B.M.S. COLLEGE OF ENGINEERING BENGALURU

Autonomous Institute, Affiliated to VTU

OOMD Mini Project Report

CROP DISEASE DETECTION

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

HARSHA B (1BM23CS107)

G M KUSUMA (1BM24CS405)

GAYATHRI S (1BM24CS406)

JAYASHREE TARAI (1BM24CS407)

Department of Computer Science and Engineering
B.M.S. College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
2023-2024

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, **HARSHA B(1BM23CS107), G M KUSUMA(1BM24CS405), GAYATHRI S(1BM24CS406), JAYASHREE TARAI(1BM24CS407)** students of 5th Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this OOMD Mini Project entitled "**CROP DISEASE DETECTION** " has been carried out in Department of CSE, B.M.S. College of Engineering, Bangalore during the academic semester March - July 2024. I also declare that to the best of our knowledge and belief, the OOMD mini Project report is not from part of any other report by any other students.

Signature of the Candidate

HARSHA B (1BM23CS107)

G M KUSUMA (1BM24CS405)

\GAYATHRI S (1BM24CS406)

JAYASHREE TARAI (1BM24CS407)

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the OOMD Mini Project titled "**CROP DISEASES DETECTION"** has been carried out by **HARSHA B(1BM23CS107), G M KUSUMA(1BM24CS405), GAYATHRI S(1BM24CS406), JAYASHREE TARAI(1BM24CS407)** during the academic year 2023-2024.

Signature of the Faculty in Charge

Table of Contents

Chapter 1: Problem Statement

Problem Statement: Crop Disease Detection Technology — Development and Implementation in an Agricultural Monitoring System
The project aims to build a system that uses image-based machine learning to detect diseases in crops (especially via leaf images) and support timely intervention for farmers. This system will help in early detection of plant diseases, reducing crop loss and improving yield, thereby supporting sustainable agriculture and food security.

Target Audience:

- **Primary users:** Farmers, agricultural extension officers, agronomists.
- **Secondary users:** Researchers in agritech, government agriculture departments, agribusiness companies.
- **Beneficiaries:** Local farming communities (more yield, less loss), and consumers (more stable supply).

Problems to Solve:

- **Early Disease Identification:** Many farmers lack rapid, reliable diagnosis tools — manual inspection is slow, requires expertise, and may be delayed.
- **Scalability:** Handling large volumes of crop images from different fields, varied lighting, and plant types.
- **Accuracy and Reliability:** Reducing false positives and false negatives in disease classification, especially across different crop species and disease types.
- **Usability:** Providing a simple interface for non-technical users (farmers) to upload leaf images and get diagnostic feedback.
- **Resource Constraints:** Working with limited compute power (e.g., mobile or edge deployment), possibly weak internet connectivity in rural areas.
- **Ethical / Practical Concerns:** Ensuring data privacy (if farmers' data is stored), giving actionable advice (not just "disease detected" but also "what to do"), and avoiding over-reliance on technology without proper agronomic recommendations.

Operational Context:

- Deployment will be via a **mobile/web app** that farmers or extension workers can use by taking photos of leaves.
- The system may also integrate with **drone or UAV imagery** (if scaled up), to monitor large fields.
- It would use a **backend server** (or edge device) for inference and maintain a **database** of images + disease labels + suggested interventions (treatment, pesticide, agronomic advice).
- The system operates in areas with limited connectivity: images may be uploaded when connectivity allows, or processed locally if possible.

Purpose:
The crop disease detection system will enable early and scalable disease diagnosis, empowering farmers to act faster to contain disease spread. It reduces dependence on agronomists for basic disease identification, saving time and cost, and boosting productivity.

Mechanism:

1. **Capture:** Farmers take photos of leaves (or other plant parts) via a mobile app / web interface, or images are collected via drones.
2. **Preprocessing:** Images are normalized — resized, enhanced, de-noised to handle variations in lighting, angle, blur.
3. **Analysis / Inference:** A trained machine learning / deep learning model (e.g., a CNN) classifies the image into one of several disease categories (or "healthy").
4. **Matching / Diagnosis:** The model outputs a disease label (with a confidence score) and suggests possible remedies or agronomic practices.
5. **Recommendation & Guidance:** The system provides recommended treatments (pesticides, organic practices), and possibly preventive advice.
6. **Data Logging & Feedback:** Stores the image, diagnosis, and user feedback. Over time, this data can be used to retrain or improve the model.

Additional Considerations:

- **Model Bias & Generalization:** Regularly validate and audit the model on different crop species / geographies to ensure fairness and broad applicability.
- **Privacy & Security:** Encrypt images during transmission, control access to stored data, and give users transparency on how their images are used.
- **Regulatory Compliance:** Align with local agricultural data policies, especially if storing data at scale

Chapter 2: Software Requirement Specification

1. Introduction

1.1 Purpose of this Document

This SRS defines the requirements for the **Crop Disease Detection System (CDDS)**. It serves as a detailed guide for stakeholders (farmers, agronomists, project managers) and the development team, describing system objectives, features, functional and non-functional requirements, constraints, and design considerations.

1.2 Scope of the Document

- Covers all functional and non-functional requirements of CDDS.
- Includes interface requirements (user interface, APIs), performance metrics, data handling, and security.
- Designed to work as a mobile and/or web application with backend model inference and storage.

1.3 Overview

The CDDS system lets users (farmers, extension workers) upload images of crop leaves or plant parts. The system preprocesses images, feeds them into a disease detection ML model, and returns a diagnosis (disease + confidence). It also recommends action (treatment/prevention). The system stores data for future learning and supports offline operation or delayed upload.

2. General Description

2.1 Product Perspective

- Modular architecture: front-end app (mobile/web), backend server (model + database), optional edge component.
- Integration: possibility to connect with existing agricultural advisory systems, drone imaging platforms, or agritech databases.
- Data Sources: Leaf images (via phone, drone), metadata (crop type, location, date), user feedback.

2.2 Product Features

- **Disease Identification:** Classify images into healthy or disease categories.
- **Image Upload & Capture:** From device camera or file.
- **Preprocessing Module:** Normalize images, enhance quality, remove noise.
- **Model Prediction:** Use a trained deep learning model (e.g., CNN) to detect disease.
- **Recommendation Engine:** After diagnosis, recommend measures (treatments, agro-practices).
- **User Feedback Loop:** Allow users to confirm or correct diagnosis to improve model over time.

- **Audit / History Log:** Store past diagnoses, images, and feedback.
- **Role-based Access:** Different roles (farmer, extension officer, admin) with different permissions.
- **Offline Support:** Cache images or queue upload when connectivity is low.
- **Reporting / Analytics:** Dashboard for agronomists or system admins to see disease trends, image statistics.

2.3 User Types / Roles
- **Farmers:** Upload leaf images, see diagnosis, view recommendations.
- **Agricultural Extension Officers / Agronomists:** Use the system to monitor disease occurrences, validate predictions, provide advice.
- **Admins / System Maintainers:** Train / retrain models, manage database, monitor usage.

2.4 Operating Context
- Runs on **Android / iOS mobile**, or via **web browser**.
- Backend server (cloud or local) with GPU (or CPU) to run inference.
- Database for storing images, predictions, feedback.

3. Functional Requirements

3.1 User Authentication and Access Control
- Registration / Login (email, phone, or social login)
- Role-based permissions (farmer, agronomist, admin)
- Optional two-factor authentication for sensitive roles (admin).

3.2 Image Capture and Preprocessing
- Allow users to capture images via device camera.
- Allow upload of images from gallery.
- Preprocess image: resize, normalize, denoise, possibly segment leaf region.

3.3 Disease Detection / Prediction
- Run ML inference on preprocessed image.
- Support classification into multiple diseases + "healthy" class.
- Show confidence scores for each prediction.
- Provide top-k predictions, not just the top one (if helpful).

3.4 Recommendation Engine
- Based on predicted disease, give customized treatment advice (chemical / organic / best practices).
- Provide preventive advice (crop rotation, resistant variety, cultural practices).
- Optionally, link to relevant agronomic resources / articles.

3.5 Feedback and Learning Loop
- Let user confirm or correct prediction.
- Store feedback to improve data quality.
- Optionally, use feedback data to retrain or fine-tune the model periodically.

3.6 Data Storage and Audit
- Maintain a log of all uploaded images, predictions, user feedback.
- Allow agronomists / admin to access historical data.
- Provide analytics (how many images per disease, trends over time).

3.7 Offline / Delayed Upload Support
- If user is offline, cache images locally.
- Upload queued images when connection restores.

3.8 API Integration
- Provide REST APIs for: image upload, prediction request, feedback submission, fetching history.
- Secure API endpoints (authentication, rate limiting).

4. Interface Requirements

4.1 Software Interfaces
- Backend: REST API (HTTPS)
- ML Model: inference module (e.g., TensorFlow, PyTorch) integrated into backend.

4.2 User Interfaces
- **Mobile UI (Farmer):** Simple screen to capture/upload image, view diagnosis and advice, give feedback.
- **Web / Admin Dashboard:** For agronomists/admins to view history, statistics, retraining controls.
- **Analytics / Reports Page:** For aggregated disease trends, usage statistics.

4.3 Communication Interfaces
- Use HTTPS (TLS) for secure data transmission.
- Optional: Push notifications (for users) when diagnosis is ready or when recommendations are updated.

5. Performance Requirements

- **Response Time:** Model inference should ideally happen within **2–5 seconds** for a single image (depending on deployment).

- **Concurrent Users:** System should support (say) **hundreds of users simultaneously** (depending on scale).
- **Storage:** Start with, for example, **500 GB** of image storage, scalable as new data comes.
- **Accuracy:** Aim for **≥ 90% classification accuracy** (or a reasonable benchmark based on dataset).
- **Error Rates:** Strive for low false positive / false negative rates; acceptable threshold to be defined based on domain (e.g., < 5% false negative).

6.Design Constraints

- **Model Limitations:** The accuracy depends on the dataset diversity (crop species, disease types, image quality).
- **Hardware Constraints:** If deployed on mobile, model size and inference speed matter.
- **Connectivity:** Rural areas may have intermittent Internet; system must handle offline scenarios.
- **Data Privacy / Regulations:** Agricultural data (images of farms) may be sensitive; need to conform to any data-privacy laws or policies.

7. Non-Functional Requirements

- **Security:** Encrypt data in transit (HTTPS) and at rest (e.g., AES). Implement role-based access.
- **Scalability:** System should scale (both storage and compute) as more images are collected / more users onboard.
- **Reliability / Availability:** Aim for high uptime (e.g., 99%) for the backend.
- **Portability:** The mobile app should work across Android and iOS; web app compatible with major browsers.
- **Usability:** The UI for the farmer should be very simple, with minimal steps.
- **Maintainability:** Logging, metrics, easy retraining mechanisms.
- **Ethical Use / Transparency:** Show how predictions are made (confidence), explain limitations, ask for user consent to store images.

8. Preliminary Schedule & Budget (Example)

8.1 Schedule

- **Phase 1 – Requirement Gathering & Design:** 2 months
- **Phase 2 – Dataset Collection & Model Training:** 4 months
- **Phase 3 – App Development (Mobile + Web):** 3 months
- **Phase 4 – Testing & QA:** 2 months
- **Phase 5 – Deployment & Maintenance:** Ongoing (with periodic retraining)
  Total Duration: ~ 11–12 months

8.2 Budget (Example – assuming small scale / academic project)

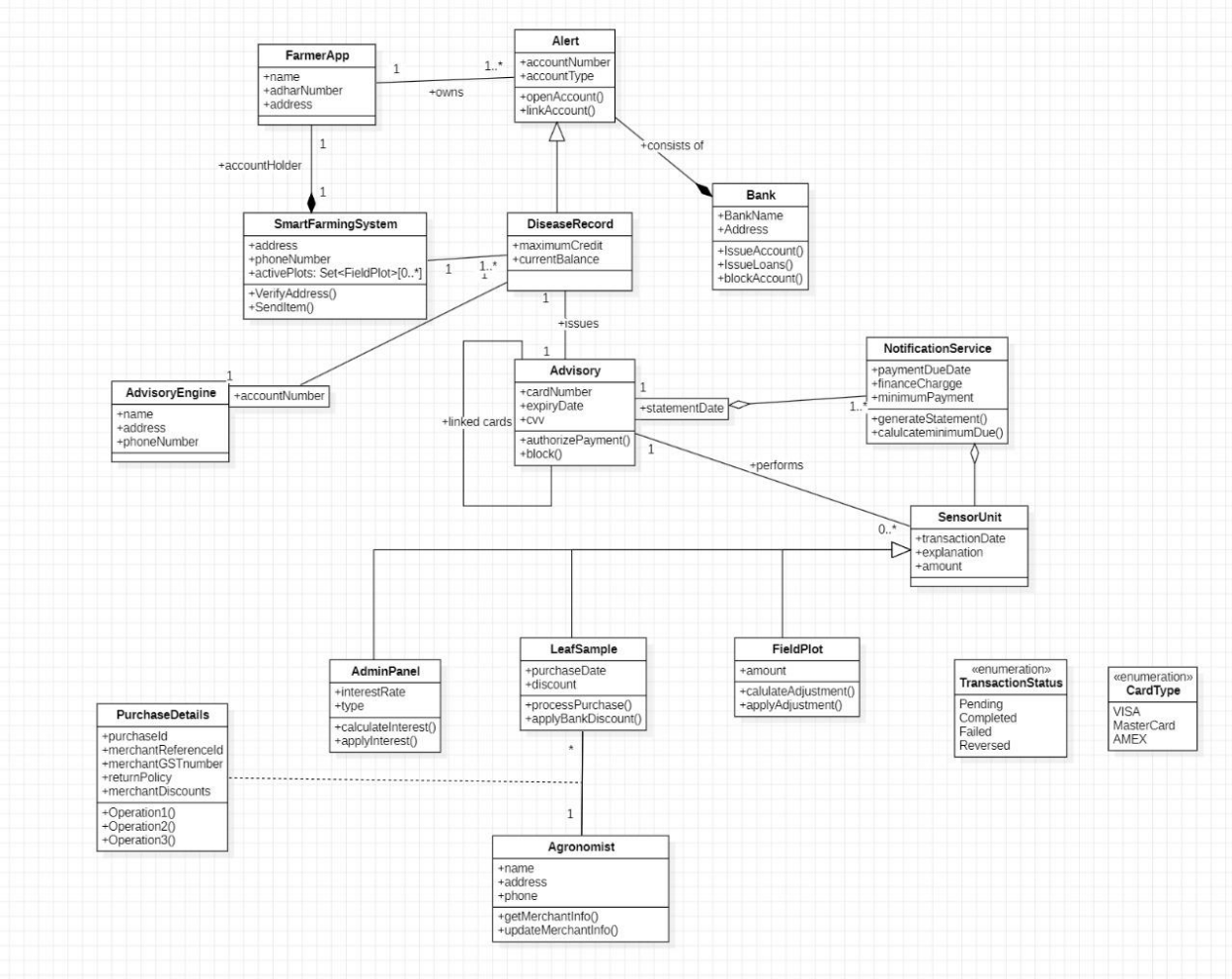
- **Model Development & Training:** ₹ 200,000

- **App Development:** ₹ 150,000
- **Testing & QA:** ₹ 50,000
- **Infrastructure / Hosting:** ₹ 50,000 (or more, if cloud GPU)
- **Miscellaneous / Contingency:** ₹ 50,000
- **Total Budget:** ~ ₹ 500,000 (just as a rough estimate — adjust as per real costs)

9. Conclusion

The **Crop Disease Detection System (CDDS)** will empower farmers and agricultural stakeholders with a powerful, scalable, and user-friendly tool for diagnosing plant diseases through image recognition. By combining deep learning models, secure data handling, feedback loops, and agronomic guidance, this system can significantly reduce crop loss, improve yield, and contribute to sustainable farming practices.

Chapter 3: Class Modeling



The Smart Farming Crop Disease Detection System enhances agricultural productivity by using sensors, advisory engines, and digital farmer services to detect plant diseases early and provide real-time suggestions. The following descriptions explain the relevance of each class in the UML diagram and how they support efficient farm management:

1. Farmer App

This class represents the mobile or web application used by farmers. It stores basic farmer details like name, Aadhaar number, and address. The Farmer App interacts with the Smart Farming System and helps farmers receive notifications, disease alerts, and advisory messages.

2. Smart Farming System

This is the **central controller** of the entire system.
It manages:

- Farmer accounts
- Field plots
- Disease records
- Communication with sensors and advisory engines

It verifies farmer addresses and sends important information such as disease alerts, recommendations, and reports.

3. Field Plot

Represents an individual agricultural plot owned by a farmer.
It maintains details such as:

- Field size or crop area
- Amount (crop yield or resource usage)
  It also supports adjustments like irrigation data, nutrient changes, or disease severity updates.

4. Sensor Unit

The Sensor Unit captures real-time environmental and soil parameters from the field.
Attributes include:

- Transaction date
- Explanation (e.g., "moisture level low")
- Amount (value recorded)

It automatically sends data to the Notification Service, enabling early detection of diseases.

5. Notification Service

Responsible for generating alerts and reminders for the farmer.
It performs:

- Calculation of minimum attention required (e.g., watering, spraying)
- Generating disease-related statements or recommendations

This ensures farmers are quickly informed when a disease risk is detected.

6. Disease Record

This class stores the complete history of diseases detected in a farmer's plots. Key details include:

- MaximumCredit (severity threshold)
- CurrentBalance (current disease intensity)

It connects farmers, advisory services, and banks for possible compensation or crop insurance processing.

7. Advisory

Represents advisory recommendations linked to a disease record. Attributes include:

- Card number / expiry date / CVV (used metaphorically here for linking farmer accounts)
- Functions to authorize or block advice, similar to validating expert recommendations

This class ensures that advice sent to the farmer is valid and verified.

8. Advisory Engine

This is the intelligence module of the system. It uses:

- AI-based disease prediction
- Weather forecasts
- Crop databases

It generates expert recommendations for farmers based on sensor inputs and disease records.

9. Admin Panel

Used by system administrators or agricultural officers. Admin tasks include:

- Calculating interest rates (for loans/subsidies)
- Applying interest policies
- Managing backend configurations

This class supports the financial and policy-related parts of the system.

10. Purchase Details

Stores purchase-related information for farmers.
Examples:

- Pesticides
- Fertilizers
- Sensors or tools

It records merchant details, discounts, transactions, and return policies.
It also provides multiple operations related to payments or invoicing.

11. Leaf Sample

Represents leaf samples collected for laboratory or AI-based disease diagnosis.
Includes:

- Purchase date
- Discount
- Processing functions

Leaf samples help in confirming diseases when automated detection is uncertain.

12. Agronomist

Agronomists are experts who provide personalized guidelines.
This class stores their:

- Name
- Contact information

They update merchant info and provide expert-level assistance for disease treatment.

13. Bank

Banks issue credit support or loans to farmers when disease impact affects crop yield.
Functions include:

- IssueAccount
- IssueLoans
- BlockAccount

This class becomes important in compensation workflows or recovery programs.

14. Alert

Represents alert messages delivered to farmers regarding:

- Disease outbreaks
- Sensor warnings
- Payment dues
- Advisory notifications

Alerts ensure rapid awareness and immediate farmer engagement.

15. Transaction Status (Enumeration)

Defines the possible states of a transaction:

- Pending
- Completed
- Failed
- Reversed

Used for purchase, advisory payments, and service charges.

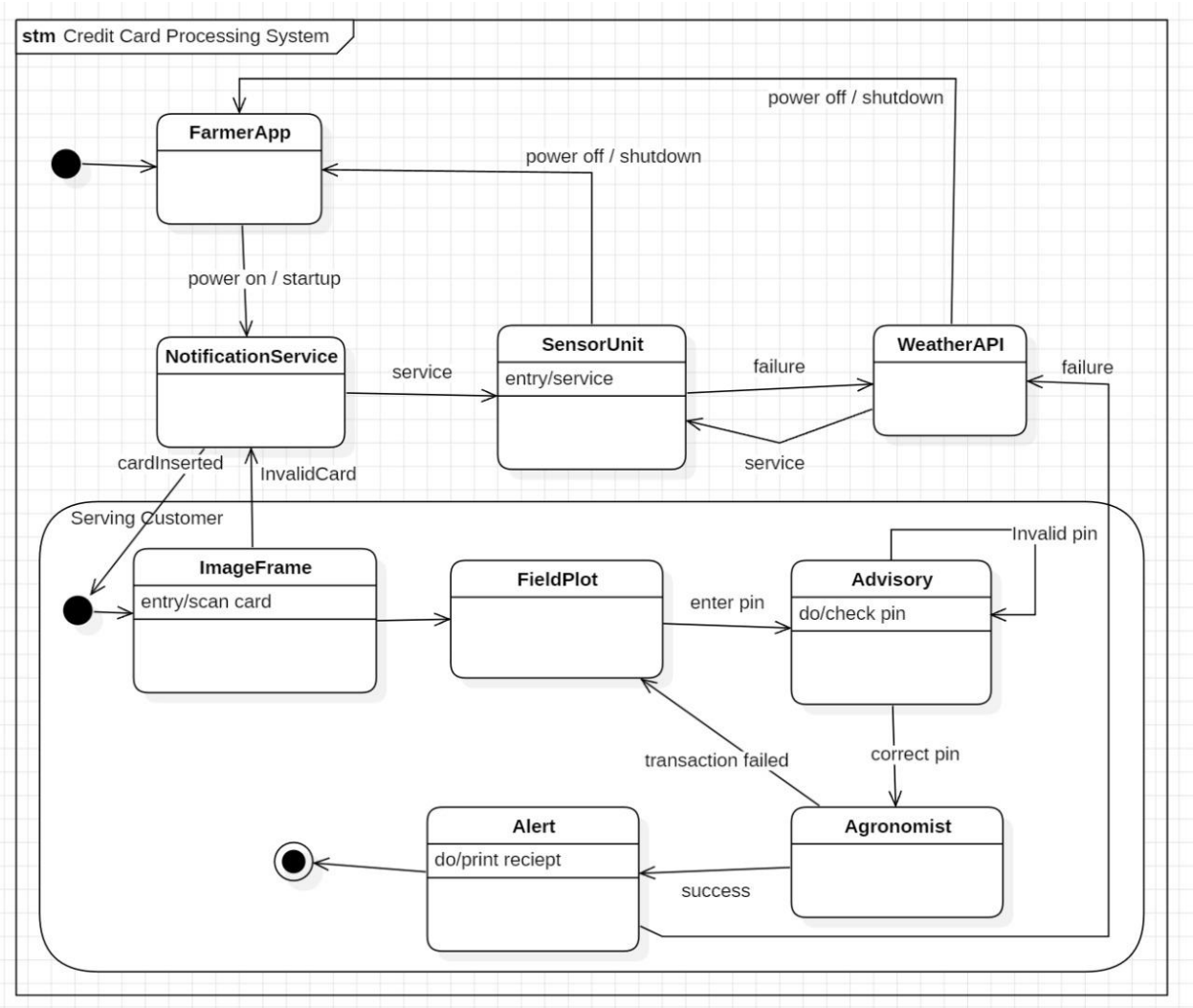
16. Card Type (Enumeration)

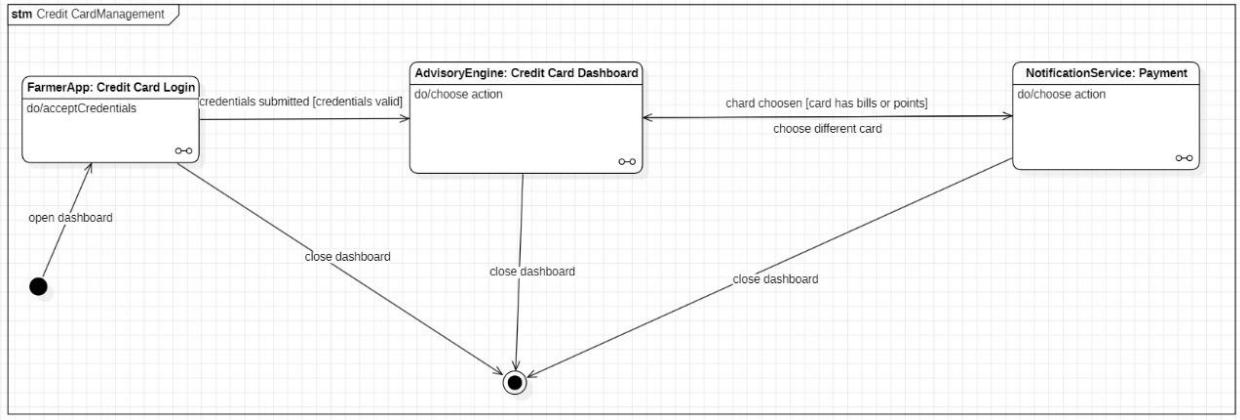
Represents available card types for payment or linking financial accounts:

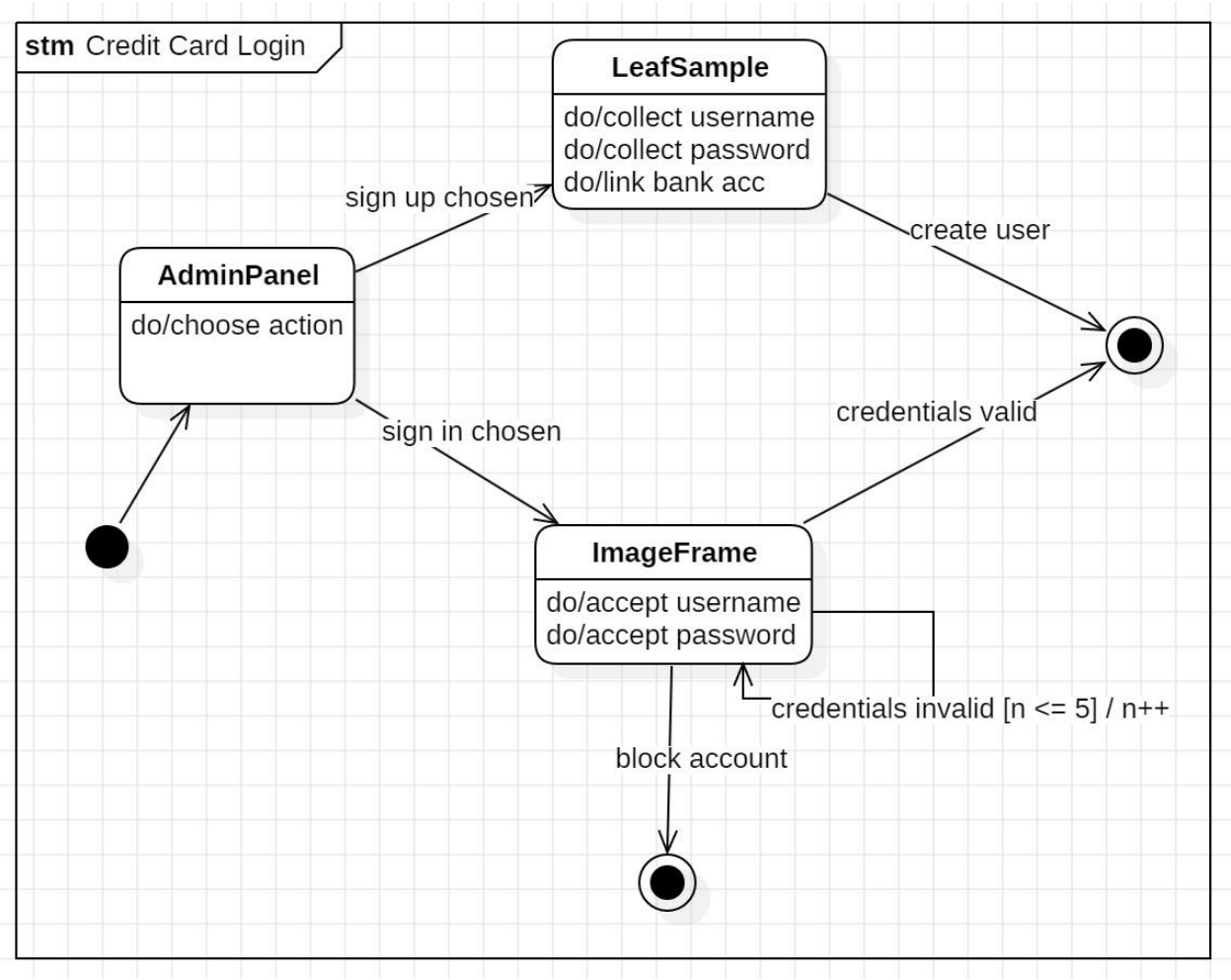
- VISA
- MasterCard
- AMEX

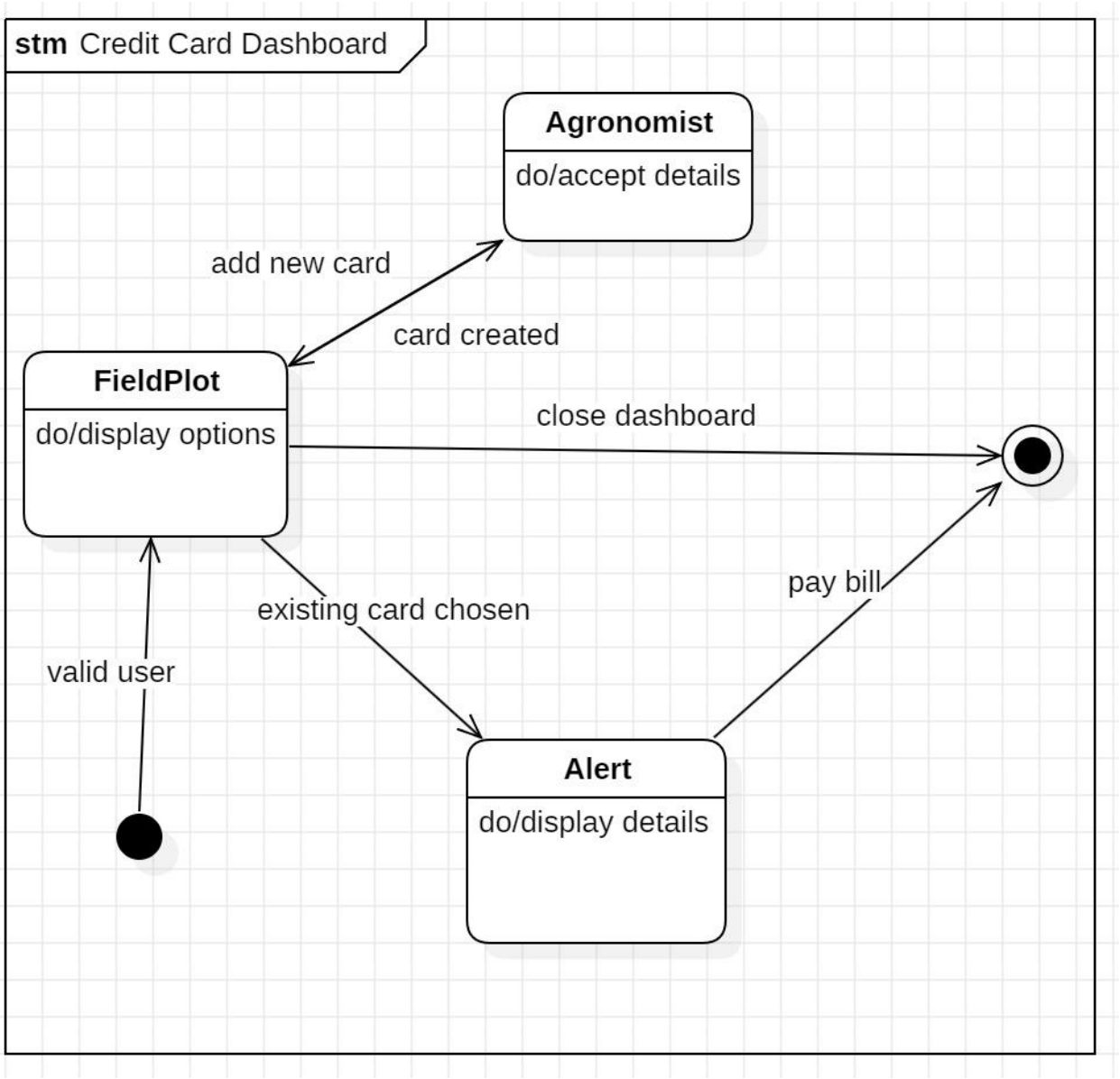
This is part of the advisory and purchase workflow.

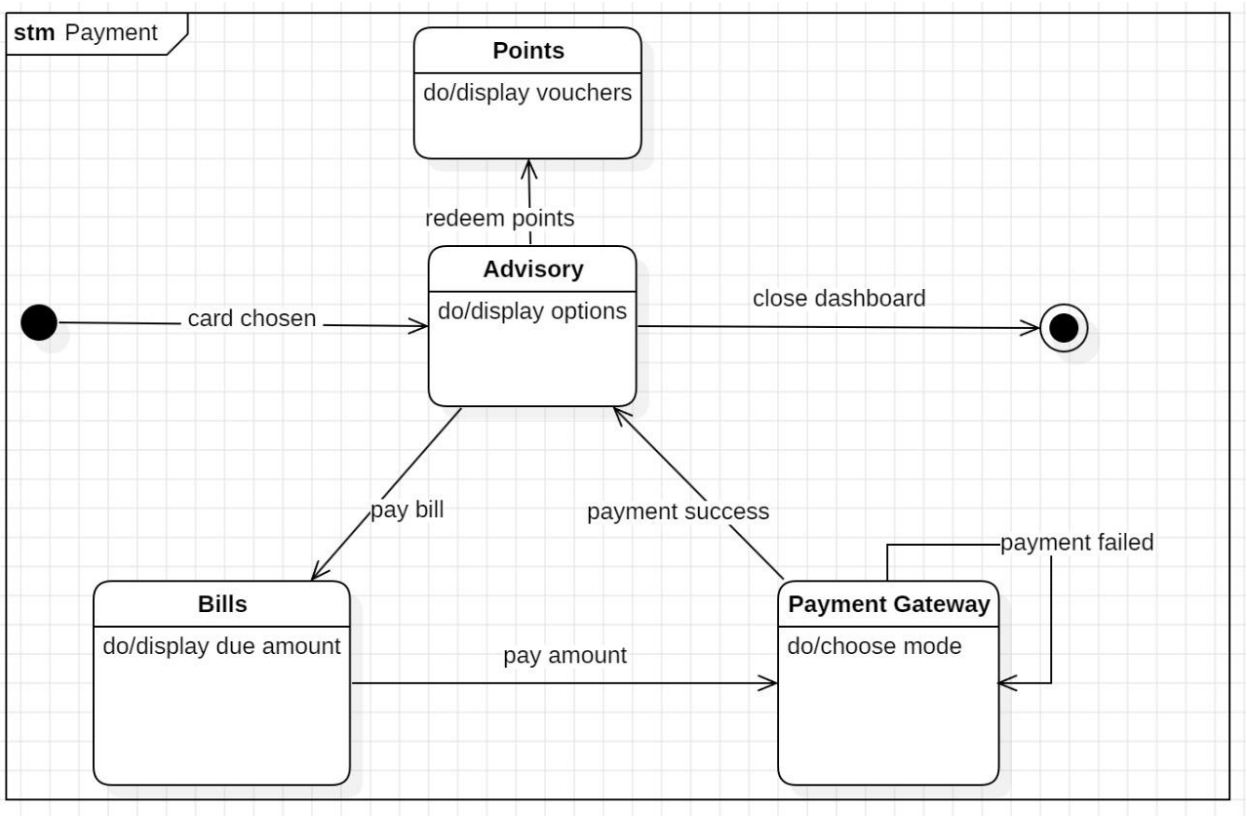
Chapter 4: State Modeling




stm Credit CardManagement
FarmerApp: Credit Card Login
do/acceptCredentials
credentials submitted [credentials valid]
AdvisoryEngine: Credit Card Dashboard
do/choose action
chard choosen [card has bills or points]
choose different card
NotificationService: Payment
do/choose action
open dashboard
close dashboard
close dashboard
close dashboard


stm Credit Card Login
LeafSample
do/collect username
do/collect password
do/link bank acc
sign up chosen
create user
AdminPanel
do/choose action
sign in chosen
credentials valid
ImageFrame
do/accept username
do/accept password
credentials invalid [n <= 5] / n++
block account


stm Credit Card Dashboard
Agronomist
do/accept details
add new card
card created
FieldPlot
do/display options
close dashboard
existing card chosen
pay bill
valid user
Alert
do/display details



Relevance of Each State (Diagram 1: Credit Card Management)

1. FarmerApp: Credit Card Login

- Accept Credentials: This state allows the user to enter login details and submit them for validation.
- Access Control: Ensures that only users with valid credentials can proceed to the dashboard.
- Session Initialization: Prepares the session environment after user login is attempted.

2. AdvisoryEngine: Credit Card Dashboard

- Choose Action: Displays all available credit-card actions (check bills, points, history) for the user to select.
- Card Verification: Validates the selected card before transferring the user to the payment module.

- Decision Handling: Routes the user to the appropriate next state based on their chosen option.

3. NotificationService: Payment

- Choose Action: Allows users to perform payment-related operations such as bill payment or checking pending dues.
- Payment Processing: Handles payment initiation and ensures correct transaction flow.
- Alternate Card Handling: Allows users to return and choose a different card when required.

4. Final State (Close Dashboard)

- Session Termination: Ends the current dashboard session when the user closes or exits.
- Cleanup: Ensures all temporary data is cleared for security and performance.
- User Redirection: Returns the user to a safe termination point after dashboard closure.

Relevance of Each Event (Diagram 1)

1. **Open Dashboard:** Initiates the login flow and transitions the system to the credential entry state.
2. **Credentials Submitted:** Sends the entered login details to the system for validation.
3. **Card Chosen:** Occurs when a card with bills or reward points is selected for further action.
4. **Choose Different Card:** Redirects back to the dashboard to allow another card to be selected.
5. **Close Dashboard:** Ends dashboard activity and transitions the system to the final state.

Relevance of Each State (Diagram 2: Credit Card Processing System)

1. FarmerApp

- Startup State: Represents the beginning of system operation after power-on.
- User Interaction Hub: Serves as the primary interface for the farmer to access card services.

- Shutdown Handling: Ensures proper termination when the system powers off.

2. NotificationService

- Card Insert Processing: Detects inserted cards and sends relevant messages to connected units.
- Invalid Card Handling: Notifies the system when an invalid card is detected.
- Customer Serving Transition: Directs the user to ImageFrame for card scanning.

3. SensorUnit

- Service Entry: Activates sensors for processing card-related information.
- System Coordination: Sends service updates and reacts to failures detected in communication.
- WeatherAPI Interaction: Exchanges status signals with the WeatherAPI during operations.

4. WeatherAPI

- Environmental Check: Provides weather-based data to the sensor unit if required.
- Failure Response: Sends failure signals when API service is unavailable.

5. ImageFrame

- Scan Card: Captures the card image for validation and forwards details to FieldPlot.
- Customer Entry: Marks the beginning of customer-serving workflow.

6. FieldPlot

- Card Data Analysis: Processes scanned card data for further actions.
- PIN Entry Transition: Moves to the Advisory state when PIN entry is required.

7. Advisory

- Check PIN: Validates PIN entered by the user.

- Error Handling: Sends "Invalid PIN" signals if verification fails.
- Success Routing: Directs the workflow to the Agronomist state if the PIN is correct.

8. Agronomist

- Successful Authentication: Represents successful completion of card authentication.
- Transaction Approval: Authorizes further operations after PIN verification.

9. Alert

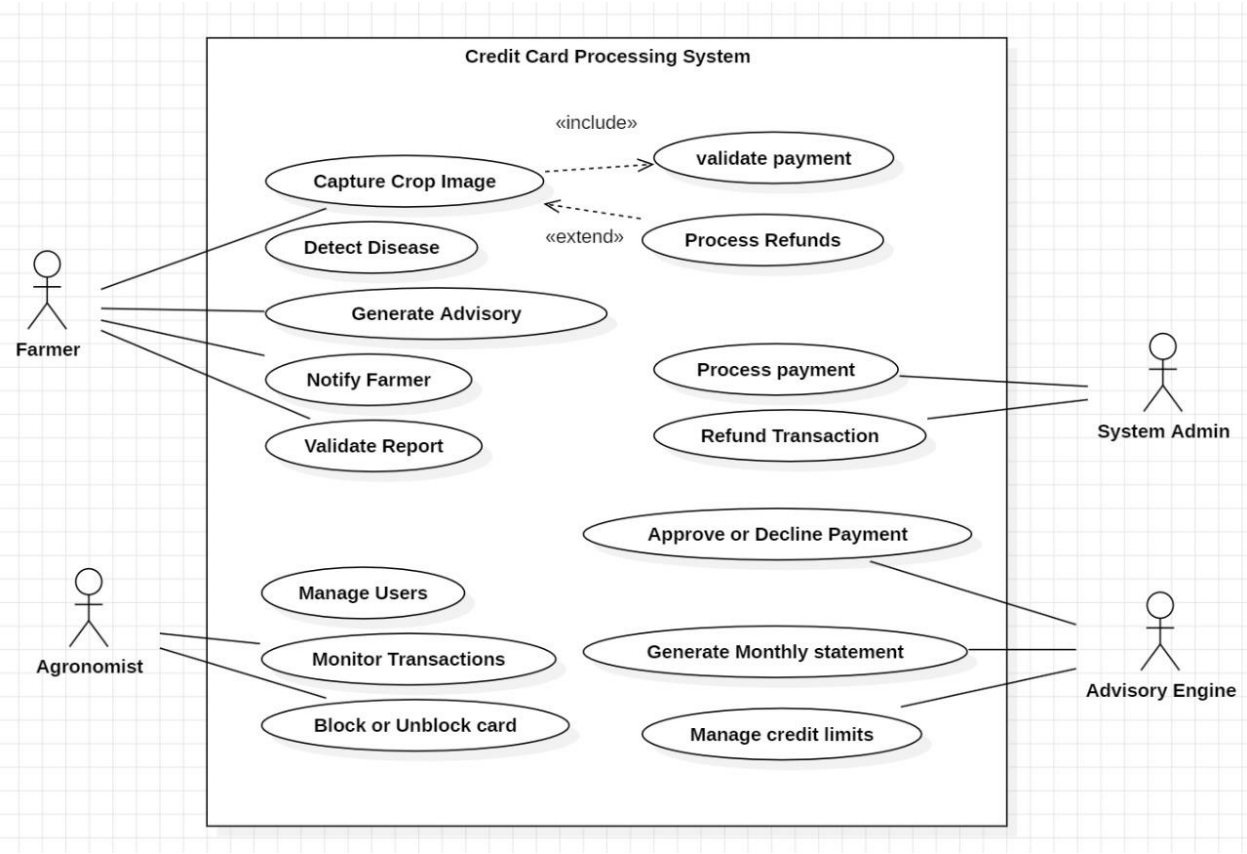
- Print Receipt: Generates and prints receipts following transaction completion.
- End of Transaction: Serves as the last state in customer-serving workflow.

Relevance of Each Event (Diagram 2)

1. **Power On / Startup:** Activates the FarmerApp and begins system initialization.
2. **Power Off / Shutdown:** Terminates system activity safely across all modules.
3. **Card Inserted:** Triggers NotificationService to start card validation.
4. **Invalid Card**: Indicates that scanned card data does not match system requirements.
5. **Service**: Enables communication between NotificationService, SensorUnit, and WeatherAPI.
6. **Failure:** Signals a communication or processing failure requiring corrective action.
7. **Scan Card:** Begins the card scanning process in the ImageFrame state.
8. **Enter PIN:** Moves the workflow from FieldPlot to Advisory for PIN verification.
9. **Invalid PIN:** Notifies that the user entered an incorrect PIN.
10. **Correct PIN:** Confirms successful user authentication and sends the system to Agronomist.
11. **Transaction Failed:** Sends user to Alert for issue notification.
12. **Success:** Completes the workflow and prints the required receipt.

Chapter 5: Interaction Modeling

USE CASE:



1. System Initialization

- Represents the beginning of all FRTC operations.
- Allows administrators to load system modules, activate camera networks, initialize AI models, and prepare the system for real-time monitoring.

2. System Shutdown

- Enables administrators to safely shut down the system.
- Ensures ongoing recognition, monitoring, and alerting processes are completed or terminated properly to avoid data loss.

3. Alert Management

- Citizens can report crimes or suspicious behavior.

- Alerts are generated and forwarded to officers, enabling immediate response and effective policing.
- Ensures timely awareness of threats or incidents.

4. Officer Interaction

- Allows officers to interact directly with the system during investigations.
- Officers receive alerts, acknowledge incidents, query suspect data, and take appropriate actions based on system insights.

5. Real-Time Monitoring

- Continuously analyzes camera feeds to detect suspicious activity or anomalies.
- Supports live surveillance of public areas, enhancing situational awareness for law enforcement.

6. Data Maintenance

- Ensures that all criminal records, logs, and facial recognition datasets are properly updated.
- Maintains system integrity by preventing outdated or corrupted data from affecting operations.

7. Escalation

- Handles incidents considered high-risk or requiring involvement beyond the responding officer.
- Notifies senior officers, specialized units, or administrators depending on severity.

8. Query Criminal Database

- Officers can retrieve detailed criminal histories, case files, and other vital information.
- Allows quick verification of suspects, supporting faster decision-making during field operations.

9. Feedback Submission

- Citizens and officers can provide feedback about system performance, false positives, or incident handling.
- Helps improve system reliability, transparency, and overall efficiency.

10. Fraud Detection

- Identifies potential spoofing attempts, such as using masks, printed photos, or digital manipulation.

- Protects the system from misuse and ensures only real faces are processed.

11. Data Archiving

- Stores old logs, recognition events, alerts, and criminal records in the archive for compliance and future reference.
- Helps optimize storage while preserving important data.

12. Face Recognition

- Core function that detects, analyzes, and identifies faces using machine learning.
- Supports suspect tracking, criminal identification, and real-time surveillance.

13. Backup and Recovery

- Creates secure system backups to prevent data loss during failures or cyber-attacks.
- Supports restoration of crucial data and system functionalities.

14. Model Update

- AI models are periodically retrained and updated with new datasets.
- Helps the system adapt to new patterns, facial variations, and evolving criminal behavior.

Interaction of Actors

1. Administrator

- Responsible for starting and shutting down the system.
- Manages high-level operations, configurations, and escalated incidents.
- Ensures the system remains functional, secure, and updated.

2. Citizen

- Acts as the initiator of crime reports or suspicious activity alerts.
- Helps bridge community involvement with law enforcement operations.

3. Officer

- Interacts heavily with the system through real-time monitoring, alert handling, and database queries.
- Uses insights provided by the FRTC to identify suspects, respond quickly, and manage incidents effectively.

4. Camera System

- Serves as the main source of real-time visual inputs.
- Captures live feeds to support continuous monitoring, face recognition, and fraud detection.

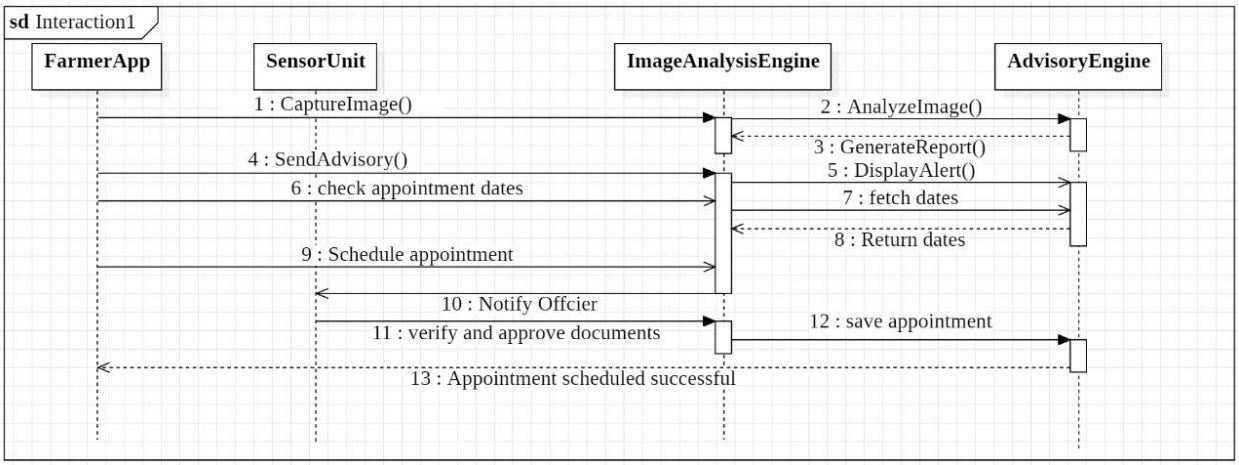
5. Notification System

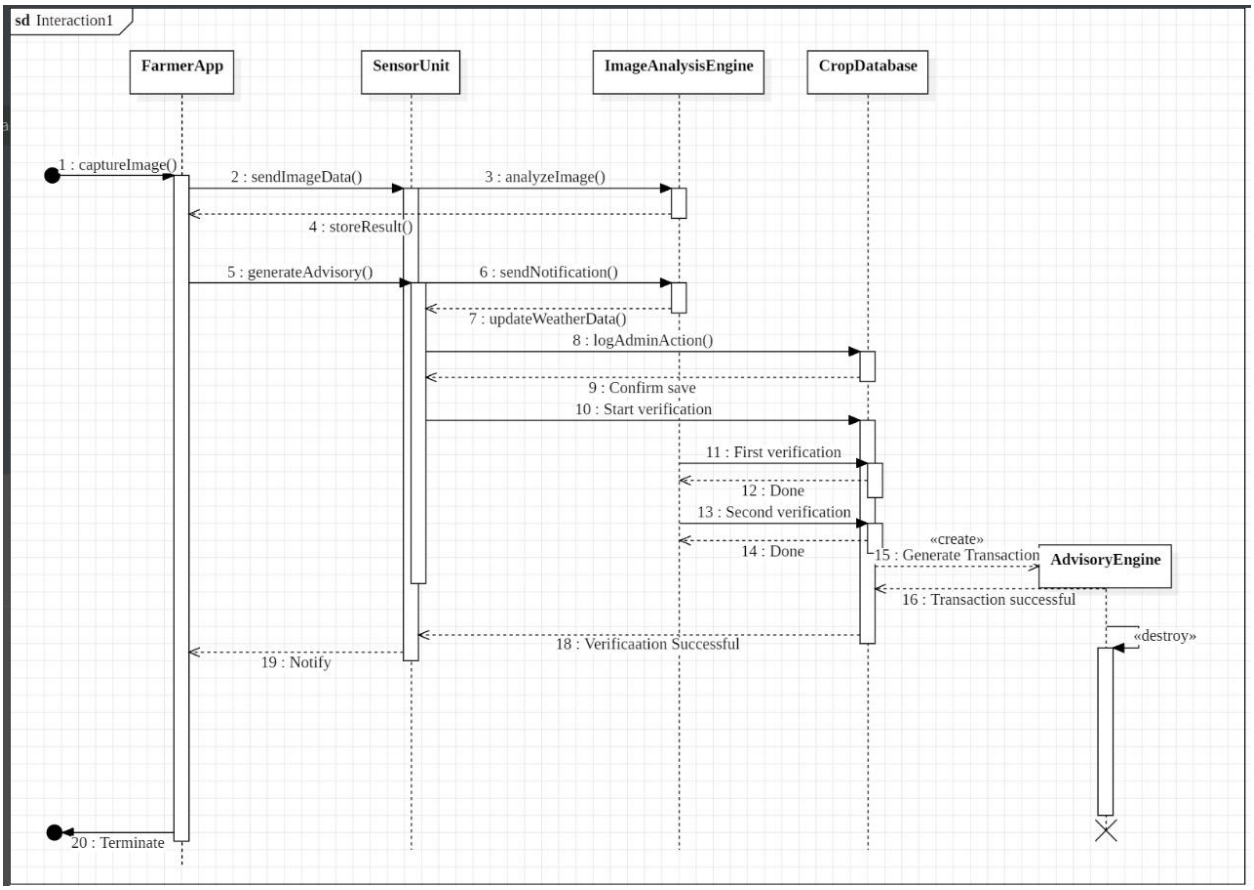
- Sends alerts, updates, escalations, and status messages to officers, administrators, and other involved parties.
- Ensures timely communication for critical responses.

6. Criminal Database

- Stores records of known offenders, past criminal activities, and suspect profiles.
- Supports identification processes and helps officers retrieve detailed insights during investigations.

SEQUENCE DIAGRAM





Relevance of Each Message (Sequence Diagram – Interaction1)

1. CaptureImage() – FarmerApp → SensorUnit

- Initiates the process by capturing an image from the farmer's device and sending it to the sensor unit for further analysis.

2. AnalyzeImage() – SensorUnit → ImageAnalysisEngine

- Sends the captured image to the analysis engine to perform processing, feature extraction, and validation.

3. GenerateReport() – ImageAnalysisEngine → AdvisoryEngine

- Produces a detailed advisory report based on the analyzed image and forwards it to the advisory component.

4. SendAdvisory() – SensorUnit → ImageAnalysisEngine

- Requests the analysis engine to process and prepare advisory information for the user.

5. DisplayAlert() – ImageAnalysisEngine → AdvisoryEngine

- Triggers an alert message when important findings or warnings need to be shown to the user.

6. Check Appointment Dates – SensorUnit → ImageAnalysisEngine

- Requests available appointment dates based on system data and farmer's requirements.

7. Fetch Dates – ImageAnalysisEngine → AdvisoryEngine

- The analysis engine queries the advisory engine to retrieve suitable appointment dates.

8. Return Dates – AdvisoryEngine → ImageAnalysisEngine

- Sends a list of available appointment dates back to the image analysis engine for further processing.

9. Schedule Appointment – FarmerApp → ImageAnalysisEngine

- The farmer confirms a preferred date, and the app sends a request to schedule the appointment.

10. Notify Officer – ImageAnalysisEngine → SensorUnit

- Informs the relevant officer or unit that an appointment scheduling request has been initiated.

11. Verify and Approve Documents – SensorUnit → ImageAnalysisEngine

- Confirms the verification of the farmer's documents and approves them for scheduling.

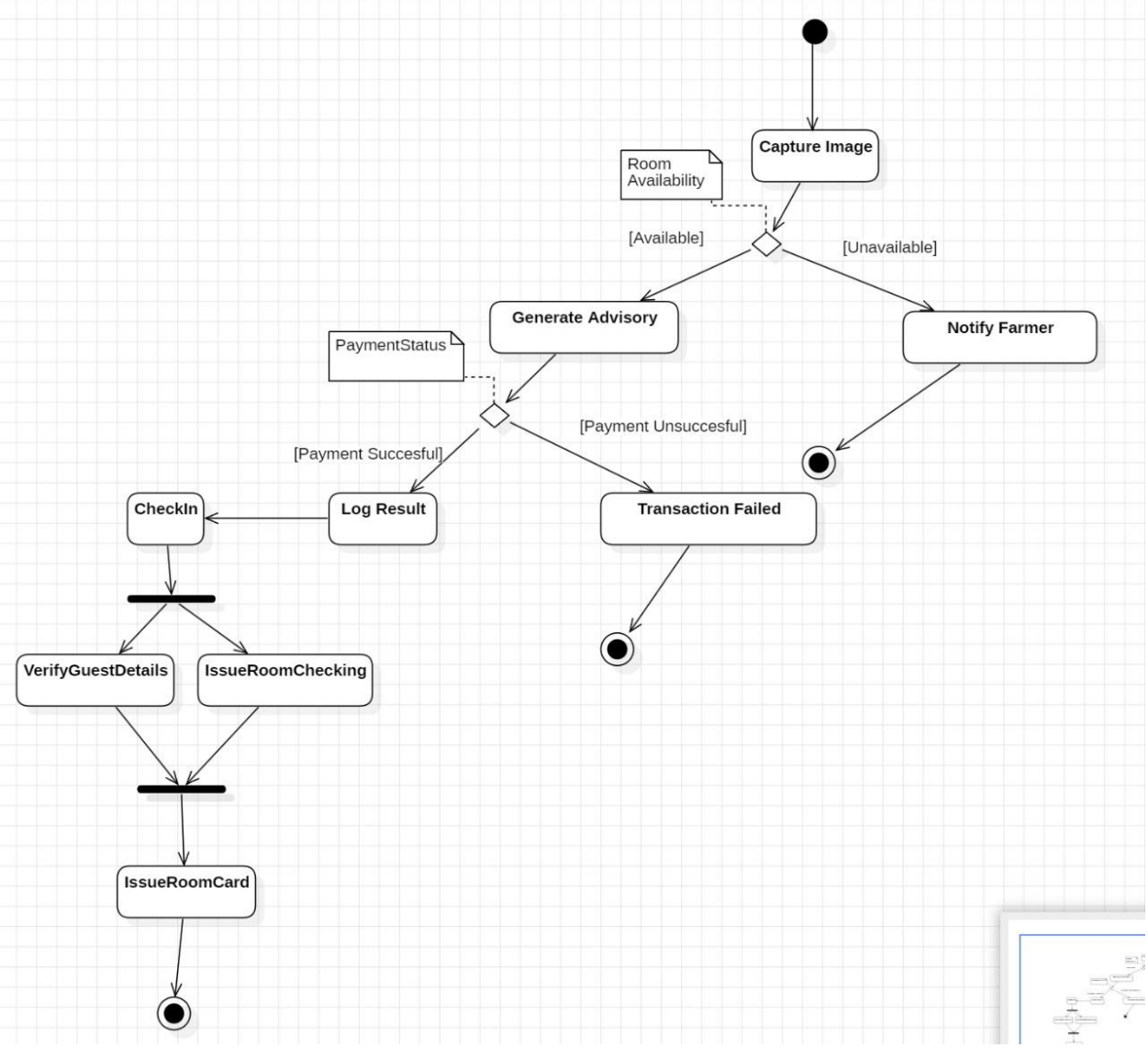
12. Save Appointment – ImageAnalysisEngine → AdvisoryEngine

- Officially stores and confirms the appointment details in the advisory engine's database.

13. Appointment Scheduled Successful – ImageAnalysisEngine → FarmerApp

- Sends a final confirmation message indicating that the appointment was scheduled successfully.

ACTIVITY DIAGRAM:



This activity diagram provides a comprehensive representation of how the system processes room availability, payment handling, guest verification, and final room card issuance. It illustrates the flow of control across multiple system components, demonstrates decision-making points, and shows how parallel actions merge into final outcomes. The diagram displays the beginning of the process, how the system evaluates conditions (such as availability and payment success), and how tasks proceed toward successful check-in or transaction failure.

Swimlanes and Their Roles

1. Image Capture / Room Availability Swimlane

Responsible for capturing initial image input and determining room availability. This lane handles:

- Starting system activity
- Capturing images
- Checking and logging room availability
  It delivers essential input used by downstream modules.

2. Advisory Generation / Payment Evaluation Swimlane

This lane processes the captured information to:

- Generate advisory results
- Evaluate payment status
- Log relevant outcomes
  It ensures that payment verification and advisory creation are validated before forwarding to check-in or failure paths.

3. Notification / Failure Handling Swimlane

In charge of:

- Notifying relevant personnel when rooms are unavailable
- Marking transactions as failed when payment is unsuccessful
- Logging and ending the process when no further steps can be taken
  This lane ensures the system communicates failures effectively.

4. Check-In Management Swimlane

Once payment is confirmed, this lane handles guest intake procedures:

- Check-in initiation
- Verification of guest details
- Room readiness confirmation
- Issuance of room card
  This part ensures smooth processing of guests who have met all requirements.

Splitting and Merging of Control

1. Splitting

The control flow splits after:

- **Image Capture:**
  The process divides into two paths:
  1. One path checks **room availability**.
  2. The other logs or supports parallel activities in advisory generation.
- **Payment Status Decision:**
  After generating the advisory, control splits to determine:
  - Payment successful → proceed to check-in
  - Payment unsuccessful → proceed to transaction failure

2. Merging

- After completing guest verification and room preparation tasks, control merges before issuing the room card.
- In the event of failed payment or unavailable room, flow merges into termination points marking the end of the process path.

Explanation of the Diagram Components

1. Actions Under Image Capture / Availability Checking

- **Capture Image:**
  The system begins by capturing an image for analysis.
- **Room Availability Check:**
  Determines if a room is available based on detected inputs.
- **Log Activity:**
  The system logs captured data or availability status for record keeping.
- **Notify Farmer:**
  If a room is unavailable, a notification is sent to the responsible person.

2. Actions Under Advisory & Payment Processing

- **Generate Advisory:**
  Creates guidance or instructions based on captured data.

- **Payment Status Evaluation:**
  Checks whether the payment required for check-in is successful.
- **Payment Successful:**
  If payment is verified, the system proceeds to check-in.
- **Payment Unsuccessful:**
  If payment fails, the system marks the transaction as unsuccessful.
- **Log Result:**
  Records the payment and advisory results for audit purposes.

3. Transaction Failure Handling

- **Transaction Failed:**
  This activity is triggered when payment is unsuccessful.
  The flow ends here with a final state.

4. Check-In Management Activities

- **Check-In:**
  Initiates the guest onboarding process once all conditions are met.
- **Verify Guest Details:**
  Confirms the identity, credentials, and booking information of the guest.
- **Issue Room Checking:**
  Ensures the assigned room is ready and validated for occupancy.
- **Issue Room Card:**
  Provides the guest with their room access card after all verification steps.
- **End State:**
  Marks completion of the successful check-in process.

Chapter 6: UI Design with Screenshots

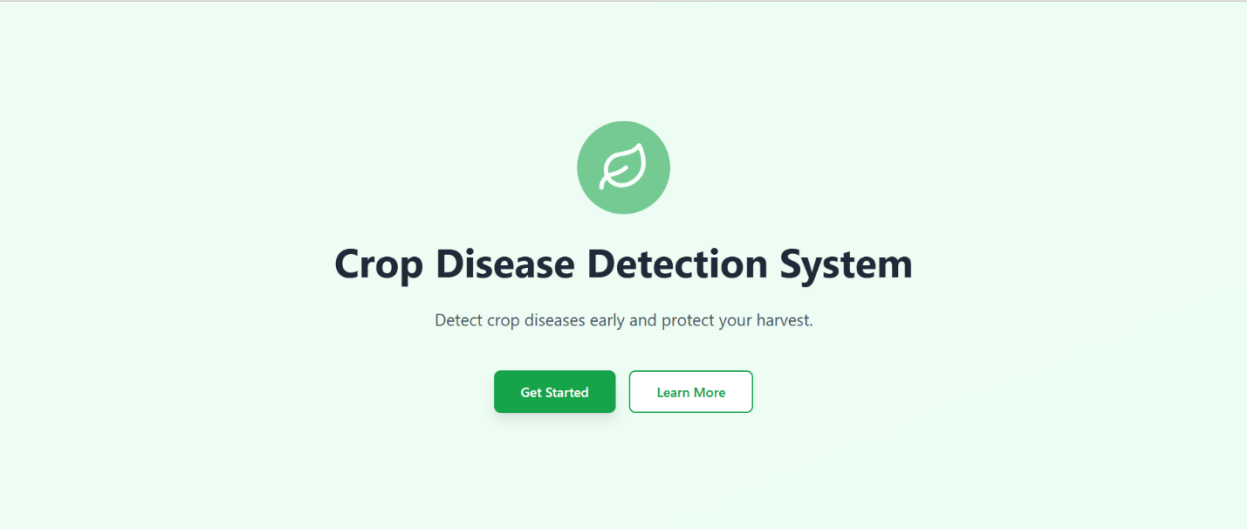


Figure 1 :Welcome Page

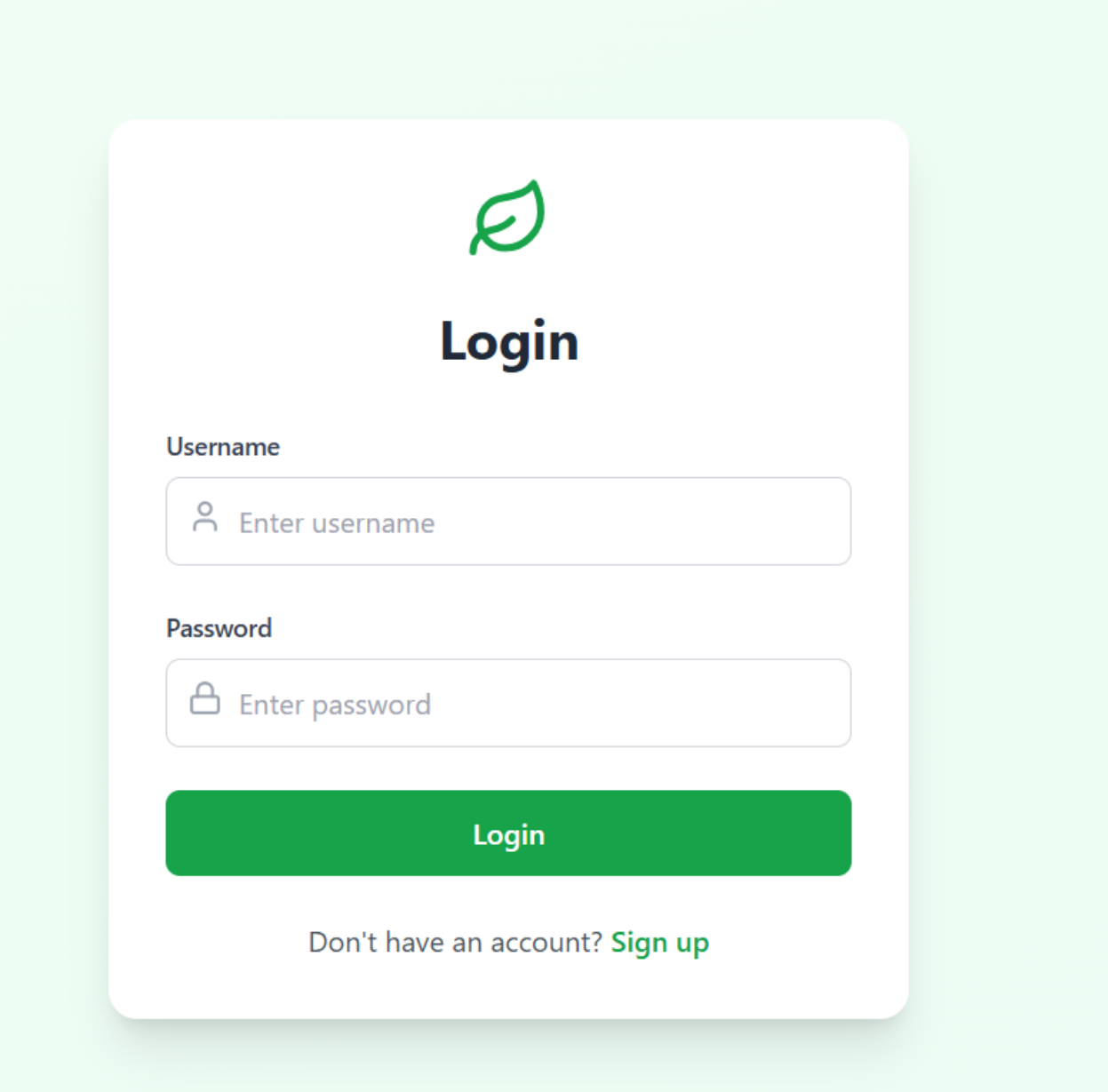


Figure 2 : Login

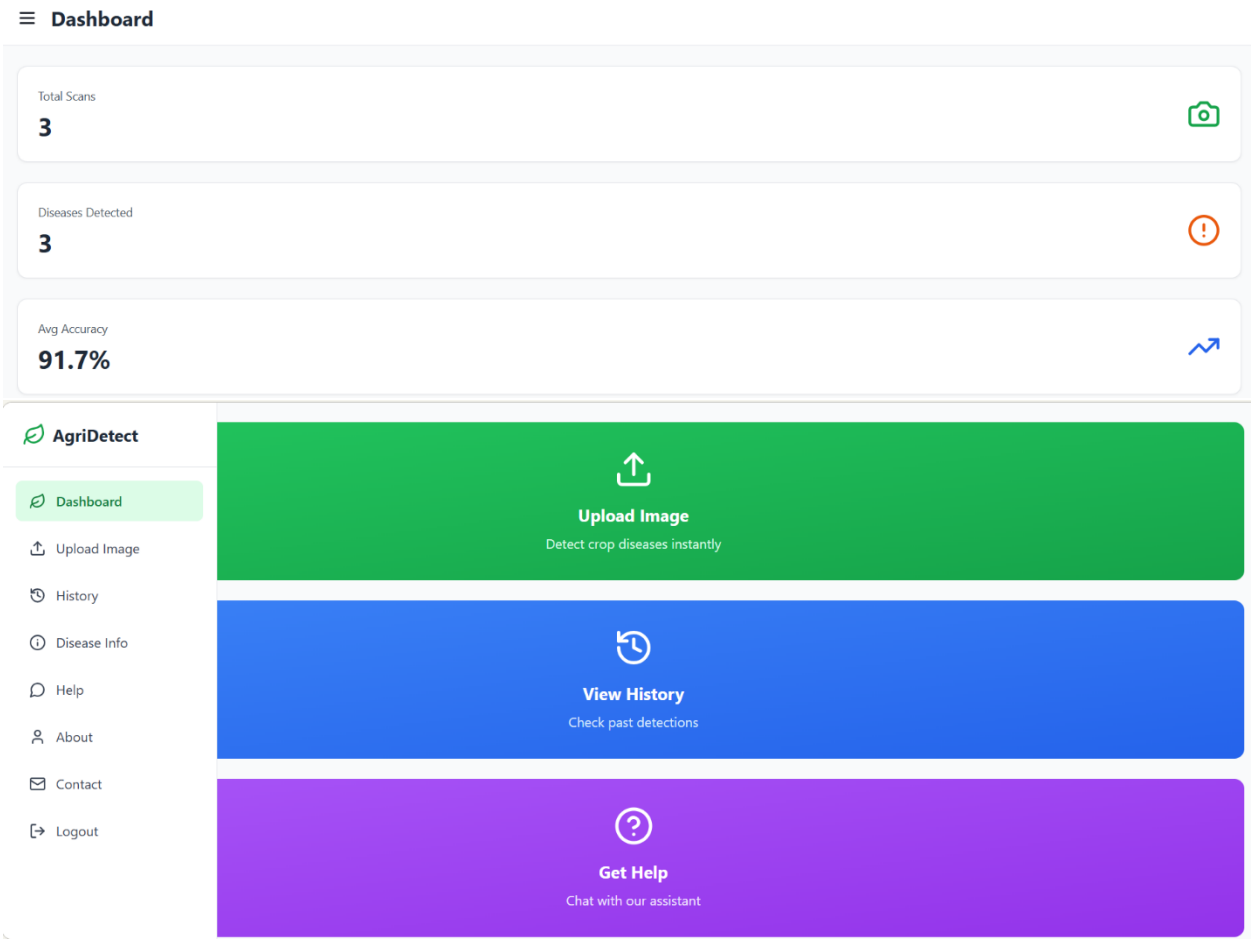


Figure 3: Dashboard and home page

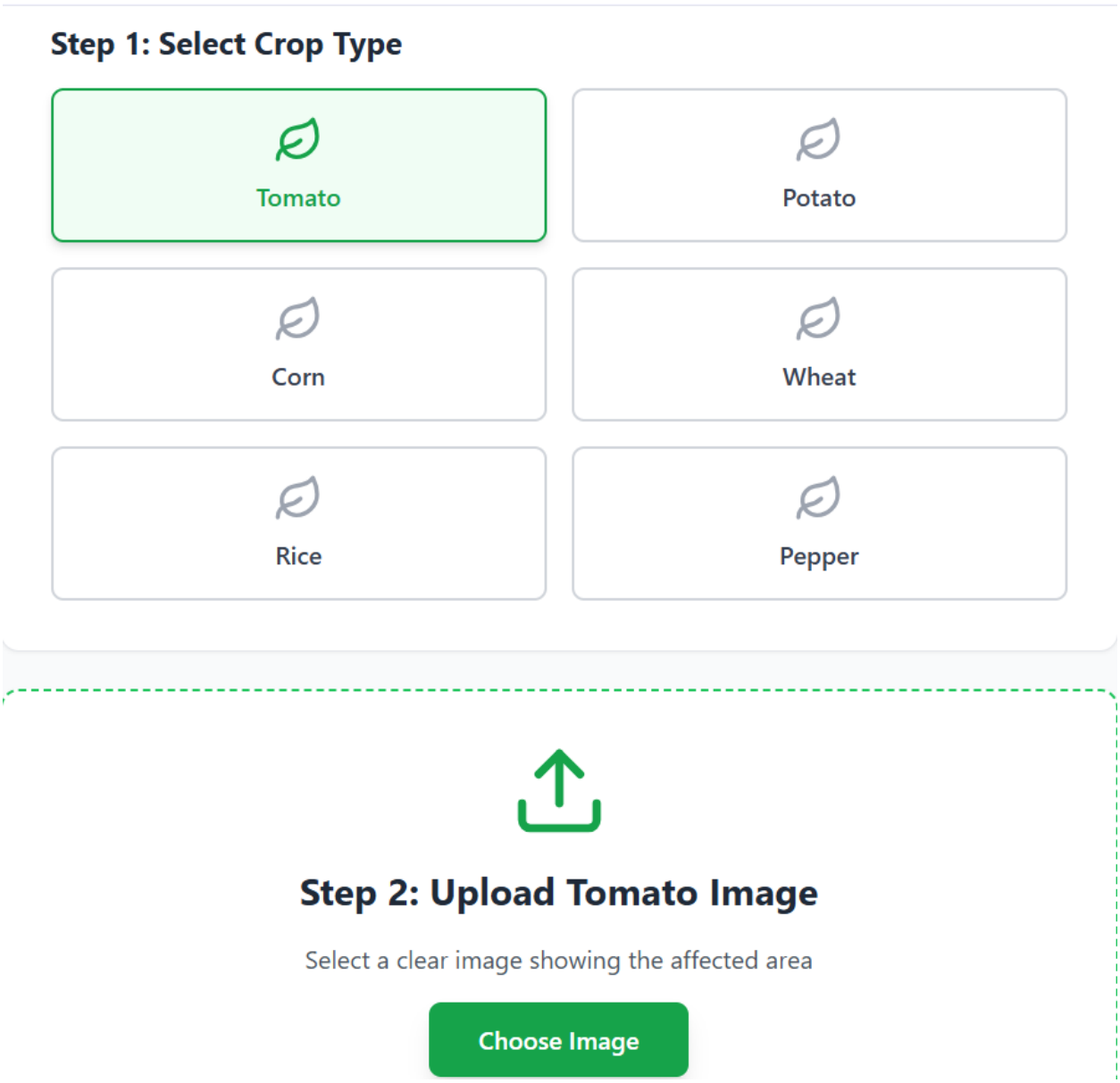


Figure 4 : Selecting and uploading crop images

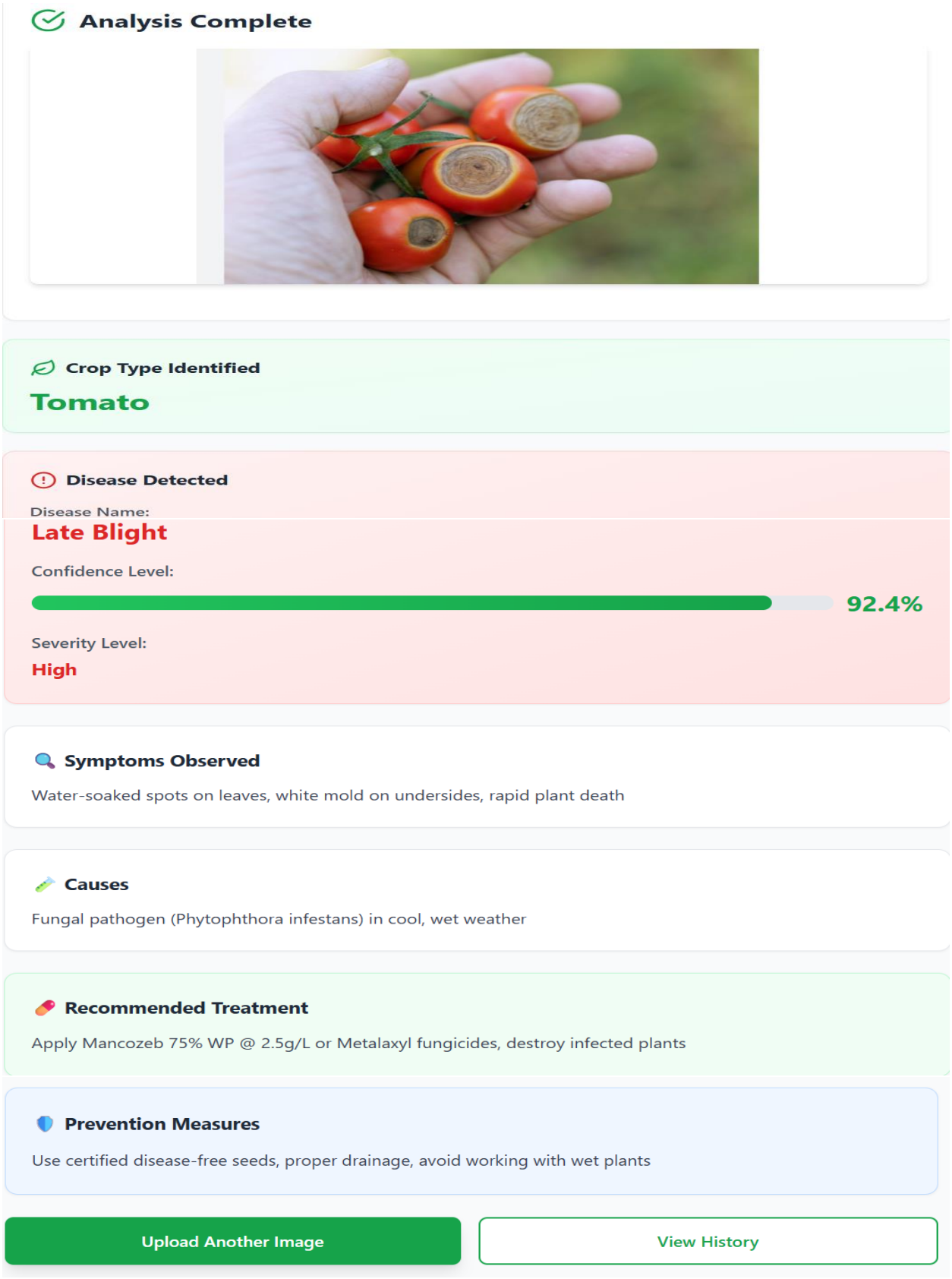


Figure 5 : Analysing disease Page

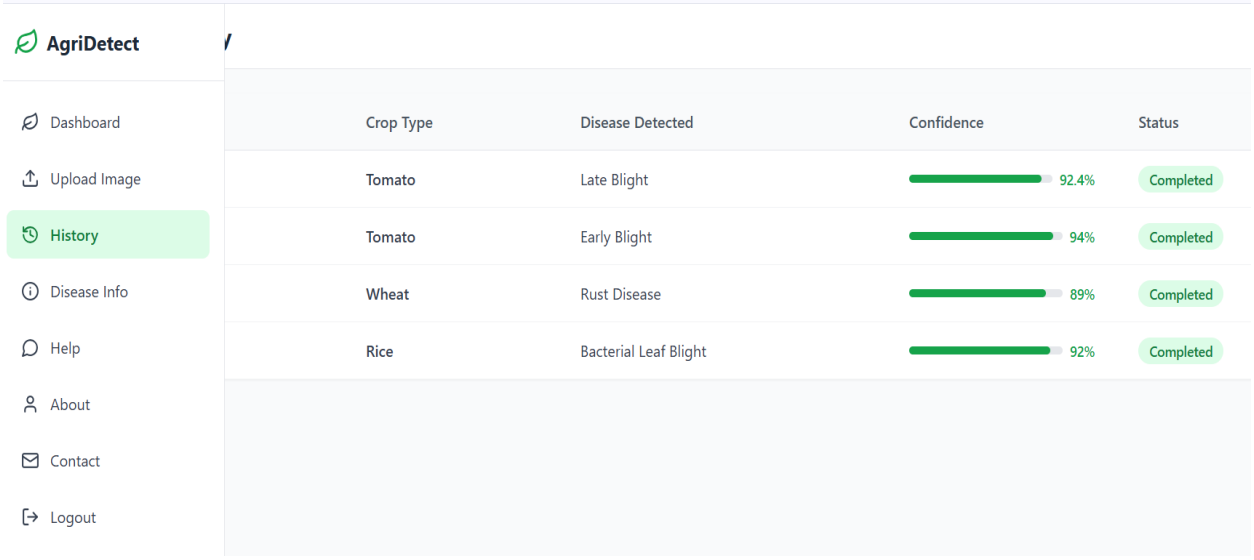


Figure 6 : History Page

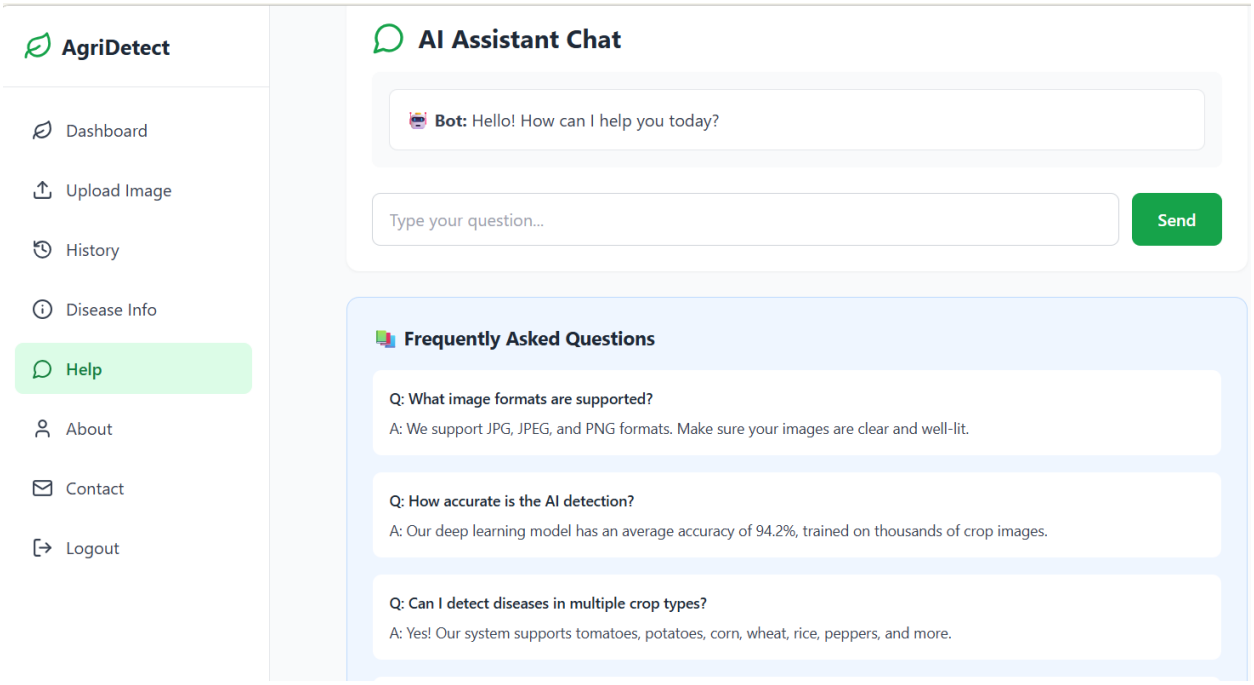


Figure 7 : Help Page

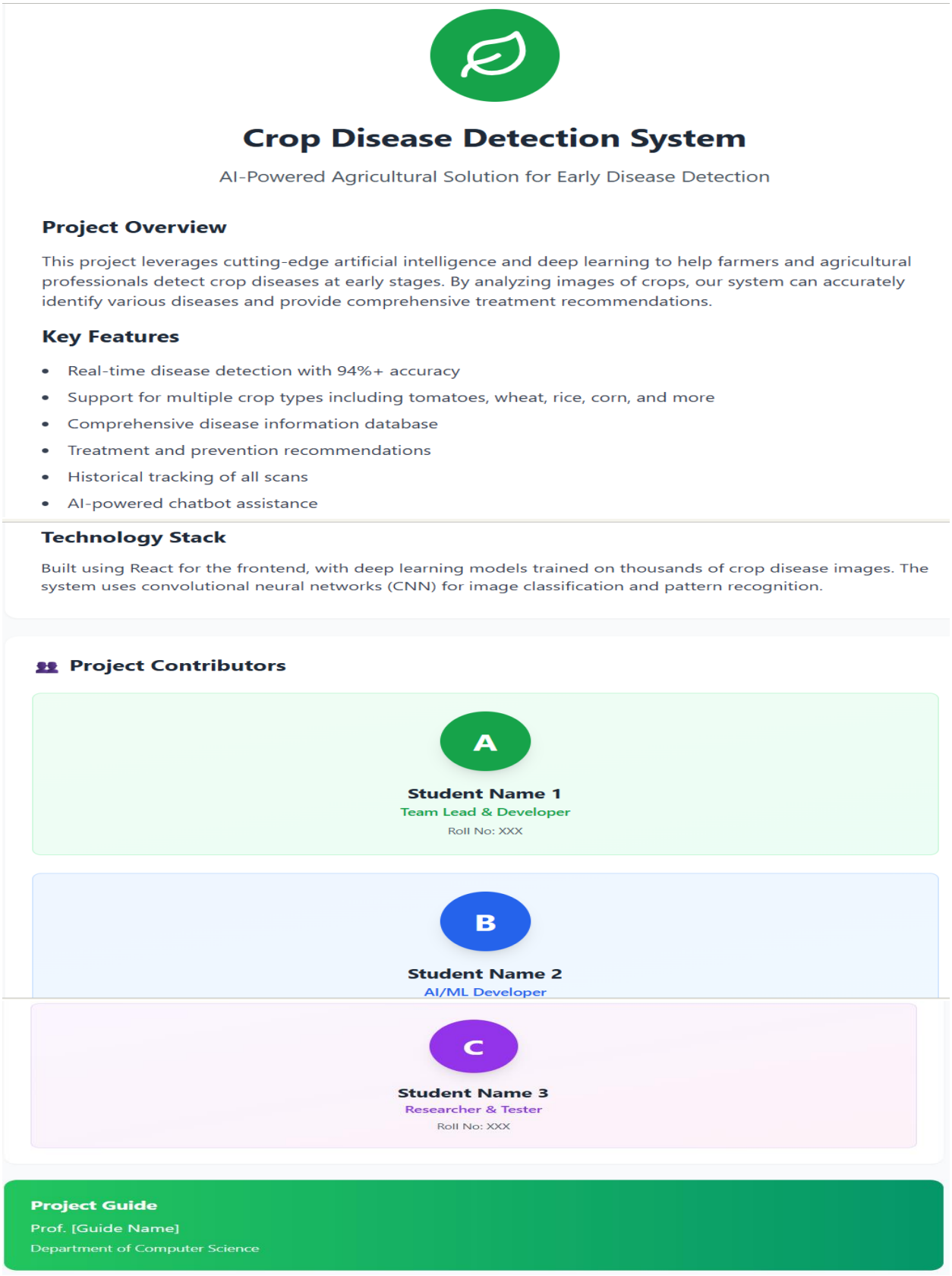


Figure 8 : About

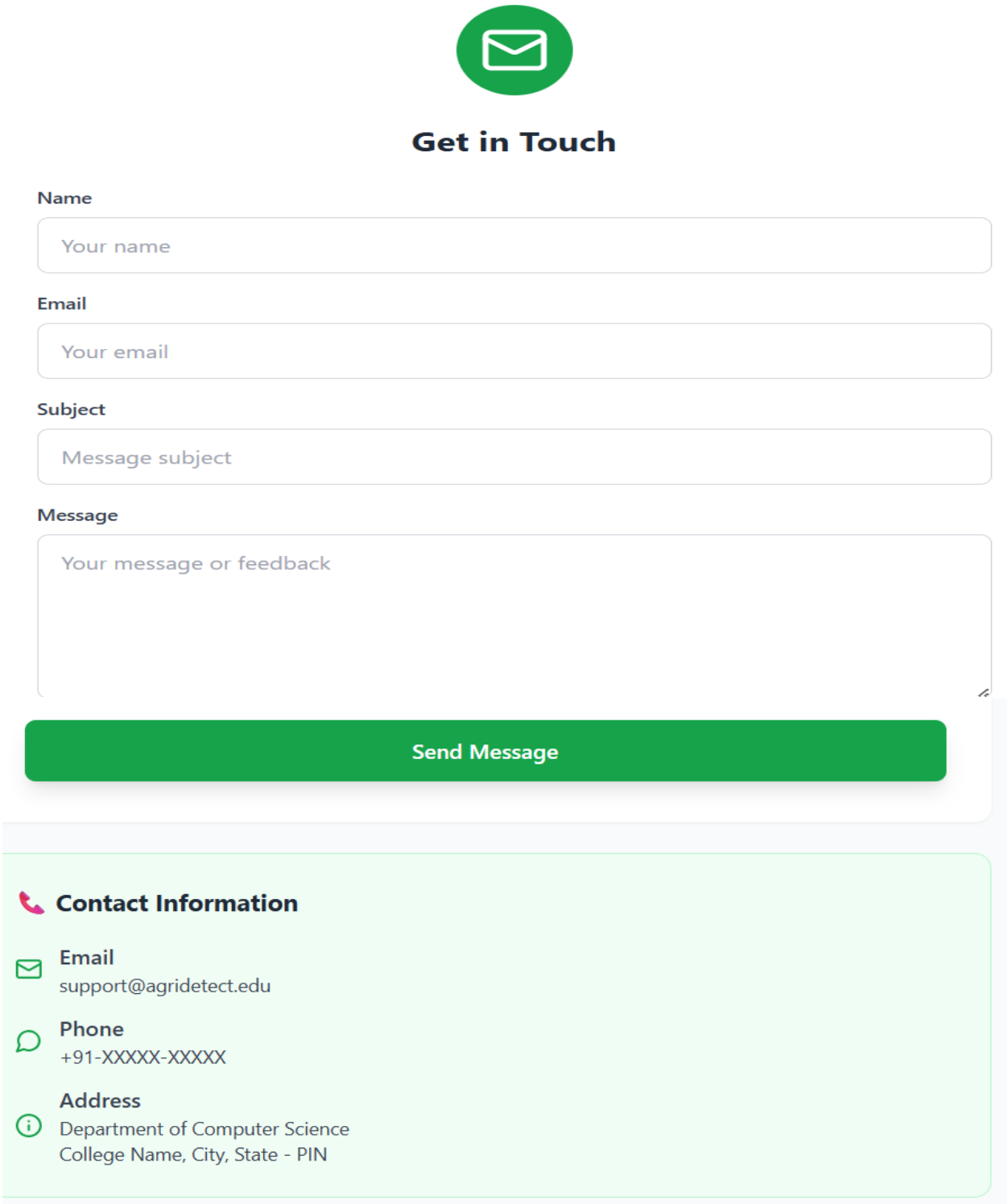


Figure 9: Contact

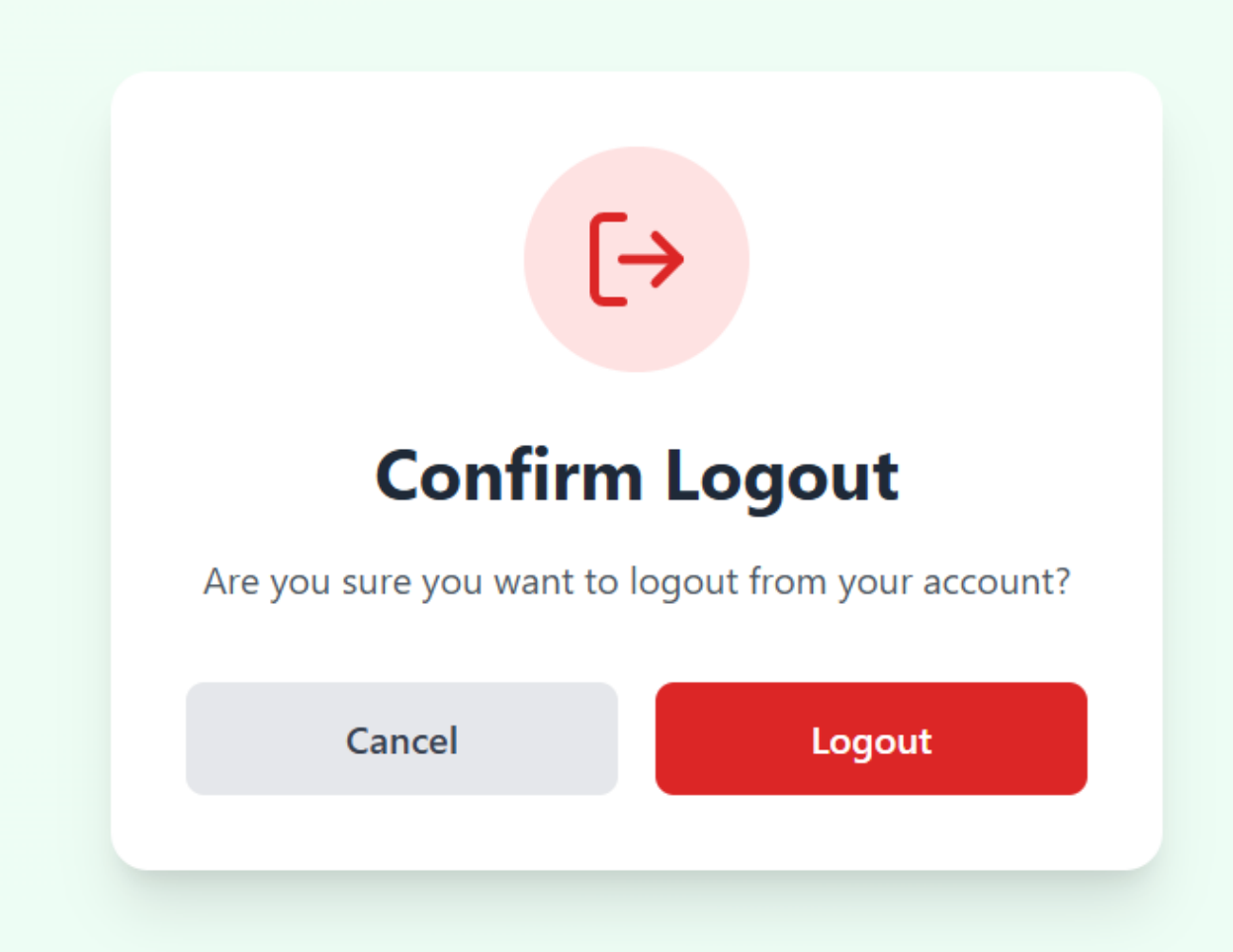


Figure 10 : Logout Page

Conclusion

the **Crop Disease Detection System** stands as a powerful and practical tool to support modern agriculture by enabling early and accurate detection of plant diseases. Leveraging image-based machine learning, the system empowers farmers to upload crop images, obtain reliable diagnoses, and receive tailored advisory recommendations for disease management. This automation helps reduce manual effort, minimize crop losses, and improve resource efficiency.

The modular architecture — consisting of a mobile/web application, a deep-learning-based diagnosis engine, sensor integration, and a feedback loop — ensures scalability and continuous improvement. By incorporating human agronomist input and continuous model updating, the system is designed to adapt to evolving disease patterns and diverse crop species.
Moreover, the system's data management capabilities, including secure storage, archiving, and model retraining, lay the foundation for robust long-term operations. Through this design, the solution not only improves immediate crop health outcomes but also builds valuable data resources for future research and optimization.

Ultimately, this project demonstrates how precision agriculture technologies can bridge the gap between farmers and expert agronomic knowledge, contributing to higher yields, reduced economic losses, and sustainable farming practices. With continued development — such as expanding the disease dataset, integrating real-time field monitoring, and improving model explainability — the system has the potential to become a cornerstone of smart, data-driven agriculture.