FLORA

Farming Life-cycle Observation with Real-time Analytics

UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA

DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

Department of Information Engineering, Computer Science and Mathematics

University Of L'Aquila, Italy

Professor Davide Di Ruscio

Software Engineering for Internet of Things

Studente	**Matricola**	**Email**
Calogero Carlino	302154	calogero.carlino@student.univaq.it
Daniele Borgna	302121	daniele.borgna@student.univaq.it
Luca Francesco Macera	302123	lucafrancesco.macera@student.univaq.it

Table of contents

1. Introduction

Our project will monitor data related on both environment and plants that span from weather data such as temperature, humidity, rainfall, solar radiation, light intensity, CO2 level and soil moisture; going over plant's health data such as ph, concentration of mineral salts, chlorophyll content, leaf temperature, plant stress levels and pest infestations; and lastly to crop yield data such as plant height and canopy density.

The goal of the system is to streamline agricultural processes and empower farmers with better control over their fields, crops, and the large volumes of data they generate. Also allowing the visualization with a user friendly application showing gauges, graphs or other descriptive diagrams.

The system will analyze the data and alert the farmer in case of hazards concerning plant health like rotting, lack of water or excessive exposure to sunlight.

2. Goals

The goals of the systems will be:

- Reliable IoT system for real-time monitoring of crops and fields.
    - Develop a robust IoT system capable of tracking the temperature, humidity, and air quality
- Detection of changes in the environment and the plants health levels
    - Implement systems to detect abnormal changes in temperature, humidity, and air quality
    - Implement systems to detect abnormal changes in ph, mineral salts and water levels.
- Smart alert system to notify the farmers in case of plant hazards.
    - Integrate an alarm system that notifies the users when the registered levels go above the configured thresholds.
- User-friendly web-based dashboard accessible to the farmers
    - Create a dashboard that provides real-time updates on plant health and crop fields' environmental-related condition

3. Functional Requirements

- **FR1: Add monitoring sensors**
  Description: The system can be scaled horizontally by adding new sensors for plant and weather monitoring
  Type: Functional Requirement - Monitoring
  Priority: **High**
- **FR2: Plant canopy density monitoring**
  Description: The system monitors each plant canopy density and sends the data to the time-series database
  Type: Functional Requirement - Monitoring
  Priority: **Medium**
- **FR3: Plant chlorophyll content monitoring**
  Description: The system monitors each plant chlorophyll content and sends the data to the time-series database
  Type: Functional Requirement - Monitoring
  Priority: **Medium**
- **FR4: Plant Height monitoring**
  Description: The system monitors each plant height and sends the data to the time-series database
  Type: Functional Requirement - Monitoring
  Priority: **Low**
- **FR5: Plant Humidity monitoring**
  Description: The system monitors each plant humidity and sends the data to the time-series database
  Type: Functional Requirement - Monitoring
  Priority: **High**
- **FR6: Plant Temperature monitoring**
  Description: The system monitors each plant temperature and sends the data to the time-series database
  Type: Functional Requirement - Monitoring
  Priority: **High**
- **FR7: Plant Ph Level monitoring**
  Description: The system monitors each plant pH level and sends the data to the time-series database
  Type: Functional Requirement - Monitoring
  Priority: **High**
- **FR8: Data analysis**
  Description: The system analyzes data stored in the time-series database
  Type: Functional Requirement - Analysis
  Priority: **High**
- **FR9: Alert**
  Description: The system alarms the farmers if the sensors readings are above

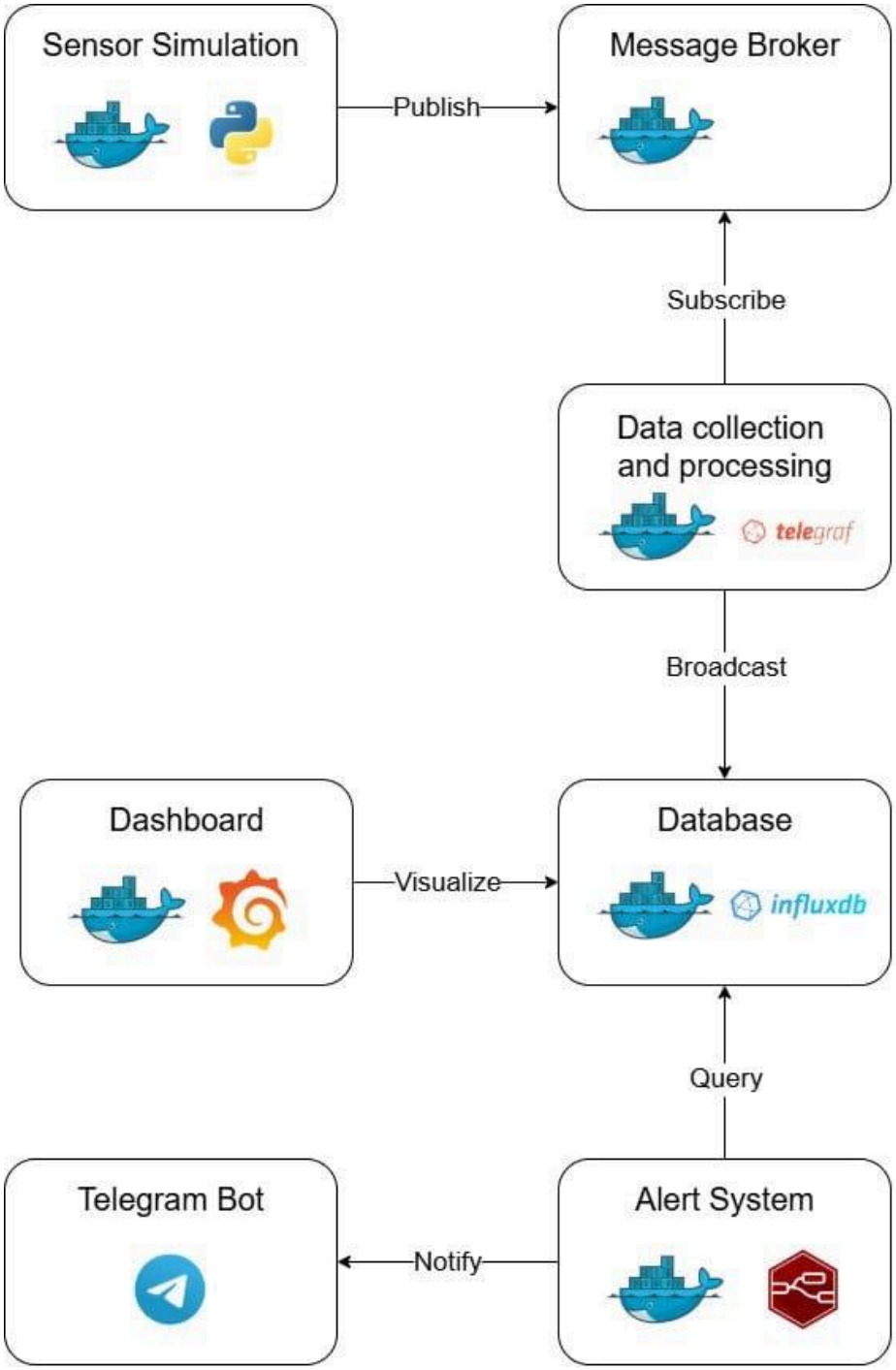
safe thresholds
Type: Functional Requirement - Alarm
Priority: **High**

- **FR10: Data Visualization**
  Description: The farmers can visualize each plant data in a dashboard
  Type: Functional Requirement - Visualization
  Priority: **Low**

4. Non Functional Requirements

- **NFR1: Scalability**
  Description: The system must scale horizontally and vertically to accommodate an increasing number of sensors and users.
- **NFR2: Reliability**
  Description: High system availability is critical, with redundant components to ensure continuous operation during peak loads or failures.
- **NFR3: Performance**
  Description: The system must process and store data at an adequate rate with minimal latency.
- **NFR4: Usability**
  Description: The user interface should be intuitive, providing clear visual feedback and easy navigation.
- **NFR5: Maintainability**
  Description: The system architecture should allow easy maintenance, including updates and scaling, with minimal downtime.
- **NFR6: Energy Efficiency**
  Description: The system must be optimized for energy consumption, especially concerning the sensors deployed in the field and the central data processing units.

5. System Architecture



6. System Functionalities

Sensor configuration

The different types of sensors and their respective threshold limits are defined in a JSON configuration file that is used to dynamically manage all the devices used by the system.

```json
{
    "sensors": [
        {
            "name": "temperature",
            "unit": "°C",
            "min": 0,
            "max": 50,
            "threshold_min": 10,
            "threshold_max": 30,
            "chart_type": "disc"
        },
        {
            "name": "humidity",
            "unit": "%",
            "min": 0,
            "max": 100,
            "threshold_min": 40,
            "threshold_max": 60,
            "chart_type": "disc"
        },
```

Sensor data generation

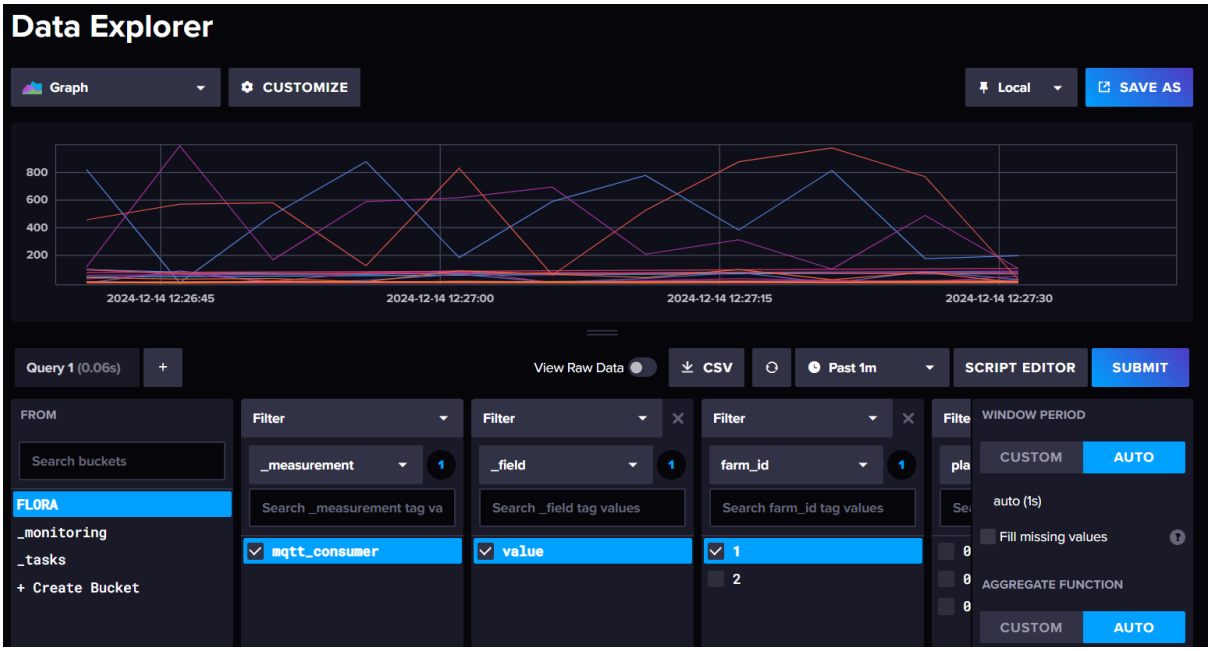
The sensor readings are automatically generated using a collection of Python scripts that simulate a complete environment with a semi-realistic season and weather cycle. This ensures that all the values obtained from the simulated sensors are more realistic.

Data communication

The data generated by the Python scripts is published directly to the MQTT broker for further usage.

Data storage

Every time a new value is published to the MQTT broker it’s immediately processed by the Telegraf agent and stored into the InfluxDB database.



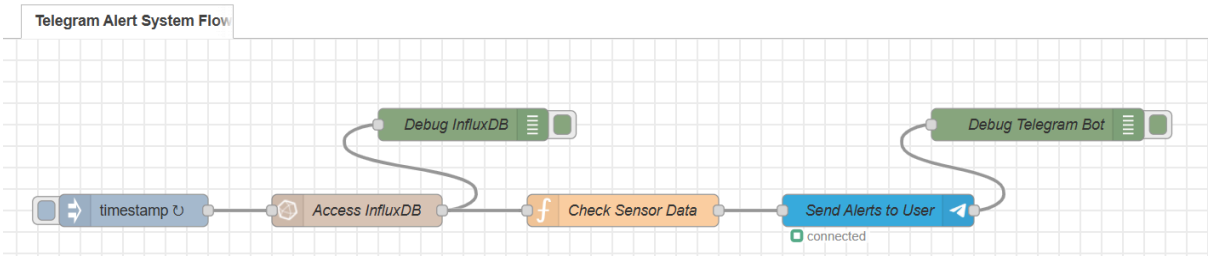
Data visualization

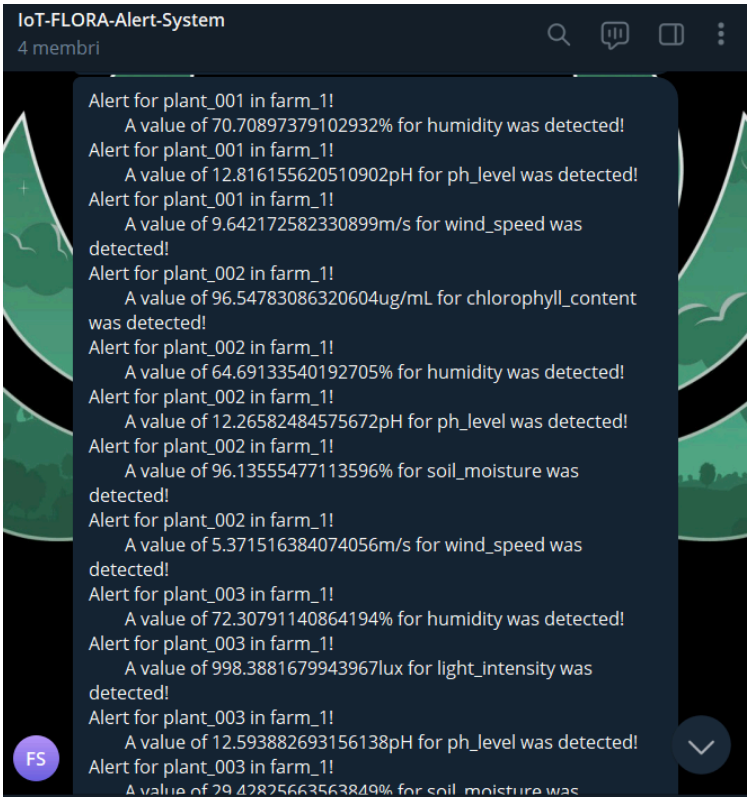
The values stored in the database are displayed on a Grafana dashboard, where data can be filtered by farm id and plant id, allowing the users to focus on the incoming values for a specific plant.

The dashboard is built dynamically, depending on settings specified into the JSON config file.

Alert notification

Every 5 seconds, the system checks the latest values stored in the database and sends an alert if a reading exceeds the safe limits set for that specific type of sensor. The alert is sent using Telegram as the messaging service.





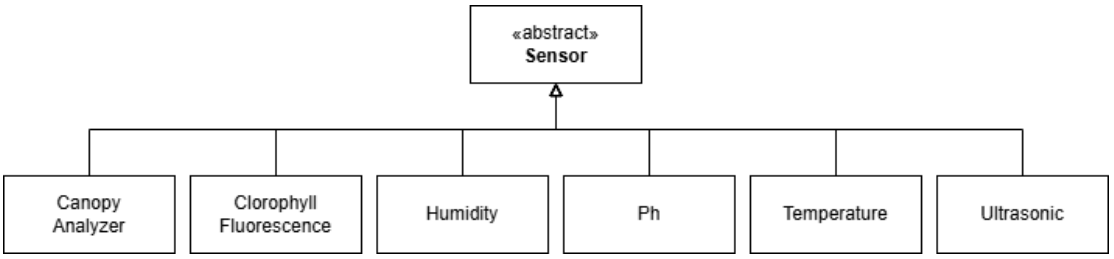
7. Implementation details

Sensor data generation

To enhance the realism of data generation, we implemented a mechanism to simulate data based on the following factors:

- **Season**
- **Geographical region in Italy** (North, Center, South)

Specifically, six types of sensors have been implemented, each extending the abstract class `Sensor`.



Each type of sensor implements a logic for data generation based on hardcoded boundaries, determined by the environment (a combination of zone and season, along with two flags: `is_raining` and `is_snowing`) provided to the constructor. For instance, the boundaries for the humidity sensor are as follows:

	North	**Center**	**South**
Winter	75-90	70-85	65-80
Spring	60-80	55-75	50-70
Summer	50-70	40-60	40-60
Autumn	70-85	65-80	60-75

The data is taken from real world sensor reading across all Italian regions

In addition to the boundaries, some logic specific to each sensor is hardcoded to avoid generating completely random values. For instance, in the case of humidity, the returned value significantly increases during rain or snow. Similarly, for the ultrasonic sensor (used to measure height), the value gradually increases over time, reaches a peak, and then drops to a very low level (to simulate plant pruning by the farmer) before starting to rise again.

The configuration of which sensors are active or inactive is managed through a JSON configuration file (the one shown in the "*Sensor Configuration*" section). However, for hardcoded sensors, the thresholds specified in the JSON file are not applied. Instead, different boundaries and thresholds are used for each environment.

In addition to the hardcoded sensors, it is possible to define a new sensor by simply specifying its name, unit of measurement, minimum and maximum boundaries, minimum and maximum thresholds, and the chart type (e.g., line or disc) in the JSON configuration file.

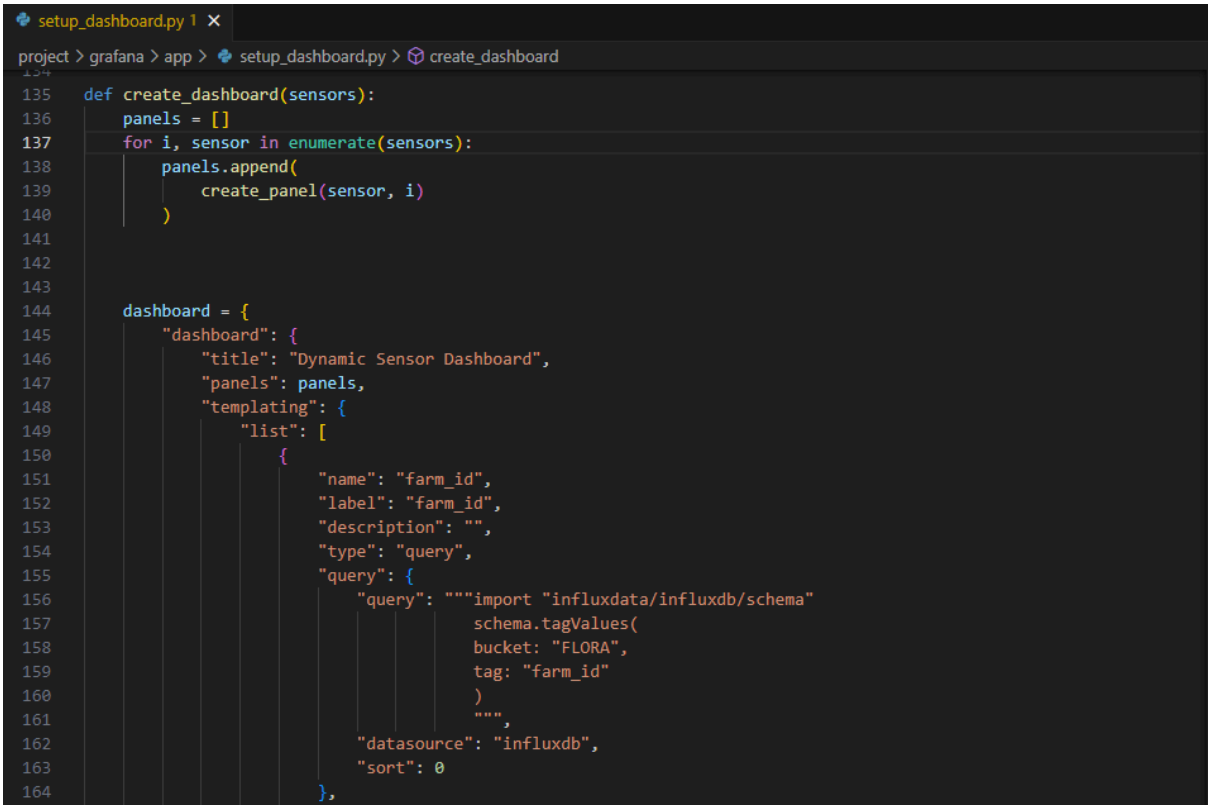
```
        {
            "name": "wind_speed",
            "unit": "m/s",
            "min": 0,
            "max": 10,
            "threshold_min": 1,
            "threshold_max": 5,
            "chart_type": "disc"
        }
    ]
}
```

Example of custom sensor definition

To implement these custom sensors, a `GenericSensor` class is used. This class generates random values within the specified boundaries, with a 0.1 probability of producing values outside the defined thresholds.

Grafana dashboard

Grafana encodes dashboards into JSON files and allows dashboard creation via API through an HTTP request. Specifically, we generate the JSON structure based on the sensors specified in the configuration file (the one shown in the "*Sensor Configuration*" section) and then push it to Grafana using the APIs. This approach enables the dynamic creation of dashboards, ensuring they reflect the sensors defined in the configuration file.
The logic for the Grafana dashboard generation can be found inside the *setup_dashboard.py* file inside the *grafana/app* folder.



```
setup_dashboard.py 1
project > grafana > app > setup_dashboard.py > create_dashboard
135  def create_dashboard(sensors):
136      panels = []
137      for i, sensor in enumerate(sensors):
138          panels.append(
139              create_panel(sensor, i)
140          )
141
142
143
144      dashboard = {
145          "dashboard": {
146              "title": "Dynamic Sensor Dashboard",
147              "panels": panels,
148              "templating": {
149                  "list": [
150                      {
151                          "name": "farm_id",
152                          "label": "farm_id",
153                          "description": "",
154                          "type": "query",
155                          "query": {
156                              "query": """import "influxdata/influxdb/schema"
157                                          schema.tagValues(
158                                          bucket: "FLORA",
159                                          tag: "farm_id"
160                                          )
161                                          """,
162                              "datasource": "influxdb",
163                              "sort": 0
164                          },
```

setup_dashboard.py code snippet

Add is executed via ENTRYPOINT in the grafana dockerfile

```
31   # Comando per eseguire lo script
32   ENTRYPOINT ["sh", "-c", "python /app/setup_dashboard.py & exec ../run.sh"]
```

Nodered Environment Variables

To use the same sensor thresholds defined in the JSON sensor configuration files (the one shown in the "*Sensor Configuration*" section) into the Node-Red flows, we modified the ENTRYPOINT of the Dockerfile as follows:

```
# Installa il parser json
RUN apk add --no-cache jq

USER node-red

# Load environment variable containing sensors configuration
ENTRYPOINT ["/bin/sh", "-c", "export SENSORS=$(cat /config/sensors-config.json | jq -c .) && exec npm start --cache /data/.npm -- --userDir /data"]
```

Starting from a new console and environment, a global variable named SENSORS is initialized with the data parsed from the JSON sensor configuration file. Then, the Node-RED process is executed in the same console, enabling the use of the SENSORS variable and the rest of the environment variables specified in the Docker Compose file within the Node-RED flow modules.

8. Conclusions

FLORA represents an IoT-Based system in smart-agriculture. This project provides a smart and reliable solution for monitoring and maintaining the health of plants in a farm, helping to prevent critical hazards or bad harvest.
This system not only ensures a proper plant life-style by monitoring critical parameters such as canopy density, chlorophyll content, plant height, humidity, temperature, pH, with any additional parameters a farmer might wish to keep an eye on, but also simplifies field maintenance and reduces the farmers' stress by minimizing the need for daily field inspections.
The possibility to set custom thresholds for the different monitored parameters enhances the system's functionality.
Furthermore, real-time data monitoring enables prompt responses to sudden environmental changes or plant-related hazards, preventing potential plant death or points of no return.
By incorporating IoT devices to their full potential, this project introduces innovative changes to traditional farming practices, paving the way for a simpler, smarter, and stress-free farming experience.

9. Future prospects

The future of FLORA could focus on several enhancements to further elevate its capabilities, such as integrating an autonomous system to fully automate plant irrigation and control temperature and humidity through planning and executing situation-tailored actions. Another possibility is developing a user-friendly mobile application for remote monitoring and control (even over the autonomous part of the system) which would enhance accessibility and convenience for users.