**Batch: A-3 Roll No.: 16010122104**

**Experiment No 3**

**Group No:**

|  |
| --- |
| **Title: Prepare design document and Plan of project** |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Objective:** Chapter No.3 of Mini Project Report will include detailed design document and plan of implementation of the project

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
|  | **At the end of successful completion of the course the student will be able to** |
| CO2 | Identify various hardware and software requirements for problem solution |
| CO5 | Prepare a technical report based on the Mini project. |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

**1.**

**2.**

**3.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**The students are expected to prepare chapter no 3 in the format given below**

**Chapter 3**

**Design Document and Project plan**

*A* ***design document*** *is crucial in a software project because it serves as a blueprint that outlines the architecture, components, data flow, and technical specifications of the system before implementation* ***Clear Vision & Planning*** *will improve collaboration with in the team members. The*

A **design document** is crucial in a software project because it serves as a blueprint that outlines the architecture, components, data flow, and technical specifications of the system before implementation.

Typically a design document will have following sections:

1. **Introduction**

Purpose of the Document

The purpose of this document is to provide a comprehensive design blueprint and implementation plan for the AI-driven crop disease prediction and management system. It outlines the system's architecture, components, data flow, technical specifications, and project execution strategy. This document serves as a guide for developers, researchers, and stakeholders involved in building, deploying, and maintaining the system.

Expected Audience

The expected audience for this document includes:

* Developers: To understand the technical details and implementation steps.
* Researchers: To explore the system's design and methodology for potential improvements or academic studies.
* Project Stakeholders: To review the project's scope, deliverables, and implementation plan.
* Farmers and Agricultural Experts: Indirectly, to understand how the system will address their needs through its functionalities.

Scope of the Project (Brief)

The project focuses on developing an AI-driven system for early detection and management of crop diseases, with an emphasis on cassava diseases such as Cassava Mosaic Disease (CMD), Cassava Brown Streak Disease (CBSD), Cassava Bacterial Blight (CBB), and Cassava Green Mite (CGM). The system leverages deep learning models like RexNet-150 for image-based disease classification and integrates a user-friendly web interface to provide real-time predictions and actionable recommendations.

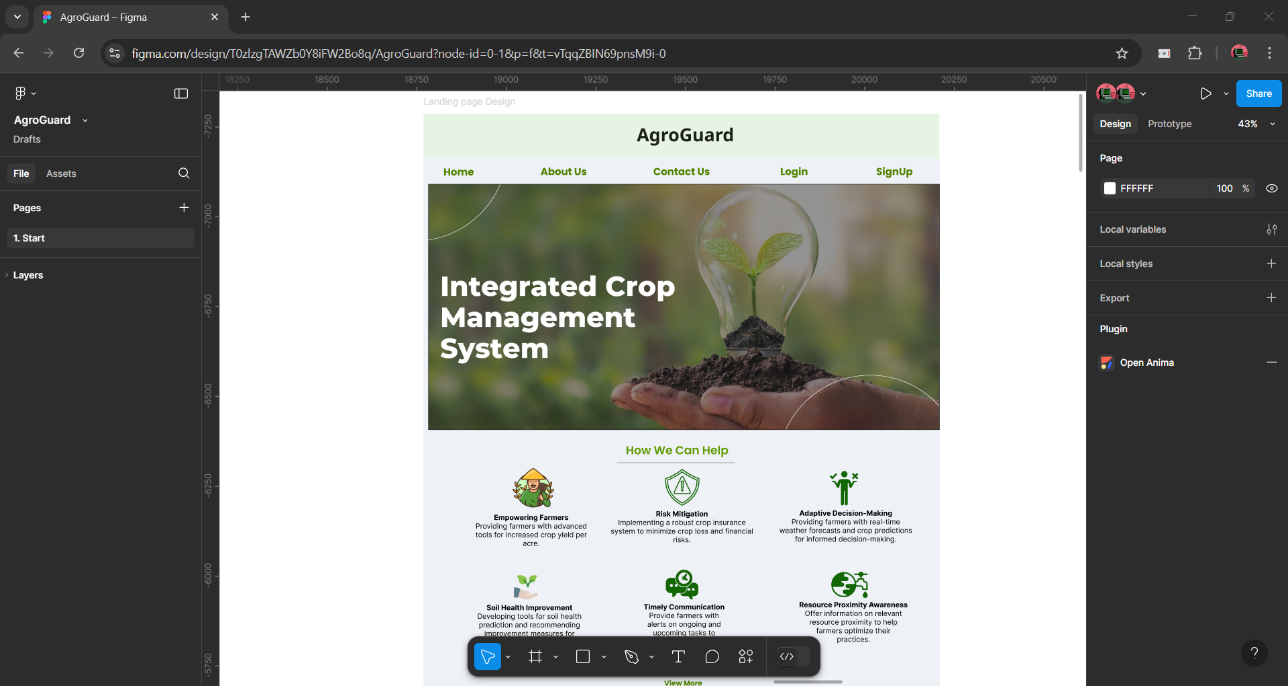
Definitions, Acronyms, and Abbreviations

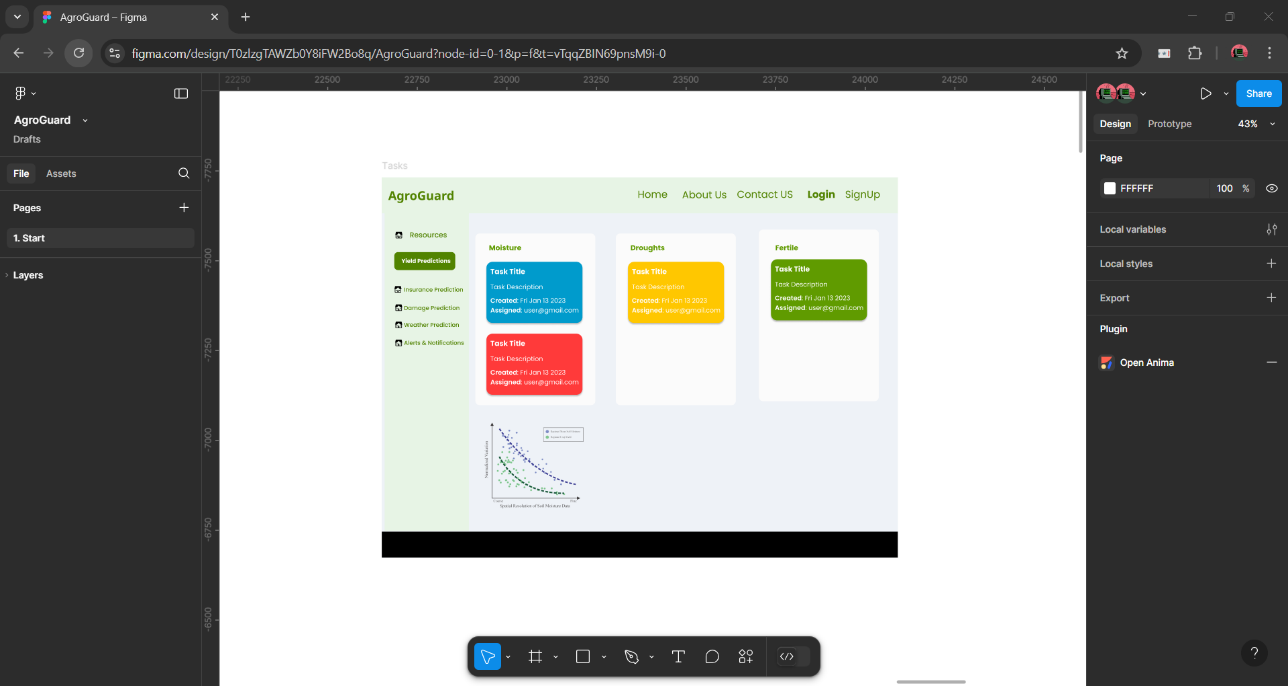
* AI: Artificial Intelligence
* CNN: Convolutional Neural Network
* CMD: Cassava Mosaic Disease
* CBSD: Cassava Brown Streak Disease
* CBB: Cassava Bacterial Blight
* CGM: Cassava Green Mite
* SRS: Software Requirements Specification
* UX/UI: User Experience/User Interface

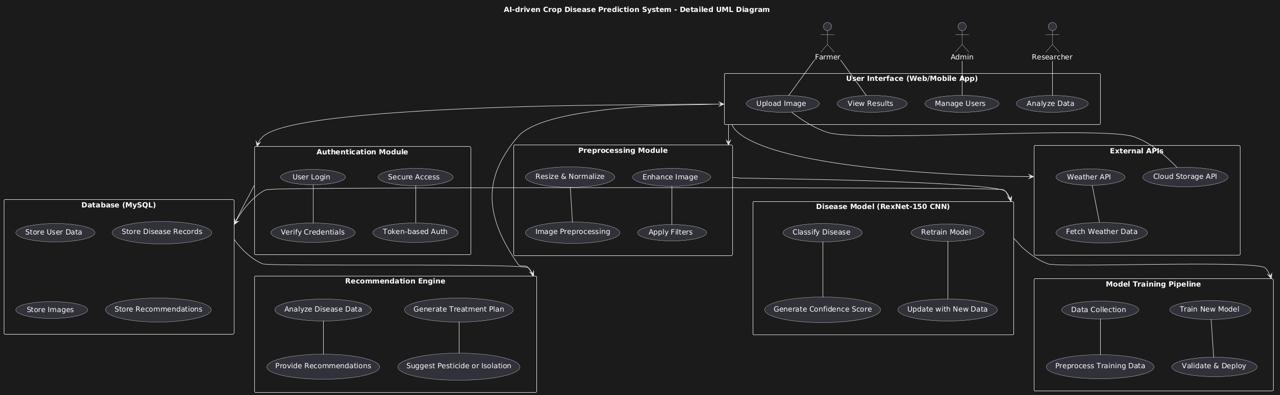
1. **System Over View**

2.1 System Architecture

The AI-driven crop disease prediction and management system utilizes a modular architecture designed for scalability and efficiency.







Description of Component Interactions

* User Interface: Farmers interact with the system via a web or mobile application to upload leaf images for disease diagnosis.
* Preprocessing Module: Uploaded images are resized, normalized, and enhanced to ensure compatibility with the RexNet-150 CNN model.
* Disease Model: The RexNet-150 deep learning model processes the preprocessed images to classify diseases into categories such as Cassava Mosaic Disease (CMD), Cassava Brown Streak Disease (CBSD), Cassava Bacterial Blight (CBB), Cassava Green Mite (CGM), and Healthy.
* Database: The MySQL database stores user profiles, disease records, uploaded images, and tailored recommendations.
* Recommendation Engine: Based on the disease classification, the engine provides actionable insights, such as pesticide usage or isolation measures, to mitigate crop loss.

The system is designed with the following key principles:

1. **Scalability**:
   * Supports multiple crops beyond cassava in future iterations.
   * Handles increasing user traffic through efficient database management and cloud-based deployment.
2. **Security**:
   * Ensures data privacy through encrypted communication between components.
   * Implements authentication mechanisms for user accounts.
3. **Performance**:
   * Achieves high accuracy (>92%) in disease detection using the RexNet-150 model.
   * Provides real-time predictions (<2 seconds per image).
4. **Maintainability**:
   * Modular architecture allows easy updates to individual components without disrupting the system.
   * Regular model retraining ensures adaptability to new diseases or environmental conditions.
5. **User Accessibility**:
   * Designed for non-expert users with an intuitive interface and multilingual support.
   * Accessible via both web and mobile platforms for broader reach.
6. **Environmental Sustainability**:
   * Promotes organic treatment recommendations to minimize pesticide usage and environmental impact.

**3. Detailed Design**

3.1 Module Description

The system is divided into the following modules, each with specific responsibilities and interactions:

1. User Interface Module:
   * Responsibilities: Provides a web/mobile interface for farmers to upload leaf images and view results.
   * Interactions: Sends images to the preprocessing module and displays disease classification results and recommendations.
2. Preprocessing Module:
   * Responsibilities: Processes uploaded images by resizing, normalizing, and enhancing them for compatibility with the disease model.
   * Interactions: Receives images from the user interface and sends preprocessed images to the disease model.
3. Disease Model Module:
   * Responsibilities: Uses RexNet-150 CNN to classify diseases based on input images into categories such as CMD, CBSD, CBB, CGM, or Healthy.
   * Interactions: Takes preprocessed images as input and outputs disease labels and confidence scores to the database.
4. Database Module:
   * Responsibilities: Stores user data, uploaded images, disease records, and recommendations.
   * Interactions: Retrieves and stores data for the recommendation engine and user interface.
5. Recommendation Engine Module:
   * Responsibilities: Analyzes disease classification results to provide actionable insights (e.g., pesticide use or isolation measures).
   * Interactions: Fetches disease data from the database and sends recommendations to the user interface.

3.2 Data Flow & Components

The system's data flow is illustrated below using a UML sequence diagram:

1. Farmer uploads an image via the web/mobile app.
2. The preprocessing module processes the image (resize, normalize).
3. The processed image is sent to the RexNet-150 CNN model for classification.
4. The model outputs a disease label and confidence score.
5. The database stores this output along with user data.
6. The recommendation engine retrieves disease information from the database and generates actionable insights.
7. The user interface displays the results and recommendations.

3.3 Database Design

The database schema includes the following tables:

| Table Name | Description | Key Fields |
| --- | --- | --- |
| Users | Stores user information | UserID, Name, Email, Password |
| Images | Stores uploaded leaf images | ImageID, UserID, ImagePath, Date |
| Diseases | Stores disease categories and descriptions | DiseaseID, Name, Description |
| Results | Links image classifications with diseases | ResultID, ImageID, DiseaseID, ConfidenceScore |
| Recommendations | Stores treatment or management suggestions | RecommendationID, DiseaseID, Action |

* Relationships:
  + One-to-Many between Users and Images.
  + One-to-Many between Diseases and Results.
  + One-to-One between Diseases and Recommendations.
* Indexing Strategies:
  + Indexing on primary keys (UserID, ImageID, etc.).
  + Foreign key indexing for faster joins between tables.

3.4 User Interface Design

The UI/UX design prioritizes simplicity and accessibility for non-expert users:

1. Design Elements:
   * Clear upload button for leaf images.
   * Minimalistic layout displaying disease diagnosis results with confidence scores.
   * Recommendation section providing actionable insights in simple language.
2. Wireframes:
   * Home Screen: Includes an image upload option and navigation menu.
   * Results Screen: Displays disease classification, confidence score, and recommendations.
   * History Screen: Allows users to view past diagnoses.
3. Navigation Flow:
   * Home → Upload Image → View Results → Access Recommendations → Return to Home or History.

3.5 External Interfaces

The system integrates with several external components:

1. APIs:
   * Weather API: To incorporate weather conditions into disease prediction models in future iterations.
   * Cloud Storage API (e.g., AWS S3): For storing uploaded images securely.
2. Third-party Services:
   * TensorFlow/Keras libraries for implementing RexNet-150 CNN.
   * MySQL for database management.
3. Hardware Components:
   * Mobile devices or cameras for capturing leaf images.
   * Cloud-based servers for hosting the web application and running inference models.

**4. Project and Implementation Plan**

4.1 Deliverables

The following deliverables will be provided as part of the project:

1. Modules and Code:
   * Fully functional modules for:
     + Image Preprocessing
     + Disease Classification
     + Recommendation Engine
   * Flask-based web application source code.
2. User Documentation:
   * A comprehensive user manual explaining how to use the system, upload images, and interpret results.
3. Installation (Deployment) Document:
   * Step-by-step instructions for deploying the system on a local server or cloud platform.
   * Details on setting up required dependencies.
4. Testing Reports:
   * Detailed reports on unit testing, integration testing, and system testing results.
5. Presentation Slides:
   * A summary of the project for presenting to stakeholders or evaluators.

4.2 Team Roles and Responsibilities and Delivery Schedule

| Name of the Task | Developer | Tester | Approver |
| --- | --- | --- | --- |
| Data Collection and Preprocessing | Avnish | Raahi | Kashish |
| Model Training | Kashish | Avnish | Raahi |
| Web Application Development (Frontend) | Raahi | Kashish | Avnish |
| Backend Integration | Avnish | Raahi | Kashish |
| Database Design and Setup | Kashish | Avnish | Raahi |
| Recommendation Engine Implementation | Raahi | Kashish | Avnish |
| System Testing and Debugging | Avnish | Raahi | Kashish |
| Deployment and Final Documentation | Kashish | Avnish | Raahi |

4.3 Risk Management Plan

The following risks have been identified along with mitigation strategies:

1. Dataset Imbalance:
   * *Risk*: Limited data for certain cassava diseases may lead to biased predictions.
   * *Solution*: Apply data augmentation techniques (e.g., flipping, rotation) to balance the dataset. Use synthetic data generation if necessary.
2. Model Overfitting:
   * *Risk*: The model may overfit due to limited training data.
   * *Solution*: Use regularization techniques such as dropout layers and early stopping during training.
3. Deployment Challenges:
   * *Risk*: Issues may arise while deploying the system on a server or cloud platform.
   * *Solution*: Provide detailed deployment documentation and test deployment on multiple platforms beforehand.
4. Low-Quality Images from Users:
   * *Risk*: Poor-quality images uploaded by users may reduce classification accuracy.
   * *Solution*: Include preprocessing steps like noise reduction and normalization to handle low-quality images.
5. System Downtime or Failures:
   * *Risk*: The system might experience downtime due to server issues or bugs.
   * *Solution*: Implement robust error handling in the backend and maintain a backup server for critical operations.
6. User Adoption Challenges:
   * *Risk*: Farmers may find it difficult to use the system due to technical barriers.
   * *Solution*: Design an intuitive user interface with multilingual support and provide a user guide.
7. Data Privacy Concerns:
   * *Risk*: Users may hesitate to upload images due to privacy concerns.
   * *Solution*: Use secure communication protocols (e.g., HTTPS) and ensure that uploaded images are anonymized.

**5. Testing & Deployment Plan**

5.1 Testing Strategy

1. Unit Testing:
   * Tests individual modules (image preprocessing, disease classification, recommendation engine) to ensure they perform as expected.
   * Focuses on input validation and error handling.
2. Integration Testing:
   * Verifies seamless interaction between modules (e.g., preprocessing → disease model → database).
   * Ensures data flow and communication between components are functioning correctly.
3. System Testing:
   * Tests the entire system end-to-end, including user interface, backend logic, and database interactions.
   * Validates overall functionality, accuracy of disease prediction, and recommendation generation.
4. User Acceptance Testing:
   * Conducted with farmers and stakeholders to assess usability and effectiveness.
   * Ensures the system meets user requirements and expectations.

5.2 Deployment Plan

1. Deployment Methods:
   * Deploy the system on a cloud server (e.g., AWS or Google Cloud) for scalability.
   * Use Docker containers to package the application for consistent deployment across environments.
2. Environment Setup:
   * Install necessary dependencies (Python, Flask, TensorFlow, MySQL).
   * Configure cloud storage for image uploads and backups.
3. Rollback Strategies:
   * Maintain version control using Git to revert to previous stable versions in case of deployment failure.
   * Implement a backup server to minimize downtime during rollback operations.