Deen Dayal Upadhyaya Gorakhpur University
A State University established under Uttar Pradesh State University Act 1973
(Accredited A++ by NAAC)

Submitted in Partial Fulfillment of the requirement
For the award of

BACHELOR OF COMPUTER APPLICATION
Session: 2024–2025

<u>A PROJECT REPORT ON</u>

PLANT DISEASE DETECTION

(Web-based Application for Identifying Plant Diseases with Treatment Suggestions)

Prepared By:

1.	**Humaira siddiqui**	**Roll No: 2514057050002**
2.	**Shriddhi Yadav**	**Roll No: 2514057050051**
3.	**Nasim Khan**	**Roll No: 2514057050044**
4.	**Farhan Ahmad**	**Roll No: 2514057050036**
5.	**Shivanee Gupta**	**Roll No: 2514057050015**

Guidance By:

Anjum Ahsan

(*Assistant Professor)*

Aisha Ikhlaq

(*Assistant Professor)*

Deen Dayal Upadhyaya Gorakhpur University
A State University established under Uttar Pradesh State University Act 1973
(Accredited A++ by NAAC)

ISLAMIA COLLEGE OF COMMERCE BUXIPUR,GORAKHPUR

CERTIFICATE

This is to certify that the project **"Plant Disease Detection"** is a bonafide work carried out by **Humaira siddiqui, Shriddhi Yadav, Nasim Khan, Farhan Ahmad, and Shivanee Gupta**, students of **Bachelor of Computer Applications (BCA)** under our guidance and supervision.

The report is submitted in the partial fulfillment of the requirement for the award of the **Degree of Bachelor of Computer Applications** from the **ISLAMIA COLLEGE OF COMMERCE, BUXIPUR, GORAKHPUR** for the session **2023–2024**.

Anjum Ahsan
(Asst. Professor)

Aisha Ikhlaq
(Asst. Professor)

Mohd. Sharjeel Lari
(Head of Department)

Department of Computer

Science

ISLAMIA COLLEGE OF COMMERCE BUXIPUR,GORAKHPUR

CANDIDATE DECLARATION

We hereby certify that the work which is being presented in the project entitled "Plant Disease Detection " in the partial fulfillment of the requirement for the award of the degree of Bachelor of Computer Applications (BCA) in the Department of Computer Science of Islamia College of Commerce, affiliated to Deen Dayal Upadhyay Gorakhpur University, Gorakhpur, is an authentic record of our own work carried out during the period from 02 February 2025 to 08 May 2025 under the supervision of Mrs. Anjum Ahsan, Department of Computer Science, Islamia College of Commerce, Buxipur.

The matter presented in this project has not been submitted by us for the award of any other degree of this or any other Institute/University.

Humaira siddiqui
Shriddhi Yadav
Nasim Khan
Farhan Ahmad
Shivanee Gupta

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Anjum Ahsan **Aisha Ikhlak**
(Asst. Professor) ***(Asst. Professor)***

The BCA Viva-Voce examination of Shobhit, Sameer, Shruti, Sapna, Anjali, graduate students, has been held on ___________.

Signature of H.O.D. **Signature of External Examiner**
Department of Computer Science

ISLAMIA COLLEGE OF COMMERCE BUXIPUR,GORAKHPUR

Acknowledgement

It gives us great pleasure and satisfaction in presenting this report on our project work, *Plant Disease Detection* , as a part of the **Bachelor of Computer Applications (BCA)** graduation course. A project of this nature requires guidance, cooperation, and encouragement from several people, and we are fortunate to have received all of these in abundance.

We would like to extend our heartfelt gratitude to all those who have helped us in successfully completing this project.

We are especially thankful to **Mr. Mohd Sharjeel Lari (Head of Department)**, for his constant support, encouragement, and valuable insights throughout the course of this project. His willingness to help and provide direction has played a vital role in shaping our understanding and approach.

We are also grateful to all our faculty members for equipping us with the knowledge and skills necessary to undertake this project, and for creating an academic environment where learning thrives.

This project involved technologies such as **React** for the frontend, **Django and Python** for the backend for plant disease detection. The practical implementation of these technologies has enhanced our technical expertise and real-world problem-solving abilities.

We have made sincere efforts to ensure that the project is informative, accurate, and meaningful. We have remained honest and dedicated in our work, and we hope this effort reflects in the final outcome.

Lastly, we thank our peers, family, and friends for their constant motivation and encouragement during every phase of this journey.

Thank you all for making this project a reality.

ABSTRACT

Agriculture is a cornerstone of the economy and food system, especially in countries where a majority of the population depends on farming for their livelihood. One of the critical challenges in modern agriculture is the early and accurate identification of plant diseases, which, if left untreated, can cause extensive damage to crops, reduce yield quality, and impact food supply chains. Traditionally, farmers rely on visual inspections and expert consultations to identify plant diseases, which are often time-consuming, subjective, and not always accessible, particularly in rural or remote areas.

This project, titled **"Plant Disease Detection"**, aims to offer a practical and efficient solution by developing a web-based application that allows users to identify plant leaf diseases through image analysis. Users can click or upload a photo of a diseased leaf, and the system will analyze the image to identify the type of disease and provide treatment suggestions. The objective is to assist farmers and plant enthusiasts in diagnosing plant health problems quickly and accurately, without needing direct access to agricultural experts.

The system is developed with a **React.js** frontend for a user-friendly interface, while the backend is implemented in **Django (Python)** to manage data processing and model integration. A trained image classification model has been incorporated to process the uploaded leaf images and recognize patterns corresponding to various plant diseases. Based on the results, the application also recommends general treatment measures, which can help users manage and prevent further damage to their crops.

This project bridges the gap between modern technology and traditional farming by offering an accessible tool that promotes timely intervention, minimizes crop loss, and supports sustainable agricultural practices. It also enhances the decision-making capability of farmers, contributing to better productivity and food security.

INTRODUCTION TO PROJECT

Agriculture plays a vital role in sustaining the economy and feeding the growing population. However, one of the major challenges faced by farmers is the presence of plant diseases that can significantly reduce crop yield and affect the quality of produce. Early detection and proper treatment of these diseases are essential to prevent large-scale damage and to ensure healthy agricultural practices.

Traditionally, the identification of plant diseases has relied on the experience of farmers or consultation with agricultural experts. However, this method can be time-consuming, inconsistent, and often inaccessible, especially in rural or under-resourced areas. To address this problem, there is a need for a solution that is fast, accurate, and user-friendly.

The project titled **"Plant Disease Detection"** is designed to help farmers and plant growers detect diseases in plants through a web-based application. The user can upload or capture a photo of a plant leaf showing symptoms, and the system will analyze the image and identify the disease from known patterns. Along with the disease name, the system also suggests suitable treatment options or preventive measures.

This application is developed using **React.js** for the frontend to provide a smooth and interactive user experience, and **Django (Python)** for the backend to handle data processing and server-side logic. The image analysis is performed using a trained model that has been developed to recognize and classify common plant diseases.

The main goal of this project is to assist in quick disease identification and offer treatment suggestions, making plant care more efficient and accessible. It contributes to improving agricultural outcomes, reducing crop losses, and empowering farmers with practical tools for better crop management.

OBJECTIVE OF THE PROJECT

The primary objectives of the **"Plant Disease Detection"** project are as follows:

1. **Early Disease Detection:**
   To develop a system that enables farmers and gardeners to detect plant diseases at an early stage by analyzing images of plant leaves. Early detection helps prevent the spread of diseases and minimizes crop damage.

2. **Automated Image Analysis:**
   To build a web-based platform where users can simply upload or take a photo of a plant leaf, and the system will automatically analyze the image to identify potential diseases based on a trained model.

3. **User-Friendly Interface:**
   To create a simple and intuitive user interface using **React.js**, ensuring that the application is accessible to both tech-savvy and non-tech-savvy users, particularly farmers and gardening enthusiasts.

4. **Disease Classification and Diagnosis:**
   To implement an effective disease classification model that can identify and categorize plant diseases based on common patterns and symptoms. This helps in providing accurate results to users.

5. **Treatment Recommendations:**
   To suggest potential treatments and preventive measures once a disease is identified. The application will provide actionable guidance to help users manage the disease and protect their crops.

6. **Support for Sustainable Agriculture:**
   To contribute to the sustainable management of crops by enabling farmers to make timely decisions regarding disease control, which will lead to reduced crop losses, increased productivity, and healthier farming practices.

7. **Accessible and Scalable Solution:**
   To design a solution that can be easily accessed via a web browser, ensuring accessibility to users in rural and remote areas with limited access to expert consultation.

8. **Integration of Modern Technology in Agriculture:**
   To incorporate modern technology into traditional agricultural practices, thus enhancing the efficiency and effectiveness of disease management in farming.

TECHNOLOGIES USED

The **"Plant Disease Detection"** project incorporates a variety of technologies that enable it to function efficiently and effectively. The key technologies used in this project are:

1. **React.js (Frontend):**
   React.js is a popular JavaScript library used for building user interfaces, especially single-page applications. It is chosen for this project due to its component-based architecture, which helps in creating a dynamic and responsive user interface. React allows for a smooth user experience by efficiently updating and rendering only the components that need to be changed.

2. **Django (Backend):**
   Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It is used for building the backend of the application, handling server-side operations such as managing data requests, processing images, and serving results to the frontend. Django's built-in tools for security, database management, and scalability make it an ideal choice for this project.

3. **Python (Programming Language):**
   Python is used for the backend logic, especially for integrating the image classification model and handling data processing. Its simplicity, readability, and extensive support for libraries make it perfect for handling image processing and implementing the disease detection algorithm.

4. **TensorFlow/Keras (Machine Learning Framework):**
   TensorFlow and Keras are open-source machine learning frameworks used for building and training the image recognition model. They are used to create a convolutional neural network (CNN) that can classify images of plant leaves and identify disease patterns. These frameworks provide a wide range of tools and pre-trained models, making it easier to develop and deploy the disease detection model.

5. **OpenCV (Computer Vision Library):**
   OpenCV (Open Source Computer Vision Library) is used for image processing tasks such as image enhancement, feature extraction, and manipulation. OpenCV helps in processing the plant leaf images uploaded by the user, ensuring that the model receives high-quality images for accurate disease detection.

6. **HTML/CSS (Web Design):**
   HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) are used for structuring and styling the web pages. HTML is used to create the basic structure of the webpage, while CSS is used to make the application visually appealing and responsive across different devices.

7. **JavaScript (Scripting Language):**
   JavaScript is used to add interactivity to the web pages, allowing users to interact with the application seamlessly. It is used for tasks like handling user input, managing form submissions, and making asynchronous API requests.

8. **SQLite (Database):**
   SQLite is a lightweight, serverless database used to store user data, plant disease information, and other relevant details. It ensures efficient and fast data retrieval while maintaining the integrity and security of the stored data.

9. **Cloud Hosting (Optional):**
   For deploying the application, cloud hosting services like **Heroku**, **AWS**, or **Google Cloud** can be used to host both the frontend and backend. These services offer scalability and flexibility for growing user demands.

10. **Git/GitHub (Version Control):**
    Git is used for version control to track changes made to the codebase during development. GitHub, a web-based platform, is used for code collaboration, backup, and easy deployment. It helps in maintaining a history of the project and collaborating with team members.

SCOPE OF THE PROJECT

The **Plant Disease Detection** project holds significant potential for improving agricultural practices and ensuring the health of plants and crops. The scope of this project can be understood in terms of its features, applications, and potential impact on the farming community and plant health management systems.

1. **Early Detection of Plant Diseases:**
   The scope of the project includes early identification of plant diseases from images of plant leaves. This allows farmers to take timely action before the disease spreads, reducing crop damage and increasing overall productivity. Early detection is crucial for minimizing the use of harmful pesticides and optimizing the use of agricultural resources.

2. **Web-Based Platform:**
   The project provides a web-based solution, making it easily accessible from any device with an internet connection. It does not require specialized software installations, making it user-friendly for farmers in rural and remote areas. The scope of the web application

allows users to interact with the system without needing technical expertise.

3. **Disease Identification and Classification:**
   The project uses machine learning to classify plant diseases based on uploaded leaf images. This system is capable of handling various types of plant diseases and can be updated to include new diseases in the future. This ensures that the scope of the application can be expanded to cater to a wider range of plants and diseases over time.

4. **Treatment Suggestions:**
   After identifying the disease, the project provides suggestions for treatment and preventive measures. These recommendations include appropriate pesticides, fungicides, or natural remedies that can be used to combat the identified disease. The scope of the project includes offering personalized, actionable advice based on the disease diagnosis.

5. **Scalability and Extensibility:**
   The system is designed to be scalable and can be extended to include additional features such as a larger database of plant diseases, integration with agricultural weather forecasting systems, or mobile app development for even greater accessibility. The project also has the potential to integrate with IoT (Internet of Things) devices to monitor plant health continuously.

6. **User-Centric Design:**
   The application's scope extends to providing a user-friendly interface that requires minimal technical knowledge, making it accessible to a wide range of users, from professional farmers to hobbyist gardeners. The design focuses on simplicity, with clear instructions and easy navigation to enhance user engagement.

7. **Support for Sustainable Agriculture:**
   By enabling farmers to identify diseases early, the project supports sustainable agricultural practices. It promotes the efficient use of pesticides and chemicals, which reduces the environmental impact and fosters eco-friendly farming practices. This can ultimately contribute to healthier ecosystems and better-quality produce.

---

8. **Educational Tool:**
   Beyond just disease detection, the project can be used as an educational tool for teaching plant health and disease management. It can be incorporated into agricultural training programs or workshops to help farmers understand how diseases spread and

how they can manage plant health more effectively.

9. **Global Reach:**
   While the initial focus may be on local agriculture, the project has the potential to reach a global audience. As the system evolves, it can be adapted to identify diseases in various plants from different regions, making it applicable worldwide. This global scope allows for collaboration with international agricultural research organizations.

10. **Continuous Improvement:**
    The system's machine learning model can be continually trained with new data, improving its accuracy and the range of diseases it can detect. The scope of the project includes ongoing updates to the disease database, ensuring that the tool remains relevant as new diseases emerge and agricultural practices evolve.

TABLE OF CONTENTS

- **Benefits of RAD**
- **Objective of Using RAD**

7. DIAGRAMS & CHARTS

- **ER Diagram**
- **Dataflow Diagram**
- **Database Schema**
- **Gantt Chart** (Project Timeline)
- **Agile Model** (Sprint Breakdown)

8. TECHNOLOGIES USED

- **Frontend:** React.js
- **Backend:** Django (Python)
- **Database:** SQLite/MySQL
- **Machine Learning:** TensorFlow, Keras, PyTorch
- **Image Processing:** OpenCV, Pillow
- **Deployment:** Cloud Hosting

9. SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

- **Functional Requirements**
  - User Interface
  - Image Upload
  - Disease Detection
  - Treatment Recommendations
  - User Feedback
- **Non-Functional Requirements**
  - Performance, Reliability, Security, Scalability
- **Hardware Requirements**

10. DATABASE DESIGN

- **MySQL Tables & Schema**
- **Role of MySQL in the Project**

11. ROLE OF OPENCV & MACHINE LEARNING

- **Image Preprocessing** (Resizing, Noise Removal, Thresholding)
- **CNN Model for Disease Classification**
- **Training & Deployment**

12. IMPLEMENTATION (CODE)

- **Frontend (React.js)**
  - Dashboard
  - Detection Plant
  - Header & Footer
  - Login & Registration
  - Plant Library
- **Backend (Django & Python)**
  - `settings.py`
  - `models.py`
  - `views.py`
  - `utils.py` (OTP, Image Embedding)

13. CONCLUSION & FUTURE SCOPE

- Impact on Agriculture
- Scalability & Future Enhancements

System Analysis

System Analysis involves a detailed study of the existing processes and requirements to understand the functionality needed for the new system. It identifies the problems in the current manual or semi-automated approaches and proposes efficient solutions that the new system will deliver. In the context of this project, system analysis focuses on understanding the agricultural landscape, challenges in disease identification, and how the software solution can assist in disease detection and treatment guidance.

---

1. Existing System

In many parts of the world, including rural areas of India, farmers rely on their own experience or local agricultural experts to identify plant diseases. The methods are often manual, time-consuming, and inaccurate. Some major drawbacks of the current approach include:

- **Lack of accurate diagnosis** leading to improper treatment.
- **Limited access to experts** in remote areas.
- **Delayed identification** resulting in widespread crop damage.
- **No digital record** or tracking of disease patterns and treatments.

---

2. Proposed System

The proposed system is a web-based platform that enables users to upload an image of a plant leaf to detect diseases and get treatment suggestions. This is achieved using image processing and machine learning models. The system is designed to:

- Identify plant diseases with high accuracy based on leaf images.
- Provide suitable treatment recommendations instantly.

- Allow farmers or users to operate the system through a simple and user-friendly web interface.
- Store historical data for future analysis or record keeping.

---

3. System Requirements

Functional Requirements:

- Upload a plant leaf image.
- Analyze the image and identify possible diseases.
- Display the name of the disease (if any) and suggest treatment options.
- Maintain a history of user uploads and predictions.

Non-Functional Requirements:

- The system should be accessible 24/7 through a web browser.
- Fast and accurate processing of uploaded images.
- Secure and scalable backend to manage user and image data.
- Cross-platform compatibility for ease of access from multiple devices.

---

4. Feasibility Study

a. Technical Feasibility:

- The technologies used (React, Django, Python libraries like TensorFlow/Keras) are reliable and widely supported.
- Availability of open-source datasets for training the model.
- Sufficient computing power for model inference through web servers.

b. Operational Feasibility:

- Easy-to-use interface makes it suitable even for users with minimal technical background.
- Can be deployed online and accessed from remote areas with internet access.

c. Economic Feasibility:

- Minimal cost involved as most tools and platforms used are open-source.
- Reduces long-term losses due to crop damage, proving cost-effective for farmers.

---

5. Benefits of the Proposed System

- **Time-Saving:** Quickly identifies plant diseases without the need for a field expert.
- **Accuracy:** Provides better accuracy than traditional methods.

- **Availability:** Can be used from anywhere and at any time.

- **Cost-Efficient:** Reduces dependency on paid expert consultations.
- **Data Storage:** Keeps a digital history for analysis and improvement.

Various Phases in RAD (Rapid Application Development)

Rapid Application Development (RAD) is a software development methodology that emphasizes quick development and iteration of prototypes over strict planning and testing phases. It is particularly suitable for projects like Plant Disease Detection, where requirements may evolve based on user feedback and model performance. The following are the key phases involved in RAD:

---

1. Business Modeling Phase

In this phase, the flow of information between various business functions is analyzed. For the Plant Disease Detection project, it involves understanding how users (farmers, agricultural experts) interact with the system to diagnose plant diseases and obtain treatment suggestions. This includes:

- User roles and requirements
- Data sources (e.g., plant leaf images)
- Communication flow and expected outcomes

---

2. Data Modeling Phase

This phase focuses on identifying and designing the data objects that are essential for the application. For this project:

- Image data (uploaded leaf photos)

- Disease labels and treatment information

- User profiles and upload history
- Database schema for storing images, results, and feedback

---

3. Process Modeling Phase

In this phase, the data objects identified are transformed to achieve business information flow. This includes:

- How an uploaded image is processed
- How the disease is detected from the image
- How the result is communicated to the user
- How treatment suggestions are generated and displayed

---

4. Application Generation Phase

This is where the actual system is built. It includes:

- Frontend development using **ReactJS**
- Backend API and logic using **Django and Python**
- Integration of the trained **Machine Learning model**
- Deployment of the web application for user access

This phase also includes prototyping, testing individual components, and ensuring integration between frontend, backend, and components.

---

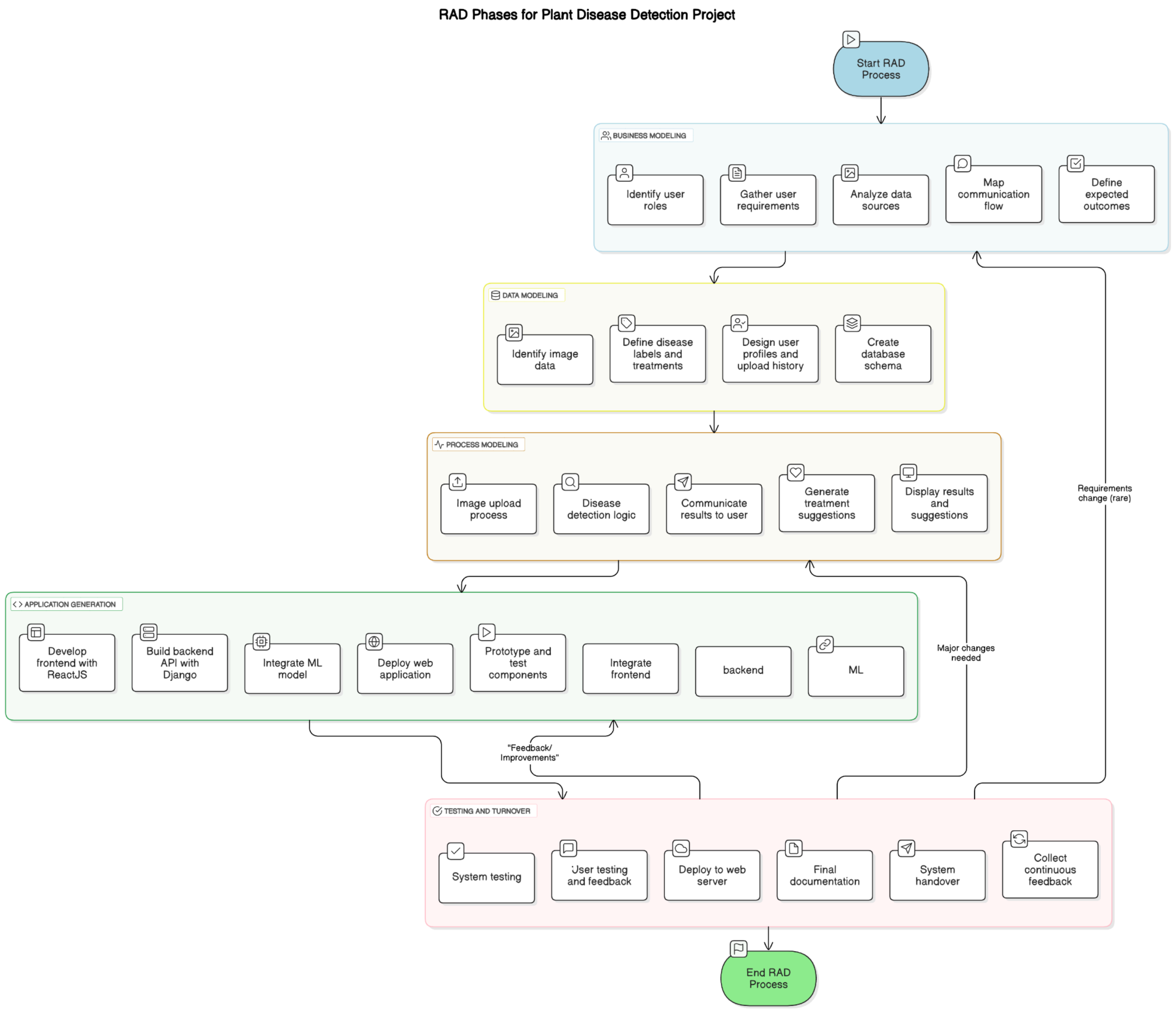
5. Testing and Turnover Phase

This phase includes:

- System testing to ensure functionality, accuracy, and performance
- User testing and feedback collection
- Deployment on a web server for real-time access
- Final documentation and system handover

Continuous feedback is collected and rapid iterations are made to improve the system based on user experience and disease detection performance.

RAD Model


RAD Phases for Plant Disease Detection Project
Start RAD Process
BUSINESS MODELING
Identify user roles
Gather user requirements
Analyze data sources
Map communication flow
Define expected outcomes
DATA MODELING
Identify image data
Define disease labels and treatments
Design user profiles and upload history
Create database schema
PROCESS MODELING
Image upload process
Disease detection logic
Communicate results to user
Generate treatment suggestions
Display results and suggestions
Requirements change (rare)
APPLICATION GENERATION
Develop frontend with ReactJS
Build backend API with Django
Integrate ML model
Deploy web application
Prototype and test components
Integrate frontend
backend
ML
Major changes needed
"Feedback/ Improvements"
TESTING AND TURNOVER
System testing
User testing and feedback
Deploy to web server
Final documentation
System handover
Collect continuous feedback
End RAD Process

Benefits of the Rapid Application Development (RAD) Model

1. **Faster Development:**
   - RAD focuses on rapid prototyping and iterative releases, reducing the overall development time.
   - In this project, it allowed for a quick rollout of the web-based interface and early testing of plant disease detection.
2. **User Involvement and Feedback:**
   - Continuous involvement of users (e.g., farmers, students, or testers) ensured the application was user-friendly and effective.
   - Regular feedback helped refine the interface and improve the accuracy of disease detection suggestions.
3. **Flexibility to Changes:**
   - RAD allows changes to be made even in later stages of development.
   - If any enhancement or modification was needed in detection logic or treatment suggestions, it could be easily incorporated.
4. **Early System Delivery:**
   - A working model of the system could be shown early in the development process, giving users a glimpse of what to expect.
   - This helped build confidence and allowed for real-world testing with actual plant leaf images.
5. **Reduced Development Risk:**
   - Frequent iterations help identify issues early, reducing the chance of project failure.

    - Bugs or inefficiencies in the disease detection model could be fixed in short cycles without delaying the project.

6. **Better Productivity:**
    - Because of parallel development and quick feedback cycles, the development team stayed productive and focused.
    - Tasks like frontend, backend, and model integration were handled simultaneously.
7. **Efficient Resource Utilization:**
    - Resources such as tools, developers, and testing environments were utilized effectively due to the modular and iterative nature of RAD.

Objective of Using RAD (Rapid Application Development) in the Plant Disease Detection Project

The primary objective of adopting the **Rapid Application Development (RAD)** model in the *Plant Disease Detection* project is to ensure **quick and efficient delivery** of a functional, user-centric application while maintaining high quality and adaptability throughout the development process. The key goals are:

---

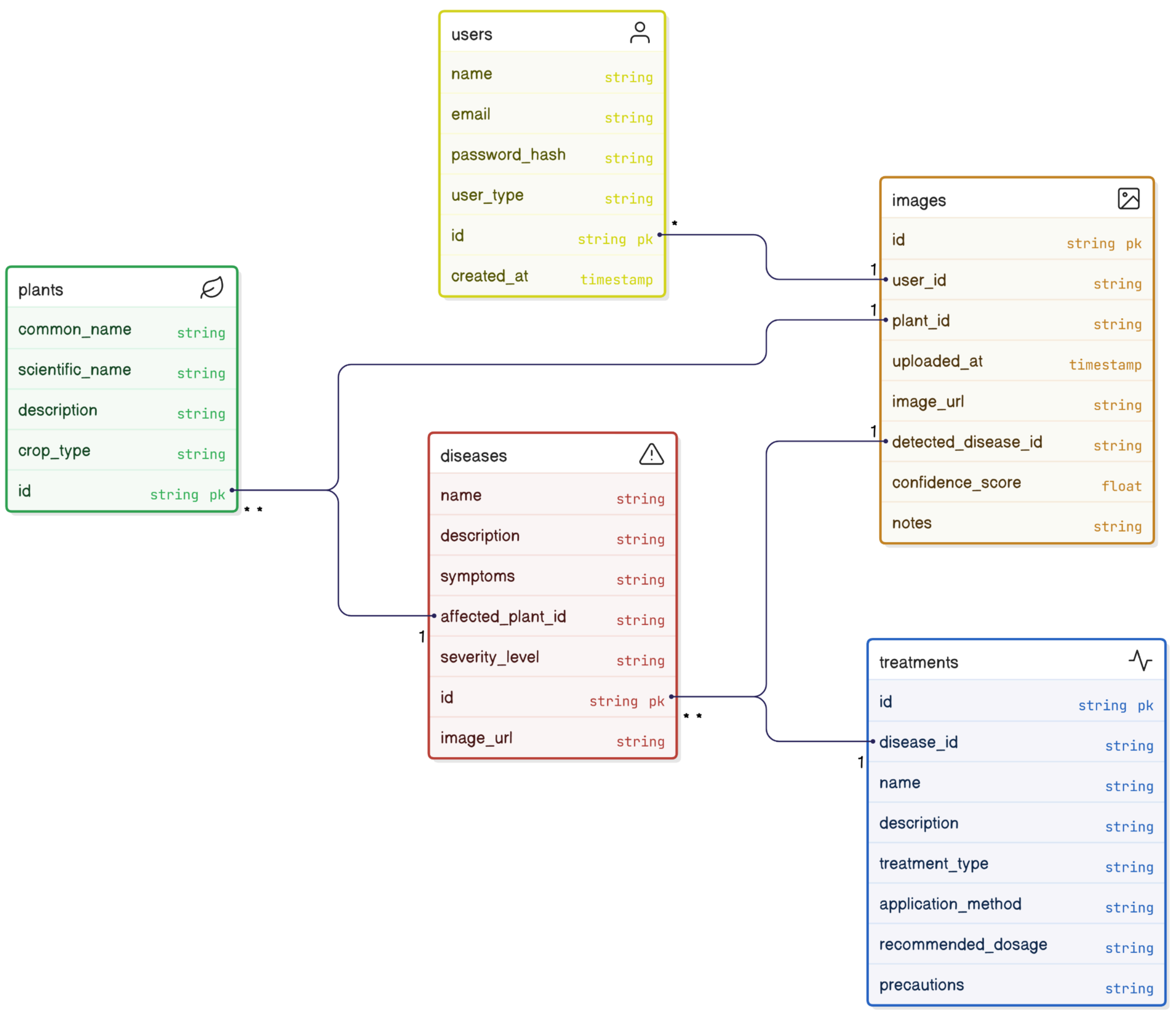
1. **Quick Prototyping and Delivery:**
   - To rapidly build working prototypes of the plant detection interface and image processing modules so users can test and validate the functionalities early in the process.
2. **User Feedback Integration:**
   - To involve end users (e.g., students, agricultural experts, or farmers) in the development cycle by collecting and incorporating their feedback to enhance the usability and accuracy of disease detection.
3. **Parallel Development Process:**
   - To allow frontend (React), backend (Django), and model integration teams to work simultaneously, thereby reducing development time and improving team efficiency.
4. **Flexibility to Incorporate Changes:**
   - To support ongoing improvements in disease identification methods and treatment suggestions as new requirements or suggestions arise.
5. **Low Risk and Better Quality Control:**
   - To detect issues and bugs early in the development lifecycle by testing prototypes frequently, ensuring a more stable and refined final product.

---

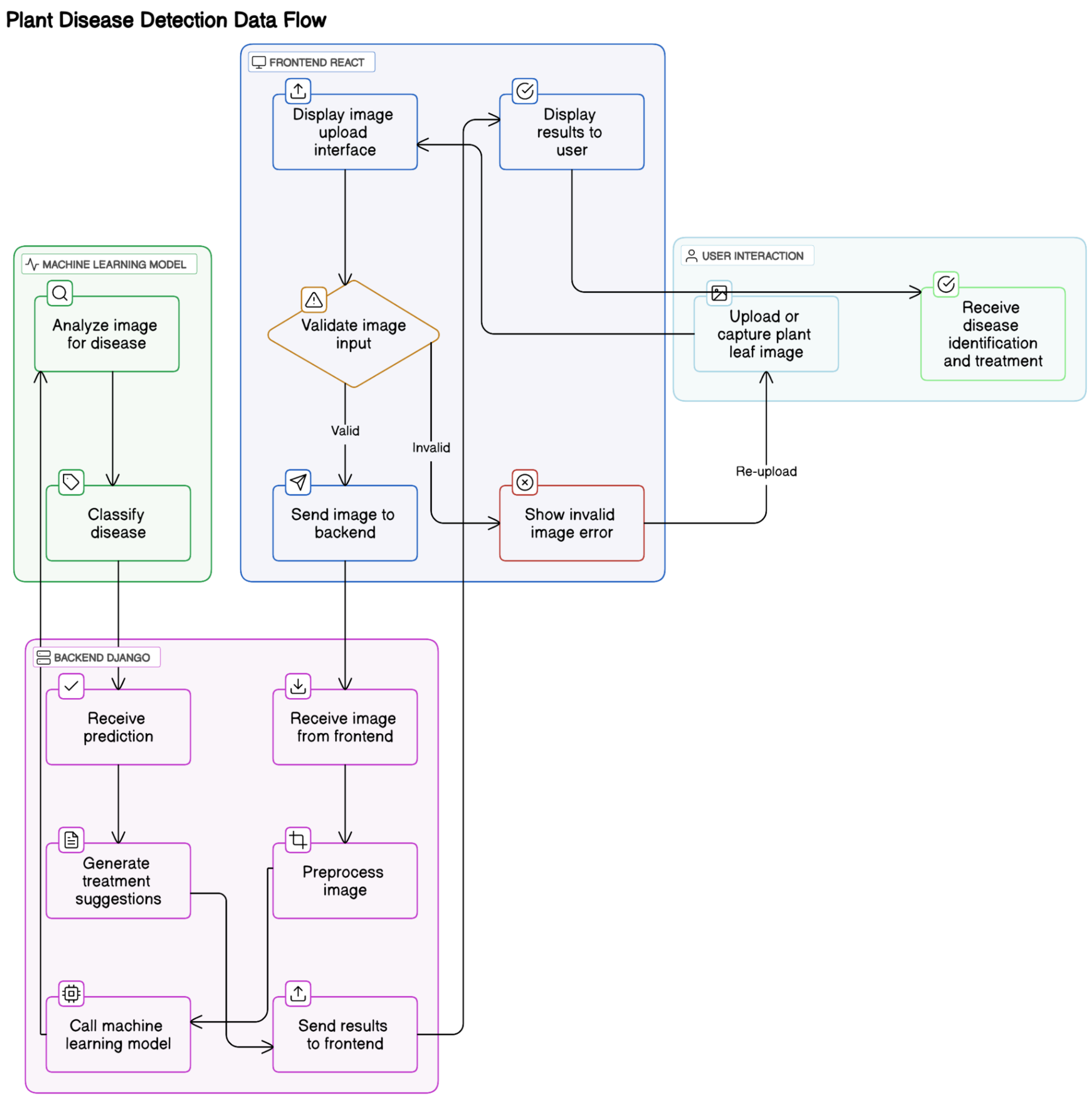
6. **Efficient Use of Resources:**
    - To optimize the use of time, tools, and team members by focusing on iterative modules that can be independently built, tested, and integrated.
7. **Early Delivery of Usable Modules:**
    - To provide a semi-functional system early for demonstration and real-world testing with plant images, making it easier to validate and enhance the final application.

ER Diagram Of The Project:

Plant Disease Detection System Data Model

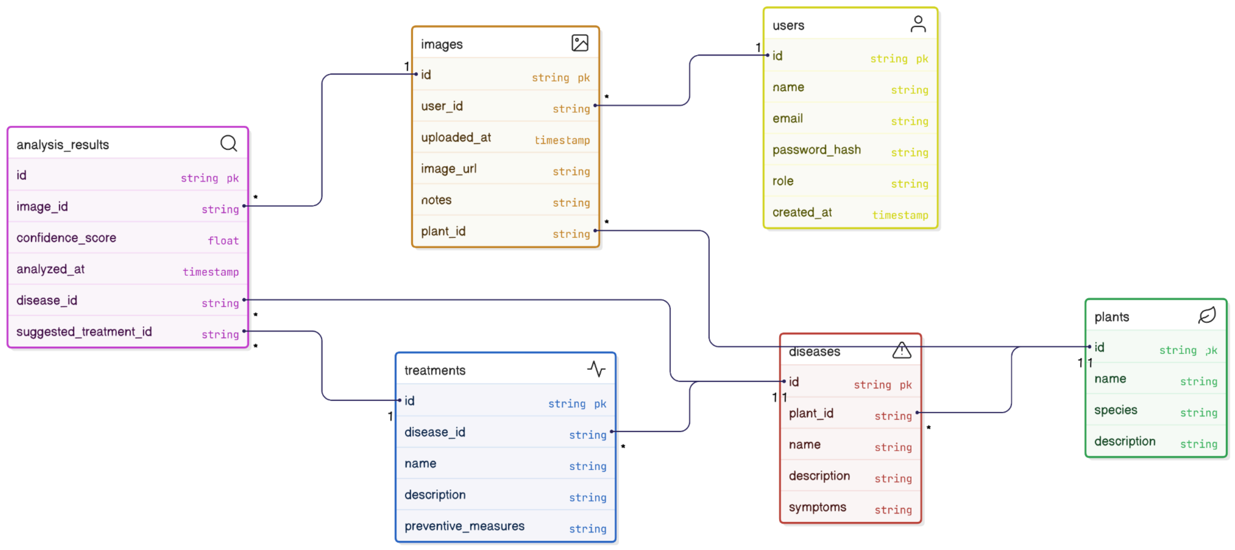


Dataflow Diagram Of the Project:


Plant Disease Detection Data Flow
FRONTEND REACT
Display image upload interface
Display results to user
Validate image input
Valid
Invalid
Send image to backend
Show invalid image error
MACHINE LEARNING MODEL
Analyze image for disease
Classify disease
USER INTERACTION
Upload or capture plant leaf image
Receive disease identification and treatment
Re-upload
BACKEND DJANGO
Receive prediction
Receive image from frontend
Generate treatment suggestions
Preprocess image
Call machine learning model
Send results to frontend

Database Diagram

Plant Disease Detection Data Model

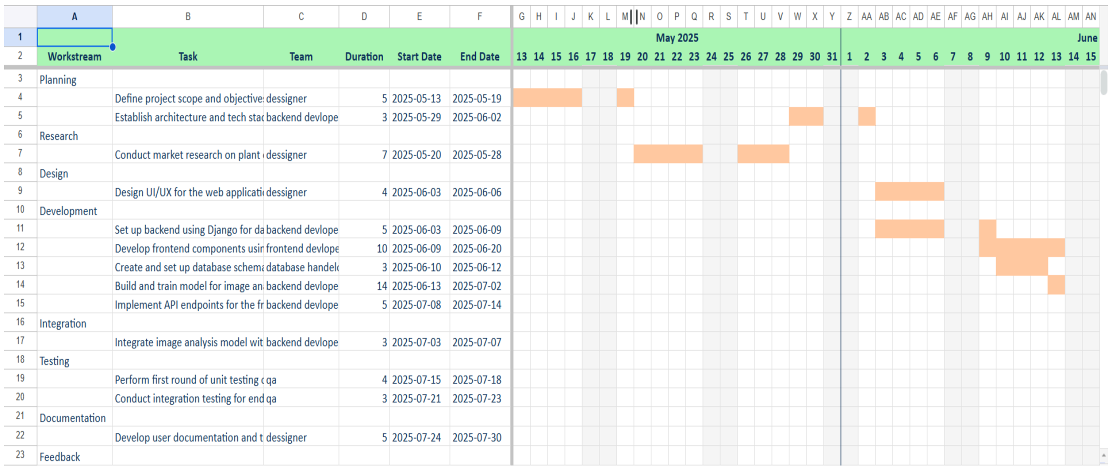


GANTT CHART

In the *Plant Disease Detection* project, the **Gantt Chart** serves as a visual project management tool that outlines the timeline and progress of various tasks across the project's development lifecycle. It displays each major phase of the project, the tasks within those phases, their start and end dates, and the duration of each task. The Gantt Chart begins with the **Project Planning Phase**, including requirement gathering and initial research, followed by the **Design Phase**, which involves preparing wireframes and the system architecture.

Next comes the **Development Phase**, where frontend development using React and backend development using Django and Python are carried out. This also includes integrating the machine learning model for disease detection. After development, the **Testing Phase** is initiated, where unit testing, integration testing, and user acceptance testing (UAT) are conducted. Once the system is thoroughly tested, the project moves into the **Deployment Phase**, where the application is hosted and made available to users.

Finally, the **Documentation and Report Submission Phase** ensures that all project details, findings, and user guides are properly documented and submitted. The Gantt Chart helps in tracking the timely completion of each task, ensuring the project remains on schedule and all deliverables are met efficiently.



	A	B	C	D	E	F
1						
2	Workstream	Task	Team	Duration	Start Date	End Date
3	Planning					
4		Define project scope and objective:	dessigner	5	2025-05-13	2025-05-19
5		Establish architecture and tech stac	backend devlope	3	2025-05-29	2025-06-02
6	Research					
7		Conduct market research on plant (	dessigner	7	2025-05-20	2025-05-28
8	Design					
9		Design UI/UX for the web applicati	dessigner	4	2025-06-03	2025-06-06
10	Development					
11		Set up backend using Django for da	backend devlope	5	2025-06-03	2025-06-09
12		Develop frontend components usir	frontend devlope	10	2025-06-09	2025-06-20
13		Create and set up database schema	database handel	3	2025-06-10	2025-06-12
14		Build and train model for image an	backend devlope	14	2025-06-13	2025-07-02
15		Implement API endpoints for the fr	backend devlope	5	2025-07-08	2025-07-14
16	Integration					
17		Integrate image analysis model wit	backend devlope	3	2025-07-03	2025-07-07
18	Testing					
19		Perform first round of unit testing (	qa	4	2025-07-15	2025-07-18
20		Conduct integration testing for end	qa	3	2025-07-21	2025-07-23
21	Documentation					
22		Develop user documentation and t	dessigner	5	2025-07-24	2025-07-30
23	Feedback					

	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1								May 2025											
2	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
3																			
4	🟧	🟧	🟧	🟧			🟧												
5																	🟧	🟧	
6																			
7								🟧	🟧	🟧	🟧			🟧	🟧	🟧			
8																			
9																			
10																			
11																			
12																			
13																			
14																			
15																			
16																			
17																			
18																			
19																			
20																			
21																			
22																			
23																			

	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN
1															June
2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3															
4															
5		🟧													
6															
7															
8															
9			🟧	🟧	🟧	🟧									
10															
11			🟧	🟧	🟧	🟧			🟧						
12									🟧	🟧	🟧	🟧	🟧		
13										🟧	🟧	🟧			
14													🟧		
15															
16															
17															
18															
19															
20															
21															
22															
23															

Agile Model

Requirement Gathering

- Collect initial requirements from users like farmers or gardeners.
- Identify key features: image upload, disease detection, and treatment suggestions.

Sprint Planning

- Divide the project into small sprints (1–2 weeks each).
- Set clear goals for each sprint (e.g., UI design, backend integration, model output display).

Sprint 1: Basic UI Development

- Develop a user-friendly frontend using React.
- Allow users to upload images of plant leaves.

Sprint 2: Backend Setup

- Build Django backend to handle image upload and user data.
- Connect the frontend to backend via APIs.

Sprint 3: Disease Detection Integration

- Connect the backend to the machine learning model for image analysis.

- Display detected disease name and confidence score to the user.

Sprint 4: Treatment Suggestion Module

- Link disease names to relevant treatment suggestions stored in a database.
- Show treatment steps or precautions after disease detection.

Sprint 5: Testing and Feedback

- Conduct unit testing, integration testing, and user testing.
- Collect user feedback to improve UI/UX and functionality.

Sprint 6: Enhancements and Bug Fixes

- Add features like disease history, treatment updates, and admin control.
- Fix any bugs reported during testing or from user feedback.

Sprint 7: Deployment and Documentation

- Deploy the project on a web server.
- Prepare user manual, project report, and system documentation.

Review and Final Presentation

- Review the final product.
- Present the project to the evaluation committee.

<u>Software Requirements Specification (SRS)</u>

Project Title: Plant Disease Detection System

1. Frontend: React.js

React.js is a powerful JavaScript library used for building user interfaces, especially for single-page applications where fast performance and smooth user experience are essential.

- It allows **component-based development**, which makes the code reusable and easier to maintain.
- The **virtual DOM** in React ensures that updates to the UI are efficient and only the parts that need to change are re-rendered.
- React helps create a **responsive and interactive user interface**, where users can upload images, view predictions, and get results without the page reloading.
- It supports seamless integration with backend APIs and libraries such as Axios or Fetch for HTTP communication.

---

2. Backend: Django (Python Framework)

Django is a high-level Python web framework that follows the **Model-View-Template (MVT)** architecture and promotes rapid development.

- It handles **server-side logic**, such as receiving uploaded images, processing them, and returning results to the frontend.
- Django provides a built-in **admin panel** for managing data models and users if needed.
- It's **secure by default**, preventing many common web vulnerabilities like SQL injection and cross-site scripting (XSS).
- The framework is scalable and integrates well with **machine learning models**, making it ideal for AI-based applications.

---

3. Database: SQLite / PostgreSQL

Databases are essential for storing persistent data such as user submissions, prediction results, and feedback.

- **SQLite** is a lightweight database that comes pre-installed with Python. It's perfect for **local development and testing** because it stores all data in a single file and requires minimal setup.
- **PostgreSQL** is a powerful, open-source, and production-grade database used for hosting the project live. It supports advanced features like concurrent transactions, JSON fields, and indexing for faster queries.
- Both databases are used with Django via its **ORM (Object-Relational Mapping)** system, which allows you to interact with the database using Python code instead of raw SQL.

---

4. Image Processing: OpenCV / Pillow

Before a machine learning model can analyze an image, it must be pre-processed.

- **OpenCV** (Open Source Computer Vision Library) is a widely-used library for computer vision tasks such as resizing, cropping, filtering, color conversion, and noise removal.

- **Pillow** (Python Imaging Library fork) is a lightweight alternative for image manipulation like rotation, format conversion, or enhancing contrast and brightness.

- These tools help in **cleaning and preparing the input images** before sending them to the ML model for prediction, ensuring higher accuracy and consistency.

---

5. Machine Learning Libraries: TensorFlow, Keras, PyTorch

These are the core technologies that enable the disease detection functionality.

- **TensorFlow** (developed by Google) and **PyTorch** (developed by Facebook) are two of the most popular open-source libraries used for creating, training, and deploying deep learning models.

- **Keras** is an easy-to-use high-level API that runs on top of TensorFlow, making model development simpler and faster.

- These frameworks support **Convolutional Neural Networks (CNNs)** which are highly effective in image classification tasks like identifying leaf diseases.

- The model, once trained on a dataset of diseased leaf images, can be **saved and loaded into the Django backend**, where it is used for making predictions on new images uploaded by users.

---

6. Operating System: Cross-Platform Compatibility

The project is designed to work on all major operating systems:

- **Windows** – Suitable for development with tools like PyCharm, VS Code, or Anaconda.
- **macOS** – Preferred by developers using Unix-based environments and tools.
- **Linux** – Ideal for deploying the project on cloud servers (like AWS, DigitalOcean, or Heroku) due to its performance, flexibility, and command-line utilities.
- The technology stack is **platform-independent**, which means it can be developed on one OS and deployed on another with minimal changes.

Use of MySQL in the Plant Disease Detection Project

1. User Information Management

- **Purpose:** To register and authenticate users such as farmers, agricultural officers, or general users.
- **MySQL Table Example:** `users`
- **Fields:** `user_id`, `name`, `email`, `password`, `role`, `created_at`
- **Why:** Securely manages access to the system and keeps user records.

🖼️ 2. Image Upload History

- **Purpose:** To store metadata related to images uploaded by users for disease detection.
- **MySQL Table Example:** `image_uploads`
- **Fields:** `upload_id`, `user_id`, `image_path`, `upload_time`, `plant_type`
- **Why:** Keeps track of what plant images were uploaded, when, and by whom.

🧪 3. Prediction Results

- **Purpose:** To save the results given by analyzing plant images.
- **MySQL Table Example:** `predictions`

- **Fields:** `prediction_id`, `upload_id`, `disease_name`, `confidence_score`, `predicted_at`
- **Why:** Enables users to view past diagnoses and supports data analysis.

📝 4. Disease Details and Suggestions

- **Purpose:** To store detailed descriptions of plant diseases, symptoms, causes, and remedies.
- **MySQL Table Example:** `diseases`
- **Fields:** `disease_id`, `plant_type`, `disease_name`, `symptoms`, `causes`, `treatment`
- **Why:** Helps in providing rich information and recommendations to users post-prediction.

💬 5. User Feedback

- **Purpose:** To collect user feedback on prediction accuracy and application experience.
- **MySQL Table Example:** `feedback`
- **Fields:** `feedback_id`, `user_id`, `prediction_id`, `rating`, `comments`, `submitted_at`
- **Why:** Improves system quality by letting developers assess user satisfaction.

🔄 Why MySQL for This Project?

- ✅ **Efficient Data Storage:** Handles structured tabular data like user records and prediction history.
- ✅ **Easy Integration with Django:** Django supports MySQL using `mysqlclient` or `PyMySQL`.
- ✅ **Relational Integrity:** Maintains relationships using primary and foreign keys between tables like `users`, `uploads`, and `predictions`.
- ✅ **Reliable Performance:** Manages multiple users and concurrent data operations without lag.
- ✅ **Security:** Supports user roles and authentication to protect sensitive data.
- ✅ **Query Capability:** Enables complex queries to generate disease reports or usage statistics.

📊 Example Relationship:

- A **user** uploads an **image** → the system generates a **prediction** → stores details in the **database**.
- All entries are **linked using foreign keys**, enabling traceability from user to result.

Role of OpenCV in Plant Disease Detection Project

In this project, the primary goal is to detect diseases in plant leaves by analyzing uploaded images. Before these images are passed, they must be **pre-processed** for accuracy and consistency.

🔧 Why Preprocessing is Necessary

Plant leaf images can be captured under various conditions—different lighting, angles, backgrounds, and resolutions. These factors can negatively impact the model's prediction if not handled properly. OpenCV helps normalize the image input by performing a series of image-processing tasks.

🧰 How OpenCV is Used in This Project

1. Image Resizing

- **Purpose:** Models require input images of a fixed size (e.g., 224x224 pixels).
- **OpenCV Function:** `cv2.resize()`
- **Project Use:** Ensures every uploaded leaf image is of uniform size for consistent input.

2. Color Conversion

- **Purpose:** Convert colored images to grayscale or different color spaces (like HSV).
- **OpenCV Function:** `cv2.cvtColor(image, cv2.COLOR_BGR2RGB)`
- **Project Use:** Converts images to the color format required by the model.

---

3. Noise Removal

- **Purpose:** Clean unnecessary noise from the image which might confuse the model.
- **OpenCV Function:** `cv2.GaussianBlur()`, `cv2.medianBlur()`
- **Project Use:** Smoothens the image to reduce background noise and highlight important leaf features.

4. Thresholding & Masking

- **Purpose:** Extract the leaf region from the background.
- **OpenCV Function:** `cv2.inRange()`, `cv2.bitwise_and()`
- **Project Use:** Helps the model focus on the leaf rather than the background (like soil or hand).

5. Image Enhancement

- **Purpose:** Improve contrast, sharpness, or brightness to highlight disease patterns (e.g., spots, color changes).
- **OpenCV Function:** `cv2.equalizeHist()` or manual adjustment of pixel values.
- **Project Use:** Makes disease symptoms clearer to the model.

6. Edge Detection (Optional)

- **Purpose:** Detect shapes, outlines, or spots on the leaf surface.
- **OpenCV Function:** `cv2.Canny()`
- **Project Use:** Can help in visual analysis or feature extraction, if needed.

🔄 Workflow Example in Project

1. User uploads an image of a plant leaf.
2. Image is passed to the **OpenCV preprocessing pipeline**:
   - Resize the image.
   - Convert color format.
   - Remove noise.
   - Mask out background.
   - Enhance the leaf features.
3. Processed image is sent to the **Django model**.
4. Model returns prediction (e.g., "Tomato Leaf Blight" with 91% accuracy).

✅ Benefits of Using OpenCV in Your Project

- **Higher Prediction Accuracy:** Clean and standardized inputs improve model performance.
- **Automation:** Reduces the need for manual image editing before analysis.
- **Speed:** OpenCV is optimized for performance and works fast even with large image datasets.
- **Flexibility:** Supports many preprocessing options that can be customized for various plant types or diseases.

Role of Machine Learning Libraries (TensorFlow, Keras, and PyTorch) in Plant Disease Detection Project

In the **Plant Disease Detection** project, the key functionality revolves around detecting diseases in plant leaves by analyzing their images. This task falls under **image classification** using **Deep Learning** techniques, particularly **Convolutional Neural Networks (CNNs)**. To accomplish this, we leverage popular machine learning libraries like **TensorFlow**, **Keras**, and **PyTorch** to build, train, and deploy our model. Here's how each of these technologies fits into the project:

---

1. TensorFlow and Keras

TensorFlow:

- **Overview:** Developed by Google, TensorFlow is one of the most popular open-source libraries for **Deep Learning**. It provides an ecosystem of tools that help in building complex models, training them efficiently, and deploying them into production.
- **Project Use:** In this project, TensorFlow is used to handle the **heavy lifting** of training and deploying the plant disease detection model. It supports CNNs, which are ideal for processing image data and detecting intricate patterns (like disease symptoms on leaves).
- **Key Benefits:**
  - **Scalability:** TensorFlow can handle large datasets of plant leaf images for training the model.
  - **Optimized Performance:** It uses advanced optimizations and parallel processing to ensure fast model training.
  - **Deployment:** Once trained, the model can be easily exported and deployed via the Django backend.

Keras (as an API for TensorFlow):

- **Overview:** Keras is a high-level neural networks API that runs on top of TensorFlow, making it more user-friendly. It simplifies model development, allowing for easy configuration of layers, activation functions, loss functions, and optimization methods.
- **Project Use:** In this project, Keras is used to **design and train the CNN**. It abstracts away much of the complexity of working with raw TensorFlow, making the process more efficient and less error-prone. Keras allows easy experimentation with different model architectures and hyperparameters, which is crucial for image classification tasks like plant disease detection.
- **Key Benefits:**
  - **Faster Development:** Keras allows for quicker iteration and prototyping.
  - **Easy-to-understand API:** The code is cleaner, and it's easier to modify or extend.
  - **Prebuilt Layers:** Keras provides a variety of layers like **Conv2D**, **MaxPooling2D**, and **Dense**, which are essential for CNNs in image processing tasks.

---

2. PyTorch

Overview:

- **PyTorch**, developed by Facebook, is another powerful open-source machine learning library. It's known for its dynamic computation graph and ease of use for research purposes.
- **Project Use:** PyTorch is an alternative to TensorFlow for developing the disease detection model. It is particularly favored for rapid experimentation and flexibility.

- **Key Benefits:**
    - **Dynamic Graphs:** PyTorch allows for dynamic changes to the model during training, making it easier to debug and modify the architecture.
    - **Community and Resources:** PyTorch has a large community and a wealth of resources available, which can help in fine-tuning the plant disease detection model.
    - **Integration:** The model built using PyTorch can be integrated with Django backend just like TensorFlow, making it suitable for use in real-time predictions.

---

3. Convolutional Neural Networks (CNNs) for Disease Detection

Convolutional Neural Networks (CNNs) are the **foundation of the image classification model** used in this project. CNNs are designed to automatically detect patterns such as edges, textures, and shapes in images. This makes them perfect for plant disease detection, as the model can learn to recognize specific patterns in the plant leaf images associated with different diseases.

- **How CNNs Work in This Project:**
    - **Input Layer:** The leaf images are resized and preprocessed (using libraries like OpenCV) before being passed to the CNN.
    - **Convolutional Layers:** These layers automatically extract features like edges, spots, and colors from the leaf image, which are important indicators of plant diseases.
    - **Pooling Layers:** Pooling is applied to reduce the dimensionality and computational cost, while retaining the essential features from the image.
    - **Fully Connected Layers:** These layers take the features extracted by the convolutional layers and use them to classify the image into one of the categories
    -

- (e.g., “Healthy”, “Blight”, “Rust”, etc.).

- **Output Layer:** The final output layer generates the predicted class label for the image.

---

4. Training the Model

- **Dataset:** A dataset of labeled images of plant leaves (with and without diseases) is used to train the model. This dataset is preprocessed and split into training and testing sets.
- **Model Training:** Using TensorFlow/Keras (or PyTorch), the CNN model is trained on the labeled images. The model learns to associate specific features in the leaf images with various diseases.
- **Evaluation:** The trained model is evaluated using accuracy, precision, recall, and F1-score metrics to measure its performance in disease detection.

---

5. Model Deployment

Once the model is trained and evaluated, it can be deployed on the **Django backend** of the application. This allows users to upload images of plant leaves, and the model will predict the disease based on the input image.

- **Saving the Model:** After training, the model is saved as a file (e.g., `.h5` for Keras/TensorFlow or `.pth` for PyTorch).
- **Backend Integration:** The saved model is loaded into the Django backend, which handles image uploads, passes them through the model, and returns the predicted result (disease and confidence).

Functional Requirements

The **Functional Requirements** describe the essential features and functionalities that the system must offer in order to meet user expectations and perform its intended purpose. Below are the key functionalities for the **Plant Disease Detection** project:

---

1. User Interface

- **Requirement:** The system must provide a user-friendly and intuitive interface that allows users to interact with the system easily.

- **Details:** The user interface (UI) should be clean, simple, and easy to navigate. It will include basic options such as uploading plant leaf images, viewing results, and interacting with the system. The design should cater to non-technical users, such as farmers and gardeners, making it accessible even for those with minimal technical skills.

2. Image Upload

- **Requirement:** Users should be able to upload images of plant leaves in commonly used formats such as JPEG and PNG.

- **Details:** The system should support various image formats (JPEG, PNG, and possibly others), allowing users to upload clear and high-quality images of plant leaves. These images will then be processed by the machine learning model for disease detection. The upload interface will be simple, with an option to browse and select files or drag-and-drop the images.

3. Disease Detection

- **Requirement:** After uploading the image, the system should process the image using a trained machine learning model and predict the disease affecting the plant.

- **Details:** Once the user uploads a leaf image, it is passed through the machine learning model that has been trained on a large dataset of plant leaf images. The model will analyze the image to detect specific symptoms of diseases and output the disease type

---

- (e.g., blight, rust, mildew, etc.). The result should be displayed on the screen within a few seconds.

4. Treatment Recommendation

- **Requirement:** Based on the detected disease, the system should recommend appropriate treatments, such as fertilizers, pesticides, or other remedies.
- **Details:** Once the disease is identified, the system will provide the user with suggestions for the most effective treatments. These recommendations can include the use of specific chemicals or organic treatments, such as fertilizers or pesticides, and guidance on when and how to apply them. The system can also suggest remedies if the disease is preventable or treatable with non-chemical methods.

5. User Feedback (Optional)

- **Requirement:** Users should be able to rate the accuracy or usefulness of the disease detection and treatment recommendations.
- **Details:** An optional feature where users can provide feedback on the accuracy of the disease detection and the helpfulness of the treatment suggestions. This feedback will help improve the system over time and ensure that the recommendations are relevant and effective. Users can rate the system's suggestions on a scale (e.g., 1-5 stars) and optionally provide comments.

6. Admin Panel (Optional)

- **Requirement:** The system should provide an administrative interface for managing disease categories, updating treatment guidelines, and overseeing system operations.
- **Details:** The admin panel will allow the system administrators to add, update, or delete disease categories and treatment guidelines. They can also monitor the system's activity, such as tracking the number of users, feedback submissions, and the status of the model's predictions. Admins will be able to maintain and update the system's database as needed.

Non-Functional Requirements

Non-functional requirements define how well the system should perform and the conditions under which it should operate. These factors are crucial for ensuring that the system is reliable, efficient, and scalable. Below are the non-functional requirements for the **Plant Disease Detection** project:

---

1. Performance

- **Requirement:** The system should return results within 5 seconds after an image is uploaded.
- **Details:** The system must process the uploaded image and provide disease detection results quickly. This ensures a seamless user experience, especially for users who may not have the patience to wait for long processing times. The performance of the backend model should be optimized to return predictions in under 5 seconds, ensuring minimal delay for users.

2. Reliability

- **Requirement:** The system should have minimal downtime and function consistently.
- **Details:** The system must be highly reliable and operate without unexpected crashes or errors. It should be available for users 24/7, with minimal maintenance downtime. Regular updates and checks should be performed to ensure smooth functioning, especially for machine learning model predictions. The backend should be well-tested to handle any errors or exceptions gracefully.

3. Security

- **Requirement:** All uploaded images and user information should be securely stored and handled.
- **Details:** The system must ensure the privacy and security of user data, especially the uploaded plant leaf images, which might contain sensitive agricultural data. Secure

---

methods like **HTTPS**, **encryption**, and **access controls** should be implemented to protect user information from unauthorized access. Additionally, the system should comply with applicable data protection regulations.

4. Scalability

- **Requirement:** The system should be able to handle increased user load as the number of users grows over time.
- **Details:** The backend architecture should be scalable, meaning that as the user base increases, the system can handle more traffic and requests without performance degradation. The cloud infrastructure or server setup should allow for easy scaling, including load balancing and adding additional resources as required.

5. Usability

- **Requirement:** The interface must be intuitive and accessible, even to users with basic technical skills.
- **Details:** The system should be easy to use and intuitive, providing clear instructions and user-friendly navigation. The design should ensure that even users with minimal technical expertise can understand how to upload images, view predictions, and follow treatment recommendations without difficulty. The system should provide helpful prompts and feedback to guide users through the process.

<u>Hardware Requirements</u>

The hardware requirements define the minimum system specifications needed for deploying and running the project. These include the local testing and production environments to ensure smooth operation of the system:

---

1. Processor

- **Requirement:** Intel i5 or AMD equivalent.
- **Details:** The system should be able to run the backend services and machine learning model effectively. A mid-range processor like Intel i5 or its AMD equivalent is sufficient for local testing. However, for larger-scale production environments, more powerful processors may be required to ensure efficient handling of multiple requests simultaneously.

2. RAM

- **Requirement:** At least 8 GB of RAM.
- **Details:** The system needs sufficient memory to run backend services and process machine learning models effectively. At least 8 GB of RAM will ensure that the model runs efficiently without slowdowns or memory-related issues.

3. Storage

- **Requirement:** Minimum 50 GB of free storage.
- **Details:** The system needs sufficient disk space to store user-uploaded images, logs, and database files. The storage should be scalable to accommodate a growing number of images and system activity data over time.

---

4. GPU (Optional)

- **Requirement:** NVIDIA GPU with CUDA support (for model training or retraining).
- **Details:** While not mandatory for deploying the model, a **GPU** can significantly speed up the **training** process of the machine learning model. If the model needs to be retrained frequently, using a GPU with CUDA support (like an NVIDIA card) can drastically reduce the time taken to train the model on large datasets.

DATA OBJECT DATA TYPE

1. User Data

This includes information about the users who upload plant leaf images and receive disease predictions.

Data Object: User

- **user_id** (Integer/Primary Key): Unique identifier for each user.
- **username** (String): The user's login name.
- **email** (String): The user's email address.
- **password_hash** (String): The hashed password for secure authentication.
- **feedback** (Integer/Optional): User feedback score on disease detection accuracy (1 to 5 stars).
- **created_at** (Datetime): Timestamp when the user registered.

2. Image Data

This includes the information regarding the uploaded plant leaf images.

Data Object: Image

- **image_id** (Integer/Primary Key): Unique identifier for each image.
- **user_id** (Integer/Foreign Key): User who uploaded the image.
- **image_file** (String): Path or URL to the uploaded image file.
- **upload_timestamp** (Datetime): The time when the image was uploaded.
- **image_format** (String): The format of the image (JPEG/PNG).
- **image_size** (Integer): The file size of the uploaded image (in KB or MB).

3. Disease Data

This includes the disease prediction results returned by the system.

Data Object: DiseasePrediction

- **prediction_id** (Integer/Primary Key): Unique identifier for each disease prediction.
- **image_id** (Integer/Foreign Key): The image associated with the prediction.
- **predicted_disease** (String): The name of the detected disease (e.g., blight, rust, mildew).
- **confidence_score** (Float): The confidence level (0.0 to 1.0) of the prediction.
- **prediction_timestamp** (Datetime): The time when the prediction was made.

4. Treatment Data

This includes the treatment recommendations based on the detected diseases.

Data Object: TreatmentRecommendation

- **recommendation_id** (Integer/Primary Key): Unique identifier for each treatment recommendation.

- **prediction_id** (Integer/Foreign Key): The prediction related to this treatment.
- **disease_name** (String): The name of the detected disease.
- **treatment_type** (String): Type of treatment (e.g., chemical, organic, or natural).
- **treatment_details** (String): Detailed description of the suggested treatment (e.g., pesticides, fertilizers, watering tips).
- **recommended_by** (String/Optional): Admin or system-based recommendation.

5. Feedback Data

This includes feedback provided by the users regarding the predictions and treatment recommendations.

Data Object: UserFeedback

- **feedback_id** (Integer/Primary Key): Unique identifier for each feedback entry.
- **user_id** (Integer/Foreign Key): User who provided the feedback.
- **prediction_id** (Integer/Foreign Key): The prediction for which feedback is provided.
- **rating** (Integer): Rating for the accuracy of disease prediction (1 to 5 stars).
- **comments** (String/Optional): Additional comments provided by the user.
- **feedback_timestamp** (Datetime): The time when the feedback was submitted.

6. Admin Data (for optional admin panel functionality)

This includes information and actions that can be managed by system administrators.

Data Object: Admin

- **admin_id** (Integer/Primary Key): Unique identifier for each admin.
- **username** (String): Admin's username.
- **email** (String): Admin's email address.
- **password_hash** (String): The hashed password for admin authentication.
- **role** (String): The role of the admin (e.g., super-admin, content manager).
- **last_login** (Datetime): Timestamp of the last admin login.

7. Disease Categories Data (Optional for Admin)

This includes disease categories managed by admins.

Data Object: DiseaseCategory

- **category_id** (Integer/Primary Key): Unique identifier for each disease category.
- **category_name** (String): Name of the disease category (e.g., fungal, bacterial).
- **category_description** (String): Description of the category and its common diseases.

---

8. Database Table Representations

In SQL, the above objects would translate to tables with specific data types. Here’s a possible SQL representation:

User Table

sql
CopyEdit
```
CREATE TABLE User (
  user_id INT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(50) NOT NULL,
  email VARCHAR(100) NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  feedback INT DEFAULT NULL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

-

Image Table

sql
CopyEdit
```
CREATE TABLE Image (
  image_id INT PRIMARY KEY AUTO_INCREMENT,
  user_id INT,
  image_file VARCHAR(255) NOT NULL,
  upload_timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
  image_format VARCHAR(10),
  image_size INT,
  FOREIGN KEY (user_id) REFERENCES User(user_id)
);
```

DiseasePrediction Table

sql
CopyEdit
```
CREATE TABLE DiseasePrediction (
  prediction_id INT PRIMARY KEY AUTO_INCREMENT,
  image_id INT,
  predicted_disease VARCHAR(100),
  confidence_score FLOAT,
  prediction_timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (image_id) REFERENCES Image(image_id)
);
```

-

TreatmentRecommendation Table

sql
CopyEdit

```
CREATE TABLE TreatmentRecommendation (
  recommendation_id INT PRIMARY KEY AUTO_INCREMENT,
  prediction_id INT,
  disease_name VARCHAR(100),
  treatment_type VARCHAR(50),
  treatment_details TEXT,
  recommended_by VARCHAR(100),
  FOREIGN KEY (prediction_id) REFERENCES
DiseasePrediction(prediction_id)
);
```

-

UserFeedback Table

sql
CopyEdit

```
CREATE TABLE UserFeedback (
  feedback_id INT PRIMARY KEY AUTO_INCREMENT,
  user_id INT,
```

```
 prediction_id INT,
  rating INT,
  comments TEXT,
  feedback_timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES User(user_id),
  FOREIGN KEY (prediction_id) REFERENCES
DiseasePrediction(prediction_id)
);
```

-

Admin Table

sql
CopyEdit

```
CREATE TABLE Admin (
  admin_id INT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(50) NOT NULL,
  email VARCHAR(100) NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  role VARCHAR(50) DEFAULT 'admin',
  last_login DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

-

DiseaseCategory Table

sql
CopyEdit

```
CREATE TABLE DiseaseCategory (
  category_id INT PRIMARY KEY AUTO_INCREMENT,
  category_name VARCHAR(100),
  category_description TEXT
```

- `);`

FRONTEND CODE (React)

DASHBOARD

```
import React, { useState, useRef } from 'react';
import farhandon from '../../images/dashimage2.png';
import image6 from '../../images/dashimage6.png';
import image7 from '../../images/dashimage7.png';
import image8 from '../../images/dashimage8.png';
import Garden1 from '../../images/garden1.jpg';
import Garden2 from '../../images/garden2.jpg';
import Garden3 from '../../images/garden3.jpg';


const DashboardSection = () => {
  const [selectedImage, setSelectedImage] = useState(null);
  const fileInputRef = useRef(null);

  // Handle file selection for the "Scan the plant" button
  const handleScanClick = () => {
    fileInputRef.current.click(); // Trigger file input click to open
camera/gallery
  };

  const handleImageUpload = (e) => {
    const file = e.target.files[0];
    if (file) {
      const reader = new FileReader();
      reader.onloadend = () => {
        setSelectedImage(reader.result); // Set the selected image to
state
      };
      reader.readAsDataURL(file); // Read the selected image
    }
  };

  return (
    <>
      {/* Dashboard Section */}
      <section className="relative w-full h-[650px] bg-cover bg-center
text-white flex items-center pl-[120px]" style={{ backgroundImage:
"url('../../src/images/dashimage1.png')" }}>
        <div className="absolute top-0 left-0 w-full h-full bg-black
opacity-50"></div>
        <div className="relative z-10 max-w-[700px]">
          <h1 className="text-[56px] font-bold mb-[30px]">Welcome to My
Garden!</h1>
          <p className="text-[28px] leading-8">
            Plants are essential to life on Earth. <br />
            They provide oxygen, food, and <br />
            help maintain ecological balance.
          </p>
          <button onClick={() => window.location.href = '/FullLibrary'}
className="bg-green-500 text-white px-[20px] py-[10px] rounded-[25px]
font-bold hover:bg-green-600 transition duration-300 mt-[20px]">
            🌿 Explore My Library
          </button>
        </div>
      </section>

      {/* About Section */}
      <section className="flex items-center justify-center p-[50px]
bg-white gap-[40px]">
        <img src={farhandon} alt="About Us" className="w-[40%]
rounded-[10px] border-[5px] border-green-500" />
        <div className="flex-1 text-left">
          <h2 className="text-[32px] font-bold text-green-500">About
Us</h2>
          <p className="text-[16px] leading-6">Bringing nature and
technology together!</p>
          <h3 className="text-[24px] font-bold text-black mt-[20px]">Who
We Are?</h3>
          <p className="text-[16px] leading-6">
            We are a smart plant identification and analysis platform that
helps you scan
            any plant and instantly get detailed information about it.
Whether you don't
            know the name of a plant or need insights about its diseases
and benefits,
            our AI-powered technology is here to assist you.
          </p>
        </div>
      </section>

      {/* Explore Section */}
      <section className="flex justify-between items-center bg-green-500
text-white p-[20px] px-[50px]">
        <div className="max-w-[60%]">
          <h2 className="text-[28px] font-bold">Check out the most
searched and rarest plants from around the world.</h2>
          <p className="text-[20px]">Learn fascinating facts and discover
new species!</p>
        </div>
        <button onClick={handleScanClick} className="bg-white
text-green-500 border-none px-[20px] py-[10px] text-[18px] font-bold
rounded-[25px] cursor-pointer flex items-center">
          Scan the plant 📷
        </button>
        {/* Hidden file input for selecting an image */}
        <input
          ref={fileInputRef}
          type="file"
          accept="image/*"
          capture="environment" // This helps on mobile devices to open
the camera by default
          style={{ display: 'none' }} // Hide the input
          onChange={handleImageUpload}
        />
      </section>

      {/* Image Preview */}
      {selectedImage && (
        <div className="flex justify-center mt-[20px]">
          <img src={selectedImage} alt="Selected Plant"
className="w-[300px] h-[250px] object-cover rounded-[10px]" />
        </div>
      )}

      {/* Plant Cards Section */}
      <section className="text-center p-[50px]">
        <h2 className="text-[36px] font-bold text-green-500">Explore</h2>
        <p className="text-[20px] mb-[20px]">Start Exploring Today! Scan,
Learn, and Grow with Us!</p>
        <div className="flex flex-wrap justify-center gap-[20px]">
          <div className="bg-white shadow-lg rounded-[10px]
overflow-hidden w-[300px] text-center pb-[20px]">
            <img src={image6} alt="Corpse Flower" className="w-full
h-[200px] object-cover" />
            <p className="p-[15px] text-[16px]">The Corpse Flower blooms
once every 7-10 years and is known for its pungent odor resembling
rotting.</p>
            <button onClick={() => window.location.href = '/FullLibrary'}
className="bg-white text-green-500 border-[2px] border-green-500 px-[15px]
py-[10px] text-[16px] font-bold rounded-[5px] cursor-pointer
transition-all ease-in-out duration-300 hover:bg-green-500
hover:text-white">
              Learn More
            </button>
          </div>
          <div className="bg-white shadow-lg rounded-[10px]
overflow-hidden w-[300px] text-center pb-[20px]">
            <img src={image7} alt="Ghost Orchid" className="w-full
h-[200px] object-cover" />
            <p className="p-[15px] text-[16px]">This rare orchid grows in
swampy forests and is incredibly difficult to cultivate outside its
natural habitat.</p>
            <button onClick={() => window.location.href = '/FullLibrary'}
className="bg-white text-green-500 border-[2px] border-green-500 px-[15px]
py-[10px] text-[16px] font-bold rounded-[5px] cursor-pointer
transition-all ease-in-out duration-300 hover:bg-green-500
hover:text-white">
              Learn More
            </button>
          </div>
          <div className="bg-white shadow-lg rounded-[10px]
overflow-hidden w-[300px] text-center pb-[20px]">
            <img src={image8} alt="Jade Vine" className="w-full h-[200px]
object-cover" />
```

```
            <p className="p-[15px] text-[16px]">Jade Vines display
mesmerizing turquoise petals and are pollinated by bats in the wild.</p>
            <button onClick={() => window.location.href = '/FullLibrary'}
className="bg-white text-green-500 border-[2px] border-green-500 px-[15px]
py-[10px] text-[16px] font-bold rounded-[5px] cursor-pointer
transition-all ease-in-out duration-300 hover:bg-green-500
hover:text-white">
              Learn More
            </button>
          </div>
        </div>
        <button onClick={() => window.location.href = '/FullLibrary'}
className="mt-[30px] bg-green-500 text-white px-[30px] py-[12px]
text-[18px] font-bold rounded-[25px] hover:bg-green-600 transition
duration-300">
          🌱 Explore My Library
        </button>
      </section>

      {/* My Garden Section */}
      <section className="text-center p-[50px]">
        <h2 className="text-[36px] font-bold text-green-500">My
Garden</h2>
        <p className="text-[20px] mb-[20px]">Your personal plant
collection, all in one place! Manage, track, and care for your plants
effortlessly.</p>
        <div className="flex justify-center gap-[20px]">
          <div className="text-center">
            <img src={Garden1} alt="Plant 1" className="w-[300px]
h-[250px] object-cover rounded-[10px]" />
          </div>
          <div className="text-center">
            <img src={Garden2} alt="Plant 2" className="w-[300px]
h-[250px] object-cover rounded-[10px]" />
          </div>
          <div className="text-center">
            <img src={Garden3} alt="Plant 3" className="w-[300px]
h-[250px] object-cover rounded-[10px]" />
          </div>
        </div>
        <button onClick={() => window.location.href = '/FullLibrary'}
className="mt-[30px] bg-green-500 text-white px-[30px] py-[12px]
text-[18px] font-bold rounded-[25px] hover:bg-green-600 transition
duration-300">
          🌱 Explore My Library
        </button>
      </section>
    </>
  );
};

export default DashboardSection;
```

DETECTION PLANT

```
import React, { useEffect, useState } from 'react';

const DetectionPlant = () => {
  const [detectionResult, setDetectionResult] = useState(null);

  useEffect(() => {
    const storedResult = localStorage.getItem('detectionResult');
    if (storedResult) {
      try {
        const parsed = JSON.parse(storedResult);
        setDetectionResult(parsed);
        console.log('Detection result loaded from localStorage:', parsed);
      } catch (error) {
        console.error('❌ Error parsing detection result:', error);
        alert('Invalid detection result format.');
      }
    } else {
      alert('❌ No detection result found in localStorage.');
    }
  }, []);

  if (!detectionResult) {
    return (
      <div className="p-4 text-lg">
        <p>Loading detection result...</p>
      </div>
    );
  }

  return (
    <div className="p-6">
      <h1 className="text-4xl font-bold mb-6 text-green-800">🌿 Disease
Detected</h1>

      <div className="mb-6">
        <img
          src={detectionResult.image}
          alt={detectionResult.disease_name || 'Detected Plant'}
          className="rounded-xl max-w-full h-auto mx-auto shadow-2xl
border-4 border-green-200"
        />
      </div>

      <div className="mb-6 text-xl">
        <p className="mb-2"><strong>Disease Name:</strong>
{detectionResult.disease_name}</p>
        <p className="mb-2"><strong>Plant Name:</strong>
{detectionResult.plant_name}</p>
        <p className="mb-2"><strong>Match Score:</strong>
{detectionResult.match_score.toFixed(2)}</p>
      </div>

      <div className="mb-6 text-xl">
        <h2 className="text-2xl font-semibold text-red-700 mb-2">🩺
Symptoms:</h2>
        <p>{detectionResult.symptoms}</p>
      </div>

      <div className="mb-6 text-xl">
        <h2 className="text-2xl font-semibold text-yellow-700 mb-2">🌱
Causes:</h2>
        <p>{detectionResult.causes}</p>
      </div>

      <div className="mb-6 text-xl">
        <h2 className="text-2xl font-semibold text-blue-700 mb-2">🧪
Solution:</h2>
        <pre className="whitespace-pre-wrap bg-blue-50 p-4 rounded-xl
border border-blue-200 shadow-inner">
          {detectionResult.solution}
        </pre>
      </div>
    </div>
  );
};

export default DetectionPlant;
```

HEADER

```
import React, { useRef, useState } from 'react';
import { Link } from 'react-router-dom';
import axios from 'axios';
import imagelogo from '../../images/logo.png';

const Header = () => {
  const fileInputRef = useRef(null);
  const [isScanning, setIsScanning] = useState(false);
  const [detectionResult, setDetectionResult] = useState(null); // Result
from API
  const [showModal, setShowModal] = useState(false); // Modal state

  const handleScanClick = () => {
    fileInputRef.current.click();
  };

  const handleImageUpload = async (e) => {
    const file = e.target.files[0];
    if (!file) return alert('Please select an image first.');
    setIsScanning(true);

    const formData = new FormData();
    formData.append('image', file);

    try {
      const response = await
axios.post('http://localhost:8000/user/detect/', formData, {
        headers: { 'Content-Type': 'multipart/form-data' },
      });

      const data = response.data;
      console.log('Detection Result:', data);

      if (data.disease_name) {
        setDetectionResult(data);
        setShowModal(true); // Show modal on success
      } else {
        alert('❌ No match found. Please try again.');
      }
    } catch (error) {
      console.error('Scan failed:', error);
      alert('⚠️ Failed to scan image. Please try again.');
    } finally {
      setIsScanning(false);
    }
  };

  return (
    <>
      <header className="flex items-center justify-between bg-white p-4
text-black shadow-md">
        {/* Logo */}
        <div className="flex items-center">
          <img src={imagelogo} alt="Logo" className="w-10 h-10 mr-3" />
          <h2 className="text-[#0D7903] font-bold text-lg">Plant Detection
System</h2>
        </div>

        {/* Center Button */}
        <div className="absolute left-1/2 transform -translate-x-1/2">
          <Link to="/FullLibrary">
            <button className="bg-[#0D7903] text-white px-5 py-2
rounded-full font-semibold shadow hover:bg-green-700 transition
duration-300">
              Explore My Library
            </button>
          </Link>
        </div>

        {/* Right Section */}
        <div className="flex gap-3">
          <Link to="/login">
            <button className="bg-white text-green-500 border-2
border-green-500 px-4 py-2 rounded-md font-bold cursor-pointer">
              Login
            </button>
          </Link>

          <button
            onClick={handleScanClick}
            className="bg-white text-[#0D7903] border-2 border-[#0D7903]
px-4 py-2 rounded-md font-bold cursor-pointer flex items-center"
            disabled={isScanning}
          >
            {isScanning ? 'Scanning...' : 'Scan the plant'} <span
className="ml-1">📷</span>
          </button>

          {/* Hidden input */}
          <input
            type="file"
            accept="image/*"
            ref={fileInputRef}
            onChange={handleImageUpload}
            style={{ display: 'none' }}
          />
        </div>
      </header>
      {/* ✅ Modal */}
      {showModal && detectionResult && (
  <div className="fixed inset-0 z-50 bg-black bg-opacity-40 flex
justify-center items-center">
    <div className="relative bg-white p-6 rounded-2xl shadow-xl max-w-2xl
w-full max-h-[80vh] overflow-y-auto">

      {/* ❌ Close button */}
      <button
        className="absolute top-2 right-2 text-gray-500 hover:text-red-600
text-2xl font-bold z-10"
        onClick={() => setShowModal(false)}
      >
        &times;
      </button>

      <h2 className="text-2xl font-bold mb-4 text-green-700
text-center">🌿 Disease Detected</h2>
      <img
        src={detectionResult.image}
        alt={detectionResult.disease_name}
        className="rounded-lg w-full h-auto mb-4 border"
      />

      <div className="space-y-2 text-gray-700 text-sm">
        <p><strong>Disease:</strong> {detectionResult.disease_name}</p>
        <p><strong>Plant:</strong> {detectionResult.plant_name}</p>
        <p><strong>Match Score:</strong>
{detectionResult.match_score.toFixed(2)}</p>
        <p><strong>Symptoms:</strong> {detectionResult.symptoms}</p>
        <p><strong>Causes:</strong> {detectionResult.causes}</p>
        <div>
          <p><strong>Solution:</strong></p>
          <pre className="bg-gray-100 p-3 rounded-md overflow-auto
whitespace-pre-wrap text-sm">{detectionResult.solution}</pre>
        </div>
      </div>

      <div className="mt-6 flex justify-center">
        <button
```

```
          className="bg-red-500 text-white px-6 py-2 rounded-md
hover:bg-red-600 transition duration-200"
          onClick={() => setShowModal(false)}
        >
          Close
        </button>
      </div>
    </div>
  </div>
)}


    </>
  );
};

export default Header;
```

FOOTER

```
import React from 'react';
import logo from '../../images/logo.png';

const Footer = () => {
  const styles = {
    footer: {
      backgroundColor: 'green',
      color: 'white',
      display: 'flex',
      flexDirection: 'column',
      alignItems: 'center',
      padding: '30px',
    },
    topSection: {
      display: 'flex',
      justifyContent: 'space-between',
      alignItems: 'center',
      width: '100%',
    },
    leftSection: {
      width: '30%',
    },
    logo: {
      display: 'flex',
      alignItems: 'center',
      marginBottom: '10px',
    },
    logoImage: {
      width: '50px',
      height: '50px',
      marginRight: '10px',
      filter: 'none',
    },
    centerSection: {
      width: '40%',
      textAlign: 'center',
    },
    contactItem: {
      display: 'flex',
      alignItems: 'center',
      marginBottom: '5px',
    },
    icon: {
      marginRight: '10px',
    },
    rightSection: {
      width: '20%',
      textAlign: 'right',
    },
    verticalLine: {
      height: '100px',
      width: '2px',
      backgroundColor: 'white',
      margin: '0 20px',
    },
    quickLinks: {
      listStyleType: 'none',
      padding: 0,
    },
    quickLink: {
      marginBottom: '5px',
      cursor: 'pointer',
    },
    bottomText: {
      marginTop: '20px',
      fontSize: '14px',
      textAlign: 'center',
    },
  };

  return (
    <footer style={styles.footer}>
      <div style={styles.topSection}>
        <div style={styles.leftSection}>
          <div style={styles.logo}>
            <img src={logo} alt="Logo" style={styles.logoImage} />
            <h2>Plant Detection System</h2>
          </div>
          <p>Plants are essential to life on Earth.<br/>They provide
oxygen, food, and help maintain ecological balance.</p>
        </div>

        <div style={styles.verticalLine}></div>

        <div style={styles.centerSection}>
          <h3>Contact Us</h3>
          <div style={styles.contactItem}>
            <span style={styles.icon}>📞</span>
            <span>9696389966, 7237070470, 8004000321</span>
          </div>
          <div style={styles.contactItem}>
            <span style={styles.icon}>✉️</span>
            <span>gorakhpur3221@gmail.com</span>
          </div>
          <div style={styles.contactItem}>
            <span style={styles.icon}>📍</span>
            <span>Near SBI Regional Office, Tarmandal, Gorakhpur</span>
          </div>
        </div>

        <div style={styles.verticalLine}></div>

        <div style={styles.rightSection}>
          <h3>My team </h3>
          <ul style={styles.quickLinks}>
            <li style={styles.quickLink}>Humaira Aziz Siddiqui</li>
            <li style={styles.quickLink}>Farhan Ahmad</li>
            <li style={styles.quickLink}>Nasim Khan</li>
            <li style={styles.quickLink}>Shridhi Yadav</li>
            <li style={styles.quickLink}>Shivani Gupta</li>
          </ul>
        </div>
      </div>

      <div style={styles.bottomText}>
        @ Powered by TNS
      </div>
    </footer>
  );
};

export default Footer;
```

LOGIN

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom"; // Import useNavigate for
navigation
import BaseUrl from "../../api/BaseUrl"; // Import the BaseUrl for API
calls
import loginpage from "../../images/loginpage.png"; // Ensure the image
path is correct

const Login = () => {
  const navigate = useNavigate(); // Use useNavigate for navigation
  const [mobileNumber, setMobileNumber] = useState(""); // Track mobile
number input
  const [otp, setOtp] = useState(new Array(6).fill("")); // OTP input
state (6-digit OTP)
  const [isOtpSent, setIsOtpSent] = useState(false); // Track if OTP has
been sent
  const [confirmationCode, setConfirmationCode] = useState(""); //
Confirmation code state

  // Function to send OTP (GET request)
  const sendOtp = async () => {
    if (!mobileNumber) {
      alert("Please enter a valid mobile number.");
      return;
    }

    try {
      const response = await
BaseUrl.get(`/user/login/?mobile_number=${mobileNumber}`, {
        headers: {
          Cookie: "csrftoken=kokqfiPPrzFLvJPlnYD3tzAGg8fvjXgE", // CSRF
token
        },
      });

      if (response.status === 200 && response.data.message === "OTP sent
successfully") {
        setIsOtpSent(true); // Mark OTP as sent
        console.log("OTP Sent Response:", response.data);
      } else {
        alert("Failed to send OTP: " + response.data.message);
      }
    } catch (error) {
      console.error("Error sending OTP:", error);
      alert("Error sending OTP, please try again.");
    }
  };

  // Function to handle OTP change
  const handleOtpChange = (target, index) => {
    const value = target.value;

    // Update OTP array by setting value at the specific index
    setOtp((prevOtp) => {
      const newOtp = [...prevOtp];
      newOtp[index] = value.slice(0, 1); // Ensure only one character is
entered
      return newOtp;
    });

    // Move focus to the next input field if a digit is entered
    if (value && index < otp.length - 1) {
      const nextInput = document.querySelectorAll('input')[index + 1];
      nextInput?.focus();
    }
  };

  // Function to confirm login (POST request)
  const handleSubmitOtp = async () => {
    const otpValue = confirmationCode; // Use confirmationCode directly

    // Check if the OTP value is empty before making the API request
    if (!otpValue) {
      alert("Please enter a valid OTP.");
      return;
    }

    try {
      const response = await BaseUrl.post(
        "/user/login/",  // Your login endpoint
        {
          mobile_number: mobileNumber,  // Mobile number from state
          otp: otpValue,  // Pass the OTP directly from confirmationCode
state
        },
        {
          headers: {
            "Content-Type": "application/json",  // Ensure correct content
type is set
```

```
            Cookie: "csrftoken=kokqfiPPrzFLvJPlnYD3tzAGg8fvjXgE", // CSRF
token if needed
          },
        }
      );

      // Check if the response status is 201 (successful login)
      if (response.status === 200) {
        console.log("Login Success:", response.data);

        // Store token and login status in localStorage
        const token = response.data.access_token;
        localStorage.setItem("access_token", token);
        localStorage.setItem("userLoggedIn", "true");

        // Navigate to the home page after login
        navigate("/");
      } else {
        alert("Failed to confirm OTP: " + response.data.message);
      }
    } catch (error) {
      console.error("Error verifying OTP:", error);
      alert("Error verifying OTP, please try again.");
    }
  };



  return (
    <div className="w-full h-screen flex items-center justify-center
bg-gray-200">
      <div className="w-full max-w-[1100px] h-[600px] flex items-center
justify-center">
        <div className="flex w-full h-full bg-white rounded-[20px]
shadow-xl overflow-hidden">
          {/* Left Side: Login Form */}
          <div className="w-[55%] p-[50px] flex flex-col justify-center
items-center bg-white">
            <h2 className="text-[28px] font-bold mb-[25px]">Login</h2>
            {/* Mobile Number Input */}
            <div className="flex items-center border border-gray-300
rounded-[8px] p-[12px] mb-[18px] w-full">
              <span className="mr-[12px] text-[20px]
text-[#4caf50]">📱</span>
              <input
                type="text"
                placeholder="Enter the Mobile Number"
                className="w-full border-none outline-none p-[12px]
text-[18px]"
                value={mobileNumber}
                onChange={(e) => setMobileNumber(e.target.value)}
                disabled={isOtpSent} // Disable mobile number input if OTP
is sent
              />
            </div>

            {/* Confirmation Code Input */}
            <div className="flex items-center border border-gray-300
rounded-[8px] p-[12px] mb-[18px] w-full">
              <span className="mr-[12px] text-[20px]
text-[#4caf50]">🔑</span>
              <input
                type="text"
                placeholder="Confirmation Code"
                className="w-full border-none outline-none p-[12px]
text-[18px]"
                value={confirmationCode}
                onChange={(e) => setConfirmationCode(e.target.value)}
                disabled={!isOtpSent} // Disable confirmation code field
until OTP is sent
              />
            </div>

            {/* Resend Code */}
            <p className="text-[16px] mt-[8px]">
              Lost your code?{" "}
              <span className="text-[#4caf50] cursor-pointer"
onClick={sendOtp}>
```

```
                Resend Code
              </span>
            </p>

            {/* Confirm Button */}
            <button
  className="w-full bg-[#4caf50] text-white py-[14px] rounded-[8px]
text-[18px] font-bold cursor-pointer mt-[12px] hover:bg-[#388e3c]"
  onClick={isOtpSent ? handleSubmitOtp : sendOtp} // Trigger
handleSubmitOtp if OTP is sent, else sendOtp
  disabled={!mobileNumber || (isOtpSent && !confirmationCode)} // Disable
button until mobile number is entered or OTP is sent but confirmation code
is missing
>
  {isOtpSent ? "Confirm" : "Send OTP"} {/* Conditional button text */}
</button>



            {/* Divider */}
            <div className="text-center mt-[18px] text-[16px]
text-[#888]">Or</div>

            {/* Create Account Button */}
            <button
              className="w-full border-2 border-[#4caf50] text-[#4caf50]
bg-white py-[14px] rounded-[8px] cursor-pointer text-[18px] font-bold
mt-[12px] hover:bg-[#e8f5e9]"
              onClick={() => navigate("/register")}
            >
              Create An Account
            </button>
          </div>

          {/* Right Side: Image */}
          <div className="w-[45%] flex items-center justify-center
overflow-hidden">
            <img
              src={loginpage}
              alt="Plant"
              className="w-full h-full object-cover rounded-r-[20px]"
            />
          </div>
        </div>
      </div>
    </div>
  );
};

export default Login;
```

REGISTER

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom"; // Import useNavigate for
navigation in React Router v6
import BaseUrl from "../../api/BaseUrl"; // Import the BaseUrl for API
calls
import registerImage from "../../images/registerpage.png"; // Ensure the
image path is correct

const Register = () => {
  const navigate = useNavigate();
  const [name, setName] = useState("");
  const [mobileNumber, setMobileNumber] = useState("");
  const [confirmationCode, setConfirmationCode] = useState("");
  const [isOtpSent, setIsOtpSent] = useState(false); // Track if OTP has
been sent
  const [otpReceived, setOtpReceived] = useState(""); // Store OTP for
validation (optional)

  // Function to send OTP (GET request)
  const sendOtp = async () => {
    if (!mobileNumber) {
      alert("Please enter a valid mobile number.");
      return;
    }
    try {
      const response = await
BaseUrl.get(`/user/register/?mobile_number=${mobileNumber}`, {
        headers: {
          Cookie: "csrftoken=kokqfiPPrzFLvJPlnYD3tzAGg8fvjXgE", // CSRF
token
        },
      });

      if (response.status === 200 && response.data.message === "OTP sent
successfully") {
        setIsOtpSent(true); // Mark OTP as sent
        setOtpReceived(response.data.otp); // Optionally store OTP (if
needed)
        console.log("OTP Sent Response:", response.data);
      } else {
        alert("Failed to send OTP: " + response.data.message);
      }
    } catch (error) {
      console.error("Error sending OTP:", error);
      alert("Error sending OTP, please try again.");
    }
  };

  // Function to confirm registration (POST request)
 // Function to confirm registration (POST request)
// Function to confirm registration (POST request)
const confirmRegistration = async () => {
  if (!confirmationCode) {
    alert("Please enter the confirmation code.");
    return;
  }

  try {
    const response = await BaseUrl.post("/user/register/", {
      name: name,
      mobile_number: mobileNumber,
      otp: confirmationCode,
    });
    console.log("Registration Response:", response.data);

    // Check if registration is successful (status code 201)
    if (response.status === 201 && response.data.message === "User
registered successfully") {
      // Redirect to the login page on successful registration
      navigate("/login");
    } else {
      alert("Failed to confirm registration: " + response.data.message);
    }
  } catch (error) {
    console.error("Error confirming registration:", error);
    alert("Error confirming registration, please try again.");
  }
};



  return (
    <div className="w-full h-screen flex items-center justify-center
bg-gray-200">
      <div className="w-full max-w-[1100px] h-[600px] flex items-center
justify-center">
        <div className="flex w-full h-full bg-white rounded-[20px]
shadow-xl overflow-hidden">
          {/* Left Side: Register Form */}
          <div className="w-[55%] p-[50px] flex flex-col justify-center
items-center bg-white">
            <h2 className="text-[28px] font-bold
mb-[25px]">REGISTRATION</h2>

            {/* Name Group (First Name, Last Name) */}
            <div className="flex gap-[15px] w-full mb-[18px]">
              <input
                type="text"
                placeholder="First Name"
                className="w-[50%] p-[12px] border border-[#ccc]
rounded-[8px] text-[16px]"
                value={name}
                onChange={(e) => setName(e.target.value)}
              />
              <input
                type="text"
                placeholder="Last Name"
                className="w-[50%] p-[12px] border border-[#ccc]
rounded-[8px] text-[16px]"
              />
            </div>

            {/* Mobile Number Input */}
            <div className="flex items-center border border-[#ccc]
rounded-[8px] p-[12px] mb-[18px] w-full">
              <span className="mr-[12px] text-[20px]
text-[#4caf50]">📱</span>
              <input
                type="text"
                placeholder="Enter the Mobile Number"
                className="w-full border-none outline-none p-[12px]
text-[18px]"

                value={mobileNumber}
                onChange={(e) => setMobileNumber(e.target.value)}
              />
            </div>

            {/* Confirmation Code Input */}
            <div className="flex items-center border border-[#ccc]
rounded-[8px] p-[12px] mb-[18px] w-full">
              <span className="mr-[12px] text-[20px]
text-[#4caf50]">🔑</span>
              <input
                type="text"
                placeholder="Confirmation Code"
                className="w-full border-none outline-none p-[12px]
text-[18px]"

                value={confirmationCode}
                onChange={(e) => setConfirmationCode(e.target.value)}
                disabled={!isOtpSent} // Disable confirmation code field
until OTP is sent
              />
            </div>
            {/* Resend Code */}
            <p className="text-[16px] mt-[8px]">
              Lost your code?{" "}
              <span className="text-[#4caf50] cursor-pointer"
onClick={sendOtp}>
                Resend Code
              </span>
            </p>

            {/* Confirm Button */}
            <button
              className="w-full bg-[#4caf50] text-white py-[14px]
rounded-[8px] text-[18px] font-bold cursor-pointer mt-[12px]
hover:bg-[#388e3c]"
              onClick={isOtpSent ? confirmRegistration : sendOtp} // Call
sendOtp if OTP not sent, else confirmRegistration
              disabled={!mobileNumber || (isOtpSent && !confirmationCode)}
// Disable button until mobile number is entered or OTP is sent but
confirmation code is missing
            >
              {isOtpSent ? "Confirm" : "Send OTP"} {/* Conditional button
text */}
            </button>
          </div>

          {/* Right Side: Image */}
          <div className="w-[45%] flex items-center justify-center
overflow-hidden">
            <img
              src={registerImage}
              alt="Register"
              className="w-full h-full object-cover rounded-r-[20px]"
            />
          </div>
        </div>
      </div>

      {/* Already have an account? Login Link */}
      <p
```

```
        className="absolute top-[15px] right-[20px] text-[16px]
text-[#333] cursor-pointer"
        onClick={() => navigate("/login")}
      >
        Already have an account?{" "}
        <span className="text-[#4caf50] font-bold
hover:underline">Login</span>
      </p>
    </div>
  );
};

export default Register;
```

LIBRARY PAGE OF PLANTS

```
import React, { useState } from 'react';
import { Link } from 'react-router-dom'; // ✅ Link imported from
react-router-dom
import image1 from "../../images/Turmeric.jpg";
import image2 from "../../images/Tulsi.jpg";
import image3 from "../../images/Aloe Vera.jpg";
import image4 from "../../images/Peppermint.jpg";
import image5 from "../../images/Neem.jpg";
import image6 from "../../images/Moringa.jpg";
import image7 from "../../images/Licorice.jpg";
import image8 from "../../images/Lavender.jpg";
import image9 from "../../images/Hibiscus.jpg";
import image10 from "../../images/Ginger.jpg";
import image11 from "../../images/Giloy.jpg";
import image12 from "../../images/Garlic.jpg";
import image13 from "../../images/Fenugreek.jpg";
import image14 from "../../images/Clove.jpg";
import image15 from "../../images/Cinnamon.jpg";
import image16 from "../../images/Cardamom.jpg";
import image17 from "../../images/Brahmi.jpg";
import image18 from "../../images/Bael.jpg";
import image19 from "../../images/Ashwagandha.jpg";
import image20 from "../../images/Amla.jpg";

const allImages = [
  { id: 1, url: image1, name: "Turmeric" },
  { id: 2, url: image2, name: "Tulsi" },
  { id: 3, url: image3, name: "Aloe Vera" },
  { id: 4, url: image4, name: "Peppermint" },
  { id: 5, url: image5, name: "Neem" },
  { id: 6, url: image6, name: "Moringa" },
  { id: 7, url: image7, name: "Licorice" },
  { id: 8, url: image8, name: "Lavender" },
  { id: 9, url: image9, name: "Hibiscus" },
  { id: 10, url: image10, name: "Ginger" },
  { id: 11, url: image11, name: "Giloy" },
  { id: 12, url: image12, name: "Garlic" },
  { id: 13, url: image13, name: "Fenugreek" },
  { id: 14, url: image14, name: "Clove" },
  { id: 15, url: image15, name: "Cinnamon" },
  { id: 16, url: image16, name: "Cardamom" },
  { id: 17, url: image17, name: "Brahmi" },
  { id: 18, url: image18, name: "Bael" },
  { id: 19, url: image19, name: "Ashwagandha" },
  { id: 20, url: image20, name: "Amla" },
];

const PlantLibrary = () => {
  const [visibleImages, setVisibleImages] = useState(6);

  const loadMore = () => {
    setVisibleImages((prev) => prev + 6);
  };

  return (
    <div className="p-6">
      <h1 className="text-2xl font-bold text-center text-[#0D7903] mb-6">
        Plant Disease Library
      </h1>

      <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 
gap-6">
```

```
        {allImages.slice(0, visibleImages).map((img) => (
          <Link to={`/plant/${img.name.toLowerCase().replace(/\s/g,
"-")}`} key={img.id}>
            <div className="bg-white shadow-lg rounded-md overflow-hidden
cursor-pointer hover:shadow-xl transition">
              <img src={img.url} alt={img.name} className="w-full h-48
object-cover" />
              <div className="p-4 text-center
font-semibold">{img.name}</div>
            </div>
          </Link>
        ))}
      </div>

      {visibleImages < allImages.length && (
        <div className="flex justify-center mt-8">
          <button
            onClick={loadMore}
            className="bg-[#0D7903] text-white px-6 py-2 rounded-full
hover:bg-green-700 transition duration-300"
          >
            Load More
          </button>
        </div>
      )}
    </div>
  );
};

export default PlantLibrary;
```

App.jsx

```
import "./App.css";
import { BrowserRouter as Router, Routes, Route, Navigate } from
"react-router-dom";

// Import your components
```

```
import Dashboard from "./components/Dashboard/Dashboard";
import Header from "./components/header/Header";
import Footer from "./components/footer/Footer";
import Login from "./components/Login/Login";
import Register from "./components/Register/Register";
import Library from "./components/Library/FullLibrary";
import Turmaric from "./components/LibraryPages/Turmaric";
import Tulsi from "./components/LibraryPages/Tulsi";
import AloeVera from "./components/LibraryPages/AloeVera";
import Sidebar from "./components/Sidebar/Sidebar";
import Peppermint from "./components/LibraryPages/Peppermint";
import Neem from "./components/LibraryPages/Neem";
import Moringa from "./components/LibraryPages/Moringa";
import Licorice from "./components/LibraryPages/Licorice";
import Lavender from "./components/LibraryPages/Lavender";
import Hibiscus from "./components/LibraryPages/Hibiscus";
import Ginger from "./components/LibraryPages/Ginger";
import Giloy from "./components/LibraryPages/Giloy";
import Garlic from "./components/LibraryPages/Garlic";
import Fenugreek from "./components/LibraryPages/Fenugreek";
import Clove from "./components/LibraryPages/Clove";
import Cinnamon from "./components/LibraryPages/Cinnamon";
import Cardamom from "./components/LibraryPages/Cardamom";
import Brahmi from "./components/LibraryPages/Brahmi";
import Bael from "./components/LibraryPages/Bael";
import Ashwagandha from "./components/LibraryPages/Ashwagandha";
import Amla from "./components/LibraryPages/Amla";
import Detectionplant from "./components/DetectionPlant/Detactionplant";

function App() {
  return (
    <Router>
      <div className="app-layout">
        {/* Render Header and Footer only on pages other than Login and
Register */}
        <Routes>
          {/* Define the route for Login */}
          <Route path="/login" element={<Login />} />

          {/* Define the route for Register */}
```

```
          <Route path="/register" element={<Register />} />
          <Route path="/FullLibrary" element={<Library />} />
          <Route path="/Sidebar" element={<Sidebar />} />

          <Route path="/plant/turmeric" element={<Turmaric />} />
          <Route path="/plant/tulsi" element={<Tulsi />} />
          <Route path="/plant/aloe-vera" element={<AloeVera />} />
          <Route path="/plant/Peppermint" element={<Peppermint />} />
          <Route path="/plant/Neem" element={<Neem />} />
          <Route path="/plant/Moringa" element={<Moringa />} />
          <Route path="/plant/Licorice" element={<Licorice />} />
          <Route path="/plant/Lavender" element={<Lavender />} />
          <Route path="/plant/Hibiscus" element={<Hibiscus />} />
          <Route path="/plant/Ginger" element={<Ginger />} />
          <Route path="/plant/Giloy" element={<Giloy />} />
          <Route path="/plant/Garlic" element={<Garlic />} />
          <Route path="/plant/Fenugreek" element={<Fenugreek />} />
          <Route path="/plant/Clove" element={<Clove />} />
          <Route path="/plant/Cinnamon" element={<Cinnamon />} />
          <Route path="/plant/Cardamom" element={<Cardamom />} />
          <Route path="/plant/Brahmi" element={<Brahmi />} />
          <Route path="/plant/Bael" element={<Bael />} />
          <Route path="/plant/Ashwagandha" element={<Ashwagandha />} />
          <Route path="/plant/Amla" element={<Amla />} />
          <Route path="/Detactionplant" element={<Detectionplant />} />

          {/* Default route will redirect to Dashboard */}
          <Route path="/" element={<Navigate to="/dashboard" />} />


          {/* All other routes (Dashboard) will show Header and Footer */}
          <Route
            path="/dashboard"
            element={
              <>
                <Header />
                <Dashboard />
                <Footer />
              </>
            }
          />
        </Routes>
      </div>
    </Router>
  );
}

export default App;
```

BACKEND CODE (Django , MAchine Learning Models, My Sql)

Setting.py

```
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY =
'django-insecure-hym^f!uwscvjl04u4f6!=+ji8&xv1vlfkcqe%s5ps$#s)o4tzj'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ["*"]

# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'corsheaders',                  # ✅ Added for CORS
    'rest_framework',
    'user',
]

MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',              # ✅ CORS
middleware should be at the top
    'django.middleware.common.CommonMiddleware',          # ✅
CommonMiddleware just after corsheaders
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'plantdiseas.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'plantdiseas.wsgi.application'

# Database
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Password validation
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator'
,
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_TZ = True

# Static files
STATIC_URL = 'static/'

# Media files
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'

# Default primary key field type
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

```
# ✅ CORS Configuration
CORS_ALLOWED_ORIGINS = [
    "http://localhost:5175",  # React app (Vite) port
]

# ✅ Allow credentials if needed
CORS_ALLOW_CREDENTIALS = True
```

admin.py

```
from django.contrib import admin
from .models import UserRegistration, PlantDisease

# Register UserRegistration model
class UserRegistrationAdmin(admin.ModelAdmin):
    list_display = ('name', 'mobile_number')  # Fields to display in the
list view
    search_fields = ('name', 'mobile_number')  # Enable search
functionality by name and mobile number
    list_filter = ('name',)  # Add filter options based on name

admin.site.register(UserRegistration, UserRegistrationAdmin)

# Register PlantDisease model
class PlantDiseaseAdmin(admin.ModelAdmin):
    list_display = ('plant_name', 'disease_name', 'symptoms', 'causes',
'solution')  # Fields to display
    search_fields = ('plant_name', 'disease_name')  # Enable search
functionality by plant and disease name
    list_filter = ('plant_name', 'disease_name')  # Filter by plant name
and disease name

admin.site.register(PlantDisease, PlantDiseaseAdmin)
```

apps.py

```
from django.apps import AppConfig


class UserConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'user'
```

models.py

```
from django.db import models

class UserRegistration(models.Model):
    name=models.CharField(max_length=100)
    mobile_number=models.CharField(max_length=15)




class PlantDisease(models.Model):
    image = models.ImageField(upload_to='disease_images/')
    plant_name = models.CharField(max_length=100)
    disease_name = models.CharField(max_length=100)
    symptoms =models.CharField(max_length=100)
    causes = models.CharField(max_length=100)
    solution = models.CharField(max_length=100)
    embedding = models.BinaryField(help_text="Binary data representing the
image embedding.")  # Will store image feature vector     # Automatically
updated on save

    def __str__(self):
        return f"{self.disease_name} affecting {self.plant_name}"
```

utils.py

```python
from twilio.rest import Client
import random


def send_otp_via_twilio(phone_number, otp):
    account_sid = 'ACfb785182d1ed9fba6a26ed4722a0a11a'
    auth_token = '79ebd5f0700eac1658cfe9a352354066'
    twilio_number = '+19033453741'

    client = Client(account_sid, auth_token)

    message = client.messages.create(
        body=f"Your OTP is: {otp}",
        from_=twilio_number,
        to=phone_number
    )

    print(f"OTP sent to {phone_number}: {otp}")
    return otp



from torchvision import models, transforms
from PIL import Image
import torch
import io

# Load pretrained model
model = models.resnet18(pretrained=True)
model.eval()

# Transform input image
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

def get_image_embedding(image_file):
    img = Image.open(image_file).convert('RGB')
    img_tensor = transform(img).unsqueeze(0)
    with torch.no_grad():
        features = model(img_tensor)
    return features.squeeze().numpy()
```

views.py

```
import random
from django.core.cache import cache
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from rest_framework_simplejwt.tokens import RefreshToken
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
from .utils import get_image_embedding, send_otp_via_twilio
from .models import  PlantDisease, UserRegistration
import json
import pickle
from rest_framework.parsers import MultiPartParser



class UserRegistrationApi(APIView):

    def get(self, request):
        mobile_number = request.query_params.get('mobile_number')

        if not mobile_number:
            return Response({"error": "Mobile number is required"},
status=status.HTTP_400_BAD_REQUEST)

        otp = random.randint(100000, 999999)
        cache.set(f"user_otp_{mobile_number}", otp, timeout=300)
        formatted_number = f"+91{mobile_number}" # 5 min = 300 sec
        send_otp_via_twilio(formatted_number, otp)
```

```python
        return Response({
            "message": "OTP sent successfully",
            "otp": otp  # 🔁 NOTE: In production, don't return OTP in
response
        }, status=status.HTTP_200_OK)

    def post(self, request):
        name = request.data.get('name')
        mobile_number = request.data.get('mobile_number')
        otp = request.data.get('otp')

        if not all([name, mobile_number, otp]):
            return Response({"error": "Name, mobile number, and OTP are
required"}, status=status.HTTP_400_BAD_REQUEST)

        otp_stored = cache.get(f"user_otp_{mobile_number}")

        if not otp_stored:
            return Response({"error": "OTP expired or not requested"},
status=status.HTTP_400_BAD_REQUEST)

        if str(otp_stored) != str(otp):
            return Response({"error": "Invalid OTP"},
status=status.HTTP_400_BAD_REQUEST)

        # ✅ Save user
        user, created =
UserRegistration.objects.get_or_create(mobile_number=mobile_number,
defaults={'name': name})

        if not created:
            return Response({"message": "User already exists"},
status=status.HTTP_200_OK)

        cache.delete(f"user_otp_{mobile_number}")

        return Response({"message": "User registered successfully"},
status=status.HTTP_201_CREATED)
```

```
class UserLoginApi(APIView):

    def get(self, request):
        mobile_number = request.query_params.get('mobile_number')

        if not mobile_number:
            return Response({"error": "Mobile number is required"},
status=status.HTTP_400_BAD_REQUEST)

        try:
            user =
UserRegistration.objects.get(mobile_number=mobile_number)
        except UserRegistration.DoesNotExist:
            return Response({"error": "Mobile number not registered"},
status=status.HTTP_404_NOT_FOUND)

        otp = random.randint(100000, 999999)
        cache.set(f"otp_{mobile_number}", otp, timeout=300)  # 5 minutes
        formatted_number = f"+91{mobile_number}"
        send_otp_via_twilio(formatted_number, otp)

        return Response({"message": "OTP sent successfully", "otp": otp},
status=status.HTTP_200_OK)

    def post(self, request):
        mobile_number = request.data.get('mobile_number')
        otp = request.data.get('otp')

        if not mobile_number or not otp:
            return Response({"error": "Mobile number and OTP are
required"}, status=status.HTTP_400_BAD_REQUEST)

        otp_stored = cache.get(f"otp_{mobile_number}")
        if otp_stored and str(otp_stored) == str(otp):
            try:
                user =
UserRegistration.objects.get(mobile_number=mobile_number)
            except UserRegistration.DoesNotExist:
                return Response({"error": "User not found"},
status=status.HTTP_404_NOT_FOUND)

            cache.delete(f"otp_{mobile_number}")

            refresh = RefreshToken.for_user(user)
            refresh['mobile_number'] = user.mobile_number
            refresh['user_type'] = 'user'
            refresh['user_id'] = user.id

            return Response({
                "message": "Login successful",
                "token_type": "access",
                "access": str(refresh.access_token),
                "refresh": str(refresh),
                "mobile_number": user.mobile_number,
                "user_id": user.id,
            }, status=status.HTTP_200_OK)

        return Response({"error": "Invalid or expired OTP"},
status=status.HTTP_400_BAD_REQUEST)




from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.parsers import MultiPartParser
import pickle
from .models import PlantDisease
from .utils import get_image_embedding  # Assuming the function
get_image_embedding is defined in utils.py

from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.parsers import MultiPartParser
import pickle
from .models import PlantDisease
from .utils import get_image_embedding  # Assuming you have a function to
get image embeddings
```

```
class UploadDiseaseAPIView(APIView):
    parser_classes = [MultiPartParser]

    def post(self, request):
        # Extract the data from the request
        image = request.FILES.get('image')
        plant_name = request.data.get('plant_name')  # Plant name
        disease_name = request.data.get('disease_name')  # Disease name
        symptoms = request.data.get('symptoms')  # Symptoms of the disease
        causes = request.data.get('causes')  # Causes of the disease
        solution = request.data.get('solution')  # Solution for the
disease

        # Validate that all required fields are provided
        if not all([image, plant_name, disease_name, symptoms, causes,
solution]):
            return Response({'error': 'All fields (image, plant_name,
disease_name, symptoms, causes, solution) are required!'}, status=400)

        # Get image embedding
        embedding = get_image_embedding(image)

        # Store embedding as binary
        embedding_bytes = pickle.dumps(embedding)

        # Create the PlantDisease object with the provided data
        plant_disease = PlantDisease.objects.create(
            image=image,
            plant_name=plant_name,
            disease_name=disease_name,
            symptoms=symptoms,
            causes=causes,
            solution=solution,
            embedding=embedding_bytes
        )

        return Response({'status': 'Disease data saved successfully'})
```

```
class DetectDiseaseAPIView(APIView):
    parser_classes = [MultiPartParser]

    def post(self, request):
        # Get the uploaded image file
        image = request.FILES['image']
        input_embedding = get_image_embedding(image)

        # Retrieve all plant diseases from the database
        all_diseases = PlantDisease.objects.all()
        best_match = None
        highest_score = 0.0

        # Iterate through all diseases to find the best match based on
cosine similarity
        for disease in all_diseases:
            db_embedding = pickle.loads(disease.embedding)
            score = cosine_similarity([input_embedding],
[db_embedding])[0][0]

            # Debug: Log the similarity score for each disease
            print(f"Matching {disease.disease_name}: Score {score}")

            # Update the best match if a higher score is found
            if score > highest_score:
                highest_score = score
                best_match = disease

        # Debug: Log the final best match and its score
        print(f"Best Match: {best_match.disease_name if best_match else
'No match found'} with score {highest_score}")

        # Confidence threshold (adjust if necessary)
        if best_match and highest_score > 0.6:  # Confidence threshold
(adjust as needed)
            return Response({
                'image': request.build_absolute_uri(best_match.image.url),
# Full URL of the image
```

```
                'match_score': highest_score,
                'disease_name': best_match.disease_name,
                'plant_name': best_match.plant_name,
                'symptoms': best_match.symptoms,
                'causes': best_match.causes,
                'solution': best_match.solution
            })
        else:
            return Response({'message': 'No match found. Try again or
upload new data.'})
```

manage.py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'plantdiseas.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

Conclusion & Future Scope

Conclusion

The Plant Disease Detection System using machine learning provides an efficient, accurate, and user-friendly solution to a persistent problem in agriculture: the early detection and treatment of plant diseases. By leveraging deep learning models such as CNNs through libraries like TensorFlow, Keras, or PyTorch, this system can identify plant diseases from leaf images with high accuracy.

The web-based interface allows farmers or users to upload images and receive instant diagnosis and treatment recommendations, significantly reducing dependency on physical inspections and expert consultations. This not only saves time but also helps in preventing the spread of diseases at an early stage, which can otherwise lead to major crop losses.

Impact on Agriculture

This system can have a transformative effect on the agricultural sector, especially in rural or under-resourced areas:

- **Early Detection**: Helps farmers catch plant diseases early, leading to faster and more effective treatment.
- **Cost-Effective**: Reduces the need for expensive expert consultations and laboratory testing.
- **Knowledge Sharing**: Educates farmers about various diseases and available treatments.
- **Increased Yield**: Timely and accurate disease management contributes to healthier crops and higher yields.

Scalability & Future Enhancements

The system has been designed with scalability in mind and can be enhanced further in the following ways:

1. **Mobile App Integration**
   A mobile application can be developed to allow farmers to use the service even without a desktop or internet connection, enabling offline image analysis.

2. **Multilingual Support**
   Adding support for regional languages will increase accessibility for non-English-speaking users.

3. **Expanded Disease Database**
   The system can be continuously updated with new disease types, symptoms, and treatment options for various crops.

4. **Real-time Weather Integration**
   Weather data can be incorporated to suggest disease risks based on humidity, temperature, and rainfall.

5. **User Community and Feedback Loop**
   A community-driven platform can be developed where users can share their experiences, give feedback, and vote on the effectiveness of different treatments.

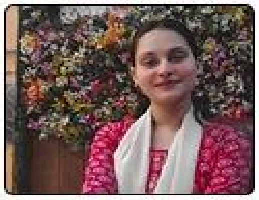
6. **AI Model Improvement**
   As more data is collected, the machine learning model can be retrained periodically to improve its accuracy and performance.

7. **API Support**
   The system can offer APIs for integration with government agricultural portals or third-party platforms.

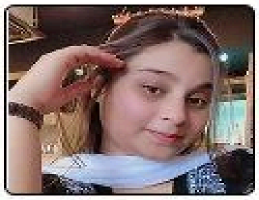
INTRODUCTION OF OUR TEAM

Nasim khan
Frontend developer

Farhan Ahmad
Frontend developer



Shivani
database handling, testing documentation , ppt and all page designer



Humaira siddiqui
Backend developer

Shriddhi yadav
Backend developer

PLANT DISEASE DETECTION

"VERY THANKFULL TO DEPARTMENT OF BCA & ISLAMIA COLLEGE OF COMMERCE GORAKHPUR ""