Protocol Management Software

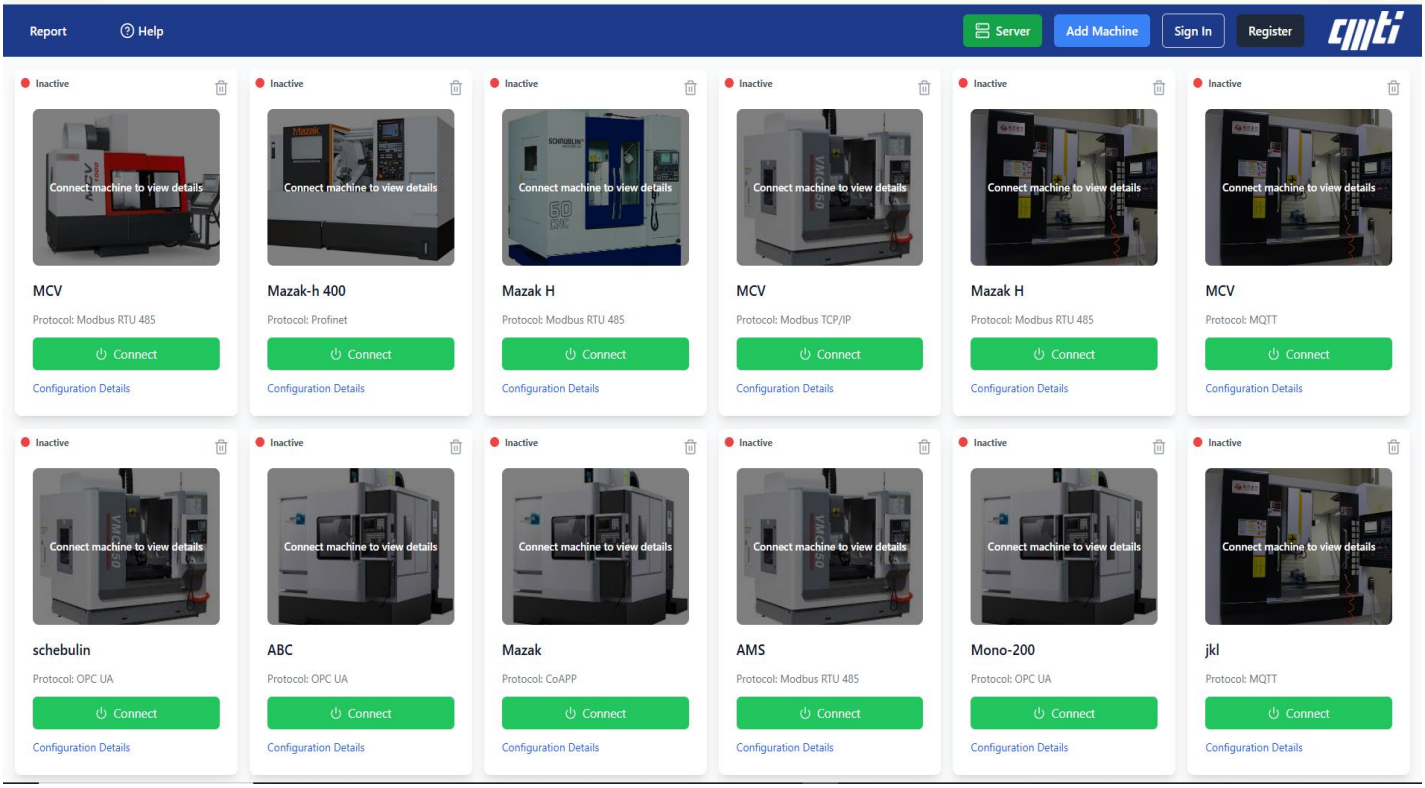
Problem Statement:

This project aims to retrieve and process data from industrial and IoT communication protocols, including Modbus, OPC UA, PROFINET, PROFIBUS, EtherCAT, CAN open, BACnet, HTTP, and MQTT. The objective is to establish seamless connectivity, enable real-time data exchange, and integrate data from various industrial automation and IoT systems for monitoring, analysis, and decision-making. The solution should ensure compatibility, scalability, and secure data handling across diverse protocols and devices.

Main Dashboard:

1. Machine Connectivity Management

- The system displays a list of industrial machines along with their respective communication protocols (e.g., Modbus RTU 485, Modbus TCP/IP, ProfiNet, OPC UA, MQTT).
- Users can connect machines to retrieve real-time data and monitor their status.



2. Machine Status Indication

- Each machine card indicates whether the machine is **Active** or **Inactive**, allowing users to identify the connectivity status at a glance.
- Inactive machines display a red status indicator and a prompt to connect the machine for details.

3. User Actions and Controls

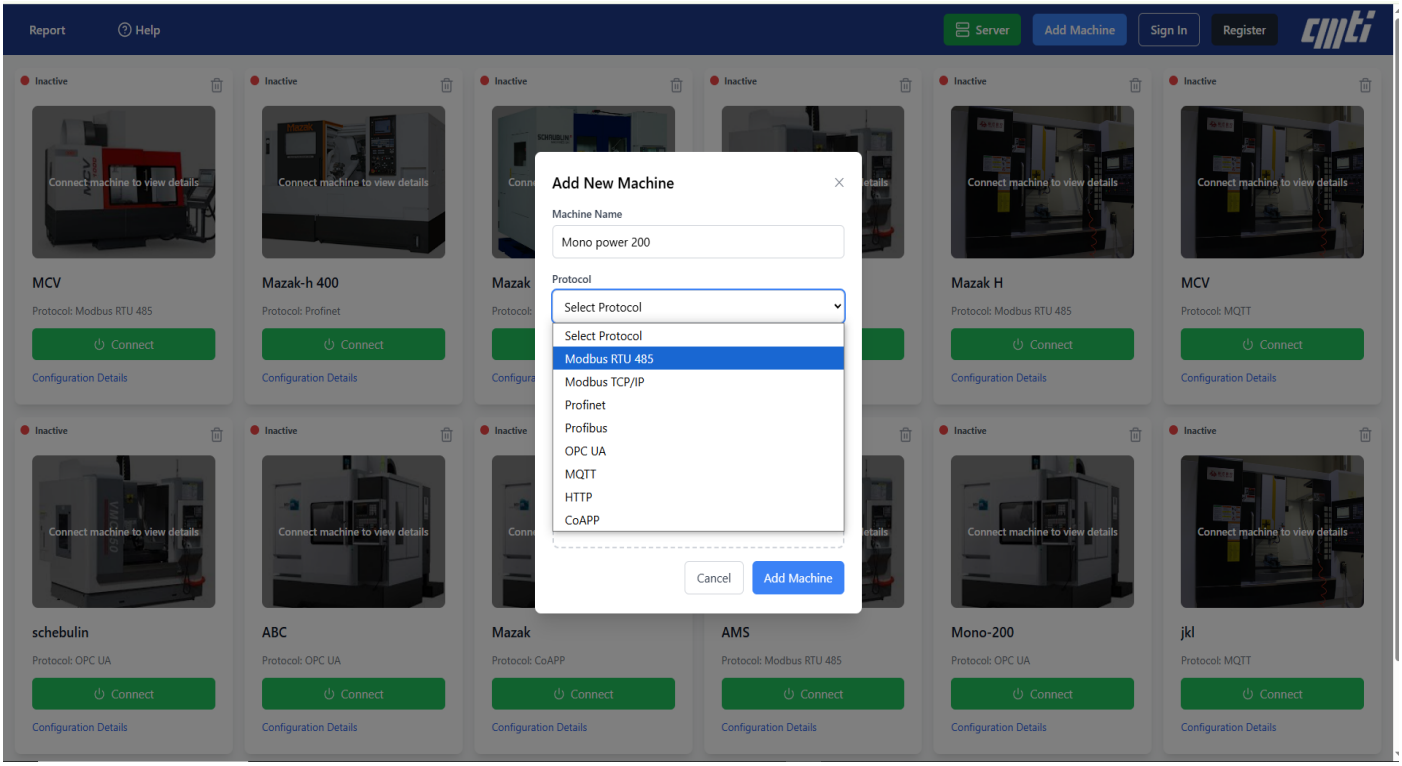
- Users can interact with each machine through the **"Connect"** button to establish a connection.
- Additional actions such as **viewing configuration details** and **deleting a machine** are available for better management.

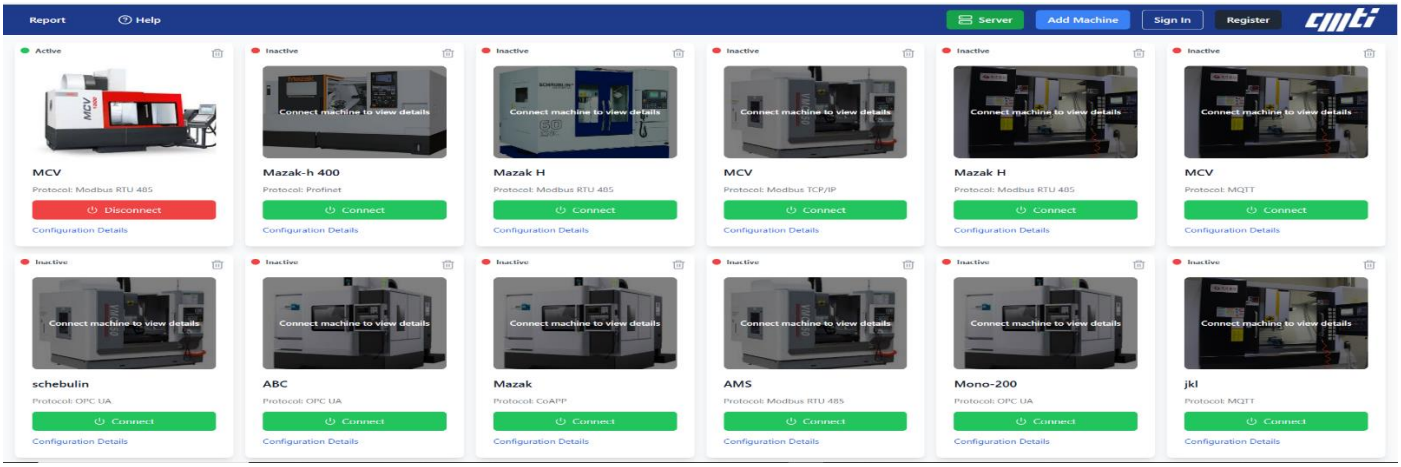
4. User Authentication & Management

- The interface provides **Sign In**, **Register**, and **Add Machine** options for user authentication and machine management.
- Users need to log in to access full functionality, ensuring controlled access and data security.

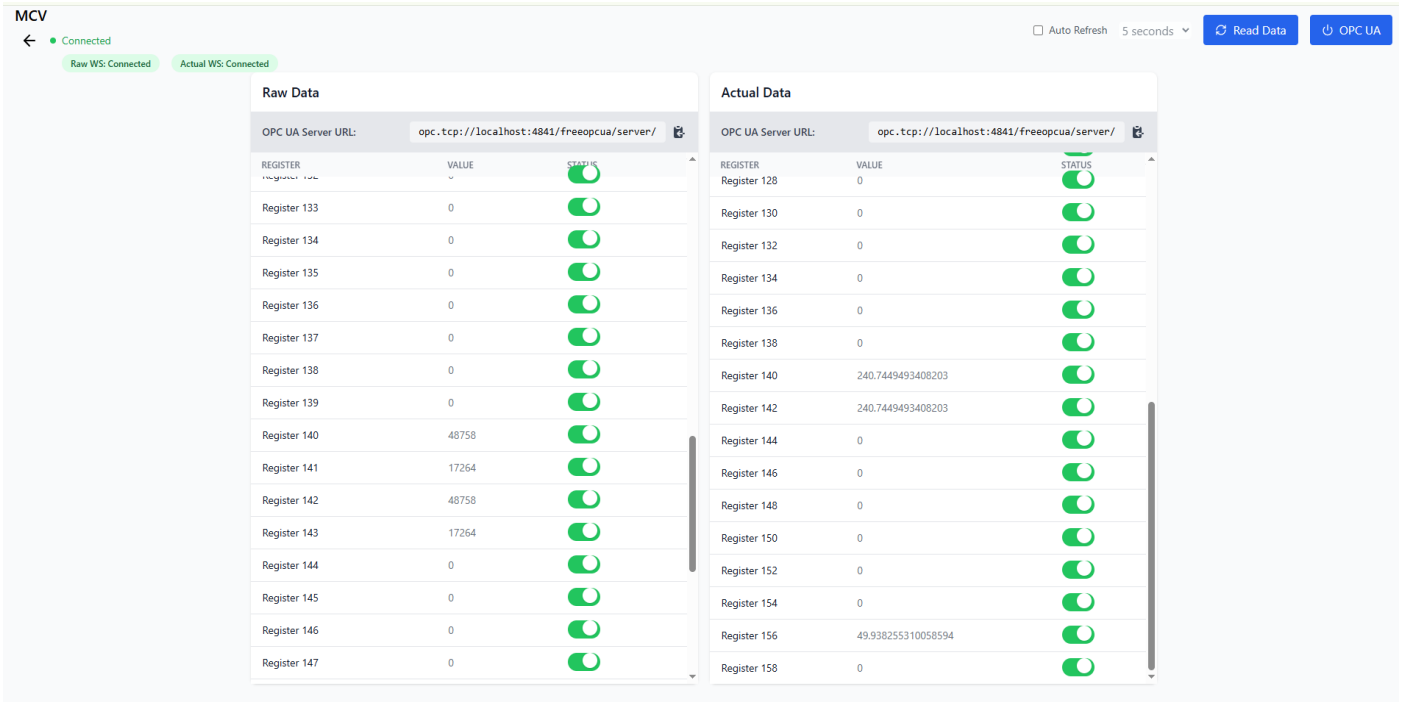
Industrial Protocols:

1. Modbus RTU -Rs 485:

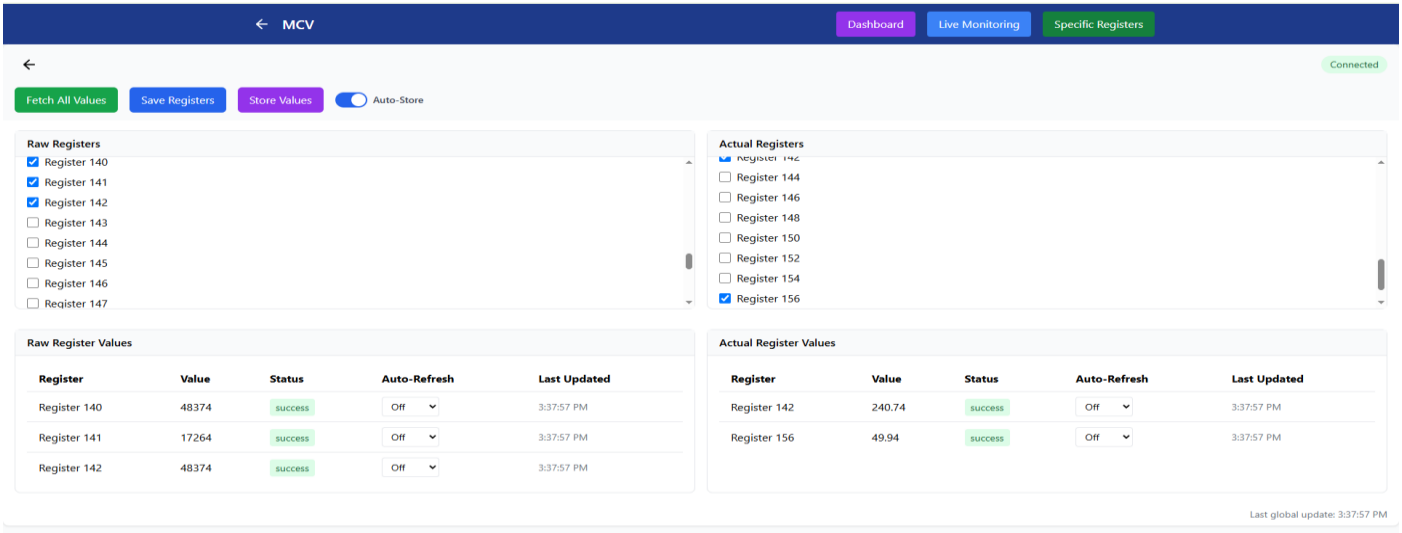




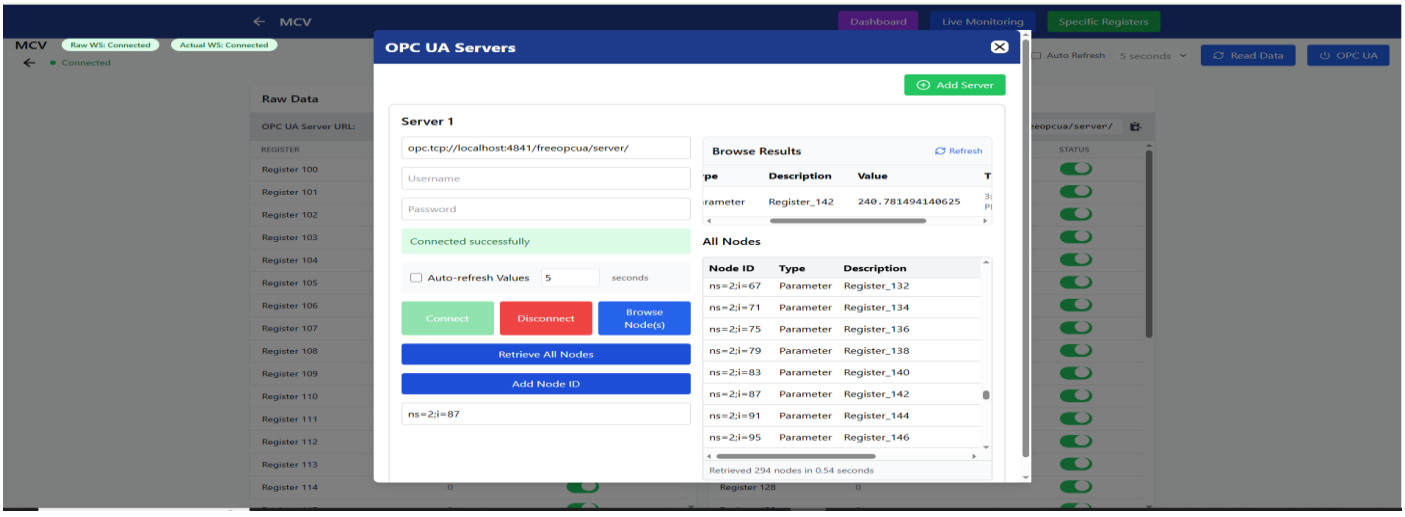
Live Monitoring



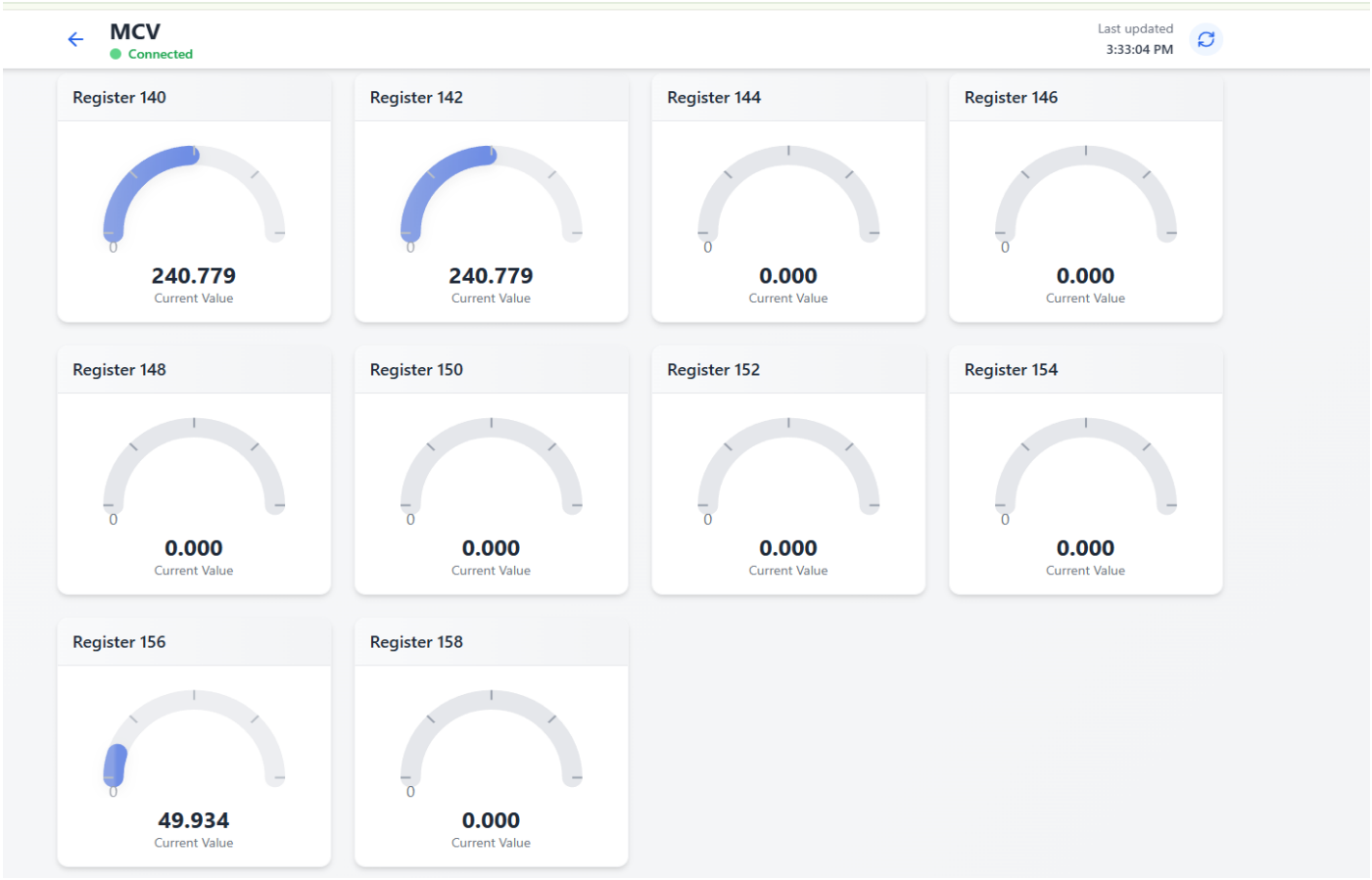
Specific Registers



OPC UA Quick Client



Dashboard of Registers data



Overview

Modbus RTU (Remote Terminal Unit) is a serial communication protocol used for industrial automation systems, allowing communication between devices connected via an **RS-485** network. It is widely used for connecting PLCs, sensors, and controllers to SCADA and monitoring systems.

➢ Modbus RTU Communication Basics

- **Protocol Type:** Request-Response (Master-Slave)
- **Physical Layer:** RS-485
- **Data Transmission:** Half-Duplex
- **Error Checking:** CRC-16 (Cyclic Redundancy Check)
- **Message Structure:** Address, Function Code, Data, CRC

- **RS-485 Communication Parameters**

When configuring **Modbus RTU over RS-485**, the following communication parameters need to be set:

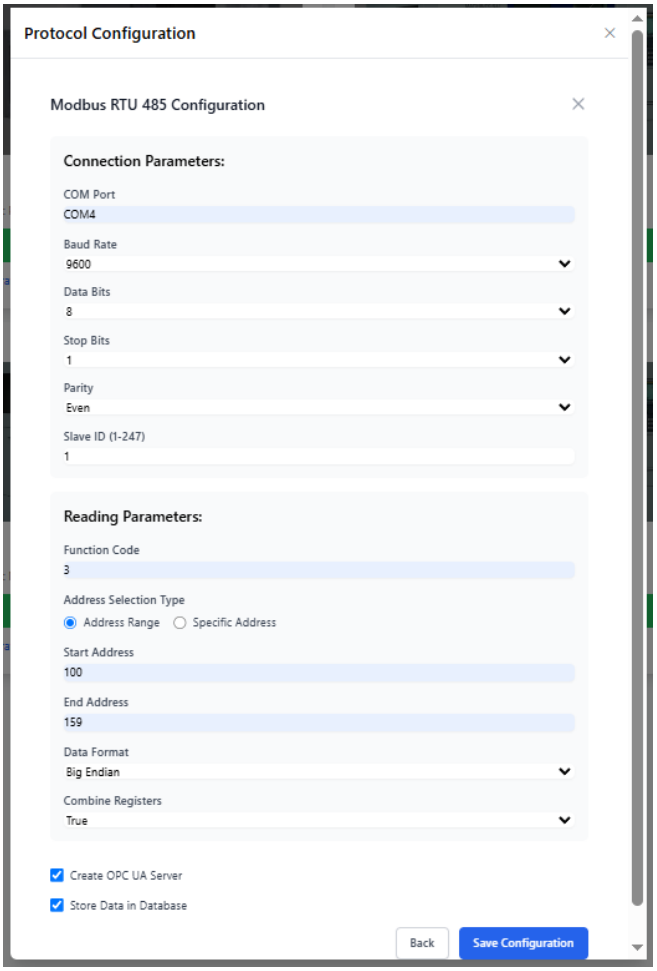
Connection Parameters

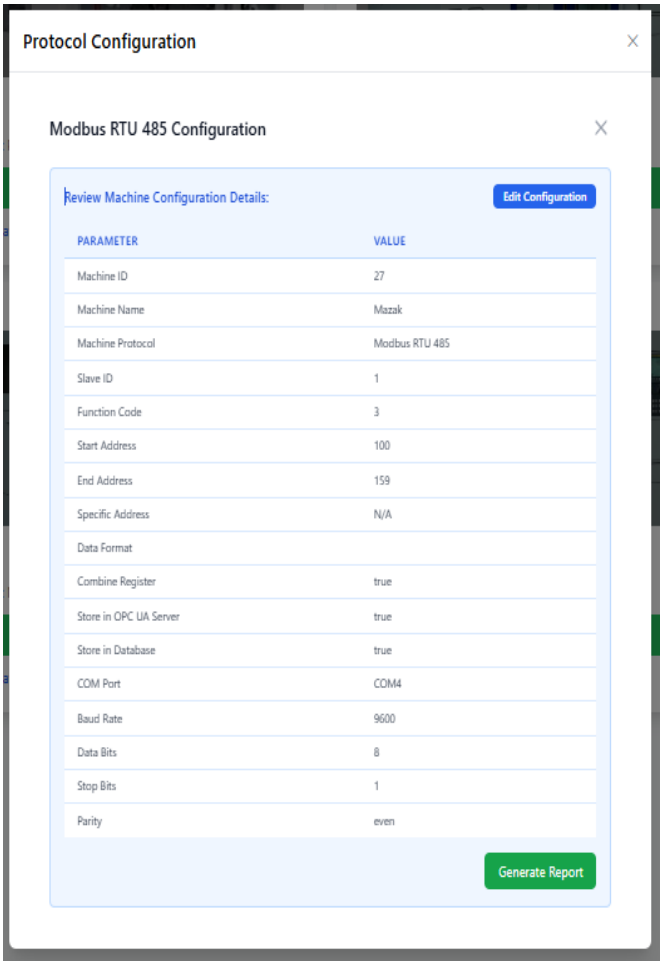
Parameters	**Description**
COM Port	Serial port used for communication (e.g., COM4)
Baud Rate	Speed of data transmission (e.g., 9600, 19200, 115200 bps)
Data Bits	Number of bits in each data byte (typically 8)
Stop Bits	Defines the end of a character (1 or 2)
Parity	Error-checking mechanism (None, Even, Odd)
Slave ID	Address of the device (1-247)

Reading Parameters

Parameters	**Description**
Function Code	Defines the type of request (e.g., 3 for Read Holding Registers)
Address Selection Type	Defines whether to read a range or a specific address
Start Address	First register to read (e.g., 100)
End Address	Last register to read (e.g., 159)
Data Format	Big Endian / Little Endian
Combine Registers	Whether to combine multiple registers for 32-bit or floating-point data

Configuration Details for Modbus RTU-RS 485





➢ Data Handling & Storage

- **Create OPC UA Server**: Enables seamless integration with SCADA and IoT platforms.
- **Store Data in Database**: Logs real-time machine parameters for historical analysis.

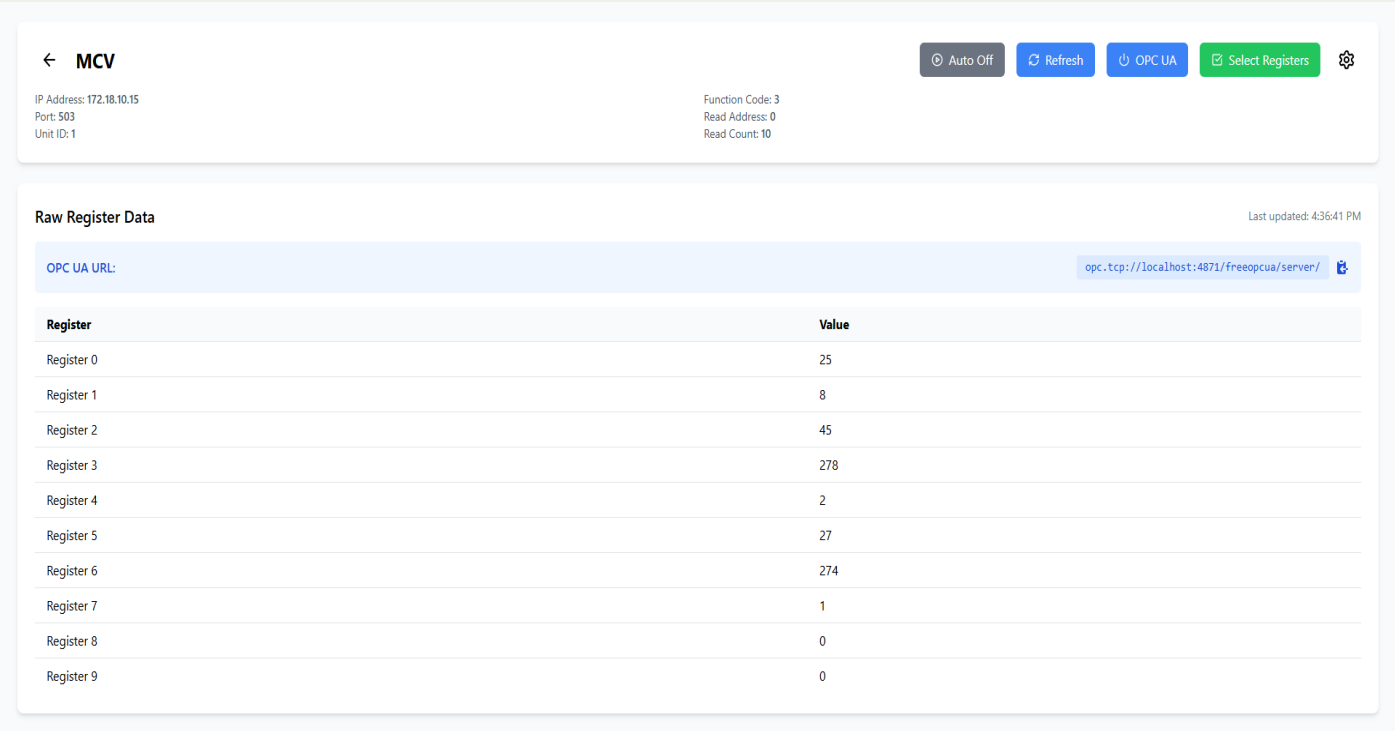
➢ Stat Card Generation (Based on Marked Section)

- The **"Generate Report"** button is used to create a **Stat Card**, summarizing key configuration parameters such as:

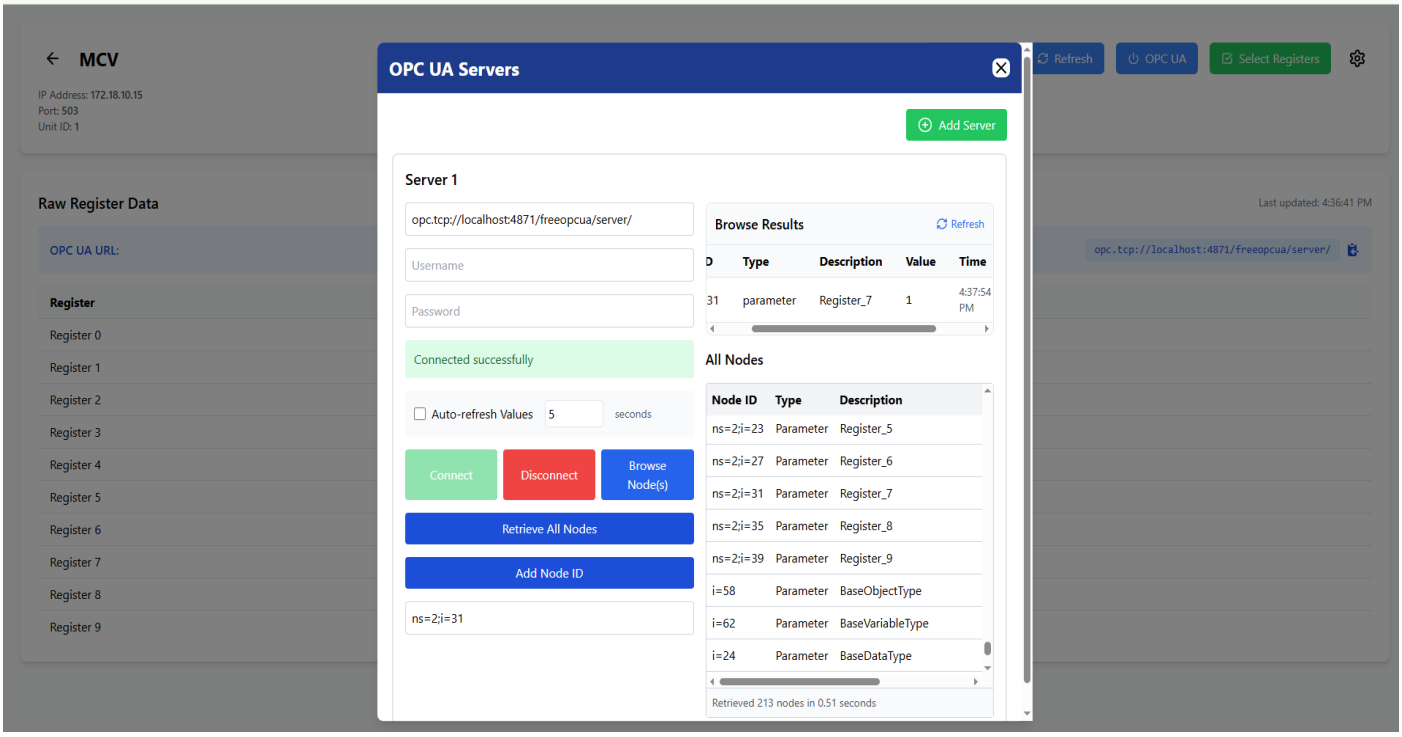
- Machine ID
- Machine Name
- Communication Settings
- Modbus Addressing
- Data Storage Preferences

This stat card provides a quick overview of the device configuration and can be used for diagnostics and validation.

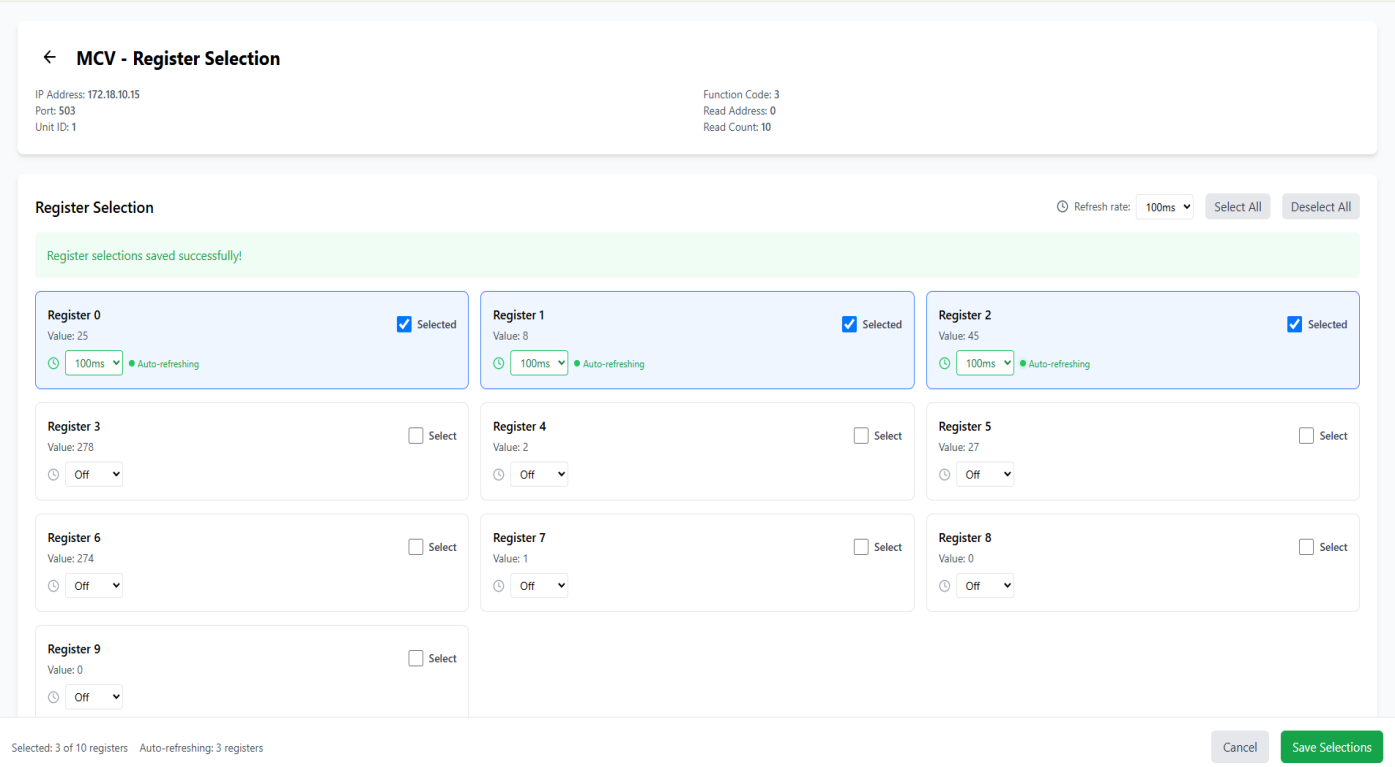
2. Modbus TCP/IP:



OPC UA Quick Client



Specific Registers Data



Overview

The interface allows users to add and manage manufacturing machines, that communicates via Modbus TCP/IP protocol. Modbus TCP/IP is a wireless communication protocol used for industrial automation systems.

➢ **Modbus TCP/IP communication Basics:**

- **Protocol Type:** Request-Response (Client-Server)
- **Physical Layer:** Ethernet
- **Data Transmission:** Full-Duplex
- **Error Checking:** TCP/IP checksum
- **Message Structure:** MBAP Header (7 bytes), Function Code, Data

➢ **TCP/IP Communication Parameters**

When configuring Modbus TCP/IP, the following communication parameters need to be set:

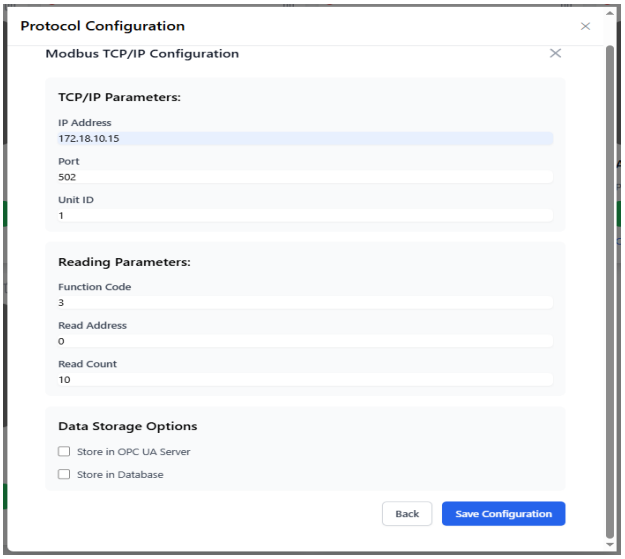
Connection Parameters

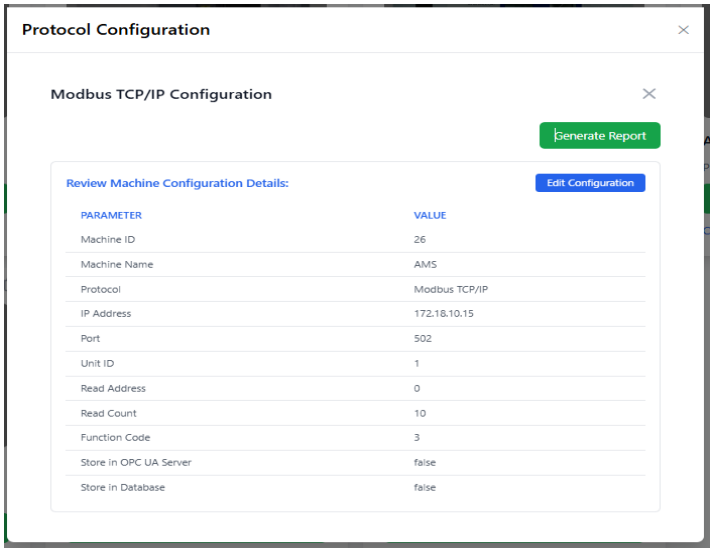
Parameters	Description
IP Address	Network address of the device (e.g., 172.18.10.15)
Port	Communication port (standard is 502)
Unit ID	Address of the device on internal sub-networks (1-247)

Reading Parameters

Parameters	Description
Function Code	Defines the type of request (e.g., 3 for Read Holding Registers)
Read Address	First register to read (e.g., 0)
Read Count	Number of consecutive registers to read (e.g., 10)
Data Format	Big Endian / Little Endian (implicit)

Configuration Details for Modbus TCP/IP





➢ **Data Handling & Storage**

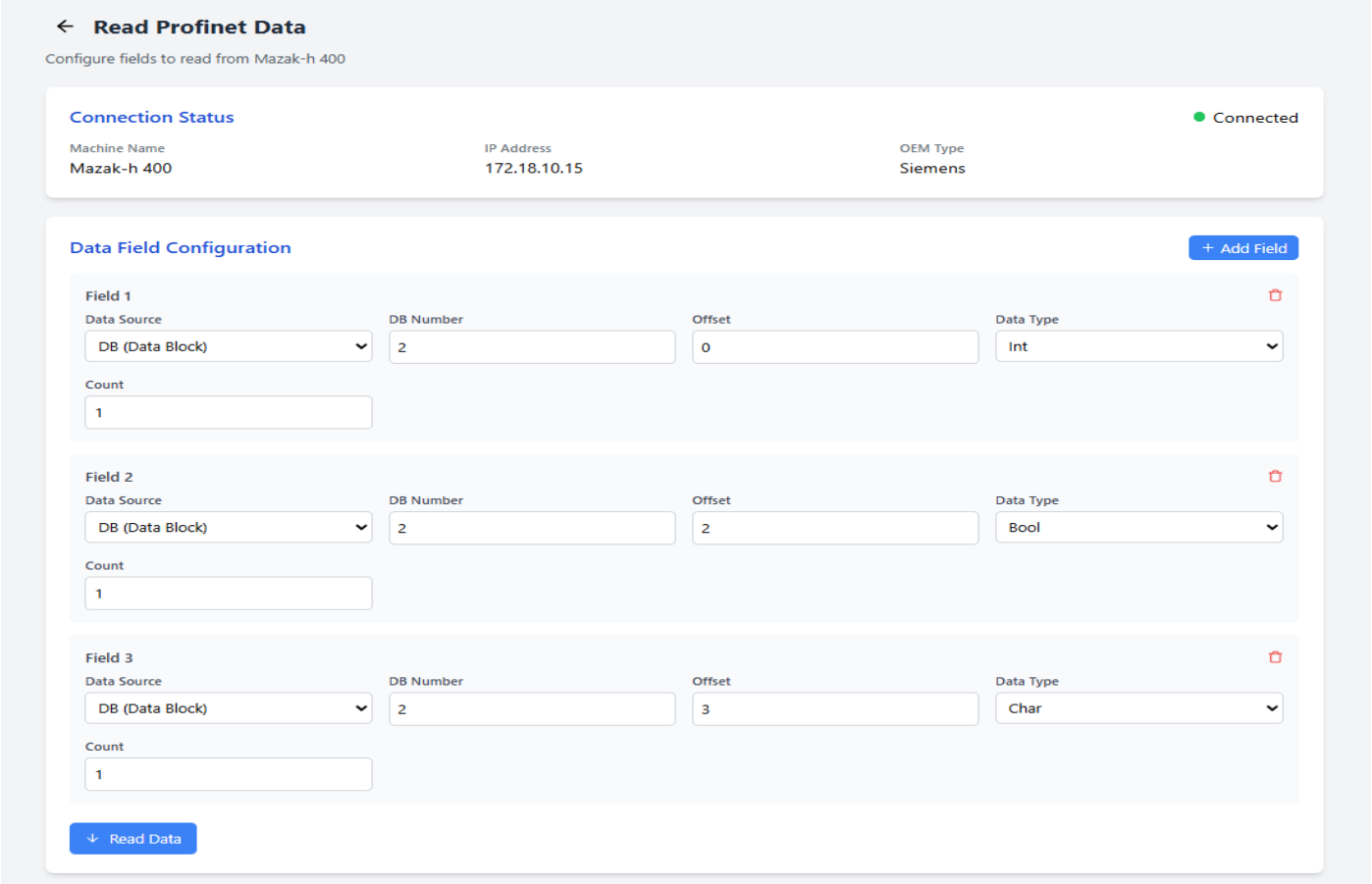
- **Create OPC UA Server**: Enables seamless integration with SCADA and IoT platforms.
- **Store Data in Database**: Logs real-time machine parameters for historical analysis.

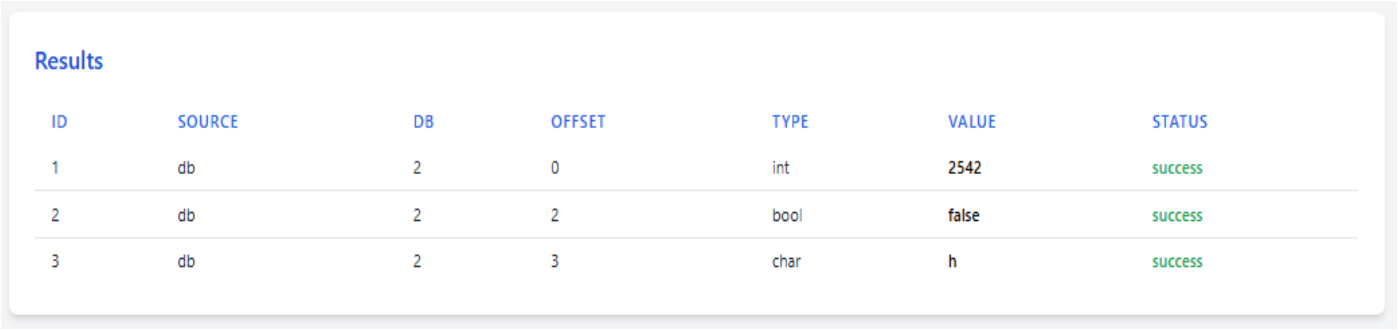
➢ **Stat Card Generation (Based on Marked Section)**

- The **"Generate Report"** button is used to create a **Stat Card**, summarizing key configuration parameters such as:

- Machine ID
- Machine Name
- Communication Settings
- Modbus Addressing
- Data Storage Preferences

3. ProfiNet:





Overview

PROFINET is an industrial Ethernet standard for automation that enables real-time communication between controllers and devices in industrial environments. It supports high-speed data exchange and is widely used for connecting PLCs, HMIs, and field devices from various manufacturers, particularly in Siemen's environments.

➢ PROFINET Communication Basics

- **Protocol Type:** Industrial Ethernet (IEC 61158)
- **Physical Layer:** Standard Ethernet
- **Data Transmission:** Full-Duplex
- **Error Detection:** Standard Ethernet CRC

- **Communication Types:** TCP/IP, RT (Real-Time), IRT (Isochronous Real-Time)

➢ **ProfiNet Communication Parameters**

When configuring **PROFINET**, the following communication parameters need to be set:

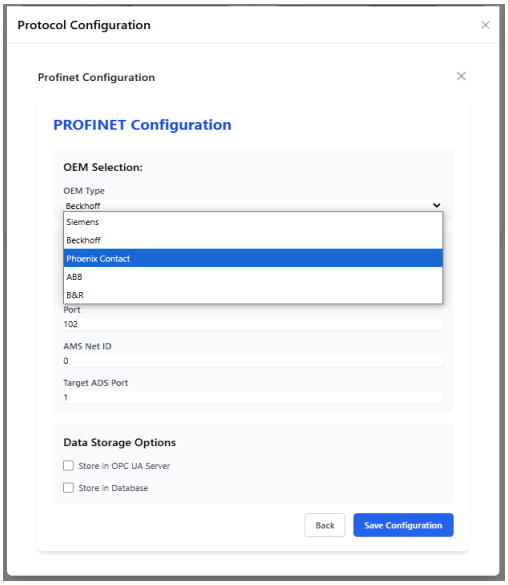
Connection Parameters for Siemen's

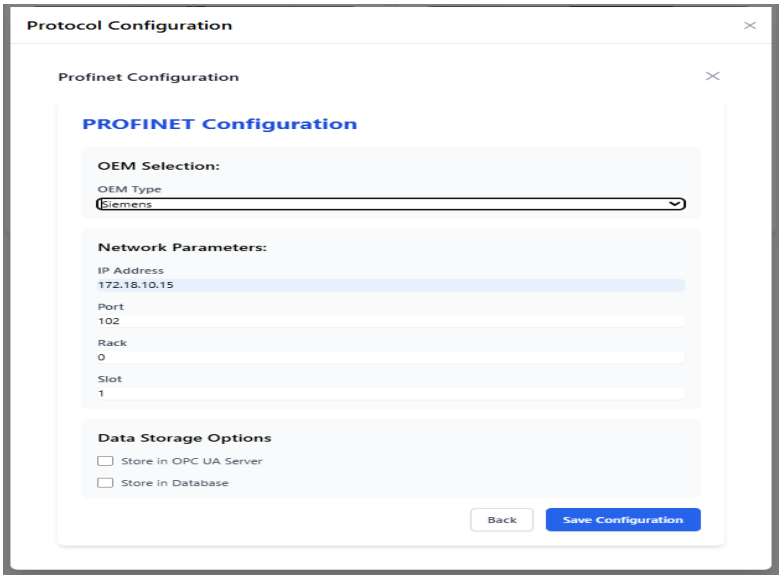
Parameters	**Description**
OEM Type	Device manufacturer (e.g., Siemens, Simatic Comfort, ABB)
IP Address	Network address of the device (e.g., 172.18.10.15)
Port	Communication port (e.g., 102)
Rack	Physical rack number in the hardware configuration (e.g., 0)
Slot	Slot number of the module in the rack (e.g., 1)

Reading Parameters

Parameters	**Description**
Data Source	Type of data to access (DB, Input, Output, Merker)
DB Number	Number of the data block to read (e.g., 2)
Offset	Address offset within the data block (e.g., 10)
Data Type	Format of the data (String, Bool, Integer, etc.)

Configuration Details for ProfiNet





➢ **Data Handling & Storage**

- **Create OPC UA Server**: Enables seamless integration with SCADA and IoT platforms.
- **Store Data in Database**: Logs real-time machine parameters for historical analysis.

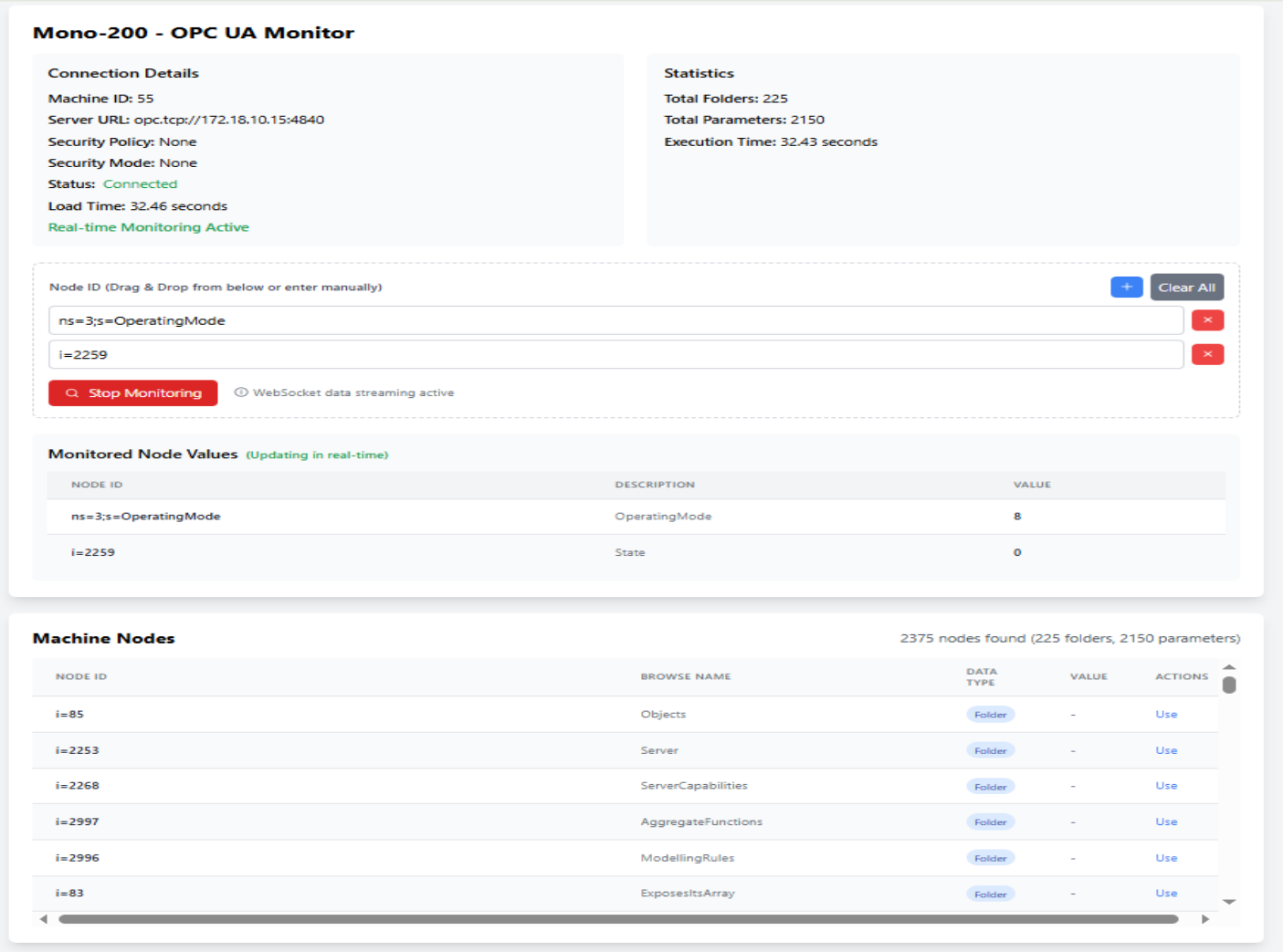
➢ **Stat Card Generation (Based on Marked Section)**

- The **"Generate Report"** button is used to create a **Stat Card**, summarizing key configuration parameters such as:

- Machine ID
- Machine Name
- Communication Settings
- Modbus Addressing
- Data Storage Preferences

This stat card provides a quick overview of the device configuration and can be used for diagnostics and validation.

4. OPC UA:



Overview

OPC UA (OPC Unified Architecture) is a platform-independent, service-oriented architecture that integrates all the functionality of individual OPC Classic specifications into one extensible framework. It provides a secure and reliable way to transfer data between industrial devices, control systems, and enterprise applications.

➢ OPC UA Communication Basics

- **Protocol Type:** Client-Server and Publish-Subscribe
- **Physical Layer:** Ethernet/IP-based
- **Data Transmission:** Full-Duplex
- **Error Handling:** Built-in error recovery mechanisms
- **Security:** Integrated authentication, encryption, and data integrity

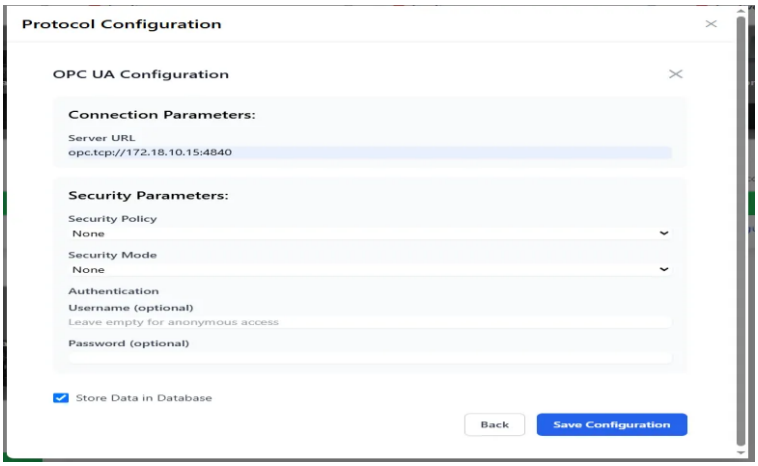
➢ OPC UA Communication Parameters

When configuring OPC UA, the following communication parameters need to be set:

Connection Parameters

Parameters	Description
Server URL	Endpoint URI of the OPC UA server (e.g., opc.tcp://172.18.10.15:4840)
Security Policy	Encryption policy (None, Basic128Rsa15, Basic256, etc.)
Security Mode	Security level (None, Sign, Sign and Encrypt)
Username	User credential for authentication (optional)
Password	Password for authentication (optional)

Configuration Details for OPCUA



- **Data Handling & Storage**

- **Store Data in Database**: Logs real-time machine parameters for historical analysis.

- **Stat Card Generation (Based on Marked Section)**
- The **"Generate Report"** button is used to create a **Stat Card**, summarizing key configuration parameters such as:

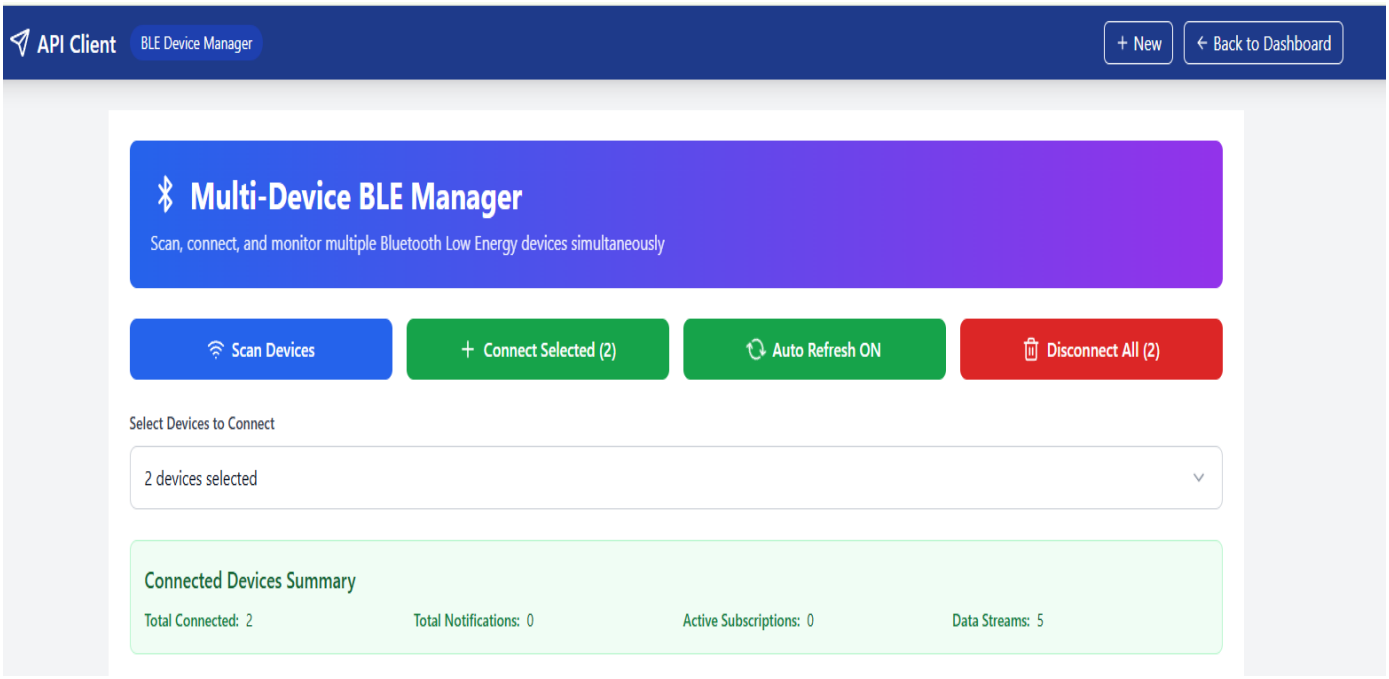
- Machine ID
- Machine Name
- Communication Settings
- Modbus Addressing
- Data Storage Preferences

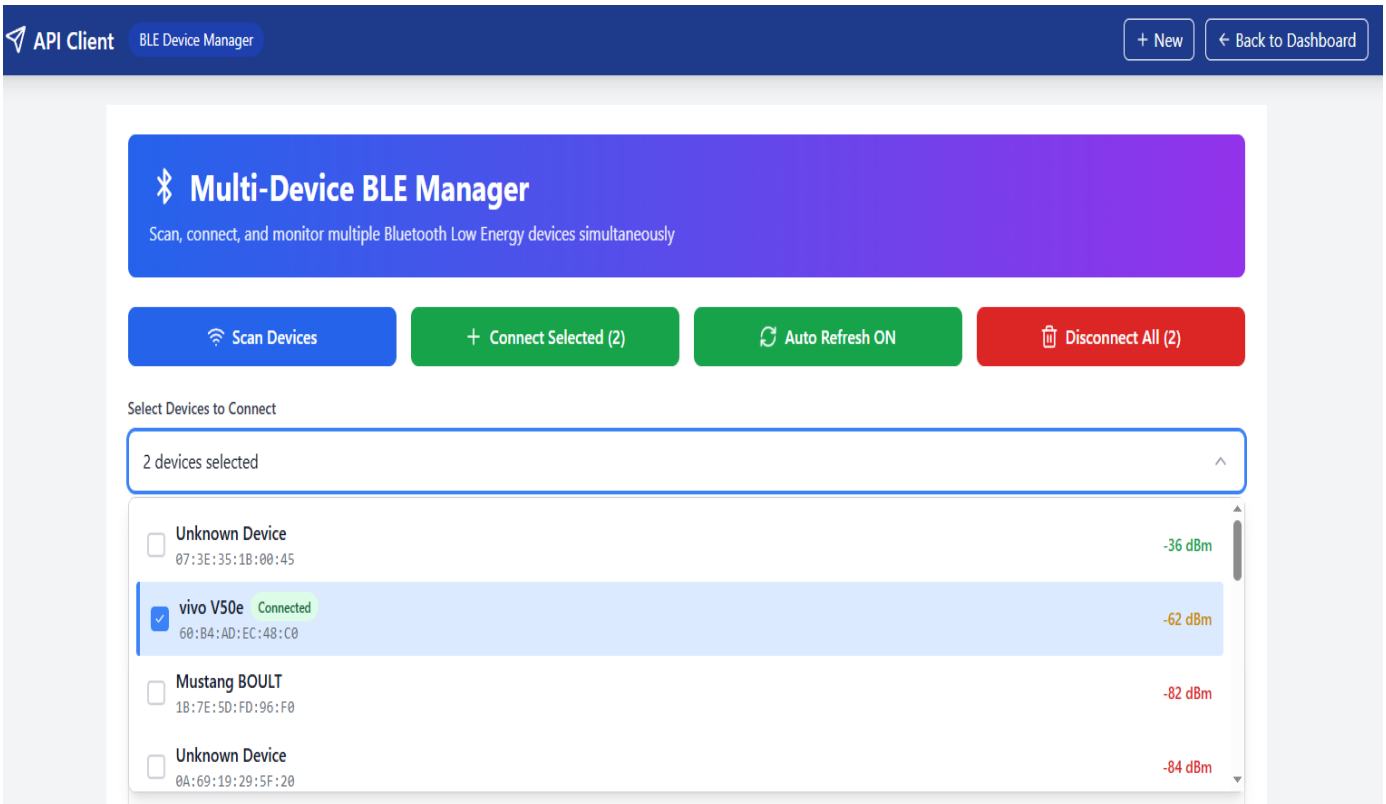
This stat card provides a quick overview of the device configuration and can be used for diagnostics and validation.

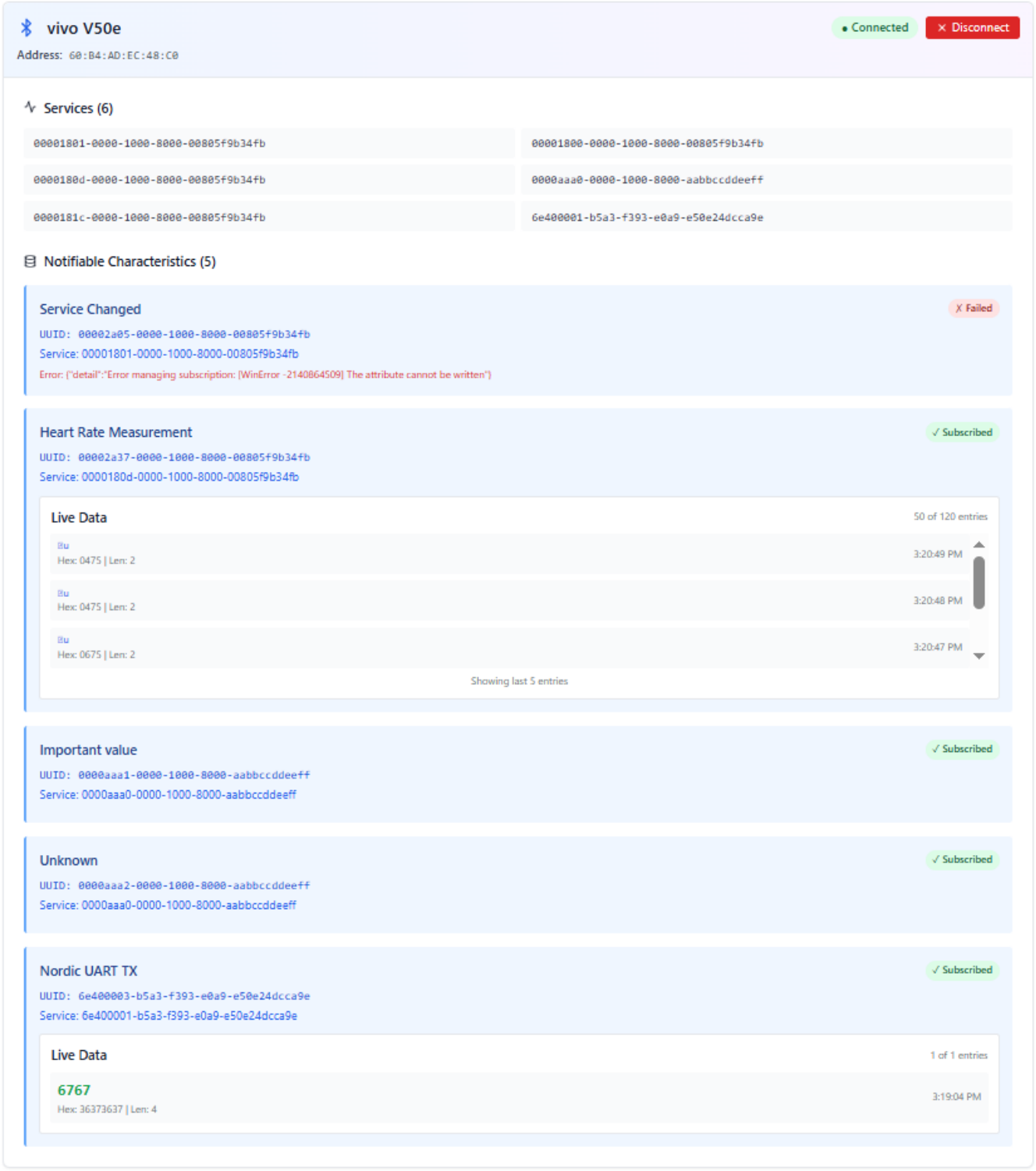
5. BLE Device Manager

Overview

The BLE Device Manager is a comprehensive web-based application designed to scan, connect, and monitor multiple Bluetooth Low Energy (BLE) devices simultaneously. The application provides real-time data streaming, notification management, and device characteristic monitoring capabilities.







System Architecture

Core Components

1. **Single Device Manager**: Individual device connection and monitoring
2. **Multi-Device Manager**: Simultaneous management of multiple BLE devices
3. **Device Scanner**: BLE device discovery and listing
4. **Characteristic Monitor**: Real-time monitoring of device services and characteristics
5. **Notification Handler**: Management of device notifications and subscriptions

Workflow Description

1. Device Discovery Phase

Process Flow:

1. Click "Scan for Devices" button to initiate BLE device discovery
2. System scans for available BLE devices in range
3. Discovered devices are listed with:
    - Device name (e.g., "vivo V50e", "Mustang BOULT")
    - MAC address
    - Signal strength (RSSI in dBm)
    - Connection status

2. Device Selection and Connection

Single Device Mode:

- Select individual device from dropdown
- View device-specific information:
    - MAC Address
    - Available services
    - Device characteristics

Multi-Device Mode:

- Select multiple devices using checkboxes
- Displays total selected devices count
- Batch connection capabilities
- Real-time connection status monitoring

3. Service and Characteristic Discovery

Once connected, the system automatically discovers:

Standard BLE Services:

- **Generic Access Profile (GAP)**: 00001800-0000-1000-8000-00805f9b34fb
- **Generic Attribute Profile (GATT)**: 00001801-0000-1000-8000-00805f9b34fb
- **Device Information Service**: 0000180a-0000-1000-8000-00805f9b34fb
- **Custom Services**: Device-specific UUIDs

Service Characteristics:

- **Heart Rate Measurement**: 00002a37-0000-1000-8000-00805f9b34fb
- **Battery Level**: 00002a19-0000-1000-8000-00805f9b34fb
- **Device Name**: 00002a00-0000-1000-8000-00805f9b34fb
- **Manufacturer Name**: 00002a29-0000-1000-8000-00805f9b34fb

4. Data Monitoring and Notifications

Notification Management:

- Subscribe to characteristic notifications
- Real-time data streaming

- Configurable notification intervals
- Data logging and timestamp recording

Supported Data Types:

- Heart rate measurements
- Battery level monitoring
- Custom sensor data
- Device status information

Technical Features

Device Connection Management

Connection States:

- **Scanning**: Discovering available devices
- **Connected**: Active connection established
- **Disconnected**: Connection terminated
- **Failed**: Connection attempt unsuccessful

Connection Statistics:

- Total Connected Devices: Real-time count
- Total Notifications: Cumulative notification count
- Active Subscriptions: Number of active characteristic subscriptions
- Data Streams: Number of active data streams

Multi-Device Capabilities

Simultaneous Operations:

- Connect up to multiple devices concurrently
- Individual device monitoring
- Batch operations (connect/disconnect all)
- Auto-refresh functionality for device status

Device Management:

- Signal strength monitoring (RSSI)
- Connection quality assessment
- Automatic reconnection handling
- Device filtering and sorting

Data Processing

Real-time Monitoring:

- Live characteristic value updates
- Timestamp-based data logging
- Configurable sampling rates
- Data export capabilities

Notification Handling:

- Automatic subscription management
- Error handling and retry mechanisms
- Data validation and parsing
- Custom notification filters

User Interface Components

Control Buttons

1. **Scan Devices**: Initiates BLE device discovery
2. **Connect Selected**: Establishes connections to selected devices
3. **Auto Refresh ON/OFF**: Toggles automatic device list refresh
4. **Disconnect All**: Terminates all active connections

Information Panels

1. **Device List**: Shows discovered and connected devices
2. **Connection Results**: Displays connection status summary
3. **Characteristic Monitor**: Real-time characteristic values
4. **Notification Panel**: Active notifications and subscriptions

Status Indicators

- **Signal Strength**: RSSI values in dBm
- **Connection Status**: Visual indicators (Connected/Disconnected)
- **Subscription Status**: Active/Inactive notification states
- **Data Stream Status**: Real-time data flow indicators

Error Handling

Common Error Scenarios:

- Device connection timeout
- Service discovery failures
- Characteristic read/write errors
- Notification subscription failures
- Device disconnection handling

Recovery Mechanisms:

- Automatic retry logic
- Connection state management
- Graceful error reporting
- User notification system

Security Considerations

BLE Security Features:

- Device pairing management
- Encrypted connections
- Authentication handling
- Access control mechanisms

Performance Metrics

Key Performance Indicators:

- Connection establishment time
- Data throughput rates
- Notification latency
- Device discovery time
- System resource utilization

API Integration

The system provides API endpoints for:

- Device management
- Data retrieval
- Configuration settings
- Status monitoring
- Batch operations

Future Enhancements

Planned Features:

- Device grouping capabilities
- Advanced filtering options
- Data analytics dashboard
- Export functionality
- Mobile application support
- Cloud data synchronization

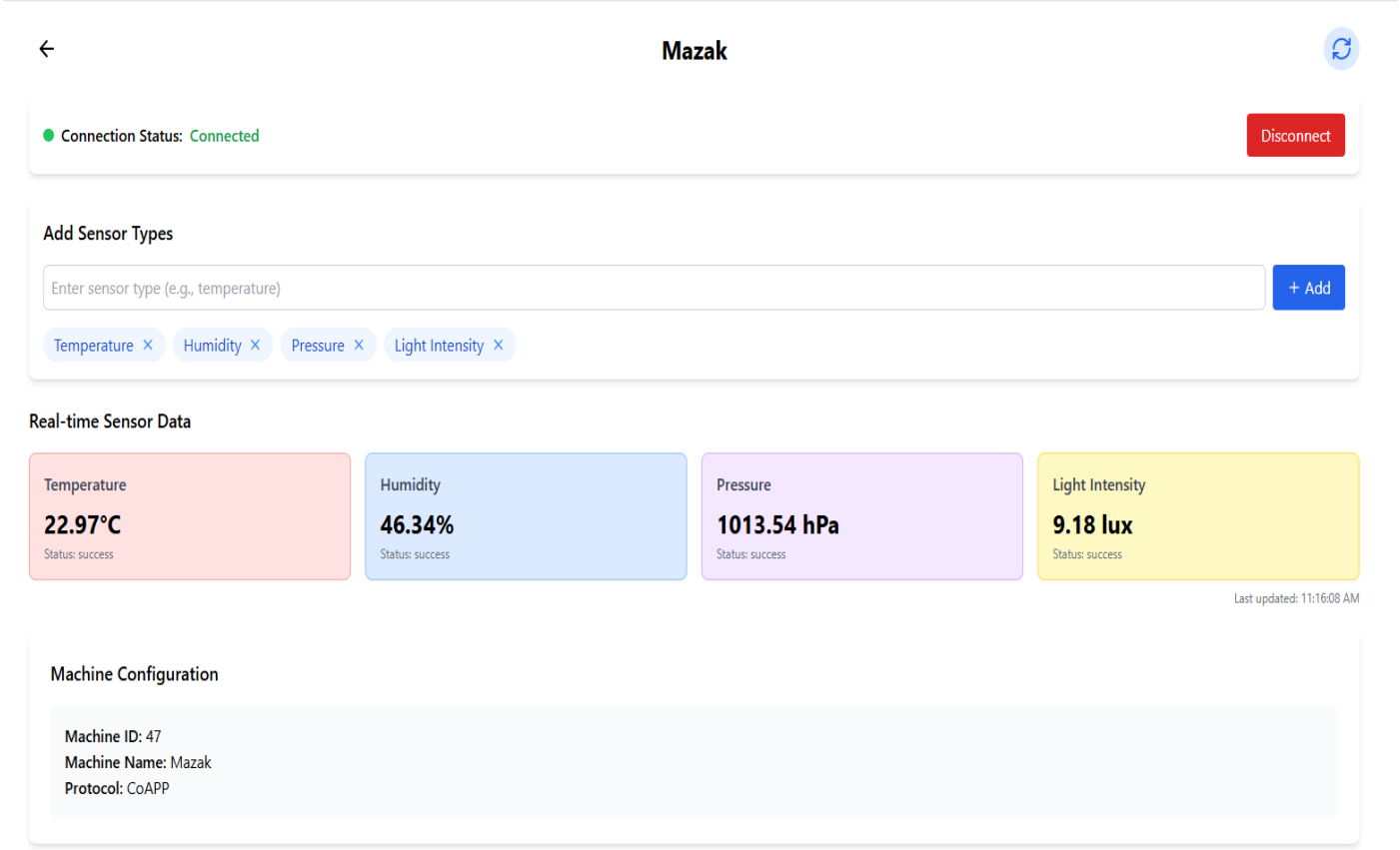
Troubleshooting Guide

Common Issues:

1. **Device Not Found**: Check Bluetooth permissions and device proximity
2. **Connection Failed**: Verify device compatibility and power status
3. **No Notifications**: Confirm characteristic subscription settings
4. **Data Loss**: Check connection stability and buffer settings

6. CoAP Protocol Overview

CoAP (Constrained Application Protocol) is a specialized web transfer protocol for use with constrained nodes and constrained networks in the Internet of Things, designed for machine-to-machine (M2M) applications such as smart energy.



CoAPP Dashboard Workflow Description

Based on monitoring dashboard, here's the workflow description:

System Overview

Machine Name: Mazak (Machine ID: 47)
Protocol: CoAPP (appears to be a variant or custom implementation of CoAP)
Connection Status: Connected (Active real-time monitoring)

Sensor Configuration

The system monitors four primary environmental and operational parameters:

1. **Temperature Sensor**
   - Current Reading: 22.97°C
   - Status: Success
   - Display: Pink/Rose colored card
2. **Humidity Sensor**
   - Current Reading: 46.34%
   - Status: Success
   - Display: Blue colored card

3. **Pressure Sensor**
    - Current Reading: 1013.54 hPa
    - Status: Success
    - Display: Purple colored card
4. **Light Intensity Sensor**
    - Current Reading: 9.18 lux
    - Status: Success
    - Display: Yellow colored card

Workflow Process

Data Collection Flow:

1. **Sensor Data Acquisition**: Four sensor types continuously collect environmental data from the Mazak machine
2. **Protocol Communication**: Data transmission occurs via CoAPP protocol over the network connection
3. **Real-time Processing**: System processes incoming sensor data and updates status indicators
4. **Dashboard Visualization**: Color-coded cards display current readings with success status
5. **Timestamp Management**: Last update timestamp (11:16:08 AM) tracks data freshness

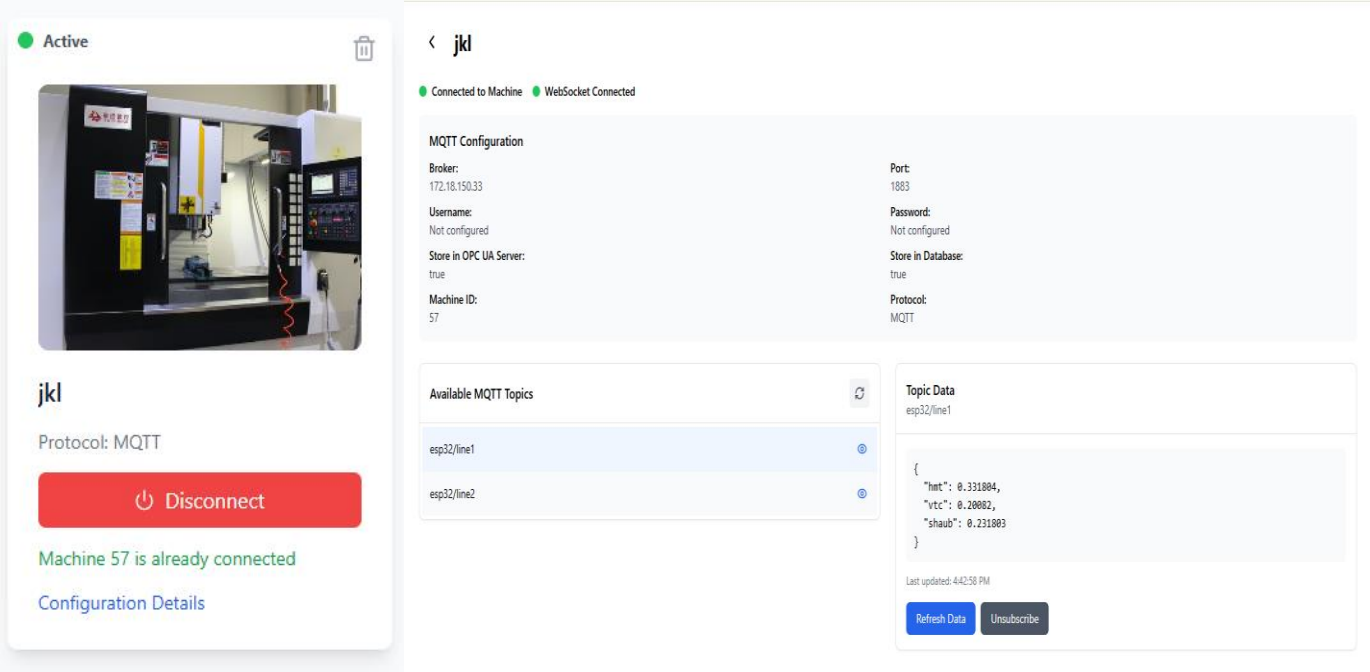
Management Features:

- **Dynamic Sensor Configuration**: "Add Sensor Types" functionality allows runtime addition of new sensor parameters
- **Connection Management**: Real-time connection status monitoring with disconnect capability
- **Status Monitoring**: Individual sensor success/failure status tracking
- **Real-time Updates**: Continuous data refresh with timestamp tracking

This system appears designed for industrial monitoring of the machine environment, providing operators with real-time visibility into critical environmental parameters that could affect machine performance and operations.

7. **MQTT Protocol Documentation and Interface Guide**

MQTT (Message Queuing Telemetry Transport) is a lightweight, open-source messaging protocol designed for machine-to-machine (M2M) communication and Internet of Things (IoT) applications. It follows a publish-subscribe pattern and is ideal for devices with limited bandwidth, high latency, or unreliable networks.



Key Features of MQTT

- **Lightweight**: Minimal code footprint and bandwidth usage
- **Reliable**: Three levels of Quality of Service (QoS) for message delivery
- **Scalable**: Supports thousands of connected devices
- **Secure**: Built-in security features with SSL/TLS support
- **Bi-directional**: Enables two-way communication between devices and servers

MQTT Architecture

MQTT uses a **publish-subscribe** model with three main components:

1. **Publisher**: Device or application that sends messages
2. **Subscriber**: Device or application that receives messages
3. **Broker**: Central server that routes messages between publishers and subscribers

How MQTT Works

1. Devices connect to an MQTT broker
2. Publishers send messages to specific **topics**
3. The broker receives and distributes messages to all subscribers of that topic
4. Subscribers receive messages from topics they've subscribed to

MQTT Topics

Topics are UTF-8 strings used to filter messages. They use a hierarchical structure with forward slashes as separators:

- Example: home/livingroom/temperature
- Wildcards: + (single level), # (multi-level)

Interface Explanation

Based on the interface shown in your image, here's how each component works:

Connection Status Indicators

- 🔴 **Disconnected from Machine**: Shows the physical device connection status
- 🟢 **WebSocket Connected**: Indicates successful connection to the web interface

MQTT Configuration Section

Broker Settings

- **Broker**: 172.18.150.33 - The IP address of your MQTT broker server
- **Port**: 1883 - Standard MQTT port (non-encrypted)
- **Username**: Not configured - No authentication username set
- **Password**: Not configured - No authentication password set

Storage and Protocol Settings

- **Store in OPC UA Server**: true - Data is being stored in an OPC UA server
- **Store in Database**: true - Data is being saved to a database
- **Machine ID**: 57 - Unique identifier for this specific machine
- **Protocol**: MQTT - Confirms MQTT protocol is being used

Available MQTT Topics Section

This section shows all the topics your device is publishing to:

- **esp32/line2** - Topic for data from line 2 of the ESP32 device
- **esp32/line1** - Topic for data from line 1 of the ESP32 device

Each topic has an 👁 **eye icon** that allows you to:

- View the topic in real-time
- Subscribe to receive updates
- Monitor data flow

Topic Data Panel

Shows real-time data from the selected topic (esp32/line1):


```
{
  "nnt": 0.366343,
  "vtc": 0.280584,
```

"shaub": 0.319935
}

This JSON data contains sensor readings or measurements with three parameters:

- **nnt**: Numeric value (possibly a sensor reading)
- **vtc**: Voltage or another measurement
- **shaub**: Another sensor parameter

Data Controls

- **Last updated**: 9:44:03 AM - Timestamp of the most recent data
- **Refresh Data** button - Manually refresh the data display
- **Unsubscribe** button - Stop receiving updates from this topic

How to Use This Interface

1. **Monitor Connection**: Check the status indicators at the top
2. **View Available Topics**: Browse topics in the left panel
3. **Subscribe to Topics**: Click the eye icon next to any topic
4. **View Real-time Data**: Selected topic data appears in the right panel
5. **Refresh Data**: Use the refresh button to get the latest values
6. **Manage Subscriptions**: Use unsubscribe to stop monitoring specific topics

Troubleshooting Tips

- If "Disconnected from Machine" shows red, check physical device connection
- Ensure the broker IP (172.18.150.33) is accessible on your network
- Port 1883 should be open and not blocked by firewalls
- If no data appears, verify the ESP32 device is publishing to the correct topics

Data Storage

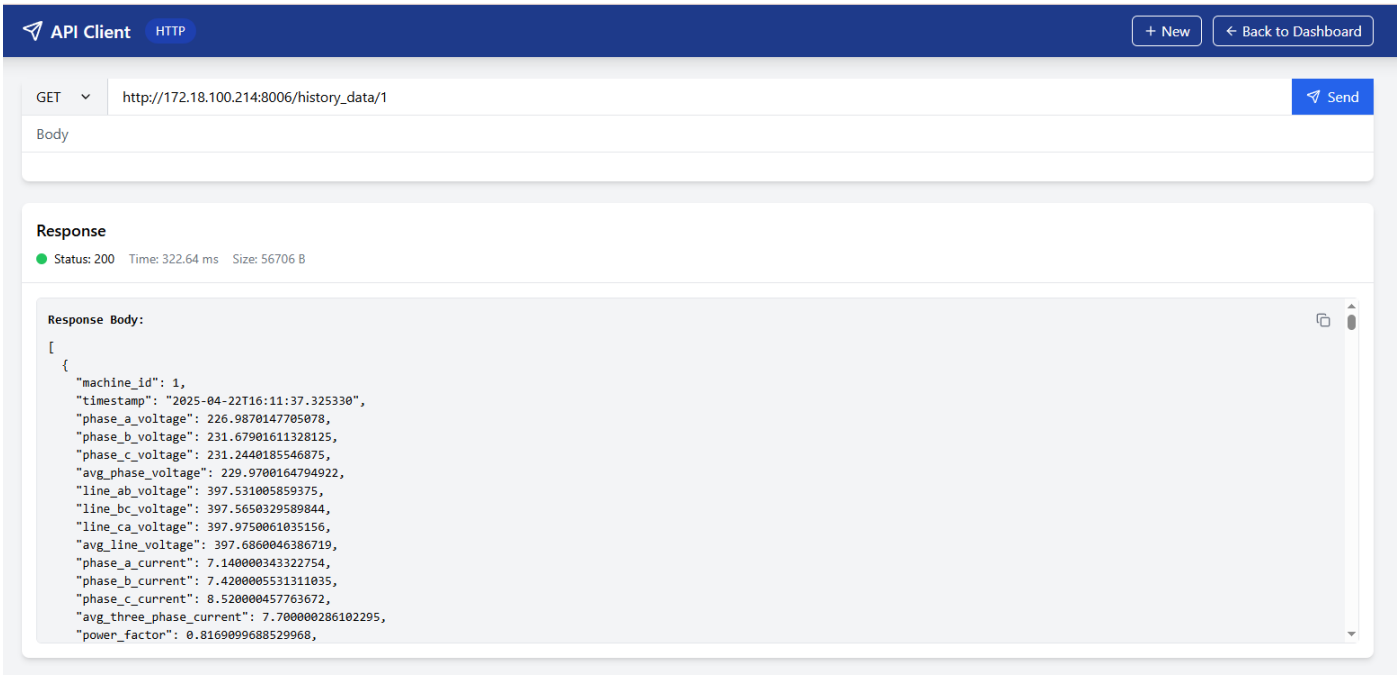
Your system is configured to:

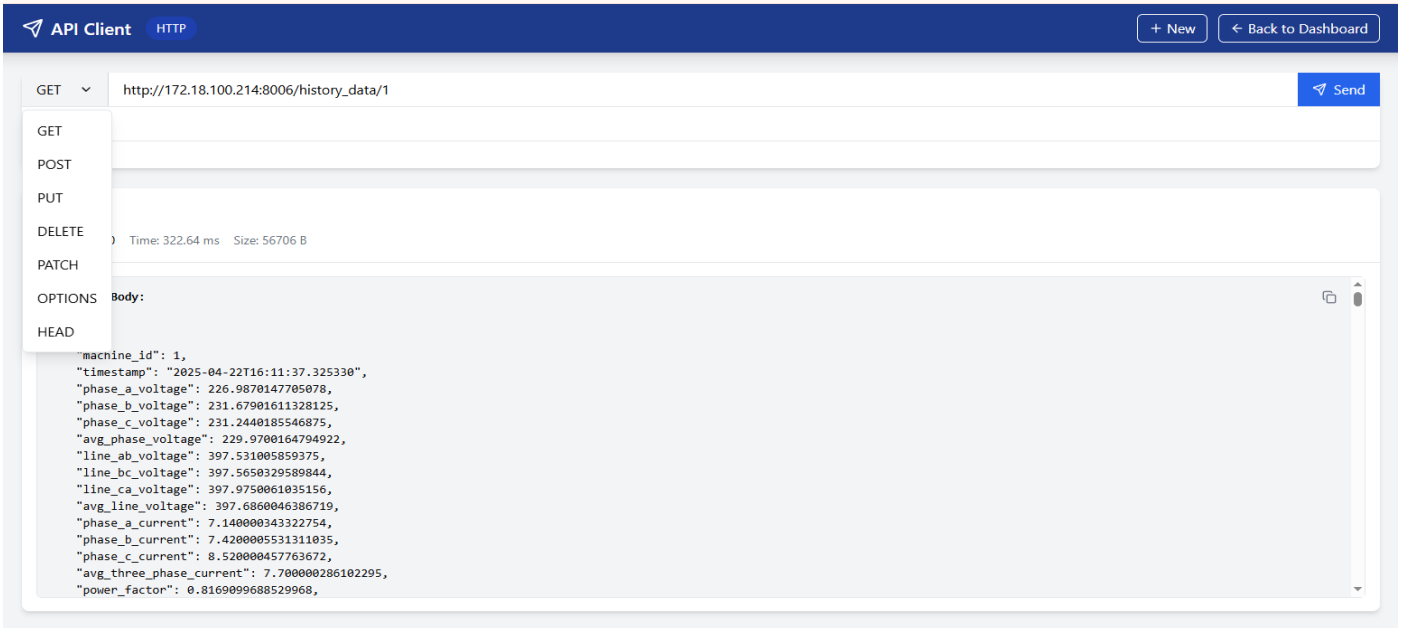
- Store data in an OPC UA server for industrial automation integration
- Save data to a database for historical analysis and reporting
- Use Machine ID 57 for data organization and identification

This setup provides a complete IoT data pipeline from sensor collection through MQTT to storage and visualization.

8. HTTP Protocol and API Client Interface Documentation

HTTP (HyperText Transfer Protocol) is the foundation of data communication on the World Wide Web. It's an application-layer protocol that defines how messages are formatted and transmitted between web browsers, servers, and other web services.





Key Features of HTTP

- **Stateless**: Each request is independent and contains all necessary information
- **Request-Response Model**: Client sends requests, server sends responses
- **Text-Based**: Uses human-readable text format
- **Port 80/443**: Standard ports for HTTP (80) and HTTPS (443)
- **Connectionless**: Connection is closed after each transaction (HTTP/1.0) or kept alive (HTTP/1.1+)

HTTP Architecture

HTTP follows a **client-server** model:

1. **Client** (browser, app, API client) sends HTTP requests
2. **Server** processes requests and sends HTTP responses
3. **Messages** contain headers, methods, status codes, and optional body data

HTTP Methods:

HTTP defines several methods (also called verbs) that indicate the desired action for a resource:

GET

- **Purpose**: Retrieve data from server
- **Characteristics**:
  - Safe operation (no side effects)
  - Idempotent (same result each time)
  - Data sent via URL parameters
  - Cacheable
- **Use Cases**: Fetching web pages, API data retrieval, downloading files

POST

- **Purpose**: Submit data to server to create new resources
- **Characteristics**:
  - Not safe (has side effects)
  - Not idempotent (multiple calls may create multiple resources)
  - Data sent in request body
  - Not cacheable
- **Use Cases**: Form submissions, creating new records, file uploads

PUT

- **Purpose**: Update or create a resource with complete replacement
- **Characteristics**:
  - Not safe (has side effects)
  - Idempotent (same result each time)
  - Data sent in request body
  - Replaces entire resource
- **Use Cases**: Updating user profiles, replacing configuration settings

DELETE

- **Purpose**: Remove a resource from the server
- **Characteristics**:
  - Not safe (has side effects)
  - Idempotent (deleting same resource multiple times has same effect)
  - Usually, no request body
- **Use Cases**: Deleting records, removing files, user account deletion

PATCH

- **Purpose**: Partial update of a resource
- **Characteristics**:
  - Not safe (has side effects)
  - Not necessarily idempotent
  - Data sent in request body
  - Updates only specified fields

- **Use Cases**: Updating specific fields, incremental changes

OPTIONS

- **Purpose**: Get information about communication options for a resource
- **Characteristics**:
  - Safe operation
  - Used for CORS preflight requests
  - Returns allowed methods and headers
- **Use Cases**: API discovery, CORS handling, checking server capabilities

HEAD

- **Purpose**: Get headers for a resource without the body
- **Characteristics**:
  - Safe operation
  - Idempotent
  - Same as GET but without response body
- **Use Cases**: Checking resource existence, getting metadata, cache validation

---

API Client Interface Analysis

Interface Overview

Your API client is a web-based tool for testing and interacting with HTTP APIs. It provides a clean interface for making HTTP requests and viewing responses.

Current Request Configuration

Endpoint: http://172.18.100.214:8006/history_data/1

- **Protocol**: HTTP (not HTTPS)
- **Host**: 172.18.100.214 (local network IP)
- **Port**: 8006 (custom port)
- **Path**: /history_data/1 (endpoint for historical data with ID 1)

Request Details

- **Method**: GET (currently selected)
- **Body**: Empty (appropriate for GET requests)
- **Purpose**: Retrieving historical data for machine/device ID 1

Response Information

Status Indicators

- **Status**: 200 (Success - OK)
- **Time**: 322.64 ms (Response time)
- **Size**: 56706 B (Response size - approximately 56KB)

Response Data Analysis

The API returns electrical/power monitoring data in JSON format:

```
{
  "machine_id": 1,
  "timestamp": "2025-04-22T16:11:37.325330",
  "phase_a_voltage": 226.9870147705078,
  "phase_b_voltage": 231.67901611328125,
  "phase_c_voltage": 231.24401855468875,
  "avg_phase_voltage": 229.9700164794922,
  "line_ab_voltage": 397.531005859375,
  "line_bc_voltage": 397.565032958984,
  "line_ca_voltage": 397.975006103516,
  "avg_line_voltage": 397.6860046386719,
  "phase_a_current": 7.140000343322754,
  "phase_b_current": 7.420000553131035,
  "phase_c_current": 8.520000457763672,
  "avg_three_phase_current": 7.700000286102295,
  "power_factor": 0.8169099688529968
}
```


Data Parameters Explained:

- **machine_id**: Identifier for the electrical equipment
- **timestamp**: When the measurement was taken
- **phase_*_voltage**: Individual phase voltages (A, B, C)
- **avg_phase_voltage**: Average of all three phase voltages
- **line_*_voltage**: Line-to-line voltages (AB, BC, CA)
- **avg_line_voltage**: Average line voltage
- **phase_*_current**: Current measurements for each phase
- **avg_three_phase_current**: Average current across all phases
- **power_factor**: Efficiency ratio (0-1, closer to 1 is better)

Interface Workflow

Step 1: Method Selection

1. Click the method dropdown (currently showing "GET")
2. Select appropriate HTTP method from the list:
    - GET: Retrieve data
    - POST: Create new data
    - PUT: Update/replace data
    - DELETE: Remove data
    - PATCH: Partial update
    - OPTIONS: Get server capabilities
    - HEAD: Get headers only

Step 2: URL Configuration

1. Enter the complete API endpoint URL
2. Include protocol (http/https), host, port, and path
3. Add query parameters if needed (for GET requests)

Step 3: Request Body (if applicable)

1. For POST, PUT, PATCH requests, add JSON data in the Body section
2. Set appropriate Content-Type headers
3. Format data according to API requirements

Step 4: Send Request

1. Click the blue "Send" button
2. Wait for response (time shown in status)
3. Review response data in the Response Body section

Step 5: Response Analysis

1. Check status code (200 = success, 400s = client error, 500s = server error)
2. Review response time and size
3. Analyze returned data structure and values
4. Use copy button to export response data

Interface Controls

- **+ New**: Create a new API request
- **← Back to Dashboard**: Return to main dashboard
- **Method Dropdown**: Select HTTP method
- **URL Field**: Enter API endpoint
- **Send Button**: Execute the request
- **Copy Button**: Copy response data
- **Body Section**: Add request payload (for POST/PUT/PATCH)

Best Practices for API Testing

1. **Start with GET**: Always test data retrieval first
2. **Check Status Codes**: Ensure requests return expected status
3. **Validate Data**: Verify response structure and values
4. **Test Error Cases**: Try invalid IDs or malformed requests
5. **Monitor Performance**: Check response times for optimization
6. **Document Results**: Save successful request configurations

Troubleshooting Common Issues

- **Connection Refused**: Check if server is running on specified port
- **404 Not Found**: Verify endpoint URL and path
- **401/403 Unauthorized**: Add authentication headers if required
- **500 Server Error**: Check server logs for backend issues
- **Timeout**: Increase timeout settings or check network connectivity

This interface is ideal for testing your electrical monitoring API, allowing you to retrieve historical power data, test different endpoints, and validate your system's API responses.

9. Conclusion

The Protocol Management Software is a comprehensive industrial IoT integration platform that enables seamless connectivity and data collection from diverse manufacturing machines and devices across multiple communication protocols including Modbus, PROFINET, OPC UA, MQTT, BLE, and HTTP. It serves as a universal data gateway that bridges legacy industrial equipment with modern IT infrastructure, providing real-time monitoring, centralized data storage, and unified dashboard management. The software eliminates protocol compatibility barriers by automatically converting and standardizing data from different sources into consistent formats for database storage and OPC UA server integration. Its primary value proposition is enabling digital transformation for industrial operations by unlocking data from previously isolated systems, facilitating predictive maintenance, performance optimization, and data-driven decision making. This solution is particularly valuable for organizations seeking to modernize their industrial data infrastructure while preserving existing equipment investments and ensuring scalable, secure, and reliable industrial data management.