# GreenEye: Intelligent Plant Species and Health Recognition System

## Project Objective

GreenEye is a deep learning-based system that can **identify plant species** and **detect plant diseases** from leaf images. Users, such as farmers, gardeners, and botanists, will upload images of plant leaves and receive instant feedback regarding both the plant species and its health status.

The solution targets both **species classification** and **disease detection**, offering personalized care suggestions based on the predictions.

---

## Core Features

- **Image Upload Interface:** Users can either capture a new image or upload an existing leaf photo.
- **Species and Health Prediction:** The system predicts the species name (e.g., "Maple", "Apple tree") and the health status (e.g., "Healthy", "Leaf Blight", "Powdery Mildew").
- **Information Page:** After prediction, the system displays brief care instructions or disease management tips.
- **History Log (Optional):** Users can track and review their previous scans.
- **Multi-Leaf Input (Enhanced Accuracy):** Users can upload multiple images of the same plant (e.g., different leaves) to improve the accuracy and reliability of predictions.
- **Admin Dashboard:** An admin panel allows monitoring of system performance, managing retraining needs, and tracking model drift.

---

## Dataset Usage Plan

The following datasets will be utilized for model training and evaluation:

| Dataset | Purpose | Justification |
|---|---|---|
| **Pl@ntNet-300K** | Plant species classification | Large-scale, real-world images covering over 1,000 species |
| **PlantVillage** | Disease detection | Comprehensive healthy/diseased samples for fine-tuning models |
| **Plant Leaves for Image Classification (Kaggle)** | Prototyping | High-resolution images suitable for quick experiments and validation |
| **plant_leaves (TensorFlow** | Prototyping | Easy integration into TensorFlow pipelines |

| Dataset | Purpose | Justification |
|---|---|---|
| **Datasets)** | | for model validation |
| **Strategy:** | | |

Initial prototyping will be conducted using the smaller Kaggle and TensorFlow datasets. Once the pipeline is validated, full-scale training will be performed using Pl@ntNet-300K and PlantVillage datasets.

---

# System Architecture Overview

The GreenEye system will be composed of four major architectural layers: Frontend Interface, Backend API Layer, Machine Learning Model Server, and Database Layer. These layers will communicate over secure APIs to deliver a seamless experience to users.

## 1. Frontend Interface

**Technology:** React.js (for web) or React Native (for mobile)

**Main Responsibilities:**

- Allow users to **upload one or multiple images** (multi-leaf input support).
- Provide real-time **feedback to the user** (loading indicators, success/failure notifications).
- Display **prediction results** including:
    - Plant species name
    - Health status (healthy or disease diagnosis)
    - Care recommendations
- Provide a **user-friendly dashboard** where users can view their **past history** of scans (if logged in).
- Perform **basic client-side validation** (e.g., acceptable image types and sizes).

**Communication:**

- Sends image files and metadata via HTTP POST requests to the backend API.
- Receives prediction results (species, disease status, suggestions) via JSON responses.

## 2. Backend API Layer

**Technology:** FastAPI (preferred for speed and auto-documentation) or Flask

**Main Responsibilities:**

- **Receive image inputs** from the frontend.
- **Preprocess images** if needed (resize, normalize before sending to ML model).

- **Route images to the correct ML model** (species classifier and disease detector).
- Manage **multi-leaf input** by aggregating predictions across multiple images for higher accuracy.
- **Store prediction results** along with user metadata (timestamp, session ID, etc.).
- **Handle user authentication** and session management if user history tracking is enabled.
- **Serve care instructions** dynamically based on prediction results.
- Provide an **admin API** for accessing system performance data (optional).

**Communication:**

- Acts as the bridge between the frontend, the ML Model Server, and the Database.

## 3. Machine Learning Model Server

**Technology:** TensorFlow Serving or TorchServe, depending on final model framework

**Main Responsibilities:**

- Host the trained **species classification model** and **disease detection model**.
- Accept **preprocessed image data** from the backend.
- Perform **real-time inference** and return:
    - Predicted species (top-k predictions if needed)
    - Disease classification (including a "healthy" class)
    - Confidence scores for each prediction
- Allow **model versioning** to easily deploy improved models without system downtime.

**Inference Flow:**

1. Receive a POST request with an image tensor.
2. Run inference through the appropriate model.
3. Return prediction results in JSON format.

**Optimization Considerations:**

- Models will be optimized for faster inference through quantization or pruning.
- GPU-enabled servers may be used for scaling inference under high user loads.

## 4. Database Layer

**Technology:** MongoDB Atlas (cloud) or Firebase Firestore

**Main Responsibilities:**

- Store **user profiles** (if user accounts are implemented).
- Save **prediction history** for logged-in users, including:
    - Uploaded image metadata
    - Predicted species
    - Detected diseases
    - Timestamps and session information
- Store **model performance logs** (optional) to track the drift or need for retraining.
- Provide **query APIs** for frontend dashboard rendering (e.g., user history timeline).

## 5. Deployment and Infrastructure

**Containerization:**

- Docker will be used to containerize the frontend, backend API, and model server.
- Separate containers for each component allow for scalability and easier maintenance.

**Orchestration and Cloud Hosting:**

- Deployed via Kubernetes or lightweight orchestration tools (e.g., Docker Compose during development).
- Hosting platforms: AWS EC2 + S3 for image storage, Google Cloud Platform, or Azure.

**Continuous Integration / Continuous Deployment (CI/CD):**

- GitHub Actions or GitLab CI/CD pipelines for automated testing, building, and deployment.

**Security:**

- HTTPS for secure communication between frontend, backend, and model server.
- JWT tokens for user authentication if profiles are implemented.
- Database access rules to enforce data security.

---

# Machine Learning Workflow

1. **Data Preprocessing**
    - Normalize images
    - Data augmentation (rotation, zoom, crop, brightness changes)
    - Handle class imbalance if necessary
2. **Model Development**
    - Transfer learning with pre-trained ImageNet models

- Fine-tuning on species and disease datasets
- Exploring multitask learning: a model that predicts species and disease together

3. **Evaluation Metrics**
    - Classification Accuracy
    - Precision, Recall, and F1-Score per class
    - Confusion Matrix Analysis
    - ROC-AUC Curve for disease detection performance
4. **Deployment and Optimization**
    - Quantize models for efficiency
    - Optimize for low-latency inference
    - Set up CI/CD for automatic deployment

---

# Why GreenEye?

GreenEye isn’t just another “plant project” — it’s a proper workout for our machine learning and engineering muscles. The project tackles a *double challenge*: not only will we be training the system to recognize a mind-boggling variety of plant species, but we will also be teaching it to play plant doctor and spot diseases. Between the huge, messy datasets and the twofold prediction task, we are guaranteed enough technical depth to keep every group member happily (and sometimes chaotically) busy.

Scalability is baked right in. What starts as a simple “What plant is this?” app could easily blossom (pun intended) into a full-scale agricultural advisor, a mobile farming assistant, or even a citizen science tool to crowdsource plant health data globally. In short: today it's leaves, tomorrow it could save crops across continents.

GreenEye also hits a real-world pain point: not everyone has access to expensive lab tests or expert botanists when a mysterious leaf spot appears. With GreenEye, a simple photo upload could bring peace of mind — or at least a good nudge toward treatment — within seconds. It's fast, accessible, and could genuinely help farmers, gardeners, and curious plant parents alike.

Finally, let’s talk team dynamics. Unlike projects that dump all the fun onto one unlucky soul, GreenEye divides the work beautifully. Requirements gathering, dataset wrestling, model training, API building, frontend wizardry, deployment drama — everyone gets a share of the action. Each member can specialize, collaborate, and claim their rightful moment of glory when the system finally identifies that leaf as "Mango tree, mild mildew infection" instead of, say, a "banana ghost."