Technical Report

**Generate a detailed Word document and UML diagrams for this component**

Generated on: June 15, 2025

# \*\*Technical Report: Plant Diagnosis Component\*\*

This report documents the React component responsible for plant diagnosis, analyzing its purpose, key modules, and data models. The component utilizes user input (image, plant type, watering frequency, language) to send a request to a backend server for analysis and displays the results.

## 1. \*\*Purpose\*\*

The primary purpose of this React component (`Analyze`) is to provide a user interface for diagnosing plant health. Users can upload an image of their plant, provide details about the plant and its care, and receive a diagnosis (healthy or unhealthy) along with recommendations. The application integrates image upload, webcam capture, and server-side image analysis.

## 2. \*\*Key Modules/Classes/Functions\*\*

1. React Modules: The component utilizes core React functionalities like `useEffect`, `useRef`, and `useState` for managing component state and lifecycle. `useNavigate` from `react-router-dom` is used for navigation.

2. Framer Motion: `motion` from `framer-motion` is used for animations and transitions, enhancing the user experience.

3. Axios: `axios` is employed to make HTTP requests to the backend server for image analysis.

4. React Icons: Various icons from `react-icons/fa` are integrated for visual clarity (e.g., back arrow, camera, checkmark, warning triangle).

5. React Webcam: The `Webcam` component enables users to capture images directly from their webcam.

6. Core Functions:

`handleImageChange`: Handles image upload from the user's gallery.

`dataURLtoFile`: Converts a data URL (received from the webcam) into a File object for server submission.

`handleCapture`: Captures an image from the webcam.

`handleAnalyze`: Sends a POST request to the backend server (`https://backend-lj86.onrender.com/analyze`) with the image and user input data. Handles loading state, error handling, and result display.

## 3. \*\*Data Models/Entities\*\*

The component interacts with the following data models:

1. Form Data: This is represented as a `FormData` object, containing:

`image`: A File object representing the uploaded or captured plant image.

`plantType`: A string (e.g., "neem", "guava").

`waterFreq`: A string representing watering frequency in days.

`language`: A string specifying the preferred language for the results (e.g., "english", "telugu").

2. Result Data: This is received from the backend server as a JSON object with the following structure:

`prediction`: A string indicating the plant's health status (e.g., "Healthy\_Plants", "Diseased\_Plant").

`recommendation`: A string containing recommendations based on the diagnosis.

## 4. \*\*Technology Stack\*\*

The component is built using:

React: For the user interface.

Axios: For making API calls.

Framer Motion: For animations and transitions.

React Icons: For visual elements.

React Webcam: For webcam integration.

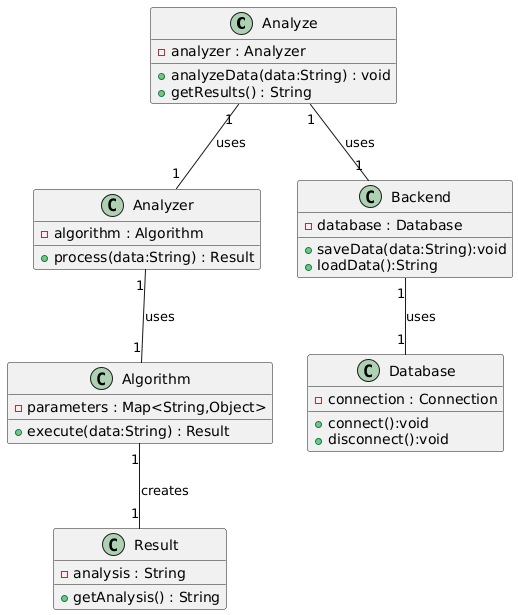
## 5. \*\*Conclusion\*\*

The `Analyze` component presents a user-friendly interface for plant diagnosis, effectively integrating various libraries to handle user input, image processing, server communication, and result display. The clear separation of concerns and well-defined functions make it maintainable and scalable. Further documentation, including UML diagrams, would enhance understanding of the component's interactions and data flow.

# Diagrams

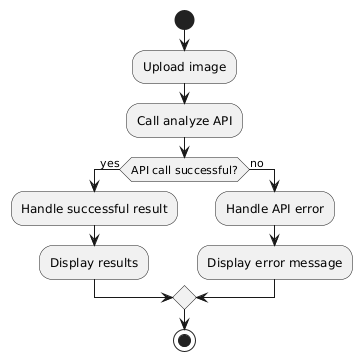
## Class Diagram

\*\* Illustrates the classes, attributes, and methods of the system, including the `Analyze` component and potentially backend classes.



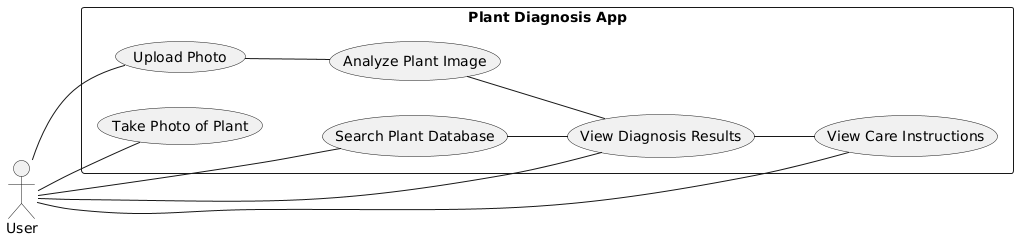
## Activity Diagram

\*\* Models the flow of actions within the `handleAnalyze` function, including steps like image upload, API call, and result handling.



## Use Case Diagram

\*\* Depicts the interactions between users (Plant Diagnosis app user) and the system (Plant Diagnosis app), showing how users analyze plants.



## Component Diagram

\*\* Shows the React components and how they are composed to form the overall UI. Would show relationships of `Analyze` with other components not present in this code snippet.



## State Machine Diagram

\*\* Models the various states of the `Analyze` component (e.g., loading, error, success) and the transitions between them.

