

**CPS3962project**

Dr.Hamza Djigal

**Name**：Jin zijian

**ID**:1236608

**Topic**: IoT-Enabled Smart Agriculture System

**Objective:**Build a system that uses IoT sensors to monitor crops and soil conditions, applying object-oriented principles to manage different components such as irrigation and fertilization.

**Date:** May.11

**1. Requirement Analysis**

**1.1 Problem Definition**

The increasing demand for sustainable agricultural practices and the need for efficient crop management has made it essential to leverage technology. Traditional farming methods often struggle with issues such as insufficient irrigation, improper fertilization, and delayed monitoring of crop health. This project proposes an IoT-enabled smart agriculture system that will leverage IoT sensors to monitor environmental factors like soil moisture, temperature, and humidity. The system will automate irrigation and fertilization based on real-time data, improving crop yield, reducing resource wastage, and enhancing farm management.

**1.2 Stakeholders**

The key stakeholders involved are:

Farmers: Primary users who will monitor crop conditions, adjust system settings, and receive alerts.

Agriculture Experts: Users who analyze the data provided by the system to make recommendations on crop management.

System Administrators: Technicians responsible for maintaining the system, performing software updates, and managing IoT devices.

Government and Environmental Agencies: Interested in the sustainability and efficiency of agricultural practices, ensuring compliance with regulations.

Technology Vendors: Providers of IoT sensors, controllers, and other hardware components.

**1.3 Functional Requirements**

The system must support the following core functionalities:

Data Collection: Collect environmental data (soil moisture, temperature, humidity, light intensity) via IoT sensors.

Real-Time Data Processing: Process incoming sensor data to monitor crop health and environmental conditions.

Irrigation Control: Automatically trigger irrigation systems when soil moisture drops below predefined thresholds.

Fertilization Control: Automatically trigger fertilization processes based on soil conditions.

Alerts and Notifications: Send alerts to farmers in case of critical changes or system malfunctions.

Data Storage and Retrieval: Store sensor data and system logs for later retrieval and analysis.

Manual Override: Allow farmers to manually override the system if necessary.

**1.4 Non-Functional Requirements**

The system must exhibit the following qualities:

Real-Time Performance: Ensure that decisions (irrigation, fertilization) are made within sub-second latency.

Scalability: The system must scale to support multiple fields, farms, and sensor types.

Reliability: Ensure that the system operates continuously with minimal downtime.

Security: Secure sensitive farm data and restrict unauthorized access to the system.

User-Friendly: The system interface should be easy for farmers to use, with clear indicators and simple controls.

Maintainability: The system must support remote diagnostics and regular updates without requiring significant downtime.

**2. System Architecture**

The system follows an object-oriented architecture, with key modules:

**Sensor:** Responsible for collecting environmental data.

**IrrigationController:** Controls the irrigation system based on soil moisture.

**FertilizationController:** Controls the fertilization system based on soil conditions.

**DataLogger:** Stores sensor data and system logs.

**AlertSystem:** Sends alerts to farmers when thresholds are exceeded.

**UserInterface:** Allows farmers to monitor system status and configure settings.

**CloudStorage:** Stores data for later retrieval and analysis.

**3. Use Case Diagram**

**Include use cases such as:**

Monitor Crop Conditions

Control Irrigation System

Generate Alerts

Visualize Data

Configure System Settings

**Actors:**

Farmer (user)

IoT Sensors (automated actor)

Cloud Server (data handler)

**4. Class Diagram**

The following is the class diagram for the IoT-Enabled Smart Agriculture System, illustrating the key components and their relationships:

**Main Classes:**

**Sensor**

Attributes: ID, location, type

Methods: collectData(), sendData()

**IrrigationController**

Attributes: ID, status

Methods: triggerIrrigation(), adjustSettings()

**FertilizationController**

Attributes: ID, status

Methods: triggerFertilization(), adjustSettings()

**DataLogger**

Attributes: dataID, timestamp

Methods: storeData(), retrieveData()

**AlertSystem**

Attributes: alertID, type

Methods: sendAlert(), acknowledgeAlert()

**UserInterface**

Attributes: userID, interfaceSettings

Methods: displayData(), receiveUserInput()

**CloudStorage**

Attributes: storageID, data

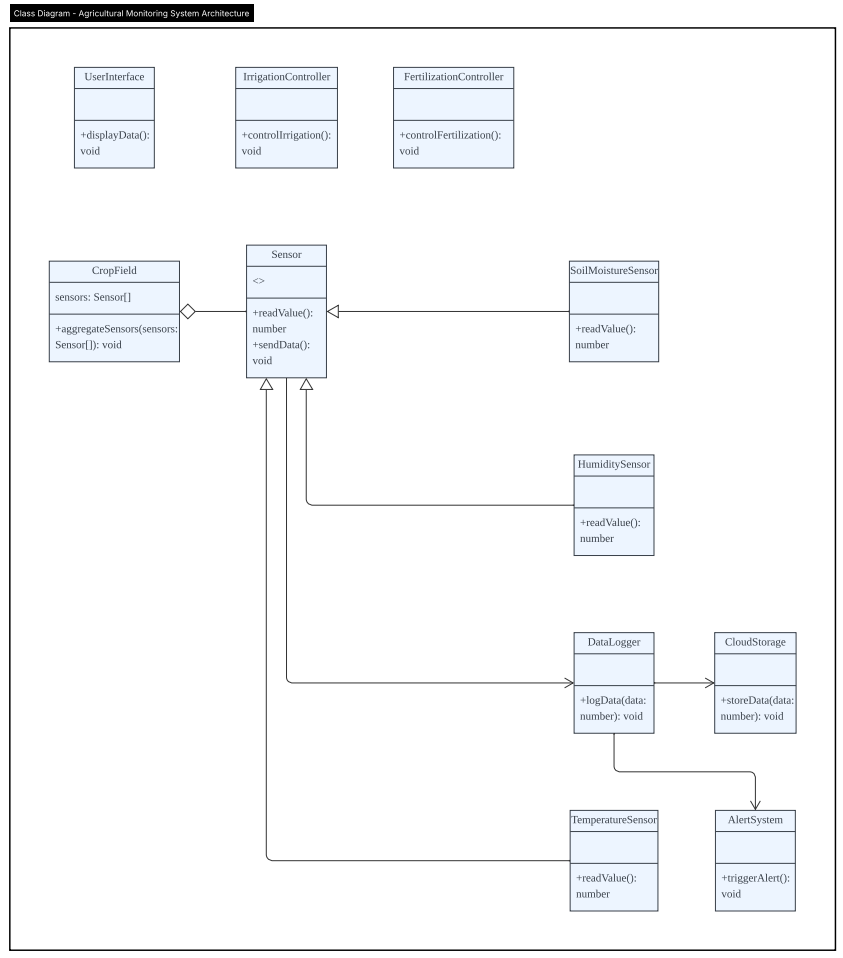
Methods: storeData(), retrieveData()

**Relationships:**

The CropField class aggregates multiple Sensor instances to collect environmental data.

Sensor sends data to DataLogger, which stores it for analysis.

IrrigationController and FertilizationController are triggered based on data collected from sensors.



**5. Sequence Diagram**

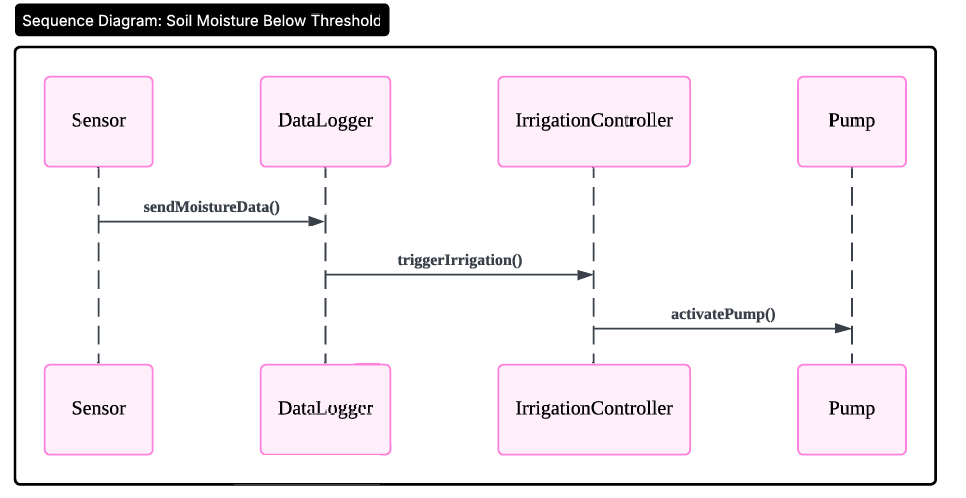
Example: Soil Moisture Below Threshold

**Sensor** → Data Logger: send MoistureData()

**DataLogger** → Irrigation Controller: triggerIrrigation()

**IrrigationController** → Sensor/Pump: activatePump()

This sequence diagram illustrates how the system responds when soil moisture falls below the threshold: the sensor sends data to the data logger, which in turn triggers the irrigation system to activate the pump.



**6. Activity Diagram**

**Data Flow:**

Sensor collects data

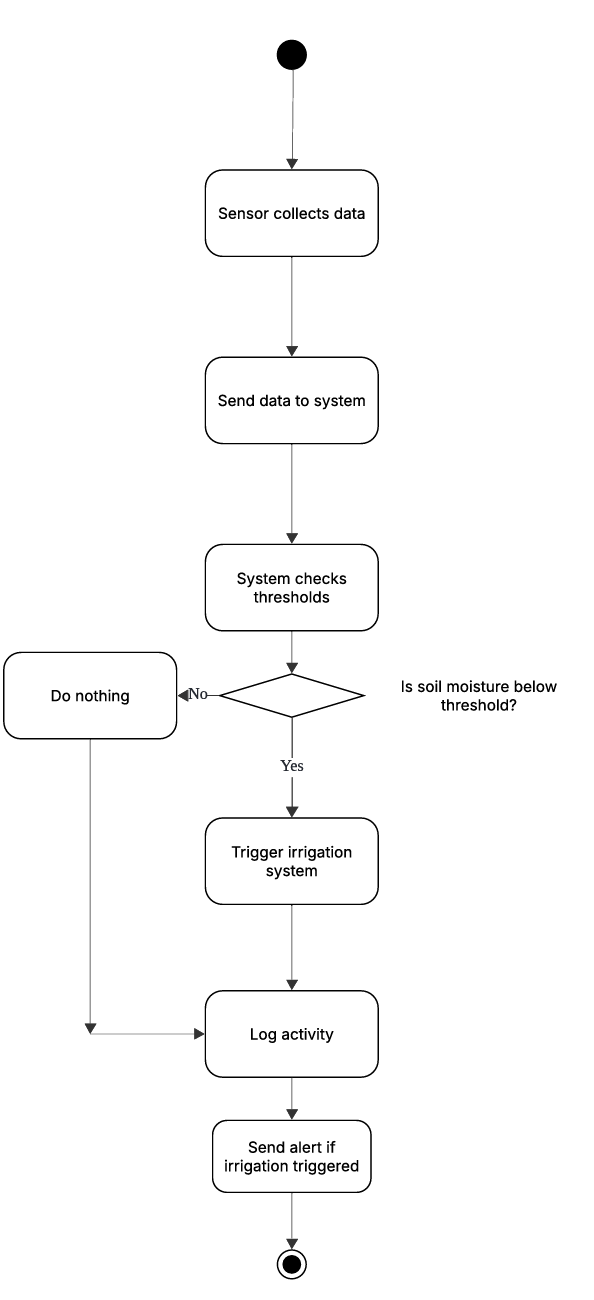
Data sent to the system

System checks thresholds

If below threshold → Trigger irrigation

Log the activity and send an alert

This activity diagram represents the full process from data collection to irrigation activation and alert notification, ensuring the system reacts appropriately when moisture levels drop too low.



**7. Testing and Validation**

**Unit Testing:**Each module is tested in isolation to ensure that its internal logic works correctly. For example:

Test that Sensor.collectData() accurately retrieves environmental parameters under different conditions.

Validate that IrrigationController.triggerIrrigation() activates only when thresholds are breached.

Verify DataLogger.storeData() correctly writes data to the storage with timestamps.

**Integration Testing:**Ensure smooth interaction between modules:

Sensor → DataLogger → Controller communication flows are tested.

Confirm that the CloudStorage component correctly receives and retrieves sensor data.

Simulate manual override from UserInterface to check system response.

**Performance Testing:**Assess system efficiency under various loads：

Test how the system responds when hundreds of sensors are active simultaneously.

Measure latency between data collection and system actuation (e.g., irrigation activation).

Evaluate the alerting mechanism under rapid data fluctuations (e.g., sudden temperature drop).

**Stress Testing:**

Simulate worst-case scenarios such as sensor failure, data spikes, or network delay.

Monitor if the system gracefully degrades or fails over to safe default states.

System Validation:

Ensure that all functional and non-functional requirements are met.

Verify automatic irrigation and fertilization logic.

Conduct user acceptance testing with real farmers or agriculture experts.

**8. Deployment and Maintenance**

**Deployment:**

Install IoT sensors (e.g., soil moisture, temperature, humidity) in strategic locations across the field.

Set up edge computing units or microcontrollers (e.g., ESP32 or Arduino) to collect and transmit data.

Connect the system to a centralized cloud database (e.g., AWS, Firebase) for data storage and analysis.

Deploy a web or mobile user interface for real-time monitoring and manual control.

Perform pilot testing in a single field before full-scale rollout.

**Maintenance:**

Sensor Calibration: Periodically calibrate sensors to maintain accuracy (especially soil moisture and temperature sensors).

Firmware Updates: Push over-the-air (OTA) updates to microcontrollers or control modules.

System Monitoring: Implement a self-check module for sensors and controllers that reports anomalies to admins.

Data Backup: Schedule regular backups of environmental data, system logs, and configuration settings to prevent data loss.

Hardware Inspection: Conduct monthly inspections of wiring, pump mechanisms, and field equipment.

User Training: Train farmers or system users on how to interpret data and interact with the interface (mobile or desktop).

Incident Handling: Develop standard procedures for handling sensor failures, false alerts, and unexpected weather anomalies.