Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

Modbus gateway

Students

David Braile, Lorenzo Ronzani

Supervisor

Mirko Gelsomini

Co-Supervisor

Customer

ESAM

Degree course

MSE Computer Science

Module

TSM_IoT

Year

2024

Date

June 4, 2024

Contents

List of Figures

Abstract

This project focuses on developing a Modbus gateway using the ESP32 [1] microcontroller, aimed at transforming non-IoT Modbus devices into IoT-enabled ones by making their data accessible over a network. The ESP32 is configured to function both as a Modbus master and slave, facilitating robust communication and data management. A web application, hosted on the ESP32, provides an intuitive interface for users to configure registers, perform CRUD (Create Read Update Delete) operations, and monitor historical data. Developed in HTML, CSS, and JavaScript, the webapp fits within the ESP32's memory constraints, ensuring efficient performance. The project successfully implements a seamless Wi-Fi setup through a captive portal, enabling easy network integration. Key achievements include reliable Modbus communication, effective data visualization, and user-friendly configuration. Future enhancements will focus on external data storage, migrating the web application to a public server for improved scalability and customization, and optimizing the system for better performance. This project not only bridges the gap between traditional and modern devices but also provides a flexible, scalable solution for industrial automation.

Keywords: Modbus gateway, ESP32, IoT, Web Interface

Chapter 1

Introduction

This project aims to develop a Modbus [1] gateway using the ESP32 [2] microcontroller. The primary goal is to collect data from various Modbus devices and make it **accessible** over a network, leveraging the powerful capabilities of the ESP32. In essence, this project transforms non-IoT devices into IoT-enabled ones using the ESP32, providing enhanced connectivity and functionality.

1.1 Project goals

The project goals can be categorized into three main areas:

- **WiFi setup**: The ESP32 needs to function as an access point and create a captive portal. This feature allows users to easily connect the device to a new network.
- **Web interface**: The ESP32 should serve a web page accessible via the new IP address. This web interface will include:
  - **Configuration page** where users can perform CRUD [5] (Create, Read, Update, Delete) operations on registers and other Modbus settings.
  - **Analytics page** displaying real-time values and a short history of the data.
- **Backend**: The ESP32 must operate as both a Modbus master and slave, ensuring proper communication with other devices. This functionality is crucial for data exchange and interoperability within the network.

1.2 Document overview

This document is organized into several key sections, each providing detailed information about different aspects of the project.

To begin with, the **Analysis section** (Chapter 2) contains comprehensive information about the current state of the technologies used in the project. Additionally, it includes an analysis of the development environment, providing the context and background necessary for understanding the project's foundation.

Moving on, in the **Implementation section** (Chapter 3), you will find a detailed explanation of how the project is structured to achieve its goals. This section delves into the technical aspects, outlining the architecture, components, and methodologies employed during development.

Next, the **User guide** (Chapter 4) offers a visual and descriptive guide on how to use the product. It provides step-by-step instructions to ensure users can effectively set up and operate the Modbus gateway.

In the **Results section** (Chapter 5), the outcomes of the project are presented, demonstrating how the goals were met.

Lastly, the **Conclusions section** (Chapter 6) provides a reflective analysis of the entire project, discussing achievements, challenges, and insights gained. It also outlines potential future developments and enhancements to further improve the product's features and capabilities.

Chapter 2

Analysis

Throughout this project, it is essential to study and understand various technologies that play a critical role in ensuring the project's success. A thorough comprehension of these technologies is necessary to grasp how the different components work together and to facilitate the development and implementation processes.

2.1 State of the art

This subsection provides relevant information about the technologies used in the project. It covers current advancements and best practices, offering a foundation for understanding the capabilities and limitations of the tools and methods employed.

2.1.1 ESP32

The ESP32, developed by Espressif Systems, is a versatile microcontroller with built-in Wi-Fi and Bluetooth capabilities, making it ideal for IoT applications. It features a dual-core processor, various peripherals (ADCs, DACs, UARTs, SPIs, I2Cs, PWMs), and operates efficiently with low power consumption. The ESP32's robust development ecosystem, including the ESP-IDF and Arduino IDE compatibility, makes it accessible for developers. Its applications range from smart home devices to industrial automation, showcasing its flexibility and performance in connected solutions.

2.1.2 ESAM E2002

The ESAM E2002 [3] is a DIN rail network analyzer designed for measuring electrical parameters in A.C. lines across various configurations (3-4 wires or ARON). It features galvanically isolated voltage inputs and supports current measurements from 0.05 to 5 A directly or through current transformers (CT). The device includes an RS485 ModBus RTU interface for remote monitoring and two digital outputs for alarms or pulse outputs. The E2002 can

also be connected to the E3000 [4] module for additional analog output capabilities, making it ideal for interfacing with PLCs and industrial PCs

2.1.3 Modbus protocol

Modbus, developed by Modicon in 1979, is a standard communication protocol for industrial automation. It enables efficient data exchange between devices, using a master-slave architecture to control and monitor equipment.

2.1.3.1 Modbus RTU

Modbus RTU (Remote Terminal Unit) is a binary data format used over serial lines like RS-232 or RS-485. It is efficient and straightforward, making it ideal for slower serial connections in industrial environments. While it is easy to implement and widely supported, its limited range and slower speed compared to Ethernet can be drawbacks.

2.1.3.2 Modbus TCP

Modbus TCP (Transmission Control Protocol) extends Modbus for Ethernet networks, encapsulating RTU messages within TCP/IP packets. This allows for greater distances, higher data transfer rates, and more complex network topologies. It also integrates well with modern IT and IoT infrastructures, though it is more complex to set up and manage, with higher associated costs.

2.1.3.3 Master-Slave Communication

In Modbus, the master-slave architecture is fundamental. The master device initiates all communication, sending requests to slave devices and processing their responses. This setup is simple and deterministic, ensuring predictable and reliable operation. However, the master device is a single point of failure, and the configuration can limit flexibility and scalability.

2.1.4 Modbus master library

The ModbusRTUMaster library is designed to facilitate communication between devices using the Modbus RTU protocol. It enables an ESP32 or other microcontroller to act as a Modbus master, allowing it to read from and write to Modbus slave devices. The library is straightforward to use, providing functions for reading and writing to holding registers, input registers, coils, and discrete inputs, thus simplifying the integration of Modbus RTU into various projects.

2.1.5 Modbus slave library

The ModbusRTUSlave library allows an Arduino device to function as a Modbus RTU slave. It supports various Modbus function codes, enabling the slave to respond to read and write requests from a Modbus master. This library is compatible with multiple Arduino boards and can work with both hardware and software serial ports, making it versatile for different applications.

2.2 Development environment

The development environment for this project comprises several key components:

- **ESP32**: The microcontroller responsible for Modbus communication and hosting the web server. The ESP32 integrates Wi-Fi and Bluetooth capabilities, making it a versatile choice for IoT applications.
- **ESAM E2002**: This is an electrical component that communicates with the ESP32 in master mode. It measures electrical parameters in A.C. lines and provides data via Modbus RTU protocol, which the ESP32 reads and processes.
- **ModScan software**: This software tool is used to test and monitor Modbus communication. It communicates with the ESP32 in slave mode, allowing for the verification and debugging of the Modbus RTU setup and data exchange.

Chapter 3

Implementation

This section details the implementation of the Modbus gateway project, covering the hardware setup, software configuration, and the integration process. It includes technical descriptions of how the ESP32, ESAM E2002, and ModScan software are configured and interconnected to achieve the project's goals.

3.0.1 General overview

The project consists of four main developed components, as illustrated in Fig. 3.1. These components work together to create a functional Modbus gateway using the ESP32.

- **Webapp**: This is the visual interface of the project, enabling users to perform CRUD operations on the registers, view register values, and see a short history of stored values. It also allows users to toggle the slave mode on the ESP32 on/off and adjust swap settings.
- **ESP32**: The core of the project, the ESP32 acts as a Modbus gateway capable of communicating in both slave and master modes. Additionally, it functions as a web server, facilitating the connection and information exchange with the webapp.
- **ESAM E2002**: This device is transformed into an IoT-enabled device through this project. It communicates electrical parameters to the ESP32 via Modbus RTU acting as a slave.
- **ModScan**: A Modbus master software tool that communicates with the Modbus slave (ESP32), used for testing and verifying the Modbus communication setup.

All communications in the system are bidirectional, with the ESP32 serving as the central hub of the project. Despite its small size and low cost, the ESP32 plays a critical role in integrating and managing the various components.

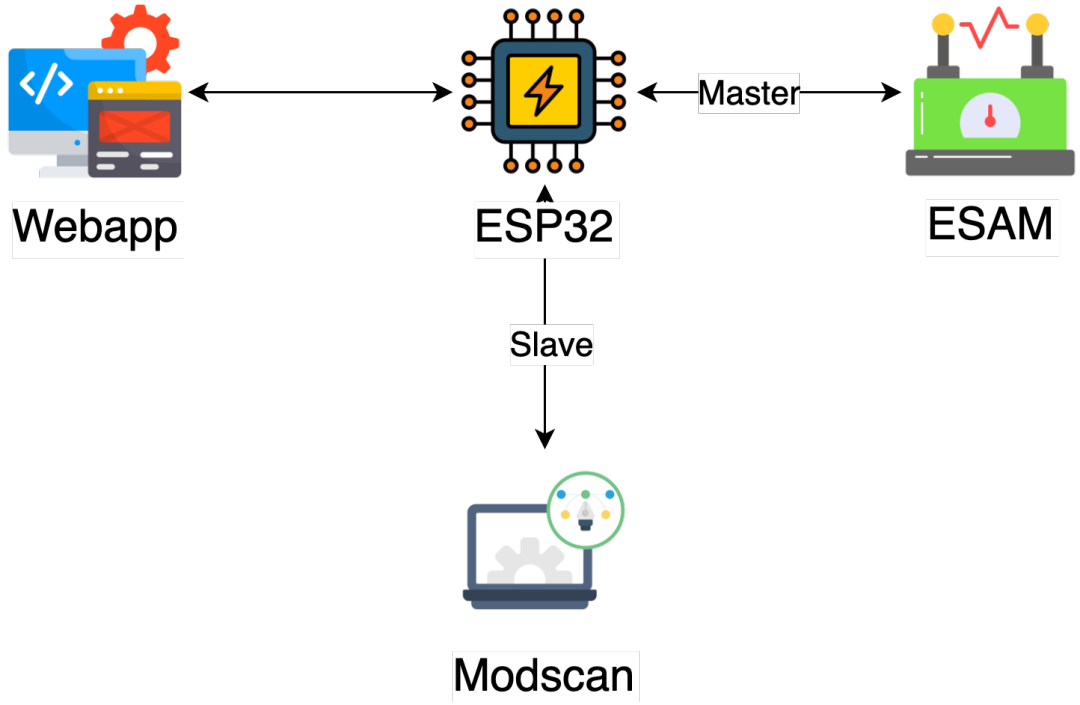

Figure 3.1: Components schema

3.0.2 ESP32

The ESP32 is a sophisticated component that orchestrates information flow between all actors in the system. It has a robust architecture that ensures stability, ease of customization, and upgradability. The main architectural components are as follows:

- **HoldingRegister**: This class represents a single modbus register.
- **ModbusRegister**: This class manages a list of HoldingRegisters, exposing CRUD operations and handling the list's size.
- **ModbusModule**: The core of modbus communication, this class operates as a master (to communicate with the ESAM E2002) or as a slave (to communicate with ModScan). It handles various types of values and integrates conversion for incoming and outgoing data.
- **NumberConverter**: A static class responsible for handling number conversions.
- **CustomModbusRTUSlave**: A fork of the ModbusRTUSlave library, this class adds the feature of passing a callback to the performOperation, allowing behavior to be injected from outside.
- **WebServerConfig**: This component manages the web server, handling both initialization and ongoing external interactions.

- **WebPages**: A class that stores HTML, CSS, and JavaScript files necessary for the web interface.

3.0.3 Wi-Fi management

Wi-Fi management is a crucial aspect of any IoT device, as it enables the device to **communicate** with the external world. In this project, the ESP32 creates an access point that allows users to set up a real Wi-Fi connection through a captive portal. This process involves configuring a web server and a DNS server correctly to achieve the **desired workflow**, these technologies reassure a correct interception and handling of DNS queries and client redirections. The access point facilitates the initial connection, and the **captive portal** guides the user through connecting the ESP32 to a Wi-Fi network, ensuring the device can function effectively within an IoT ecosystem.

3.0.4 Modbus management

Modbus management is handled by the **ModbusModule**, the core component of this project. This module allows the ESP32 to function both as a master and a slave, communicating via Modbus RTU protocol. By integrating other components, the ModbusModule can manage lists of **HoldingRegisters**, enabling the reading and retrieving of values efficiently. This dual functionality is crucial for interfacing with both ModScan and the ESAM E2002, facilitating seamless data exchange and control within the Modbus network.

3.0.5 Webserver

The Webserver is supported by the **WebServerConfig** class, created to configure and manage a web server that interfaces with Modbus functions. This class is the main actor for the creation of Wi-Fi access point, establishes routes for web server endpoints, and handles client requests for various operations. To achieve **seamless** communication and functionality, the WebServerConfig class integrates several essential components such as WiFi, WebServer, DNSServer, and WebPages.
The main part of this class is the routes handling:

- /: handles the root request and call the captive portal page.
- /**captivePortalScript.js**: handles the JavaScript of the portal page.
- /**scanNetworks**: scans for available Wi-Fi networks and returns the results in JSON.
- /**connect**: handles the connection to a certain network given the right credentials.
- /**configuration**: gives to the user the CSS and HTML of the configuration page.
- /**configurationScript.js**: handles the JavaScript for the configuration page.

- /**holdingRegisters and** /**holdingRegister**: return information about Modbus holding registers in JSON format.
- /**add, /edit, and** /**delete**: handle the addition, modification, and deletion of holding registers.
- /**slave and** /**swap**: handle the modification of the slave mode and swap mode settings.
- **default**: catches undefined requests and redirect them on the root request.

3.0.6 Webapp

The webapp serves as the **graphical interface** that allows users to interact with the ESP32. Hosted directly on the ESP32, this setup simplifies the project architecture by eliminating the need for external hosting services. The webapp facilitates **bidirectional communication** with the ESP32, functioning both as a configuration tool and an analytics platform. Developed in pure HTML, CSS, and JavaScript, the webapp is **lightweight and efficient**, ensuring it operates effectively within the limited memory constraints of the ESP32. Through this web interface, users can manage device settings, monitor real-time data, and access historical data, all within a streamlined and user-friendly environment.

Chapter 4

User guide

This section provides comprehensive instructions on how to set up, configure, and use the ESP32 Modbus gateway. It includes step-by-step procedures for connecting to the device, accessing the web interface, and managing modbus registers. This section is designed to help users get the most out of their ESP32 modbus gateway with clear and concise guidance.

4.1 Schematic

The first step in using your ESP32 Modbus gateway is to replicate the schematic shown in Fig. 4.1, which represents the main circuit. This setup includes the following components:

- **ESP32**: The central microcontroller responsible for Modbus communication and hosting the web server.
- **2x TTL to RS485 Converters**: hese converters facilitate communication between the ESP32 and Modbus devices by converting TTL signals to RS485 signals.
- **Modbus Device**: Any device that communicates using the Modbus protocol, which will be interfaced with the ESP32.
- **Modscan**: A Modbus master software tool used to communicate with Modbus slave devices, allowing you to monitor and control the Modbus network.

Carefully follow the schematic to connect these components correctly, ensuring that the wiring is precise to avoid any communication issues. Once the circuit is replicated, you can proceed with configuring the ESP32 and accessing the web interface for further setup and management.

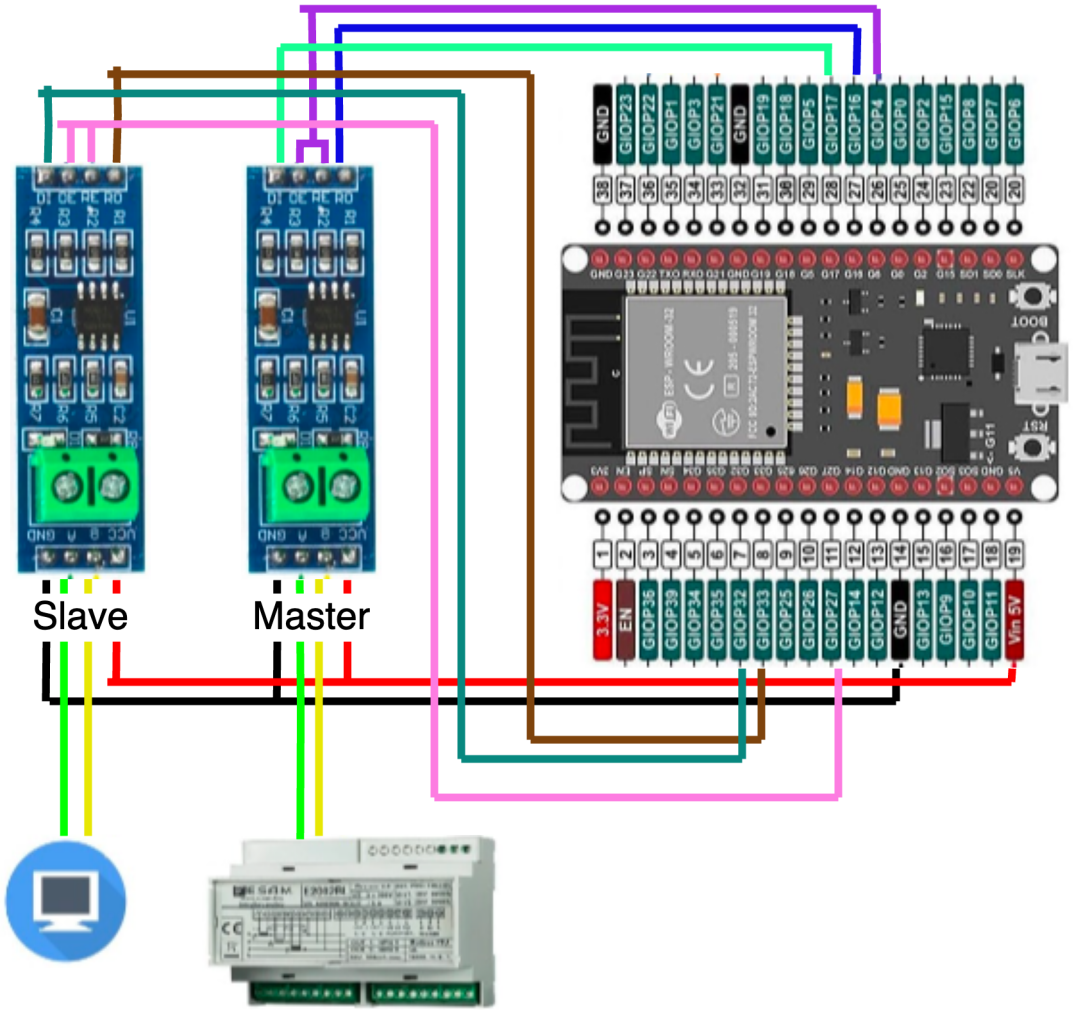

Figure 4.1: Schematic

4.2 Modbus gateway

Once you have correctly reproduced the schematic, you can proceed with powering on the ESP32, uploading the code, and following the subsequent steps to complete the setup. Here's a concise guide to help you get started:

1. **Power on the ESP32**:
   - Connect the ESP32 to a power source using a USB cable or another suitable power supply.
   - Ensure that all connections are secure and that the ESP32 is receiving power.
2. **Upload the Code**:
   - Connect the ESP32 to your computer using a USB cable.

- Open your preferred development environment (e.g., Arduino IDE, PlatformIO).
- Load the code for the Modbus gateway project into the development environment.
- Select the correct board and port settings for the ESP32 in the IDE.
- Upload the code to the ESP32 by clicking the upload button in the IDE.

4.3 Wi-Fi Setup

When the ESP32 is powered on and the code is uploaded, it will broadcast an endpoint called ‘CONFIGURE ME’. Follow these steps to connect and configure the ESP32:

1. **Connect to the ‘CONFIGURE ME’ endpoint**:
   - Use your computer or mobile device to search for available Wi-Fi networks.
   - Connect to the network named “CONFIGURE ME.”
2. **Access the captive portal**:
   - Once connected, a captive portal will automatically open (Fig. 3).
   - If the portal does not open automatically, open a web browser and enter any URL to be redirected to the captive portal page.
3. **Set up Wi-Fi connection**:
   - On the captive portal page, enter the credentials for the Wi-Fi network you want the ESP32 to connect to.
   - Submit the information to configure the ESP32 with the new Wi-Fi network.
4. **ESP32 connects to new network**:
   - The ESP32 will restart and attempt to connect to the provided Wi-Fi network.
   - Upon successful connection, the ESP32 will obtain a new IP address.
   - The Wi-Fi needs to have an available internet connection because the webapp download an external JavaScript file.
5. **Retrieve the new IP address**:
   - The new IP address will be displayed on the captive portal page after successful connection.
   - Note down this IP address as it will be used to access the web interface for further configuration and monitoring.

By following these steps, the ESP32 will be integrated into your Wi-Fi network, and you can begin using the web interface to manage Modbus settings and monitor data.

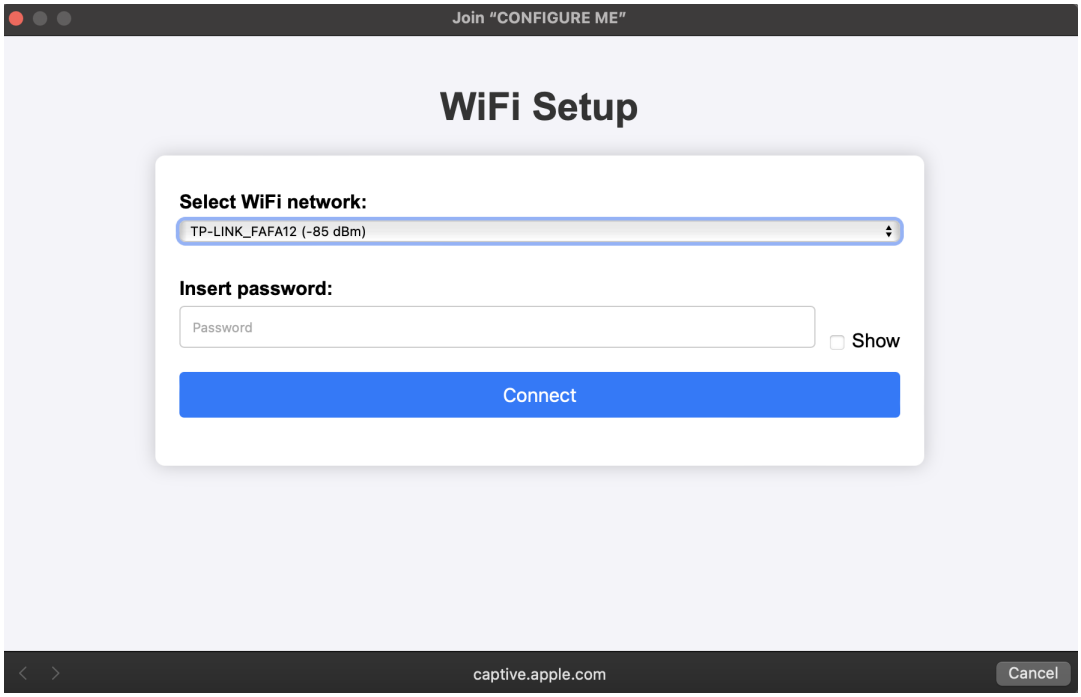

Figure 4.2: Captive portal

4.4 Configuration page

Now you are able to navigate to the endpoint **IP/configuration** where you will find a page similar to Fig. 4.3. On this page, you can view the registers and perform actions such as create, edit, and delete using the form shown in Fig. 4.4. Additionally, you can configure the Modbus module and view the values history as illustrated in Fig. 4.5.

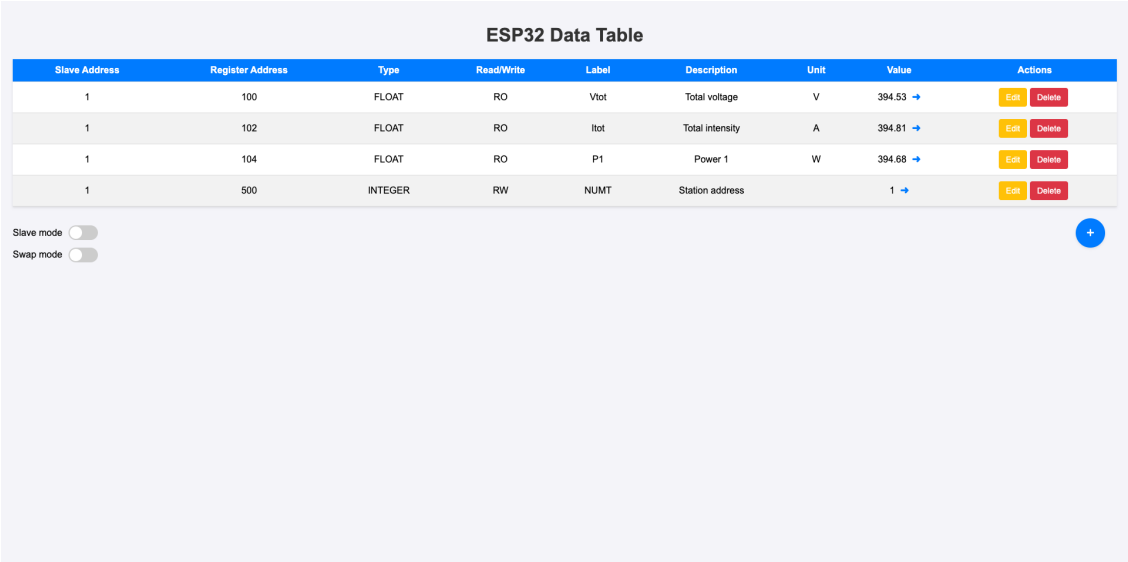

Figure 4.3: Configuration page

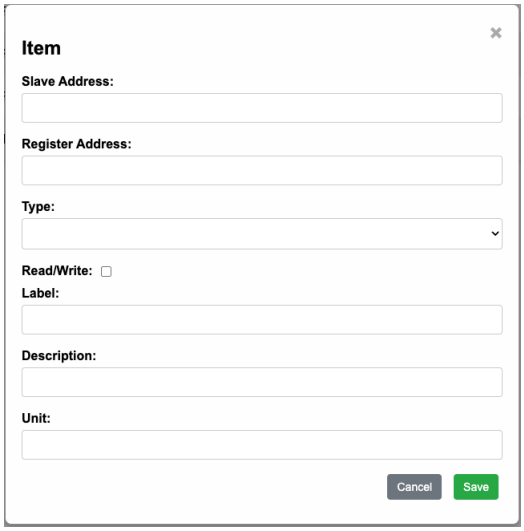

Figure 4.4: Item form

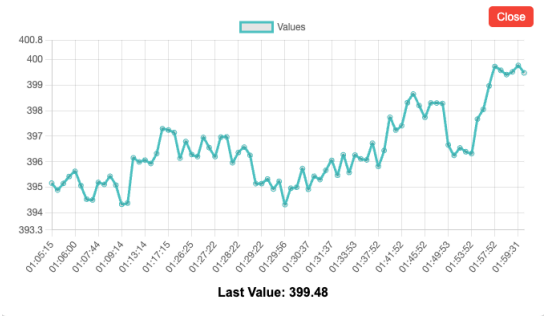

Figure 4.5: History graph

The values are updated every 15 seconds to avoid overloading the ESP32.

1. **Accessing the configuration page**:
   - Open a web browser and enter the IP address obtained during the Wi-Fi setup process, followed by `/configuration` (e.g., `192.168.1.10/configuration`).
   - The configuration page (Fig. 4.3) will be displayed.
2. **Managing registers**:
   - On the configuration page, you can view a list of Modbus registers.
   - Use the provided form (Fig. 4.4) to create, edit, or delete registers.
   - Enter the necessary details and submit the form to perform the desired action.
3. **Setting ModbusModule configuration**:
   - Adjust the settings of the Modbus module as required.
   - Ensure the configuration aligns with your specific Modbus communication needs.
4. **Viewing values history**:
   - The configuration page also includes a section to view the historical values of the registers (Fig. 4.5).
   - Values are updated every 15 seconds to provide real-time data without overloading the ESP32.

By following these steps, you can effectively manage and monitor your Modbus devices using the ESP32 web interface.

Chapter 5

Results

We successfully developed an ESP32 program that functions as a Modbus gateway. The system is capable of **communicating** through the Modbus protocol, acting both as a master and a slave. It can read and store register values and serve endpoints along with a web application for **graphical interaction**.
The implemented workflow allows users to complete a Wi-Fi setup, starting from an exposed endpoint and obtaining an IP address for further configuration. Once connected, users can navigate to the endpoint to access the web application. This application provides tools to configure registers using CRUD operations, view **historical data** from the registers, and manage internal settings.
Additionally, the system enables users to operate at the Modbus gateway level, allowing for slave mode activation and data swapping. The core Modbus module, which is **central** to the project, facilitates communication in both **master** and **slave** modes with any device implementing the Modbus protocol.
Through these functionalities, the ESP32 program efficiently bridges non-IoT devices to the network, providing robust Modbus communication capabilities and **user-friendly** configuration and monitoring tools.

Chapter 6

Conclusions

During the project, we achieved significant success in our goal of developing a Modbus gateway on an ESP32. One of the most noteworthy aspects is the ability to connect any Modbus device to the internet using a **low-cost board** that acts as an intermediary. This effectively makes non-IoT devices **accessible** via network connections.

Throughout the development process, we encountered numerous **challenges**, particularly in understanding the Modbus protocol, which was quite different from our previous studies. Despite these difficulties, we managed to start from scratch and, within a few months, created a functioning solution in a field outside our comfort zone.

The integration of hardware and software communication added another **layer of complexity**, but overcoming these hurdles was immensely rewarding. The project has been a great learning experience, expanding our knowledge and skills significantly. The final result is a robust Modbus gateway that effectively **bridges** non-IoT devices to modern networks, demonstrating our ability to tackle and solve complex problems.

6.1 Future developments

Looking ahead, there are several key areas for future development:

- **External data storage**: Typically, IoT devices store data on external web servers, enabling access from various locations rather than just within the same network. Implementing this would enhance the accessibility and usability of our system.

- **Web application hosting**: If data is stored externally, hosting the web application on a publicly accessible web server becomes a logical step. This approach would allow for extensive customization of the frontend without being constrained by the ESP32's limited memory. While maintaining some basic functions on the ESP32 could be beneficial, a publicly accessible web app would offer a better user experience.

- **Improved scalability**: Removing the web server role from the ESP32 would significantly improve scalability. Without the limitations of the ESP32's processing power, the system could handle more devices and more complex operations, making it more robust and efficient.

By focusing on these areas, we can significantly enhance the capabilities and performance of the Modbus gateway, ensuring it meets the demands of modern IoT applications.

6.2 Thanksgivings

We want to extend our gratitude to ESAM for providing us with the opportunity to work on a real-world project and for supplying the necessary project materials. Additionally, we would like to thank Prof. Mirko Gelsomini for his inestimable support throughout the project and for bridging the gap between academic learning and real-world applications.

Bibliography

[1] Espressif, https://www.espressif.com/en. Last accessed 04 Jun 2024

[2] Modbus Organisation, https://modbus.org/. Last accessed 04 Jun 2024

[3] Esam device E2002, https://esam.biz/prodotto/e2002/#dati-generali. Last accessed 04 Jun 2024

[4] Esam device E3000, https://esam.biz/prodotto/e3000/. Last accessed 2024 Jun 2024

[5] CRUD operations, https://www.codecademy.com/article/what-is-crud. Last accessed