99	5184
weighted avg	0.99	0.99	0.99	5184

Figure 5.5 (Classification report for fertilizer model)

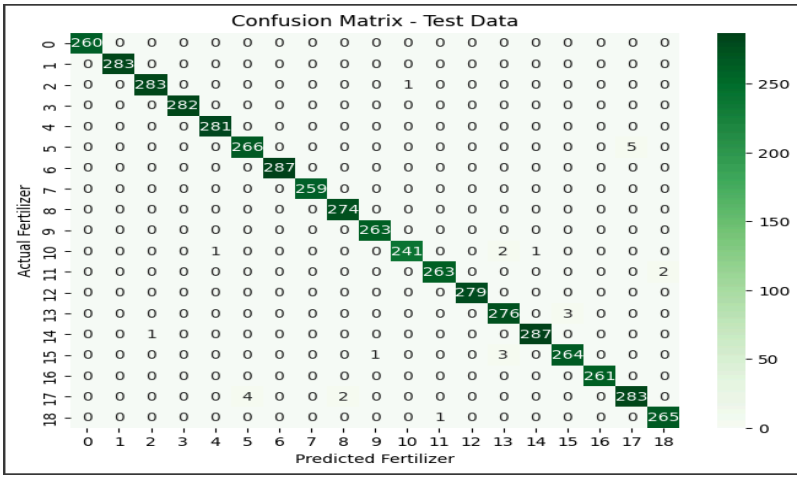


Figure 5.6 (Confusion matrix for fertilizer model)

5.2.2 Speed:

- Prediction Time:
  - The models were optimized for fast inference, ensuring near real-time predictions on mobile devices.
  - API integration were used to reduce latency
- Image Upload and Processing Time:
  - Firebase Cloud Storage and optimized image handling ensured quick upload and classification.

5.●.3 **Scalability**:

- Firebase was used to manage authentication, database, and storage, providing a scalable infrastructure capable of handling many concurrent users.
- The system design supports adding more models or expanding the dataset in future updates.

5.●.4 Resource Efficiency:

- Pre-trained models such as ResNet50 were fine-tuned to reduce memory usage while maintaining accuracy.
- Feature engineering and SMOTE balancing techniques helped boost model performance without requiring additional data.

5.●.3 User Feedback (User-Centric Performance):

- Usability and prediction performance were evaluated during user testing sessions.
- feedback was incorporated to improve both model predictions and app design.

Chapter 6

Results & Discussion

6.1 Introduction

This chapter presents the outcomes of implementing and testing the proposed mobile application integrated with three AI-based models: Plant Disease Detection, Crop Recommendation, and Fertilizer Recommendation. Each model was evaluated for performance, accuracy, and practical usability. We also discuss how these results support the goals of improving agricultural productivity and assisting farmers with informed decision-making**.**

6.2 Summary of findings.

Plant Disease Detection Model:

- Model Used: ResNet50 (PyTorch)
- Achieved Training Accuracy: 99.33%
- Validation Accuracy: 99.42%
- Test Accuracy: 96.30%
- Result: The model effectively identified 38 plant diseases from leaf images, showing high generalization on unseen data.

Crop Recommendation Model:

- Model Used: Random Forest Classifier
- Achieved Training Accuracy: 99.93%
- Validation Accuracy: 99.15%
- Test Accuracy: 100%

- Result: Successfully recommended the most suitable crop based on soil nutrient content, pH, rainfall, and temperature.

Fertilizer Recommendation Model:

- Model Used: XGBoost Classifier
- Training Accuracy: 99.99%
- Test Accuracy: 99.48%
- Result: Accurately suggested the best fertilizer based on enhanced engineered features like NPK ratios and crop-specific nutrient needs.

Mobile Application Performance:

- The integration of ML models into the Android app using APIs ensured smooth operation.
- Firebase Authentication and Realtime Database allowed secure login and result storage.
- Users can upload images, receive results, and track history efficiently.

6.3 Interpretation of results

The primary objective of the project was to develop a mobile application that assists farmers in diagnosing plant diseases, recommending suitable crops, and suggesting appropriate fertilizers based on soil and environmental data. The integration of three machine learning models into a user-friendly Android application successfully achieved this goal.

- The Plant Disease Detection model achieved high performance, with over 96% test accuracy, effectively classifying 38 types of plant diseases using a pretrained ResNet50 model.
- The Crop Recommendation model using Random Forest reached 100% test accuracy, confirming its ability to make reliable suggestions based on key agricultural features.
- The Fertilizer Recommendation model, enhanced through feature engineering and built with XGBoost, achieved an impressive 99.48% test accuracy, meeting expectations in terms of precision and reliability.

Overall, the application met its functional and technical objectives by:

- Delivering accurate predictions.

- Offering real-time feedback to users.
- Ensuring secure data storage and user authentication via Firebase.
- Providing a scalable solution adaptable to future model updates or additional features.

6.4 Limitations of the proposed solution.

Despite the success of the project, a few limitations were identified:

- Dataset Limitations: The models were trained on publicly available datasets which may not fully reflect local conditions
- Internet Dependency: While the app processes predictions through lightweight APIs, a stable internet connection is required for full functionality, especially for Firebase services.
- Limited Crop and Disease Coverage: Although the models handle a wide range of crops and diseases, they are not exhaustive. Some local or rare conditions may not be recognized.
- Language Accessibility: The application currently supports a single language interface. Broader accessibility could be enhanced by supporting multiple local languages.

1

Chapter 7

Conclusion & Future Work

7.1 Conclusion

Since the last few years, precision agriculture has become a growing necessity, especially in regions where access to agricultural experts is limited, and productivity is highly dependent on timely interventions. Farmers often face challenges in identifying plant diseases, choosing the right crops, and applying the proper fertilizers, which ultimately affects yield and sustainability. With the increasing adoption of mobile technology, we saw an opportunity to empower farmers with an intelligent solution that runs right from their phones.

To address these critical issues, we developed Planty Care, an AI-powered Android application designed to act as a virtual agricultural assistant. For plant disease detection and treatment recommendation, we used a fine-tuned ResNet-50 model, trained to classify diseases based on leaf images and provide tailored treatment solutions. For crop prediction based on soil type and environmental conditions, we implemented a Random Forest model due to its robustness and interpretability in handling structured data. To guide farmers in selecting the appropriate fertilizer based on nutrient deficiencies, we used XGBoost, a powerful gradient boosting algorithm known for its accuracy and performance.

Each model was carefully trained, validated, and optimized to suit real-world conditions, considering the challenges that farmers face in rural and data-limited environments. The application was also designed with an

easy-to-use interface and multiple features such as image-based diagnosis, soil-specific crop suggestions, and intelligent fertilizer matching to ensure that users receive reliable, fast, and actionable recommendations.

By combining deep learning and machine learning with accessible mobile technology, Planty Care contributes toward smarter farming practices, aiming to improve productivity, reduce loss, and support sustainable agriculture — especially in developing regions where such tools can make the biggest difference.

7.2 Future Work :

We aim to enhance Planty Care by integrating advanced smart agricultural technologies and AI capabilities. Future improvements may include real-time sensor integration (such as soil moisture, pH, temperature, and light sensors), AI-based image analysis for plant growth and disease detection, and GPS-powered alerts for potential outbreaks. We also plan to implement real-time camera scanning for instant disease diagnosis, develop a smart scheduling system to assist in crop management, and track disease progression over time through timeline-based reports. Additionally, an in-app agricultural marketplace will be introduced, allowing users to purchase recommended products directly. These

features will not only improve the accuracy of predictions and recommendations, but also make the system more interactive, proactive, and accessible for all farmers, regardless of their technical background.

References

- J. Ma et al. (2018) – Recognition method for cucumber diseases using deep CNN (Comput. Electron. Agric.)
- K.P. Ferentinos (2018) – Deep learning models for plant disease detection and diagnosis (Comput. Electron. Agric.)
- J. Lu et al. (2017) – In-field automatic wheat disease diagnosis system (Comput. Electron. Agric.)
- Y. Lu et al. (2017) – Identification of rice diseases using deep CNNs (Neurocomputing)
- B. Liu et al. (2017) – Identification of apple leaf diseases based on deep neural networks (Symmetry)
- J.G.A. Barbedo (2018) - Factors influencing deep learning for plant disease recognition (Biosystems Engineering).
- Horea Mureșan et al. (2018) - Fruit recognition using deep learning (Acta Universitatis Sapientiae Informatica).
- S.H. Lee (2015) - Deep-plant: plant identification with CNNs (IEEE International Conference on Image Processing).
- Krizhevsky et al. (2012). "ImageNet Classification with Deep Convolutional Neural Networks"