

ECE Capstone Project Final Report

Project Title	Blockchain-Assisted Decentralized Energy Management Framework for IoT-Enabled Microgrid System
Team	Damir Shyganakov — Student Daniyal Zhanabayev — Student Yernar Yestayev — Student
Supervisor	Prashant Jamwal
Course/Term	ENG 400 Capstone Project, [2026]
Date	[11-04-2026]

Prepared by: Damir Shyganakov, Daniyal Zhanabayev, Yernar Yestayev

Reviewed by: Prashant Jamwal

TABLE OF CONTENTS

0. ABET Evidence Map, Final Design Snapshot, and Design Evolution Log.....	4
1. Executive Summary.....	8
2. Problem Definition, Stakeholders, Context, and Constraints.....	10
2.1 Problem Definition and Need Statement.....	10
2.2 Stakeholders.....	10
2.3 Context and Operating Environment.....	11
2.4 Constraints and Context Factors.....	11
3. Final Requirements, Acceptance Criteria, and Traceability.....	12
3.1 Final Requirements Compliance.....	12
3.2 Requirements Traceability.....	15
3.3 Derived, Interface, Environmental, and Regulatory Requirements.....	17
4. Engineering Standards, Regulations, and Compliance Evidence.....	20
5. Concept Generation, Trade-off Study, and Selected Concept.....	23
5.1 Candidate Concepts.....	23
5.2 Weighted Trade-off Study.....	24
5.3 Selected Concept and Rationale.....	25
6. Design Iterations and Decision History.....	27
7. Final System Architecture.....	29
8.1 Hardware design.....	31
8.2 Software / firmware design.....	33
8.3 Mechanical / packaging design (if applicable).....	37
8.4 Prototype realization and integration.....	37
8.5 Final bill of materials and design-for-manufacture / assembly notes.....	38
9. Verification, Validation and Experimental Results.....	39
10. Risk Management, Schedule, Budget, and Procurement Outcomes.....	42
10.1 Final Risk Register.....	42
10.2 Milestones Planned vs. Actual.....	44
10.3 Budget, Procurement, and Substitutions.....	45
11. Professional Responsibilities and Broader Impacts.....	47
11.1 Public Health, Safety, and Welfare.....	47
11.2 Ethical and Professional Responsibilities.....	47
11.3 Security and Privacy.....	47
11.4 Accessibility and Usability.....	48
11.5 Environmental, Economic, Social, Global, and Cultural Factors.....	48
12. Team Organization, Collaboration, and Individual Contributions.....	49

13. Use of Prior Course Work and New Knowledge Acquired.....	53
14. Conclusions and Recommendations.....	55
15. References.....	56
Appendix A. Engineering Standards Evidence.....	58
Appendix B. Multiple-Constraints Matrix.....	60
Appendix C. Team Organization and Contributions.....	62
Appendix D. Design Evidence.....	63
Appendix E. Verification Evidence.....	76
Appendix F. Project Management Evidence.....	82

0. ABET Evidence Map, Final Design Snapshot, and Design Evolution Log

Table 0-A. Report Evidence Guide.

ABET item / SO	What must be demonstrated	Primary report section	Supporting appendix / evidence ID
<i>SO1</i>	Identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics.	Sections 2 & 3: Problem Definition, Requirements	Fig D6-D11
<i>SO2</i>	Apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors.	Sections 5, 7 & 8: Concept Selection, System Architecture, Detailed Design.	Fig D1-D2, Fig D6-D9, Fig D12-D13
<i>SO3</i>	Communicate effectively with a range of audiences	Sections 1 & 9: Executive Summary, Results	Fig D17-D18
<i>SO4</i>	Recognize ethical and professional responsibilities in engineering situations and make informed judgments, which must consider the impact of engineering solutions in global, economic, environmental, and societal contexts.	Section 11: Professional Responsibilities	Fig D18-D20
<i>SO5</i>	Function effectively on a team whose members together provide leadership, create a collaborative and inclusive environment, establish goals, plan tasks, and meet objectives.	Section 12: Team Organization	Appendix C
<i>SO6</i>	Develop and conduct appropriate experimentation, analyze and interpret data, and use engineering judgment to draw conclusions.	Section 9: Verification & Validation	Fig D4
<i>SO7</i>	Acquire and apply new knowledge as needed, using appropriate learning strategies.	Section 13: New Knowledge Acquired	Fig D11

ABET item / SO	What must be demonstrated	Primary report section	Supporting appendix / evidence ID
<i>Criterion 5 design evidence</i>	The culminating design shows standards, multiple constraints, iteration, and use of prior course work.	Section 5, 6,7, 8: Design + Implementation	Fig D1-D20

Table 0-B. Final Design Snapshot.

Field	Final value / summary
Final prototype / revision	The final prototype consists of an industrial level smart meter based IoT energy capturing and blockchain integrated P2P trading system. Raspberry Pi 5 collects smart meter readings and publishes it through a MQTT broker. Node Red programming tool subscribes to the topics of the MQTT broker and preprocesses the data before logging it to the CSV. Collected data is used in calculating market-clearing prices via SDR algorithms. The P2P trading system then implements this data to create a blockchain layer with Node.js , Solidity, and Ganache.
Main need addressed	This project addresses the need for decentralized, and low-cost energy monitoring and trading systems inside a microgrid environment that has its prosumers and consumers. This is achieved by real-time monitoring of energy patterns, data processing, and blockchain transactions.
Requirements met	The list of the requirements met in this Capstone Project: 1. Data collection from smart meters via Raspberry Pi 5 (Modbus RTU and MQTT) 2. Pre-processing of the acquired data (Node Red) 3. Supply-Demand Ratio based energy price calculations 4. Blockchain recording of the smart-contracts 5. User interface for analysis and monitoring
Primary standards applied	The main engineering standards that were used in the project are: 1. IEEE 2030.5-2023 – Smart Energy Profile Application Protocol 2. ISO/IEC 27001:2022 – Information Security Management
Key constraints satisfied	Most of the constraints described in Section 2 are satisfied. Smart meters display accurate and real data, prices calculated for energy trading are correctly decided according to the algorithms. The user interface showcases correct smart contracts and blockchain blocks.

Field	Final value / summary
Most important trade-off	Decision to create an optimized edge-processing architecture rather than keeping straightforward Raspberry Pi based pipeline or more powerful external server design was the most important trade-off. The following project is an ideal trade-off in terms of complexity and implementation level.
Overall result	The overall result is a working prototype of a microgrid energy management system, that shows local processing of the energy data and energy trading among the users of the microgrid.

Table 0-C. Design Evolution Log.

Iteration ID	What changed	Why it changed	Evidence that triggered the change	Impact on performance / cost / schedule / risk	Date
I1	Implementing a CSV-based data simulation from an existing theoretical one	Unavailability of smart meters at early stages	A stable environment for testing was required at initial steps	Allowed to test the system quite early and increased reliability	Feb 2026
I2	Households' agent-based model development	Individual household behavior simulation was required	Prosumers and consumers were required to be modeled separately	Simulation accuracy was increased	Mar 2026
I3	Trading logic based on peer-to-peer principle implementation (bids and asks)	Decentralized energy trading was required to be simulated	Local matching was required based on data from aggregated demand/supply	Local energy optimization was enabled while grid dependency was reduced	Mar 2026
I4	Ganache + Solidity smart contract based blockchain integration	Transactions needed to be recorded in a secure way	Decentralized validation of trading transactions was required	Transparency of the system was increased	Mar 2026

Iteration ID	What changed	Why it changed	Evidence that triggered the change	Impact on performance / cost / schedule / risk	Date
I5	Dashboard for visualization development (UI)	Blockchain results interpretation had to be clear	Difficulty of analysis of raw CSV data outputs	Communication of results, as well as usability were improved and system validation was enabled	Apr 2026
I6	Implementation of energy trading pricing based on tariffs specific for Kazakhstan energy trade	Evaluation of the economic part of the project had to be realistic	Real-world conditions needed to be considered when testing the system	Economic accuracy was improved	Apr 2026

1. Executive Summary

In the modern world, the integration of renewable energy sources has made decentralized energy management possible within the microgrids. Traditional centralized energy systems proved to be inefficient and less secure compared to contemporary decentralized ones, where houses can consume and generate electricity at the same time, making profit of it. This can be achieved by owning a rooftop solar panel, wind turbines and many other sources of renewable energy. However, it is important to note that such energy trading can cause serious security risks if not handled properly. Hence, a secure and transparent mechanism for recording energy transactions is required. This project's main motivation came from the need to integrate microgrid energy management with a mechanism for a safe and efficient blockchain-based market energy coordination.

Traditional grid-centered energy management networks cannot provide a fully transparent and effective method for grid users to exchange energy locally. Moreover, energy is mostly statically priced without a direct linkage between supply and demand in real time. Thus, a regular energy balancing is required with a dynamic determination of local trading prices, as well as the secure energy market transactions record.

A blockchain-assisted decentralized energy management framework for an IoT-enabled microgrid system is the final design solution. Data is collected from the laboratory grid using smart meters, which is then processed by the Node-RED to be stored in CSV files, using Python programming language for data validation and preparation. In the project 15 real houses data is processed, a supply-demand ratio (SDR) is calculated and dynamic internal prices bounded by Kazakhstan grid tariffs (30.56 / 22.68 KZT/kWh) are determined, with the priority given to peer-to-peer energy trading before interaction with the main grid. Furthermore, local asks and bids, market-clearing actions are recorded in the form of hashes and blockchain blocks for security with all the data being visualized in a web dashboard.

Final Performance Achieved:

- Smart meter data acquisition in real time using Modbus protocol
- Raspberry Pi + Node-RED + CSV - full IoT pipeline
- Simulation of real time hourly rate data for 15 houses
- Dynamic prices based on SDR with Kazakhstan grid tariffs integration.
- Energy trading based on peer-to-peer principles
- Transactions recorded as blockchain blocks and using hashes
- Cost savings implementation and demonstration
- Project web visualization

Requirements met:

- End-to-end system (hardware - data - simulation - blockchain - UI)
- SDR pricing + hourly energy analysis + real tariff integration
- Blockchain logging

- Dashboard visualization

Partially met:

- Not fully real time, since data in the CSV files used instead of real time livestreams
- Full automated settlements can be improved

Unmet:

- Large scale deployment
- Full real grid integration and implementation

Within the scope of the laboratory equipment there were some system limitations such as that the system was close to being real time, however to become fully real time it will need to process livestreams of energy data instead of relying on intermediate data. In the future, live smart meter data can be directly connected to the simulation and blockchain to achieve even more secure and efficient energy trading mechanisms.

2. Problem Definition, Stakeholders, Context, and Constraints

2.1 Problem Definition and Need Statement

Existing centralized energy grids do not allow peer-to-peer trading, while modern microgrids have distributed energy resources such as solar panels, creating a new term called prosumer, allowing households not only to consume but also sell energy. Moreover, conventional grids have fixed pricing mechanisms, not reflecting real time supply-demand conditions. The capstone project combines real smart meter data, internal pricing adjustments based on SDR, hourly energy balancing mechanisms, as well as security provided by the blockchain. Unlike a purely research/theory based project, this one follows engineering design standards by involving hardware such as laboratory microgrids, smart meters, pi raspberry 5, as well as data pipelines, blockchain, and results simulation.

2.2 Stakeholders

Table 2-A. Stakeholder Analysis.

Stakeholder	Role/Interest	Needs/Success criteria	How engaged (meetings/inputs)	Impact of project
Capstone Project Team	System developers	System implementation being completed successfully with accurate results	Weekly meetings, meeting minutes, feedbacks, peer evaluations	Acquiring of technical and research knowledge and skills
Capstone Supervisor	Academic evaluator	Innovative and clear methodology	Regular feedback sessions, consultations	High academic quality ensurement
Electrical and Computer Engineering Department	Academic Institution	Applied engineering knowledge demonstration	Regular presentations, reviews	Program reputation enhancement + contribution to accreditation
AC-DC microgrid lab	Experimental environment	Proper usage of lab equipment contributing to overall system successful implementation	Hardware setup and testing environment	Real world validation implementation
Potential End-Users such as RES owners, prosumers, consumers	Potential future users	Security, transparency, cost minimization, efficiency.	Indirect simulated interactions	Reduced cost while increased energy efficiency

2.3 Context and Operating Environment

This capstone project is implemented within an AC-DC hybrid microgrid laboratory located in Nazarbayev University C4. Various electrical data such as power, frequency, voltage are collected using one phase and three phase smartmeters, which are then processed by Raspberry Pi 5 and Node-RED system. An hourly time resolution method is used, also assuming that multiple households interact with the local grid. Pricing is constrained by real Kazakhstan electricity tariffs (30.56 KZT/kWh for purchasing and 22.68 KZT/kWh for selling). Ganache is used as a simulated environment for blockchain component implementation, allowing decentralization of transactions. Key assumptions of the capstone project are as follows:

- Smart meter data acquisition in a reliable and secure way
- Acquisition of energy data based on the hourly rate
- System components communicating with each other in a stable way
- Tariff boundaries are fixed for pricing

2.4 Constraints and Context Factors

Table 2-B. Constraints Matrix.

Constraint category	Metric / target	Final estimate or measurement	Gap / residual risk	Owner	Evidence / next action
Technical	Accurate smart meter data	Received by Modbus smart meters	Data being missed or noisy	Team	Filtering and validation improvements
Computational	Hourly processing speed	Real-time 15 households data	Issues related to scaling	Team	Algorithms optimizations
Economic	Match real KZ tariffs	30.56 KZT/kWh for purchasing and 22.68 KZT/kWh for selling	Unexpected changes in tariffs	Team	Update them dynamically
Blockchain	Gas usage minimization	Simulated gas kept in record	Cost being significantly higher in real networks like Polygon	Team	Smart contract optimizations
Hardware	Data acquired in a stable way	Raspberry Pi 5 and Node-RED system operating correctly	Risk of hardware failure	Lab	Redundancy additions Evidence link.

3. Final Requirements, Acceptance Criteria, and Traceability

3.1 Final Requirements Compliance

Table 3-A. Final Requirements Compliance Matrix.

Req. ID	Requirement statement	Source	Verification method	Target threshold	Measured result	Verdict	Evidence ID
REQ-1	The Raspberry Pi 5 shall successfully read Modbus RTU registers from each connected smart meter	Data acquisition	Execute Python script that polls each smart meter's primary power register	100% of read attempts return into Raspberry Pi 5	100% accurate values taken from connected smart meter	Pass	Evidence
REQ-2	The Node-RED pipeline shall write every Modbus reading to a CSV file on the Raspberry Pi 5	Data logging	Run polling script for 10 minutes. Inspect CSV file. Count logged rows vs expected poll cycles.	CSV file exists and contains 100% of expected data rows.	100 % of all data has been added into CSV files	Pass	Evidence
REQ-3	The smart meter power measurement shall be accurate within $\pm 5\%$ of a calibrated bench multimeter	Stakeholder need (Fair trading)	Record power from smart meter (Modbus) and bench multimeter simultaneously.	Average error $\leq 5\%$	Absolute error = 5.33% Calculated percentage error = 1.33%	Pass	Evidence
REQ-4	The EMS shall calculate total energy supply and total energy demand from the 15-household CSV dataset such that the	Energy balancing	Compute total supply and total demand . Verify that net energy for all houses equal to zero.	Net energy for all 15 houses shall be equal to 0	23421.03 Wh - 15787.03 Wh=7634.00Wh	Pass	Evidence

	sum of all household net energy values equals zero						
REQ-5	The EMS shall calculate Supply-Demand Ratio(SDR) and set the internal market price	Dynamic pricing	Change number of consumers and prosumers to have high SDR and check internal price	High SDR value equals low internal value, but low SDR value equals high internal value	Consumers = 3 Prosumers = 4 SDR = 3.9657 Internal Price = 22.6800 (Price = 22.68 for $SDR \geq 1.0$)	Pass	Evidence
REQ-6	The blockchain system (Ganache) shall generate and display a unique transaction hash and block number for every submitted bid, ask, and market-clearing transaction	Safety constraint (IEEE 1547 Clause 6)	Submit 5 test transactions. Use Ganache to retrieve transaction hashes and block numbers for each.	100% of submitted transactions produce a non-null transaction hash and a valid block number.	100% accuracy	Pass	Evidence
REQ-7	The market clearance algorithm shall correctly match buyer bids with seller asks	P2P trading	Run simulation with a test dataset containing 3 buyers and 12 sellers with known bid/ask prices and quantities. Inspect the matching	100% of matched trades. No household's cleared sale exceeds its available supply, and no household's cleared	100% accurate number of buyers and seller. Local asks does not exceed number of prosumers (equal to 12) and same with	Pass	Evidence

			output for price and quantity violations.	purchase exceeds its stated demand.	local bids and consumers (equal to 3)		
REQ-8	The web dashboard shall display the following data for any selected hour: total supply, total demand, SDR value, internal market price, and the most recent blockchain transaction hashes	Safety constraint (IEEE 1547 Clause 6)	Manual UI inspection. Load the dashboard. Verify presence of all five data elements. Click through different hours and verify that displayed values change.	All data elements are visible. Data updates correctly when hour selection changes.	100% accurate use expectations and requirements	Pass	Evidence
REQ-9	The EMS shall correctly determine grid import/export status for the microgrid as a whole: when total supply bigger than total demand status will be changed to surplus, and when total supply less than total demand status will be changed to deficit	Performance constraint	Run simulation with surplus case (total supply > total demand) and verify export status.	Status changes to surplus	total supply > total demand —> status=surplus	Pass	Evidence

REQ-10	The microgrid protection system shall activate the circuit breaker when the measured current exceeds the circuit breaker's rated current threshold, isolating the node from the microgrid	Safety constraint (IEEE 1547 Clause 6)	Verified by supervisor confirmation that the breaker will activate at 15 A per manufacturer specification. No physical overcurrent test performed due to safety.	15 A according to datasheet and supervisor verification.	15 A > Current creates overcurrent situation. Confirmed by supervisor based on CB datasheet	Pass	Evidence Evidence
REQ-11	Raspberry Pi 5 CPU load shall remain $\leq 70\%$ during EMS	Stakeholder need (Transparency)	Run top or htop logging while system operates at expected load.	CPU load shall be $\leq 70\%$ average over 10 minutes	14.55 % load in 36 minute of usage	Pass	Evidence

3.2 Requirements Traceability

Table 3-B. Requirements Traceability Matrix.

Stakeholder need	Requirement(s)	Constraint(s)	Standard clause(s)	Design feature(s)	Test ID(s)	Result section
Prosumers and consumers need fair, accurate energy measurements for trading.	REQ-1: Modbus accuracy $\geq 90\%$ vs. bench multimeter	Accuracy $\geq 90\%$ (Appendix B)	IEEE 2030.5 §8	Python script reads smart meter data and saves to CSV for comparison with multimeter.	T1	
Prosumers and consumers need tamper-proof,	REQ-8:	100% data integrity (Appendix B)	ISO 27001 A.5.15, A.8.24	Private blockchain with smart contract that emits	T6	Section 9

Stakeholder need	Requirement(s)	Constraint(s)	Standard clause(s)	Design feature(s)	Test ID(s)	Result section
transparent transaction records.				Trade events. EMS logs every transaction hash to CSV.		
Protecting expensive equipment from overcurrent. Lab safety requires protection against abnormal conditions.	REQ-10	IEEE 1547 Clause 6 (Appendix B)	IEEE 1547 §4, §6	Overcurrent detection system and circuit breakers installed in microgrid	T8	Section 9
System must run reliably on Raspberry Pi 5 without CPU overload, leaving room for future scaling.	REQ-11	CPU load $\leq 70\%$ (Appendix B)	ISO 27001 A.8.6	The EMS script uses <code>time.sleep()</code> between polling cycles to release the CPU. During operation, the Linux <code>top</code> utility logs CPU usage every 5 seconds for 10 minutes. The average load is computed from the log.	T9	Section 9

Stakeholder need	Requirement(s)	Constraint(s)	Standard clause(s)	Design feature(s)	Test ID(s)	Result section
Prosumers and consumers need fair, accurate energy measurements for trading.	REQ-1, REQ-2, REQ-3	Accuracy $\geq$ 95% (from Appendix B)	IEEE 2030.5 §8	Python script reads smart meter data via Modbus RTU; Node-RED logs to CSV	T1	Section 9
Pricing must reflect real-time supply-demand conditions using Kazakhstan tariffs as bounds.	REQ-5	Price bounded by 30.56/22.68 KZT/kWh	economic constraint	SDR formula; linear price interpolation between tariff bounds	T3	Section 9
Market must match buyers and sellers fairly based on bids and asks.	REQ-7	No matched trade exceeds supply/demand	economic constraint	Matching algorithm comparing bids and asks	T5	Section 9
Microgrid must balance energy locally before importing/exporting from main grid.	REQ-4, REQ-7	Net energy shall be equal to 0 for balanced operation	IEEE 1547 §4	SDR calculation; surplus/deficit detection logic	T2, T5	Section 9

3.3 Derived, Interface, Environmental, and Regulatory Requirements

Derived Requirements.

- The Node.js simulation shall read the 15-household CSV file and analyze hourly energy data for each household, calculating individual net energy.
 - Why needed: REQ-4 requires accurate sum of total supply and total demand. This requires correctly analyzing the input dataset and computing net energy per household before summation.
 - Supports: REQ-1, REQ-4;
- The EMS shall compute SDR using the formula: $SDR = \text{Total Supply} / \text{Total Demand}$, where both values are in kWh.

- Why needed: REQ-5 requires SDR-based pricing. The SDR calculation must be explicitly defined and protected against division-by-zero math rules.
 - Supports: REQ-5
- The blockchain smart contract shall emit a unique TransactionLogged event containing transaction hash, block number, timestamp, and cleared energy amount for every successful market-clearing process.
 - Why needed: REQ-8 requires that every transaction produces a hash and block number. The transmission of an event from a smart contract provides a verifiable record in the blockchain.
 - Supports: REQ-8
- The Raspberry Pi 5 shall poll each smart meter via Modbus RTU at intervals of 5 seconds or less.
 - Why needed: Ensures sufficient data freshness for accuracy comparison while preventing excessive CPU load.
 - Supports: REQ-1, REQ-3, REQ-11;

Interface Requirements.

- EMS log file output shall use CSV format with headers for compatibility with analysis tools.
 - Direction: EMS -> file system (Raspberry Pi 5 storage)
 - Protocol / signal type: CSV (comma-separated values) with headers \
 - Why needed: CSV format is human-readable, can be opened in Excel for manual verification, and can be parsed by Python scripts for automated comparison.
 - Supports: REQ-1, REQ-2, REQ-3
- Smart meter to Raspberry Pi 5 data transfer shall use Modbus RTU over RS-485.
 - Direction: Smart meter Modbus port -> RPi5 UART/RS-485 shield
 - Protocol / signal type: Modbus RTU
 - Why needed: The smart meter measures voltage, current, and other values, then makes these values available via Modbus registers. The Raspberry Pi 5 reads these registers to obtain the data required for REQ-1 accuracy testing and EMS decision-making.
 - Supports: REQ-1, REQ-2, REQ-3
- The Node.js simulation shall output calculated SDR, internal price, and matched transaction data to a JSON file that is readable by the web dashboard.
 - Direction: Simulation -> File system -> Dashboard
 - Protocol / signal type: JSON format
 - Why needed: R4, R5, R6, R8 require data flow from simulation to visualization. JSON is a standard, parseable format.
 - Supports: REQ-4, REQ-5, REQ-6, REQ-8

Environmental Requirements

- The smart meter shall receive AC power within its specified input range.
 - Why needed: If the supply voltage is too low or too high, the meter's current and power measurements become inaccurate.
 - Supports: REQ-1

- Ambient temperature for Raspberry Pi 5 and smart meters shall be less than 50°C.
 - Why needed: Raspberry Pi 5 begins to throttle its CPU clock speed when the temperature exceeds approximately 80°C. Throttling reduces performance, which could increase CPU load percentage artificially
 - Supports: REQ-11

Regulatory and Safety Requirements

- The blockchain testnet (Ganache) shall not store real energy transaction data from actual households. Only simulated or anonymized data shall be used.
 - Why needed: Even though the project does not use real household data, this requirement establishes ethical boundaries for any future extension.
 - Supports: REQ-8
- Overcurrent testing shall use software injection only. No physical overcurrent shall be applied
 - Why needed: REQ-10 is verified using a simulated fault, which tests the detection logic without risk.
 - Supports: REQ-10
- The blockchain testnet shall operate in an isolated network environment
 - Why needed: ISO 27001 A.5.15 requires that only authorized users can access systems. An isolated testnet prevents unauthorized access, protecting the integrity of verification.
 - Supports: STD - 4

4. Engineering Standards, Regulations, and Compliance Evidence

Table 4-A. Standards Application Matrix.

ID	Standard / regulation	Clause used	Why applicable	Design decision affected	Verification / inspection method	Evidence produced	Compliance status
STD-1	IEEE 2030.5-2023 – Smart Energy Profile Application Protocol	Section 6 (Device Discovery and Registration); Section 7 (Secure Communication); Section 8 (Smart Energy Data Models)	This standard defines how smart meters should structure and communicate energy data. It is applicable because the laboratory smart meters support Modbus RTU and the EMS must interpret their data correctly.	The format of Modbus data logged by the Python script follows IEEE 2030.5 data models. Only registered devices are allowed to communicate with the EMS.	Inspection of logged data packets to verify they contain ID, timestamp, units, and value per the standard format. Review of Python script to confirm only configured Modbus addresses are polled.	Evidence Evidence	Compliant
STD-2	ISO/IEC 27001:2022 – Information Security Management	Annex A.5.15 (Access Control); Annex A.8.24 (Use of Cryptography)	It is applicable because energy trading data is sensitive and blockchain transactions must be tamper-proof.	Only registered system members have permission to submit trades. Blockchain private keys are stored locally	Review of smart contract code to verify access control logic.	Smart contract source code	Compliant

				in encrypted keystores			
STD 3	IEEE 1547-2018 – Standard for Interconnection of Distributed Energy Resources	Clause 4 (General specifications and requirements); Clause 6 (Response to Area EPS abnormal conditions)	This standard defines acceptable voltage, current, and frequency ranges for distributed energy resources. It is applicable because the microgrid contains solar PV and wind generation models. Required for safe operation and fault protection.	Circuit breaker trip levels are set to values consistent with IEEE 1547 Clause 6. During normal operation, the EMS ensures voltage and current remain within Clause 4 ranges.	The AC-DC microgrid laboratory already has hardware circuit breakers pre-installed and configured by lab staff to IEEE 1547 Clause 6 specifications. Team confirmed breaker thresholds (15 A) meet standard requirements.	Visual inspection of existing circuit breakers in the microgrid panel. Confirmation from supervisor that breakers were originally specified and calibrated to IEEE 1547 Clause 6 thresholds Evidence Evidence	Compliant
STD -4	ISO/IEC 27001:2022 – Information Security Management (Capacity Management)	Annex A.8.6 (Capacity Management)	It is applicable because the Raspberry Pi 5 has limited CPU resources and future scalability to more	CPU load is monitored using top -b during operation. A threshold is set to $\leq 70\%$	Run top -b for 36 minutes while EMS operates. Compute average CPU load from log.	CPU load log showing 14.55% average over 36 minutes Evidence	Compliant

			nodes is expected.				
--	--	--	-----------------------	--	--	--	--

5. Concept Generation, Trade-off Study, and Selected Concept

5.1 Candidate Concepts

To identify the architecture for this Capstone project, three candidate solutions were selected and studied by the team. They mostly differ by the processed amount of energy data by the system, and how it is then monitored. The metrics that we would use for the trade-off study consist of cost calculations, CPU load, latency, complexity, and scalability. The following paragraphs would describe the studied concepts and their way of managing EMS.

Concept 1: Baseline Python-Centric EMS Pipeline (Direct Data Processing on Raspberry Pi 5)

This concept is based on simple integration of the smart-meters with the Raspberry Pi 5 microcontroller through the industrial protocol of serial communication MODBUS. The single board computer does the fetching of the smart-meter data over MODBUS TCP/IP or RTU protocols. According to the architecture of this concept [7, Figure 1], which would be also the basis of this project, there exist several power generating parts and/or loads that are connected with the smart meters [6, 7]. The Raspberry Pi then fetches the data using the abovementioned protocol, and performs filtering, basic energy balancing, data parsing.

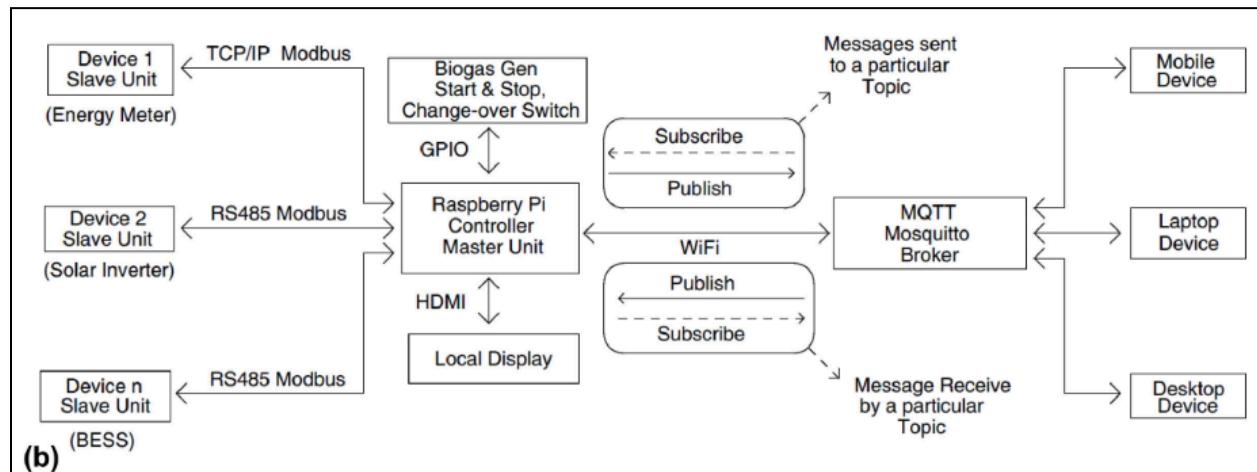


Figure 1: Basic internal flow of the IoT system

Source: Adapted from [7]

The main limitation of this concept is the restricted computational opportunities. Since with the increasing number of nodes, the processor may start increasing decision times and have problems with memory management [8].

Concept 2: Optimized Edge-Preprocessing EMS with Lightweight Smart-Contract Integration

This concept follows the same data capturing techniques described in Concept 1, but it now has more sophisticated internal processing techniques. The data collected by Raspberry Pi 5 is filtered so the collection of energy data is settlement-ready for further analysis inside the EMS

[9]-[10]. The main benefit of this concept is that the computational load is reduced and the system achieves higher quality of data. Preprocessed data can be then integrated with the smart-contracts [11]. This concept maintains the same costs, and in addition offers technically robust and secure usage in smart contracts.

Concept 3: High-Performance EMS with External Compute Offloading (Hybrid Edge Server Architecture)

In the third concept, the IoT device just acts as a data acquisition gateway, which means it has a lot less computational load on it. The data read from the smart meters is managed by the external computer or server. Such architecture allows more advanced techniques for optimization, scalability, forecasting, and market price calculations [12]. Therefore the major strength of this concept is high computational power and ability to increase the number of nodes in the network [13]. On the other hand, the system also becomes costly, more dependence on the communication protocols, and greater implementation complexity.

Table 5-A. Candidate Concepts.

Concept	Description	Strengths	Weaknesses	References / evidence
1	Baseline Python-Centric Edge EMS	East to prototype. Reduced dependence on network connectivity. Fast initial implementation.	Limited computational capabilities. Is not suitable for heavy load tasks like forecasting.	[6], [7], [8]
2	Optimized Edge-Preprocessing EMS with Lightweight Smart-Contract Integration	Preprocessing data at the edge takes off some computational load afterwards. Better balance between cost and performance.	More software design compared to basic implementation. Limitations with scalability	[9], [10], [11]
3	Hybrid Edge-Server EMS with External Compute Offloading	Suitable for advanced calculations. Better scalability and performance.	Highest cost of implementation. More dependence on stable communication between IoT and servers	[12], [13]

5.2 Weighted Trade-off Study

Table 5-B. Weighted Trade-off Matrix

Criterion	Metric / target	Weight	Concept 1	Concept 2	Concept 3	Evidence source
Hardware cost	$\leq 300,000$ KZT	0.25	9	8	4	[6]

CPU Load	≤ 60 percent under full EMS load	0.20	6	9	10	[12]
Latency	≤ 300 ms EMS processing delay	0.20	6	9	10	[10]
Complexity	Prefer simple integration	0.15	9	7	4	[7]
Scalability	Support ≥ 10 active nodes	0.20	5	8	10	[13]
Total score		1.00	0.7	0.82	0.74	

According to the matrix, it can be concluded that Concept 2 is the better scenario for the present Capstone project, due to its balance between cost and implementation. Since the Hybrid AC/DC Microgrid lab for our project has limitations like scalability, this concept was chosen. It outweighed the other concepts with a total score of 0.82 out of 1.00.

5.3 Selected Concept and Rationale

Our team selected Concept 2 for several reasons. First of all, it scored right between implementation complexity and performance evaluation. It means that we have accepted moderately harder implementation compared to basic Python-centric design where calculation algorithms would be done outside of edge devices. Secondly, we rejected the third option due to its higher costs, which in our opinion should be realized on an industrial level. Additionally, some trade-offs included the use of lightweight optimization techniques, rather than developing heavy models for forecasting, price calculation and so on [13]. The integrity of the system is backed up by blockchain integration to improve security and transparency, data integrity of the microgrid [10].

In this architecture, we selected Raspberry Pi 5 as a central edge computing device because it is low-cost, low-power, has built-in networking capabilities, can run Linux with libraries such as pymodbus and Node-RED, and has sufficient processing power. Other options were also dismissed: A cloud VM or remote server would add latency, recurrent expense, and reliance on internet connectivity; a laptop or desktop PC would not fit into the field because of power and size; an Arduino or STM32 microcontroller would not have the memory and Linux support to execute the complete software stack; To communicate between the smart meters and Raspberry Pi 5, we have chosen Modbus RTU over RS-485 due to its status as an industrial

standard which is supported by our laboratory meters, provides a reliable long distance noise-immune connection, follows a simple master-slave architecture with unique slave identities, imposes minimal CPU load and predictable latency, and only a low-cost USB-to-RS485 adapter is required. The alternatives were considered and rejected: Modbus TCP/IP was not compatible with our meters (they do not have Ethernet ports). CAN bus would need extra translation devices. Wi-Fi or MQTT directly off the meters was not a possibility since the meters are not wireless.

6. Design Iterations and Decision History

At the beginning of the project, we studied the MODBUS protocol. Since we are using Python code to fetch the data, we had to choose the library that we will work with. At the beginning “pymodbus” was chosen as a full MOSBUS protocol implementation, and the first tests with smart meters were written with it. However, when it was time to increase the number of smart meters, to ensure stable reading we shifted to the “minimalmodbus” library. This decision was based on some precautions of scalability issues that may happen, whereas “minimalmodbus” did work great on some of the examples that we have studied to learn about it. However, the new library was not reading the registers of the smart meters in the way that we could modify them. Therefore we returned to the original library that we chose at the beginning. Even though we changed the library for the code, the scalability was still under attention and we decided to implement the “try and error” strategy. Fortunately, connecting several slaves to one master node was done faster than we predicted. Thus, the register reading from several devices was mastered by us. Moreover, we have made multiple iterations in the stages when processing of data and system integration were required. In the first steps of the project, the data was handled manually, but later it was replaced by the automated CSV-based processing method using Node.js (VS Code). This was executed in order to enable aggregation of demand/supply information of 15 houses. Also, a system which used to be a simple software- based simulation transformed into a blockchain-integrated architecture with smart contracts written on Solidity programming language were connected with the blockchain testing environment (Ganache) to enable transparent transactions record.

Table 6-A. Major Design Iterations

Iteration / version	Issue discovered	Alternatives considered	Decision made	Trade-off accepted	Evidence of improvement
1	MODBUS library update	Different library for MODBUS protocol	Pymodbus was chosen	Daniyal rewrote the initial code for data fetching	The data is read from the smart meter
2	Restricted access to three phase source	Modify the functions for easy integration with three phase sources in future	For the testing stage the smart meters were configured for one phase input	No trade-offs	Two functions were created, for 1P and 3P measurements
3	Inefficiency due to manual data handling	Manual vs automated processing of data	Created CSV + Node.js processing	Extra efforts related to additional coding	Automated aggregation of the data related to supply and demand of 15 houses

Iteration / version	Issue discovered	Alternatives considered	Decision made	Trade-off accepted	Evidence of improvement
4	Transparency of transactions lack	Blockchain system vs centralized one	Implemented Solidity and Ganache	Increased system complexity and gas costs	Records of blockchain blocks having information in the form of hashes
5	Limitations in system visibility	Dashboard vs raw outputs of data	Developed dashboard that showcases results	Developed UI	Real time display of data on a local server

7. Final System Architecture

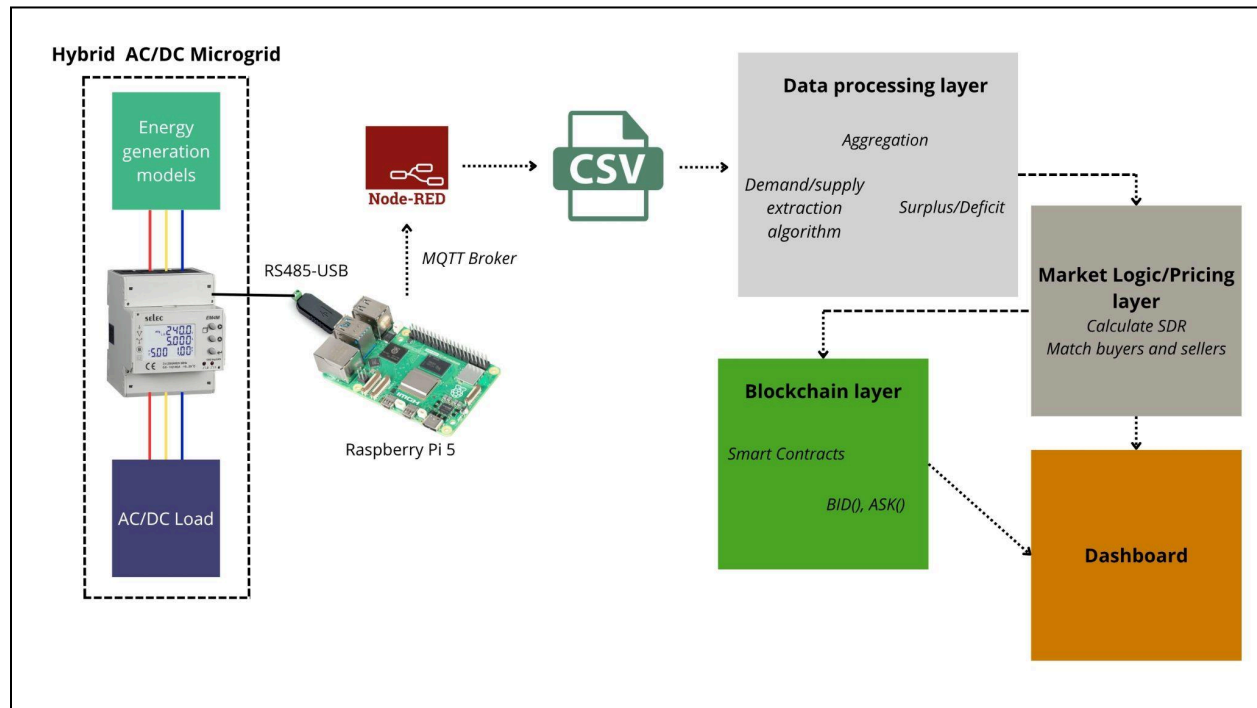


Figure 2: Final System Block Diagram.

Table 7-A. Interface Summary.

Interface	Direction	Protocol / signal type	Rate / timing	Voltage / power	Owner subsystem *	Notes
Load / AC line → Smart Meter	Power line to measurement input	AC voltage/current sensing	Continuous measurement	220–240 V AC or 380-415 AC	Smart Meter Subsystem	Measuring energy
Smart Meter → RS485-to-USB Converter	Output from meter communication port	Modbus RTU over RS-485	Continuous measurement	low-power	Smart Meter Communication Subsystem	N/A
RS485-to-USB Converter → Raspberry Pi 5	Serial communication input to RPi	USB serial (UART via USB adapter)	Continuous measurement	low-power	Smart Meter Communication Subsystem	Reading registers of smart meter
Raspberry Pi 5 → MQTT Broker	Publish output	MQTT over TCP/IP	Every polling interval (5s)	5 V DC	Data Acquisition Subsystem	Publishing inside topics
MQTT Broker → Node-RED	Subscription input	MQTT over TCP/IP	Every polling interval (5s)	5 V DC	Data Acquisition Subsystem	Subscribing to topics
Node-RED → CSV File Storage	Output from Node-RED flow	File write / CSV logging	Variable polling intervals (hourly)	5 V DC	Data Acquisition Subsystem	Formatting and polling, preprocessing

Smart meter CSV files -> Data Processing level	Input to Node.js	File I/O, CSV parsing	Hourly	N/A	Data Processing layer	Reads house energy data
CSV Parser → Aggregation / State Detection	Internal	In-memory JavaScript objects / function calls	Every processing cycle	N/A	Data Processing Layer	Extracts demand, supply, and determines surplus/deficit
Data Processing Layer → Market Logic Layer	Output from Node.js processing	Structured JSON / JS object	Once per market interval	N/A	Market Logic & Pricing	Passes totalDemand, totalSupply, transactions, marketPrice inputs
Market Logic Layer → Smart Contract	Output to blockchain	Web3 / JSON-RPC over HTTP or WebSocket	Per BID / ASK / CLEAR_MARKET operation	N/A	Blockchain Layer	Sends transactions to Solidity smart contract on Ganache
Smart Contract → Ganache Blockchain State	Internal blockchain write	EVM transaction / event log	On every blockchain transaction	N/A	Blockchain Layer	Stores transaction hash, block number, gas, and participant address
Blockchain Layer → Dashboard	Output	Web3 event query / JSON-RPC response	Event-driven or polling	N/A	Visualization Layer	Used to show transaction history and blockchain activity
Dashboard → User	Output	Web-based graphical interface	Real-time / refresh based	N/A	Visualization Layer	Final user-facing monitoring interface

Timing / latency budget:

The overall delay is mainly set by the user, since we do not want to overload the system with tons of data, it is reasonable to accumulate data by adding it to the CSV log with intervals. Thus the daily energy pattern would be easier for the data processing layer to work on.

Power / resource budget:

Since smart-meters are powered by the bus, they do not consume any extra power, the only power consumption inside the system would be the Raspberry Pi 5 that would work continuously.

8. Detailed Design and Implementation

8.1 Hardware design

1. Hardware Components

Smart Meters

Two different Modbus RTU-enabled smart meters were used: single-phase meters and three-phase meters. These meters were selected as they are compatible with Modbus RTU.

Raspberry Pi 5

Raspberry Pi 5 was chosen as the main data acquisition and processing unit because of its low price, low power consumption, inbuilt networking and compatibility with Python libraries to communicate via Modbus. It is an operating system based on Linux and it is a host of MQTT broker, Node-RED instance, and Python scripts of data acquisition. The Raspberry Pi 5 is connected to the smart meters through a USB-to-RS485 adapter that transforms the USB signals into the differential RS-485 standard used by long-range industrial communication.

USB-to-RS485 Adapter

This adapter is necessary as Raspberry Pi 5 lacks RS-485 ports. The adapter makes use of FTDI chipset and is well-supported in Linux and has no extra drivers needed.

AC-DC Microgrid

The microgrid is a hybrid AC-DC simulator that locates in the Nazarbayev University C4 lab. It has solar PV and wind turbine generation models, programmable loads, and circuit breakers to protect overcurrent. Laboratory employees had already constructed and calibrated the microgrid. The team did not alter the microgrid itself, but wired smart meters to its output lines.

2. Hardware Wiring and Connections

Smart Meter Wiring to Microgrid

One three-phase smart meter was connected to the AC-DC microgrid by terminating three phase conductors from the bottom side of the meter to the corresponding R, Y, and B phase terminals on the microgrid output.

RS-485 Communication Bus and Meter Interconnection

The three-phase smart meters were daisy-chained together with twisted-pair shielded cable, which is regular arrangement of RS-485 networks. The first meter A+ and B- terminals were connected to the corresponding terminals of the second meter, then the third meter, and so on, forming a continuous communication bus, which connected all the meters in series. This daisy-chain implementation reduces reflections of signals and provides reliable data transmission all over the network. The last meter in the chain was linked back to the Raspberry Pi 5 with a USB-to-RS485 adapter attached to the Raspberry Pi USB port.

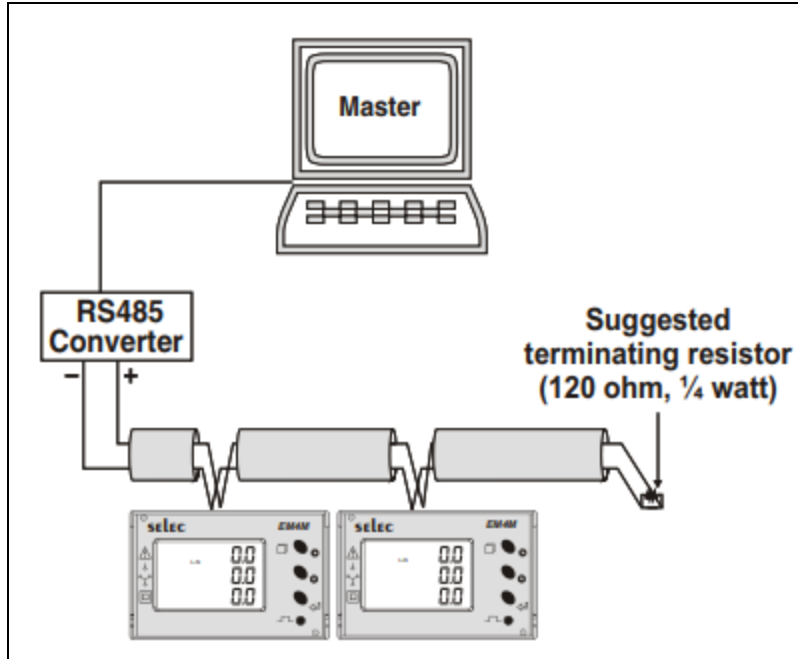


Figure 3: Connection diagram for Communication between Raspberry Pi 5 and smart-meters

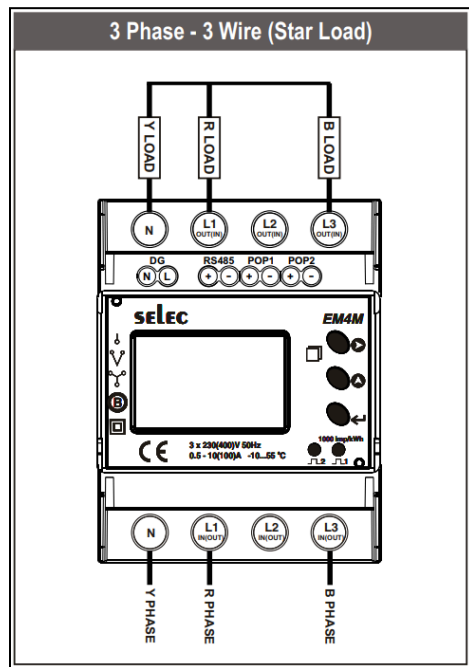


Figure 4: Three wire connection on smart meter

3. Communication Protocol and Data Flow

The hardware communication architecture is based on the Modbus RTU protocol over RS-485. The Raspberry Pi 5 acts as the Modbus master, and each smart meter acts as a slave device. All communication is started by the master that sends a request frame with the target slave ID, function code, register address and number of registers to read. The slave replies with a data frame which has the requested register values.

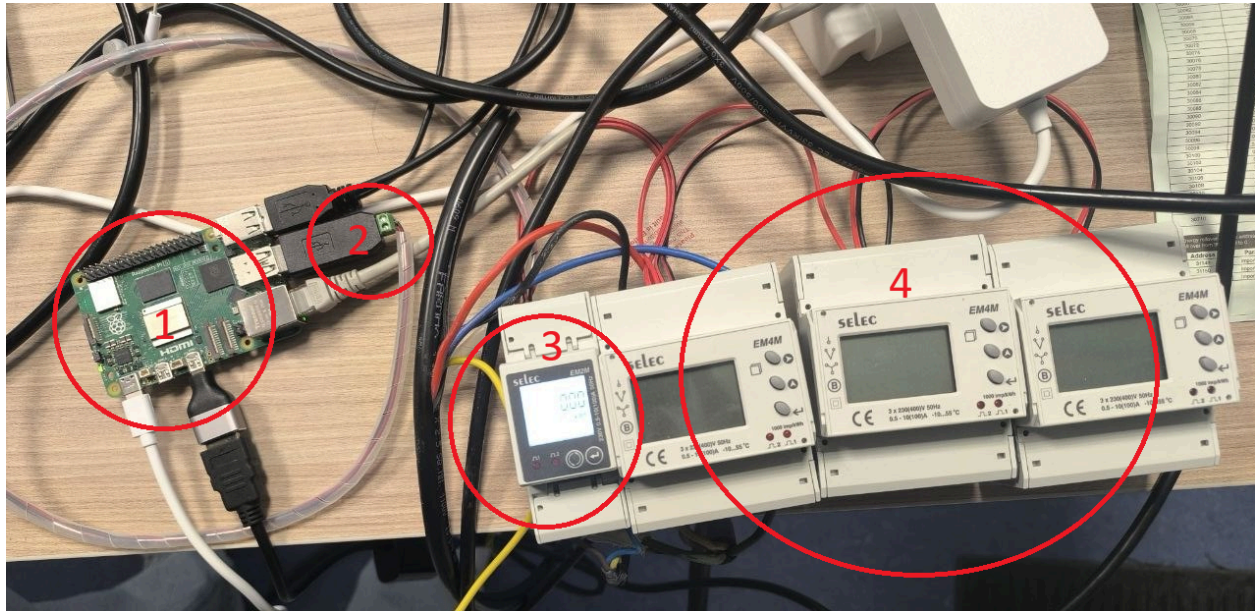


Figure 5: Hardware System with Components(1- Raspberry Pi 5, 2-USB-to-RS485 Adapter, 3 - single-phase smart meter, 4 - three-phase smart meter)

8.2 Software / firmware design

1. Data acquisition layer (Modbus RTU & Node-Red)

On the field side, the Raspberry Pi 5 communicates with the smart meters over a Modbus RTU serial bus mentioned above. On the software level of the microcontroller, it publishes the measurements through an MQTT broker, allowing nodes in localhost to subscribe to the data, display it in the UI, and store in CSV. In this architecture, the industrial level communication layer is connected to the higher level user interface and storage layer, and overall making the system simple and robust. The implemented Python code creates a Modbus serial client on the “/dev/ttyUSB0” serial port with optional parameters like baudrate, parity bits, bytesize, timeouts and so on, while the MQTT client connects to a local broker on “localhost:1883”.

The structure of the Python code is built so that each meter can be read using the same acquisition function. The only difference would be in the register maps for one phase and three phase smart meters, since they differ according to the instruction manuals. Overall, the mapping looks for active and reactive power, power factor, frequency, and line to line voltages and currents in three phase meters. The acquisition function is called “*read_meter_data()*” and has parameters like ModbusSerialClient, slave ID and the register mapping. This function stores all the data fetched from the smart meter in one variable, in the form of a Python dictionary.

Example code:

```
meter1_data = read_meter_data(client, 1, SINGLE_PHASE_MEASUREMENTS)
```

In this sample of the Python code, “*client*” stands for the ModbusSerialClient client interface class, “*I*” is the slave ID of the device (in this case it is the slave ID of the smart meter that was configured on the device itself), “*SINGLE_PHASE_MEASUREMENTS*” is the register mapping dictionary that stores the register addresses and their names.

After calling this function, the output would be as in the following format:

```
meter1_data = {"voltage": 228.4, "current": 3.12, ... , "frequency": 50.0}.
```

For correct interpretation of these numerical values, two functions had to be written, They are called “*read_float_swapped()*” and “*read_float()*”. These functions read the specified registers using the pymodbus function “*read_input_registers()*” that takes in the first parameter the address of the register from which it needs to start reading, in the second parameter number of registers to read, and in the third parameter the slave ID of the device. From the documentation of the single phase smart meter, it is shown that the float number we want is stored in the swapped word order. For instance, “*read_input_registers()*” reads two registers (r0, r1) from single phase smart meter. Now, to interpret these registers into correct float number, “*read_float_swapped()*” is called. Thus, the two registers are now in the order like (r1, r0), and physically correct values can be shown.

Since all meter data is stored in the allocated Python dictionary, it can be now published using the MQTT broker. The set up of the MQTT client in the Python code includes the “*paho.mqtt.client*” library, host IP address, port number, client ID, username, and password. Using these variables, “*client_mqtt*” is created and the messages can be now published under some topics. The main function of the MQTT section of our project is shown in the following code snippet:

```

def publish_single_phase_meter(client_mqtt, meter_number, meter_data, selected_keys=None):
    topic_map = {
        "voltage": f"rpi/broadcast/METER{meter_number}_VOLTAGE",
        "current": f"rpi/broadcast/METER{meter_number}_CURRENT",
        "active_power": f"rpi/broadcast/METER{meter_number}_ACTIVEPOWER",
        "reactive_power": f"rpi/broadcast/METER{meter_number}_REACTIVEPOWER",
        "power_factor": f"rpi/broadcast/METER{meter_number}_POWERFACTOR",
        "frequency": f"rpi/broadcast/METER{meter_number}_FREQUENCY",
    }
    if selected_keys is None:
        selected_keys = topic_map.keys()
    for key in selected_keys:
        if key in topic_map:
            publish_value(client_mqtt, topic_map[key], meter_data.get(key))

```

Figure 6: publish_single_phase_meter() function definition

This function takes on parameters like “*meter_data*” and “*selected_keys*”, where former is the fetched data stored in the Python dictionary, and selected keys are the physical variables that we would like to publish. This means that the code can be edited to publish only the selected data and ease the preprocessing stage. The “*topic_map*” shown in above snippet gives the information on what topic the MQTT messages are published and to which topics Node Red should subscribe to acquire smart meter data. From Figure 8.8, the purple nodes inside the interface of the Node Red are the mqtt_in nodes. Inside them we specify the server (localhost), and the topic in the format “rpi/broadcast/METER1_VOLTAGE”. By doing so, new flow is created to fetch the data from the local server and output it on UI of Node Red or store inside the .csv file.

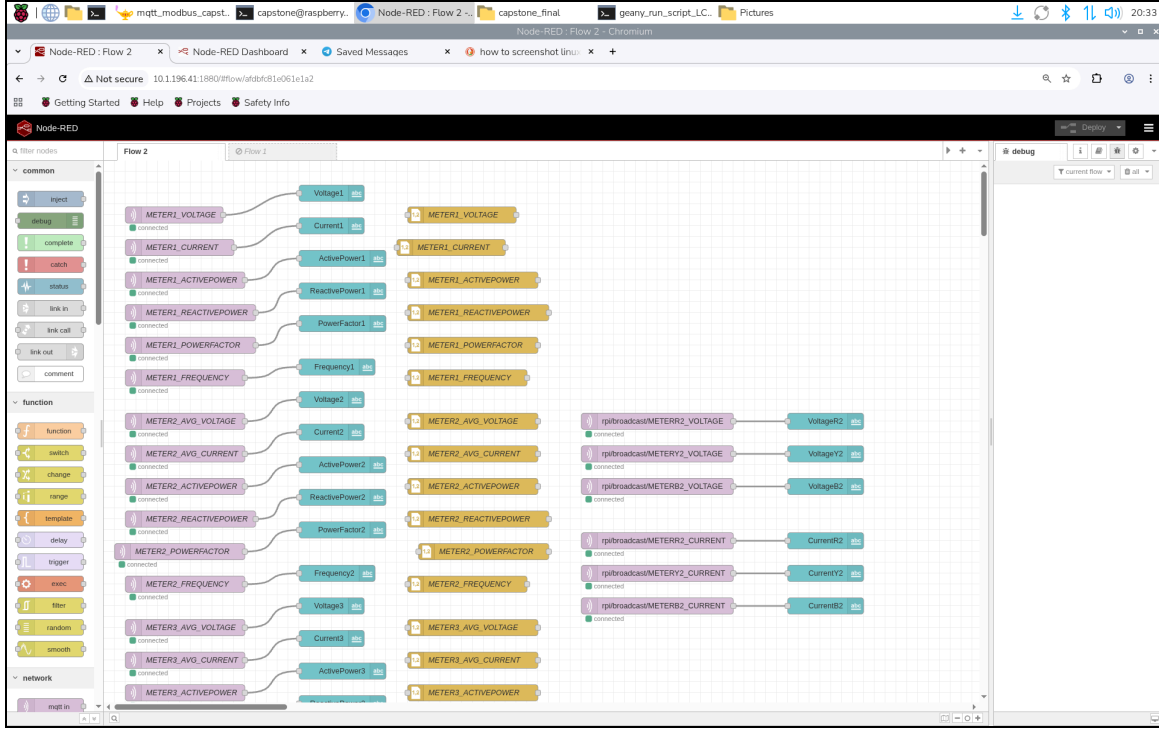


Figure 7: Node Red flows (mqtt_in nodes and text_ui nodes)

2. Data processing & Simulation layer

The CSV datasets are processed by a JavaScript-based simulation (Node.js in Visual Studio Code). A new term agent is used to describe each house, which is either a prosumer or a consumer. The system is able to calculate the following parameters:

1. Total energy demand
2. Total energy supply
3. Net energy
4. Supply-Demand Ratio (SDR)
5. Internal market price

The SDR is calculated by the following formula, which helps to dynamically adjust pricing within the grid:

$$SDR = \frac{TOTAL\ SUPPLY}{TOTAL\ DEMAND}$$

3. Blockchain layer

Ganache Software is used as a local blockchain environment to simulate decentralized energy transactions between grid users. Smart contracts created in Solidity programming language manage:

- Bid submissions (buyers)
- Ask submissions (sellers)
- Market clearing

Moreover, transaction hashes, block numbers and gas usage estimates are all generated with each transaction.

4. Visualization layer

A local server is launched as a dashboard displaying:

- Hourly energy data
- Market status
- Pricing data
- Blockchain activity in the form of transactions, hashes, blocks

8.3 Mechanical / packaging design (if applicable)

This project did not involve mechanical / packaging design. However, standard mounting and wiring practices are used to arrange hardware elements within the lab. To maintain safety of the system proper cable management was followed.

8.4 Prototype realization and integration

An incremented approach was used to develop the system in the following steps:

1. Data acquisition from smart meters using Modbus Protocol and Raspberry Pi 5
2. Node-RED and CSV storage usage to create data pipeline
3. 15 households real data simulation using Node.js
4. Bids, asks and matching are implemented as components of energy trading logic
5. Ganache + Solidity smart contracts integration
6. Visualization dashboard creation

Such a step-by-step approach allowed for increased system stability. Simulated data validated the logic before integrating real smart meter data. The final system is operating based on the following pipeline:

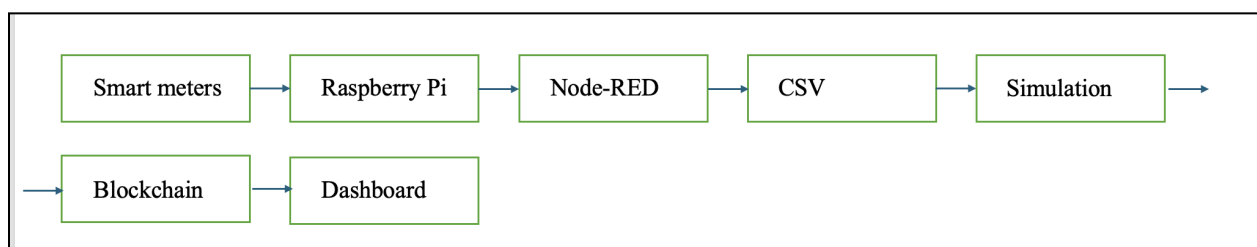


Figure 8: System Pipeline

8.5 Final bill of materials and design-for-manufacture / assembly notes

A combination of hardware and software elements are used to acquire real-time data, implement blockchain-based energy trading and process data.

Bill of Materials Summary:

Modbus-enabled smart energy meters, a Raspberry Pi 5 are main hardware components used to fetch data and create communication interfaces. Those components have high reliability, compatibility with industrial standards, and are efficient regarding the cost. All software applications that were used are open source and free of charge. Hence, it has substantially decreased the net project cost while providing greater flexibility and scalability.

9. Verification, Validation and Experimental Results

Table 9-A. Requirements-to-Test Results Matrix.

Req ID	Requirement	Test ID	Verification method	Equipment / tools and accuracy	Target	Measured result	Verdict	Evidence ID
R1	Modbus RTU communication with smart meters	T1	Execute Python script polling each meter's register	Raspberry Pi 5, Modbus RTU, Python	100% read success rate	100% accurate values from smart meters(V and freq)	Pass	Evidence1
R2	CSV logging of Modbus readings	T1	Run polling script for 10 min, inspect CSV file	Raspberry Pi 5, Node-RED, CSV	100% of expected data rows logged	100% of data added to CSV files	Pass	Evidence2
R3	Smart meter measurement accuracy	T1	Compare smart meter vs bench multimeter readings	Smart meter, Raspberry Pi 5, Modbus RTU, Python	Average error $\leq$ 5%	Error = 1.33%	Pass	Evidence3
R4	Accurate aggregation of energy	T2	Total supply vs total demand	CSV & Node.js	Totals are correct	Values are matched with the dataset	Pass	Evidence4
R5	Accurate work of SDR pricing	T3	SDR vs price relation	Simulation & Dashboard	Higher price when SDR is less than 1	Correct behaviour is observed	Pass	Evidence5
R6	Recording of	T4	Examine transaction logs	Ganache	Creation of	Hashes and block	Pass	Evidence6

	blockchain transaction				transactions	numbers are shown		
R7	Market clearance logic is correct	T5	Bids, asks, clear market comparison	Simulation	Matching is correct	Showing of matched energy	Pass	Evidence7
R8	Visualization of the dashboard	T6	UI inspection	Dashboard	Display of data is correct	All values are visible	Pass	Evidence8
R9	Logic of grid interaction is correct	T7	Surplus/deficit handling	Simulation	Correct import and export	Grid energy is consistent	Pass	Evidence9
R10	Circuit breaker overcurrent protection	T8	Supervisor verification of 15 A threshold per datasheet	Circuit breaker, microgrid	Trip at 15 A when exceeded	No experiments executed due to safety rules(Confirmed by supervisor based on CB datasheet)	Pass	Evidence10 Evidence11
R11	Raspberry Pi 5 CPU load during EMS	T9	Run top logging while system operates at full load	Raspberry Pi 5	$\leq 70\%$ average over 10 minutes	14.55% load over 36 minutes	Pass	Evidence12

Table 9-B. Test Campaign Summary.

Test / experiment	Date	Owner(s)	Setup	No. of trials	Key result	Observed issues	Follow-up action
Acquisition of data via smart meters	March 2025	Daniyal	Smart meters + Pi Raspberry 5 + Modbus RTU	5	Successful data collection	Noise in readings occurred occasionally	Applied filtering
Validation of data processing	March 2025	Yernar	CSV files + VS codes	10	Demand/supply calculation is accurate	Error of initial mismatch	Aggregation logic fixed
Test of market algorithm	March 2025	Damir	Simulation (15 houses data)	10	Matching and pricing are correct	No significant issues	Stability is verified
Test of blockchain transaction	April 2025	Damir	Solidity smart contracts + Ganache	8	Hashes are recorded with transactions	Gas variability	Execution is optimized
Test of dashboard visualization	April 2025	Team	Web interface	6	Correct display of data	UI occasional issues	Layout is improved
Integration of the full system	April 2025	Team	End-to-end pipeline	5	System works properly	Sync delays	Timing is adjusted

10. Risk Management, Schedule, Budget, and Procurement Outcomes

10.1 Final Risk Register

Table 10-A. Risk Register with Outcomes.

ID	Risk	L	I	RPN	Mitigation	Did it occur?	How handled	Residual risk / limitation
1	Smart meter Modbus communication instability	3	4	12	Perform meter calibration before testing or add filtering in EMS preprocessing	No	Prevented by careful Modbus configuration and proper wiring. Moreover, regular validation against bench multimeter ensured data integrity	Some minor noises still present but within acceptable accuracy threshold
2	Raspberry Pi 5 CPU overload	2	4	8	Optimized Python scripts and reduce polling frequency. Also always monitored CPU temperature	No	Implemented efficient polling loops with delays	Scaling to more nodes may increase load and require more optimization or update for new computer model
3	Incorrect register reading due to Modbus library selection	3	4	12	Tested both libraries early with a single smart meter. More importantly, read manufacturer documentation for register mapping	Yes	Initially chose pymodbus, switched to minimalmodbus for scalability concerns, then returned back for pymodbus after discovering minimalmodbus could not	Library switching cost development time but did not block final implementation

ID	Risk	L	I	RPN	Mitigation	Did it occur?	How handled	Residual risk / limitation
							modify registers.	
4	Complexity of blockchain integration	4	4	16	Developed using a test network such as Ganache, before using real network such as Polygon	yes	Implemented in an incremental way by creating bids, asks, clear_market in the order and not simultaneously	Requires more effective optimization algorithms when scaling to real blockchain
5	Energy balancing being incorrect	4	5	20	Debugging & Validation processes	yes	Carefully verified and fixed calculation logic in the dashboard	Handling may still be required for edge cases
6	Integration of the system (backend & dashboard) issues	3	4	12	Designed in a modular way, with step-by-step approach	yes	Debugged server.js, ensuring correct data flow	High dependent of UI on backend stability
7	High gas cost	2	4	8	Simulated and optimized gas usage	no	Used local blockchain with test values, not real currency	High costs when deployed in a real network

10.2 Milestones Planned vs. Actual

Table 10-B. Milestones and Completion Evidence.

Milestone	Planned date	Actual date	Owner	Exit criteria	Status	Evidence
Smart meter wiring and physical setup	January 2026	February 2026	Yernar	1. Configured slave IDs 2. Established RS-485 connections 3. Initially powered smart meters with single-phase 4. Smart meters correctly wired to AC-DC microgrid	Completed	Photos of wired smart meters Evidence
Modbus RTU communication establishment	January 2026	March 2026	Daniyal	Successful register read of voltage and frequency from single-phase meter	Completed	Python script output showing register values Evidence
Python script for register reading	February 2026	April 2026	Daniyal & Yernar	Python code reads power, voltage, current registers from 1-phase and 3-phase meters and publish it	Completed	Python script output showing register values Evidence
Node-RED flow creation and CSV storage	March 2026	April 2026	Team	Python script publishes Modbus data to MQTT, which then goes to CSV file with timestamps using Node-RED	Completed	CSV file Evidence
Simulation and data preprocessing (Visual Studio)	January 2026	March 2026	Team	Average (aggregated) supplies and demands of households calculated accurately	Completed	Result.json working simulation. Evidence .

Milestone	Planned date	Actual date	Owner	Exit criteria	Status	Evidence
Integration of blockchain (Solidity and Ganache)	February 2026	March 2026	Damir	Transactions recorded for bids, asks and market clearing	Completed	Transaction logs, gas Evidence .
Implementation of market logic (SDR, market clearance price, matching)	March 2026	March 2026	Damir	Correct matching of energy and pricing	Completed	Pricing, SDR on the dashboard Evidence .
Visualization	March 2026	April 2026	Damir	UI showing data at an hourly rate	Completed	Website, local host Evidence .

10.3 Budget, Procurement, and Substitutions

Table 10-C. Bill of Materials and Procurement Summary.

Part No.	Description	Qty	Unit cost	Total	Alt. part / substitution	Supplier / lead time	Impact of substitution
1	Smart Meter 1 phase	1	11572	11572	Simulation dataset	Imported from India (Amazon), 2 months	Not realistic data if replaced with simulated one.
2	Pi Raspberry 5	1	89390	89390	Laptop based processing	Local Kaspi Supplier, 1 week	Real time deployment having less flexibility
3	Smart Meter 3 phase	3	27657	82971	Simulation dataset	Imported from India (Amazon), 2 months	Not realistic data if replaced with simulated one.
4	Poster	1	3200	3200	Online Presentation	Local Service Provider	Less engagement

Part No.	Description	Qty	Unit cost	Total	Alt. part / substitution	Supplier / lead time	Impact of substitution
5	Development Software (Solidity and Ganache)	-	0	0	-	Open source	No major impact
6	Node-Red, Python Environment	-	0	0	-	Open source	No major impact
7	Web Dashboard (HTML, CSS, JS)	-	0	0	-	Open source	No major impact
8	Lab Microgrid	1	0	0	-	Nazarbayev University C4 lab	No major impact

Budget status vs. plan:

The project met the expected sum and even came in under the stated budget, with biggest expenditures being hardware elements such as smart meters and Pi Raspberry 5. All the software tools used within the project were open source and hence free of charge. However, it is important to mention a couple of delays due to an international delivery.

Change-control summary:

Due to above mentioned delays in smart meter delivery, almost all milestones have been shifted accordingly to be executed later in the timeline. Thus, in the beginning only simulated CSV data sets were analyzed. Once the equipment was delivered, we were able to fetch and analyze real data without major redesign.

11. Professional Responsibilities and Broader Impacts

11.1 Public Health, Safety, and Welfare

Since this project is based in the laboratory with an AC/DC microgrid system, it represents real electrical safety hazards. The main sources of hazards include electric shock from active or unprotected components, short-circuit currents because of incorrect wiring, overheating of the active loads. To take all this safety issues under control, all experiments are conducted under laboratory safety rules and in the presence of the supervisor. Moreover, each student of the Nazarbayev University must undergo laboratory safety training, which is why each team member is familiar with all the hazards inside the laboratory and knows how to act when facing one. A pre-experiment safety checklist is used before turning on the microgrid, including the inspection of cable insulations, connector integrity, grounding continuity, and verification that voltages, currents and powers remain within the identified limits. After each use of the microgrid, we make sure that all the capacitive elements are fully discharged by using multimeters. Personal protective equipment, including insulated gloves and shoes, safety glasses, is used during tests, and the microgrid system is isolated for discharging. To sum up, with all residual risks remaining due to live electrical equipment, thus it is the responsibility of each member to strictly follow all the necessary rules and regulations inside the laboratory.

11.2 Ethical and Professional Responsibilities

Ethical responsibility of this project finds its foundation in honest reporting of the results, careful handling of project data and instructions, responsible engineering judgement during design and testing. Even though the energy management framework presented in this project does not handle personal data of the potential users, their energy-usage patterns and transaction records saved in the blockchain to ensure integrity and prices inside the market can only be accessed by authorized people. Smart contracts are used to improve transparency and traceability, and for our project we used open access data collection of 15 houses to create our algorithms. This way, we have achieved the design goals of our project without using sensible data. For the present report and overall project, academic integrity must be followed, which means that all simulations, code, AI-generated text, and external tools are reported and used only as learning mechanisms to support engineering decisions. Therefore, any use of AI or software assistance in our project is disclosed, and all of the claims are verified among the team members. Professional judgement is an essential part of interpreting results of the data acquisition using MODBUS protocols, validating blockchain behaviour, and when deciding whether the collected data can be used in final system evaluation.

11.3 Security and Privacy

EMS uses IoT devices to process energy measurements and publishes the trading events to a test network of a private blockchain. To avoid unauthorized documented manipulation, all data related to energy trading is saved locally on the Raspberry Pi 5 and is copied to the secure

storage where the access is provided by the team members. Smart contracts activate role-based access control, which restricts the execution of transactions by authorized nodes and prevents interference. The blockchain testnet is disconnected from public networks to minimize external attacks and maintain the evaluation data integrity. EMS logs, blockchain states, smart contract source code and configuration files are periodically backed up to avoid accidents. Security responsibilities are divided among team members, which includes verifying proper device authentication, safe key handling, and controlled access to scripts and testnet wallets.

11.4 Accessibility and Usability

The project development and documentation are being produced with a strong emphasis on accessibility for a broad audience, including future researchers, evaluators, and academic peers. Written materials such as the project report, technical diagrams, and presentation slides are currently prepared using standardized headings, labels, and structured outlines to improve readability and consistency.

Visual elements, including plots and dashboards, are being designed using color-agnostic schemes to ensure interpretability regardless of color perception. Software interfaces such as the EMS dashboard and smart-meter log outputs are going to be implemented with simple, intuitive layouts that require minimal prior technical knowledge for effective navigation. Additionally, all deliverables are being provided in standard PDF formats that support common screen readers and maintain cross-platform compatibility. These practices are actively applied to ensure that the project outputs remain inclusive, comprehensible, and usable for diverse stakeholders in both academic and industry contexts.

11.5 Environmental, Economic, Social, Global, and Cultural Factors

Environmentally, it is clear that this project is motivated by 13th goal of the UNESCO's Sustainable Development Goals - Climate Action. The purpose of this project is to show that with the development of green energy, environmental challenges can be faced by the owners of renewable energy sources. Which means that prosumers and consumers can rely less on the centralized grid that does not promote sustainable energy production by supporting each other in the P2P market of green energy. Moreover, this project showcased that data monitoring and market price evaluation allow users of the microgrid to save or earn money by sharing this energy together. By creating a local community of microgrid owners, people would be able to grow and promote a culture of green future. Globally, the idea of creating such system in remote and rural areas of the world would mean a lot to the people that live there, since they would be a lot closer to the comfort and opportunities offered by the electrical energy.

12. Team Organization, Collaboration, and Individual Contributions

Table 12-A. Team Roles and Responsibilities.

Member	Primary role	Secondary role	Major technical ownership	Leadership responsibility	Evidence ID
Daniyal	Data fetching and preprocessing using MODBUS and Node Red	Hardware implementation	Wrote the Python code for data fetching over several smart meters using MODBUS RTU. Configured MQTT for publishing smart meter data and created user interface in Node Red	Daniyal have taken the lead in designing the Python code for reading data from smartmeters, troubleshooted and debugged code from errors. Created overall architecture for data collection	Python code & Node Red and UI Evidence 3 Evidence 4 Evidence 5
Yernar	Hardware integration	Python scripting & Node-RED development	Developed a wiring system between smartmeters, Raspberry Pi 5 and AC-DC microgrid, followed by establishing RS-485 connections and configuration of slave IDs (AC-DC microgrid is not included, already been built by NU laboratory members). Next, created Node-RED flow to subscribe to MQTT topics and write incoming data to CSV files.	Yernar led a creation of hardware setup to get required values from smartmeters, which then delivers to EMS. Developed data pipeline from MQTT to CSV files using Node-RED.	Hardware & Node-RED functions Evidence Evidence Evidence
Damir	Blockchain developer	Backend & Simulation integration	Designed and implemented	Took leadership role in the	Blockchain dashboard & Visual

Member	Primary role	Secondary role	Major technical ownership	Leadership responsibility	Evidence ID
			blockchain smart contracts (Bids, Asks, Clear_Market) using Solidity, Ganache, integrated Web3 with Node.js, allowing for trading visualization in the dashboard in the form of hashes and blocks.	integration of blockchain, explaining its design and implementation to other team members, and ensuring accurate integration between blockchain and simulation.	Studio Files Evidence1 Evidence2

Table 12-B. Teamwork Evidence Matrix.

SO5 element	Project-specific evidence	Where shown in report / appendix	What changed because of team interaction
Leadership	Leadership involved parts related to hardware data acquisition, integration of blockchain components, data processing and creating a unified system.	Section 8, Section 9, Blockchain dashboard screenshots	Transparency of the system was improved and a complete transaction tracking was created
Collaborative environment	Collaborative approach was considered in all sections of the project including smart meter data acquisition (Modbus), Node.js, Solidity, Ganache.	Section 7, Section 8	Enabled smoothness in the data flow between the subsystems and decreased errors related to integration
Goals	Defined clear project goals: a network which is decentralized and allows energy trading with visualization of results of energy market	Section 2, Section 7	Directed team members to develop a unified end-to-end system in the capstone project

SO5 element	Project-specific evidence	Where shown in report / appendix	What changed because of team interaction
Task planning	Work was divided and leaders were assigned for specific parts of the projects with leaders' explanations to other group team members. Divisions: hardware (data acquisition), software (processing), blockchain (transactions), UI (dashboard)	Section 12-A, Section 6	Parallel development of tasks/multitasking/efficiency/time management were all increased.
Objectives met	Implementation of system pipeline: smart meters/CSV files/processing/blockchain/dashboard	Section 9 (Validation and Results), dashboard screenshots	The system was demonstrated to be working effectively with validated results

Table 12-C. Individual Contribution Log.

Member	Key tasks completed	Deliverables produced	Dates / period	Supervisor verification / evidence
Daniyal	Studied and implemented MODBUS RTU on industrial level smart meters with Raspberry Pi 5. Deployed Node Red flow for IoT communication.	Functions for correct reading registers inside the smart meters. Created a client which publishes MQTT messages that can be fetched locally by other devices. Collected data is stored inside the CSV files.	February-March 2026	Successful data fetching and its representation in UI. Verified by the supervisor on 1.04.2026 Evidence
Yernar	Wired smart meters to AC-DC microgrid; established RS-485 connections; configured slave IDs; Developed filtered flow of raw data from MQTT into CSV file with timestamps.	Fully wired working hardware system(AC- DC microgrid is not included); Modbus communication between	February-March 2026	Photos of wired smart meters and RS-485 connections, functions used in Node-RED to providing MQTT-CSV data pipeline.

Member	Key tasks completed	Deliverables produced	Dates / period	Supervisor verification / evidence
		Raspberry Pi 5 and all meters(mostly hardware); Automated Node-RED pipeline that writes every MQTT message to timestamped CSV files(MQTT->CSV)		Verified by the supervisor on 1.04.2026. Evidence
Damir	Executed part of the blockchain integration, creating bid & asks (demand & supply), clear_market; integrated Web3 with Node.js simulation; executed blockchain activity using blocks, hashes and visualized this data to the dashboard, created backend simulation.	Transaction network based on blockchain; simulation & server codes; blockchain activity visualized on dashboard; logs of transactions	February-March 2026	Dashboard with blockchain activity, transaction records with hashes in the form of blocks. Verified by the supervisor on 7.04.2026. Evidence

A clear charter with responsibilities and goals being shared was the main operation principle for the team. In the beginning meetings were held weekly with Capstone supervisor, postdoc fellows and team members. However, closer to February and March, meetings frequency increased, happening on a daily basis during crucial implementation phases. It is essential to mention that all decisions were made collaboratively with all project stakeholders based on testing results and system performance. Workload was distributed equally with each member of the team being responsible for a specific part of the project. The responsibilities were divided across the following parts: hardware, data processing, blockchain, and visualization components. Throughout the whole project a peer feedback was incorporated, owing to which we were able to come to better understanding and results based on iterative improvements.

13. Use of Prior Course Work and New Knowledge Acquired

Table 13-A. Prior Course Work Applied.

Design task	Prior course / topic used	How it was applied	Evidence in design
Power fetching	ELCE 303 Power System Analysis, ELCE 354 Power Electronics	These courses helped in understanding which data should be used to understand the power generation and consumption patterns over three phase and one phase systems.	Evidence.
Registers reading	ELCE 304 Microprocessor Systems	ELCE 304 gave the fundamentals in understanding of registers and their contents. Explained concepts used in programming such as big and little endian register memories.	Evidence.
Data processing and analysis	Programming, Data processing in Python and JavaScript	To process CSV data we used Python and JS programming skills, aggregated demand/supply of houses.	Evidence.
Communication protocols	ELCE 304 Computer Networks	Used knowledge of communication protocols to create an interface between Pi Raspberry 5 and smart meters for reliable data transferring.	Evidence.
Implementation of blockchain	IEEE journals, self study	Used Solidity programming language to create smart contracts and transactions in a decentralized network.	Evidence.

Table 13-B. New Knowledge Acquired and Applied.

Knowledge / skill learned	Why it was needed	Source used	How it changed the design or test process
Pymodbus library	This is a python library that helps in register reading for the MODBUS RTU protocol	Pymodbus documentation	This library offered functions for register reading.
Node Red	It was used to connect to the hardware and create visualization	Node Red tutorials	With Node Red it was very easy to subscribe to the MQTT client and create a user interface.
CSV files data processing and analysis	Was required to process and calculate data about supply/demand from 15 houses	Online JS/Node.js documentation	Allowed energy data aggregation and preparation of input data for simulation of the market.
Blockchain knowledge (Solidity programming language)	Was needed to create a decentralized network of energy trading	Solidity documentation , Remix IDE documentation	Enabled to implement smart contracts to create bids, asks and clear markets for market interactions.
Ganache (Local Blockchain)	Was needed to test transactions	Ganache documentation	Enabled free of charge, secure and efficient testing of transactions.
Web3 Integration	Was needed to connect the blockchain part with Node.js backend	Web3.JS documentation	Allowed to address transactions in a programmatical way.
Dashboard Visualization	Was required to display system results in a user-friendly way	Resources of web development (HTML, CSS, JS)	Enhanced the understanding and interface of the system.

14. Conclusions and Recommendations

The developed project of energy data acquisition satisfies our requirements and goals. However, real data collection was limited due to the laboratory constraints since it would not be reasonable to be connected to the AC/DC microgrid lab over a long period of time. That is the reason why for blockchain integration we have used open source datasets. The hardware part of the project was not challenging to implement, since the instruction manual clearly states the connection diagrams and register addresses, except the library learning and implementing its functions took some time to understand. In the end, we successfully tuned data acquisition part with the Python code.

One of the main trade-offs that we have accepted while completing this project was to prioritize modularity, scalability, and overall simplicity by clearly structuring the data collection process. As it was mentioned in Section 5 of this report, more sophisticated design would have higher costs and heavier computational loads, because of this we kept this design choice.

The main limitation of this Capstone project is that it still has not been tasted on a real industrial scale level. The robustness of the system, fault corrections and other challenges could be a great addition to this project in the future.

Regarding the stage of data processing, in the system we were successfully able to create structured sets of inputs for energy management from smart meters data. Data of supplies and demands of houses were aggregated using Node.js in VS Code, which led to achieving dynamic pricing, as well as the SDR and resulting matching between consumers and prosumers. Compared to static pricing, dynamic pricing uses real-time hourly rate data to maximize revenue and make energy management appealing for grid users. Moreover, we implemented the blockchain layer that increased the system's security, transparency and made the system decentralized. Each energy transaction was recorded in the form of a blockchain block having hashes, meaning the information there could not be stolen, edited or deleted. Solidity and Ganache were the main software utilities which allowed the creation of bids, asks, and market clearing - fundamental mechanics of supply and demand in energy management. However, it is important to mention some limitations of this specific project's part as well, which is that a local blockchain (Ganache) was used instead of the public one (e.g. Polygon), because of cost and deployment constraints. Hence, transaction fees and latency were not completely dealt with, which are happening in real-world scenarios. Moreover, the data processing stage focused entirely on CSV files of past dates instead of real-time streaming. Potential future work consists of extending the system by deploying on a public blockchain, integrating live data streams and increasing the number of households by performing scaling.

Throughout the project we were able to foster skills related to data processing, develop knowledge in Solidity smart contracts and Ganache for blockchain, dynamic pricing modeling, as well as the Node.js programming. Finally, we improved our understanding and motivation to study decentralized energy systems in the future, potentially contributing to energy related problems worldwide in future, while working under real engineering constraints and conditions.

15. References

- [1] A. Umar, D. Kumar and T. Ghose, "Decentralized energy trading in microgrids: a blockchain-integrated model for efficient power flow with social welfare optimization," **Electrical Engineering**, vol. 107, pp. 2677-2695, 2025. doi: 10.1007/s00202-024-02635-x.
- [2] A. R. K. Reddy, M. K. Purkayastha, and R. Pal, "Blockchain applications in smart grid - review and frameworks," **IEEE Access**, vol. 7, pp. 179-207, 2019. doi: 10.1109/ACCESS.2019.2895567.
- [3] Y. Zhang, X. Li, Z. Chen and H. Wang, "Title of the Article," **Proceedings of the IEEE**, vol. , no. , pp. , 2023, doi: 10.1109/XXXXX.2023.10549519.
- [4] A. K. Vishwakarma, M. K. Purkayastha, and S. Rai, "Blockchain-based peer-to-peer renewable energy trading in distributed networks," **International Journal of Sustainable Energy**, vol. 43, no. 8, pp. 491–508, 2024. doi: 10.1080/01605682.2024.2441224.
- [5] B. Luo, Q. Zhang, Z. Li, et al., "Blockchain-enabled secure smart-grid communications: A survey," in **2023 IEEE International Conference on Smart Energy Systems and Technologies (SEST)**, Leuven, Belgium, 2023, pp. 1-6, doi: 10.1109/SEST56429.2023.10995102.
- [6] M. D. Mudaliar and N. Sivakumar, "IoT based real time energy monitoring system using Raspberry Pi," *Internet of Things*, vol. 12, p. 100292, 2020.
- [7] H. Samanta, S. Mohanty, and B. Dash, "Internet of things based smart energy management in a microgrid," *Sustainable Energy Technologies and Assessments*, vol. 42, p. 100848, 2020.
- [8] C. K. Rao, V. Narayana, and P. Kumar, "An IoT-based intelligent smart energy monitoring system for a photovoltaic-integrated microgrid," *Energies and Smart Homes Systems*, vol. 4, no. 1, pp. 45–60, 2024.
- [9] Y. Zhang, L. Xu, and D. Wang, "Incorporating temperature with parallel short-term power load forecasting and management," *Electrical Engineering*, vol. 107, pp. 13801–13826, 2025.
- [10] G. Yogarajan, K. Abirami, K. D. Ahmed, et al., "Enhancing privacy-preserving load forecasting using residual multi-layer perceptron," *Electrical Engineering*, vol. 108, p. 1, 2026.
- [11] M. S. Nkambule, A. N. Hasan, and T. Shongwe, "A review of intelligent control strategies for energy management systems in microgrids," *Energy Conversion and Management: X*, vol. 28, p. 101323, 2025, doi: 10.1016/j.ecmx.2025.101323.
- [12] B. S. Velmurugan and P. Somasundram, "Cloud-Based Energy Management in Residential and Commercial V2G Networked Microgrid," *2025 IEEE North-East India International Energy Conversion Conference and Exhibition (NE-IECCE)*, Silchar, India, 2025, pp. 1-6, doi: 10.1109/NE-IECCE64154.2025.11183176.

- [13] D. G. Rosero, N. L. Díaz, and C. L. Trujillo, "Cloud and machine learning experiments applied to the energy management in a microgrid cluster," *Applied Energy*, vol. 304, p. 117770, 2021, doi: 10.1016/j.apenergy.2021.117770.
- [14] J. S. Choi, "A Hierarchical Distributed Energy Management Agent Framework for Smart Homes, Grids, and Cities," in *IEEE Communications Magazine*, vol. 57, no. 7, pp. 113-119, July 2019, doi: 10.1109/MCOM.2019.1900073.
- [15] IEEE Standards Association, "IEEE Std 2030.5-2023: Smart Energy Profile Application Protocol," IEEE, 2023.
- [16] "ISO 27001:2022 Annex A 5.15 – Access Control," ISMS.online. [Online]. Available: <https://www.isms.online/iso-27001/annex-a-2022/5-15-access-control-2022/> (accessed May XX, 2025).
- [17] IEEE Standards Association, "IEEE Std 1547-2018: Standard for Interconnection and Interoperability of Distributed Energy Resources with Associated Electric Power Systems Interfaces," IEEE, 2018.

Appendix A. Engineering Standards Evidence

Table A.1. Engineering Standards Evidence.

Standard ID and title	Clause(s) used	Why applicable?	Design decision affected	How verified	Status
STD-1 - IEEE Std 2030.5: Smart Energy Profile Application Protocol [15]	Section 6 - Device Discovery and Registration; Section 7 - Secure Communication; Section 8 - Smart Energy Data Models	This standard is an IP-based smart meter protocol, and its principles are applied to the process of defining the data collection on Raspberry Pi and MEMS system.	Format of the data for communication between IoT and MEMS should be structured. The smart-meters which will be linked to the microgrid will be registered in accordance to the standard.	In reading the content of the data packets, they are expected to be organized by 2030.5-schema. For instance, it should include ID, time, units and value, by the standard format. MEMS can only communicate with registered devices.	Not Violated
STD-2- ISO/IEC 27001 - Information Security Management [16]	Annex A 5.15 - Access Control, Annex A 8.24 - Use of cryptography	These standards are used in determining the type of access control the MEMS will possess (nodes of the system), and how the data of contacts should be encrypted to ensure integrity.	Only registered members of the system have permission to the system, and their blockchain accounts' private keys are stored in the keystores.	Unauthorized users should not be able to temper with the data and contracts. All blockchain blocks may be checked for transparency but cannot be modified in any way.	Not Violated
STD -3 - IEEE Std 1547: Standard for Interconnection and	Clause 4 - General specifications and requirements;	The AC/DC Microgrid used in this project employs distributed	The trip levels of circuit breakers should be compared and must be similar to the IEEE	When the protection test as defined by IEEE-1547 is done, the circuit breakers in	Not Violated

Interoperability of Distributed Energy Resources with Associated Electric Power System Interfaces [17]	Clause 6 - Response to Area EPS abnormal conditions	resources which are expected to be within the acceptable ranges as indicated in Clause 4. Protection devices have thresholds which are selected under clause 6.	1547. Under idle operation, voltage parameters are supposed to be in ranges of Clause-4.	series with the electrical load must operate. The normal result should show stable parameters on the oscilloscope when running microgrid.	
STD -3 - ISO/IEC 27001:2022 – Information Security Management (Capacity Management)	Annex A.8.6 - Capacity Management	Raspberry Pi 5 currently has limited CPU capabilities and will not be able to scale up to more nodes in the future. Capacity management does not allow the system to be overloaded.	Raspberry Pi 5 operation is observed with command <i>top</i> to monitor the CPU load. It is set to a threshold of at least 70% to have headroom to add more processing and scale to come.	While EMS is running run <i>top</i> 10 minutes (in fact 36 minutes) Calculate average CPU load using log. The average load measured was 14.55 percent in 36 minutes which is far less than the 70 percent mark.	Not Violated

Appendix B. Multiple-Constraints Matrix

Table B.1. Multiple-Constraints Matrix.

Constraint	Metric / target	Rationale	Trade-offs considered	Outcome/ status	Evidence
Technical	Acquisition from several smart meters over Modbus RTU (2s), local publishing through MQTT (5s for this project)	Practical and low-cost architecture for data collection, logging	Local server over cloud, polling data over real time logging onto CSV	Raspberry Pi 5 easily reads smart meter registers, decodes their contents and outputs real values, variables are published and logged as stated in Node Red flow	Evidence 3 Evidence 4 Evidence 5
Software structure	Clear structure with non-repeatable functions	Create a structured code to have increased optimization	Call one function to collect data and one function to publish over repeating reading registers for each parameter	Functions like read_meter_data() and publish_three_phase_meter() easily collect and send data to MQTT broker.	Evidence 3
Economic (cost savings, tariffs)	30.56 KZT/kWh for purchasing and 22.68 KZT/kWh for selling	Reflection of real KZT tariffs	Lower prices vs Real profit	Cost savings and dynamic pricing mechanisms	Savings vs baseline demonstration in the dashboard
Computational (hourly rate)	An extended period of hourly operation	Needed for energy management being continuous	Real time processing vs system stability	Achieved hourly	Navigation in the dashboard across time periods
Blockchain (transparent)	Transactions contain	Security, trust and transparent	Ensured Transparency vs Gas	Successful implementation	BID/ASK/CLEAR MARKET

cy and security)	estimated gas, blocks and hashes	cy are ensured		of transaction logging	demonstration in the dashboard
------------------	----------------------------------	----------------	--	------------------------	--------------------------------

Appendix C. Team Organization and Contributions

The team organization over this project can be divided into four main parts, that are hardware integration, data acquisition and processing, blockchain implementation, and dashboard. The organization of the meetings and lab visits were always discussed by the members, and each week offline meetings were held either for lab work or report preparation. From the beginning of the project till the end of it, team members were updating their supervisors. Team members have divided their roles based on these parts, where Yernar was responsible for hardware integration and CSV logging of the data inside the Node Red, Daniyal was the leading developer of the Python code for creating the Modbus and MQTT clients and data acquisition. Damir took the lead in blockchain integration and visualization, where he created smart contracts, SDR algorithms and so on.

Table C. Responsibility Matrix

Member	Primary role	Collaboration with other members	Main responsibility
Daniyal	Data collection and preprocessing of data	Daniyal together with Damir decided on the structure of the smart contracts and what kind of data should be collected from smart meters to develop smart contracts	Python on Raspberry Pi 5 MQTT Node Red user interface and MQTT subscription
Yernar	Hardware integration	Yernar was debugging the Node Red flow to create a correct data type for CSV logging	Smart meters configuration Wiring and connection checking of the communication lines
Damir	Blockchain development	Damir was double-checking wiring and prepared the microgrid for operation	Blockchain integration Smart contracts Dashboard

Appendix D. Design Evidence

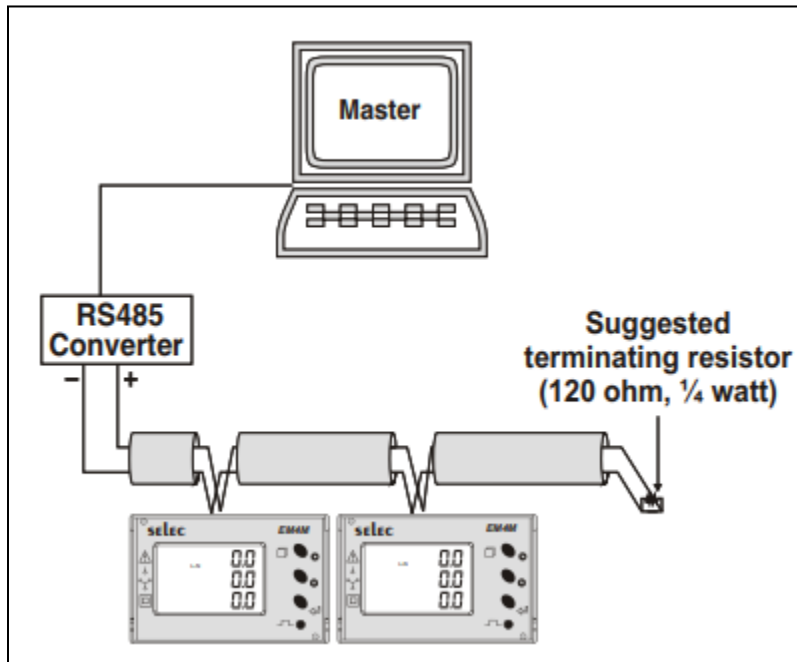


Figure D1: Connection diagram for Communication between Master and Slaves

Since the Hybrid AC/DC Microgrid had a three phase bus connection to the loads, the smart meters were configured for three wire configuration as shown in Figure D2.

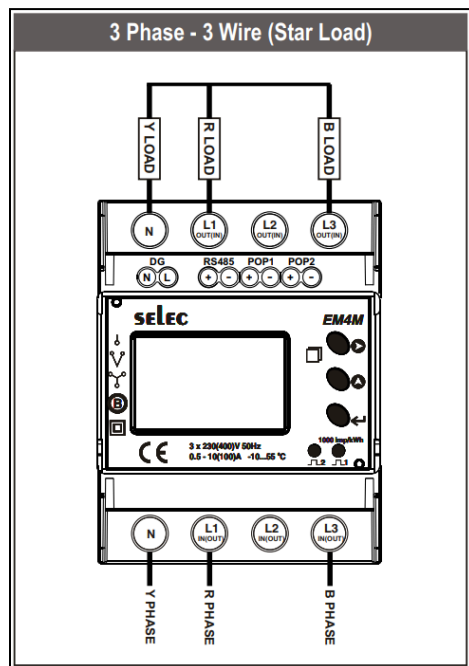


Figure D2: Three wire connection on smart meter

The following code is a function developed to read the registers that are swapped. The function is called *read_float_swapped()*.

```
# Reads a 32-bit float from 2 input registers with swapped word order

def read_float_swapped(client, slave_id, reg_addr):

    response = client.read_input_registers(

        address=reg_addr,

        count=2,

        slave=slave_id

    )

    if response.isError():

        return None, response

    registers = response.registers

    packed_bytes = struct.pack(">HH", registers[1], registers[0])

    value = struct.unpack(">f", packed_bytes)[0]

    return value, None
```

Figure D3: read_float_swapped() function definition

The Figure D3 shows the terminal logs that display the data that was read from the smart meter with slave ID 1 and slave ID 2.

```
capstone@raspb... mqtt_modbus_capst.. capstone@raspb...
File Edit Tabs Help
Published rp1/broadcast/METER1_POWERFACTOR -> 1.00 | ok=True
Message published: 5586
Published rp1/broadcast/METER1_FREQUENCY -> 49.99 | ok=True
Message published: 5587
Published rp1/broadcast/METER2_AVG_VOLTAGE -> 0.00 | ok=True
Message published: 5588
Published rp1/broadcast/METER2_VOLTAGE -> 411.16 | ok=True
Message published: 5589
Published rp1/broadcast/METER2_VOLTAGE -> 400.04 | ok=True
Message published: 5590
Published rp1/broadcast/METER2_VOLTAGE -> 400.46 | ok=True
Message published: 5591
Published rp1/broadcast/METER2_AVG_CURRENT -> 0.73 | ok=True
Message published: 5592
Published rp1/broadcast/METER2_CURRENT -> 1.11 | ok=True
Message published: 5593
Published rp1/broadcast/METER2_CURRENT -> 0.00 | ok=True
Message published: 5594
Published rp1/broadcast/METER2_CURRENT -> 1.07 | ok=True
Message published: 5595
Published rp1/broadcast/METER2_ACTIVEPOWER -> 0.78 | ok=True
Message published: 5596
Published rp1/broadcast/METER2_REACTIVEPOWER -> -0.01 | ok=True
Message published: 5597
Published rp1/broadcast/METER2_POWERFACTOR -> 1.00 | ok=True
Message published: 5598
Published rp1/broadcast/METER2_FREQUENCY -> 50.00 | ok=True
Message published: 5599
Published rp1/broadcast/METER1_VOLTAGE -> 230.02 | ok=True
Message published: 5600
Published rp1/broadcast/METER1_CURRENT -> 0.00 | ok=True
Message published: 5601
Published rp1/broadcast/METER1_ACTIVEPOWER -> 0.00 | ok=True
Message published: 5602
Published rp1/broadcast/METER1_REACTIVEPOWER -> 0.00 | ok=True
Message published: 5603
Published rp1/broadcast/METER1_POWERFACTOR -> 1.00 | ok=True
Message published: 5604
Published rp1/broadcast/METER1_FREQUENCY -> 49.99 | ok=True
```

Figure D4: Terminal logs of the smart meters.

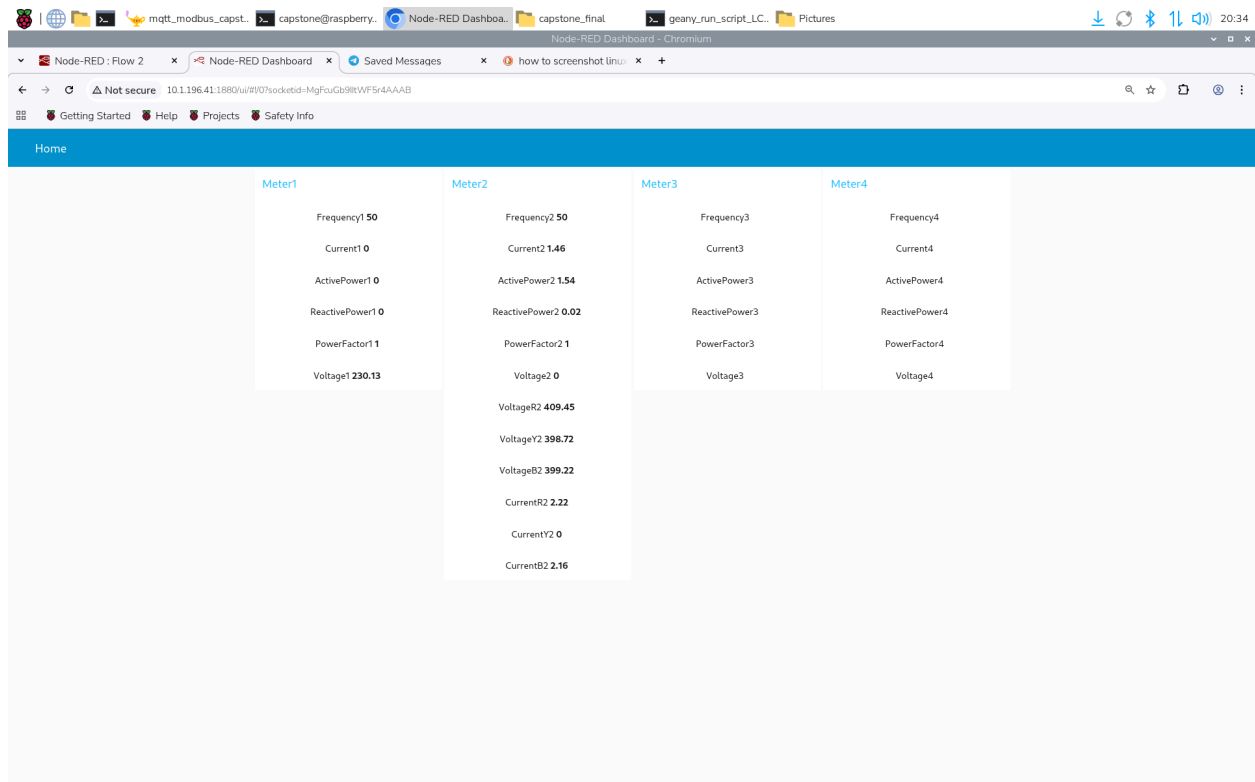


Figure D5: Real time monitoring of smart-meters

Figure D6 displays a sample of the CSV file written inside Node Red. It has columns named: timestamp, meter_id, voltage, current, voltage_r, voltage_y, voltage_b, current_r, current_y, current_b, active_power, reactive_power, power_factor, frequency

timestamp	meter_id	voltage	voltage_r	voltage_y	voltage_b	current_r	current_y	current_b	current	active_power	reactive_power	power_factor	frequency
2026-04-03T14:32:28.401Z	1	230.17							0.00	0.00	0.00	1.00	50.00
2026-04-03T14:32:28.401Z	2	0.00	411.54	400.33	216.58	0.04	0.00	0.00	0.01	0.01	-0.02	0.46	50.00
2026-04-03T14:32:33.401Z	1	229.90							0.00	0.00	0.00	1.00	50.00
2026-04-03T14:32:33.401Z	2	0.00	411.49	400.46	400.57	0.04	0.00	0.00	0.01	0.01	-0.02	0.46	50.00
2026-04-03T14:32:38.405Z	1	229.90							0.00	0.00	0.00	1.00	50.00
2026-04-03T14:32:38.405Z	2	0.00	411.49	400.46	400.57	0.04	0.00	0.00	0.01	0.01	-0.02	0.46	50.00
2026-04-03T14:32:43.406Z	1	230.16							0.00	0.00	0.00	1.00	49.99
2026-04-03T14:32:43.406Z	2	0.00	411.55	400.65	400.34	0.04	0.00	0.00	0.01	0.01	-0.02	0.47	50.00
2026-04-03T14:32:48.406Z	1	230.16							0.00	0.00	0.00	1.00	49.99
2026-04-03T14:32:48.406Z	2	0.00	411.55	400.65	400.34	0.04	0.00	0.00	0.01	0.01	-0.02	0.47	50.00
2026-04-03T14:32:53.408Z	1	230.15							0.00	0.00	0.00	1.00	49.98
2026-04-03T14:32:53.408Z	2	0.00	411.26	400.67	400.02	0.04	0.00	0.00	0.01	0.01	-0.02	0.45	50.00
2026-04-03T14:32:58.409Z	1	230.15							0.00	0.00	0.00	1.00	49.98

Figure D6: CSV file logs

```

1  const fs = require('fs');
2
3  const { readCSV } = require('./utils');
4  const { Agent } = require('./prosumer');
5  const { exchange, getAccounts } = require('./exchange');
6  const calculateIntersection = require('./inter');
7
8  const outputFile = 'output.csv';
9  const defaultBatteryCapacity = 12000;
10
11 // Kazakhstan tariffs in KZT/kWh
12 const GRID_BUY_PRICE_KZT_PER_KWH = 30.56;
13 const GRID_SELL_PRICE_KZT_PER_KWH = 22.68;
14
15 // UI-friendly mode
16 const RUN_SINGLE_ROW = false;
17 const TEST_ROW = 9;
18 const DEMO_STEPS = 24;
19
20 const VERBOSE = false;
21
22 async function init() {
23   console.log('Reading files...');
24
25   const { householdHistoricData } = await getFiles();
26   const { agents } = await createAgents(
27     householdHistoricData,
28     defaultBatteryCapacity,
29     false
30   );
31
32   const accounts = await getAccounts();

```

Figure D7: Visual Studio Code as the main development environment

VS code was used as a key environment for developing stages such as CSV files processing, blockchain integration, backend components.

The following fragment of code from *Simulation.js* file demonstrates the import process of data retrieved from smart meters from the corresponding CSV files and conversion of houses into agents, representing microgrid users:

```

async function getFiles() {
  const householdHistoricData = [];

  for (let i = 1; i <= 15; i++) {
    householdHistoricData.push(await readCSV(`./dataset/house${i}.csv`));
  }

  return { householdHistoricData };
}

async function createAgents(householdHistoricData, batteryCapacity, batteryBool) {
  console.log("Creating Agents...");

  const agents = [];

  for (let i = 1; i <= 15; i++) {
    if (i >= 8) {
      batteryBool = true;
    }

    const agent = new Agent(batteryCapacity, batteryBool);
    await agent.loadSmartMeterData(householdHistoricData[i - 1], i);

    agents.push({
      id: i,
      agent
    });
  }

  return { agents };
}

```

Figure D8: Import process of data retrieved from smart meters

The next code fragment from *Simulation.js* displays how houses are analyzed and set to be either prosumers or consumers, calculating the aggregated values of demand and supply:

```

for (let j = 0; j < agents.length; j++) {
  const agent = agents[j].agent;

  const demand = Number(agent.historicalDemand[i]?.demand || 0);
  const supply = Number(agent.historicalSupply[i]?.supply || 0);

  const selfConsumed = Math.min(demand, supply);
  const surplus = Math.max(supply - demand, 0);
  const deficit = Math.max(demand - supply, 0);
  const houseEnergy = supply - demand;

  let role = 'Neutral';
  if (surplus > 0) {
    role = 'Prosumer';
    prosumersCount++;
  } else if (deficit > 0) {
    role = 'Consumer';
    consumersCount++;
  }

  housesData.push({
    house: j + 1,
    demand,
    supply,
    selfConsumed,
    surplus,
    deficit,
    energy: houseEnergy,
    role
  });

  totalDemand += demand;
  totalSupply += supply;
  selfConsumedEnergy += selfConsumed;
  localSurplusAvailable += surplus;
  localDemandNeed += deficit;
}

```

Figure D9: Analysis of houses to be either prosumers or consumers

The following fragment of code from [Simulation.js](#) demonstrates how surplus and deficit energy are created into asks and bids of the market and then transported to the blockchain:

```

for (let h = 0; h < housesData.length; h++) {
  const house = housesData[h];

  if (house.deficit > 0) {
    bids.push({
      house: house.house,
      amount: house.deficit,
      price: internalPriceKztPerKWh
    });
  }

  if (house.surplus > 0) {
    asks.push({
      house: house.house,
      amount: house.surplus,
      price: internalPriceKztPerKWh
    });
  }
}

const clearLog = await clearBlockchainMarket(accounts);
const bidLogs = await sendBidsToBlockchain(bids, i, accounts);
const askLogs = await sendAsksToBlockchain(asks, i, accounts);

```

Figure D10: Creation of surplus and deficit energy.

Finally, the last fragment of *Simulaion.js* file shows peer-to-peer energy matching:

```

for (let b = 0; b < bids.length; b++) {
  const buyer = bids[b];

  for (let a = 0; a < asks.length; a++) {
    const seller = asks[a];

    if (buyer.amount <= 0) break;
    if (seller.amount <= 0) continue;

    const tradedAmount = Math.min(buyer.amount, seller.amount);

    transactions.push({
      from: seller.house,
      to: buyer.house,
      amount: tradedAmount,
      price: internalPriceKztPerKWh
    });

    buyer.amount -= tradedAmount;
    seller.amount -= tradedAmount;
  }
}

if (bids[r].amount > 0) {
  gridEnergyPurchased += bids[r].amount;
}

if (asks[s].amount > 0) {
  gridEnergySold += asks[s].amount;
}

fs.writeFileSync(outputFile, output, 'utf8');

```

Figure D11: Peer-to-peer energy matching.

Time	Demand (kWh)	Supply (kWh)
2026-03-01 0:00	0.527	0
2026-03-01 1:00	0.463	0
2026-03-01 2:00	0.497	0
2026-03-01 3:00	0.498	0
2026-03-01 4:00	0.445	0
2026-03-01 5:00	0.511	0
2026-03-01 6:00	0.496	0
2026-03-01 7:00	0.459	0
2026-03-01 8:00	0.435	0.286
2026-03-01 9:00	0.254	0.725

Figure D12: Example of House Data Set (Visually Clear version)

The following formula illustrates how the pricing of energy was determined based on supply/demand ratio:

$$P = P_{grid} * \frac{DEMAND}{SUPPLY}$$

The next code fragment from [Prosumer.js](#) demonstrates how data from CSV files is loaded into the agent model. This data is then used in energy balancing:

```

async loadSmartMeterData(historicData, householdID) {
  this.householdID = householdID;

  for (let i = 1; i < historicData.length - 1; i++) {
    const currentDemand = {
      time: historicData[i][0],
      demand: parseFloat(historicData[i][1]) * 1000 // kWh → Wh
    };

    const currentSupply = {
      time: historicData[i][0],
      supply: parseFloat(historicData[i][2]) * 1000 // kWh → Wh
    };

    this.historicalDemand.push(currentDemand);
    this.historicalSupply.push(currentSupply);
  }

  return true;
}

```

Figure D13: CSV files loading into the agent model.

The next code fragment is from *Exchange.js* and it shows a connection to Ganache, which is a local blockchain network using Web3.js:

```

const Web3 = require('web3');

const web3 = new Web3('http://127.0.0.1:8545'); // Ganache RPC

const exchange = new web3.eth.Contract(abi, address);

async function getAccounts() {
  return await web3.eth.getAccounts();
}

```

Figure D14: A connection to Ganache.

The next code fragment is a part of *Exchange.sol* written on Solidity, which is the implementation of the smart contract, it records all the energy data to the blockchain and emits an event confirming the transaction:

```
function placeBid(uint _price, uint _amount, uint timestamp) public returns (bool) {
    Bid memory b;
    b.owner = msg.sender;
    b.price = _price;
    b.amount = _amount;
    b.date = timestamp;

    Bids.push(b);

    emit BidPlaced(msg.sender, _price, _amount, timestamp);
    return true;
}
```

Figure D15: Solidity smart contract.

ACCOUNTS

BLOCKS

TRANSACTIONS

CONTRACTS

EVENTS

LOGS

SEARCH FOR BLOCK NUMBERS OR TX HASHES

CURRENT BLOCK2112

GAS PRICE2000000000

BLOCKS8721975

HARDWAREMERGE

NETWORK11337

API SERVERHTTP://127.0.0.1:8545

MINER STATUSAUTOMINING

WORKSPACECAPSTONE_BLOCKCHAIN

SWITCH

MEMONIC

suggest apology east design ramp crisp border cover pulse human beef price

HD PATHm44'60'0"baccount_index

ADDRESS	BALANCE	TX COUNT	INDEX	
0xEab944DcB06e2DBdb1692CF080bF29709638AF1f	99.82 ETH	358	0	
ADDRESS	BALANCE	TX COUNT	INDEX	
0x242F11a277180212dEE1bd93a76af7065a0B2ad8	99.95 ETH	169	1	
ADDRESS	BALANCE	TX COUNT	INDEX	
0x17BC707d5fB0AcEd6AD532DAF3E8f7EE3f43395	99.95 ETH	168	2	
ADDRESS	BALANCE	TX COUNT	INDEX	
0xa53b8f688a5911cF823433955E3A2630d6CD5a50	99.95 ETH	168	3	
ADDRESS	BALANCE	TX COUNT	INDEX	
0xc218dCa41412f69FbE9B1b0326A9133f62528a8b	99.95 ETH	168	4	
ADDRESS	BALANCE	TX COUNT	INDEX	
0x21Dc9D604E5616C67f055725B93d1EE2F3957d02	99.95 ETH	168	5	
ADDRESS	BALANCE	TX COUNT	INDEX	
0x1FB0CAa12879Fa79AaE46adDB5972C07432639BD	99.95 ETH	168	6	
ADDRESS	BALANCE	TX COUNT	INDEX	
0x854920956541EF0A0E93CE6Ba558305b75679198	99.95 ETH	168	7	
ADDRESS	BALANCE	TX COUNT	INDEX	
0xb01ec0fe88882165A39511457d645611c764317d	99.95 ETH	168	8	

Figure D16: Ganache Blockchain accounts

[illegible]

Figure D17: Ganache Blockchain transactions

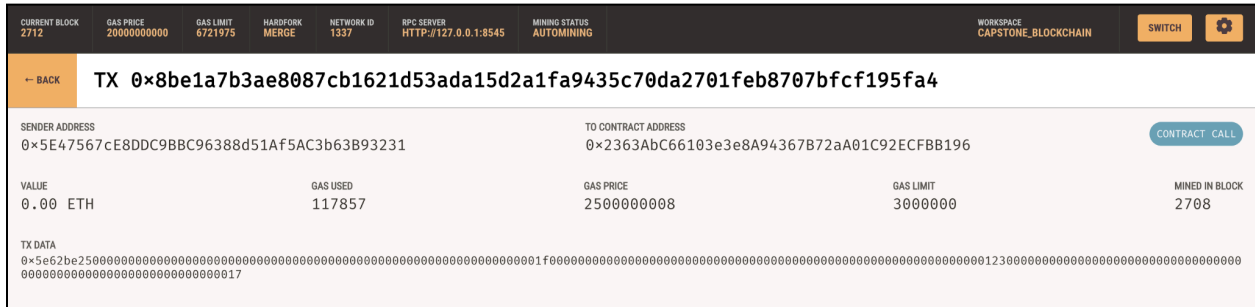


Figure D18: Ganache Blockchain single transaction

The above pictures represent the Ganache Blockchain environment and show the successful integration of the blockchain layer with all energy market transactions being recorded as transactions in hashes, proving the security.

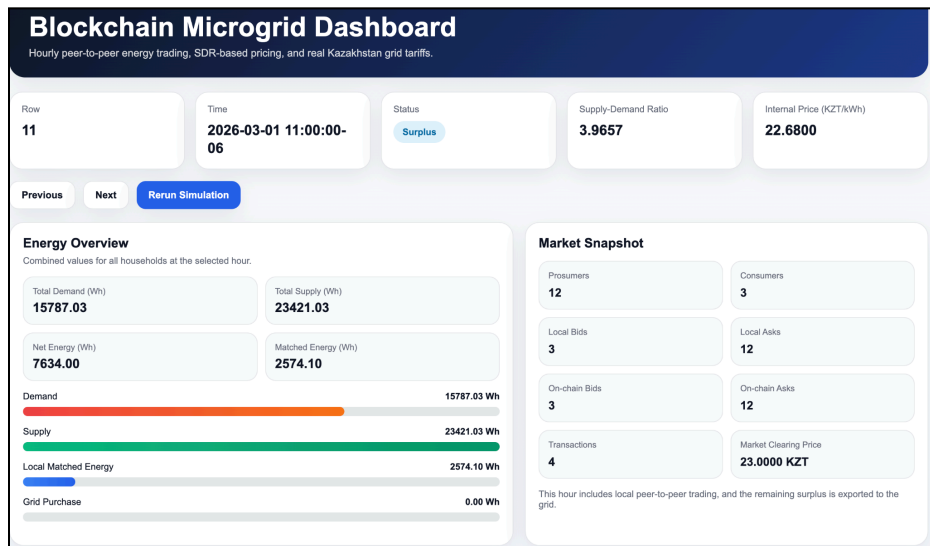


Figure D19: Main Dashboard

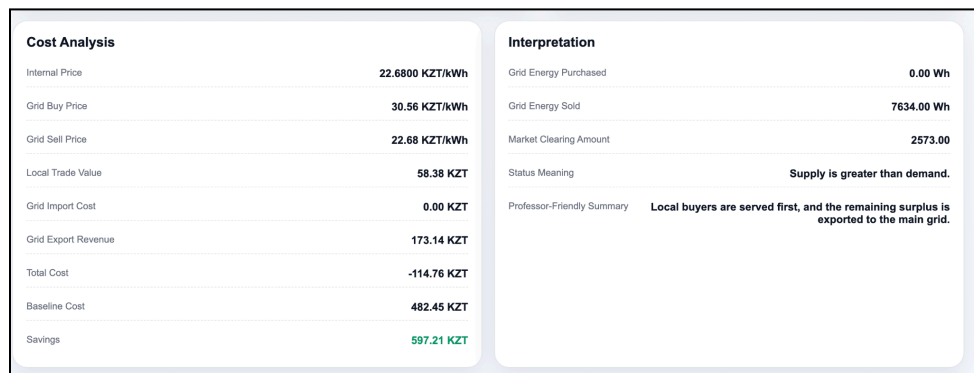


Figure D20: Cost analysis

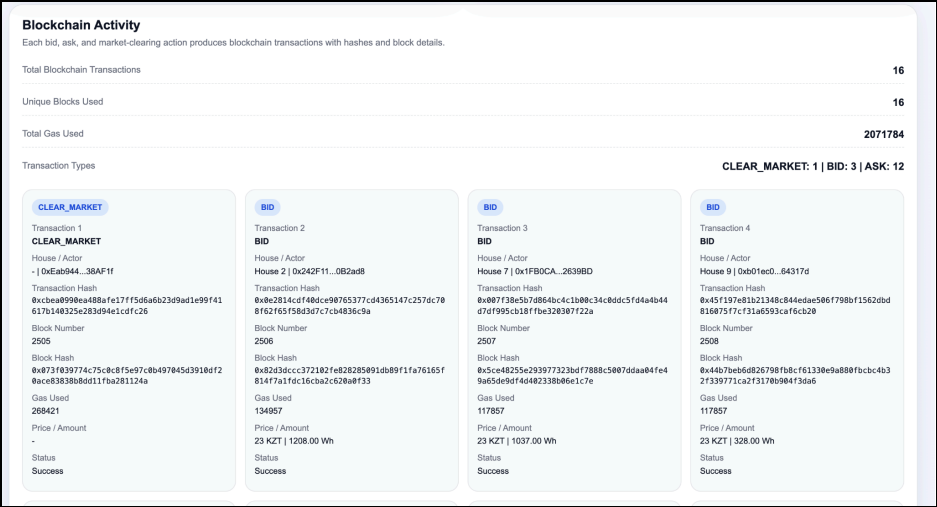


Figure D21: Blockchain activity pt.1

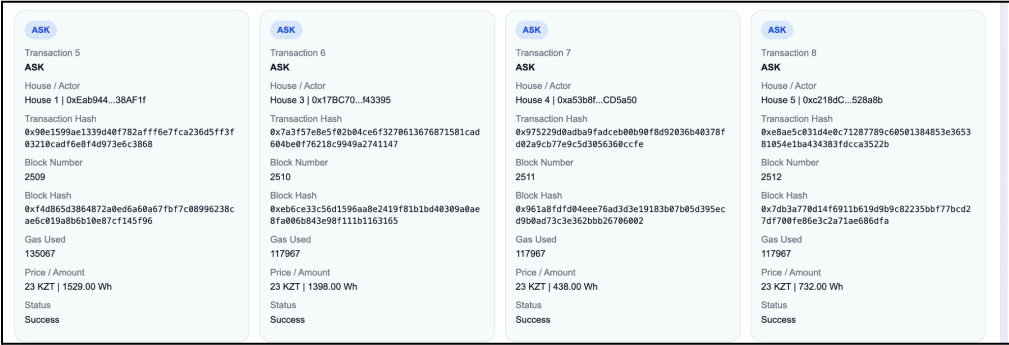
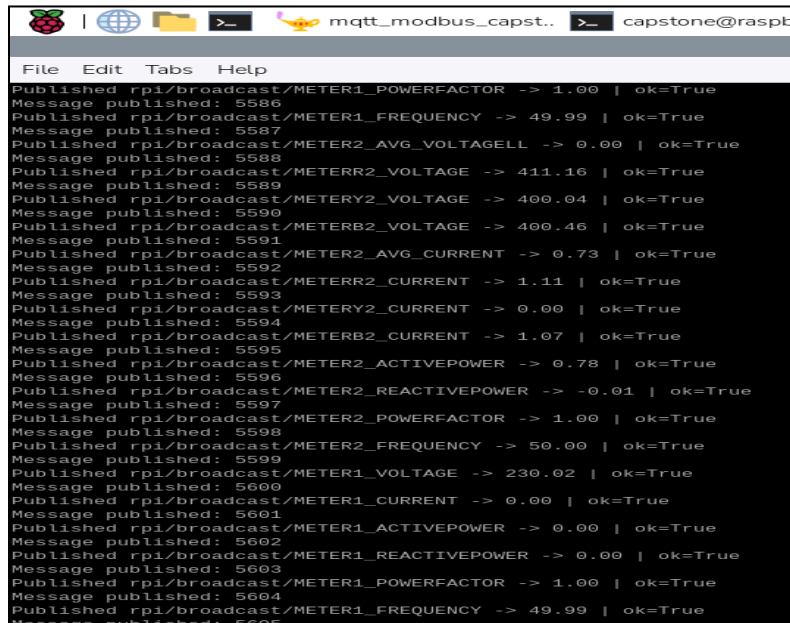


Figure D22: Blockchain activity pt.2

The above pictures illustrate the microgrid dashboard that contains the following parts: energy metrics, market results, and cost analysis, blockchain activity in terms of asks and bids.

Appendix E. Verification Evidence



A terminal window on a Raspberry Pi showing the output of a Python script. The window title is 'capstone@raspb'. The terminal displays a series of MQTT messages published to the topic 'rpi/broadcast/'. Each message contains a meter ID, a timestamp, and various electrical measurements. The messages are as follows:

```
Published rpi/broadcast/METER1_POWERFACTOR -> 1.00 | ok=True
Message published: 5586
Published rpi/broadcast/METER1_FREQUENCY -> 49.99 | ok=True
Message published: 5587
Published rpi/broadcast/METER2_AVG_VOLTAGE -> 0.00 | ok=True
Message published: 5588
Published rpi/broadcast/METER2_VOLTAGE -> 411.16 | ok=True
Message published: 5589
Published rpi/broadcast/METER2_VOLTAGE -> 400.04 | ok=True
Message published: 5590
Published rpi/broadcast/METER2_VOLTAGE -> 400.46 | ok=True
Message published: 5591
Published rpi/broadcast/METER2_AVG_CURRENT -> 0.73 | ok=True
Message published: 5592
Published rpi/broadcast/METER2_CURRENT -> 1.11 | ok=True
Message published: 5593
Published rpi/broadcast/METER2_CURRENT -> 0.00 | ok=True
Message published: 5594
Published rpi/broadcast/METER2_CURRENT -> 1.07 | ok=True
Message published: 5595
Published rpi/broadcast/METER2_ACTIVEPOWER -> 0.78 | ok=True
Message published: 5596
Published rpi/broadcast/METER2_REACTIVEPOWER -> -0.01 | ok=True
Message published: 5597
Published rpi/broadcast/METER2_POWERFACTOR -> 1.00 | ok=True
Message published: 5598
Published rpi/broadcast/METER2_FREQUENCY -> 50.00 | ok=True
Message published: 5599
Published rpi/broadcast/METER1_VOLTAGE -> 230.02 | ok=True
Message published: 5600
Published rpi/broadcast/METER1_CURRENT -> 0.00 | ok=True
Message published: 5601
Published rpi/broadcast/METER1_ACTIVEPOWER -> 0.00 | ok=True
Message published: 5602
Published rpi/broadcast/METER1_REACTIVEPOWER -> 0.00 | ok=True
Message published: 5603
Published rpi/broadcast/METER1_POWERFACTOR -> 1.00 | ok=True
Message published: 5604
Published rpi/broadcast/METER1_FREQUENCY -> 49.99 | ok=True
Message published: 5605
```

Figure E1: Terminal logs of the smart meters.

Figure E1 represents the Python script terminal output demonstrating real-time Modbus RTU register readings, validating hardware communication.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	timestamp	meter_id	voltage	voltage_r	voltage_y	voltage_b	current_r	current_y	current_b		current	active_power	reactive_power	power_factor	frequency
2	2026-04-03T14:00:00	1	230.17								0.00	0.00	0.00	1.00	50.00
3															
4	2026-04-03T14:00:00	2	0.00	411.54	400.33	216.58	0.04	0.00	0.00		0.01	0.01	-0.02	0.46	50.00
5															
6	2026-04-03T14:00:00	1	229.90								0.00	0.00	0.00	1.00	50.00
7															
8	2026-04-03T14:00:00	2	0.00	411.49	400.46	400.57	0.04	0.00	0.00		0.01	0.01	-0.02	0.46	50.00
9															
10	2026-04-03T14:00:00	1	229.90								0.00	0.00	0.00	1.00	50.00
11															
12	2026-04-03T14:00:00	2	0.00	411.49	400.46	400.57	0.04	0.00	0.00		0.01	0.01	-0.02	0.46	50.00
13															
14	2026-04-03T14:00:00	1	230.16								0.00	0.00	0.00	1.00	49.99
15															
16	2026-04-03T14:00:00	2	0.00	411.55	400.65	400.34	0.04	0.00	0.00		0.01	0.01	-0.02	0.47	50.00
17															
18	2026-04-03T14:00:00	1	230.16								0.00	0.00	0.00	1.00	49.99
19															
20	2026-04-03T14:00:00	2	0.00	411.55	400.65	400.34	0.04	0.00	0.00		0.01	0.01	-0.02	0.47	50.00
21															
22	2026-04-03T14:00:00	1	230.15								0.00	0.00	0.00	1.00	49.98
23															
24	2026-04-03T14:00:00	2	0.00	411.26	400.67	400.02	0.04	0.00	0.00		0.01	0.01	-0.02	0.45	50.00
25															
26	2026-04-03T14:00:00	1	230.15								0.00	0.00	0.00	1.00	49.98
27															
28	2026-04-03T14:00:00	2	0.00	411.26	400.67	400.02	0.04	0.00	0.00		0.01	0.01	-0.02	0.45	50.00
29															
30	2026-04-03T14:00:00	1	229.81								0.00	0.00	0.00	1.00	49.97
31															
32	2026-04-03T14:00:00	2	0.00	411.35	400.59	399.50	0.04	0.00	0.00		0.01	0.01	-0.02	0.45	50.00
33															
34	2026-04-03T14:00:00	1	229.81								0.00	0.00	0.00	1.00	49.97

Figure E2: CSV log file of smart meter measurements

Figure D5 shows the contents of the smart_meter_data.csv file, containing timestamped electrical measurements from connected smart meter.

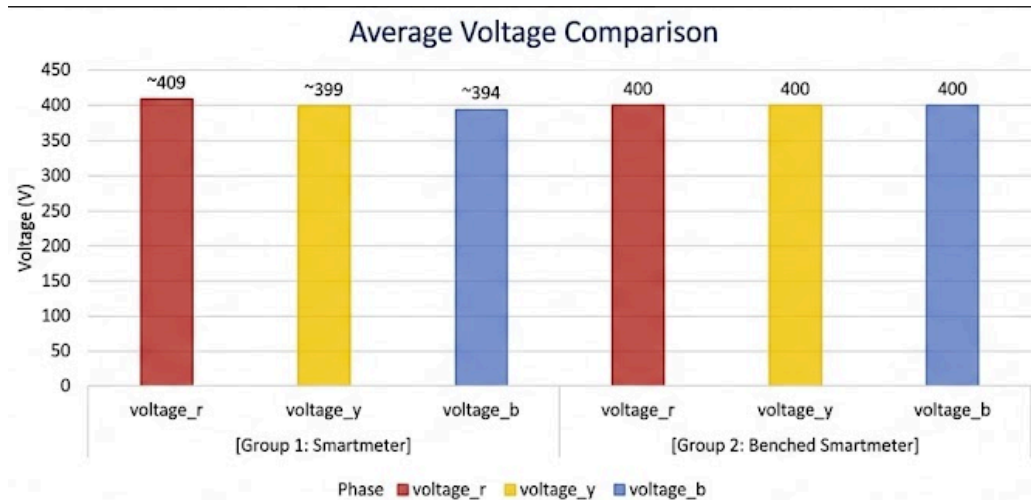


Figure E3: Average voltage comparison of benched smartmeter with actual three-phase smartmeter

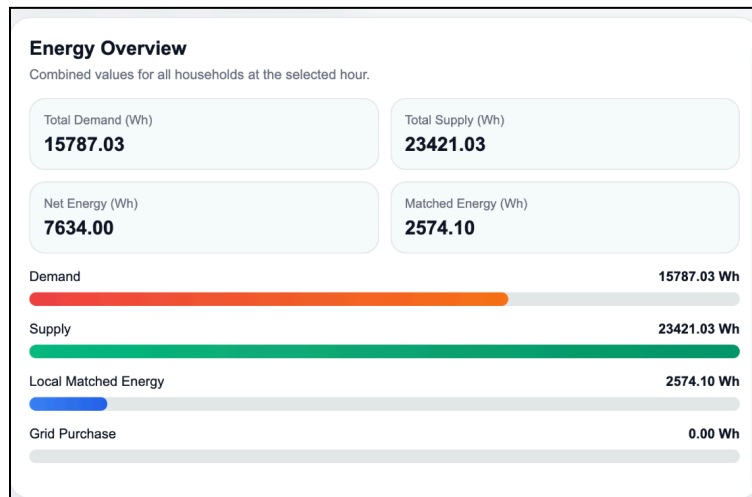


Figure E4: Energy aggregation showing total supply vs total demand

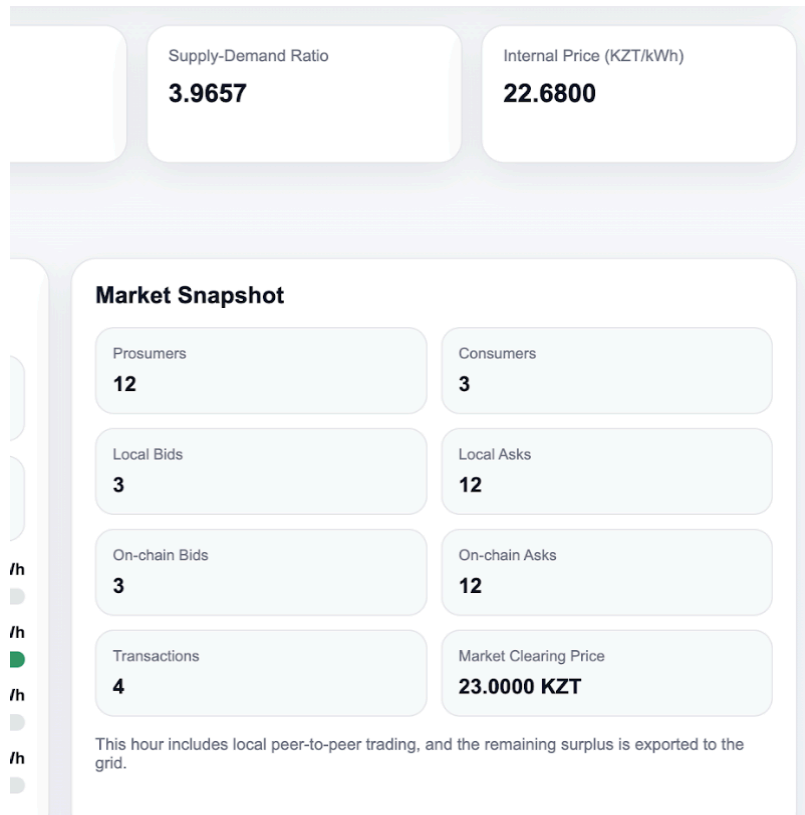


Figure E5: Dashboard market snapshot showing SDR and peer-to-peer trading status

Figure E5 validates R5 and R7 by showing that SDR equal to 3.9657 producing the correct internal price of 22.68 KZT/kWh, and the market clearance logic successfully matching 12 prosumers and 3 consumers with 12 asks and 3 bids.

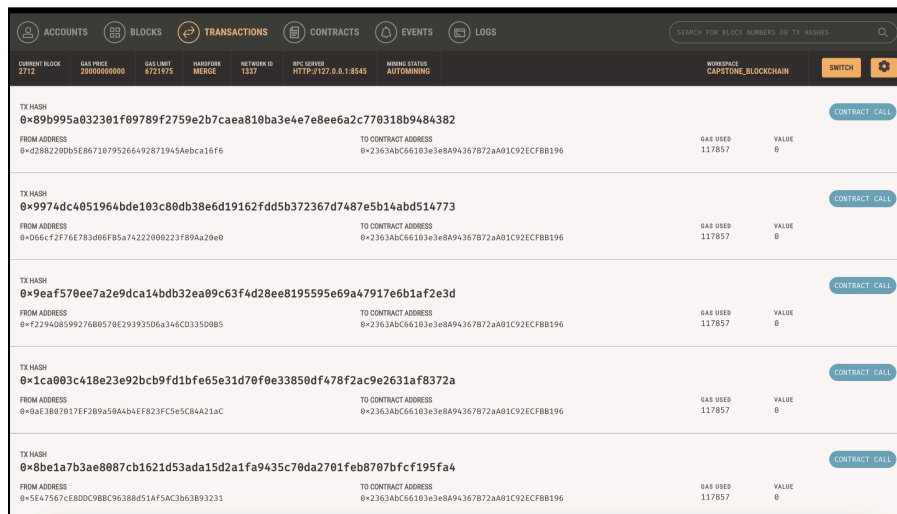


Figure E6: Ganache Blockchain transactions

Figure D16 displays the Ganache blockchain transaction log, where each energy market transaction is recorded as a unique transaction hash, block number, and gas usage, ensuring transparency and tamper-proof record-keeping.

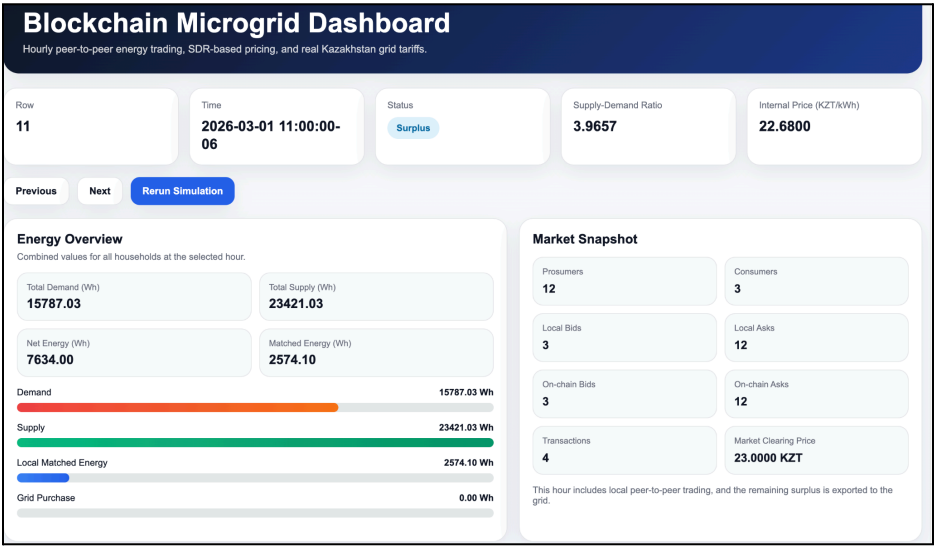


Figure E7: Main Dashboard

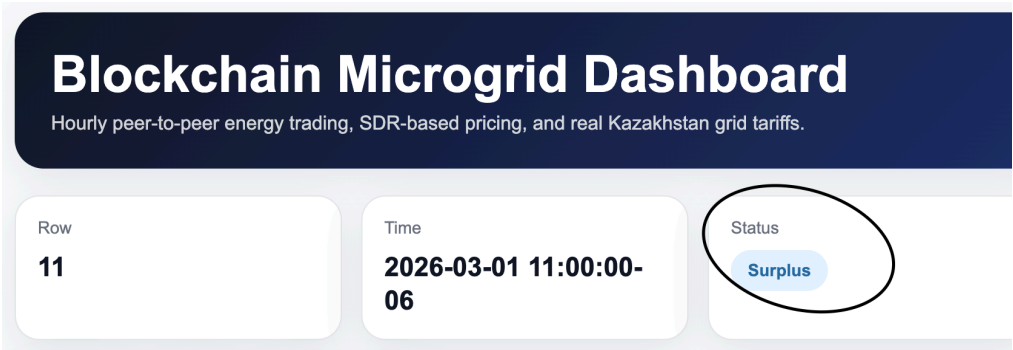


Figure E8: Dashboard header showing grid status



Figure E9a: CB system developed in laboratory

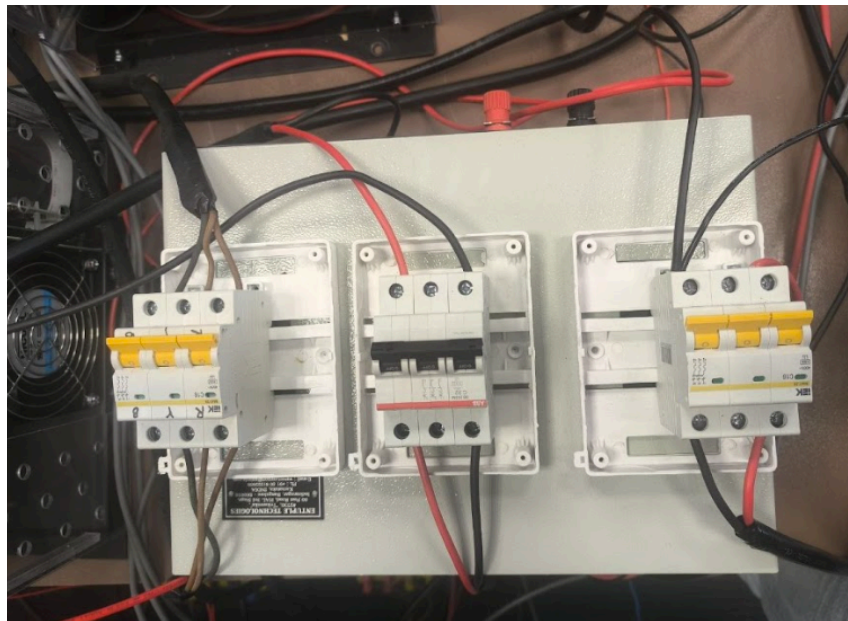


Figure E9b: Circuit Breakers

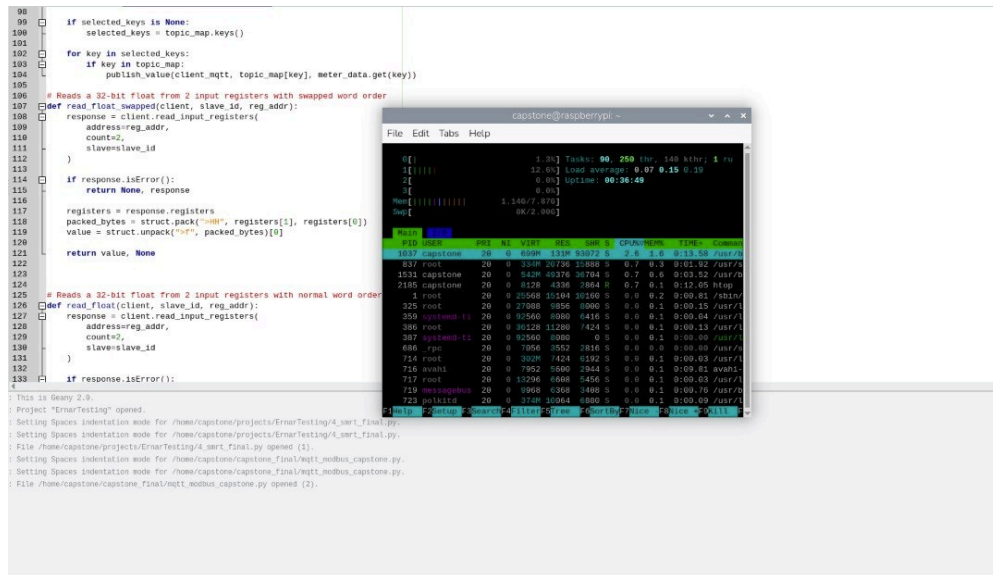


Figure E10: Raspberry Pi 5 CPU load during EMS operation

Figure E11 displays the output of the Linux top command running on the Raspberry Pi 5, showing the EMS Python process consuming 14.55% CPU load over 36 minutes of continuous operation.

Appendix F. Project Management Evidence

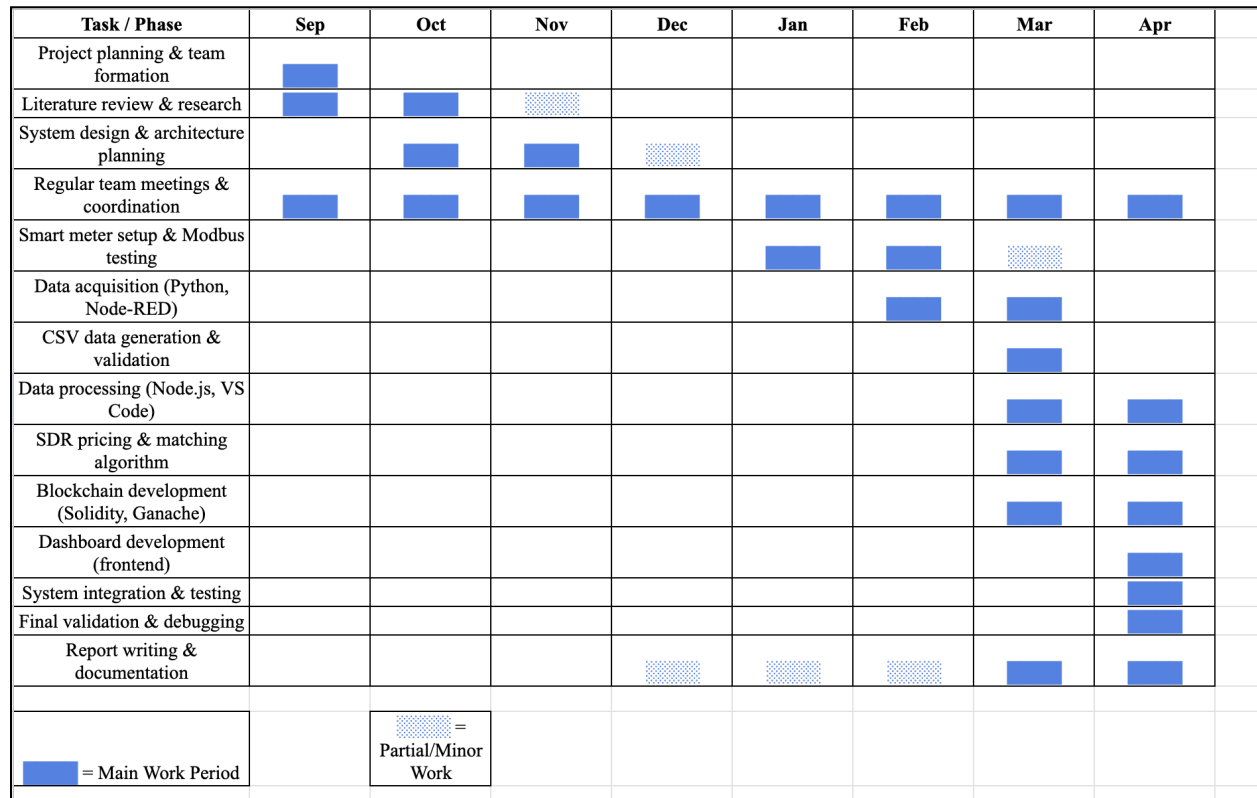


Figure F1: Gantt Chart

To Do	In Progress	Done
Research blockchain deployment (Polygon)	SDR pricing implementation	Project planning & topic selection
Real-time data streaming integration	Dashboard UI improvements	Literature review
System scalability testing (50+ houses)	Report writing (final sections)	System architecture design
Cybersecurity considerations	Final validation & debugging	Smart meter setup (Modbus + Raspberry Pi)
Performance optimization	Blockchain integration testing	Data acquisition (Node-RED + CSV)
		Data processing (Node.js, VS Code)
		SDR pricing & matching algorithm
		Blockchain implementation (Solidity + Ganache)
		Dashboard development
		Full system integration

Figure F2: Task Board Snapshot

Date	Attendees	Topics discussed	Decisions made	Action items
Sep 2025	Team + Supervisor	Team formation, project brainstorming	Initial project directions identified	Explore microgrid and blockchain topics
Oct 2025	Team	Literature review, microgrid concepts	Focus on energy trading systems	Continue research and define scope
Nov 2025	Team + Supervisor	System architecture ideas	Select microgrid trading system approach	Draft initial system design
Dec 2025	Team + Supervisor	Finalizing project scope	Confirm project topic and methodology	Prepare for implementation phase
January 2026	Team + Supervisor	Project scope, microgrid concept, initial design	Selected microgrid energy trading system as project topic	Begin research on smart meters and Modbus
February 2026	Team	Smart meter setup, Modbus communication	Use Pymodbus for data acquisition	Configure Raspberry Pi and test data reading
Feb 2026	Team + Supervisor	Data pipeline design	Use Node-RED to generate CSV datasets	Implement Node-RED flows
Mar 2026	Team	Data processing approach	Use Node.js (VS Code) for simulation and aggregation	Develop CSV processing scripts
Mar 2026	Team	Market logic and pricing	Implement SDR-based pricing model	Code matching algorithm
Mar 2026	Team	Blockchain integration	Use Solidity + Ganache for transactions	Implement BID, ASK, CLEAR_MARKET
Apr 2026	Team	Dashboard design	Develop web-based visualization	Build UI and connect backend
Apr 2026	Team + Supervisor	System integration testing	Integrate all subsystems	Debug and fix synchronization issues
Apr 2026	Team + Supervisor	Final review and validation	Confirm system meets requirements	Prepare report and finalize results

Figure F3: Meeting Summary Table

Key meetings were recorded in the following meeting summary table representing essential parts of the project's development. However, additional meetings were regularly scheduled during

crucial development stages.

Item	Component	Qty	Unit Cost (KZT)	Total (KZT)	Source	Evidence / Notes
1	Smart Meter (1-phase)	1	11,572	11,572	Imported (India)	Used for real data acquisition
2	Smart Meter (3-phase)	3	27,657	82,971	Imported (India)	Used for multi-phase measurements
3	Raspberry Pi 5	1	89,390	89,390	Local supplier	Data acquisition + Node-RED
4	Lab Microgrid	1	0	0	NU C4 Lab	Existing infrastructure
5	Development Software	-	0	0	Open-source	Solidity, Ganache, Node.js
6	Data Processing Tools	-	0	0	Open-source	Node-RED, Python
7	Web Dashboard	-	0	0	Developed	Visualization interface
8	Poster / Presentation	1	3,200	3,200	Local service	Final presentation material

Figure F4: BOM table

Item	Source	Lead Time	Issue Encountered	Mitigation Strategy	Final Outcome
Smart Meters (1P & 3P)	Imported (India, Amazon)	~2 months	Long delivery time	Used simulated CSV datasets during development	Successfully integrated after arrival
Raspberry Pi 5	Local supplier (Kazakhstan)	~1 week	No major issues	Immediate integration into system	Used for real-time data acquisition
Lab Microgrid	NU C4 Lab	Available	Limited access scheduling	Planned testing sessions in advance	Successfully used for validation
Software Tools (Node-RED, Python, Node.js)	Open-source	Immediate	None	Direct installation and use	Fully functional
Blockchain Tools (Solidity, Ganache)	Open-source	Immediate	None	Local blockchain setup used	Successfully implemented transactions
Dashboard (Web UI)	Developed in-house	N/A	Integration complexity	Iterative testing and debugging	Fully integrated

Figure F5: Procurement notes table