SOICT

Lecturer: Dr. Nguyen Duc Tien

Lab Manage Control

Graduation II Report

Student:
Doan Thi Thu Quyen 20226063

Contents

1 Objective

This report documents the learning and research process for the Lab Manage Control project, serving as a foundation and a stepping stone for my future graduation thesis.

2 Introduction: Adopting System Thinking

The primary takeaway from this project was shifting from writing isolated code to developing a **System Thinking mindset**.
The core problem was not just creating software, but solving a "blind spot" in computer lab management: administrators cannot verify the status of every machine or identify the user sitting at it without manual inspection.
The solution required bridging the gap between Software (Centralized Management) and Hardware (Physical Interaction), proving that effective IoT solutions must integrate both domains seamlessly.

3 Architectural Insights and Communication Protocols

A significant portion of the learning curve involved selecting the right protocols for a distributed network environment.

3.1 MQTT

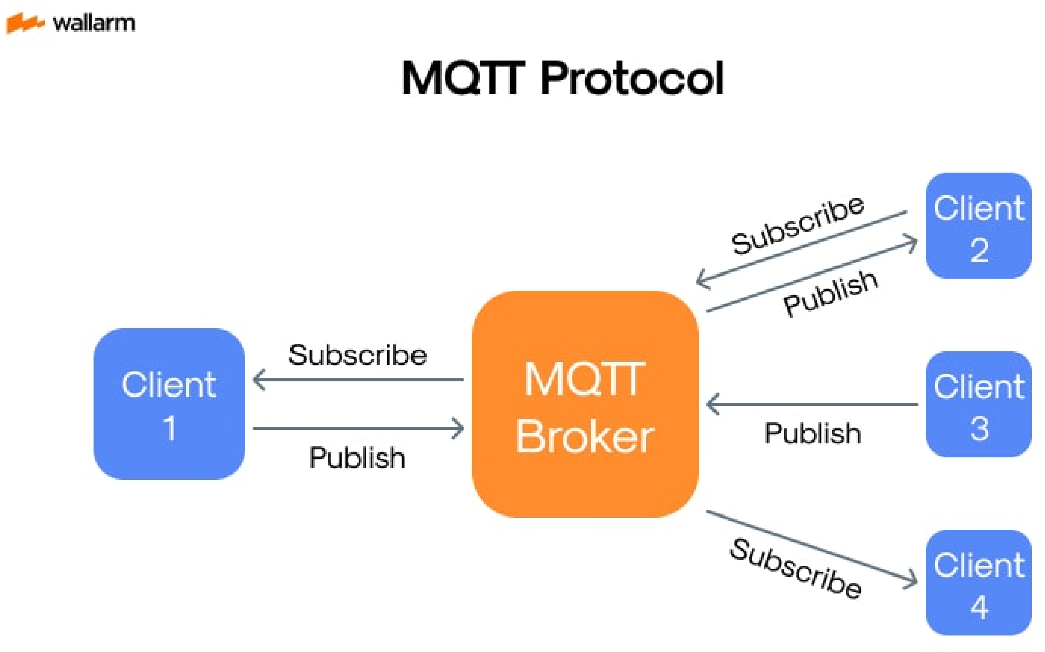


Figure 1: MQTT Protocol

Initially, HTTP or WebSocket might seem like standard choices. However, this project demonstrated that MQTT is far superior for this specific use case due to network instability in lab environments (described by some belowed reasons).

- **Decoupling via Publish/Subscribe**: The system decouples the sender (Terminal Service) from the receiver (LocalCenterForm). If one side disconnects, the system does not hang or crash.
- **Retain Flag** By using the MQTT Retain Flag, the Management App receives the last known status of a computer immediately upon subscribing, without waiting for the next heartbeat cycle. Standard HTTP requests cannot achieve this efficiency without complex polling mechanisms.

3.2 Reliable Serial Communication

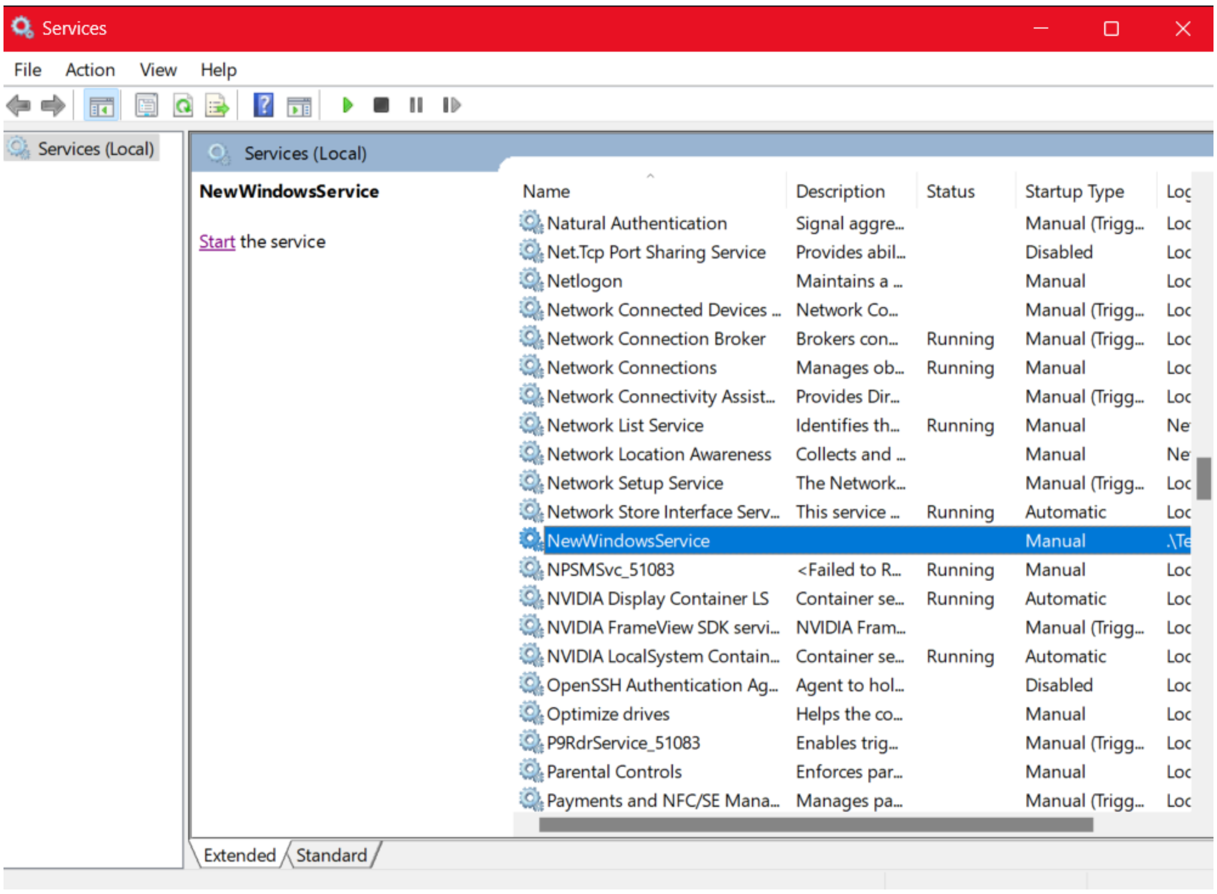
For local communication between the PC and the embedded device, the project was choosen to prioritize stability over wireless convenience.

- **Decision**: Instead of using Bluetooth or WiFi (which are prone to interference or disconnection in sleep modes), Serial (USB/COM) was selected for its absolute stability and high speed (115200 baud).
- **Data Integrity**: JSON encapsulation for serial data transmission. Sending structured data (e.g., "Room": "B1", "IP": "...") prevents bit-shift errors and makes debugging significantly easier compared to sending raw bytes.

4 Software Development Journey

The software stack provided deep insights into the MVC (Model-View-Controller) pattern and Windows internal architecture.
During the software development process for this system, the project was implemented with five core technical areas.

4.1 Windows Service and Background Process Management



Name	Description	Status	Startup Type	Log
Natural Authentication	Signal aggre...		Manual (Trigg...	Loc
Net.Tcp Port Sharing Service	Provides abil...		Disabled	Loc
Netlogon	Maintains a ...		Manual	Loc
Network Connected Devices ...	Network Co...		Manual (Trigg...	Loc
Network Connection Broker	Brokers con...	Running	Manual (Trigg...	Loc
Network Connections	Manages ob...	Running	Manual	Loc
Network Connectivity Assist...	Provides Dir...		Manual (Trigg...	Loc
Network List Service	Identifies th...	Running	Manual	Ne
Network Location Awareness	Collects and ...		Manual	Ne
Network Setup Service	The Network...		Manual (Trigg...	Loc
Network Store Interface Serv...	This service ...	Running	Automatic	Loc
NewWindowsService			Manual	.\Te
NPSMSvc_51083	<Failed to R...	Running	Manual	Loc
NVIDIA Display Container LS	Container se...	Running	Automatic	Loc
NVIDIA FrameView SDK servi...	NVIDIA Fram...		Manual (Trigg...	Loc
NVIDIA LocalSystem Contain...	Container se...	Running	Automatic	Loc
OpenSSH Authentication Ag...	Agent to hol...		Disabled	Loc
Optimize drives	Helps the co...		Manual	Loc
P9RdrService_51083	Enables trig...		Manual (Trigg...	Loc
Parental Controls	Enforces par...		Manual	Loc
Payments and NFC/SE Mana...	Manages pa...		Manual (Trigg...	Loc

Figure 2: Window Service Example

- **Session Isolation**: I learned to build a **Windows Service** (NewWindowsService) that runs in Session 0. Unlike standard forms, this service auto-starts with the OS and operates without a GUI, which is essential for security and consistency.
- **Debugging Solution**: Since services have no interface, the project was choosen to integrated the **Log4net** library to implement "Rolling File" logging. This ensures that history is preserved for debugging without consuming infinite disk space (rotating 10 files, 50MB each)
- **Permission Management**: The Service needs to run under a "Local Account" with **administrative privileges** to execute sensitive commands like Shutdown or Kill Process effectively.

4.2 Asynchronous Communication and Data Handling (MQTT and JSON)

- **Async/Await Pattern with MQTTnet**: Using the **MQTTnet** library to connect to the HiveMQ Broker. The asynchronous programming model (async Task) was applied for PublishAsync and SubscribeAsync functions.
  Futhermore, the project used the **Retain Flag**. When sending machine status messages (Online/Offline) or configuration data, the code set **WithRetainFlag()**.

This ensures that when the Management Machine (LocalCenterForm) starts up or reconnects to the network, it immediately receives the last known status of the client machine without waiting for the client to resend data.

- **Topic Hierarchy**: The software code designed a tree-based topic structure: Room/MachineID/Topic (e.g., B1-201/24/thongtin),. This makes data subscription flexible: one can subscribe to a specific branch or use wildcards (#) to listen to everything.

4.3 Serialization and Deserialization in Hardware Communication

Communication between C# (.NET) and C++ (ESP32) via COM ports often suffers from data corruption.

- **The Challenge**: Sending raw strings can easily lead to missing characters or difficulties in parsing data fields.
- **Solution**: Using JSON libraries (implied Newtonsoft.Json on C# and ArduinoJson on ESP32) to encapsulate data into objects.
- **Implementation**:
  - Sending: "Room": "B1-201", "IP": "192.168.1.10"
  - Receiving: Using SerialPort.ReadLine() to read a complete JSON line, then Deserializing it into an object to extract the RFID tag ID.

4.4 Decoupling Strategy in Configuration

- **The Problem**: Hard-coding room numbers or machine IDs into the C code makes installing the software on 50 different computers a disaster (requiring 50 different .exe builds).
- **Solution**: Using Settings.settings combined with a separate Console Application (TerminalConf). Configuration data is stored in a **user.config** XML file located in the AppData folder. TerminalConf.exe accepted command-line arguments. When this file is run, it directly modifies the user.config file without needing to recompile the Service. A single installer **(Setup.exe)** is used for all machines. Technicians only need to run one command line to identify the machine after installation.

4.5 Full-Stack Integration

The project bridged disparate technologies to create a cohesive data flow:
RFID Handling: Reading hardware signals -> **Web API Call** (ASP.NET hosted on Render.com) -> **Database Query**(PostgreSQL) -> **UI Display** (Windows Forms)

- **Technology Stack Selection**:

Figure 3: Web Tech Stack

  - **Backend Framework**: ASP.NET Web API. Controller-Action model was choosen to organize code clearly, specifically implementing HTTP GET methods to retrieve student data via UID
  - **Database**: Unlike a simple text file, The relational database was used to store student attributes (UID, Name, DOB, Department, MSSV). DBeaver was used for database management and query testing, which gave practical SQL experience.
  - **Cloud Deployment**: Instead of running the database on localhost, the web was deployed the API and Database to the cloud using Render.com.

- **End-to-End Data Flow**
  - **Physical Layer**: The student scans a card. The ESP32 reads the UID 7cccf1 and sends it via Serial to the Client PC.
  - **Service Layer**: The Terminal Service (Client PC) receives the UID and publishes it to the MQTT Broker (studentTagId topic).
  - **Application Layer**: The LocalCenterForm (Admin PC) receives the UID via MQTT.
  - **Cloud Layer**: The LocalCenterForm (Admin PC) receives the UID via MQTT.
    - The Desktop App uses HttpClient to send a GET request to the cloud endpoint,
    - The Web API queries PostgreSQL, serializes the result into JSON, and returns it.
    - The Desktop App deserializes the JSON and displays on the teacher's screen

5 Hardware Engineering Journey

The hardware phase marked the transition from theoretical logic to physical product realization.

5.1 Understanding Hardware Components and Design

- **Central Processing Unit**: The core of the system is the ESP32-WROOM-32, which acts as the central controller connecting all other peripherals.
- **Key Components**:

- **Power Supply**: A voltage regulator (AMS1117-3.3) is used to stabilize the power for the circuit
- **User Interface**:
  - OLED Display: Used for visualizing system information
  - Control Buttons: The system includes 4 distinct buttons for user input
  - Feedback: A Speaker provides audio confirmation, and an LED Ring offers visual effects (such as breathing effects or color changes)

- **Communication and Sensors**:
  - RFID Module: For reading RFID tags
  - USB-to-Serial: Facilitates communication and code uploading
- **Connectivity and Operation Logic**:
  - Reset and Boot Sequence:
    - The RST button is connected to the EN pin, and the BOOT button is connected to the IO0 pin
    - Normal Reset: Pressing the RST button resets the board to normal operation
    - Flash Mode: To enter the code download mode, the user must hold the BOOT button, press RST, release RST, and finally release BOOT
- **Interaction**: User actions via the buttons or Serial commands trigger responses from the LED ring and the piezo speaker to confirm operations

5.2 Schematic Draw

Based on the existing design documentation, the **schematic was redrawn** to support the PCB design process. The EasyEDA software was utilized for this purpose.
To support the schematic design process, I referred to drawing methods and learned through EasyEDA tutorial videos on EasyEDA Channel on YouTube.
After drawing the schematic, the **Design Rule Check** (DRC) was performed to identify and check for schematic errors.

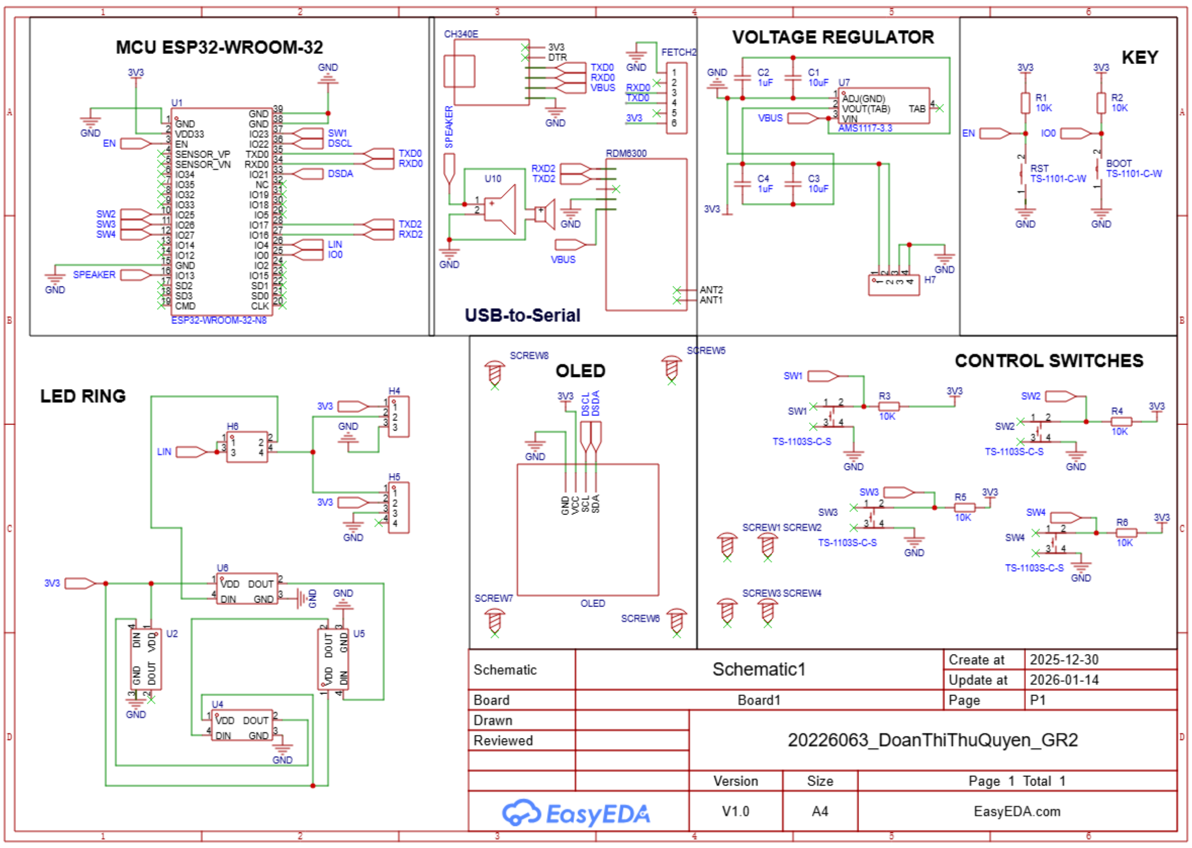


Figure 4: Schematic Design

5.3 PCB Design

After completing the schematic, **additional footprints** were **created** for devices that lacked predefined footprints. The missing device specifications were researched to ensure accurate footprint design.
The schematic was then converted into a PCB design in EasyEDA.
Next, the **components were rearranged** to optimize the PCB layout, and appropriate layers were assigned to each footprint.
Finally, **routing** was **performed**, and a **Design Rule Check** (DRC) was conducted to verify the PCB design.

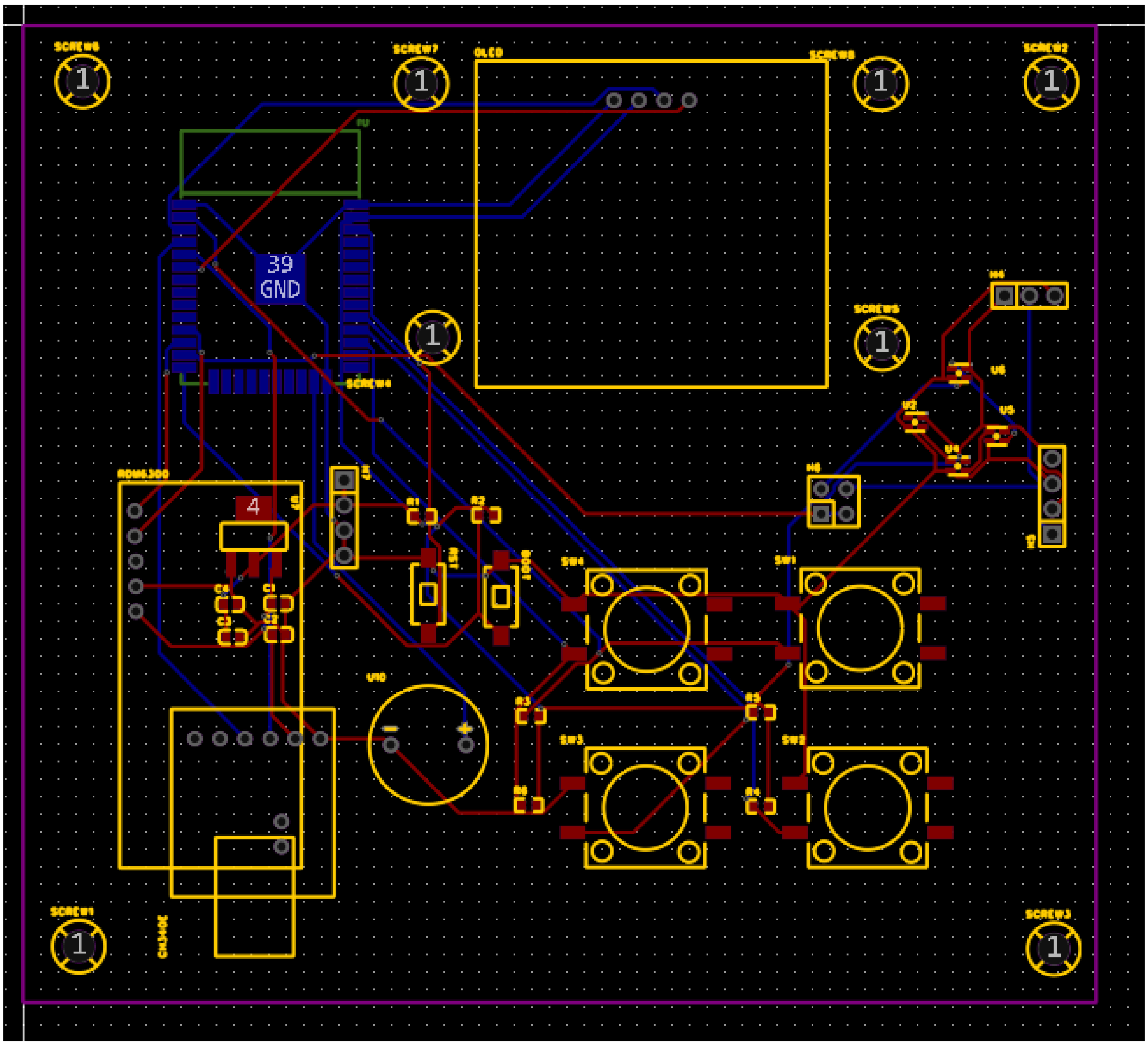


Figure 5: PCB Design

However, this PCB design has **not** yet been **fully completed** or **optimized**. **Further verification** and **modification** is required, including checks on trace spacing and component clearance. Additional components may need to be added or existing ones modified, if necessary, to optimize the design and ensure the success of subsequent PCB fabrication and assembly.

6 Conclusion

This report summarizes the **learning** and **exploration** process of **relevant technologies**, the **approach to problem** analysis, and **the design** of a complete IoT application. It begins with **a study of hardware** aspects, including component selection, hardware architecture, and the interconnection between components. The process of learning how to **draw schematics and design PCBs** is also presented, as these steps form the foundation for future circuit design, implementation, and hardware development for the project.

In addition to hardware exploration, the report covers the **study of software-related** technologies that are essential for building a complete IoT system. This includes learning the **C# programming language**, understanding **system-level software design** concepts, and studying various communication **protocols** commonly used in IoT applications. These software technologies provide a solid technical foundation for designing, implementing, and integrating the application layer in later stages of the project.

Overall, this report represents a comprehensive **preparation phase**, combining both hardware and software perspectives. The knowledge and skills gained through this learning process establish a strong foundation for the subsequent development, implementation, and optimization of the final project, ensuring that future work can be carried out in a structured, efficient, and technically sound manner.