A Midterm Progress Report

on

KISAN MITRA: SMART FARMING ASSISSTANT

Submitted in partial fulfillment of the requirements for the award of

BACHELOR OF TECHNOLOGY

Computer Science and Engineering

Batch 2022-2026

SUBMITTED BY

DIYA BAWEJA (2203425, 2215036)

GURJOT KAUR (2203433, 2215044)

RAHUL SACHDEVA (2203536, 2215145)

UNDER THE GUIDANCE OF

PROF. HARDEEP SINGH KANG

Department of Computer Science and Engineering

GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA

Contents

1 Introduction

1.1 Overview

Kisan Mitra is an innovative solution aimed at transforming traditional farming by leveraging advanced machine learning (ML) techniques for plant disease detection. The project integrates an intelligent ML model with a recommendations database and weather-based alert systems, providing farmers with actionable insights to improve crop health and yield. The system is designed for mobile deployment, ensuring that farmers can receive real-time assistance in the field.

1.2 Motivation

Timely and accurate detection of plant diseases is critical for reducing crop losses and optimizing treatment strategies. Manual inspections are often error-prone and inefficient, leading to delayed responses. Kisan Mitra addresses these challenges by:

- Automating disease identification through state-of-the-art image analysis.
- Providing tailored treatment recommendations using a comprehensive database.
- Issuing proactive alerts by incorporating weather and location data.

1.3 Objectives

1. To develop an application system supporting intelligent plant disease identification by analysing plant leaf images using machine learning techniques.
2. To provide a machine learning based recommendation system that suggests targeted remedies and preventive measures based on classified disease.
3. To provide alerts to farmers in textual and graphical form using weather and location monitoring about potential disease outbreaks or climate problems.

1.4 Project Scope

The project primarily targets common crops (e.g., Apple, Tomato, Potato) and their associated diseases. While the current development focuses on the core ML model, Recommendation Database and a basic UI design, future work will expand functionality to include full user interaction, backend integrations, and comprehensive analytics.

2 System Requirements

2.1 Hardware Requirements

- **Mobile Device:** ARM-based smartphone or tablet with a minimum of 2 GB RAM.
- **Server/Cloud:** A cloud-based server or on-premises machine capable of hosting the Node.js backend and TensorFlow ML inference engine.
- **Sensors:**
  - **Camera:** For capturing plant leaf images.
  - **GPS:** For location-based alert generation.

2.2 Software Requirements

Operating Systems:

- **Mobile:** Android/iOS for the Flutter application.
- **Server:** Linux/Windows for Node.js backend.

Programming Languages and Frameworks:

- **Flutter (Dart):** For the front-end of the mobile application.
- **Node.js:** For backend services (REST APIs, data management).
- **TensorFlow/TensorFlow Lite:** For training and deploying the ML model.

Database:

- SQL/NoSQL database (e.g., MySQL, MongoDB) for storing user data, disease information, treatment recommendations, and weather logs.

APIs:

- **Weather API:** (e.g., OpenWeatherMap) for retrieving real-time weather data.
- **Geolocation Services:** For mapping user locations with disease-prone areas.

3 Software Requirement Analysis

3.1 Problem Definition

Farmers frequently struggle with accurately diagnosing plant diseases in a timely manner, resulting in inefficient use of resources and potential crop loss. Kisan Mitra addresses this issue by:

- Automating disease diagnosis through ML-driven image analysis.
- Delivering precise treatment recommendations based on a dedicated recommendations database.
- Providing early warnings through integration of weather data and geolocation information.

3.2 Modules and Functionalities

3.2.1 ML Model Module

- **Purpose:** Analyze plant leaf images and classify them into disease categories using an innovative ensemble teacher-student model.
- **Key Feature:** Employs multi-level distillation to capture both high-level predictions and detailed feature representations.

3.2.2 Recommendation Module

- **Purpose:** Query a structured recommendations database to provide targeted remedies and preventive measures based on the identified disease.
- **Data Source:** Curated treatment protocols, pesticide guidelines, and preventive strategies linked to specific diseases.

3.2.3 App Module (Flutter + Node.js)

- **Purpose:** Provide a user interface for the system.

- **Current Status:** The mobile UI is in a preliminary design stage, developed with Flutter. It currently offers a polished frontend design without integrated essential functionalities such as image capture or classification display.
- **Future Enhancements:** Advanced features like user authentication, push notifications, and information guide dashboards will be incorporated in subsequent iterations.

3.2.4 Alert Module

- **Purpose:** Monitor weather and location data to generate proactive alerts on potential disease outbreaks or adverse climate conditions.
- **Integration:** Connects with a weather API to fetch real-time data and correlate it with disease occurrence patterns.

4 Software Design

4.1 Data Flow and Model Training

Below is a high-level workflow diagram illustrating the model training process:

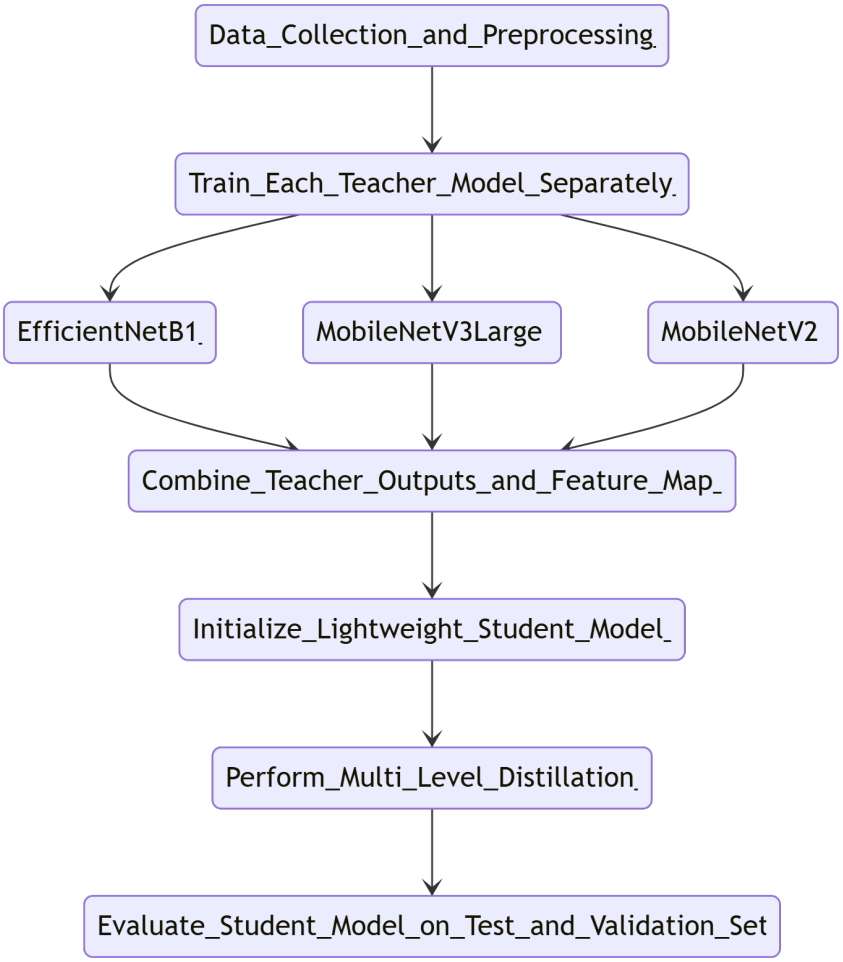


Figure 1: Model Training Workflow

4.1.1 Data Collection & Preprocessing

- **Dataset Integration:**
  - Merging images from multiple sources (e.g., PlantVillage) into a unified directory.
  - Creating combined class names to avoid conflicts and performing oversampling to balance classes.
- **Image Augmentation:**

  - Techniques such as rotation, flipping, and color adjustments improve model generalization.

4.1.2 Teacher Models

- **Architectures Used:** EfficientNetB1, MobileNetV3Large, MobileNetV2.
- **Feature Extraction:** Each teacher model is trained independently to extract complementary features from the dataset.
- **Ensemble Fusion:** The outputs and intermediate feature maps of the teacher models are dynamically weighted (learnable weights) and fused to guide the student model.

4.1.3 Student Model

- **Architecture:** A custom, lightweight Convolutional Neural Network (CNN) utilizing depthwise separable convolutions and squeeze-and-excitation blocks.
- **Distillation Strategy:**
  - **Output-Level Distillation:** Aligns the student's softened predictions (using a high temperature) with the teacher ensemble's output.
  - **Feature-Level Distillation:** Matches intermediate feature maps between the student and the fused teacher features.

Benefits: This multi-level approach ensures that the student model retains the rich semantic information learned by the teacher ensemble while maintaining a compact form factor for mobile deployment.

4.1.4 Evaluation & Inference

- **Metrics:** Accuracy, precision, recall, F1 score, inference latency, and model size.
- **Deployment Considerations:** The student model is optimized for low latency and minimal resource consumption, making it ideal for mobile deployment.

4.2 System Deployment Architecture

Below is the high-level architecture diagram for system deployment:

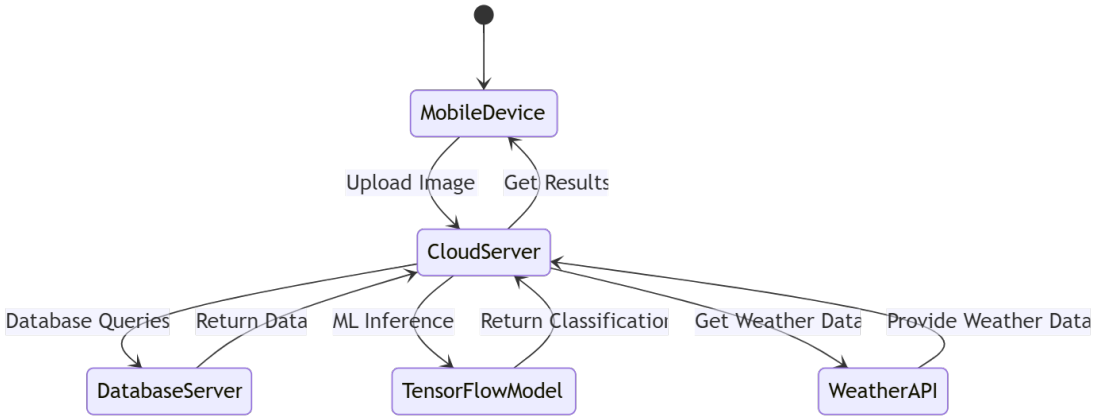


Figure 2: System Deployment Architecture

- **Mobile Device (Flutter):** Provides a frontend interface for user interactions. Currently, the design is in place without active functionalities.
- **Cloud Server (Node.js):** Manages API requests and orchestrates interactions between the mobile app, ML model, and weather service.
- **Database Server:** Stores user profiles, logs, and the recommendations database.
- **Weather API:** Supplies real-time climate data for alert generation.

4.3 Database Design (E-R Diagram)

The following Entity-Relationship (ER) diagram represents the database schema:

Users Table

- **id (integer, primary key)**: Unique identifier for each user.
- **name (varchar)**: Stores the name of the user.
- **phone (varchar)**: Contact number of the user.
- **profile_image (varchar)**: URL or path to the user's profile picture.

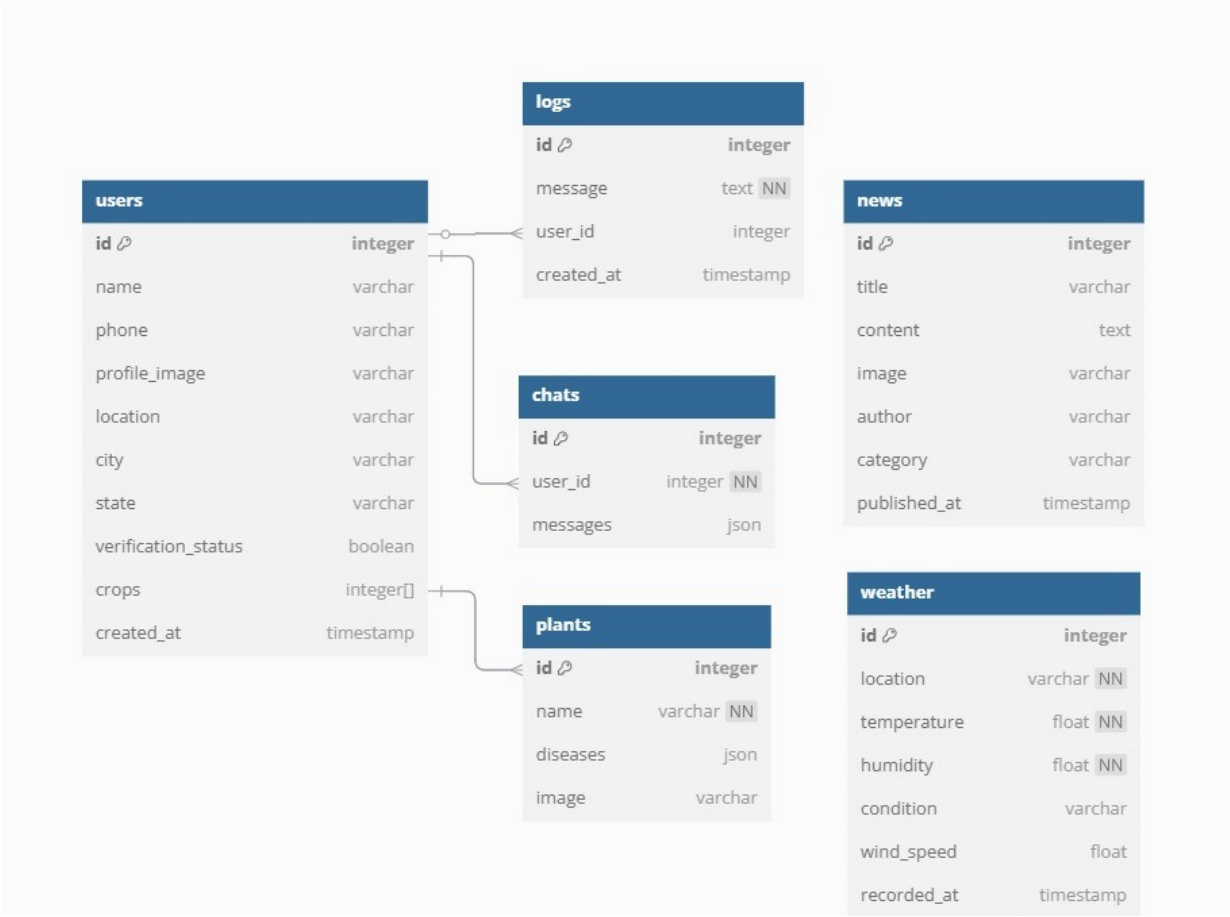


Figure 3: E-R Diagram for Database

- **location (varchar)**: User's general location.
- **city (varchar)**: User's city.
- **state (varchar)**: User's state.
- **verification_status (boolean)**: Indicates whether the user is verified.
- **crops (integer[])**: Array of crops associated with the user.
- **created_at (timestamp)**: Timestamp of user account creation.

Logs Table

- **id (integer, primary key)**: Unique identifier for each log entry.
- **message (text, NOT NULL)**: Log message details.
- **user_id (integer, foreign key)**: References the user who created the log.

- **created_at (timestamp)**: Timestamp of log creation.

Chats Table

- **id (integer, primary key)**: Unique identifier for each chat entry.
- **user_id (integer, foreign key, NOT NULL)**: References the user associated with the chat.
- **messages (json)**: Stores chat messages in JSON format.

Plants Table

- **id (integer, primary key)**: Unique identifier for each plant.
- **name (varchar, NOT NULL)**: Name of the plant.
- **diseases (json)**: List of diseases affecting the plant.
- **image (varchar)**: URL or path to an image of the plant.

News Table

- **id (integer, primary key)**: Unique identifier for each news article.
- **title (varchar)**: Title of the news.
- **content (text)**: Main content of the news.
- **image (varchar)**: URL or path to an image related to the news.
- **author (varchar)**: Name of the news author.
- **category (varchar)**: Category of the news (e.g., agriculture, weather).
- **published_at (timestamp)**: Timestamp of when the news was published.

Weather Table

- **id (integer, primary key)**: Unique identifier for each weather record.
- **location (varchar, NOT NULL)**: Location where the weather data was recorded.
- **temperature (float, NOT NULL)**: Temperature at the recorded location.
- **humidity (float, NOT NULL)**: Humidity level at the location.
- **condition (varchar)**: Weather condition (e.g., sunny, rainy).
- **wind_speed (float)**: Wind speed at the location.
- **recorded_at (timestamp)**: Timestamp of weather data recording.

4.4 Notification Feature

To provide alerts to farmers in textual and graphical form using weather and location monitoring about potential disease outbreaks or climate problems.

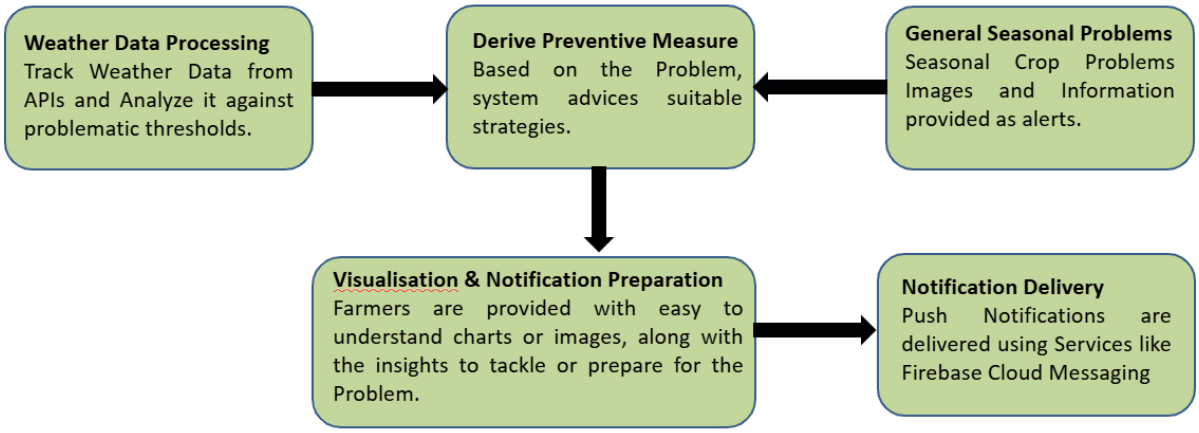


Figure 4: Notification Feature

5 Testing Module

5.1 Testing Strategies

- **Unit Testing:** Each component (data loaders, model layers, UI elements) is tested in isolation to ensure they function as expected.
- **Integration Testing:** Verifies seamless interaction between the Flutter frontend, Node.js backend, and TensorFlow model, ensuring data flows correctly through the system.
- **System Testing:** Validates the complete pipeline—from data input (image capture placeholder) to processing and output display—using simulated user scenarios.
- **Performance Testing:** Measures inference latency, memory usage, and overall response times, particularly on mobile devices.
- **User Acceptance Testing (UAT):** Early stage testing with potential users (farmers and agricultural experts) to validate UI design, usability, and the relevance of recommendations.

5.2 Example Test Cases

Table 1: Test Cases

Description	**Input**	**Expected Output**
Disease detection (Healthy Leaf)	Leaf image of healthy plant	System outputs "Healthy"
Disease detection (Diseased leaf)	Leaf image with a disease	Correct Disease Name
Recommendation check	Disease classification result	Relevant Suggestions
Weather alert generation	Location + Weather data	Alert displayed

6 Performance of the Project Developed (So Far)

6.1 Data Preparation & Oversampling

- Successfully merged images from multiple sources into a unified dataset with combined class names.
- Applied oversampling to balance the dataset, ensuring a more uniform distribution across disease classes, thereby improving model training effectiveness.

6.2 Teacher-Student Model Architecture

6.2.1 Teacher Models

- **Architectures Used:** EfficientNetB1, MobileNetV3Large, and MobileNetV2.
- **Outcome:** Each model captures unique feature representations, and when fused using a learnable weighted scheme, they provide a robust supervisory signal to the student.

6.2.2 Ensemble Fusion & Distillation

- **Weighted Fusion:** The teacher outputs and feature maps are dynamically weighted to form a comprehensive supervisory signal.
- **Multi-Level Distillation:**
  - **Output-Level:** Student predictions are aligned with the softened outputs of the teacher ensemble using KL divergence.
  - **Feature-Level:** Intermediate representations of the student are aligned with the fused teacher features, ensuring deeper semantic consistency.

Result: Preliminary evaluations indicate that the distilled student model achieves high accuracy ($\sim$ 97%+) while maintaining a significantly reduced model size, making it well-suited for mobile deployment.

Below is a table summarizing the key evaluation metrics based on the results:

Table 2: Key Evaluation Metrics

Metric	Value
Test Accuracy	98.08%
Precision (Macro Avg)	98.0%
Recall (Macro Avg)	98.0%
F1 Score (Macro Avg)	98.0%
Inference Latency	∼44 ms/step
Model Size	1.6 MB

6.3 Mobile UI & Recommendations Database

- **Mobile UI**
  - **Current Status:** The Flutter-based mobile UI is in the preliminary design phase. It features a polished frontend layout; however, essential functionalities like AI models are yet to be integrated.
  - **Future Enhancements:**
    - Integration of backend functionalities for image capture.
    - Addition of user authentication, push notifications, and data analytics dashboards.
- **Recommendations Database**
  - **Purpose:** Stores disease-to-treatment mappings. Enables the system to provide actionable recommendations based on disease classification.
  - **Integration:** Although basic queries are functional, the database integration with the mobile app is scheduled for future development.

6.4 Preliminary Observations

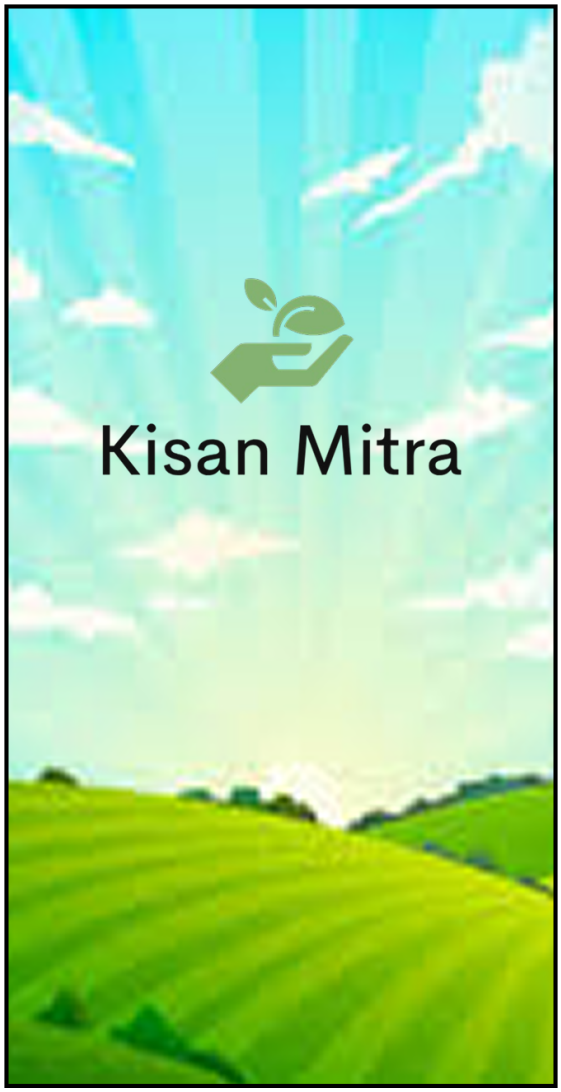
- **Model Accuracy:** The teacher-student ensemble approach shows promising accuracy improvements over single-model baselines.
- **Inference Efficiency:** Early tests indicate that the student model operates with low latency on mobile hardware.

- **Scalability:** The modular design allows easy incorporation of new diseases, treatment protocols, and extended functionalities.

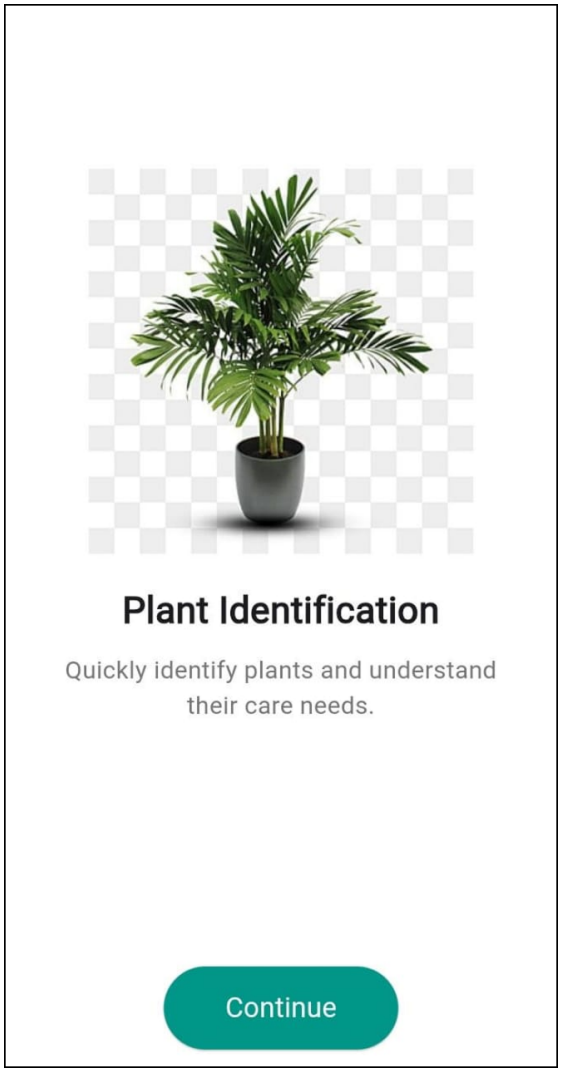
7 Output Screens

7.1 Current State

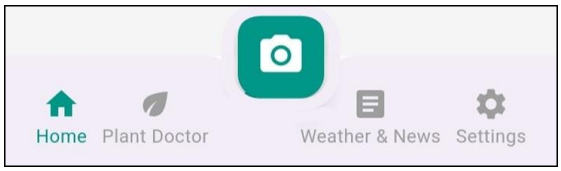
Mobile UI: The frontend, built using Flutter, is fully designed to provide a seamless user experience.



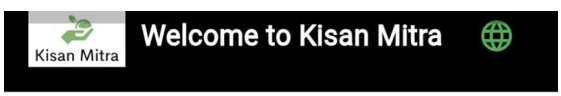
(a) UI Screen 1



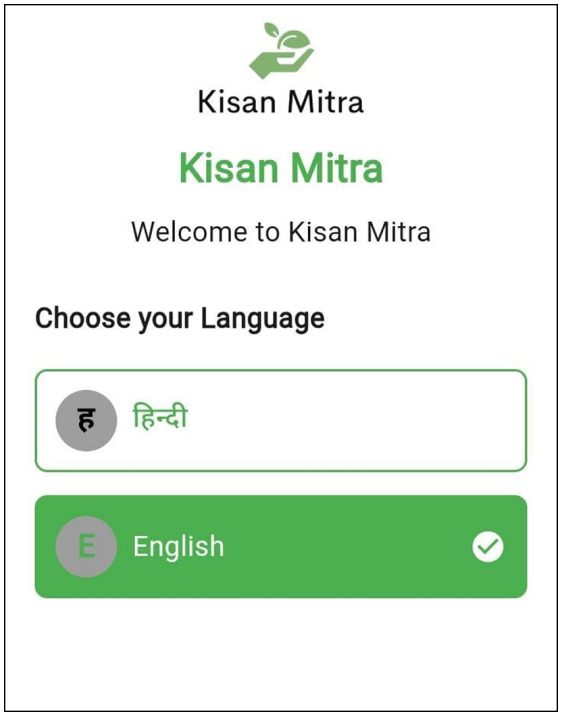
(b) UI Screen 2



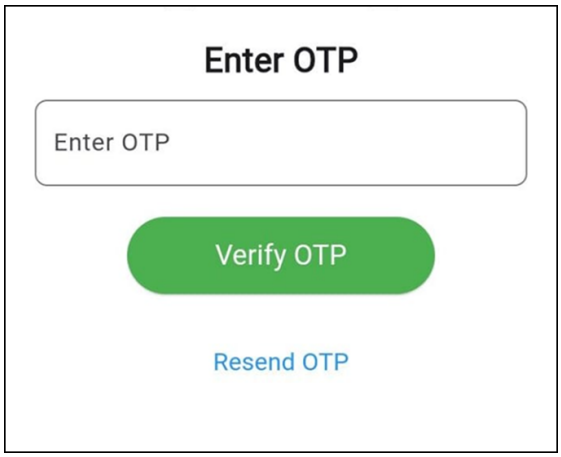
(c) UI Screen 3



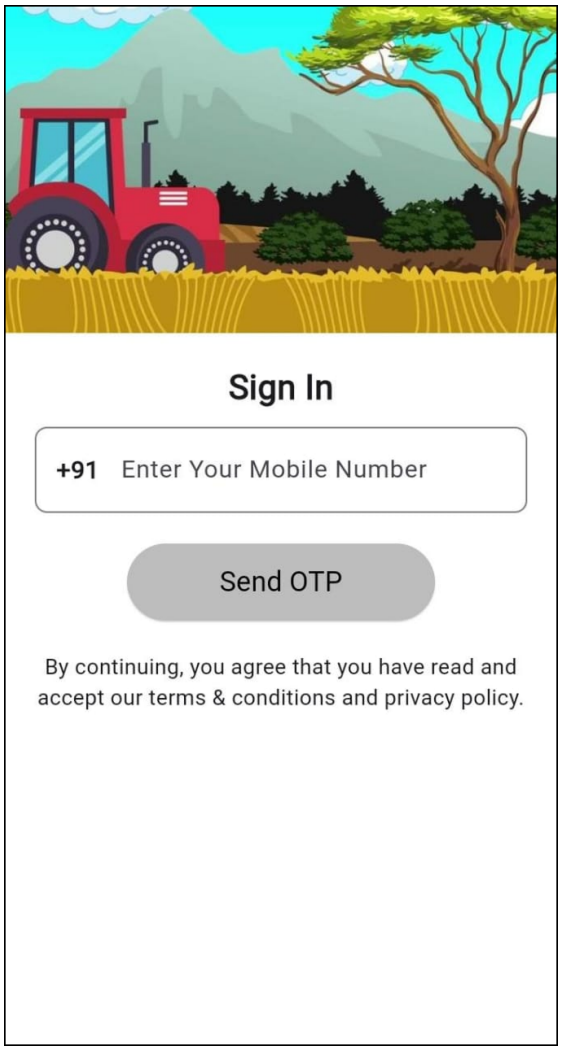
(d) UI Screen 4



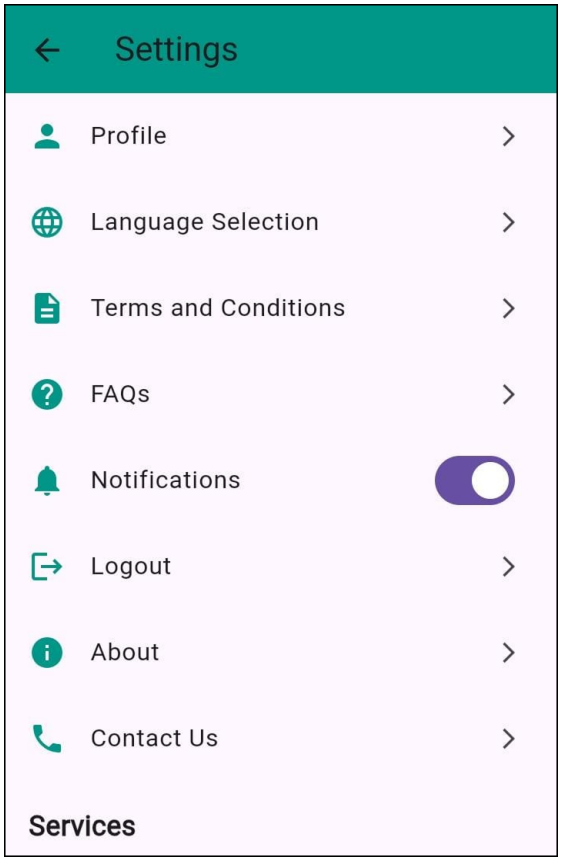
(a) UI Screen 5



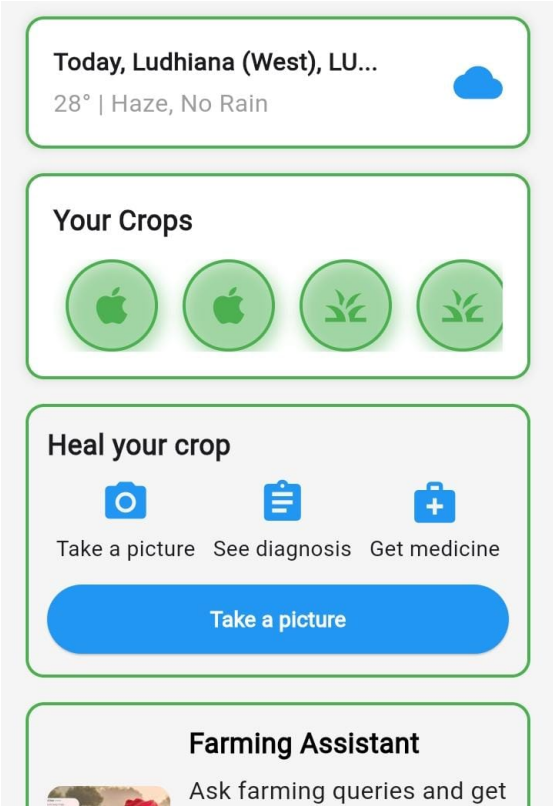
(b) UI Screen 6



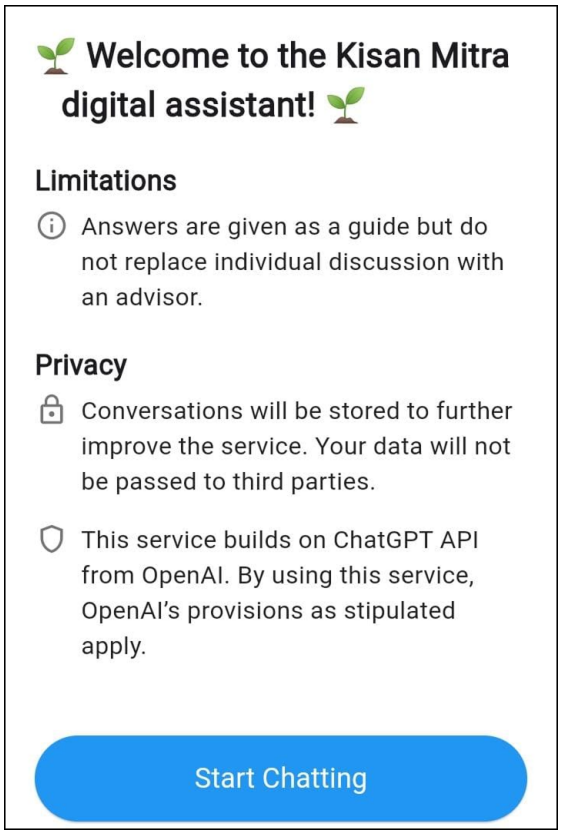
(c) UI Screen 7



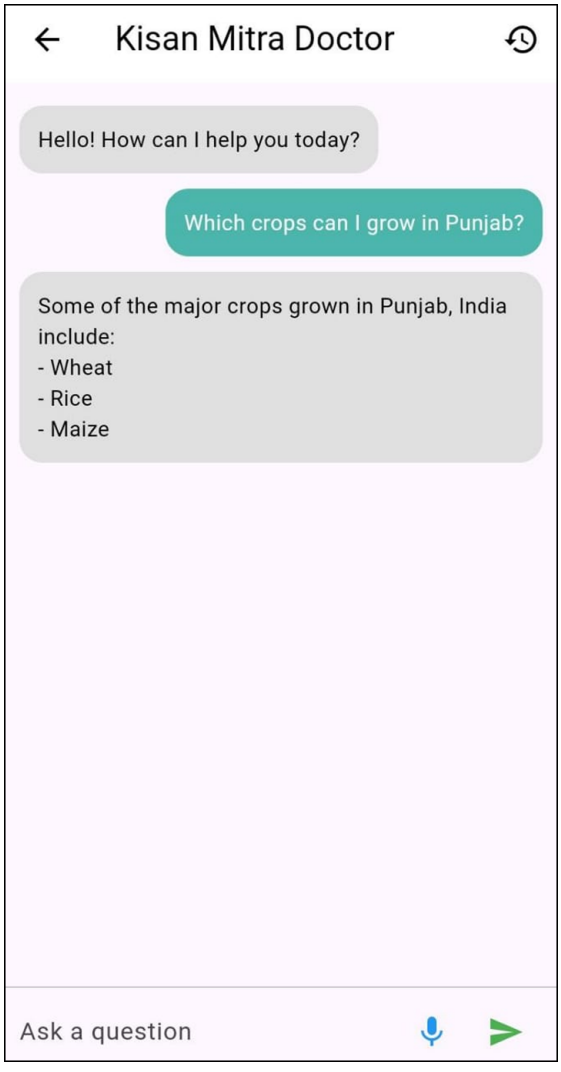
(d) UI Screen 8



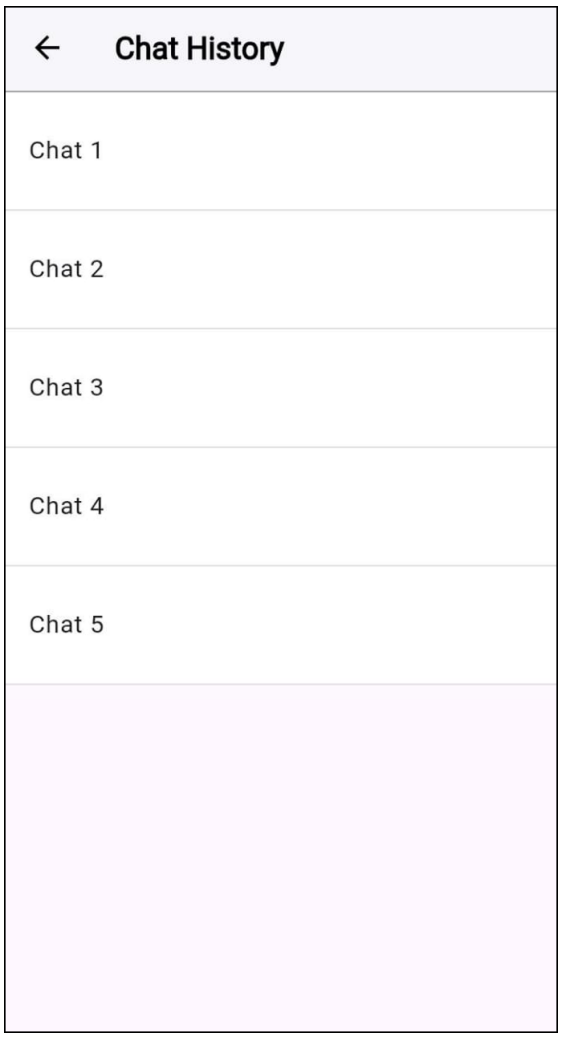
(a) UI Screen 9



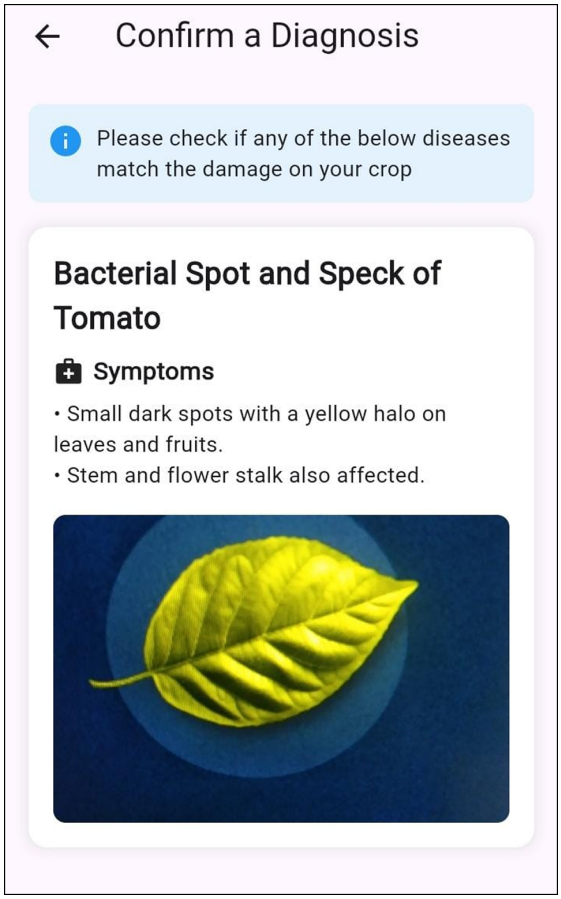
(b) UI Screen 10



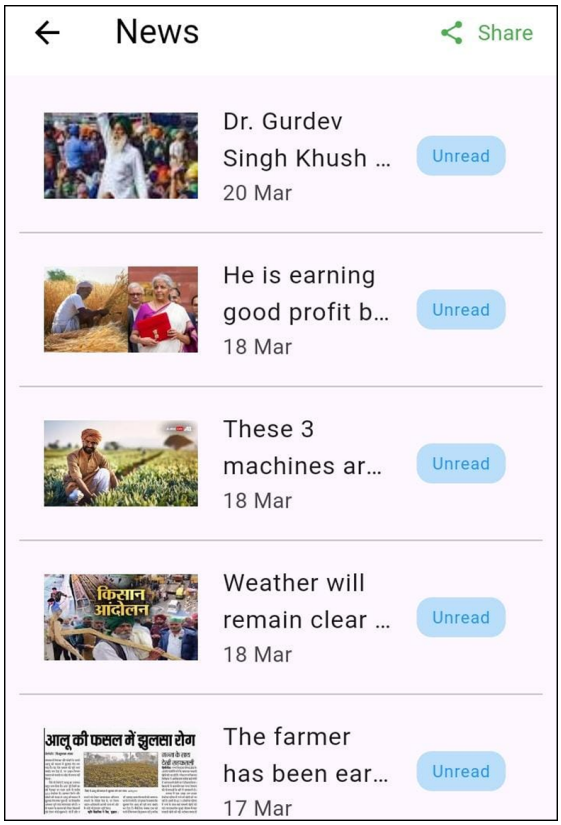
(c) UI Screen 11



(d) UI Screen 12

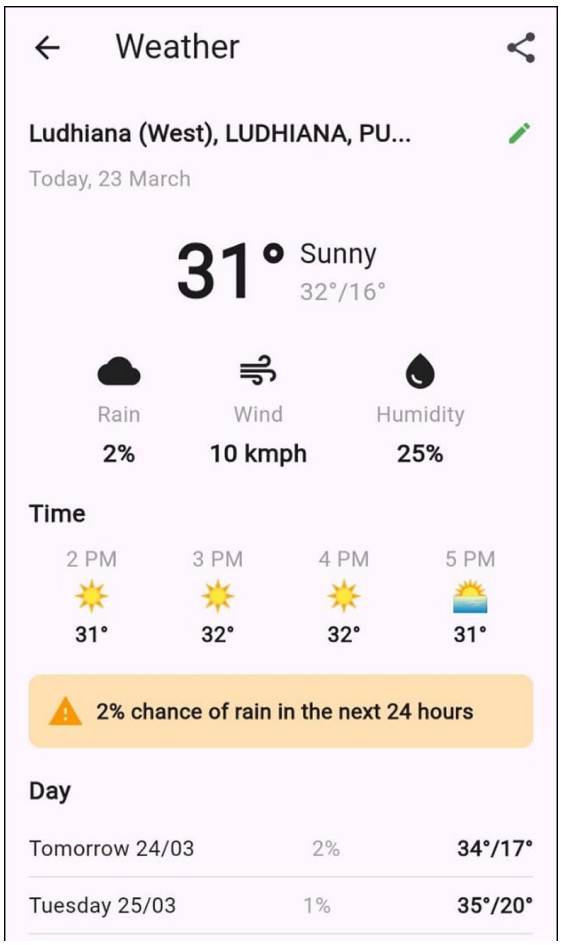


(a) UI Screen 13

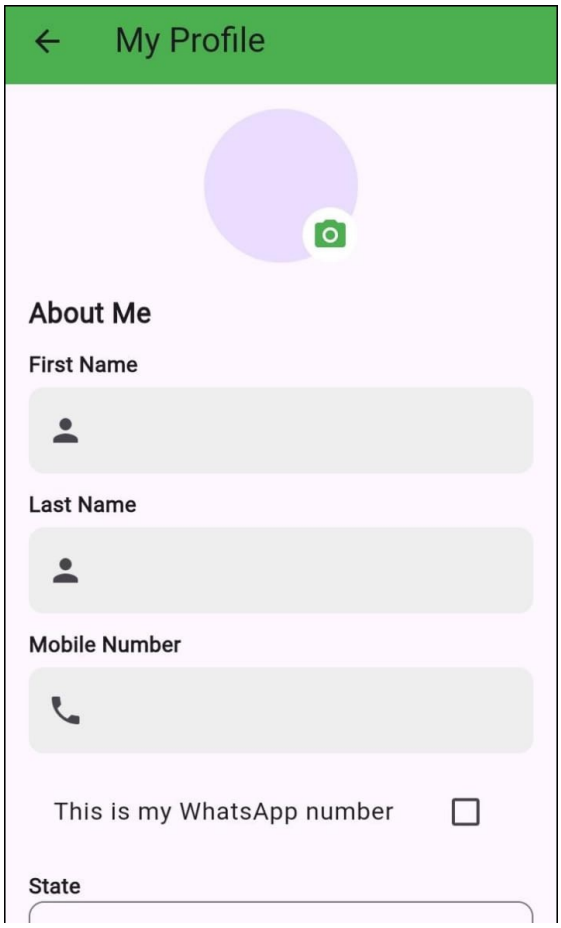


(b) UI Screen 14

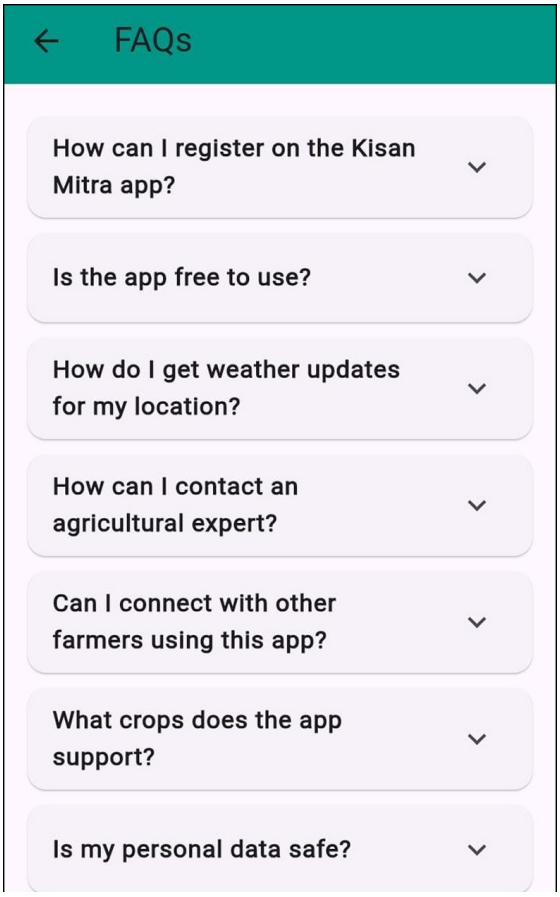
(c) UI Screen 15



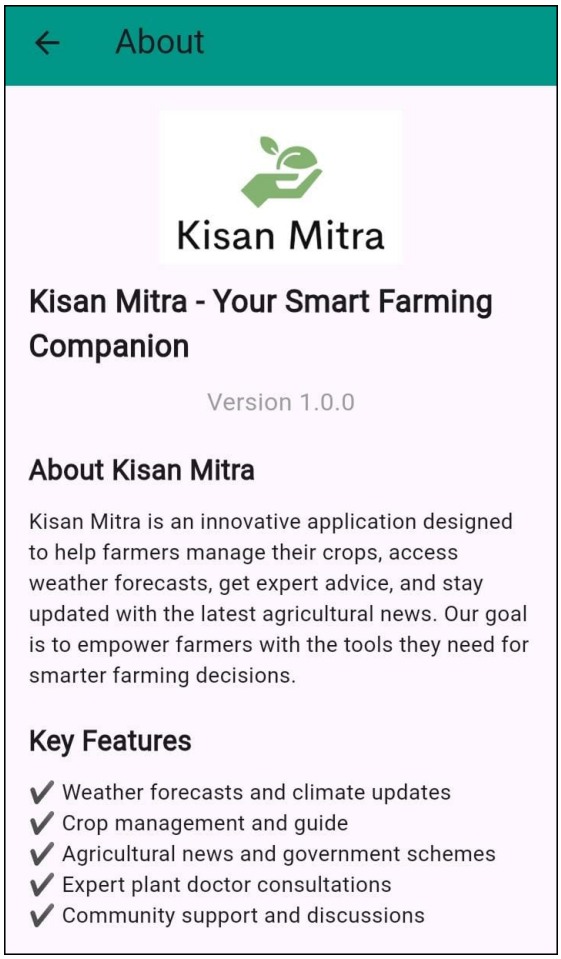
(d) UI Screen 16



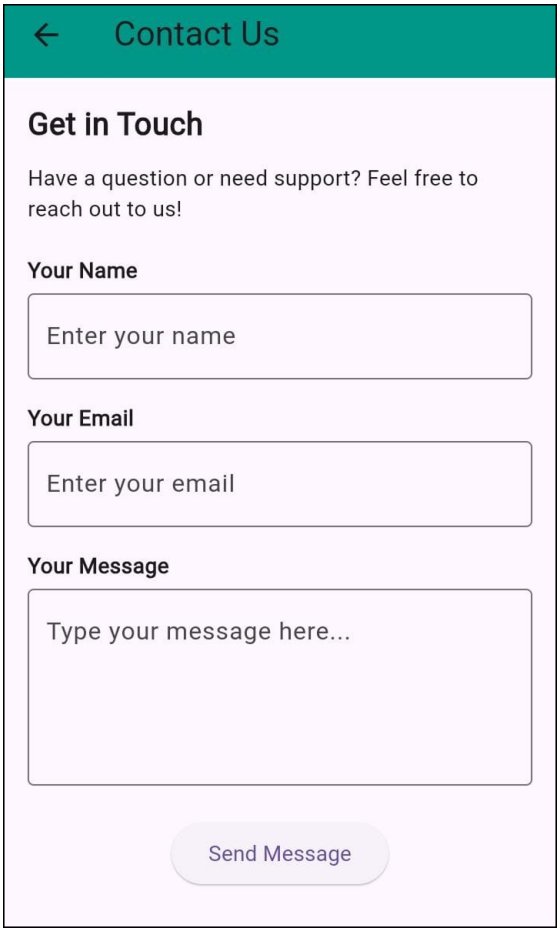
(a) UI Screen 17



(b) UI Screen 18



(c) UI Screen 19



(d) UI Screen 20

7.2 Model Classification Results and Confusion Matrix on Test Set

The results indicate robust performance with high accuracy and minimal misclassifications across classes, as shown by the confusion matrix.

```
43/43 ━━━━━━━━━━━━━━━━━━━━ 3s 44ms/step
Test Accuracy: 0.9808
Classification Report:
```

	precision	recall	f1-score	support
Apple_Apple Scab	1.00	1.00	1.00	51
Apple_Black Rot	1.00	1.00	1.00	50
Apple_Cedar Apple Rust	1.00	1.00	1.00	44
Apple_Healthy	0.94	1.00	0.97	51
Bell Pepper_Bacterial Spot	1.00	0.98	0.99	48
Bell Pepper_Healthy	0.96	0.98	0.97	50
Cherry_Healthy	0.98	1.00	0.99	46
Cherry_Powdery Mildew	1.00	1.00	1.00	43
Corn (Maize)_Cercospora Leaf Spot	0.95	0.98	0.96	41
Corn (Maize)_Common Rust	0.98	1.00	0.99	48
Corn (Maize)_Healthy	1.00	1.00	1.00	47
Corn (Maize)_Northern Leaf Blight	0.98	0.94	0.96	48
Grape_Black Rot	1.00	0.98	0.99	48
Grape_Esca (Black Measles)	0.98	1.00	0.99	48
Grape_Healthy	1.00	1.00	1.00	43
Grape_Leaf Blight	0.93	1.00	0.97	43
Peach_Bacterial Spot	1.00	1.00	1.00	46
Peach_Healthy	1.00	0.91	0.95	44
Potato_Early Blight	0.96	1.00	0.98	49
Potato_Healthy	0.98	1.00	0.99	46
Potato_Late Blight	0.98	0.98	0.98	49
Strawberry_Healthy	1.00	1.00	1.00	46
Strawberry_Leaf Scorch	1.00	1.00	1.00	45
Tomato_Bacterial Spot	0.95	0.93	0.94	43
Tomato_Early Blight	0.96	0.96	0.96	48
Tomato_Healthy	0.96	1.00	0.98	49
Tomato_Late Blight	0.98	0.94	0.96	47
Tomato_Septoria Leaf Spot	0.98	0.93	0.95	44
Tomato_Yellow Leaf Curl Virus	1.00	0.94	0.97	49
accuracy			0.98	1354
macro avg	0.98	0.98	0.98	1354
weighted avg	0.98	0.98	0.98	1354

Figure 10: Classification Report

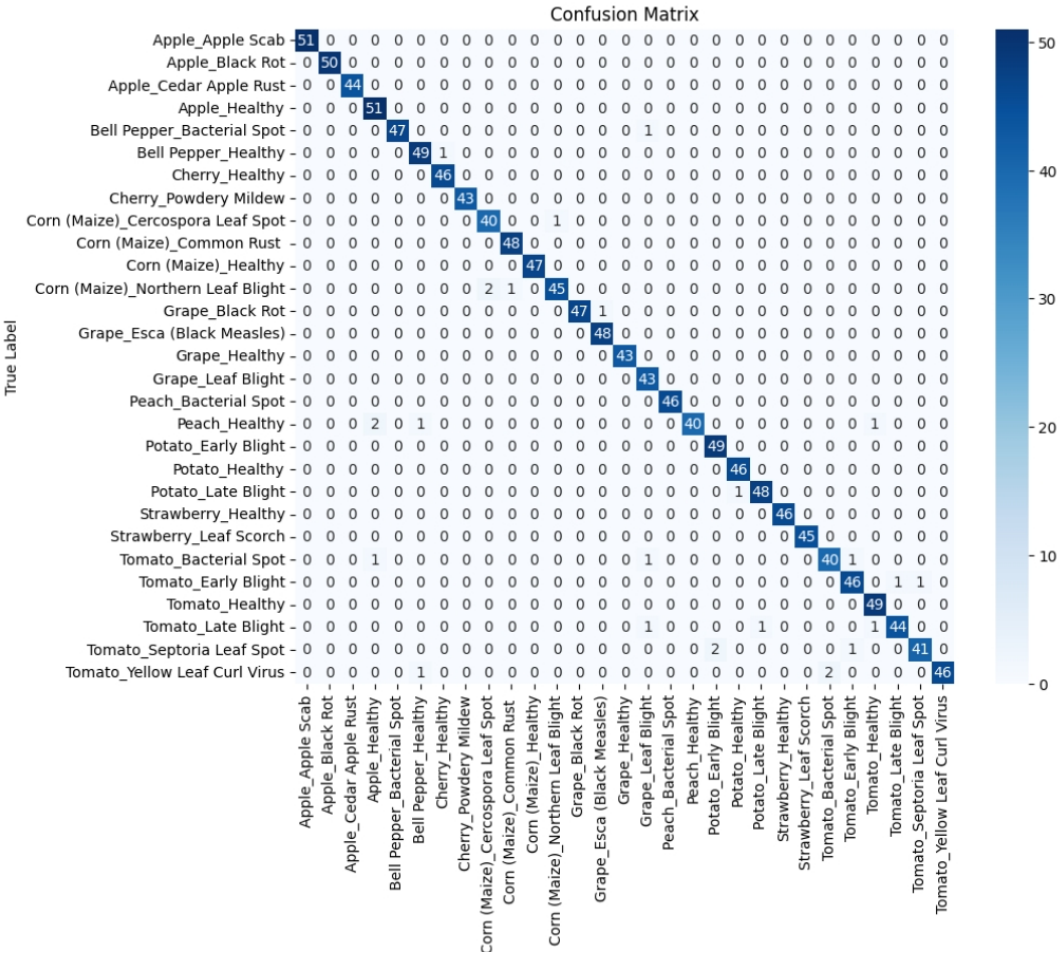


Figure 11: Confusion Matrix

7.3 Recommendations Result

Figure 12: Sample Recommendations Screen

7.4 Planned Future Enhancements

- **Functional Modules:**
  - Image Capture: Integrate camera functionalities to allow users to capture and upload plant images.
  - Classification Display: Dynamically show disease classification results and confidence scores.
  - User Authentication & Dashboard: Enable secure logins and personalized data analytics for user engagement.
  - Push Notifications & Alerts: Implement real-time alerts based on weather data and disease outbreak predictions.

References

- PlantVillage Dataset – Kaggle Dataset
  https://www.kaggle.com/datasets/tushar5harma/plant-village-dataset-updated
- MobileNetV3 – Howard et al., Searching for MobileNetV3 — IEEE Conference Publication — IEEE Xplore https://ieeexplore.ieee.org/document/9008835
- EfficientNet – Tan & Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks
  https://arxiv.org/abs/1905.11946
- MobileNetV2 – Chetan, G., Lavanya, K., Vamsi Krishna, S. B. S. S. S., & Venkata Lakshmi, D. (2024). Plant Disease Identification Using MobileNetV2 and InceptionV3. IEEE. https://ieeexplore.ieee.org/document/10867205
- Knowledge Distillation – Huang, Q., Wu, X., Wang, Q., Dong, X., Hao, G., et al. (2023). Knowledge Distillation Facilitates the Lightweight and Efficient Plant Diseases Detection Model, SPJ.
  https://spj.science.org/doi/10.34133/plantphenomics.0062
- TensorFlow Documentation – https://www.tensorflow.org/api_docs
- Flutter Documentation – https://docs.flutter.dev/
- Node.js Documentation – https://nodejs.org/api/all.html