**Plant Growth Analytics System Documentation**

**Introduction**

The Plant Growth Analytics System is a monitoring and analytics solution designed to process sensor data, providing insights for optimising plant growth conditions. This document outlines the design decisions, assumptions, key API endpoints, database optimisations, and examples of usage.

**Design Decisions and Rationale**

**Database Design**

* + **Rationale**: PostgreSQL is known for its powerful support for relational data structures and features designed for scalability and performance. Its native capabilities for handling time-series data, combined with its compatibility with advanced extensions such as UUID and indexing, make it an optimal choice for the system's requirements.
  + **Improvements Made**:
    - Added indexes on frequently queried columns (zone\_id, plant\_id, timestamp) to optimise search queries. Indexes on zone\_id, plant\_id, and timestamp enable fast filtering for analytics queries such as growth rates, optimal conditions, and yield predictions based on specific zones or plants.
    - UUIDs already existed. Using it as the primary key ensures that the DB is unique and scalable. The uuid-ossp extension is already enabled.
    - The Database is dockerised ensuring easy consistency across environments.
  + **Future considerations:**
    - The data from sensors is time-series in nature. If the records are higher in volume, considering that they grow beyond 1512 records, the following implementations can be done:
      1. Partitioning the plant\_readings table by time for faster queries.
      2. Use of EXPLAIN ANALYZE to verify and tune query performance.
    - If the analytics queries are slow, usage of materialised views can be helpful to store frequently accessed metrics.
    - While the current schema design avoids redundancy, further normalisation (for example: creating separate tables for zones and plants) could reduce the storage overhead in high-scale environments.
    - As the data expands, other optimisation techniques like sharding or caching can be implemented.

**API Framework**

1. **FastAPI**:
   * **Rationale**: FastAPI was chosen for its modern, asynchronous architecture, built-in support for data validation with Pydantic, and ease of integration with JWT authentication.
   * **Approach**:
     + Sensitive endpoints like login and registration are protected using JWT tokens. The code employs *fastapi\_jwt\_auth* for creating and validating these tokens.
     + Utilised FastAPI's dependency injection to handle common tasks like authentication.
2. **Sensor Data Endpoints:**

These endpoints are responsible for managing and retrieving sensor data.

* + **Batch Upload Sensor Data:**

**Endpoint:** /api/v1/sensor-data/batch  
This endpoint allows the uploading of multiple sensor readings at once. It accepts a list of sensor data objects and saves them in the database.

* + **Single Sensor Data Upload:**  
    **Endpoint:** /api/v1/sensor-data/single  
    This endpoint is used for uploading a single sensor reading, including temperature, humidity, soil moisture, and light levels for a plant.
  + **Get Sensor Data by Zone**:  
    **Endpoint:** /api/v1/sensor-data/{zone\_id}  
    This endpoint retrieves sensor data for all plants in a specific zone, based on the zone ID. The data includes various environmental metrics like temperature, humidity, and soil moisture.
  + **Get Sensor Data by Zone and Plant**:  
    **Endpoint:** /api/v1/sensor-data/{zone\_id}/{plant\_id}  
    This endpoint retrieves sensor data for a specific plant in a specific zone. It returns all available sensor data for the given plant within the specified zone.

1. **Analytics Endpoints**:

These endpoints are designed to provide actionable insights such as:

* + **Growth Rate Trends:**  
    **Endpoint:** /api/v1/analytics/growth-rate/{plant\_id}  
    This endpoint calculates the growth rate of a plant by comparing plant heights over time.
  + **Optimal Environmental Conditions:**  
    **Endpoint:** /api/v1/analytics/optimal-conditions/{species\_id}  
    This endpoint calculates the optimal environmental conditions for a plant species based on sensor data (temperature, humidity, soil moisture).
  + **Yield Predictions:**  
    **Endpoint:** /api/v1/analytics/yield-prediction/{zone\_id}  
    This endpoint predicts the yield of plants in a specific zone based on historical data.

**Containerisation**

* **Docker**:
  + Both the backend and database are containerised to ensure portability and simplify setup.
  + A docker-compose.yml file orchestrates the services, allowing seamless multi-container deployment.

**Assumptions Made**

1. **Sensor Data**:
   * Assumed data is reliable and pre-validated by the source systems.
2. **API Design**:
   * Assumed that each species\_id and zone\_id has unique and consistent mappings.
   * Used RESTful principles to design endpoints for clarity and scalability.

**Key API Endpoints and Usage Examples**

**1. Data Ingestion**

* **POST** /api/v1/sensor-data/batch
  + **Description**: Ingests multiple sensor data records.
  + **Request Body**:

[

{

"zone\_id": "zone1",

"plant\_id": "rose\_1",

"temperature": 22.51,

"humidity": 60.0,

"soil\_moisture": 0.695,

"light\_level": 350.0

},

{

"zone\_id": "zone2",

"plant\_id": "tulip\_1",

"temperature": 20.0,

"humidity": 55.0,

"soil\_moisture": 0.695,

"light\_level": 300.0

}

]

* + **Response**: 201 Created

**2. Optimal Conditions Analysis**

* **GET** /api/v1/analytics/optimal-conditions/{species\_id}
  + **Description**: Determines the ideal growing conditions for a plant species.
  + **Response Example**:

{

"zone\_id": "zone\_1",

"plant\_id": "tomato\_1",

"temperature": 25.22,

"humidity": 64.19,

"soil\_moisture": 0.7,

"light\_level": 803.5,

"id": "778e5eee-aa6a-42db-a73d-d86f7b0551f9"

}

**Future Considerations:**

* Additional security measures such as token expiration and refreshing can be considered
* To prevent over-usage, rate limiting on certain endpoints can be implemented

**Setup Instructions**

1. **Clone Repository**:
   * git clone <repo\_url>
   * cd plant-growth-analytics
2. **Run Docker Compose**:
   * docker-compose up --build
3. **Access the API**:
   * API URL: http://localhost:8000

**AI Usage Documentation**

* Used AI to guide the initial setup of Docker
* Cross-checked the entire project with the requirements to double-check that all conditions were met
* For troubleshooting various compatibility issues that arose from using the lower versions of docker and other libraries
* Used to resolve the errors and in debugging

**Additional Future Considerations**

* More folders and modularity for better organisation.
* Front-end implementation with React.js or Vue.js.
* Machine learning models for further data analysis and predictions.
* Data smoothing techniques for improved accuracy.
* Detailed analytics and visual reporting.
* Usage of environment variables to store sensitive information (e.g., database password)

**Conclusion**

The Plant Growth Analytics System satisfies the core requirements by enabling efficient sensor data ingestion, providing actionable insights through analytics endpoints, and optimising the database for performance. Containerisation ensures ease of deployment, making the system scalable and robust.