Crimson Publishers
Wings to the Research

Review Article

Designing the Software Layer of an IoT-Based Integrated Greenhouse Control System

ISSN: 2832-4463

Mona Kouhi*

Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Iran

***Corresponding author:** Mona Kouhi, Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Iran

Submission: June 07, 2025
Published: October 10, 2025

Volume 5- Issue 1

How to cite this article: Mona Kouhi*. Designing the Software Layer of an IoT-Based Integrated Greenhouse Control System. COJ Rob Artificial Intel. 5(1). COJRA. 000601. 2025.
DOI: 10.31031/COJRA.2025.05.000601

Abstract

In the agricultural industry, the process of planting and growing plants requires a lot of constant care and maintenance. The use of small greenhouses instead of large agricultural land leads to better control of the process and the production of high-quality agricultural products. Also, to control these distributed greenhouses from one point, it is necessary to use IoT technology. In this work, a software layer is developed, which includes a set of hardware devices (including a network of sensors, relays, an Arduino board, and a Raspberry Pi board) and a software control system to control the distributed greenhouses from one point remotely. The mentioned software layer (including a web application, middleware, and database) communicates with the hardware control center to remotely monitor and control greenhouse conditions such as temperature, humidity, light, etc. In addition, a scheduler programmed with Python multithreading is provided for the parallel execution of system commands. The SCADA (Supervisory Control and Data Acquisition) system is used to upgrade this system to a SCADA system or to integrate it into a SCADA system in the future. The proposed software architecture is presented with 4 + 1 views. The Architecture Tradeoff Analysis Method (ATAM) is applied to evaluate different scenarios. The trade-off analysis has proven the suitability and competence of the proposed architecture.

Keywords: Internet of things; Greenhouse control system; Middleware; SCADA; Software layer

Introduction

Today, in the agricultural industry, to produce better products, the process of planting and growing plants needs to be done with more care. For this purpose, instead of using large agricultural lands, smaller greenhouses can be used, which can lead to better control of the process and production of higher-quality agricultural products [1]. Also, to control these distributed greenhouses from one point, using Internet of Things technology is the best way. The concept of the Internet of Things uses network technology for controlling greenhouses remotely. The hardware part of this agricultural IoT includes temperature, humidity, and light sensors, as well as processors with big data processing capabilities; these hardware devices connect to short wireless communication technologies such as Bluetooth and Wi-Fi. The sensor network is created with web technology and combined in the form of the wireless sensor network to remotely control and monitor the data generated by the sensors [2].

In [3], an ON-OFF mechanism for controlling plant growth parameters was presented. In [4], various application areas and smartphone applications are briefly discussed. In [5], Arduino and Raspberry Pi were presented to develop monitoring and control systems with different parameter aspects. In [6] they have focused on monitoring three parameters namely soil moisture, temperature, and humidity using IoT technology.

In the proposed architecture, there are 5 components: Arduino board (including sensors, relays, relay board, and Arduino board), Raspberry Pi board, gateway, web server, and device browser. The collected data is sent from the sensors to the Arduino board, then to

the Raspberry Pi board, and from there to the database. The web server reads new data from the database and displays it in the device browser. Administrators or farmers can check the received data and the state of the greenhouse and make the desired changes or execute a command. When the user executes a command, the command is stored in the database. The Raspberry Pi board reads the data from the database and sends it to the Arduino board. Then according to the data sent, commands are applied to the desired relay and turn it on or off. In this system, we have used Python's multithreaded programming to execute multiple parts of our code simultaneously [7]. For example, in the proposed system, the commands come from 3 sources: 1. User Manual Command, 2. SchOnTime (commands that need to be executed at a specific time), 3. SchPeriodic (commands that need to be executed periodically). When these commands are executed, the program queues them and according to their arrival time they are executed in order and the commands apply to the requested relay. This feature speeds up the execution of the greenhouse commands, improves the system performance, and reduces costs.

Regarding these issues, we present an integrated control system for greenhouses that includes both modeling and architecture of a smart greenhouse. In the proposed system, we have used IoT technology in the development of greenhouse integrated control system to fulfill the following purposes:

A. Provide an open platform for connecting IoT nodes in the greenhouse environment.

B. Intelligently control greenhouses and IoT nodes.

C. Enable access to multiple geographically distributed greenhouses from one point.

D. Reduce the production cost of agricultural products and improve crop quality.

To achieve these goals, we developed an IoT-based system that can connect all IoT nodes and access the greenhouse environment to control it from one point remotely. This system can not only reduce the cost but also increase the quality of agricultural products.

The developed system is superior to existing IoT-based greenhouse management systems for the following points:

a. Queuing of commands that are entered into the system simultaneously (commands such as User Manual Commands, SchOnTime Commands, and SchPeriodic Commands)

b. Mapping the proposed system to the SCADA system for future expansion of this system to SCADA

c. Using ATAM to demonstrate the suitability and suitability of the proposed architecture

This paper is organized as follows: Section 2 presents previous related work. Section 3 deals with the modeling and architecture of the greenhouse integrated control system. Section 4 deals with the architecture evaluation methods to study the quality characteristics of the system. Finally, Section 5 concludes the paper. A list of Acronyms used in the manuscript is mentioned in Table 1.

Table 1: Acronyms & glossary.

Acronym	Description
HMI	Human-Machine Interface
RTU	Remote Terminal Unit
PLC	Programmable Logic Controller
SCADA	Supervisory Control and Data Acquisition
API	Application Programming Interface
REST API	RESTful web API
CRUD	Create, Read, Update, Delete
UML	Unified Modeling Language
FIFO	First-In, First-Out
I/O	Input/Output
TLS	Transport Layer Security
mTLS	Mutual TLS
PIN	Personal Identification Number
AC	Air Conditioner
TDS	Total Dissolved Solids
ATAM	Architectural Tradeoff Analysis Method

Related Works

In [8], agricultural IoT system solutions are proposed for monitoring the growth of tomato fruit by extending the connection with Slack Bot API to notify farmers about the status of tomatoes. They also performed image analysis by combining Deep Learning for tomato fruit detection, image processing for colour feature extraction, and machine learning for 6 growth stage classifications. In their tomato recognition system, they were able to recognize both green and red fruits from background images. In their system, they did not develop a user interface for the administrator and the farmer to control the greenhouse conditions. However, in our proposed system, we developed a web application for the system to manage greenhouse conditions remotely.

In [9], a mobile greenhouse environment monitoring system based on IoT architecture is proposed. For the first time, a Raspberry Pi and an Arduino chip are combined for environmental monitoring in agricultural greenhouses, with the former serving as a data server and the latter as a master chip for the mobile system. A four-layer system architecture is built, which provides a motion control function. And all four layers of the architecture are used on the mobile system. The system can be operated in three modes to realize automatic multi-point environment information acquisition for greenhouses and crop image acquisition at a low cost. In their system, they haven't developed middleware for their devices to connect and receive and send data simultaneously.

In [10], high-tech monitoring systems are used in Mediterranean greenhouse structures with little technical effort. Their project included 1) a network of sensors for climate, root zone, and plant monitoring, 2) mathematical models for plant management and preliminary simulations, 3) closed cultivation systems, and

4) innovative devices for plant defense and nutrition based on NTP. Integration of the above technologies and their control as a whole into the same intelligent control system will enable precise management of the growing environment with reduced chemical use and high quality and yield of the final product. In this project, the necessary changes are made manually in the greenhouse after sending the data from the sensors through the web server. However, in our proposed system, these commands are executed completely automatically using our components.

In [11], IoT applications in smart greenhouses and their benefits are mentioned. In this paper, it is shown that IoT enables control of environmental parameters, humidity, water and energy consumption, temperature data, CO2 levels, etc. It also minimizes pesticide treatments to prevent important diseases that plants are infected with and the ability to monitor and control favourable crop conditions in a protected environment from damage caused by climate change or other weather conditions.

In [12], the different operating strategies to optimize and improve the energy balance were investigated. They integrated a new system with the SCADA software and advanced algorithms such as Model Predictive Control using the WEST computing platform.

In [13], a system for improving the quality and upgrading of products is proposed that uses temperature sensors to monitor air and soil moisture, which are integrated into the microcontroller of the rubber foot and provide the results through the application installed on the smartphone. The results of the sensors are entered into the database installed on the Piebars, which can be used to analyze the agricultural data.

In [14], a system for controlling and monitoring greenhouse temperature using IoT technologies is proposed. In this system, a Petri net model was used to both monitor the greenhouse environment and generate a suitable reference temperature that is later sent to a temperature control block. They also developed an energy-efficient system design that processes large amounts of IoT Big Data collected by sensors using a dynamic graph data model that can be used for future analysis and prediction of production, crop growth rate, energy consumption, and other related issues. In their system, they have not provided a user interface for remote control of the greenhouse.

In [15], the integration of a third-generation smart embedded system based on the Raspberry Pi, climate sensors, and IoT analytics is proposed. Their proposed work can be physically installed in a greenhouse environment to record climate parameter data. The gateway nodes have control over forwarding this data to agricultural experts via a web browser on the Internet. Based on the received information, ES activates intelligent decision-making by implementing an appropriate arrangement to control climate parameter values. In their study, they didn't consider the error detection capability, as this could compromise the safety of the system and the greenhouse.

In [16], an IoT-based smart solar agricultural robot was proposed. Their system operated entirely on solar energy. The Raspberry Pi was the heart of the Agrobot and monitored the humidity and temperature of the farm, while the Arduino controlled the six DC motors. A solar panel in the battery also provided the power needed to run the Agrobot. Your system is not safe and can cause problems.

In [17], a system is proposed that receives three parameters from the sensors and activates the relays when the actual values are above the thresholds. Moreover, these values are stored in the cloud database so that they can be retrieved at any time and from anywhere. The automatic control of climatic conditions in the greenhouse is also illuminated. For example, various seasonal crops can only be grown under certain conditions, according to their research. Onions, garlic, shallots, etc. are the winter crops that need cold conditions to grow, and cucumbers, melons, etc. are the summer crops that need temperate or hot climatic conditions. Their prototype consisted of humidity sensors, temperature, and humidity sensors, a Raspberry Pi, and water pipes to supply water from tanks controlled by DC motors.

In [18], an integrated farm monitoring system using a smartphone application and the Internet of Things is presented. With this system, farmers can remotely monitor soil moisture, leaf wetness duration, soil pH, temperature, and ambient humidity, and manage the data received from the sensors using a mobile app and a web app. In addition, their proposed system can analyze the weather and soil conditions in a specific area where the plant is located and provide new insights for decision-making. In their proposed system, they haven't developed a web app for remote control.

In [19], an automated intelligent greenhouse based on an Adaptive Neuro-Fuzzy Inference System (ANFIS) and the Internet of Things (IoT) is proposed. Their work combines IoT technology with a fuzzy set to identify data threats in network transmission. The result of this work is an efficient, cost-effective, secure, and easy-to-use greenhouse maintenance system that ensures data traceability and durability for tailored quality indoor agriculture.

In [20], a literature-centered study on localization methodologies is presented for humanoid and autonomous robots, emphasizing sensor fusion, perspective transformation, and SLAM-based approaches to achieve robust real-time localization in dynamic and GNSS-denied environments. It has surveyed a broad range of works-from probabilistic localization and graph optimization to multi-sensor fusion, visual-inertial SLAM, and 3D reconstruction-to highlight how integrated sensing and perspective transformations contribute to accurate pose estimation and robust perception for humanoid robots and related platforms.

In [21], the integration of IoT, Digital Twin (DT), and Augmented Reality (AR) technologies to improve the management, monitoring, and control of agricultural greenhouses is investigated. In this research, a prototype IoT system is developed that provides real-time visualization and monitoring of sensor data collected from microcontroller-based sensor nodes installed in the greenhouse. The data is transmitted using the MQTT protocol, and AR is also used to visualize and control the greenhouse environment.

In [22], the use of the Architectural Trade-Off Analysis Method (ATAM) in the evaluation and integration of complex Systems of Systems (SoS) is investigated, with a specific telecommunications

SoS as a case study. It also uses ATAM qualitatively to identify and analyze architectural trade-offs that affect qualities such as performance, reliability, and maintainability in the context of SoS. This research is relevant to Cyber-Physical Systems (CPS) because many CPS are essentially systems that consist of multiple interconnected components. The use of ATAM in this context has contributed to informed architectural decisions to manage complexity, resolve conflicts, and improve the effectiveness of system integration.

In [23], a soft motion control platform based on a real-time Linux system is proposed. The platform aims to provide strong openness, easy operability, and compatibility with various sensors. In this research, real-time Linux is integrated with Ethernet fieldbus technologies to build a versatile motion control system.

Proposed System

The IoT-based greenhouse integrated control system consists of a hardware section that includes sensors, relays, a Raspberry Pi board, an Arduino board, and a software section that controls the hardware devices. In the following, these sections will be explained.

Proposed architecture

In this proposed system, we have tried to use the structure of the Supervisory Control and Data Acquisition (SDACA) system to design our system [24,25]. Therefore, we mapped our components to SCADA components. SCADA has 5 components: Supervisory Computers, Remote Terminal Units (RTU), Programmable Logic Controllers (PLC), Communication Infrastructure, Human Machine Interface (HMI) [24,26].

According to the SCADA components, we have considered the following components in the proposed system:

a) Web server: The core of our system that receives information from hardware devices and sends them the correct commands.

b) Arduino board: Receives the data from sensors and relays and sends it to the Raspberry Pi board.

c) Raspberry Pi board: The Raspberry Pi board receives the data from the Arduino board and sends it to the web server.

d) Gateway: Connects the web server to the Raspberry Pi board and the Arduino board to transfer data between them.

e) Device browser: A device browser is a software application that presents information to the user in a graphical format. In this system, it is developed using Python programming language (Flask framework). Figure 1 shows the components of the proposed system.

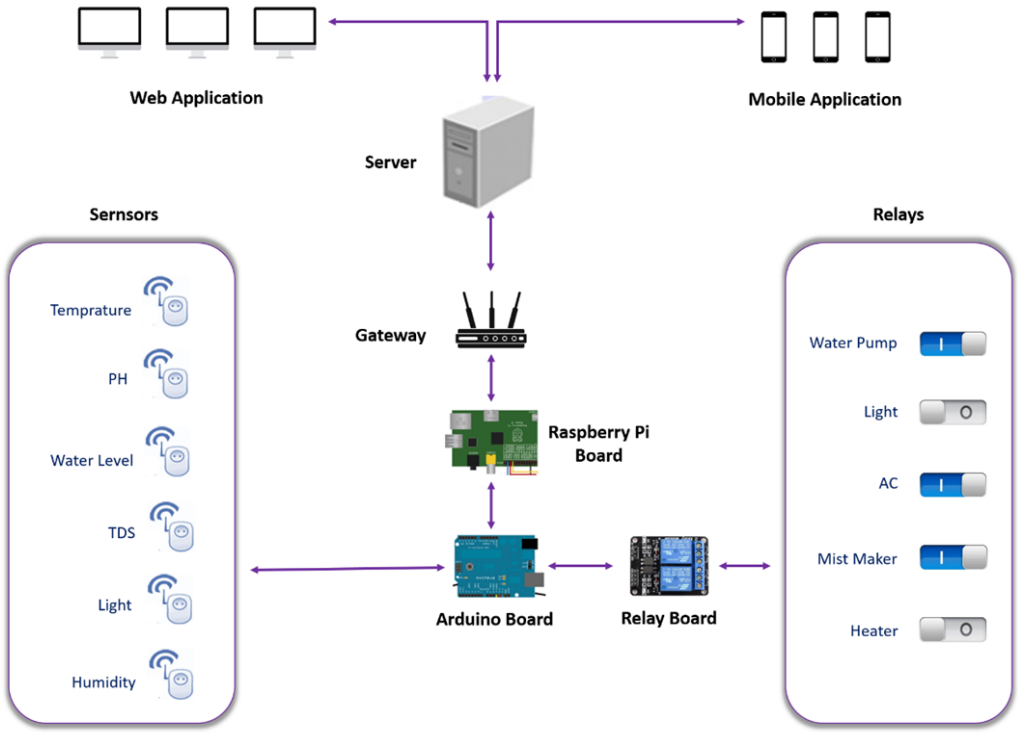

Figure 1: The components of the proposed system.

The following are some useful features required to build a SCADA system:

1. The ability to communicate with devices and PLCs. This is required for real-time monitoring and control of devices.
2. A sub-framework for developers to implement communication protocols not provided by the framework.
3. A user framework for control and monitoring.

4. Built-in support for connecting to and using common SQL databases, such as MySQL, SQL Server, Oracle, SQLite, and PostgreSQL.

5. Ability to access and use HTTP-based REST (Representational State Transfer) APIs (Application Programming Interfaces) provided by external applications. This enables SCADA applications to send requests to external applications and receive responses.

6. Ability to create HTTP-based REST (Representational State Transfer) APIs (Application Programming Interfaces) that external applications can access and use. This allows SCADA applications to receive information requests from external applications and send responses.

7. Full support for sending and receiving emails, including attachments.

According to the above points, we have performed the following functionalities in the proposed system to adapt our system to the SCADA system:

a. Connecting between devices such as sensors, relays, Arduino boards, and Raspberry Pi boards through USB, cable, power, and wireless internet and transferring data between them.

b. Developing a web server using Python programming language (Flask framework) as the core of this system.

c. Developing a web application that allows users to control and monitor the state of the greenhouse and issue commands.

d. Using SQLite database to store data is very easy and fast.

e. Implementing Restful APIs to send requests to external applications and receive responses.

f. Implementing Restful APIs to receive requests to external applications and send responses.

g. Creating a log file to store data at specific times on the local PC and creating a notification service to inform the system administrator about greenhouse conditions as well.

The following table (Table 2) shows the REST APIs of the system components and their message formats.

Table 2: Message format of the system components.

Communication Flow	Endpoint / Message Format (JSON Snippet)
Sensor → RPi	POST /sensor/datajson {“sensor id”: “S1”, “value”: 23.5, “timestamp”: “2025-08-17T12:00:00Z”}
RPi → DB	POST /db/storejson {“sensor_id”: “S1”, “value”: 23.5, “timestamp”: “2025-08-17T12:00:00Z”}
DB → Server	GET /data?sensor_id=S1&from=2025-08-17T00:00:00Z&to=2025-08-17T12:00:00ZResponse: json [{“sensor_id”: “S1”, “value”: 23.5, “timestamp”: “2025-08-17T12:00:00Z”}]
Server → RPi	POST /rpi/commandjson {“command”: “activate”, “device”: “fan”, “duration”: 300}
RPi → Arduino	Serial message: json {“action”: “relay_on”, “relay_id”: 2}
Arduino → Relays	Electrical signal (no JSON, hardware level)

The proposed software architecture is described by diagrams and charts showing the components and their relationships and constraints. The proposed software architecture is expressed by using 4 + 1 views [27].

The four views of this model are the logical view, the process view, the development view, and the physical view. In addition to these views, the model also uses use cases to describe the architecture.

a) Logical view: This view shows the capabilities that the designed system provides to end users.

b) Process view: This view shows the dynamic aspects of the system and describes the system processes and their interactions as well as the runtime behavior of the system. This view covers the concurrency, distribution, performance, and scalability aspects.

c) Development view: This view describes the designed system from the software developers’ point of view and deals with software management. This view is also referred to as the implementation view.

d) Physical view: This view shows the system from a system engineer’s perspective and focuses on the topology of the physical layer software components and their physical connections.

e) Scenarios: Scenarios describe the architecture by describing the sequence of relationships between objects as well as the processes. This view is also referred to as the use case view [27].

The following are 4 + 1 views for the proposed architecture.

Usecase view: Different systems may have different requirements, and developers are responsible for assessing each need and designing a system that meets those requirements. These requirements can generally be divided into two categories: functional requirements and non-functional requirements. Functional requirements refer to the ability of the system to perform the task for which it was designed. For each of the subsystems, several primary and functional requirements must be satisfied [28].

For a better understanding of the proposed system behavior, the use case diagram [29] of this system is shown in Figure 2.

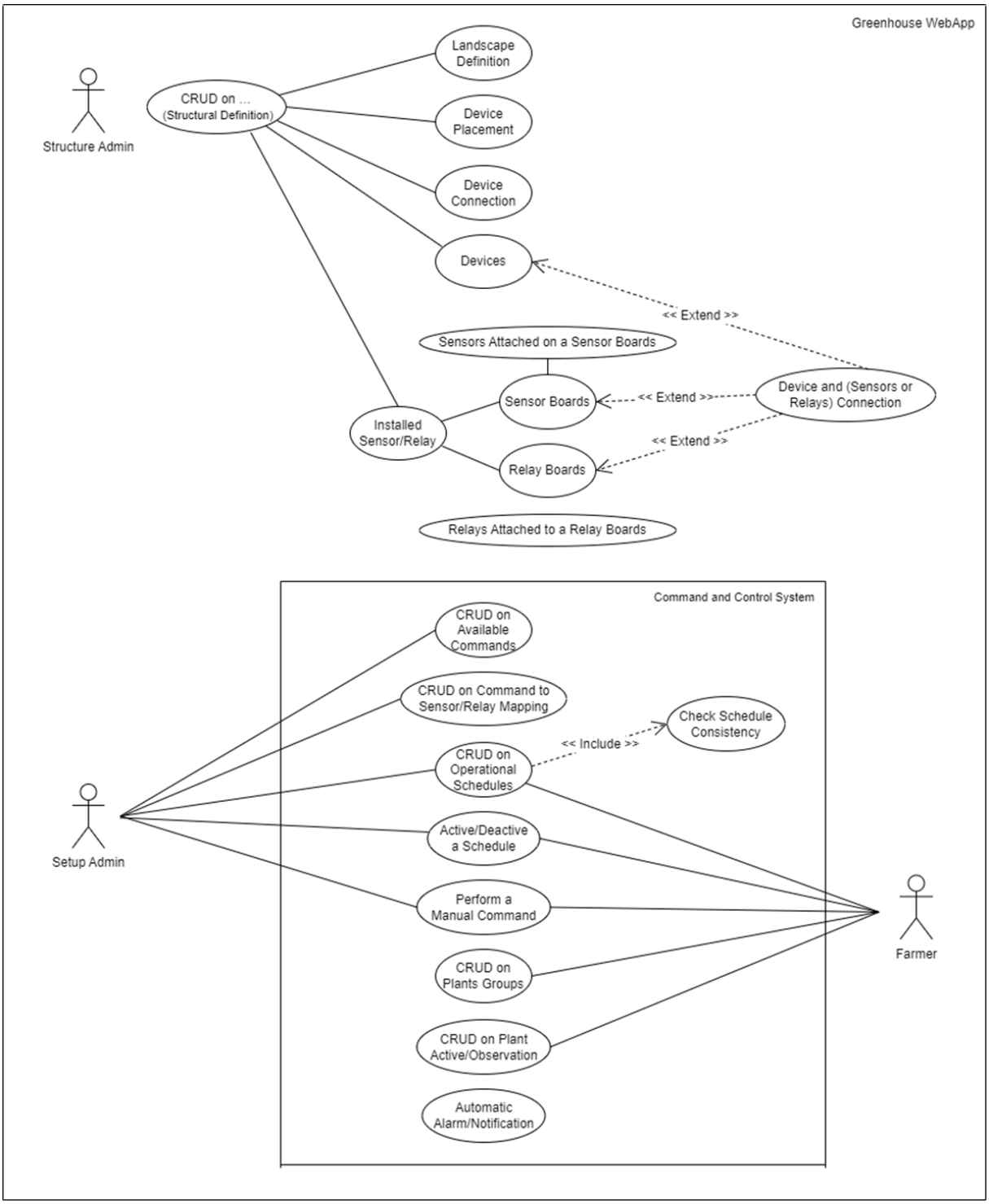

Figure 2: General use case diagram for the proposed system.

According to Figure 2, the Structure Admin is generally responsible for the overall system structure and focuses on the landscape, devices, connections, location of devices, and installed sensors/relays. The Setup Admin focuses more on commands, scheduling, and task management. The third component is the user (farmer) who can manage schedules, and plants and execute manual commands.

Deployment view: To describe the deployment view, we used the deployment diagram. A deployment diagram shows the hardware components, software components, and the connections of the different parts of the system [30]. Figure 3 shows how the components are arranged in the system.

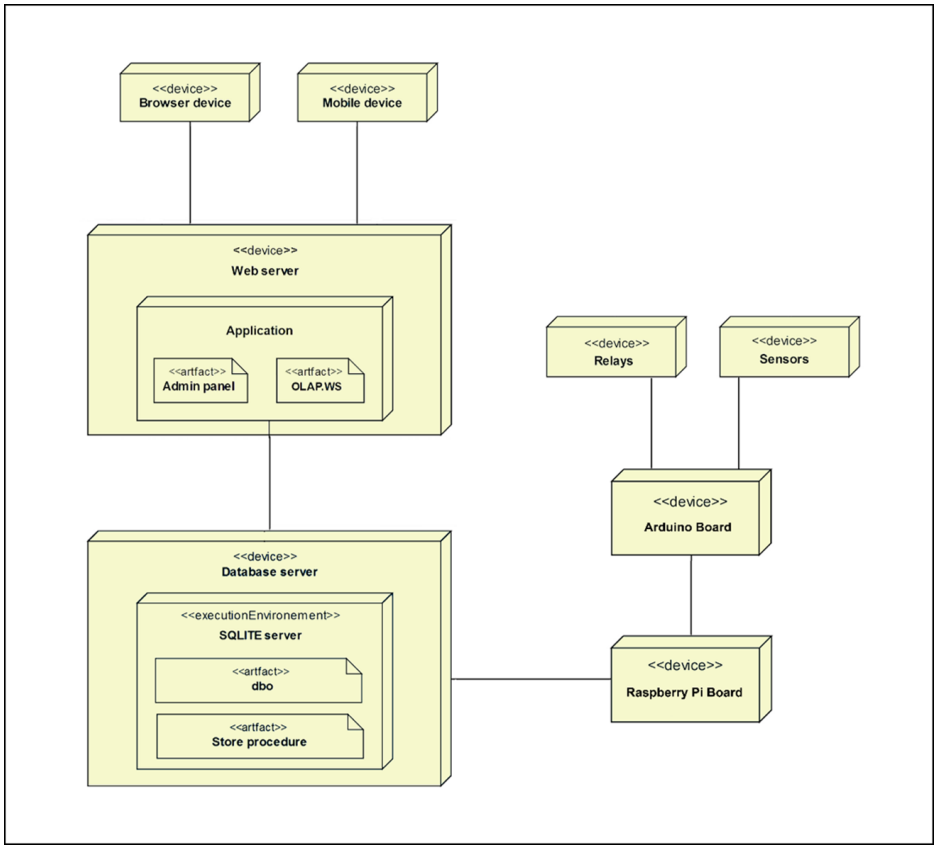

Figure 3: System deployment diagram.

In order to install and make the system available in a real-world environment, the purpose of each part and component of the system and their location must be determined. The components of this system include the following 5 components:

A. Device browser component (component for mobile devices): Connection between users and the web server.

B. Web server component: receiving data and executing functions and commands and controlling all conditions.

C. Gateway component: Transferring data from the Raspberry Pi to the web server and vice versa.

D. Raspberry Pi board component: data processing and comparison: when data is sent from the sensor to the Raspberry Pi, the Raspberry Pi stores it. On the other hand, sensor data is expected to change when a command is executed, and Raspberry Pi stores the new data as well. One of the most important tasks of Raspberry Pi in this system is to compare the stored data (previous data) with the new data to prevent system failure.

E. Arduino board component: Data transfer between sensors, relays, and Raspberry Pi.

a. Relay component: Changing device state to "on" or "off".

b. Sensor component: Sending device data to the Arduino board.

All components can be accessed via wireless networks, cables, and USB. In the proposed system, the components are connected as follows:

1. The sensors and relays are connected to the Arduino board through a cable.

2. The Arduino board and the Raspberry Pi board are connected via USB.

3. The Raspberry Pi board is connected to the web server via the gateway.

4. Finally, the device browser component is connected to the web server component via the Internet.

Sequence diagram: A sequence diagram is used to represent the routine sequence of system processes. It specifies a sequence of events to perform an action. This diagram is used in the analysis and design phase to understand how the proposed system works [31].

A. Sequence diagram from the perspective of the remote user: The remote user who wants to use the system first logs in. If the entered username and password are correct, the user can select an option from the menu. However, if the entered username and password are not correct, the user can try again. On the main page, the user can select 3 options: 1. Get the latest greenhouse data. 2. Monitor the live status using the live camera. 3. CRUD settings of the greenhouse. And then the user logs out. Figure 4. shows the sequence diagram from the perspective of the remote user.

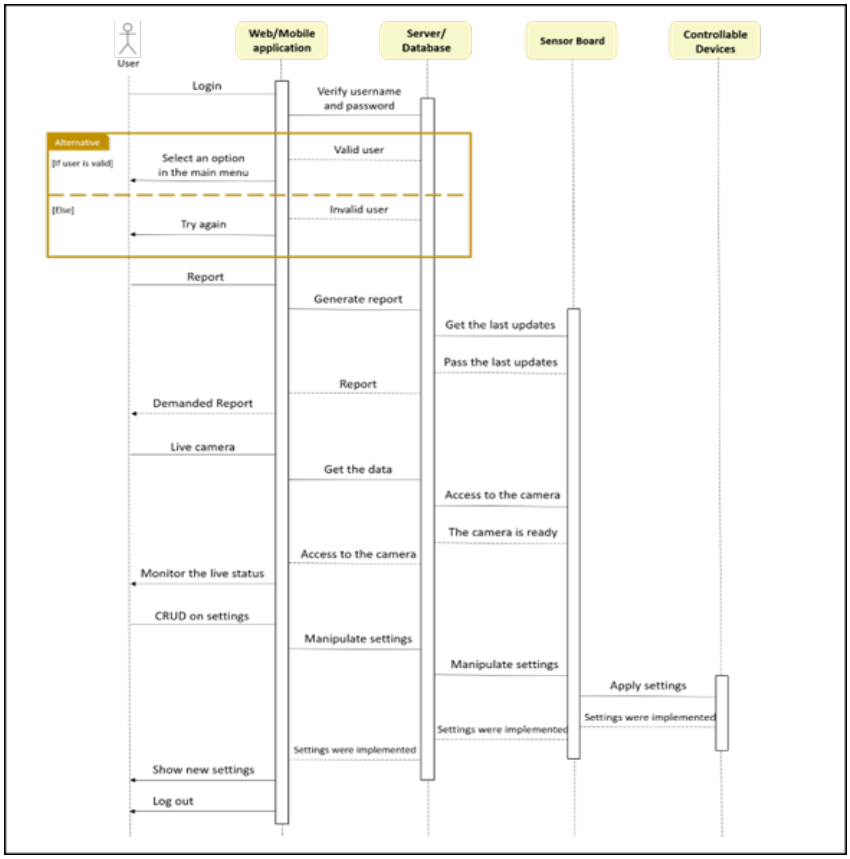

Figure 4: Sequence diagram from the perspective of the remote user.

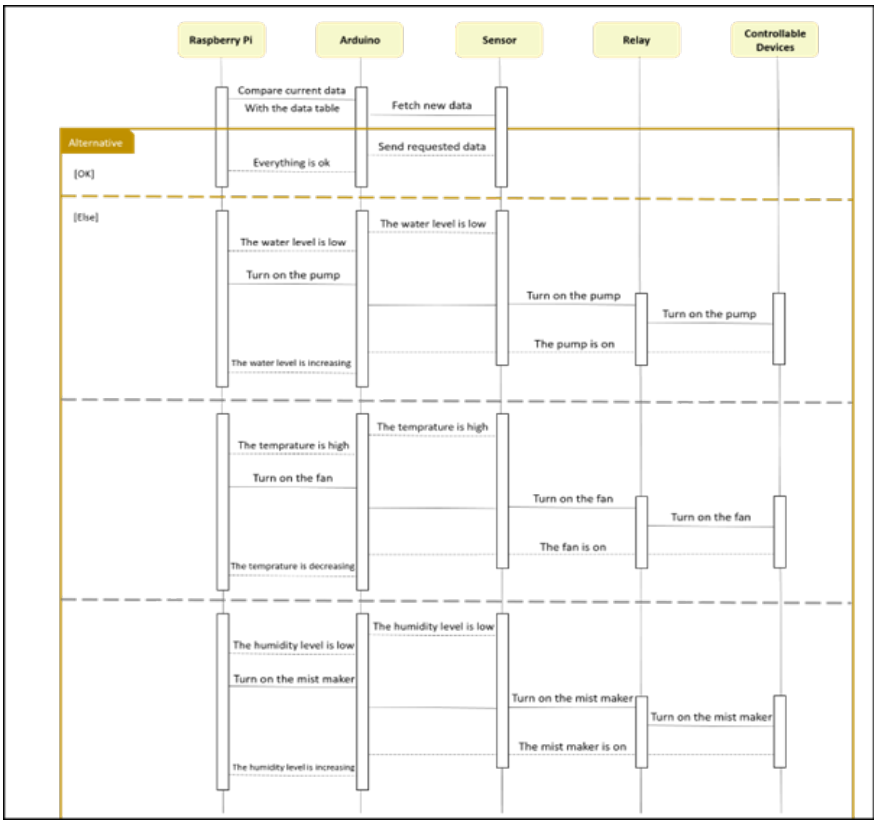

Figure 5: Sequence diagram from the system perspective.

B. Sequence diagram from the system perspective: To check the general condition of the greenhouse, the Raspberry Pi compares the data sent by the sensors with the data table. If the conditions in the greenhouse are suitable, no action is taken. However, if even one of the conditions was not suitable, a command must be executed. For example, if the water level is low, the Raspberry Pi sends a command to the relay via the Arduino to turn on the pump. After the pump is turned on, the water level is raised. Other cases are shown in Figure 5.

System workflow: According to Figure 6, the workflow of the system from the remote user's point of view consists of 7 sections:

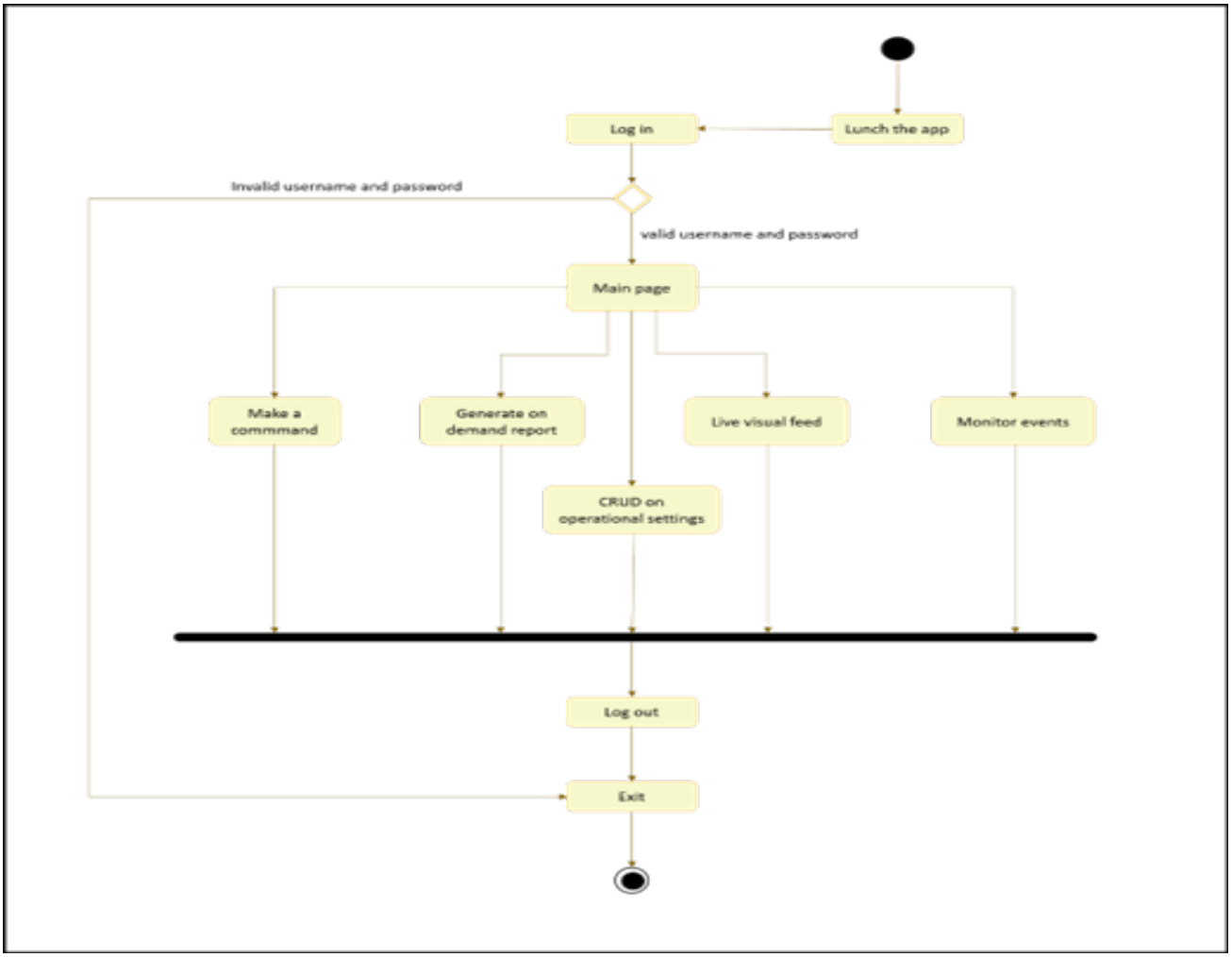

Figure 6: System workflow from the perspective of the remote user.

A. To access the system, the user enters his username and password and sends the information to the central server, which verifies the entered information with the information of the approved users in the database. If the information is correct, the user is redirected to the main menu of the system.

B. The user is redirected to the main page of the system and can select the desired option from the defined options and is redirected to the page associated with each option to continue the process. These options include checking the current system status, generating reports, viewing the greenhouse environment live, and attempting to manually disable and enable any of the system settings.

C. If the user selects the option to view the system status, the corresponding request is sent to the central server and the latest information collected by the sensors is sent to the user and displayed to the user in the web app.

D. If the user selects the report generation option, he will be redirected to the corresponding page. By applying the desired filters, the report request is sent to the server, and the information is retrieved from the database and sent to the user. In this section, the user can download these reports to their smart device.

E. If the user selects the option of live video of the greenhouse, the request is sent to the server, and if there is no error in the communication, the user can view the video online.

F. If the user selects the option of manual settings, under certain circumstances he can disable some elements that can

be disabled, but it is recommended to leave the activities to the system and not make any changes.

G. The user can exit the system by selecting the Exit (Logout) option (Figure 6).

Design a scheduler

In the proposed system, a scheduler is designed and programmed using Python's multithreaded programming [32]. The system uses a queue to store incoming commands from three sources: User Manual Command, SchOnTime command (commands that must be executed at a specific time), and SchPeriodic command (commands that must be executed periodically). Commands are executed in order of their arrival time, suggesting a First-In, First-Out (FIFO) queueing policy. However, since the commands are executed "according to their arrival time," the queue incorporates a timestamp to prioritize commands, indicating a FIFO policy with timestamp-based priority.

Figure 7 shows the process of queuing the commands from their sources to the relays in the designed schedule.

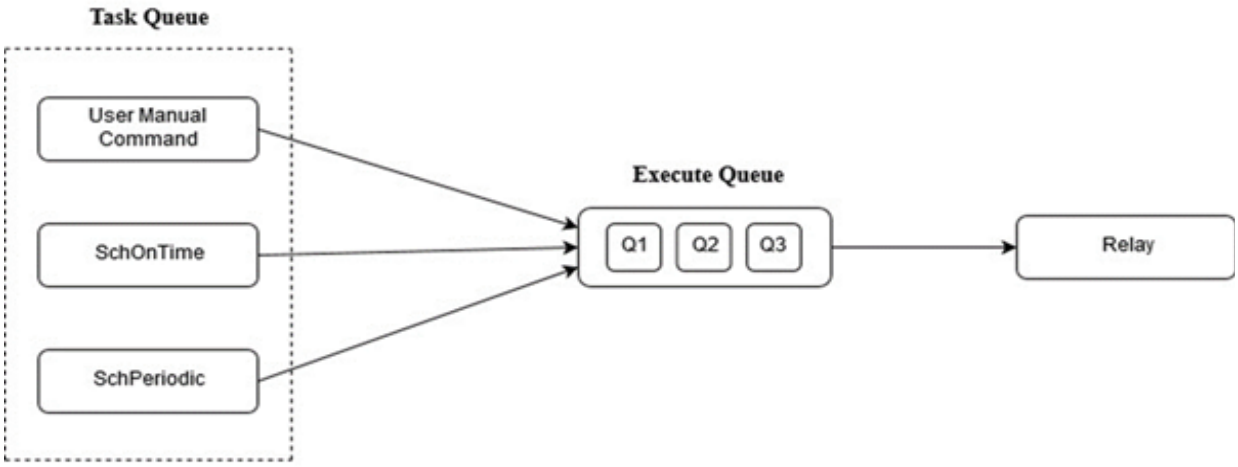

Figure 7: Queue in the designed schedule.

```python
import threading

shared_resource = 0

lock = threading.Lock()

def increment():
    global shared_resource
    for _ in range(100000):
        with lock:
            shared_resource += 1

UserManualCommand = threading.Thread(target=increment)
SchOnTimeCommand = threading.Thread(target=increment)
SchPeriodicCommand = threading.Thread(target=increment)

UserManualCommand.start()
SchOnTimeCommand.start()
SchPeriodicCommand.start()

UserManualCommand.join()
SchOnTimeCommand.join()
SchPeriodicCommand.join()
```

Figure 8: Lock function for command scheduling.

Furthermore, since this system is a multi-threaded system and in multi-threaded systems, threads may access shared resources simultaneously, leading to possible conflicts and data corruption, we used synchronization mechanisms to coordinate the execution of threads to maintain data integrity and prevent possible corruption. So, we used the lock function (or mutex), which is a primitive synchronization function that allows only one thread to access a shared resource at a time. So, we used the lock function (or mutex), which is a primitive synchronization function that allows only one thread to access a shared resource at a time. Figure 8 shows a code snippet that uses the lock function to prevent priority conflict.

On the other hand, since many commands to control devices (Lights, water pump, etc.) may reach the actuators at the same time, for this purpose

a. Each actuator command is executed in its own action thread. This helps to isolate long-running or blocking I/Os without stopping the main program.

b. Use the _do_with_timeout function, which causes the hardware call to be executed in a separate thread and imposes a timeout. If the hardware operation does not finish within the timeout, a Timeout Error error occurs.

c. If a timeout occurs for executing a command, the command can be retried up to the number of retries.

d. When turning on an actuator with a specified timeout, the code keeps the actuator on for the same number of seconds (unless canceled), then automatically turns it off.

e. Central management Actuator Manager keeps track of active actions and prevents overlapping and conflicting commands in an actuator/action pair by canceling any existing tasks before starting a new one.

Figure 9 shows a portion of the program code that demonstrates how threads handle long-running actuator commands.

```python
import threading
import time
import queue
from enum import Enum
from typing import Callable, Optional
import RPi.GPIO as GPIO


class ActuatorName(Enum):
    LIGHTS = "lights"
    WATER_PUMP = "water_pump"
    AIR_CONDITIONER = "air_conditioner"
    HEATER = "heater"
    MIST_MAKER = "mist_maker"

class CommandResult(Enum):
    SUCCESS = "success"
    TIMEOUT = "timeout"
    FAILURE = "failure"
    CANCELLED = "cancelled"

class ActuatorDriver:
    def __init__(self, name: ActuatorName):
        self.name = name

    def activate(self) -> None:
        self._activate()

    def deactivate(self) -> None:
        self._deactivate()

    def _activate(self) -> None:
        raise NotImplementedError

    def _deactivate(self) -> None:
        raise NotImplementedError

class GPIO(ActuatorDriver):
    def _activate(self) -> None:
```

```python
    def _activate(self) -> None:
        print(f"[{self.name.value}] ACTIVATED")
        time.sleep(0.5)

    def _deactivate(self) -> None:
        print(f"[{self.name.value}] DEACTIVATED")
        time.sleep(0.2)

class Action(threading.Thread):
    def __init__(
        self,
        actuator: ActuatorDriver,
        action: str,  # "on" or "off"
        duration: Optional[float] = None,
        timeout: float = 5.0,
        retries: int = 0,
        on_complete: Optional[Callable[[ActuatorDriver, CommandResult], None]] = None
    ):
        super().__init__()
        self.actuator = actuator
        self.action = action
        self.duration = duration
        self.timeout = timeout
        self.retries = retries
        self.on_complete = on_complete
        self._cancel_event = threading.Event()
        self._result: Optional[CommandResult] = None

    def run(self):
        attempt = 0
        while attempt <= self.retries and not self._cancel_event.is_set():
            attempt += 1
            try:
                if self._cancel_event.is_set():
                    self._set_result(CommandResult.CANCELLED)
                    break

                if self.action == "on":
                    self._do_with_timeout(self.actuator.activate, "activate")
                    if self.duration and self.duration > 0:
```

```
                    if self.duration and self.duration > 0:
                        start = time.time()
                        while (time.time() - start) < self.duration:
                            if self._cancel_event.is_set():
                                break
                            time.sleep(0.1)
                        if not self._cancel_event.is_set():
                            self._do_with_timeout(self.actuator.deactivate, "deactivate after duration")
                    else:
                        pass
                elif self.action == "off":
                    self._do_with_timeout(self.actuator.deactivate, "deactivate")
                else:
                    raise ValueError("Unsupported action")
                self._set_result(CommandResult.SUCCESS)
                break
            except TimeoutError:
                if attempt > self.retries:
                    self._set_result(CommandResult.TIMEOUT)
                    break
                else:
                    continue
            except Exception as ex:
                print(f"Actuator error on {self.actuator.name}: {ex}")
                if attempt > self.retries:
                    self._set_result(CommandResult.FAILURE)
                    break
                else:
                    continue

        if self.on_complete:
            self.on_complete(self.actuator, self._result or CommandResult.FAILURE)

    def _do_with_timeout(self, func: Callable[[], None], phase: str) -> None:
        finished = threading.Event()
        exception_holder = {}

        def target():
            try:
                func()
```

```
                func()
            except Exception as e:
                exception_holder["exc"] = e
            finally:
                finished.set()

        t = threading.Thread(target=target, name=f"{self.actuator.name.value}-{phase}")
        t.start()
        t.join(self.timeout)

        if not finished.is_set():
            raise TimeoutError(f"{phase} timed out after {self.timeout}s")
        if "exc" in exception_holder:
            raise exception_holder["exc"]

    def _set_result(self, res: CommandResult):
        self._result = res

    def cancel(self):
        self._cancel_event.set()

class ActuatorManager:
    def __init__(self, drivers: dict[ActuatorName, ActuatorDriver]):
        self.drivers = drivers
        self.lock = threading.Lock()
        self.active_actions: dict[str, Action] = {}

    def _make_key(self, actuator: ActuatorName, action: str) -> str:
        return f"{actuator.value}:{action}"

    def run(self, actuator: ActuatorName, action: str, duration: Optional[float] = None,
            timeout: float = 5.0, retries: int = 0, on_complete: Optional[Callable[[ActuatorDriver, CommandResult], None]] = None):
        if actuator not in self.drivers:
            raise ValueError(f"Unknown actuator: {actuator}")

        driver = self.drivers[actuator]
        key = self._make_key(actuator, action)

        with self.lock:
            if key in self.active_actions:
```

```
            if key in self.active_actions:
                self.active_actions[key].cancel()
                del self.active_actions[key]

            task = Action(
                actuator=driver,
                action=action,
                duration=duration,
                timeout=timeout,
                retries=retries,
                on_complete=on_complete,
            )
            self.active_actions[key] = task
            task.start()

    def wait_all(self, timeout: Optional[float] = None) -> None:
        start = time.time()
        while True:
            with self.lock:
                if not self.active_actions:
                    return
                running = list(self.active_actions.values())
            for t in running:
                t.join(timeout=0.1)
            if timeout is not None and (time.time() - start) > timeout:
                return
            time.sleep(0.05)


if __name__ == "__main__":
    drivers = {
        ActuatorName.LIGHTS: GPIO(ActuatorName.LIGHTS),
        ActuatorName.WATER_PUMP: GPIO(ActuatorName.WATER_PUMP),
        ActuatorName.AIR_CONDITIONER: GPIO(ActuatorName.AIR_CONDITIONER),
        ActuatorName.HEATER: GPIO(ActuatorName.HEATER),
        ActuatorName.MIST_MAKER: GPIO(ActuatorName.MIST_MAKER),
    }

    manager = ActuatorManager(drivers)
```

```
    manager = ActuatorManager(drivers)

    def on_complete(driver: ActuatorDriver, result: CommandResult):
        print(f"Completion: {driver.name.value} -> {result.value}")

        manager.run(
        ActuatorName.LIGHTS,
        action="on",
        duration=10.0,
        timeout=3.0,
        retries=1,
        on_complete=on_complete
    )

    manager.run(
        ActuatorName.WATER_PUMP,
        action="on",
        duration=None,
        timeout=5.0,
        retries=0,
        on_complete=on_complete
    )

    time.sleep(2)
    manager.run(
        ActuatorName.WATER_PUMP,
        action="off",
        duration=None,
        timeout=5.0,
        retries=0,
        on_complete=on_complete
    )

    manager.wait_all(timeout=30.0)
```

Figure 9: Handling actuator commands by threads.

Security

In this section, system security is examined.

Authentication and access control:

A. Intended controls:

a. API keys/JWTs: Token-based authentication is used to identify and authorize both data sources (sensors) and control recipients (actuators) interfacing with the Raspberry Pi and the web server.

b. Scopes and roles: Tokens are restricted to specific operations, such as:

I. Read access for sensor data streams

II. Write/execute access for actuator commands

c. Token lifetimes: Short-lived tokens are preferred with refresh capabilities to minimize risk if a token is compromised.

d. Replay protection: Nonces or time-bound tokens are implemented for critical command paths to mitigate replay attacks.

B. Known limitations:

a) Token storage security on resource-constrained devices (sensors/actuators) varies; Secure storage methods are chosen to suit the hardware.

b) Token validation relies on secure key management; Rotations and revocations are maintained to prevent the use of compromised keys.

1. c) API keys for non-human clients may be less expressive than JWTs/OAuth2; Strict monitoring of API keys is intended.

Device whitelisting:

1. Intended controls:

I. A whitelist of approved devices (sensors, the Raspberry Pi edge gateway, and authorized client endpoints) allowed to communicate with the system is maintained.

II. Device identifiers are bind to credentials (per-device keys or certificates) to establish trust at the edge and between the web server and gateway.

III. Revocation paths are implemented for decommissioned or compromised devices.

2. Known limitations

I. Whitelist management can be challenging in large or dynamic deployments (e.g., seasonal sensors or movable units).

II. Compromised whitelisted devices plus stolen credentials can broaden attack surface.

III. Device identity spoofing risk; Device certificates are considered.

Transport security (gateway ↔ server):

1. Intended controls

a) Data is encrypted in transit using TLS.

b) Mutual TLS (mTLS) is implemented to authenticate both the Raspberry Pi gateway and the web server endpoints.

c) Strong certificate validation is implemented and a PIN code is included to prevent MITM.

d) Secure channels are ensured for both sensor data uplinks (edge to server) and actuator command paths (server to edge).

2. Known limitations

a) TLS setup and certificate lifecycle management (issuance, rotation, revocation) require coordinated operations; Coordinated operations are in place.

b) mTLS adds overhead on the edge device; It has been ensured that provisioning and rotation processes are robust.

c) Proxies, load balancers, or network devices must preserve end-to-end security; Correct configurations are in place to ensure and enhance protections.

The hardware view

The hardware part of the proposed system consists of 2 components: an Arduino board (with sensors, relays, relay board, and Arduino board) and a Raspberry Pi board. The connection between these components is such that the sensors send the data of temperature, water level, brightness, and humidity to the Arduino board, the Arduino board sends the data to the Raspberry Pi board, the Raspberry Pi board sends the data to the database and from there the data is sent to the server. After viewing the data on the web, the user (or administrator) can execute a command that can turn a relay on or off, set a time for a command, etc. Each of these changes is stored in the database, then the Raspberry Pi retrieves the data from the database and sends it to the Arduino to finally apply it to the desired relay. The data between these components is transferred via wireless internet, USB, power, and cable. Figure 10 shows the hardware components of the proposed system.

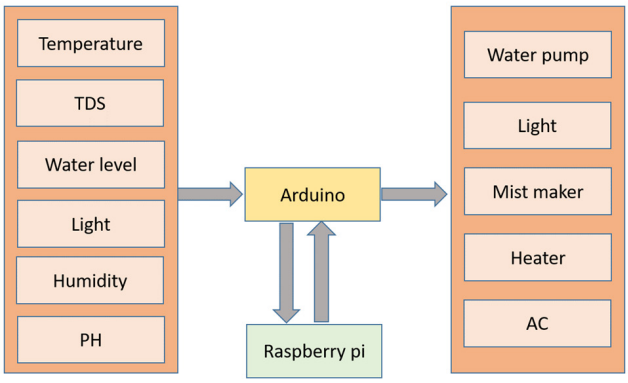

Figure 10: The hardware components of the proposed system.

Figure 11: The environment of the created greenhouse.

There are 5 devices installed in this greenhouse to change the conditions. The water pump, light, mist maker, heater, and AC are the devices that receive data from the Arduino and change the conditions of the greenhouse for better plant growth. Figure 11 shows the overall view of the greenhouse environment created and how these devices are connected in the distributed systems research lab.

Result

A. After setting up the greenhouse and verifying all connections and equipment, we initiated planting and cultivation.

B. Strawberry seedlings were grown with remote control of environmental conditions from the server side, minimizing on-site labor interventions. Under these conditions, the first harvestable fruits were obtained at approximately three months, which is comparable to our baseline manual practice (2.5-3.5 months for similar cultivars, 120 plants per batch). Fruit size and visual quality were consistent with the baseline range, and environmental setpoints were maintained within target tolerances (temperature ±0.5°C, RH ±3%, EC ±0.1 mS/cm).

Architectural Analysis

Architectural assessment methods are used to investigate or measure the quality characteristics and quality assurance of software products. Software architecture assessment is an effective mechanism for improving the quality of software systems [33]. Various techniques are used to evaluate software architecture. One of them is a scenario-based technique. Since it is related to a specific system and is part of the system development process, it basically covers all aspects of the system and performs a complete evaluation of the system. ATAM (Architecture Trade-off Analysis Method) is a method used to evaluate the quality attributes of a software architecture. It is used to reduce risks in software architecture in the early stages of the software development life cycle [34]. The ATAM consists of four phases:

Phase 0: Preparation, planning, and stakeholder recruitment

Preparation, planning, stakeholder recruitment, and team formation are done in this phase. In addition, the components of the integrated greenhouse control system are introduced and the tasks of each team member are described in this phase. Table 3 shows the members of the evaluation team.

Table 3: Members of the evaluation team.

No.	Expertise	Role
1	Software Engineer	Developer, team leader
2	Software Engineer	Developer, evaluation leader
3	Senior Architect	Developer, process observer
4	Agricultural Engineer	User, questioner
5	Farmer	User, questioner

Phase 1: Evaluation process

This phase includes steps 1 to 6.

Step 1: Present ATM: This step presents the concept of the process to all stakeholders of the process and answers the questions of the participants. This step familiarizes the participants with the process.

Step 2: Present business drivers: In this step, the business drivers of the system are presented. For this purpose, in this step, a view of the integrated greenhouse control system based on the Internet of Things is examined. This view includes the following:

A. The main important requirements of the system:

I. To identify the system requirements, a use case diagram is shown in Figure 1.

B. The objectives of the integrated greenhouse control system:

I. Provide an open platform for connecting IoT nodes in the greenhouse environment.

II. Intelligently control greenhouses and IoT nodes.

III. Enable access to multiple geographically distributed greenhouses from one point.

IV. Reduce the production cost of agricultural products and improve crop quality.

C. The following quality attributes are ranked as high priority:

I. Availability: The integrated control system for greenhouses should be available at all times.

II. Interoperability: This system consists of specific and separate components, each performing specific tasks in the system. These components and the relationship between them are designed to harmonise with each other.

III. Performance: When new data arrives in this system during peak load and information processing, the system performs the processing as quickly as possible and applies the necessary changes and commands.

Step 3: Present architecture: In this step, a brief overview of the architecture is presented by the architect with an appropriate level of detail. In addition, the main components and tasks of the system, as well as the relationships between them, were explained to provide knowledge to the team members. A description of the architecture can be found in Section 3.4.

Step 4: Identify architectural approaches: At this stage, the architect presents specific architectural approaches to the team, and then the proposed architecture is discussed. The analysis performed at this stage is used as a basis for the activities of the next stages.

a) The system design is based on the component that satisfies the reusability characteristic.

b) Divide the system into subsystems that achieve high variability.

c) Reduce the complexity of the system design with a component-based design that increases comprehensibility.

d) Increases data security on components and communication between components to achieve data integrity and performance assurance.

Step 5: Generate quality attribute utility tree: In this step, technical requirements of the system are defined and the quality attributes of the system are generated. Quality attributes such as availability, interoperability, and performance are reviewed. Table 4 shows the quality attributes of the system, and the associated scenarios [35] (Table 4).

Table 4: The quality attributes and their related scenarios.

Quality Attributes	Scenario
Availability	The system must be available under all circumstances. Therefore, if a problem occurs in the system, this problem will be reported to the server administrator through a notification, and the server administrator will find and fix the problem.
Interoperability	The system components should be coordinated and compatible with each other.
Performance	When new data arrives during the peak time and data processing, the proposed system will perform the processing as soon as possible and apply the necessary changes and commands.

Step 6: Analyze architectural approaches: In this step, each scenario is prioritized based on importance and application, and then the scenarios that have the highest priority are mapped on the architecture.

A. Availability: The integrated greenhouse control system must be available at all times. Therefore, when the system encounters problems, a message is sent to the system administrator to check the server and fix the problem. If the error is caused by the designed website, users can use smartphones to connect to the server until the problem is fixed. In Tables 5&6, two scenarios of availability are examined according to their priority.

Table 5: Analysis of the availability 1 quality attribute.

Quality Attribute	Scenario
Availability 1	Source of stimulus: Within the system Stimulus: An internal error may occur, such as a dramatic sudden difference in the values of the sensors - or no change in the new information coming from the sensors after the relay state is changed, etc. Artifact: Server Environment: After receiving information from the database Response: Send a message to the system administrator Response measurement: In case of failure, there is no downtime.

Table 6: Analysis of the availability 2 quality attribute.

Quality Attribute	Scenario
Availability 2	Source of stimulus: An external factor generates the stimulus. Stimulus: The failure to send information from the sensors to the server Artifact: Server Environment: When sensors send information Response: Send a message to the system administrator Response measurement: in case of failure, there is no downtime.

B. Interoperability: This attribute allows different systems and components to work together seamlessly and effectively, improving the overall efficiency and effectiveness of the system. In this system, for coordination and communication between components, the same data format (Json) is considered for receiving and sending data in different parts of the system. In Table 7, the quality characteristic of interoperability is examined.

Table 7: Analysis of the interoperability quality attribute.

Quality Attribute	Scenario
Interoperability	Source of stimulus: Information is entered. Stimulus: Sending information to databases and relays Artifact: Relays Environment: System is detected before execution time. Response: The information is converted to the standard required by the destination components (relays). Response measurement: The information is exchanged properly with high probability

C. Performance: Performance refers to the ability of a system to meet its functional and non-functional requirements efficiently and effectively. In this system, events are responded to in the shortest possible time because of the multi-threaded and parallel instructions. When a new event occurs during peak

hours, the system can respond to it as quickly as possible. In Table 8, the performance quality characteristic is examined.

Table 8: Analysis of the performance quality attribute.

Quality Attribute	Scenario
Performance	Source of stimulus: From within the system, an event is reached by the sensors. Stimulus: All information is sent to the server for processing. Artifact: Server Environment: At peak time on the server Response: The server performs the processing as fast as possible. Response measurement: The information is processed as fast as possible.

Phase 2: Evaluation process 2

In this phase, steps 7 to 9 from the ATAM analysis phase are performed.

Step 7: Brainstorm scenarios: In this step, the evaluation team, which includes the stakeholders, architects, and other technical experts, brainstorm a set of scenarios that represent the usage of the system or product under evaluation and the meeting participants vote for them. Table 9 shows the high priority scenarios and their associated quality attributes.

Table 9: High priority scenarios and related quality attributes.

No.	Scenario	Quality Attribute
1	The system is resistant to unauthorized intrusion.	Security
2	All commands are processed as fast as possible.	Performance
3	All problems and bugs must be resolved quickly.	Availability
4	System errors in processing must be supported.	Reliability
5	New requests in the system are created in the shortest possible time without side effects.	Modifiability
6	All components of the system work in harmony with each other.	Function
7	The system components should be coordinated and compatible with each other.	Interoperability
8	In complex cases, it is possible to extend the framework of the system architecture.	Variability
9	Enable a secure data structure.	Security
10	The system can be easily migrated from one hardware/software environment to another.	Portability

In this step, members vote on the scenarios. The number of votes is calculated in the form of Phrase 2:

Phrase 2 calculates the number of votes [35]. Each member can cast three votes on the scenarios. The members vote for the scenarios according to their expertise. Table 10 shows the voting results of the members. In Table 11, the scenarios are arranged in descending order according to the number of votes.

Table 10: The voting results of the members.

Scenario Number	Quality Attribute	User Vote	System Developer Vote	System Architecture Vote	Total Vote
2	Performance	1	1	1	3
3	Availability	1	1	1	3
4	Reliability	1	0	1	2
5	Modifiability	1	1	0	2
6	Function	0	0	1	1
7	Intero-perability	1	1	1	3
8	Variability	0	1	0	1

Table 11: Sorted scenarios based on the number of votes.

Scenario Number	Quality Attribute	Scenario
3	Availability	All problems and bugs must be solved quickly.
7	Interoperability	System components should be coordinated and compatible with each other.
2	Performance	All commands are processed as fast as possible.
5	Modifiability	New requests in the system are created in the shortest possible time without side effects.
4	Reliability	User errors during processing must be supported.
6	Function	All components of the system work in harmony with each other.
8	Variability	In complex cases, it is possible to extend the framework of the system architecture.

Step 8: Analyze architectural approaches 2: This step is similar to step 6 and analyzes the scenarios created by stakeholders in the previous step. This step may lead to the discovery of scenarios not yet analyzed in Step 6.

a. Modifiability: Modifiability tests whether the system can be easily modified in a short period of time. This architecture has a high degree of modifiability because all components are separate. If a specific component needs to be added to this architecture, the adaptability can be done at the lowest cost. Table 12 examines the modularity quality attribute.

Table 12: Analysis of the modifiability quality attribute.

Quality Attribute	Scenario
Modifiability	Source of stimulus: Developer Stimulus: Quality attribute Artifact: System user interface or system components Environment: Design time Response: Modifies without affecting other functions. Response measurement: Extent to which this affects other functions or quality attributes.

b. Reliability: Reliability examines the behavior of the system in response to errors in the system. The proposed system detects all invalid inputs and data and denies the input. If there is a significant difference between the data coming from the sensors in a certain period of time, the system notices this difference and sends an error message to the server and notifies the server administrator about these errors. In Table 13, the reliability quality attribute is examined.

Table 13: Analysis of the reliability quality attribute.

Quality Attribute	Scenario
Reliability	Source of stimulus: Through the sensors Stimulus: In a period of time, different data come from a sensor. Artifact: Server Environment: At the processing time Response: Send a message to the system administrator Response measurement: In case of failure, there is no downtime.

Step 9: Present results: At the end of the second phase, the evaluation team reviews all the results and the results of the analysis performed in the previous phase are presented to the stakeholders. These results include:

a) The architectural approaches

b) The set of scenarios and their prioritization

c) The utility tree

d) The risks

e) The sensitivity points and tradeoff points

f) The project risks:

I. The processes for troubleshooting are not fully defined. Multiple fault detection scenarios have been explored, although customers may encounter a fault that has not been explored.

II. There are issues with documentation. There is documentation such as UML diagrams, but no explanation of system development, testing, and maintainability.

Phase 3: Report

In this phase, the tangible result is a written report. The outcome of the meeting held in this phase facilitates the rapid preparation of the final written report. The preparation of the final document is a simple task that brings together the results of the previous steps.

Conclusion

In this paper, the software layer of the IoT-based integrated greenhouse control system is proposed. A scheduler is designed that can execute the system commands in parallel. Also, the SCADA system is mentioned to upgrade this system with SCADA or integrate it with SCADA in the future. The proposed software architecture is presented with 4 + 1 views. The Architecture Tradeoff Analysis Method (ATAM) is applied to evaluate different scenarios. The trade-off analysis has proven the suitability and competence of the proposed architecture. Since the proposed system is only designed to control and monitor one greenhouse, a system that can control and monitor multiple greenhouses simultaneously can be developed for the future of this research. Cloud technology can also be used to store and analyze data in these systems with multiple greenhouses.

References

1. Li Z, Wang J, Higgs R, Zhou L, Yuan W (2017) Design of an intelligent management system for agricultural greenhouses based on the internet of things. 22017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), Guangzhou, China, pp: 154-160.
2. Fang T, Yang Y (2022) Distributed communication protocol in wireless sensor network based on internet of things technology. Wireless Personal Communications 126(3): 2361-2377.
3. Sofwan A, Sumardi S, Ahmada AI, Ibrahim I, Budiraharjo K, et al. (2020) Smart Greet things: Smart greenhouse based on internet of things for environmental engineering. 2020 International Conference on Smart Technology and Applications (ICoSTA), Surabaya, Indonesia, pp: 1-5.
4. Farooq MS, Riaz S, Abid A, Abid K, Azhar Naeem M (2019) A survey on the role of IoT in agriculture for the implementation of smart farming. IEEE Access 7: 156237-156271.
5. Tolentino LKS, Celline PDP, Jatt DI, Benjamin ENJ, Luigi James DS, et al. (2021) Development of an IoT-based intensive aquaculture monitoring system with automatic water correction. International Journal of Computing and Digital Systems 10(1): 1355-1365.
6. Nath SD, Shahadat Hossain M, Hasan M, Chakma R, Akber Chowdhury I, et al. (2021) Design and implementation of an IoT based greenhouse monitoring and controlling system. Journal of Computer Science and Technology Studies (JCSTS) 3(1): 1-6.
7. Abbasi M, Rafiee M, Khosravi MR (2020) Investigating the efficiency of multithreading application programming interfaces for parallel packet classification in wireless sensor networks. Turkish Journal of Electrical Engineering & Computer Sciences 28(3): 1699-1715.
8. Kitpo N, Kugai Y, Inoue M, Yokemura T, Satomura S (2019) Internet of things for greenhouse monitoring system using deep learning and bot notification services. 2019 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, Nevada, USA, pp: 1-4.
9. Geng X, Zhang Q, Wei Q, Zhang T, Cai Y, et al. (2019) A mobile greenhouse environment monitoring system based on the internet of things. IEEE Access 7: 135832-135844.
10. Burchi G, Chessa S, Gambineri F, Kocian A, Massa D, et al. (2018) Information technology-controlled greenhouse: A system architecture. 2018 IoT Vertical and Topical Summit on Agriculture-Tuscany (IOT Tuscany), IEEE, Tuscany, Italy.
11. Bersani C, Ruggiero C, Sacile R, Soussi A, Zero E (2022) Internet of things approaches for monitoring and control of smart greenhouses in industry 4.0. Energies 15(10): 3834.
12. Drewnowski J (2019) Advanced supervisory control system implemented at full-scale WWTP-A case study of optimization and energy balance improvement. Water 11(6): 1218.
13. Dedeepya P, Srinija USA, Gowtham Krishna M, Sindhusha G, Gnanesh T (2018) Smart greenhouse farming based on IOT. 2018 2nd International Conference on Electronics, Communication and Aerospace Technology (ICECA), IEEE, Coimbatore, India.
14. Subahi AF, Bouazza KE (2020) An intelligent IoT-based system design for controlling and monitoring greenhouse temperature. IEEE Access 8: 125488-125500.
15. Arshad J, Tariq R, Saleem S, Rehman AU, Munir HM, et al. (2020) Intelligent greenhouse monitoring and control scheme: An arrangement of Sensors, Raspberry Pi based Embedded System and IoT platform. Indian Journal of Science and Technology 13(27): 2811-2822.

16. Suma N (2020) IOT based smart agriculture monitoring system. Regular.
17. Danita M, Mathew B, Shereen N, Sharon N, John Paul J (2018) IoT based automated greenhouse monitoring system. 2018 2nd International Conference on Intelligent Computing and Control Systems (ICICCS), IEEE, Madurai, India.
18. Patil MA, Adamuthe AC, Umbarkar AJ (2020) Smartphone and IoT based system for integrated farm monitoring. Techno-Societal 2018, Springer International Publishing, Switzerland, pp: 471-478.
19. Soheli SJ, Jahan N, Hossain B, Adhikary A, Rahman Khan A, et al. (2022) Smart greenhouse monitoring system using internet of things and artificial intelligence. Wireless Personal Communications 124(4): 3603-3634.
20. Nadiri F, Banirostam T, Rad AB (2025) On a novel localization methodology for humanoid soccer robots via sensor fusion and perspective transformation. International Journal of Intelligent Robotics and Applications.
21. Slimani H, El Mhamdi J, Jilbab A (2025) Real-time greenhouse management using IoT, digital twin, and augmented reality for optimal control and decision-making. Acta IMEKO 14(2): 1-15.
22. Berezovskyi A, Inam R, El-khoury J, Mokrushin L, Fersman E (2024) Integrating systems of systems with a federation of rule engines. Journal of Industrial Information Integration 38: 100545.
23. Wang H, Wang X (2025) Exploration of soft motion control platform for real-time Linux system and ethernet fieldbus. Lecture Notes in Education, Arts, Management and Social Science 3(1): 35-40.
24. Raghunandan K (2022) Supervisory Control and Data Acquisition (SCADA). Introduction to Wireless Communications and Networks, Springer International Publishing, Switzerland, pp: 321-337.
25. Nurjannah DR, Supriadi D, Sutiawan A, Kustiawan I (2020) Designing smart greenhouse systems using SCADA based on IoT. IOP Conference Series: Materials Science and Engineering 850: 1-6.
26. Ara A (2022) Security in Supervisory Control and Data Acquisition (SCADA) based Industrial Control Systems: Challenges and solutions. IOP Conference Series: Earth and Environmental Science 1026(1): 012030.
27. Kruchten PB (1995) The 4+1 view model of architecture. IEEE Software 12(6): 42-50.
28. Maier MW, Emery D, Hilliard R (2001) Software architecture: Introducing IEEE standard 1471. Computer 34(4): 107-109.
29. Aquino ER, Saqui-Sannes PD, Vingerhoeds RA (2021) A methodological assistant for UML and SysML use case diagrams. Model-Driven Engineering and Software Development, Springer International Publishing, Switzerland, pp: 298-322.
30. Bernardi S, Gómez A, Merseguer J, Perez-Palacin D, Requeno JI (2022) DICE simulation: A tool for software performance assessment at the design stage. Automated Software Engineering 29(1): 36.
31. Tatale S, Chandra Prakash V (2022) Combinatorial test case generation from sequence diagram using optimization algorithms. International Journal of System Assurance Engineering and Management 13(1): 642-657.
32. Jaworsky M, Ziadé T (2021) Expert python programming: Master python by learning the best coding practices and advanced programming concepts. (4th edn), Packt Publishing, Birmingham, England, UK, p. 630.
33. Babar MA, Kitchenham B, Zhu L, Gorton I, Jeffery R (2006) An empirical study of groupware support for distributed software architecture evaluation process. Journal of Systems and Software 79(7): 912-925.
34. Kazman R, Barbacci M, Klein M, Jeromy Carriere S, Woods SG (1999) Experience with performing architecture tradeoff analysis. Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002), IEEE, Los Angeles, California, USA.
35. Bass L, Clements P, Kazman R (2012) Software architecture in practice. (3rd edn), Pearson Education, London, UK.

