

**Disease Detection In Crops**

**Team No - A9**

|  |  |  |
| --- | --- | --- |
| **Sl.no** | **Names** | **SRN** |
| 1 | Rohit Rathod | 02FE22BCS085 |
| 2. | Sakshi Sonoli | 02FE22BCS095 |
| 3. | Tanvi Patil | 02FE22BCS166 |
| 4. | Satvik Bellad | 02FE22BCS113 |

**Under the guidance of : Prof. Asharani Patil**

**Faculty Incharge : Prof. Shankar Biradar**

Department of Computer Science and Engineering,

### KLE Technological University’s

**Dr. M. S. Sheshgiri College of Engineering and Technology, Belagavi – 590 008**

**DECLARATION**

We hereby declare that the matter embodied in this report entitled “Disease Detection In Crops” submitted to KLE Technological University for the course completion of Mini Project (15ECSW301) in the 5th Semester of Computer Science and Engineering is the result of the work done by us in the Department of Computer Science and Engineering, KLE Dr. M. S. Sheshgiri College of Engineering, Belagavi under the guidance of Prof. Asharani Patil, Department of Computer Science and Engineering. We further declare that to the best of our knowledge and belief, the work reported here in doesn’t form part of any other project on the basis of which a course or award was conferred on an earlier occasion on this by any other student(s), also the results of the work are not submitted for the award of any course, degree or diploma within this or in any other University or Institute. We hereby also confirm that all of the experimental work in this report has been done by us.

Belagavi – 590 008 Date :08-01-2025

Rohit Rathod Sakshi Sonoli

(02FE22BCS085) (02FE22BCS095)

Tanvi Patil Satvik Bellad

(02FE22BCS166) (02FE22BCS113)

**CERTIFICATE**

This is to certify that the project entitled “Disease Detection In Crops” submitted to KLE Technological University’s Dr. MSSCET, Belagavi for the partial fulfillment of the requirement for the course - Mini Project (15ECSW301) by ”Rohit Rathod, Sakshi Sonoli , Tanvi Patil ,Satvik Bellad ” students in the Department of Computer Science and Engineering, KLE Technological University’s Dr. MSSCET, Belagavi, is a bonafide record of the work carried out by them under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any other course completion.

Belagavi – 590 008 Date :08-01-2025

Prof. Asharani Patil Prof. Shankar Biradar

(Course Mentor) (Course Coordinator)

Dr. Rajashri Khanai (Head of the Department)

**ABSTRACT**

Agriculture plays a vital role in sustaining the global economy and ensuring food security. However, plant diseases remain a persistent challenge, causing substantial losses in crop yield and quality. This project aims to develop an intelligent, AI-powered plant disease detection system that empowers farmers by accurately identifying plant diseases and providing actionable treatment suggestions. Leveraging deep learning models and a robust dataset of plant images, the system analyzes plant leaf images to detect diseases with high precision.

The system integrates seamlessly with an Android-based mobile application, offering user-friendly functionalities such as image-based disease detection, detailed disease insights, and personalized treatment recommendations. Farmers can upload images of diseased plants, receive real-time diagnoses, and access curated suggestions to mitigate the impact of diseases. Additionally, the app features a secure user authentication system to protect user data and ensure personalized interaction.

By bridging the gap between modern technology and traditional farming practices, this project enhances agricultural productivity, reduces dependency on expert intervention, and promotes sustainable farming. The solution is scalable, adaptable to various crops, and serves as a comprehensive tool for empowering farmers in their fight against plant diseases.

**CONTENTS**

1. **Introduction** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
   1. [Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1](#_TOC_250024)
   2. [Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1](#_TOC_250023)
      1. [Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1](#_TOC_250022)
2. **Literature Survey** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
3. **Design Space** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
   1. Understanding the Problem . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
      1. [Problem Space . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8](#_TOC_250021)
      2. [State of Art . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8](#_TOC_250020)
      3. [Internal External Factors: . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10](#_TOC_250019)
   2. #BP 1.2 PROBLEM FLOW . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
      1. [Solution Space . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14](#_TOC_250018)
      2. [Features . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15](#_TOC_250017)
      3. [Modules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15](#_TOC_250016)
      4. [Relationship between modules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16](#_TOC_250015)
      5. [Society . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17](#_TOC_250014)
      6. End Users . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
      7. Privacy and Security . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
4. Requirements Engineering. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
   1. [Actors. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21](#_TOC_250013)
   2. [General Requirements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21](#_TOC_250012)
      1. [Infrastructure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21](#_TOC_250011)
      2. [Competitor . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21](#_TOC_250010)
   3. [Functional and Non Functional Requirements . . . . . . . . . . . . . . . . . . . . . . . 23](#_TOC_250009)
      1. [Functional Requirements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23](#_TOC_250008)
      2. Non Functional Requirements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
   4. [Use Case Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25](#_TOC_250007)

**5: System Modeling** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26

* 1. [UML Diagrams . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26](#_TOC_250006)
     1. Component Diagrams . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
     2. Activity Diagrams . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
     3. Sequence Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
     4. [State Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38](#_TOC_250005)
  2. Model Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43

**6: Implementation** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44

* 1. [Algorithm Development . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44](#_TOC_250004)
  2. [User interface design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48](#_TOC_250003)

**7: Testing** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50

* 1. [Testing report . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50](#_TOC_250002)
  2. [Testing Tools . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55](#_TOC_250001)

**Results and outcomes** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56

[**Conclusion** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 58](#_TOC_250000)

Photos of project execution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56

# List of Tables

7.1 Test Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52

# List of Figures

1. Use Case Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
2. Component Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
3. Activity Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
4. Sequence diagram 32
5. State Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . 38
6. Login Page 48
7. Signup Page 48
8. Main Page 49
9. Home Page 49
10. User Main Page. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
11. Database and Model Result . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57

## Chapter 1: Introduction

### Background:

Plant diseases pose a significant threat to agricultural productivity, leading to economic losses and reduced food security. Early and accurate detection of plant diseases is essential for effective management and prevention of crop damage. This project utilizes a MobileNet-based AI model integrated into a mobile application to enable real-time disease detection and provide actionable treatment suggestions to farmers.

### Problem Statement

To design a mobile application that leverages AI-based plant disease detection, providing farmers with an accessible, efficient, and reliable solution to identify plant diseases directly from leaf images. This system aims to offer real-time, accurate diagnoses and treatment suggestions, eliminating the need for expert analysis or costly laboratory tests, and helping farmers make informed decisions to mitigate crop damage effectively

### Objectives

The objective of this project is to develop a mobile application that utilizes a MobileNet-based AI model to detect plant diseases from leaf images. The app will integrate TensorFlow Lite to enable real-time disease detection, ensuring quick and accurate results. By leveraging AI, the app aims to provide accurate disease identification and relevant treatment suggestions to farmers, helping them manage their crops effectively. Additionally, the app will maintain a recent history of disease predictions, allowing users to reference recent results for future decision-making.

## Chapter 2: Literature Survey

## 1.Detection of Plant Diseases Using Machine Learning and Deep Learning Approaches

1. Author(s): C. Jackulin, S. Murugavalli
2. Proposed System: This study highlights the use of machine learning (ML) and deep learning (DL) techniques to detect and classify plant diseases. By leveraging AI-based image processing methods, it provides a solution for identifying diseases in plant leaves using visual symptoms, ultimately enhancing crop yield and reducing agricultural losses.
3. Performance: The study compares various ML and DL models, noting that DL methods, such as Convolutional Neural Networks (CNNs), outperform traditional ML techniques in accuracy and robustness. The use of large datasets and GPU computational power has further boosted the efficiency of DL models.
4. Advantages:

* Improved accuracy in early disease detection.
* Reduced manual labor and reliance on expert pathologists.
* Scalable solutions for large-scale farming operations.

1. Disadvantages:

* High computational and memory requirements for training DL models.
* Performance may degrade in scenarios with insufficient or poor-quality training data.

**2.Deep Learning Enabled Multi-Class Plant Disease Detection Model**

1. Author(s): Arunabha M. Roy and Jayabrata Bhaduri
2. Proposed System: This study proposes a deep learning-based model using an optimized version of the YOLOv4 algorithm to detect multiple classes of plant diseases, specifically targeting apple plants. The model incorporates enhancements in feature extraction and spatial pooling to improve detection accuracy and speed in complex orchard environments.
3. Performance: The model achieved a mean average precision (mAP) of 91.2% and an F1-score of 95.9% with a detection rate of 56.9 frames per second (FPS). These results outperform the original YOLOv4 model, with improvements in precision (+9.05%) and F1-score (+7.6%).
4. Advantages:

* High accuracy in detecting fine-grained, multi-scale disease patterns.
* Efficient real-time detection capabilities in complex scenarios.
* Enhanced robustness through data augmentation and modified network architecture.

1. Disadvantages:

* Increased computational complexity compared to traditional models.
* Slight compromise in detection speed for achieving higher accuracy.

**3.Imaging Techniques for Plant Disease Detection**

1. Author(s): Vijai Singh, Namita Sharma, Shikha Singh
2. Proposed System: This paper reviews various imaging techniques for identifying and classifying plant diseases. Techniques such as thermal imaging, hyperspectral imaging, fluorescence imaging, and 3D imaging are explored to provide a comprehensive understanding of their effectiveness in detecting plant diseases early. These methods leverage advancements in computer vision and machine learning for precision agriculture.
3. Performance: The reviewed imaging systems and classification models demonstrate high accuracy and early detection capabilities, crucial for mitigating crop losses. Techniques like hyperspectral imaging combined with deep learning models have shown significant potential in handling complex disease patterns.
4. Advantages:

* Early and accurate detection of diseases in crops.
* Reduction in agricultural losses through timely interventions.
* Scalability for use in large agricultural setups.

1. Disadvantages:

* High costs associated with imaging equipment and computational requirements.
* Limited efficacy in outdoor conditions with varying environmental factors.

**4.Deep Learning Techniques for Plant Disease Diagnosis**

1. Author(s): Aanis Ahmad, Dharmendra Saraswat, Aly El Gamal
2. Proposed System: This study surveys the use of deep learning techniques for diagnosing plant diseases. It highlights the integration of Convolutional Neural Networks (CNNs) with various datasets, emphasizing their ability to detect, classify, and estimate the severity of diseases across different crops.
3. Performance: Deep learning models have demonstrated high accuracy (up to 99%) in identifying plant diseases using RGB imagery, especially in controlled environments. However, cross-dataset generalization remains a challenge, with accuracies dropping significantly when tested on field-acquired images.
4. Advantages:

* These techniques enable automated and efficient disease diagnosis, reducing dependence on manual inspections.
* They support early detection, multi-disease identification, and severity estimation, contributing to precision agriculture.

1. Disadvantages:

* High computational requirements and the need for extensive labeled datasets limit scalability. Additionally, the models struggle with real-world conditions, such as varying field environments, due to poor generalization.

**5.Advanced Methods of Plant Disease Detection**

* 1. Author(s): Federico Martinelli, Riccardo Scalenghe, Salvatore Davino, et al.
  2. Proposed System: This paper reviews modern techniques for plant disease detection, emphasizing methods like DNA-based, serological, and remote sensing technologies. It explores innovative approaches such as VOC analysis, biosensors, and spectroscopy to enhance early detection capabilities.
  3. Performance: Advanced methods like spectroscopy and VOC profiling offer precise early-stage diagnosis and rapid results, surpassing traditional methods like visual inspections and serological assays in terms of speed and accuracy. However, their effectiveness may vary depending on environmental conditions and the stage of infection.
  4. Advantages:
  + These techniques provide non-invasive, rapid, and scalable solutions for plant health monitoring.
  + They also reduce the economic impact by enabling early intervention.
  1. Disadvantages:
  + High costs, technical complexity, and the requirement for specialized equipment limit their widespread adoption.
  + The scalability of these methods is further constrained by environmental and sample variability.

**6.Advances in Plant Disease Detection and Monitoring**

* 1. Author(s): Ilaria Buja, Erika Sabella, Anna Grazia Monteduro, et al.
  2. Proposed System: This paper reviews diagnostic advancements for plant disease detection, emphasizing portable, non-invasive, and IoT-enabled technologies. It discusses methods like VOC profiling, wearable sensors, and microfluidic devices, alongside traditional assays.
  3. Performance: These technologies offer early detection with high sensitivity and specificity. Wearable and IoT devices enable real-time, on-field analysis. However, performance may be hindered by environmental variables and the complexity of field conditions.
  4. Advantages:
  + Enhanced diagnostic precision and speed facilitate timely interventions.
  + Integration with IoT systems supports precision agriculture, reducing pathogen spread and crop losses.
  1. Disadvantages:
  + High costs, technical complexity, and dependency on specialized equipment or infrastructures limit accessibility.
  + Field applications can face challenges due to environmental interference and variability in sample quality.

## Chapter 3: Design Space:

## Problem Space:

Agriculture, being the backbone of many economies, is plagued by the persistent challenge of plant diseases. These diseases significantly impact crop yields, leading to food shortages, financial losses for farmers, and disruptions in the agricultural supply chain. Farmers often rely on traditional methods to detect and address diseases, which are prone to delays and inaccuracies. Many lack access to expert agronomists or resources to implement timely solutions, leaving them vulnerable to recurring outbreaks.

Furthermore, traditional approaches to disease detection are labor-intensive, subjective, and inconsistent. The overuse of chemical pesticides due to misdiagnoses not only harms the environment but also increases the cost of production, reducing profitability for farmers. Hence, there is a critical need for a system that combines precision, accessibility, and ease of use to detect plant diseases early and suggest actionable treatments.

### State of Art:

**Understanding the Role of Collaboration in Learning**

Collaborative learning environments are pivotal for enhancing understanding and engagement among students. There is a proven positive impact on academic performance when students engage in structured group activities. By working together, learners not only exchange ideas but also improve critical thinking skills and develop solutions that are more innovative and robust.

**Challenges in Existing Platforms**

1. **Social Media Tools**

* While accessible and familiar, social media tools lack dedicated academic features such as real-time collaboration, structured learning aids, and moderated environments.

1. **Video Conferencing Applications**

* Tools like Zoom and Google Meet are designed for formal meetings rather than collaborative study. They lack shared workspaces, integrated repositories for resources, and tools like collaborative document editing.

1. **Discussion Boards and Online Forums**

* These cater to asynchronous communication, making them unsuitable for real-time collaboration and interactive study sessions.

**Requirements for Effective Study Platforms**

To overcome the challenges above, effective platforms should include:

* Real-time communication tools (text, voice, and video) with features like screen sharing and document editing.
* Flexibility for group creation and management to accommodate varying schedules and needs.
* Intuitive user interfaces designed to ensure usability for individuals with different levels of technical expertise.
* Integrated resource repositories for easy sharing and access to academic content.

**Technological Innovations Observed**

1. **Cloud-Based Infrastructure**

* Scalable cloud platforms support seamless handling of multiple users and ensure low-latency interactions, making them ideal for collaborative learning tools.

1. **AI-Driven Recommendations**

* Artificial intelligence is being increasingly used to recommend study groups, resources, and learning paths based on a user’s interests, academic progress, and learning style.

**Safety and Moderation**

To maintain a respectful and effective learning environment, platforms must include:

* Content filtering to ensure discussions remain appropriate.
* Reporting systems to handle misuse or harassment.
* Moderation tools for administrators to maintain order and ensure a safe environment.

**Gap Identified in Current Research**

Most existing platforms prioritize either real-time interaction (e.g., video calls) or resource sharing (e.g., repositories) but fail to provide a holistic experience that combines communication, learning, and collaboration tools into one platform.

**Gaps in Current Platforms**

While many platforms for plant disease detection focus on specific plant categories, our project addresses this limitation by encompassing almost all plants. This comprehensive approach ensures that stakeholders from various agricultural regions have access to a unified tool. Existing solutions are often constrained to a smaller scope of crops or specific plants (e.g., tomatoes, rice, or wheat), leaving gaps in utility for farmers growing less common crops.

By bridging this gap, our solution provides a broader, all-encompassing platform that caters to diverse agricultural needs, ensuring maximum usability and inclusivity.

.

### Internal External Factors:

**Internal Factors**

Internal factors are elements within the project team or organization that influence the

success and functionality of the system. These factors are within the control of the team.

1. **Team Expertise and Skills**

* Availability of skilled developers, data scientists, and UI/UX designers to develop and deploy the system.
* Experience in Android app development, machine learning, and integration with APIs.

1. **Dataset Quality**

* The availability and quality of plant disease datasets, ensuring sufficient, well-labeled, and diverse images for training the model.
* Maintenance of a clean and organized dataset for optimal model performance.

1. **Technology Stack**

* Selection of the right tools, frameworks, and libraries (e.g., TensorFlow, Keras, MongoDB, Android Studio).
* Efficient use of resources such as a robust server backend and cloud storage.

1. **Budget and Resources**

* Adequate financial resources to acquire software licenses, cloud services, and hosting platforms.
* Access to computational resources like GPUs for training large-scale models.

1. **Project Management and Timeline**

* A well-defined project timeline and milestones.
* Clear communication within the team to avoid delays.

1. **System Scalability**

* Designing the application to handle increased users and requests in the future.
* Regular updates and feature enhancements based on user feedback.

1. **Data Security**

* Implementation of proper security measures for storing user and model data.
* Ensuring compliance with data protection laws and standards.

**External Factors**

External factors are elements outside the control of the project team but can significantly influence the project's success.

1. **User Demographics**

* The target audience, such as farmers, agricultural researchers, and plant disease experts, and their technological literacy.
* Accessibility features to accommodate users from rural areas with low-tech skills.

1. **Market Competition**

* Presence of similar plant disease detection platforms in the market and their functionalities.
* Differentiation of the app by offering support for a wider range of plants and diseases.

1. **Internet Connectivity**

* Dependence on internet availability in rural and remote areas for real-time data and feature updates.
* Offering offline functionality to overcome connectivity limitations.

1. **Regulatory and Legal Factors**

* Compliance with agricultural and data privacy laws.
* Obtaining necessary permissions for using plant-related datasets or offering suggestions.

1. **Environmental and Seasonal Factors**

* Changes in plant disease patterns due to environmental conditions.
* Seasonality affecting the type and frequency of diseases reported by users.

1. **Technological Advances**

* Integration of emerging technologies like AI, IoT sensors, and satellite imaging for enhanced features.
* Staying updated with advancements in machine learning algorithms and cloud-based solutions.

1. **User Feedback and Adoption Rate**

* Gaining insights from early adopters to refine features.
* Encouraging wide adoption by building trust through accurate disease detection and helpful suggestions.

1. **External Partnerships**

* Collaboration with agricultural organizations, research institutes, or government bodies.
* Building trust through endorsements from credible entities.

1. **Global and Local Issues**

* Addressing food security challenges and promoting sustainable farming practices.
* Catering to specific regional agricultural needs and addressing endemic plant diseases.

### Problem Flow

### Solution Space

The solution space outlines the possible approaches and tools to solve the identified problem in the project. This project focuses on providing a plant disease detection system that is accessible, reliable, and user-friendly for farmers, researchers, and agricultural experts.

**Key Aspects of the Solution Space:**

1. **Integration of Machine Learning (ML):**

* Use deep learning models (Mobile-Net) trained on diverse plant disease datasets to accurately detect plant diseases.
* Integrate the model for communication using Flask API.
* Real-time image analysis for quick results.

1. **User-Centric Mobile Application:**

* Android application with an intuitive UI/UX for easy interaction.
* Allow users to register and login for authentication purpose.
* Store the most recent history of the user so that user can easily access it.

1. **Treatment Suggestions:**

* Generate specific and actionable treatment recommendations based on the detected disease.

1. **Data Collection and Feedback:**

* Allow users to upload images for analysis, contributing to the dataset for future improvements.
* Provide feedback mechanisms for users to report inaccuracies.

1. **Scalable Backend Architecture:**

* Cloud-based storage for model deployment and image data.
* MongoDB for database management.

### Features

### Frontend Features (App):

### User Authentication (Login/Sign-up): Allows users to create an account and log in.

### Upload Image for Prediction: Users can upload images of plant leaves to get predictions about the disease.

### Disease Prediction and Suggestions: Shows predicted plant disease and suggestions for treatment.

### Profile Management: Displays the user's profile (e.g., name, email), but does not store prediction history.

### Backend Features:

### User Registration & Authentication: Handles user sign-up, login, and authentication.

### Disease Prediction API: Provides an API to receive image data, process it with the machine learning model, and return predictions.

### No Database for History: Does not store the history of predictions or suggestions, only current data.

### Machine Learning Features:

### Plant Disease Detection Model: Trained to detect diseases based on images and provide corresponding classification.

### Model Deployment: Deployed to the backend, allowing real-time predictions.

### Modules

### Frontend Modules (App):

### Authentication Module: Handles user login, sign-up, and user profile management.

### Prediction Module: Allows users to upload an image and receive a disease prediction along with suggestions.

### Recent History Saving Module: Displays Recent search of the user for easy access.

### Backend Modules (Node.js):

### User Authentication Module: Manages sign-up, login, and session handling.

### Disease Prediction API Module: Receives image data, calls the ML model for predictions, and sends the response to the frontend.

### Model Deployment Module: Hosts and manages the machine learning model for predictions.

### Suggestions Generation Module: Based on the prediction, this module provides treatment suggestions for the detected disease.

### Machine Learning Modules:

### Data Preprocessing Module: Handles image preprocessing before sending it to the model.

### Model Inference Module: Takes an image, passes it to the trained model, and returns the prediction.

### Relationship between modules

**Frontend <-> Backend:**

* **Authentication:** The Frontend communicates with the Backend to handle user login and sign-up.
* **Prediction Requests:** The Frontend sends the plant image to the Backend for prediction and receives the prediction and suggestion.

**Backend <-> Machine Learning:**

* **Prediction Request:** The Backend sends the plant image to the ML model for prediction.
* **Model Inference:** The ML model processes the image and returns the prediction and suggestion, which the Backend sends back to the Frontend.

**Frontend <-> Local Storage:**

* **User credentials storage:** Store the user detail along with password in form of hashed.

**User Uploads Image**: The user uploads a plant image to get a prediction.

**Backend Processes Image**: The app sends the image to the Backend for processing and prediction.

**Prediction Response**: The Backend calls the ML model, processes the image, and sends back the prediction and treatment suggestions.

**Displaying Prediction**: The Frontend (app) displays the prediction and suggestions to the user.

### Society

* **Agricultural Productivity Improvement**:

The app's plant disease detection system provides farmers with timely and accurate diagnoses of diseases, which can prevent crop loss and improve yields. This can enhance food security by enabling farmers to take preventive measures before diseases spread extensively.

* **Sustainability in Farming**:

By allowing farmers to detect diseases early, the app helps reduce the use of chemical pesticides and fertilizers, promoting more sustainable and eco-friendly farming practices.

* **Access to Knowledge**:

Farmers, especially in rural or remote areas, might have limited access to agricultural extension services. The app provides an easy-to-use platform for accessing critical agricultural knowledge, which can empower farmers with information to make better decisions.

* **Healthcare for Plants**:

Just like healthcare for humans, this app offers preventive care for plants. By diagnosing diseases early, it prevents plant health from deteriorating and ensures better management of agricultural resources.

* **Economic Growth for Farmers**:

Increased crop health and better disease management lead to improved agricultural production, boosting the income of farmers. The financial benefits can enhance the overall economy in regions reliant on agriculture.

### End Users

* **Farmers**:

The primary end users are farmers who grow crops like millets, fruits, and vegetables. They benefit from disease detection, prevention advice, and actionable suggestions to improve crop health and productivity.

Farmers with little technical knowledge will find the app useful as it offers a simple, user-friendly interface.

* **Agricultural Scientists and Researchers**:

The app can be used by researchers or agronomists who are working to study plant diseases or evaluate the effectiveness of different treatments. They can use the app's prediction and suggestion features for analysis and improvements in agricultural practices.

* **Agri-Tech Startups or NGOs**:

Organizations supporting rural or small-scale farmers may use the app to provide farmers with data-driven insights into plant diseases, crop management, and sustainable farming practices.

* **Consumers and Marts**:

Consumers interested in ensuring healthy plants and crops will benefit by receiving better produce. This can extend to marts or agricultural supply chain entities that are concerned with the quality and health of crops.

* **Extension Officers/Advisors**:

Agricultural extension workers can use the app to provide on-ground support to farmers. They can use the app’s suggestions for advice during farm visits.

### Privacy and Security:

* **Data Collection & Privacy:**

**Personal Information**: The app may collect basic personal data such as user credentials (username, email) for authentication purposes. It is important to ensure that this personal data is securely stored, and not shared with third parties without user consent.

**Sensitive Data**: The app collects images of plant leaves for disease prediction. These images do not contain personally identifiable information, but it is important to ensure that users provide informed consent for the use of their images.

* **Data Security:**

**Encryption**: All data transmitted between the mobile app, backend server, and the machine learning model should be encrypted using secure protocols like HTTPS to prevent data interception and tampering.

**Password Hashing with MD5**:

For storing user passwords securely, MD5 (Message Digest Algorithm 5) is used to hash passwords before storing them in the database. However, MD5 has known vulnerabilities, and it is not recommended for new applications. For enhanced security, consider using more secure hashing algorithms like SHA-256, bcrypt, or Argon2.

If MD5 is being used, ensure that **salting** (adding a random string to the password before hashing) is implemented. Salting helps prevent attacks such as rainbow table attacks, where precomputed hash values are used to reverse-engineer passwords.

**Example for MD5 with Salting:**

Before storing the password in the database, the password is concatenated with a random salt (a unique string generated for each user).

The salted password is then hashed using the MD5 algorithm, and the resulting hash value is stored in the database. This makes it harder for attackers to retrieve the original password.

**Authentication & Authorization**: Implement secure login mechanisms where user passwords are compared against their hashed versions stored in the database. Only hashed versions of passwords should be used, and passwords should never be stored or transmitted in plaintext.

## Chapter 4: Requirements Engineering

### Actors:

• User: Individuals who uploads plant leaf images for disease detection and re-

ceives results with treatment suggestions.

• System: A software application that Processes images using the AI model, iden-

tifies diseases, and provides predictions.

• Backend API: Manages image uploads, disease predictions, Retrieving data and

history storage.

### General Requirements:

### Infrastructure:

* A smartphone with the Android operating system and a stable internet connection

to support image uploads and API communication.

* Server infrastructure for hosting the backend, storing user data, and processing

disease predictions using TensorFlow.js.

### Competitor:

* Existing plant disease detection solutions often lack user-friendly mobile inter-

faces, real-time results, or comprehensive treatment suggestions.

* The proposed solution stands out by integrating a lightweight MobileNet model

for efficient real-time disease detection.

### Functional and Non-Functional Requirements:

### Functional Requirements:

**1. Disease Detection and Diagnosis :**

* The system must allow users to upload images of crops via the mobile app.
* The app must analyze the uploaded images using an ML model to identify plant diseases.
* The system must provide detailed diagnostic results, including the disease name.

**2. Treatment Recommendations:**

* The app must generate treatment suggestions based on the identified disease, including:
* Recommended pesticides and fertilizers.
* Organic or sustainable treatment methods.
* Preventive measures to avoid further infection.

**3. User Registration and Authentication**

* Users must be able to register and log in securely to access the app's features. The app must store user preferences and interaction history.

**4. Secure Storage**

* Store User details in the database
* Store the password as hash value using MD5(Hash Algorithm)

### 4.3.1 Non-Functional Requirements:

**1.Usability**

* The interface must be intuitive, requiring minimal technical knowledge for operation . The app must provide clear, context-sensitive help and guidance.

**2.Scalability**

* The system must support an increasing number of users, regions, crops, and diseases without a drop in performance.
* It should allow seamless integration of new features, datasets, and ML model updates.

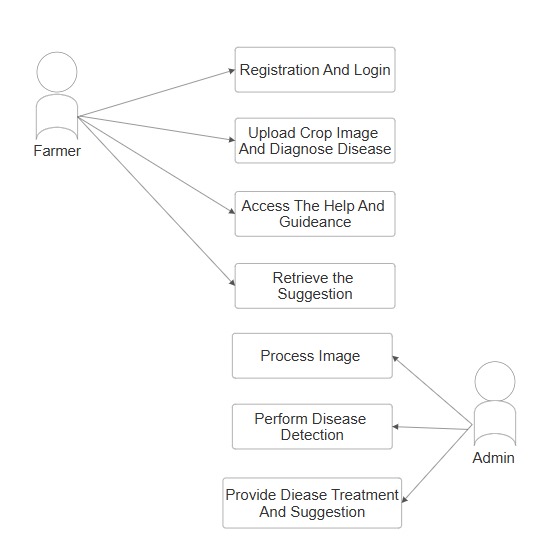
**3.Minimal Latency**

* The system must display prediction results with minimal delay after image upload.
* A lightweight AI model and optimized backend processes ensure faster predictions.

**4.Accuracy of Results**

* The system must accurately detect plant diseases from uploaded leaf images.
* A well-trained AI model on a diverse, labeled dataset ensures minimal false positives and negatives.

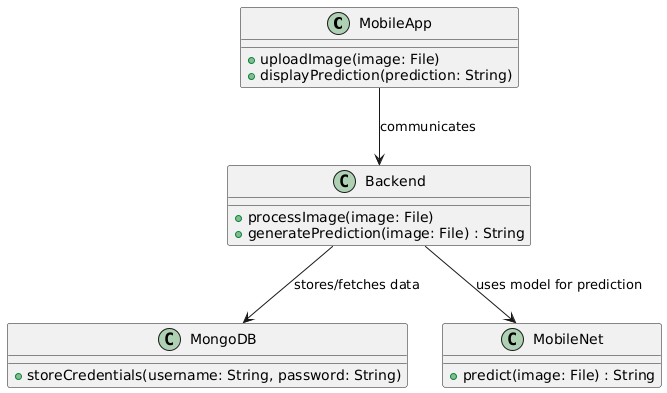
### 4.4: Use Case Diagram:

****

**Figure 1. Use Case Diagram**

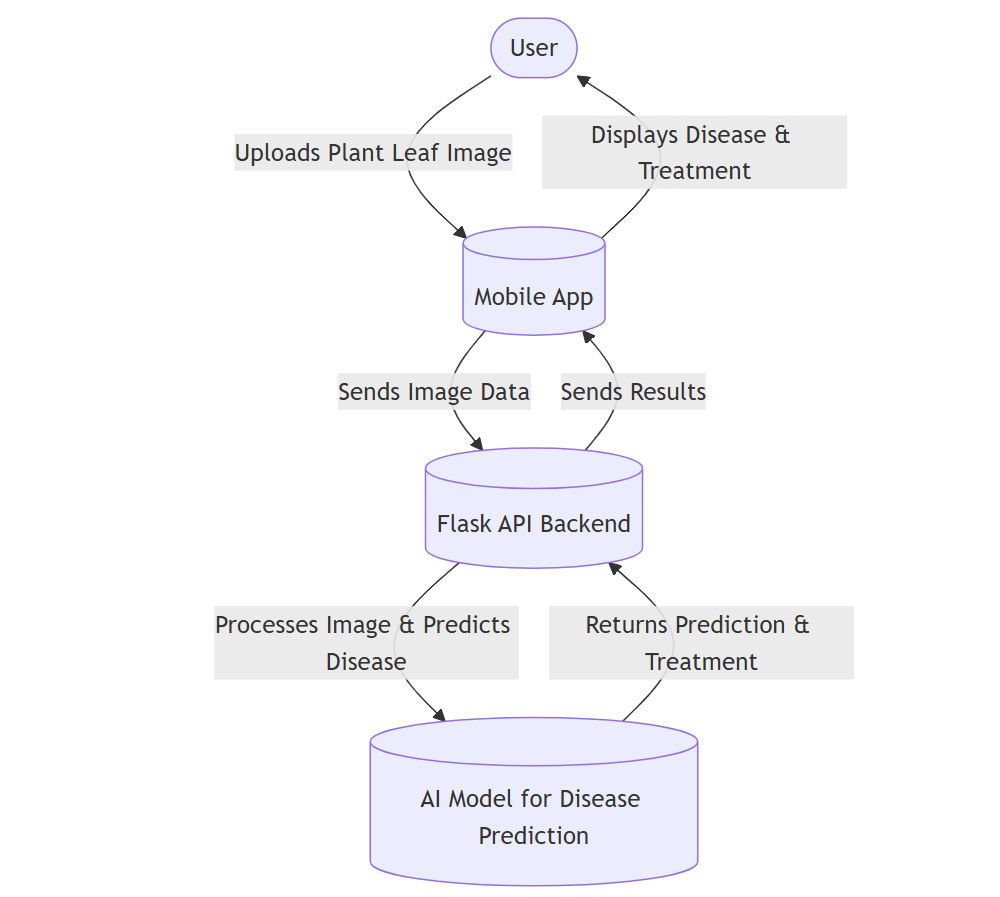
## Chapter 5: System Modeling

### UML Diagrams



**Figure 2.Uml Diagram**

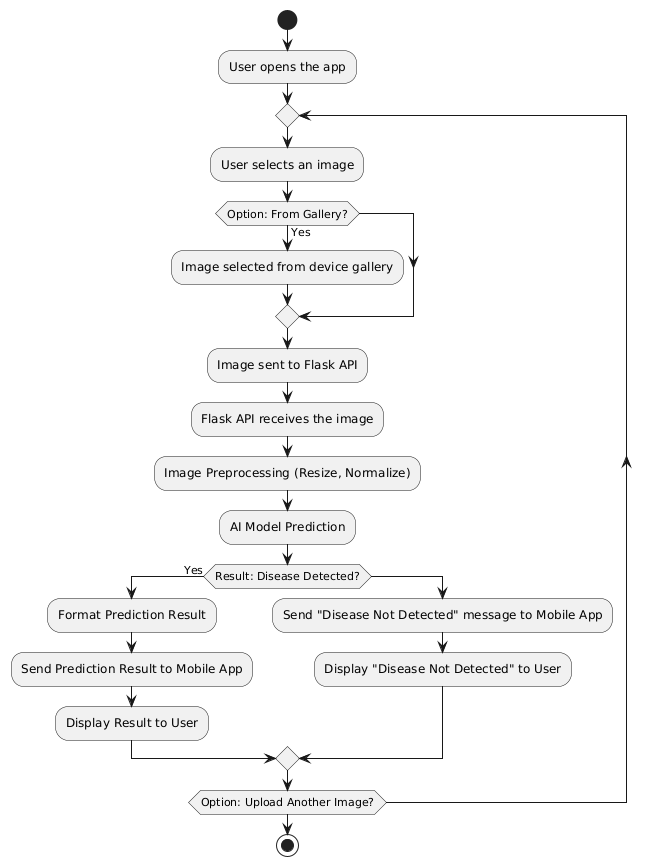
### Component Diagram

****

**Figure 3.Component Diagram**

A component diagram provides a high-level view of the system's architecture by modeling the system's components and their relationships. It illustrates how various software components interact, dependencies between them, and the overall structure of the system.

#### Activity Diagram



**Figure 4.Activity Diagram**

### Sequence Diagrams:

### Login:

### PlantUML diagram

**Figure 5. Sequence1**

**Explanation:**

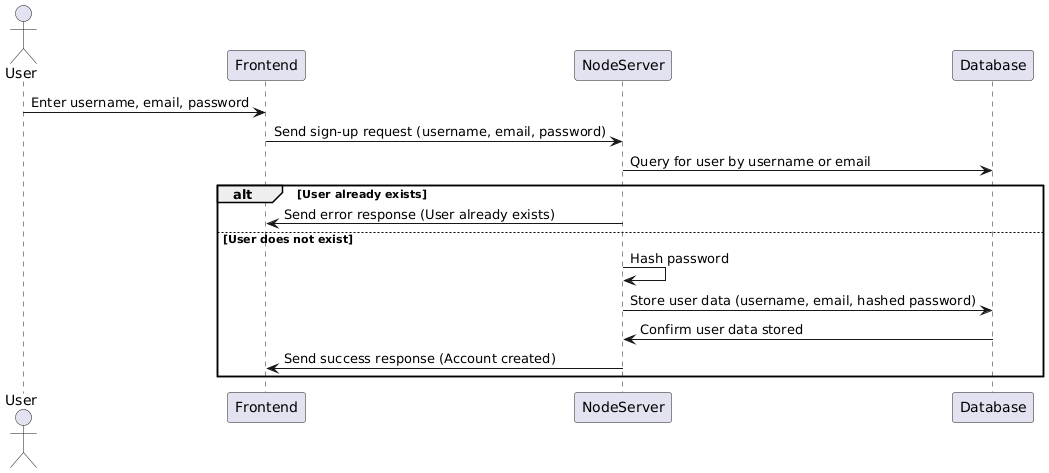
1. **User:** The user enters their credentials (username and password) in the frontend app.
2. **Frontend:** The frontend app sends a login request to the backend (Node.js server) with the username and password.
3. **NodeServer:** The Node.js server receives the request and queries the database to check if the user exists.
4. **Database:** The database returns the user’s data (usually the hashed password) to the Node.js server.
5. **Password Verification:**

* If the user is found, the backend compares the entered password with the hashed password stored in the database.
* If the passwords match, the backend sends a success response to the frontend (e.g., a JWT or session token for further authentication).
* If the password does not match, the backend sends an error response.

1. **User not found:** If the username doesn't exist, the backend returns an error response saying "User not found."

This diagram should help you visualize how the backend handles the login process in your Node.js app. Let me know if you need further assistance!

**Sign-Up:**



**Figure 6.Sequence2**

**Explanation:**

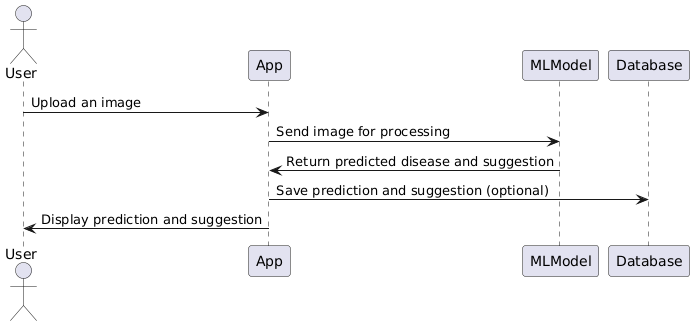
1. **User:** The user enters their credentials (username, email, password) in the frontend.
2. **Frontend:** The frontend sends a sign-up request with the entered username, email, and password to the backend (Node.js server).
3. **NodeServer:** The Node.js server receives the request and queries the database to check if the username or email already exists.
4. **Database:** The database checks whether the user with the same username or email exists.

* If the user already exists, the backend sends an error response back to the frontend (e.g., "User already exists").
* If the user does not exist, the backend proceeds to hash the password before storing it in the database.

1. **NodeServer:** The Node.js server hashes the password to enhance security.
2. **Database:** The server stores the user's data (username, email, and hashed password) in the database.
3. **NodeServer:** The backend confirms that the user data has been successfully stored.
4. **Frontend:** The frontend receives the success response, and the user is notified that their account has been created.

This sequence diagram provides a clear flow of how the sign-up process would work in your Node.js backend. Let me know if you'd like further details or modifications!

**Prediction:**



**Figure 7.Sequence3**

**Explanation:**

**User Uploads an Image:**

* The **User** initiates the process by uploading an image (likely of a plant) through the **App**. The image contains the plant that the user wants to analyze for disease.

**2. App Sends Image for Processing:**

* Once the user uploads the image, the **App** sends the image to the **MLModel** for disease detection. The **MLModel** is the core part of your system that processes the image and identifies any diseases or issues based on the image input.

**3. MLModel Returns Predicted Disease and Suggestion:**

* The **MLModel** processes the image and, based on its training, returns the prediction result. This includes:
  + The type of plant disease detected.
  + Suggestions or treatment recommendations for the detected disease.

**4. App Displays Prediction and Suggestion:**

* The **App** receives the prediction and suggestions from the **MLModel** and displays this information to the **User**. The user is shown the diagnosis (disease) and suggested actions, such as possible treatments or preventative measures.

**5. Optional: Save Prediction and Suggestion:**

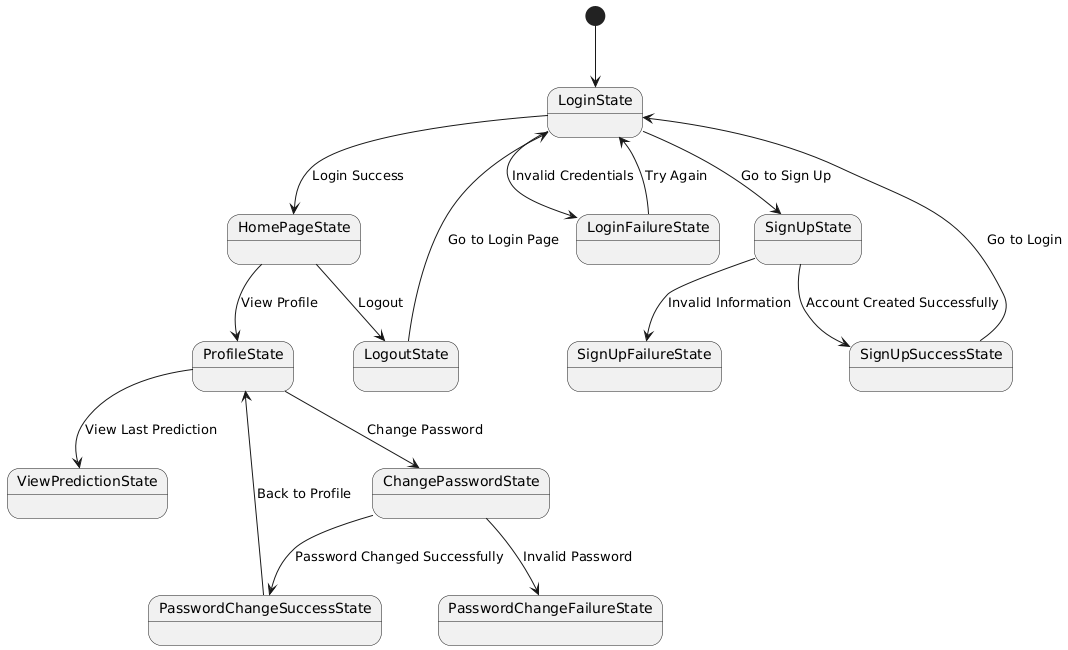
* This step is optional, but if implemented, the **App** can choose to save the prediction and suggestion in the **Database**. This allows the user to view their previous results at any time (e.g., through their profile or history page).

**6. Database:**

* The **Database** stores information such as user details, uploaded images, predictions, suggestions, and any other relevant data. If the system is designed to save the predictions and suggestions, the **App** will interact with the **Database** to store these details for future use.

This **Sequence Diagram** illustrates the flow of interactions between the **User**, **App**, **MLModel**, and **Database** in your system.

#### State Diagram



**Figure 8.State Diagram**

**Explanation of States:**

1. **LoginState**:

* This is the initial state when the user tries to log in with their credentials.
* If credentials are valid, the system transitions to the HomePageState.
* If the credentials are invalid, it moves to the LoginFailureState.

1. **SignUpState**:

* When the user selects "Sign Up," they are directed to this state.
* If the user provides invalid information during sign-up, it transitions to SignUpFailureState.
* If the sign-up is successful, it transitions to SignUpSuccessState.

1. **HomePageState**:

* This is the main state after successful login. The user can either view their profile or log out.

1. **ProfileState**:

* This state is for managing the user profile. From here, the user can either view their last prediction or change their password.

1. **ChangePasswordState**:

* If the user opts to change their password, this state handles that process.
* If the password is successfully changed, the state transitions to PasswordChangeSuccessState.
* If the password change fails (e.g., invalid current password), it transitions to PasswordChangeFailureState.

1. **LogoutState**:

* This state represents logging the user out and directing them back to the login page.

1. **Password Hashing & Security Considerations:**

* While this state diagram represents the flow of actions, password handling and hashing (MD5, bcrypt, etc.) would occur during transitions from the login or sign-up states, specifically when user credentials are submitted.
* MD5 hashing (or ideally more secure algorithms) is done before storing or validating passwords.

#### Components and Communication:

#### Frontend (Android App):

#### User Interface:

#### This represents the user-facing interface of the app where users interact, such as uploading images for prediction and viewing results.

#### Authentication & Profile:

#### Handles the login, signup, and profile management features.

#### Sends authentication requests to the Backend and stores profile data locally if needed.

#### Prediction & Suggestions:

#### Takes image input from the user, sends it to the Backend for processing, and displays the prediction and suggestion to the user.

#### Local Storage (SharedPreferences):

#### Temporarily stores the most recent prediction and suggestion for easy access from the profile.

#### Backend (Node.js API):

#### Authentication Service:

#### Validates user credentials, either for login or signup, and returns appropriate responses (such as authentication tokens).

#### Prediction API:

#### This API handles the communication between the app and the ML model.

#### It processes the image sent by the app, calls the prediction model, and returns the disease prediction and suggestions.

#### User Management:

#### Manages user profile information, including saving and updating the profile.

#### Machine Learning (ML Model):

#### Plant Disease Detection Model:

#### This is the core model that handles image analysis and predicts the disease along with suggestions. The model is called by the Backend's Prediction API.

#### User Authentication:

#### The User Interface communicates with Authentication & Profile when the user logs in or signs up.

#### The Authentication & Profile component sends the credentials to the Backend (Node.js API), which calls the Authentication Service to validate the user and return a response (success or failure).

#### Prediction Request:

#### When a user uploads a plant image, the User Interface sends the image to the Prediction & Suggestions component.

#### The Prediction & Suggestions component sends the image to the Backend (Node.js API), which in turn sends the image to the Prediction API.

#### The Prediction API interacts with the Plant Disease Detection Model, which processes the image and returns a prediction (disease type and suggestions).

#### Storing the Prediction:

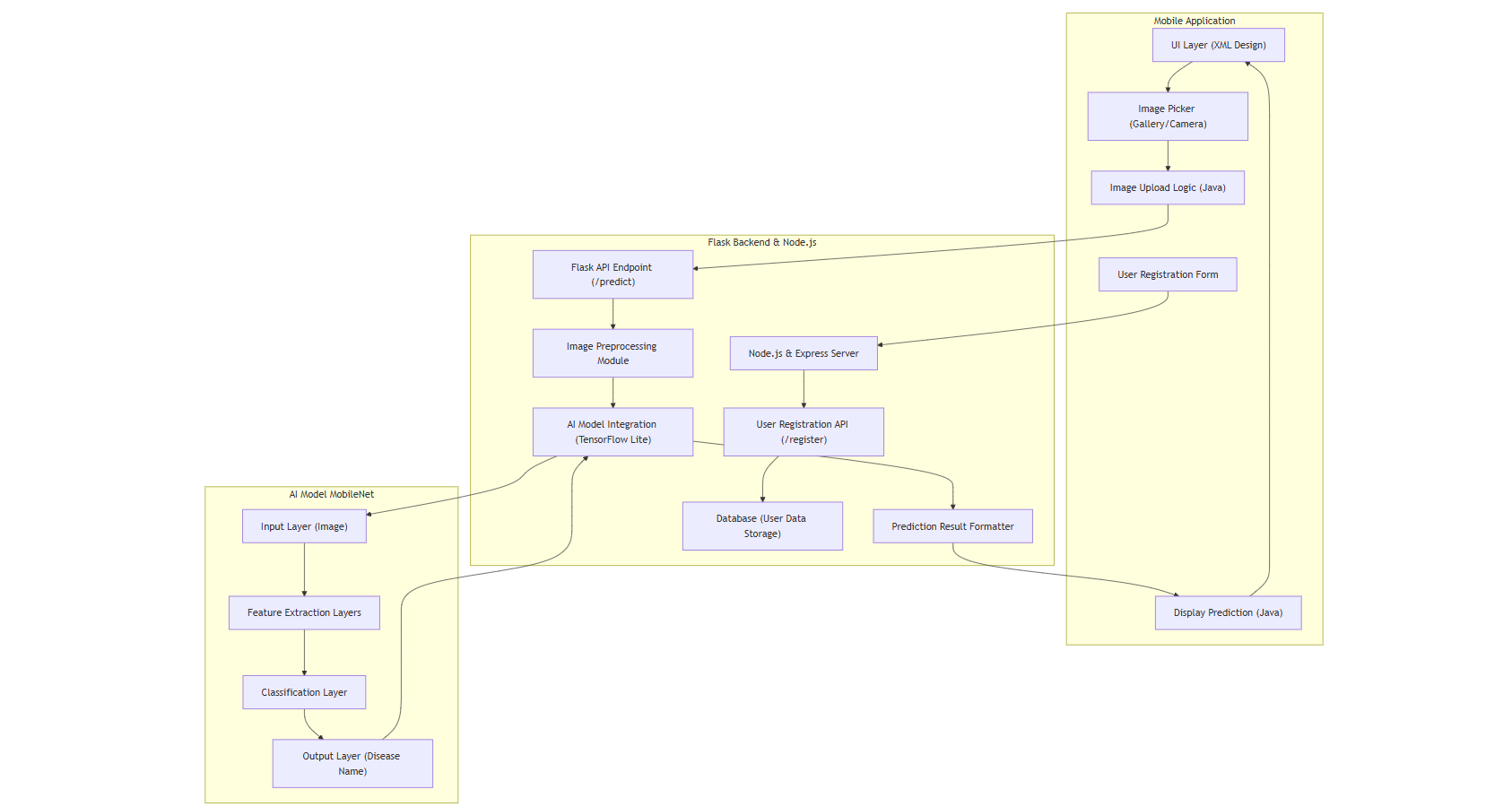
#### Once the prediction is returned, the Prediction & Suggestions component stores the result in Local Storage (SharedPreferences) for future access (like when the user visits their profile).

#### User Profile Management:

#### The Authentication & Profile component handles updating and managing user profile information. Any changes in the profile are sent to the User Management component in the Backend.

#### If there are changes related to profile data, such as updating profile details, the Backend stores or updates that information.

**5.2 Model Architecture :**



**Figure 9.Model Architecture**

## Chapter 6: Implementation

### Algorithm Development

**1.Mobile App Development**

**Algorithm for Image Upload and Prediction Display:**

1. **Start**:

* User opens the app and sees the home screen.

1. **Image Upload**:

* User clicks the "Upload" button.
* Show options to pick an image from the gallery or use the camera.

1. **Image Selection**:

* Once the user selects or captures an image, the app processes the image.
* Send the image to the backend server for disease prediction.

1. **Send Image to Server**:

* The app sends the selected image to the Flask backend using an HTTP POST request.
* The image is encoded (e.g., base64) before transmission.

1. **Display Prediction**:

* After receiving the prediction result from the backend, the app parses the response.
* Display the disease name and treatment suggestions on the Prediction Results Page.

1. **End**.

**2. Backend (Server)**

**Algorithm for Image Processing and Prediction:**

1. **Start**:

* The server is running with Flask and is waiting for incoming image requests.

1. **Receive Image**:

* The Flask backend listens for POST requests with an image in the request body.
* The image is temporarily saved on the server for processing.

1. **Preprocess Image**:

* Convert the image to the format required by the AI model (resize, normalization, etc.).
* Ensure compatibility with the trained model's input dimensions.

1. **Predict Disease**:

* The preprocessed image is passed to the AI model (MobileNet).
* The model predicts the disease by analyzing the image.

1. **Generate Treatment Suggestions**:

* Based on the model’s output, generate suggestions or recommendations for treatment.
* These suggestions can be mapped to the predicted disease category.

1. **Send Prediction and Suggestions Back**:

* Package the predicted disease name and treatment suggestions into a JSON response.
* Send the response back to the mobile app.

1. **Save User Data** (Optional):

* If the user opts to store the result, the server stores the user's profile and prediction in the MongoDB database.

1. **End**.

**3. Model Implementation**

**Our Dataset consists of 90000+ images with 73 classes. We have Created a dataset by integrating different datasets on plant diseases.**

**Algorithm for Training and Deployment:**

1. **Start**:

* Gather the dataset of plant leaf images categorized into different disease classes.
* Prepare the dataset by integrating images from multiple sources.

1. **Data Preprocessing**:

* Resize images to the model-compatible resolution (e.g., 224x224).
* Normalize pixel values (e.g., scale from 0 to 1).
* Apply data augmentation (rotation, flipping, etc.) to improve model generalization.

1. **Model Architecture**:

* Use MobileNet as the base model for plant disease classification.
* Add custom layers on top (e.g., dense layers) to adjust the model for the specific task.

1. **Model Training**:

* Train the model using the prepared dataset.
* Use Adam and Categorical cross entropy for model training with the appropriate optimizer and loss function.
* Monitor training using metrics like accuracy, precision, and recall.

1. **Model Evaluation**:

* After training, evaluate the model on the validation dataset.
* Check if the accuracy meets the desired threshold.

1. **Model Export**:

* Once satisfied with the accuracy, export the trained model in the .keras format.

1. **Deploy Model**:

* Convert the trained model to be compatible with Flask (e.g., use TensorFlow or Keras for serving).
* Deploy the model on the backend server to handle real-time predictions.

1. **End**.

**4. Integration Between Components**

**Algorithm for Communication Between Mobile App, Backend, and Model:**

1. **User Uploads Image**:

* Mobile app sends an image to the backend server using an HTTP POST request.

1. **Backend Receives Image**:

* The Flask backend receives the image.
* The server processes the image and forwards it to the trained MobileNet model.

1. **Prediction by Model**:

* The model processes the image and predicts the disease and treatment suggestions.
* The mobile app parses the response and displays the prediction and suggestions to the user.

1. **End**.

### User interface design:

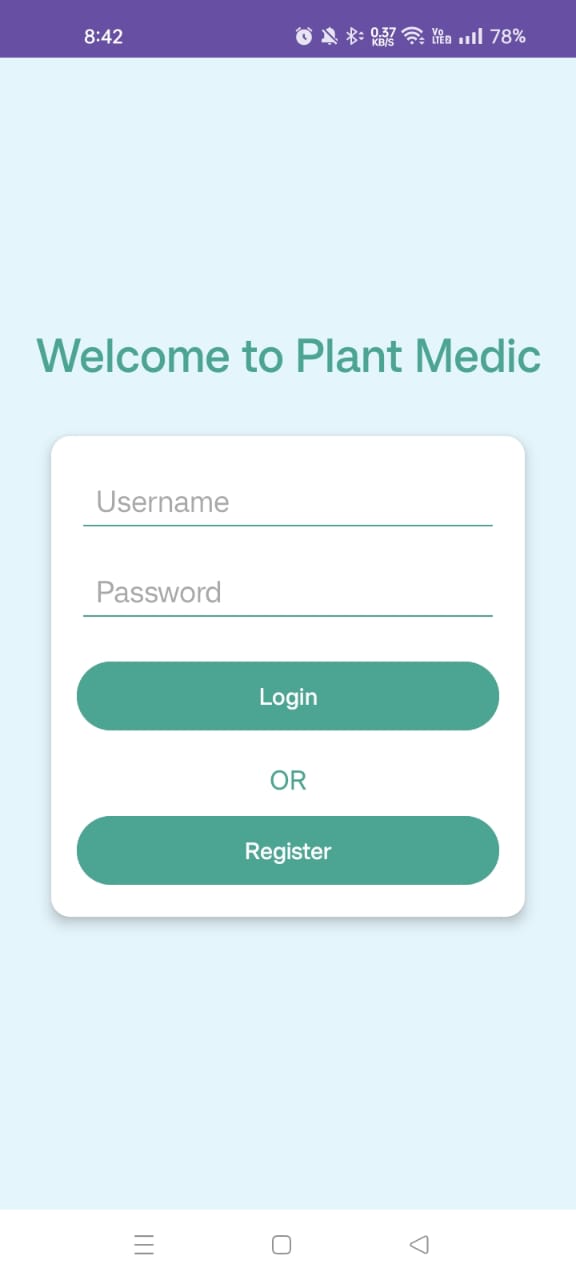
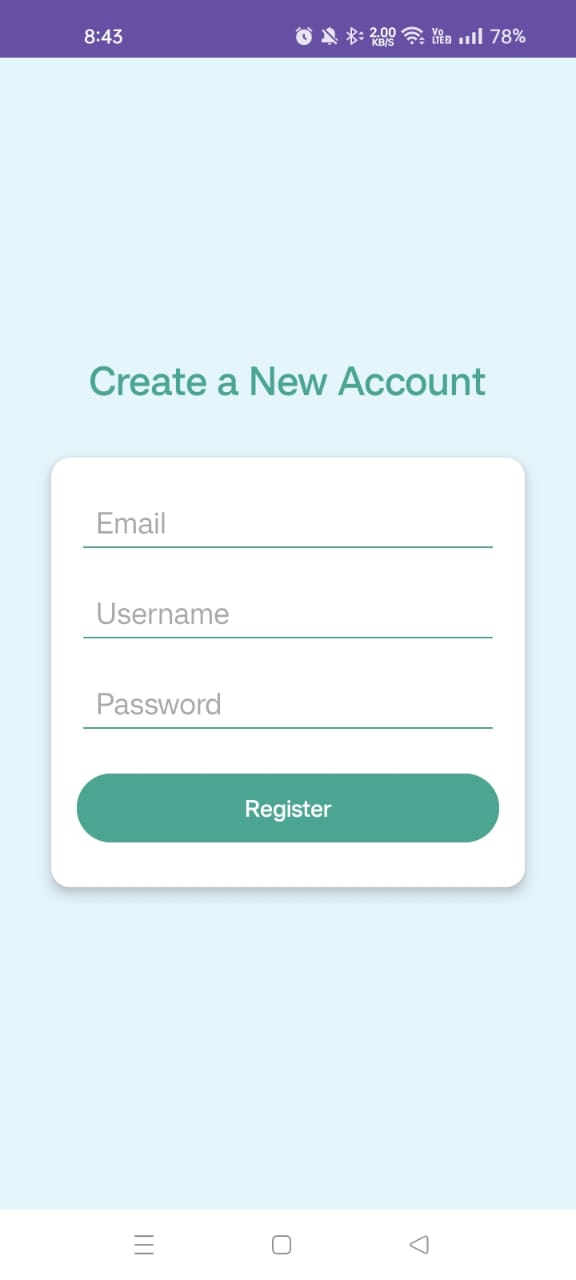
** **

Fig. 10. Login Page Fig. 11. Signup Page

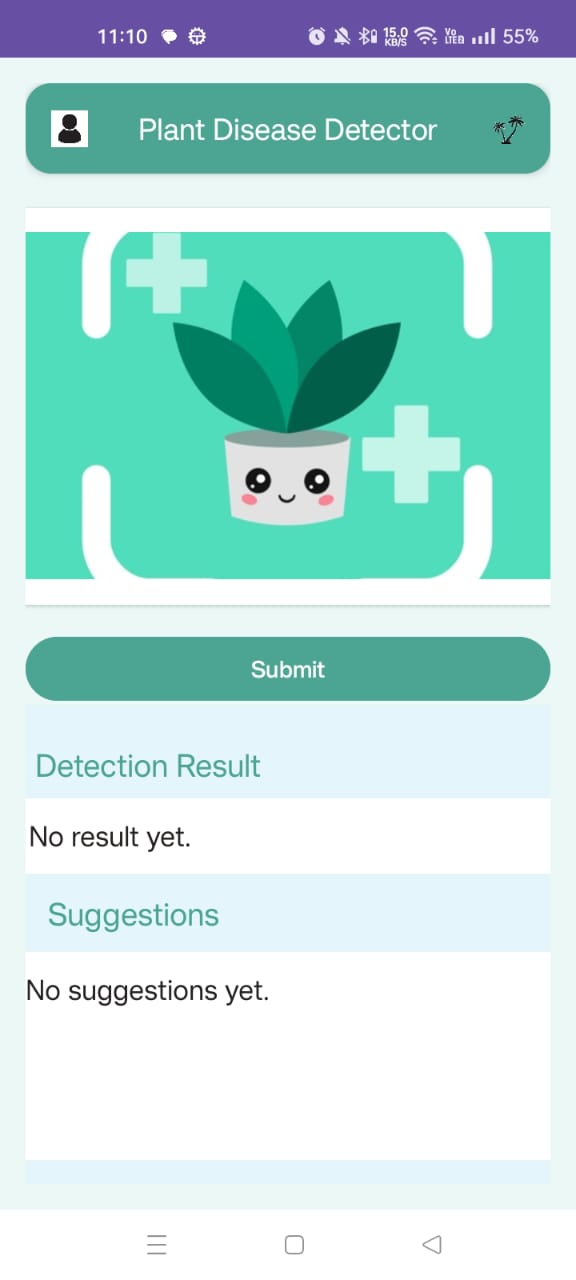
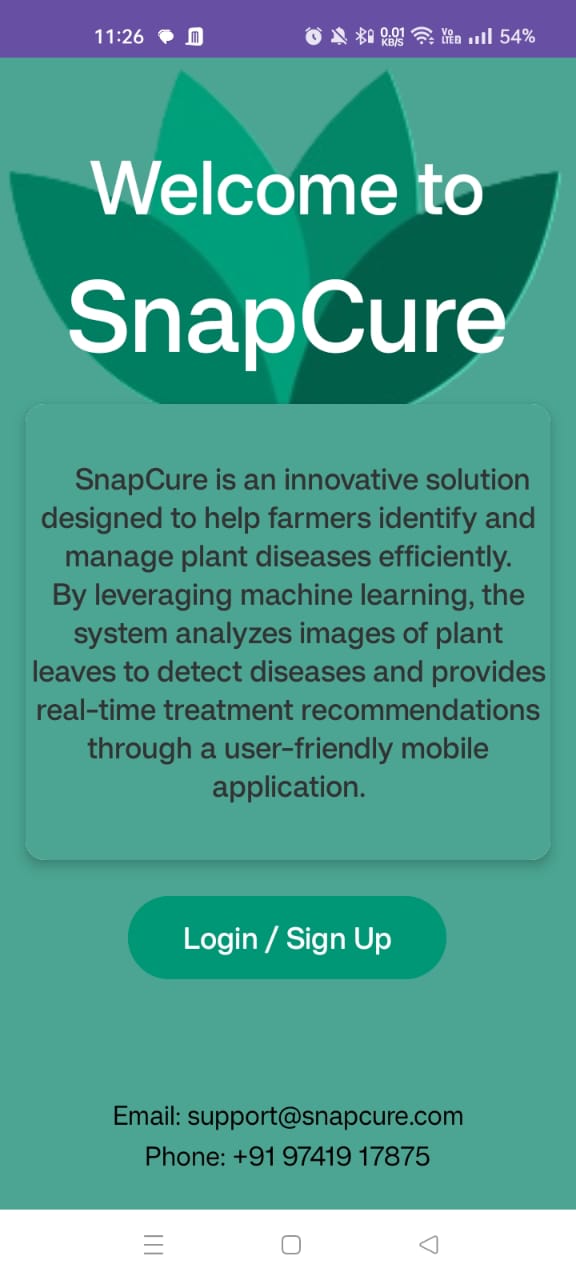
**** 

Fig 12. Main Page Fig 13. Home Page

## Chapter 7: Testing

### Testing Report:

**Initial Testing in Mobile App (Frontend)**

The testing begins with the mobile app, focusing on its functionalities and ensuring seamless user experience.

* **Tested Features**:
  + **Image Upload Handling**: Verified the image picker functionality for selecting images from the gallery or capturing through the camera.
  + **Navigation**: Ensured smooth navigation between screens (Home Screen, Upload Section, and Prediction Results Page) using intents.
  + **Prediction Display**: Checked that the detected disease name and treatment suggestions are correctly displayed on the results page.

**2. Backend Testing (Server)**

The server-side logic was tested to ensure accurate predictions and smooth communication between the frontend and backend.

* **Tested Features**:
  + **Image Upload Handling**: Confirmed that images are successfully received and stored temporarily on the Flask server.
  + **AI Model Integration**: Verified that the AI model processes the uploaded image and returns accurate predictions.
  + **API Response**: Ensured that the Flask API returns the predicted disease name and treatment suggestions in a proper JSON format.

**3. Database Testing**

The database was tested for proper storage and retrieval of user credentials, prediction history, and related data.

* **Tested Features**:
  + User data and predictions were successfully stored in MongoDB.
  + Retrieval of stored data was verified for display on the app's user interface.

**4. Model Testing**

The MobileNet model used for disease detection was rigorously tested to ensure accuracy and performance.

* **Tested Features**:
  + **Data Preprocessing**: Verified resizing and augmentation techniques to ensure compatibility with the model.
  + **Prediction Accuracy**: Tested the model on a validation dataset and achieved satisfactory accuracy metrics.
  + **Deployment Compatibility**: Ensured the trained model was properly converted to .keras format and integrated with Flask for server-side predictions.

**5. System-Wide Testing**

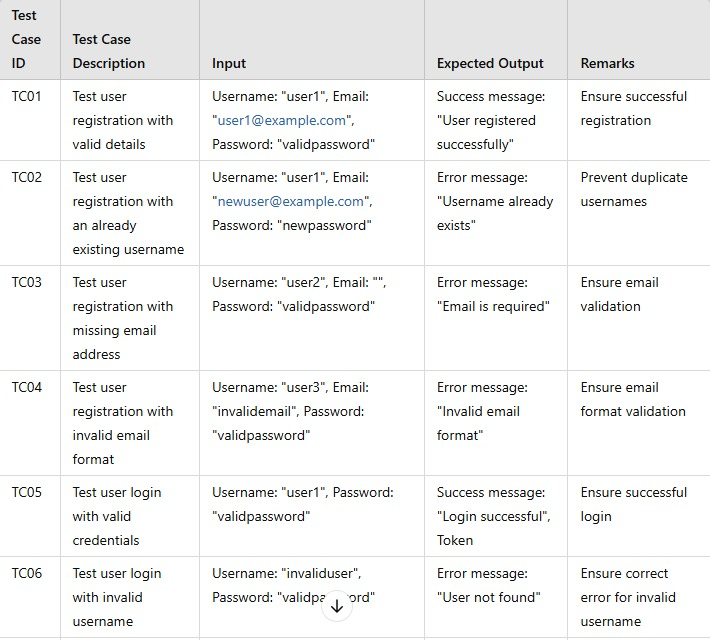
The entire system, including the app, backend server, and database, was tested as a unified system to ensure end-to-end functionality.

* **Tested Features**:
  + Image uploads from the app were successfully processed by the backend and returned accurate predictions.
  + The app correctly displayed the results and saved user data for future reference.

**6. Observations**

* The **image upload feature** performed well under different scenarios (camera and gallery inputs).
* Predictions were accurate for most disease classes, with an overall **model accuracy above 90%** during testing.
* The **navigation and UI design** were user-friendly, ensuring smooth app usage.
* Backend API responses were consistent, with no significant delays in processing.

### Test Cases:





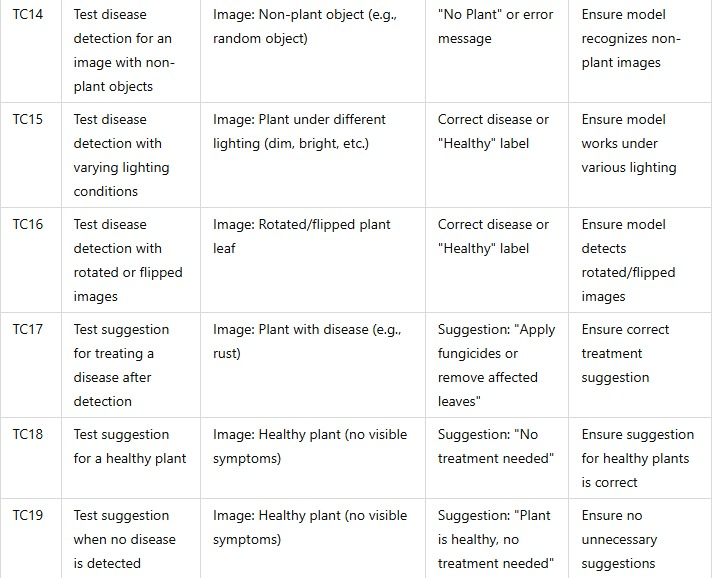


Table 1: Test cases

#### Test Methodologies:

* + - Unit Testing: Focused on testing individual using mock data.
    - Integration Testing: Checked how well different modules interacted with each other.
    - System Testing: Verified that all modules worked as expected when integrated into the full system. Ensured that user registration, login, disease detection, suggestion worked together seamlessly

### Testing Tools:

The following tools were used to facilitate the testing process:

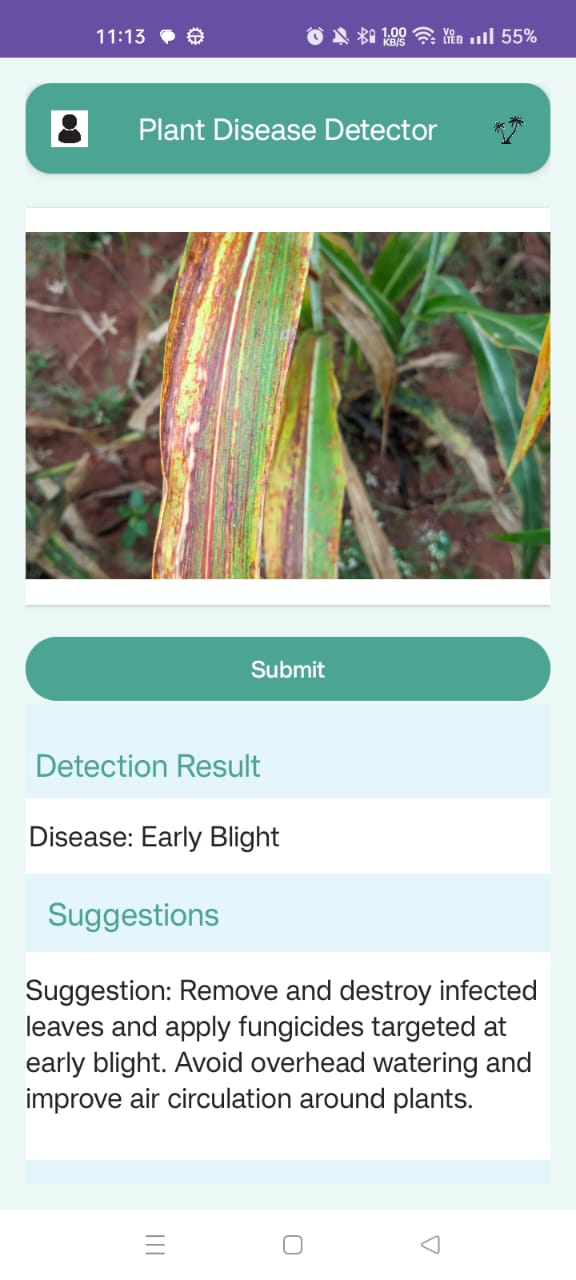
**Testing Tools Used**

The same tools used during development were utilized for testing the project:

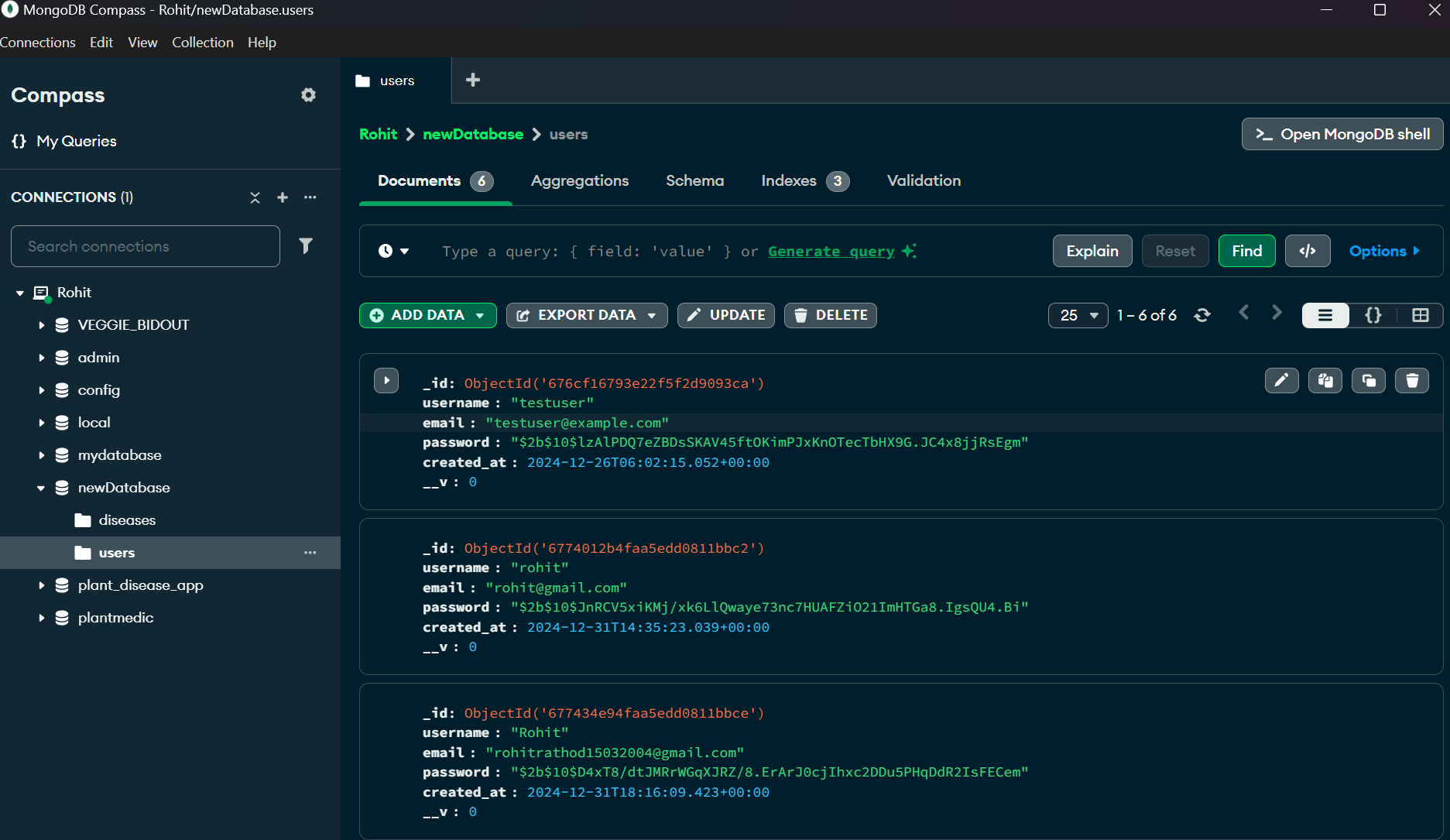
* **Frontend Tools**:
  + **Android Studio**: Debugged the app for smooth UI and functional logic.
  + **Java**: Verified the image picker, intents, and other logic.
* **Backend Tools**:
  + **Flask**: Tested API endpoints and server-side processing.
  + **Postman**: Verified API responses for accuracy and format.
* **Database Tools**:
  + **MongoDB Compass**: Validated database entries and queries.
* **Model Development Tools**:
  + **TensorFlow/Keras**: Ensured the model's accuracy during training and testing phases.

### Result:

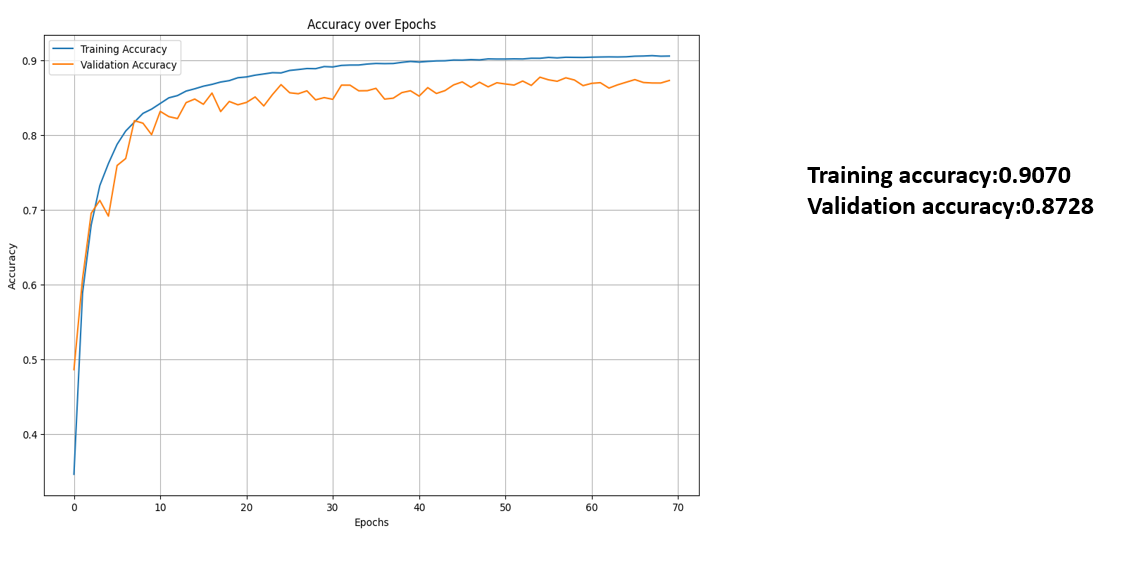
### User Main page: Here user uploads the pictures and retrieve the disease and the suggestion.



**Backend/Database Results:** User details are stored along with the password in hash.



**Model result:** The picture displays the Accuracy vs Epochs along with the accuracy results.



### Conclusion:

The Plant Disease Detection System effectively combines a user-friendly mobile app, a secure Flask backend, and a MobileNet-based machine learning model to provide accurate disease predictions and treatment suggestions. With a focus on user privacy, data security, and scalability, the system offers farmers a reliable tool for improving crop health and productivity. This project highlights the potential of AI in addressing agricultural challenges and promoting sustainable farming practices.

