Technical Report

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

Generated on: June 15, 2025

# Table of Contents

# Technical Report: Plant Analysis Component

This report documents the React component `Analyze`, designed for plant disease diagnosis. It utilizes user inputs, image processing, and a backend API for analysis and result display.

## 1. Project Purpose

The `Analyze` component provides a user interface for diagnosing plant health. Users can upload images of plants, specify plant type, watering frequency, and preferred language. The component then sends this data to a backend API for analysis, receiving a prediction (healthy or specific disease) and a recommendation. The results are displayed to the user, along with options for further information if the plant is unhealthy.

## 2. Key Modules, Classes, and Functions

The component relies on several key modules and functions:

1. React Modules: `useEffect`, `useRef`, `useState`, `useNavigate` are used for state management, lifecycle methods, and navigation.

2. UI Libraries: `motion` (from `framer-motion`) provides animations for a smoother user experience. React Icons (`react-icons/fa`) are used for visual elements.

3. Image Handling: `Webcam` (from `react-webcam`) allows users to capture images directly from their webcam. The `dataURLtoFile` function converts data URLs to File objects for submission.

4. API Interaction: `axios` is used to make POST requests to the backend API ("`https://backend-lj86.onrender.com/analyze`"). The API call handles authentication using a token from `localStorage`.

5. Core Functions:

`handleImageChange`: Manages image uploads from the user's gallery.

`dataURLtoFile`: Converts a data URL (from webcam capture) into a File object.

`handleCapture`: Captures an image from the webcam.

`handleAnalyze`: Sends data to the backend API for analysis and updates the UI accordingly.

## 3. Data Models and Entities

The component interacts with the following data models:

1. Frontend State: The component manages various states using React's `useState` hook:

`image`: Stores the uploaded or captured image file (File object).

`plantType`: Selected plant type (string, e.g., "neem").

`waterFreq`: Watering frequency in days (string).

`language`: Preferred language for the recommendation (string).

`result`: Analysis result from the backend (object containing `prediction` and `recommendation`).

`error`: Error messages (string).

`loading`: Loading status (boolean).

`preview`: URL of the image preview (string).

`useCamera`: Boolean indicating whether the webcam is being used.

2. Backend API Response: The backend API returns a JSON object with the following structure:

`prediction`: String indicating the plant's health status (e.g., "Healthy\_Plants", "disease\_name").

`recommendation`: String containing the recommendation based on the prediction.

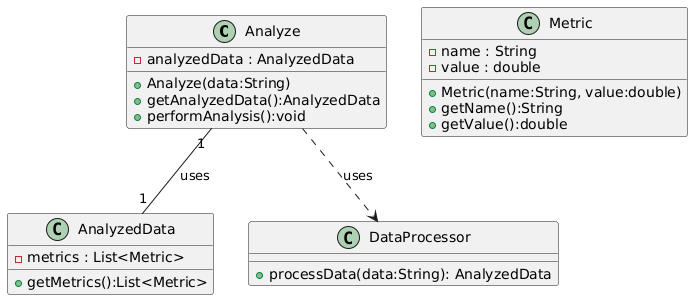
## 4. Conclusion

The `Analyze` component effectively integrates various libraries to create a user-friendly interface for plant disease diagnosis. Its modular design promotes maintainability and extensibility. Further development could focus on error handling and improved user feedback mechanisms. The integration with a backend API allows for sophisticated analysis beyond the scope of the client-side code.

# Diagrams

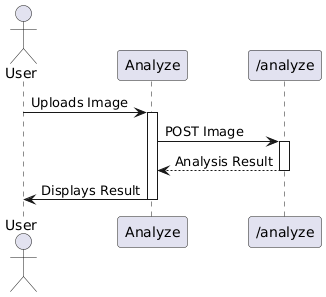
## Class Diagram

Shows the classes, attributes, and methods of the `Analyze` component and potentially relevant backend classes (if their structure is known).



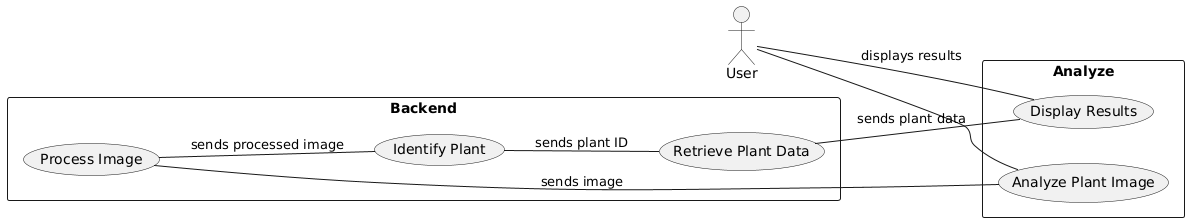
## Sequence Diagram

Illustrates the sequence of interactions between the `Analyze` component, the user, and the backend's `/analyze` endpoint during the image analysis process.



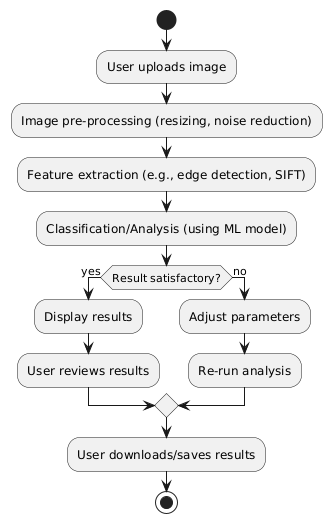
## Use Case Diagram

Depicts the interactions between the user and the system (the `Analyze` component and backend) to achieve specific goals, such as analyzing a plant image.



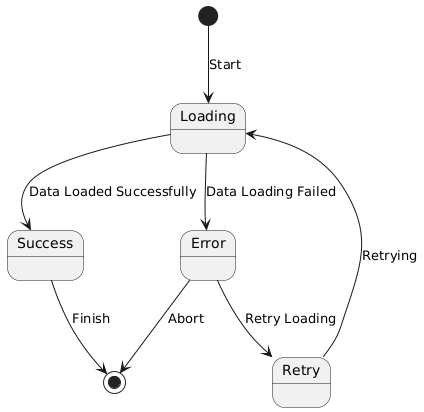
## Activity Diagram

Models the workflow of the image analysis process, including user actions, image processing, and the display of results.



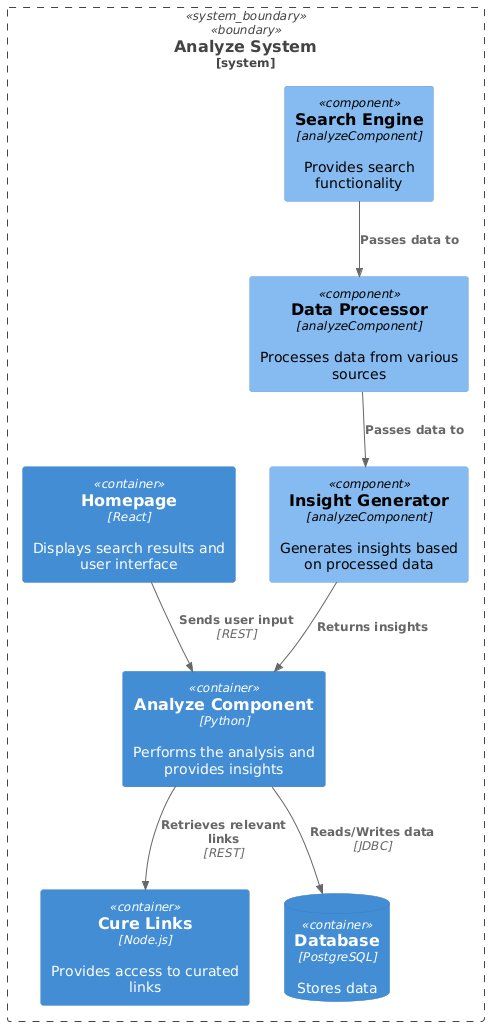
## State Machine Diagram

Shows the different states of the `Analyze` component (e.g., loading, error, success) and the transitions between these states.



## Component Diagram

(If you want to represent the overall application architecture) Shows the different components and how they are related. In this case, it would include `Analyze` component's relation to other components like the homepage, cure links, etc.



## Deployment Diagram

Illustrates how the React application and the backend Node.js server are deployed to their respective environments (Render for the backend).

