TAZROUT

IoT Irrigation System

ESP32 Sensor Node Development Plan

Module: ESP32 Sensor & Network Communication Layer
Simulation-first approach using Wokwi platform

Developer:	KHENFRI , Moussa
Responsibility:	ESP32 Simulation & Network Layer
Project Duration:	8 weeks (Feb 1 - Mar 28, 2026)
Time Commitment:	15-18 hours/week
Platform:	Wokwi ESP32 Simulator
Version:	2.0 (Enhanced Edition)

Prepared: January 31, 2026

Contents

1 Project Overview

1.1 Objective

This document outlines the development plan for the ESP32 sensor node module of the TAZROUT IoT irrigation system. The primary objective is to design and implement a fully functional sensor and network communication layer capable of:

- Real-time soil moisture monitoring
- Environmental condition tracking (temperature, humidity)
- Reliable data transmission via WiFi and MQTT protocol
- Remote command reception and processing
- Comprehensive data logging and validation

1.2 Technical Specifications

Component	**Specification**
Microcontroller	ESP32 (simulated in Wokwi)
Soil Sensor	Capacitive moisture sensor (analog simulation)
Environment Sensor	DHT22 (temperature & humidity)
Communication	WiFi + MQTT protocol
Actuator	5V relay controlling water pump (simulation)
Control Modes	Manual (MQTT command) & Automatic (threshold-based)
Data Format	JSON
Development Platform	Wokwi ESP32 Simulator

Table 1: ESP32 Module Technical Specifications

1.3 Module Scope and Boundaries

1.3.1 In Scope

This module is responsible for:

- ESP32 firmware development and configuration
- Sensor interfacing and calibration
- Analog and digital I/O operations
- JSON data serialization
- Basic relay control logic
- State machine design
- Error handling and recovery mechanisms
- Module-level documentation

1.4 Learning Outcomes

Upon completion of this 8-week development plan, the following competencies will be achieved:

- Proficiency in ESP32 programming using Arduino IDE
- Sensor interfacing and calibration techniques
- Understanding of analog/digital I/O operations
- WiFi network configuration on embedded systems
- MQTT protocol implementation and message queuing
- JSON data serialization for IoT applications
- Relay control and actuator management
- State machine design patterns
- Error handling strategies for embedded systems
- Code modularization and documentation best practices
- Version control using Git and GitHub
- Technical documentation and presentation skills
- Team collaboration in distributed development

2 Master Timeline

2.1 8-Week Overview

Week	Focus Area	Deliverable	Hours
1	ESP32 Fundamentals	LED control + Analog I/O	15
2	Sensor Integration	Multi-sensor system	17
3	WiFi & MQTT Publishing	Publishing sensor data to broker	16
4	MQTT Bidirectional	Receiving commands + state machine	18
5	Relay Control	Basic relay control via MQTT	16
6	Error Handling	Network recovery + validation	17
7	Documentation	Complete technical documentation	15
8	Team Integration	Final integration & testing	16
		Total:	**130**

Table 2: 8-Week Development Timeline

2.2 Weekly Checkpoint System

Each week follows a structured development cycle:

1. **Planning Phase:** Review previous week progress, identify blockers, plan current week tasks
2. **Development Phase:** Implementation and coding (distributed across available time slots)
3. **Testing Phase:** Unit testing and debugging
4. **Integration Phase:** Ensure new components work with existing codebase
5. **Documentation Phase:** Update technical documentation and commit to repository

2.3 Time Allocation Strategy

Based on schedule availability:

- **Friday:** Deep work sessions (4-6 hours) - primary development
- **Saturday:** Integration and testing (4-6 hours) - build and verify
- **Tuesday/Thursday:** Afternoon sessions from 12:00 (3-4 hours each) - focused tasks
- **Monday/Wednesday/Sunday:** Light work (1-2 hours max) - reading, planning, quick fixes

Total weekly commitment: 15-18 hours

3 Week 1: ESP32 Fundamentals

February 1-7, 2026

3.1 Learning Objectives

- Understand ESP32 architecture, pinout, and capabilities
- Master digital output operations (GPIO control)
- Implement analog input reading (ADC operations)
- Utilize Serial communication for debugging
- Learn non-blocking code patterns using millis()
- Set up development environment and version control

3.2 Detailed Task Breakdown

3.2.1 Day 1: Environment Setup (2-3 hours)

Tasks:

1. Create Wokwi account at `https://wokwi.com`
2. Review Wokwi ESP32 documentation
3. Create first project: "ESP32 LED Blink"
4. Understand project structure: diagram.json, sketch.ino
5. Verify simulation functionality
6. Save project and archive URL

Success Criteria:

- LED blinks at 1Hz frequency
- Simulation runs without errors
- Project URL is accessible and shareable

3.2.2 Day 2: Serial Communication (2-3 hours)

Tasks:

1. Configure Serial Monitor in Wokwi
2. Implement formatted output for different data types
3. Create simulated temperature readings
4. Debug using Serial.print() statements
5. Experiment with different baud rates
6. Document Serial communication patterns

3.2.3 Day 3-4: Traffic Light System (4-5 hours)

Tasks:

1. Add 3 LEDs to circuit: Red (GPIO 13), Yellow (GPIO 12), Green (GPIO 14)
2. Wire LEDs with 220 ohm resistors
3. Define pin constants and state variables
4. Implement traffic light sequence: Green(5s) → Yellow(2s) → Red(5s)
5. Convert blocking delay() to non-blocking millis()
6. Implement state machine using enum
7. Print current state and remaining time to Serial
8. Verify smooth state transitions

Deliverable: Non-blocking traffic light with state-based control.

3.2.4 Day 5: Analog Input (3-4 hours)

Tasks:

1. Add potentiometer to circuit (GPIO 34 - ADC1_CH6)
2. Read raw ADC values using analogRead()
3. Understand ESP32 ADC characteristics (12-bit resolution, 0-4095 range)
4. Map values to percentage: map(value, 0, 4095, 0, 100)
5. Display potentiometer value on Serial Monitor
6. Calibrate by recording minimum and maximum physical values
7. Implement moving average filter (last 5 readings)
8. Test accuracy across full potentiometer range

Calibration Process:

1. Record ADC value at potentiometer minimum position
2. Record ADC value at potentiometer maximum position
3. Use these values for accurate percentage mapping

3.2.5 Day 6: PWM LED Control (3-4 hours)

Tasks:

1. Add LED for PWM control (GPIO 26)
2. Configure PWM: ledcSetup(channel, frequency, resolution)
3. Attach pin to channel: ledcAttachPin(pin, channel)
4. Read potentiometer value

5. Map to PWM duty cycle: map(potValue, 0, 4095, 0, 255)
6. Control LED brightness: ledcWrite(channel, dutyCycle)
7. Display brightness percentage on Serial Monitor
8. Implement smooth fade effect using millis()

3.2.6 Day 7: Git Setup & Weekly Review (2-3 hours)

Tasks:

1. Initialize local Git repository
2. Organize Week 1 code into appropriate directories
3. Create README.md with project description
4. Document Week 1 learnings and challenges

3.3 Week 1 Deliverables

Required Outputs

1. Working Wokwi project: PWM-controlled LED with potentiometer
2. Organized local code repository
3. README.md documenting Week 1 progress
4. Technical notes on ESP32 fundamentals
5. List of questions or challenges encountered

Team Sharing:

- Wokwi project URL
- Screenshot of Serial Monitor output
- Brief summary of learning outcomes

4 Week 2: Sensor Integration

February 8-14, 2026

4.1 Learning Objectives

- Interface with analog moisture sensor (simulated)
- Integrate DHT22 digital temperature/humidity sensor
- Implement sensor calibration techniques
- Build data logging and filtering systems
- Handle sensor errors and validation
- Create structured data management

4.2 Detailed Task Breakdown

4.2.1 Day 8: Moisture Sensor Research & Setup (2-3 hours)

Research Tasks:

1. Compare capacitive vs resistive moisture sensors
2. Understand sensor output characteristics
3. Study voltage-to-moisture relationship
4. Review sensor placement considerations for soil monitoring
5. Identify operating range and power requirements
6. Plan calibration methodology

Documentation: Create technical notes covering:

- Sensor operating principles
- Expected voltage ranges
- Calibration requirements
- Potential sources of error

4.2.2 Day 9: Moisture Sensor Implementation (3-4 hours)

Tasks:

1. Add potentiometer as moisture sensor simulation (GPIO 35 - ADC1_CH7)
2. Measure and record "dry" value (potentiometer at minimum)
3. Measure and record "wet" value (potentiometer at maximum)
4. Implement readMoisture() function
5. Convert raw ADC to percentage with bounds checking
6. Add moisture status categorization: Dry (0-30%), Optimal (30-70%), Wet (70-100%)
7. Test across full potentiometer range
8. Create visual feedback using LED indicators

4.2.3 Day 10-11: DHT22 Integration (4-5 hours)

Tasks:

1. Add DHT22 sensor to Wokwi circuit (GPIO 16)
2. Include DHT library: #include ¡DHT.h¿
3. Initialize sensor with correct pin and type
4. Implement temperature reading function
5. Implement humidity reading function
6. Add NaN (Not-a-Number) error checking
7. Implement retry logic on sensor failure (maximum 3 attempts)
8. Display all three sensor values in formatted output
9. Test sensor response to simulated environmental changes

4.2.4 Day 12-13: Data Logging System (4-5 hours)

Tasks:

1. Create SensorData structure
2. Implement circular buffer for last 10 readings
3. Calculate moving average for each sensor
4. Find minimum and maximum values in buffer
5. Detect rapid changes exceeding threshold
6. Format timestamps from millis()
7. Create displaySensorData() function
8. Implement logging at 30-second intervals
9. Simulate EEPROM persistence for data backup

4.2.5 Day 14: Multi-Sensor Integration (3-4 hours)

Tasks:

1. Combine all sensor code into single organized file
2. Create updateSensors() function (called every 5 seconds)
3. Implement status LED system with color coding
4. Add sensor health monitoring
5. Create JSON-formatted output structure
6. Test edge cases: sensor disconnection, extreme values
7. Optimize memory usage

8. Add comprehensive code comments

Status LED Logic:

- **Green:** All sensors OK, optimal conditions
- **Yellow:** Warning (low moisture OR high temperature)
- **Red:** Critical (sensor failure OR multiple issues)

JSON Output Format:

```
{
    "node_id": "ESP32_001",
    "moisture": 45.2,
    "temperature": 24.5,
    "humidity": 62.0,
    "timestamp": 123456,
    "status": "OK"
}
```

4.3 Week 2 Deliverables

Required Outputs

1. Multi-sensor reading system (moisture + temperature + humidity)
2. Data logging system with moving average filter
3. Status LED indicator system
4. Sensor calibration documentation
5. Updated repository with Week 2 code

Team Sharing:

- Wokwi project URL (working demo)
- Screenshots of Serial Monitor output showing all sensor readings
- Calibration values and methodology document
- Brief technical report: "Integrated 3 sensors with data validation"

5 Week 3: WiFi & MQTT Publishing working with network team

February 15-21, 2026

5.1 Learning Objectives

- Configure ESP32 WiFi connectivity
- Understand MQTT protocol architecture
- Implement MQTT publishing
- Serialize sensor data to JSON format
- Handle network connectivity issues
- Test with external MQTT broker

5.2 Detailed Task Breakdown

5.2.1 Day 15-16: WiFi Configuration (4 hours)

WiFi Setup Tasks:

1. Include WiFi library: #include ¡WiFi.h¿
2. Connect to Wokwi-GUEST network (Wokwi simulation)
3. Implement connection timeout mechanism (30 seconds)
4. Add automatic reconnection logic
5. Display IP address upon successful connection
6. Implement WiFi status LED indicator
7. Test connection stability under various conditions

Reconnection Logic:

1. Monitor WiFi status in main loop
2. Attempt reconnection if disconnected
3. Implement exponential backoff for retries
4. Log all connection events with timestamps

5.2.2 Day 17-19: MQTT Publishing (6 hours)

MQTT Setup Tasks:

1. Install PubSubClient library in Wokwi
2. Research MQTT fundamentals: brokers, topics, QoS levels
3. Connect to public MQTT broker (test.mosquitto.org:1883)
4. Understand topic naming conventions

5. Create MQTT client instance
6. Implement connection function with client ID

JSON Payload Creation:

1. Include ArduinoJson library
2. Create JSON document
3. Add sensor readings
4. Add metadata (node_id, timestamp, status)
5. Serialize to string
6. Publish to topic: "tazrout/sensors/data"

Testing Tasks:

1. Install MQTT Explorer (desktop application) or use HiveMQ web client
2. Subscribe to "tazrout/sensors/data" topic
3. Verify JSON messages are received
4. Check message structure and data validity
5. Test publishing frequency (every 30 seconds)
6. Implement publish success/failure handling

5.2.3 Day 20-21: Network Error Handling (3 hours)

Tasks:

1. Implement MQTT keep-alive mechanism
2. Add connection state monitoring
3. Create reconnection logic for MQTT
4. Implement Last Will and Testament (LWT)
5. Test behavior during network failures
6. Add comprehensive error logging
7. Optimize code for reliability

5.3 Week 3 Deliverables

Required Outputs

1. ESP32 publishing sensor data via MQTT
2. WiFi connection management with auto-reconnect
3. JSON-formatted sensor data messages
4. MQTT topic structure documentation
5. Network error handling implementation

Team Sharing:

- Working Wokwi project URL
- MQTT topic documentation
- Screenshots from MQTT Explorer showing published messages
- Demo video (1-2 minutes) showing WiFi connection and MQTT publishing

6 Week 4: MQTT Bidirectional Communication

February 22-28, 2026

6.1 Learning Objectives

- Implement MQTT subscription
- Parse incoming JSON commands
- Design and implement state machine
- Handle command acknowledgments
- Build robust message buffering
- Implement Quality of Service (QoS) levels

6.2 Detailed Task Breakdown

6.2.1 Day 22-23: MQTT Subscription (4 hours)

Subscription Tasks:

1. Subscribe to command topic: "tazrout/commands"
2. Implement callback function for received messages
3. Parse incoming message payload
4. Validate message format
5. Extract command parameters
6. Log all received commands

Command Parsing:

1. Use ArduinoJson to deserialize incoming messages
2. Validate required fields (node_id, command, timestamp)
3. Check if node_id matches this device
4. Implement command types:
    - status_request - Send current sensor readings
    - config_update - Update configuration parameters
    - relay_control - Manual relay on/off command

Command Format Example:

```
{
    "node_id": "ESP32_001",
    "command": "status_request",
    "timestamp": 1675234567
}
```

6.2.2 Day 24-25: Command Processing (4 hours)

Tasks:

1. Create command handler functions for each command type
2. Implement command execution logic
3. Add command validation (parameter checking)
4. Create acknowledgment message structure
5. Publish acknowledgment to "tazrout/ack" topic
6. Implement command logging with timestamps
7. Handle invalid or malformed commands
8. Test each command type individually

Acknowledgment Format:

```
{
    "node_id": "ESP32_001",
    "command": "status_request",
    "result": "success",
    "timestamp": 1675234567
}
```

6.2.3 Day 26-27: State Machine Implementation (6 hours)

State Design Tasks:

1. Define system states
2. Create state transition rules
3. Implement state-specific behaviors
4. Add state timeout mechanisms
5. Create state logging system
6. Implement state visualization via LED
7. Test all possible state transitions

State LED Indicators:

- INIT: Blue blinking
- IDLE: Green solid
- READING: Yellow blinking
- PUBLISHING: Cyan blinking
- RECEIVING: Magenta blinking
- ERROR: Red solid

6.2.4 Day 28: Advanced Error Handling (4 hours)

Tasks:

1. Implement QoS levels (0, 1, 2) for MQTT
2. Create message buffer for offline periods
3. Implement message retry logic
4. Add heartbeat mechanism (publish every 60 seconds)
5. Enhance Last Will and Testament
6. Test network failure scenarios
7. Monitor and optimize memory usage
8. Create comprehensive error log

6.3 Week 4 Deliverables

Required Outputs

1. Bidirectional MQTT communication system
2. State machine implementation with visual feedback
3. Command processing with acknowledgments
4. Message buffering for network resilience
5. State diagram documentation
6. Integration test report

Team Sharing:

- Working Wokwi project with full pub/sub functionality
- State machine diagram (visual representation)
- MQTT topic structure document (updated)
- Demo video (2-3 minutes) showing:
  - Publishing sensor data
  - Receiving and processing commands
  - State transitions
  - Error recovery

7 Week 5: Relay Control Basics Optional

March 1-7, 2026

7.1 Learning Objectives

- Understand relay operation principles
- Implement safe relay control
- Add manual/automatic control modes
- Build basic irrigation logic
- Implement safety mechanisms
- Create irrigation event logging

7.2 Detailed Task Breakdown

7.2.1 Day 29-30: Relay Setup & Manual Control (4 hours)

Tasks:

1. Add relay module to Wokwi circuit (GPIO 25)
2. Understand relay as digital output (HIGH = ON, LOW = OFF)
3. Implement controlRelay() function
4. Add relay status LED indicator
5. Implement manual control via MQTT command
6. Test relay ON/OFF operations
7. Verify relay response time

7.2.2 Day 31-32: Safety Mechanisms (4 hours)

Safety Features to Implement:

1. Maximum ON time limit (5 minutes per cycle)
2. Minimum OFF time (cooldown period: 10 minutes)
3. Emergency stop command (immediate shutdown)
4. Watchdog timer (auto-off if no command for 10 minutes)
5. Sensor validation before irrigation
6. Safety check function called before every relay operation

7.2.3 Day 33-34: Automatic Mode (4 hours)

Tasks:

1. Implement dual control modes: MANUAL and AUTO
2. Define moisture thresholds for automatic control
3. Create mode switching functionality
4. Implement automatic irrigation logic
5. Add mode status to published data
6. Test mode transitions
7. Verify threshold-based activation

7.2.4 Day 35: Event Logging & Documentation (4 hours)

Tasks:

1. Create irrigation event logging system
2. Log: start time, end time, duration, reason (manual/auto)
3. Implement event buffer (last 20 events)
4. Add event retrieval command
5. Document all safety features
6. Create relay operation flowchart
7. Test comprehensive scenarios
8. Update technical documentation

7.3 Week 5 Deliverables

Required Outputs

1. Working relay control via MQTT (manual mode)
2. Comprehensive safety mechanisms
3. Automatic mode with threshold-based control
4. Irrigation event logging system
5. Safety features documentation
6. Relay operation flowchart

Team Sharing:

- Wokwi project URL with relay functionality
- Safety features documentation
- Demo video showing:

  - Manual relay control
  - Automatic mode operation
  - Safety mechanism activation

- Event log sample data

8 Week 6: Error Handling & Robustness

March 8-14, 2026

8.1 Learning Objectives

- Implement comprehensive error handling
- Add network recovery mechanisms
- Create sensor data validation
- Build system health monitoring
- Optimize code for reliability
- Conduct stress testing

8.2 Detailed Task Breakdown

8.2.1 Day 36-37: Network Recovery (5 hours)

Tasks:

1. Implement robust WiFi reconnection with exponential backoff
2. Enhance MQTT reconnection logic
3. Add message queue persistence during disconnection
4. Implement Last Will and Testament (LWT) properly
5. Create heartbeat mechanism
6. Add network quality monitoring (WiFi RSSI)
7. Test various network failure scenarios

8.2.2 Day 38-39: Sensor Validation (4 hours)

Tasks:

1. Implement value range validation for each sensor
2. Add sensor drift detection
3. Create sensor failure detection mechanism
4. Implement sensor health status tracking
5. Add sensor diagnostics to published data
6. Test with artificially corrupted sensor data

8.2.3 Day 40-41: System Health Monitoring (4 hours)

Tasks:

1. Create SystemHealth structure
2. Implement comprehensive health checks
3. Add memory usage monitoring
4. Create health report publishing
5. Implement soft reset on critical errors
6. Add diagnostic mode for troubleshooting
7. Monitor and log all system metrics

8.2.4 Day 42: Stress Testing (4 hours)

Test Scenarios:

1. 24-hour continuous operation test (simulated)
2. Rapid sensor value changes
3. Multiple sequential network failures
4. MQTT broker disconnection/reconnection cycles
5. Sensor failure scenarios
6. Memory leak detection
7. Buffer overflow testing
8. Command flooding test

8.3 Week 6 Deliverables

Required Outputs

1. Robust network recovery implementation
2. Comprehensive sensor validation system
3. System health monitoring dashboard
4. Stress test results report
5. Error handling documentation
6. Memory optimization report

Team Sharing:

- Updated Wokwi project with all robustness features
- Stress test report with results
- System health monitoring output samples
- Network recovery demonstration video

9 Week 7: Documentation & Code Quality

March 15-21, 2026

9.1 Learning Objectives

- Refactor code for modularity and maintainability
- Write comprehensive technical documentation
- Create user guides and API documentation
- Prepare professional deliverables
- Produce demonstration materials

9.2 Detailed Task Breakdown

9.2.1 Day 43-44: Code Refactoring (5 hours)

Refactoring Tasks:

1. Split code into modular files
2. Create header files for each module
3. Standardize function naming conventions
4. Add comprehensive code comments
5. Remove duplicate code
6. Optimize memory usage
7. Implement consistent error codes
8. Create configuration header file

Modular File Structure:

```
esp32-sensor-node/src/
 main.ino                    // Main program loop
 config.h                    // Configuration constants
 sensors.h                   // Sensor function declarations
 sensors.cpp                 // Sensor implementations
 network.h                   // WiFi/MQTT declarations
 network.cpp                 // Network implementations
 relay.h                     // Relay control declarations
 relay.cpp                   // Relay implementations
 state_machine.h             // State machine declarations
 state_machine.cpp           // State machine logic
 utils.h                     // Utility functions
```

9.2.2 Day 45-46: Technical Documentation (5 hours)

Documentation Tasks:

1. Write comprehensive README.md
2. Create CHANGELOG.md with version history
3. Document MQTT API (topics and message formats)
4. Create sensor calibration guide
5. Write troubleshooting guide
6. Add inline code documentation (Doxygen style)
7. Create FAQ section

9.2.3 Day 47-48: Visual Documentation (5 hours)

9.2.4 Day 49: Demo Video Production (5 hours)

9.3 Week 7 Deliverables

Required Outputs

1. Refactored, modular codebase
2. Complete README.md with all sections
3. MQTT API documentation
4. System architecture diagrams
5. Calibration and troubleshooting guides
6. Professional demo video (7-10 minutes)
7. All code committed with proper version control

Team Sharing:

- Final GitHub repository with clean structure
- Complete documentation package
- Demo video (unlisted YouTube link)
- Presentation slides (for Week 8)

10 Week 8: Team Integration & Final Delivery

March 22-28, 2026

10.1 Learning Objectives

- Integrate ESP32 module with team's backend
- Conduct comprehensive system testing
- Prepare final presentation
- Deliver professional demonstration
- Complete project handoff

10.2 Detailed Task Breakdown

10.2.1 Day 50-51: Integration Meeting & Setup (4 hours)

Meeting Agenda:

1. Review ESP32 module capabilities and limitations
2. Discuss backend API endpoints
3. Align on MQTT topic structure (if changes needed)
4. Confirm JSON message formats
5. Identify any required modifications
6. Plan integration testing approach
7. Set timeline for integration milestones

Integration Checklist:

- Verify MQTT broker address (production vs test)
- Confirm topic naming convention
- Validate message payload structures
- Test authentication (if required)
- Verify timestamp format compatibility
- Check data type consistency
- Confirm error code standards

10.2.2 Day 52-53: Integration Testing (6 hours)

Test Scenarios:

1. End-to-end data flow test
   - ESP32 → MQTT → Backend → Database
   - Verify data appears in backend logs
   - Check database entries for accuracy
2. Command flow test
   - Frontend/Backend → MQTT → ESP32
   - Test all command types
   - Verify acknowledgment reception
3. Dashboard integration
   - Confirm sensor data displays correctly
   - Test real-time updates
   - Verify control buttons functionality
4. Error scenario testing
   - Sensor failure handling
   - Network disconnection recovery
   - Invalid command handling

10.2.3 Day 54-55: Final Presentation Preparation (4 hours)

10.2.4 Day 56: Final Delivery & Demonstration (6 hours)

Pre-Delivery Checklist:

- Final code review and cleanup
- All documentation complete and accurate
- Demo video uploaded and accessible
- Backup video prepared for live demo
- Presentation slides finalized
- All team members informed of their roles

Deliverables Package:

1. GitHub Repository
   - Clean, organized codebase
   - Complete documentation
   - README with quick start guide
   - All commits properly tagged
2. Wokwi Project

- Final working simulation
- Public/shareable link
- Embedded in documentation

3. Documentation Package
   - Technical documentation
   - API reference
   - User guide
   - Calibration procedures
   - Troubleshooting guide
4. Presentation Materials
   - PowerPoint/PDF slides
   - Demo video (7-10 minutes)
   - Integration test report
5. Individual Report
   - Personal learnings and reflections
   - Contribution summary
   - Challenges overcome
   - Skills acquired

Live Demonstration Checklist:

- Wokwi simulation ready
- MQTT Explorer open and connected
- Serial Monitor configured
- Backup video ready (in case of technical issues)
- Questions anticipated and answers prepared

10.3 Week 8 Deliverables

Final Submission Package

1. Fully integrated ESP32 module working with team system
2. Final presentation slides (PDF format)
3. Live demonstration or backup video
4. Complete GitHub repository with all code and docs
5. Integration test results
6. Individual contribution report
7. Lessons learned document

10.4 Collaboration Guidelines

- Each team member works primarily in their designated folder
- Coordinate before making changes to shared files (main README, docs/)
- Communicate major changes in team chat before pushing
- Pull latest changes before starting new work
- Commit frequently (daily when actively developing)
- Write clear commit messages
- Tag releases at major milestones

11 Success Metrics & Monitoring

11.1 Weekly Self-Assessment

At the end of each week, evaluate progress using the following criteria (1-5 scale):

Criterion	Assessment Questions
Understanding (1-5)	Do I understand the core concepts covered this week? Can I explain them to someone else?
Code Quality (1-5)	Is my code clean, well-commented, and following best practices?
Documentation (1-5)	Have I documented all functions, decisions, and learnings adequately?
Time Management (1-5)	Did I stay on schedule? Was my time allocation effective?
Problem Solving (1-5)	How effectively did I handle challenges? Did I seek help when needed?
Team Collaboration (1-5)	Did I communicate effectively? Are deliverables ready for integration?

Action Plan Based on Scores:

- Score 4-5: Excellent progress, continue current approach
- Score 3: Good progress, identify minor improvements
- Score 1-2: Needs attention - review concepts, seek help, adjust timeline

11.2 Red Flags and Corrective Actions

Red Flag	Description	Action
Schedule Slip	Behind by 2+ days	Reassess priorities, ask for help, adjust next week's plan
Concept Gap	Not understanding fundamentals	Re-study materials, watch additional tutorials, schedule team discussion
Debugging Loop	Code not working after 3+ hours	Take a break, use different debugging approach, ask team/online communities
No Commits	Haven't pushed to Git in 3+ days	Commit current work immediately, establish daily commit habit
Burnout Signs	Feeling overwhelmed or unmotivated	Take a rest day, break tasks into smaller pieces, discuss with team

11.3 Progress Tracking

Use a simple weekly checklist to track completion:

```
Week X Progress Tracker

Tasks Completed: [X/Y]
Hours Spent: [Actual vs Planned]
```

```
Blockers: [List any obstacles]
Learnings: [Key takeaways]
Next Week Preparation: [What to focus on]

Status: [On Track / Slight Delay / Needs Adjustment]
```

12 Resources and References

12.1 Essential Documentation

- **Wokwi Documentation:** https://docs.wokwi.com
- **ESP32 Arduino Core:** https://docs.espressif.com/projects/arduino-esp32
- **MQTT Protocol Essentials:** https://www.hivemq.com/mqtt-essentials/
- **ArduinoJson Documentation:** https://arduinojson.org/v6/doc/
- **PubSubClient Library:** https://pubsubclient.knolleary.net/
- **DHT Sensor Library:** https://github.com/adafruit/DHT-sensor-library

12.2 Video Tutorials (Recommended)

ESP32 Fundamentals:

- "ESP32 Tutorial for Beginners" - Random Nerd Tutorials
- "Wokwi ESP32 Simulator Complete Guide"
- "ESP32 Analog Input and ADC Tutorial"

Sensor Integration:

- "DHT22 Temperature & Humidity Sensor with ESP32"
- "Capacitive Soil Moisture Sensor Tutorial"
- "ESP32 Multi-Sensor Data Logging"

Network Communication:

- "MQTT Protocol Explained" - The Engineering Mindset
- "ESP32 WiFi and MQTT Complete Tutorial" - DroneBot Workshop
- "ESP32 MQTT with JSON - Complete Guide"

Advanced Topics:

- "ESP32 State Machine Design Patterns"
- "Error Handling in Embedded Systems"
- "ESP32 Relay Control and Safety Mechanisms"

12.3 Example Projects for Reference

1. **Smart_Farming** by agungferdi
   - GitHub: https://github.com/agungferdi/Smart_Farming
   - Features: ESP32, DHT22, soil moisture, MQTT via HiveMQ
   - Architecture: IoT device → MQTT → Backend → Database
   - Useful for: MQTT implementation patterns
2. **Smart-Irrigation-System** by Circuit-Digest

- GitHub: https://github.com/Circuit-Digest/Smart-Irrigation-System-Using-ESP32-Blynk
- Features: ESP32, soil sensor, DHT22, relay, Blynk app
- Control modes: Automatic + manual
- Useful for: Relay control logic and UI ideas

3. **Plantwatery** by Lumics
   - GitHub: https://github.com/Lumics/Plantwatery
   - Features: Autonomous system with solar power
   - Advanced: OTA updates, MQTT communication
   - Useful for: System architecture and optimization
4. **LazyGardener** by pilotak
   - GitHub: https://github.com/pilotak/LazyGardener
   - Features: ESP32 sprinkler controller
   - Integration: HomeAssistant
   - Useful for: Home automation integration patterns

12.4 Community Resources

- ESP32 Forum: https://esp32.com/
- Arduino Forum - ESP32 Section
- Reddit: r/esp32
- Stack Overflow - esp32 tag
- Discord: ESP32 Development Community

12.5 Tools and Software

- **Arduino IDE:** Primary development environment
- **Wokwi:** ESP32 online simulator
- **MQTT Explorer:** Desktop MQTT client for testing
- **HiveMQ Web Client:** Browser-based MQTT testing
- **Git:** Version control system
- **VS Code:** Alternative code editor (optional)
- **Serial Monitor:** For debugging (built into Arduino IDE / Wokwi)

13 Appendix

13.1 Glossary of Terms

- **ADC (Analog-to-Digital Converter):** Hardware that converts continuous analog signals to discrete digital values
- **Broker:** MQTT server that routes messages between publishers and subscribers
- **GPIO (General Purpose Input/Output):** Programmable pins on microcontroller
- **JSON (JavaScript Object Notation):** Lightweight data interchange format
- **MQTT (Message Queuing Telemetry Transport):** Lightweight messaging protocol for IoT
- **PWM (Pulse Width Modulation):** Technique for controlling power to devices through rapid on/off cycling
- **QoS (Quality of Service):** MQTT message delivery guarantee level (0, 1, or 2)
- **State Machine:** Design pattern that manages system states and transitions

13.2 Contact Information

For questions or collaboration:

- **Developer:** KHENFRI Moussa
- **Email:** moussakhanfri04@gmail.com

Document Version 2.0
Last Updated: January 31, 2026

Prepared for TAZROUT IoT Irrigation System Project