🌱 Smart Greenhouse Control System

A comprehensive IoT solution for automated greenhouse monitoring and control, featuring real-time sensor data, actuator control, and a modern web interface.

license MIT ESP32 Compatible Next.js 15.3

📋 Table of Contents

🎯 Overview

This Smart Greenhouse Control System provides automated environmental monitoring and control for your greenhouse. It combines ESP32 microcontroller hardware with a modern Next.js web application to deliver real-time monitoring of temperature, humidity, pH levels, water levels, and air quality, while allowing remote control of fans, pumps, LED lighting, and a motorized roof vent system.

Key Capabilities

- **Real-time Environmental Monitoring**: Track temperature, humidity, pH, water levels, and air quality
- **Remote Actuator Control**: Control fans, water pumps, LED grow lights, and motorized roof vents
- **Web-based Dashboard**: Modern, responsive interface accessible from any device
- **Automated Roof Control**: Stepper motor-driven roof vent system for temperature regulation
- **Live Camera Feed**: Monitor your greenhouse visually in real-time
- **RESTful API**: Easy integration with other automation systems

---

✨ Features

Sensors

- **DHT22**: Temperature and humidity monitoring
- **pH Sensor**: Soil/water pH measurement (0-14 range)
- **Ultrasonic Sensor**: Water level detection
- **MQ135**: Air quality monitoring (CO2, NH3, NOx detection)

Actuators

- **Exhaust Fan**: Temperature regulation and air circulation
- **Water Pump 1 & 2**: Automated irrigation system

- **LED Grow Lights**: Supplemental lighting control
- **Stepper Motor**: Automated roof vent opening (0-100mm travel)

Web Interface

- Real-time sensor data visualization
- Individual actuator control switches
- Batch control (all pumps on/off, emergency stop)
- Manual roof vent positioning
- Live camera feed integration
- Connection status monitoring
- Debug console for development

🏗️ System Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                    Web Frontend (Next.js)                    │
│  - React Dashboard                                           │
│  - Real-time Data Display                                    │
│  - Actuator Controls                                         │
└──────────────────────┬──────────────────────────────────────┘
                       │
                       │ HTTP/REST API
                       │
┌──────────────────────▼──────────────────────────────────────┐
│                   ESP32 Microcontroller                      │
│  - Web Server (Port 80)                                      │
│  - Sensor Reading Logic                                      │
│  - Actuator Control Logic                                    │
│  - WiFi Communication                                        │
└───┬──────────────────┬──────────────────┬───────────────────┘
    │                  │                  │
    │                  │                  │
┌───▼───┐         ┌────▼────┐       ┌─────▼─────┐
│Sensors│         │ Relays  │       │  Stepper  │
│       │         │         │       │   Motor   │
│- DHT22│         │- Fan    │       │  Driver   │
│- pH   │         │- Pump 1 │       │           │
│- Ultra│         │- Pump 2 │       │(Roof Vent)│
│- MQ135│         │- LED    │       │           │
└───────┘         └─────────┘       └───────────┘
```

🔧 Hardware Requirements

Core Components

Component	Specification	Quantity	Purpose
ESP32 Development Board	WiFi enabled	1	Main controller
DHT22	Temperature/Humidity	1	Climate monitoring

pH Sensor	Analog output	1	Soil pH measurement
HC-SR04	Ultrasonic sensor	1	Water level detection
MQ135	Air quality sensor	1	Gas detection
4-Channel Relay Module	5V/10A	1	Actuator switching
Stepper Motor	NEMA 17 or similar	1	Roof vent control
Stepper Driver	A4988/DRV8825	1	Motor control
Power Supply	12V 2A	1	Motor power
5V Adapter	2A minimum	1	ESP32 & sensors

Pin Connections

ESP32 GPIO Mapping

Relays (Active LOW - LOW=ON, HIGH=OFF)

- GPIO 5: Fan Relay
- GPIO 4: Pump 1 Relay
- GPIO 14: Pump 2 Relay
- GPIO 12: LED Relay

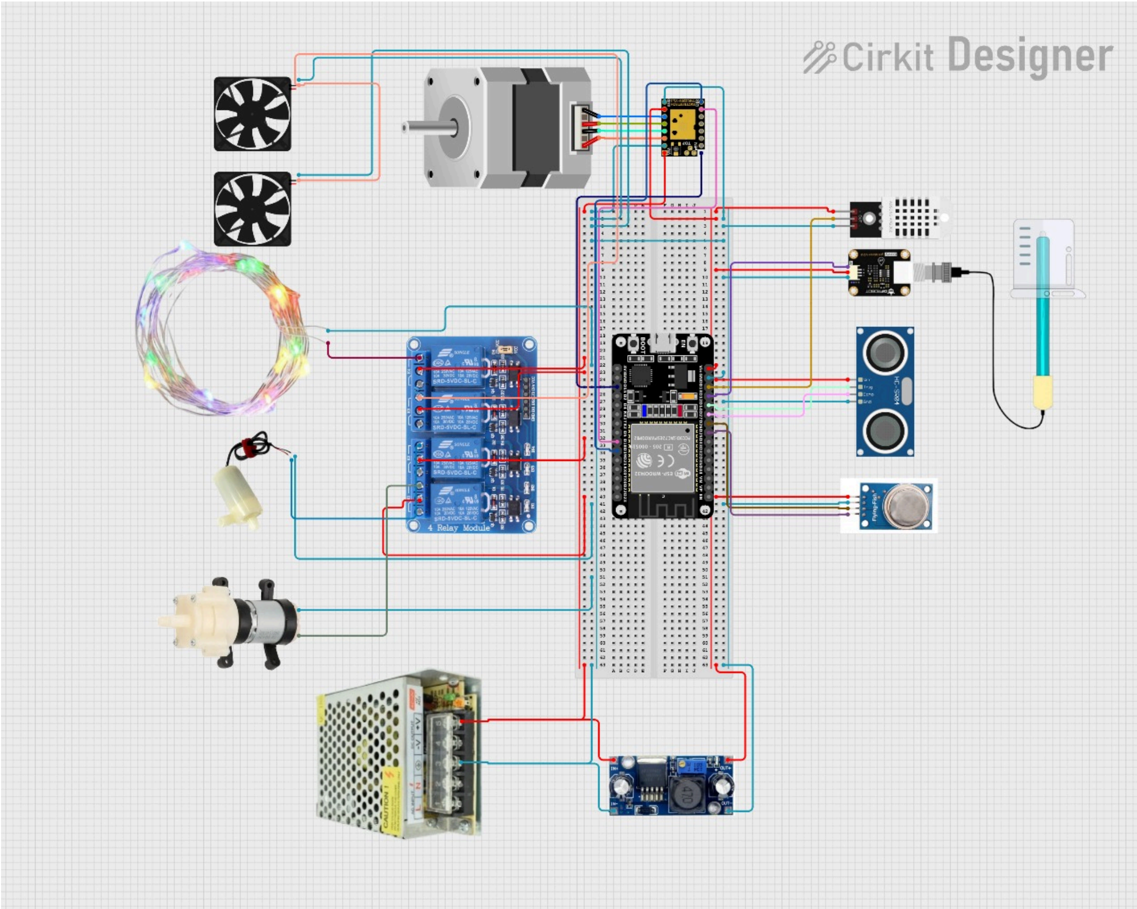
Sensors

- GPIO 2: DHT22 Data Pin
- GPIO 34: pH Sensor Analog Input
- GPIO 0: Ultrasonic Trigger
- GPIO 13: Ultrasonic Echo

Stepper Motor

- GPIO 27: Enable Pin (EN)
- GPIO 26: Direction Pin (DIR)
- GPIO 25: Step Pin (STEP)

Complete Circuit Diagram



Complete wiring schematic showing all connections between ESP32, sensors, relays, and actuators

Wiring Diagram Notes

1. **Relays**: Connect relay module to ESP32 5V, GND, and control pins. Connect AC/DC loads to relay outputs.
2. **DHT22**: Connect to 3.3V, GND, and data pin with 10kΩ pull-up resistor.
3. **pH Sensor**: Connect to 3.3V, GND, and ADC pin (GPIO 34).
4. **Ultrasonic**: Connect VCC to 5V, GND to GND, trigger and echo to respective pins.
5. **Stepper**: Connect driver to 12V power supply, with EN/DIR/STEP pins to ESP32.

💻 Software Stack

ESP32 Firmware

- **Platform**: Arduino IDE / PlatformIO
- **Language**: C++
- **Libraries**:
  - WiFi.h (ESP32 WiFi)
  - WebServer.h (HTTP server)
  - DHT.h (DHT sensor)

Frontend

- **Framework**: Next.js 15.3

- **Language**: TypeScript
- **UI Library**: React 19
- **Styling**: Tailwind CSS
- **Components**: Radix UI, Lucide Icons
- **State Management**: React Hooks

Backend

- **Runtime**: Node.js
- **Framework**: Express.js
- **Database**: LibSQL (SQLite-based)
- **ORM**: Drizzle ORM
- **Authentication**: Better-Auth

📦 Installation

1. ESP32 Setup

Using Arduino IDE

1. **Install Arduino IDE** (v1.8.19 or later)

2. **Add ESP32 Board Support**:

    - File → Preferences
    - Add to "Additional Boards Manager URLs":

    ```
    https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
    ```

    - Tools → Board → Boards Manager
    - Search "ESP32" and install "esp32 by Espressif Systems"

3. **Install Required Libraries**:

    - Sketch → Include Library → Manage Libraries
    - Install: `DHT sensor library by Adafruit`

4. **Configure WiFi**:

    - Open `esp.md` or `esp32_greenhouse.ino`
    - Update WiFi credentials:

    ```
    const char* ssid = "YOUR_WIFI_SSID";
    const char* password = "YOUR_WIFI_PASSWORD";
    ```

5. **Upload Code**:

    - Connect ESP32 via USB
    - Select board: Tools → Board → ESP32 Dev Module
    - Select port: Tools → Port → (your COM port)
    - Click Upload button

6. **Get ESP32 IP Address**:

- Open Serial Monitor (115200 baud)
- Reset ESP32
- Note the IP address displayed (e.g., 192.168.1.100)

2. Frontend Setup

```bash
# Navigate to frontend directory
cd frontend

# Install dependencies
npm install

# Create environment file
cp .env.example .env.local

# Edit .env.local and add ESP32 IP
# NEXT_PUBLIC_ESP_IP=192.168.1.100

# Run development server
npm run dev
```

The dashboard will be available at `http://localhost:3000`

3. Backend Setup

```bash
# Navigate to backend directory
cd backend

# Install dependencies
npm install

# Create environment file
cp .env.example .env

# Configure database connection in .env

# Run development server
npm run dev
```

⚙️ Configuration

ESP32 Configuration

Edit the following constants in the ESP32 code:

```cpp
// WiFi Settings
const char* ssid = "YOUR_NETWORK";
const char* password = "YOUR_PASSWORD";
```

```
// Stepper Motor Settings
const float steps_per_mm = 400.0;        // Steps per millimeter
const float travel_distance_mm = 100.0;  // Maximum travel (mm)
const int speedDelay = 250;              // Speed in microseconds

// Timing
const unsigned long WIFI_CHECK_INTERVAL = 30000;   // 30 seconds
const unsigned long SENSOR_READ_INTERVAL = 5000;   // 5 seconds
```

Frontend Configuration

Create `frontend/.env.local` :

```
# ESP32 IP Address (found in Serial Monitor)
NEXT_PUBLIC_ESP_IP=192.168.1.100

# API endpoints (if using backend)
NEXT_PUBLIC_API_URL=http://localhost:3001
```

Backend Configuration

Create `backend/.env` :

```
PORT=3001
DATABASE_URL=file:./greenhouse.db
NODE_ENV=development
```

📡 API Documentation

Base URL

```
http://<ESP32_IP_ADDRESS>/api
```

Endpoints

GET /api/status

Returns current system status including all sensor readings and device states.

Response:

```
{
  "temperature": 25.5,
  "humidity": 60.2,
  "ph": 1234,
  "distance": 35.7,
  "fan": "0",
  "pump1": "1",
  "pump2": "1",
  "led": "0",
```

```
  "stepper_enabled": "1"
}
```

Note: Device states use inverted logic:

- `"0"` = Device ON
- `"1"` = Device OFF

GET /api/control

Controls devices via query parameters.

Parameters:

Parameter	Values	Description
`fan`	`0, 1`	Control fan (0=ON, 1=OFF)
`pump1`	`0, 1`	Control pump 1
`pump2`	`0, 1`	Control pump 2
`led`	`0, 1`	Control LED lights
`stepper_enable`	`0, 1`	Enable/disable stepper motor
`stepper_move`	`1-100`	Move forward (mm)
`stepper_move_back`	`1-100`	Move backward (mm)

Examples:

```
# Turn fan ON
curl "http://192.168.1.100/api/control?fan=0"

# Turn pump 1 OFF
curl "http://192.168.1.100/api/control?pump1=1"

# Enable stepper motor
curl "http://192.168.1.100/api/control?stepper_enable=1"

# Move roof vent forward 50mm
curl "http://192.168.1.100/api/control?stepper_move=50"

# Move roof vent backward 50mm
curl "http://192.168.1.100/api/control?stepper_move_back=50"
```

Response:

```
OK
```

Or specific message for stepper operations:

```
Moved forward 50 mm
```

Error Responses:

```
ERROR: Stepper disabled. Enable it first!
```

📖 Usage Guide

Initial Setup

1. **Power on ESP32** and ensure it connects to WiFi
2. **Open Serial Monitor** to verify connection and note IP address
3. **Access web dashboard** at `http://localhost:3000`
4. **Test connection** using the "Test Connection" button

Monitoring Sensors

The dashboard automatically updates sensor readings every 5 seconds:

- **Temperature**: Optimal range 20-30°C
- **Humidity**: Optimal range 20-60%
- **pH Level**: Optimal range 6.0-7.5
- **Water Level**: Distance in cm (lower = more water)
- **Air Quality**: MQ135 PPM reading

Sensors display color-coded status:

- 🟢 **Normal**: Within optimal range
- 🟡 **Warning**: Outside optimal range

Controlling Actuators

Individual Control

Click on any actuator card to toggle its state:

- Fan
- LED Strip
- Pump 1
- Pump 2
- Stepper Motor Enable

Batch Control

- **All Pumps ON**: Activate both pumps simultaneously
- **All Pumps OFF**: Deactivate both pumps
- 🚨 **ALL OFF**: Emergency stop - turns off all devices

Roof Vent Control

1. **Enable Stepper Motor** by clicking the Stepper Motor card
2. **Set Movement Distance** (1-100mm) in the input field
3. **Click Movement Buttons**:
   - ➡️ Move Forward: Opens the roof vent
   - ⬅️ Move Backward: Closes the roof vent

Important Notes:

- Motor movement is blocking - wait for completion before next command

- Total travel range: 100mm
- Step resolution: 400 steps/mm
- Always enable motor before attempting movement

Camera Feed

View live camera feed from your greenhouse directly in the dashboard.

🐛 Troubleshooting

ESP32 Issues

WiFi Connection Failed

Symptom: ESP32 cannot connect to WiFi

Solutions:

- Verify SSID and password in code
- Check if WiFi network is 2.4GHz (ESP32 doesn't support 5GHz)
- Move ESP32 closer to router
- Check Serial Monitor for error messages

Sensors Reading Zero or NaN

Symptom: Sensor values show 0 or invalid readings

Solutions:

- Check sensor wiring and power connections
- Verify correct GPIO pins in code
- Test sensors individually with example sketches
- Check sensor power requirements (3.3V vs 5V)

Relays Not Switching

Symptom: Devices don't respond to control commands

Solutions:

- Verify relay module is powered (5V + GND)
- Check if relays use active HIGH or LOW logic
- Test relay manually by connecting GPIO directly to GND
- Ensure relay current rating matches load

Stepper Motor Not Moving

Symptom: Motor doesn't respond or stutters

Solutions:

- Check 12V power supply to driver
- Verify EN pin is LOW (motor enabled)
- Test driver with simple example code
- Check motor wiring (A+, A-, B+, B-)
- Adjust speed delay (increase value to slow down)

Frontend Issues

Cannot Connect to ESP32

Symptom: "Connection Error" displayed on dashboard

Solutions:

- Verify ESP32 IP address in `.env.local`
- Check ESP32 is powered on and connected to WiFi
- Test API directly: `http://<ESP_IP>/api/status` in browser
- Ensure ESP32 and computer are on same network
- Disable browser CORS plugins if any

Sensor Data Not Updating

Symptom: Values stuck or not refreshing

Solutions:

- Check browser console for errors (F12)
- Verify ESP32 `/api/status` endpoint responds correctly
- Clear browser cache
- Check if DATA_FETCH_INTERVAL is set correctly (default: 5000ms)

Controls Not Working

Symptom: Clicking buttons has no effect

Solutions:

- Check debug console in dashboard (development mode)
- Verify rate limiting isn't blocking commands (1 second cooldown)
- Test API directly using curl commands
- Check browser console for JavaScript errors

Network Issues

IP Address Changes

Symptom: ESP32 IP changes after router restart

Solutions:

- Set static IP in router DHCP settings
- Use ESP32 MAC address to reserve IP
- Update `.env.local` if IP changes
- Consider using mDNS (greenhouse.local)

🔒 Security Considerations

1. **Network Security**:
   - Keep ESP32 on isolated IoT network
   - Use WPA2 encryption for WiFi
   - Change default passwords
2. **API Security**:
   - Implement authentication (currently open)

    - Add rate limiting for production
    - Use HTTPS for public access

3. **Physical Security**:

    - Protect ESP32 from moisture
    - Use proper enclosure for electronics
    - Implement emergency shutoff switch

🚀 Future Enhancements

- [ ] Add authentication to ESP32 API
- [ ] Implement automation rules (if temp > X, turn on fan)
- [ ] Data logging and historical charts
- [ ] Mobile app (React Native)
- [ ] Email/SMS alerts for critical conditions
- [ ] Integration with weather APIs
- [ ] Machine learning for optimal scheduling
- [ ] Multi-greenhouse support

📄 License

This project is licensed under the MIT License - see the LICENSE file for details.

🙏 Acknowledgments

- ESP32 community for excellent documentation
- Next.js team for the amazing framework
- Open source contributors for libraries used
- Greenhouse automation community for inspiration