



JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY
SCHOOL OF ELECTRICAL, ELECTRONIC AND INFORMATION ENGINEERING
DEPARTMENT OF TELECOMMUNICATION AND INFORMATION
ENGINEERING
BSC PROJECT REPORT

A Blockchain-Enabled Solar Energy Management and Communication System

PRESENTED BY:

1. ENE221-0095/2019 - TREVOR MURIUKI GITURU
2. ENE221-0094/2020 - MICHAEL MURAGE NDEGWA

A project report submitted to the Department of Telecommunication and Information Engineering in partial fulfilment of the requirements for the award of the degree of Bachelor of Science in Telecommunication and Information Engineering.

MAY 2025

DECLARATION

This project report is my original work and has not been presented for a degree in any other University.

Signature: _____ **Date:** _____

Name of Candidate: Trevor Muriuki Gituru

Registration Number: ENE221-0095/2019

This project report is my original work and has not been presented for a degree in any other University.

Signature: _____ **Date:** _____

Name of Candidate: Michael Murage Ndegwa

Registration Number: ENE221-0094/2020

This project report has been submitted for examination with my approval/knowledge as university supervisor.

Signature: _____ **Date:** _____

Name of Supervisor: Dr. Lawrence Ngugi

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to everyone who helped us finish this project successfully.

First and foremost, we would like to express our profound thanks to Dr. Lawrence Ngugi, our supervisor, for his tremendous advice, encouragement, and support during the course of this research. His knowledge and perceptive criticism have greatly influenced the course and results of our work.

We also like to thank Jomo Kenyatta University of Agriculture and Technology's Department of Telecommunication and Information Engineering for providing the tools and supportive environment needed to complete this project.

We would especially want to thank our friends and coworkers, whose cooperation and helpful critique motivated us to think critically and challenge ourselves.

Finally, we would want to sincerely thank everyone who has contributed, no matter how small, to making this project possible

ABSTRACT

This project addressed the inefficiencies in solar energy management and utilization in Kenya by leveraging blockchain technology and advanced communication networks. Even with Kenya's great potential for solar energy and its progress in adopting renewable energy, issues including lack of transparency, a lack of financial incentives, and ineffective energy management still exist. These obstacles make it difficult for small-scale solar energy producers to integrate into the national grid and efficiently monetize excess energy. A transparent, decentralized system for controlling energy production, consumption, and transactions is created by integrating blockchain technology into the suggested solution. By introducing a token-based approach, the system promotes financial empowerment by enabling small-scale producers to earn tradable tokens for excess energy. Blockchain also addresses trust issues and encourages adoption by guaranteeing real-time transaction verification and tamper-proof data storage. Used a hybrid communication infrastructure to enable this system, which consists of Arduino micro controller for local communication. This configuration optimizes energy use while guaranteeing scalability and dependability by enabling real-time monitoring, effective energy distribution, and strong analytics. Higher adoption of solar energy systems, better financial incentives for small-scale producers, improved grid reliability, and greater transparency in energy transactions are all anticipated results of this initiative. The initiative supports worldwide renewable energy breakthroughs and Kenya's Vision 2030 sustainability goals by encouraging clean energy and lowering dependency on fossil fuels.

Table of Contents

DECLARATION	i
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
Table of Contents	v
LIST OF ABBREVIATIONS	x
LIST OF TABLES	xi
LIST OF FIGURES	xii
CHAPTER ONE: INTRODUCTION	xiii
1.1 Background Information	1
1.2 Problem Statement	2
1.3 Objectives of the study	2
1.3.1 Main Objective	2
1.3.2 Specific objectives	3
CHAPTER TWO: LITERATURE REVIEW	4
2.1 Introduction	4
2.2 Solar Energy Management	5
2.2.1 Challenges in solar energy management	5
2.3 Smart Grids and Distributed Energy Resources (DERs)	5
2.3.1 Characteristics of Smart Grids in Solar Energy Management	6
2.3.2 Ways in which blockchain enhances smart grids	6
2.3.3 Key characteristics of DERs in solar energy system:	6
2.3.4 Ways in which blockchain enhance DERs	7
2.3.5 The Role of Blockchain in Enhancing Smart Grids and DERs	7
2.4 Energy Storage and Optimization	7
2.5 Blockchain Technology in Energy Management	8
2.5.1 Blockchain Fundamentals	8

2.5.2 Blockchain for Peer-to-Peer Energy Trading	8
2.5.3 Smart Contracts in Energy Systems.....	8
2.5.4 Energy Storage Optimization.....	9
2.6 Blockchain Technology in Data.....	9
2.7 Communication Systems in Blockchain-Enabled Solar Energy Networks	9
2.7.1 Energy communication network	10
2.8 Data Transmission Technology	10
2.8.1 Internet of Things (IoT) and Blockchain Integration.....	10
2.8.1.1 Role of IoT in Data Transmission:.....	10
2.8.2 Wireless communication Network.....	11
2.8.3 Data Transmission for Energy Storage and Grid Integration	11
2.8.3.1 Grid Communication Protocols	11
2.8.3.2 Virtual Power Plant Communication	11
2.8.4 Role of Blockchain in Data Transmission in solar energy system	12
2.9 Potential benefits of Blockchain in Solar Energy Systems.....	12
2.10 Challenges and Future Directions	12
2.10.1 Scalability and Energy Efficiency of Blockchain Networks	12
2.10.2 Regulatory and Security Concerns.....	13
2.11 Existing Research and studies.....	13
2.12 IoT and Smart Metering for Energy Monitoring	14
2.13 Identified Gaps and Limitations.....	14
2.14 Contribution of This Project	16
CHAPTER THREE: METHODOLOGY	17
3.1 System Design Overview.....	17
3.2 Hardware Components.....	17
3.3 System Architecture.....	18
3.3.1 Description of the components use	18

Solar panel.	18
3.3.2 Solar Energy Generation and Storage	24
3.3.3 Load Management	24
3.3.4 Data Acquisition (Monitoring)	25
3.3.5 Software development	25
3.3.5.1 Overview	25
3.3.5.2 Arduino Software.....	26
3.3.5.3 Power Measurement.....	26
3.3.5.4 Bidirectional Communication	27
3.3.5.5 User Interface.....	28
3.3.5.6 Access Control Logic (ACL)	29
3.3.5.7 Persistence.....	29
3.3.5.8 Timing and Responsiveness.....	30
3.3.5.9 Estate Hub Software	30
3.3.5.10 Robust Arduino Device Communication.....	30
3.3.5.11 Communication Management at the Estate Hub	32
3.3.5.12 Blockchain Integration with Starknet Sepolia	35
3.3.5.13 Smart Contract Methodology: Solaris Conexus Token (SCT)	38
3.3.5.14 Design Strategy	38
3.3.5.15 Token Transfers	39
3.3.5.16 Trade Lifecycle Management	40
3.3.5.17 Consumption and Supply Updates.....	41
3.3.5.18 Contract Upgrades	42
3.3.5.19 Frontend Application (Web Interface).....	42
3.3.5.20 Authentication.....	43
3.3.5.21 Dashboard Home	44
3.3.5.22 Profile.....	45

3.3.5.23 Trade	46
3.3.5.24 Purchase	47
3.3.5.25 Device	47
3.3.5.26 Central Backend Server	48
3.3.5.27 Database Tables (ORM Models)	49
3.3.5.28 API Layer and Communication	49
3.4 Hardware Implementation	51
3.5 Software implementation	52
3.6 Testing and Troubleshooting	53
CHAPTER FOUR: RESULTS AND DISCUSSIONS	54
4.1 Solar Panel	54
4.2 Battery	54
4.2.1 Monitoring battery capacity	55
4.3 Loads (Bulbs).....	55
4.3.1 Household A	55
4.3.2 Household B.....	55
4.4 The feature codes	56
4.5 Power losses.....	59
4.6 Frontend	59
4.6.1 Authentication.....	59
4.6.2 Profile.....	61
4.6.3 Token Purchase.....	64
4.6.4 Trades.....	66
4.6.5 Devices.....	68
4.6.6 Home.....	69
4.7 Central Backend.....	70
4.8 Smart Contract SCT	73

4.9 Estate Hub	74
CHAPTER SIX: CONCLUSION	78
5.1 Introduction.....	78
CHAPTER SIX: REFERENCES.....	80

LIST OF ABBREVIATIONS

A: Amperes

ABI: Application Binary Interface

API: Application Programming Interface

CSS: Cascading Style Sheet

DC: Direct current.

DERs: Distributed Energy Resources.

EVs: Electric Vehicle.

H: Hours.

HTTP: Hyper Text Transfer Protocol

I: Current.

IoT: Internet of things.

JSON: Java script Object Notation

JWT: JSON Web Token

KPLC: Kenya power and lighting company.

LCDs: Liquid crystal display

MOSFETS: Metal-oxide-semiconductor field-effect transistor

MQTT: Message Queuing Telemetry Transport

ORM: Object Relationship Mapping

PAYG: Pay As You Go

PVs: Photo voltaic.

P: Power

SDK: Software Development Kit

SCT: Solaris Conexus Token

STRK: Stark

URL: Uniform Resource Locator

V: voltage

W: watts

WIFI: Wireless Fidelity.

LIST OF TABLES

Table 1: Gaps and Limitations	15
Table 2 :Hardware Components.....	17

LIST OF FIGURES

Figure 1:Solar panel	19
Figure 2:battery	19
Figure 3: Current sensor	20
Figure 4: Step up converter	21
Figure 5: Charge controller	22
Figure 6: MOSFET	23
Figure 7: MOSFET	24
Figure 8:Flow chart Diagram	25
Figure 9:Power measurement code	26
Figure 10:Bidirectional communication	27
Figure 11:User interface	28
Figure 12: Access Logic Unit	29
Figure 13:Persistence	29
Figure 14: Time and Responsiveness	30
Figure 15: Arduino communication	32
Figure 16: Arduino Communication	32
Figure 17: Communication Management	34
Figure 18: Communication management	35
Figure 19: Blockchain Intergration with starknet	37
Figure 20: Design strategy	38
Figure 21: Token transfer	39
Figure 22: Trade lifecycle management	41
Figure 23: Consume logic	41
Figure 24:Contract upgrade	42
Figure 25:House A Implementation	51
Figure 26: House A and B Setup	52
Figure 27:House ID	57
Figure 28:Connection nature	57
Figure 29:Current Value	58
Figure 30:Token balance	58
Figure 31:LED status	58
Figure 32:Authentication	60

Figure 33:Login	61
Figure 34:Profile.....	62
Figure 35:Email confirmation	63
Figure 36:Braavos wallet.....	63
Figure 37:Personal details	64
Figure 38:purchase token	65
Figure 39:updated power Tokens	65
Figure 40:Transaction details	66
Figure 41:Trades.....	66
Figure 42:trade	67
Figure 43:Trade history	68
Figure 44:addition of devices	68
Figure 45:Household details.....	69
Figure 46:Resident dashboard	69
Figure 47:Braavos wallet.....	70
Figure 48:central backend	71
Figure 49: receiving Backend Toggle	71
Figure 50: Client Connection Setting	72
Figure 51:database schema.....	73
Figure 52: Smart Contract SCT	73
Figure 53: Smart Contract SCT	74
Figure 54:callable functions	74
Figure 55:Voyager events	74
Figure 56: Estate Hub.....	75
Figure 57:Power Consumption history.....	76
Figure 58:estate hub receiving MQTT messages	76
Figure 59:estate hub receiving an MQTT command.....	77

CHAPTER ONE: INTRODUCTION

1.1 Background Information

Solar energy is one of the most sustainable and readily available renewable energy sources, providing a feasible answer to the world's energy concerns. Germany, Australia, and India have effectively implemented solar energy regulations and technology, such as net metering and decentralized energy models. These initiatives have allowed households to donate excess energy to the grid, get cash compensation, and actively engage in energy markets. Such initiatives have advanced the adoption of renewable energy in certain regions while also providing economic benefits to small-scale producers.

Kenya, with its advantageous location on the equator, has a similar opportunity to become a solar energy pioneer. The country has plenty of sunshine all year, making it a perfect location for solar power generation. The proliferation of small-scale solar systems, particularly through new financing mechanisms such as PAYG, has increased access to clean energy in off-grid locations. This progress has reduced reliance on fossil fuels, increased sustainability, and provided electricity to marginalized communities.

However, Kenya's solar energy sector continues to encounter serious obstacles. Unlike other nations that offer direct cash incentives for excess energy, Kenya's current net metering legislation, enforced by the KPLC, only allows for energy credits. This diminishes the economic benefits for small-scale producers and the motivation to use solar systems. Households frequently create excess energy that goes unused because there is no system in place to monetize or trade this surplus energy, preventing them from fully using their investment.

Furthermore, a lack of openness in energy transactions breeds mistrust among users. Many PAYG systems do not keep detailed records of energy production, usage, or compensation, leaving users unclear about the financial and operational value of their solar systems. Furthermore, Kenya's existing communication infrastructures are inadequate for supporting advanced energy management options such as peer-to-peer energy trading or decentralized energy systems. These obstacles jointly impede the scalability and efficiency of solar energy adoption in the country.

To address these difficulties, combining blockchain technology with an efficient communication infrastructure offers a breakthrough possibility. Blockchain provides a secure, decentralized, and transparent platform for tracking and trading surplus energy, promoting trust and fairness in energy transactions. By integrating this with strong communication networks,

Kenya can transform its solar energy sector, producing a model that maximizes renewable energy use while also aligning with the country's overall sustainability goals.

1.2 Problem Statement

While global advances in solar energy, such as those in Germany, Australia, and India, show the potential of decentralized energy systems, Kenya's solar energy sector faces significant challenges that impede progress. Despite the country's enormous solar resources and rapid adoption of Pay-As-You-Go (PAYG) solar systems, substantial gaps prevent the full use of this renewable energy.

Key challenges include:

- i. **Insufficient Financial Incentives:** The KPLC's current net metering policy provides energy credits rather than direct financial compensation, deterring households from investing in solar systems.
- ii. **Lack of Transparency:** PAYG systems frequently fail to offer accurate records of energy output, consumption, and compensation, which breeds mistrust and limits adoption.
- iii. **Inadequate Communication Infrastructure:** Kenya lacks the necessary infrastructure for real-time monitoring, peer-to-peer energy trading, and distributed energy management.

These challenges not only limit the scale and efficiency of solar energy use, but also jeopardize Kenya's ability to reach its sustainability targets. To overcome these obstacles and realize the country's solar energy potential, a new system that improves transparency, incentivizes energy production, and incorporates advanced communication technologies is urgently required.

1.3 Objectives of the study

1.3.1 Main Objective

To accelerate the adoption of renewable energy in Kenya, an innovative solar energy management system will be designed and implemented that incorporates blockchain technology and advanced communication networks, enhancing transparency, incentivizing surplus energy production, and fostering peer-to-peer energy trading while addressing scalability, efficiency, and trust challenges.

1.3.2 Specific objectives

- i. Create a blockchain-based energy management system: Create and install a secure, permission-based blockchain for transparently recording and managing energy production, consumption, and transaction data.
- ii. Create an efficient communication network: Implement a hybrid communication system that combines the entire system.
- iii. Introduce a token-based incentive mechanism: Create a system that pays homes with tradable energy tokens for surplus energy, providing immediate cash benefits while incentivizing sustainable energy production.
- iv. Facilitate peer-to-peer energy trading: Allow for the decentralized exchange of surplus energy via blockchain smart contracts, hence stimulating local energy markets and maximizing use.
- v. Enable real-time monitoring and insights: Implement technologies for real-time monitoring of energy consumption and output, backed up by analytics dashboards, to improve decision-making and system optimization.

CHAPTER TWO: LITERATURE REVIEW

2.1 Introduction

As the demand for decentralized, sustainable energy systems grows, integrating blockchain technology with solar energy management has become a significant research focus. Researchers aim to create secure, transparent, and autonomous energy trading and monitoring systems that empower users to manage energy production and consumption efficiently. This literature review explores existing work related to blockchain in energy systems, solar energy storage and load management technologies. It highlights the current gaps this project aims to address.

Kenya, with its advantageous location on the equator, has a similar opportunity to become a solar energy pioneer. The country has plenty of sunshine all year, making it a perfect location for solar power generation. The proliferation of small-scale solar systems, particularly through new financing mechanisms such as PAYG, has increased access to clean energy in off-grid locations. This progress has reduced reliance on fossil fuels, increased sustainability, and provided electricity to marginalized communities.

However, Kenya's solar energy sector continues to encounter serious obstacles. Unlike other nations that offer direct cash incentives for excess energy, Kenya's current net metering legislation, enforced by the KPLC, only allows for energy credits. This diminishes the economic benefits for small-scale producers and the motivation to use solar systems. Households frequently create excess energy that goes unused because there is no system in place to monetize or trade this surplus energy, preventing them from fully using their investment.

Furthermore, a lack of openness in energy transactions breeds mistrust among users. Many PAYG systems do not keep detailed records of energy production, usage, or compensation, leaving users unclear about the financial and operational value of their solar systems. Furthermore, Kenya's existing communication infrastructures are inadequate for supporting advanced energy management options such as peer-to-peer energy trading or decentralized energy systems. These obstacles jointly impede the scalability and efficiency of solar energy adoption in the country.

2.2 Solar Energy Management

Solar energy systems, particularly photovoltaic (PV) systems, are being deployed at an increasing rate globally, as they offer a sustainable and renewable energy source. Solar energy is also a clean form of energy. However, solar power production is intermittent due to varying weather conditions and the time of day, which makes energy storage and distribution complex.

2.2.1 Challenges in solar energy management.

- i. **Energy Storage:** Batteries and other storage systems are required to hold excess energy for later use, but these systems are expensive and can be difficult to integrate into existing grid structures.
- ii. **Energy Trading:** Solar energy producers, especially in decentralized systems, may have excess energy that can be traded with others. However, current energy trading systems can be inefficient and expensive.
- iii. **Monitoring and Control:** Solar systems require continuous monitoring for performance optimization. Traditional centralized systems may be unable to control the dynamic nature of solar energy generation and consumption.
- iv. **Interruptive:** Solar power generation depends on sunlight, which vary. To ensure consistent supply, solar energy needs to be stored or supplemented by other energy sources, which can be costly and complex to manage.

2.3 Smart Grids and Distributed Energy Resources (DERs)

The concept of smart grids has been widely explored to integrate renewable energy into the existing energy infrastructure. A smart grid is an electrical grid that uses digital communication technology to monitor and manage the flow of electricity from both conventional and renewable sources. Distributed Energy Resources (DERs), such as solar panels, energy storage systems (batteries), and electric vehicles (EVs), are essential components of a smart grid. These resources can be managed more effectively with decentralized and automated solutions, such as blockchain, which allow for secure and transparent peer-to-peer energy trading.

Research done on blockchain-based smart grids has shown that blockchain can enable real-time data sharing between energy producers and consumers, ensuring transparency and reducing the risks associated with fraud, data manipulation, and energy theft [1].

2.3.1 Characteristics of Smart Grids in Solar Energy Management

Automation and Control: With Smart Grids, energy distribution can be optimized based on demand. Blockchain-enabled smart contracts can automate energy transactions, for example, automatically selling excess energy from solar panels to nearby consumers or storing it in batteries when there is a surplus.

Two-way Communication: Smart Grids use real-time data collection and communication between the energy producer (e.g., solar panels) and the consumer. The consumer is able to know the amount of energy consumed and amount needed to pay. Blockchain enhances this communication by providing a secure, immutable ledger for tracking and verifying energy transaction.

Energy Flow Monitoring: Smart Grids can monitor energy flow continuously, but the integration of blockchain guarantees that this data is secure, transparent, and tamper-proof. This feature is important for solar energy systems, where data such as energy production, consumption, and storage need to be accurate for billing, settlements, and performance monitoring.

2.3.2 Ways in which blockchain enhances smart grids

Grid Resilience: By decentralizing energy generation and distribution, Smart Grids integrated with blockchain can become more resilient to system failures. Distributed generation (such as solar) can provide backup power to the grid, ensuring that localized issues do not lead to widespread blackouts.

Improved Security and Trust: Blockchain enhances cybersecurity of Smart Grids by ensuring that all data exchanged between producers, consumers, and other participants are secure and cannot be tampered with.

Decentralized energy trading. Through blockchain solar energy can be traded between users in a decentralized way without requiring a centralized utility.

2.3.3 Key characteristics of DERs in solar energy system:

- i. **Decentralized generation.** It enables energy to be generated locally and not relying on a centralized power generation plant.
- ii. **Energy Storage:** Many DER systems include battery storage that allows excess solar energy to be stored for later use, improving the stability and reliability of solar-powered systems.

- iii. Scalability: DERs are adaptable and scalable, allowing users to add more generation or storage capacity as needed. An example is solar Panel; one can install a certain number and then add the panels with time based on energy demand.

2.3.4 Ways in which blockchain enhance DERs

- i. Transparent and Secure Transactions: Blockchain ensures that all transactions between solar producers and energy consumers are recorded transparently and securely. This ensures that solar energy exchange is verifiable and fair.
- ii. Peer-to-Peer Energy Trading: Blockchain facilitates peer to peer energy trading between solar producers and consumers. Energy producers with excess solar energy can sell their excess directly to consumers. This decentralized trading system can operate efficiently with the help of smart contracts.
- iii. Optimized Energy Storage: Blockchain can help track and manage the use of energy storage systems. Smart contracts can automatically store excess solar energy in batteries when there is a surplus and distribute it when demand is high.

2.3.5 The Role of Blockchain in Enhancing Smart Grids and DERs

Energy Data management: data generated by Smart Grids and DERs, including energy production, consumption, and storage, is critical for optimizing energy systems. Blockchain ensures that this data is accurate, tamper-proof, and accessible to all stakeholders involved in energy trading, monitoring and control.

Grid Stability and Resilience: Blockchain can improve the stability and resilience of Smart Grids and DERs by allowing local energy production to contribute to grid balancing. Solar energy systems can support grid demand by supplying energy during peak times or during outages.

Smart Contracts for Automation: Smart contracts help in managing decentralized energy transactions. These contracts can automate everything starting with the distribution of energy credits to payments for energy consumption.

2.4 Energy Storage and Optimization

Efficient storage of solar energy remains a significant challenge, as energy demand often peaks when solar energy production is low. Technologies such as lithium-ion batteries,

flow batteries, and even hydrogen storage systems are being developed to mitigate this challenge. Blockchain can support decentralized energy storage systems by ensuring transparent management of energy credits and maintaining secure transaction records.

Previous research explored blockchain's potential to improve energy storage and optimization by enabling microgrid operators to share storage capacity and allocate energy in a decentralized manner. By incorporating smart contracts, blockchain could automate the buying and selling of energy storage capacities based on supply and demand.

2.5 Blockchain Technology in Energy Management

2.5.1 Blockchain Fundamentals

Blockchain is a distributed ledger technology that securely records transactions across a decentralized network of computers. Each block in a blockchain contains data and is linked to previous blocks, forming a chain that is resistant to tampering or alteration. The most well-known application of blockchain is in cryptocurrency, but its potential extends to many industries, including energy management.[11]

2.5.2 Blockchain for Peer-to-Peer Energy Trading

Blockchain allows for peer-to-peer energy trading, where individuals or organizations with surplus solar energy can sell their excess energy to others. This eliminates intermediaries, reduces costs, and enhances energy efficiency by enabling decentralized transactions.

Research highlighted the potential of blockchain-based peer-to-peer energy trading systems in solar energy markets. These systems ensure transparency in transactions, provide secure payment mechanisms, and enable automatic settlements using smart contracts [2,9,12].

2.5.3 Smart Contracts in Energy Systems

Smart contracts are self-executing contracts with the terms of the agreement directly written into code. In the context of solar energy, smart contracts can automate various processes such as energy trading, billing, and payments, based on predefined conditions. A smart contract ensures that once the conditions are met, the agreed-upon transaction occurs automatically without the need for intermediaries.

Previously it was demonstrated that blockchain-integrated smart contracts could automate the distribution of solar energy credits based on real-time data from solar energy

producers and consumers. This reduces the complexity and human intervention involved in managing energy systems.

Blockchain can enhance grid management by enabling more efficient and automated communication between solar power producers, consumers, and utilities. By integrating blockchain into smart grids, utilities can have better control over energy distribution, balancing supply and demand more effectively.

2.5.4 Energy Storage Optimization

Solar energy requires efficient storage systems to ensure a reliable supply. Blockchain can facilitate the management of energy storage, enabling real-time tracking of energy stored and consumed. This can be implemented by technologies like LoRa technology. It can also support the optimization of energy distribution by providing transparency and reliability.

Security: Blockchain uses encryption to secure data, ensuring that energy consumption and production data cannot be tampered with, increasing the integrity of energy systems.

2.6 Blockchain Technology in Data

It enhances transparency because every step of production, distribution and consumption process is recorded on the blockchain making it easy to be accessible.

Enhance secure data recording. Blockchain provide a secure platform for recording generation and consumption of energy. This makes sure that the transactions are verifiable which help to build trust among the stakeholders.

Enhance better grid management. Blockchain helps to provide real time update on energy usage data. This allows effective balancing of supply and demand on the grid improving operations and reducing peak load requirements.

2.7 Communication Systems in Blockchain-Enabled Solar Energy Networks

The integration of communication systems within blockchain-enabled solar energy networks is critical for real-time data exchange, control, and monitoring. Communication networks ensure that devices, sensors, and smart meters can interact efficiently within the solar energy system. An example of technology used for communication is LoRa gateway, Fiber optic and Zigbee.

2.7.1 Energy communication network

Power line communication: in solar energy system it can be used for local communication between smart meters, solar panels, inverters, and other devices within a microgrid.

Ethernet and Fiber Optics: For large-scale solar farms or centralized systems, Ethernet and fibre optics offer high-speed data transmission capabilities, ensuring that real-time data from thousands of sensors can be efficiently transmitted to centralized control systems and the blockchain network.

2.8 Data Transmission Technology

Data transmission technologies play a critical role in ensuring real-time, secure, and efficient communication between solar energy systems, blockchain network and users.

The technologies to be used in data transmission are.

2.8.1 Internet of Things (IoT) and Blockchain Integration

The Internet of things plays a crucial role in modern solar energy management systems by enabling the collection of real-time data from solar panels, batteries, and other devices.

Blockchain's decentralized nature ensures that data from these devices are recorded efficiently and transparently. Integrating blockchain with IoT devices enables energy systems to self-optimize and respond to demand fluctuations efficiently.

A study done on Internet of things-based solar energy management with blockchain integration demonstrated that this combination could significantly improve the scalability and efficiency of solar energy systems. Data collected from solar panels and smart meters were securely stored on the blockchain, allowing for accurate performance monitoring and real-time system optimization.

2.8.1.1 Role of IoT in Data Transmission:

Communication with Blockchain: IoT devices can communicate directly with blockchain networks through gateways or intermediary systems, sending data for validation, recording, and smart contract execution.

Data Collection: IoT sensors attached to solar panels and energy storage systems gather key data points such as energy production, consumption, temperature, battery status, and overall system performance.

Real-Time Monitoring: These devices send data in real time to a central system, allowing energy producers, grid operators, and consumers to track energy usage, storage levels, and solar power generation.

2.8.2 Wireless communication Network

- i. Zigbee technology: ZigBee is a low-power, low-data-rate wireless communication protocol designed for smart devices in energy management systems. It is well-suited for home automation, smart meters, and energy management in decentralized systems
- ii. Lora technology: Lora is a low-power wide-area network protocol for IoT devices that need to transmit small amounts of data over long distances. Advantage of LoRa is that it consumes low power it is long range and scalable.
- iii. Wi-Fi: Wi-Fi is commonly used in smart devices for short-range communication. Used in residential and commercial solar energy systems. It allows for high-speed data transmission and is easy to deploy in environments where a local network is available.

2.8.3 Data Transmission for Energy Storage and Grid Integration

Integrating solar energy with storage systems and the broader grid involves data transmission for optimizing energy distribution and ensuring system stability. Some of the protocols that enhances this include:

2.8.3.1 Grid Communication Protocols

Protocols like IEC 61850 (for electrical substation automation) and Modbus (for data exchange between intelligent devices and control systems) are widely used for data transmission in energy grid management. These protocols can be adapted for solar systems that interface with the grid or energy storage devices.

2.8.3.2 Virtual Power Plant Communication

A Virtual Power Plant integrates multiple decentralized energy systems (including solar power, storage, and other renewables) into a single network. Communication technologies such

as Distributed Network Protocol and Automated Demand Response are used for grid communication in Virtual Power Plant.

2.8.4 Role of Blockchain in Data Transmission in solar energy system

- i. Transparent Transactions: Blockchain guarantees that all participants in the energy ecosystem can view and verify data related to energy production, consumption, and trading.
- ii. Data integrity and security: Blockchain ensures that all the energy transaction records are securely stored and easily accessible. Once a record of energy transaction is stored it cannot be altered ensuring integrity and security of data.

2.9 Potential benefits of Blockchain in Solar Energy Systems

1. Cost efficiency. By using blockchain technology it can potentially reduce the transactions associated with energy trading. By removing intermediaries and automating processes through smart contracts, blockchain reduces transaction costs. This can lead to more affordable solar energy for consumers and higher profits for solar producers.

2. Transparency. Blockchain enables data stored in the platform to be publicly accessible. This helps to improve trust among participants in the solar energy market. This is especially important in peer-to-peer energy trading, where participants need assurance that transactions are fair and secure.

3. Scalability: Blockchain can support the growth of decentralized solar energy systems by allowing the seamless addition of new producers and consumers without disrupting the existing infrastructure.

4. Increased security. Decentralized architecture of blockchain minimizes of centralized data storage making it less vulnerable to cyber security attacks thus improving security.

2.10 Challenges and Future Directions

2.10.1 Scalability and Energy Efficiency of Blockchain Networks

Besides Blockchain having the potential in solar energy management, scalability remains a key challenge. Traditional blockchain systems, such as Bitcoin, consume a lot of

energy for transaction validation. The environmental impact of these blockchain systems needs to be considered in the context of solar energy management.

2.10.2 Regulatory and Security Concerns

The use of blockchain in energy systems introduces new security concerns, particularly with the integration of smart contracts and IoT devices. Ensuring the integrity and security of data shared across a decentralized network requires strong cryptography.

2.11 Existing Research and studies

Several studies have demonstrated blockchain's potential for revolutionizing energy markets:

- Mengelkamp et al. (2018) proposed peer-to-peer (P2P) energy trading platforms where residential solar energy producers could sell excess energy using smart contracts. Their work showed that blockchain can eliminate intermediaries and lower transaction costs.[3]
- Sousa et al. (2019) explored blockchain for local energy communities, allowing microgrid participants to trade energy autonomously while maintaining secure records.
- Power Ledger (commercial project) has implemented real-world trials of blockchain-enabled energy trading in Australia, showing significant promise for decentralized energy exchange networks. Power ledger helps create a market for those small-scale generations. It allows peer-to-peer energy trading between neighbors, businesses and the grid. The technology is scalable. And it allows power to be bought and sold in real time.[4]
- Peer to peer energy trading among smart homes. This paper evaluates the impact of Peer-to-Peer (P2P) energy trading among the smart homes in a microgrid. Recent trends show that the households are gradually adopting renewables (e.g., photovoltaics) and energy storage (e.g., electric vehicles) in their premises. This research addresses the energy cost optimization problem in the smart homes which are connected together for energy sharing. The contributions of this paper is first, we propose a near-optimal algorithm, named Energy Cost Optimization via trade, which coordinates peer to peer energy trading among the smart homes with a Demand Side Management system. Our results show that, for real

datasets, 99% of the solutions generated by the energy cost optimization-Trade algorithm are optimal solutions. Second, peer to peer energy trading in the microgrid potentially results in an unfair cost distribution among the participating households. We address this unfair cost distribution problem by enforcing Pareto optimality, ensuring that no households will be worse off to improve the cost of others. Finally, we evaluate the impact of renewables and storage penetration rate in the microgrid. Our results show that cost savings do not always increase linearly with an increase in the renewables and storage penetration rate. Rather they decrease gradually after a saturation point.[5]

- Research in microgrid solar energy systems has focused heavily on battery storage management, load prioritization, and system optimization using conventional control systems like microcontrollers (e.g., Arduino).[6]
- Some studies highlight the importance of monitoring solar power production and consumption for maximizing efficiency. They described current experience and trials in East Timor with solar photovoltaic (PV) technology by introduction of solar home systems. Also proposed Solar Home Systems that are commercially disseminated and used them cost efficiently to substitute kerosene and dry cell batteries to reduce gas emissions and thus make a significant contribution to climate protection. [7]

2.12 IoT and Smart Metering for Energy Monitoring

IoT based smart meter. A smart meter is designed for both industrial and domestic users. The meter is responsible for acquiring data, displaying and updating/uploading over the cloud to allow the grid owner to access that data, monitor and take respective actions (if necessary). The models are designed using Arduino. Wireless connectivity is achieved through Bluetooth modules and circuits are designed to depict the actual operations of the system. Architecture of the proposed system. [8 ,10]

Systems like these, however, often involve sophisticated hardware setups (Wi-Fi modules, GSM, cloud databases), which can add complexity and cost.

2.13 Identified Gaps and Limitations

Despite the promising developments, several limitations persist in current research and implementations:

Aspects	Limitations in existing work
Hardware simplification	Most blockchain energy projects use complex IoT architectures; very few demonstrate basic, cost-effective setups using simple microcontrollers like Arduino
Focus on Large Systems	Research largely focuses on community- scale or grid-scale energy trading. Little work exists on small, individual solar energy systems suitable for rural, off-grid users.
Direct Load Control Integration	Few studies have tightly integrated load management (switching loads ON/OFF) with blockchain recording in real time. Most focus only on energy transactions, not physical control.
Real-Time Current Monitoring	Although current sensing is standard, few works combine live current monitoring, user-controlled load operation, and immediate blockchain transaction logging.
Cost and Accessibility	Implementations often assume high-end hardware availability (Wi-Fi, cloud access). This limits adoption in low- resource environments, where a minimal, affordable system is needed.
Security and Usability	While blockchain ensures security at the data level, user interfaces (keypad inputs, LCDs) in many systems are not optimized for simple, non-technical users, leaving a usability gap.

Table 1: Gaps and Limitations

2.14 Contribution of This Project

This project addresses the identified gaps by:

- Building a simple, low-cost solar energy monitoring and control system using Arduino, MOSFETS, and basic sensors.
- Focusing on individual-level solar usage with real-time monitoring and control, suitable for small homes, off-grid users, or educational purposes.
- Ensuring live recording of load activation and energy usage onto a private blockchain network without complex IoT frameworks.
- Emphasizing ease of use, with user interaction through a basic keypad and LCD display, making the system accessible to non-technical users.

CHAPTER THREE: METHODOLOGY

3.1 System Design Overview

This project aimed to design and implement a small-scale solar energy management system enabling peer-to-peer energy trading between two households using blockchain technology. The system is based on:

- Solar energy generation (individual solar panels per household)
- Energy storage (battery)
- Load consumption (bulbs).
- Real-time current sensing.
- Microcontroller-based data acquisition (Arduino).
- Web interface displaying consumption.
- Blockchain for secure and transparent energy transactions.

3.2 Hardware Components

COMPONENT	DESCRIPTION
8.4V Solar Panels	Solar energy generation for each household
Battery	Energy storage for each household (12 V ,7.2Ah)
MOSFETs and a manual switch	Automation of ON/OFF control of bulbs
Current Sensors	To measure real-time energy usage (e.g., ACS712 sensor)
Arduino Boards	Data acquisition, processing, and control
Website Dashboard	For monitoring energy consumption data
Blockchain Backend	Smart contract for peer-to-peer energy trading records
Charge controller	It is used to regulate charging of the battery to avoid overcharging.
Boost dc-dc converter	Used to increase the input voltage to a higher output voltage while reducing current to conserve energy.

Table 2 :Hardware Components

3.3 System Architecture

One household A has:

- Solar panels (8.4V and 9V).
- One 12 V battery
- Current sensor (ACS712)
- Arduino controller
- Two bulbs as load (7W and 3W)
- A keypad.
- Voltage divider.

The Arduino monitors the current consumption and controls various components like the keypads, LCD, MOSFETS. The website displays live consumption per household. The blockchain smart contracts handle the trading records between households.

Household B has:

- Current sensor (ACS712)
- Arduino controller
- One bulb (7 Watts)
- A keypad.
- Voltage divider (5V).

The first household supply excess power to the second household up to a certain level.

3.3.1 Description of the components use

Solar panel.

The solar panel is used to collect light energy and convert it to electrical energy.

It helps in providing renewable power to home premises.

Connected two solar panels in series with a rating of 8.4V and 9 V for battery charging.

The total voltage output of the two solar panels is 17.4 V. The solar panel voltage was stepped down to about 14.4 V that is optimum for charging the 12V battery using a buck dc-dc converter.



Figure 1: Solar panel

Battery

The role of the battery is to store energy that is collected from the solar panel. The type of battery used is a 12V 7.2Ah/20HR sealed lead acid battery. It supplies DC current to the bulbs used.



Figure 2: battery

Current sensor

A current sensor is used to measure the current flowing through the load, whether AC or DC. It uses the principal Hall effect where when current flows through the conductor inside the sensor it creates a magnetic field. The sensor produces analog output proportional to the current. The specification of the current sensor used is a 10 A ACS712. It has three pins. The

ground pin is connected to the ground of the circuit, the power pin is connected to the 5V of the Arduino input and the Output pin is connected to the analog pin of the Arduino. It has two terminals (P+ and P-). P+ is connected to the power source and P- is connected to the loads.



Figure 3: Current sensor

Boost converter

A boost converter is used to increase the input voltage to a higher output voltage. Its main role is to power devices or charge batteries that require a higher voltage than the input source provides. Some of the key roles of the boost converter is to boost voltage, enable devices to run at a stable voltage even as battery voltage drops, used to match a low voltage solar panel to a higher battery charging voltage.

For this project the boost converter is used to ensure that the 12v dc bulbs used run at a stable voltage even when the battery capacity is slightly below 12v.

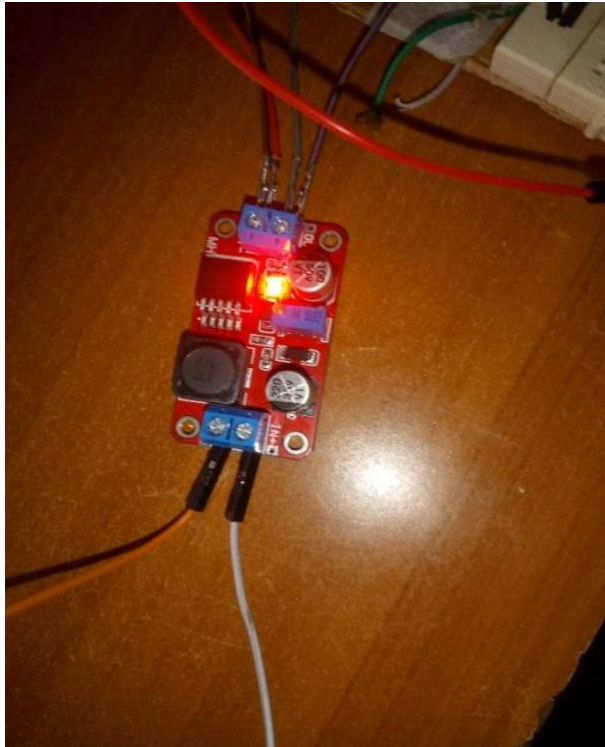


Figure 4: Step up converter

Charging controller

A charging controller is used to protect the battery against overcharging. Once the battery is full the charge controller does not allow any more voltage to flow to the battery from the solar panel. This helps to improve the lifespan of the battery. Designed the charging controller using the following components:

- Buck dc dc converter-It is used to step down solar panels voltage from 17.4V to 14.4V optimum for battery charging.
- Schottky diode-Used to prevent reverse flow of current
- Arduino micro controller-Reads the capacity of the battery and is responsible for switching ON and OFF the MOSFET.
- MOSFET-Act as a switch, it turns OFF when the battery is full to prevent overcharging.
- Capacitors-used two 0.22 μ F electrolytic capacitors which are sued for high frequency filtering. Also used 470 μ F and 1000 μ F capacitors used tonto reduce ripple and buffers load changes. The 470 μ F and 0.22 μ F capacitors are connected in parallel to each other and are connected between the positive terminal of the solar panel and the input positive terminal of the buck converter. Placed the 1000 μ F and 0.22 μ F in parallel to each other between the Positive output of the buck converter and the ground.

- 10K Ω and 100 Ω resistors. -10K Ω resistor is connected between the gate of the MOSFET and the source. Prevents floating current when switching OFF the MOSFET. The 100 Ω resistor is connected between the gate of the MOSFET and the pin going to the Arduino (pin 12).
- Current sensor. Senses the amount of current flowing to the battery. Ground of the Current sensor is connected to the common ground of the circuit, The Vcc pin is connected to the 5V input of the Arduino, Out is connected to the analog pin of the Arduino (A2). One of the terminals of the current sensor (P+) is connected to the positive terminal of the buck converter output, the other terminal of the current sensor (P-) is connected to the positive terminal of the battery.
- Voltage divider. It is used to read the capacity level of the battery and is monitored by the Arduino.

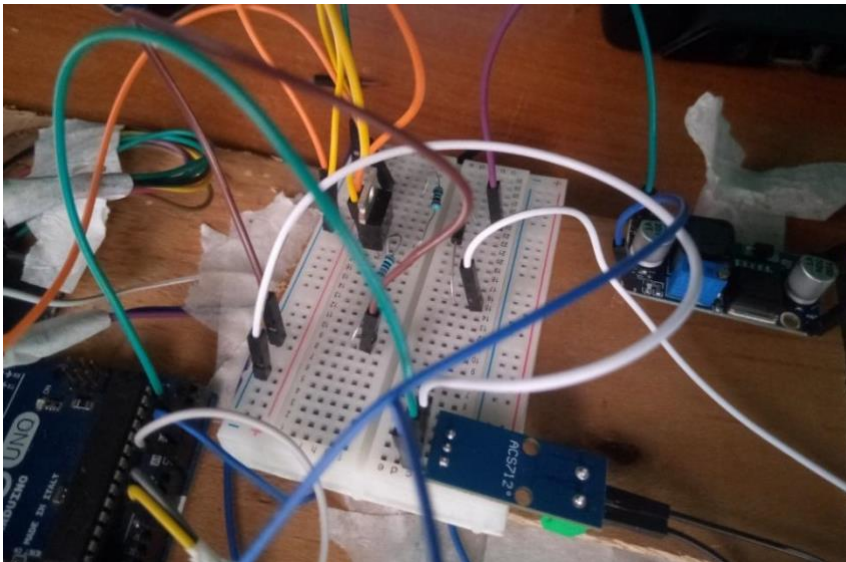


Figure 5: Charge controller

Design procedure

- The positive terminal of the solar panel is connected to the anode terminal of the diode.
- The ground of the solar panel is connected to the ground of the input of the buck converter.
- The positive input of the buck converter is connected to the cathode of the diode.
- The output ground (Vout-) of the buck converter is connected to the common ground of the circuit.
- The positive output terminal of the buck converter is connected to one of the terminals of the current sensor (P+).

- The Battery positive terminal is connected to the other terminal of the current sensor (P-). The negative terminal of the battery is connected to the drain of the MOSFET.
- The MOSFET source is connected to the ground, the gate is connected to the digital pin of the Arduino (pin 12), the drain is connected to the negative terminal of the battery.

Arduino Micro controller

A micro-controller is a programmable device that includes a microprocessor, memory, and input/output signal lines on a single chip fabricated using very large-scale integrated technology (VLSI). The microcontroller that was selected for the project is the Arduino UNO. The Arduino Uno is built around the ATmega328P microprocessor. It features 14 digital input/output pins (6 of which can function as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header, and a reset button. The Arduino Uno is programmed using the Arduino Integrated Development Environment (IDE). It can be powered either through a USB connection or through an external power supply. When using an external power supply, the recommended voltage range is 7 to 12 volts. Supplying less than 7V may result in the 5V pin providing insufficient voltage, potentially causing instability. Conversely, supplying more than 12V could cause the voltage regulator to overheat, risking damage to the board. In this project, power is supplied directly through the 5V pin.

MOSFET

Used IRF540 N-Channel MOSFET. It has three terminals; gate, drain and source. In this project it was used as a switch where the drain is connected to the negative terminal of the power source, the source is connected to the common ground, the gate is connected to the Arduino digital pin (example pin 12) and it used to control the switching mechanism. There is also connected to 10K Ω resistor between the source and the gate.



Figure 6: MOSFET

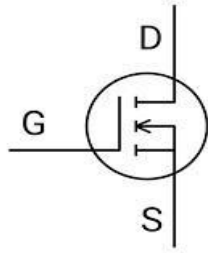


Figure 7: MOSFET

Voltage divider

A voltage divider is used to help reading the value of the voltage capacity of the battery. Since the Arduino input is 5V and the battery output is 12V, the 12V is scaled down to 5V a value that is readable by the Arduino. Used $6.8K\Omega$ and $4.7K\Omega$ resistors connected in series. The positive input of the battery is connected to one end of the $6.8K\Omega$ resistor. The other end of the $6.8K\Omega$ is connected to the same junction with one end of the $4.7K\Omega$ resistor. The negative terminal of the of the battery is connected to the other end of the $4.7K\Omega$ resistor. Then the analog pin of the Arduino is connected at the junction of the two resistors.

Keypad

It is used to feed numerical values and commands into the system requesting a service like enquiring the amount of power tokens.

3.3.2 Solar Energy Generation and Storage

The two solar panels rated at 8.4V,0.59A and 9V,0.5A respectively are connected in series to generate energy and store it to the battery. The voltage generated by the solar panels is regulated from 17.4V to 14.4V by a buck dc converter to optimize the voltage used to charge the battery. Each household draws energy from the 12V battery.

3.3.3 Load Management

Household A uses two bulbs with a rating of 3W A and 7W respectively.

Household B uses one bulb with a rating of 7W. The switching of the bulb is controlled by a MOSFET which enhances automatic switching.

We implemented both the manual switching part and the automatic part using MOSFETs in household A. There is also a main switch that is used to cut down the power

being supplied to the household that is purchasing power when the household stops to sell power.

3.3.4 Data Acquisition (Monitoring)

Current sensors (ACS712) were used to measure real-time current flowing through the bulbs. Arduino reads current values, calculates power ($P = V \times I$), and tracks energy consumption over time.

3.3.5 Software development

3.3.5.1 Overview

The Solaris Conexus system adopts a modular and layered software architecture designed to facilitate decentralized solar energy monitoring, peer-to-peer energy trading, and seamless interaction between physical hardware and blockchain infrastructure. The system is composed of five major software components, each playing a distinct role in the data flow and energy transaction process as follows:

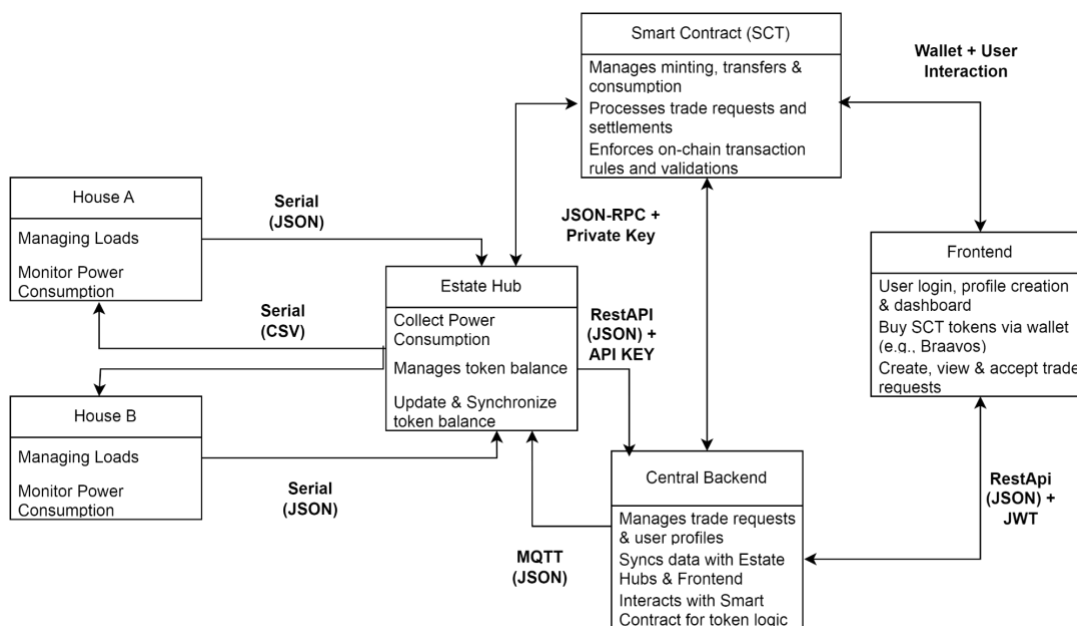


Figure 8:Flow chart Diagram

3.3.5.2 Arduino Software

The embedded system for power monitoring and control is developed using an Arduino-based microcontroller programmed in C++. This system interfaces with both hardware components (sensors, keypad, LCD, EEPROM, etc.) and the serial interface (Python-based backend) for bidirectional communication. The code implements the following key functionalities:

3.3.5.3 Power Measurement

The software running on the Arduino Uno performs real-time power measurement by continuously sampling analog signals from current and voltage sensors. The analog readings are processed through scaling and calibration equations to derive current (in amperes) and voltage (in volts). These measurements form the basis for computing power consumption and making energy management decisions. The code uses standard Arduino functions such as `analogRead()` to obtain raw values, which are then converted using empirically determined constants to reflect accurate physical quantities.

```
void updatePower() {
    if (millis() - lastCurrentUpdate >= currentInterval) {
        lastCurrentUpdate = millis();

        const int samples = 50;
        long currentSum = 0;
        long voltageSum = 0;

        for (int i = 0; i < samples; i++) {
            currentSum += analogRead(A0); // Read current sensor
            voltageSum += analogRead(A1); // Read voltage divider
            delay(2); // Minimal delay for stability
        }

        // --- Calculate average values ---
        float avgCurrentADC = currentSum / (float)samples;
        float avgVoltageADC = voltageSum / (float)samples;

        // --- Convert ADC to real-world values ---
        float voltageCurrent = avgCurrentADC * (5.0 / 1023.0);
        current = (voltageCurrent - 2.5) / 0.185; // Adjust based on sensor

        float voltageAtA1 = avgVoltageADC * (5.0 / 1023.0);
        batteryVoltage = voltageAtA1 * ((R1 + R2) / R2);
    }
}
```

Figure 9: Power measurement code

3.3.5.4 Bidirectional Communication

To facilitate external integration and remote control, the system employs bidirectional serial communication via the Arduino's USB interface using the Serial library. Sensor data—comprising current, voltage, and relay status—is transmitted as structured JSON-like strings to a Python script running on a laptop or gateway device. In return, the Arduino listens for incoming serial data packets, which include crucial control parameters such as the user's energy balance and a connection status flag. This two-way communication mechanism ensures real-time coordination between the embedded system and higher-level energy management software.

```
void acceptSerialInput() {
    if (Serial.available()) {
        lastSerialInputTime = millis();
        String input = Serial.readStringUntil('\n');
        input.trim();
        if (input.length() > 0) {
            int commaIndex = input.indexOf(',');
            if (commaIndex > 0) {
                float receivedBalance = input.substring(0, commaIndex).toFloat();
                int receivedInstruction = input.substring(commaIndex + 1).toInt();

                accountBalance = receivedBalance;

                switch (receivedInstruction) {
                    case 1:
                        connectionStatus = true;
                        break;
                    case 0:
                        connectionStatus = false;
                        break;
                    case 2:
                        ledState = !ledState;
                        digitalWrite(LED_PIN, ledState ? HIGH : LOW);
                        EEPROM.update(0, ledState);
                        break;
                }
            }
        }
    }
}

void sendSerialOutput() {
    if (connectionStatus) {
        String message = "{";
        message += "\"current\": " + String(current, 2) + ",";
        message += "\"voltage\": " + String(batteryVoltage, 2) + ",";
        message += "\"req\": " + String(ledState ? "true" : "false") + "\"";
        message += "}";
        Serial.println(message);
    } else {
        Serial.println("{\"device_id\": \"\" + device_id + "\"}");
    }
}
```

Figure 10: Bidirectional communication

3.3.5.5 User Interface

An interactive user interface is implemented through a 4x4 matrix keypad and an I2C 16x2 LCD display. The keypad allows users to input commands by pressing specific digits to view different types of system data, such as current, voltage, LED (relay) status, and connection availability. The LCD updates accordingly to provide immediate feedback, thereby enabling the user to monitor the system's state without needing access to a computer. The interface is user-friendly, responsive, and designed to operate under limited processing and memory constraints typical of microcontroller environments.

```
void acceptKeypadInput() {  
    char input = keypad.getKey();  
    if (!input) return;  
  
    if (input == 'C') {  
        mode = -1;  
    } else if (input >= '0' && input <= '6') {  
        mode = input - '0';  
    } else if (input == 'A') {  
        connectionStatus = !connectionStatus;  
    }  
}
```

```
switch (mode) {  
    case 0:  
        lcd.setCursor(0, 0);  
        lcd.print("Device ID:");  
        lcd.setCursor(0, 1);  
        lcd.print(device_id);  
        break;  
  
    case 1:  
        lcd.setCursor(0, 0);  
        lcd.print("Connection:");  
        lcd.setCursor(0, 1);  
        lcd.print(connectionStatus ? "1 (ON)" : "0 (OFF)");  
        break;  
  
    case 2:  
        lcd.setCursor(0, 0);  
        lcd.print("Avg Current:");  
        lcd.setCursor(0, 1);  
        lcd.print(current, 2);  
        lcd.print(" A");  
        break;
```

```
    case 3:  
        lcd.setCursor(0, 0);  
        lcd.print("Balance:");  
        lcd.setCursor(0, 1);  
        if (connectionStatus) {  
            lcd.print(accountBalance, 2);  
            lcd.print(" SCT");  
        } else {  
            lcd.print("unable fetch");  
        }  
        break;  
  
    case 4:  
        ledState = !ledState;  
        digitalWrite(LED_PIN, ledState ? HIGH : LOW);  
        EEPROM.update(0, ledState);  
        lcd.setCursor(0, 0);  
        lcd.print("LED Toggled");  
        lcd.setCursor(0, 1);  
        lcd.print(ledState ? "Now ON" : "Now OFF");  
        mode = -1; // Reset mode  
        break;
```

Figure 11: User interface

3.3.5.6 Access Control Logic (ACL)

At the heart of the control system is a multi-condition Access Control Logic (ACL) module that governs the operation of the relay controlling power output. The master switch is activated only when three critical conditions are simultaneously satisfied: the user's balance is greater than zero, the connection to the broader network is verified as active, and the user has manually toggled the switch (via keypad input). This logic is checked continuously in the main execution loop, ensuring that power is only supplied when all policy and operational constraints are met. This safeguards the system against unauthorized or unintended power usage.

```
// === MASTER SWITCH CONTROL BASED ON BALANCE & CONNECTION ===  
if (accountBalance > 0 && connectionStatus) {  
    digitalWrite(MASTER_PIN, HIGH); // Turn ON master switch  
} else {  
    digitalWrite(MASTER_PIN, LOW); // Turn OFF master switch  
}
```

Figure 12: Access Logic Unit

3.3.5.7 Persistence

The system incorporates data persistence using the Arduino's onboard EEPROM to retain key user settings across power cycles. Specifically, the state of the relay (on/off) is stored in EEPROM memory whenever a change is made via the keypad. On startup or reset, the program reads the stored value and restores the previous state, providing a seamless user experience and enhancing reliability. This feature ensures that user preferences are maintained even in the event of power loss or system reboot.

```
EEPROM.get(0, ledState); // safer EEPROM read  
pinMode(LED_PIN, OUTPUT);  
digitalWrite(LED_PIN, ledState ? HIGH : LOW);
```

Figure 13: Persistence

3.3.5.8 Timing and Responsiveness

To manage updates and delays without blocking the main loop, the software uses the `millis()` function for non-blocking time management. This allows concurrent operations—such as sensor reading, serial communication, and LCD updates—to execute smoothly without halting the system. As a result, the program maintains high responsiveness and supports real-time interaction, a key requirement for embedded systems operating in dynamic environments.

```
if (ledState != prevLedState || millis() - lastSendTime >= sendInterval) {  
    sendSerialOutput();  
    lastSendTime = millis();  
    prevLedState = ledState;  
}
```

Figure 14: Time and Responsiveness

The full source code, implementing all the described modules and logic, is available for review and reuse. It provides a comprehensive template for embedded energy monitoring and control in decentralized environments. It can be accessed through the following repository link: [Arduino House-Level Power Management](#).

3.3.5.9 Estate Hub Software

The Estate Hub acts as the intermediary between house-level Arduino monitors and the central blockchain system. It is designed for resilience, continuous operation, and real-time communication with distributed nodes. Implemented in Python, the software establishes serial communication with Arduino devices, processes incoming energy metrics, and relays actionable instructions.

3.3.5.10 Robust Arduino Device Communication

At the core of the Estate Hub’s functionality is a robust communication interface with Arduino-based power monitoring devices deployed throughout the estate. These Arduino devices play a crucial role in capturing real-time electrical parameters, such as current and voltage, from individual homes. To manage this communication effectively, the Estate Hub leverages a dedicated software component: the *Arduino Device* class.

This class is designed to establish a reliable and persistent serial connection to each Arduino through a specified port and baud rate (typically 9600). Recognizing that hardware

interfaces can occasionally be unpredictable, the *connect_with_retry()* method introduces resilience by continuously attempting to establish a connection every 30 seconds until successful. This ensures that the hub remains capable of recovering automatically from temporary disconnections or power cycles.

Upon successful connection, the device enters a handshake phase where it listens for a specially formatted JSON message from the Arduino. This message must contain a unique *device_id*, which serves as the device's fingerprint within the backend system. By validating this identifier against records in the database, the hub ensures that the data it receives is correctly mapped to the corresponding estate unit. Once the handshake is complete and the device is authenticated, the system transitions into active monitoring.

Beyond simply establishing a connection, the *Arduino Device* class is designed with stateful intelligence. It maintains attributes like the latest current and voltage readings, a timestamp for when data was last received, and operational parameters such as instruction and balance. These values can be updated in real-time and used to control how the Arduino behaves, for instance, by adjusting how power is distributed or when certain loads should be disabled to conserve energy. This feedback loop is made possible through the *send_data()* method, which transmits instructions and account balances back to the Arduino in a structured format.

Moreover, the class is prepared for integration with blockchain-based identity or accounting systems. For example, it includes fields like *account address* and *update* flags, which enable more advanced features such as authenticating energy consumption or credits using decentralized identities.

Finally, the *run_with_retry()* method ties everything together into a continuous loop that monitors device status, handles reconnections gracefully, and ensures that the Estate Hub can function autonomously. Should a disconnection or unexpected error occur, the system waits for a short interval and then retries the connection—eliminating the need for manual intervention.

In summary, the *Arduino Device* class embodies the principles of resilience, modularity, and intelligent control. It not only ensures steady communication with hardware devices but also lays the foundation for real-time energy management, decentralized

accounting, and automated decision-making across the estate's energy network. The sample of the class is as shown below:

```
class ArduinoDevice:
    def __init__(self, port, baud=9600):
        self.port = port
        self.baud = baud
        self.serial = None
        self.device_id = None
        self.current = 0.0
        self.voltage = 0.0
        self.timestamp = None
        self.id = None # DB Device.id numeric
        self.balance = 0.0
        self.instruction = 1 # Default instruction
        self.account_address = None
        self.update = 0

    def connect_with_retry(self):
        """Try to connect to the Arduino port every 30 seconds until successful."""
        while self.serial is None:
            try:
                print(f"[Info] Trying to connect to {self.port}...")
                self.serial = serial.Serial(self.port, self.baud, timeout=1)
                time.sleep(2) # Allow Arduino to reset
                print(f"[Success] Connected to {self.port}")
            except serial.SerialException:
                print(f"[Warning] {self.port} not available. Retrying in 30 seconds...")
                time.sleep(30)
```

Figure 15: Arduino communication

```
def send_data(self):
    """
    Sends balance and instruction to Arduino as a comma-separated string.
    Format: "<balance>,<instruction>\n"
    """
    message = f"{self.balance},{self.instruction}\n"
    try:
        self.serial.write(message.encode())
        print(f"[Sent] {message.strip()}")
    except Exception as e:
        print(f"[Error] Failed to send data: {e}")
```

Figure 16: Arduino Communication

The full implementation of the *Arduino Device* class, including methods for connection handling, data parsing, and instruction transmission, can be accessed through the following link: [Arduino Device Module](#)

3.3.5.11 Communication Management at the Estate Hub

The Estate Hub serves as the critical communication bridge between local solar energy devices within an estate and the central hub that oversees the entire energy network. This communication is managed through two distinct channels: outgoing data submissions to the

central hub via a secured REST API, and incoming control commands from the central hub received via MQTT messaging.

For outbound communication, the Estate Hub periodically aggregates energy consumption and production data collected from multiple solar-powered houses. This aggregated data is then transmitted securely to the central hub's REST API endpoint. To authenticate and authorize these requests, the Estate Hub includes a unique API key in the request headers, ensuring that only trusted estate hubs can submit data. The use of REST API enables standardized, stateless, and efficient data exchange, allowing the central hub to process real-time energy statistics and update the blockchain ledger accordingly.

In parallel, the Estate Hub maintains a persistent connection to an MQTT broker, subscribing to a dedicated topic named *juja/commands*. Through this MQTT topic, the central hub pushes real-time operational commands and configuration updates to the Estate Hub. MQTT is chosen for its lightweight publish-subscribe messaging model, which is well-suited for low-bandwidth, high-latency, or unreliable network conditions often encountered in distributed IoT systems. The Estate Hub listens to this topic continuously and reacts promptly to commands such as token minting requests, device resets, or changes in operating parameters.

The dual communication strategy, REST API for reliable data submission and MQTT for instantaneous command delivery, ensures that the Estate Hub remains in sync with the central hub's directives while providing consistent energy data updates. This architecture also improves fault tolerance: if the REST API connection is temporarily disrupted, data submission can be retried without blocking command reception via MQTT.

Security considerations are paramount in this communication framework. The API key in REST requests prevents unauthorized data injection, while MQTT communications are secured via TLS encryption and client authentication to protect against interception or spoofing.

The below function queries the central hub for the estate's registered devices and stores them locally in a database using an API key for authentication.

```

def fetch_and_store_devices():
    url = f"{BACKEND_URL}/hubs/devices"
    params = {"api_key": BACKEND_KEY}
    response = requests.get(url, params=params)
    if response.status_code != 200:
        print("Failed to fetch devices:", response.text)
        return

    devices = response.json().get("devices", [])
    db = SessionLocal()

    # Prepare map for devices with account_address
    address_to_device = {}
    for d in devices:
        acct_addr = d.get("account_address")
        if acct_addr:
            address_to_device[acct_addr] = d

    # Batch get balances using asyncio
    async def fetch_balances(addresses):
        tasks = [balanceOf(addr) for addr in addresses]
        results = await asyncio.gather(*tasks, return_exceptions=True)
        return dict(zip(addresses, results))

    balances = asyncio.run(fetch_balances(list(address_to_device.keys())))

```

Figure 17: Communication Management

The MQTT client subscribes to the topic juja/commands to receive commands like "stream". Upon receiving this command, it triggers stream_power_data (), which sends power consumption data real time as shown below

```

def on_message(client, userdata, msg):
    print(f"[MQTT] Message received on {msg.topic}: {msg.payload.decode()}")

    try:
        command = json.loads(msg.payload.decode())

        if command.get("action") == "send_history":
            from .utils import publish_historical_data
            publish_historical_data(hours=command.get("hours", 24))

        elif "device" in command and "instruction" in command:
            device_id = command["device"]
            instruction = command["instruction"]

            db_generator = get_db()
            db: Session = next(db_generator)
            try:
                device = db.query(Device).filter(Device.id == device_id).first()
                if device:
                    device.instruction = instruction
                    db.commit()
                    print(f"[MQTT] Updated instruction for device {device_id} to {instruction}")
            except:
                pass

            else:
                print(f"[MQTT] Device ID {device_id} not found in database.")

        elif command.get("command") == "stream":
            device_id = command["device"]
            if not streaming_flags.get(device_id):
                streaming_flags[device_id] = True
                threading.Thread(target=stream_power_data, args=(device_id, client), daemon=True).start()
                print(f"[Stream] Started streaming for device {device_id}")
            else:
                print(f"[Stream] Already streaming for device {device_id}")

        elif command.get("command") == "stop":
            device_id = command["device"]
            streaming_flags[device_id] = False
            print(f"[Stream] Stopped streaming for device {device_id}")

        else:
            print(f"[MQTT] Unknown command: {command}")

    except json.JSONDecodeError:
        print("[MQTT] Invalid JSON payload")

```

Figure 18: Communication management

3.3.5.12 Blockchain Integration with Starknet Sepolia

The estate hub integrates directly with the Starknet Sepolia testnet to facilitate secure and transparent energy token transactions using the Solaris Conexus Token (SCT) smart contract. This integration enables each estate to participate in decentralized energy trading by interacting with blockchain-based logic, including checking token balances, purchasing tokens, and signing peer-to-peer energy trades. These interactions ensure a tamper-proof, traceable, and automated mechanism for managing energy credits within and across estates.

To initiate blockchain communication, the estate hub configures a Starknet account using the *starknet_py SDK*. The setup includes the estate's wallet address, public and private keys, and the node URL of Starknet Sepolia. Once initialized, the estate connects to the SCT smart contract deployed at a known address, and its Application Binary Interface (ABI) is preloaded for direct function calls.

One of the core functionalities provided is checking the token balance of any participating house. Using the *balanceOf()* function, the estate hub queries the blockchain with the hexadecimal wallet address of a house and retrieves the corresponding SCT balance. This allows the estate to track energy credits per household and ensure up-to-date synchronization with blockchain records before approving local energy consumption or sale.

Another critical feature is the execution of peer-to-peer energy trades via blockchain. The *consume()* function plays a vital role by finalizing a trade once power has been consumed. It does this by deducting the corresponding number of SCT tokens from the buyer and permanently destroying them, effectively reducing the total token supply. This ensures that tokens are only retained when energy is available and consumed responsibly. By tying token destruction directly to power usage, *consume()* enforces transparency and prevents token inflation, all while maintaining an immutable and decentralized record of each transaction on-chain.

By connecting local energy monitoring to the Starknet blockchain, the estate hub ensures a high level of trust, security, and accountability. This system lays the groundwork for a scalable and decentralized energy economy where households can autonomously manage and trade renewable energy tokens in a fair and open environment.

The figure below shows the account creation using a users account information then initializing the SCT contract instance. The 2 methods of *balanceOf* & *consume* is as shown below:

```

account = Account(
    client=client,
    address=WALLET_ADDRESS,
    key_pair=KeyPair.from_private_key(
        key=WALLET_PRIV
    ),
    chain=StarknetChainId.SEPOLIA,
)

contract_address = SCT_ADDRESS

# When ABI is known statically just use the Contract constructor
contract = Contract(address=contract_address, abi=contract_abi, provider=account, cairo_version=1)

async def balanceOf(account_address: str) -> int:
    # Calling contract's function doesn't create a new transaction, you get the function's result.
    (saved,) = await contract.functions["balanceOf"].call(int(account_address, 16))
    return saved

```

```

async def consume(account: str) -> str:
    """
    Destroy tokens using the provided StarkNet address.
    Returns the transaction hash.
    """
    buyer_int = int(account, 16)
    try:
        # Call the contract's 'consume' method
        # Assuming amount is a Decimal like Decimal("10.5")
        invocation = await contract.functions["consume"].invoke_v3(
            account=buyer_int,
            amount=int(1),
            auto_estimate=True,
        )
        # Wait for transaction to be accepted
        await invocation.wait_for_acceptance()
        return hex(invocation.hash)

    except Exception as e:
        print(f"Error during contract invocation: {e}")
        raise HTTPException(
            status_code=500,
            detail=f"Failed to invoke contract: {str(e)}"
        )

```

Figure 19: Blockchain Intergration with starknet

3.3.5.13 Smart Contract Methodology: Solaris Conexus Token (SCT)

The Solaris Conexus Token (SCT) smart contract is the core engine powering decentralized peer-to-peer energy trading on the Solaris Conexus platform. Built using Cairo and deployed on the Starknet blockchain, SCT is designed to handle the full lifecycle of Power Tokens, digital units representing energy value within the ecosystem. It supports minting, transfers, trade agreement management, consumption tracking, and contract upgrades, all while ensuring operations are recorded transparently on-chain.

3.3.5.14 Design Strategy

The development of the Solaris Conexus Token (SCT) smart contract was guided by three core principles: security, modularity, and transparency. Every function within the contract was structured to ensure data integrity and strict control over sensitive operations. Critical actions like token minting and contract upgrades are restricted to the contract owner, verified through role checks. State variables such as total supply, user balances, and trade mappings were designed for efficient access and consistency, providing a solid foundation for interacting with the energy token ecosystem.

To reinforce these controls, helper functions like *_is Owner* and *_sufficient Balance* were embedded within the contract to validate permissions and state conditions. The mint function, for example, ensures that only authorized addresses can create new tokens:

```
fn mint(ref self: ContractState, account: ContractAddress, amount: u256){
    let owner: ContractAddress = (get_caller_address());
    assert!(self._isOwner(owner), "UNAUTHORIZED ACCOUNT");

    let new_free_token: u256 = self.account_balances.entry(account).read() + amount;
    let new_supply: u256 = self.total_supply.read() + amount;
    self.total_supply.write(new_supply);
    self.account_balances.entry(account).write(new_free_token);
    self.emit(Mint{account, amount});
}
```

Figure 20: Design strategy

This disciplined structure ensures the SCT contract remains resilient, adaptable, and trustworthy throughout its lifecycle.

3.3.5.15 Token Transfers

In the Solaris Connexus Token (SCT) contract, token transfers form the backbone of energy trading by enabling residents to securely and directly exchange Power Tokens with one another. This function simulates real-world peer-to-peer energy transactions by updating token balances instantly and reliably on the blockchain. To maintain system integrity, the contract enforces strict checks to ensure that senders cannot transfer more tokens than their available balance. This prevents accidental overdrafts and helps maintain trust in the decentralized trading environment.

The *transfer* function operates by first verifying the sender's balance, then deducting the transferred amount from the sender while crediting the recipient accordingly. These balance updates are performed atomically to avoid discrepancies or race conditions. Additionally, each successful transfer emits a Transfer event, which serves as a public record for blockchain explorers and backend systems, providing transparency and traceability of all token movements. This mechanism plays a crucial role in fostering accountability and supporting audit processes within the platform.

Here is the core transfer method implemented in Cairo:

```
fn transfer(ref self: ContractState, recipient: ContractAddress, amount: u256){
    let sender: ContractAddress = (get_caller_address());

    assert!(self._sufficientBalance(sender, amount), "INSUFFICIENT BALANCE");
    let new_balance: u256 = self.account_balances.entry(sender).read() - amount;
    self.account_balances.entry(sender).write(new_balance);
    let new_balance: u256 = self.account_balances.entry(recipient).read() + amount;
    self.account_balances.entry(recipient).write(new_balance);
    self.emit(
        Transfer{
            from: sender,
            to: recipient,
            amount
        }
    );
}
```

Figure 21: Token transfer

3.3.5.16 Trade Lifecycle Management

One of the standout features of the Solaris Conexus Token (SCT) contract is its robust system for managing the lifecycle of energy trade agreements between users. This functionality goes beyond simple token transfers by enabling structured peer-to-peer trades, complete with trade states, ownership verification, and blockchain-logged activity. The contract maintains a trade registry, which acts as a decentralized ledger of all active and historical trades, ensuring each transaction follows a well-defined lifecycle from initiation to closure.

The life cycle begins with trade creation; users can request a trade by specifying token amounts and expected recipients. Once created, trades are marked as "Pending" and can then be activated through the sign Trade function, which assigns a buyer and transitions the trade to an "Active" state. The system enforces role-based checks to ensure only the contract owner can activate trades, protecting users from malicious or unintended activity. Each state transition is validated with assertions, and every significant change triggers a blockchain event for transparency. This design allows for subsequent actions like repayments, defaults, or deletions, all tracked on-chain for trust and auditability.

Here's the function responsible for creating & signing trades:

```
fn createTrade(ref self: ContractState, amount: u256){
    let lender: ContractAddress = (get_caller_address());
    let owner: ContractAddress = self.owner.read();
    assert!(lender != owner, "INSUFFICIENT AUTHORITY");
    self.transfer(owner, amount);
    let trade_id: u64 = self.trade_counter.read();
    let trade: Trade = Trade {
        id: trade_id,
        lender,
        borrower: Option::None,
        amount,
        balance: amount,
        status: TradeStatus::Pending
    };
    let local_id: u64 = self._insertTrade(trade);
    let trade_event: TradeEvent = TradeEventImpl::new(trade, local_id);
    self.emit(trade_event);
}
```

```

fn signTrade(ref self: ContractState, buyer: ContractAddress, trade_id: u64) {
  let owner: ContractAddress = get_caller_address();
  let mut trade: Trade = self._getTrade(trade_id);
  assert!(self._isOwner(owner), "UNAUTHORIZED ACCOUNT");
  assert!(trade.status.to_u8() == 0, "TRADE IS ALREADY ACTIVE");
  trade.sign(buyer);
  self.trade.at(trade_id).write(Option::Some(trade));
  let Trade_event: TradeEvent = TradeEventImpl::new(trade, trade_id);
  self.emit(Trade_event);
}

```

Figure 22: Trade lifecycle management

3.3.5.17 Consumption and Supply Updates

To ensure that energy usage is accurately mirrored on-chain, the Solaris Conexus Token (SCT) contract includes dedicated functionality for token consumption. This reflects the real-world scenario where residents utilize stored or purchased energy, and their token balances should be updated accordingly. The consume function allows the contract owner to decrement both the user's token balance and the overall token supply, ensuring accountability and preventing token inflation.

This functionality is particularly critical in systems where consumption data is derived from real-time monitoring devices, such as Arduino-based sensors installed in homes or hubs. Once energy usage is measured, a backend service can securely trigger the consume method on behalf of the system, creating a seamless link between the physical energy consumption and its digital representation.

Here is the implementation of the consume logic:

```

fn consume(ref self: ContractState, account: ContractAddress, amount: u256){
  let call_account: ContractAddress = (get_caller_address());
  assert!(self._isOwner(call_account), "UNAUTHORIZED ACCOUNT");
  assert!(self._sufficientBalance(account, amount), "INSUFFICIENT BALANCE");

  let new_free_token: u256 = self.account_balances.entry(account).read() - amount;
  let new_supply: u256 = self.total_supply.read() - amount;
  self.total_supply.write(new_supply);
  self.account_balances.entry(account).write(new_free_token);
  self.emit(Consume{account, amount});
}

```

Figure 23: Consume logic

3.3.5.18 Contract Upgrades

Given the evolving nature of decentralized applications, the Solaris Conexus Token (SCT) contract incorporates a robust upgrade mechanism. This design choice ensures that the smart contract can adapt over time, whether to patch vulnerabilities, introduce new features, or optimize existing logic, without requiring a complete system overhaul. Upgrades are performed securely and are strictly controlled by the contract owner, preserving user trust and system integrity.

The upgrade process leverages Starknet's class replacement feature. A new class hash representing the updated contract version is passed to the upgrade function, which then replaces the contract's logic while maintaining its current state. This approach combines flexibility with stability, allowing the system to evolve while safeguarding critical data such as user balances and trade histories.

The implementation is as follows:

```
fn upgrade(ref self: ContractState, new_class_hash: ClassHash) {
    let caller: ContractAddress = get_caller_address();
    assert!(self._isOwner(caller), "INSUFFICIENT AUTHORITY");
    assert!(!(new_class_hash.is_zero()), "Class hash cannot be zero");
    syscalls::replace_class_syscall(new_class_hash).unwrap();
    self.emit(Upgrade{by: caller});
}
```

Figure 24: Contract upgrade

By including a final event emission, every upgrade action is recorded on-chain, creating an immutable log that promotes accountability. This mechanism ensures the smart contract remains future-proof while continuing to serve the needs of the decentralized energy trading ecosystem.

The full implementation of the Solaris Conexus Token smart contract, including its modular architecture and complete method definitions, can be reviewed here: [Solaris Conexus Token Smart Contract](#)

3.3.5.19 Frontend Application (Web Interface)

The frontend of the Solaris Conexus platform was developed using React with the Next.js framework to provide a fast, modular, and responsive user experience. The goal was to create a lightweight, accessible interface that allows residents to interact securely and

efficiently with the system. The architecture follows a component-based structure, ensuring maintainability and scalability. Features were designed with user flow and ease-of-use in mind, leveraging Tailwind CSS for styling, React hooks for state management, and secure communication with the backend via RESTful APIs.

The application includes core modules such as authentication (login/registration), a user dashboard, profile management, device monitoring, token trading, and token purchasing. These modules are integrated with external services, including Starknet wallet for blockchain interactions. Each part contributes to the overall goal of giving users full control over their energy data and tokens in a clean and intuitive environment. The following sections briefly outline how each module was built to support this vision.

3.3.5.20 Authentication

The authentication module is a critical component of the frontend application, designed to securely manage user access and protect sensitive information. This module supports two main functions: user registration and login. Both processes are implemented using React's modern features, including hooks and controlled form inputs, to ensure a smooth and interactive user experience. The frontend communicates with the backend API, which is built using FastAPI, to handle all authentication-related operations such as validating credentials and issuing authentication tokens.

During user registration, the frontend presents a form that collects essential user details like username, email, and password. Before submission, the form performs basic validation to ensure all required fields are properly filled out, minimizing errors and improving data integrity. Once validated, the form sends a POST request to the backend's /register endpoint, where the server processes the information, creates a new user account, and responds with a success or error message. The frontend then provides immediate feedback to the user, confirming successful registration or highlighting any issues that need to be addressed.

The login process follows a similar approach. Users enter their credentials into a controlled form that submits a POST request to the backend's /login endpoint. Upon successful authentication, the backend returns a JSON Web Token (JWT), which the frontend securely stores in the browser's local storage. This token acts as proof of authentication for future requests, enabling the user to remain logged in without repeatedly entering credentials. By leveraging JWTs, the system maintains session security while supporting stateless communication between the client and server.

To safeguard protected routes within the application, a custom React hook called `useAuth` is utilized. This hook checks for the presence of a valid JWT in local storage whenever a protected page is loaded. If no token is found, the user is automatically redirected to the login page, preventing unauthorized access. This mechanism centralizes authentication checks and eliminates redundant code, enhancing maintainability and security throughout the frontend.

Together, these elements form a robust and user-friendly authentication flow that integrates smoothly with the backend infrastructure. The source code for the authentication module can be reviewed here: [Frontend Authentication Module](#).

3.3.5.21 Dashboard Home

The Resident Dashboard Home functions as the primary landing page after a user successfully logs into the Solaris Conexus platform. It provides an at-a-glance summary of important user metrics, including the number of registered devices, the balance of SCT tokens, and the balance of STRK tokens. This dashboard was designed using React functional components combined with hooks to efficiently manage state and lifecycle events, ensuring that data is fetched and displayed securely and dynamically.

The page layout incorporates Tailwind CSS for styling, enabling a modern, clean, and responsive design that adapts seamlessly to different screen sizes. Navigation is handled through a reusable Sidebar component, which maintains consistent user experience across various sections of the dashboard. The main content area dynamically adjusts its layout to optimize usability on both desktop and mobile devices.

Upon mounting, the Dashboard component initiates a secure API request to fetch the user's details from the backend. This request includes the JWT token stored in local storage as part of the authorization header, thereby validating the user's session and providing access to personalized data. The component handles potential errors gracefully, such as when the token is missing or expired, ensuring the app remains stable without exposing sensitive information.

To improve user experience, the dashboard employs conditional rendering to indicate loading states clearly. While data is being fetched, a loading placeholder appears, signaling to the user that the information is being retrieved. Once the data arrives, the dashboard updates automatically, displaying the latest balances and device counts in real time. This responsiveness fosters trust and keeps the user informed about their current status within the system.

Overall, the Dashboard Home combines intuitive design, secure data fetching, and dynamic rendering to offer a smooth and informative user experience. The full source code and implementation details can be accessed here: [Dashboard Home Module](#).

3.3.5.22 Profile

The Profile component is designed as a central interface where residents can view and manage their personal information within the Solaris Conexus platform. Built using React with the Next.js framework, the component leverages modern frontend development techniques, such as React hooks and context, to handle state management effectively. This ensures a responsive and interactive experience for users managing their profiles.

When the component loads, it fetches the current user's profile data from the backend API, including details like the user's name, contact information, and potentially blockchain-related account identifiers. This data retrieval happens securely through authenticated API calls, ensuring that only authorized users can access and update their own information. By pulling fresh data upon each visit, the component maintains accuracy and relevance in the user's displayed profile.

Users can edit their profile information directly within the component via integrated forms. Changes made in the input fields are tracked locally using React's state hooks, allowing immediate feedback and validation as users interact with the form. When the user submits updates, the component sends the modified data back to the backend API to persist changes in the database. The component also manages loading states and displays success or error messages, giving clear feedback on the update process.

Authentication is closely integrated into the Profile component to ensure that the profile data corresponds to the logged-in user. This includes interaction with the Starknet wallet connection system, which helps verify the user's identity and link blockchain accounts where necessary. The component's layout is built to be responsive, adapting smoothly across a variety of devices and screen sizes to ensure accessibility and ease of use for all residents.

By providing a secure and intuitive personal space, the Profile component plays a key role in empowering users to maintain control over their identity and settings within the Solaris

Conexus ecosystem. The full source code and further technical details are available here: [Profile Component Source Code](#).

3.3.5.23 Trade

The Trade component facilitates peer-to-peer energy token transactions between residents by providing a streamlined and intuitive interface for managing PowerToken trades. Developed using React in the Next.js framework, the module is designed to simplify complex blockchain interactions while ensuring security and transparency. It allows residents to initiate, view, and manage trade activities directly from their dashboards.

Upon loading, the component fetches the current user's trade history, including active and past trade requests. This history is displayed in a structured table format, offering visibility into transaction statuses, trade amounts, and counterparties. The interface dynamically adapts based on real-time data retrieved from the backend, which ensures that users stay informed of trade changes as they happen.

Users can create a new trade by filling out a simple form that includes the recipient's username and the number of tokens to transfer. When a trade is submitted, the component interacts with the connected Starknet wallet (e.g., Braavos) to initiate the blockchain transaction. This integration ensures that all transfers are authenticated and verifiable on-chain, while the frontend abstracts away the complexities of interacting with smart contracts. Feedback mechanisms such as confirmation modals and success/error alerts are built in to guide users through each action.

The component also includes functionality to cancel trade requests if they haven't been fulfilled. This ensures flexibility for users and prevents unintentional token commitments. These actions are protected by clear UI prompts and require user confirmation before any irreversible operations are performed. The component uses centralized API calls to update trade status and propagate changes across the platform.

By bridging frontend usability with backend logic and blockchain immutability, the Trade module delivers a seamless experience that empowers residents to participate in decentralized energy trading. The component supports both transparency and ease of use—two

critical pillars in the success of the Solaris Conexus ecosystem. The full implementation is available for review at: [Trade Frontend Module](#).

3.3.5.24 Purchase

The Purchase component is a crucial part of the frontend that enables residents to acquire Power tokens directly through their connected Stark Net wallets. Designed with simplicity and usability in mind, this module offers a frictionless experience for residents who need tokens for trading or energy settlement within the Solaris Conexus ecosystem.

Upon accessing the Purchase page, users are greeted with a clean and responsive form interface that allows them to specify the amount of Power Tokens they wish to buy. Once the amount is entered, the system initiates a blockchain transaction by connecting to the user's Bravos wallet. The wallet prompts the resident to confirm the transaction, which, once authorised, triggers a transfer operation on the smart contract, securely transferring the unowned tokens into their account.

To ensure clarity and responsiveness throughout the process, the component incorporates visual feedback elements such as loading animations during wallet interaction, success confirmations once tokens are received, and error messages in case of failed transactions. This approach enhances user trust while abstracting away the underlying blockchain mechanics, making the system accessible even to users without prior Web3 experience.

In addition to the on-chain operations, the component communicates with the backend API to log purchase activity. This helps in synchronizing token balances with the user's dashboard and maintaining a consistent record of transactions. This dual-layer integration, blockchain and backend, ensures both transparency and accountability in token purchases.

Overall, the Purchase module elegantly ties together smart contract functionality with intuitive frontend design. It empowers residents to take part in the decentralized energy marketplace with just a few clicks. The complete implementation can be explored here: [Purchase Frontend Module](#).

3.3.5.25 Device

The Device component forms an essential part of the frontend, offering residents a centralized space to manage and monitor the energy devices registered to their profile. These devices, often connected through microcontroller units like Arduino, are responsible for

capturing critical metrics such as energy production, consumption, voltage, and balance. The component bridges the user interface with both backend services and the physical IoT infrastructure to provide real-time insights and control.

When the Device page loads, it immediately fetches the authenticated resident's device list by querying the backend API. Each listed device displays detailed metadata, including the device name, activation status, registration date, and any recent readings available. The layout was designed using Tailwind CSS to ensure responsiveness and a clean, digestible presentation of information across different screen sizes.

Adding a new device is made simple through a registration form embedded within the interface. Residents can input device-specific data such as serial numbers, the loads connected and type of connection to the estate hub. Upon form submission, the data is validated and sent to the backend via a POST request. The backend associates the new device with the resident's account, and once confirmed, it appears in the device listing, ready for data tracking.

In addition to registration and listing, the module supports live or periodic monitoring of connected devices. It fetches updated readings, such as power consumption from the database of the estate hub, depending on the implementation setup. Residents can also interact with certain devices by toggling load states (e.g., turning appliances on or off) via frontend controls. This level of interactivity enhances energy awareness and empowers users to make data-driven decisions about their consumption.

The Device component successfully integrates frontend responsiveness with backend APIs and physical devices, serving as a critical interface in the decentralized energy trading system. For a deeper look into the implementation, the full source code is available here: [Device Module](#).

3.3.5.26 Central Backend Server

The backend of this project is built using Fast API, a high-performance web framework for building APIs with Python. It serves as the central system for managing devices, hubs, MQTT communication, and smart contract integration via Starknet. This architecture is designed for scalability, modularity, and integration with IoT and blockchain technologies.

3.3.5.27 Database Tables (ORM Models)

At the heart of this backend lies a structured database powered by SQLAlchemy ORM, which models the main entities and relationships in the system. These models allow the backend to maintain clear associations between users, their devices, and the local hubs through which they communicate.

One of the core models is the Hub, which represents a local gateway in a home, building, or estate. Each hub has a unique name and an API key used for secure identification. It also records timestamps for registration and last activity, which are essential for monitoring connectivity.

The Device model captures the identity and behaviour of connected devices, such as solar meters or power switches. Each device is linked to a user and a hub, storing information such as the type of device, how it's connected (wired or wireless), its operating status, and the configuration of its power load pins. This allows the backend to understand what each device does and how it's being used in real-time.

Users are managed through the User model, which holds core identity information like usernames and email addresses. To support blockchain-based features, the Profile model extends each user with a Starknet wallet address. This address is essential for interacting with smart contracts and tracking token balances or energy transactions.

Relationships between these models are carefully designed. A single hub can be associated with many devices. Users may control multiple devices, and each device belongs to both a user and a specific hub. Additionally, every user has one profile, enabling seamless integration with blockchain interactions.

This structure ensures that data flows naturally throughout the system. Whether a device sends power usage updates or a user checks their token balance, the backend can route, store, and respond to these actions efficiently. Thanks to SQL Alchemy, these relationships are expressed in clean Python code, making it easy to develop and maintain even as the platform grows.

For a deeper look into the implementation, the full source code is available here: [Database Models](#).

3.3.5.28 API Layer and Communication

The user-related API routes focus on registration, authentication, and account management. When a user signs up, they are linked to profiles and devices they own or control. These routes enable secure onboarding and are responsible for generating authentication

credentials. The system ensures that users are verified before they can access sensitive information such as their energy usage or initiate trades.

Devices are at the heart of the system—sensors, smart plugs, and controllers report to the backend through the API. Device routes handle registration, status updates, and load configuration. When a new device is added, it is assigned a unique identifier, connection type, and metadata such as estate or user ownership. Devices can also be updated or queried through the API. For example, the */devices* route allows a hub to fetch all devices under its control using a secure API key.

Each device entry also links back to a user, and through the user's profile, it can be associated with a Starknet account address, which becomes important for blockchain-based token operations.

The profile routes provide a structured identity layer for users. A profile stores extended metadata, such as the user's account address used for blockchain transactions. While user routes handle login credentials and IDs, profiles attach contextual and blockchain-relevant information to those users. This separation keeps the system modular and clean, allowing flexibility in how profiles are extended in the future.

The trade routes facilitate energy token trading between users. Users can initiate a trade request, accept a pending trade, or delete a trade using secure, wallet-connected operations. These routes are tightly integrated with the Starknet token smart contract, ensuring that each trade results in a real token transfer that is recorded both off-chain (in the database) and on-chain (on Starknet).

The trade API also provides access to user trade history, empowering participants to monitor their peer-to-peer energy transactions in a transparent and verifiable manner.

Finally, token-related routes bridge the core system with the blockchain. These include operations like buying, creating Trade, and checking balance of each operation communicates with the deployed Starknet Cairo smart contract, using the Starknet Python SDK (*starknet_py*) to execute secure calls or transactions. For instance, `balanceOf` allows a user or hub to query their current token balance, which reflects their energy credits or consumption.

These routes work asynchronously, and each call is handled carefully to provide fast responses while ensuring synchronization with the on-chain state. Advanced resource bounds are applied during transaction execution to avoid overuse or failure due to gas limits.

For a deeper look into the implementation, the full source code is available here: [Dashboard routes](#).

3.4 Hardware Implementation

Household A is equipped with solar panels rated at 17.4V and 5W peak power.

The solar panel was connected to a charge controller rated at 12V to control the charging of the battery. The battery stores the energy. Uses two 7W and 3W battery. There is a current sensor that tracks the current consumption of both bulbs. Uses an Arduino microcontroller that helps in reading the current sensor values, coordinated the key pad and the LCD to help it display current values. There are two modes of switching the manual way and automatic way for each bulb. The automatic switching method uses MOSFETs. There is also a keypad that is used by the customer to enquire on the power consumption history, power token balances. There is a voltage divider that helps to monitor the level of the battery.

House B has one 7W bulb. Uses the 12V battery. Has a 10A current sensor that tracks the current consumed by the bulb when it is on. Has an Arduino micro controller which helps to read the value of the current from the current sensor, controls the LCDs and the keypads. Has a keypad that is used to feed command requests into the system. The LCD displays the value of the current detected by the current sensor.

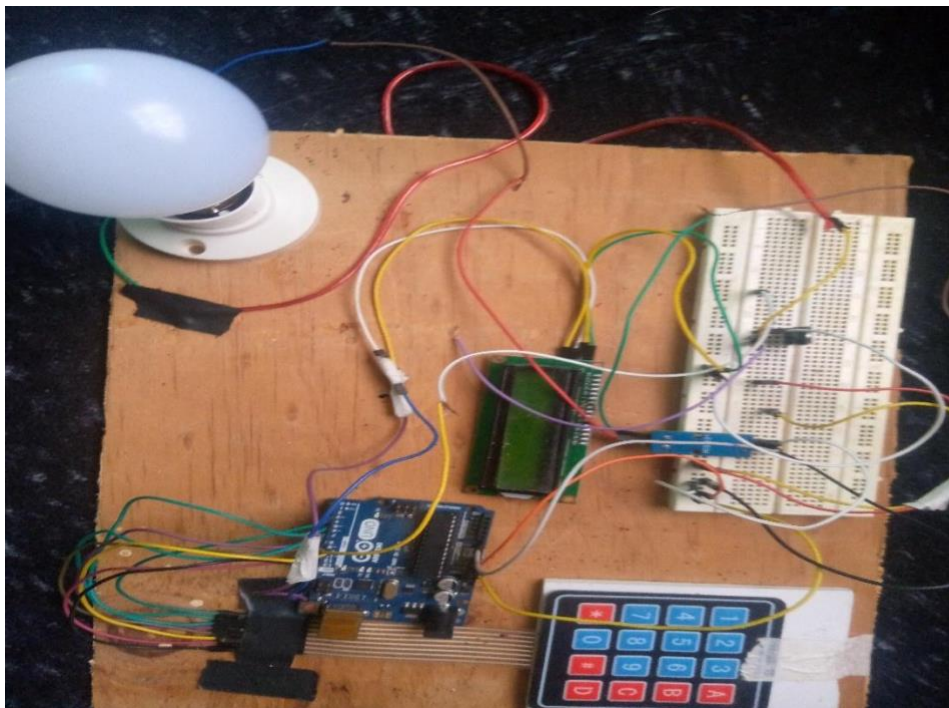


Figure 25: House A Implementation

Below is a diagram that shows the entire system starting with the solar panels which charge a 12 V battery. The bulbs draw energy from the battery once switched on. Two 7 w bulbs use MOSFETS as switches and one 3W bulb uses a manual switch. The keypad can be

used to switch on and off the bulbs, can be used to display current values, can be used to display the token balances and show the household Identification number.



Figure 26: House A and B Setup

3.5 Software implementation

Website and Blockchain implementation.

The website is able to display the graph of power consumption, the blockchain smart contracts containing the power tokens available (SCT). The cryptocurrency (STRK) balance of the customer.

The website dashboard also displays the trade history initiated by the two households.

3.6 Testing and Troubleshooting

Unit testing was conducted for each module (sensors, Arduino control, server API, blockchain). Used a multimeter to measure the continuity of the circuit, measure the battery capacity and confirm the current flowing through the bulbs to confirm current sensed by the current sensor.

Integration testing verified that data flows correctly from hardware sensors to the web dashboard and blockchain system. Calibration routines were refined to minimize sensor error (<5% deviation).

CHAPTER FOUR: RESULTS AND DISCUSSIONS

Below is a clear description and analysis of the project from the project.

4.1 Solar Panel

Connection of two solar panels in series with a rating of 8.4V and 9V.

The rating of one solar panel is:

$$\text{Peak voltage}=8.4V$$

$$\text{Peak power}=5W$$

$$\text{Peak current}=0.59A$$

The rating of the other solar panel is:

$$\text{Voltage}=9V.$$

$$\text{Power}=3W.$$

$$\text{Current}=0.5Ah.$$

Total nominal voltage of the solar panels.

$$9V + 8.4V = 17.4V$$

In a series connection, the lowest current rating limits the entire string.

So, the total current is minimum $(0.59A, 0.5A) = 0.5A$.

$$\text{Total current flow is}=0.5A.$$

Total power output of the solar panel:

$$\text{Total voltage} \times \text{Limited current}$$

$$17.4V \times 0.5A = 8.7W$$

With a eight hours of sunlight per day total energy generated is:

$$\text{Total power} \times \text{Hours of sunlight} = 8.7W \times 8hrs = 69.6Wh$$

4.2 Battery

The battery is used to supply power to both households

The battery is 12V with 7.2Ah/20HR:

The current that can be supplied by the battery is

$$\text{Current} = \text{Capacity} / \text{time}$$

$$7.2Ah / 20hrs. = 0.36A.$$

Capacity of the battery is given by:

$$7.2Ah \times 12V = 86.4Wh$$

Analysis of the time in hours of charging the battery when it is fully drained:

Energy of the battery/ Power of the solar panels:

$$86.4Wh / 8.7W = 9.93 \text{ HOURS.}$$

4.2.1 Monitoring battery capacity

Monitoring the battery capacity is very essential. This is of essence when trading power. One is able to monitor the level of the battery and decide to sell power until a particular level. For this project we used a voltage divider to help us monitor the voltage level of the battery through the Arduino micro controller. Since the input of an Arduino is 5V we scaled down the 12v to 5V using resistors of 6.8k Ω and 4.7k Ω connecting them in series.

$$V_{out} = V_{in} (R1 / (R1 + R2))$$

$$12V (4.7k\Omega / (4.7k\Omega + 6.8K\Omega)) = 4.9V \text{ approximated to } 5V.$$

4.3 Loads (Bulbs)

4.3.1 Household A

Household A uses two bulbs rated at 3W and 5W at 12V.

Theoretical computation of current of both bulbs of current consumed:

$$3W / 12V = 0.25 \text{ A}$$

$$7W / 12V = 0.58A$$

The two bulbs are connected in parallel the total amount of current that is drawn is:

$$0.25 + 0.58 = 0.83A.$$

For house A the two bulbs consume power for five hours a day.

So, the total amount of energy consumed by the bulbs is:

$$\text{Current drawn} \times \text{hours of usage} =$$

$$0.83A \times 5 \text{ hours} = 4.1667 \text{ Ah.}$$

4.3.2 Household B

Household B uses a 7W bulb as the load.

Current drawn by the bulb from the battery is

$$7W / 12V = 0.58A.$$

The bulb is required to draw power from the 12V battery for four hours per day.

The amount of energy drawn by the bulb is:

$$\text{Current drawn} \times \text{Usage hours.}$$

$$0.58A \times 4 \text{ hours} = 2.32 \text{ Ah.}$$

The percentage the bulb consumes from the battery is:

$$\text{Energy drawn from the bulb} / \text{Energy capacity of the battery} \times 100\%$$

$$2.32 \text{ Ah} / 7.2 \text{ Ah} \times 100\% = 32.389\%$$

So, household A supply about 32 % of its total energy to household B.

Power token: One power token is equivalent to 50W and one power token costs 0.0001 Starknets.

$$\text{Power} = \text{Voltage} \times \text{Current drawn}$$

$$= 12V \times 0.83A = 9.96W$$

For four hours that the bulbs draw power from the bulb

$$\text{Power} = 9.96W \times 4 \text{ Hours} = 39.84Wh.$$

Power factor of the bulbs:

$$\text{Power Factor} = \text{Apparent Power (kVA)} / \text{Real Power (kW)}$$

In DC circuits, voltage and current are always in phase there's no reactance, only resistance.

So;

$$\text{In DC: Real Power} = \text{Apparent Power Factor} = 1.$$

A DC bulb draws current directly through a resistive load:

- No phase shift (since there's no AC waveform).
- No reactive power.
- Hence, power factor is effectively 1 by definition.

Since we are using DC power and simple resistive bulbs, the power factor is 1.

4.4 The feature codes

Feature codes are identifiers used in software or hardware systems to enable, track, or manage specific functions, capabilities, or configurations. They are especially common in systems where functionality is modular or license-based.

The feature codes used include:

- When one press the 0 from the keypad one is able to note the device ID. It will display whether it is household one or household two.

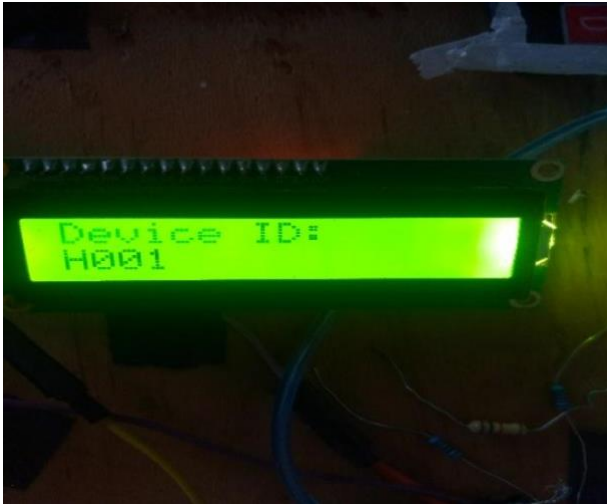


Figure 27:House ID

- When one presses 1 from the keypad it displays the nature of connection whether off or on. This is used to show whether house A is supplying power to house B. If it is supplying power the status of the connection will be ON and if it is not supplying the status will be OFF

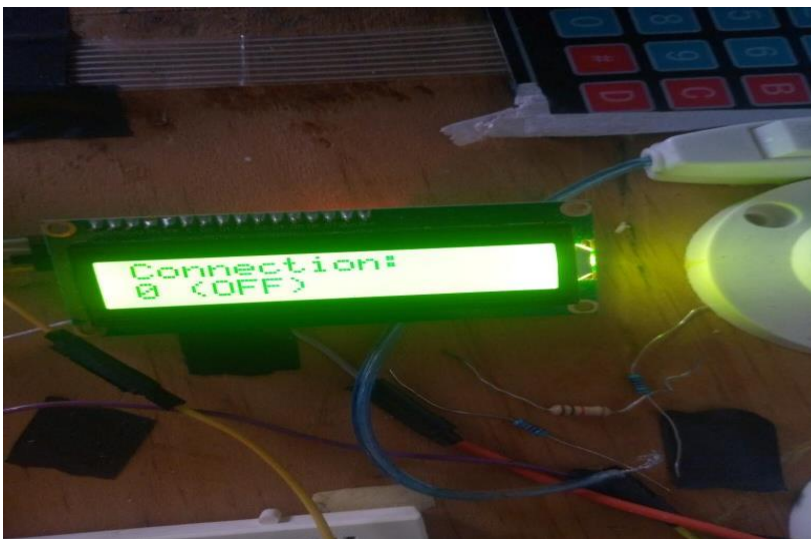


Figure 28:Connection nature

- Code 2 displays the average current the load is using. If one wants to see the amount of current being use by the load when they are on will press code 2 from the keypad.

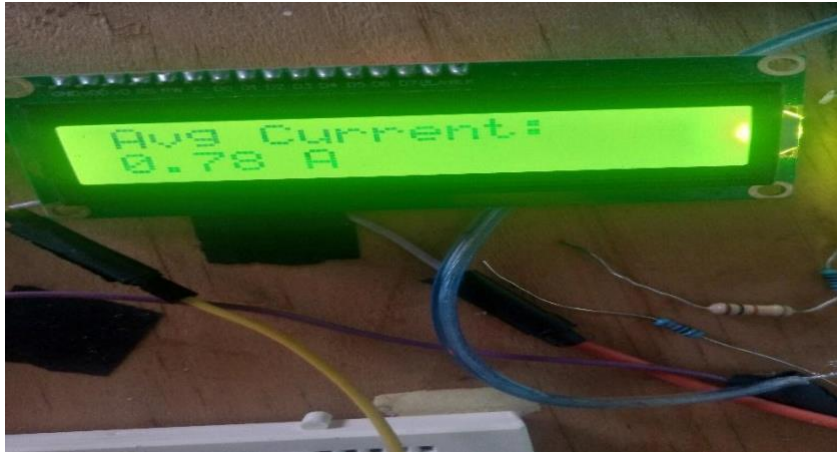


Figure 29:Current Value

- Code 3 displays the balance of the power token by the customer from house B.



Figure 30:Token balance

- Code 4 switches on and of the bulb and indicates that the status of the bulb whether there ON or OFF



Figure 31:LED status

4.5 Power losses

Possible sources of power losses in:

i. Solar panel

- Reflection Losses: Some sunlight is reflected off the glass surface of the panel.
- Temperature Losses: Efficiency drops as temperature increases.
- Mismatch Losses: Variability between panels (e.g., shading or different specs).
- Dirt and Dust: Reduces light hitting the surface.
- Wiring Losses (DC): Resistance in cables from panels to the battery.

ii. Battery

- Charge/Discharge Inefficiency: Batteries lose energy during both charging and discharging (efficiency ~80–95% depending on type).
- Internal Resistance: Causes heat and energy loss.
- Self-discharge: Batteries naturally lose charge over time, even when not used.
- Temperature Effects: Cold or very hot environments reduce performance.
- Aging: Capacity fades over time, increasing effective loss.

iii. Wires

- Ohmic Losses: Energy lost due to resistance in cables (depends on cable gauge and length).
- Loose/Corroded Connections: Increase resistance, cause hotspots.
 - Undersized Wires: Cause significant voltage drops and heat.

4.6 Frontend

4.6.1 Authentication

Figure below demonstrates the user registration process. In this example, a user with the username giturutrevor and email giturutrevor@gmail.com is registering for an account. For security reasons, the password is not displayed, but it is securely hashed and stored in the backend database upon submission.

Register

Username

giturutrevor

Email

giturutrevor@gmail.com

Password

.....

Confirm Password

.....|

☐ Show Passwords

Register

Already have an account? [Login](#)

Figure 32:Authentication

The figure below shows the login process for the user giturutrevor using the same credentials entered during registration. After submitting the form, a confirmation message appears indicating successful authentication. The system then redirects the user to the dashboard interface.

The image shows a login interface. At the top, the word "Login" is centered in a large, bold, black font. Below it, a light green rectangular box contains the text "Login successful! Redirecting..." in a green font, with a green progress bar underneath. The form then has two input fields: "Email" with the value "giturutrevor@gmail.com" and "Password" with masked characters ".....". Below the password field is a checkbox labeled "Show Password". At the bottom of the form is a large blue button with the text "Login" in white. Below the button is a link that says "Don't have an account? Register". The entire form is set against a light gray background.

Login

Login successful! Redirecting...

Email

giturutrevor@gmail.com

Password

.....

☐ Show Password

Login

[Don't have an account? Register](#)

Figure 33:Login

4.6.2 Profile

The figure below illustrates the process of creating a user profile. After successfully logging in, the user proceeds to complete their profile by providing additional personal details

Profile Settings

First Name

Gituru

Last Name

Trevor

Date of Birth

08 / 25 / 2000

Gender

Male

Phone Number

0701342609

Account Address

0x4b1cea62c4bf725c69b8f1e12ed5c7525bbf4bada1a3712cc73fb4c9e4adb5a

Connected: 0x4b1c...db5a

Copy

Disconnect

Email Confirmed

✓

Save Changes

Figure 34:Profile

This interface appears immediately after the user provides their profile details. A confirmation email containing a One-Time Password (OTP) is automatically sent to the registered email address. The user is prompted to enter this OTP to verify their email and activate the account. This step enhances account security and ensures that the email is valid and accessible by the user.

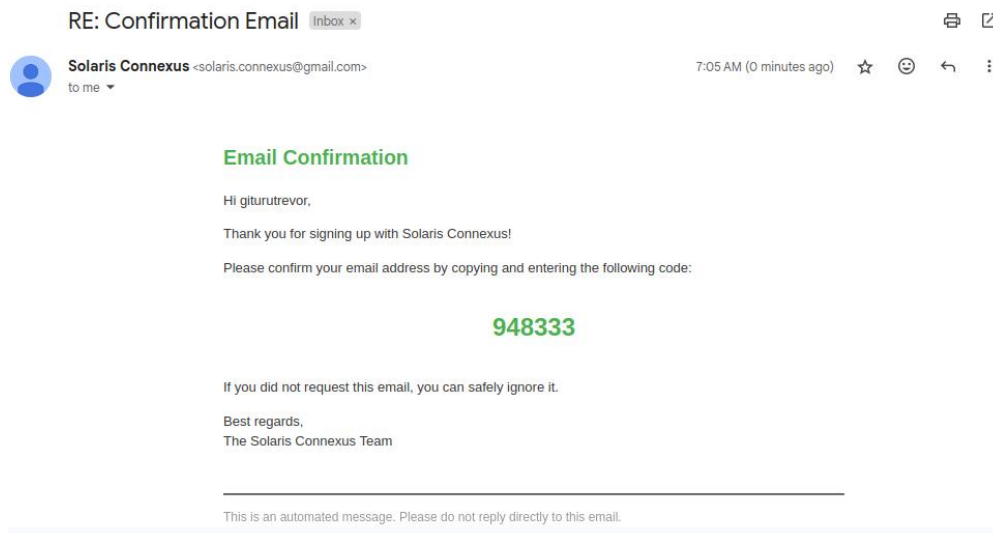


Figure 35:Email confirmation

After email verification, the user is prompted to connect their blockchain wallet. In this case, the user connects a Braavos wallet by clicking the "Connect Wallet" button. Upon successful connection, the user's wallet address (0x4b1cea...adb5a) is linked to their profile. This wallet address is crucial for enabling blockchain-based transactions within the system, including token management and power trading.

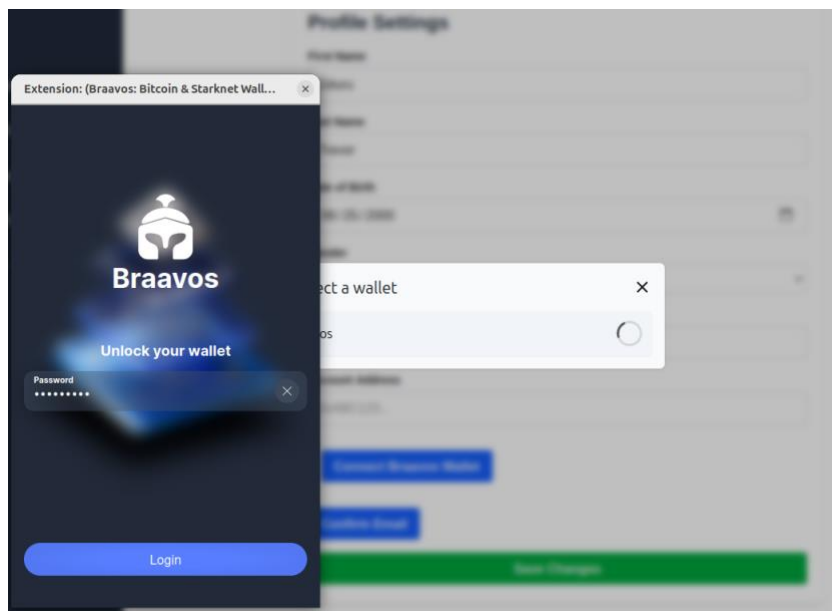


Figure 36:Braavos wallet

The final figure displays a profile summary card showing the user's personal information in a clean and structured format. After completing email verification and wallet connection, the system presents the user's profile:

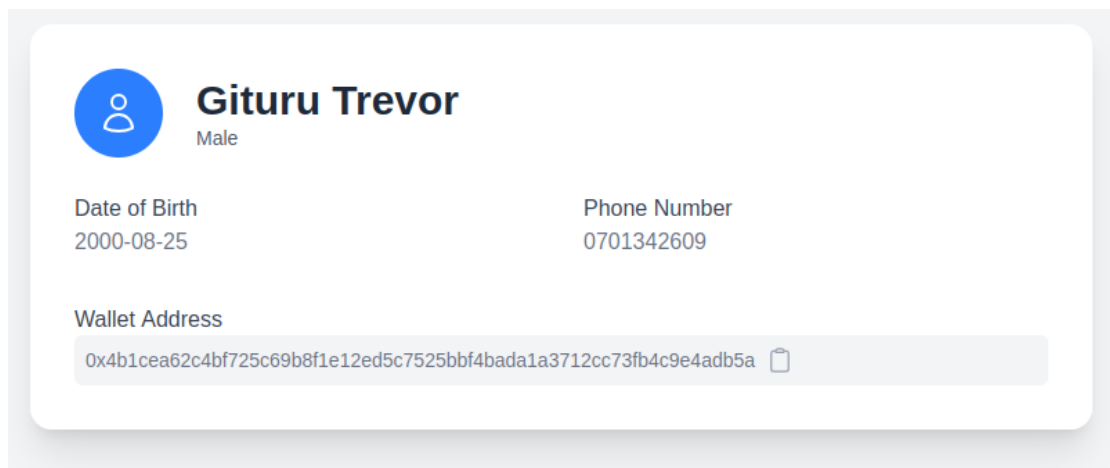


Figure 37: Personal details

4.6.3 Token Purchase

After completing profile setup and wallet connection, the user navigates to the token purchase page. This interface allows the user to acquire Solar Credit Tokens (SCT), which are used within the system for transactions such as energy trading or service access.

The user is prompted to enter the amount of SCT they wish to purchase. In this demonstration, the user inputs 5 SCT. Upon confirming the transaction, the connected Braavos wallet is triggered to process the payment. The system automatically calculates the cost in STRK tokens deducting from the wallet balance to complete the purchase.

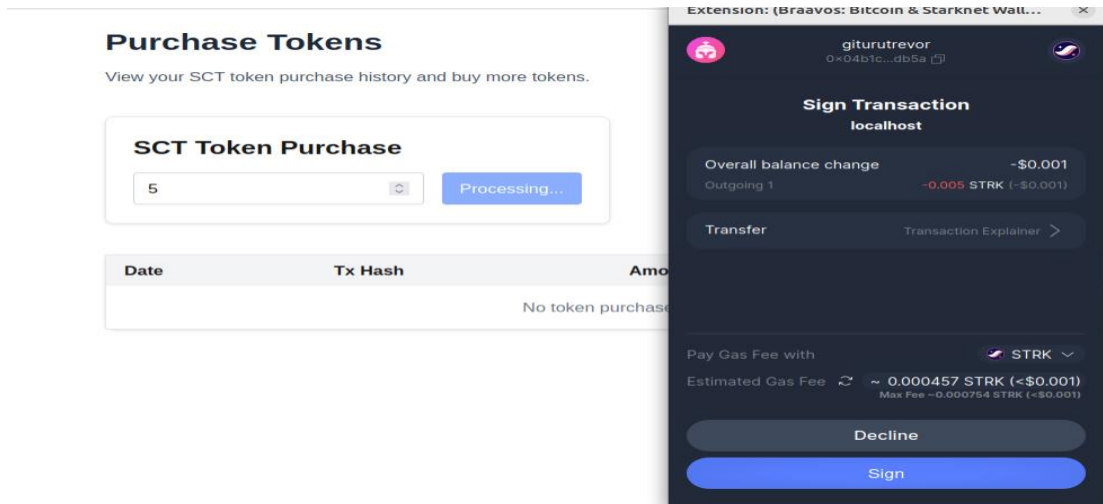


Figure 38:purchase token

The figure below displays the updated token purchase history table immediately after a successful transaction.

Purchase Tokens

View your SCT token purchase history and buy more tokens.

SCT Token Purchase

Date	Tx Hash	Amount	Status
5/28/2025	0x5c24...94c4	5 STC	Confirmed

Figure 39:updated power Tokens

The figure below shows the detailed transaction information displayed on StarkScan after clicking the transaction hash (tx hash) of a token purchase. StarkScan provides a comprehensive view of the transaction, including sender and receiver addresses, amount transferred, transaction status, block details, and timestamp, ensuring transparency and on-chain verification.

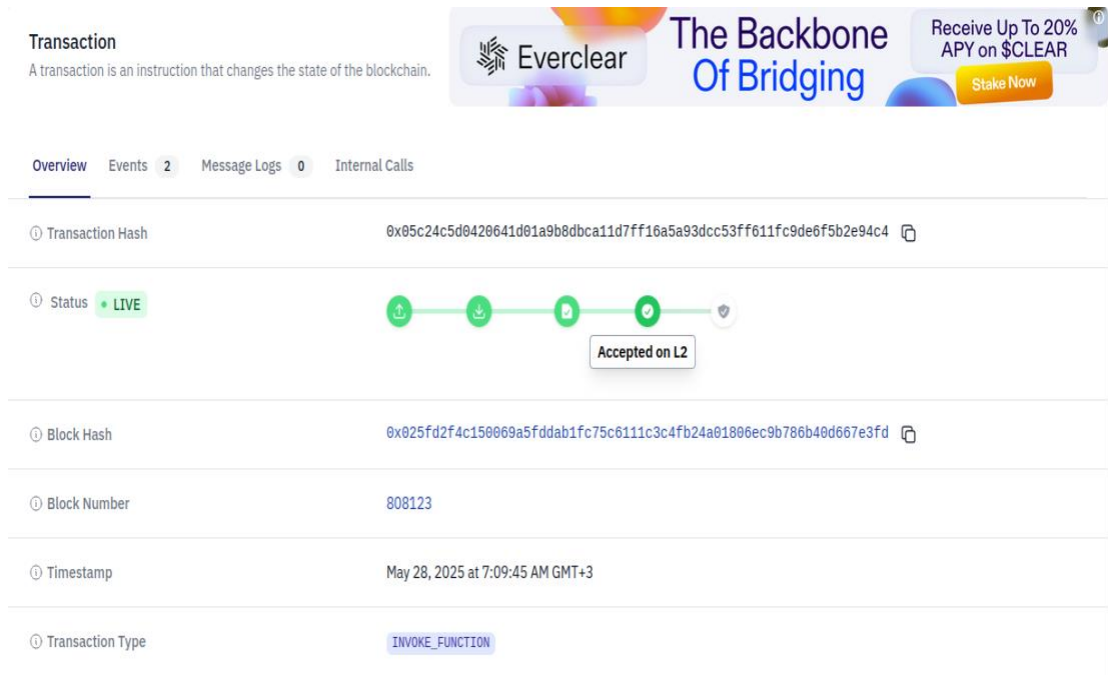


Figure 40: Transaction details

4.6.4 Trades

The figure below shows the trade section where the user has initiated a trade request to exchange 2 SCT tokens for 0.02 STRK. The 2 SCT tokens have been transferred to a smart contract acting as a middle party to securely manage the trade.

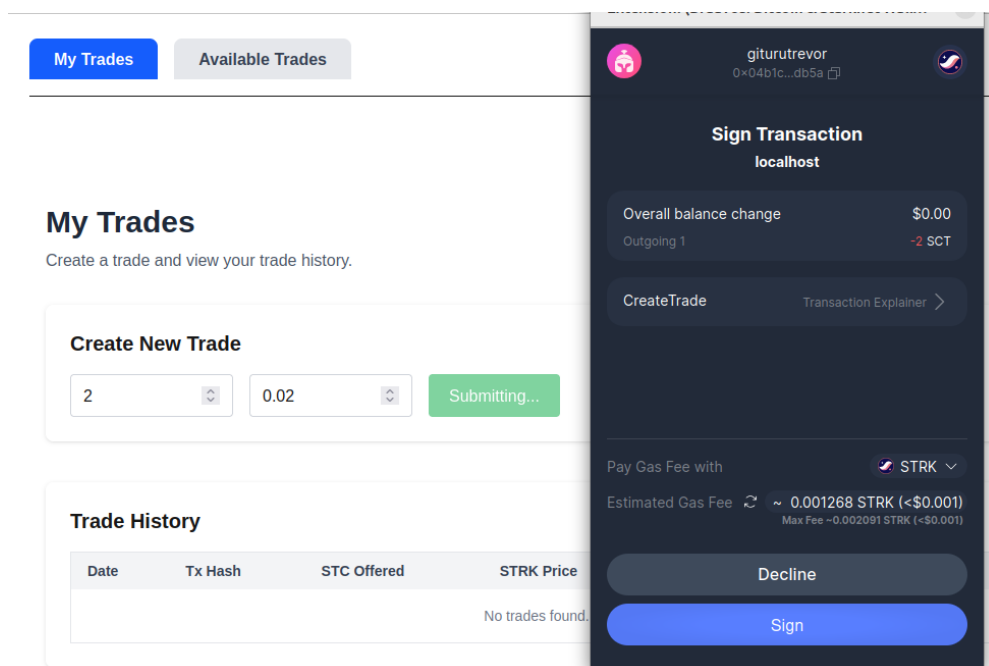


Figure 41: Trades

The figure below demonstrates the user deleting an existing trade request by calling the deleteTrade function through their connected Braavos wallet. This action cancels the pending trade and releases any locked tokens back to the user's account.

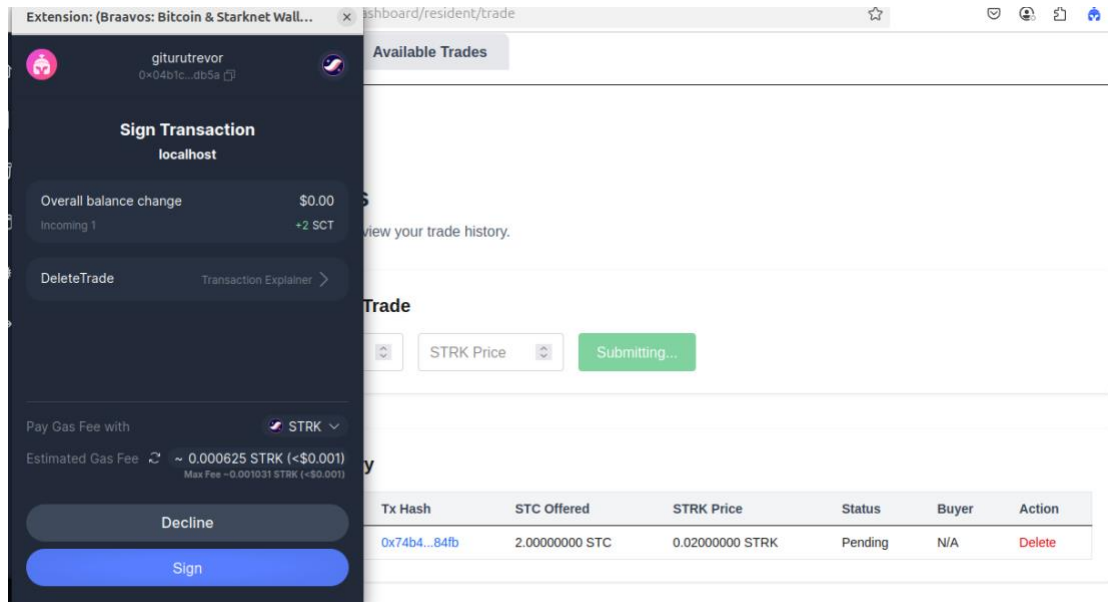


Figure 42:trade

The figure below displays the updated trade history table after the user cancelled a trade request. The cancelled trade is clearly marked, reflecting the current status and confirming that the associated tokens have been released.

My Trades

Create a trade and view your trade history.

Create New Trade

STC Offered

STRK Price

Create Trade

Tx Submitted: 0x171a1a0d264084b8c0e8d1543d5c2f407861e2048300c264053f0e71cfe1faf

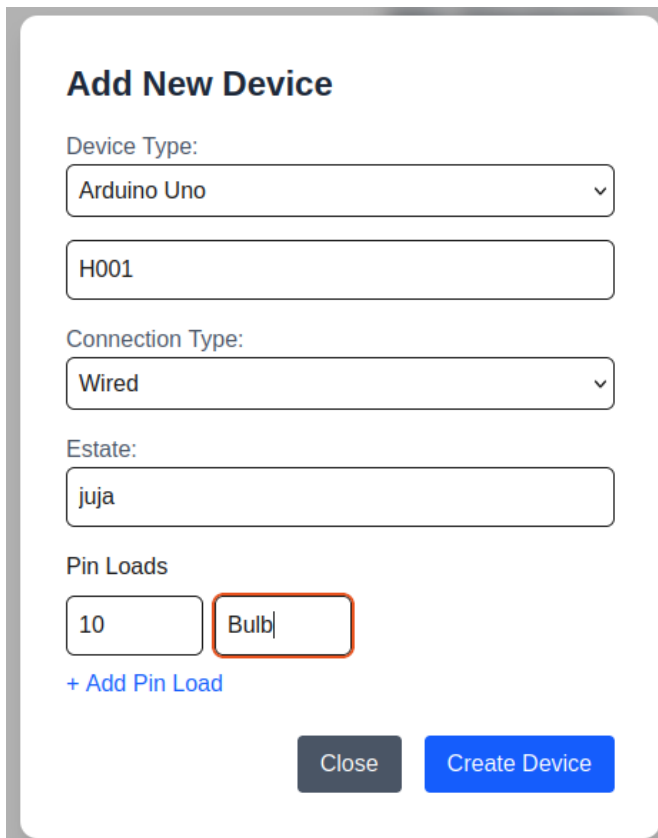
Trade History

Date	Tx Hash	STC Offered	STRK Price	Status	Buyer	Action
5/28/2025	0x171a...1faf	2.00000000 STC	0.02000000 STRK	Cancelled	N/A	

Figure 43: Trade history

4.6.5 Devices

The figure below shows the user adding a new device by specifying its load (power consumption) and the connected pin on the controller, along with selecting the estate to which the device belongs.



The image shows a web form titled "Add New Device". It contains several input fields and buttons. The "Device Type" is a dropdown menu with "Arduino Uno" selected. Below it is a text field containing "H001". The "Connection Type" is another dropdown menu with "Wired" selected. The "Estate" field is a text box containing "juja". Under "Pin Loads", there are two input boxes: one with "10" and another with "Bulb" which is highlighted with a red border. Below these is a blue link "+ Add Pin Load". At the bottom right are two buttons: "Close" (grey) and "Create Device" (blue).

Add New Device

Device Type:
Arduino Uno

H001

Connection Type:
Wired

Estate:
juja

Pin Loads

10 Bulb

[+ Add Pin Load](#)

Close Create Device

Figure 44: addition of devices

The figure below shows the created device's information, including two options: one to toggle the device's load on or off, and another to display real-time power consumption. This interface enables users to actively manage and monitor their devices within the estate.

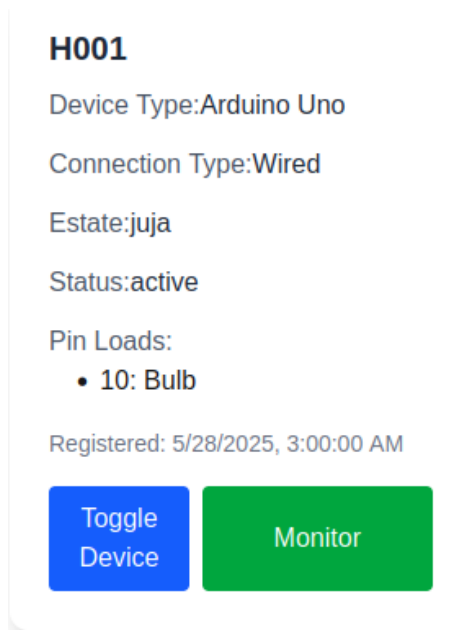


Figure 45: Household details

4.6.6 Home

The figure below shows the home section dashboard summarizing key user information, including the number of connected devices, current STRK token balance, and SCT (Solar Credit Token) balance. This provides users with a quick snapshot of their energy assets and system status.

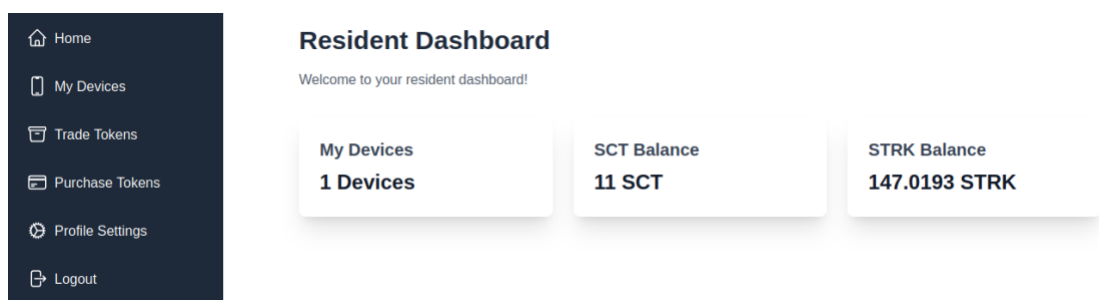


Figure 46: Resident dashboard

The figure below displays the Braavos wallet account interface showing the user's SCT (Solar Credit Token) balance. It includes options to transfer tokens to other users and a history

of signed transactions, allowing users to manage their token holdings and review past activities directly within the wallet.

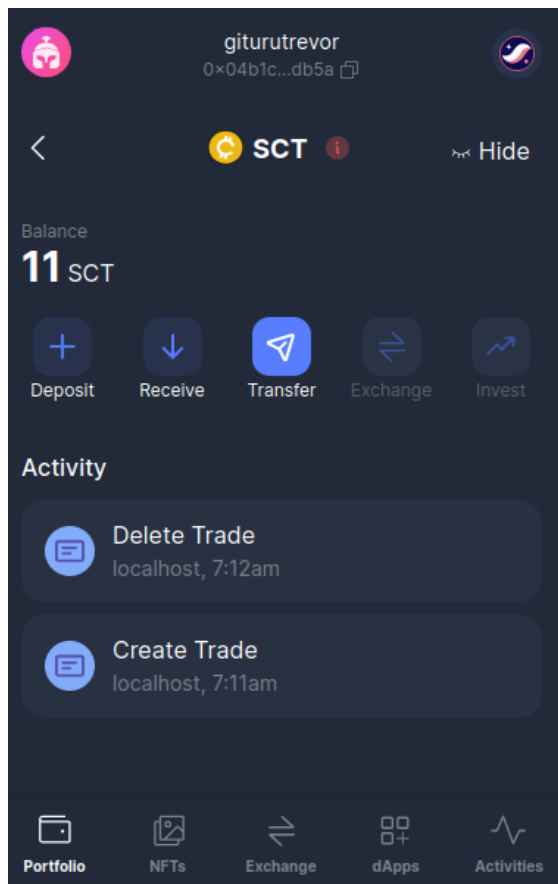


Figure 47: Braavos wallet

4.7 Central Backend

The figure below shows the FastAPI backend server starting up and handling incoming GET and POST requests from the frontend. The server processes these requests and responds with HTTP status code 200, indicating that the requests were successfully received and handled.

```

ubuntu@499235-web-02:~/solaris_conexus/backend-central$ source myenv/bin/activate
(myenv) ) ubuntu@499235-web-02:~/solaris_conexus/backend-central$ python3 main.py
/home/ubuntu/solaris_conexus/backend-central/myenv/lib/python3.12/site-packages/pydantic/_internal/_config.py:373: UserWarning:
Valid config keys have changed in V2:
* 'orm_mode' has been renamed to 'from_attributes'
  warnings.warn(message, UserWarning)
INFO: Started server process [18476]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: 41.139.178.89:27105 - "OPTIONS /register HTTP/1.1" 200 OK
INFO: 41.139.178.89:27105 - "POST /register HTTP/1.1" 200 OK
INFO: 41.139.178.89:32205 - "OPTIONS /login HTTP/1.1" 200 OK
INFO: 41.139.178.89:32205 - "POST /login HTTP/1.1" 200 OK
INFO: 41.139.178.89:27225 - "OPTIONS /get_user_details HTTP/1.1" 200 OK
INFO: 41.139.178.89:27225 - "GET /get_user_details HTTP/1.1" 200 OK
INFO: 41.139.178.89:27225 - "GET /get_user_details HTTP/1.1" 200 OK
INFO: 41.139.178.89:31882 - "POST /login HTTP/1.1" 200 OK
INFO: 41.139.178.89:31882 - "GET /get_user_details HTTP/1.1" 200 OK
INFO: 41.139.178.89:31882 - "GET /get_user_details HTTP/1.1" 200 OK
INFO: 41.139.178.89:27530 - "OPTIONS /get_profile HTTP/1.1" 200 OK
INFO: 41.139.178.89:27530 - "GET /get_profile HTTP/1.1" 200 OK
INFO: 41.139.178.89:27530 - "GET /get_profile HTTP/1.1" 200 OK
INFO: 41.139.178.89:31735 - "OPTIONS /create_profile HTTP/1.1" 200 OK
INFO: 41.139.178.89:27942 - "OPTIONS /confirm_email HTTP/1.1" 200 OK
INFO: 41.139.178.89:27942 - "GET /confirm_email HTTP/1.1" 200 OK
INFO: 41.139.178.89:28546 - "OPTIONS /create_profile HTTP/1.1" 200 OK
INFO: 41.139.178.89:28546 - "POST /create_profile HTTP/1.1" 201 Created
INFO: 41.139.178.89:30834 - "GET /get_profile HTTP/1.1" 200 OK
INFO: 41.139.178.89:30598 - "OPTIONS /get_token_purchases HTTP/1.1" 200 OK
INFO: 41.139.178.89:30598 - "GET /get_token_purchases HTTP/1.1" 200 OK
INFO: 41.139.178.89:30598 - "GET /get_token_purchases HTTP/1.1" 200 OK
INFO: 41.139.178.89:30459 - "OPTIONS /add_token_purchase HTTP/1.1" 200 OK
INFO: 41.139.178.89:30459 - "POST /add_token_purchase HTTP/1.1" 201 Created
INFO: 41.139.178.89:30459 - "GET /get_token_purchases HTTP/1.1" 200 OK
INFO: 41.139.178.89:29023 - "OPTIONS /get_user_trade HTTP/1.1" 200 OK
INFO: 41.139.178.89:29023 - "GET /get_user_trade HTTP/1.1" 200 OK
INFO: 41.139.178.89:29023 - "GET /get_user_trade HTTP/1.1" 200 OK
INFO: 41.139.178.89:30288 - "OPTIONS /create_trade HTTP/1.1" 200 OK
INFO: 41.139.178.89:30288 - "POST /create_trade HTTP/1.1" 201 Created
INFO: 41.139.178.89:30288 - "GET /get_user_trade HTTP/1.1" 200 OK
INFO: 41.139.178.89:29347 - "OPTIONS /cancel_trade HTTP/1.1" 200 OK
INFO: 41.139.178.89:29347 - "POST /cancel_trade HTTP/1.1" 200 OK
INFO: 41.139.178.89:29347 - "GET /get_user_trade HTTP/1.1" 200 OK
INFO: 41.139.178.89:29471 - "OPTIONS /get_user_details HTTP/1.1" 200 OK
INFO: 41.139.178.89:29471 - "GET /get_user_details HTTP/1.1" 200 OK
INFO: 41.139.178.89:29471 - "GET /get_user_details HTTP/1.1" 200 OK

```

Figure 48:central backend

The figure below illustrates the backend receiving toggle and real-time stream requests from the frontend for device control and monitoring. It creates a WebSocket connection to enable continuous real-time data streaming. The backend processes these requests and publishes relevant messages to the Hive MQTT broker, facilitating communication with the estate hub for live energy management.

```

[MQTT] Published to juja/commands: {"device": 1, "instruction": 2}
INFO: 41.139.178.89:32927 - "GET /toggle_device?device_id=1 HTTP/1.1" 200 OK
[MQTT] Connected with result code 0
INFO: ('41.139.178.89', 26441) - "WebSocket /device/stream?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJnaXR1cnV0cmV2b3IiLCJleHAiOiJlNDg0MDY3NTF9.5uKSG1SWJ1APFtJFnEgu3rB-AtubxPgKvZnGVAPG6II" [accepted]
[MQTT] Listener registered for juja/power/1
[MQTT] Published to juja/commands: {"device": 1, "command": "stream"}
INFO: connection open
INFO: ('41.139.178.89', 32947) - "WebSocket /device/stream?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJnaXR1cnV0cmV2b3IiLCJleHAiOiJlNDg0MDY3NTF9.5uKSG1SWJ1APFtJFnEgu3rB-AtubxPgKvZnGVAPG6II" [accepted]
[MQTT] Listener registered for juja/power/1
[MQTT] Published to juja/commands: {"device": 1, "command": "stream"}
INFO: connection open
INFO: connection closed
INFO: connection closed

```

Figure 49: receiving Backend Toggle

The figure below shows the server publishing messages to the HiveMQTT broker on the /juja/commands topic. These messages contain commands and control signals that are sent to the estate hub for managing connected devices and energy distribution.

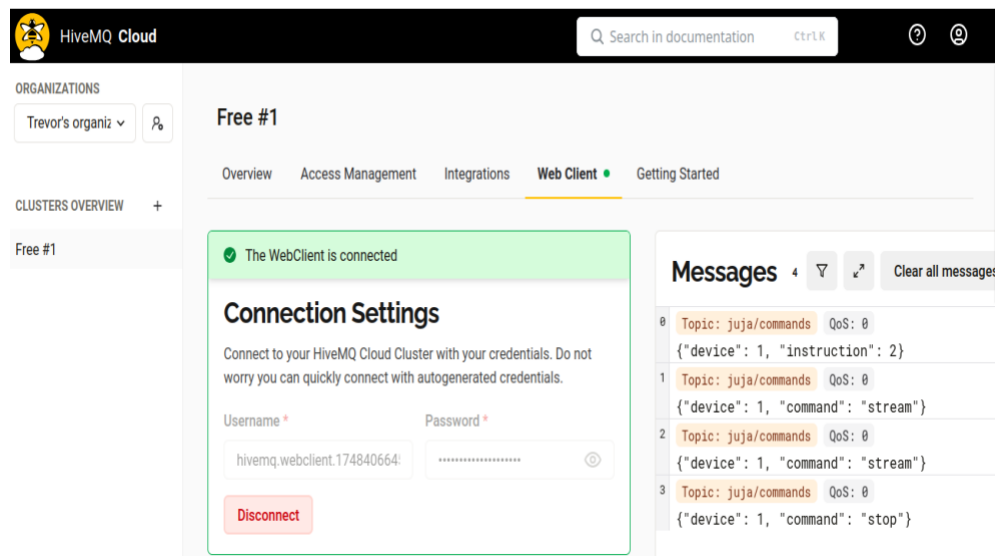


Figure 50: Client Connection Setting

The following image displays all the tables created in the PostgreSQL database along with sample data from the user's table. This overview highlights the database schema and shows how user information is stored within the system.

```

ubuntu@499235-web-02:~$ sudo -u postgres psql
Password for user postgres:
psql (12.22 (Ubuntu 12.22-0ubuntu0.20.04.4))
Type "help" for help.

postgres=# \c solaris
You are now connected to database "solaris" as user "postgres".
solaris=# \dt
          List of relations
Schema |      Name      | Type | Owner
-----+-----+-----+-----
public | alembic_version | table | postgres
public | devices         | table | postgres
public | hubs            | table | postgres
public | profiles        | table | postgres
public | token_purchases | table | postgres
public | trade_requests  | table | postgres
public | users           | table | postgres
public | wallets         | table | postgres
(8 rows)

solaris=# TABLE devices;
 id | device_type | device_id | connection_type | estate | status |      pin_loads      | created_at | user_id
-----+-----+-----+-----+-----+-----+-----+-----+-----
  1 | Arduino Uno | H001      | Wired           | juja   | active | [{"pin": "10", "load": "Bulb"}] | 2025-05-28 |      2
(1 row)

```

Figure 51:database schema

4.8 Smart Contract SCT

This image displays key details of the deployed smart contract on Voyager, including the contract class hash, type, name, number of token holders, and total token supply. It gives a high-level summary of the contract's identity and distribution metrics.

The screenshot shows the Voyager interface for a smart contract named "SolarisConexusToken SCT". At the top, there's a banner for "slise" with a promotion: "Only users who have staked tokens with more than 0.01 ETH with balance on Metamask who own in-game NFT will see this ad". Below the banner, the contract details are listed:

- Contract:** 0x0696e1ba36b1b46ccfda4a6ae963b12c932e36d1a1008d5d0a03033b12d86827
- Balance:** 0.0 ETH
- Source Code Status:** Not Verified
- Type:** ERC20
- Token:** SolarisConexusToken SCT
- Number of Holders:** 4
- Token Total Supply:** 10,000
- Created On:** May 12 2025 10:49:18
- Deployed At:** 0xb518...86933b
- Deployed By:** 0x040d...74d90b
- Class Hash:** 0x013491a662b3d34c29ac324735aa7ffebfc6f70bb5d54e71cb89d563a87259c7

Figure 52: Smart Contract SCT

The figure below shows the ERC-20 token balance held by the smart contract, specifically displaying a balance of 0.216 STRK. This confirms the contract's ability to store and interact with Starknet tokens.

Portfolio

Tokens **NFTs**

List of verified ERC20 tokens owned by this contract

VERIFIED 1 ALL

Tokens Value: \$0.04

ASSET	SYMBOL	CONTRACT ADDRESS	BALANCE	VALUE
StarkNet Token	STRK	0x0471...938d	0.216	\$0.035382

Figure 53: Smart Contract SCT

This image presents the list of callable functions within the smart contract, including create trade, delete trade, and buy. These functions define the core interactions available for token trading and management.

Contract data

Transactions Token Transfers Bridge Txns Events Messages **Account Calls** Token Holders Code Class History Read Contract Write Contract Read Storage

ID	TXN HASH	CALL	FROM	BLOCK	AGE
808129_1_2	0x171a...1faf		0x04b1...db5a	808129	36 minutes ago
808126_1_2	0x74b4...84fb		0x04b1...db5a	808126	37 minutes ago
808123_3_2	0x5c24...94c4		0x040d...d90b	808123	39 minutes ago
805535_9_2	0x598e...c009		0x040d...d90b	805535	a day ago
805529_2_2	0x3656...820a		0x074d...f42e	805529	a day ago
805473_5_2	0xa195...9442		0x040d...d90b	805473	a day ago
805469_4_2	0x37d3...1007		0x074d...f42e	805469	a day ago
805453_0_2	0x6691...9101		0x040d...d90b	805453	a day ago

Figure 54: callable functions

The figure below shows the emitted events captured on Voyager, such as Trade and Transfer. These events provide transparency into the contract's operations, allowing users and developers to track key interactions and token movements on-chain.

Events

Transactions Token Transfers Bridge Txns **Events** Messages Account Calls Token Holders Code Class History Read Contract Write Contract Read Storage

ID	NAME	BLOCK	TXN HASH	AGE
808129_1_1		808129	0x171a...1faf	41 minutes ago
808129_1_0		808129	0x171a...1faf	41 minutes ago
808126_1_1		808126	0x74b4...84fb	42 minutes ago
808126_1_0		808126	0x74b4...84fb	42 minutes ago
808123_3_0		808123	0x5c24...94c4	44 minutes ago

Figure 55: Voyager events

4.9 Estate Hub

The figure below illustrates the initialization of the estate hub, which begins by subscribing to the MQTT topic 'juja/commands'. It then listens to the serial port for incoming signals from connected devices. In this instance, the hub detects device ID H001 and responds

by sending a JSON payload to the Arduino with two key parameters: balance set to 10 and instruction set to 1, which indicates the load should be connected. The [Raw] log section captures real-time data from the Arduino, including current, voltage, and load status (noting the load is currently off). This data is then stored in a database, and a confirmation message prints the data entry ID upon successful insertion.

```
(venv2) razaoul@trevor-HP-650-Notebook-PC:~/Documents/software_dev/solaris_conexus/estate-backend$ python3 src/main.py
[Info] Trying to connect to /dev/pts/4...
[Main] Started monitoring 1 devices.
[MQTT] Connected with result code 0
[MQTT] Subscribed to juja/commands
[Success] Connected to /dev/pts/4
[Raw] {"device_id": "H001"}
[Info] Received device ID: H001
[Sent] 10,1
[Raw] {"current": 0.44, "voltage": 4.13, "req": "false"}
[Info] Received 'current' reading; exiting connect()
[Raw] {"current": 0.38, "voltage": 3.76, "req": "false"}
[Info] Received 'current' reading; exiting connect()
[Sent] 10,1
[Raw] {"current": 0.42, "voltage": 3.82, "req": "false"}
[Decoded] Current: 0.42 A, Voltage: 3.82 V, Request: false
[Info] Power consumption recorded successfully: {6112}
[Redis] Accumulated: 15.661200000000077 W for H001
```

Figure 56: Estate Hub

The figure below illustrates how power consumption data for each device is stored in Redis, with new values added every 3 seconds. Once the total consumption reaches 50W, the system automatically triggers the token destruction function, which burns a token from the user's balance. Upon successful execution, the transaction hash (tx hash) is displayed for reference. The user's token balance is then updated in real time, showing the new balance of 9 SCT tokens.


```

[Info] Power consumption recorded successfully: {6135}
[Redis] Accumulated: 50.991700000000008 W for H001
[Action] Threshold reached! Minting tokens on-chain...
[Sent] 10,1
[Raw] {"current": 0.21, "voltage": 3.8, "req": "false"}
[Decoded] Current: 0.21 A, Voltage: 3.8 V, Request: false
[Info] Power consumption recorded successfully: {6136}
[Redis] Accumulated: 1.78970000000000796 W for H001
[Sent] 10,1
[Raw] {"current": 0.49, "voltage": 3.7, "req": "false"}
[Decoded] Current: 0.49 A, Voltage: 3.7 V, Request: false
[Info] Power consumption recorded successfully: {6137}
[Redis] Accumulated: 3.60270000000000795 W for H001
[Sent] 10,1
[Raw] {"current": 0.26, "voltage": 3.51, "req": "false"}
[Decoded] Current: 0.26 A, Voltage: 3.51 V, Request: false
[Info] Power consumption recorded successfully: {6138}
[Redis] Accumulated: 4.515300000000008 W for H001
0xfe9116b5b4da45a61c17a1c53fac9efd0bad22e0080e0c4b1b48a716c8473c
[Sent] 10,1
[Info] Updated device 5 token_balance to 9

```

Figure 57:Power Consumption history

The figure below shows the estate hub receiving MQTT messages instructing it to start and later stop streaming power consumption data for Device 1. These commands are published to the subscribed MQTT topic, and the hub responds accordingly by initiating or terminating the real-time data stream from the specified device.

```

[MQTT] Message received on juja/commands: {"device": 1, "command": "stream"}
[Stream] Started streaming for device 1
-----
[MQTT] Message received on juja/commands: {"device": 1, "command": "stop"}
[Stream] Stopped streaming for device 1

```

Figure 58:estate hub receiving MQTT messages

The figure below displays the estate hub receiving an MQTT command to toggle the load status. The command specifies device with id of 5 and instruction 2 meaning to switch state of load. Upon processing the command, the hub sends it to the connected device, resulting in the load status changing to true (indicating the load is now turned on).

```
[MQTT] Message received on juja/commands: {"device": 5, "instruction": "2"}
[MQTT] Updated instruction for device 5 to 2
[Raw] {"current": 0.44, "voltage": 4.13, "req": "false"}
[Decoded] Current: 0.44 A, Voltage: 4.13 V, Request: false
[Info] Power consumption recorded successfully: {6199}
[Redis] Accumulated: 6.062900000000077 W for H001
[Sent] 8,2
[Raw] {"current": 0.4, "voltage": 3.96, "req": "true"}
```

Figure 59:estate hub receiving an MQTT command

CHAPTER SIX: CONCLUSION

5.1 Introduction

In order to provide a decentralized, effective, and transparent framework for energy production, consumption, and trade, the suggested energy management system incorporates cutting-edge technology such as blockchain, Internet of Things-based communication, and renewable energy generation. The system tackles important issues in contemporary energy management by utilizing sophisticated hardware and software, secure blockchain-based transactions, strong communication infrastructure, and home integration.

Households should be empowered by the system's ability to optimize energy use, provide excess energy, and profit financially via tokenized energy trading. The blockchain ensures transparency, security, and confidence in every transaction, while the communication protocols and infrastructure guarantee dependable, low-latency data flow. Furthermore, the system may be adjusted to meet growing demand and user involvement thanks to the utilization of scalable and effective hardware and software tools.

By lowering dependency on conventional energy sources, this all-encompassing strategy encourages sustainability, increases user cost-effectiveness, and offers a scalable option for wider deployment. The suggested techniques provide a route to a creative, decentralized energy environment and are in line with contemporary energy objectives.

The increasing global emphasis on sustainable energy solutions has highlighted the need for innovative systems that not only harness renewable resources efficiently but also ensure transparency and user empowerment. This project, Blockchain Enabled Solar Energy, successfully demonstrated a low-cost, user-friendly prototype that integrates solar energy storage, load management, real-time monitoring, and blockchain-based transaction recording. Through systematic implementation, the project achieved the key objectives:

- **Solar Energy Utilization:** A simple yet effective setup using a solar-charged power bank provided a clean and stable power source for DC loads such as LEDs and phone charging.
- **Load Control:** The use of BJTs as electronic switches controlled by Arduino enabled precise and user-driven management of energy consumption.
- **Real-Time Monitoring:** Incorporating a current sensor with LCD display allowed for transparent and immediate feedback on energy usage.

- **Secure Data Recording:** The integration of a private blockchain network ensured that all energy transactions—such as load activation, current drawn, and timestamps—were permanently and immutably recorded using smart contracts.

Compared to existing work in the field, this project addressed notable gaps by focusing on individual-level solar applications, minimizing hardware complexity, and maximizing accessibility. Unlike large-scale blockchain energy trading platforms that require sophisticated and costly IoT frameworks, this design remains affordable, compact, and easily replicable, particularly for rural or off-grid users.

Challenges during development, such as ensuring stable serial communication and accurate current sensing, were overcome through careful system design and iterative testing. The result is a working model that not only validates the concept but also lays the foundation for future enhancements, including wireless communication, dynamic load prioritization based on available energy, and scaling to community-based solar microgrids.

In conclusion, this project exemplifies how blockchain technology can be meaningfully combined with renewable energy systems even at a small scale, promoting energy democratization, transparency, and user autonomy. It paves the way for further research into scalable, decentralized, and secure solar energy networks suited to diverse environments around the world.

CHAPTER SIX: REFERENCES

- [1] Zhang et al. (2020)
- [2] Andoni et al. (2019)
- [3] Mengelkamp, Esther, Benedikt Notheisen, Carolin Beer, David Dauer, and Christof Weinhardt. "A blockchain-based smart grid: towards sustainable local energy markets." *Computer Science-Research and Development* 33 (2018): 207-214.)
- [4] <https://international.austrade.gov.au/en/news-and-analysis/success-stories/powerledger-creates-the-worlds-first-new-energy-trading-platform>.
- [5] Muhammad Raisul Alam, Marc St-Hilaire, Thomas Kunz, Peer-to-peer energy trading among smart homes *Applied Energy*, Volume 238, 2019, Pages 1434-1443, ISSN 0306-2619, (<https://www.sciencedirect.com/science/article/pii/S0306261919300935>).
- [6] Parida, B., Iniyan, S., & Goic, R. (2011). A review of solar photovoltaic technologies. *Renewable and Sustainable Energy Reviews*, 15(3), 1625–1636. <https://doi.org/10.1016/j.rser.2010.11.032>.
- [7] www.dl.edi-info.ir was first indexed by Google in March 2016.
- [8] Sorrentino, M., Delp, G., & Kassem, S. (2020). Blockchain-based Smart Contracts for Renewable Energy Management: A Case Study. *Energy Technology*, 8(12), 2202-2211.
- [9] Zhao, Y., Zhang, H., & Xu, Z. (2018). Blockchain-based Peer-to-Peer Energy Trading System: A Survey. *Renewable and Sustainable Energy Reviews*, 88, 256-264.
- [10] Sorrentino, M., Delp, G., & Kassem, S. (2020). Blockchain-based Smart Contracts for Renewable Energy Management: A Case Study. *Energy Technology*, 8(12), 2202-2211.
- [11] Basu, R., Mishra, S., & Nanda, S. (2017). Blockchain in Energy Sector: A Systematic Review. *Energy Reports*, 3, 11-18.
- [12] Liu, Y., Li, H., & Ma, X. (2019). Blockchain-based Solar Energy Management System for Peer-to-Peer Trading. *Renewable and Sustainable Energy Reviews*, 98, 345-357.
- [13] Power Ledger. (2021). Power Ledger: Enabling a Decentralized Energy Future. Retrieved from Power Ledger.
- [14] Liu, Y., Li, H., & Ma, X. (2019). Blockchain-based Solar Energy Management System for Peer-to-Peer Trading. *Renewable and Sustainable Energy Reviews*, 98, 345-357.