Assignment: Modbus RTU (over TCP) System Monitoring

Date: 2025-11-24
Candidate: Hasintha Dilshan - Software Engineer

1. Objective

Build a System monitoring solution with the following components:

1. Modbus Master & REST APIs (Spring Boot)
    - Schedules and executes Modbus RTU requests over TCP to remote Modbus clients.
    - Exposes REST APIs for job scheduling and status.

2. Modbus Slave (C/C++ CLI Tool)
    - Runs on Ubuntu 22.04 or 24.04.
    - Responds to Modbus RTU requests over TCP.
    - Provides CPU, RAM, and Disk usage metrics.

3. Web UI (React)
    - Allows users to schedule monitoring jobs.
    - Polls the REST API for job status and displays metrics.

2. System Architecture

Components:

1. **Web UI (React)**

    - Inputs: target server IP, CRON expression.
    - Submits job creation request via REST.
    - Polls for job status & data.
    - Displays metrics (CPU, RAM, Disk usage) visually.

2. **Spring Boot Backend (Modbus Master)**

    - REST API endpoints for job management:

HTTP Method	Endpoint	Description
POST	/jobs	Schedule a new monitoring job
GET	/jobs/{jobId}	Retrieve status and metrics for a specific job.
GET	/jobs	List all jobs, optionally filtered by status. (optional)
DELETE	/jobs/{jobId}	Stop a scheduled job. (optional)

Example Job Status JSON

```
{
   "jobId":"76221913-32ea-4e74-8289-0285677271ca",
   "status":"RUNNING", // RUNNING | STOPPED
   "executions":[
      {
         "telemetry":{
            "cpu":45,
            "ram":60,
            "disk":70
         },
         "status":"COMPLETED", // PENDING | COMPLETED |
ERROR_APP | ERROR_TCP | ERROR_TIMEOUT | ERROR_MODBU
         "exeAt":1763717292000
      },
      {
         "telemetry":{
         },
         "status":"ERROR_TIMEOUT",
         "exeAt":1763717392000
      }
   ],
   "sceduledAt":1763717292000
}
```

- Connects to Modbus Slave over TCP port 5000 and reads input registers (0x04) using RTU framing. The Modbus protocol must be implemented from scratch, without using any third-party Modbus libraries. Support for additional or unused Modbus function codes is optional.

Reads holding registers:

Register	Metric	Description
0x04	CPU Usage	Percentage (0.00–100.00)
0x06	RAM Usage	Percentage (0.00–100.00)
0x08	Disk Usage	Root partition usage (0.00–100.00)

- Updates job status (PENDING, RUNNING, COMPLETED, ERROR).

3. **Modbus Slave (C/C++ CLI Tool)**

   - Command-line tool that can run on Ubuntu 22.04 or Ubuntu 24.04.
   - Accepts a Slave Address as a required command-line argument.

     Eg:

     ```
     ./system_monitor --slave_address 1
     ```

   - Listens on TCP port 5000 for Modbus RTU requests.
   - Returns CPU, RAM, and Disk usage metrics in response to holding register reads.

3. Workflow

1. The user enters the Target IP and the CRON expression in a Web UI (React form).
2. Frontend sends POST /jobs to Spring Boot backend.
3. Backend schedules a job
4. At the scheduled time (according to the CRON expression), the backend connects to the Modbus Slave over TCP and sends Modbus RTU read requests for registers 0x04, 0x06, and 0x08.
5. Modbus slave parses the request & returns the requested data
6. Backend
    a. Parses response
    b. Hold results
    c. Updates job status
    d. Persists results & status in RDBMS (optional)
7. Web UI polls /jobs/{jobId} to show status and metrics.

4. Deliverables

- A GitHub repository link containing:
    - All 3 application components
    - API & USER documents
    - dockerr-compose.yml
- A Google Drive link to a demonstration video showcasing the complete workflow described in Section 3 of this document.

5. Evaluation Criteria

Components/Area	**Weight**
Modbus Master & REST APIs (Spring Boot)	35%
Modbus Slave (C/C++ CLI Tool)	25%
Web UI (React)	10%
Other non-functional, optional requirements & out-of-the-box thinking	30%