# 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 responsible for analyzing plant health based on user-provided image and information.

**1. Project Purpose:**

The primary purpose of this React component (`Analyze`) is to provide a user interface for plant disease diagnosis. Users upload or capture an image of a plant, specify the plant type, watering frequency, and preferred language. The component then sends this data to a backend server for analysis and displays the results, including a prediction of the plant's health and relevant recommendations.

**2. Key Modules, Classes, and Functions:**

The component leverages several key modules and functions:

* **`axios`:** For making HTTP requests to the backend server for analysis.

* **`motion` (from `framer-motion`):** For creating smooth animations and transitions within the UI.

* **`React` Hooks (`useEffect`, `useRef`, `useState`):** For managing component state, lifecycle events, and DOM references.

* **`FaArrowLeft`, `FaCamera`, `FaCheckCircle`, `FaExclamationTriangle`, `FaImages`, `FaSpinner` (from `react-icons/fa`):** For displaying Font Awesome icons in the UI.

* **`useNavigate` (from `react-router-dom`):** For navigation within the application.

* **`Webcam` (from `react-webcam`):** For capturing images from the user's webcam.

* **`handleImageChange`:** Updates the component's state with the selected image file and sets a preview URL.

* **`dataURLtoFile`:** Converts a data URL (obtained from the webcam) to a File object suitable for form submission.

* **`handleCapture`:** Captures an image from the webcam, converts it to a File object, and updates the component state.

* **`handleAnalyze`:** Handles the submission of the analysis form. It sends the image and other user input to the backend API, handles loading states, errors, and displays the results.

**3. Data Models or Entities:**

The component interacts with the following data:

* **Image Data:** A `File` object representing the uploaded or captured image.

*   **Plant Information:**

*   `plantType` (string):  The type of plant (e.g., "neem", "guava").

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

*   `language` (string):  Preferred language for results.

*   **Analysis Result:**

*   `prediction` (string): The predicted health status of the plant (e.g., "Healthy_Plants", a specific disease).

*   `recommendation` (string): Recommendations based on the prediction.

*   **Error Messages:**  Strings indicating errors during image upload or analysis.

*   **Loading State:** A boolean indicating whether the analysis is in progress.

**4. Component Structure:**

The `Analyze` component renders a form for user input, including image upload/capture options, dropdowns for plant type, watering frequency, and language selection.  The form uses controlled components, with state managed using React Hooks.  Upon submission, it sends a POST request to a backend API (`https://backend-lj86.onrender.com/analyze`) using `axios`. The component also handles loading and error states and displays the analysis results clearly.  The UI incorporates animations using `framer-motion`.

**5.  External Dependencies:**

The component depends on the following external packages:

* `axios`

* `framer-motion`

* `react-icons/fa`

* `react-router-dom`

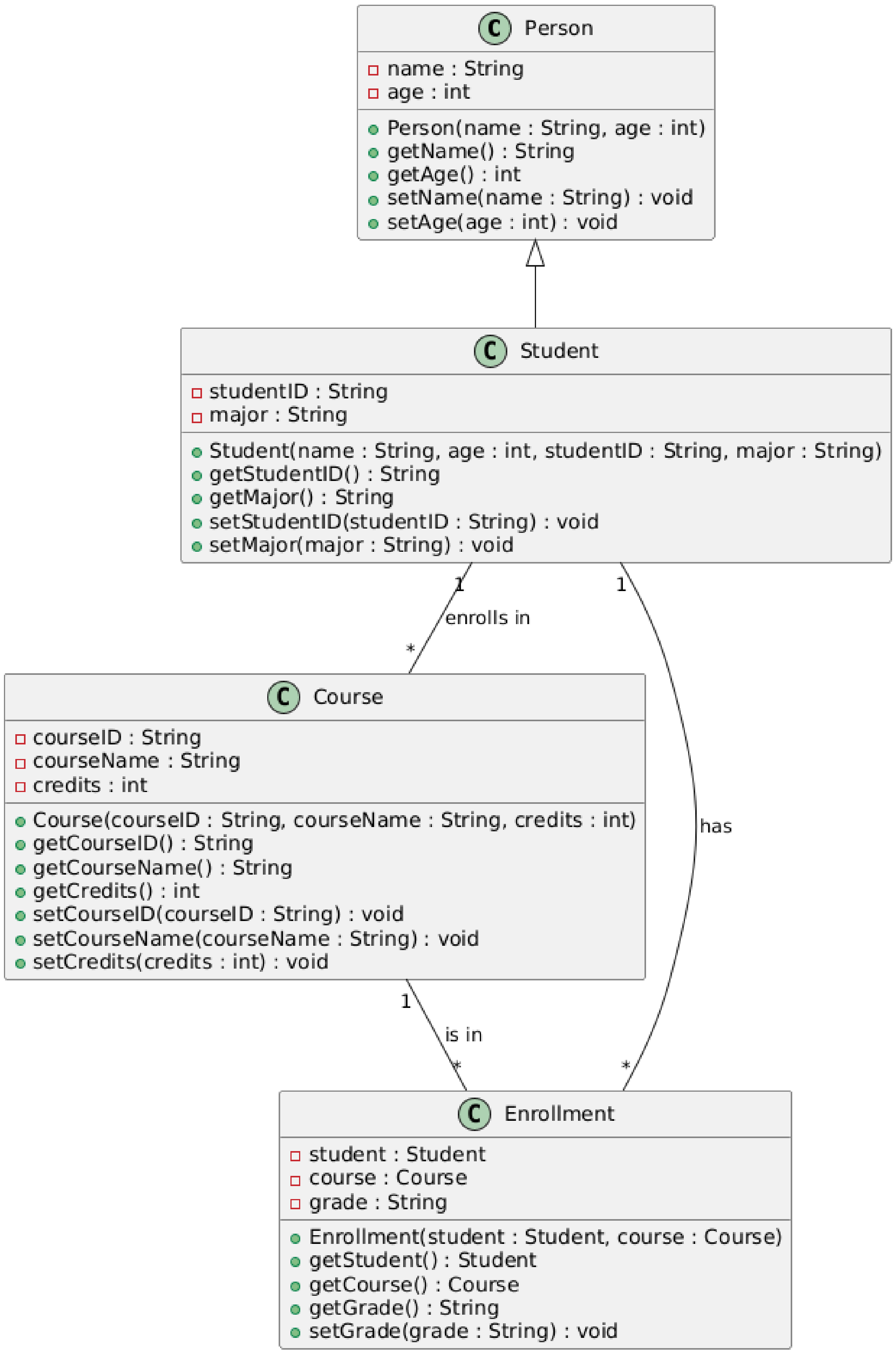
* `react-webcam`

This report provides a high-level overview of the plant analysis component. A more detailed analysis, including UML diagrams, would require further investigation into the backend API and detailed design specifications.

## Diagrams

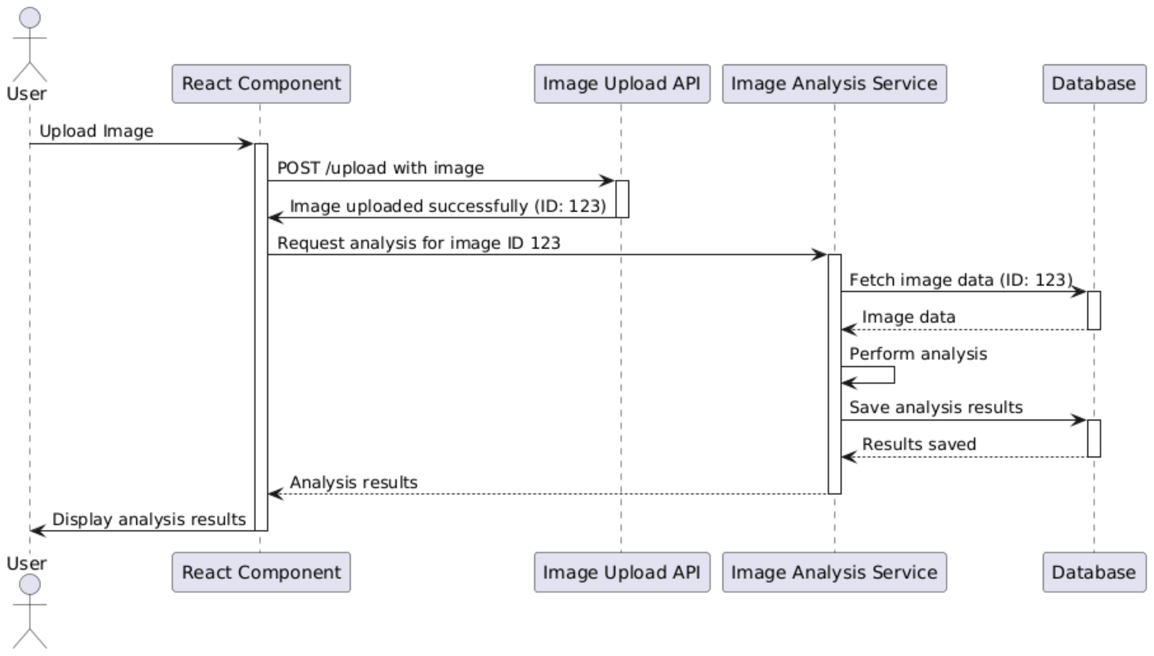
### Class Diagram

**  Illustrates the classes, their attributes, and methods, including relationships between classes (like inheritance or association).


Person
- name : String
- age : int
+ Person(name : String, age : int)
+ getName() : String
+ getAge() : int
+ setName(name : String) : void
+ setAge(age : int) : void
Student
- studentID : String
- major : String
+ Student(name : String, age : int, studentID : String, major : String)
+ getStudentID() : String
+ getMajor() : String
+ setStudentID(studentID : String) : void
+ setMajor(major : String) : void
1
enrolls in
*
1
has
*
Course
- courseID : String
- courseName : String
- credits : int
+ Course(courseID : String, courseName : String, credits : int)
+ getCourseID() : String
+ getCourseName() : String
+ getCredits() : int
+ setCourseID(courseID : String) : void
+ setCourseName(courseName : String) : void
+ setCredits(credits : int) : void
1
is in
*
Enrollment
- student : Student
- course : Course
- grade : String
+ Enrollment(student : Student, course : Course)
+ getStudent() : Student
+ getCourse() : Course
+ getGrade() : String
+ setGrade(grade : String) : void

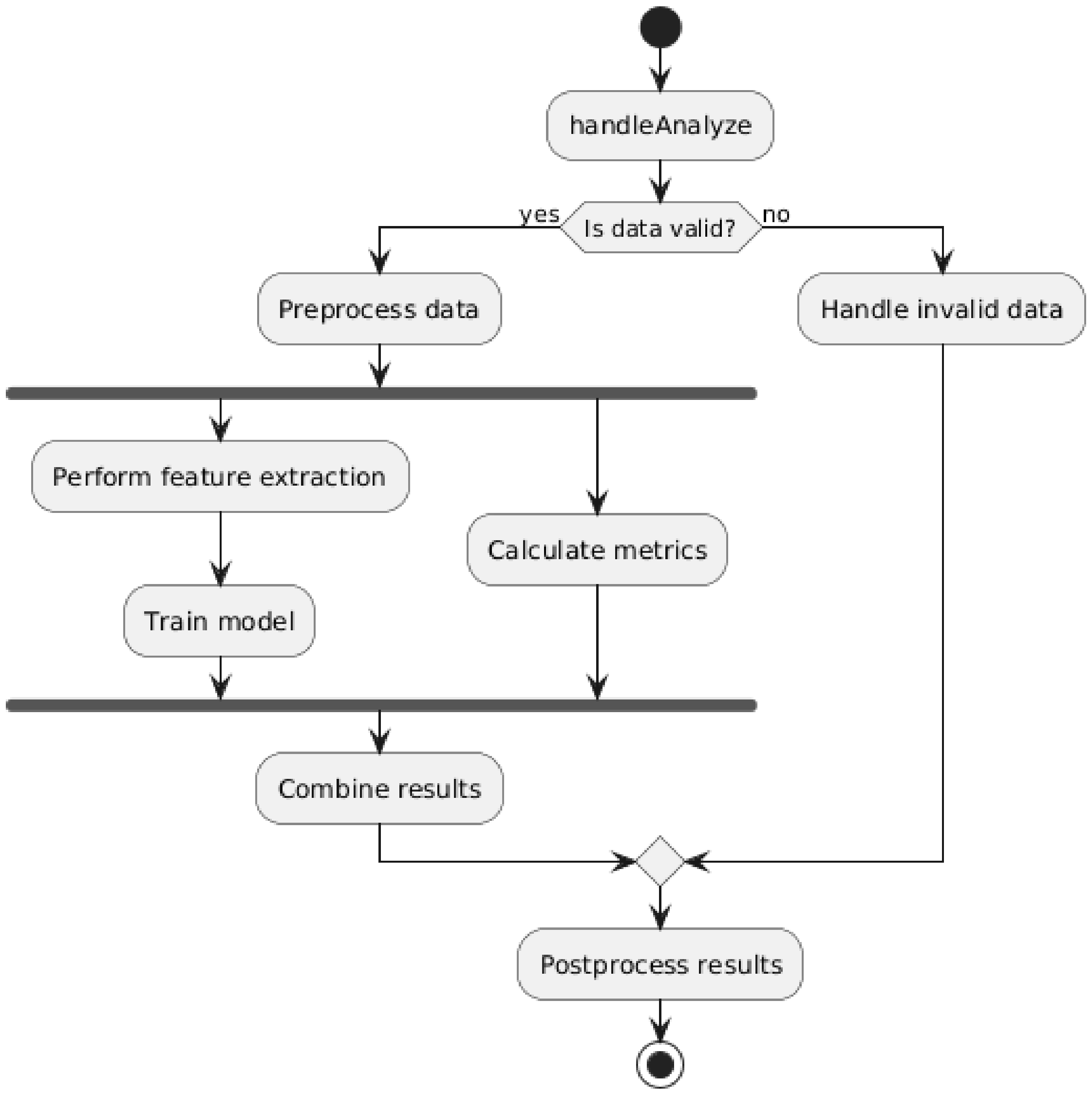
## Sequence Diagram

** Shows the interactions between objects (including the React component and backend) over time, in the context of a specific use case (e.g., image analysis).



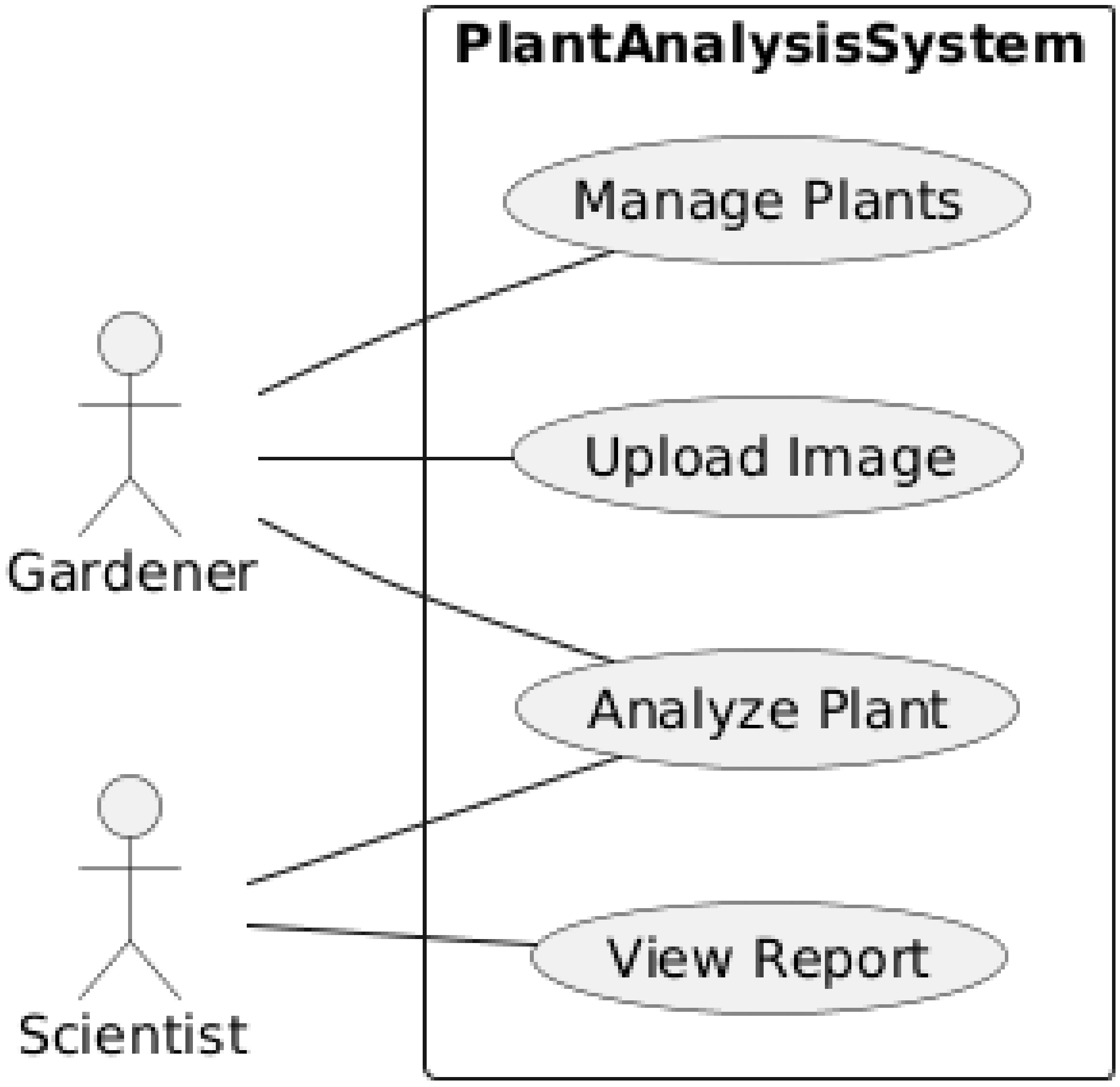
## Activity Diagram

** Models the flow of control within a process, such as the `handleAnalyze` function, showing decision points and parallel activities.



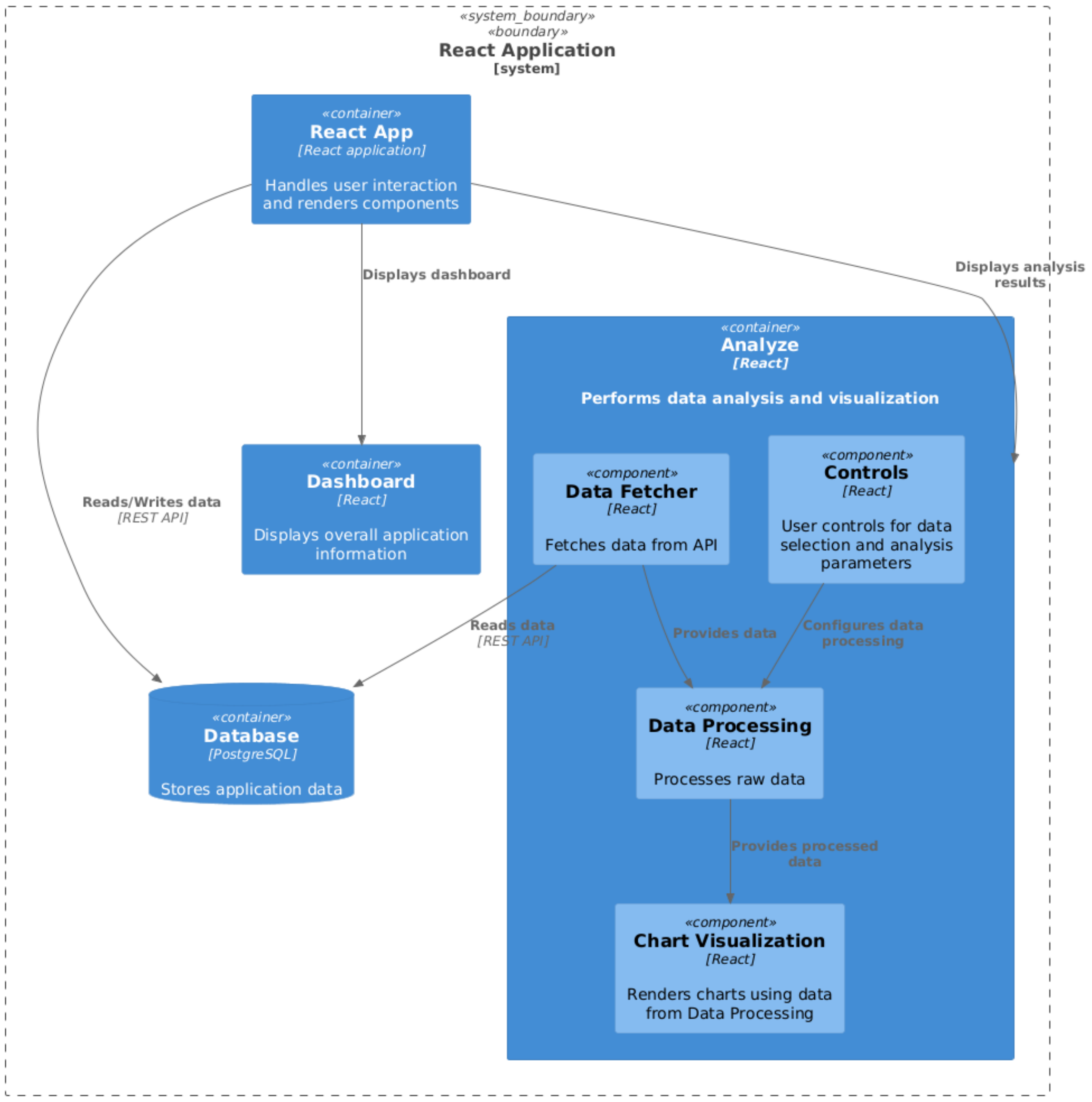
## Use Case Diagram

** Depicts the interactions between users (or actors) and the system, showing high-level functionalities (e.g., Analyze Plant).



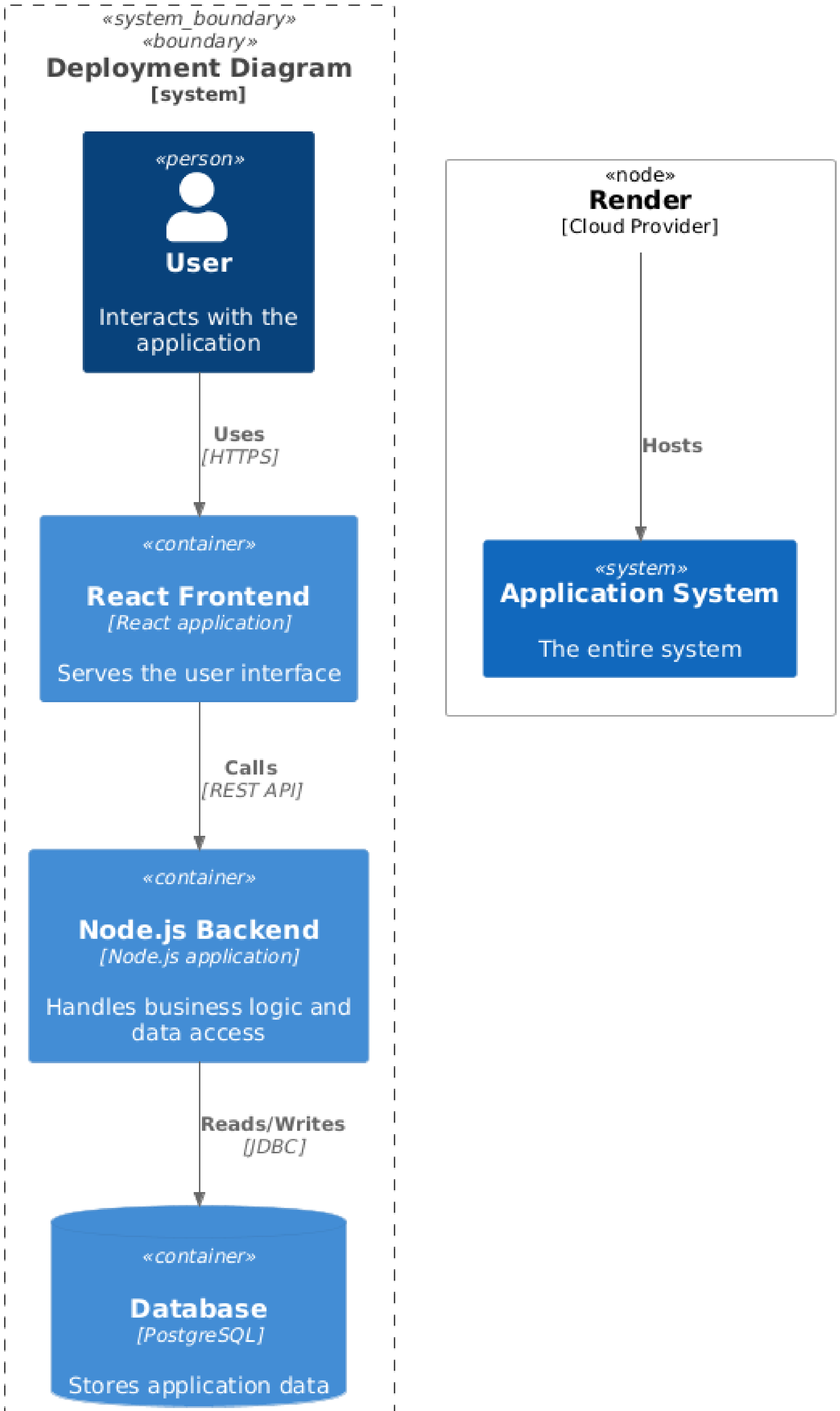
### Component Diagram

** (Specifically for React) Shows the React components and their relationships, including data flow and dependencies.  This would be especially useful to map out how the sub-components within `Analyze` are connected.



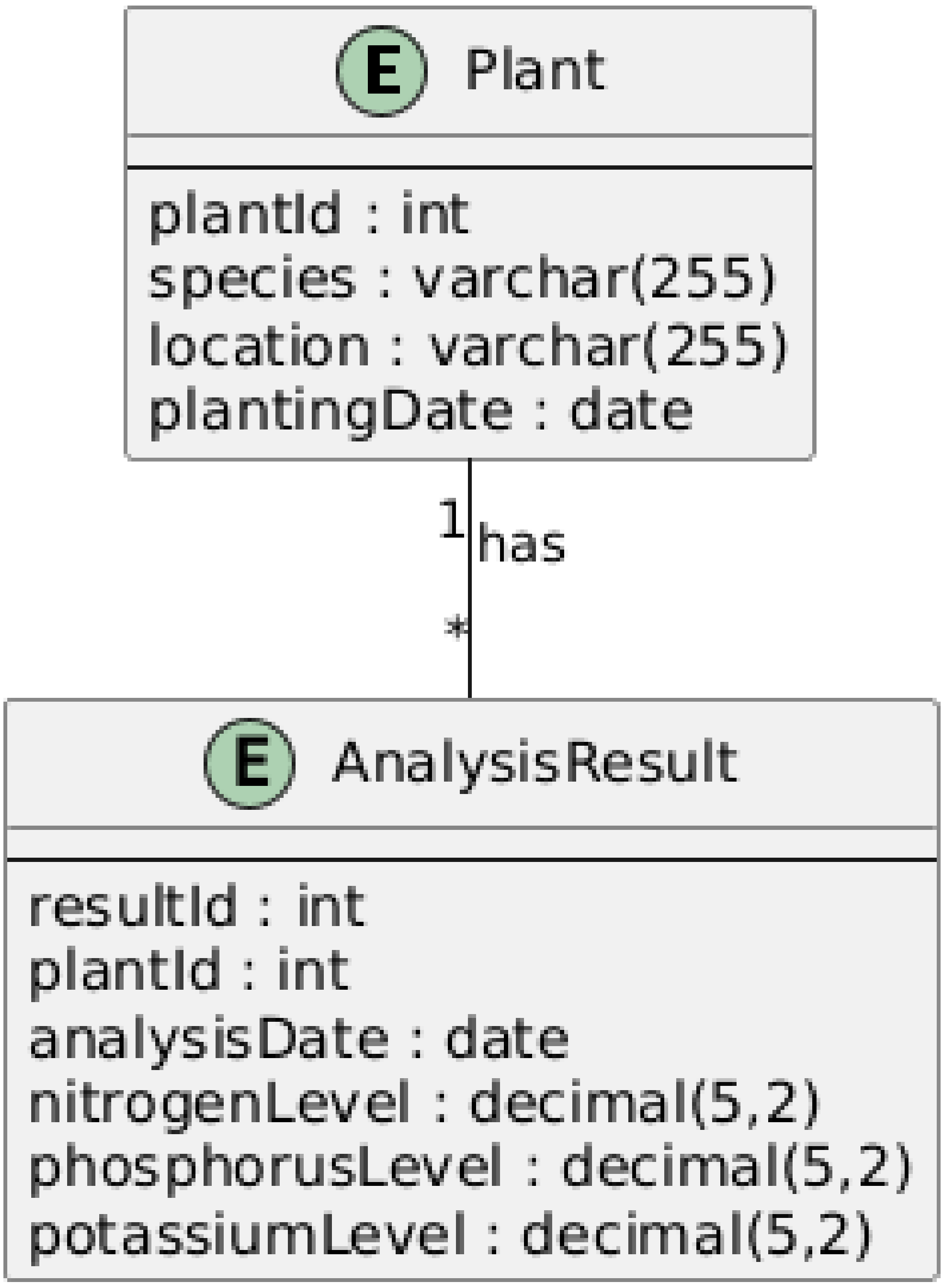
## Deployment Diagram

** Illustrates the physical deployment of the system, showing nodes (like the React frontend and the Node.js backend on Render) and their connections.


«system_boundary»
«boundary»
Deployment Diagram
[system]
«person»
User
Interacts with the application
Uses
[HTTPS]
«container»
React Frontend
[React application]
Serves the user interface
Calls
[REST API]
«container»
Node.js Backend
[Node.js application]
Handles business logic and data access
Reads/Writes
[JDBC]
«container»
Database
[PostgreSQL]
Stores application data
«node»
Render
[Cloud Provider]
Hosts
«system»
Application System
The entire system

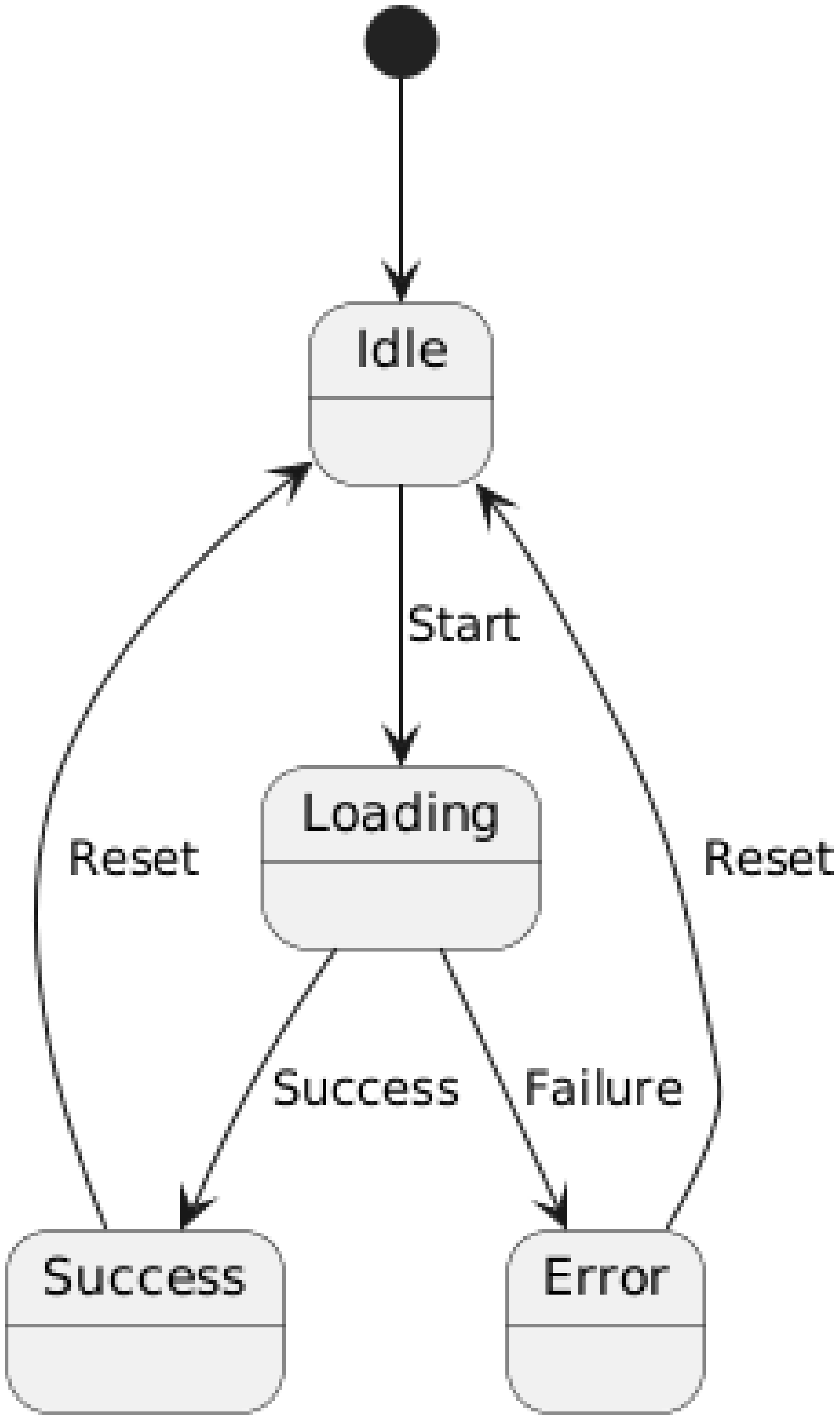
## Data Model Diagram Entity-Relationship Diagram - Erd

** Represents the structure of the data used by the system, including entities (e.g., Plant, AnalysisResult) and their attributes.


E
Plant
plantId : int
species : varchar(255)
location : varchar(255)
plantingDate : date
1
has
*
E
AnalysisResult
resultId : int
plantId : int
analysisDate : date
nitrogenLevel : decimal(5,2)
phosphorusLevel : decimal(5,2)
potassiumLevel : decimal(5,2)

## State Diagram

** A state diagram would be useful for modeling how the state of the application changes from one state to another in response to user actions. For example, the application could transition between an idle state, loading state, error state, and success state.


Idle
Start
Loading
Reset
Reset
Success
Failure
Success
Error