Sapienza
Università di Roma

Dipartimento di Informatica

MSc in Cybersecurity

Energy Heaven

Blockchain and Distributed Ledger Technologies

Professor:
Claudio Di Ciccio

Students:
Andriani Simone
Marcuccio Francesco

Academic Year 2022/2023

Contents

1 Preface

1.1 Introduction: Energy Heaven

Energy Heaven is an idea born out of the desire to contribute to the green development of the planet. It is aimed at the exchange of renewable energy within energy communities, through the use of blockchain.

This project was realised as a team as the final project of the Blockchain and Distributed Ledger Technologies course of the MSc in Cybersecurity at 'La Sapienza' University of Rome.

1.2 Team and responsabilities

The team consists of the following members with their respective responsibilities during the development process:

- Andriani Simone: Project design, software architecture, token engineering, smart contract development, javascript development, report writing.
- Marcuccio Francesco: Project design, software architecture, smart contract development, middle-ware, server back-end and front-end development.

1.3 Project phases

The project implementation phases were as follows:

- Project conception and design;
- Infrastructure setup: setting up Ganache and Truffle;
- Software architecture: conceptual diagramming of the main components;
- Smart contract programming using Solidity;
- Middle-ware programming using javascript;
- Front-end development using html, css and javascript;
- Server development using php;
- Report writing.

2 Background

2.1 History

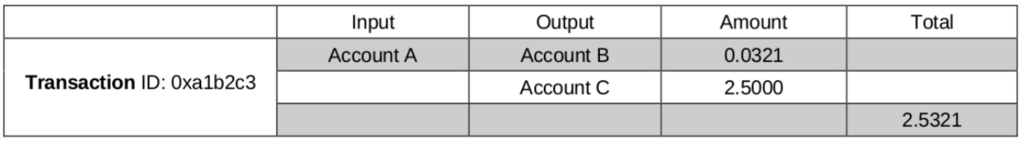
The history of Blockchain has its foundations back in the 1980s-1990s with the definition of the Anonymous e-cash protocols. In 1991, the idea of a signed information chain used as an electronic ledger was introduced to easily prove that none of the signed documents in the collection had been altered. The next steps were the Wei Dai's b-money (1998), to create money through solving computational puzzles and the Finney's Reusable Proof-of-Work (2004). In 2008 it was published an initial paper describing the Bitcoin electronic cash solution, by Satoshi Nakamoto.

2.2 Rationale

Today we can define blockchains as follows: Blockchains maintain a ledger and implement a specific kind of distributed ledger technology. A distributed ledger is an append-only store of transactions which is distributed across many machines. Being 'append-only' is important: new transactions can be added, but old transactions cannot be deleted or modified. A blockchain is a distributed ledger that is structured into a linked list of blocks. Each block contains an ordered set of transactions. Typical solutions use cryptographic hashes to secure the link from a block to its predecessor. [1]

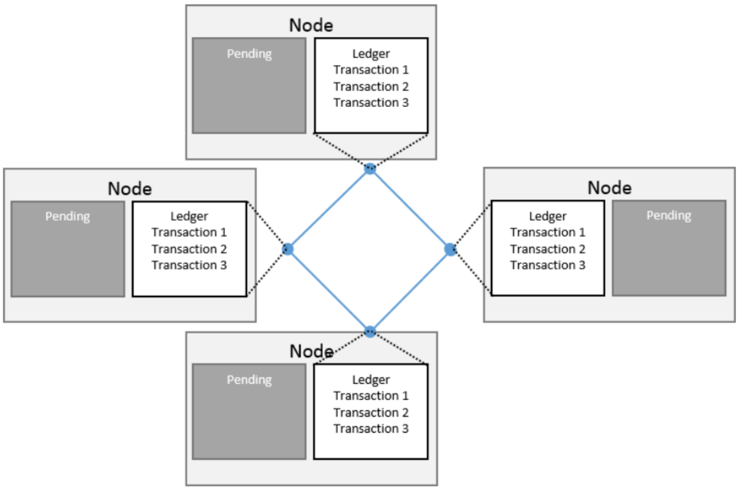
2.3 Concepts

Transaction A transaction is a recording of a transfer of assets among parties (accounts). Each transaction is identified by a transaction ID [2].



	Input	Output	Amount	Total
Transaction ID: 0xa1b2c3	Account A	Account B	0.0321	
		Account C	2.5000	
				2.5321

Distribuited Ledger A ledger is a collation of transactions that is not stored centrally but in each node of the network, for this reason it is ”distribuited”. Candidate ransactions are submited to the ledger, they are propagated to the other nodes in the network. The distributed transactions wait in a queue until they are added to the blockchain by a mining node [2].



Mining nodes Mining nodes are the subset of nodes that maintain the blockchain by publishing new blocks. Transaction are added to the blockchain when a mining node publishes a block. The mining node to trust in the publication of blocks is defined, for example, thanks to the Proof of Work model (solving a computationally intensive puzzle) or to the Proof of Stake model [2].

2.4 Application domain

Blockchains were first used for cryptocurrency but are now being used for many other purposes.

2.4.1 Enterprise and Industry

Data Management A blockchain can create a metadata layer for decentralized data sharing and analytics [1].

Supply Chain When tracking physical assets through changes in ownership and handling, key events and agreements can be recorded and communicated through data stored on a blockchain [1].

2.4.2 Financial Services

Digital Currency New forms of money can be implemented on blockchains, but these can also serve as a foundation for incentive models that support integrity for many blockchain systems. Blockchains allow digital currency to be transferred between parties, often without those transfers being processed or recorded by banks or payment services [1].

2.4.3 Government Services

Registries and Identity Including the identities and attributes of persons, companies, or devices, licensing, qualifications, and certifications. Storing registry entries or cryptographic certification of registry entries on a blockchain can facilitate access to and validation against the register. Blockchains could be used to share authenticated identifiers for individuals and companies, and these identifiers could in turn also enable many other blockchain applications [1].

Taxation Possible applications range from automated collection of tax using smart contracts to improved compliance by authoritative publication of taxation regulation and calculation tools as smart contracts on blockchain [1].

3 Context

3.1 Aim of the DApp and other details

Purpose: The purpose of the DApp 'Energy Heaven' is to offer energy communities a platform for buying and selling electricity from renewable sources by creating a self-supporting system that does not need third-party central entities.

No TTP (Trusted Third-Party): The absence of third-party central entities is achieved through the blockchain technology that is at the heart of the project. This results in the total elimination of costs related to the central entity, forgetting forever the concept of taxes, electricity bills and energy cost increases decided from the top.

Actors: Energy Heaven is a DApp for the communication of the different actors in an energy community for the purpose of energy exchange. The different actors are consumers and producers, specifying that a consumer can also become a producer.

Green Tokens: The medium of energy exchange are 'GreenTokens', an internal fungible token obtainable through an exchange from Ethereum units into tokens. GreenTokens can be spent to buy energy, this will result in a gain of GreenTokens for the sellers involved. GreenTokens can also be used to enable a consumer to become a producer. In addition, productivity rewards are periodically awarded proportionally to sellers based on their sales. At any time, any user can use GreenTokens for reverse exchange in order to regain units of Ethereum.

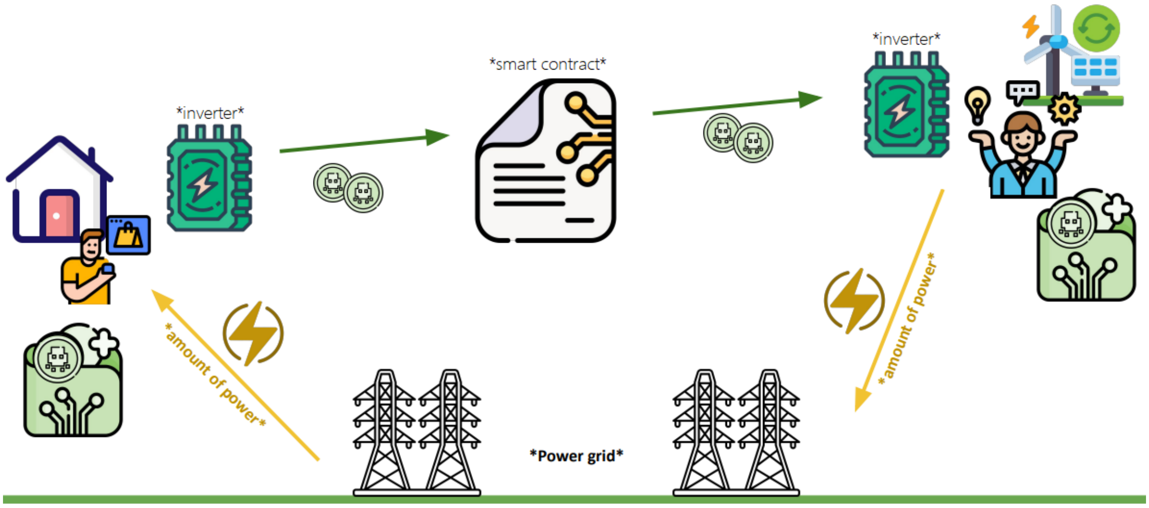
Inverters: It is essential to specify that the buying and selling of energy through the DApp can only take place through 'inverter' devices. These are special electricity meters that record the passage of energy (incoming in the case of purchase or outgoing in the case of sale) and communicate this to the DApp via the appropriate buy and sell functions. Without these devices, interaction is not possible (in our project, we emulated these devices through the use of manual buttons that abstraction the passage of energy).

Earnings: The gains are assured for all actors. Those who produce energy and sell it at the most competitive price manage to earn GreenTokens, even if there are other producers who have the same low price, because the buying is implemented in a split manner by all the most competitive sellers. This implementation ensures a system of self-regulating competition for which there can be no speculation on the price of energy since only the most competitive prices are chosen. Furthermore, the minimum price is chosen by the community's self-regulation based on the costs incurred by producers to produce energy. This is why consumers also gain from buying energy at extremely competitive and attractive prices.

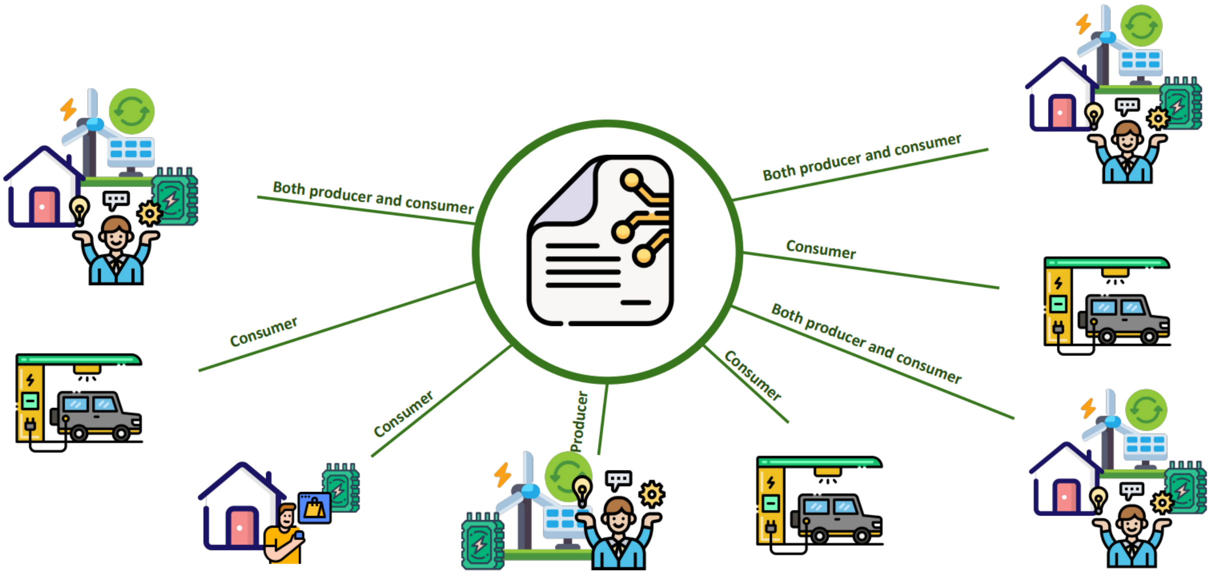
In addition, there is the periodic, proportional allocation of productivity bonuses to energy sellers.

Practically: In more detail Energy Heaven is a Smart Contract whose interaction can take place via the inverters, also thanks to a comfortable user interface in manual mode or automatically managed by the inverters.

Schema: The following is the schema of energy selling and buying:

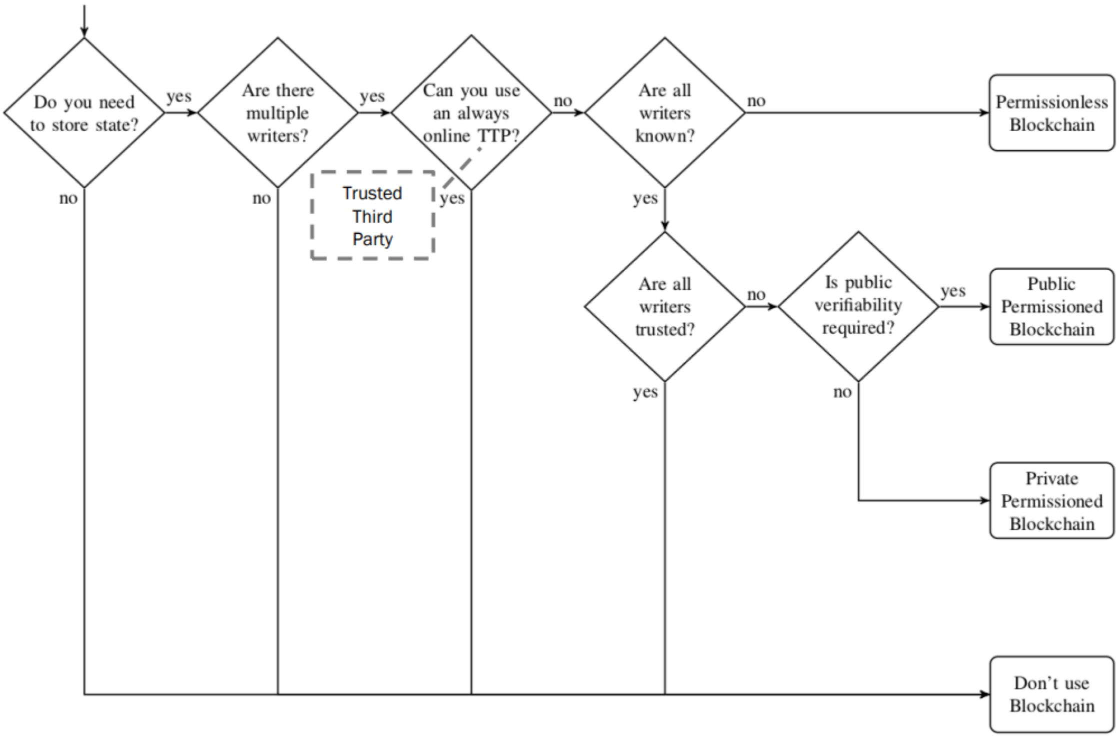


Network: The network may be complex:


Both producer and consumer
Both producer and consumer
Consumer
Consumer
Both producer and consumer
Consumer
Producer
Consumer

3.2 Why using a blockchain

We can answer this question by analysing the Wüst Gervais model.


Do you need to store state?
yes
no
Are there multiple writers?
yes
no
Can you use an always online TTP?
no
yes
Trusted Third Party
Are all writers known?
no
yes
Permissionless Blockchain
Are all writers trusted?
no
yes
Is public verifiability required?
yes
no
Public Permissioned Blockchain
Private Permissioned Blockchain
Don't use Blockchain

1 First we need a state, in fact all exchanges of energy and tokens must be registered for each user of the contract.

2 Certainly there are more writers, in fact every user who interacts with the smart contract is capable of causing a write in it. For example, acquiring tokens causes a write in the user's balance, similarly buying and selling energy causes a write in another component.

3 Using a TTP could be an alternative, but our system has its strength precisely in the absence of a TTP. Foregoing a TTP means having a system free of central impositions, without taxation and without higher costs.

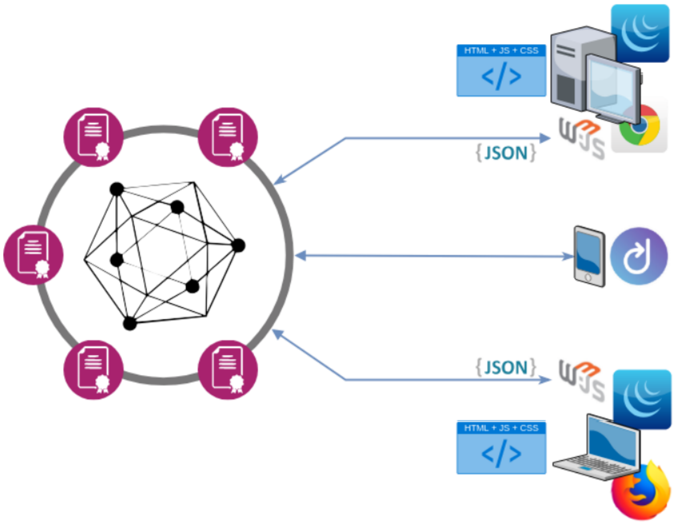
4 The users are not all known. Of course, anyone is free to make the identity associated with a certain address publicly known in the Blockchain, but this is not required.

3.3 Type of blockchain to use in production

Considering that not all users are known, considering that all users must be able to write in the state of the contract, considering that a Trusted Third Party always online would imply all the famous centralisation problems that the blockchain overcomes and higher costs for the system, the most suitable type of blockchain is Permissionless Blockchain.

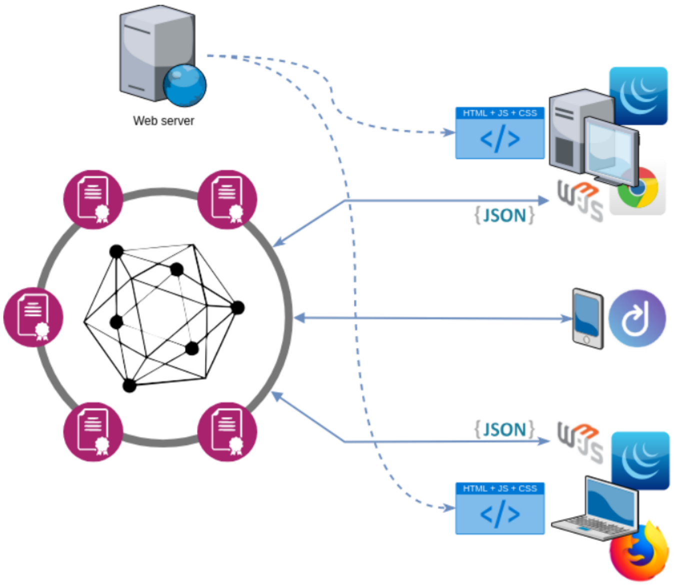
4 Architecture

The general architecture of a Web 3.0 distributed application is as follows:



At the centre is the blockchain, on its nodes the smart contract is executed. It is possible to interact with it thanks to web interfaces.

Certainly, in addition to the contracts running on blockchain, an important role is played by the servers used to record information that would otherwise burden the memory of the smart contract. In this project, too, we have realised something of this kind.



4.1 Main components

4.1.1 Front-end

The languages and technologies used on the front-end side are: Javascript, HTML and CSS. In particular, html was used to give an initial structure to the web page of the DApp interface. Subsequently, using javascript it was possible to give dynamism to the web page. Finally, CSS was used to refine the graphic appearance of the interface.

4.1.2 Middle-ware

The middle-ware is still represented by the aforementioned javascript script, in the .js file with which the html interface interacts. This file communicates with the smart contract functions via the Web3js API.

4.1.3 Back-end

A php server was used to handle the whole aspect of registering new users and logging in. All huge amounts of information, for example the information about the history of energy purchases and sales, is also stored here (so that the graphs of the interface can show them). This detailed information is useful for users but it was not a good choice to store it in the memory of the smart contract, so having a server was very useful.

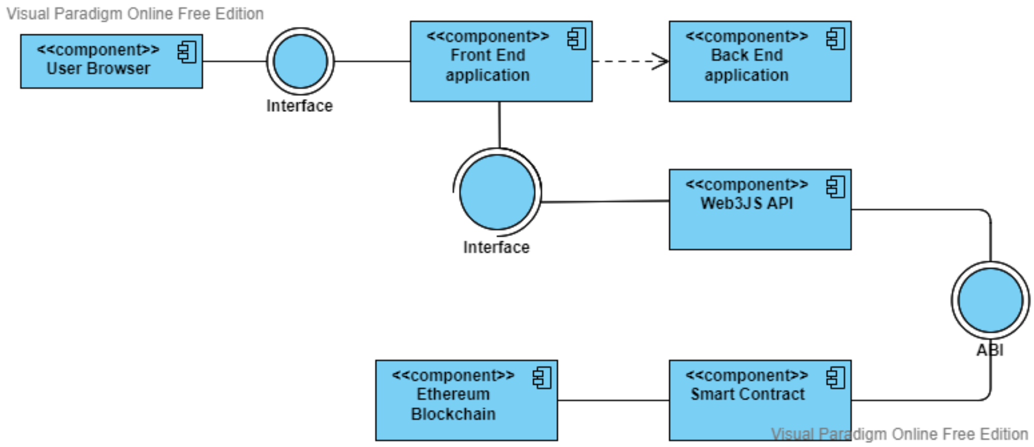
4.1.4 Smart Contract

The smart contract is the real heart of the DApp. It contains all the functions necessary for the life of the tokens, to manage the token balance of each user and to coordinate the buying and selling of energy. This smart contract is written to run on the Ethereum blockchain, but for obvious reasons has been deployed on ganache (which is as our own Ethereum blockchain).

4.2 Components diagram

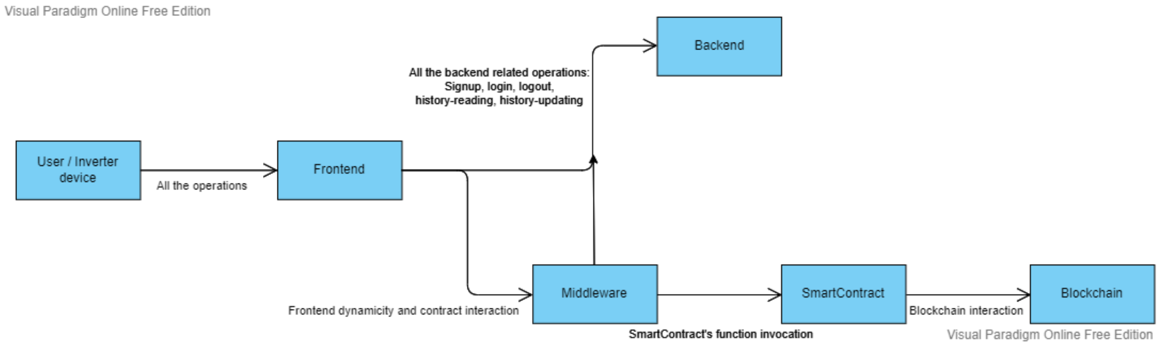
In the terminology of the UML modelling language, a component diagram is a diagram whose purpose is to represent the internal structure of the modelled software system in terms of its main components and the relationships between them. A component is understood to be a software unit with a precise identity, as well as well-defined responsibilities and interfaces [3].

In the following diagram, therefore, the components of our system are shown (here, the front-end was conceptually aggregated with the javascript file):


Visual Paradigm Online Free Edition
<<component>> User Browser
Interface
<<component>> Front End application
<<component>> Back End application
Interface
<<component>> Web3JS API
ABI
<<component>> Ethereum Blockchain
<<component>> Smart Contract
Visual Paradigm Online Free Edition

4.3 Collaboration among components

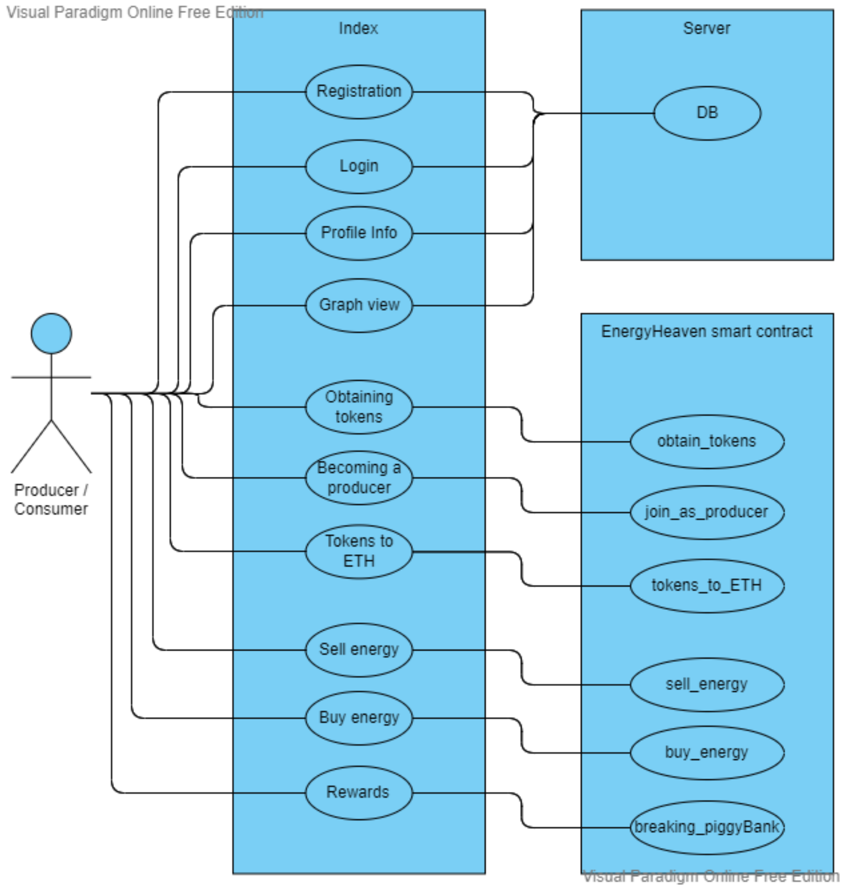
The components interact in the following way, first there is the web interface, it communicates with the javascript file and the server. The javascript file (middleware) communicates both with the server and, via Web3.js, with the contract. The latter runs on the blockchain.


Visual Paradigm Online Free Edition
Backend
All the backend related operations:
Signup, login, logout,
history-reading, history-updating
User / Inverter device
All the operations
Frontend
Frontend dynamicity and contract interaction
Middleware
SmartContract
SmartContract's function invocation
Blockchain
Blockchain interaction
Visual Paradigm Online Free Edition

4.4 Use cases diagram

Use Case Diagrams are diagrams dedicated to the description of the functions or services offered by a system, as perceived and used by the actors interacting with the system. They are mainly used in the context of the Use Case View of a model, in which case they can be considered as a tool for representing the functional requirements of a system. [4]

As can be seen in the following graph, all operations concerning the profile and the display of graphs are performed by interacting with the server, whereas the other interaction operations are performed, thanks to the previously explained javascript file, with the contract. Here, in the first block, the front-end has been conceptually aggregated with the javascript file.

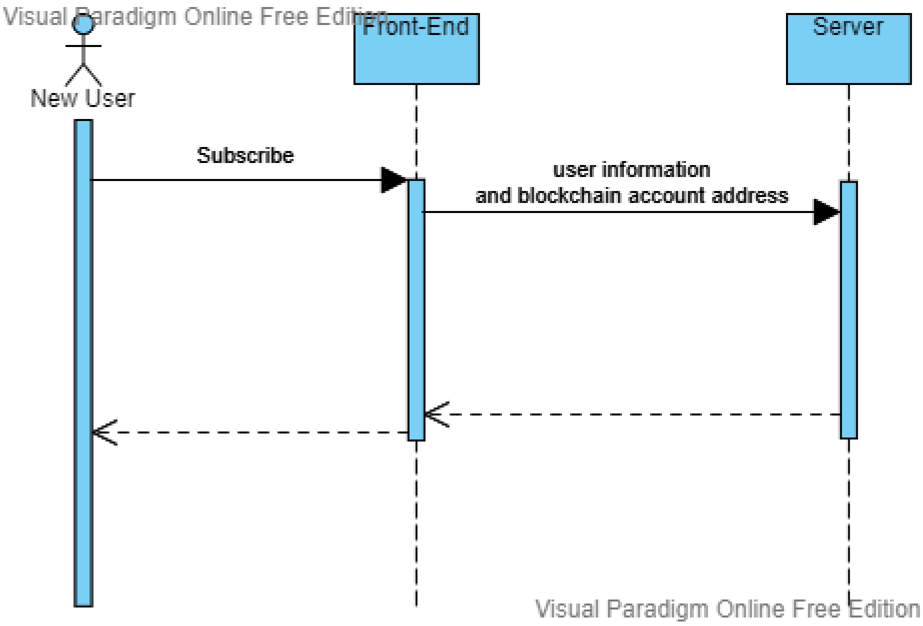


4.5 Sequence diagram

A Sequence Diagram is a diagram provided by the UML used to describe a scenario. The Sequence Diagram describes the relationships, in terms of messages, between Actors, Business Objects, Objects or Entities of the system being represented. A message is information that is exchanged between two entities. Usually the sender of the message, the active party, is the Actor. [5]

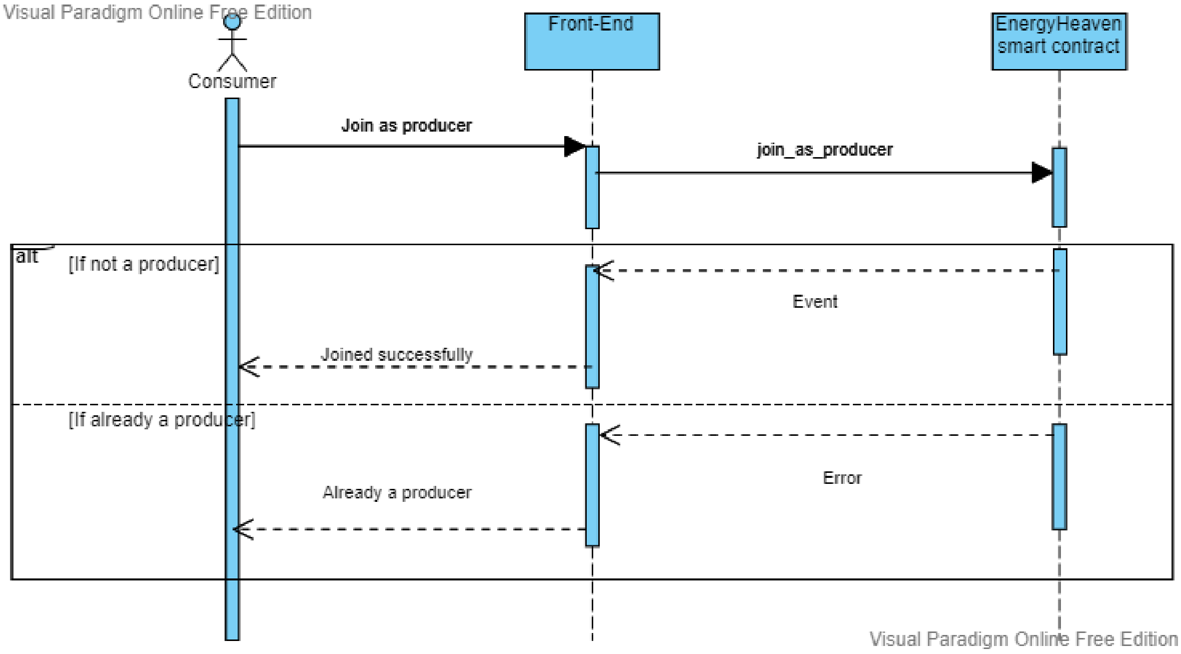
Note that the 'front-end' part includes both the actual html front-end and the js middle-ware. Instead, the server is understood as the files and functions relating to the back-end part.

Subscription


Visual Paradigm Online Free Edition
New User
Front-End
Server
Subscribe
user information
and blockchain account address
Visual Paradigm Online Free Edition

The registration of a new user in the system is done through the front-end interface, by indicating the user data. These data are transmitted to the server and recorded in order to guarantee subsequent access.

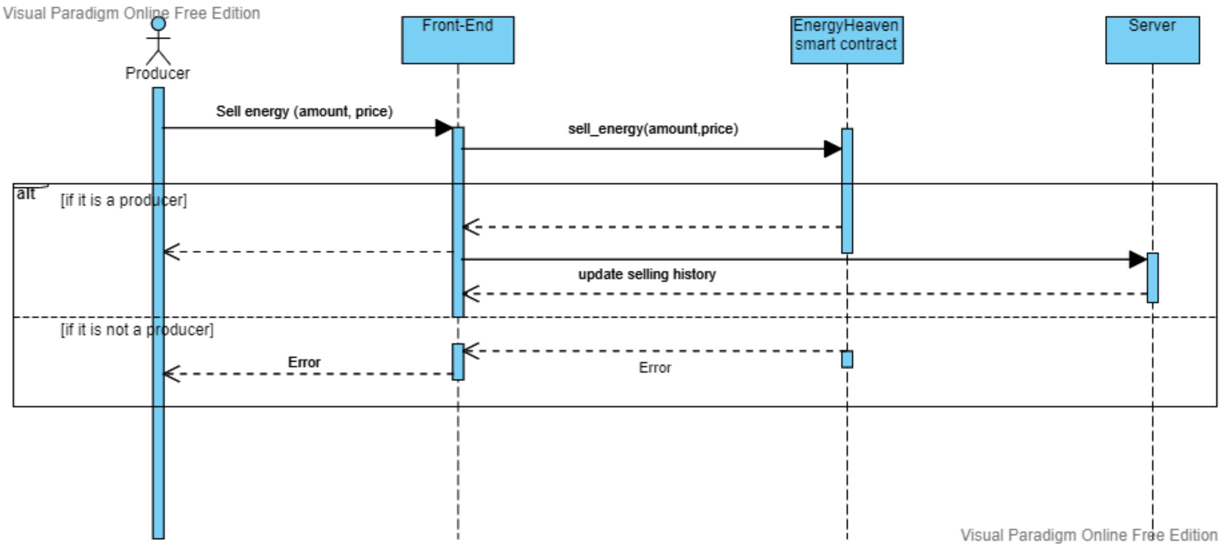
Becoming a producer



When a user wants to become a producer, thanks to the appropriate button in the front-end, a contract function is invoked which only performs the action if the user is not already a producer.

This involves the payment of some tokens that go into a piggy bank (we will see its purpose later).

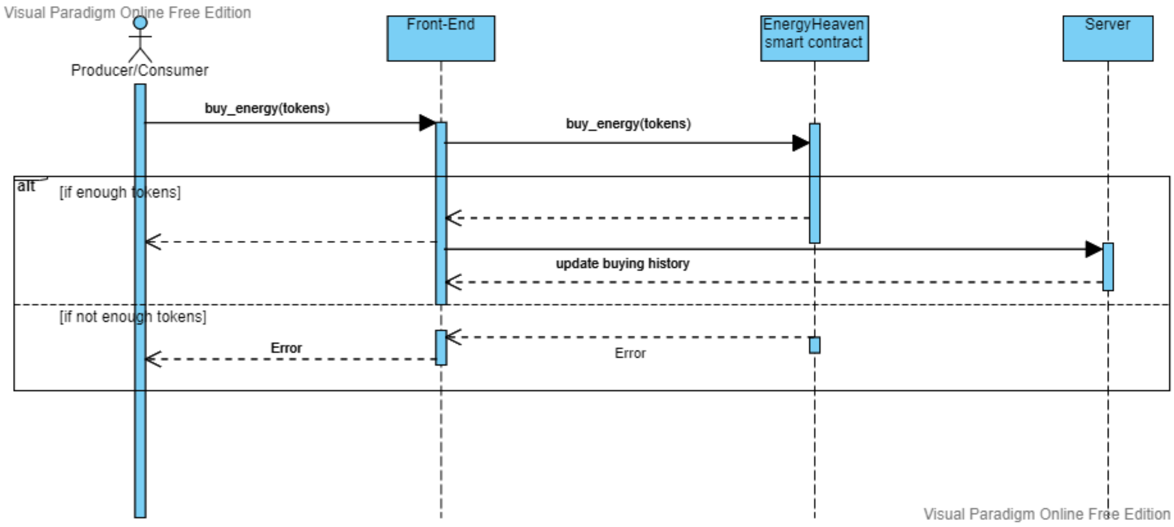
Sell energy



Visual Paradigm Online Free Edition
Producer
Front-End
EnergyHeaven smart contract
Server
Sell energy (amount, price)
sell_energy(amount,price)
alt
[if it is a producer]
update selling history
[if it is not a producer]
Error
Error
Visual Paradigm Online Free Edition

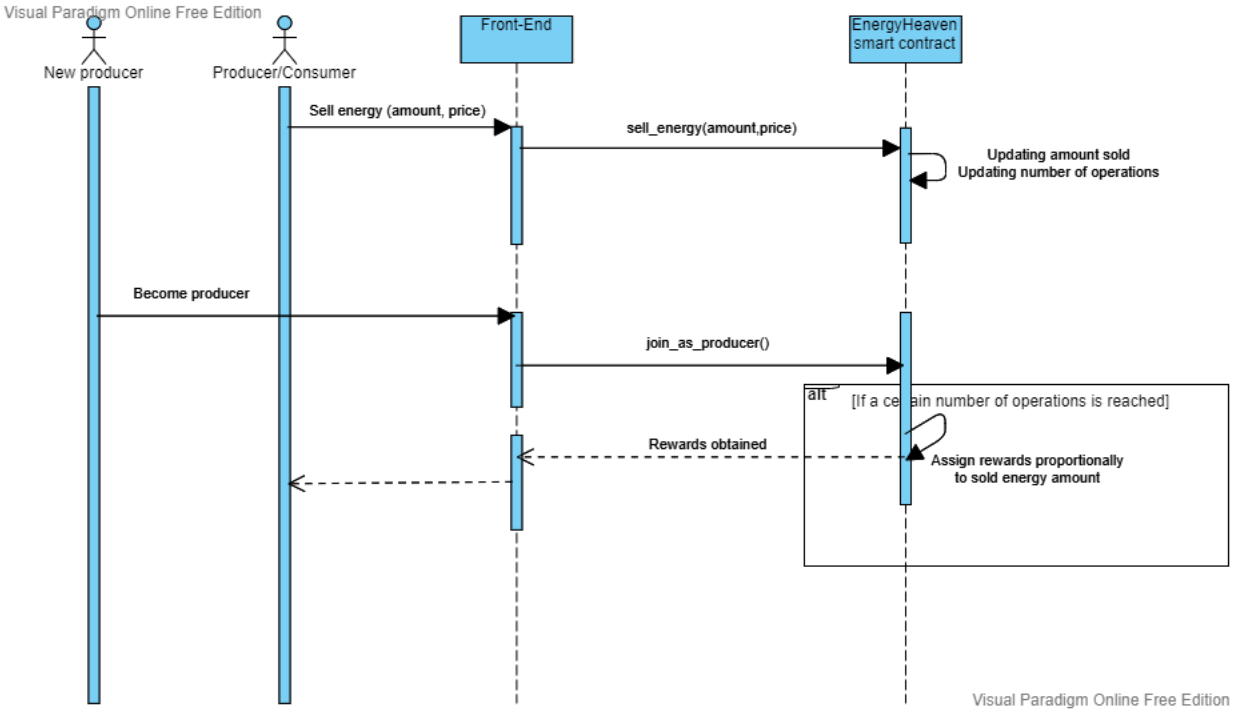
The sale of energy takes place, in actual use of the service, via the inverter. It is the inverter that automatically invokes the energy sale functions. This can only take place if the user is a producer. However, selling can also take place, and be set up, manually. It is always done through the front-end that invokes the appropriate function of the contract. In addition, in order to display graphs of the sales data of an individual user, each sales transaction is recorded in the server (note that the fact of recording this information in the server, and not in the contract, is designed to save memory in the contract).

Buy energy



Energy purchase takes place, in actual use of the service, via the inverter. it is the inverter that automatically invokes the energy purchase functions as required. This can only take place if the user has enough tokens. However, purchasing can also take place, and be set, manually. It is always done via the front-end that invokes the appropriate contract function. In addition, in order to display graphs of an individual user's purchase data, each purchase transaction is recorded in the server (note that the fact that this information is recorded in the server, and not in the contract, is designed to save memory in the contract).

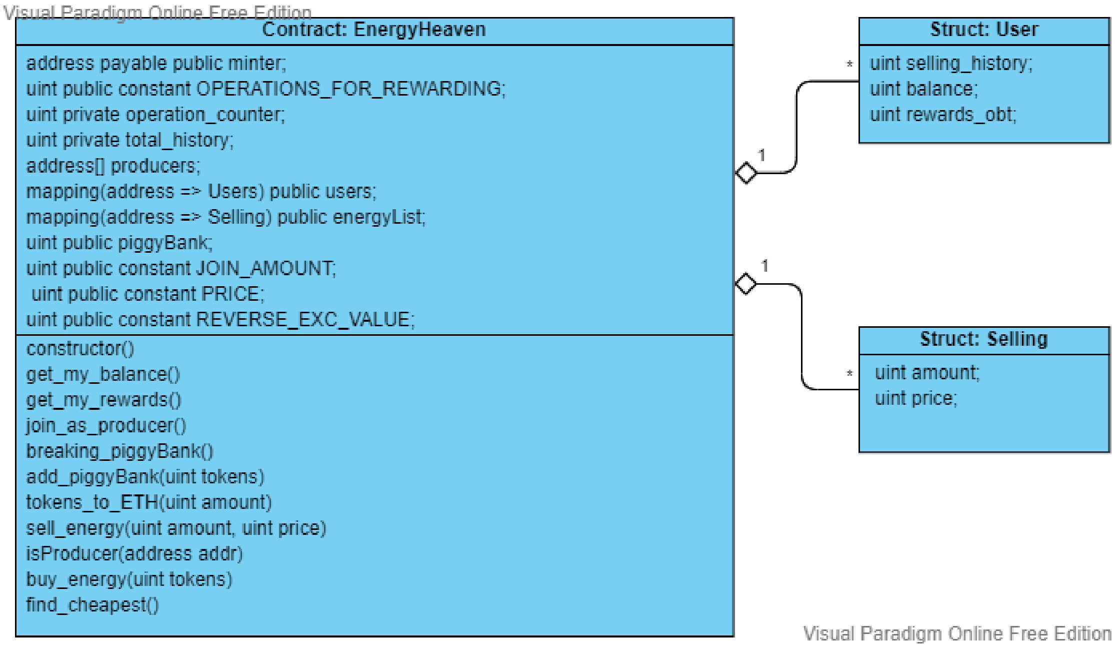
Obtain rewards


Visual Paradigm Online Free Edition
New producer
Producer/Consumer
Front-End
EnergyHeaven smart contract
Sell energy (amount, price)
sell_energy(amount,price)
Updating amount sold
Updating number of operations
Become producer
join_as_producer()
alt
[If a certain number of operations is reached]
Rewards obtained
Assign rewards proportionally to sold energy amount
Visual Paradigm Online Free Edition

Every amount of energy sold for each selling user is recorded and a counter is incremented for each sale transaction. Each time a user becomes a new producer the threshold of transactions for the allocation of rewards (also called the breaking of the piggy bank) is checked, if it has been reached then the amount of the piggy bank is allocated in proportion to the amount of energy sold. This is a productivity reward.

4.6 Concept diagram of the Smart Contract

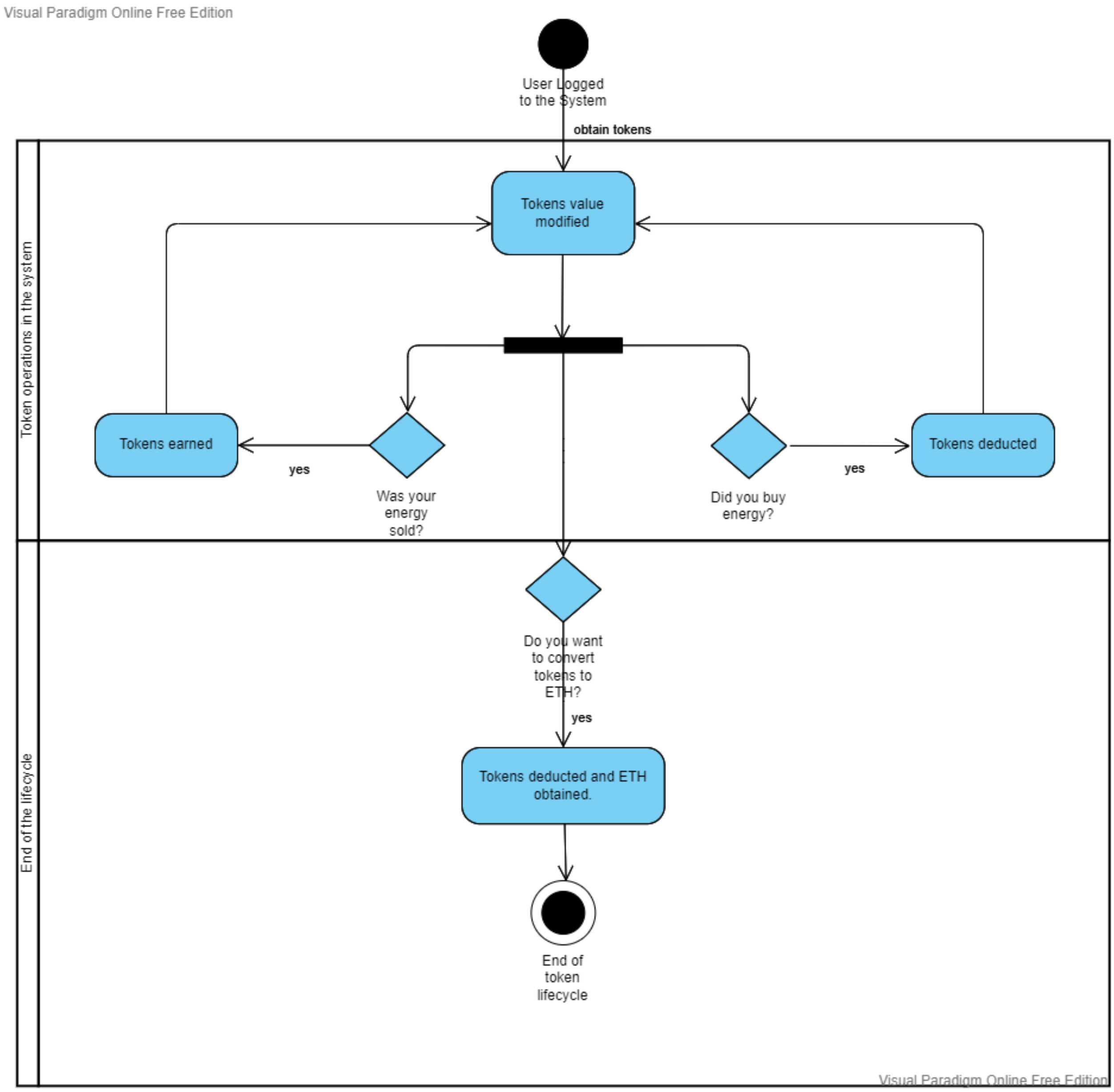
This diagram shows the contract, divided into its components: ’attributes’ and ’functions’. Struct specifications are used in the attributes, which are placed in an aggregation relationship.


Contract: EnergyHeaven
address payable public minter;
uint public constant OPERATIONS_FOR_REWARDING;
uint private operation_counter;
uint private total_history;
address[] producers;
mapping(address => Users) public users;
mapping(address => Selling) public energyList;
uint public piggyBank;
uint public constant JOIN_AMOUNT;
uint public constant PRICE;
uint public constant REVERSE_EXC_VALUE;
constructor()
get_my_balance()
get_my_rewards()
join_as_producer()
breaking_piggyBank()
add_piggyBank(uint tokens)
tokens_to_ETH(uint amount)
sell_energy(uint amount, uint price)
isProducer(address addr)
buy_energy(uint tokens)
find_cheapest()
1
*
Struct: User
uint selling_history;
uint balance;
uint rewards_obt;
1
*
Struct: Selling
uint amount;
uint price;

4.7 Activity diagram for the tokens' lifecycle

Here we can observe the life cycle of tokens. First, they are obtained by exchanging ETHs (subunits of them). This causes the value of the tokens to change (in the user's balance sheet), then according to the transactions performed, this value continues to change. Energy sold increases its token count, energy bought decreases it.

When one chooses to convert a token back into ETH, then its life cycle ends.

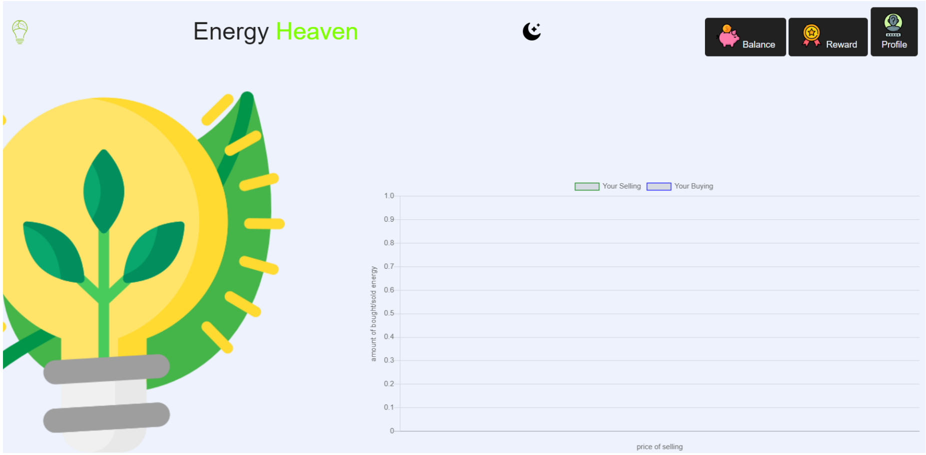

Visual Paradigm Online Free Edition
User Logged to the System
obtain tokens
Token operations in the system
Tokens value modified
Tokens earned
yes
Was your energy sold?
Did you buy energy?
yes
Tokens deducted
End of the lifecycle
Do you want to convert tokens to ETH?
yes
Tokens deducted and ETH obtained.
End of token lifecycle
Visual Paradigm Online Free Edition

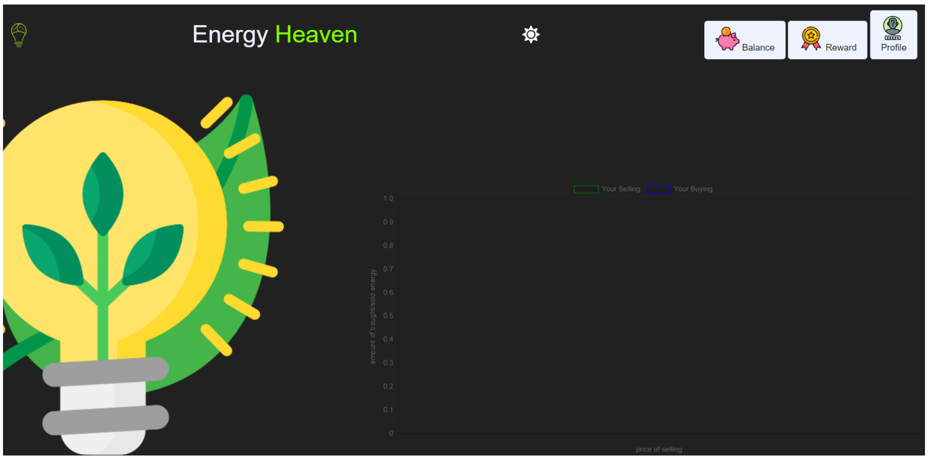
5 Implementation and Demo

5.1 Implementation

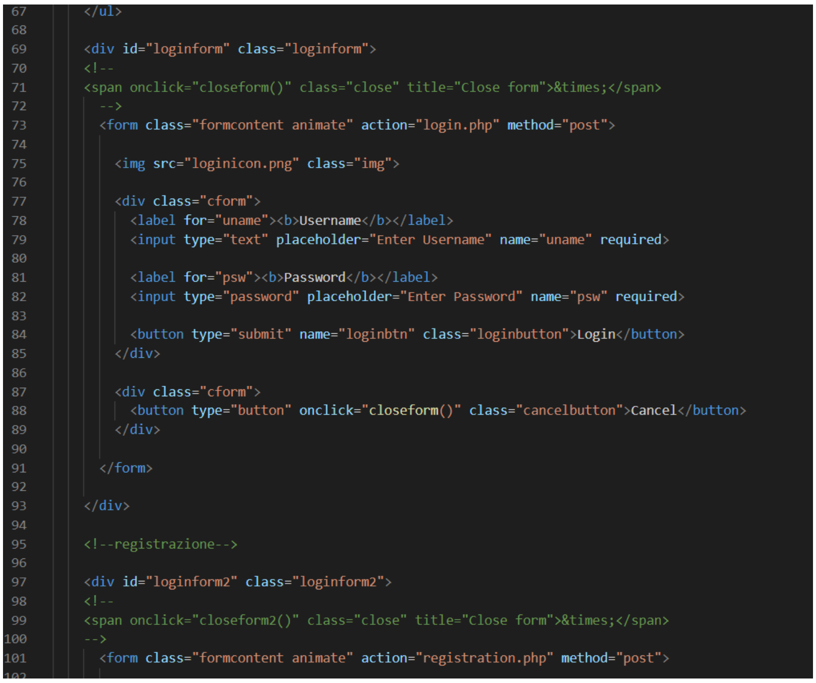
5.1.1 Front-end

The DApp interface consists of a single, simple, usable and user-friendly page. The front-end was implemented with a file that contains the html part related to the graphical interface and the interaction with the various php and javascript functions.



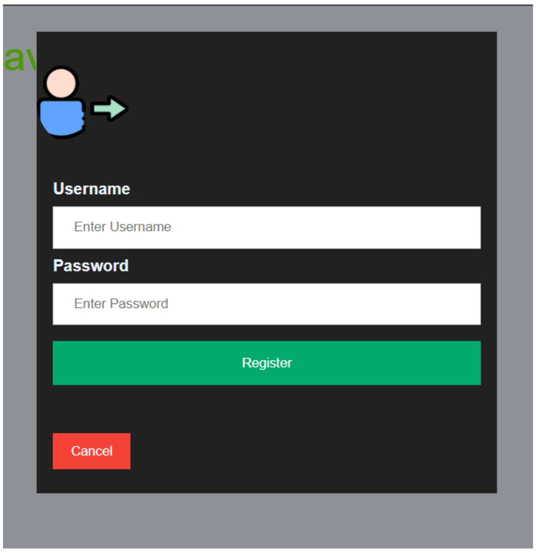


The following is an illustrative code fragment:

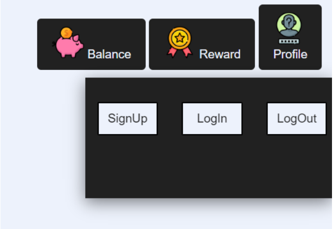


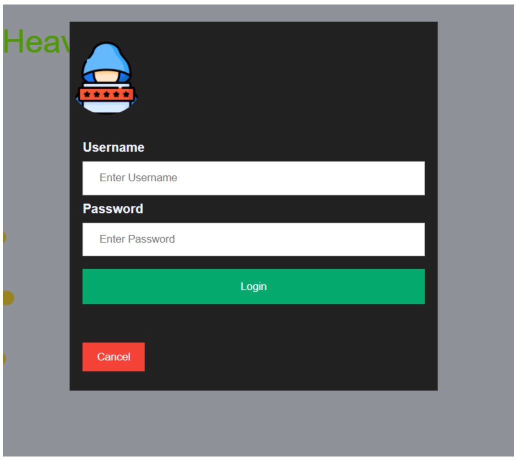
```
    </ul>

    <div id="loginform" class="loginform">
    <!--
    <span onclick="closeform()" class="close" title="Close form">&times;</span>
      -->
      <form class="formcontent animate" action="login.php" method="post">

        <img src="loginicon.png" class="img">

        <div class="cform">
          <label for="uname"><b>Username</b></label>
          <input type="text" placeholder="Enter Username" name="uname" required>

          <label for="psw"><b>Password</b></label>
          <input type="password" placeholder="Enter Password" name="psw" required>

          <button type="submit" name="loginbtn" class="loginbutton">Login</button>
        </div>

        <div class="cform">
          <button type="button" onclick="closeform()" class="cancelbutton">Cancel</button>
        </div>

      </form>

    </div>

    <!--registrazione-->

    <div id="loginform2" class="loginform2">
    <!--
    <span onclick="closeform2()" class="close" title="Close form">&times;</span>
    -->
      <form class="formcontent animate" action="registration.php" method="post">
```

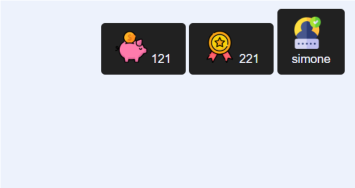
SignUp Within the interface it is possible to register a new user via the appropriate button.



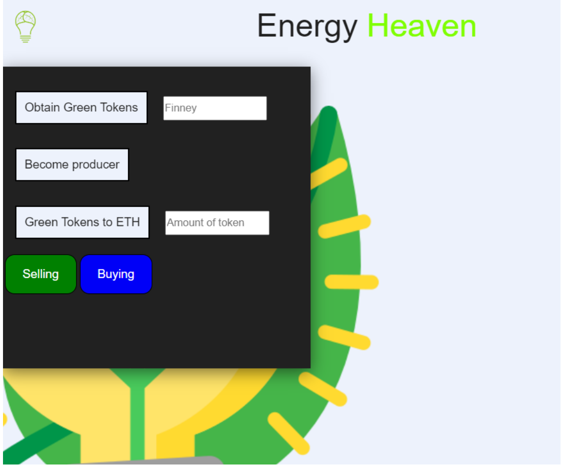
Login and Logout It is of course possible to log in and log out.

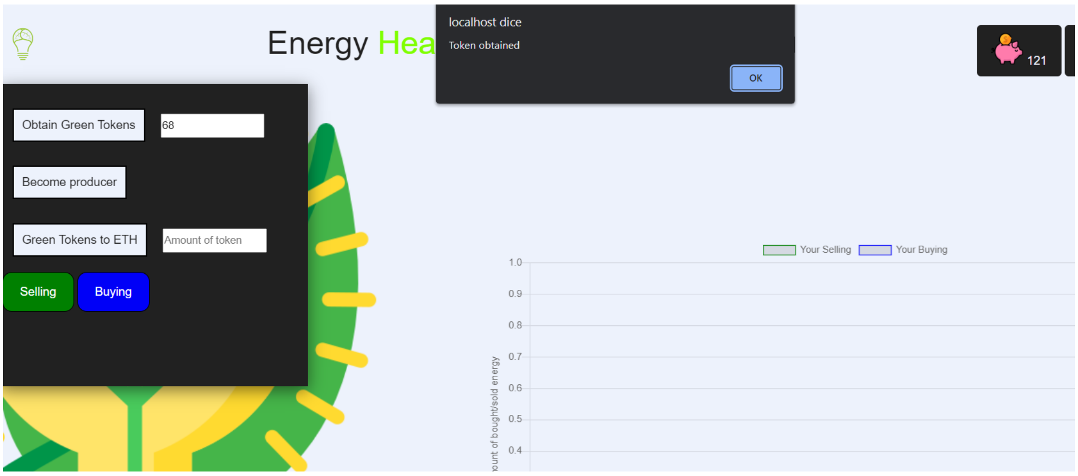






Contract interaction Interaction with the contract is done through the side window that allows you to obtain tokens, exchange tokens to ETH, become a producer, sell and buy energy. It must be specified that in the real application of the system, the buying and selling of electricity is automated by the 'inverter' device, the user can however manually set the selling price. Since we do not have such a device, and cannot practically apply our system for real, we have emulated the inverter through the use of manual buttons that abstract the passage of energy.





localhost dice

Token obtained

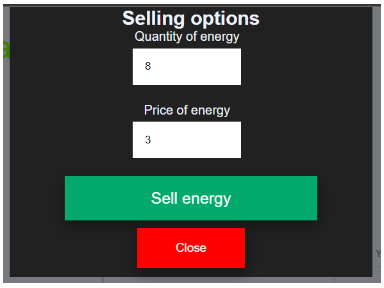
OK

Obtain Green Tokens: 68

Become producer

Green Tokens to ETH: Amount of token

Selling Buying



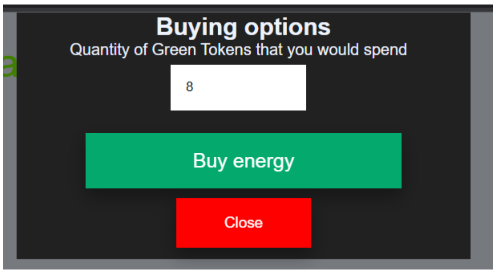
Selling options

Quantity of energy: 8

Price of energy: 3

Sell energy

Close



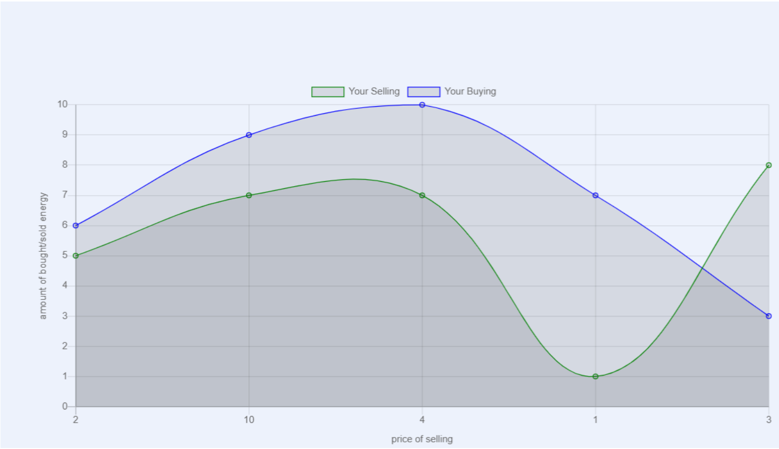
Buying options

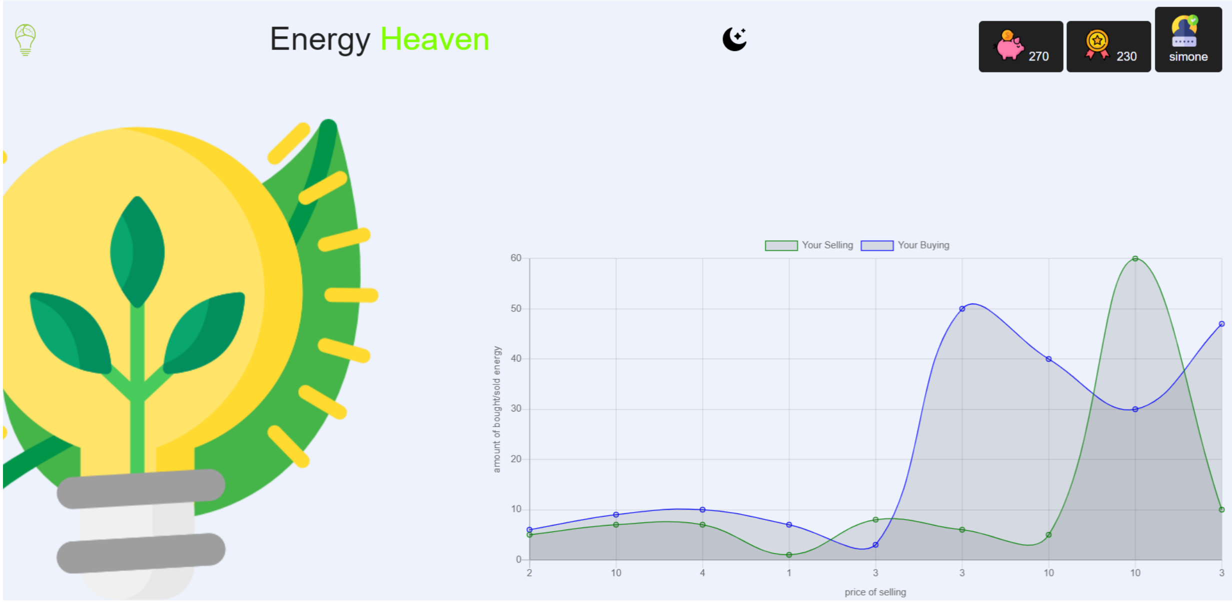
Quantity of Green Tokens that you would spend: 8

Buy energy

Close

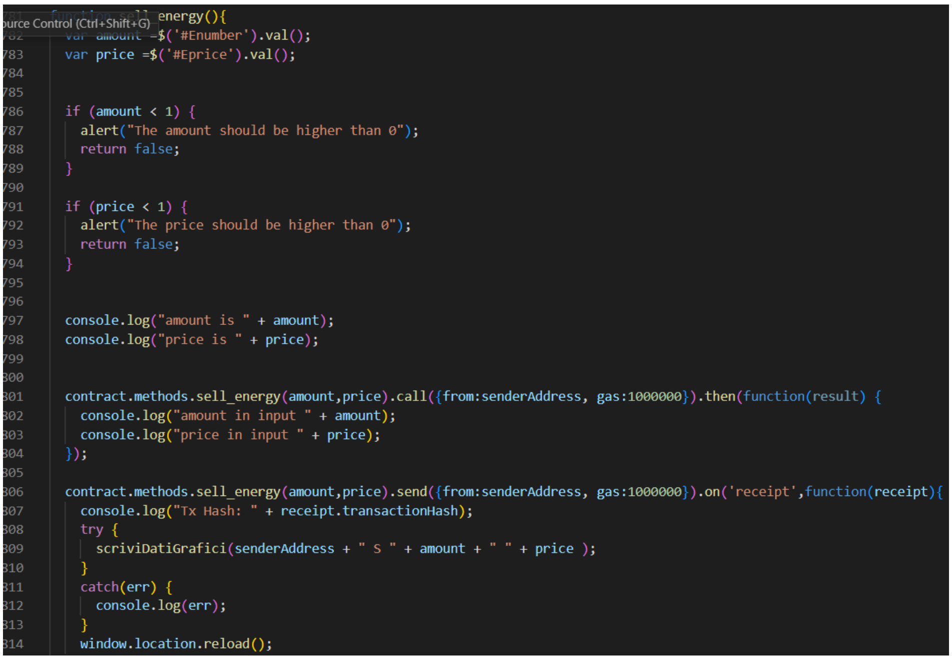
Charts Through the comfortable graphs you can see your sales and purchase history.





5.1.2 Middle-ware

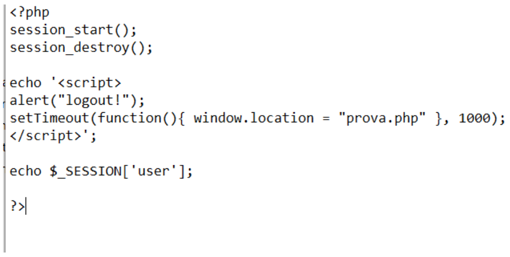
The file with the .js extension allows the interface to be dynamic and interact with the contract via the Web3.js API. It handles the display and updating of graphics. The following is an illustrative code fragment:



```
function sell_energy(){
  var amount =$('#Enumber').val();
  var price =$('#Eprice').val();


  if (amount < 1) {
    alert("The amount should be higher than 0");
    return false;
  }

  if (price < 1) {
    alert("The price should be higher than 0");
    return false;
  }


  console.log("amount is " + amount);
  console.log("price is " + price);


  contract.methods.sell_energy(amount,price).call({from:senderAddress, gas:1000000}).then(function(result) {
    console.log("amount in input " + amount);
    console.log("price in input " + price);
  });

  contract.methods.sell_energy(amount,price).send({from:senderAddress, gas:1000000}).on('receipt',function(receipt){
    console.log("Tx Hash: " + receipt.transactionHash);
    try {
      scriviDatiGrafici(senderAddress + " S " + amount + " " + price );
    }
    catch(err) {
      console.log(err);
    }
    window.location.reload();
```

5.1.3 Back-end

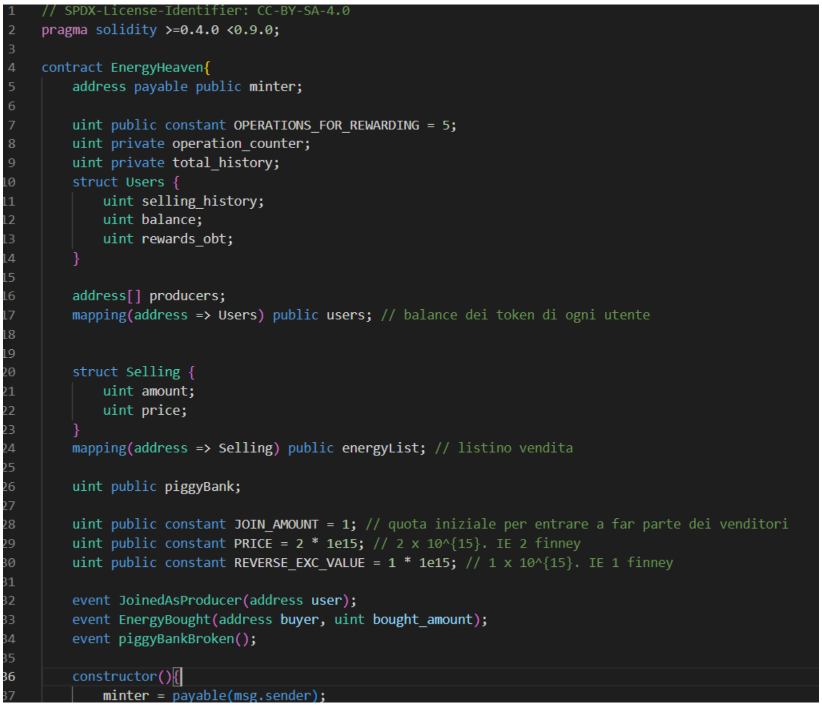
The php functions allow us to manage login, logout, the registration of new users and the saving of information to be displayed in graphs on the server. For this reason we use xampp to run the application. The following is an illustrative code fragment:



```
<?php
session_start();
session_destroy();

echo '<script>
alert("logout!");
setTimeout(function(){ window.location = "prova.php" }, 1000);
</script>';

echo $_SESSION['user'];

?>
```

5.1.4 Smart Contract

The smart contract is the heart of the system, written in Solidity, it implements all the functions that manipulate its state and interact with the blockchain. It is important to specify that the constant values assigned and used in the contract are illustrative. The following is an illustrative code fragment:



```
// SPDX-License-Identifier: CC-BY-SA-4.0
pragma solidity >=0.4.0 <0.9.0;

contract EnergyHeaven{
    address payable public minter;

    uint public constant OPERATIONS_FOR_REWARDING = 5;
    uint private operation_counter;
    uint private total_history;
    struct Users {
        uint selling_history;
        uint balance;
        uint rewards_obt;
    }

    address[] producers;
    mapping(address => Users) public users; // balance dei token di ogni utente


    struct Selling {
        uint amount;
        uint price;
    }
    mapping(address => Selling) public energyList; // listino vendita

    uint public piggyBank;

    uint public constant JOIN_AMOUNT = 1; // quota iniziale per entrare a far parte dei venditori
    uint public constant PRICE = 2 * 1e15; // 2 x 10^{15}. IE 2 finney
    uint public constant REVERSE_EXC_VALUE = 1 * 1e15; // 1 x 10^{15}. IE 1 finney

    event JoinedAsProducer(address user);
    event EnergyBought(address buyer, uint bought_amount);
    event piggyBankBroken();

    constructor(){
        minter = payable(msg.sender);
```

6 Conclusions

What we have just discussed is a distributed application that can concretely help the environment and citizens' pockets. Indeed, it incentivises and encourages energy communities based on the self-production and exchange of clean and renewable energy. Moreover, the fact that it is blockchain-based, without the need for a TTP, totally negates any problems related to taxes, top-down price increases and speculation.

6.1 Known issues and limitations

There are some small problems that can be solved by funding in this area, and refining the project.

Firstly, the fact that it is necessary to have special devices that combine electricity management (production and consumption) with interaction with the smart contract running on the blockchain. Funding in this direction could lead to the large-scale production of similar devices, with low prices for their purchase.

Secondly, there is the risk that these devices could be tampered with, to operate in a byzantine manner, damaging our system. In this case, the solution could be, for example, to create strong integrity challenges that guarantee that the devices will not be tampered with.

6.2 Future works

Our project is a starting point for increasingly flexible and functional systems.

For example, currently energy is sold by default by sellers who sell it at the lowest price, equally, creating a highly competitive but fair market (which has the merit of incentivising lower prices and the convenience of the system). This can be extended with the possibility of choosing the seller from whom someone wants to buy energy entirely, perhaps because he is a friend of his, or because he prefers him to another (due to circumstances in the physical world) and wants to buy energy entirely from him.

Another extension of the project, depending on the type of physical device used, could be the production of charging stations for electric cars such that they can interact with the blockchain via the driver's profile.

Similarly, in any case where there may be a trade in electricity our system can be used. All you need is an inverter device that interacts with the smart contract running on the blockchain.

References

[1] Xiwei Xu - Ingo Weber - Mark Staples, Architecture for Blockchain Applications, 2019.

[2] Claudio Di Ciccio, Bitcoin – behind the scenes and overview, 2022.

[3] Wikipedia, Component Diagram, 2015.

[4] Wikipedia, Use Case Diagram, 2022.

[5] Wikipedia, Sequence Diagram, 2022.