

# Design and implementation of a smart greenhouse system using Lora Wireless Technology

Nguyen Thi Huyen Trang<sup>1</sup>, Phan Thi Quynh Huong<sup>2</sup>, Tran Thi Tra Vinh<sup>3</sup>

<sup>1,2,3</sup> Vietnam Korea University of Information and Communications Technology – Danang University, Viet Nam

---

**Abstract:** Greenhouse farming has become an increasingly popular method of crop production, offering greater control over environmental factors and the potential for higher yields. However, the management of these complex systems can be challenging, requiring real-time monitoring and precise adjustments to ensure optimal growing conditions. To address this, the design and implementation of a smart greenhouse system leverage cutting-edge technologies such as MongoDB for data storage, OutSystems for application development, and LoRa for wireless communication. The implementation results in improved resource efficiency, increased crop yields, and reduced labor costs. By combining the strengths of MongoDB, OutSystems, and LoRa, the smart greenhouse system demonstrates a robust, scalable, and cost-effective solution for modern agriculture.

**Keywords:** Smart IoT, MongoDB, Outsystems, Lora, sensor.

---

Date of Submission: 13-06-2024

Date Of Acceptance: 27-06-2024

---

## I. INTRODUCTION

The agricultural sector has seen a significant transformation with the advent of Internet of Things (IoT) and cloud computing technologies. This paper presents the design and implementation of a smart greenhouse system that leverages the power of MongoDB, OutSystems, and LoRa to enhance the efficiency and productivity of greenhouse operations. The system employs a variety of sensors to monitor and control critical parameters such as temperature, humidity, soil moisture, and nutrient levels, enabling precise control and optimization of the growing environment. The data collected from these sensors is stored in a NoSQL MongoDB database and accessed through a user-friendly web-based interface for data visualization, alerting, and remote control of the greenhouse systems, developed using the OutSystems low-code platform. Furthermore, the system utilizes LoRa connectivity to enable reliable and long-range communication between the greenhouse nodes and the central control system, overcoming the limitations of traditional wireless technologies in greenhouse settings.

The contents of paper are arranged in sequence: Part 1 will be the design a smart greenhouse with its system requirements, part 2 is steps to implement the system and result, part 3 is the conclusion and give some evaluations on the built system.

## II. SYSTEM DESIGN

### 1.1 Necessary requirements

The system needs to have flexible interaction between hardware and software components to ensure flexibility in greenhouse operation and management.

Ensure the system is designed and implemented according to safety standards, especially in the use of electrical components and equipment.

The system needs to be able to integrate data from sensors combined with devices to optimize device management and control.

Optimize energy used to reduce environmental impact and ensure stable energy sources.

System design for easy maintenance and repair, reducing operating interruption time.

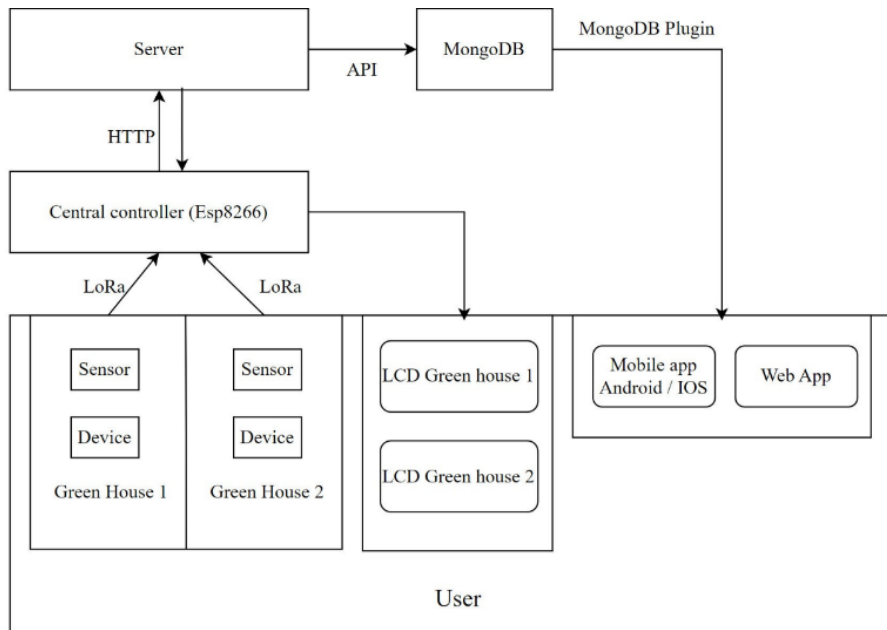
Ensure strong connectivity with new technologies such as Outsystem and NodeJS to deploy and store database applications through MongoDB.

The system needs remote connection and monitoring to manage and monitor it anywhere.

Design applications and integration processes to ensure ease of use and integration with other systems.

## 1.2 System Structure

**Figure1. System structure**



Two operating modes of the system: automation and manual mode

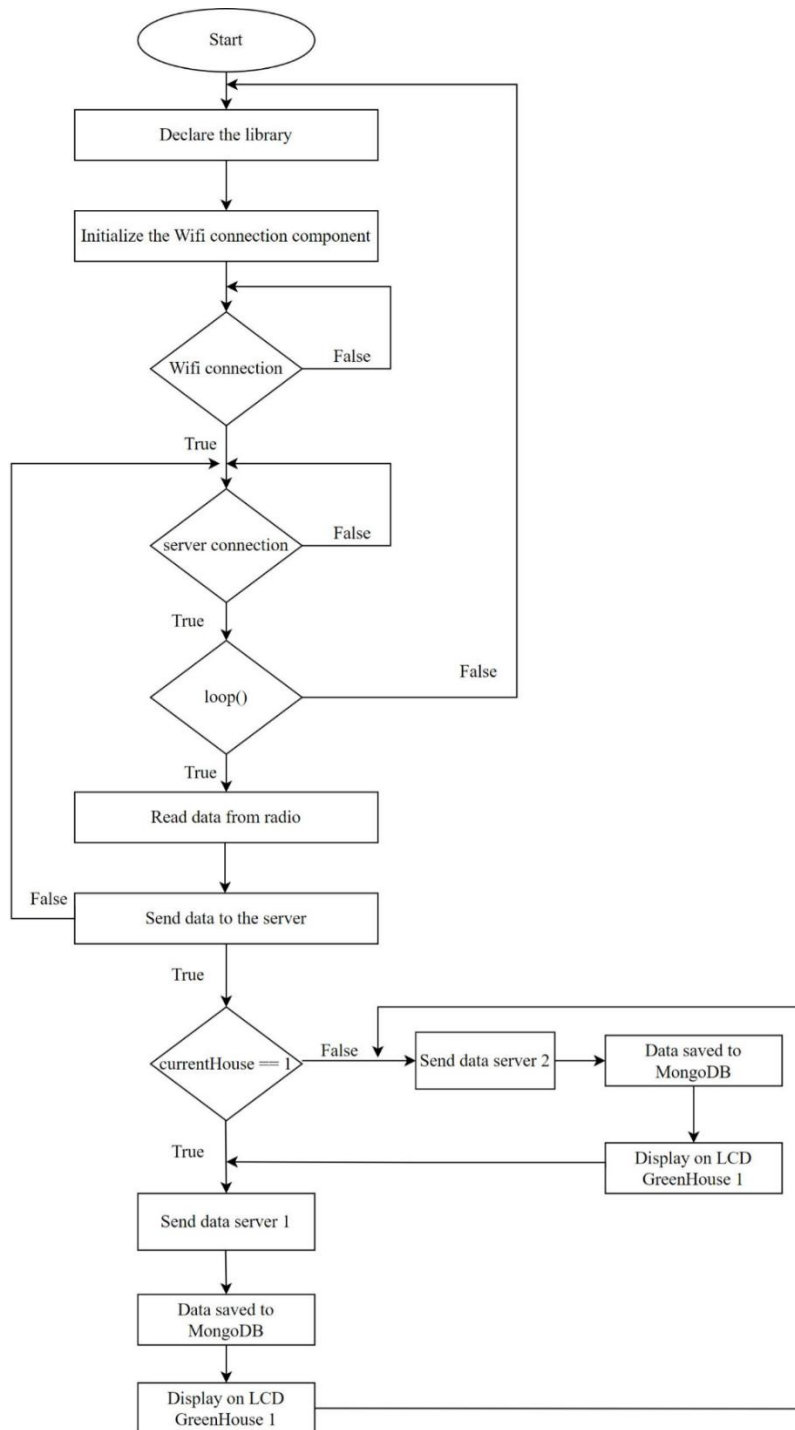
The model applies design and implementation using advanced technology and is processed automatically to produce the most optimal and effective results. In addition, when unforeseen problems arise, they can be used manually to monitor and manage them immediately.

In automation mode: Sensors trigger actions, and data is sent via LoRa.

In manual mode: Central device receives and sends sensor values to the server. Server saves data to MongoDB, displayed on an LCD. Mobile/web apps, through Outsystem, connect to MongoDB for real-time monitoring.

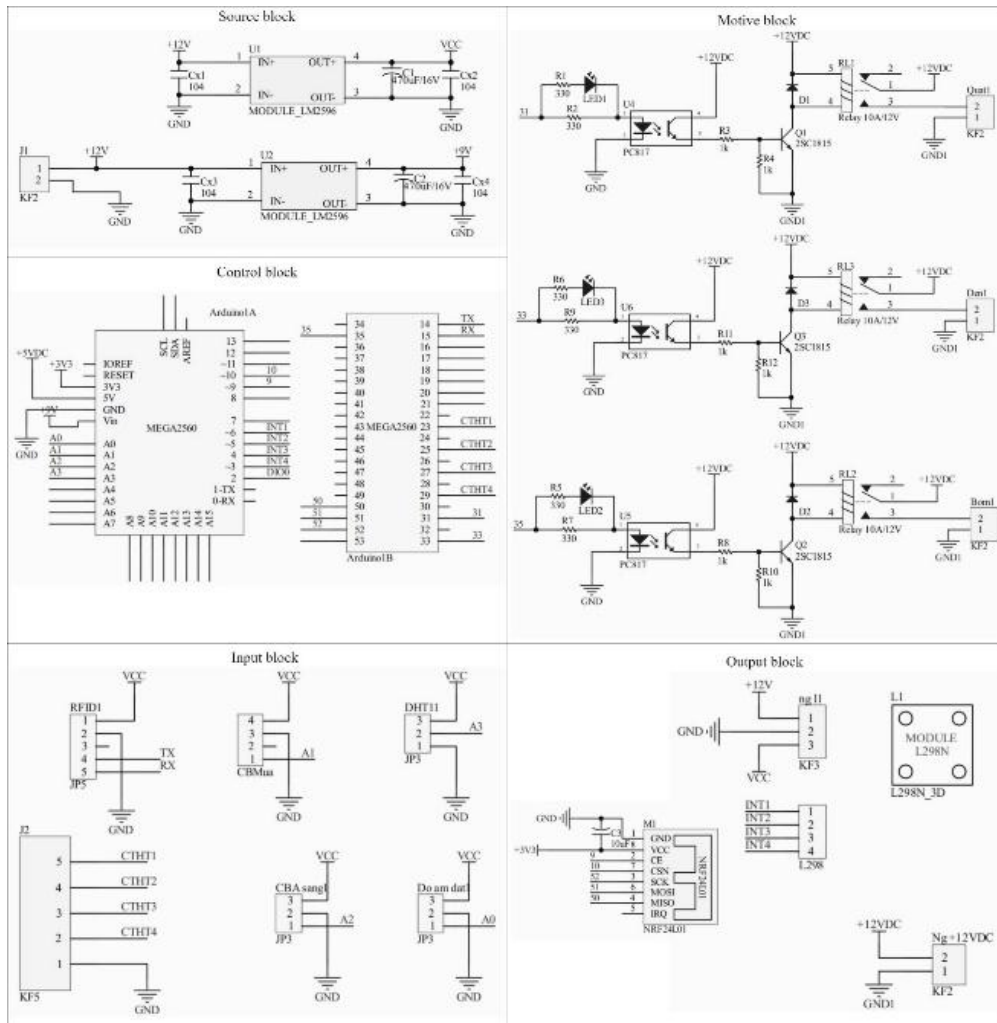
1.3 Algorithm Flowchart

Figure2. Diagram of central control algorithm



1.4 Circuit Design

Figure3. Greenhouse principle circuit



**Source block**

Configure the power parameters for the circuit, defining important parameters such as voltage, current, and other properties of the power source. In this case, the power block is used to integrate the LM2596 module, which reduces voltage and stabilizes the current in the circuit.

**Input block**

Responsible for collecting data from the environment and converting it into control signals. The input block includes input data such as RFID, rain sensor, DHT11, light sensor, soil moisture sensor, and limit switch for the door system.

**Control block**

Performs important functions in controlling and managing an automation system. Includes wireless communication, sensor management. Use control logic to perform operations such as turning on/off LEDs, lights, fans, doors, roofs, and send control commands to the motive block to initiate control actions.

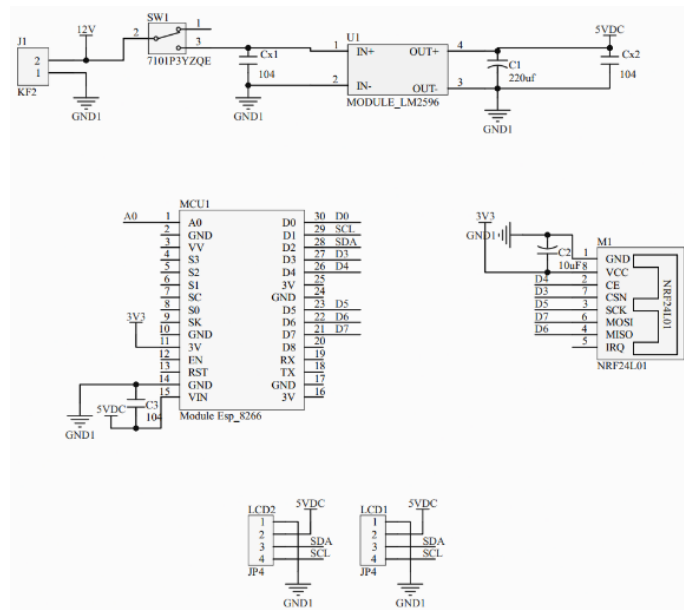
**Motive block**

In the output block, receive data from the greenhouse control block and perform the corresponding control logic actions. Use relay to control fans, lights and pumps. If the value is 1 then the device will be off and if the value is 2 then the device will be on.

**Output block**

In the output block, receive data from the greenhouse control block and perform the corresponding control actions. Use the LN2983 module to control devices with large currents, such as doors and curtains. Send all sensor data and device status via LoRa Module SX1278 to the central device.

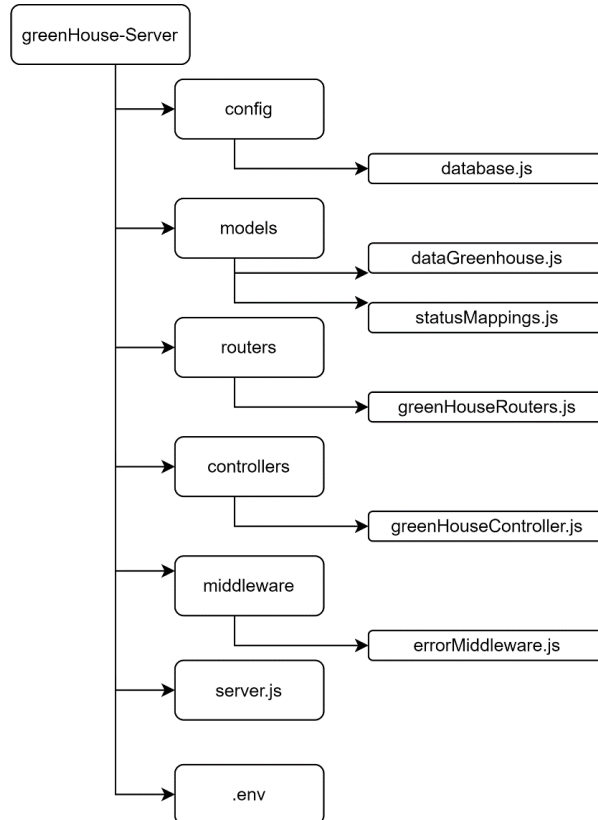
Figure4. Greenhouse central monitoring principle circuit



Use module nRF24L01 to receive data from the greenhouse in the greenhouse system. Then use ESP8266 to process the data and send data to the Server, updating information on the LCD screen corresponding to each greenhouse. In addition, use the LM2595 module to control the central on/off.

### 1.5 Design of Server Structure

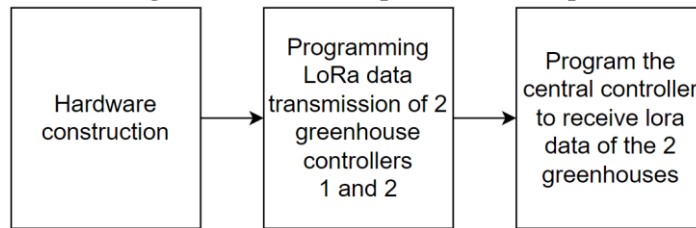
Figure5. Server structure diagram



### III. SYSTEM IMPLEMENTATION

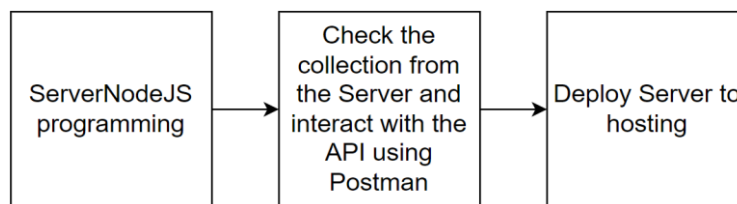
#### 3.1. Hardware Implementation

Figure6. Hardware implementation steps



#### 3.2 Build Server

Figure7. Steps to build a server



##### 3.2.1 Set up MongoDB database

- Create a MongoDB account and create a project to store the database.
- Create a database and two collections, greenhouse 1 and greenhouse 2, to store the data.

##### 3.2.2 Programming for NodeJS Server

- Initialize the server with Mongoose for MongoDB connection via URI.
- Define MongoDB schemas for greenhouse data, ensuring clear mappings for component status. Process and store input data in the respective MongoDB databases for each greenhouse.

Figure8. Mapping values corresponding to states

```
const rainMap = createStatusMap(1, 2, 'No Rain', 'Rain');
const lightSensorMap = createStatusMap(1, 2, "It's Dark", 'Bright Sky');
const ledMap = createStatusMap(1, 2, 'Light Off', 'Light On');
const fanMap = createStatusMap(1, 2, 'Fan Turns Off', 'Fan Turns On');
const pumpMap = createStatusMap(1, 2, 'Pump Turns Off', 'Pump Turns On');
const doorMap = createStatusMap(1, 2, 'Close Door', 'Open Door');
const roofMap = createStatusMap(1, 2, 'Close Roof', 'Open Roof');
```

- Process and store input data in respective MongoDB databases.
- Set up Express router for greenhouse data routes.
- Implement robust error handling.

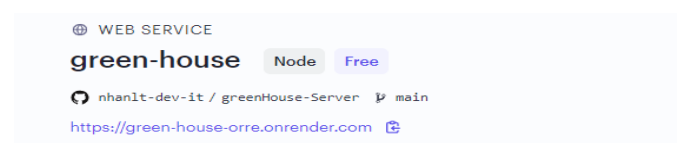
##### 3.2.3 Test the Server

Use virtual data created from Postman to GET to check if the value is sent and saved to MongoDB database, if yes the server is working normally, if not then the server is experiencing an error.

##### 3.2.4 Deploy Server to hosting

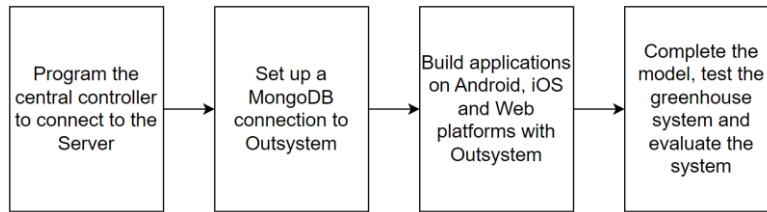
After testing the Server to operate normally, the Server will be deployed to hosting so it can run online 24/7 without needing to use it local.

Figure9. Server implementation results



### 3.3 Implement hardware and software connections

**Figure10. Steps to implementation hardware and software connection**



#### 3.3.1 Establish ESP8266 connection with server

Declare the variables used to connect to the server and specify the Server address and data transfer port.

**Figure10. Declare the connection variable to the server**

```
const char* serverAddress = "green-house-orre.onrender.com";
const int serverPort = 443;
```

Send all data from greenhouse 1 and greenhouse 2 to the Server via the https protocol in JSON format. The data contains input parameters corresponding to the sensor and device status.

#### 3.3.2 Results of connecting ESP8266 to MongoDB

**Figure11. Greenhouse data at MongoDB**

|                 |   |    |    |     |   |     |     |
|-----------------|---|----|----|-----|---|-----|-----|
| datagreenhouse1 | 0 | 0B | 0B | 4KB | 1 | 4KB | 4KB |
| datagreenhouse2 | 0 | 0B | 0B | 4KB | 1 | 4KB | 4KB |

### 3.4 Set up Outsystem connecting to MongoDB database



#### 3.4.1 Set up Outsystem system

Set up and select the MongoDB database for the Outsystem system, initialize personal information such as name, password, server name,... to connect. After successful connection, data will be obtained from 2 collections of 2 greenhouses. Proceed to push that data to the Outsystem system to deploy the application.

#### 3.4.2 Results of building Outsystem with MongoDB

Data has been successfully integrated from MongoDB into the OutSystem.

**Figure12. Integrate MongoDB data into Outsystem**

|                     |  |
|---------------------|--|
| App name            |  MongoDB Connector Outsystem                      |
| Module to consume   | MongoDB_IS <br>MongoDB integration service module |
| Other modules       | MongoDB_DRV<br>MongoDB driver module   |
| Assigned connection | MongoDB Connection Outsystem - GreenHouse<br>ltnhan19ce  |

### 3.5 Multi-platform Outsystem application programming

#### 3.5.1 Programming mobile and web applications

Create a UserExtend database to store user account information and build application logic to handle the flow of application activities. Use MongoDB\_IS to get data to design and build information pages about greenhouse system management.

### 3.5.2 Results of application implementation

Install apps:

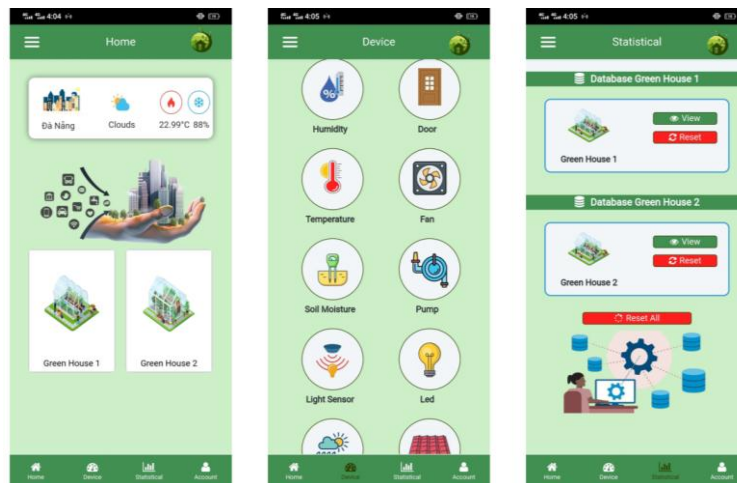
For Android applications, the application can be installed through installing the APK file by scanning the Native Platform code. In addition, Android and iOS can be quickly accessed by scanning the Web App code to conveniently monitor and manage the greenhouse system.

Figure13. QR code Native Platform and Web App



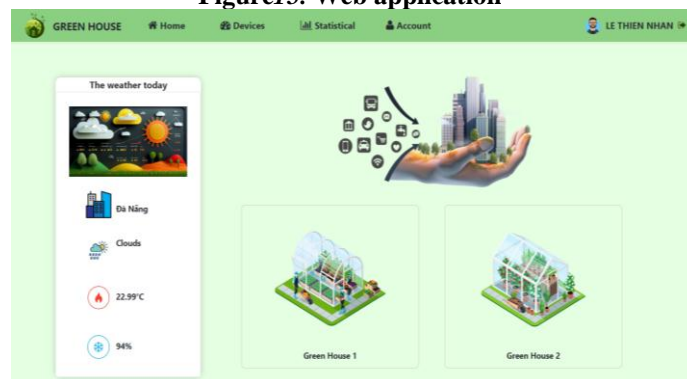
- Mobile application:

Figure14. Greenhouse App mobile



- Web application:

Figure15. Web application





### 3.6 Greenhouse system results

**Figure16. Smart greenhouse system**



### 3.7 Testing system

- Sensors display accurate values on the LCD screen.
- Devices are optimized based on sensor data.
- Greenhouses send correct data to the server.
- Android and iOS apps monitor sensors and devices effectively.

### 3.8 Evaluating greenhouse

- Design complies with set requirements and safety standards for electrical components.
  - Maintenance facilitated through clear design blocks.
  - Addresses equipment automation needs based on design processes.
  - Data problems resolved by utilizing MongoDB for storage.
  - Server built but requires improvement for real-time performance.
  - Outsystem app employed for cross-platform development.
- Cross-platform applications enable easy monitoring and management of sensors on any device.

## IV. CONCLUSION

The project successfully designed and implemented a greenhouse controller ensuring interoperability between sensors and devices. It features a beautiful, detailed greenhouse system model and a self-built NodeJS server deployed on render.com, with a reliable Server-MongoDB connection for data storage. OutSystems was configured for cross-platform development, supporting Android, iOS, and Web applications. These applications effectively monitor sensor data and device status, providing real-time updates, statistical displays, and database reset features. However, real-time requests for unloaded data require a 30-second wait to prevent server crashes, and the system currently cannot add control features to the application due to conflicts with the sensor system. Future development directions include using more stable hosting to enable real-time data access, adding manual device control features that avoid sensor conflicts, and developing new functions to check sensor status.

## REFERENCES

- [1]. Gonzalez, V., & Gonzalez, F., 2020. Design and Implementation of a Smart Greenhouse System Using IoT and Cloud Computing. *International Journal of Computer Applications*, 176(1), 1-7.
- [2]. Silva, L., & Pereira, R., 2018. Developing Cross-Platform Mobile Applications Using OutSystems.. *Software Engineering Journal*, 33(3), 211-220.
- [3]. Zhang, X., Li, Y., & Wang, Z., 2021. Application of MongoDB in Real-time Data Management for Smart Agriculture. *Journal of Database Management*, 32(4), 45-57.
- [4]. Silva, L., & Pereira, R., 2018. Developing Cross-Platform Mobile Applications Using OutSystems, *Software Engineering Journal*, 33(3), 211-220.
- [5]. Smith, J., & Brown, K., 2020. LoRa Technology in Smart Farming: A Case Study on Greenhouse Monitoring. *Wireless Sensor Networks Journal*, 15(2), 112-119.

- [6]. Patel, H., & Desai, R., 2019. Integration of IoT, Big Data, and Cloud Computing for Smart Agriculture. *International Journal of Advanced Research in Computer Science*, 10(5), 66-74.
- [7]. Brown, T., & Miller, J., 2021. Real-time Data Processing and Analysis in Smart Greenhouses. *Agricultural Systems Journal*, 95(4), 325-335.
- [8]. Williams, S., & Jones, P., 2019. Challenges and Solutions in Developing Smart Greenhouse Systems. *Journal of Agricultural Engineering Research*, 84(3), 178-189.