UNIVERSITY OF PATRAS
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

DIVISION OF TELECOMMUNICATIONS & INFORMATION TECHNOLOGY
LABORATORY OF WIRED TELECOMMUNICATIONS

«Designing and implementing Local Energy Market application using Blockchain technologies»

D I P L O M A   T H E S I S

KONSTANTINOS SEKARAS

SUPERVISOR: DENAZIS SPYRIDON

PATRAS – MARCH, 2023

CERTIFICATION

It is certified that the Diploma Thesis titled

Designing and implementing Local Energy Market application using Blockchain technologies

of the Department of Electrical and Computer Engineering student

KONSTANTINOS SEKARAS

Registration Number: 1046888

was presented publicly at the Department of Electrical and Computer Engineering at

03/06/2023

and was examined by the following examining committee:

Denazis Spyridon, Professor, ECE (supervisor)

Dimitrios Lymberopoulos, Professor, ECE (committee member)

Alexios Birbas, Professor, ECE (committee member)

The Supervisor	The Director of the Division
Denazis Spyridon Professor	Michael Logothetis Professor

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΕΡΓΑΣΤΗΡΙΟ ΕΝΣΥΡΜΑΤΗΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΣ

Σχεδιασμός και υλοποίηση Local Energy Market εφαρμογής με τη χρήση Blockchain τεχνολογίας

Δ Ι Π Λ Ω Μ Α Τ Ι Κ Η   Ε Ρ Γ Α Σ Ι Α

Σεκαρά Κωνσταντίνου του Γεωργίου

Αριθμός μητρώου: 1046888

ΕΠΙΒΛΕΠΩΝ: Δενάζης Σπυρίδων

ΠΑΤΡΑ - ΜΑΡΤΙΟΣ 2023

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η Διπλωματική Εργασία με τίτλο

Σχεδιασμός και υλοποίηση Local Energy Market εφαρμογής με τη χρήση Blockchain τεχνολογίας

του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

ΚΩΝΣΤΑΝΤΙΝΟΥ ΣΕΚΑΡΑ ΤΟΥ **ΓΕΩΡΓΙΟΥ**

Αριθμός Μητρώου: 1046888

Παρουσιάστηκε δημόσια στο Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών στις

06/03/2023

και εξετάστηκε από την ακόλουθη εξεταστική επιτροπή:

Δενάζης Σπυρίδων, Καθηγητής, ΗΜΤΥ (επιβλέπων)

Λυμπερόπουλος Δημήτριος, Καθηγητής, ΗΜΤΥ (μέλος επιτροπής)

Μπίρμπας Αλέξιος, Καθηγητής, ΗΜΤΥ (μέλος επιτροπής)

Ο Επιβλέπων	Ο Διευθυντής του Τομέα
Δενάζης Σπυρίδων Καθηγητής	Λογοθέτης Μιχαήλ Καθηγητής

ACKNOWLEDGMENTS

I would like to thank the supervisor Professor Denazis Spyridon and misters Georgakakos Georgios and Andriopoulos Nicholaos for their cooperation, confidence, patience and valuable contribution to the completion of this Diploma Thesis.

ABSTRACT

Designing and implementing Local Energy Market application using Blockchain technologies

STUDENT: KONSTANTINOS SEKARAS   SUPERVISOR: SPYRIDON DENAZIS

As the energy demand is skyrocketing worldwide and the climate change effects are becoming harsher, there is a need for a revitalisation of the energy production and distribution system, especially if we take into account the costly aspect of electricity storage and the relatively unpredictable behaviour of renewable energy sources. The Local Energy Market (LEM) is a concept that promises to tackle most current energy system issues, by outlining the operation of a microgrid, connected to a main grid, and describing a feasible way of connecting and integrating to a smaller scale network, multiple distributed energy sources. This idea is the outcome of a global tendency that demands systems to be more interconnected and operate in a more trustless and decentralized manner. Another new technology, which has emerged recently and follows this pattern, is the blockchain one.

Blockchain was first introduced when the Bitcoin network was deployed in 2009 and presented the world with a new, closed and trustless system, in which financial transactions could be performed in a direct, private and secure way. The Ethereum blockchain took this innovation one step further and created an environment, in which applications of every kind can be published and members of the Ethereum network can interact with them, in total safety, without relying on third parties. These applications are called Decentralized applications – Dapps – and in order to create them, developers write programs called smart contracts. This thesis showcases how one could develop a Dapp that would implement all necessary activities for the operation of a LEM.

The Dapp is run on the Ganache-CLI client which fires up a personal Ethereum blockchain, so we can test our smart contracts locally. We also use the ethereum-bridge, a private network version of the Provable oracle. Oracles are entities that help users in the system to communicate with the world beyond the strictly defined boundaries of the blockchain and gain access to data and services unavailable in it. In our case, the oracle is needed to give us the exchange rate between US dollars and ETH, Ethereum's native cryptocurrency, as well as providing us with a necessary tool for scheduling the execution of one of our contract's functions. Finally, we use the Truffle framework, an advanced smart contract developing environment that is accompanied with all tools necessary for writing and deploying smart contracts to blockchains.

After deploying our contracts, we can interact with them typing JavaScript commands in Truffle's console. Using contract instances, along with the accounts that Ganache created and we have available, we can make a simulation of the conditions and activities that take place in an LEM operating without administrative authorities and estimate the outcome of deploying our contracts in the main Ethereum network.

ΠΕΡΙΛΗΨΗ

Σχεδιασμός και υλοποίηση Local Energy Market εφαρμογής με τη χρήση Blockchain τεχνολογίας

ΦΟΙΤΗΤΗΣ: ΣΕΚΑΡΑΣ ΚΩΝΣΤΑΝΤΙΝΟΣ ΕΠΙΒΛΕΠΩΝ: ΔΕΝΑΖΗΣ ΣΠΥΡΙΔΩΝ

Οι ενεργειακές απαιτήσεις σε παγκόσμιο επίπεδο αυξάνονται συνεχώς, όπως επίσης και οι προσπάθειες κυρίως ανεπτυγμένων κρατών, για κάλυψη των αναγκών τους μέσω ανανεώσιμων πηγών ενέργειας. Η κατανάλωση συμβατικών καυσίμων δεν αποτελεί πλέον μία επιθυμητή προοπτική, λόγω των πεπερασμένων αποθεμάτων που καθιστούν απαγορευτική την μακροχρόνια εξάρτηση από αυτά, αλλά και επειδή η συγκέντρωση τους σε ορισμένα γεωγραφικά σημεία δίνει τη δυνατότητα σε συγκεκριμένες χώρες να αποκτήσουν το ρόλο του «ελεγκτή» της ενεργειακής ροής. Η ανομοιογενής κατανομή των ενεργειακών θυλάκων ανά την υφήλιο, χωρίζει τα κράτη σε προμηθευτές και καταναλωτές και είναι πλέον φυσιολογικό πως τα κράτη που δεν διαθέτουν μεγάλες ποσότητες ορυκτών καυσίμων στο υπέδαφός τους, επιδιώκουν την ενεργειακή ανεξαρτητοποίηση τους. Αν προσθέσουμε στα παραπάνω και τις ολοένα εντονότερες και εμφανέστερες επιπτώσεις της κλιματικής αλλαγής, γίνονται προφανείς οι αιτίες που οδηγούν τον ανεπτυγμένο κόσμο να πραγματοποιεί εσπευσμένες επενδύσεις σε «πράσινη» ενέργεια. Η αύξηση της εκμετάλλευσης ανανεώσιμων πηγών όμως απαιτεί και την ενσωμάτωση ενός μεγάλου αριθμού μονάδων παραγωγής, όπως φωτοβολταϊκών συστημάτων για ηλιακή ενέργεια, οι οποίες μονάδες θα είναι κατανεμημένες σε ολή την έκταση που καλύπτει το ηλεκτρικό δίκτυο, προκειμένου να αυξήσουν την αδράνειά του. Όπως είναι αναμενόμενο, αυτές οι εξελίξεις θα φέρουν αλλαγές στον τρόπο λειτουργίας των δικτύων διανομής ενέργειας και θα προκύψουν νέες προκλήσεις σχετικά με την διαχείριση των ενεργειακών πόρων. Τα δίκτυα διανομής θα πρέπει πλέον να επιχειρούν με περισσότερους αυτοματισμούς, δίχως την συνεχή επιτήρηση ενός κεντρικού επόπτη, ενώ θα απαιτείται επίσης η δυνατότητα επικοινωνίας με άλλα δίκτυα και η συνεργασία με αυτά. Η απαίτηση για διαλειτουργικότητα μεταξύ διαφόρων συστημάτων, αλλά κυρίως για αυτοματοποιημένη και αποκεντρωμένη διαχείρισή τους, είναι ένα γενικό φαινόμενο της εποχής μας και μία ακόμη τεχνολογία που έχει αναπτυχθεί τα τελευταία χρόνια και ακολουθεί αυτήν την τάση, είναι αυτή των blockchain.

Το blockchain σαν ιδέα αναπτύθηκε παράλληλα με την δημιουργία του κρυπτονομίσματος Bitcoin. Το Bitcoin έγινε γνωστό ως ένα ψηφιακό νόμισμα, ωστόσο το ίδιο όνομα χρησιμοποιείται και για το δίκτυο χρηστών στα πλαίσια του οποίου γίνεται η χρήση του. Σ' αυτό το σύστημα, τα μέλη μπορούν να πραγματοποιούν συναλλαγές άμεσα και με ασφάλεια, χωρίς να χρειάζεται η οποιαδήποτε παρέμβαση ή εποπτεία από κάποιον κεντρικό διαχειριστή, κάτι που καθιστά τις αναφερθείσες συναλλαγές απόλυτα ιδιωτικές. Αυτή η καινοτομία αποτέλεσε την έμπνευση για το πως θα μπορούσε να δημιουργηθεί ένα αποκεντρωμένο δίκτυο στο οποίο οι χρήστες θα μπορούν να υλοποιούν αλληλεπιδράσεις οποιασδήποτε μορφής, μέσα στα πλαίσια ορισμένων προκαθορισμένων κανόνων.

Το θεμέλιο πάνω στο οποίο στηρίζεται αυτό το οικοδόμημα είναι η διαμοίραση των δεδομένων. Κάθε ενέργεια εντός του δικτύου πρέπει να είναι ορατή από κάθε μέλος του

δικτύου – στο οποίο μπορούμε να αναφερθούμε και ως κόμβο – και όλοι πρέπει να αναγνωρίζουν πως τα δεδομένα αυτών είναι πανομοιότυπα. Αυτό το είδος συστήματος αποκαλείται Distributed Ledger – ή DLT – καθώς μπορεί να θεωρηθεί ως μία κοινόχρηστη λογιστική συλλογή καταγεγραμμένων συναλλαγών. Τα DLTs λοιπόν είναι ουσιαστικά βάσεις δεδομένων οι οποίες διαφοροποιούνται ανάλογα με τον τρόπο που ελέγχονται και ακολούθως καταγράφονται τα δεδομένα. Υπάρχει η περίπτωση στην οποία μόνο συγκεκριμένοι κόμβοι του δικτύου, που έχουν λάβει την απαιτούμενη άδεια, μπορούν να πραγματοποιήσουν έλεγχο και η περίπτωση που όλοι οι κόμβοι συμμετέχουν στη διαδικασία επαλήθευσης της εγκυρότητας των δεδομένων. Η δεύτερη περίπτωση πρόκειται και για το είδος DLT που αποτελεί το Bitcoin, ένα αποκεντρωμένο δικτύο δηλαδή στο οποίο δεν μπορεί κάποια κεντρική αρχή να ελέγξει την εισαγωγή πληροφοριών, το οποίο είναι και το είδος συστήματος που θα μας απασχολήσει.

Σε ένα Blockchain όπως το Bitcoin τα δεδομένα από έναν αριθμό συναλλαγών ομαδοποιούνται σε blocks και ελέγχονται από τους κόμβους μέσα από την διαδικασία της εξόρυξης δεδομένων ή αλλιώς mining. Τα μέλη του δικτύου που πραγματοποιούν το mining λέγονται miners και σε ένα αποκεντρωμένο και ανοιχτό blockchain, ο οποιοσδήποτε κόμβος μπορεί να παίξει αυτό το ρόλο. Ο λόγος για τον οποίο οι miners μπαίνουν στην διαδικασία της εξακρίβωσης της εγκυρότητας των συναλλαγών, είναι το αντίτιμο στο συνάλλαγμα του δίκτυου - ή αλλιώς κρυπτονόμισμα - που αποδίδεται στους πρώτους miners που θα ελέγξουν μία συναλλαγή, όπο με τον όρο συναλλαγή αναφερόμαστε σε ένα οποιοδήποτε είδος αλληλεπίδρασης μεταξύ δύο κόμβων. Το μέγεθος του αντιτίμου καθορίζεται σύμφωνα με το νόμο προσφοράς και ζήτησης, ενώ τα μέλη που επιθυμούν οι συναλλαγές τους να εγκριθούν ταχύτερα μπορούν να προσφέρουν ένα μεγαλύτερο ποσό για να προσφέρουν επιπλέον κίνητρα στους miners και με αυτόν τον τρόπο να θέσουν τη συναλλαγή τους σε υψηλότερη προτεραιότητα. Ο τρόπος που γίνεται το mining, διαφέρει ανάλογα με το πρωτόκολλο λειτουργίας του κάθε blockchain αλλά γενκά η συνεργασία μεταξύ των miners είναι προς το συμφέρον τους και γι’ αυτό το λόγο συνδυάζουν τις δυνατότητες τους στα λεγόμενα mining pools.

Προϋπόθεση για να γίνει κατανοητή η μέθοδος ελέγχου των συναλλαγών, είναι και η εξήγηση της έννοιας του fork. Εφόσον οι miners – ή mining pools - εργάζονται ξεχωριστά μπορεί να επαληθεύουν τις ίδιες συναλλαγές και να τις τοποθετούν σε διαφορετικά blocks τα οποία τοποθετούνται στην αλυσίδα. Έτσι, δημιουργούνται συνεχώς πολλές μικρές αλυσίδες οι οποίες «ξεπηδούν» από ένα block και θυμίζουν υπό μία έννοια τα δόντια ενός πιρουνιού. Πάντα όμως μόνο μια αλυσίδα θα καταφέρνει να επιβιώσει, όταν επιτευχθεί μια τελική συναίνεση μεταξύ των μελών και ο τρόπος που ξεχωρίζει αυτή η μία, καθορίζεται από τα πρωτόκολλα του κάθε blockchain. Στα forks λοιπόν οι περισσότερες μικρές αλυσίδες είναι βραχύβιες, εκτός από τα hard forks στα οποία μία αλυσίδα που φέρνει κάποια αλλαγή στο σύστημα επιβιώνει παράλληλα με την προηγούμενη κύρια αλυσίδα ή και ακόμη την αντικαθιστά εντελώς.

Το πιο γνωστό πρωτόκολλο συναίνεσης (consensus protocol) και η βάση λειτουργίας του Bitcoin, είναι το Proof of Work (PoW). Η κάθε συναλλαγή αποτελείται από ιδιωτικά και δημόσια δεδομένα τα οποία ενοποιούνται και μετέπειτα κρυπτογραφούνται μέσω μιας hash function. Η hash function παράγει ένα μοναδικό αποτέλεσμα, το οποίο είναι επίσης ορατό από όλους και με βάση αυτό και τα δημόσια δεδομένα κάθε miner προσπαθεί να μαντέψει τα ιδιωτικά δεδομένα και να επαληθεύσει την εγκυρότητα της συναλλαγής. Αυτό γίνεται με την προσθήκη τυχαίων δεδομένων στα γνωστά δημόσια δεδομένα και την μετέπειτα επεξεργασία του συνόλου μέσα από μια hash function. Στη συνέχεια τα δεδομένα τροποποιούνται ελαφρά και η διαδικασία αυτή επαναλαμβάνεται έως ότου τα αποτελέσματα να είναι ταυτόσημα με τα αρχικά παραχθέντα. Όπως γίνεται αντιληπτό αυτή είναι μια μέθοδος που απαιτεί αρκετή υπολογιστική ισχύ και είναι επομένως πολύ ενεργοβόρα, ειδικά αν λάβουμε υπόψη και τον παράγοντα του ανταγωνισμού. Επειδή όλοι οι miners προσπαθούν να ελέγξουν τις ίδιες συναλλαγές το πιο πιθανό είναι πως θα συμφωνήσουν στο ποια αλυσίδα είναι η σωστή, κάτι που σημαίνει πως αν κάποιος ήθελε να εγκρίνει κάποια «κακόβουλη» συναλλαγή θα έπρεπε να αποκτήσει τον έλεγχο της πλειοψηφίας των πόρων του συστήματος, κάτι που θα ήταν προφανώς φοβερά δαπανηρό. Πέρα από το Proof of Work, υπάρχουν και πολλά άλλα πρωτόκολλα, αλλά ένα ακόμη ειδικά που αξίζει να αναφερθεί είναι το Proof of Stake (PoS). Σ' αυτό το πρωτόκολλο, κάθε κόμβος μπορεί να πραγματοποιήσει mining, με τις ενέργειές του να έχουν βαρύτητα ανάλογη του ποσού που έχει στο συνάλλαγμα τού εκάστοτε blockchain. Έτσι καταλήγει να πραγματοποιεί mining μόνο για τον αριθμό συναλλαγών που αντιπροσωπεύει το ποσό στον λογαριασμό του και με αυτόν τον τρόπο εξασφαλίζεται πως δεν θα προχωρήσει σε κακόβουλες ενέργειες οι οποίες θα υποβάθμιζαν την αξία των καταθέσεών του. Το Proof of Stake λοιπόν παρέχει μια πολύ πιο οικονομική και κομψή λύση που παρακάμπτει το ενεργοβόρο “arms race” του PoW.

Το blockchain που μας απασχολεί κυρίως σε αυτήν την εργασία είναι το Ethereum, το οποίο διαθέτει το δικό του κρυπτονόμισμα το οποίο καλείται Ether (ETH). Το Ethereum έχει την ιδιαιτερότητα ότι δίνει τη δυνατότητα στα μέλη του δικτύου του να πραγματοποιήσουν οποιουδήποτε είδους δραστηριότητα, σε αντίθεση πχ με το Bitcoin στο οποίο μπορούν μόνο να συναλλάσσονται οικονομικά. Αυτό γίνεται γιατί τα μέλη του έχουν πρόσβαση σε κομμάτια κώδικα, τα οποία μπορούν να ορίσουν σύνθετες αλληλεπιδράσεις μεταξύ κόμβων, τα οποία λέγονται smart contracts. Τα smart contracts σαν έννοια προϋπήρχαν του Ethereum, αλλά με την εμφάνιση των blockchain εμφανίστηκαν νέες ιδέες για τον τρόπο αξιοποίησης τους. Πρόκειται για προγράμματα τα οποία είναι αποθηκευμένα στο ledger και εκτελούνται όταν ικανοποιηθεί κάποια συνθήκη. Ο ρόλος τους είναι να εκτελούν αυτομάτως αιτιοκρατικές «συμφωνίες» μεταξύ κόμβων, με το αποτέλεσμα της κάθε εκτέλεσης να εξαρτάται από την παραμετροποίηση ορισμένων μεταβλητών. Χρησιμοποιώντας τα smart contracts, οι προγραμματιστές αναπτύσσουν ορισμένες εφαρμογές, ορατές από όλα τα μέλη του δικτύου, που λόγω του αποκεντρωμένου χαρακτήρα της διαχείρισής και πρόσβασης αυτών λέγονται Decentralized Apps (Dapps). Έτσι το Ethereum αποτελεί ένα σύστημα που αποτελείται από το σύνολο των Dapps και των smart contracts, στο οποίο σύστημα ένας κόμβος μπορεί θεωρητικά να πραγματοποιήσει οποιαδήποτε ενέργεια θα ήταν δυνατή και στο Διαδίκτυο και γι' αυτό το λόγο, το δίκτυο των εφαρμογών αυτών λέγεται web3, μία αντίθεση στην έννοια του web2 που είναι ο γνωστός Παγκόσμιος Ιστός.

Παρά τη σύνθετη φύση των smart contracts πάντως, ο τρόπος που αποθηκεύονται τα δεδομένα στο Ethereum είναι παρόμοιος με αυτόν του Bitcoin, ενώ και ο έλεγχος των δραστηριοτήτων στο blockchain γίνεται μέσω mining που υπακούει στο Proof of Work πρωτόκολλο. Η ανάπτυξη εφαρμογών στο Ethereum και η εκτέλεσή αυτών από άλλους κόμβους απαιτεί έναν έλεγο εγκυρότητας, του οποίου το κόστος καθορίζεται από ένα μέγεθος που λέγεται gas. Το gas είναι ένα νομισματικό μέγεθος το οποίο είναι ανάλογο της υπολογιστικής ισχύος που απαιτείται για την εκτέλεση κάποιας λειτουργίας ενός smart contract ή και για το σύνολο των λειτουργιών του, όταν έχουμε την περίπτωση της αρχικής ανάπτυξης στο blockchain. Το gas έχει άμεση σχέση με το αντίτιμο που αποδίδεται για τον έλεγχο των συναλλαγών και συνεπώς η αξία της μονάδας του καθορίζεται από τη ζήτηση και την δραστηριότητα των miners.

Παρόλο που το Ethereum έχει καινοτομήσει στο χώρο των blockchain, όπως και το Bitcoin πιο πριν, συνεχίζει να εξελίσσεται. Τον τελευταίο καιρό ετοιμάζεται η μετάβασή του σε μια νέα εκδοχή του που θα αποκαλείται Ethereum 2.0, με την μεγάλη αλλαγή να είναι η υιοθέτηση του Proof of Stake πρωτοκόλλου. Αυτή όμως η μεγάλη αλλαγή δεν θα εκφραστεί με κάποιο απλό fork, αλλά με την μετάβαση σε ένα ολόκληρο σύστημα από αλυσίδες που θα υπακούουν στο PoS και θα μοιράζονται το φόρτο αποθήκευσης και επεξεργασίας δεδομένων. Το πρώτο βήμα που είναι η δημιουργία μίας αλυσίδας που υπακούει στο PoS, έχει ήδη γίνει και αναμένεται η φάση στην οποία θα ενωθεί με την κύρια αλυσίδα, ώστε να φέρει την αλλαγή στο πρωτόκολλο, δίχως να χαθεί το ιστορικό των προηγούμενων συναλλαγών.

Η τάση λοιπόν προς χρήσιν πιο αποκεντρωμένων συστημάτων με περισσότερους αυτοματισμούς, επηρεάζει και τον τρόπο συναλλαγής ενέργειας. Στα σύγχρονα ενεργειακά συστήματα υπάρχει η ανάγκη για την άμεση επικοινωνία μεταξύ καταναλωτών και παραγωγών, ειδικά αν ληφθεί υπ όψιν πως οι διαχωριστικές γραμμές αυτών γίνονται ολοένα και πιο δυσδιάκριτες. Ολοένα και περισσότερα μέλη του δικτύου δεν έχουν απλώς ένα παθητικό ρόλο καταναλωτή, αλλά συνεισφέρουν στη συνολική παραγωγή του συστήματος αξιοποιώντας Ανανεώσιμες Πηγές Ενέργειας όπως η ηλιακή, με την χρήση φωτοβολταϊκών πάνελ. Η ένταξη επιπρόσθετων πόρων ενέργειας κατανεμημένων σε όλη την έκταση του συστήματος (Distributed Energy Resources – DERs), συνιστά μια διαφοροποίηση από την παραδοσιακή δομή του συστήματος παραγωγής και διανομής ενέργειας, με τις μεγάλες παραγωγικές μονάδες οι οποίες καλύπτουν τη ζήτηση σχεδόν σε όλη την έκταση του δικτύου, σε μία πιο αποκεντρωμένη. Αυτή η αλλαγή βοηθά στην αύξηση του ποσοστού της ενέργειας από ΑΠΕ στη συνολική προσφορά, αλλά και στην πιο ομοιόμορφη κατανομή των ενεργειακών πόρων στο δίκτυο, κάτι που συνεπάγεται μεγαλύτερη ισορροπία και αδράνεια του συστήματος.

Το κυριότερο ζήτημα που καλούνται να αντιμετωπίσουν τα συστήματα διανομής ενέργειας, είναι η άμεση μεταφορά αυτής από την παραγωγή προς την κατανάλωση. Η αποθήκευση της ενέργειας είναι μια ιδιαίτερα απαιτητική διαδικασία για την οποία χρειάζονται να δαπανηθούν μεγάλα χρηματικά ποσά και λοιποί πόροι. Ο στόχος λοιπόν είναι να υπάρχει όσο το δυνατόν πιο ακριβής πρόβλεψη για την μελλοντική ζήτηση, έτσι ώστε να καλύπτεται ακριβώς από την παραγωγή. Αυτή η επιδιωκόμενη ισορροπία είναι πιο εύκολο να επιτευχθεί με την παρουσία πολλών διαφορετικών ενεργειακών πορών που λειτουργούν αυτόνομα και έχουν μια πιο προβλέψιμη συμπεριφορά σε μεγάλη κλίμακα, απ' ότι με την αποκλειστική χρήση μεγάλων ηλεκτροπαραγωγικών μονάδων η οποία συνεπάγεται πως η απόδοση του συστήματος είναι άμεσα εξαρτώμενη από την απόδοση αυτών. Η επιτυχημένη ένταξη των παραγωγών-καταναλωτών απαιτεί την ύπαρξη ενός έξυπνου δικτύου (smart grid), το οποίο παρακολουθεί τις ενέργειές τους και καθορίζει αυτόματα τις τιμές στις οποίες θα πωλείται η ενέργεια. Αυτές οι τιμές μπορεί να προκύψουν μέσα από διάφορους μηχανισμούς αγοράς ενέργειας (Energy Markets) οι οποίοι διαμορφώνουν τα κόστη, δίνοντας έμφαση σε διαφορετικά χαρακτηριστικά του εκάστοτε συστήματος, ανάλογα με την περίπτωση.

Η προαναφερθείσα παγκόσμια τάση προς την επίτευξη διασυνδεσιμότητας μεταξύ διαφόρων συστημάτων δεν αφήνει ανεπηρέαστο ούτε τον τομέα των ενεργειακών δικτύων, καθώς διαμορφώνεται το υπόβαθρο για τον δημιουργία συμπλεγμάτων που αποτελούνται από μικρότερα συστήματα τα οποία λειτουργούν με μια σχετική αυτονομία. Το μεγαλύτερο προτέρημα της λειτουργίας μικρότερης κλίμακας δικτύων είναι η δυνατότητα για ομαλότερη ένταξη των DERs στην συνολική παραγωγή, καθώς ένα τέτοιο δίκτυο αναπτύσσεται διαρκώς με μοναδικό στόχο την αύξηση της παραγωγικότητας του η οποία μπορεί να γίνει μόνο με την συμμετοχή πολλών μικρότερων παραγωγών οι οποίοι εκμεταλλεύονται ΑΠΕ. Οι παραγωγοί αυτοί με τη σειρά τους επιλέγουν την ένταξη στο τοπικό δίκτυο καθώς η επιδιωκόμενη αυτονομία, μίας για παράδειγμα μικρής επαρχιακής πόλης, μπορεί να θωρακίσει την τοπική ενεργειακή κοινότητα (Local Energy Community – LEC) από απότομες μεταβολές των τιμών ρεύματος, ενώ ο κάθε καταναλωτής μπορεί να εντοπίσει εύκολα την προέλευση της ενέργειάς του και να επιλέξει τον πάροχο, ανάλογα με τις προτιμήσεις του για την μέθοδο παραγωγής, ακόμη κι αν αυτό συνεπάγεται μεγαλύτερο κόστος κιλοβατώρας. Με την ένταξη περισσότερων DERs στο ευρύτερο δίκτυο επιτυγχάνεται λοιπόν η αύξηση του ποσοστού των ενεργειακών αναγκών που καλύπτονται από ΑΠΕ, σημειώνοντας βέβαια πως η απρόβλεπτη συμπεριφορά των DERs μπορεί να αναταράσσει κάπως την ισορροπία παραγωγής-κατανάλωσης, όταν αναφερόμαστε σε ένα αποκομμένο και αυτόνομο, μικρότερης κλίμακας σύστημα.

Σε ένα τέτοιο «πρότυπο» δικτύου που στηρίζεται στην αυτόνομη λειτουργία των DERs, γίνεται σαφές πως οι δαπάνες για την αξιοποίηση οποιουδήποτε είδους κεντρικού διαχειριστή ή επόπτη είναι περιττές. Οι συναλλαγές θα πραγματοποιούνται μέσω αυτοματισμών, ενώ οι λεπτομέρειες αυτών θα ρυθμίζονται άμεσα μεταξύ των ενδιαφερόμενων μερών, με την προϋπόθεση βέβαια πως οι μηχανισμοί του συστήματος εγγυώνται ασφάλεια και ταχεία υλοποίηση. Είναι απευθείας εμφανές λοιπόν πως η λειτουργία ενός Energy Market σε τοπικό επίπεδο (Local Energy Market – LEM) έχει απαιτήσεις που μπορούν να καλυφθούν από την τεχνολογία των blockchain. Σκοπός της εργασίας αυτής λοιπόν, είναι η προσομοίωση της λειτουργίας ενός LEM με τη χρήση ενός Dapp το οποίο θα «τρέχει» -μέσα στα πλάισια του Ethereum blockchain- όλες τις απαραίτητες διαδικασίες για να επιτευχθεί η οικονομικώς βέλτιστη μεταφορά ενέργειας από τους παραγωγούς στους καταναλωτές.

Αυτό που προσδίδει στο Ethereum την ιδιαίτερη ταυτότητά του, είναι το σύνολο των πρωτοκόλλων τα οποία διαχειρίζονται την κοινόχρηστη βάση που περιέχει δεδομένα για τις συναλλαγές μεταξύ των κόμβων. Έτσι, πέραν από το κανονικό blockchain το οποίο αποκαλούμε Mainnet, υπάρχουν και άλλα δίκτυα που υπακούουν σ’ αυτά τα πρωτόκολλα, στα οποία αναφερόμαστε ως Testnet, κάποια από τα οποία μάλιστα αξιοποιούν και διαφορετικές μεθόδους συναίνεσης. Τα Testnet, όπως δηλώνει και το όνομά τους, είναι χρήσιμα για να ελέγχουμε τη λειτουργία των Dapps που αναπτύσσουμε, γιατί τα νομίσματα που χρησιμοποιούνται για τις συναλλαγές εκεί δεν έχουν πραγματική αξία και έτσι μπορούμε να δοκιμάσουμε διάφορες ενέργειες χωρίς να φοβόμαστε το κόστος τους σε gas. Στη συγκεκριμένη εργασία όμως και ειδικά στα αρχικά στάδιά της, χρειάζονταν συνεχείς και τακτικοί έλεγχοι της λειτουργίας της κάθε συνάρτησης του smart contract, όπως επίσης χρειάστηκε και η δυνατότητα για την υλοποίηση σύνθετων σεναρίων τα οποία περιέγραφαν συγκεκριμένες συναλλαγές μεταξύ μελών του δικτύου. Τα Testnet λοιπόν δεν μπορούσαν να ικανοποιήσουν αυτές τις ανάγκες και γι’ αυτό οι δοκιμές του project πραγματοποιήθηκαν στα πλαίσια ενός private blockchain.

Οι δοκιμές σε ένα private blockchain μπορούν να συγκριθούν με την ανάπτυξη web εφαρμογών σε έναν τοπικό server. Ένας προγραμματιστής δημιουργεί ένα τοπικό δίκτυο, οι κόμβοι του οποίου επικοινωνούν μόνο μεταξύ τους, απομονωμένοι από τον έξω κόσμο, επιτρέποντάς μας να ελέγχουμε πλήρως όλες τις δραστηριότητές τους και να επισκοπήσουμε αιτιοκρατικώς τις εξελίξεις εκεί. Επίσης, μπορούμε να ρυθμίσουμε τις παραμέτρους του mining και του compiling, κάτι που μας προσφέρει τη δυνατότητα να ελέγχουμε ταχύτερα τις λειτουργίες των smart contract και να επαναλαμβάνουμε τις δοκιμές πιο τακτικά. Το πρόγραμμα που δημιουργεί το προσωπικό μας Ethereum blockchain λέγεται Ganache, τρέχει στο command line του συστήματος και είναι ένα εργαλείο που αναπτύχθηκε από το Truffle Suite, στο οποίο θα αναφερθούμε και παρακάτω. Το Ganache μας παρέχει μια πληθώρα επιλογών για τη διαχείριση του δικτύου αλλά και των κόμβων και των λογαριασμών του, δίνοντάς μας συνεχώς μία πλήρη εικόνα του συστήματος και των δραστηριοτήτων που λαμβάνουν χώρα σ' αυτό, άρα και όλων των απαραίτητων πληροφοριών για να προσομοιάσουμε υποθετικά σενάρια για το Mainnet.

Ένα ακόμη εργαλείο που χρησιμοποιείται σ' αυτό το project είναι το Provable oracle. Εφόσον οι εφαρμογές του web3 καλούνται να αποτελέσουν μια εναλλακτική, βελτιωμένη εκδοχή του Παγκόσμιου Ιστού που έχουμε γνωρίσει, η πρόσβαση των Dapps σε δεδομένα από τον πραγματικό κόσμο και γενικά σε πληροφορίες εκτός του blockchain είναι απαραίτητη. Η ίδια η έννοια όμως του blockchain στηρίζεται στην πλήρη απομόνωσή του από άλλα δίκτυα, κάτι που εγγυάται και την ασφάλεια των συναλλαγών. Η λύση σε αυτό το ζήτημα έρχεται με τα oracles τα οποία είναι προγράμματα που ουσιαστικά λειτουργούν ως APIs (Application Programming Interface) για επικοινωνία με τον έξω κόσμο. Χάρη στα oracles, τα smart contracts μπορούν να επικοινωνήσουν με εξωτερικές πηγές για να λάβουν αλλά και να παραδώσουν δεδομένα, καθιστώντας έτσι εφικτή την οποιαδήποτε δραστηριότητα που θα μπορούσε να πραγματοποιηθεί και στον Παγκόσμιο Ιστό. Όπως είναι αναμενόμενο όμως, αυτή η «εξαίρεση» στην απομόνωση του δικτύου εγείρει κάποια σημαντικά ερωτήματα σχετικά με το αν αποδυναμώνεται η συνολική ασφάλεια του συστήματος. Το κάθε oracle αποτελεί μια οντότητα που δεν στηρίζεται στην αποκεντρωμένη λειτουργία, κάτι που σημαίνει πως η χρήση του απαιτεί και ένα επίπεδο εμπιστοσύνης προς τους δημιουργούς και διαχειριστές του, ενώ υπάρχει πάντα το ρίσκο της έκθεσης του σε κάποιο κακόβουλο στοιχείο, το οποίο θα μπορούσε για παράδειγμα να παρέχει στο blockchain ανακριβή δεδομένα.

Σε κάθε περίπτωση το Provable oracle που αξιοποιείται σε αυτήν την εργασία καταφέρνει να επιλύσει αποτελεσματικά τα ζητήματά μας. Ο λόγος που έγινε η επιλογή του είναι το ότι μας παρέχει ένα ειδικό εργαλείο που λέγεται ethereum-bridge, το οποίο χρησιμοποιείται αποκλειστικά για την επικοινωνία των πηγών δεδομένων του Provable με private blockchains. Μας ενημερώνει λοιπόν διαρκώς με την πραγματική αξία του ETH σε δολάρια, αλλά επίσης καταφέρνει να λύσει ένα σημαντικό ζήτημα που αφορά την προγραμματισμένη εκτέλεση στο μέλλον μιας λειτουργίας του smart contract, καθώς κάθε φορά που ζητάμε δεδομένα από το provable, μπορούμε να προσδιορίσουμε ως παράμετρο της κλήσης μας μία χρονική καθυστέρηση -μεταξύ άλλων- στην τελική αποστολή του αιτήματος. Η κάθε κλήση, εκτός από την πρώτη που είναι δωρεάν, έχει ένα συγκεκριμένο κόστος σε gas που προκύπτει από τις παραμέτρους και τα χαρακτηριστικά του αιτήματος, το αντίτιμο του οποίου καταβάλλεται από έναν λογαριασμό τον οποίο έχουμε τη δυνατότητα να ορίσουμε.

Το γενικότερο πλαίσιο στο οποίο αναπτύχθηκε η εφαρμογή είναι η δομή του Truffle. Το Truffle Suite, πέρα από εξειδικευμένα προγράμματα όπως το Ganache, μας παρέχει και ένα σύνολο από εργαλεία που χρησιμοποιούμε μέσω command line, κατάλληλα για τη διαχείριση smart contracts, το οποίο αποκαλείται απλώς Truffle. Τα smart contracts στο

Ethereum μπορούν να γραφτούν σε ορισμένες εξειδικευμένες γλώσσες προγραμματισμού, εκ των οποίων η συχνότερα χρησιμοποιούμενη και αυτή στην οποία έχουν συνταχθεί τα smart contract μας, είναι μία αντικειμενοστραφής, εμπνευσμένη από τη C++ γλώσσα, που λέγεται Solidity. Το Truffle λοιπόν μάς διαθέτει έναν compiler ο οποίος διατρέχει όλα τα solidity αρχεία ενός project και μετά το compilation δημιουργεί ένα νέο directory εντός του project, στο οποίο αποθηκεύει ορισμένα αρχεία τα οποία καλούνται artifacts. Το κάθε artifact περιέχει διάφορες πληροφορίες για ένα συγκεκριμένο smart contract, αλλά και δεδομένα που χρειάζονται για την ανάπτυξή του στο blockchain. Χρησιμοποιώντας το Truffle, μπορούμε να ορίσουμε τα αρχεία που θα αναπτυχθούν, όπως και τη σειρά με την οποία θα εξελιχθεί αυτή η διαδικασία. Το Truffle μας επιτρέπει να διαχειριστούμε κάποια Javascript αρχεία εντός του project, που λέγονται migrations, τα οποία περιγράφουν αναλυτικά στον κώδικά τους όλη τη διαδικασία για τον τρόπο αναπτύξεως των smart contract στο Ethereum. Τα migrations εντοπίζουν τα artifacts που αντιστοιχούν στα smart contract που μας ενδιαφέρουν και αξιοποιώντας τα δεδομένα τους, υλοποιούν το σενάριο που έχει στο μυαλό του ο προγραμματιστής. Στη συνέχεια, προσδιορίζουμε την ταυτότητα του δικτύου και είμαστε έτοιμοι να αλληλεπιδράσουμε με το Dapp μας, χρησιμοποιώντας την κονσόλα του Truffle. Εκεί, πληκτρολογούμε javascript εντολές που απευθύνονται σε λειτουργίες των smart contract, προσιτές από συγκεκριμένους ή και όλους τους χρήστες του δικτύου, ενώ μας παρέχεται και η web3.js συλλογή βιβλιοθηκών, απαραίτητη για περιπλάνηση στον κόσμο του Ethereum. Αξίζει να σημειωθεί πως το Truffle μας δίνει και τη δυνατότητα να καθορίσουμε σύνθετες αλληλεπιδράσεις με τα smart contract μας και με άλλους κόμβους, χωρίς να χρειάζεται να τις καταγράφουμε σαν αλληλουχίες εντολών που εκτελούνται ξεχωριστά στην κονσόλα. Αυτό γίνεται χάρη στα external scripts, javascript αρχεία που φροντίζουμε να συνδέσουμε στο δίκτυο που τρέχουν τα smart contract και τα οποία καταγράφουν στον κώδικά τους προετοιμασμένες ιδέες του ιδιοκτήτη του λογαριασμού, για τις συναλλαγές με την εφαρμογή. Τέλος, ένα ακόμη χρήσιμο εργαλείο του Truffle, είναι το αυτοματοποιημένο πλαίσιο για testing που μας προσφέρει. Οι προγραμματιστές μπορούν να γράψουν test σε javascript και typescript τα οποία προσομοιάζουν τις λειτουργίες των smart contract που σχεδιάζουν, προκειμένου να δοκιμάσουν την εφαρμογή των ιδεών τους, χωρίς μάλιστα να χρειαστεί να αναπτύξουν τον κώδικα σε κάποιο δίκτυο. Φυσικά, υπάρχει και η δυνατότητα για συγγραφή test σε solidity για όσους θέλουν να πραγματοποιήσουν δοκιμές με μεγαλύτερη εγγύτητα σε πραγματικές καταστάσεις.

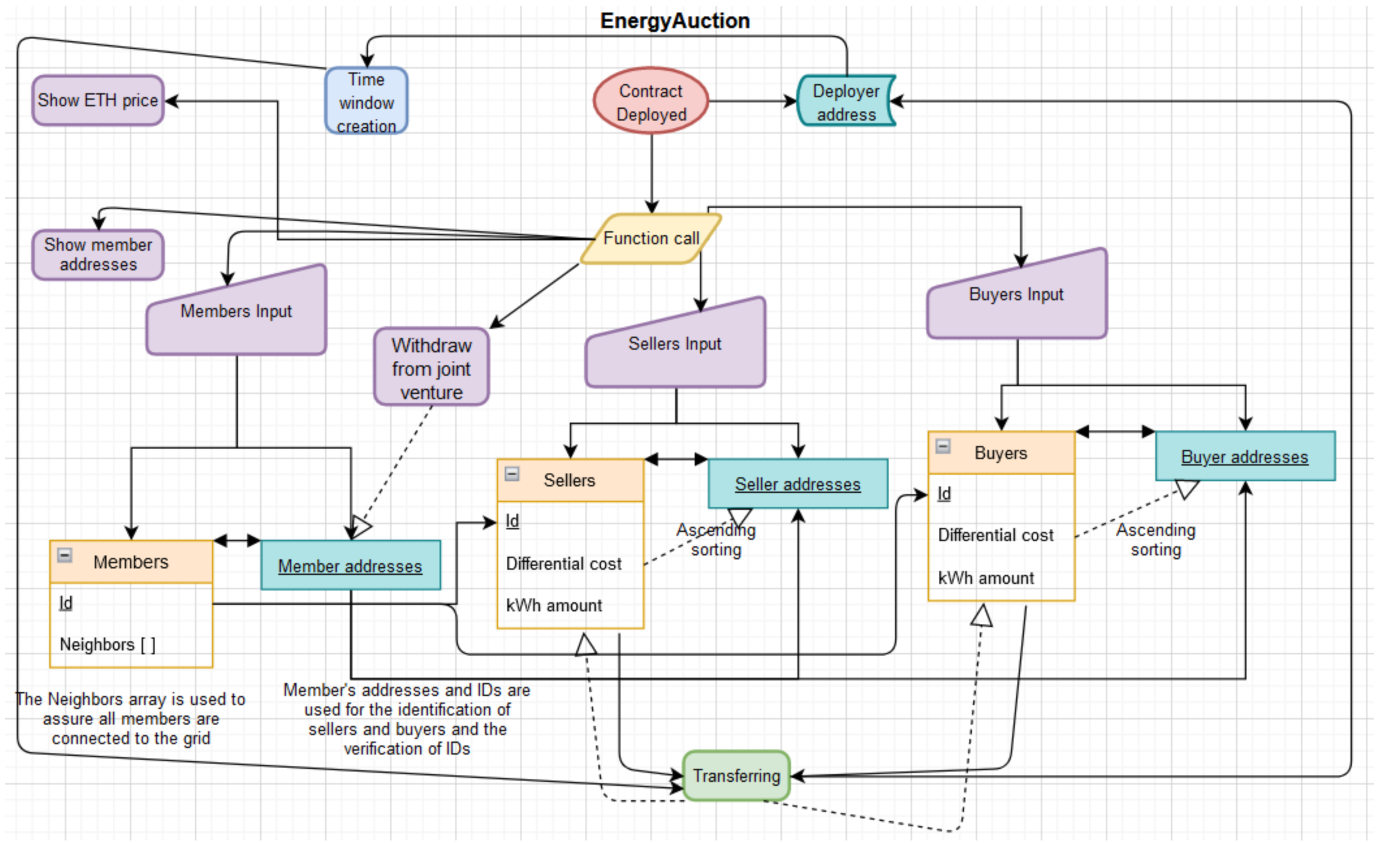
Κατ' αρχάς λοιπόν για να «τρέξουμε» την εφαρμογή μας είναι να δημιουργήσουμε το ιδιωτικό δίκτυο με το Ganache. Στην περίπτωσή μας, χρησιμοποιούμε το command line ενός Ubuntu λειτουργικού συστήματος και δημιουργούμε το προσωπικό μας blockchain, το οποίο έχει συγκεκριμένο network id και port μέσω του οποίου συνδεόμαστε. Σ' αυτό το δίκτυο δημιουργούνται αυτόματα 10 διευθύνσεις που έχουν από 100 Ether έκαστη στον λογαριασμό της, αν και αυτά τα χαρακτηριστικά μπορούμε φυσικά να τα τροποποιήσουμε. Το επόμενο βήμα είναι να συνδέσουμε το ethereum-bridge με το δίκτυό μας, μία ενέργεια η οποία απαιτεί κάποια mining funds καθώς το oracle ουσιαστικά θα αναπτύξει κάποια δικά του smart contracts για να επιτελέσει τη λειτουργία του. Έτσι λοιπόν, πέραν από τα στοιχεία του δικτύου, ορίζουμε και έναν λογαριασμό που θα επιμεριστεί αυτό το κόστος, ο οποίος θα ήταν προτιμότερο να μην συμμετέχει έπειτα στη διαδικασία των συναλλαγών ενέργειας, καθώς αυτό είναι ένα κόστος που δεν θα υπήρχε σε φυσιολογικές περιστάσεις. Το επόμενο βήμα είναι να κάνουμε, με τη βοήθεια του Truffle, compile και migrate τα smart contracts μας που αποτελούν το Dapp, αλλά και να προσδιορίσουμε τον λογαριασμό του υπεύθυνου για την ανάπτυξη της εφαρμογής, οποίος θα αναλάβει και τα κόστη για το mining. Όπως θα δούμε και στη συνέχεια, αυτός ο λογαριασμός θα αποζημιωθεί και θα έχει μάλιστα και κέρδος. Τέλος, ενεργοποιούμε την κονσόλα του Truffle και είμαστε έτοιμοι να αλληλεπιδράσουμε με την εφαρμογή.

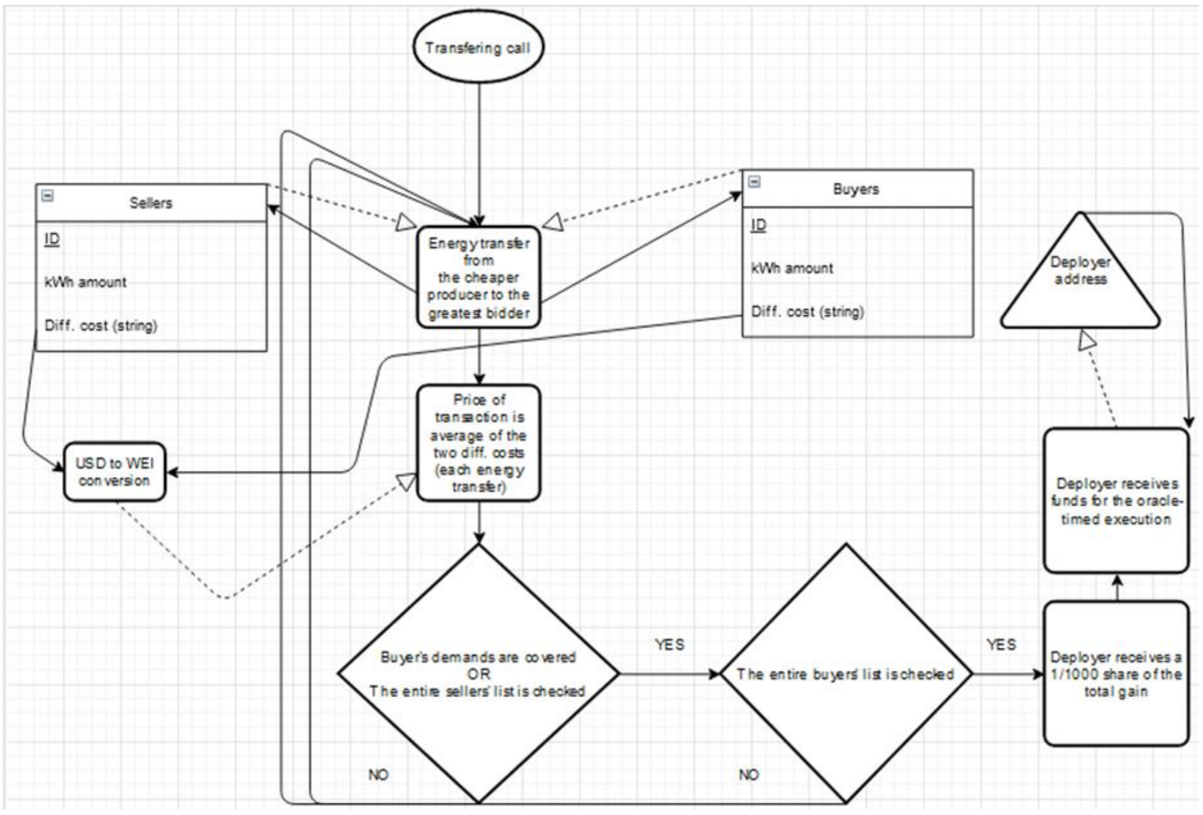
Το πρώτο βήμα είναι να ενεργοποιηθεί ένα χρονικό «παράθυρο», μέσα στο οποίο οι παραγωγοί και αγοραστές ενέργειας θα εκφράσουν τις προσφορές και τις απαιτήσεις τους. Κάθε 15 λεπτά, η εφαρμογή επεξεργάζεται τα δεδομένα της και πραγματοποιεί μια δημοπρασία, στην οποία ο παραγωγός που προσφέρει την kWh στην χαμηλότερη τιμή, την παραχωρεί στον καταναλωτή που κάνει την καλύτερη προσφορά. Αν οι απαιτήσεις σε ενέργεια του αγοραστή ικανοποιηθούν, ο παραγωγός δίνει την περίσσειά του σ' αυτόν που κάνει την αμέσως επόμενη καλύτερη προσφορά, ενώ αντίστοιχα αν δεν του παρέχει την ποσότητα που χρειάζεται, ο καταναλωτής απευθύνεται στον πωλητή με την επόμενη φθηνότερη kWh. Αυτή η διαδικασία επαναλαμβάνεται έως ότου δεν είναι δυνατό να πραγματοποιηθεί άλλη συναλλαγή και ιδανικά στο τέλος δεν πρέπει να υπάρχει περίσσευμα ενέργειας στο δίκτυο, ούτε κάποιο μέλος που δεν έχει ικανοποιήσει τις ενεργειακές του ανάγκες. Αυτό είναι και το κίνητρο για τους αγοραστές και τους πωλητές, ώστε να πραγματοποιούν πιο συμφέρουσες οικονομικά προσφορές. Η διαδικασία αυτή λοιπόν είναι πλήρως αυτοματοποιημένη, κάτι που σημαίνει πως κάποιες λειτουργίες πρέπει να εκκινούνται κάθε 15 λεπτά, χωρίς να υπάρχει φυσικά ένα φυσικό πρόσωπο που να επιβλέπει την εφαρμογή. Εδώ το ethereum-bridge μας δίνει τη λύση μέσα από την προαναφερθείσα δυνατότητα για προγραμματισμό μελλοντικής εκτέλεσης εντολών και την αξιοποιούμε δημιουργώντας μια αναδρομική διαδικασία εκκίνησης και τερματισμού του δεκαπεντάλεπτου χρονικού «παραθύρου». Η αρχική εντολή για το ξεκίνημα λοιπόν αυτού του κύκλου, μπορεί να δοθεί μόνο από το λογαριασμό που αντιστοιχεί στον υπεύθυνο για την ανάπτυξη της εφαρμογής, κάτι που σημαίνει βέβαια πως αυτός επιμερίζεται όλα τα έξοδα για το mining αυτών των διαδικασιών. Η πρώτη κλήση του ethereum-bridge είναι όπως είπαμε δωρεάν, οι επόμενες ωστόσο έχουν κόστος, για το οποίο θα αποζημιωθεί. Πρέπει όμως να αναφερθεί πως αυτή η διαδικασία υπονομεύει κάπως τον αποκεντρωμένο χαρακτήρα της όλης εφαρμογής.

Αφού γίνει η εκκίνηση το κάθε μέλος του δικτύου πραγματοποιεί μια σειρά από ενέργειες και αναλαμβάνει φυσικά και το κόστος εκτέλεσης αυτών. Αρχικά δηλώνουν τη συμμετοχή τους στο LEM και τους δίνεται ένα αναγνωριστικό που έχει το ρόλο του αριθμού μητρώου. Σ' αυτό το βήμα, πραγματοποιείται έλεγχος για την μοναδικότητα του αναγνωριστικού, αλλά και για την επικοινωνία που έχει το κάθε νέο μέλος με τους ήδη υπάρχοντες κόμβους του δικτύου, ώστε να αποφευχθεί η περίπτωση που κάποιος είναι απομονωμένος και δεν μπορεί να συναλλάξει ενέργεια με τους υπόλοιπους. Στη συνέχεια τα μέλη που επιθυμούν να πουλήσουν ενέργεια, καταθέτουν με το αναγνωριστικό τους μια προσφορά στην οποία κοινοποιούν την ποσότητα σε kWhs που έχουν διαθέσιμη και την τιμή στην οποία την προσφέρουν. Αντίστοιχα, οι καταναλωτές καταθέτουν τις απαιτήσεις τους σε ενέργεια, την τιμή στην οποία είναι διατεθειμένοι να αγοράσουν, αλλά δεσμεύουν επίσης στο λογαριασμό του smart contract το ποσό που προκύπτει από τον πολλαπλασιασμό αυτών των δύο στοιχείων, ώστε όταν οριστικοποιηθεί η συμφωνία, το smart contract να εκκινήσει τη διαδικασία για την αποστολή των χρημάτων. Οι αγοραστές μπορούν να μάθουν την αξία του Ether πραγματοποιώντας μία κλήση χωρίς χρέωση στην εφαρμογή, η οποία αξία μας είναι γνωστή από το Provable και η οποία ανανεώνεται κάθε 15 λεπτά, σε κάθε νέο κύκλο δηλαδή. Έπειτα, μετατρέπουν το ποσό από δολάρια Αμερικής σε Ether και το καταθέτουν, έχοντας ως διαβεβαίωση πως σε περίπτωση που δεν καταναλωθεί ολόκληρο, το περίσσευμα θα τους επιστραφεί. Αξίζει να αναφερθεί επίσης πως σε όλες αυτές τις διαδικασίες υπάρχει έλεγχος ταυτοποίησης, ενώ στην περίπτωση των αγοραστών υπάρχει και έλεγχος για την επάρκεια του ποσού που κατατέθηκε.

Μετά το πέρας των 15 λεπτών λαμβάνει χώρα η διαδικασία που περιγράψαμε ανωτέρω. Όταν ο αλγόριθμος καταλήξει σε ένα ζευγάρι παραγωγού-καταναλωτή, αποδίδει στον παραγωγό ένα ποσό που αντιστοιχεί στο γινόμενο του αριθμού kWhs που συναλλάχθηκαν, επί το μέσο όρο των αξιών που είχαν ορίσει και οι δύο στην προηγούμενη φάση και στον καταναλωτή επιστρέφει τα υπόλοιπα από τα χρήματα που είχε αρχικά καταθέσει, καθώς αυτά θα είναι σίγουρα περισσότερα ή ισάξια από αυτά που συναλλάσσονται τελικώς. Η εφαρμογή στέλνει επίσης ένα σήμα που σηματοδοτεί την άδεια για μεταφορά της ηλεκτρικής ενέργειας, ενώ ένα μικρό ποσοστό του συνολικού ποσού θα πάει στον λογαριασμό του υπεύθυνου για την ανάπτυξη. Τέλος, καθαρίζονται οι προσφορές και οι αιτήσεις ενέργειας ώστε στο νέο κύκλο να ξεκινήσει η διαδικασία από την αρχή, αν και η λίστα με τα μέλη φυσικά δεν διαγράφεται. Αν κάποιο μέλος όμως επιθυμεί να αποχωρήσει από το LEM έχει τη δυνατότητα να πραγματοποιήσει την κατάλληλη κλήση για να διαγραφεί από το σύστημα και να αφήσει το αναγνωριστικό του.

Στα παρακάτω σχεδιαγράμματα έχουμε μια συνολική επισκόπηση της όλης εφαρμογής, αλλά και της συνάρτησης transferring, η οποία υλοποιεί όλες τις διαδικασίες που «τρέχουν» με το πέρας του δεκαπενταλέπτου.




Transfering call
Sellers
ID
kWh amount
Diff. cost (string)
Energy transfer from the cheaper producer to the greatest bidder
Buyers
ID
kWh amount
Diff. cost (string)
Deployer address
USD to WEI conversion
Price of transaction is average of the two diff. costs (each energy transfer)
Deployer receives funds for the oracle-timed execution
Buyer's demands are covered OR The entire sellers' list is checked
YES
The entire buyers' list is checked
YES
Deployer receives a 1/1000 share of the total gain
NO
NO

ΕΥΧΑΡΙΣΤΙΕΣ

Για τη διεκπεραίωση της παρούσας Διπλωματικής Εργασίας, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή Σπυρίδων Δενάζη και τους κυρίους Γεώργιο Γεωργακάκο και Νικόλα Ανδριόπουλο για τη συνεργασία, την εμπιστοσύνη, την υπομονή και την πολύτιμη συμβολή τους στην ολοκλήρωση αυτής.

Contents

1. Introduction

1.1. Contemporary issues and concerns regarding energy markets

The traditional route of electrical energy begins with large thermal power generators that produce the bulk of energy that covers the demands of a certain area. We can consider these areas comprising of entire countries or groups of countries, but also of specific regions that need to be examined as separate – energy wise – entities, such as isolated islands or any other land mass with a peculiar formation that requires a special attention to its energy management. The energy required is then transported across the corresponding region throughout a high voltage transmission network which guarantees the reliability of such a system. Underlying this grid, a distribution network is set up and it conducts the energy to low and medium voltage levels so it can be accessed by consumers. This is a form of a mainly unidirectional power flow, from the energy generators to consumers connected in medium and low levels of voltage. In this system, the consumers, who constitute the majority of actors, do not affect essentially its overall operation and only a handful of actors consisting of producers and system operators have an active role, by coordinating the system’s function.

Times nevertheless are changing and new challenges as well as opportunities appear, putting the structure described above into question. The global economy is experiencing a significant growth of businesses and population, with increasing demands for energy. A raise in production is entailed, which cannot be expected to be achieved simply by conventional energy systems. Most large power generators operate using fossil fuel like coal, a solution that used to be considered the cheapest and simplest one. Climate change however, increases the need for reduced dependency on conventional resources and several nations are aiming at increasing the dependency on renewable energies. Moreover, not all regions have access to such fossil fuels, meaning their energy production, and by extension their overall economic performance, is bound to the demands and policies of certain suppliers. Owing to these factors there has been an increase in the share of renewable energy towards the total energy of the world overall [1].

The introduction of more green energy sources into the distribution grid can be achieved through governmental planning, but also with the addition of distributed energy resources (DERs) throughout the network. Over the past years, the costs of renewable energy technologies have declined steadily due to technological advances and combined with an increase in the environmental concern of customers and legislators, it is easier than ever for many consumers to provide the grid with additional energy, mainly exploiting solar power. This way, they take on the role of the “prosumer”, an ordinary consumer which actively produces his own energy and proceeds to trade it with other members of the network. Technical challenges arise in the system operation, such as supporting the grid’s capacity, forecasting the intermittent behaviour of DERs, and avoiding grid congestion. New smarter grids are being developed with smart meters and information and communication tools (ICTs) to facilitate the transition towards new systems. Apart from these new technologies though, we also have new business models being discussed and developed to enhance the integration of DERs into distribution grids and provide services to smart grid stakeholders by empowering prosumers.

The work of this thesis describes one of these new business models and offers a potential implementation method. The main issue is designing a system in which prosumers have a prominent role, so they can essentially affect the flow of electricity and increase the share of renewables in the total mix. Taking advantage of the smart grid and the transfer of data between different prosumers (which we will consider “nodes” of the network) we can set up a digital platform in which energy is exchanged between nodes through auction mechanisms and in which each consumer can have additional information regarding the origin of his energy. We consider the grid to be applied in a relatively small geographical area and thus we create something called a Local Energy Market (LEM). A local market’s needs and demands can be met more easily with the collective efforts of simple, individual small-scale producers, especially if we consider the absence of major, large-scale industrial establishments in the area, whose energy demands would necessitate the assistance of larger, thermal power generators. An LEM is also considered to be a relatively autonomous system, meaning that our platform must be functional without depending on external aid, while also maintaining security and transparency for all interactions between nodes. The innovation that blockchain technology has brought the world over the past few years, is the perfect tool for the creation of a platform that provides total security, transparency and autonomy, while also ensuring that all processes occurring within the system will be carried out automatically, without control from a central coordinator. At its core, a blockchain is a Distributed Ledger, a database in which all records are linked together via a cryptographic method and many recently developed blockchain platforms do not have some of the characteristics listed above, like the decentralization aspect, but most of them satisfy the criteria of a blockchain platform as they were noted down after the launch of the first and most famous blockchain, Bitcoin.

This thesis details the creation of an application, operating on the Ethereum Blockchain, which implements all the procedures and energy auction mechanisms that occur in an LEM environment. Ethereum is the second largest blockchain and a platform that allows users to perform more complex interactions with each other, than those taking place in Bitcoin. This application is supported by a smart contract, a computer program that is intended to automatically execute, control or document events and actions according to the terms of a detailed agreement between different parties. Smart contracts allow for all kinds of potential interactions and thus are the basis for the development of an application customized to our needs. This application has not been tested in a real-life scenario and neither has been deployed on the actual Ethereum blockchain, in which transactions cost real money, but has rather been tested on a local environment operating in a similar way to Ethereum. Members of the LEM can be considered as nodes of the Ethereum network who access our application, which utilizes the information from the smart grid to identify them and regulate the energy transactions, while ensuring security, transparency and decentralization.

Lots of other projects however have been developed or are being developed at the moment, trying to tackle the issues we mentioned before. Some of them have been the inspiration for our own project, or highlighted the need for the creation of more Local Energy Communities.

1.2. New energy market models

The emergence of prosumers and smart grids creates opportunities for energy trading transactions that can occur between entities such as the prosumers themselves, the grids and the ever-challenging, costly energy storage. As energy is the most critical component for growing the economy, this paradigm shift in energy trading requires a system that is secure, efficient and fosters energy economics. The integration of these DERs along the distribution grid however, creates local variations that can affect the optimal operation of transmission and distribution networks and disrupt the balance between production and consumption. Despite this, local supply variation can be matched with local demand variation, resulting in an internal way to solve the problem and producing a potential business model to increase the hosting capacity of the distribution grid without investing in it. Many state of the art projects have developed in the past few years aiming for further research and breakthroughs in the application of these ideas.

A key project covering the issue is CoordiNet which received funding from the European Union's Horizon 2020 research and innovation programme and was completed in 2022 after 42 months of running demos in three European countries [2]. The project's purpose was to increase the coordination between distribution system operators (DSO), transmission system operators (TSO), the end consumers and the grid service providers which help keeping the operation within acceptable limits for the security of supply and enable TSOs and DSOs to use the grid more efficient. In the Spanish demo, several demand-side and renewable resources, mainly wind power, were integrated, with most units already participating in electricity markets. However, Voltage Control, Local Congestion Management and Controlled Islanding, being new services, required the development of products and market frameworks to be defined and agreed between the TSOs and DSOs. Active engagement of stakeholders and the use of cascading funds attracted a significant number of flexibility services providers (FSPs) to participate in the project. This way, a Local Market Platform and control systems were developed, along with hardware devices implemented to monitor and control flexibility providers, connected to the distribution networks. The platform accommodated the market clearing algorithms to determine the accepted orders and prices and defined the interactions with the actors. Overall, the Spanish demo showcased that local markets are compatible with the participation of small FSPs that are not able to participate in other types of flexibility markets and generally highlighted the need for interoperability of distinct flexibility markets. The second demo took place in Sweden and tested the performances of two different market models for the interaction between TSOs and DSOs, the Multi-level and the Distributed Market Model. This demo was supported by the development of a marketplace and flexibility tool focused on a holistic approach bringing together technical, cultural, political, and financial perspectives. It demonstrated how DSOs can utilize flexibility for the operation of the network in an increasingly dynamic and digitalized way. The use of flexibility was proven beneficial to alleviating network congestion, with the prerequisite of having sufficient market liquidity. Its conclusion, however, showed that existing regulations will not be sufficient to unlock the targeted level of flexibility. The final demo was conducted in two testing sites in Greece, in which a mix of assets, such as wind farms, photovoltaic stations, batteries, small diesel gensets, and buildings, were connected to test possible flexibility services. Just as in the Swedish demo, the Multi-level and the Distributed Market Model were investigated and the results confirmed that the introduction of a Local Market can enable the procurement of ancillary services from resources connected to the distribution grid. This market also allowed

DSOs to have a more proactive role in the operation of the system while increasing the integration of DERs. The overall demo analysis indicated that the Multi-level market had more advantages than the Distributed Market and confirmed the importance of local markets encouraging the procurement of ancillary services from distributed energy resources.

Another major effort to examine the future of energy markets is the EM-Power Europe project [3]. It is the international exhibition for energy management and integrated energy solutions and focuses on transitioning the existing power grids, to modern smart grids, through digitalization. It has already conducted pilots in lots of European countries, in which economic benefits for peer-to-peer trading were provided throughout the community and external agents also participated aiming for profit. Thus, the economic feasibility of energy communities was illustrated and in places such as Norway, parts of the project were commercialized. More specifically, EMPOWER pilots in Hvaler, showed that load reduction can be offered to the DSO and thus contribute to congestion management. In the German pilot, the load shifting potential in households was greater than that of public buildings. In Hvaler, EMPOWER essentially formed Norway's first operational microgrid and proved both the technical and the economic feasibility of energy communities and local trading. Solar and wind power was harnessed and residential homes – consumers and prosumers alike – along with a municipal recycling facility and storage units were integrated to local energy market operations.

Amongst other projects however, a new trend seems to arise in the development of state-of-the-art projects regarding energy markets. The Pebbles project running in Germany [4] has developed a platform for local energy trading, functioning on the basis of the blockchain technology. Pebbles' goal was to design, develop and operate the blockchain platform - using a peer-to-peer or 'P2P' model – and to integrate grid usefulness into the market mechanism in regional "energy-supply areas". Pebbles analyzed the grid's operation in all kinds of ways, from physical measurements and IT to business models and energy markets. This way, it can be used in several areas within the field of energy supply, but also utilize its P2P platform to promote cooperation between technology providers, platform providers, platform users and DSOs and eventually generate innovative business models for the stakeholder which are based in the exchange of grid services and direct trade. Generally, P2P models are being examined by lots of non-blockchain related projects, such as CoordiNet's Swedish demo. Blockchain's appeal however is founded on its infrastructure which assures a high degree of process automation and data security. But at this point we should examine what exactly blockchain is and why it can be a helpful tool in developing Local Energy Communities, in more detail.

1.3. Blockchain technology in the service of energy distribution

As the population and its energy demands are growing worldwide, so is energy production. Traditional means of production and distribution seem to be struggling to cover the needs of everyday people, but also those of the industrial sector, especially if we take into account the slowly but steadily increasing awareness for the climate change and its effects, which is directly linked to fossil fuel consumption. Conventional fuel reserves also happen to be limited in supply, making them unfit for meeting the demand in the long-term, but are also concentrated in certain geographical spots, a territorial patchiness that separates countries to suppliers and buyers. Considering the costly and challenging reality of electricity storage, lots of electricity providers, mainly in the advanced world, are looking for new ways to produce and transfer cheap and mostly clean energy to their communities. Since large amounts of green energy, require a significant amount of production units, e.g. solar panels, a new complex challenge regarding the distribution has emerged, meaning that there is a need for more distributed networks that will also be able to cooperate with other systems. Of course, all kinds of systems today are generally expected to be more interoperable, following the demands of globalization, but are also expected to function automatically, without being surveilled by a central coordinator. Many new technologies have emerged in recent years in the service of this tendency and one that could provide some solutions to the energy issues we face today, is the blockchain technology.

The first one to use the term "chain of blocks" was Satoshi Nakamoto, in the paper he published on 31 October 2008 [5] and on the 3rd of January 2009, with the creation of the Bitcoin network, the public became aware of a first implementation of the methods described in it. Using a digital currency called Bitcoin, the network of the same name, allowed for peer-to-peer transactions, carried out by its members, without the need for an administrative authority. People from around the globe could exchange goods directly, privately, in a totally secure way, without having to rely on any intermediaries. This example of a new, radical way of doing transactions in a completely decentralized environment, would soon provide the inspiration for more ways the blockchain technology could improve any kind of interactions between different parties, agreeing on certain principles regarding the rules of their engagement. The Ethereum Blockchain provides one of the most well-known decentralized environments [6], used for the design of applications called Dapps (Decentralized Applications) that are accessible to everyone in the system, developed writing computer programs called smart contracts.

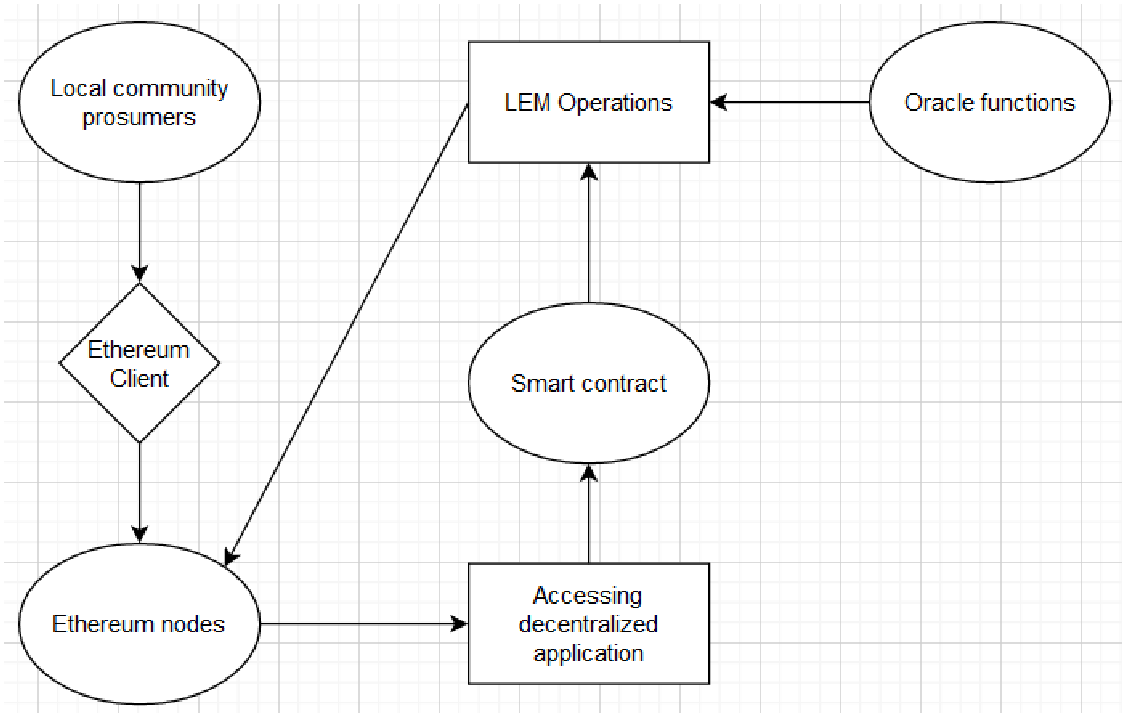


Figure 1. The project's overview

1.4. Blockchain as a distributed ledger

The most obvious issue Bitcoin had to tackle first, was the way the data from any exchange would be stored. Data should supposedly be shared between all the members of a network, also referred to as "nodes", in order to form a truly, fully decentralized and trustless space for transactions, which would also guarantee safety for all nodes by shutting off any malicious or accidental attempts to redirect the assets to different nodes [7]. That meant that it was the nodes' responsibility to check every activity on the network and through a majority consensus, approve its execution and register the data in a shared database. And apart from the absence of authorities able to interfere on the system and have unlimited access to all kinds of data, this new method should also guarantee higher levels of security than conventional payment networks, in order to provide incentives for participation. Thus there was a need for a common, shared ledger of transactions - or Distributed Ledger (Distributed Ledger Technology-DLT) as it is called - that would also give the majority of the nodes, the ability to validate every new recording in the ledger.

DLTs are databases that are spread across multiple sites, countries or institutions and can be either private, permissioned or unpermissioned [8]. Private DLTs allow only verified participants to join, through an authentic and verified invitation. A permissioned DLT can have an owner or administrator that assigns to certain trusted actors the duty of checking every new entry in the ledger and maintaining its proper functioning. These shared ledgers are considered "distributed", because all members in the network can see the stored data. An unpermissioned DLT has no such owners and all of its nodes can take part in the checking of every new registration, a prodedure that is carried out through one of many different consensus protocols available. Everyone can contribute data to the ledger and everyone has identical copies of the data. Blockchains can be considered DLTs in which the information of any activity gets stored in a specific block, which also contains a timestamp and information from the previous block, processed through a cryptographic hash function, this way ensuring that every block has information about all the preceding ones. That means that trying to manipulate the system or accidentally install false data, would require from one to continue "building" a whole different chain, from that point onwards, than the one the others would expect to see. Whether someone could do that though, depends on the level of control he would have over the system, which differs according to the consensus protocol. Blockchains can also be divided into private, permissioned and unpermissioned. Generally, unpermissioned blockchains – just like all unpermissioned DLTs – are much slower at adding new records. The Bitcoin network is an example of one unpermissioned blockchain and therefore is considered decentralized and censorship resistant.

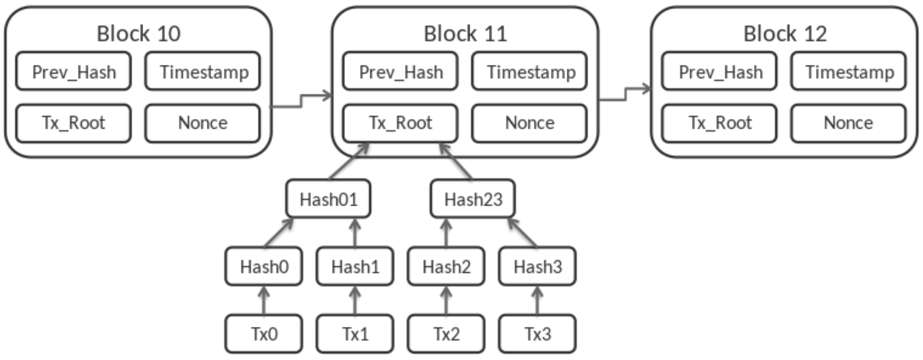


Figure 2. Bitcoin Blockchain structure. Transaction data are also processed by hash functions

1.5. Mining

Like we've mentioned, the nodes in any public blockchain will work together to verify which activities' execution should be approved and thus, which data should be added to the next block. This procedure, which forms the new blocks, is called mining and it is the backbone of the blockchain functionality. The actions the miners are supposed to perform, as well as the way the nodes are granted permission to mine, vary from one protocol to another. Generally speaking, the responsibility of verifying transactions belongs to nodes called Blockchain Validators, but on most unpermissioned blockchains, such as Bitcoin, everyone is free to become a validator and essentially a miner, since only they have economic motivations, although managing to profit from mining is a difficult task that eventually comes down to committing to a competitive "race" [9].

The main incentive for mining, is the prize of the transaction fees, rewarded to the first person that completes any activity check, whereas the term transaction, is used to describe any kind of interaction between two or more members in the network. When the different parties agree on a transaction, it is broadcasted on all nodes and when the broadcast is detected by the miners, the next step is to validate it as "legitimate". The fees provide the necessary compensation for the miners' actions, but if we take into consideration the large numbers of transactions that need to be processed, fee prices might get higher if the people behind the exchange wish for it to become a higher priority and thus be completed faster. Moreover, depending on the protocol, mining may be an extremely resource-consuming procedure and taking competition into consideration, we can see how fee prices could maintain an ascending tendency, just so that mining would keep on being profitable. By the time a blockchain gets big enough, mining starts attracting lots of attention, causing the competition to grow stronger and the expenses to skyrocket. This is why many miners decide to join mining pools and combine their efforts with others, with which they are going to share their collective gains. There are many though who believe that this practice opposes the decentralized character of the blockchain. One last thing that should be noted, is that the Bitcoin network rewards with new “coins” a miner for every new block he creates, but Bitcoin as an exchange has a finite number of currency units, which means that in the future, transaction fees will be the only source of income for miners.

After the validation, miners will examine some other transactions and lastly, put them all together in a block. Considering that miners work independently, many blocks for the same transactions could emerge and start developing their own chains, creating what we call a fork. Eventually though, a consensus will be reached among them and only one chain will continue growing and become part of the distributed ledger.

1.6. Forks

Most forks are short lived and are formed due to separate efforts during the mining process, in a big, distributed system. When the forks are not accidental though and a new branch, bringing some protocol change, is developing and does not get abandoned, we can say that we have a hard fork. These hard forks can be used to add new features to a blockchain, but they can also be used to reverse the effects of hacking, or avert catastrophic bugs, as was the case with a fork on the Bitcoin network, on 6 August 2010. In case of a hard fork, the rules on which the mining is based are essentially changed, so that the blocks produced according to the new rules, would be considered invalid by the old ones. Sometimes, old branches may keep on adding new blocks, in case there is a certain strategy by the designers of a permissioned blockchain, or a major dispute inside the community of an unpermissioned one, concerning the policy change.

1.7. Blockchain consensus protocols

There are many different ways in which the nodes will reach a consensus and the main ones are listed below. Something that should be noted, is that permissioned or public blockchains may also follow some of these protocols, although only authorized participants have the rights to perform the necessary activities for their execution.

- **Proof of Work** (PoW) is the most famous and widespread blockchain consensus protocol. It was originally introduced as a term in a paper released in 1999 [10] although similar ideas had been expressed before. It is the basis on which Bitcoin operates and thus it became the first ever blockchain protocol. Basically, any transaction consists of public and private data, merged together and processed through a hash function. This function manages to produce a unique output for every different piece of input (although the function cannot be considered injective), which cannot though lead back to the original information. Miners then, given the public data and the products of the hashing, try to guess the original elements of the transaction, so they can verify it and along with other transactions, add it to a new block. This is done by adding a nonce to the public data and essentially, repeatedly altering it slightly, until the merged data will produce the same hashing result as the one we already have. As one can imagine, this is a process requiring a lot of computing resources and more importantly, lots of energy. Given the competition and the fact that mining rewards are given to the first miner who mines a new block, which means that the difficulty of getting them is proportional to the demand, we can see that Proof-of-work mining can become an expensive pursuit, but also makes any attempt to cheat the system futile. Most nodes will track the "valid" path and ignore other branches, unless the attacker has at least half of the computing resources of the network and therefore the ability to guide the rest of the members, something extremely unlikely to happen, the bigger the blockchain is. Moreover, in order to achieve a "51% attack", one would have to add more computational units to the network and consecutively, increase the difficulty of the attempt. It should also be noted that this energy-consuming character of the protocol, has made all blockchain systems implementing it, receive major criticism over the big carbon footprint they result in.

- **Proof of Stake** dates back to 2011 and is one of the most highly anticipated protocols, to be implemented to a big blockchain. Proof of Stake (PoS) as a concept, states that any member can become a validator and is attributed mining power, proportional to the amount of tokens he holds [11]. The main benefit of this protocol, is rendering the mining "arms race" taking place on PoW blockchains obsolete. This arms race generates a massive carbon footprint and the massive energy demands could force miners to sell their awarded coins for fiat money, which could lead to a downward movement in the price of the cryptocurrency. Moreover, most of the mining in major PoW networks like Bitcoin, is now done by large, well-financed pools, that exclude the general public from the mining process, but also contradict the original vision of their designers, for a trustless, decentralized distributed ledger. In PoS, instead of utilizing energy to answer PoW puzzles, a miner is limited to mining a percentage of transactions that is reflective of their ownership stake. This way, mining becomes a more open and decentralized procedure that should not spark as much competition. Of course, Proof-of-Stake, being a more sophisticated approach to consensus protocols, does not guarantee the solid results, regarding security, that Proof-of-Work has proven to give, but offers some interesting alternative solutions to many safety issues. For example, in the –unlikely-case of the 51% attack, any attempts of the attacker to deceive the network in which he holds a majority share, would not be in his best interests. A majority stake owner has more incentives to maintain a secure network, in order to keep a high value of his token holdings. For the time being, Cardano is the biggest blockchain platform, implementing Proof of Stake.

- **Proof of Authority** (PoA) is a protocol that can be used on public networks, but is suited mainly for private ones. In PoA-based networks, transactions are verified by validators, who consist of approved accounts. Validators run software allowing them to put these transactions in blocks. The process is automated and does not require validators to be constantly monitoring their computers. It, however, does require maintaining the computer (the authority node) uncompromised. The main incentive for honest activity, comes from the efforts one makes to earn such a position, but also from the fact that nobody wants a negative reputation attached to his identity. This is, naturally, a more "classic" and centralized way of checking the activity in a system. The most notable PoA platform right now, is VeChain.

There are also many other consensus protocols, like Proof of Space that uses disk storage to validate transactions, or Proof of Burn, but those that were mentioned above are the most notable ones and also the only ones that will bother us in this thesis.

1.8. Smart contracts

Like we've already mentioned, a blockchain is a database containing the data of multiple interactions between different members of a network. As a technology, it provides total security during these interactions, meaning that every activity is guaranteed to be executed properly and as intended, within the blockchain. In order for this to happen, the network has to be a closed system, so that the validators will check the soundness of all actions and come to a consensus, unaffected from any outside influences. The first and most well-known blockchain, Bitcoin, is a monetary system with a digital cryptocurrency that can be used as a token, only within its confines. Inside Bitcoin, all interactions are essentially transactions, money transfers from one party to another. But as we said, the distributed ledger can contain data from any kind of activity between the nodes and therefore, can support the execution of complex and elaborate operations and the development of applications, functional within the bounds of the network.

A smart contract is a computer program or a transaction protocol, which can describe the implementation of a complex agreement, between many different parties. The concept existed long before the appearance of the first blockchains, but it soon became obvious how this new technology could utilize the full potential of this idea, promising true independence from any authorities or intermediaries and absolute safety for the members. In a blockchain, smart contracts are simply programs stored on it that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any third party's involvement or time loss. They can also automate a workflow, triggering the next action when conditions are met [12]. Within a smart contract, there can be as many stipulations as needed to satisfy the participants that the task will be completed satisfactorily. To establish the terms of agreement, participants must determine how transactions and their data are represented on the blockchain, consent on the "if/when...then…" rules that govern those transactions, explore all possible exceptions, and define a framework for resolving disputes.

1.9. Ethereum

Ethereum is the most known platform that supports the deployment and execution of smart contracts. It was first conceived as a concept in 2013 by programmer Vitalik Buterin and it was to be a decentralized, open-source blockchain. In 2014, development work commenced, supported by crowdfunding and in 2015, it was finally introduced to the public. Like most blockchains, it took inspiration from Bitcoin, but built on this groundwork to make it something more than a payment platform. It forms a whole different space for all kinds of activities and could potentially even be the framework for a new Internet.

Ethereum operates on a single, canonical computer (called the Ethereum Virtual Machine, or EVM) that represents the distributed ledger, as every member of the network agrees on its state and holds a copy of it [13]. Like all blockchains, it has its own native cryptocurrency called Ether (ETH) - and its denomination called Wei – which is not hard capped and allows for a market for computation. Every time a node wishes to execute an ETH transaction, it broadcasts an encrypted request to the EVM, to perform an arbitrary computation and then other participants verify, validate, and carry out this computation - a series of events that essentially constitute the mining process. This execution causes a state change in the EVM, which is committed and propagated throughout the entire network and the cryptographic mechanisms ensure that this change won't be easily altered or revoked.

All these interactions though are not usually the result of direct communication between two parties and/or the ad hoc synthesis of new code. Rather, developers create applications into EVM storage and then users try to execute some of their operations, modifying their requests through various parameters, which have been defined by the programmer. Thus, Ethereum is a universe of all these decentralized applications (Dapps), whose functionality is supported by the EVM, which combine a frontend user interface and a backend, which consists of one, or many, smart contracts. This is the great innovation of Ethreum, it makes transactions simply the medium needed for the use of an application and this application could deal with any ordinary subject such as gaming, technology, or finance. The concept of DeFi (Decentralized Finance) was born with Bitcoin, but smart contracts managed to go one step further and make even digital money programmable. Now, developers on the network can use this new feature presented to them and create their own tokens, in the confines of the blockchain, such as ERC-20 (a technical standard for Ethereum implementation of monetary assets) cryptocurrencies. And since Ethereum is open-source, all smart contracts in it are accessible and transparent, which means that a Dapp can even include contracts authored by somebody else. All this backend code runs on the decentralized peer-to-peer network that is Ethereum, giving these applications their name, but the frontend can also get hosted on decentralized storage such as IPFS [14]. This user interface can be written in any language, just like most apps, but smart contracts have to be programmed in specific languages. The most frequently used of them is Solidity, a high level, object-oriented, curly-bracket type language that has been highly influenced by C++ and also the one that we are dealing with in this thesis. Another very well-known one is Vyper, a pythonic programming language that has less intricacies than Solidity and is easier to read, but has less features. For those with an experience on working with Ethereum and a general programming experience, Yul and its extension Yul+ are also available, as intermediate languages for the EVM that support Ewasm, an Ethereum flavored WebAssembly [15]. We should also note the existence of Fe, a statically typed language for the Ethereum Virtual Machine, which is inspired by Python and should be easy for all to learn, but is still on the early stages of development.

Apart from the innovation of Dapps though, the way data are stored into the Blockchain and mining is performed, is really similar to Bitcoin. Users have wallets of accounts made up of a cryptographic pair of a public and private key and these accounts are specifically called externally-owned. Every time a user makes a transaction, he reaches out to others' public keys through his own public key, but signs the transaction with his private one. Essentially, users holding cryptocurrency, is identical to holding private keys and this is why we say an account is an object with an ether balance that can send transactions on Ethereum. Apart from users though, smart contracts have accounts as well, which can also hold ETH and interact with other smart contracts to perform transactions. The main difference between the two is that in order to create a contract account you first have to deploy it, which will cost you the miners’ fees and also, in order for a contract account to interact with others, it first has to be stimulated by receiving a transaction.

The cost of the deployment of a smart contract and of the execution of its functions, constitutes the gas. Gas is a monetary unit that measures the amount of computational effort these activities require and is linked to the rewards for mining a new block and thus, its price is determined by the demand and the mining activity. Ethereum recently had a major upgrade on the way gas was determined, with the London fork. Prior to this, gas was simply the multiplication of the gas price and the gas limit, which itself is the limit of gas units the person making the transaction call is willing to give. Afterwards, a miner would examine the call and after adding the data to a new block, he would receive the fees, just like the PoW protocol indicates. It should be noted that in case the gas limit does not cover the EVM’s demands for executing the operation, the transaction will not be completed, but the gas will have been consumed, causing the account to lose these funds. With the London upgrade, gas is more easily estimated and the transaction fee market is more efficient, with users making more accurate and purposeful submissions of gas limit. This fork introduced a base fee for transactions, which varies in accordance with the network demand and defines every block’s size. Block sizes used to be fixed, but now every block has a target of 15 million gas. Observing the size of the previous block, a prediction is made and its base fee changes, proportionally to the overshooting or undershooting of the target, making long-term overshooting not economically viable. After the mining of a new block, the base fee is “burned” and removed from circulation, removing the possibility of mining empty blocks for profit, but also suggesting that now the only sources of income for miners, are the priority fees that users give. These “tips” must also be determined and their price will be altered, accordingly to competition. Finally, gas limit is replaced by max fee - in which base fee and priority tip are taken into account – and after the mining process, any difference between the total cost and the max fee is refunded to the transaction sender. As one can imagine, the newest upgrade has made predicting the max fee significantly easier and wallets that support it may even make recommendations.

Upgrades however, were not the only incentives that led to some of Ethereum’s hard forks. Back in 2016, a decentralized autonomous organization (DAO) raised 150 million USD worth of Ether and was afterwards hacked, posing a great threat to the entire system [16]. The DAO was a cooperative collectively owned by its members, who acted as investor-stakeholders. DAO tokens were distributed to different entities seeking investment in exchange for ETH, meaning the owners could benefit from dividends, just like ordinary investors, but also through the increase of the token price with the augmented participation of successful companies. Vulnerabilities in DAO’s code however, allowed certain individuals to drain the DAO’s wallet smart contracts, obtaining approximately 60 million USD, at a time when the DAO contracts’ balances consisted of 14% of all ETH in circulation at the time. After a long debate on how they would respond to this attack, the community decided to execute a hard fork, meaning the chain would return to the stage before the investments, thus returning all contracts’ balances to the state they were back then. Certain members however refused to take part in this process, considering such an intervention a violation of Ethereum’s principle values and continued building on the original branch, resulting to a current competition between two operating Ethereum chains. Ethereum Classic (ETC) as it is called did not follow the fork, which meant the attackers could profit from their Ether gains, since the attack was not erased from the blockchain’s memory.

Generally, Dapps present an alternative on the way people carry out everyday activities on the Internet, however a comparison between Ethereum and the modern day Internet is not easy. In contrast to web2, the current form of the World Wide Web, Ethereum introduced web3, as the network of applications running in the blockchain. Essentially, web3 is a collection of libraries that allow one to interact with a local or remote node using HTTP, IPC or WebSocket [17]. Developers have strong incentives to build their projects in web3. First and most important, is the ability to program anything they wish, since Ethreum is turing-complete. Secondly, the blockchain is permissionless, which means that all members can use a service freely, without anyone blocking them or denying them access. Moreover, all payments, including the deployment, don't require personal data and can be completed with the network's native token, ether. There are also the benefits of a decentralized system, meaning that tech giants that build apps in it, do not have to maintain a server system that could also go down, since all computers that operate as the EVM, provide the computational resources. Apart from these benefits of decentralization though, including the ones that were mentioned in previous sections, operating on a classic, centralized system also gives some advantages. Centralization offers faster processing of transactions and any changes to the state do not have to be propagated to the entire network. Also, bigger and more successful applications, would not prefer to put large portions of their code in a blockchain, since deployment is more expensive. But most importantly, the main issue with decentralized networks, is the complexity of surfing in them. In order to interact with any web3 applications, you need additional software equipment and also the knowledge of how you should proceed, in the extra steps required for accessing the Dapps' content.

Just like Bitcoin before it, Ethereum brought forth many new methods and ideas, on how interactions between the nodes of a network could be improved and this is why amongst cryptocurrencies, ether is second only to Bitcoin in market capitalization. Still, like all new technologies, it has issues that need resolving and generally, its members are trying to upgrade and optimize it constantly. Most importantly though Ethereum has to deal with competition, since new blockchains that support the execution of smart contracts, appear constantly. Its biggest "rival" right now is Cardano, an open-source blockchain platform, the first one to be founded on peer-reviewed research, which operates on Proof-of-Stake consensus protocol called Ouroboros. This means that it has none of the disadvantages of Proof-of-Work systems and this is why Ethereum is aiming to transition from PoW to PoS. This change, will be implemented along with some other major upgrades the community has planned, in order to present an entirely different version of Ethereum, which they refer to as Ethereum 2.0. The phase 0 of this new project, the Beacon chain, was already shipped in 2020 and was basically the creation of a new chain that operates on a PoS protocol and will be used to coordinate the entire network. The Beacon chain cannot handle accounts or smart contracts and at the moment is running separately from the Mainnet, but these two chains will eventually be linked in the phase 1 of the transition, called the Merge. The Beacon chain will be able to control and coordinate entities called Shard chains (not to be confused with forks), with the Mainnet being expected to be its first shard. This will signal Ethereum's transition to PoS processing of smart contracts, a change that will not affect in any other manner its operation, or its account balances, since the full history and current state of the EVM, will be brought along. The phase 1.5 will be the introduction of more Shard chains, a multi-phase upgrade to improve Ethereum's scalability and capacity [18]. These Shards will "split" the database and reduce the network's congestion, increasing the speed of transaction verification. This is a great alternative to simply augmenting the database's size, because this way there will be no great demands of computational power, in order to participate or run a client, making the system even more decentralized and therefore, secure. Initially, shards will simply help with the verification process, making it faster and smoother, but will not handle transactions on their own. Eventually though, they will be added extra functionality and will be expected to operate like the Mainnet.

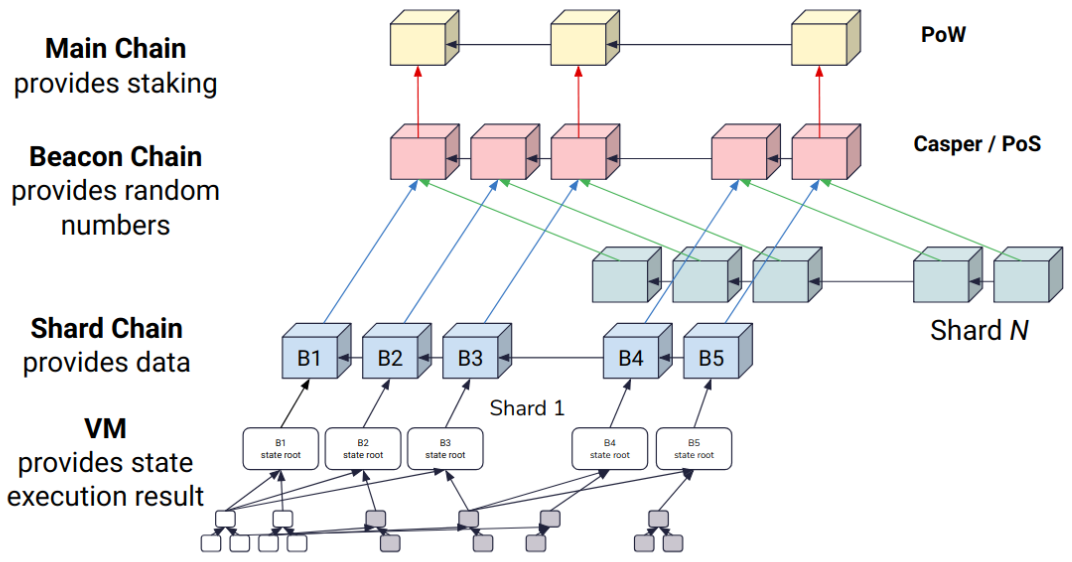


Figure 3. An overview of ETH 2.0

1.10. Energy Markets

The demands for constant and reliable energy flow are increasing worldwide. Taking the costly aspect of electricity storage into consideration, we can see how a distribution grid for multiple energy producers and consumers, is necessary for immediate energy transfer. What is equally important though, is making sure that the direct pricing configuration is beneficial for all parties. Nowadays, electricity markets are based on competition but also contain regulated agents. However, technological advances, such as smart meters and information and communication tools (ICTs), combined with the continuous installations of distributed energy resources (DERs) operating all around the distribution network, are evolving the grid from a very centralized and unidirectional flow to a bidirectional one. If we also take into consideration the increasing efforts by the global community for the integration of more renewable energy resources, aiming to tackle the climate change, it becomes clear that new challenges are currently being faced by the electricity sector. This emphasis on detaching from the conventional fuel consumption is also linked to the finiteness of the fossil fuel resources and the consequent issues that emerge on planning for long-term energy provision, but also to many countries' efforts of gaining independence from those who are in possession of the fossil fuel reserves, which are not evenly scattered around the globe.

Systems now have to deal with the unpredictable behavior of DERs, along with problems regarding grid capacity and congestion. This need to process a steady stream of information and resolving issues, leads us down the road of smart grids. The traditional approach of power systems is based on large power generators that cover the demand. The consumers, who constitute the majority of the community, have a passive role, while the system's control is coordinated by certain actors, such as system operators and generators. A smart grid is an electricity network that can intelligently integrate the actions of all users connected to it, including consumers, producers, but also prosumers, the newly defined category of consumers that also supply the grid with energy. The prosumers' activities are independent of those of traditional electricity companies, making their behavior hard to regulate and their integration to the system tricky, but their presence is becoming increasingly important. Due to their flexibility and big numbers, they can push for a system's decentralization, but they can also increase its overall share of renewable energy, since renewables are the main sources of energy exploited by small-scale producers. Energy markets are an integral part of the complex operation of smart grids, since they are a way of organizing the distribution of commodities efficiently when conditions enhance perfect competition between actors [19]. This distribution however presents some particularities, if we reckon the need for almost immediate energy consumption, following the production process. As we mentioned, storing electricity is not easy, especially in the case of large-scale networks, something that drives the system to utilize flexible mechanisms that can keep the balance between production and consumption. Energy markets are constantly evolving to deal with these challenges and assure reliable and continuous delivery of electricity, in order to maintain the system's operation as stable and smooth as possible.

Competition to energy supply, accompanied by the privatization of utilities, has led to lots of different types of energy markets, regarding their organization. Each of these types is specially designed for a specific scenario and thus they are all complementary to each other and are supposed to be combined. They usually vary depending on the system's rules and control protocols, especially regarding the regulatory bodies and their functionality. The most traditional utility model, initially chosen by most countries, is the regulated monopoly, in which all activities in the system are observed by a central authority, which is also regulating the energy flow according to a defined status. The data received, help the authority to regulate the tariff or price for electricity, but political decisions linked to irrelevant issues, could have an immediate impact on the financial results. Generally though, especially in the developed world, we are slowly moving towards smart grids and decentralized power systems that provide a more secure environment for prosumers and DERs.

Energy markets can also be categorized according to the procedure that defines the energy price. Bilateral trades are executed when a buyer reaches an agreement with a unique seller and sets up a contract for energy exchange. These agreements don't have the involvement of a third party and so bilateral contracts are formed when the two agents reach an agreement. Regardless of the kind contract though, the main aspect of these agreements is that the settled price is set independently and is agreed upon by the involved agents, something entirely different from an auction mechanism. Usually, it is not the producers or consumers that define the pricing, especially in the case of larger grids. In the course of the liberalization of the energy market and the increase of the active market players, there was a need to establish rules for energy purchase and exchange. Auction theory provided the optimal solution, describing explicit models of automated price making. There are many kinds of auctions that appear in energy markets, like the classic and well-known English or ascending-bid auction, in which the price for a commodity is increased successively, based in rounds, until eventually one bidder remains and wins the prize, but the main point is that auction procedures appear really often in most power systems. Competition is the best way to ensure the lowest price for energy dispatch and an auction has proven to be a very effective mechanism to ensure a fair and transparent process, based on a known set of rules that are determined by the market operator, but also a factor of attraction of even more market players [19]. The main parts of an auction are bidding, clearing, and pricing.

1.11. Local Energy Market

The decentralization of traditionally rigid electric power systems is leading the way towards micro and local energy markets. In the era of globalization all systems are expected to connect and be interoperable with each other, forming a set consisting of different modules which, in large scale, behaves as one giant homogenous entity, whose behavior is standardized and predictable. Apart from the need for interconnectivity though, in the case of energy systems there is also the issue of connecting and monitoring members of a growing community in the modern digital and ever-changing world.

One of the advantages of being local and small scale is fulfilling easily the preferences of consumers, some of them for example might be willing to pay more for the energy they consume if it meets their environmental preferences, or if they can trace it back to a certain producer. Most importantly though, a smaller structure makes it easier to integrate more renewable energy sources (RES), in a distributed way, in both transmission and distribution grids. This is especially important for the case of DERs, whose activities create local variations that can undermine the optimal operation of the entire network. The increasing numbers of DERs could only enhance these effects, but they can be overlooked in a local scale, since local supply variation can be matched with local demand variation, resulting in an increase of the hosting capacity of the distribution grid, without making major investments on the entirety of the network. This is the basic idea that leads to the concept of the business model of a micro or local energy market, in which a prosumer shares his excess energy in his vicinity, or an ordinary consumer purchases the surplus of energy generated locally. Unfortunately, most of existing energy markets do not provide end users and especially prosumers, the tools needed to become more active parts of the community, but what is even more absent is the framework for increased participation, on the consumers' part. The optimal energy flow and pricing can be achieved only through the involvement and engagement of everyday citizens in the market activities.

In order to create a Local Energy Community (LEC), while maintaining the reliability and security of supply that traditional, centralized power systems provide, there is a need for infrastructure investments. That means that the development of local and micro power markets, requires the existence of a new type of grid called microgrid. The electrical energy community has not come up with a single definition of the concept just yet, but it is widely accepted that a microgrid is essentially an electrical network that can connect or disconnect form the broader grid, which integrates loads and DERs and acts as a single entity, from both the grid and the market perspective [20]. One of the details that are still discussed, is the scale of the voltage and the power ratings that can define a system as a microgrid. According to professor Sumper [19], if the voltage level is less than 1000 Volt, the grid is considered to be Low Voltage (LV), while voltage in the range of 1 to 69 kV, is peculiar to Medium Voltage (MV). As for the total load and generation power range installed in a microgrid, there is a wide spectrum of possible prices, varying from a few kilowatts to hundreds of megawatts, but for the purposes of this thesis, we will only focus on LV microgrids with a total power consumption and generation, from a few kilowatts up to tens or even hundreds of kilowatts, which is the expected scale for residential communities.

As we mentioned, the microgrid will be the bedrock for the LEC and an energy market will consequently be developed. There are two distinct kinds of small-scale markets which differ in functionality but also on the parts of the microgrid they operate on. The Micro power market (μM) is a trading arena for energy products within the microgrid and can be seen as an energy management system (EMS). It utilizes some traditional market rules, to manage the DERs located within a microgrid and set some boundaries for their activities, although it can leave some room for flexibility. In case of a private ownership of the local grid, the market can be considered centralized, meaning that there is a microgrid operator regulating it. A Local power Market (LM) is essentially a trading arena existing within the LEC and we can consider two different markets of this kind, one providing flexibility services and another one focusing on energy, the latter being referred to as Local Energy Market – LEM. Both can function on a centralized manner or implement peer-to-peer (P2P) methods on their operation, but both have common objectives regarding the local community - encourage it to generate energy and create new prosumers, whose active participation in the flexible environment that is formed, will benefit all connected to the local grid. LEMs and the peer-to-peer approach to their operation, will be of most interest to us.

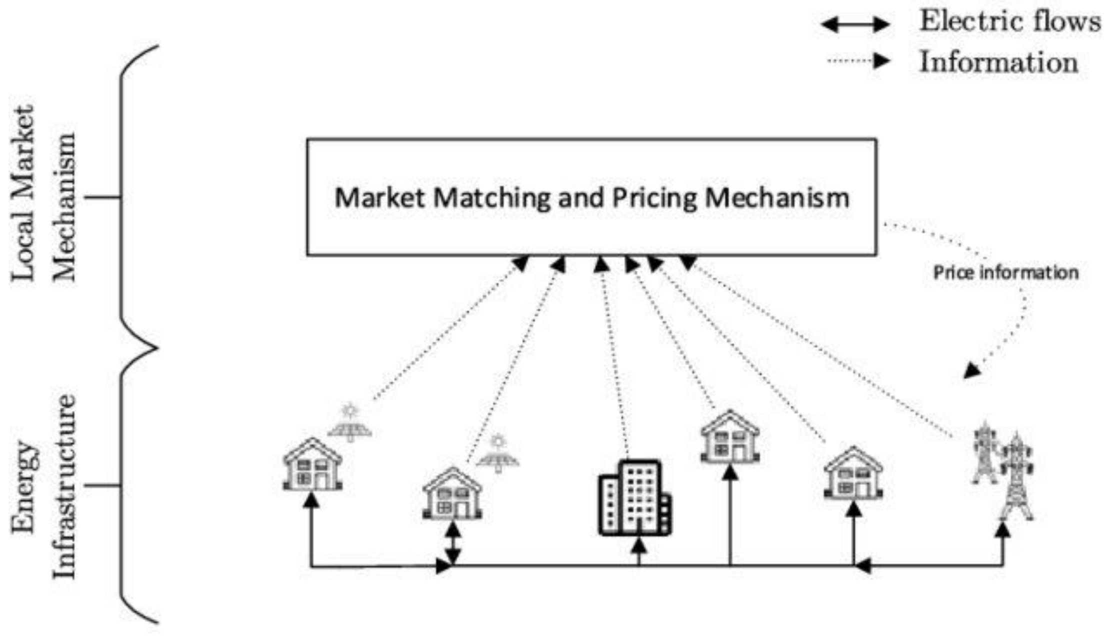


Figure 4. Framework of a local energy market based on a microgrid community

1.12. Peer-to-Peer energy trading

The global tendency towards a more decentralized society could not have left the field of energy trading unaffected. Operators and all kinds of intermediaries, monitoring the activities between consumers and producers are being increasingly considered of secondary importance for power systems, since they are no longer considered providers of security, or the financially optimal choice for a system's regulation. As DERs keep on being integrated in the grid, providing it with additional green energy and ICTs keep on advancing, an increasing number of grid members embrace the role of prosumers, thus undertaking an active behavior towards the system. In a system in which members assume more responsibility for their actions, regarding consumption, production and storage, the ideal scenario would be the development of a framework, providing total safety and transparency, for interpersonal interactions between different parties, regarding energy transaction.

This ideal design is called the Full Peer-to-Peer and is the simplest form of a P2P market [21]. Any figures of authority are non-existent and each agent is guaranteed total privacy, meaning they can share the power and price that they are willing to trade, without revealing sensible information. Other forms of P2P markets exist though, such as the Community-based market, a more traditionally formed structure, with a community manager having the duty of managing the trading activities, but also being the intermediator between his community and the rest of the system. The efficiency of this model is based on the notion that members share common interests and goals, like the preference of green energy, but the most important is the coordination and collaboration between the prosumers, even though some models describe a distinction between prosumers on different energy production classes. Naturally, there is also a combination of the 2 designs described above, the Hybrid P2P market, which has different layers for trading energy. In the higher layers, which are comprised of the prosumers that transfer the largest amounts of energy, interactions are P2P and members can even interact with existing markets. In the lowest layers though, Community-based markets are formed and the trading inside each community is monitored by a community manager, who is obviously trading on a higher layer. This also means that nested communities can be formed, forming member subsets of the higher layers.

Many ideas and concepts expressed above are yet to be implemented in real life, but as we mentioned in the previous section, smaller-scale, local markets can prove to be more flexible and deal effectively with the issues coming up, in a rapidly changing environment. Local Energy Markets in particular, are consumer-centric and can be the driving force to bring a bottom-up transformation of the entire electricity market [22]. They can support the integration of additional DERs, with no concerns for the system's balance, creating a comprehensive system of prices and charges that provides incentives for prosumers' participation, amplifying the overall inertia of the system, but also rendering useless all direct efforts of any central operator to interfere with the trade. A LEM defines a decentralized structure where all participants cooperate with what resources they have available for commons-based production, trading or distribution of goods or services, but there is no precise definition on the size of the microgrid, both on geographical and numerical sense. Right now, there are many proposals on how an energy market can reach the point of operating in such a decentralized and balanced manner, including the idea of a blockchain-based microgrid energy market, without any central coordination.

2. Connecting with the chain

2.1. Ethereum chains

What we call Ethereum is essentially a database shared between members of a network, which confirm to certain protocols. That means that there can be multiple networks implementing these protocols, operating independently of each other. The primary public Ethereum production blockchain is called the Mainnet and it's the environment in which all actual transactions take place, meaning that there is ETH transfer with each transaction and the currency has real value. Apart from the Mainnet however, there are other similar public chains called testnets, used to test smart contract deployments or protocol upgrades. Navigating the Mainnet always requires caution, since real money are handled when accessing Dapps, which is why most projects have copies deployed to testnests that one can interact with, meaning that conditions in testnets are quite similar to the Mainnet. Naturally, ETH in these networks have no value and thus, a user can get an unlimited supply for experimentation through testnet faucets, webapps which provide the user with the amount of testnet ETH he demands. The most accurate representation of the Mainnet is Ropsten, a proof-of-work testnet, but other testnets operate on proof-of-authority consensus mechanisms, because providing incentives for mining and making it profitable, on a proof-of-work testnet, is particularly hard [23]. These testnets are comprised of Rinkeby, for those running the Geth client, Kovan, for those running OpenEthereum clients, and Görli, a testnet that works across clients.

When we refer to the blockchain, a client is an implementation of Ethereum used by a member to run a node. There are 3 types of nodes, each one managing data in a different way and having different synchronization strategies, the latter meaning that getting up to date with the EVM's state does not take the same time for each type. Full nodes, provide the most complete processing of data, since they store the entirety of the blockchain data, participate in block validation by verifying all blocks and states, and serve the network and provide data on request, which means that all information for the state can be derived from a full node. Alternatively, users can run light nodes, especially if they have low capacity devices such as embedded devices or mobile phones, which only store the blocks' header data and request everything else, and can only verify the validity of the data against the state roots located in the headers. There are also nodes used primarily by block explorers, wallet vendors and chain analytics, called archive nodes. These nodes store everything kept in a full node and build an archive of historical states, meaning that data can amount to terabytes. Running a node will benefit the entire system, especially in the case of a full node, since data processing becomes more distributed and the PoW consensus protocol is enforced, making the system more secure, reliable and balanced, but members can also benefit greatly from it. A user's own node can verify all the transactions and blocks against consensus rules by itself and thus does not have to rely completely on other nodes. Moreover, they won't have to leak their addresses and balances to random nodes in the network, since everything can be checked with their own clients. Generally speaking, running a node guarantees self-sufficiency, privacy and enables one to use Ethereum in a truly trustless manner [24].

When working on a project though, some more things need to be taken into consideration. Projects and Dapps especially in the early stages of their development, need to be tested thoroughly and on a regular basis. Similarly to how web developers initially attempt to test their projects on local servers, Dapp developers can create a local blockchain instance for much faster iteration than a public testnet. In the context of Ethereum, a network is private only if its nodes are not connected to a public one, allowing to deterministically seed the local blockchain with data, instantly mine transactions and generally debug and check smart contracts' logs easily. Any of the existing Ethereum clients, such as the ones mentioned above, can be configured to run a private network and there also are tools such as the Hardhat Network, which comes built-in with Hardhat - an Ethereum development environment for professionals, but we will only deal with tools from the Truffle Suite.

2.2. Truffle and Ganache

The Truffle Suite is basically an open source project that provides developers with a number of different tools, for simpler and more efficient smart contract deployment and Dapp creation. A private network can be created, using the development framework of the same name they have published, through Truffle Develop. Truffle Develop is a development blockchain built directly into Truffle that can be set up with a single command. After installing Truffle, all that is needed is to type the following simple command into a terminal:

```
truffle develop
```

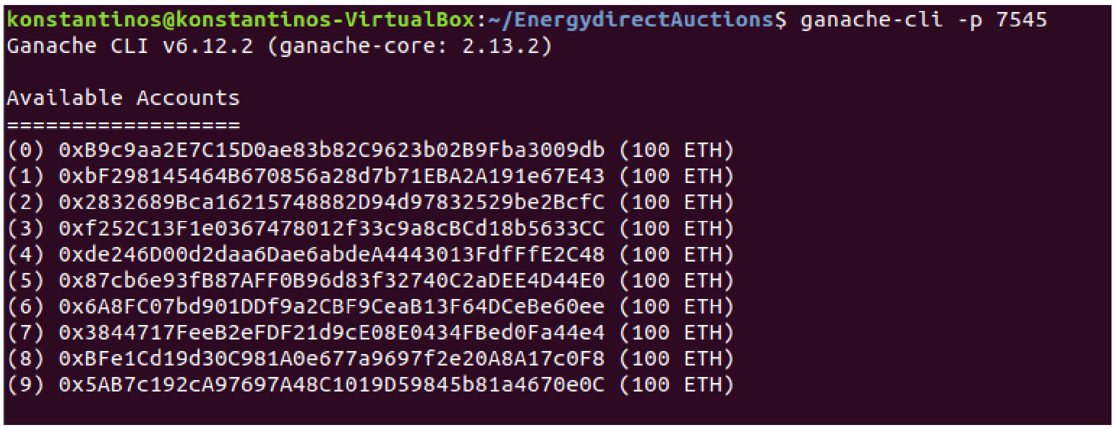
This will run the client locally, on port 9545. The first time a user runs Truffle Develop, a random mnemonic will be generated and will persist on this specific user, for all future instances. Mnemonics are collections of simple words, readable by humans, unique for each wallet and are supposed to be used as a backup in case the user needs to recover the wallet at a later date, or if he forgets/loses login information. In the case of Truffle Develop, the mnemonic can be converted to a binary seed, from which 10 random accounts are created, but it should be noted that private network mnemonics are not safe to use in public testnets or the Mainnet. Once launched, Truffle Develop will give access to the truffle console and all relative commands, which we will explore later in depth.

The Truffle Suite offers though an even better option of client for developers. Ganache-GUI, or simply Ganache, is a personal blockchain for Ethereum development that runs on desktop. After every launch, it runs locally on port 7545, although the port and host IP can be changed manually, and just like Truffle Develop, it creates 10 different accounts, with the balance of 100 ETH, that are generated through a mnemonic which can be either random or be configured manually. Through a well-designed and simple interface, the developer can quickly see the current status of all accounts, including their addresses, private keys, transactions and balances. Generally, Ganache focuses on smart contracts and their relative transactions, with a built-in block explorer that's examining all blocks and transactions, but also with the log output of the internal blockchain, which can provide useful debugging information. Ganache also offers the option to choose which Ethereum fork it has to implement and offers advanced mining controls, including gas price and limit configuration and mining time.

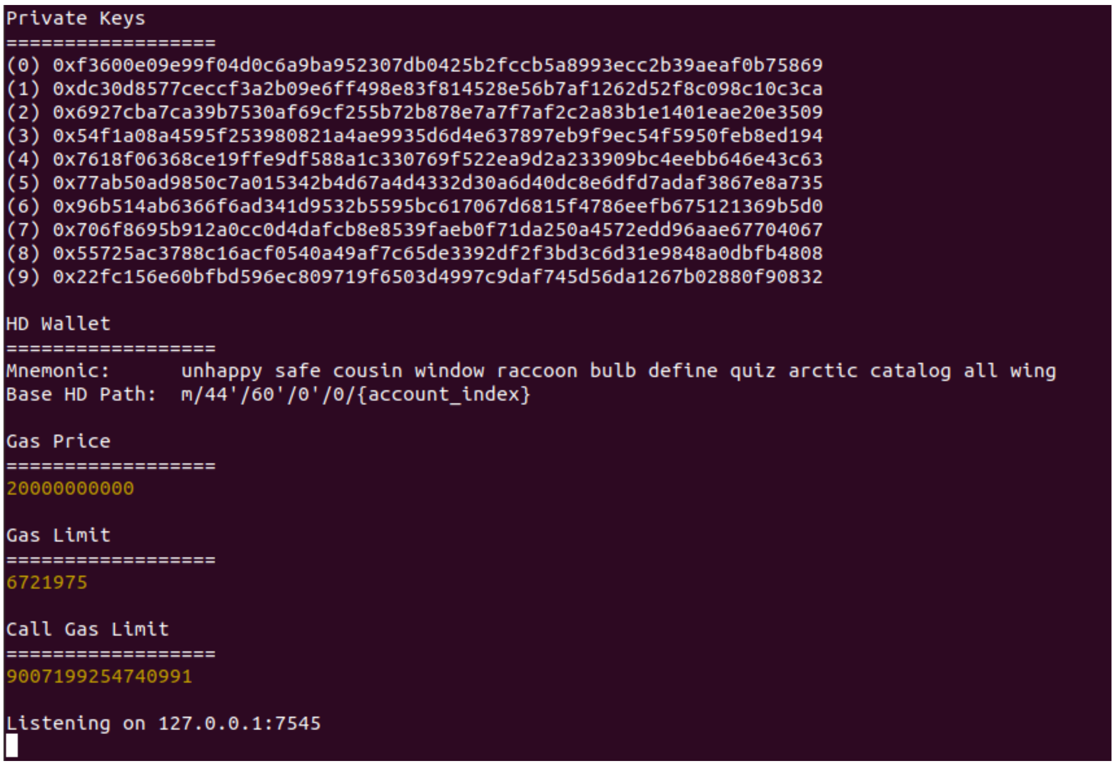
Ganache-GUI has certain issues and usually bugs out when working with oracles – the oracles' role will be explained later. For that reason in this thesis we are using Ganache-CLI, the command line version of Ganache. Ganache-CLI gives the user all the tools the desktop version provides, regarding account management, mining configuration and general blockchain and network options. It can be launched simply by typing the following simple command:

```
ganache-cli <options>
```

Whereas <options> can be any kind of option regarding the blockchain [25]. After the launch and the contract deployment, on the specific private network Ganache has created, the terminal will display a stream of information regarding the transactions and mining.



```
konstantinos@konstantinos-VirtualBox:~/EnergydirectAuctions$ ganache-cli -p 7545
Ganache CLI v6.12.2 (ganache-core: 2.13.2)

Available Accounts
==================
(0) 0xB9c9aa2E7C15D0ae83b82C9623b02B9Fba3009db (100 ETH)
(1) 0xbF298145464B670856a28d7b71EBA2A191e67E43 (100 ETH)
(2) 0x2832689Bca16215748882D94d97832529be2BcfC (100 ETH)
(3) 0xf252C13F1e0367478012f33c9a8cBCd18b5633CC (100 ETH)
(4) 0xde246D00d2daa6Dae6abdeA4443013FdfFfE2C48 (100 ETH)
(5) 0x87cb6e93fB87AFF0B96d83f32740C2aDEE4D44E0 (100 ETH)
(6) 0x6A8FC07bd901DDf9a2CBF9CeaB13F64DCeBe60ee (100 ETH)
(7) 0x3844717FeeB2eFDF21d9cE08E0434FBed0Fa44e4 (100 ETH)
(8) 0xBFe1Cd19d30C981A0e677a9697f2e20A8A17c0F8 (100 ETH)
(9) 0x5AB7c192cA97697A48C1019D59845b81a4670e0C (100 ETH)
```



```
Private Keys
==================
(0) 0xf3600e09e99f04d0c6a9ba952307db0425b2fccb5a8993ecc2b39aeaf0b75869
(1) 0xdc30d8577ceccf3a2b09e6ff498e83f814528e56b7af1262d52f8c098c10c3ca
(2) 0x6927cba7ca39b7530af69cf255b72b878e7a7f7af2c2a83b1e1401eae20e3509
(3) 0x54f1a08a4595f253980821a4ae9935d6d4e637897eb9f9ec54f5950feb8ed194
(4) 0x7618f06368ce19ffe9df588a1c330769f522ea9d2a233909bc4eebb646e43c63
(5) 0x77ab50ad9850c7a015342b4d67a4d4332d30a6d40dc8e6dfd7adaf3867e8a735
(6) 0x96b514ab6366f6ad341d9532b5595bc617067d6815f4786eefb675121369b5d0
(7) 0x706f8695b912a0cc0d4dafcb8e8539faeb0f71da250a4572edd96aae67704067
(8) 0x55725ac3788c16acf0540a49af7c65de3392df2f3bd3c6d31e9848a0dbfb4808
(9) 0x22fc156e60bfbd596ec809719f6503d4997c9daf745d56da1267b02880f90832

HD Wallet
==================
Mnemonic:      unhappy safe cousin window raccoon bulb define quiz arctic catalog all wing
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
==================
20000000000

Gas Limit
==================
6721975

Call Gas Limit
==================
9007199254740991

Listening on 127.0.0.1:7545
```

Figure 5. Running the client and the first pop-up information

3. Oracles

3.1. Connecting with the outside world

As we already mentioned, a blockchain is an isolated system. Since smart contracts though are supposed to be the basis for the creation of applications, which should be able to manage and process information from the entire world, there is the issue of accessing data so that web3 can truly be the environment to host all kinds of activities that take place on systems like the Internet. Oracles are the tools used by smart contracts to contact the resources outside the blockchain (off-chain), like data sources, legacy systems, and advanced computations, useful particularly when the contracts are connected to real-world events. They essentially act like application programming interfaces (API) to the world outside the blockchain [26]. These kinds of oracles are called inbound and they are the most frequently used ones, but after a two-way line of communication is established, it is of course possible to transmit data off-chain and we refer to the oracles performing these actions as outbound.

Obviously, this connection with the outside world constitutes a major exception to the fundamentals of the blockchain philosophy and thus, some major security concerns regarding the use of oracles have emerged. Oracles are entities controlled in a centralized manner by independent third parties and their use requires a certain amount of trust by the contract participants. Moreover, there is always the danger of the oracle being compromised and consequently compromising the entire blockchain, by channeling all kinds of altered data. The oracle problem as it's called, could be solved through the use of decentralized oracles, which implement a system that isn't controlled by a single entity and can achieve remarkable security results by distributing trust among many different data sources, just like a blockchain does.

3.2. Provable

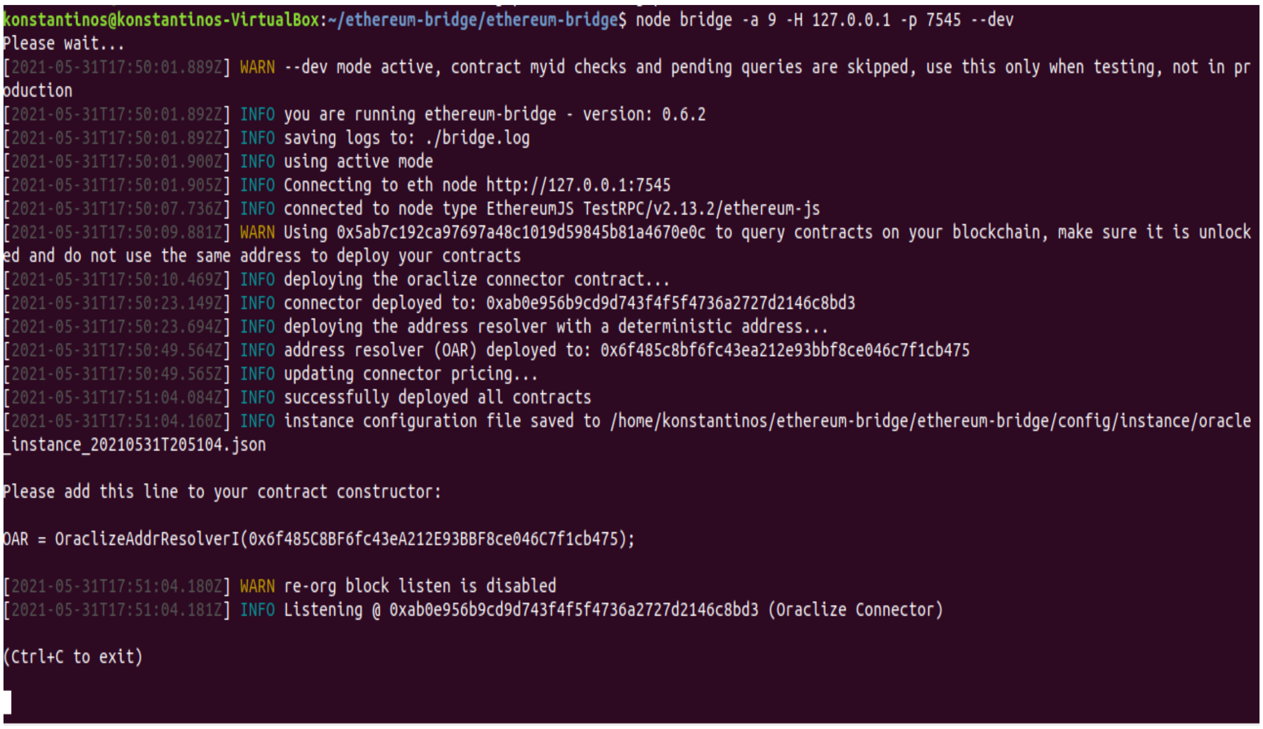
For this thesis, we are in need of an oracle to offer us a constant stream of data regarding the price of Ether (ETH), but also to provide us some services regarding the implementation of time windows. We will be using the Provable oracle, formerly known as Oraclize. Provable serves thousands of requests every day on platforms like Ethereum, Rootstock, R3 Corda, Hyperledger Fabric and EOS, and provides access to different types of data sources, such as URL, WolframAlpha, IPFS, random (untampered random bytes coming from a secure application running on a Ledger Nano S) and computational results. Any data requests to Provable need to indicate the data type, optionally provide an authenticity proof and finally, form a specific command containing all necessary parameters called a query [27].

Provable also offers a helpful tool that is specifically designed to interact with private blockchain instances, using the Node.js runtime environment, called Ethereum-bridge. There is the option to either use it on the default active mode, meaning that contracts are deployed using a specified account, in the port in which the blockchain is deployed, or use it on broadcast mode, which generates a new address locally and deploys contracts, while transactions are broadcasted to the blockchain node. Generally, an account is needed to provide the "funds" for the initial deployment of the service, just as if it were a smart contract, and also all actions Ethereum-bridge is performing, are considered transactions which produce transaction logs. This service, is constantly contacting the node we are running and the continuous requests it makes, is the reason the desktop version of Ganache would bug out and we had to resort to the simpler command line version of Ganache-CLI. This Provable oracle contract will be deployed to another specific address that needs to be different from the contracts', which is not of great concern since we are working locally, and the contracts we want to deploy as developers, need to include this oracle address in their own code. When running the Node.js command that will launch Ethereum-bridge, we can also add optional flags that will define the oracle's functionality [28].

Every time we are in need of Provable's services, we make a request in the form of a "provable_query" command. A query is a line of code, added to a smart contract, in which we define the source for the type of data we demand, along with an argument that specifies the input we will get. All Provable-related commands are linked to functions included into a smart contract called "usingProvable.sol", which is essential for the activation of the oracle service. Additional arguments can be added to such a command, such as determining custom gas limit and gas price. If no settings are specified, Provable will use the default values of 200,000 gas and 20 GWei, but altering these values, and particularly gas limit, can be mandatory in case we want to perform a complex operation. Just like any other transaction, queries need to be mined and the fees, which will be provided through the contract's balance, depend on the execution costs of the performed operations. Provable will set those parameters accordingly as specified in the smart contract, for contract-wide settings, and in the "provable_query" function, for query-specific settings.

Another very interesting option though we are presented with, is the ability to schedule the execution of a query in a future date. A time delay, expressed in seconds, can be added as an argument in a query command and determine the point in time, in which we want this specific command to be executed. This query, which could have no arguments other than the delay, could be recursively called with the help of a function named "_callback", thus enabling us to periodically reprocess a series of operations. The "_callback" function is accessed every time we request a query and is responsible for fetching the oracle's response, but could include other commands as well, such as calling for another function that contains the original query. Consequently, a potential delay between the oracle request and the call of the "_callback" function, could be used to implement a set of activities, only for the oracle to be recalled at the end of this time window and the same series of actions to take place repeatedly. So, besides simply arranging a query for the future, we are able to create a time loop, which could either repeat indefinitely, or have a specific amount of iterations.

Every call of the "provable_query" request returns a unique ID referred to as "queryId" which is a parameter of the "_callback" function. Verifying the "queryId" was generated by a valid call, ensures the response was not processed multiple times and that helps avoid misuse of the smart contract logic. Thanks to this identification, we can also be provided with authenticity proofs regarding the incoming data and their source. The authentication process requires the use of the "provable_setProof" function and a different definition of "_callback", with the addition of two more arguments including "queryId".



```
konstantinos@konstantinos-VirtualBox:~/ethereum-bridge/ethereum-bridge$ node bridge -a 9 -H 127.0.0.1 -p 7545 --dev
Please wait...
[2021-05-31T17:50:01.889Z] WARN --dev mode active, contract myid checks and pending queries are skipped, use this only when testing, not in pr
oduction
[2021-05-31T17:50:01.892Z] INFO you are running ethereum-bridge - version: 0.6.2
[2021-05-31T17:50:01.892Z] INFO saving logs to: ./bridge.log
[2021-05-31T17:50:01.900Z] INFO using active mode
[2021-05-31T17:50:01.905Z] INFO Connecting to eth node http://127.0.0.1:7545
[2021-05-31T17:50:07.736Z] INFO connected to node type EthereumJS TestRPC/v2.13.2/ethereum-js
[2021-05-31T17:50:09.881Z] WARN Using 0x5ab7c192ca97697a48c1019d59845b81a4670e0c to query contracts on your blockchain, make sure it is unlock
ed and do not use the same address to deploy your contracts
[2021-05-31T17:50:10.469Z] INFO deploying the oraclize connector contract...
[2021-05-31T17:50:23.149Z] INFO connector deployed to: 0xab0e956b9cd9d743f4f5f4736a2727d2146c8bd3
[2021-05-31T17:50:23.694Z] INFO deploying the address resolver with a deterministic address...
[2021-05-31T17:50:49.564Z] INFO address resolver (OAR) deployed to: 0x6f485c8bf6fc43ea212e93bbf8ce046c7f1cb475
[2021-05-31T17:50:49.565Z] INFO updating connector pricing...
[2021-05-31T17:51:04.084Z] INFO successfully deployed all contracts
[2021-05-31T17:51:04.160Z] INFO instance configuration file saved to /home/konstantinos/ethereum-bridge/ethereum-bridge/config/instance/oracle
_instance_20210531T205104.json

Please add this line to your contract constructor:

OAR = OraclizeAddrResolverI(0x6f485C8BF6fc43eA212E93BBF8ce046C7f1cb475);

[2021-05-31T17:51:04.180Z] WARN re-org block listen is disabled
[2021-05-31T17:51:04.181Z] INFO Listening @ 0xab0e956b9cd9d743f4f5f4736a2727d2146c8bd3 (Oraclize Connector)

(Ctrl+C to exit)
```

Figure 6. Deployment of ethereum-bridge and following information and warnings. Notice the use of the word "Oraclize" (OAR), Provable's former name

4. Truffle environment

4.1. The Truffle Framework

Like we mentioned, the Truffle Suite offers a set of different tools, suitable for smart contract developers, like the Ganache client. It is often though mixed up with the development environment of the same name, since it's what it's mostly known for. This framework, provides us with all means necessary to interact with our contracts and define the parameters of their deployment. This toolbox comes in the form of commands suited for a terminal and most of them need an existing project to be run against [29].

Truffle will interact with contracts that are placed on a "contracts" folder within the project and they need to have the .sol extension, meaning they are written in Solidity. Upon installation, Truffle will come with a solidity compiler (solc) and will compile any contracts that do not have artifacts in the "./build/contracts" directory of the project, meaning they are being compiled for the first time. These artifacts of the compilation, which are .json files, contain various types of contract-related information like the ABI, the contract bytecode and the compiler version that was used, but most importantly are necessary for the later deployment of the contracts to a network and they are utilized for this purpose when we run some JavaScript files called migrations. It should be noted that these artifacts' names do not correspond to the name of the source file, but to the name of the contract as is defined in the code.

Migrations are responsible for staging the deployment tasks, and they're written under the assumption that the deployment needs will change over time. As the project evolves, more contracts may need to be appended, or already existing files to change their role, and new migration scripts are supposed to regulate the new deployment changes. These scripts identify all smart contracts relevant to the project and their codes explicitly describe the process of development in the blockchain. Every time we run the migrations, all scripts located in the project's "migrations" directory start to be executed from the last migration that was run, running only newly created ones. Every script that is run, locates the reflected artifacts of every smart contract which will be deployed and afterwards implements the scenario the developer has in mind. In order to take advantage of the migrations feature, we need to include in the "contracts" directory a specific contract called "Migrations" that is run when we deploy contracts for the first time and is responsible for storing the number of the last deployment script applied. Deployment scripts are numbered, meaning they can be sorted ascendingly by a number, which is the first character in their file names. The Migrations contract keeps track of the sequence of completed migrations and indicates the following script to be processed. After the deployment, an artifact of "Migrations" will be created in the "./build/contracts" directory and a script called "1_initial_migration", located in the "migrations" directory, will seek this artifact to initialize the rest of the migration process. Both files "Migration.sol" and "1_initial_migration.js" can be found in the Truffle docs, but can also spawn when running initially in a project the command:

```
truffle init
```

This is the simplest way in which we can get started with a project, but Truffle provides us with lots of other options in the form of Truffle Boxes. These "boxes" are helpful boilerplates which correspond to different application themes and form the basis for the following development. They may contain libraries, modules, contracts and even fully developed Dapps which can serve as examples and inspiration. New Tuffle Boxes are being created by the Truffle community, which means there is a plethora of options for most types of projects we may wish to develop.

After the deployment, when the contracts are ready to use, Truffle offers some options on how we can interact with them, especially when we are trying to test them in a private network. The first option is the "truffle develop" command we have already mentioned, but if the user has another client like Ganache already running and has just run the migrations needed for deployment, he can use Truffle's console. The console is a JavaScript environment and can be used for testing and debugging purposes, but also to perform any transactions by hand, in case of multiple accounts in a private network for example. Of course, these console commands can also help us migrate to any live, public Ethereum network that we identify. Additionally, Truffle provides developers with the option of using external scripts to interact with the contracts, through:

```
truffle exec <path/to/file.js>
```

These external scripts may describe a specific process that would be counter-productive to be implemented through the Solidity contracts, or could be utilized to execute complex transaction scenarios at once, avoiding the procedure of executing a sequence of commands one-by-one. Recently, Truffle also added the option of running third-party, plugin commands, although this feature is new and still in a barebones state.

It's not a surprise that Truffle also comes with the web3.js package, for "surfing" the Ethereum world. The Web3 toolkit is the basis of all Ethereum activity and knowing how to use it can give the developer access to all options on interactions and configuration. It is also a very powerful tool for testing, since when we check the functionality of our contracts inside a private network, Web3 is the means for receiving data on any activities and controlling the behavior of different accounts – remember that a contract is also considered an account within a network.

Most of the choices on the options regarding details of the deployment, will be made and determined in a JavaScript file located in the root of the project, called "truffle-config.js". This particular file can for example offer the build configuration for an application, if it requires tight integration with Truffle, although most users won't need to bother with that. It is mostly used to configure network options, such as specifying which networks are available for deployment during migrations, as well as specific transaction parameters when interacting with each network (such as gas price, address used during migrations, etc.). The "development" option is the default and the one that we need when we are testing in a private network. For every network, apart from the transactions' parameters, we also determine the host IP, the port and the network ID. With this file, we can also configure the console environment, choose a preferred provider for each network and, among many other options, define the way the compilers – for each language – operate, although of course the solidity compiler is the most customizable one.

One more remarkable feature of Truffle, is the automated testing framework it comes standard with. In the project root directory, there is a "test" folder, in which we can add files whose code is supposed to imitate a smart contract scenario and quickly display the outcomes. Running tests is done typing the command:

```
truffle test
```

This is a much faster way for a developer to test the codes' logic and architecture, than compiling and deploying actual contracts to any Ethereum-type network. Moreover, the `truffle test --debug` flag and associated `debug()` global function can make the debugging process much simpler, by interrupting a test and inspecting certain operations. This is a great addition to the already existing debugging tools of Truffle, regarding the execution of smart contracts. Since we only care about the code's logic we can write tests in JavaScript and TypeScript and examine their structure and design before converting them to contracts, but we can also write Solidity tests, suited for advanced, bare-to-the-metal scenarios.

Figure 7. Ganache and Truffle Suite logos paired with the Ethereum vector in the middle

5. EnergydirectAuctions

5.1. The project

The purpose of this project is to create a Dapp that will implement the procedures that take place in a decentralized Local Energy Market. It will first create the microgrid of consumers and prosumers and afterwards receive their inputs on energy (kWh) demands and offerings, along with the declared kWh prices (differential costs). Every 15 minutes, the smart contract will process the inputs and will indicate which producer should send a certain amount of energy to cover the demand of a specific consumer, performing the financial transaction at the same time. The system prioritizes the producers' requests to make sure no excessive energy is left in the grid, since electricity cannot be easily stored and will have to be discarded, the very problem we are trying to fix. The seller with the lowest price per kWh will sell to the buyer with the best offer, performing a very simple auction per say for every producer's energy offer, repeating the process towards the sellers that make the most expensive offers, until no more transactions can occur. After the energy transfer is established, it is determined that the agreed amount will be bought by the consumer, for the price of the mean of the kWh differential costs of the two parties. The overall goal, is to distribute the energy in the most beneficial way for the entire network, while at the same time making sure that all participants in energy transfers, will have some profit.

This project was designed on an Ubuntu 64-bit operating system and its core, is the "EnergyAuction.sol" contract, which we will deploy in a private network, formed by Ganache-CLI. We also use Provable's ethereum-bridge and a contract needed for activating the oracle service called "usingProvable.sol". The oracle will be responsible for generating the 15-minute time window, while also fetching the current ETH price for every iteration.

5.2. Preparations

The first step is to install the Node.js runtime and its package manager, npm. We need to make sure we have cURL installed, or if that's not the case, open a terminal and type:

```
sudo apt install curl
```

Afterwards, we need to install nvm, a Node version manager, by running the install script through the command:

```
curl -o- https://raw.githubusercontent.com/nvm-
sh/nvm/v0.39.0/install.sh | bash
```

With nvm installed, we restart the terminal and can proceed to install Node.js. Running the "nvm" command will display all available sub-commands and will show it has been installed correctly. We get the latest version of Node.js by typing:

```
nvm install --lts
```

If we want to make sure we have the latest npm version as well we can type:

```
npm install –g npm
```

The Node.js and npm versions can be displayed with the "npm –v" and "node –v" commands.

We proceed with the global Ganache-CLI installation. For the latest stable release of ganache-cli, we run:

```
npm install ganache-cli@latest --global
```

To get ethereum-bridge, we first create a new ethereum-bridge directory. From this directory, we use npm once more to type:

```
npm install -g ethereum-bridge
```

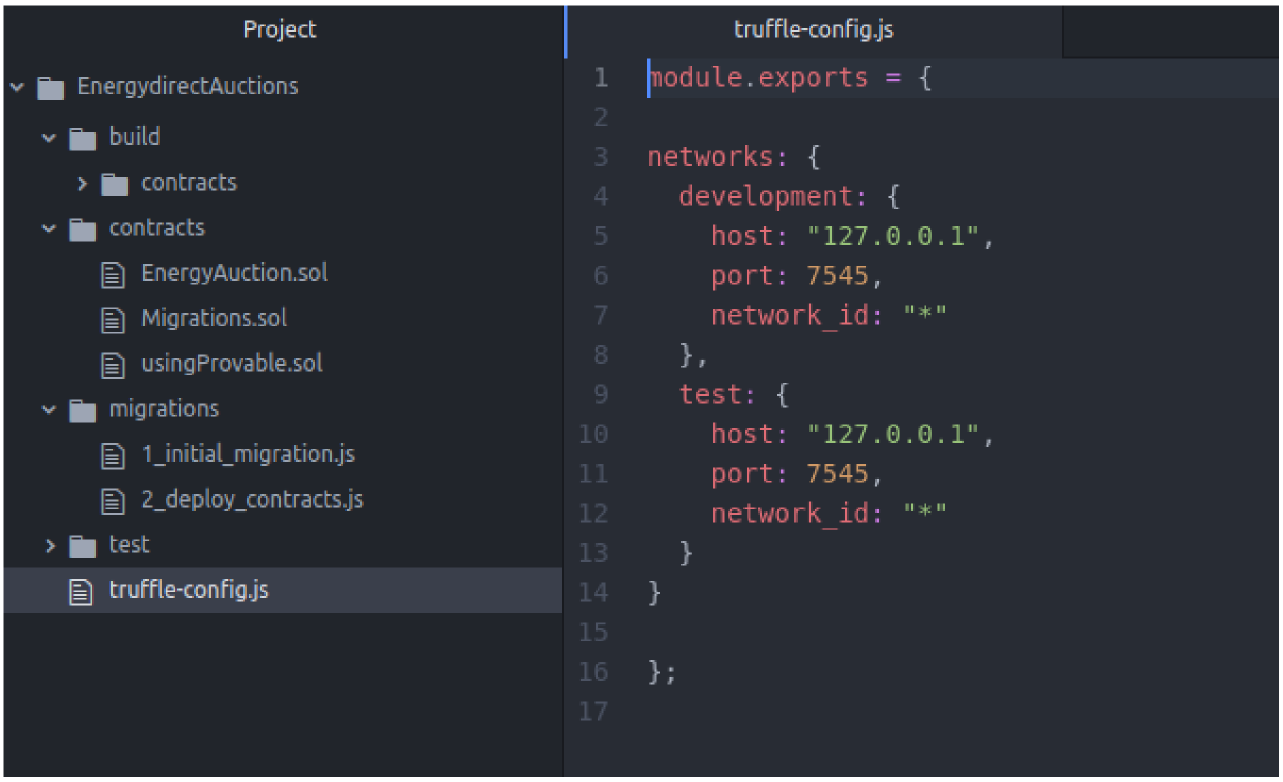
The final thing we have to install is the Truffle framework. We simply run:

```
npm install -g truffle
```

And now we begin to create the project. We make a new “EnergydirectAuctions” directory, to host our projects and when in it, we type in a terminal the command:

```
truffle init
```

This will create all the necessary directories that we see in figure 8, as well as the truffle-config.js file. We make sure to add the specifics of a development network to the configuration file, which will be hosted locally, on port 7545, that will be matched with any network ID. A test network can also be defined, but the default option of the client will be the development one. In the “contracts” directory, we add the “EnergyAuction” contract, the “usingProvable” contract, which we will import to the main one so we can use the Provable oracle, and finally the “Migrations” contract. “Migrations” can be found within the Truffle documentation and “usingProvable”, likewise, is available in Provable’s GitHub page [30]. In the “migrations” directory we have our first migration file, which we need in order to deploy the Migrations contract, and the second script that deploys the rest of the contracts.



```
module.exports = {

networks: {
  development: {
    host: "127.0.0.1",
    port: 7545,
    network_id: "*"
  },
  test: {
    host: "127.0.0.1",
    port: 7545,
    network_id: "*"
  }
}


};

```

Figure 8. The project directory and the configuration file

5.3. Running

We open a terminal and the first thing we do is run our Ganache-CLI client by typing:

```
ganache-cli -p 7545
```

This will create a local network on port 7545, as we specified, with 10 accounts of 100 ETH balance, but also with specific, though configurable, gas limit, gas price etc, as we can see in figure 5. We are also presented with each account's public and private keys, along with the mnemonic that produces these accounts.

Afterwards, we need to prepare our oracle. Opening a second terminal, we move to the "~/ethereum-bridge/ethereum-bridge/ directory and using node.js, we type the command we have shown in figure 6:

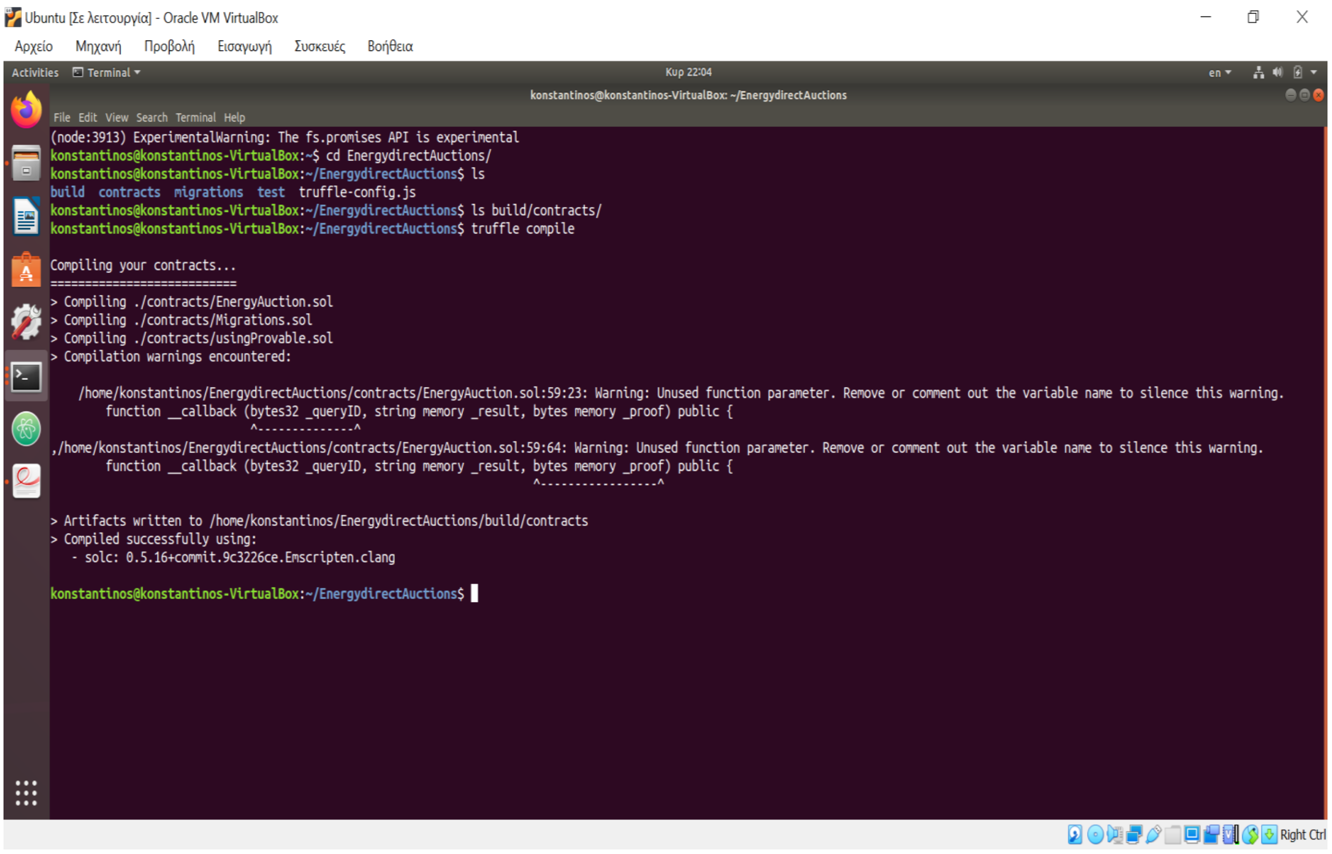
```
node bridge -a 9 -H 127.0.0.1 -p 7545 --dev
```

With this command, we tell the oracle to seek the network we have deployed locally, on port 7545, but also to skip some checks and queries, because we are in development mode, meaning we are simply testing. We also determine that the 10th of the created accounts (numbering from 0-9) will be the one to funnel the funds for the "mining" of the ethereum-bridge contracts.

Warning - at this moment, the "./build/contracts/" directory will have the ethereum-bridge artifacts, but it is advised to make sure this folder is clear before running the Ganache-CLI client, due to some bugs that may occasionally occur.

The only thing that's left is deploying our contracts. First, we open a third terminal, get to "~/EnergydirectAuctions" directory and compile them running:

```
truffle compile
```


Ubuntu [Σε λειτουργία] - Oracle VM VirtualBox
Αρχείο Μηχανή Προβολή Εισαγωγή Συσκευές Βοήθεια
Activities Terminal
Κυρ 22:04
konstantinos@konstantinos-VirtualBox: ~/EnergydirectAuctions
File Edit View Search Terminal Help
Right Ctrl

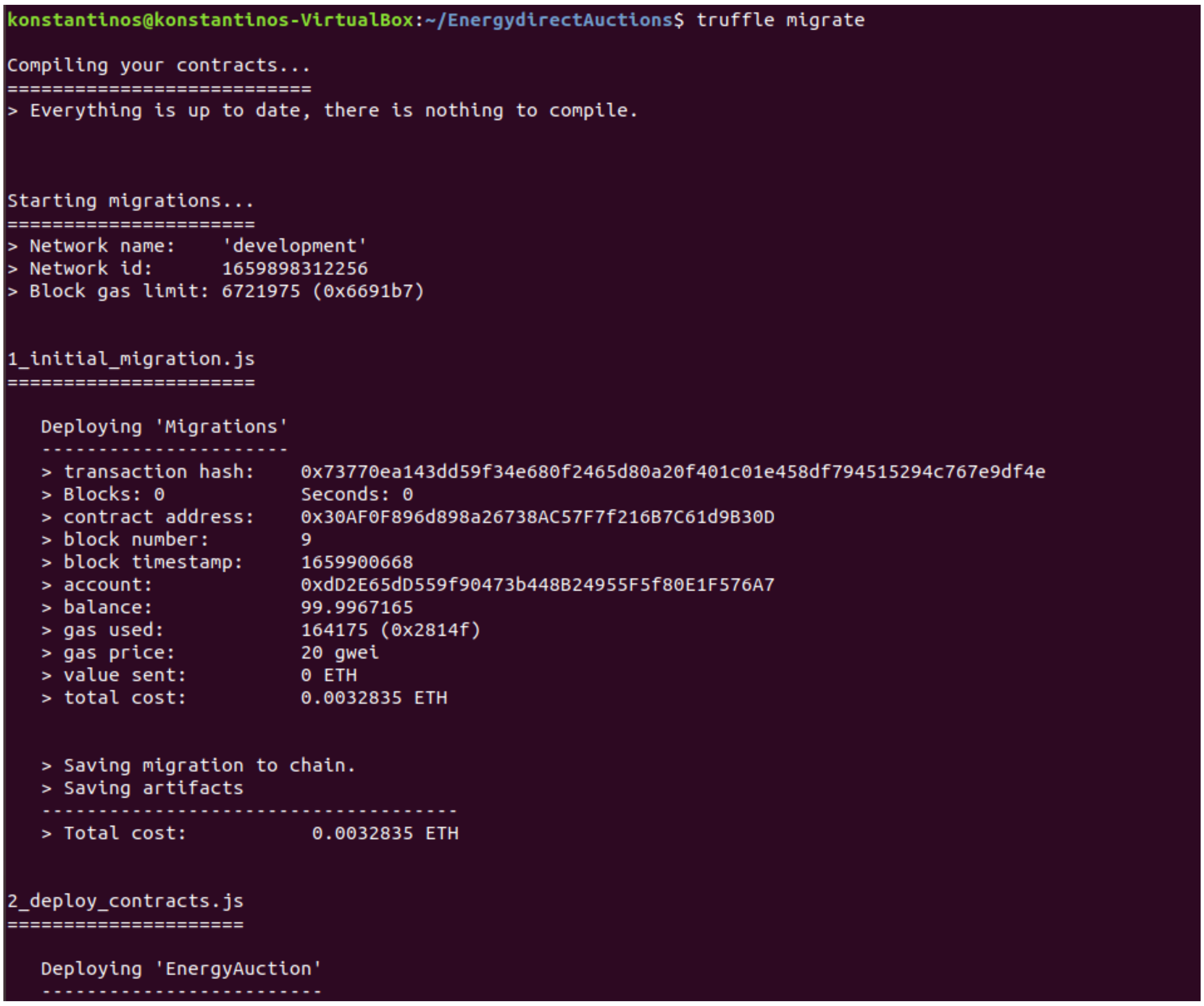
```
(node:3913) ExperimentalWarning: The fs.promises API is experimental
konstantinos@konstantinos-VirtualBox:~$ cd EnergydirectAuctions/
konstantinos@konstantinos-VirtualBox:~/EnergydirectAuctions$ ls
build  contracts  migrations  test  truffle-config.js
konstantinos@konstantinos-VirtualBox:~/EnergydirectAuctions$ ls build/contracts/
konstantinos@konstantinos-VirtualBox:~/EnergydirectAuctions$ truffle compile

Compiling your contracts...
===========================
> Compiling ./contracts/EnergyAuction.sol
> Compiling ./contracts/Migrations.sol
> Compiling ./contracts/usingProvable.sol
> Compilation warnings encountered:

    /home/konstantinos/EnergydirectAuctions/contracts/EnergyAuction.sol:59:23: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
        function __callback (bytes32 _queryID, string memory _result, bytes memory _proof) public {
                             ^-------------^
,/home/konstantinos/EnergydirectAuctions/contracts/EnergyAuction.sol:59:64: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
        function __callback (bytes32 _queryID, string memory _result, bytes memory _proof) public {
                                                                      ^----------------^

> Artifacts written to /home/konstantinos/EnergydirectAuctions/build/contracts
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang

konstantinos@konstantinos-VirtualBox:~/EnergydirectAuctions$ 
```

Figure 9. Initial commands and compilation

It will be noticed there is a warning for two unused parameters on one of EnergyAuction's functions, but these parameters are needed, since this particular function communicates with the oracle and it needs to include them as arguments to be recognized.

The compilation of all three of our contracts is successful and we move on with the deployment:

```
truffle migrate
```



```
konstantinos@konstantinos-VirtualBox:~/EnergydirectAuctions$ truffle migrate

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



Starting migrations...
======================
> Network name:    'development'
> Network id:      1659898312256
> Block gas limit: 6721975 (0x6691b7)


1_initial_migration.js
======================

   Deploying 'Migrations'
   ----------------------
   > transaction hash:    0x73770ea143dd59f34e680f2465d80a20f401c01e458df794515294c767e9df4e
   > Blocks: 0            Seconds: 0
   > contract address:    0x30AF0F896d898a26738AC57F7f216B7C61d9B30D
   > block number:        9
   > block timestamp:     1659900668
   > account:             0xdD2E65dD559f90473b448B24955F5f80E1F576A7
   > balance:             99.9967165
   > gas used:            164175 (0x2814f)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.0032835 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:           0.0032835 ETH


2_deploy_contracts.js
=====================

   Deploying 'EnergyAuction'
   -------------------------
```

Figure 10. Migration command and execution of migration scripts

The first thing this command does, is locate the existing contract artifacts in the ".build/contracts" directory and afterwards check if the respective contracts have been modified in any way. Since they have not, there is no need for any additional compilation. Afterwards, Truffle checks the first file in the "./migrations" directory, which is "1_initial_migration.js", which indicates the "Migrations.sol" file as the first contract to be deployed. Since this contract is deployed in the blockchain, the migration itself is considered to be a transaction and Truffle provides us with a set of relevant data. First of all, we have a transaction hash which, combined with the rest of the data, will provide us with a unique identifier. Next, we have information on the amount of blocks and the time required for the deployment. Following details are the contract's address, the timestamp and the number of the block in which the transaction is stored, the address of the account that handles the deployment and its consequent balance, after the mining, the value of ETH that was sent to the contract's balance and finally, the amount of gas that was used, its price and the total cost of the deployment. When we run the "Migrations" contract, the rest of our deployment scripts are ready to be executed and migrate the rest of our smart contracts. The second script that is run is "2_deploy_contracts.js" and our main smart contract called "EnergyAuction.sol" gets deployed consecutively.

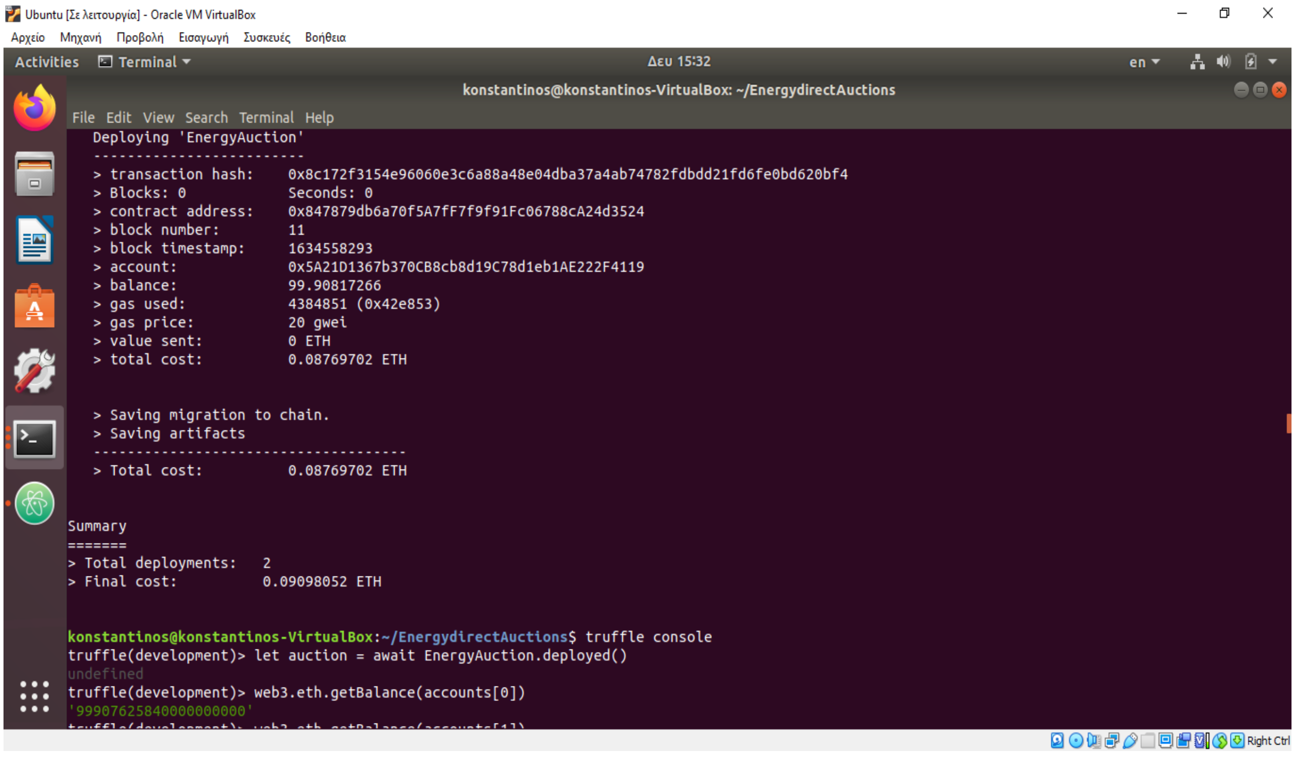
The mining cost is handled by default by the first account (account no 0), although that is also configurable. The first account will play the role of the Dapp developer and deployer in general, who needs to have the profit he's aiming for, especially since he was charged with the mining expenses - expenses that amount to 0.09098052 ETH. As it will be noticed later though, this account seems to have spent more ETH than the total migration cost. The reason for this, is the automatic mining of the "constructor" functions in all three contracts, immediately after they are deployed. Smart contracts don't have to include the "constructor" function, but when they do, it is the very first contract function to be called and its mining is handled by the default miner. Keep in mind that these figures are based on the gas price and gas limit that Ganache determined initially.

5.4. Interacting with the contract and its functions

In order to interact with the EnergyAuction contract we need to call on its functions, so that we can dictate which specific tasks we want it to perform. Some functions are private or internal, meaning they are not visible from external users, but public ones can be accessible to all, although additional identity checks can be implemented, as we will see. In any case, to perform such actions we need to use the console, which we get when we type:

```
truffle console
```

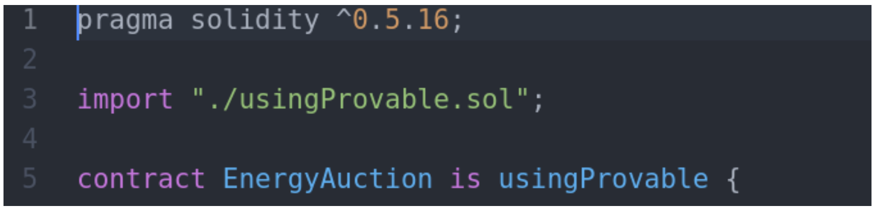
The console will only allow us to communicate with an instance of the contract and that means that to get one, we have to make an asynchronous call, like the one in figure 11.


Ubuntu [Σε λειτουργία] - Oracle VM VirtualBox
Αρχείο Μηχανή Προβολή Εισαγωγή Συσκευές Βοήθεια
Activities Terminal
Δευ 15:32
en
konstantinos@konstantinos-VirtualBox: ~/EnergydirectAuctions
File Edit View Search Terminal Help
Right Ctrl

```
   Deploying 'EnergyAuction'
   -------------------------
   > transaction hash:    0x8c172f3154e96060e3c6a88a48e04dba37a4ab74782fdbdd21fd6fe0bd620bf4
   > Blocks: 0            Seconds: 0
   > contract address:    0x847879db6a70f5A7fF7f9f91Fc06788cA24d3524
   > block number:        11
   > block timestamp:     1634558293
   > account:             0x5A21D1367b370CB8cb8d19C78d1eb1AE222F4119
   > balance:             99.90817266
   > gas used:            4384851 (0x42e853)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.08769702 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.08769702 ETH


Summary
=======
> Total deployments:   2
> Final cost:          0.09098052 ETH


konstantinos@konstantinos-VirtualBox:~/EnergydirectAuctions$ truffle console
truffle(development)> let auction = await EnergyAuction.deployed()
undefined
truffle(development)> web3.eth.getBalance(accounts[0])
'99907625840000000000'
```

Figure 11. The final migration screen, the deployment of the console, the "auction" instance of the contract and a Web3 command showing us the balance of the first account

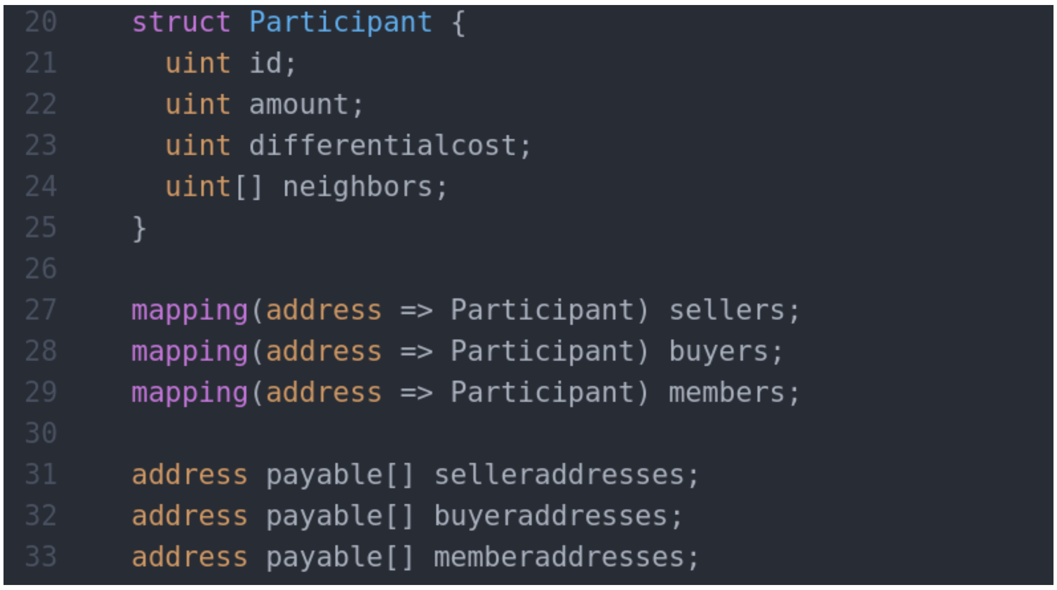
We have identified the instance as "auction" and afterwards made a Web3 call to fetch us the balance of account no 0. In the above figure we can see that the remaining funds are less than 100 ETH (the initial balance) minus the final cost of all deployment expenses. As we mentioned, the reason for this is the mining of the "constructor" functions.



```solidity
pragma solidity ^0.5.16;

import "./usingProvable.sol";

contract EnergyAuction is usingProvable {
```

Figure 12. First lines of code in "EnergyAuction.sol"

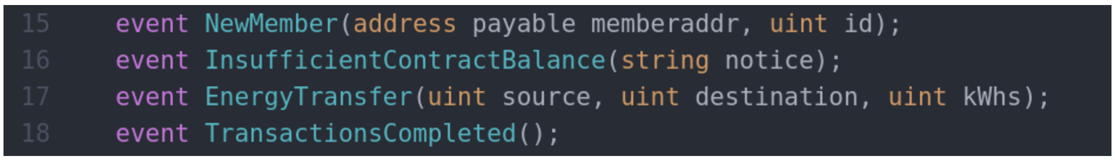
The contract itself is simple in its concept. At its start the solidity version used is defined and the "usingProvable.sol" contract is imported. It is important to use a 0.5.x solidity version, since at the time this thesis is written the "usingProvable" contract requires one of these solidity versions to be used, for successful oracle calls. Next, we have the events' and global variables' definitions, along with the creation of a "Participant" struct, which will be used as the standard for the characteristics of all blockchain members who wish to be a part of the LEM. Every Participant will have a unique id, an array indicating the neighboring Participants, the amount of energy in kWhs that are to be transferred and the differential cost of those kWhs. We will also bind each member address to a specific instance of the "Participant" struct, so the Id will refer to a unique account.



```solidity
    struct Participant {
      uint id;
      uint amount;
      uint differentialcost;
      uint[] neighbors;
    }

    mapping(address => Participant) sellers;
    mapping(address => Participant) buyers;
    mapping(address => Participant) members;

    address payable[] selleraddresses;
    address payable[] buyeraddresses;
    address payable[] memberaddresses;
```

Figure 13. "Participant" and addresses declaration

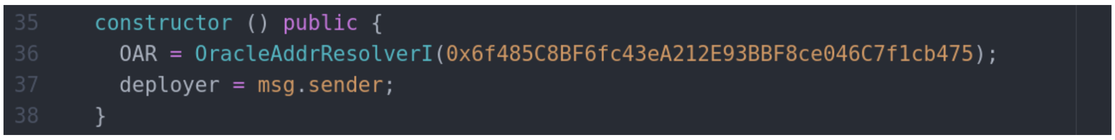
We also proceed to declare some Events. We can understand Events as data storages ready to transmit their information to all members of the blockchain. Every time there is a specific occurrence we want to notify the network about, we run some code with the keyword "emit" alongside the Event's name and we store the relevant information inside the blockchain. Afterwards, the nodes can listen to such Events using the console in a way we will explain below.



```
    event NewMember(address payable memberaddr, uint id);
    event InsufficientContractBalance(string notice);
    event EnergyTransfer(uint source, uint destination, uint kWhs);
    event TransactionsCompleted();
```

Figure 14. Event declaration

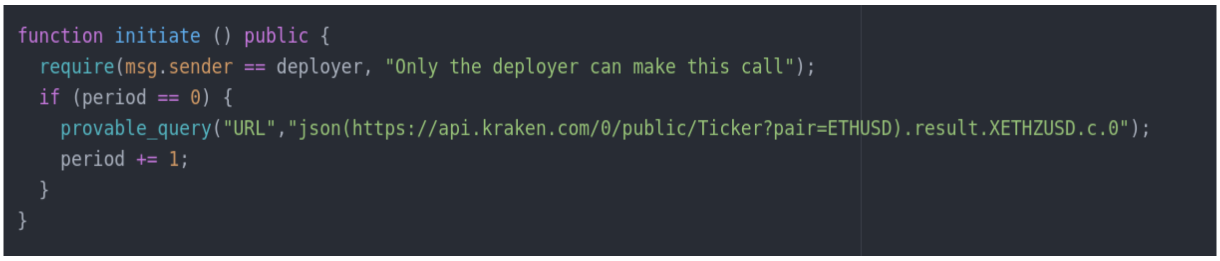
Inside the constructor, which is the first function that is run when the contract is deployed, we identify the first account as the deployer and we follow the instruction we had in figure 6 regarding the OAR.



```
    constructor () public {
      OAR = OracleAddrResolverI(0x6f485C8BF6fc43eA212E93BBF8ce046C7f1cb475);
      deployer = msg.sender;
    }
```

Figure 15. “constructor” code

The 15-minute cycle will commence when an account first decides to fetch the current Ether/USD exchange and this is why there is an “initiate” function, with an entry check to ensure access only to the deployer.

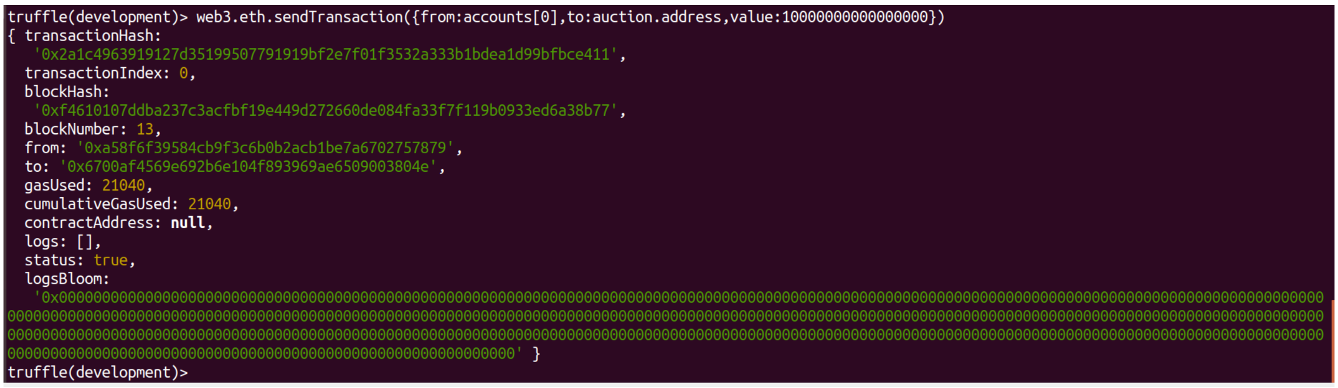


```
function initiate () public {
  require(msg.sender == deployer, "Only the deployer can make this call");
  if (period == 0) {
    provable_query("URL","json(https://api.kraken.com/0/public/Ticker?pair=ETHUSD).result.XETHZUSD.c.0");
    period += 1;
  }
}
```

Figure 16. “initiate” function

The deployer makes the first call to ethereum-bridge and Provable provides us with the tool of delaying the response, thus giving us the 15-minute loop. The first oracle call is granted for free, but all others require mining funds. Account no.9 was responsible for covering the fees for the particular case of ethereum-bridge’s launch, but the rest of the oracle’s demands can be considered as some of the LEM’s collective expenses and as such, it’s the deployer who covers the fees. Of course, this account has to profit enough from its share in each cycle’s total system profit, so it can be compensated for these additional expenses.

Running a Web3 command, we transfer the amount of 0.01 ETH into our smart contract’s balance, just the necessary amount for the call we want to make. The contract is addressed simply as “auction”, the identification that was given to its instance when we made the asynchronous call, after launching the console. When we initiate the cycle, it is the contract itself that signals the oracle query, meaning the funds will be drawn straight from the address of its own balance.

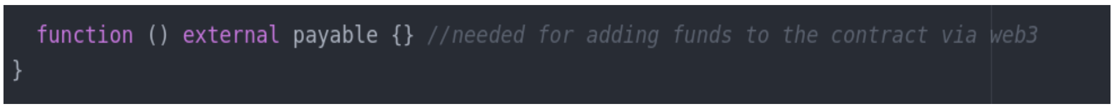


```
truffle(development)> web3.eth.sendTransaction({from:accounts[0],to:auction.address,value:10000000000000000})
{ transactionHash:
   '0x2a1c4963919127d35199507791919bf2e7f01f3532a333b1bdea1d99bfbce411',
  transactionIndex: 0,
  blockHash:
   '0xf4610107ddba237c3acfbf19e449d272660de084fa33f7f119b0933ed6a38b77',
  blockNumber: 13,
  from: '0xa58f6f39584cb9f3c6b0b2acb1be7a6702757879',
  to: '0x6700af4569e692b6e104f893969ae6509003804e',
  gasUsed: 21040,
  cumulativeGasUsed: 21040,
  contractAddress: null,
  logs: [],
  status: true,
  logsBloom:
   '0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000' }
truffle(development)>
```

*Figure 17. Account no. 0 transfers gas price * gas limit amount of Wei to the contract's balance*

As we see in the figure above, a series of data appears after we run this command. We are presented with a transaction hash which, combined with the rest of the data, will provide us with a unique identifier for this money transfer, but we also get an index showing us that this is the very first of a series of - potential - transactions which can be analyzed in these logs. Following details are the addresses of the sender and receiver accounts, the number of the block in which the transaction is stored and the block's hash, the amount of gas that was used along with additional gas expenses that can arise from consequent activities, a NULL indication regarding the contract's address since it is not a contract making the call and a TRUE status confirming the transaction was completed. Additional information can also be examined if we explore the details in the "logs[]" array.

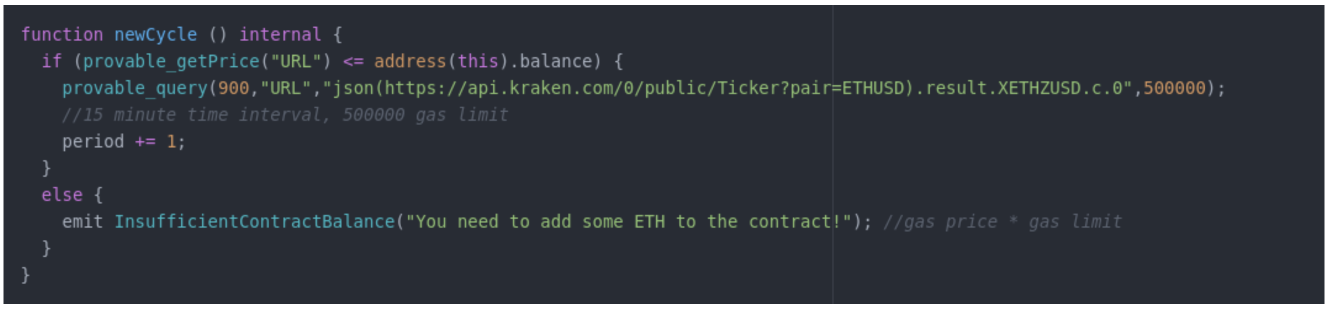
Operating the loop was a major issue, because when we access a function, only one account can make the call, meaning that the mining expenses will not be shared like they should and the person taking the responsibility has to rely on out-of-blockchain agreements on being compensated. Moreover, operations can't be forward-planned easily in a blockchain, since miners cannot be expected to verify activities, on certain time schedules. This is a general issue in blockchain functionality and some solutions such as the Ethereum Alarm Clock (EAC) [31] have been proposed, but for a private network project, Provable gives us the solution. Of course, the fact that the deployer has the responsibility of maintaining the cycle, goes against the decentralized spirit of the blockchain principles. The first Provable call is always for free and on this occasion, we use it to get the ETH price and schedule the next cycle. The ETH price can be seen by all, through a free-to-call public function that simply returns the ETH price in USD. In order to schedule the next cycle, the deployer needs to send the contract the necessary funds, which are the gas price multiplied by the gas limit. Ganache defined the gas price as 20GWei, but the gas limit for an oracle call can be manually set to 500,000 by adding another parameter to the command that makes the call. The oracle will draw this fee straight from the contract balance and this is why the deployer will first send these funds to the contract through the console, using Web3, before initiating the entire procedure. The empty external function at the end of the contract, is used for this exact Web3 transaction, enabling the contract to receive funds directly.



```
  function () external payable {} //needed for adding funds to the contract via web3
}
```

Figure 18. The external function which enables direct fund transfers to the contract

Since we want this 15 minute cycle to loop indefinitely, it is needed to include a new function in the contract that creates the next cycle. This function is called “newCycle” and is labeled as internal, meaning it can be accessed only by the contract itself, following a series of procedures taking place within the contract.

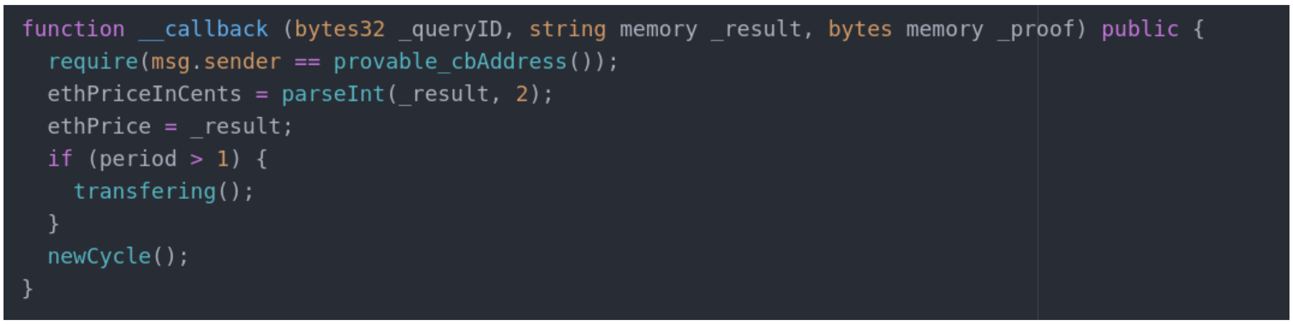


```
function newCycle () internal {
  if (provable_getPrice("URL") <= address(this).balance) {
    provable_query(900,"URL","json(https://api.kraken.com/0/public/Ticker?pair=ETHUSD).result.XETHZUSD.c.0",500000);
    //15 minute time interval, 500000 gas limit
    period += 1;
  }
  else {
    emit InsufficientContractBalance("You need to add some ETH to the contract!"); //gas price * gas limit
  }
}
```

Figure 19. “newCycle” function

This function includes a call to a provable query that comprises three arguments. The first argument is the number of seconds for which we wish to delay the return of our query so the cycle can be completed, in this case 900 seconds, equal to 15 minutes. The second one is the type of source in which we seek our data and the third one is the URL address of the ETH to USD conversion value. The final argument is the custom gas limit, set to 500,000. This new gas limit multiplied with the gas value, which is 20GWei, results in the required amount of Wei that we need to send into the contract’s balance. Since this amount, which we can see in figure 17, is necessary for a successful call, there is also a check of the contract’s balance to make sure the query can be completed. That is the reason for which the timely transferring of funds from the deployer’s account is essential for the operation of the LEM. Like we mentioned, this process is inconsistent with the decentralized nature of the project, although it can become an automated one and raise no concerns for the overall process.

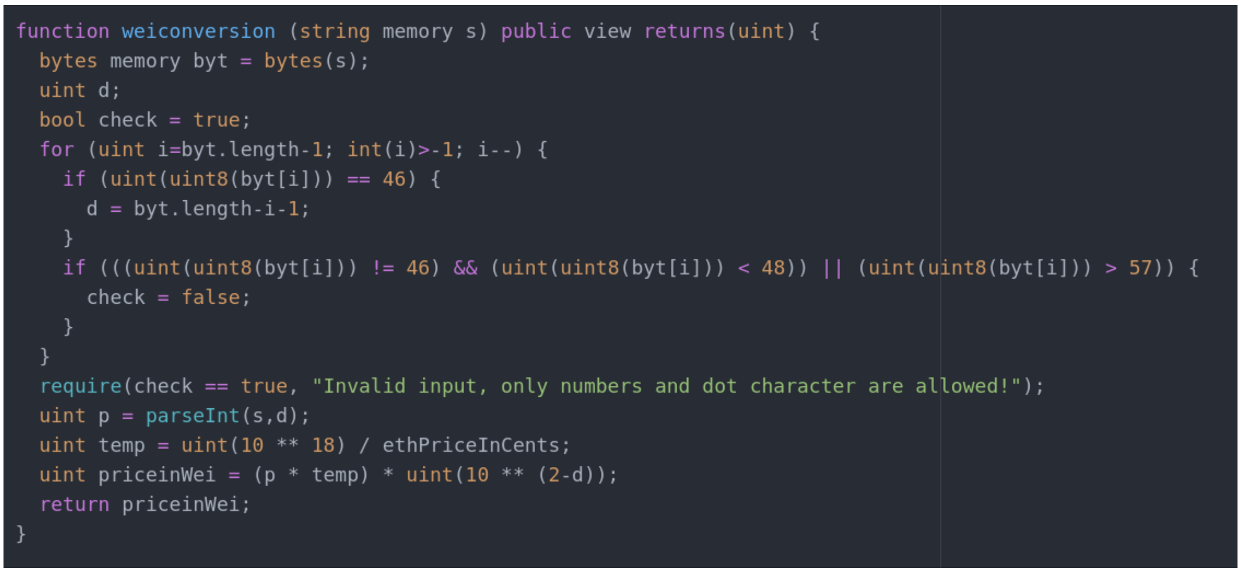
The call is scheduled to return the data in a separate function after the 15-minute delay. “ethereum-bridge” is the only entity with access to this function, something we are assured of with an identity check in the second line of its code, but only recognizes a very specific form of this function. It has to be identified as “_callback” and include three arguments to store the data the oracle will return. In our case we only make use of the “_result” string, which brings us the current ETH to USD exchange rate. This string is passed through a certain function available from the “usingProvable” contract that turns it into a “uint” variable, expressed with the precision of 2 decimal points, although we store the original string as well. After the information is fetched, a call to a “transferring” function is made that completes the procedures required for the energy transfer occurring at the end of each cycle. Finally, we initiate a new cycle, which will result in the access of “_callback” once again, after the passing of 15 minutes.



```
function __callback (bytes32 _queryID, string memory _result, bytes memory _proof) public {
  require(msg.sender == provable_cbAddress());
  ethPriceInCents = parseInt(_result, 2);
  ethPrice = _result;
  if (period > 1) {
    transfering();
  }
  newCycle();
}
```

Figure 20. "_callback" function

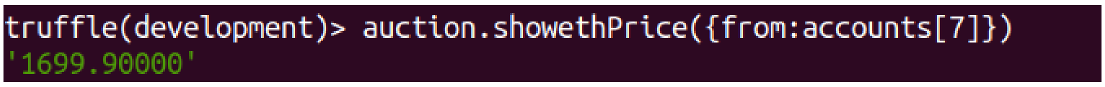
Another important function working in the background, although public, is the function the converts strings to Wei. Members of the network will obviously want to give inputs of an ordinary currency, but all financial transfers in the contract are expected to be done using Wei. This function also uses the tool from the "usingProvable" contract we utilized in "_callback" to make the process simpler. One issue that Solidity has, is that it does not make mathematical operations with decimal numerals. That means that when we have to perform some numerical division, before making a transfer, a remainder of Wei will be left on the contract's balance, although this problem is somewhat trivial, since Wei is a denomination with a really insignificant value, considering that $1\ ETH = 10^{18}\ Wei$. It also means that when members don't give an integer as an input, their only option is to present it as a string and then have this conversion function turn it into an integer (Wei). Users can input any particular string, but this function will accept only those that include exclusively numbers and the "." character.



```
function weiconversion (string memory s) public view returns(uint) {
  bytes memory byt = bytes(s);
  uint d;
  bool check = true;
  for (uint i=byt.length-1; int(i)>-1; i--) {
    if (uint(uint8(byt[i])) == 46) {
      d = byt.length-i-1;
    }
    if (((uint(uint8(byt[i])) != 46) && (uint(uint8(byt[i])) < 48)) || (uint(uint8(byt[i])) > 57)) {
      check = false;
    }
  }
  require(check == true, "Invalid input, only numbers and dot character are allowed!");
  uint p = parseInt(s,d);
  uint temp = uint(10 ** 18) / ethPriceInCents;
  uint priceinWei = (p * temp) * uint(10 ** (2-d));
  return priceinWei;
}
```

Figure 21. The "weiconversion" function

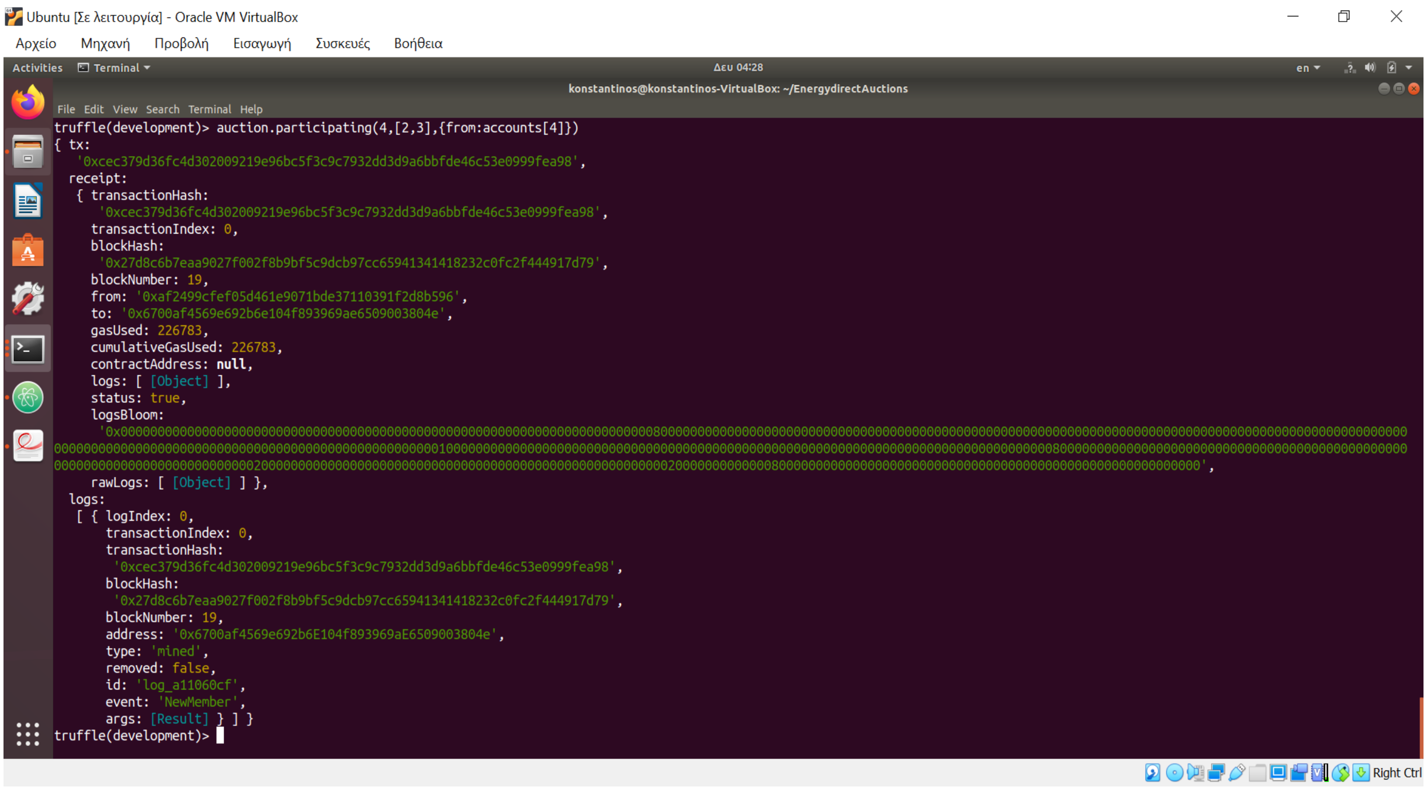
This conversion procedure may be automatic, but still it is important for all members to know the current ETH price. Generally speaking, perspective on the currency's value is useful, but this knowledge is also necessary because any direct fund transfer from member to member, has to be made in Wei. Accounts can call a specific contract function and after they learn the ETH to USD exchange, can convert the number they wish with the help of a calculator.



```
truffle(development)> auction.showethPrice({from:accounts[7]})
'1699.90000'
```

Figure 22. Calling for ETH price

The only thing that’s left is dealing with the LEM operations. Before anything else, members need to state their participation, giving a unique ID and declaring which other participants they are neighbouring with, to ensure there will be no isolated node in the network making the energy transfer impossible. This is a mandatory procedure for someone to either offer or buy energy in any particular cycle and when it’s completed, the new member’s input will be stored in an instance of a structure called simply “members”, which is based on the “Participant” struct. Each instance’s data is bound to a specific address that is appended to an array called “memberaddresses” as a new element. “memberaddresses” is the array of all members’ addresses and various checks have been included in the function in order to make sure that every account will be tied to a single, unique Id. Using this list of addresses on the “members” struct, our contract can find the relevant data for every account that is part of the community. Members also have the option to alter this data, as long as they don’t appropriate another’s Id.

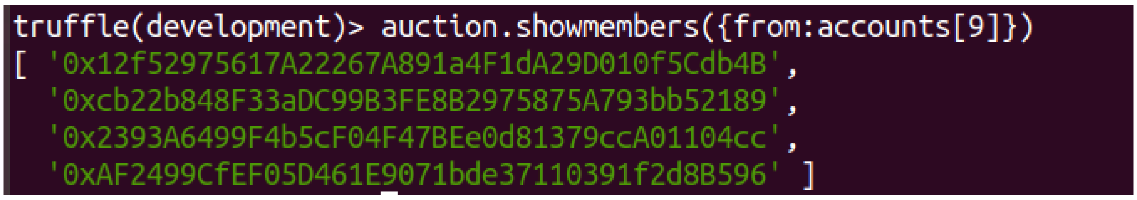


```
truffle(development)> auction.participating(4,[2,3],{from:accounts[4]})
{ tx:
   '0xcec379d36fc4d302009219e96bc5f3c9c7932dd3d9a6bbfde46c53e0999fea98',
  receipt:
   { transactionHash:
      '0xcec379d36fc4d302009219e96bc5f3c9c7932dd3d9a6bbfde46c53e0999fea98',
     transactionIndex: 0,
     blockHash:
      '0x27d8c6b7eaa9027f002f8b9bf5c9dcb97cc65941341418232c0fc2f444917d79',
     blockNumber: 19,
     from: '0xaf2499cfef05d461e9071bde37110391f2d8b596',
     to: '0x6700af4569e692b6e104f893969ae6509003804e',
     gasUsed: 226783,
     cumulativeGasUsed: 226783,
     contractAddress: null,
     logs: [ [Object] ],
     status: true,
     logsBloom:
      '0x000000000000000000000000000000000000000000000000000000000000000000000000800000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000800000000000000000000000000000000000000000000000000
00000000000000000000000000000000200000000000000000000000000000000000000000000000000020000000000008000000000000000000000000000000000000000000000000000000000',
     rawLogs: [ [Object] ] },
  logs:
   [ { logIndex: 0,
       transactionIndex: 0,
       transactionHash:
        '0xcec379d36fc4d302009219e96bc5f3c9c7932dd3d9a6bbfde46c53e0999fea98',
       blockHash:
        '0x27d8c6b7eaa9027f002f8b9bf5c9dcb97cc65941341418232c0fc2f444917d79',
       blockNumber: 19,
       address: '0x6700af4569e692b6E104f893969aE6509003804e',
       type: 'mined',
       removed: false,
       id: 'log_a11060cf',
       event: 'NewMember',
       args: [Result] } ] }
truffle(development)>
```

Figure 23. Account no 4 calls the “participating” function, stating an ID and his neighbours’ IDs

As we see, the data we receive after this call are somewhat different from previous ones. The first thing we get is the transaction's hash which is essentially its ID, but afterwards we are presented with a receipt containing the rest of the details. This receipt is given to the specific account – in our case account no 4 – that makes the call, since he interacted directly with a contract function. In the receipt we get information like those demonstrated in figure 17, including transaction index, block hash and number, the addresses of both account no 4 and the contract's, the total amount of gas used in all relative activities and the status of our transaction which confirms it was completed successfully. We also have a more detailed look of the "logs[]" array, without explicitly choosing to explore it, because of an emitted event. When the "participating" function is run, we want to notify the community of the addition of another member, so we emit the "NewMember" event, which includes two arguments for the new participant's address and Id. These arguments can be further examined in the "args[]" array. Apart from event related data, we can also see the logs' index, their Id and the fact that this particular transaction has been mined.

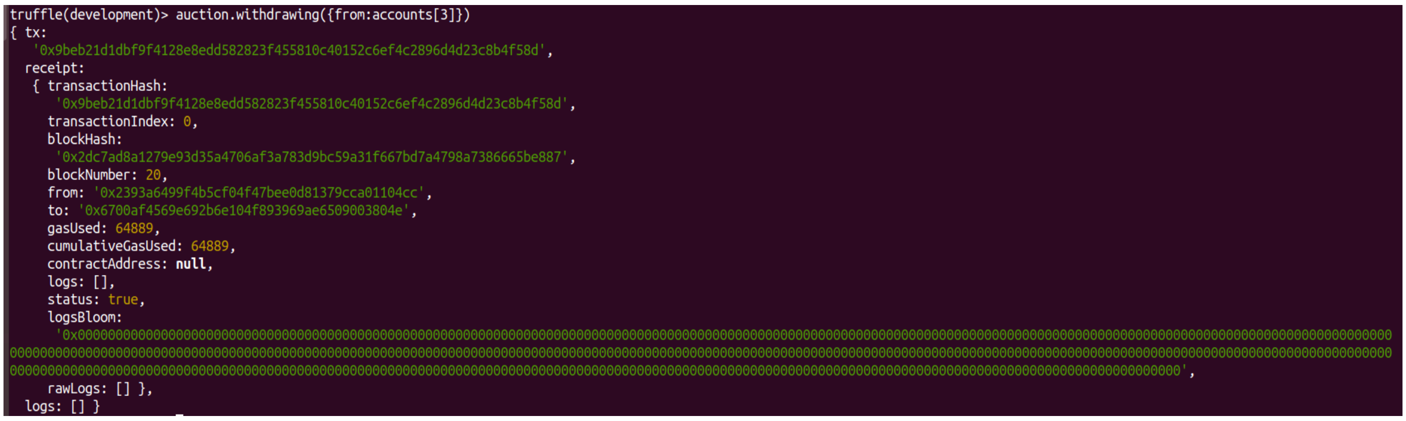
Participation can only be allowed after the deployer has chosen to initiate the whole process and every new member is added taking into consideration the existence of a connection to other pre-existing members. Since any node in the blockchain may want to know who is already participating, there is a public function, accessible to all, presenting this information.



```
truffle(development)> auction.showmembers({from:accounts[9]})
[ '0x12f52975617A22267A891a4F1dA29D010f5CdB4B',
  '0xcb22b848F33aDC99B3FE8B2975875A793bb52189',
  '0x2393A6499F4b5cF04F47BEe0d81379ccA01104cc',
  '0xAF2499CfEF05D461E9071bde37110391f2d8B596' ]
```

Figure 24. Calling the "showmembers" function

The community's members though may also choose to remove themselves from all relevant activities and not participate in the LEM anymore. It is assumed however that they will continue to operate as nodes through which energy will be able to be transported, because in case a pivotal node stops operating completely, some parts of the network could suddenly become isolated from the rest. Thankfully, since we are dealing with a local community, this isn't a challenging problem to tackle. Members that want to quit call the "withdrawing" function which simply removes them from the "memberaddresses" array, making sure null elements won't be left in their place by reordering the rest of the addresses and deleting the leftover empty space. Since instances of "members" struct are bound to an address, if a particular address is removed from all lists, the entirety of relevant information will be inaccessible and practically nonexistent. The departed account's Id is also removed from the "neighbouring" lists of all nearby nodes, making it possible to be acquired by another user.

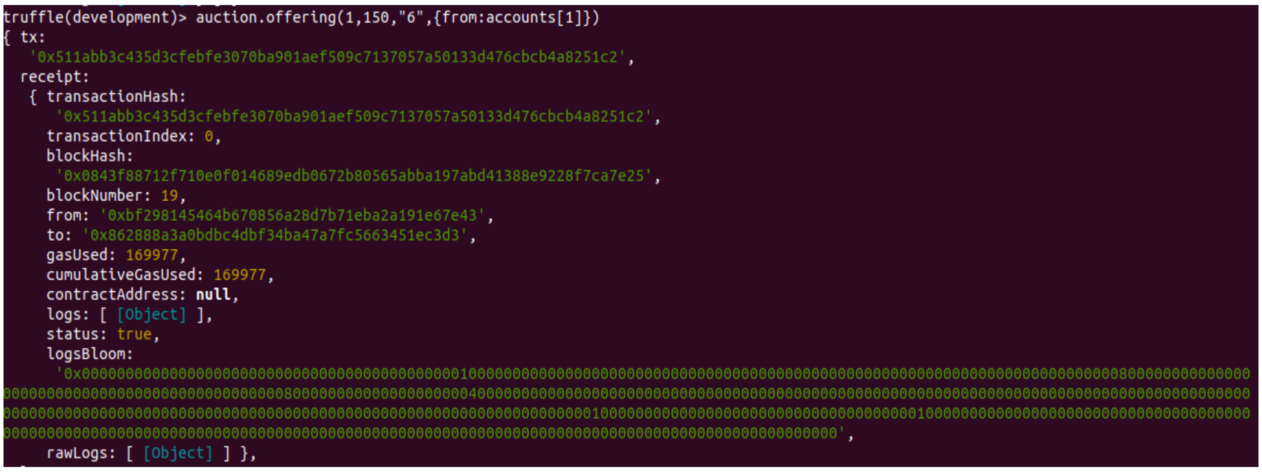


```
truffle(development)> auction.withdrawing({from:accounts[3]})
{ tx:
   '0x9beb21d1dbf9f4128e8edd582823f455810c40152c6ef4c2896d4d23c8b4f58d',
  receipt:
   { transactionHash:
      '0x9beb21d1dbf9f4128e8edd582823f455810c40152c6ef4c2896d4d23c8b4f58d',
     transactionIndex: 0,
     blockHash:
      '0x2dc7ad8a1279e93d35a4706af3a783d9bc59a31f667bd7a4798a7386665be887',
     blockNumber: 20,
     from: '0x2393a6499f4b5cf04f47bee0d81379cca01104cc',
     to: '0x6700af4569e692b6e104f893969ae6509003804e',
     gasUsed: 64889,
     cumulativeGasUsed: 64889,
     contractAddress: null,
     logs: [],
     status: true,
     logsBloom:
      '0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000',
     rawLogs: [] },
  logs: [] }
```

Figure 25. Account no 3 withdraws from the LEM

As in the case of the "participating" function, the data for a "withdrawing" call come in the form of a receipt. In the above figure we can see the transaction hash and index, the block hash and number, the address of account no 3 which makes the call, our contract's address and the amount of gas used for mining. Since there are no events emitted, the list of logs is not demonstrated automatically.

After signing up, the participants make their offers and demands, calling the respective functions for offering and buying energy. Each account's input consists of their id, the amount of kWhs they are either buying or selling and the differential cost of them. The differential cost could be an integer but just in case it isn't, members are expected to express it as a string and the "weiconversion" function deals with turning it to an integer, Wei value. Each entry is added to a new instance of structs similar to "members". In the case of "offering" we have the "sellers" struct, with data bound to addresses appended to an array called "selleraddresses". This array of addresses is then sorted in ascending order, regarding the kWhs' differential cost. That way, we can explore the energy offers from cheapest to most expensive, simply by navigating the addresses one by one and examining the corresponding "sellers" instances. It should also be noted that a readmission of offer is possible, in which case the account's previous entry is deleted.

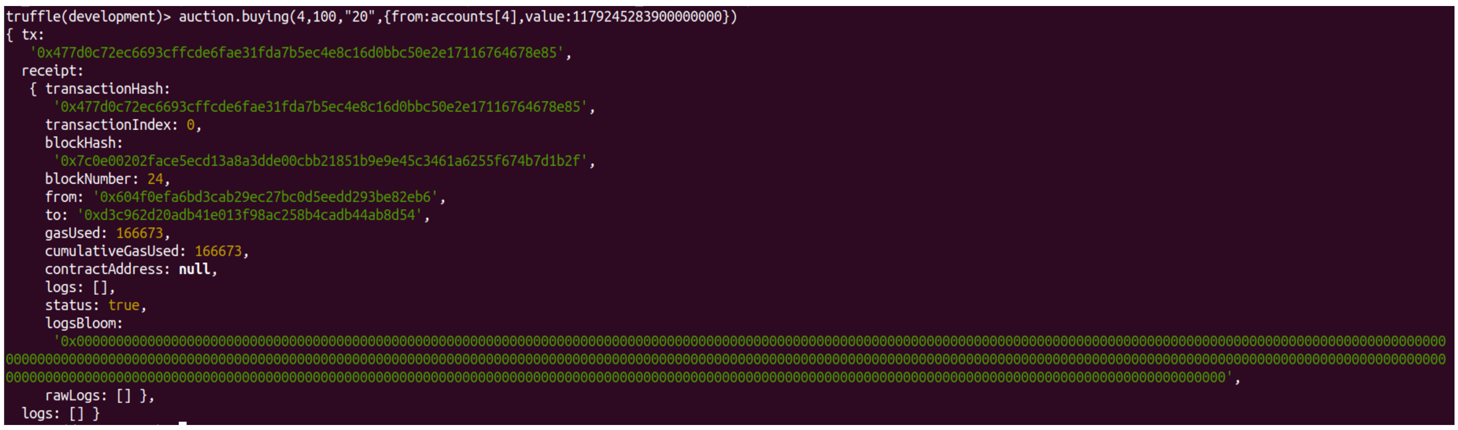


```
truffle(development)> auction.offering(1,150,"6",{from:accounts[1]})
{ tx:
   '0x511abb3c435d3cfebfe3070ba901aef509c7137057a50133d476cbcb4a8251c2',
  receipt:
   { transactionHash:
      '0x511abb3c435d3cfebfe3070ba901aef509c7137057a50133d476cbcb4a8251c2',
     transactionIndex: 0,
     blockHash:
      '0x0843f88712f710e0f014689edb0672b80565abba197abd41388e9228f7ca7e25',
     blockNumber: 19,
     from: '0xbf298145464b670856a28d7b71eba2a191e67e43',
     to: '0x862888a3a0bdbc4dbf34ba47a7fc5663451ec3d3',
     gasUsed: 169977,
     cumulativeGasUsed: 169977,
     contractAddress: null,
     logs: [ [Object] ],
     status: true,
     logsBloom:
      '0x00000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000008000000000000000
00000000000000000000000000000000800000000000000000000400000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000001000000000000000000000000000000000000100000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000',
     rawLogs: [ [Object] ] },
```

Figure 26. Account no 1 makes an energy offer and the transaction logs that consequently appear. Notice the differential cost is expected to be a string

Transaction logs are similar to those we got when we called the “withdrawing” function. In the example above, account no 1 makes an offer of 150 kWhs in the price of 6 USD and afterwards this input is stored inside the contract so it can be used by a different function later.

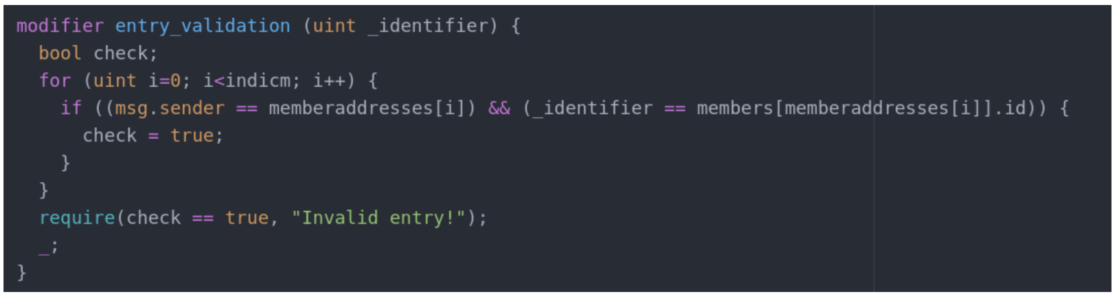
The “buying” function is really similar to the “offering” one, with the exception that people that demand energy need to offer some funds to the contract for the monetary transfers. In case their bid does not meet the multiplication product of the amount of kWhs and the differential cost they have stated, the function rejects the call. Of course, if their demands are not satisfied, the contract will make sure to hand them the respective amount of refund, so they don’t lose their money. One slight inconvenience regarding this function, is the need for the call to declare the funds that will be transferred in Wei. That means that the user will have to make the conversion from his preferred currency to Wei either by hand, or using any kind of calculator/converter. Since most conversions of this kind don’t produce integers, all we can do is try to round up the result so it can approximate the real number. This isn’t a major issue though, because Wei is a denomination of the Ether currency with a very small value. In fact, we can choose to round up (ceiling -making sure the offer isn’t insufficient) our value to the 8th digit of our number and still not even approach the value of a US Dollar cent.



```
truffle(development)> auction.buying(4,100,"20",{from:accounts[4],value:1179245283900000000})
{ tx:
   '0x477d0c72ec6693cffcde6fae31fda7b5ec4e8c16d0bbc50e2e17116764678e85',
  receipt:
   { transactionHash:
      '0x477d0c72ec6693cffcde6fae31fda7b5ec4e8c16d0bbc50e2e17116764678e85',
     transactionIndex: 0,
     blockHash:
      '0x7c0e00202face5ecd13a8a3dde00cbb21851b9e9e45c3461a6255f674b7d1b2f',
     blockNumber: 24,
     from: '0x604f0efa6bd3cab29ec27bc0d5eedd293be82eb6',
     to: '0xd3c962d20adb41e013f98ac258b4cadb44ab8d54',
     gasUsed: 166673,
     cumulativeGasUsed: 166673,
     contractAddress: null,
     logs: [],
     status: true,
     logsBloom:
      '0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000',
     rawLogs: [] },
  logs: [] }
```

Figure 27. A “buying” call

In both of these two functions, it is crucial to make sure the stated Id corresponds to the address of the correct account. Since this is a simple process we don’t want to repeat in terms of code, we implement it with a function modifier. A modifier is essentially another function which is applied to the one that’s called and depending on the satisfaction or not of some condition, the execution of the function is either permitted or blocked.



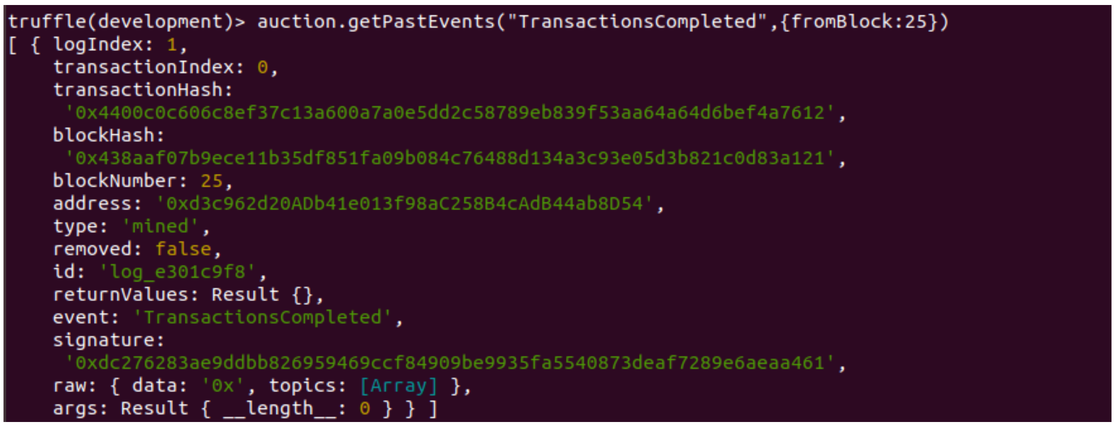
```
modifier entry_validation (uint _identifier) {
  bool check;
  for (uint i=0; i<indicm; i++) {
    if ((msg.sender == memberaddresses[i]) && (_identifier == members[memberaddresses[i]].id)) {
      check = true;
    }
  }
  require(check == true, "Invalid entry!");
  _;
}
```

Figure 28. Function modifier for entry validation

Finally, after every activity within these 15 minutes is completed, the main function of the contract is called by the oracle. This is the internal "transferring" function and it implements all the auction mechanisms needed, to determine which transactions and transfers of energy and money will occur. Giving priority to inputs of energy offers, it scans the arrays of buyers and sellers alike, aiming to give the cheapest energy produced, to consumers that are willing to pay more for it. Since both arrays are sorted ascendingly, according to the differential cost, every time a seller's production is sold out or a buyers demands are fully met, the same process is repeated with the members' inputs that are next in line. In the case of energy offers, it's the immediately following account's input, but energy buyers are scanned from the array's end moving towards the start, because the emphasis is given to bids of higher monetary value. Every time there is a match between a producer and a consumer, it is decided that the agreed amount of energy will be sold at a price equal to the mean of their kWh costing. In this community every member is a prosumer, meaning all entities that consume energy are able to produce their own, but sometimes consider covering their needs buying from others. This implies of course that sufficing the entirety of their needs with their own production is more costly than acquiring the energy of some other member and consequently, they have higher kWh differential costs. "transferring" ensures that a buyer must declare a higher differential cost than that of a seller in order to buy his energy. If this condition is not satisfied, the function doesn't scan any more seller inputs, since they are ascendingly ordered according to the kWh cost, and the buyer has to consider stating a higher price for the next cycle. This whole process ensures that no participant gets economic damage and although those who sell energy don't get the maximum profit they could, the entirety of the LEM is benefited.

A small share of the total profit in one cycle – more specifically one thousandth of it – will be granted to the deployer. Apart from that, the contract withholds a small percentage from each member's gains to cover the deployer's expenses for the oracle call. This percentage is equal for those that offered an input during these 15 minutes, which means that everyone who didn't manage to interact with some other and thus had zero gains, won't contribute to this collective effort and the deployer won't be totally compensated. This is considered however an unlikely scenario and won't affect the rest of the transactions either way, because it's one of the final issues the function deals with. Moreover, a residue of Wei is left in the contract after every cycle which could be an additional source of funds for compensation. This remainder adds up after every cycle because of the rounding up the buyers are doing when they declare their bid in the contract. The "transferring" function interacts with their inputs considering they have offered the exact amounts that correspond to the number of kWhs and their price and ends up transferring slightly inaccurate sums of money, leaving some in the contract. We avoid however to increase this remainder by using division operations, because as we already mentioned, solidity doesn't deal with decimals.

The last step is clearing the arrays of both sellers and buyers, preparing for the next cycle, and emitting an event indicating all transactions are completed.

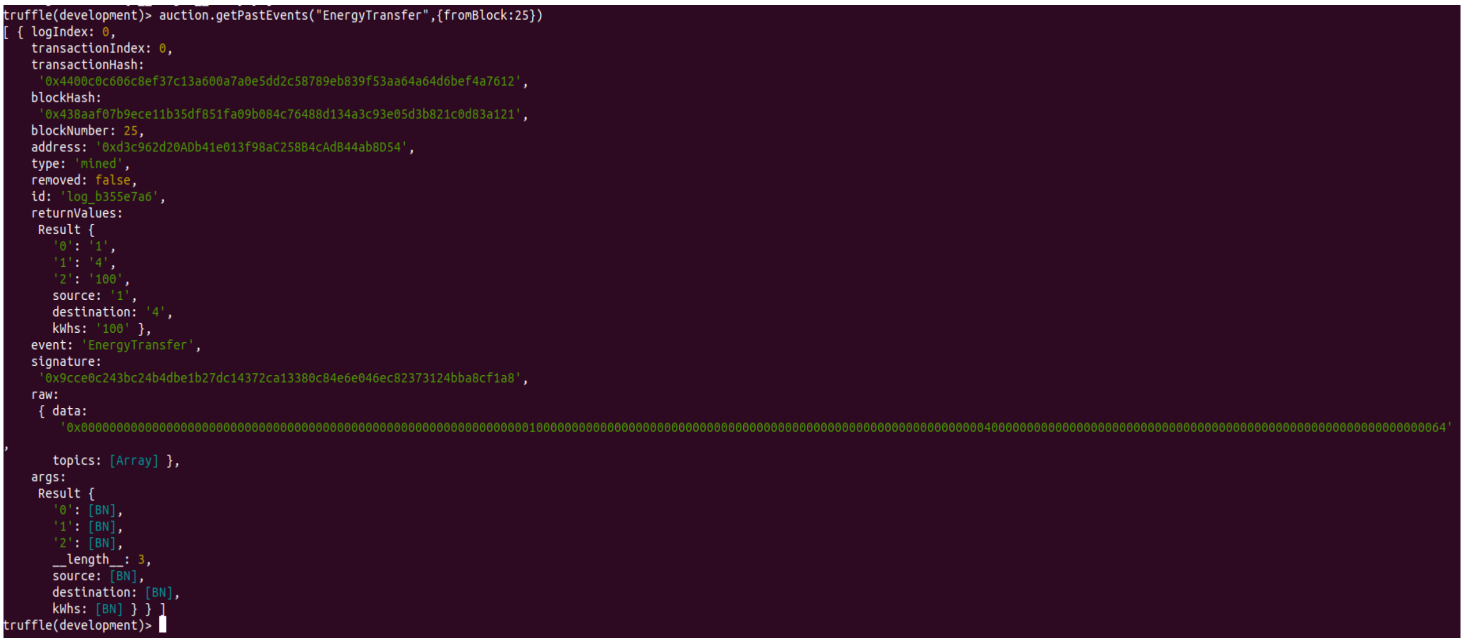


```
truffle(development)> auction.getPastEvents("TransactionsCompleted",{fromBlock:25})
[ { logIndex: 1,
    transactionIndex: 0,
    transactionHash:
     '0x4400c0c606c8ef37c13a600a7a0e5dd2c58789eb839f53aa64a64d6bef4a7612',
    blockHash:
     '0x438aaf07b9ece11b35df851fa09b084c76488d134a3c93e05d3b821c0d83a121',
    blockNumber: 25,
    address: '0xd3c962d20ADb41e013f98aC258B4cAdB44ab8D54',
    type: 'mined',
    removed: false,
    id: 'log_e301c9f8',
    returnValues: Result {},
    event: 'TransactionsCompleted',
    signature:
     '0xdc276283ae9ddbb826959469ccf84909be9935fa5540873deaf7289e6aeaa461',
    raw: { data: '0x', topics: [Array] },
    args: Result { __length__: 0 } } ]
```

Figure 29. A web3.js command that retrieves information relevant to the emission of a "TransactionsCompleted" event

This emission is the culmination of all auction operations that take place in these 15 minutes and it signals to all the transition from one cycle to another. All events transmit their data to the entirety of the blockchain and make this information universally known. In our case, logs for this event appear just as the "transferring" function is called and run, just as in the case of the "participating" function we saw in figure 23, but we can also retrieve this data with the help of the web3.js command we see above. We can utilize our previous knowledge, from the "transferring" logs, to learn the number of the block in which these data were stored and afterwards search for the logs of the "TransactionsCompleted" event, emitted during an activity taking place in the "auction" contract. After the command is executed, we receive the usual information regarding the transaction itself, the contract's address, but also an Id for this specific event and most importantly a large hexadecimal that is considered its signature and serves as a hash. Since our event has no arguments, no further details are needed.

Apart from the event above though, "transferring" also emits the "EnergyTransfer" event, which marks the agreement between two entities for an energy transaction. This particular action doesn't only have the role of informing the community and the two parties for the imminent exchange of power, but can also serve as the signal which triggers the seller's system to start moving the electricity throughout the network. Since the blockchain is a closed system and we want it to operate almost fully automatically, an event is the best method of informing the outside world to initiate an activity, as soon as the auction process is finished, especially if we take into account that events can be read by blockchain clients like Ganache-CLI.



```
truffle(development)> auction.getPastEvents("EnergyTransfer",{fromBlock:25})
[ { logIndex: 0,
    transactionIndex: 0,
    transactionHash:
     '0x4400c0c606c8ef37c13a600a7a0e5dd2c58789eb839f53aa64a64d6bef4a7612',
    blockHash:
     '0x438aaf07b9ece11b35df851fa09b084c76488d134a3c93e05d3b821c0d83a121',
    blockNumber: 25,
    address: '0xd3c962d20ADb41e013f98aC258B4cAdB44ab8D54',
    type: 'mined',
    removed: false,
    id: 'log_b355e7a6',
    returnValues:
     Result {
       '0': '1',
       '1': '4',
       '2': '100',
       source: '1',
       destination: '4',
       kWhs: '100' },
    event: 'EnergyTransfer',
    signature:
     '0x9cce0c243bc24b4dbe1b27dc14372ca13380c84e6e046ec82373124bba8cf1a8',
    raw:
     { data:
        '0x000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000040000000000000000000000000000000000000000000000000000000000000064'
,
       topics: [Array] },
    args:
     Result {
       '0': [BN],
       '1': [BN],
       '2': [BN],
       __length__: 3,
       source: [BN],
       destination: [BN],
       kWhs: [BN] } } ]
truffle(development)> 
```

Figure 30. Logs from the emission of an “EnergyTransfer” event

Running the same command as the previous one, we search for the logs of “EnegyTransfer” in the exact same block as before, because both events were emitted when the “transferring” function was run. Apart from information such as those we saw in the logs for “TransactionsCompleted”, we get additional ones regarding the arguments this event has. Three different integers are presented in the order they were originally put in and below them we can also see what each one represents. This way we understand that the community’s member with the Id of “1” has to transfer 100 kWhs worth of energy to the member with the Id “4”. After the signature, we are presented with the event’s raw logs that show the values of these three arguments can be depicted as BN (Big Numbers).

In the next figure we exhibit a flow chart, giving us a rough portrayal of the way “transferring” function operates. We did not include the emission of events before the energy transfer, or the more obscure details regarding the compensation of the deployer for setting up the oracle for the next cycle.

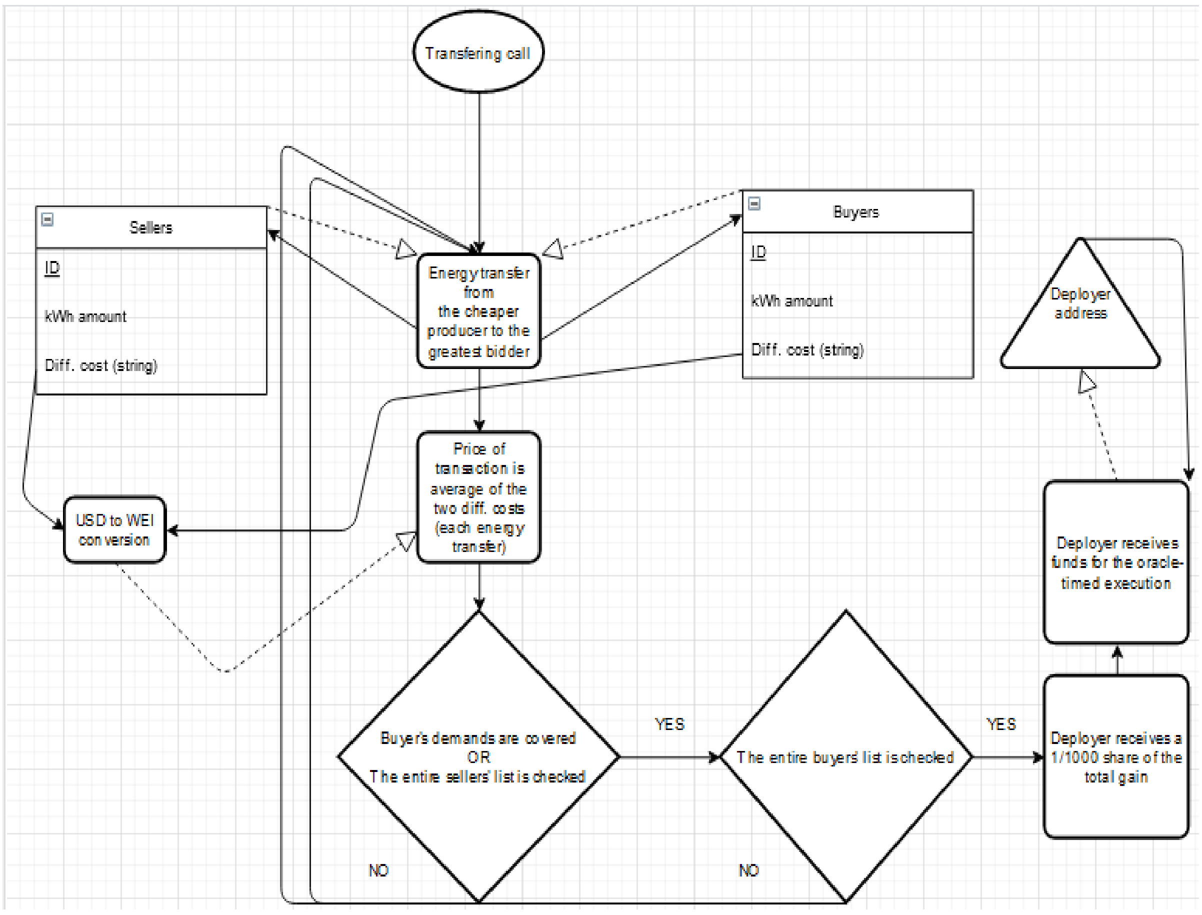

Transfering call
Sellers
ID
kWh amount
Diff. cost (string)
Buyers
ID
kWh amount
Diff. cost (string)
Energy transfer from the cheaper producer to the greatest bidder
Price of transaction is average of the two diff. costs (each energy transfer)
USD to WEI conversion
Deployer address
Deployer receives funds for the oracle-timed execution
Deployer receives a 1/1000 share of the total gain
Buyer's demands are covered OR The entire sellers' list is checked
The entire buyers' list is checked
YES
YES
NO
NO

Figure 31. Diagram explaining the "transferring" function

After the "transferring" function is run and the arrays of buyers and sellers are cleared, everything is ready for the next round of energy offers and demands. The "_callback" function, which accessed "transferring", calls "newCycle" right after and the oracle starts counting down for another repetition of the process we just described. A total overview of the "EnergyAuction" project is presented below, with a detailed showcase of all the different options a user has and the role of the deployer in the contract. The different forms of the "Participant" struct, namely "members", "sellers" and "buyers", are depicted in detail including their characteristic arguments. The "initiate", "newCycle" and "_callback" functions are just crammed in a block describing their part in "Time Window Creation" and the rest of the activities are shown to be eventually concluding with the operation of "transferring" and the subsequent transactions of energy and money between different member accounts.

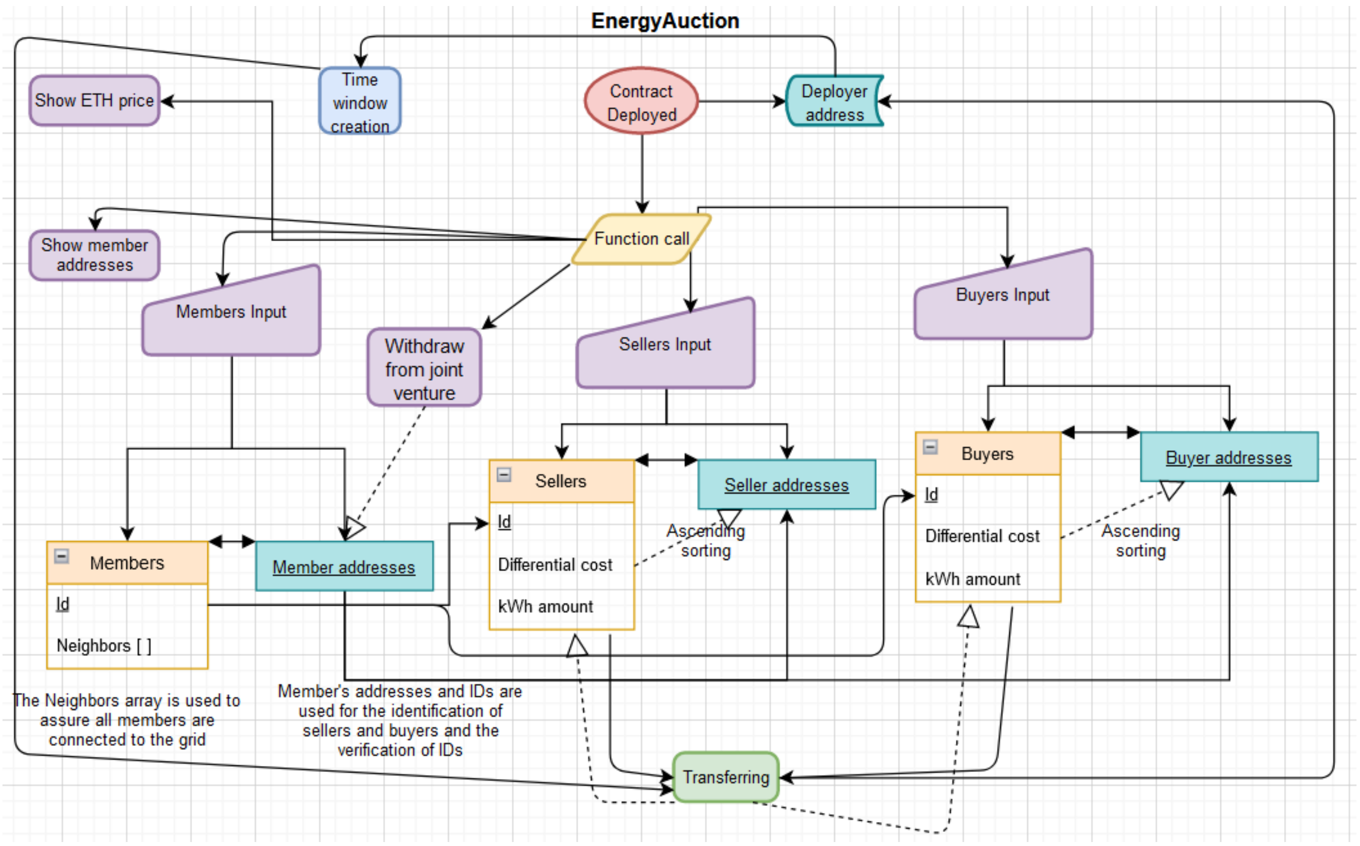


Figure 3233. A diagram with the overall architecture of the EnergyAuction contract

6. Future Work

6.1. Regarding the Dapp

First of all, the project could obviously be configured to run into a live, public network, like a Testnet, for more testing in conditions similar to those of the actual Mainnet. We could also work to find a better solution to the time window issue, either with the Ethereum Alarm Clock, or using a different method. We could also develop the code, so that it can tackle more efficiently the issues with the remainder of Wei in the contract's balance, since they could add up in the long term to hundreds or thousands of US dollars. And finally, apart from the smart contract back-end, a front-end user interface could be developed, so that our project could be shaped into a real Dapp.

6.2. Regarding the LEM

This project is simply working on the basic ideas of the operation of LEMs. Building on this bedrock, we could use advanced algorithms to store data and use oracles to make predictions on future energy demands. This way, we can minimize the risk of wasting energy and help the system become more stable and balanced. Moreover, this improvement could be a major incentive for additional members to assume the role of the prosumer, since production of energy would become a safer source of income. Lastly, we could add regulations that prioritize the consumption of green energy, a measure that combined with advanced prediction efficiency could really help integrating more renewable energy sources, which is one of the most important reasons for the LEMs' existence after all.

Figure List

References

[1] Faizan Safdar Ali, Moayad Aloqaily, Omar Alfandi, Oznur Ozkasap (2020) “Cyberphysical Blockchain-Enabled Peer-to-Peer Energy Trading”, https://ieeexplore.ieee.org/document/9187467

[2] https://private.coordinet-project.eu/files/documentos/6331d0e647262The%20CoordiNet%20Project%20-%20Final%20Results%20Report.pdf , The CoordiNet project: Paving the Way to Flexibility Markets in Europe

[3] https://www.em-power.eu/about-em-power-europe-impressions , Visited 18 November 2022

[4] https://pebbles-projekt.de/en/ , Visited 18 November 2022

[5] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system” http://nakamotoinstitute.org/bitcoin/ , Visited 29 October 2021

[6] https://ethereum.org/en/what-is-ethereum/ , Visited 29 October 2021

[7] Fabian Knirsch, Andreas Unterweger, Dominik Engel, “Implementing a blockchain from scratch: why, how, and what we learned“, https://doi.org/10.1186/s13635-019-0085-3

[8] Government Office for Science (UK), “Distributed Ledger Technology: beyond block chain” (Report January 2016), https://www.gov.uk/government/news/distributed-ledger-technology-beyond-block-chain

[9] Andreas M. Antonopoulos, “Mastering Bitcoin”, Second Edition (2017) https://github.com/bitcoinbook/bitcoinbook

[10] Jakobsson, Markus; Juels, Ari (1999). "Proofs of Work and Bread Pudding Protocols", https://doi.org/10.1007/978-0-387-35568-9_18

[11] https://www.investopedia.com/terms/p/proof-stake-pos.asp , By Jake Frankenfield, Updated April 21, 2021

[12] https://www.ibm.com/topics/smart-contracts , Visited 29 October 2021

[13] https://ethereum.org/en/developers/docs/intro-to-ethereum/ , Visited 29 October 2021

[14] https://ipfs.io/

[15] https://ethereum.org/en/developers/docs/smart-contracts/languages/ , Visited 29 October 2021

[16] https://www.gemini.com/cryptopedia/the-dao-hack-makerdao , Visited 30 October 2021

[17] https://web3js.readthedocs.io/en/v1.5.2/

[18] https://ethereum.org/en/eth2/shard-chains/ , Visited 29 October 2021

[19] Sumper, Andreas - *Micro and local power markets* (2019, John Wiley & Sons)

[20] Hatziargyriou, N., Asano, H., Iravani, R., and Marnay, C. (2007). Microgrids. IEEE Power Energy Mag. 5: 78–94.

[21] Tiago Sousa Tiago Soares Pierre Pinson Fabio Moret Thomas Baroche Etienne Sorin. “Peer-to-peer and community-based markets: A comprehensive review”, https://doi.org/10.1016/j.rser.2019.01.036

[22] N. Andriopoulos, A. Bachoumis, P. Alefragis, A. Birbas. “Optimization of a Local Energy Market Operation in a Transactive Energy Environment”, https://www.researchgate.net/publication/347266502_Optimization_of_a_Local_Energy_Market_Operation_in_a_Transactive_Energy_Environment

[23] https://ethereum.org/en/developers/docs/networks/ , Visited 29 October 2021

[24] https://ethereum.org/en/developers/docs/nodes-and-clients/, Visited 29 October 2021

[25] https://github.com/trufflesuite/ganache-cli-archive/blob/master/README.md , Visited 29 October 2021

[26] https://www.gemini.com/cryptopedia/crypto-oracle-blockchain-overview#section-blockchain-oracles-providers-of-external-data , Visited 29 October 2021

[27] https://docs.provable.xyz/ , Visited 29 October 2021

[28] https://github.com/provable-things/ethereum-bridge , Visited 29 October 2021

[29] https://trufflesuite.com/docs/truffle/ , Visited 29 October 2021

[30] https://github.com/provable-things/ethereum-api , Visited 29 October 2021

[31] https://ethereum-alarm-clock-service.readthedocs.io/en/latest/ , Visited 29 October 2021