ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΤΜΗΜΑ. ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

"Τεχνολογία Blockchain και ανάπτυξη έξυπνων συμβολαίων για την αγοραπωλησία μοναδικών τεκμηρίων"

Χρηστίδης Ιωάννης 45056

Επιβλέπουσα: Ελένη Αικατερίνη Λελίγκου Αναπληρώτρια Καθηγήτρια
Τομέας Ηλεκτρονικού Αυτοματισμού και Τηλεματικής

Σεπτέμβριος 2020

Η παρούσα διπλωματική εργασία εγκρίθηκε ομόφωνα από την τριμελή εξεταστική επιτροπή, η οποία ορίστηκε από την Γ.Σ. του Τμήματος Μηχανικών Βιομηχανικής Σχεδίασης και Παραγωγής του Πανεπιστημίου Δυτικής Αττικής, σύμφωνα με το νόμο και τον εγκεκριμένο Οδηγό Σπουδών του τμήματος.

Μέλη της Επιτροπής

Λελίγκου Ελένη-Αικατερίνη (Επιβλέπων) ELENI AIKATERINI LELIGKOU Digitally signed by ELENI AIKATERINI LELIGKOU Date: 2021.04.27 12:32:18 +03'00'

Κάντζος Δημήτριος (Μέλος) Dimitrios Kantzos Digitally signed by Dimitrios Kantzos Date: 2021.04.27 14:09:43 +03'00'

Δρόσος Χρήστος (Μέλος) Christos Drosos Digitally signed by Christos Drosos Date: 2021.04.27 14:34:33 +03'00'

Δήλωση συγγραφέα διπλωματικής εργασίας

Ο κάτωθι υπογεγραμμένος Χρηστίδης Ιωάννης του Σπυρίδων με αριθμό μητρώου 45056, φοιτητής του τμήματος Μηχανικών Βιομηχανικής Σχεδίασης και Παραγωγής, του Πανεπιστημίου Δυτικής Αττικής, πριν αναλάβω την εκπόνηση της Διπλωματικής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω :

«Είμαι συγγραφέας αυτής της διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Υπογραφή

Ημερομηνία

16-10-2020

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω θερμά την επιβλέπουσα καθηγήτρια μου κυρία Ελένη Αικατερίνη Λελίγκου για την ευκαιρία που μου έδωσε να συνεργαστούμε και να αποτελέσω μέλος της ομάδας της. Την ευχαριστώ για την εμπιστοσύνη που μου έδειξε, για την υποστήριξη, την υπομονή και την συνεχή προτροπή της να βελτιώνομαι.

Επίσης θα ήθελα να ευχαριστήσω τον Δρ. Δημήτριο Κόγια για την καθοδήγηση του στα πρώτα μου βήματα στο Blockchain καθώς και την βοήθεια που μου πρόσφερε κατά την εκπόνηση της διπλωματικής εργασίας.

Στην συνέχεια θα ήθελα να ευχαριστήσω τον Υποψήφιο Διδάκτορα Μιχάλη Ξευγένη που με τις γνώσεις του με βοήθησε να καταλάβω την ουσία του blockchain και με ενέπνευσε να ασχοληθώ με αυτό και στο μέλλον.

Θα ήθελα να ευχαριστήσω τη Γεθσημανή Παπαδοπούλου για τη βοήθεια της κατά την εκπόνηση της διπλωματικής εργασίας και την διαρκή υποστήριξη της.

Τέλος θα ήθελα να ευχαριστήσω τους γονείς μου που με στηρίζουν σε κάθε μου βήμα.

Περίληψη

Η τεχνολογία Blockchain χρησιμοποιήθηκε αρχικά από τον Shatoshi Nakamoto το 2009 για την δημιουργία της πλατφόρμας bitcoin, ενός ψηφιακού κρυπτονομίσματος που ακόμα και σήμερα θεωρείται το πιο επιτυχημένο κρυπτονόμισμα παγκοσμίως. Η πλατφόρμα του bitcoin χρησιμοποιείται κυρίως για την εκτέλεση ψηφιακών οικονομικών συναλλαγών μεταξύ αγνώστων χωρίς την ύπαρξη μιας οντότητας που μεσολαβεί για την έγκυρη ολοκλήρωση τους. Η δημιουργία του bitcoin άλλαξε ριζικά τον τρόπο που γίνονται οι διαδικτυακές συναλλαγές. Μετά από το bitcoin αναπτύχθηκαν πολλές ακόμα μορφές κρυπτονομισμάτων αλλά οι περισσότερες ακολουθούσαν παρόμοια αρχή λειτουργίας με το bitcoin χωρίς να προσφέρουν κάτι διαφορετικό. Όμως η τεχνολογία blockchain συνέχισε να εξελίσσεται και κατά συνέπεια το φάσμα των εφαρμογών στις οποίες χρησιμοποιείται συνεχώς επεκτείνεται.

Πλέον η τεχνολογία αυτή χρησιμοποιείται σε εφαρμογές της βιομηχανίας, της οικονομίας και της απλής καθημερινότητας ενός ατόμου λόγω της ασφάλειας και της εγκυρότητας που προσδίδει. Το 2013 δημιουργήθηκε, από τον Vitalik Buterin, το Ethereum blockchain, μία πλατφόρμα που υποστηρίζει τη δημιουργία κατανεμημένων εφαρμογών γνωστά ως DApps (decentralized applications). Οι κατανεμημένες εφαρμογές που αναπτύσσονται στο ethereum blockchain δεν περιορίζονται μόνο σε οικονομικές συναλλαγές όπως στο bitcoin διότι η πλατφόρμα του ethereum επιτρέπει την δημιουργία ποικίλων ειδών εφαρμογών χρησιμοποιώντας λογική που είναι αποθηκευμένη μέσα στο δίκτυο, τα γνωστά έξυπνα συμβόλαια. Τα έξυπνα συμβόλαια είναι αρχεία κώδικα που εκτελούνται με απολυτότητα και ντετερμινισμό, με αποτέλεσμα η λειτουργικότητα τους να είναι καθολική. Πιθανές εφαρμογές που βασίζονται στη λειτουργικότητα ενός έξυπνου συμβολαίου και δεν αφορούν οικονομικές συναλλαγές είναι η κατοχύρωση δικαιωμάτων, logistics ή ακόμα και εφαρμογές που προσδίδουν μια μορφή ιεραρχίας.

Η παρούσα διπλωματική εργασία δημιουργήθηκε με σκοπό να αναδείξει τις δυνατότητες και τα όρια των έξυπνων συμβολαίων καθώς και προβλήματα που μπορεί να υπάρξουν κατά την ανάπτυξη τους και κατά την εφαρμογή τους. Η εφαρμογή αναπτύσσεται σε ένα τοπικό ethereumblockchain δίκτυο και πραγματεύεται την δημιουργία ψηφιακών τεκμηρίων(token). Τα ψηφιακά τεκμήρια είναι γνωστά στη κοινότητα του ethereum σαν άυλα περιουσιακά στοιχεία τα οποία μπορούν να πάρουν τη μορφή κρυπτοοικονομίας ή να έχουν την ιδιότητα της μοναδικότητας, με αποτέλεσμα να είναι ξεχωριστά και να αποκτούν συγκεκριμένα χαρακτηριστικά. Στην συγκεκριμένη εφαρμογή αναπτύσσονται και τα δύο ήδη τεκμηρίων που αναφέρθηκαν με αποτέλεσμα να δημιουργηθεί μία ψηφιακή αγορά με μοναδικά τεκμήρια τα οποία μπορούν να δημιουργηθούν, να αγοραστούν και να πωληθούν με αντάλλαγμα τα τεκμήρια που προσομοιάζουν την κρυπτοοικονομία. Σε κάθε περίπτωση ο κοινός παράγοντας για την δημιουργία των τεκμηρίων είναι η «ιδιότητα της ιδιοκτησίας», δηλαδή ότι ένα τεκμήριο, είτε είναι μοναδικό είτε είναι νόμισμα, ανήκει σε έναν συγκεκριμένο χρήστη που έχει όλες τις δυνατότητες που του προσδίδει αυτό το τεκμήριο και κανείς άλλος χρήστης δε μπορεί να το χρησιμοποιήσει.

Summary

Blockchain technology was first used by Shatoshi Nakamoto in 2009 to create the bitcoin platform, a digital cryptocurrency that is still considered the most successful cryptocurrency in the world today. The bitcoin platform is mainly used for the execution of digital financial transactions between strangers without the existence of an entity that mediates for their valid completion. The creation of bitcoin has radically changed the way online transactions are done. Many other forms of cryptocurrencies developed after bitcoin but most followed a similar principle of operation to bitcoin without offering anything different. But blockchain technology has continued to evolve and as a result the range of applications in which it is used is constantly expanding. Now this technology is used in applications of industry, economy and simple everyday life of a person because of the security and validity it provides. In 2013, VitalikButerin created the Ethereum blockchain, a platform that supports the creation of distributed applications known as DApps (decentralized applications). Distributed applications developed in the ethereum blockchain are not limited to financial transactions such as bitcoin because the ethereum platform allows the creation of various types of applications using logic stored within the network, the well-known smart contracts. Smart contracts are code files that are executed with absoluteness and determinism, with the result that their functionality is universal. Possible applications that are based on the functionality of a smart contract and do not involve financial transactions are the securing of rights, logistics or even applications that provide a form of hierarchy. This thesis was created in order to highlight the possibilities and limits of smart contracts as well as problems that may exist during their development and after migration. The application is developed in a local ethereum blockchain network and deals with the creation of digital tokens. Digital tokens are known in the ethereum community as intangible assets that can take the form of cryptocurrencies or have the property of uniqueness, resulting in them being distinct and acquiring specific characteristics. In this application, both tokens are developed, resulting in the creation of a digital market with unique tokens that can be created, bought and sold in exchange for tokens that resemble a cryptocurrency. In any case, the common factor for the creation of tokens is ownability, which means that a token, whether it is unique or serves as a cryptocurrency, belongs to a specific user where he has all the advantages that this item provides him and no other user has them.

Πίνακας περιεχομένων

3.5.4. Proof of capacity

3.5.5. Proof of Elapsed Time

3.5.6. Byzantine fault tolerance

3.6. Bitcoin

3.7. Ethereum

3.7.1. Smart Contracts

3.7.2. Ethereum virtual machine

3.7.3. Χρήση του ethereum σε κατανεμημένες εφαρμογές(Dapps)

3.7.4. Αποκεντρωμένοι Αυτόνομοι Οργανισμοί

3.7.5. ERC-20

3.7.6. ERC-721

3.8. HyperledgerFabric

Β. Μεθοδολογία

1. Τεχνολογίες και εργαλεία

1.1. Gethsoftware

1.2. Solidity programming language

1.3. Remix-Solidity IDE

1.4. GanacheCLI

1.5. Metamask

1.6. Node Package Manager

1.7. Node.js

1.8. Truffle framework

1.9. Javascript

1.10. Web3 library

1.11. jQuery

1.12 Bootstrap

2. Επεξήγηση κατανεμημένης εφαρμογής

3. Υλοποίηση εφαρμογής

3.1. Ανάπτυξη Smart Contract

3.2. Παρουσίαση εφαρμογής

3.3. Ανάπτυξη εφαρμογής από κακόβουλο χρήστη

Γ. Συμπεράσματα

Δ. Βιβλιογραφικές αναφορές

Α. Εισαγωγή

Το blockchain είναι ένα δημόσιο καθολικό στο οποίο ο καθένας έχει πρόσβαση αλλά δεν υπάρχει κάποια κεντρική αρχή να έχει έλεγχο. Είναι μια τεχνολογία που επιτρέπει σε άτομα και εταιρείες να συνεργάζονται με εμπιστοσύνη και διαφάνεια. Μία από τις πιο γνωστές εφαρμογές των blockchain είναι τα κρυπτονομίσματα όπως το Bitcoin αλλά είναι δυνατές πολλές άλλες εφαρμογές. Η τεχνολογία Blockchain θεωρείται η κινητήρια δύναμη της επόμενης θεμελιώδους επανάστασης στην τεχνολογία πληροφοριών. Πολλές εφαρμογές της τεχνολογίας blockchain είναι ευρέως διαθέσιμες σήμερα, καθεμία από τις οποίες έχει ιδιαίτερη ισχύ σε έναν συγκεκριμένο τομέα εφαρμογών.

1. Τύποι συστημάτων

1.1. Συγκεντρωτικά συστήματα

Στα συγκεντρωτικά συστήματα όλοι οι κόμβοι συνδέονται με έναν κεντρικό διακομιστή[28]. Αυτή η αρχιτεκτονική ονομάζεται client/server. Το centralized system είναι ο συνηθέστερος τύπος συστήματος και χρησιμοποιείται σε πολλούς οργανισμούς με τον πελάτη να στέλνει αίτημα στον κεντρικό διακομιστή και να λαμβάνει απάντηση.

Βασικά χαρακτηριστικά αυτού του συστήματος είναι τα εξής:

1) Οι κόμβοι πελατών συγχρονίζονται σύμφωνα με το καθολικό ρολόι του κεντρικού κόμβου.
2) Όλοι οι κόμβοι συντονίζονται ή εξυπηρετούνται από την κεντρική μονάδα δηλαδή τον διακομιστή.
3) Κάθε κόμβος είναι εξαρτημένος από τον κεντρικό. Αυτό σημαίνει ότι όταν ο κεντρικός κόμβος δεν βρίσκεται σε λειτουργία τότε δεν υπάρχει δυνατότητα αποστολής και λήψης αιτημάτων και απαντήσεων.

Στα πλεονεκτήματα των συγκεντρωτικών συστημάτων συμπεριλαμβάνονται:

1) Εύκολη ασφάλιση δικτύου λόγω της θέσης των κόμβων. Κάθε κόμβος είναι άμεσα συνδεδεμένος με τον κεντρικό.
2) Ομαλή διαχείριση του συστήματος εφόσον το δίκτυο προσαρμόζεται στις αντίστοιχες προδιαγραφές.
3) Τα συγκεντρωτικά συστήματα είναι οικονομικότερα όταν χρησιμοποιούνται για μικρές υλοποιήσεις.
4) Εύκολες αναβαθμίσεις εφόσον μόνο ένα μηχάνημα χρειάζεται αναβάθμιση.
5) Εύκολη απόσπαση κόμβου από το σύστημα.

Παρά την πληθώρα πλεονεκτημάτων των συγκεντρωτικών συστημάτων, αυτά παρουσιάζουν μια σειρά από μειονεκτήματα και περιορισμούς που απαριθμούνται στη συνέχεια:

1) Τα συστήματα αυτά είναι εξαρτημένα από τη συνδεσιμότητα του δικτύου.
2) Η δυνατότητα δημιουργίας αντίγραφων είναι μικρότερη.

3) Δυσκολότερη συντήρηση δικτύου εφόσον αυτό σημαίνει ότι ο διακομιστής πρέπει να τεθεί εκτός λειτουργίας για το χρονικό διάστημα της συντήρησης.
4) Δεδομένου ότι ο διακομιστής έχει συγκεκριμένο αριθμό ανοιχτών πορτών(ports) που επικοινωνούν με τους κόμβους πελατών, είναι δυνατόν να υπάρξουν προβλήματα κινητικότητας το δικτύου(traffic).

1.2. Αποκεντρωμένα συστήματα

Στα αποκεντρωμένα συστήματα η συμπεριφορά του συστήματος προκύπτει από το σύνολο των αποφάσεων των κόμβων. Κάθε κόμβος παίρνει την δική του απόφαση οπότε δεν υπάρχει συγκεκριμένη οντότητα που στέλνει αιτήματα και λαμβάνει απαντήσεις.

Μερικά από τα βασικά χαρακτηριστικά των αποκεντρωμένων συστημάτων είναι:

1) Οι κόμβοι είναι ανεξάρτητοι μεταξύ τους, επομένως έχουν και διαφορετικό ρολόι.
2) Υπάρχουν περισσότερες από μία κεντρικές μονάδες. Κάθε κεντρική μονάδα ακούει στις συνδέσεις από άλλους κόμβους.
3) Κάθε κόμβος λαμβάνει τις δικές του αποφάσεις οπότε όταν υπάρξει αστοχία ενός κόμβου δεν θα εξαπλωθεί σε όλο το σύστημα.

Πλεονεκτήματα των αποκεντρωμένων συστημάτων:

1) Μειώνεται η πιθανότητα συμφόρησης του δικτύου εφόσον το φορτίο μοιράζεται στους κόμβους του.
2) Υψηλή διαθεσιμότητα δικτύου εφόσον πάντα θα υπάρχουν κόμβοι σε λειτουργία.
3) Επειδή κάθε κόμβος έχει τη δική του συμπεριφορά υπάρχει αυτονομία και έλεγχος των πόρων.

Μειονεκτήματα και περιορισμοί των αποκεντρωμένων συστημάτων:

1) Όταν κάθε κόμβος είναι αυτόνομος υπάρχει δυσκολία στην επίτευξη μεγάλων έργων λόγο απουσίας της ιεραρχίας, επιτήρησης και συνεπώς διοίκησης του συστήματος.
2) Ένα ακόμα μειονέκτημα αυτών των συστημάτων είναι η δυσκολία εντοπισμού του κόμβου που έστειλε μια απάντηση καθώς και ενός κόμβου που σταμάτησε να λειτουργεί.
3) Υπάρχει δυσκολία υλοποίησης μικρών συστημάτων λόγω του κόστους και της απόδοσης τους.

1.3. Κατανεμημένα συστήματα

Στα κατανεμημένα συστήματα η επεξεργασία αποφάσεων μοιράζεται στους κόμβους του δικτύου και η απόφαση λαμβάνεται συγκεντρωτικά χρησιμοποιώντας την πλήρη γνώση του συστήματος.

Χαρακτηριστικά των κατανεμημένων συστημάτων:

1) Για τη λήψη αποφάσεων χρησιμοποιούνται πρωτόκολλα συναίνεσης. Οι αποφάσεις μπορεί να είναι συναλλαγές, αποτέλεσμα μιας τιμής ή καταγραφή κάποιας πληροφορίας.
2) Όπως και στα αποκεντρωμένα συστήματα δεν υπάρχει κοινό ρολόι.
3) Μία μεγάλη διαφορά σε σχέση με τα αποκεντρωμένα συστήματα είναι πως αν ένας κόμβος σταματήσει τη λειτουργία του ή αν αποκτήσει “κακή βούληση” δεν θα επηρεάσει το δίκτυο εφόσον οι αποφάσεις βγαίνουν συγκεντρωτικά και το δίκτυο θα συνεχίσει τη λειτουργία του.

Πλεονεκτήματα των κατανεμημένων συστημάτων:

1) Ελάχιστη πιθανότητα συμφόρησης του δικτύου, σε αντίθεση με τα συγκεντρωτικά συστήματα, στα οποία τα αιτήματα στέλνονται μόνο σε έναν διακομιστή.
2) Μειωμένη πιθανότητα να χαθούν δεδομένα, εφόσον αυτά δε βρίσκονται μόνο σε ένα μηχάνημα.
3) Μεγαλύτερη ασφάλεια δεδομένων επειδή οι αποφάσεις παίρνονται συγκεντρωτικά.

Μειονεκτήματα των κατανεμημένων συστημάτων:

1) Η αρχιτεκτονική, καθώς και η υλοποίηση αυτών των συστημάτων είναι πολύπλοκες.
2) Ο εντοπισμός σφαλμάτων απαιτεί ιδιαίτερη προσοχή στους αλγόριθμους των συστημάτων. Η προσθήκη ενός νέου κόμβου αυξάνει την πολυπλοκότητα των αλγορίθμων.
3) Στα κατανεμημένα συστήματα υπάρχει δυσκολία στη χρονική διάταξη των συναλλαγών και των συμβάντων διότι το ρολόι κάθε κόμβου είναι ξεχωριστό.
4) Για να έχει την πλήρη εικόνα του συστήματος ένας κόμβος πρέπει να στηριχθεί σε χρονικές καταστάσεις του κάθε κόμβου και όχι στην τελική τους κατάσταση.

1.4. Ομότιμα κατανεμημένα συστήματα Peer-to-Peer

Ένα δίκτυο ονομάζεται Peer-to-peer(P2P) εάν η αρχιτεκτονική του είναι κατανεμημένη και οι συμμετέχοντες μοιράζονται ένα μέρος των πόρων τους(επεξεργαστική ισχύ, αποθηκευτικό χώρο, χωρητικότητα σύνδεσης δικτύου). Οι πόροι αυτοί είναι απαραίτητοι για την παροχή των υπηρεσιών και του περιεχομένου που προσφέρεται από το δίκτυο και είναι εύκολα προσβάσιμοι από άλλους κόμβους χωρίς την ανάγκη ενδιάμεσων φορέων. Άρα οι συμμετέχοντες ενός P2P δικτύου θεωρούνται ταυτόχρονα πάροχοι και αιτούντες των πόρων του δικτύου.Ωστόσο, μπορούμε να διακρίνουμε και υπο-ορισμούς για τα P2P δίκτυα, οι οποίοι θα αναλυθούν παρακάτω.

Τα P2P δίκτυα αναφέρονται σαν μία συλλογή από ετερογενείς κατανεμημένους πόρους που συνδέονται σε ένα δίκτυο και θεωρούνται αρχιτεκτονική αντίστροφη των Client/Server. Υπάρχουν δύο υπό-ορισμοί για αυτά τα δίκτυα, “Hybrid” P2P και “Pure” P2P.

Ορισμός “Pure” P2P

Ένα κατανεμημένο δίκτυο ονομάζεται “Pure” P2P εάν ακολουθεί τις προδιαγραφές ενός P2P δικτύου και αν οποιαδήποτε τερματική οντότητα μπορεί να αφαιρεθεί από το δίκτυο χωρίς να υποστεί κάποια απώλεια η υπηρεσία δικτύου.

Ορισμός “Hybrid” P2P

Ένα κατανεμημένο δίκτυο θεωρείται “Hybrid” P2P όταν ακολουθεί τις προδιαγραφές ενός P2P δικτύου και υπάρχει μία κεντρική οντότητα που είναι απαραίτητη για την παροχή τμημάτων των προσφερόμενων υπηρεσιών δικτύου.

1.5. Πρόβλημα Βυζαντινών στρατηγών

Ένα αξιόπιστο κατανεμημένο σύστημα πρέπει να είναι σε θέση να αντιμετωπίσει τις κακόβουλες συμπεριφορές των χρηστών του καθώς και την ύπαρξη τεχνικών προβλημάτων στους κόμβους του δικτύου ή ακόμα και στο ίδιο το δίκτυο [1]. Το παράδειγμα του προβλήματος των βυζαντινών στρατηγών εξηγεί τον όρο κακόβουλη συμπεριφορά και τα προβλήματα που μπορεί να προκαλέσει

στο δίκτυο[2]. Υποτίθεται ότι υπάρχουν στρατιωτικές μονάδες με έναν βυζαντινό στρατηγό η κάθε μια και περικυκλώνουν μία αντίπαλη πόλη. Τα δύο αντίπαλα στρατόπεδα είναι σχεδόν ισοδύναμα. Για την άλωση της πόλης πρέπει όλοι οι στρατηγοί που περικυκλώνουν την πόλη να επιτεθούν ταυτόχρονα. Για την επίτευξη αυτού του σκοπού χρησιμοποιούν αγγελιοφόρους που μεταφέρουν τις απαραίτητες πληροφορίες με τη μορφή μηνυμάτων. Η πιθανότητα να νικήσουν είναι μηδενική αν ένα τάγμα δεν επιτεθεί την ίδια χρονική στιγμή με τα υπόλοιπα. Εφόσον ο χρόνος είναι περιορισμένος οι αγγελιοφόροι πρέπει να περάσουν κρυφά μέσα από την πόλη ώστε να μεταφέρουν την πληροφορία στους άλλους στρατηγούς. Στο σημείο αυτό δημιουργούνται δύο προβλήματα. Πρώτον σε περίπτωση που συλληφθούν οι αγγελιαφόροι κατά τη διάρκεια της αποστολής τους, το γεγονός της σύλληψης δε θα γίνει άμεσα γνωστό στους στρατηγούς και δεύτερον εφόσον οι πληροφορίες είναι σε γραπτή μορφή αν συλληφθεί έστω και ένας αγγελιαφόρος τότε η πόλη θα γνωρίζει την ακριβή ώρα της επίθεσης και μπορεί κακόβουλα να στείλει λανθασμένη πληροφορία στους στρατηγούς. Η λύση στο προαναφερθέν πρόβλημα είναι είτε να δωθούν στους αγγελιοφόρους κρυπτογραφημένα μηνύματατα οποία μπορούν να αποκρυπτογραφήσουνμόνο οι μαθηματικοί των βυζαντινών στρατηγών είτε ο χρόνος που απαιτείται για την αποκρυπτογράφηση, την τροποποίηση και επανακρυπτογράφηση τους να είναι μεγαλύτερος από την διάστημα που απομένει για να ξεκινήσει η επίθεση.

Σε δεύτερο, αντίστοιχο, παράδειγμα μπορεί ένας ή περισσότεροι από τους στρατηγούς να είναι προδότες και εσκεμμένα να μεταδώσει εσφαλμένη πληροφορία. Σε αυτή τη περίπτωση, η λύση είναι οι έντιμοι στρατηγοί να είναι περισσότεροι από τους προδότες και δίνεται με τη χρήση αλγορίθμων.

1.6. Κοινή συναίνεση

Σε αντίθεση με τα συγκεντρωτικά συστήματα όπου οι αποφάσεις λαμβάνονται μόνο από τον διακομιστή στα κατανεμημένα συστήματα λαμβάνονται συναινετικά από όλους τους συμμετέχοντες. Το πρόβλημα κοινής συναίνεσης(consensusproblem)[3] και γενικότερα τα προβλήματα ομοφωνίας δημιουργούν τη βάση για την επίλυση της ανάπτυξης αξιόπιστων κατανεμημένων συστημάτων. Με την χρήση αυτών το πρωτοκόλλων οι συμμετέχοντες έχουν τη δυνατότητα να συντονίσουν τις πράξεις τους ώστε να διατηρηθεί και να διασφαλιστεί η κατάσταση του συστήματος. Η εξασφάλιση ότι όλες οι επεξεργασίες των αξιόπιστων συμμετεχόντων καταλήγουν στο ίδιο αποτέλεσμα το οποίο είχε προταθεί από προηγούμενες επεξεργασίες αποτελεί το πρόβλημα κοινής συναίνεσης. Υποθέτοντας ότι κάθε επεξεργασία "i" καταλήγει σε αποτέλεσμα "v_i" και όλες οι αξιόπιστες επεξεργασίες κατέληξαν στο ίδιο αποτέλεσμα "v" τότε το πρόβλημα κοινής συναίνεσης ορίζεται από τις παρακάτω ιδιότητες:

- **Εγκυρότητα**: Εάν μια αξιόπιστη επεξεργασία καταλήξει στο αποτέλεσμα "v", τότε "v" προτάθηκε από κάποια επεξεργασία
- **Συμφωνία**: Καμία αξιόπιστη επεξεργασία δεν πρέπει να καταλήξει σε αποτέλεσμα διάφορο του "v"
- **Ολοκλήρωση επεξεργασίας**: Κάθε αξιόπιστη επεξεργασία πρέπει να καταλήξει σε αποτέλεσμα
- **Ακεραιότητα**: Κάθε αξιόπιστη επεξεργασία καταλήγει σε αποτέλεσμα μόνο μία φορά

2. Κρυπτογραφία

Η κρυπτογραφία είναι η προστασία των δεδομένων από μη εξουσιοδοτημένα άτομα[4]. Η κρυπτογράφηση ουσιαστικά μετασχηματίζει δεδομένα σε κείμενο cyphertext δηλαδή κρυπτογραφημένο κείμενο το οποίο είναι αδύνατο να αποκρυπτογραφηθεί από τη στιγμή που δεν υπάρχει γνώση για το είδος της κρυπτογράφησης. Αποκρυπτογράφηση σημαίνει μετασχηματισμός του κρυπτογραφημένου κειμένου στο αρχικό κείμενο. Η χρήση της κρυπτογραφία φαίνεται στην επίτευξη της ταυτοποίησης (identification) δηλαδή του ισχυρισμού ότι ένας άνθρωπος είναι όντως αυτός που λέει ότι είναι, πιστοποίηση (authentication), δηλαδή η απόδειξη ότι είναι πράγματι αυτός που ισχυρίζεται και εξουσιοδότηση(authorization) που έχει να κάνει με την πρόσβαση σε συγκεκριμένους πόρους εφόσον πρώτα έχει πιστοποιηθεί.

2.1. Συμμετρική κρυπτογραφία

Στη συμμετρική κρυπτογραφία μια λιγότερο διαδεδομένη μορφή κρυπτογράφησης χρησιμοποιείται μια μορφή κλειδιού που είναι υπεύθυνη για την κρυπτογράφηση αλλά και ην αποκρυπτογράφηση της. Ο λόγος που δεν χρησιμοποιείται συχνά πλέον είναι λόγω προβλημάτων που δημιουργούνται στη διαχείριση των κλειδιών

2.2. Ασύμμετρη κρυπτογραφία

Στην ασύμμετρη κρυπτογραφία χρησιμοποιούνται δύο συμπληρωματικά είδη κλειδιών ένα ονομάζεται δημόσιο και ένα ιδιωτικό. Δεδομένα που κρυπτογραφούνται με δημόσιο κλειδί αποκρυπτογραφούνται με το ιδιωτικό κλειδί και το αντίθετο. Από την στιγμή που τα συμπληρωματικά κλειδιά δημιουργηθούν το ένα ορίζεται ως δημόσιο και το άλλο ως ιδιωτικό. Το ιδιωτικό δεν δημοσιοποιείται σε τρίτους και το γνωρίζει μόνο αυτός που το κατέχει, ενώ το δημόσιο κλειδί είναι διαθέσιμο σε κάποιον ενδιαφερόμενο. Έτσι οποιοσδήποτε χρησιμοποιώντας το δημόσιο κλειδί μπορεί να κρυπτογραφήσει ένα μήνυμα που μόνο ο κάτοχος του ιδιωτικού κλειδιού είναι σε θέση να αποκρυπτογραφήσει. Από την άλλη πλευρά ο κάτοχος του ιδιωτικού κλειδιού μπορεί να ταυτοποιηθεί καθώς μόνο αυτός μπορεί να στέλνει μηνύματα κρυπτογραφημένα με το ιδιωτικό κλειδί που κατέχει τα οποία θα έχουν νόημα εάν αποκρυπτογραφηθούν από το δημόσιο κλειδί που μπορεί να έχει πρόσβαση ο οποιοσδήποτε. Η **ασύμμετρη κρυπτογραφία** χρησιμοποιείται για την ταυτοποίηση λογαριασμών και την εξουσιοδότηση διαφόρων συναλλαγών στο blockchain.Τα δεδομένα των συναλλαγών εμπεριέχουν κλειδιά(δημόσια) για την ταυτοποίηση των λογαριασμών. Από την άλλη μεριά ο ιδιοκτήτης του λογαριασμού που παραδίδει την πληροφορία ενός πόρου μέσω μιας συναλλαγής κρυπτογραφεί ένα κείμενο με το ιδιωτικό κλειδί που κατέχει. Οι υπόλοιποι χρήστες μπορούν να επιβεβαιώσουν την εγκυρότητα της συναλλαγής χρησιμοποιώντας το δημόσιο κλειδί που όπως αναφέρθηκε παραπάνω είναι ο αριθμός του λογαριασμού του ιδιοκτήτη.

2.3. Ψηφιακές υπογραφές

Οι ψηφιακές υπογραφές χρησιμοποιούνται για τον έλεγχο της κατοχής των δεδομένων εφόσον υπάρχει κάποιος που ισχυρίζεται ότι του ανήκουν[5]. Η δημιουργία ψηφιακής υπογραφής γίνεται υπολογίζοντας την hash τιμή των δεδομένων και κρυπτογραφώντας την με το ιδιωτικό κλειδί του κατόχου των δεδομένων. Τα αρχικά δεδομένα και η υπογραφή τοποθετούνται μαζί σε ένα αρχείο και

αποτελούν ένα ψηφιακά υπογεγραμμένο μήνυμα του κατόχου των προαναφερθέντων δεδομένων. Η επαλήθευση του γεγονότος ότι όντως τα δεδομένα ανήκουν σε αυτόν που το ισχυρίζεται γίνεται λαμβάνοντας το υπογεγραμμένο αρχείο και εφαρμόζοντας στα δεδομένα του την hash συνάρτηση. Εφόσον η τιμή που θα προκύψει είναι όμοια με αυτήν που θα επιστραφεί αποκρυπτογραφώντας την ψηφιακή υπογραφή που περιέχεται στο αρχείο τα δεδομένα ανήκουν σε αυτόν που έκανε τον ισχυρισμό. Αντίστοιχα γίνεται και η αναγνώριση της απάτης

2.4 Κατακερματισμός

Οι συναρτήσεις κατακερματισμού τα λεγόμενα hashfunctions είναι προγράμματα που μετασχηματίζουν σε ακολουθίες ψηφίων σταθερού μήκους όλα τα είδη και ποσότητες δεδομένων[6][7] (Introduction to Merkle Tree, 2018). Οι συναρτήσεις αυτές πρέπει να εκτελούνται γρήγορα και να είναι ντετερμινιστικές. Ενώ υπάρχουν διάφορες συναρτήσεις κατακερματισμού η σημαντικότερη κατηγορία είναι οι κρυπτογραφικές συναρτήσεις. Τα κύρια χαρακτηριστικά της είναι:

1. Έχει μοναδική κατεύθυνση δηλαδή δεν μπορεί να υπολογισθεί η είσοδος ενώ υπάρχει παραγμένη δεδομένη έξοδος
2. Είναι τυχαία (ψευδοτυχαία) που σημαίνει ότι η επιστρεφόμενη τιμή της συνάρτησης κατακερματισμού διαφέρει κάθε φορά που αλλάζει η είσοδος και μάλιστα μη προβλέψιμα
3. Έχει ανοχή στις «συγκρούσεις» που σημαίνει ότι σχεδόν απίθανο να εμφανιστούν παραπάνω από ένα στιγμιότυπο εισόδου τα οποία να επιστρέφουν την ίδια έξοδο
4. Οι συναρτήσεις κατακερματισμού έχουν τη δυνατότητα να εφαρμοστούν σε δεδομένα από τη σύνδεση διαφορετικών και ανεξάρτητων τμημάτων δεδομένων. Από τους πιο αποδοτικούς μηχανισμούς για την επίτευξη του παραπάνω είναι τα MerkleTrees.

2.4.1.. Εφαρμογές κατακερματισμού

1. Σύγκριση δεδομένων

Οι συναρτήσεις κατακερματισμού χρησιμοποιούνται στη σύγκριση δεδομένων. Μια περίπτωση είναι ναι υπολογισθούν οι κρυπτογραφιμένες μορφές hash τιμών διαφόρων κειμένων ώστε να εντοπιστούν κοινά κείμενα με τη σύγκριση των hash τιμών τους. Λόγω της ανθεκτικότητας στις «συγκρούσεις» των κρυπτογραφικών hash συναρτήσεων ,όπως αναφέρθηκε προηγμένως, το αποτέλεσμα είναι ακριβές.

2. Ανίχνευση αλλαγών σε δεδομένα

Με το συνεχή υπολογισμό, σε διάφορες χρονικές στιγμές, των κρυπτογραφικών hash τιμών ορισμένων δεδομένων, λαμβάνεται συμπέρασμα για την πιθανότητα αλλαγής των δεδομένων στο χρονικό αυτό διάστημα. Η σύγκριση γίνεται μεταξύ των hash τιμών που αναφέρθηκαν προηγουμένως και εάν διαφέρουν μεταξύ τους τότε υπήρξε τροποποίηση των δεδομένων.

3. Αποθήκευση δεδομένων με δυνατότητα ανίχνευσης αλλαγών

Ένα MerkleTree είναι μία δενδρική δομή από μπλοκ δεδομένων που αποτελούν ένα μεγαλύτερο μπλοκ[8]. Με το Merkletree υπάρχει δυνατότητα ανίχνευσης για οποιοδήποτε από τα επιμέρους μπλοκ δεδομένων υποστεί αλλαγή. Αυτό είναι δυνατό γιατί υπάρχει αλλαγή στο σύνολο των δεδομένων.. Η ιδιαιτερότητα του MerkleTree είναι ότι σε οποιοδήποτε από τα επιμέρους μπλοκ δεδομένων που βρίσκονται στη βάση του δένδρου υπάρξει τροποποίηση τότε αυτόματα αλλάζει και η hash τιμή στη ρίζα του δένδρου. Η αποθήκευση συναλλαγών σε μορφή δεδομένων με τρόπο

που να είναι ανιχνεύσιμη η αλλαγή σε οποιαδήποτε από τις συναλλαγές ενός μπλοκ εφαρμόζεται στο blockchain.

4. Εκτέλεση υπολογιστικά δύσκολων εργασιών

Οι συναρτήσεις κατακερματισμού βρίσκουν εφαρμογή και στα λεγόμενα hashpuzzles. Στα hashpuzzles δίνονται δεδομένα για αποθήκευση που προφανώς δεν επιτρέπεται να τροποποιηθούν και το λεγόμενο nonce που είναι μια μορφή δεδομένων με σκοπό να αλλάζει κάθε φορά που χρησιμοποιείται. Ταυτόχρονα χρησιμοποιείται και μια κρυπτογραφική συνάρτηση κατακερματισμού. Το ζητούμενο είναι να παραχθεί μια τιμή hash από το κρυπτογράφηση του nonce και των δεδομένων προς αποθήκευση και αυτή η τιμή hash πρέπει να πληροί συγκεκριμένα κριτήρια τα οποία ορίζουν την δυσκολία των hashpuzzle και εξαρτάται από αυτά η αποδοχή της λύσης. Το χαρακτηριστικό των hashpuzzle βρίσκεται στον τρόπο επίλυσης τους. Για να βρεθεί η απάντηση πρέπει να δοκιμαστούν διάφορες τιμές του μέχρι να βρεθεί μία τιμή που να ικανοποιεί τους περιορισμού που προαναφέρθηκαν. Για την επαλήθευση της απάντησης αρκεί να κρυπτογραφηθούν τα αρχικά δεδομένα και το nonce με την κρυπτογραφική συνάρτηση και το αποτέλεσμα να ικανοποιεί τους περιορισμούς που είχαν τεθεί στο puzzle. Αν υπάρξει κάποια αλλαγή στα δεδομένα ή στο nonce τότε η τρέχουσα τιμή hash θεωρείται μη έγκυρη.

3. BlockchainTechnologies

3.1. Ορισμός και λειτουργία του Blockchain

Το Blockchain μπορεί να θεωρηθεί μια κατανεμημένη βάση δεδομένων στην οποία αποθηκεύονται ψηφιακές συναλλαγές(transactions) και καταγραφές ψηφιακών συμβάντων(events) σε κρυπτογραφημένη μορφή[11][12]. Οποιαδήποτε πληροφορία βρίσκεται μέσα στο blockchain είναι ανοιχτή για όλους τους συμμετέχοντες. Για τον παραπάνω λόγο το blockchain ορίζεται και σαν ένα κοινό καθολικό(ledger)[13]. Κάθε συναλλαγή επαληθεύεται με έναν αλγόριθμο κοινής συναίνεσης του συστήματος. Ο αλγόριθμος αυτός διαφέρει στα διάφορα Blockchain δίκτυα. Από τη στιγμή που μια πληροφορία εισαχθεί μέσα στο blockchain είναι αδύνατον να σβηστεί και είναι επαληθεύσιμη γι' αυτό το λόγο θεωρείται απροσπέλαστο σύστημα. Ο όρος blockchain επινοήθηκε από τον τρόπο με τον οποίο αποθηκεύονται τα δεδομένα μέσα σε αυτό καθώς αυτά οργανώνονται σε μονάδες που ονομάζονται blocks και συνδέονται σε χρονολογική σειρά σαν αλυσίδα.

Βασικά χαρακτηριστικά του Blockchain:

Το Blockchain είναι μία τεχνολογία χτισμένη πάνω σε σημαντικές ιδέες της εποχής. Τα τέσσερα βασικά χαρακτηριστικά του είναι τα παρακάτω[14][15]:

- **Κατανομή**: Σε αντίθεση με τα συγκεντρωτικά δίκτυα όπου κάθε πληροφορία πρέπει να γίνει αποδεκτή από μία κεντρική οντότητα του συστήματος την οποία πρέπει να εμπιστεύονται όλοι οι χρήστες στο blockchain δε χρειάζεται εμπιστοσύνη μεταξύ των χρηστών και των συμμετεχόντων του δικτύου. Η ιδιότητα του διαμεσολαβητή είναι περιττή εφόσον το δίκτυο εξασφαλίζει την ασφάλεια όλων των πληροφοριών που αποθηκεύονται σε αυτό με τη χρήση των αλγόριθμων συναίνεσης που συγκρατούν τη συνοχή του δικτύου.
- **Ανθεκτικότητα**: Κάθε συναλλαγή επικυρώνεται ταχύτατα εφόσον είναι έγκυρη ενώ οι έντιμοι συμμετέχοντες θα "καταψηφίσουν" οποιαδήποτε μη έγκυρη. Θεωρείται αδύνατον να διαγραφούν ή να μεταγραφούν όλες οι συναλλαγές από την στιγμή που αποθηκεύονται στο blockchain. Σε περίπτωση που σε ένα block καταγραφεί μία μη έγκυρη συναλλαγή μπορεί εύκολα να ανακτηθεί.
- **Ανωνυμία**: Για την συμμετοχή ενός χρήστη σε μία συναλλαγή χρησιμοποιείται το ζευγάρι των κλειδιών που βρίσκεται στην κατοχή τους. Το δημόσιο κλειδί αντιπροσωπεύει τον χρήστη του αλλά το όνομα του δεν φαίνεται πουθενά επειδή δεν είναι απαραίτητο για την συναλλαγή. Σημειώνεται ότι το blockchain δεν εγγυάται απόλυτη ανωνυμία λόγο ενδογενών περιορισμών
- **Επαληθευσιμότητα**: Σε κάθε blockchain δίκτυο υπάρχει μια μορφή κρυπτονομίσματος που θα αναφερθεί παρακάτω αναλυτικότερα. Το ζευγάρι κλειδιών με το οποίο ο χρήστης συνδέεται στο δίκτυο αντιστοιχίζεται με το ποσό των κρυπτονομισμάτων που κατέχει. Κάθε φορά που ο χρήστης συμμετέχει σε μία συναλλαγή ενημερώνεται και το υπόλοιπο του και η πληροφορία αυτή αποθηκεύεται στο block που εμπεριέχει τη συναλλαγή. Με αυτό τον τρόπο είναι εύκολο να εντοπιστεί και να αξιολογηθεί οποιαδήποτε συναλλαγή μέσα στο δίκτυο.

3.2. Δομή των Blocks

Αναφορικά με την αρχιτεκτονική του Blockchain, κάθε block αποτελείται από έναν blockHeader και το blockBody[16]. Μέσα στο blockHeader αποθηκεύονται οι πληροφορίες του block και εκεί περιλαμβάνονται συγκεκριμένα:

- Blockversion: Υποδεικνύει τους κανόνες επικύρωσης του block.
- Ρίζα Merkletree: Την κατακερματισμένη μορφή όλων των συναλλαγών που αποθηκευτήκαν στο συγκεκριμένο block
- Χρονική σήμανση: Υποδεικνύει την ώρα δημιουργίας του block σε δευτερόλεπτα από την ημερομηνία Ιανουάριος- 1- 1970.
- nBits: Στοχευμένο όριο της κατακερματισμένης μορφής του έγκυρου block
- Nonce: πεδίο τεσσάρων byte το οποίο ξεκινάει από το μηδέν και αυξάνεται για κάθε υπολογισμό κατακερματισμού
- Parentblockhash: Η κατακερματισμένη μορφή του προηγούμενου block σε 256-bit

Στο blockbody αποθηκεύονται κάποιες συναλλαγές και events μαζί με έναν μετρητή. Ο αριθμός των πληροφοριών που βρίσκονται σε ένα block εξαρτάται από το μέγεθος του block και της κάθε συναλλαγής. Για παράδειγμα το block του bitcoinblockchain έχει μέγεθος ενός megabyte. Για την επαλήθευση των συναλλαγών χρησιμοποιείται ασύμμετρη κρυπτογραφία και σε ένα περιβάλλον στο οποίο δεν υπάρχει εμπιστοσύνη μεταξύ των συμμετεχόντων γίνεται χρήση ασύμμετρα κρυπτογραφημένων ψηφιακών υπογραφών. Κάθε χρήστης του δικτύου κατέχει ένα ζευγάρι ιδιωτικού και δημόσιου κλειδιού. Το δημόσιο κλειδί χρησιμοποιείται για την αναγνώριση του χρήστη ενώ το ιδιωτικό στις υπογραφές των ψηφιακών συναλλαγών. Θεωρείται απαραίτητο ο χρήστης να είναι ο μόνος που κατέχει το δικό του ιδιωτικό κλειδί, οπότε όταν ένας χρήστης συμμετέχει σε μία συναλλαγή υπογράφει με το ιδιωτικό του κλειδί και στα δεδομένα της συναλλαγής φαίνεται το δημόσιο κλειδί.

3.3 Genesisblock

Το Genesisblock αποτελεί το πρώτο μπλοκ μιας αλυσίδας blockchain. Πλέον αναφέρεται ως μπλοκ μηδέν αλλά παλαιότερα συνηθιζόταν να ονομάζεται μπλοκ ένα. Το Genesisblock είναι συνήθως κωδικοποιημένο με λογισμικό εφαρμογών του blockchain που χρησιμοποιείται. Εφόσον είναι το πρώτο μπλοκ του blockchain δεν έχει προηγούμενο και δεν αναφέρει πληροφορίες για προηγούμενα μπλοκς. Αντίθετα δημιουργεί τις συνθήκες του μπλοκ οι οποίες αποθηκεύονται μέσα σε αυτό. Στο παρακάτω πίνακα (3.1) φαίνονται τα χαρακτηριστικά ενός Genesisblock. Από αριστερά στην ισότητα υπάρχουν οι μεταβλητές που αναφέρουν τα χαρακτηριστικά που έχει ένα Genesisblock και στη συνέχεια ένα blockchain πρέπει να έχει. Δεξιά στην ισότητα είναι δεδομένα των μεταβλητών. Ο συγκεκριμένος πίνακας εμφανίζει τα χαρακτηριστικά ενός Genesisblock σε bitcoinblockchain.

GetHash()= 0x000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
hashMerkleRoot = 0x4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b
txNew.vin[0].scriptSig = 486604799 4
0x736B6E616220726F662074756F6C69616220646E6F63657320666F206B6E697262206E6F20726
F6C6C65636E61684320393030322F6E614A2F33302073656D695420656854
txNew.vout[0].nValue = 5000000000
txNew.vout[0].scriptPubKey =
0x5F1DF16B2B704C8A578D0BBAF74D385CDE12C11EE50455F3C438EF4C3FBCF649B6DE611
FEAE06279A60939E028A8D65C10B73071A6F16719274855FEB0FD8A6704 OP_CHECKSIG
block.nVersion = 1
block.nTime = 1231006505
block.nBits = 0x1d00ffff
block.nNonce = 2083236893

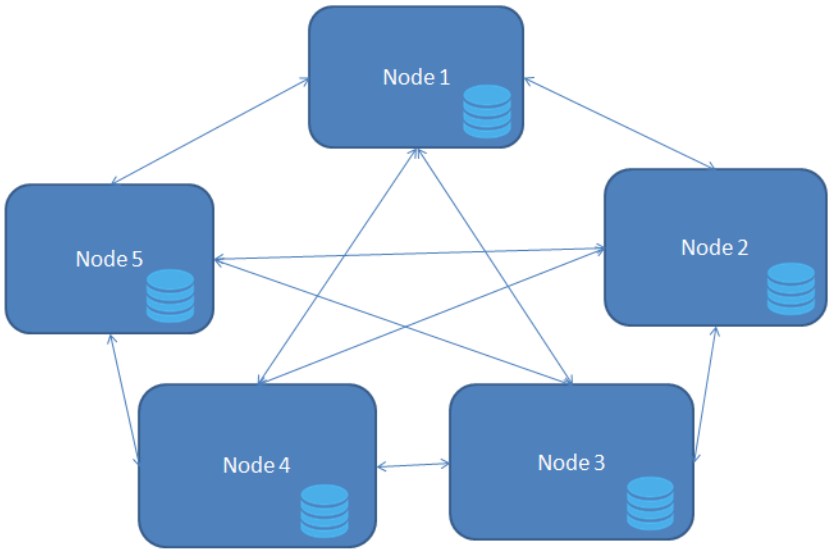
CBlock(hash=000000000019d6, ver=1, hashPrevBlock=00000000000000, hashMerkleRoot=4a5e1e,
nTime=1231006505, nBits=1d00ffff, nNonce=2083236893, vtx=1)
CTransaction(hash=4a5e1e, ver=1, vin.size=1, vout.size=1, nLockTime=0)
CTxIn(COutPoint(000000, -1), coinbase
04ffff001d0104455468652054696d65732030332f4a616e2f32303039204368616e63656c6c6f72206f6
e206272696e6b206f66207365636f6e64206261696c6f757420666f722062616e6b73)
CTxOut(nValue=50.00000000, scriptPubKey=0x5F1DF16B2B704C8A578D0B)
vMerkleTree: 4a5e1e

Πίνακας 3.1 [17]

Για την δημιουργία ενός ιδιωτικού ethereumblockchain δικτύου(εικόνα 1.1) δημιουργείται ένα genesisblock με τα χαρακτηριστικά του πίνακα που φαίνεται παρακάτω (Πινακας 3.2)

```
{
   “config”: {
      “chainId”: 0,
      “homesteadBlock”: 0,
      “eip155Block”: 0,
      “eip158Block”: 0
            },
    “nonce”: “0x0000000000000042”,
   “timestamp”: “0x0”,
   “parentHash”:
“0x0000000000000000000000000000000000000000000000000000000000000000”,
   “extraData”: “0x0”,
   “gasLimit”: “0x8000000”,
   “difficulty”: “0x400”,
   “mixhash”: “0x0000000000000000000000000000000000000000000000000000000000000000”,
   “coinbase”: “0x3333333333333333333333333333333333333333”, “alloc”: { }
}
```

Πίνακας 3.2



Εικόνα 1.1 Δίκτυο με πέντε κόμβους που στον καθένα είναι αποθηκευμένο το ίδιο blockchain.

3.3.1 Εξήγηση παραμέτρων [18]

1) Config: Η πρώτη παράμετρος του config είναι το chainId και είναι το μοναδικό γνώρισμα του blockchain ενώ οι υπόλοιπες τρεις μεταβλητές αναφέρονται σε κάποιες σταθερές του ethereum που σε ένα ιδιωτικό δίκτυο δεν είναι απαραίτητες.

2) Nonce: πεδίο 64-bit που χρησιμοποιείται σε συνδυασμό με το mixhash για να αποδείξει ότι η υπολογιστική ισχύς που χρησιμοποιήθηκε ήταν επαρκής για το μπλοκ.

3) Timestamp: το Timestamp είναι η τιμή που είναι ίση με την έξοδο του χρόνου κατά την έναρξη του μπλοκ. Χρησιμοποιείται στον μηχανισμό που επιβάλλει την ομοιόσταση ως προς τον χρόνο. Εφόσον η διαφορά χρόνου μεταξύ δύο μπλοκ είναι μικρότερη από την ορισμένη τιμή τότε αυξάνεται το επίπεδο δυσκολίας και την εγκυρότητα του επόμενου μπλοκ θα χρειαστεί επιπλέον υπολογιστική ισχύ. Αντίστροφα η δυσκολία μειώνεται εάν η χρονική διαφορά είναι μεγαλύτερη από την ορισμένη τιμή. Έστω ότι σε ένα blockchain δίκτυο τα μπλοκ πρέπει να εντάσσονται στην αλυσίδα κάθε δέκα λεπτά. Εάν ένα μπλοκ ενταχτεί στα επτά λεπτά, τότε πρέπει το δίκτυο να δυσκολέψει τον αλγόριθμο ώστε το επόμενο μπλοκ να δημιουργηθεί και να τοποθετηθεί στην αλυσίδα σε δέκα λεπτά.

4) Parenthash: Ως Parenthash ορίζεται το hashkeccak 256-bit της κεφαλίδας το γονικού μπλοκ. Έχει τη μορφή δείκτη και απαιτείται για τον σχηματισμό της αλυσίδας από μπλοκ. Παρομοιάζεται με συνδετική λίστα σε επίπεδο προγραμματιστικό. Το genesisblock δεν έχει γονικό μπλοκ οπότε τοποθετείται μηδέν στην τιμή του.

5) Extradata: Ένας αυθαίρετος πίνακας από bytes που περιέχει δεδομένα σχετικά με το genesisblock. Θεωρείται προαιρετική παράμετρος και δεν μπορεί να ξεπεράσει τα 32 byte.

6) gasLimit: Το gasLimit είναι μια βαθμιαία τιμή ίση με το τρέχον όριο δαπανών gas ανά μπλοκ. Στις συναλλαγές υπάρχει επίσης παράμετρος gasLimit και το συνολικό άθροισμα τους δε μπορεί να ξεπερνά το όριο του block.

7) Difficulty: Αυτή η παράμετρος είναι μια βαθμιαία τιμή που αντιστοιχεί στο επίπεδο δυσκολίας ενός συγκεκριμένου μπλοκ. Για τον υπολογισμό του επιπέδου δυσκολίας ενός μπλοκ χρησιμοποιείται ως παράμετρος η δυσκολία και η χρονική σήμανση του προηγούμενου αλλά στην περίπτωση του genesisblock που δεν υπάρχει προηγούμενο μπλοκ η δυσκολία καθορίζεται στην ίδια την παράμετρο. Η χρησιμότητα του επιπέδου δυσκολίας είναι στον έλεγχου του χρόνου της δημιουργίας των μπλοκ του blockchain για να διατηρηθεί συγκεκριμένη συχνότητα. Εάν η δυσκολία του προηγούμενου μπλοκ είναι υψηλή τότε οι miners πρέπει να χρησιμοποιήσουν περισσότερη υπολογιστική ισχύ, ενώ αν είναι χαμηλή τότε πρέπει να χρησιμοποιήσουν λιγότερη για την επικύρωση ενός μπλοκ.

8) mixHash: Το mixHash είναι μια hash μορφή 256-bit που σε συνδυασμό με το nonce αποδεικνύει ότι η υπολογιστική ισχύς που χρησιμοποιήθηκε για την δημιουργία του μπλοκ είναι επαρκής.

9) Coinbase: Μία διεύθυνση ethereum 160-bit όπου αποθηκεύονται όλες οι ανταμοιβές που συλλέγονται από την επικύρωση του μπλοκ. Η ανταμοιβή είναι το άθροισμα της ανταμοιβής εξόρυξης και των επιστροφών από την εκτέλεση συμβατικών συναλλαγών. Δεδομένου ότι αναφέρεται στο genesisblock η τιμή για αυτό το μπλοκ μπορεί να είναι οτιδήποτε. Για όλα τα επόμενα μπλοκ, η τιμή θα είναι μια διεύθυνση που θα καθοριστεί από τον miner που επικύρωσε αυτό το μπλοκ.

10) Alloc: Αυτή η παράμετρος χρησιμοποιείται για την προχρηματοδότηση σε ether ορισμένων διευθύνσεων. Περιέχει μία παραμέτρο που αναφέρονται οι διευθύνσεις και το ποσό του ether που θα λάβουν.

3.4. Δημιουργία ενός νέου μπλοκ

Για την πρόσθεση ενός μπλοκ στο blockchain δίκτυο το οποίο περιέχει δεδομένα νέων συναλλαγών απαιτείται η παρακάτω διαδικασία[18]:

1) Δημιουργία του merkletree. Το merkletree περιέχει τα δεδομένα των συναλλαγών που θα προστεθούν στο blockchain δίκτυο.
2) Το hash του προηγούμενου μπλοκ και η ρίζα του merkletree εμπεριέχονται στην κεφαλίδα του μπλοκ που δημιουργείται
3) Τέλος δημιουργείται ένα νέο hash για την κεφαλίδα του νέου μπλοκ η οποία πλέον είναι η τρέχουσα κεφαλή στο blockchain δίκτυο μέχρι να προστεθεί το επόμενο μπλοκ.

Η διαφορά του blockchain και μίας κοινής βάσης δεδομένων είναι στην αποθήκευση των δεδομένων. Μία κοινή βάση δεδομένων βρίσκεται σε έναν υπολογιστή και είναι πολύ εύκολο να υπάρξει τροποποίηση των δεδομένων της ενώ ταυτόχρονα είναι ευάλωτη σε επιθέσεις. Στο blockchain από την άλλη πλευρά τα δεδομένα βρίσκονται σε πολλούς κόμβους, ως αντίγραφα, οι οποίοι είναι συγχρονισμένοι μεταξύ τους μέσω ενός δικτύου. Το σύνηθες δίκτυο είναι το internet λόγω της ευκολίας στην πρόσβαση το οποίο αποτελεί ταυτόχρονα μία φθηνή λύση δικτύωσης. Η επικοινωνία επιτυγχάνεται μέσω μηνυμάτων μεταξύ των κόμβων και συμβαίνει για την ενημέρωση τους σε αλλαγές στο blockchain. Κάθε κόμβος κατέχει μια λίστα με ομότιμους κόμβους και μπορεί να επικοινωνήσει με αυτούς απευθείας. Η λίστα είναι υποσύνολο των κόμβων του δικτύου. Σε περίπτωση που κάποιος κόμβος μίας λίστας δεν ανταποκρίνεται αντικαθιστάται με έναν άλλον κόμβο. Για αυτόν τον λόγο δημιουργείται ένας αριθμός συνδέσεων από κόμβους που υπάρχουν στο δίκτυο για κάθε νέο κόμβο που προστίθεται στο blockchain.

Για την διασφάλιση της εγκυρότητας των μπλοκ στο blockchain δίκτυο χρησιμοποιείται ένα σύστημα ανταγωνιστικότητας μεταξύ των κόμβων διότι μετά από κάθε επιτυχή τοποθέτηση ενός μπλοκ υπάρχει μία ανταμοιβή για τον κόμβο που ήταν υπεύθυνος για την τοποθέτηση του. Κάθε κόμβος έχει δύο ευθύνες, η μία είναι η συνεχής προσπάθεια για την δημιουργία και τοποθέτηση ενός μπλοκ στην αλυσίδα αφού πρώτα ελεγχτεί η εγκυρότητα του μπλοκ από άλλους κόμβους και δεύτερον να ελέγχει ο ίδιος την εγκυρότητα των μπλοκ που δημιουργούνται από τους υπόλοιπους κόμβους. Η διαδικασία έχει ως εξής:

1) Όταν οι κόμβοι λάβουν ένα μπλοκ που χρειάζεται να επαληθευθεί, γίνεται άμεση προτεραιότητά τους η επεξεργασία του.
2) Οι κόμβοι λαμβάνουν κάποια δεδομένα που έχουν να κάνουν με συναλλαγές και μπλοκ τα οποία πρέπει να επαληθεύσουν εφόσον είναι έγκυρα. Έπειτα πρέπει να τα προωθήσουν στους κόμβους με τους οποίους υπάρχει επικοινωνία.
3) Εφόσον τα νέα δεδομένα συναλλαγών του κόμβου συγκεντρωθούν στο merkletree δημιουργείται ένα νέο μπλοκ για το οποίο πρέπει να λυθεί ένα hashpuzzle και ύστερα στέλνεται για επιβεβαίωση στους υπόλοιπους κόμβους.
4) Τα έγκυρα μπλοκ προστίθενται στο αντίγραφο του blockchain του κάθε κόμβου.
5) Εάν υπάρχουν δεδομένα συναλλαγών σε έναν κόμβο και αυτά τα δεδομένα τοποθετηθούν σε μπλοκ τότε όλοι οι κόμβοι που είχαν λάβει αυτά τα δεδομένα είναι υποχρεωμένοι να τα διαγράψουν.
6) Όταν ένας κόμβος εισάγει ένα νέο μπλοκ στο blockchain κερδίζει μια αμοιβή για όλες τις συναλλαγές που αποθηκεύτηκαν στο μπλοκ.

3.5. Αλγόριθμοι κοινής συναίνεσης

Είναι γνωστό ότι το blockchain είναι ένα κατανεμημένο δίκτυο που παρέχει ασφάλεια και τη δυνατότητα να μην χρειάζεται να υπάρχει εμπιστοσύνη. Δεν υπάρχει κάποια μορφή κεντρικής αρχής για την επικύρωση και την επαλήθευση μίας συναλλαγής. Κάθε συναλλαγή θεωρείται ασφαλής και επαληθευμένη[19]. Αυτό συμβαίνει διότι υπάρχει ένα πρωτόκολλο συναίνεσης σε κάθε blockchain δίκτο. Ο αλγόριθμος συναίνεσης είναι μια διαδικασία που μέσω αυτής όλοι οι κόμβοι του δικτύου συμφωνούν με την παρούσα κατάσταση του κατανεμημένου καθολικού. Έτσι οι αλγόριθμοι αυτοί προσδίδουν αξιοπιστία στο δίκτυο και δημιουργούν ένα κλίμα ασφάλειας μεταξύ των κόμβων χωρίς να υπάρχει εμπιστοσύνη μεταξύ τους. Στην ουσία ένα πρωτόκολλο συναίνεσης διασφαλίζει ότι κάθε μπλοκ που προστίθεται στο blockchain θα είναι μοναδικό διότι θα είναι αποδεχτό από κάθε κόμβο του δικτύου. Το πρωτόκολλο συναίνεσης έχει συγκεκριμένους στόχος όπως για παράδειγμα η επίτευξη μιας συμφωνίας, συνεργασία, ίσα δικαιώματα μεταξύ των κόμβων και υποχρεωτική συμμετοχή κάθε κόμβου στη διαδικασία κοινής συναίνεσης. Με αυτό τον τρόπο επιτυγχάνεται ο στόχος της συναίνεσης δηλαδή μία κοινή συμφωνία από όλο το δίκτυο.

3.5.1. ProofofWork

Ο συγκεκριμένος αλγόριθμος δημοσιεύτηκε το 1993 και χρησιμοποιήθηκε αργότερα από τον Shatoshi Nakamoto[20] στο bitcoin blockchain. Το proofofwork χρησιμοποιείται στις περισσότερες κρυπτοοικονομίες. Βασική του ιδέα είναι μια λύση που είναι δύσκολο να την εντοπιστεί αλλά εύκολο να επιβεβαιωθεί. Ο σκοπός αυτού του αλγορίθμου συναίνεσης είναι να έρθουν όλοι οι κόμβοι ενός δικτύου σε συμφωνία, ένα κλίμα που δεν χρειάζεται εμπιστοσύνη. Όλες οι συναλλαγές γίνονται αποδεκτές και το νέο μπλοκ τοποθετείται στην αλυσίδα. Οι κόμβοι χρησιμοποιούν επεξεργαστική ισχύ για να λύσουν ένα hashpuzzle ώστε να προσθέσουν το μπλοκ στο δίκτυο και από αυτό βγήκε και το όνομα του αλγορίθμου. Με το χρόνο και καθώς το δίκτυο μεγαλώνει τα hashpuzzles δυσκολεύουν. Η διαδικασία αυτή ονομάζεται mining και οι κόμβοι που συμμετέχουν σε αυτήν miners. Η διαδικασία επαλήθευσης των συναλλαγών στο μπλοκ που θα προστεθούν, η οργάνωση των συναλλαγών με χρονολογική σειρά στο μπλοκ και η ανακοίνωση του μπλοκ που εξορύσσεται σε ολόκληρο το δίκτυο δεν απαιτεί πολύ ενέργεια και χρόνο. Το μέρος που καταναλώνει ενέργεια είναι η επίλυση του hashpuzzle για να συνδέθει το νέο μπλοκ με το τελευταίο μπλοκ στο δίκτυο. Όταν ένας κόμβος (miner) λύσει το hashpuzzle το στέλνει στους υπόλοιπους κόμβους του δικτύου και κερδίζει ένα έπαθλο σε μορφή κρυπτονομίσματος που παρέχεται από το πρωτόκολλο.

3.5.2. Proofofstake

Σε αυτόν τον τύπο αλγορίθμου συναίνεσης οι επικυρωτές επενδύουν στα νομίσματα του συστήματος δίνοντας μερικά από τα δικά τους ως μερίδιο. Έπειτα όλοι οι επικυρωτές θα ξεκινήσουν να επικυρώνουν τα μπλοκ. Τοποθετούν μια μορφή στοιχήματος σε ένα μπλοκ που πιστεύουν ότι μπορεί να τοποθετηθεί στο δίκτυο. Με βάση τα μπλοκ τα οποία όντως τοποθετήθηκαν στο blockchain δίκτυο, οι επικυρωτές λαμβάνουν ανταμοιβή ανάλογα με τα «στοιχήματα» που έβαλαν. Στο τέλος επιλέγετε ο επικυρωτής που θα δημιουργήσει το μπλοκ με βάση το οικονομικό του μερίδιο στο δίκτυο. Το ethereum έχει μετατοπιστεί σε αυτόν τον αλγόριθμο.

3.5.3.Proofofburn

Με αυτόν τον αλγόριθμο οι επικυρωτές θυσιάζουν ένα ποσό από κρυπτονομίσματα στέλνοντας τα σε μια διεύθυνση από την οποία είναι αδύνατον να ανακτηστούν. Με αυτήν την δέσμευση οι επικυρωτές κερδίζουν το προνόμιο της εξόρυξης στο δίκτυο με βάση μια διαδικασία τυχαίας επιλογής. Με αυτόν τον αλγόριθμο συναίνεσης οι επικυρωτές βραχυπρόθεσμα χάνουν ένα μικρό ποσό αλλά μακροπρόθεσμα κερδίζουν ένα μεγαλύτερο. Όσο περισσότερα είναι τα νομίσματα που καίνε, τόσες περισσότερες είναι οι πιθανότητες τους να εκλεγούν για το mining του νέου μπλοκ. Μία από τις αδυναμίες αυτού του πρωτοκόλλου είναι η συνεχής αποβολή νομισμάτων χωρίς ουσιαστικό λόγο.

3.5.4. Proofofcapacity

Με τον αλγόριθμο Proofofcapacity οι επικυρωτές επενδύουν χώρο από το σκληρό δίσκο τους. Όσο περισσότερος ο χώρος που επενδύουν τόσες περισσότερες οι πιθανότητες να εκλεγούν για το mining του επόμενου μπλοκ και να κερδίσουν την αντίστοιχη ανταμοιβή

3.5.5. ProofofElapsedTime

Ο συγκεκριμένος αλγόριθμος θεωρείται από τους δικαιότερους αλγορίθμους συναίνεσης που χρησιμοποιείται για την επιλογή του κόμβου που θα επικυρώσει το επόμενο μπλοκ. Χρησιμοποιείται περισσότερο στα permissionnedblockchain δίκτυα. Κάθε επικυρωτής έχει μία δίκαιη ευκαιρία για δημιουργήσει ένα μπλοκ. Οι κόμβοι περιμένουν ένα τυχαίο χρονικό διάστημα αφήνοντας κάθε φορά απόδειξη της αναμονής τους στο μπλοκ. Τα μπλοκ που έχουν δημιουργηθεί στέλνονται και στους υπόλοιπους κόμβους. Ο νικητής και κατά συνέπεια ο κόμβος που θα προσαρτήσει το νέο μπλοκ είναι αυτός που κατέχει την ελάχιστη τιμή χρόνου κατά την απόδειξη. Επίσης σε αυτόν τον αλγόριθμο υπάρχουν αρκετοί περιορισμοί που σταματούν κόμβους από το να κερδίζουν συνέχεια ή να παράγουν πάντα το μικρότερο χρόνο.

3.6. Bitcoin

Ένας χάκερ που χρησιμοποιούσε το ψευδώνυμο ShatoshiNakamoto τον Οκτώβριο του 2008 έκανε μία δημοσίευση για τη δημιουργία ενός ψηφιακού κρυπτονομίσματος με το όνομα bitcoin. Στις 9 Ιανουρίου του 2009 η πρώτη έκδοση του bitcoin λογισμικού παραδόθηκε από τον Nakamoto στο sourceforge(αποθετήριο λογισμικού). Έπειτα δημιούργησε μία ιστοσελίδα με όνομα bitcoin.org και μέχρι το 2010 συνεργάστηκε με μία ομάδα προγραμματιστών για την ανάπτυξη του λογισμικού bitcoin. Έπειτα αποσύρθηκε[21].

Η διπλή χρέωση (doublespend), δηλαδή η πιθανότητα δημιουργίας ενός πανομοιότυπου νομίσματος, είναι ένα σοβαρό πρόβλημα που αντιμετωπίζουν τα ψηφιακά νομίσματα. Τα φυσικά νομίσματα δεν έχουν σε μεγάλο βαθμό αυτό το πρόβλημα διότι πρώτον δεν είναι δύσκολο να εντοπιστούν και δεύτερον είναι δύσκολο να δημιουργηθούν. Επίσης υπάρχει η αντίστοιχη νομοθεσία οι οποία επιβάλει σοβαρές ποινές σε όσους υπέπεσαν σε αυτό το αδίκημα. Το bitcoin με την αξιοποίηση του blockchain κατάφερε να αποτρέψει τη διπλή χρέωση. Αυτό είναι δυνατό επειδή το blockchain χρησιμοποιείται για την αποθήκευση και την εγκυροποίηση των συναλλαγών[22]. Οι συναλλαγές αποθηκεύονται σε μπλοκ και συνδέονται μεταξύ τους δημιουργώντας μία αλυσίδα. Η εγκυρότητα την συναλλαγών είναι μια διαδικασία που απαιτεί υπολογιστική ισχύ και η ασφάλεια προέρχεται το υπερβολικό κόστος που απαιτείται για να τοποθετηθούν εσφαλμένα δεδομένα στο δίκτυο. Η αποτελεσματικότητα του bitcoin έχει φανεί τα τελευταία δέκα χρόνια και δηλώνει πως αυτός ο μηχανισμός είναι ιδιαίτερα χρήσιμος. Ένας από τους τρόπους με τους οποίους θα μπορούσε κάποιος να προσπελάσει την αμυντική χρήση του blockchain είναι η επίθεση 51%. Σε περίπτωση που κάποιος κακόβουλος χρήστης κατέχει παραπάνω από το 50% της υπολογιστικής ισχύς του blockchain δικτύου τότε μπορεί επικυρώσει οποιαδήποτε συναλλαγή και συνεπώς να ακυρώσει την ορθότητα των αποφάσεων του δικτύου. Το bitcoin έχει 30000 κόμβους που εργάζονται για την επαλήθευση των συναλλαγών. Στο bitcoin υπάρχουν αρκετές κατηγορίες κόμβων.

1) Οι πλήρεις κόμβοι που έχουν μεγάλη επεξεργαστική ισχύ και αποθηκευτικό χώρο ενώ ταυτόχρονα έχουν υψηλές ταχύτητες επικοινωνίας με το internet.
2) Οι mining κόμβοι που έχουν υψηλότερες απαιτήσεις επεξεργαστικής ισχύς.
3) Υπάρχει μία κατηγορία κόμβων η οποία κανει απλά επαλήθευση των συναλλαγών.
4) Άλλη μία κατηγορία είναι οι κόμβοι που εκτέλουν λογισμικά πορτοφολιών και ως ιδιαιτερότητα αποθηκεύουν μόνο τις κεφαλίδες των μπλοκ.
5) Η τελευταία κατηγορία είναι κόμβοι που χρησιμοποιούνται ως πύλες προς άλλα συστήματα όπως δίκτυα τραπεζών

Όπως σε κάθε blockchain δίκτυο με μια μορφή κρυπτονομίσματος έτσι και στο bitcoin είναι απαραίτητη η χρήση ενός πορτοφολιού(wallet). Το πορτοφόλι bitcoin είναι μια εφαρμογή διεπαφής χρήστη με το bitcoinblockchain που εμφανίζει το ποσό των bitcoins που έχει στην κατοχή του ένας λογαριασμός, δηλαδή το υπόλοιπο του. Ο τρόπος λειτουργίας του είναι ότι δημιουργεί και αποθηκεύει ζευγάρια κλειδιών που αντιστοιχούν σε λογαριασμούς στο bitcoinblockchain. Το πορτοφόλι bitcoin έχει και τη δυνατότητα να πραγματοποιήσει συναλλαγές μεταξύ λογαριασμών.

3.7. Ethereum

Το Ethereum είναι μια παγκόσμια, αποκεντρωμένη πλατφόρμα για συναλλαγές κρυπτονομισμάτων και δημιουργία νέων κατανεμημένων εφαρμογών[23][24]. Στο Ethereum, μπορεί να γραφτεί κώδικας που ελέγχει τις συναλλαγές κρυπτονομισμάτων μεταξύ λογαριασμών και μπορούν επίσης να δημιουργηθούν εφαρμογές προσβάσιμες οπουδήποτε στον κόσμο. Δημιουργήθηκε από VitalikButerin το 2013 και σαν κρυπτονόμισμα χρησιμοποιεί το ether. Το ethereumblockchain σε αντίθεση με το bitcoin δεν είναι απλά μια πλατφόρμα για συναλλαγές. Σε αυτή τη πλατφόρμα υπάρχει η δυνατότητα να γραφτεί κώδικας που ονομάζεται έξυπνο συμβόλαιο και να αποδώσει στην πλατφόρμα επιπλέον λειτουργικότητα. Κάποια παραδείγματα είναι η ανάπτυξη έξυπνου συμβολαίου για διαδικτυακή ψηφοφορία ή συλλογικές χρηματοδοτήσεις ή για αποθήκευση περιουσιακών στοιχείων.

3.7.1. Έξυπνο συμβόλαιο

Το έξυπνο συμβόλαιο είναι απλώς μια φράση που χρησιμοποιείται για να περιγράψει μερικές γραμμές κώδικα που μπορεί να διευκολύνει τις συναλλαγές χρημάτων, περιεχομένου, ιδιοκτησίας, μετοχών ή οτιδήποτε αξίας[23]. Όταν εκτελείτε ένα έξυπνο συμβόλαιο στο blockchain λειτουργεί σαν ένα πρόγραμμα και εκτελείται αυτόματα όταν πληρούνται συγκεκριμένες προϋποθέσεις. Επειδή τα έξυπνα συμβόλαια εκτελούνται στο blockchain, λειτουργούν ακριβώς όπως έχουν προγραμματιστεί χωρίς καμία πιθανότητα λογοκρισίας, διακοπής λειτουργίας, απάτης ή παρέμβασης τρίτων. Ενώ όλες οι πλατφόρμες blockchain έχουν τη δυνατότητα επεξεργασίας κώδικα, οι περισσότερες είναι αρκετά περιορισμένες. το ethereum είναι διαφορετικό. Αντί να παρέχει ένα σύνολο περιορισμένης λειτουργικότητας, το ethereum επιτρέπει στους προγραμματιστές να δημιουργούν ό, τι εφαρμογή θέλουν. Αυτό σημαίνει ότι οι προγραμματιστές μπορούν να δημιουργήσουν χιλιάδες διαφορετικές εφαρμογές που ξεπερνούν οτιδήποτε έχει εμφανιστεί στο παρελθόν.

3.7.2. Ethereumvirtualmachine

Πριν από τη δημιουργία του ethereum, οι εφαρμογές είχαν σχεδιαστεί να λειτουργούν περιορισμένα. Το Bitcoin και άλλα κρυπτονομίσματα, για παράδειγμα, αναπτύχθηκαν αποκλειστικά για να λειτουργούν ως ψηφιακά νομίσματα peer-to-peer. Ο VitalikButerin αναγνωρίζοντας το πρόβλημα και θεωρώντας πως η επέκταση των λειτουργιών του bitcoin είναι μια διαδικασία περίπλοκη και χρονοβόρα αποφάσισε να αναπτύξει το Ethereumvirtualmachine(EVM)[23]. Το Ethereumvirtualmachine Επιτρέπει σε οποιονδήποτε προγραμματιστή να εκτελεί οποιοδήποτε πρόγραμμα, ανεξάρτητα από τη γλώσσα προγραμματισμού αρκεί να υπάρχει αρκετός χρόνος και μνήμη. Η EthereumVirtualMachine κάνει τη διαδικασία δημιουργίας εφαρμογών blockchain πολύ πιο εύκολη και αποτελεσματική. Αντί να είναι απαραίτητη η δημιουργία ενός πρωτότυπου blockchain για κάθε νέα εφαρμογή, το ethereum επιτρέπει την ανάπτυξη απεριόριστων εφαρμογών σε μια πλατφόρμα.

3.7.3 Χρήση του ethereum σε κατανεμημένες εφαρμογές(Dapps)

Το Ethereum επιτρέπει στους προγραμματιστές να δημιουργούν και να αναπτύσσουν αποκεντρωμένες εφαρμογές[23]. Μια αποκεντρωμένη εφαρμογή ή Dapp εξυπηρετεί συγκεκριμένο σκοπό για τους χρήστες της. Το Bitcoin, για παράδειγμα, είναι ένα Dapp που παρέχει στους χρήστες

του ένα σύστημα ηλεκτρονικών μετρητών peertopeer και επιτρέπει online πληρωμές Bitcoin. Επειδή οι αποκεντρωμένες εφαρμογές γράφονται σε μορφή κώδικα που εκτελείται σε ένα δίκτυο blockchain, δεν ελέγχονται από κανένα άτομο ή κεντρική οντότητα. Οποιαδήποτε υπηρεσία η οποία είναι συγκεντρωτική μπορεί να αποκεντρωθεί χρησιμοποιώντας το ethereum. Για παράδειγμα τα third-party που υπάρχουν σε εκατοντάδες διαφορετικές βιομηχανίες. Από υπηρεσίες όπως δάνεια που παρέχονται από τράπεζες σε υπηρεσίες απόδοσης τίτλων, συστήματα ψηφοφορίας και πολλά άλλα.

3.7.4. Αποκεντρωμένοι Αυτόνομοι Οργανισμοί

Το Ethereum χρησιμοποιείται για τη δημιουργία Αποκεντρωμένων Αυτόνομων Οργανισμών (DAO)[23][25]. Το DAO είναι ένας πλήρως αυτόνομος, αποκεντρωμένος οργανισμός χωρίς κανέναν ηγέτη. Τα DAO λειτουργούν με έξυπνα συμβόλαια γραμμένα στο ethereum. Τα συμβόλαια αυτά έχουν σχεδιαστεί για να αντικαταστήσουν τους κανόνες και τη δομή ενός παραδοσιακού οργανισμού, εξαλείφοντας την ανάγκη για ανθρώπους και για κεντρικό έλεγχο. Ένα DAO ανήκει σε όλους όσους αγοράζουν τα τεκμήρια του, αλλά αντί τα τεκμήρια να ισοδυναμούν με μετοχές και ιδιοκτησία, απλώς δίνουν στους κατόχους τους δικαιώματα ψήφου.

3.7.5. ERC-20

Το Ethereum χρησιμοποιείται επίσης ως πλατφόρμα για την δημιουργία και την έκδοση άλλων κρυπτονομισμάτων[23][26]. Λόγω του προτύπου τεκμηρίων ERC20(EthereumrequestforComments 20 tokenstandards) που ορίζεται από το EthereumFoundation, άλλοι προγραμματιστές μπορούν να εκδώσουν τις δικές τους εκδόσεις αυτού του τεκμηρίου σε μορφή κρυπτονομίσματος και να συγκεντρώσουν χρήματα με μια αρχική προσφορά νομισμάτων (ICO). Σε αυτήν τη στρατηγική συγκέντρωσης χρημάτων, οι εκδότες του κρυπτονομίσματος ορίζουν ένα ποσό που θέλουν να συγκεντρώσουν, το προσφέρουν σε μια πώληση πλήθους και λαμβάνουν Ether ως αντάλλαγμα. Δισεκατομμύρια δολάρια έχουν συγκεντρωθεί από τους ICO στην πλατφόρμα ethereum τα τελευταία χρόνια, και ένα από τα πιο πολύτιμα κρυπτονομίσματα στον κόσμο, το EOS, είναι κρυπτονόμισμα ERC20.

3.7.6. ERC-721

Στο Ethereum δημιουργήθηκε ένα πρότυπο που ονομάζεται τεκμήριο ERC721(EthereumrequestforComments 721 tokenstandards) για την παρακολούθηση μοναδικών ψηφιακών περιουσιακών στοιχείων[23][27]. Μία από τις μεγαλύτερες περιπτώσεις χρήσης των τεκμηρίων ERC-721 είναι τα ψηφιακά συλλεκτικά αντικείμενα, καθώς η υποδομή επιτρέπει στους ανθρώπους να αποδείξουν την ιδιοκτησία σπάνιων ψηφιακών αγαθών. Πολλά παιχνίδια κατασκευάζονται αυτήν τη στιγμή χρησιμοποιώντας αυτήν την τεχνολογία, όπως το CryptoKitties, ένα παιχνίδι με συλλεκτικές ψηφιακές γάτες που δημιουργήθηκε για να διευκολύνει τον κόσμο στην χρήση του ethereum.

Β.Μεθοδολογία

1. Τεχνολογίες και εργαλία

1.1. GethCLI

Το geth-CLI ή go-ethereumcommandlineinterface είναι η υλοποίηση ενός κόμβου Ethereum, αναπτυγμένο στη προγραμματιστική γλώσσα go. Το geth δίνει στο χρήστη τη δυνατότητα να συμμετάσχει στο ethereum δίκτυο ή να αναπτύξει ένα ιδιωτικό δίκτυο και να αλληλεπιδράσει με αυτό με διάφορους τρόπους. Δημιουργώντας έναν λογαριασμό μπορεί να εξορύξει, να κανει συναλλαγές με το κρυπτονόμισμαether, όπως και να παρατάξει ή να επικοινωνήσει με έξυπνα συμβόλαια στο δίκτυο.

1.2. Solidityprogramminglanguage

Η Solidity είναι μια αντικειμενοστρεφή γλώσσα προγραμματισμού υψηλού επιπέδου με σύνταξη παρόμοια της javascript. Χρησιμοποιείται για την δημιουργία έξυπνων συμβολαίων τα οποία ενισχύουν την εικονική μηχανή του Ethereum(EVM). Η συγκεκριμένη γλώσσα υποστηρίζει την κληρονομικότητα ιδιοτήτων των αντικειμένων και δίνει τη δυνατότητα σε αντικείμενα να αποκτήσουν ιδιότητες ήδη υπάρχων αντικειμένων. Η τάξη που χρησιμοποιείται ως βάση ονομάζεται υπερ-κλάση(superclass) και όλες οι τάξεις που κληρονομούν από αυτήν ονομάζονται υπό-κλάσεις(subclasses). Στη solidity δίνεται η δυνατότητα να δημιουργηθούν διάφορα είδη έξυπνων συμβολαίων όπως ψηφοφορίες, χρηματοδοτήσεις, πλειστηριασμοί με τα οποία επιτυγχάνεται η αποφυγή μεσαζόντων διότι όλα οι συναλλαγές επαληθεύονται από το blockchain δίκτυο, που είναι αποθηκευμένο το έξυπνο συμβόλαιο.

1.3. RemixIDE

Το RemixIDE είναι ένα εργαλείο ανοιχτού κώδικα που δίνει τη δυνατότητα στο χρήστη να αναπτύξει και να αλληλεπιδράσει με έξυπνα συμβόλαια μέσω ενός φυλλομετρητή. Υποστηρίζει επίσης δοκιμές, εντοπισμό σφαλμάτων και παράταξη σε έξυπνα συμβόλαια είτε στη τοπική εικονική μηχανή javascript που παρέχει ή σε ένα εξωτερικό blockchain δίκτυο.

1.4. Ganache-CLI

Το nodepackagemanagerGanacheCLI προσομοιάζει ένα κόμβο Ethereum τοπικά, και χρησιμοποιείται για τη δοκιμή και την ανάπτυξη έξυπνων συμβολαίων. Αρχικοποιώντας τον κόμβο δημιουργούνται αυτόματα δέκα λογαριασμοί με τα δημόσια και τα ιδιωτικά κλειδιά τους. Κάθε λογαριασμός κατέχει εκατό ether τα οποία μπορεί να χρησιμοποιήσει για να κάνει συναλλαγές στο τοπικό δίκτυο. Αξιοποιώντας το GanacheCLI αντί για οποιοδήποτε testNet αποφεύγονται τα μεγάλα διαστήματα αναμονής, το κόστος της ανάπτυξης η επιβράδυνση και επιβάρυνση του δικτύου και οι περιττές πληροφορίες που θα αποθηκεύονταν χωρίς να χρησιμοποιηθούν.

1.5. Metamask

Το metamask εξυπηρετεί ως ένας μεσολαβητής μεταξύ του χρήστη και ενός ethereum δικτύου. Για την διεπαφή του χρήστη με μία κατανεμημένη εφαρμογή απαιτείται η χρήση ενός λογαριασμού με το δημόσιο και ιδιωτικό κλειδί του. Το metamask αποθηκεύει τους λογαριασμούς του χρήστη καθώς και το ether κάθε λογαριασμού και του επιτρέπει να συνδεθεί σε οποιοδήποτε ethereum δίκτυο, χρησιμοποιώντας τους, με αποτέλεσμα να έχει πρόσβαση αλλά και την ικανότητα να αλληλεπιδράσει χρησιμοποιώντας το ether του με οποιοδήποτε dapp του εκάστοτε δικτύου . Με το metamask είναι δυνατή και η συναλλαγή ether καθώς και άλλων κρυπτονομισμάτων ανεπτυγμένων στη προγραμματικστική γλώσσα solidity, γνωστά και ως ERC-20. Διατίθεται στο κοινό σαν ένα plug-in στους φυλλομετρητέςGoogleChrome, Opera, Brave και Mozillafirefox. Αναμφισβήτητα κάνει το Ethereumπροσβάσιμο στον κόσμο όπου ήταν και ο στόχος των δημιουργών του.

1.6. Nodepackagemanager(NPM)

ToNodePackageManager ορίζεται ως ένα διαδικτυακό αποθετήριο ανοιχτού κώδικα για προγράμματα βασισμένα στο Node.js. Χρησιμοποιείται επίσης σαν πρόγραμμα διεπαφής με γραμμή εντολών για την εγκατάσταση προγραμματιστικών πακέτων καθώς και για την διαχείριση εκδόσεων και προγραμματιστικών εξαρτημάτων. Νέες βιβλιοθήκες και εφαρμογές του Node.jsαναρτούνται καθημερινά στο nodepackagemanager. Το package περιγράφεται από ένα json αρχείο σε αντίθεση με το node.js όπου χρειάζεται χειροκίνητη τοποθέτηση και αποτελείται από το αρχεία που αποθηκεύονται τα πακέτα και τα προγραμματιστικά εξαρτήματα, την ιστοσελίδα το npmjs καθώς και το πρόγραμμα διεπαφής της γραμμής εντολών.

1.7. Node.js

Το Node.js είναι σχεδιασμένο για τη δημιουργία διαδικτυακών εφαρμογών με δυνατότητα επέκτασης εφόσον θεωρείται ένα ασύγχρονο event-drivenruntimejavascript περιβάλλον. Αντιτίθεται στα κοινά μοντέλα συγχρονισμένης ταυτοποίησης όπου χρησιμοποιούνται νήματα λειτουργικού συστήματος, μια αναποτελσματική δικτύωση συστήματος και δύσκολη στη χρήση. Εφόσον δεν υφίστανται κλειδαριές, οι χρήστες απαλλάσσονται από ανησυχίες για το κλείσιμο της διαδικασίας. Ελάχιστες είναι οι λειτουργίες που εκτελούν άμεση παραγωγή από την είσοδο, με αποτέλεσμα το ανύπαρκτο μπλοκάρισμα της διαδικασίας. Το Node.js έχει χαρακτηριστικά παρόμοια με το Rubyeventmachine και το python'stwisted ωστόσο έχει ανεβάσει το επίπεδο στην ιδέα του μοντέλου event-driven, διότι εκτελεί ένα βρόγχο με events ως κατασκευή χρόνου εκτέλεσης σε αντίθεση με τα προαναφερθέντα που το παρουσιάζουν σε μορφή βιβλιοθήκης. Επίσης το node.js εκκινεί το βρόγχο απευθείας μετά την εκτελεστή του προγράμματος εισόδου και αφού δεν υπάρχουν αναφορές αποτελεσμάτων εξέρχεται από αυτό. Ενώ το node.js δεν είναι σχεδιασμένο ως σύστημα πολλαπλών νημάτων υπάρχει η δυνατότητα επέκτασης σε αυτό με απλή σύνδεση και εύκολη στην επικοινωνία.

1.8. Truffleframework

Το πλαίσιο truffle είναι ένα περιβάλλον για δοκιμές και ανάπτυξη συμβολαίων blockchain στην εικονική μηχανή ethereum (EVM). Με το truffle μειώνεται η δυσκολία στη ενσωμάτωση έξυπνων συμβολαίων για τη σύνδεση μεταξύ δικτύου και εφαρμογής, την ανάπτυξη και την διαχείριση τους. Υπάρχει η δυνατότητα εύκολης και αυτοματοποιημένης δοκιμής έξυπνων συμβολαίων καθώς και η ενσωμάτωση εξωτερικών πλαισίων(mocca) για ταχύτερη δοκιμή και ανάπτυξη. Η ικανότητα ανάπτυξης, εγκατάστασης και απεγκατάστασης έξυπνων συμβολαίων κάνει το truffle ιδιαίτερα χρήσιμο στην ανάπτυξη λογισμικού blockchain. Χρησιμοποιείται επίσης και για την διαχείριση δικτύων, δημόσιων ή ιδιωτικών αλλά και για την διαχείριση πακέτων με Ethereumpackagemanager

και nodepackagemanager, χρησιμοποιώντας το πρότυπο ERC190. Προσφέρει διαδραστική κονσόλα για την επικοινωνία με έξυπνα συμβόλαια και θεωρείται διαμορφώσιμος αγωγός κατασκευής ενσωμάτωσης συμβολαίων. Τέλος μέσα σε περιβάλλον truffle μπορεί να δημιουργηθεί δρομέας σεναρίων για δοκιμές κώδικα.

1.9. Javascript

Η γλώσσα προγραμματισμού javascript χρησιμοποιείται για την εκτέλεση προγραμμάτων από την μεριά του εξυπηρετητή αλλά και από την μεριά του πελάτη μέσω του φυλλομετρητή. Το βασικό πλεονέκτημα της javascript βρίσκεται στην δυνατότητα ανάπτυξης προγραμμάτων σε διάφορες τεχνολογίες όπως εφαρμογών desktop, κινητών, εξυπηρετητών, φυλλομετρητών, βάσεων δεδομένων και σε αρκετών άλλων είδών. Θεωρείται η πιο διαδεδομένη γλώσσα για την ανάπτυξη διαδικτυακών εφαρμογών με δυναμικό περιεχόμενο εφόσον υποστηρίζεται από όλους φυλλομετρητές.

1.10 Web3.js

Toweb3.js είναι ένα από τα σημαντικότερα πακέτα npm για την ανάπτυξη εφαρμογών blockchain. Είναι μια συλλογή από βιβλιοθήκες που δίνουν τη δυνατότητα αλληλεπίδρασης με κόμβους του ethereumblockchain είτε σε τοπικό δίκτυο είτε σε απομακρυσμένο δίκτυο χρησιμοποιώντας συνδέσεις HTTP ή IPC. Θεωρείται το πιο διαδεδομένο και εύκολο στη χρήση πακέτο για την αλληλεπίδραση χρήστη με έξυπνα συμβόλαια αποθηκευμένα στο ethereumblockchain καθώς και την χρήση των συναρτήσεων τους και τον δεδομένων τους.

1.11 jQuery

Η Βιβλιοθήκη jQuery είναι σχεδιασμένη για την απλοποίηση της χρήσης javascript στο fron-end. Είναι βιβλιοθήκη ανοιχτού κώδικα εύκολη στη χρήση, συμπαγής με διάφορα χαρακτηριστικά και θεωρείται η πιο γνωστή και χρησιμοποιημένη βιβλιοθήκη της javascript στο διαδίκτυο. Μερικά από τα βασικά χαρακτηριστικά της που την καθιστούν ιδανική για την δημιουργία και την ανάπτυξη πλούσιων ιστοσελίδων καθώς και διαδικτυακών εφαρμογών είναι:

- Ευκολία χειρισμού HTML/DOM
- Εύκολος χειρισμός CSS
- Αξιοποίηση συμβάντων (events)
- Δυνατότητα χρήσης εφέ και κινουμένων σχεδίων
- Απλοποίηση των AsynchronousJavascriptandXML γνωστά και ως AJAX
- Ύπαρξη συναρτήσεων για την τυποποίηση καθώς και την διευκόλυνση της υλοποίησης συχνών και συνηθισμένων διεργασιών

Η βιβλιοθήκη jQuery προσφέρει επιπλέον λειτουργικότητες μέσω plugins, τα επονομαζόμενα jQueryPlugins. Αυτά τα πακέτα βρίσκονται από την κοινότητα ελεύθερου λογισμικού ή α δημιουργηθούν και να μοιραστούν σε αυτήν. Με αυτό τον τρόπο καλύπτεται ένα τεράστιων φασμα λειτουργιών από αυτή τη βιβλιοθήκη.

1.12 Bootstrap

https://www.tutorialrepublic.com/twitter-bootstrap-tutorial/

Το Bootstrap είναι ένα front-endframework που χρησιμοποιείται για την ευκολότερη και ταχύτερη ανάπτυξη λογισμικού. Περιέχει κουμπιά, αναπτυσσόμενα μενού, ειδοποιήσεις και πολλά άλλα βασισμένα σε απλή HTML και CSS. Προσφέρει την δυνατότητα δημιουργίας ευέλικτων και προσαρμοστικών σελίδων χωρίς ιδιαίτερη προσπάθεια. Άλλες δυνατότητες του Bootstrap είναι:

1) Διαδραστικές ιστοσελίδες.
2) Πολλαπλές στήλες με προκαθορισμένες κλάσεις
3) Διαφορετικές και εύκολες στη δημιουργία διατάξεις φόρμας
4) Πολλές παραλλαγές γραμμών πλοήγησης
5) Εύκολη δημιουργία διατάξεων accordionκ.λ.π χωρίς την χρήση javascript αρχείων.
6) Καρτέλες για εύκολη διαχείριση του όγκου των περιεχομένων

Τα προτερήματα του bootstrap είναι ονομαστικά η εξοικονόμηση χρόνου, οι δυνατότητες διαδραστικότητας που δίνονται, εύκολο μοτίβο σχεδίασης, είναι σχεδιασμένο για τους νέους φυλλομετρητές και είναι λογισμικό ανοιχτού κώδικα.

2. Επεξήγηση κατανεμημένης εφαρμογής

Η εφαρμογή αφορά τη δημιουργία ενός έξυπνου συμβολαίου σε ένα τοπικό δίκτυο ethereum και την ανάπτυξη μίας ιστοσελίδας διεπαφής χρήστη στην οποία μέσω του metamask ένας χρήστης μπορεί να αλληλεπιδράσει με το έξυπνο συμβόλαιο. Στο έξυπνο συμβόλαιο αναπτύσεται μια μορφή κρυπτονομίσματος βασισμένη στα erc-20 standard καθώς και τεκμήρια τα οποία ακολουθούν τα standard των erc-721. Ο χρήστης ξεκινώντας στην εφαρμογη έχει τη δυνατότητα μέσω του λογαριασμού του στο metamask να δημιουργήσει τα πρώτα του 721 τεκμήρια. Ταυτόχρονα του δίνονται και μερικά 20 tokens. Έπειτα έχει τις παρακάτω δυνατότητες:

1. Ο χρήστης μπορεί να δει τα tokens που κατέχει αλλά και όσα βρίσκονται σε πώληση.
2. Ο χρήστης μπορεί να ζευγαρώσει δυο τεκμήρια 721 για να δημιουργήσει ένα νέο τεκμήριο με την προϋπόθεση ότι τα επιλεγμένα τεκμήρια μπορούν να ζευγαρώσουν.
3. Μέσα στην εφαρμογή υπάρχει ένα market στο οποίο μπορεί ένας χρήστης να πουλήσει και να αγοράσει 721 τεκμήρια χρησιμοποιώντας erc 20 tokens
4. Ο χρήστης έχει επίσης τη δυνατότητα να αγοράσει erc-20 tokens χρησιμοποιώντας ether.

Ταυτόχρονα δημιουργείται και ένα δεύτερο έξυπνο συμβόλαιο όπου αξιοποιεί το πρώτο και δίνει τη δυνατότητα στον δημιουργό του να δει όλα τα 721 token που έχουν δημιουργηθεί στο πρώτο έξυπνο συμβόλαιο. Η συγκεκριμένη εφαρμογή δημιουργείται με σκοπό να δείξει τους περιορισμούς και τον έλεγχο της ασφάλειας της εφαρμογής εφόσον στο ethereumblockchainnetwork τα έξυπνα συμβόλαια δεν είναι αποθηκευμένα σε κρυπτογραφιμένη μορφή.

3. Υλοποίηση εφαρμογής

3.1 Ανάπτυξη Έξυπνων Συμβολαίων

Κάθε έξυπνο συμβόλαιο πρέπει να αναπτύσσεται με συγκεκριμένες αρχές από την υλοποίηση εώς και τους περιορισμούς της γλώσσας στην οποία γράφεται. Το blockchain και η γλώσσα προγραμματισμού solidity δίνουν στον προγραμματιστή δυνατότητες όπου άλλες τεχνολογίες αδυνατούν να ικανοποιήσουν. Καταρχάς στην solidity κάθε μορφή λειτουργικότητας (function) πρέπει να πυροδοτηθεί από κάποιον λογαριασμό στο ethereumblockchain δίκτυο εκτός εάν χρησιμοποιείται από κάποια άλλη μέθοδο μέσα ή έξω από το έξυπνο συμβόλαιο αλλά και σε αυτή τη περίπτωση η μέθοδος αυτή έχει πυροδοτηθεί από κάποιον λογαριασμό. Ο λογαριασμός μπορεί να είναι ενός ανθρώπου ή ενός έξυπνου συμβολαίου Τα δύο βασικά ήδη μεθόδων (functions) χωρίζονται με βάση την δράση τους στο blockchain δίκτυο, αυτά που γράφουν στο δίκτυο και αυτά που διαβάζουν. Στη πρώτη κατηγορία τοποθετούνται όλες οι μέθοδοι που αλλάζουν ή προσθέτουν δεδομένα στο blockchain δίκτυο. Για να χρησιμοποιηθούν απαιτείται ο χρήστης να προσφέρει gus στο δίκτυο. Στη περίπτωση των μεθόδων που «διαβάζουν» από το blockchain δίκτυο χρησιμοποιούνται για την ανάκτηση δεδομένων από το δίκτυο χωρίς να πραγματοποιούν κάποια αλλαγή και δεν κοστίζουν gus.

Ένας από τους σημαντικότερους παράγοντες στην ανάπτυξη ενός έξυπνου συμβολαίου είναι η αρχιτεκτονική του. Η αρχιτεκτονική της εφαρμογής πρέπει να ακολουθεί ένα συγκεκριμένο μοτίβο ενώ ταυτόχρονα να καλύπτει όλες της πτυχές της εφαρμογής. Η βασική μορφή αρχιτεκτονικής στη γλωσσα προγραμματισμού solidity απαιτεί τη δημιουργία ρόλων που αντιστοιχίζονται σε λογαριασμούς. Κάθε λογαριασμός είναι αποθηκευμένος στο ethereumblockchain. Η χρήση των ρόλων είναι απαραίτητη για την δημιουργία περιορισμών στις μεθόδους του έξυπνου συμβολαίου εφόσον σε μία ολοκληρωμένη κατανεμημένη διαδικτυακή

εφαρμογή δεν πρέπει όλοι οι χρήστες να έχουν τα ίδια δικαιώματα. Ένα παράδειγμα είναι μια κατανεμημένη εφαρμογή για ψηφοφορίες. Στην περίπτωση αυτή θα πρέπει οι οργανωτές των ψηφοφοριών να είναι συγκεκριμένοι και να μην μπορεί να παρέμβει κάποιος εξωτερικός λογαριασμός στην δημιουργία μίας ψηφοφορίας. Η δημιουργία ρόλων επιτυγχάνετε είτε με δέσμευση λογαριασμών σε ρόλους είτε με μορφή διαδικότητας (αληθές, ψευδής). Οι ρόλοι μπορούν να αποδοθούν σε λογαριασμούς με τη χρήση μεθόδων τη στιγμή που το έξυπνο συμβόλαιο αποθηκεύεται στο δίκτυο ή ενώ το συμβόλαιο είναι ήδη αποθηκευμένο. Η αξιοποίηση των ρόλων είναι ένα ιδιαίτερα σημαντικό χαρακτηριστικό στις κατανεμημένες εφαρμογές διότι ενώ υπάρχουν πολλοί τρόποι για τον περιορισμό των δυνατοτήτων των χρηστών, στη συγκεκριμένη περίπτωση οι ρόλοι δίνονται μέσα στο blockchain δίκτυο το οποίο θεωρείται απροσπέλαστο και το έξυπνο συμβόλαιο δε μπορεί να αλλάξει από την στιγμή που αποθηκεύεται στο δίκτυο.

Η μέθοδος με την οποία δημιουργούνται τα δεδομένα στο ethereumblockchain δίκτυο είναι σε μορφή δομών. Κάθε δομή έχει συγκεκριμένα χαρακτηριστικά τα οποία ρυθμίζει ο δημιουργός του έξυπνου συμβολαίου. Η αποθήκευση των δεδομένων επιτυγχάνεται με την χρήση πινάκων. Στο συγκεκριμένο σημείο πρέπει να ειπωθεί ότι ο τρόπος με τον οποίο δημιουργείται μια νέα δομή και αποθηκεύεται σε έναν πίνακα είναι μέσω της χρήσης μεθόδων.

Το blockchain προσομοιώνει την έννοια της ιδιοκτησίας δηλαδή την δυνατότητα της κατοχής ενός στοιχείου από έναν λογαριασμό. Αυτό επιτυγχάνεται με δύο συνθήκες. Έστω ότι υπάρχει το αντικείμενο Α και ο λογαριασμός Β. Για να θεωρηθεί το αντικείμενο Α ιδιοκτησία του λογαριασμού Β πρέπει να υπάρχει κάποια απόδειξη για την τεκμηρίωση της ιδιοκτησίας ενώ ταυτόχρονα το αντικείμενο Α δεν πρέπει να ανήκει σε άλλο λογαριασμό. Η δεύτερη συνθήκη είναι ότι εφόσον το αντικείμενο Α ανήκει στο λογαριασμό Β ο χρήστης με τον λογαριασμό Β έχει όλα τα προνόμια που του προσδίδει το αντικείμενο Α. Ο τρόπος με τον οποίο επιτυγχάνεται η προσομοίωση της ιδιοκτησίας με τη χρήση της solidity είναι η δέσμευση των δομών που αναφέρθηκαν παραπάνω σε λογαριασμούς με την χρήση των αντιστοιχίσεων.

Σε μία κατανεμημένη διαδυκτιακή εφαρμογή, στην οποία δύο ή περισσότεροι λογαριασμοί συμμετέχουν σε μία ίδια διαδικασία, πιθανότατα με τη χρήση μίας ή περισσοτέρων μεθόδων, δε χρειάζεται να υπάρχει εμπιστοσύνη μεταξύ των χρηστών αυτών των λογαριασμών και αυτό επιτυγχάνεται μέσω της γλώσσας προγραμματισμού solidity. Παράδειγμα: Έστω ότι δημιουργείται μία κατανεμημένη εφαρμογή χρηματοδότησης ατόμων από πλήθος (crowdfounding) και η λογική της εφαρμογής είναι η εξής:

1. Χρήστης Α δημιουργεί έναν λογαριασμό και επιλέγει το ποσό το 100 ether
2. Άλλοι χρήστες χρηματοδοτούν τον χρήστη Α δίνοντας ether
3. Εάν το συνολικό ether που δώθηκε από τους χρήστες μέσα σε ένα συγκεκριμένο χρονικό διάστημα είναι μικρότερο των 100 ether τότε οι χρήστες παίρνουν πίσω το ether τους.

Στο παραπάνω παράδειγμα δε χρειάζεται να υπάρχει εμπιστοσύνη μεταξύ του χρήστη Α και των υπολοίπων χρηστών διότι το έξυπνο συμβόλαιο είναι υπεύθυνο για την τήρηση της συμφωνίας μέσω των συνθηκών που υπάρχουνε στις μεθόδους του.

Λαμβάνοντας υπ’ όψιν ότι το blockchain είναι μια «βαριά και αργή» τεχνολογία και ότι όσο μεγαλύτερη η πολυπλοκότητα του έξυπνου συμβολαίου τόσο περισσότερο gus θα χρειάζεται για την χρήση των μεθόδων του πρέπει κάποιος να αναλογιστεί που να μην χρησιμοποιήσει την solidity. Ένας άγραφος γενικός κανόνας είναι οτι δεν χρειάζεται να προστίθεται επιπλέον

λειτουργικότητα στο έξυπνο συμβόλαιο πέρα από αυτή που αφορά την ασφάλεια,την εμπιστοσύνη, τους περιορισμούς, την ιδιοκτησία και τη μοναδικότητα.

Για τη δημιουργία της κατανεμημένης εφαρμογής χρησιμοποιήθηκαν κάποια έτοιμα έξυπνα συμβόλαια ως βάση. Το πρώτο από αυτά είναι το έξυπνο συμβόλαιο ownable.sol. Αυτό το έξυπνο συμβόλαιο χρησιμοποιείται από την κοινότητα του ethereum για τον εύκολο προσδιορισμό του διαχειριστή της εφαρμογής ενώ ταυτόχρονα δίνει το δυνατότητα στον διαχειριστή να δώσει τα δικαιώματα του σε έναν άλλον χρήστη.

```
pragma solidity >=0.4.0<0.6.0;

/**
* @title Ownable
* @dev The Ownable contract has an owner address, and provides basic authorization control
* functions, this simplifies the implementation of "user permissions".
*/
contract Ownable {
  address private _owner;

  event OwnershipTransferred(
    address indexed previousOwner,
    address indexed newOwner
  );

/**
  * @dev The Ownable constructor sets the original `owner` of the contract to the sender
  * account.
  */
constructor() internal {
    _owner = msg.sender;
    emit OwnershipTransferred(address(0), _owner);
  }

/**
  * @return the address of the owner.
  */
functionowner() public view returns(address) {
return _owner;
  }

/**
  * @dev Throws if called by any account other than the owner.
  */
  modifier onlyOwner() {
    require(isOwner());
    _;
  }

/**
  * @return true if `msg.sender` is the owner of the contract.
```

```
 */
functionisOwner() public view returns(bool) {
returnmsg.sender == _owner;
 }

/**
 * @dev Allows the current owner to relinquish control of the contract.
 * @notice Renouncing to ownership will leave the contract without an owner.
 * It will not be possible to call the functions with the `onlyOwner`
 * modifier anymore.
 */
functionrenounceOwnership() publiconlyOwner {
   emit OwnershipTransferred(_owner, address(0));
   _owner = address(0);
 }

/**
 * @dev Allows the current owner to transfer control of the contract to a newOwner.
 * @param newOwnerThe address to transfer ownership to.
 */
functiontransferOwnership(address newOwner) publiconlyOwner {
   _transferOwnership(newOwner);
 }

/**
 * @dev Transfers control of the contract to a newOwner.
 * @param newOwnerThe address to transfer ownership to.
 */
function _transferOwnership(address newOwner) internal {
require(newOwner != address(0));
   emit OwnershipTransferred(_owner, newOwner);
   _owner = newOwner;
 }
}
```

Ένα άλλο έξυπνο συμβόλαιο που χρησιμοποιείται σε μορφή βιβλιοθήκης είναι το safemath.sol. Η χρήση βιβλιοθηκών στην ανάπτυξη έξυπνων συμβολαίων είναι ιδιαιτέρως σημαντική διότι στο ethereum υπάρχει ένας περιορισμός στο μέγεθος των έξυπνων συμβολαίων που αποθηκεύονται σε αυτό. Η βιβλιοθήκη safemath όπως φαίνεται και από το όνομα της χρησιμοποιείται για τις πράξεις που γίνονται στις μεθόδους του έξυπνου συμβολαίου. Παραδείγματος χάρη έστω ότι υπάρχει ένα έξυπνο συμβόλαιο για κρυπτονόμισματα, εάν αφαιρεθεί μία ποσότητα από έναν λογαριασμό πρέπει το υπόλοιπο του λογαριασμού να είναι μη-αρνητικό. Για αυτόν τον λόγο χρησιμοποιείται η βιβλιοθήκη safemath.

```
pragma solidity >=0.4.0<0.6.0;

/**
```

```
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {

/**
  * @dev Multiplies two numbers, throws on overflow.
  */
functionmul(uint256 a, uint256 b) internal pure returns (uint256) {
if (a == 0) {
return0;
    }
    uint256 c = a * b;
assert(c / a == b);
returnc;
  }

/**
  * @dev Integer division of two numbers, truncating the quotient.
  */
functiondiv(uint256 a, uint256 b) internal pure returns (uint256) {
// assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
// assert(a == b * c + a % b); // There is no case in which this doesn't hold
returnc;
  }

/**
  * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend).
  */
functionsub(uint256 a, uint256 b) internal pure returns (uint256) {
assert(b <= a);
return a - b;
  }

/**
  * @dev Adds two numbers, throws on overflow.
  */
functionadd(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
assert(c >= a);
returnc;
  }
}

/**
 * @title SafeMath32
 * @dev SafeMath library implemented for uint32
 */
library SafeMath32 {

functionmul(uint32 a, uint32 b) internal pure returns (uint32) {
if (a == 0) {
return0;
    }
    uint32 c = a * b;
assert(c / a == b);
```

```
return c;
  }

function div(uint32 a, uint32 b) internal pure returns (uint32) {
// assert(b > 0); // Solidity automatically throws when dividing by 0
    uint32 c = a / b;
// assert(a == b * c + a % b); // There is no case in which this doesn't hold
return c;
  }

function sub(uint32 a, uint32 b) internal pure returns (uint32) {
assert(b <= a);
return a - b;
  }

function add(uint32 a, uint32 b) internal pure returns (uint32) {
    uint32 c = a + b;
assert(c >= a);
return c;
  }
}

/**
 * @title SafeMath16
 * @dev SafeMath library implemented for uint16
 */
library SafeMath16 {

function mul(uint16 a, uint16 b) internal pure returns (uint16) {
if (a == 0) {
return 0;
    }
    uint16 c = a * b;
assert(c / a == b);
return c;
  }

function div(uint16 a, uint16 b) internal pure returns (uint16) {
// assert(b > 0); // Solidity automatically throws when dividing by 0
    uint16 c = a / b;
// assert(a == b * c + a % b); // There is no case in which this doesn't hold
return c;
  }

function sub(uint16 a, uint16 b) internal pure returns (uint16) {
assert(b <= a);
return a - b;
  }

function add(uint16 a, uint16 b) internal pure returns (uint16) {
    uint16 c = a + b;
assert(c >= a);
return c;
  }
}
```

Τέλος χρησιμοποιούνται δυο έξυπνα συμβόλαια με την μορφή interface ως βάση για τα κρυπτονομίσματαerc20 καθώς και τα τεκμήρια erc721.

```
pragma solidity >=0.4.0<0.6.0;

contract ERC20 {
function totalSupply20() public view returns (uint);
function balanceOf20(address tokenOwner) public view returns (uint balance);
function allowance20(address tokenOwner, address spender) public view returns (uint remaining);
function transfer20(address to, uint tokens) internal returns (bool success);
function approve20(address spender, uint tokens) internal returns (bool success);
function transferFrom20(address from, address to, uint tokens) internal returns (bool success);

   event Transfer20(address indexed from, address indexed to, uint tokens);
   event Approval20(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

```
pragma solidity >=0.4.0<0.6.0;

contract ERC721 {
  event Transfer721(address indexed _from, address indexed _to, uint256 indexed _tokenId);
  event Approval721(address indexed _owner, address indexed _approved, uint256 indexed _tokenId);

function balanceOf721(address _owner) public view returns (uint256);
function ownerOf721(uint256 _tokenId) public view returns (address);
function transferFrom721(address _from, address _to, uint256 _tokenId) internal;
function approve721(address _approved, uint256 _tokenId) internal;
}
```

Όλος ο πηγαίος κώδικας solidity θα πρέπει να ξεκινά με ένα "versionpragma", μια δήλωση για την έκδοση του κώδικα. Αυτό γίνεται για να αποφευχθούν προβλήματα με μελλοντικές εκδόσεις μεταγλωττιστών που ενδέχεται να εισάγουν αλλαγές που θα μπορούσαν να καταστρέψουν τον κώδικά.

```
pragma solidity >=0.4.0<0.6.0;
```

Το έξυπνο συμβόλαιο ξεκινά με την δήλωση της φράσης contract και το όνομα του έξυπνου συμβολαίου. Ανάμεσα στις αγκύλες τοποθετείται η λειτουργικότητα του κώδικα.

```
Contract HelloWorld {

}
```

Όταν υπάρχουν πολλαπλά αρχεία, για την εισαγωγή ενός αρχείου σε άλλο χρησιμοποιείται η λέξη import και ακολουθεί το όνομα του αρχείου που εισάγεται.

```
import "./erc20.sol";
import "./safemath.sol";
import "./ownable.sol";
```

Στη δήλωση του έξυπνου συμβολαίου μπορεί να χρησιμοποιηθεί η λέξη “is” και να ακολουθήσει το όνομα του έξυπνου συμβολαίου που βρίσκεται στο αρχείο που έγινε import έτσι ώστε να κληρονομηθούν όλες οι ιδιότητες του προηγούμενου έξυπνου συμβολαίου στο καινούριο. Πρώτα αναπτύχθηκε ένα έξυπνο συμβόλαιο για την δημιουργία κρυπτονομίσματος με τα erc-20 standard. Το συγκεκριμένο συμβόλαιο προσομοιάζει έναν λογαριασμό τραπέζης όπου αποθηκεύονται τα κρυπτονομισματα του χρήστη.

```
contract MyERC20 is ERC20, Ownable {

}
```

Για την αξιοποίηση των βιβλιοθηκών χρησιμοποιείται η λέξη “using ” και ακολουθεί το πακέτο της βιβλιοθήκης. Στο παρακάτω παράδειγμα χρησιμοποιείται η βιβλιοθληκησafemath για τους αριθμούς με 16,32 και 256 bytes.

```
using SafeMath16 for uint16;
  using SafeMath32 for uint32;
usingSafeMath for uint256;
```

Στην αρχή του έξυπνου συμβολαίου τοποθετείται το όνομα του νομίσματος και τα δεκαδικά του ψηφία. Το όνομα του είναι “gold” και έχει 1 δεκαδικό ψηφίο. Έπειτα ορίζεται ο συνολικός αριθμός κρυπτονομισμάτων του έξυπνου συμβολαίου, ένα εκατομμύριο. Στη συνέχεια δημιουργούνται μια αντιστοίχηση (mapping) που προσομοιάζει το υπόλοιπο του κάθε λογαριασμού καθώς και μία ακόμα αντιστοίχιση που προσομοιάζει το ποσό που επιτρέπεται να χρησιμοποιήσει ένας λογαριασμός από έναν άλλον. Ένα παράδειγμα είναι ο χρήστης Α επιτρέπει στο χρήστη Β να ξοδέψει 10 κρυπτονομίσματα από το υπόλοιπο του Α. Τέλος ορίζεται μια μέθοδος ως “constructor” που αρχικοποιεί το υπόλοιπο του λογαριασμου του έξυπνου συμβολαίου ίσο με τον συνολικό αριθμό των κρυπτονομισμάτων ώστε όλα τα κρυπτονομίσματα να ανήκουν στο έξυπνο συμβόλαιο. Αυτό είναι μία βασική αρχή στην ανάπτυξη των έξυπνων συμβολαίων διότι εάν τα κρυπτονομίσματα ανήκαν σε ένα λογαριασμό ενός χρήστη όπως τον δημιουργό του συμβολαίου τότε θα μπορούσε να υπάρξει κακόβουλη χρήση των κρυπτονομισμάτων.

```
string public constant name = "Gold";
string public constant symbol = "GFT";
uint8 public constant decimals = 1;

  mapping (address => uint256)  _balances;
  mapping (address => mapping (address => uint256))  _allowed;

  uint256 private _totalSupply = 1000000;

constructor() public{
```

```
  _balances[address(this)] = _totalSupply;

 }
```

Παρατηρείται παραπάνω ότι δίπλα από το "constructor" υπάρχει η λέξη public. Στην γλώσσα προγραμματισμού solidity οι μεθόδοι και οι μεταβλητές είναι δημόσιες από προεπιλογή. Αυτό σημαίνει ότι οποιοσδήποτε (ή οποιοδήποτε άλλο συμβόλαιο) μπορεί να καλέσει τη λειτουργία του συμβολαίου και να εκτελέσει τον κωδικό του. Προφανώς αυτό δεν είναι πάντα επιθυμητό γιατί μπορεί να κάνει το συμβόλαιό ευάλωτο σε επιθέσεις. Επομένως, είναι καλή πρακτική να επισημαίνονται οι μέθοδοι ως ιδιωτικές (private) από προεπιλογή και, στη συνέχεια, να δημοσιεύονται μόνο οι λειτουργίες που είναι απαραίτητο να εκτεθούν. Εκτός από τις private και τις public μεθόδους, στη Solidity υπάρχουν δύο ακόμη τύποι ορατότητας για τις μεθόδους: internal και external. Οι internal μέθοδοι είναι όμοιες με τις private, εκτός από το ότι είναι επίσης προσβάσιμες σε συμβόλαια που κληρονομούν από αυτό το συμβόλαιο. Οι external μέθοδοι είναι όμοιες με τις public, εκτός από το ότι αυτές οι λειτουργίες μπορούν να κληθούν μόνο εκτός της σύμβασης, δεν μπορούν να κληθούν από άλλες λειτουργίες εντός αυτής της σύμβασης.

Όταν μια μεταβλητή είναι public τότε αυτομάτως δημιουργείται και μια μέθοδος που την επιστρέφει. Εφόσον αρκετές από τις μεταβλητές του κώδικα είναι ιδιωτικές πρέπει να δημιουργηθούν και οι κατάλληλες λειτουργίες ώστε να μπορούν να επιστραφούν και να εμφανιστούν στην εφαρμογή. Στο παρακάτω κώδικα δημιουργούνται μέθοδοι που επιστρέφουν το συνολικό αριθμό των κρυπτονομισμάτων, το υπόλοιπο ενός λογαριασμού καθώς και τον αριθμό των κρυπτονομισμαων που επιτρέπετε να ξοδέψει ένας λογαριασμός από έναν άλλον.

```
function totalSupply20() public view returns (uint256) {
return _totalSupply;
  }

function balanceOf20(address owner) public view returns (uint256) {
return _balances[owner];
  }

function allowance20(address owner, address spender) public view returns(uint256) {
return _allowed[owner][spender];
  }
```

Παρατηρείται ότι η μέθοδος allowance20 χρειάζεται δύο λογαριασμούς για να επιστρέψει ένα αποτέλεσμα, αυτού που ξοδεύει και του λογαριασμού που ανήκουν τα κρυπτονομίσματα. Φαίνεται επίσης ότι και οι τρείς μέθοδοι έχουν έναν modifierview. Με αυτόν το modifier το έξυπνο συμβόλαιο δηλώνει ότι αυτές οι μέθοδοι δεν γράφουν στο δίκτυο αλλά μόνο ανακτούν δεδομένα.

Έπειτα δημιουργούνται οι μέθοδοι που είναι υπεύθυνες για την μεταφορά των κρυπτονομισμάτων μεταξύ λογαριασμών καθώς και οι μέθοδος που δίνει τη δυνατότητα σε έναν χρήστη να δώσει την άδεια σε έναν άλλον χρήστη να ξοδέψει κρυπτονομίσματα από το υπόλοιπο του.

```
function transfer20(address to, uint256 value) internal returns (bool) {
require(value <= _balances[msg.sender]);
require(to != address(0));

  _balances[msg.sender] = _balances[msg.sender].sub(value);
```

```
    _balances[to] = _balances[to].add(value);
    emit Transfer20(msg.sender, to, value);
returntrue;
  }

function transferFrom20(address from, address to, uint256 value) internal returns (bool) {
require(value <= _balances[from]);
require(value <= _allowed[from][msg.sender]);
require(to != address(0));

    _balances[from] = _balances[from].sub(value);
    _balances[to] = _balances[to].add(value);
    _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
    emit Transfer20(from, to, value);
returntrue;
  }

function approve20(address spender, uint256 value) internal returns (bool) {
require(spender != address(0));

    _allowed[msg.sender][spender] = value;
    emit Approval20(msg.sender, spender, value);
returntrue;
  }

//a function that allows the smart contract to send gold to a player
function transferFromContract20(address to, uint256 value) internal{

require(to != address(0));

    _balances[address(this)] = _balances[address(this)].sub(value);

    _balances[to] = _balances[to].add(value);
    emit Transfer20(address(this), to, value);
  }

}
```

Στη μέθοδο transfer20 παρατηρείται η λέξη msg.sender όπου δηλώνει τον λογαριασμό που καλεί την μέθοδο, ένα πολύ σημαντικό στοιχείο στην solidity που προσδίδει στο συμβόλαιο ασφάλεια. Παραδείγματος χάρη στο προαναφερθείσα μέθοδο απαιτείται ο επικαλεστής της μεθόδου να είναι αυτός που θα μεταφέρει τα κρυπτονομίσματα του σε κάποιον άλλον λογαριασμό αλλιώς θα μπορούσε να χρησιμοποιηθεί κακόβουλα και να επιτρέπεται η μεταφορά κρυπτονομισματων από το υπόλοιπο οποιουδήποτε λογαριασμού χωρίς την έγκριση του. Μία ακόμα λέξη κλειδί σε αυτή τη μέθοδο είναι το “require” που εκτελεί έναν έλεγχο για το αν η συνθήκη που βρίσκεται μέσα στις παρενθέσεις είναι αληθής και αν δεν είναι βγάζει σφάλμα και δεν εκτελείται η μέθοδος. Το require όπως και το msg.sender προσδίδει ένα ακόμα επιπεδο ασφάλειας στο έξυπνο συμβόλαιο. Στην παραπάνω μέθοδο το require ελέγχει αν το ποσό που θέλει να καταθέσει ο msg.sender είναι μεγαλύτερο του υπολοίπου του και αν είναι συνεχίζει τη συναλλαγή. Στις υπόλοιπες μεθόδους ακολουθεί η ίδια λογική ασφάλειας όπως και στη μέθοδο transfer20.

Στο επόμενο έξυπνο συμβόλαιο, το οποίο έχει όλες τις ιδιότητες του προηγούμενου, αναπτύσσονται η μέθοδος της αγοράς του κρυπτονομίσματος με αντάλαγμαether, το οποίο αποθηκεύεται στο έξυπνο συμβόλαιο, παρακάτω ορίζεται η αρχική τιμή ως 0.001 ether για 5 gold αλλά υπάρχει και η λειτουργία setfee που αλλάζει τον αριθμό των ether συνήθως ανάλογα με την αξία του. Τέλος υπάρχει μία μέθοδος η οποία μπορεί να χρησιμοποιηθεί μόνο από την δημιουργό του έξυπνο συμβολαίου και δίνει τη δυνατότητα να συλλέξει το ether που βρίσκεται αποθηκευμένο στο έξυπνο συμβόλαιο από τις αγορές του gold.

```
pragma solidity >=0.4.0<0.6.0;


import"./1_myerc20.sol";

contract MyERC20Use is  MyERC20 {

uintpayFee = 0.001ether;


functionwithdraw() external  onlyOwner {
//address payable _owner = address(uint160(owner()));
   address(uint160(owner())).transfer(address(this).balance);
  }

functionsetFee(uint _fee) external onlyOwner {
payFee = _fee;
  }

functionbuyGold() external payable {
require(msg.value == payFee);
   require(balanceOf20(msg.sender) <= 50);
   transferFromContract20(msg.sender, 5);
  }

}
```

Ένας τροποποιητής(modifier) μοιάζει ακριβώς με μια συνάρτηση, αλλά δεν μπορεί να κληθεί άμεσα όπως μια συνάρτηση. Αντίθετα μπορεί να ενσωματωθεί στο τέλος ενός ορισμού συνάρτησης για να αλλάξει τη συμπεριφορά της. Στον παραπάνω κώδικα χρησιμοποιείται ο modifieronlyOwner στη συνάρτηση withdraw από το προηγούμενο έξυπνο συμβόλαιο ownable που επιτρέπει μόνο σε αυτόν που είναι ιδιοκτήτης του συμβολαίου να χρησιμοποιήσει την παραπάνω συνάρτηση. Η λέξη this αναφέρεται στο λογαριασμό του έξυπνου συμβολαίου που θα αποκτήσει όταν αποθηκευτεί στο ethereumblockchain δίκτυο. Τέλος στη μέθοδο buyGold παρατηρούνται οι λέξεις κλειδιά “payable” και “msg.value”. Το “payable” είναι ένα modifier που χρησιμοποιείται σε μεθόδους που κάνουν συναλλαγές με ether. Με αυτόν τον τρόπο το έξυπνο συμβόλαιο γνωρίζει ότι πρέπει να δεχτεί ένα ποσό ether. Το msg.value δηλώνει το ποσό που πρέπει να δωθεί από το χρήστη ώστε η μέθοδος να κάνει την μεταφορά.

Το παρακάτω συμβόλαιο είναι υπεύθυνο για την δημιουργία των τεκμηρίων. Για αρχή δημιουργείται η δομή των τεκμηρίων όπου ονομάζεται card και τα βασικά χαρακτηριστικά της είναι το σχημα, το χρώμα, πόσες φορές έχει να «ζευγαρώσει», το όριο του ζευγαρώματος, και ο χρόνος που χρειάζεται μετά από κάθε ζευγάρωμα για να μπορεί να ζευγαρώσει ξανά. Στη συνέχεια δημιουργείται ένας άδειος πίνακας για την αποθήκευση των τεκμηρίων και δύο mappings που αντιστοιχίζουν λογαριασμούς με τα τεκμήρια που κατέχουν αλλά και το συνολικό αριθμό των τεκμηρίων τους.

Έπειτα αναπτύσσονται οι μέθοδοι που δημιουργούν και αποθηκεύουν τα τεκμήρια. Κάθε τεκμήριο έχει μια μορφή dna που καθορίζει τα χαρακτηριστικά του. Παρακάτω φαίνεται η μέθοδος _generateRandomDna όπου δέχεται έναν αριθμό και κρυπτογραφεί σε μορφηkeccak256 μαζί με την ώρα, τον λογαριασμό του χρήστη και τον αριθμό και επιστρέφει έναν αριθμό με 16 ψηφία που χρησιμοποιείται ως dna. Η μέθοδος _generateRandomDna χρησιμοποιείται στην createCard όπου δημιουργεί το dna δημιουργεί και την δομή του τεκμηρίου και το τοποθετεί στον πινακαCards και την αντιστοιχίζει με το λογαριασμό του χρήστη. Τέλος δημιουργούνται δύο μέθοδοι για την ανάκτηση των καρτών.

```
pragma solidity >=0.4.0<0.6.0;

import"./2_myerc20use.sol";



// This is the contract that allows the user to create cards
contract CardFactory is MyERC20Use {

//event that emits that an address has created a new card
    event NewCard(address cardSender, uintcardId);


uintcooldownTime = 1minutes;
// we want 8 digits on our dna
uintdnaDigits = 16;
// this is the uint that allows us to get digits equal to the dnaDigits
uintdnaModulus = 10 ** dnaDigits;

// the card's stats dna, type, name, lvl, atk, def, skillid, nature, uses(times that the weapon has been used), limit(times that the weapon can been used)
    struct Card {
uintcardDna;
      uint16 cardShape;
      uint16 cardColor;
      uint16 cardUses;
      uint16 cardLimit;
      uint32 readyTime;
    }

// a list of cards
Card[] cards;

// a mapping that addresses a card to a user
mapping(uint => address) publiccardToOwner;
// a mapping that states how many cards a user has
mapping(address =>uint) publicownerCardCount;


// a function that creates and returns a dna with digits equal to dna digits
function _generateRandomDna(uintfavNumber) internal view returns(uint){
// a number that has been created by hashing current time, name, the user's address, and the nonce
uintrandDna = uint(keccak256(abi.encodePacked(now, msg.sender, favNumber)));
// returns thadna
returnrandDna % dnaModulus;
    }

function _createCard(uint256 _cardDna) internal {
```

```
uint id = cards.push(Card(_cardDna, uint16((_cardDna / 1000) % 10), uint16(_cardDna % 1000), 0, uint16(_cardDna % 10 + 1), uint32(now + cooldownTime))) - 1;
cardToOwner[id] = msg.sender;
ownerCardCount[msg.sender] = ownerCardCount[msg.sender].add(1);
    emit NewCard(msg.sender, id);
  }

functioncreateCard(uintfavNumber, uinthateNumber) public {
    require(ownerCardCount[msg.sender] == 0);

    transferFromContract20(msg.sender, 5);
uintfavRandDna = _generateRandomDna(favNumber);
uinthateRandDna = _generateRandomDna(hateNumber);
    _createCard(favRandDna);
    _createCard(hateRandDna);
  }

functiongetCard(uint _cardId) public view returns(uint,uint16,uint16, uint16, uint16, uint){
return(cards[_cardId].cardDna, cards[_cardId].cardShape, cards[_cardId].cardColor, cards[_cardId].cardUses, cards[_cardId].cardLimit, _cardId);
    }

functiongetTotalCards() public view returns(uint){
return (cards.length);
    }

}
```

Στο συγκεκριμένο έξυπνο συμβόλαιο δημιουργείται η λειτουργικότητα των τεκμηρίων που έχει να κάνει με το «ζευγάρωμα». Καταρχάς δημιουργείται ένα modifieronlyOwnerOf που δηλώνει ότι μόνο αυός που κατέχει το τεκμήριο μπορεί να το ζευγαρώσει ή να το πουλισει. Στη μέθοδο createNewCard τοποθετούνται οι αριθμοί του πίνακα των δύο τεκμηρίων που ζευγαρώνουν εφόσον μπορούν και αυτό ισχύει λόγο περιορισμών που έχουν τοποθετηθεί στη μέθοδο. Αφού δημιουργηθούν τα νέα τεκμήρια ενεργοποιείται η μέθοδος triggercooldown ώστε να υπάρχει κάποιο χρονικό περιθώριο μέχρι το τεκμήριο να μπορεί να ζευγαρώσει ξανά. Ο έλεγχος αυτού επιτυγχάνεται με το modifierisReady. Όταν το carduses ενός τεκμηρίου φτάσει το cardlimit δεν μπορεί να ζευγαρώσει ξανά. Ταυτόχρονα σε αυτό το έξυπνο συμβόλαιο δημιουργήθηκε και μια μέθοδος που δίνει τη δυνατότητα στον ιδιοκτήτη του έξυπνου συμβολαίου να δημιουργήσει δικά του ξεχωριστά τεκμήρια. Τέλος υπάρχει η μέθοδος getCardsByAddress που επιστρέφει όλα τεκμήρια του λογαριασμού που την επικαλέστηκε.

```
Pragmasolidity>=0.4.0<0.6.0;

import"./3_cardfactory.sol";
// getcards

contractCardBirthisCardFactory {



modifieronlyOwnerOfCard(uint _cardId) {
require(msg.sender == cardToOwner[_cardId]);
  _;
```

```
}

// triggers a cooldown for the character in order to attack again
function _triggerCooldown(Card storage _card) internal {
    _card.readyTime = uint32(now + cooldownTime);
  }
// checks if the character is ready to attack
function _isReady(Card storage _card) internal view returns (bool) {
return (_card.readyTime<= now);
  }

functioncreateCardAdmin(uint _cardDna, uint16 _cardShape, uint16 _cardColor, uint16 _cardLimit)
publiconlyOwner{
uint id = cards.push(Card(_cardDna, _cardShape, _cardColor, 0, _cardLimit, uint32(now +
cooldownTime))) - 1;
cardToOwner[id] = msg.sender;
ownerCardCount[msg.sender] = ownerCardCount[msg.sender].add(1);
    emit NewCard(msg.sender, id);
  }

functioncreateNewCard(uint _firstCardId, uint _secondCardId) publiconlyOwnerOfCard(_firstCardId)
onlyOwnerOfCard(_secondCardId){
require(_firstCardId != _secondCardId);
    Card storage firstCard = cards[_firstCardId];
    require(_isReady(firstCard));
require(firstCard.cardUses<firstCard.cardLimit);
    Card storage secondCard = cards[_secondCardId];
    require(_isReady(secondCard));
require(secondCard.cardUses<secondCard.cardLimit);
uintnewDna = (firstCard.cardDna + secondCard.cardDna) / 2;
    _createCard(newDna);
firstCard.cardUses = firstCard.cardUses.add(1);
secondCard.cardUses = secondCard.cardUses.add(1);
    _triggerCooldown(firstCard);
    _triggerCooldown(secondCard);

  }

functiongetCardByAddress(address _owner) public view returns(uint[] memory){
uint[] memory result = newuint[](ownerCardCount[_owner]);
uint counter = 0;
for (uinti = 0; i<cards.length; i++) {
if (cardToOwner[i] == _owner) {
        result[counter] = i;
        counter++;
      }
    }
returnresult;
  }
```

```
}
```

Σε αυτό το έξυπνο συμβόλαιο τα τεκμήρια αποκτούν ιδιότητες των erc-721. Όπως και με τα κρυπτονομίσματαerc-20 δημιουργείται η κατάλληλη λειτουργικότητα για να προσομοιάσει τα 721 τεκμήρια. Οι βασικές μέθοδοι επιστρέφουν τον αριθμό των τεκμηρίων που κατέχει ένας λογαριασμός, καθώς και σε ποιον ανήκει το κάθε τεκμήριο. Υπάρχει η μέθοδος που είναι υπεύθυνη για την μεταφορά τεκμηρίων από έναν λογαριασμό σε έναν άλλον και μια μέθοδος που όταν καλεστεί δίνει άδεια σε έναν λογαριασμό να χρησιμοποιήσει ένα συγκεκριμένο τεκμήριο από αυτά που έχει στην κατοχή του ο επικαλεστής της μεθόδου.

```
Pragma solidity >=0.4.0<0.6.0;

import"./4_cardbirth.sol";
import"./erc721.sol";


contract CardOwnership is ERC721, CardBirth {

   mapping (uint => address) cardApprovals;

function balanceOf721(address _owner) public view returns (uint256) {
returnownerCardCount[_owner];
   }

function ownerOf721(uint256 _tokenId) public view returns (address) {
returncardToOwner[_tokenId];
   }

function _transfer721(address _from, address _to, uint256 _tokenId) private {
ownerCardCount[_to] = ownerCardCount[_to].add(1);
ownerCardCount[_from] = ownerCardCount[_from].sub(1);
cardToOwner[_tokenId] = _to;
     emit Transfer721(_from, _to, _tokenId);
   }

function transferFrom721(address _from, address _to, uint256 _tokenId) internal  {
     require (cardToOwner[_tokenId] == msg.sender || cardApprovals[_tokenId] == msg.sender);
     _transfer721(_from, _to, _tokenId);
   }

function approve721(address _approved, uint256 _tokenId) internal  /*onlyOwnerOfCard(_tokenId)*/ {
     require(cardToOwner[_tokenId] == msg.sender);
cardApprovals[_tokenId] = _approved;
     emit Approval721(msg.sender, _approved, _tokenId);
   }
```

```
function transferFromContract721(address _to, uint _tokenId) internal {
    require(cardToOwner[_tokenId] == address(this));
    _transfer721(address(this), _to, _tokenId);
  }
}
```

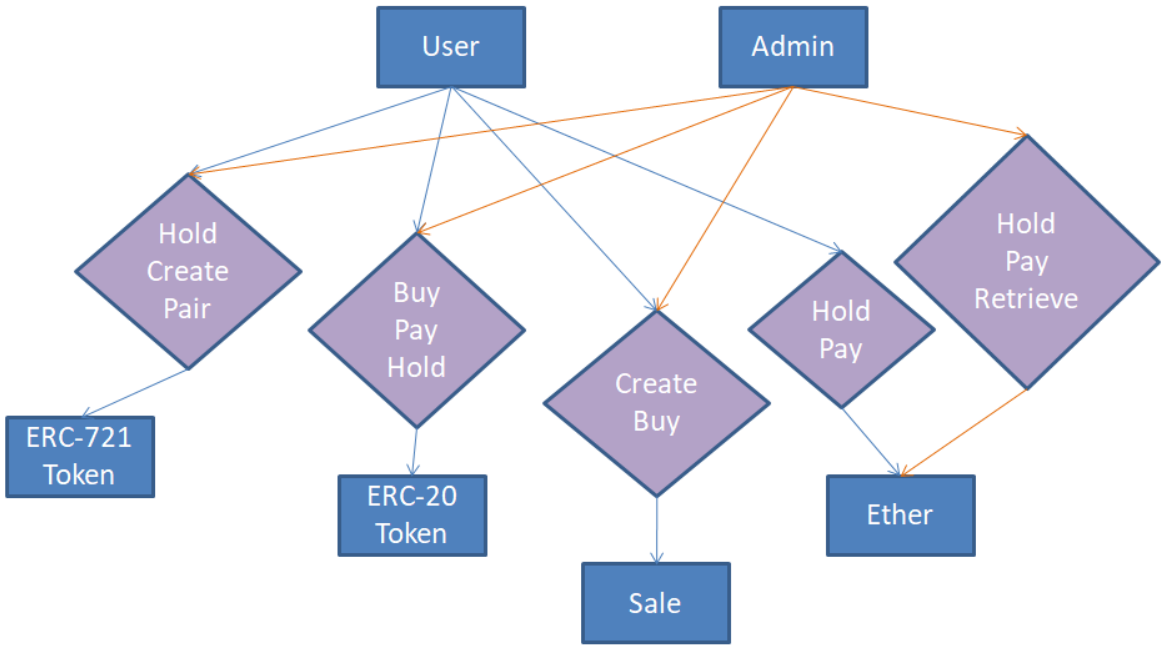
Τέλος το έξυπνο συμβόλαιο που αφορά την αγορά. Καταρχάς κληρονομεί όλες τις ιδιότητες των προηγούμενων έξυπνων συμβολαίων. Δημιουργεί τη δομή Sale που αποθηκεύει το id του τεκμηρίου την τιμή του σε κρυπτονομίσματα «gold» και το αν είναι διαθέσιμο προς πώληση με μία μοναδιαία μεταβλητή. Η μέθοδος setSale δημιουργεί ένα sale σε μορφή αγγελίας με τον περιορισμό ότι μόνο η ιδιοκτήτης του τεκμηρίου μπορεί να δημιουργήσει αγγελία για το συγκεκριμένο τεκμήριο την αποθηκεύει σε έναν πίνακα και αντιστοιχίζει το τεκμήριο με τον επικαλεστή της μεθόδου. Η επόμενη μέθοδος επιστρέφει τις αγγελίες του επικαλεστή της συνάρτησης. Η μέθοδος buySale δίνει την δυνατότητα σε έναν χρήστη να χρησιμοποιήσει τα κρυπτονομίσματα που κατέχει για να αγοράσει ένα τεκμήριο που βρίσκεται σε αγγελία από τη στιγμή που είναι ενεργή. Έπειτα η αγγελία απενεργοποιείται.Οι δύο τελευταίες λειτουργίες επιστρέφουν τα συνολικές αγγελίες και κάθε αγγελία ξεχωριστα.

```
pragma solidity >=0.4.0<0.6.0;

import"./5_cardownership.sol";

contract CardMarket is CardOwnership {

   event NewSale(address _owner, uint _tokenId, uint _tokenPrice);
   event NewBuy(address _newOwner);

   struct Sale{

uinttokenId;
uinttokenPrice;
     bool saleOn;
   }

Sale[] public sales;

mapping(uint => address) publicsaleToOwner;

mapping(address =>uint) ownerSaleCount;

functionsetSale(uint _tokenId, uint _tokenPrice) public {


       require(cardToOwner[_tokenId] == msg.sender);

uint id = sales.push(Sale( _tokenId, _tokenPrice, true)) - 1;

saleToOwner[id] = msg.sender;
```

```
ownerSaleCount[msg.sender] = ownerSaleCount[msg.sender].add(1);

        transferFrom721(msg.sender, address(this), _tokenId);

        emit NewSale(msg.sender, _tokenId, _tokenPrice);
    }


functiongetSaleByAddress(address _seller) public view returns(uint[] memory){
uint[] memory result = newuint[](ownerSaleCount[_seller]);
uint counter = 0;
for (uinti = 0; i<sales.length; i++) {
if (saleToOwner[i] == _seller) {
          result[counter] = i;
          counter++;
        }
      }
returnresult;
    }

functionbuySale(uint _saleId) public{

uint _tokenPrice = sales[_saleId].tokenPrice;
        transfer20(saleToOwner[_saleId], _tokenPrice);
        transferFromContract721( msg.sender, sales[_saleId].tokenId );
        Sale storage _sale = sales[_saleId];
        _sale.saleOn = false;
        emit NewBuy(msg.sender);

    }

functiongetTotalSales() public view returns(uint){
return (sales.length);
    }


functiongetSale(uint _saleId)public view returns(uint16, uint16, uint, bool, uint){
return(cards[sales[_saleId].tokenId].cardShape,                    cards[sales[_saleId].tokenId].cardColor,
sales[_saleId].tokenPrice, sales[_saleId].saleOn, _saleId);
    }
}
```

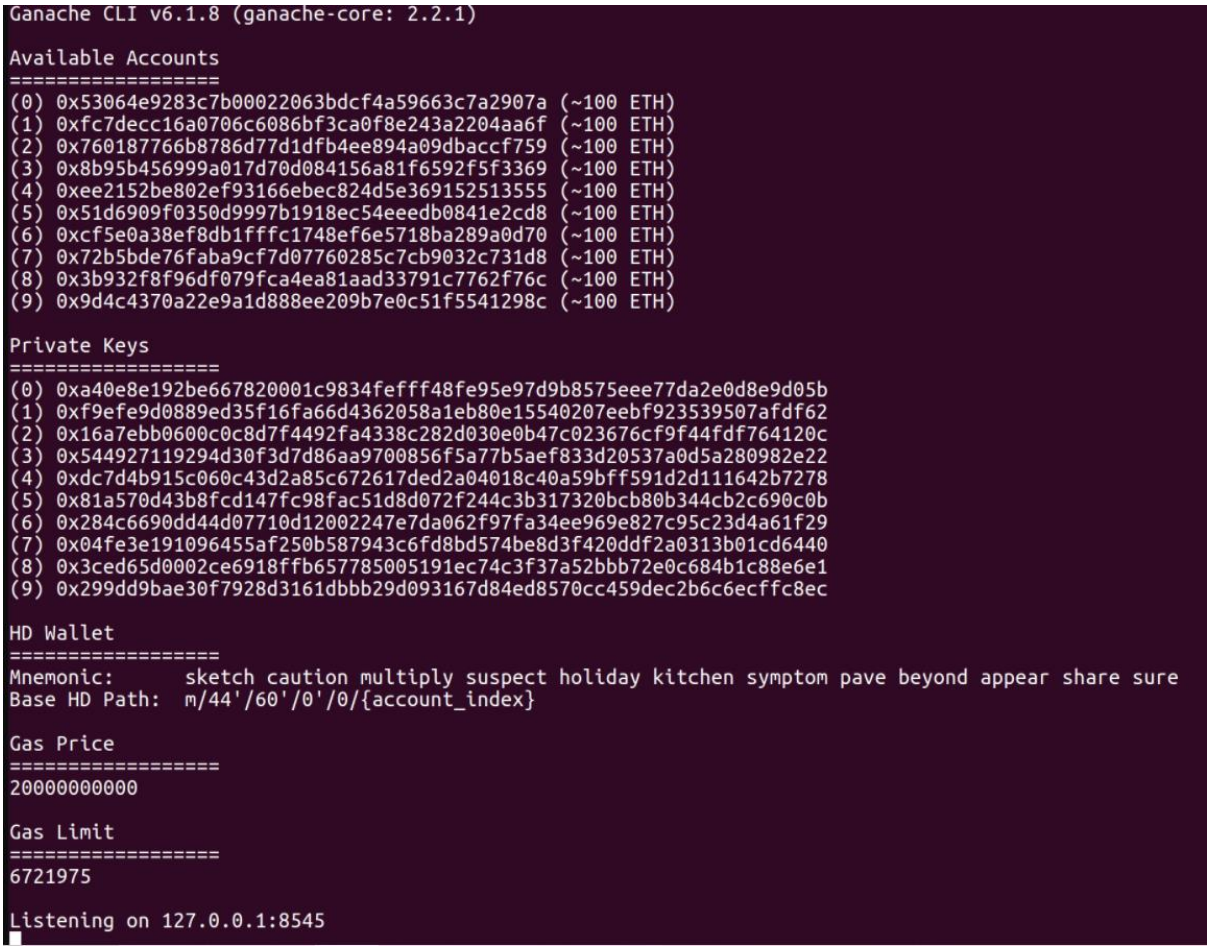
3.2 Παρουσίαση εφαρμογής

Στην παρακάτω εικόνα φαίνεται ένα τυπικό διάγραμμα οντοτήτων- σχέσεων (Εικόνα 3.1).



Εικόνα 3.1 Entity-Relationship diagram

Για την εκκίνηση του τοπικού δικτύου ganache-cli χρησιμοποιείται η εντολή ganache-cli στο τερματικό και αυτομάτως εμφανίζονται δέκα λογαριασμοί με 100 ether και τα δημόσια και ιδιωτικά κλειδιά τους όπως φαίνεται στην εικόνα 3.2. Για την παρουσίαση της εφρμογής χρησιμοποιήθηκαν τρεις από αυτούς τους λογαριασμούς με ονομασίες Cli-0, Cli-1, Cli-2.



Εικόνα 3.2 Τα δημόσια και τα ιδιωτικά κλειδιά των λογαριασμών του Ganache-Cli.

Επειδή χρησιμοποιήθηκε το Truffleframework εκτελέστηκε η εντολή truffleinit για να δημιουργήσει φάκελους και αρχεία απαραίτητα για την ανάπτυξη της δομής της εφαρμογής. Ένας από τους φακέλους που δημιουργήθηκαν στο πρόγραμμα είναι ο φάκελος contracts οπού εκεί αποθηκεύονται τα έξυπνα συμβόλαια που αναφέρθηκαν στο προηγούμενο κεφάλαιο και προϋπάρχει το συμβόλαιο με όνομα migrations. Το συμβόλαιο αυτό αναλαμβάνει την εγκατάσταση των υπόλοιπων έξυπνων συμβολαίων στο τοπικό ethereum δίκτυο που χρησιμοποιείται. Μαζί με τον φάκελο των συμβολαίων δημιουργήθηκε και ο φάκελος migrations όπου στον συγκεκριμένο φάκελο βρίσκεται ένα αρχείο javascript που ονομάζεται 1_initial_migrations.js. Το συγκεκριμένο αρχείο αξιοποιείται στην αποθήκευση του συμβολαίου migrations στο δίκτυο. Έπειτα δημιουργείται και ένα αντίστοιχο αρχείο με την ονομασία 2_contracts.js που είναι υπεύθυνο για την εγκατάσταση των έξυπνων συμβολαίων στο δίκτυο. Οι αριθμοί ένα και δύο στα ονόματα είναι απαραίτητοι διότι όταν ξεκινήσει η εγκατάσταση των συμβολαίων στο δίκτυο πρέπει πρώτα να εγκατασταθεί το συμβόλαιο migrations και μετά τα συμβόλαια της κατανεμημένης εφαρμογής. Ένα ακόμη σημαντικό αρχείο που δημιουργήθηκε κατά το truffleinit είναι το truffle-config.js στο οποίο βρίσκονται πληροφορίες για τον

μεταγλωττιστή και για το δίκτυο ethereum που γίνεται η εγκατάσταση της εφαρμογής. Στο παρακάτω αρχείο φαίνονται οι ρυθμίσεις της εφαρμογής.

```
module.exports = {

  networks: {
    development: {
      host: "127.0.0.1",
      port: 8545,
network_id: "*",
      gas: 4500000,
    },

  compilers: {
solc: {
      version: '0.4.25',
      optimizer: {
        enabled: true,
        runs: 200
      }
    }
  }
};
```

Εφόσον χρησιμοποιείται το τοπικό δίκτυο Ganache-Cli για την εγκατάσταση των έξυπνων συμβολαίων στο δίκτυο χρησιμοποιείται από προεπιλογή ο πρώτος λογαριασμός του τοπικού δικτύου που έχει δημιουργηθεί. Οπότε χρησιμοποιώντας την εντολή truffle migrate τα έξυπνα συμβόλαια μεταγλωττίζονται και έπειτα εγκαθίστανται στο τοπικό δίκτυο από τον πρώτο λογαριασμό. Αυτή η διαδικασία απαιτεί την χρήση ενός μικρού ποσού ether από το υπόλοιπο του πρώτου λογαριασμού. Στην εικόνα 3.3 παρακάτω φαίνεται το αποτέλεσμα που εμφανίζεται εφόσον δεν υπήρξε κάποιο πρόβλημα στη φάση της μεταγλώττισης ή της εγκατάστασης.

```
Compiling .\contracts\1_myerc20.sol...
Compiling .\contracts\2_myerc20use.sol...
Compiling .\contracts\3_cardfactory.sol...
Compiling .\contracts\4_cardbirth.sol...
Compiling .\contracts\5_cardownership.sol...
Compiling .\contracts\6_cardmarket.sol...
Compiling .\contracts\erc20.sol...
Compiling .\contracts\erc721.sol...
Compiling .\contracts\ownable.sol...
Compiling .\contracts\safemath.sol...
Writing artifacts to .\build\contracts

Using network 'development'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  ... 0xda7a6d505284d99d3e4a9a13483061f463e6b20ee91727015c6d3d56e00225f4
  Migrations: 0xdabb1c42fed604fc6c224dd82ada1de51dc86585
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Replacing CardMarket...
  ... 0x6d70952646459279 88fc4fdf6aa0a8c65698e6cf1d6f40b019fece39382ae439
  CardMarket: 0x736a5482a22902b2ea57bd5f945872cd71bc57f6
Saving artifacts...
PS C:\Users\kinim\consertadminnew>
```

Εικόνα 3.3

Ένα σημαντικό στοιχείο που πρέπει να αναφερθεί είναι η διεύθυνση του συμβολαίου που αποθηκεύτηκε στο δίκτυο «0x736a5482a22902b2ea57bd5f945872cd71bc57f6» η οποία θα χρησιμοποιηθεί στη συνέχεια στην επόμενη εφαρμογή.

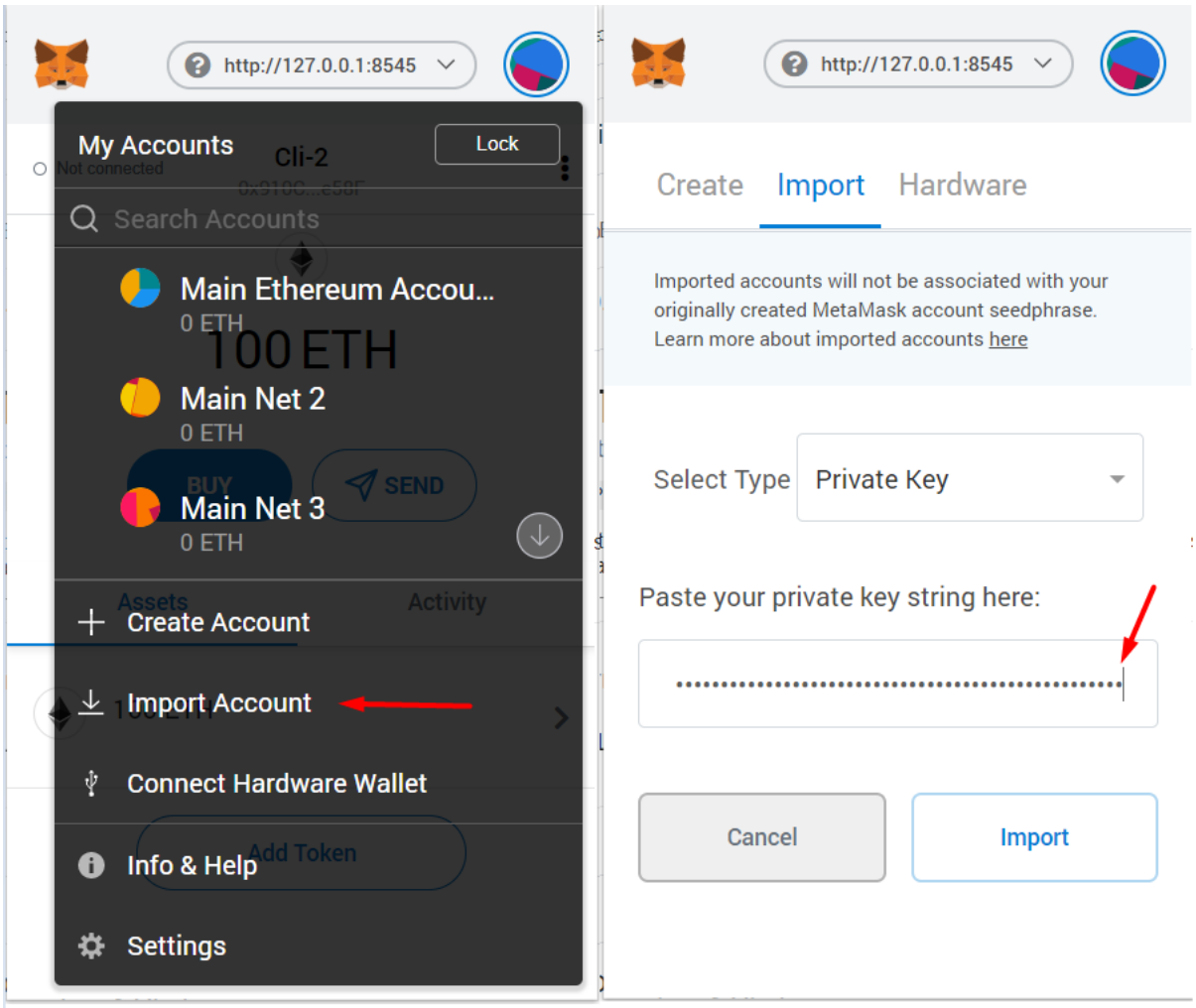
Κατά την ανάπτυξη της συγκεκριμένης εφαρμογής χρησιμοποιήθηκαν η γλώσσα σήμανσης HTML, οι βιβλιοθήκες bootstrap για την μορφοποίηση της εφαρμογής, web3 και truffle-contract για την αλληλεπίδραση του χρήστη με το έξυπνο συμβόλαιο μέσω του Metamask. Αξίζει να σημειωθεί ότι η ασφάλεια της εφαρμογής εξαρτάται απόλυτα από το έξυπνο συμβόλαιο που έχει αναπτυχθεί διότι όλοι οι περιορισμοί της εφαρμογής εφαρμόζονται μέσα στο ethereumblockchain δηλαδή στο back-end. Για αυτό το λόγο δεν υπήρξε καμία μορφή περιορισμού στο από την μεριά του front-end.

Για την σύνδεση των λογαριασμών του ganache-cli με το metamask χρησιμοποιείται το ιδιωτικό κλειδί κάθε λογαριασμού. Για τη σύνδεση του λογαριασμού που φαίνετε παρακάτω απαιτείται η διαδικασία που φαίνεται στην εικόνα 4.4.

Παράδειγμα ιδιωτικού και δημόσιο κλειδιού ενός λογαριασμού από το Ganache-Cli

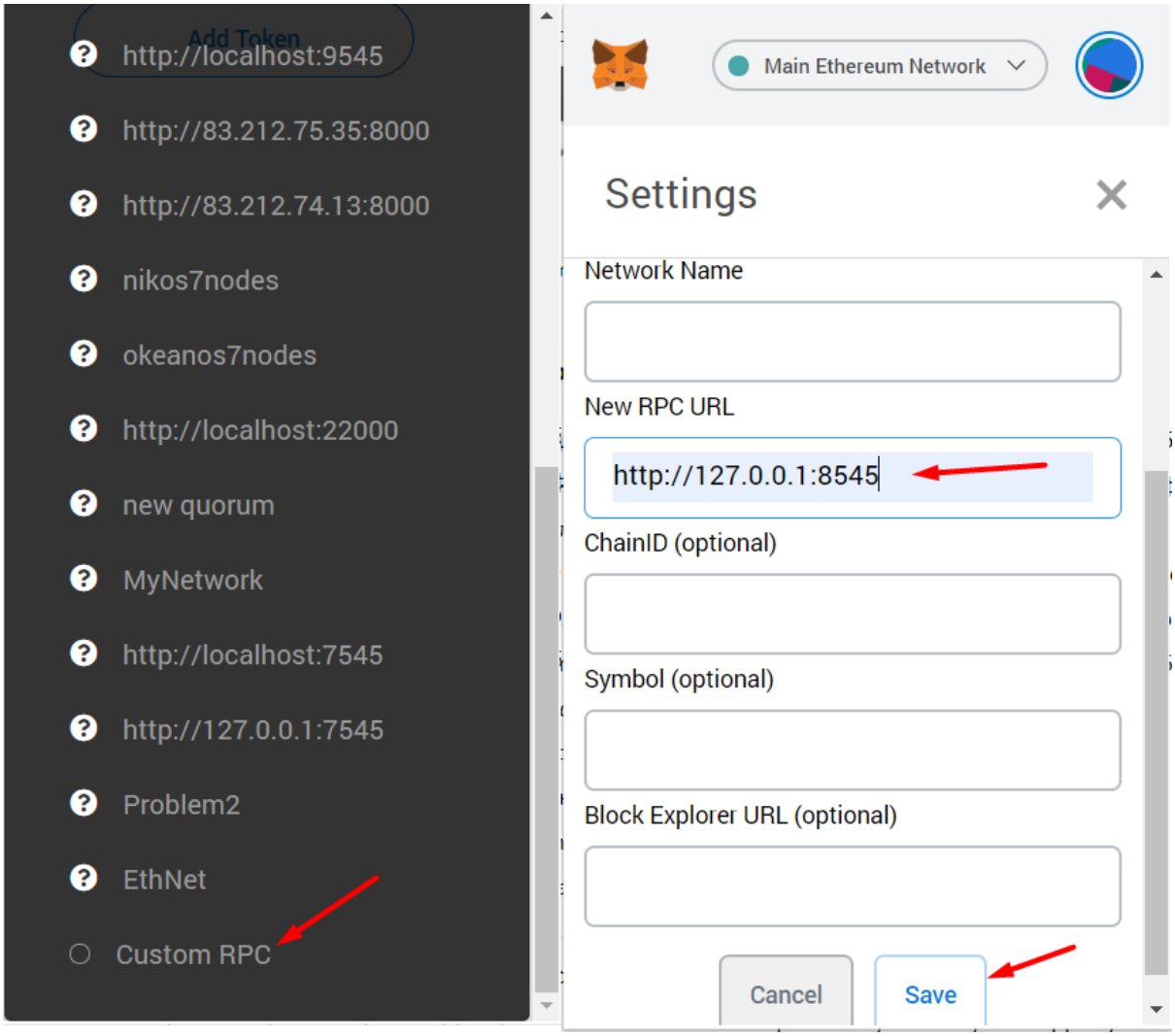
Δημόσιο : 0x910C797DA73a0FFE7995997F0374cfeF8560e58F

Ιδιωτικό : 0x8042e1e5cf1b0f536920c77fac32e462b7b8325548a355ab5690064b9064fb0f



Εικόνα 3.4 Στο βελάκι που φαίνεται στην δεξιά εικόνα τοποθετείται το ιδιωτικό κλειδί του λογαριασμού. Πατώντας import ο λογαριασμός συνδέεται με το metamask.

Έπειτα πρέπει να γίνει η σύνδεση του Metamask με το δίκτυο του Ganache-Cli. Αυτή η διαδικασία είναι απαραίτητη προκειμένου να μπορέσει ένας χρήστης να χρησιμοποιήσει την κατανεμημένη εφαρμογή που δημιουργήθηκε. Το δίκτυο είναι τοπικό οπότε χρησιμοποιείται η σύνδεση 127.0.0.1 στην πόρτα 8545 που είναι η προεπιλεγμένη πόρτα στην ακούει του τοπικό δίκτυο. Άρα στο Metamask τοποθετείται το URLhttp://127.0.0.1:8545 όπως φαίνεται στην παρακάτω εικόνα (3.5).



Εικόνα 3.5 Επιλέγεται η επιλογή CustomRPC όπως φαίνεται στην εικόνα αριστερά και έπειτα τοποθετείται το RPCURL και το δίκτυο αποθηκεύεται στο Metamask.

Με το πακέτο npmlite-server, το οποίο χρησιμοποιείται για γρήγορη ανάπτυξη εφαρμογών, ξεκινάει η εφαρμογή χρησιμοποιώντας την εντολή npmrundev στην διεύθυνση http://localhost:3000.

Στην συγκεκριμένη εφαρμογή υπάρχουν δύο είδη χρηστών: Ο admin ή owner ο οποίος είναι δημιουργός της εφαρμογής και ο απλός χρήστης. Ο απλός χρήστης έχει τις εξής δυνατότητες:

1) Μπορεί να δημιουργήσει τα πρώτα του τεκμήρια από την στιγμή που δεν είχε προηγούμενα.
2) Μπορεί μόνο να δει τα δικά του τεκμήρια καθώς και αυτά που βρίσκονται προς πώληση.
3) Μπορεί να ζευγαρώσει τα τεκμήρια του ώστε να δημιουργήσει καινούρια τεκμήρια.
4) Μπορεί να αγοράσει τα κρυπτονομίσματα που δημιουργήθηκαν στα πλαίσια της εφαρμογής χρησιμοποιώντας ether. Το ether αποθηκεύεται μέσα στο έξυπνο συμβόλαιο.
5) Μπορεί να βάλει προς πώληση τα τεκμήρια του για ένα ποσό των προαναφερθέντων κρυπτονομισμάτων.
6) Μπορεί να αγοράσει τεκμήρια που είναι προς πώληση χρησιμοποιώντας τα κρυπτονομίσματα του.

Ο admin μπορεί να κάνει όλες τις παραπάνω ενέργειες και ακόμα μπορεί να συλλέξει το ether που βρίσκεται αποθηκευμένο στο έξυπνο συμβόλαιο.

Στη συγκεκριμένη περίπτωση ο admin είναι ο δημιουργός του έξυπνου συμβολαίου καθώς και αυτός που έκανε την εγκατάσταση του συμβολαίου στο δίκτυο άρα ο πρώτος λογαριασμός στο ganache-cli κόμβο. Οι λογαριασμοί δύο και τρία είναι απλοί χρήστες. Όπως αναφέρθηκε και παραπάνω τα ονόματα των λογαριασμών είναι Cli-0, Cli-1, Cli-2 και μία υπόθεση εργασίας είναι ότι ο κάθε χρήστης χρησιμοποιεί μόνο έναν λογαριασμό ακόμη και αν δεν είναι απαραίτητο.

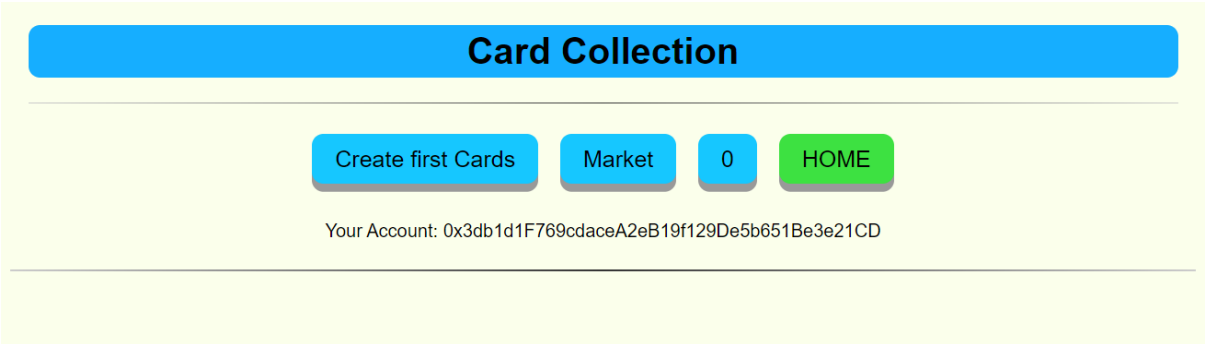
Όνομα	Ρόλος	Δημόσιο κλειδί	Ιδιωτικό κλειδί
Cli-0	Admin	0x3db1d1F769cdaceA2eB19f129De5b651Be3e21CD	0x4e15719ead0486ba47f7dc265c7a47a2f3df939913b0cc405ca339467ba80787
Cli-1	User	0x0ffDbC03DaE52a69Ee2f1DCBc35C45eBF7FC913B	0xc414305839f4ac9a9f29925e4d062a3af9565bd1d0b0b2431751487a53c4ba3c
Cli-2	User	0x910C797DA73a0FFE7995997F0374cfeF8560e58F	0x8042e1e5cf1b0f536920c77fac32e462b7b8325548a355ab5690064b9064fb0f

Πίνακας 3.1

Όταν ένας χρήστης επισκέπτεται την εφαρμογή μεταφέρεται στην αρχική σελίδα στην οποία μπορεί να δει τα τεκμήρια που έχει καθώς και το ποσό των κρυπτονομισμάτων του. Στη πρώτη επίσκεψη του χρήστη τα τεκμήρια όπως και τα κρυπτονομίσματα του είναι μηδέν (εικόνα 4.3). Τα τεκμήρια ενός χρήστη έχουν έναν δεκαεξαδικό αριθμό που προσομοιάζει μια μορφή DNA όπως προαναφέρθηκε στο κεφάλαιο της ανάπτυξης του έξυπνου συμβολαίου. Το DNA χρησιμοποιείται για να δώσει κάποια χαρακτηριστικά σε κάθε τεκμήριο όπως το σχήμα του, το χρώμα του και το πόσες φορές μπορεί «ζευγαρώσει». Το «ζευγάρωμα» τεκμηρίων είναι στην ουσία ο συνδυασμός των DNA δύο διαφορετικών τεκμηρίων. Στην εικόνα 3.6 φαίνεται ένα τεκμήριο.

Εικόνα 3.6

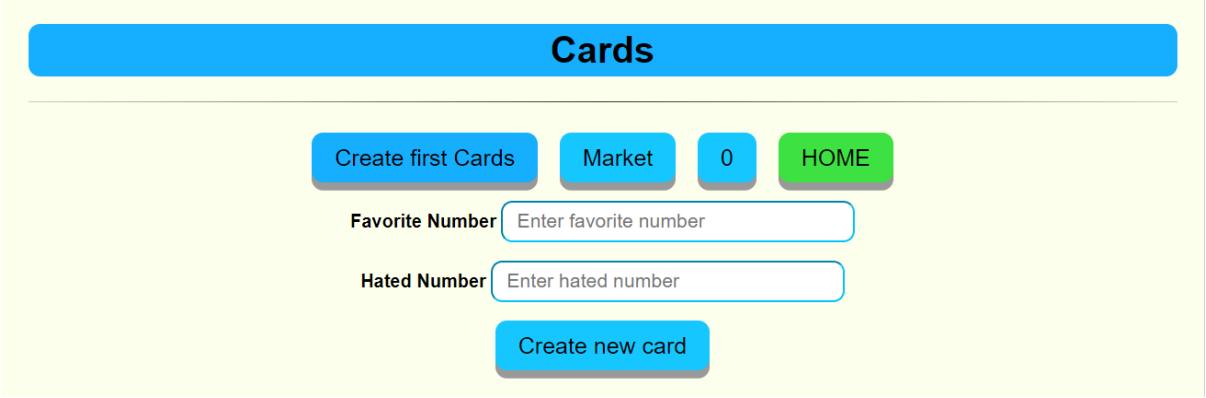
Στο συγκεκριμένο τεκμήριο το σχήμα είναι παραλληλεπίπεδο και το χρώμα γκρι.



Εικόνα 3.7 Ο χρήστης δεν έχει τεκμήρια και κρυπτονομίσματα. Ο χρήστης μπορεί να δει τον

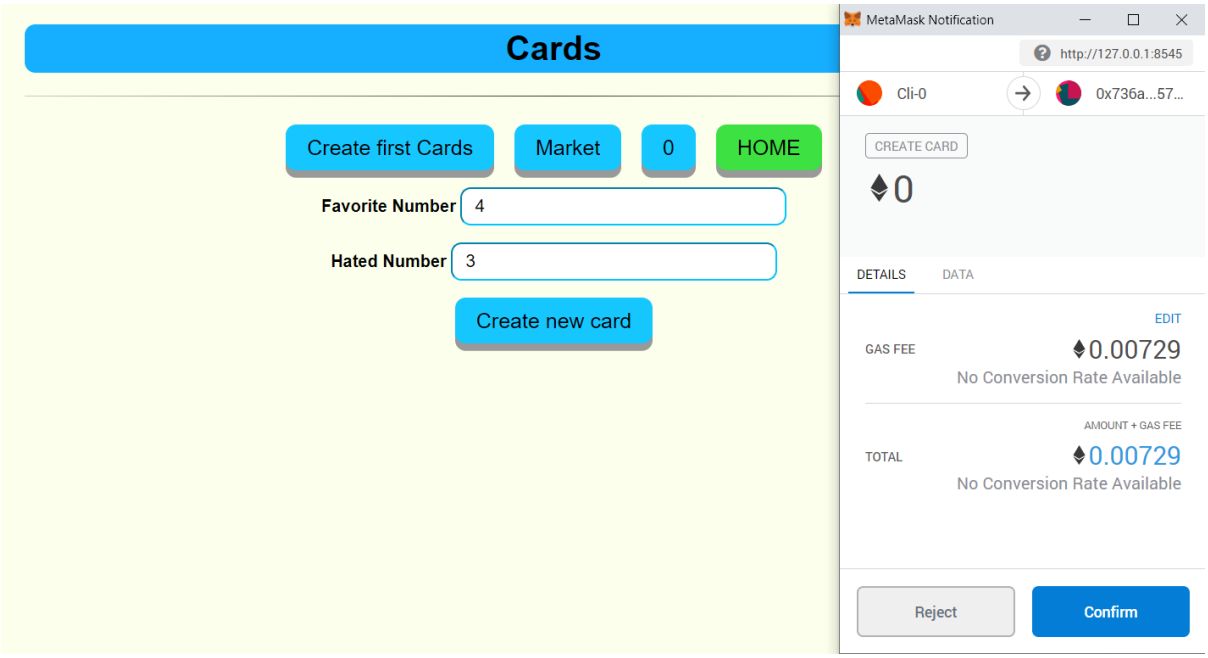
λογαριασμό με τον οποίο είναι συνδεδεμένος στο δίκτυο.

Πατώντας το κουμπί CreatefirstCards (για την εφαρμογή όλα τα τεκμήρια ονομάζονται cards αλλά θα μπορούσαν να έχουν οποιαδήποτε ονομασία) εμφανίζεται ένα νέο παράθυρο (εικόνα 3.8) στο οποίο ο χρήστης πληκτρολογεί δύο αριθμούς.



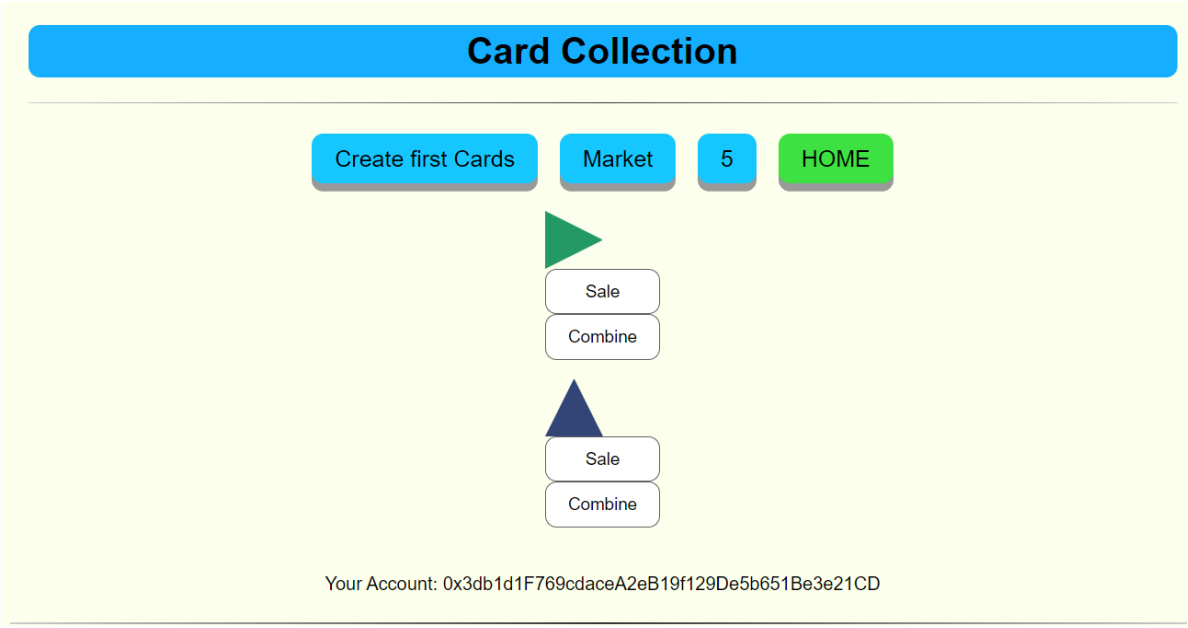
Εικόνα 3.8 Σελίδα που ο χρήστης μπορεί να δημιουργήσει τα πρώτα του τεκμήρια.

Εφόσον τοποθετηθούν οι δύο αριθμοί ο χρήστης πατάει το κουμπί CreatenewCard και εμφανίζεται ένα παράθυρο του metamask το οποίο ζητάει την επιβεβαίωση του χρήστη. Στο παράθυρο αυτό φαίνεται το κόστος της χρήσης της μεθόδου createCard που βρίσκεται στο έξυπνο συμβόλαιο. Το κόστος της μεθόδου είναι μηδεν. Παρατηρείται επίσης ότι αυτή η διαδικασία απαιτεί από τον χρήστη να σπαταλήσει ένα μικρό ποσό ether το οποίο ονομάζεται gasfee. Το συνολικό ποσό ether που θα σπαταλήσει ο χρήστης είναι το άθροισμα του κόστους της μεθόδου και το gasfee (εικόνα 3.9).



Εικόνα 3.9 Το metamask αναγνωρίζει ότι πρόκειται να χρησιμοποιηθεί μια μέθοδος που «γράφει» στο blockchain δίκτυο και εμφανίζει ένα παράθυρο που ζητάει επιβεβαίωση από τον χρήστη.

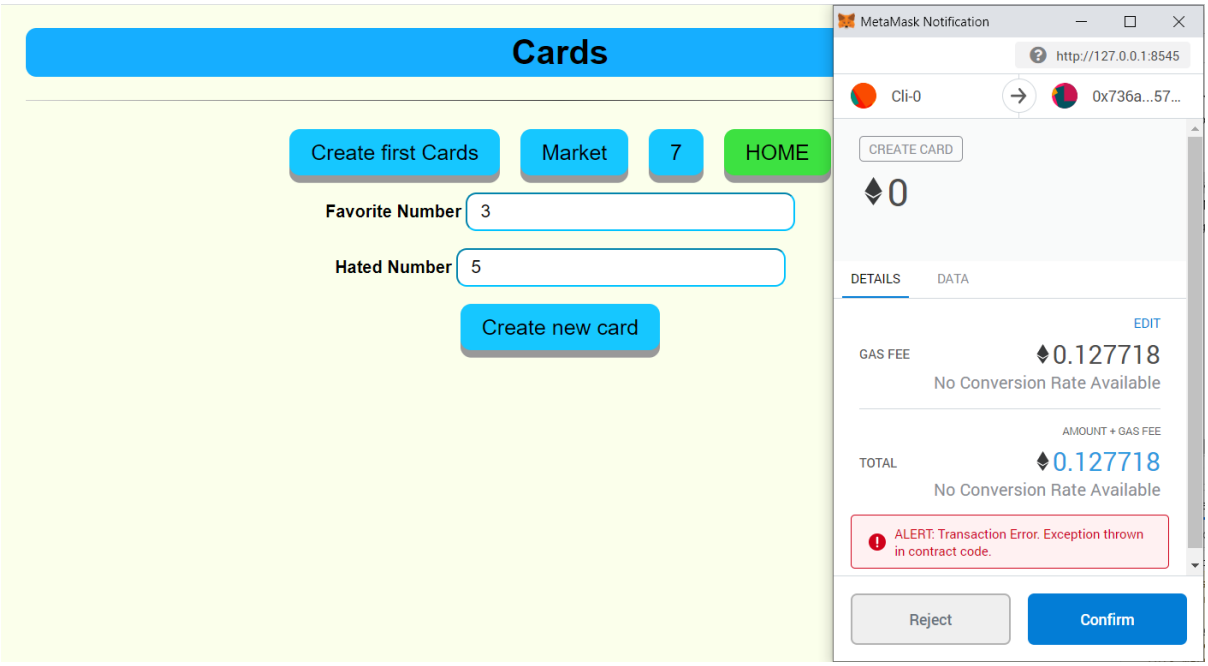
Αφού ο χρήστης πατήσει confirm η συναλλαγή ολοκληρώνεται και πλέον ο χρήστης έχει στην κατοχή του δύο τεκμήρια και ένα ποσό κρυπτονομισμάτων που δίνονται μαζί με τα τεκμήρια όπως φαίνεται στην εικόνα 3.10.



Εικόνα 3.10 Ο χρήστης έχει δύο τεκμήρια στην κατοχή του και πέντε κρυπτονομίσματα.

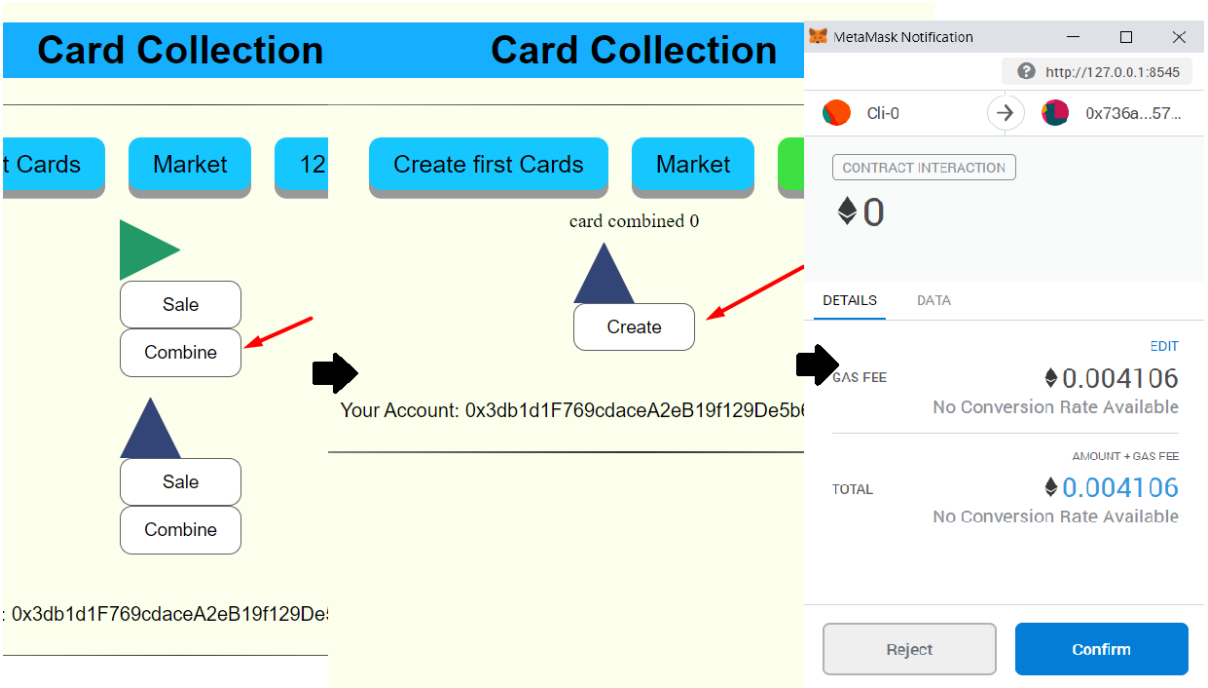
Τα τεκμήρια που δημιουργήθηκαν είναι διαφορετικά μεταξύ τους διότι έχουν διαφορετικό DNA. Στο σημείο αυτό θα πρέπει να αναφερθεί ότι ο λόγος που δημιουργούνται δύο τεκμήρια αντί για ένα είναι γιατί ο χρήστης μπορεί να «ζευγαρώσει» μόνο με τεκμήρια που έχει στην κατοχή του το οποίο θα φανεί παρακάτω.

Όπως παρατηρείται στην εικόνα 3.11 αν ο χρήστης θελήσει να ξαναδημιουργήσει τεκμήρια χρησιμοποιώντας το CreatefirstCard το metamask βγάζει μία προειδοποίηση ότι η συναλλαγή δεν θα πραγματοποιηθεί. Αυτό συμβαίνει διότι η μέθοδος που καλείται από το έξυπνο συμβόλαιο έχει ένα περιορισμό που δεν επιτρέπει στο χρήστη να χρησιμοποιήσει αυτήν την μέθοδο εφόσον έχει ήδη τεκμήρια. Εάν ο χρήστης παραβλέψει αυτήν την προειδοποίηση τότε απλά θα αφαιρεθεί το gasfee που χρειάζεται για την συναλλαγή και θα επιστραφεί σφάλμα.



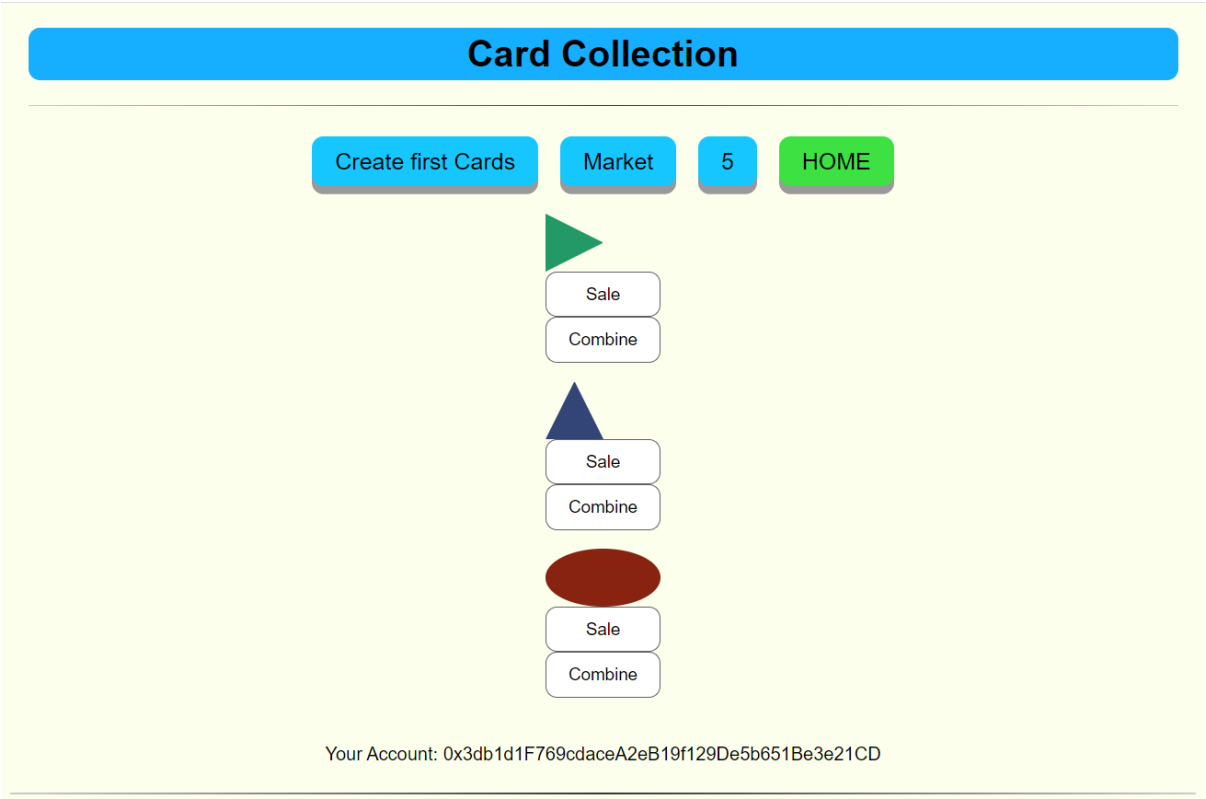
Εικόνα 3.11 Το metamask επισημαίνει ότι η συναλλαγή θα επιστρέψει σφάλμα διότι ο χρήστης έχει ήδη τεκμήρια στην κατοχή του.

Για το «ζευγάρωμα» δύο τεκμηρίων ο χρήστης πατάει το κουμπί combine σε ένα από τα τεκμήρια που θέλει να ζευγαρώσει. Έπειτα ο χρήστης επιλέγει το δεύτερο τεκμήριο και πατάει το κουμπί create. Ένα παράθυρο του metamask εμφανίζεται και ο χρήστης επιλέγει το confirm και το νέο τεκμήριο δημιουργείται (εικόνα 3.12).



Εικόνα 3.12 Διαδικασία με την οποία επιτυγχάνεται το «ζευγάρωμα» τεκμηρίων.

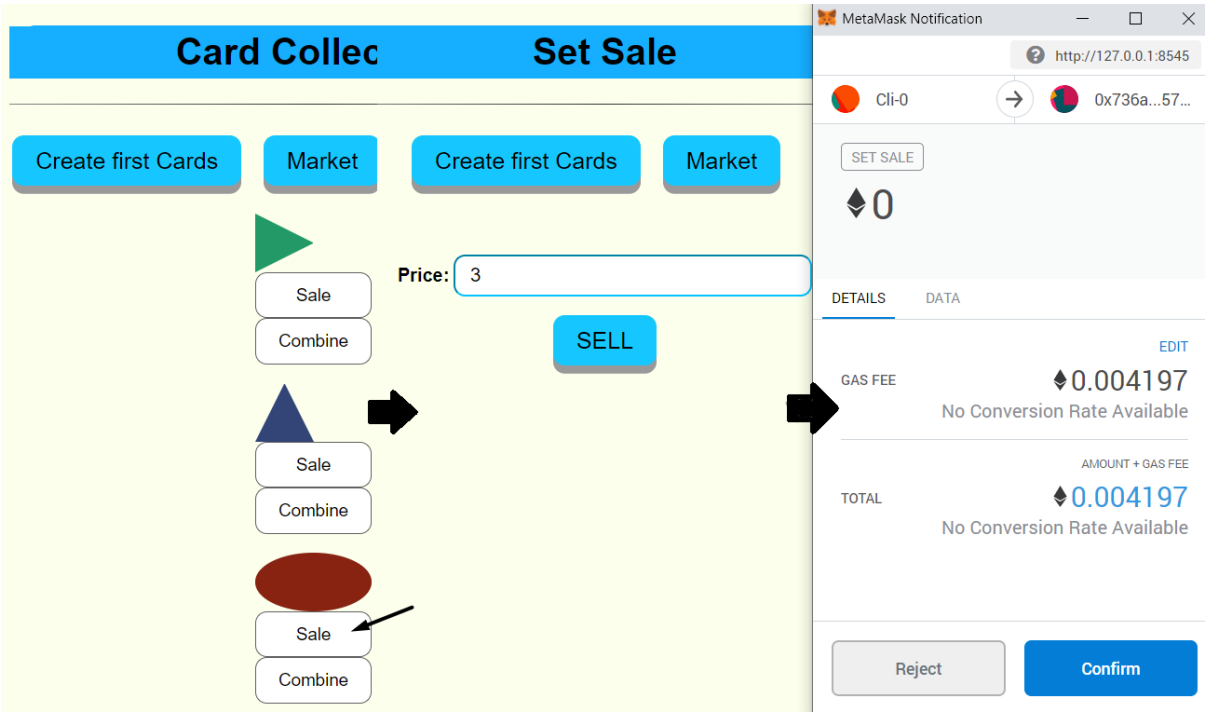
Από την στιγμή που δύο τεκμήρια «ζευγαρώσουν» γίνονται κάποιες αλλαγές στα χαρακτηριστικά τους. Όπως προαναφέρθηκε κάθε τεκμήριο έχει ένα συγκεκριμένο όριο που του επιτρέπει να «ζευγαρώσει». Κάθε φορά που συμμετέχει στο «ζευγάρωμα» τότε ένας μετρητής που βρίσκεται στα χαρακτηριστικά του αυξάνεται. Ο μετρητής αυτός ξεκινάει από το μηδέν την στιγμή που το τεκμήριο δημιουργείται. Όταν ο μετρητής γίνει ίσος με το όριο του «ζευγαρώματος» το metamask θα προειδοποιήσει τον χρήστη ότι δεν είναι δυνατή η συναλλαγή αυτή και θα βγάλει σφάλμα. Επίσης κάθε τεκμήριο έχει ένα χρονικό όριο για το πόσο συχνά μπορεί να «ζευγαρώσει» για παράδειγμα ένα λεπτό οπότε εάν ο χρήστης επιδιώξει να χρησιμοποιήσει αυτό το τεκμήριο πριν περάσει αυτό το χρονικό όριο το metamask θα προειδοποιήσει ξανά για σφάλμα. Οι παραπάνω περιορισμοί δηλώνονται μέσα στις μεθόδους του έξυπνου συμβολαίου και για αυτό το metamask μπορεί να της εντοπίσει. Παρακάτω εμφανίζεται το νέο τεκμήριο που δημιουργήθηκε (εικόνα 3.13).



Εικόνα 3.13 Το τεκμήριο που δημιουργήθηκε από το «ζευγάρωμα» των δύο προηγούμενων.

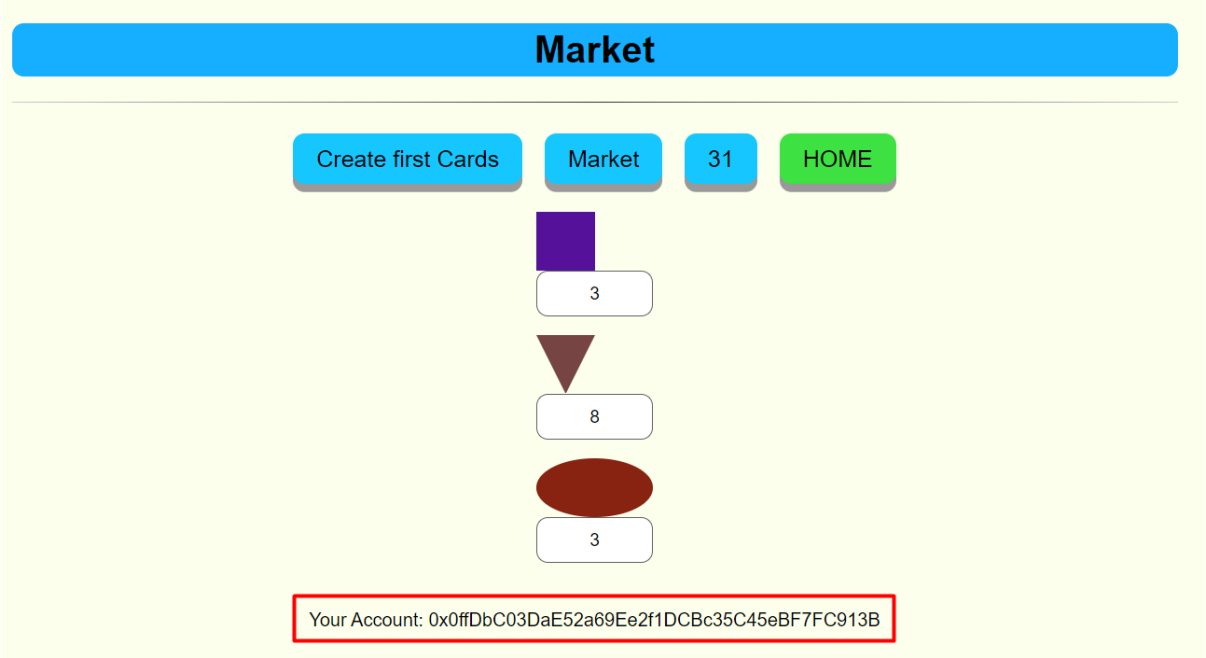
Παρατηρείται ότι νέο τεκμήριο έχει διαφορετικό DNA από τα δύο που συνέβαλαν στη δημιουργία του. Όταν δημιουργείται ένα τεκμήριο ενεργοποιείται ο μετρητής χρόνου που έχει ώστε να μην μπορεί να ζευγαρώσει απευθείας.

Για να βάλει προς πώληση ένα τεκμήριο ο χρήστης επιλέγει το κουμπί Sale κάτω από το τεκμήριο και έπειτα επιλέγει την τιμή, σε κρυπτονομίσματα της εφαρμογής, που θεωρεί ότι αξίζει. Έπειτα το metamask εμφανίζεται για την επιβεβαίωση της συναλλαγής (εικόνα 3.14).



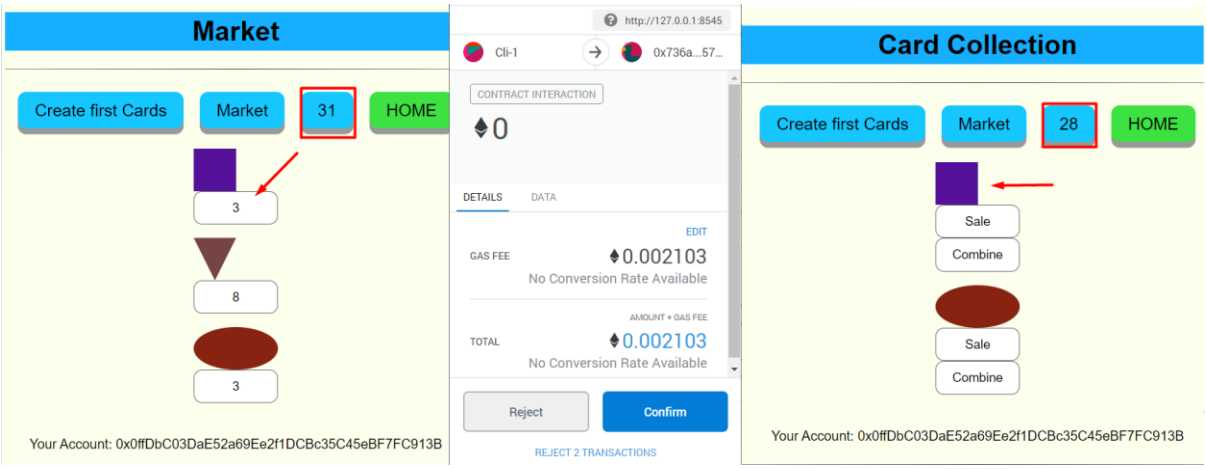
Εικόνα 3.14 Ο χρήστης βάζει προς πώληση το οβάλ τεκμήριο.

Όταν ένας χρήστη βάλει ένα τεκμήριο προς πώληση, παύει να ανήκει πλέον σε αυτόν. Το τεκμήριο ανήκει στο έξυπνο συμβόλαιο και συγκεκριμένα στην διεύθυνση του έξυπνου συμβολαίου μέχρι κάποιος χρήστης να δεχτεί την συναλλαγή. Όλα τα τεκμήρια που βρίσκονται προς πώληση είναι ορατά σε όλους τους χρήστες της εφαρμογής και έχουν την δυνατότητα να τα αγοράσουν εφόσον δώσουν το ποσό των κρυπτονομισμάτων που αντιστοιχεί στη συγκεκριμένη πώληση. Τα τεκμήρια που βρίσκονται προς πώληση εμφανίζονται στο Market (εικόνα 3.15).

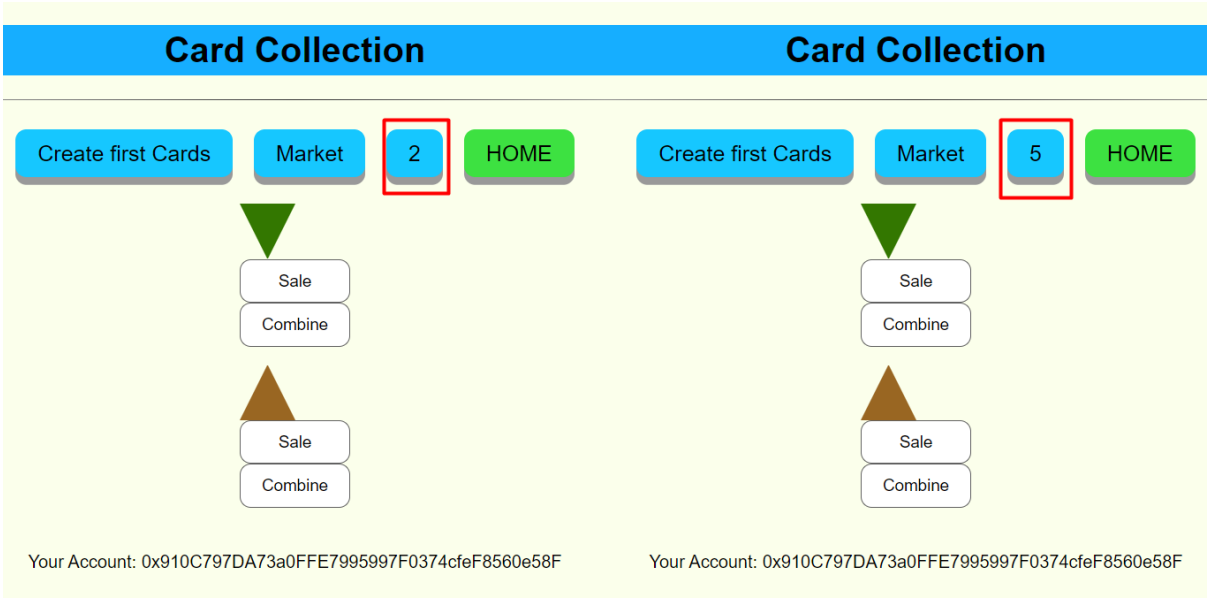


Εικόνα 3.15 Το Market της εφαρμογής. Παρατηρείται ότι ο λογαριασμός που εμφανίζεται στο κόκκινο πλαίσιο είναι ο Cli-1 ενώ ο λογαριασμός του χρήστη που έβαλε προς πώληση το οβάλ τεκμήριο ήταν ο Cli-0.

Όταν ένας χρήστης αγοράσει ένα τεκμήριο τότε το ποσό που αναγράφεται κάτω από το τεκμήριο αφαιρείται από το υπόλοιπο του και προστίθεται στο υπόλοιπο του χρήστη που είχε δημιουργήσει την πώληση. Το τεκμήριο πλέον ανήκει στον αγοραστή και έχει την δυνατότητα να το βάλει προς πώληση η να το «ζευγαρώσει» εφόσον τηρούνται οι συνθήκες που προαναφέρθηκαν (εικόνες 3.16, 3.17).

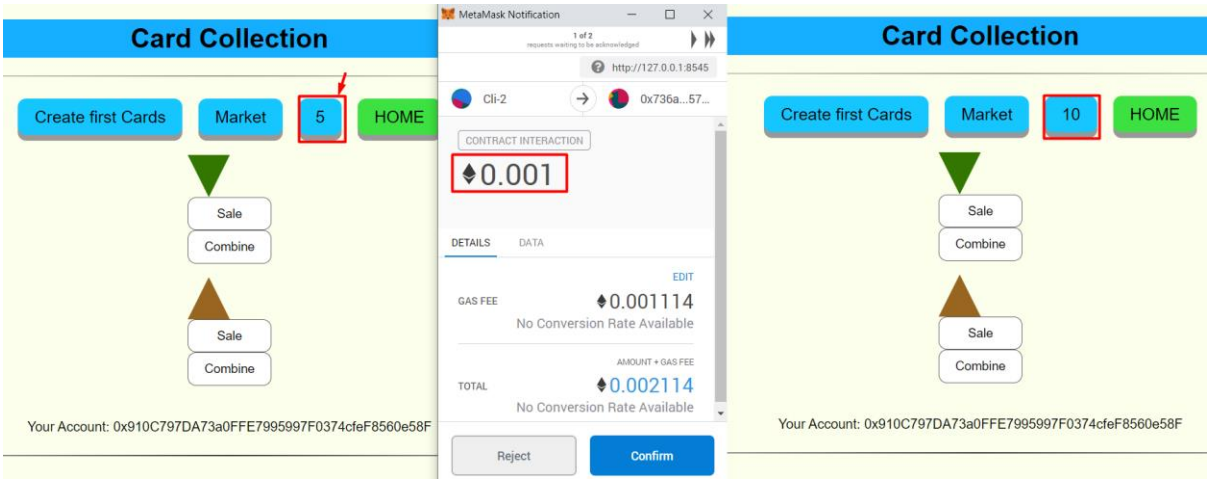


Εικόνα 3.16 Παρατηρείται ότι ο Cli-1 αγόρασε το τετράγωνο τεκμήριο για το ποσό των τριών κρυπτονομισμάτων τα οποία αφαιρέθηκαν από το υπόλοιπο του και πλέον το τεκμήριο του ανήκει.



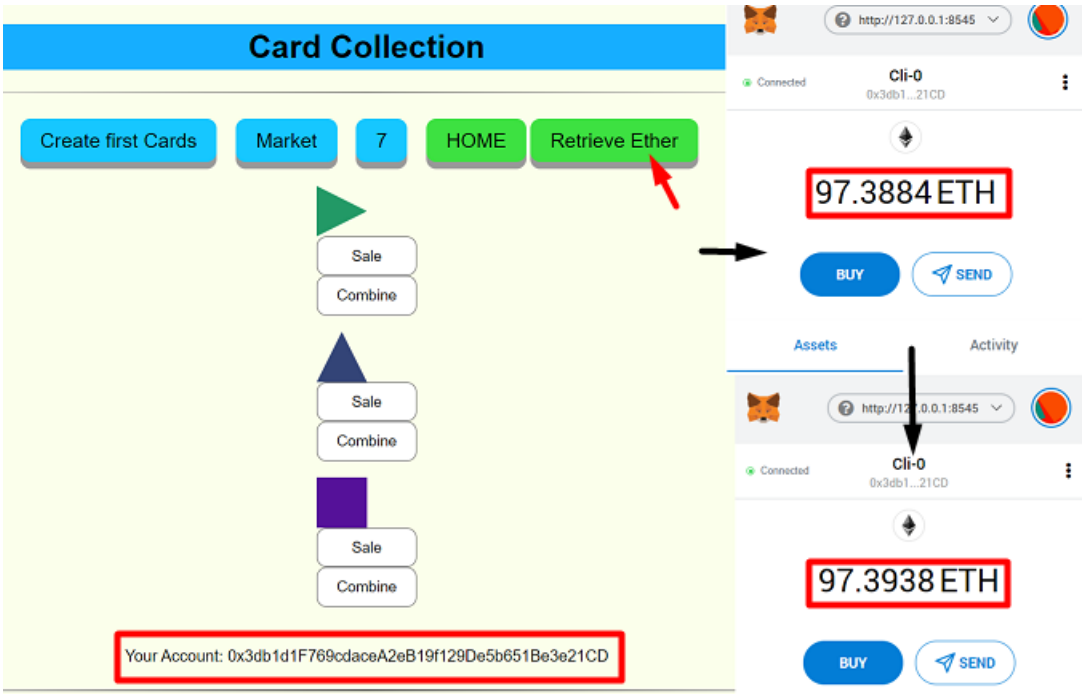
Εικόνα 3.17 Το υπόλοιπο το Cli-2 που είχε βάλει προς πώληση το τετράγωνο τεκμήριο αυξήθηκε κατά το ποσό που πληρώθηκε από τον Cli-1.

Τέλος ο χρήστης πατώντας το κουμπί που αναγράφει το υπόλοιπό του στα κρυπτονομίσματα της εφαρμογής μπορεί να αγοράσει πέντε επιπλέον κρυπτονομίσματα για το ποσό των 0.001 ether (εικόνα 3.18). Από τη στιγμή που γίνεται η αγορά των κρυπτονομισμάτων το ether ανήκει στην διεύθυνση του έξυπνου συμβολαίου και μόνο ο admin έχει την δυνατότητα να το ανακτήσει.



Εικόνα 3.18 Παρατηρείται ότι το υπόλοιπο του λογαριασμού αυξήθηκε κατά 5. Στη συγκεκριμένη συναλλαγή φαίνεται στο metamask ότι εκτός από το gasfee ο χρήστης πρέπει να πληρώσει και 0.001 ether, δηλαδή την τιμή που επιβάλλεται από την μέθοδο του έξυπνου συμβολαίου.

Όπως προαναφέρθηκε ο admin έχει όλες τις δυνατότητες που έχει ένας απλός χρήστης αλλά με την διαφορά ότι του δίνεται η δυνατότητα να ανακτήσει το ether που είναι αποθηκευμένο στην διεύθυνση του έξυπνου συμβολαίου (εικόνα 4.15) από τις αγορές των κρυπτονομισμάτων. Η σελίδα του admin έχει ένα κουμπί που γράφει RetrieveEther και πατώντας το εμφανίζεται ένα παράθυρο του metamask για την επιβεβαίωση της συναλλαγής. Είναι σημαντικό να αναφερθεί ότι μόνο ο admin έχει αυτήν την δυνατότητα. Έστω ότι ένας κακόβουλος χρήστης βρεθεί στην σελίδα του admin και πατήσει το κουμπί που αναφέρθηκε προηγμένως. Το metamask αναγνωρίζει πως ο λογαριασμός του χρήστη είναι διάφορος του admin και προειδοποιεί για σφάλμα. Αυτό συμβαίνει γιατί στο έξυπνο συμβόλαιο υπάρχει ένας περιορισμός που ελέγχει εάν ο λογαριασμός του χρήστη είναι ίδιος με αυτόν του admin.



Εικόνα 3.18 Ο admin, δηλαδή ο Cli-0, επιλέγει να ανακτήσει το ether από το έξυπνο συμβόλαιο και όπως φαίνεται το υπόλοιπο του σε ether αυξήθηκε.

3.3 Ανάπτυξη εφαρμογής από κακόβουλο χρήστη

Η συγκεκριμένη εφαρμογή αναπτύσσεται με σκοπό να αναδείξει τα προβλήματα που μπορούν να δημιουργηθούν εάν δεν υπάρχουν οι κατάλληλοι περιορισμοί στην ανάπτυξη ενός έξυπνου συμβολαίου. Στο ethereumnetwork υπάρχει η δυνατότητα ανάκτησης της διεύθυνσης ενός έξυπνου συμβολαίου που είναι αποθηκευμένο στο δίκτυο και μπορεί να χρησιμοποιηθεί από τρίτους για τους δικούς τους σκοπούς. Ένα πολύ απλό παράδειγμα είναι το παρακάτω:

Έστω ότι υπάρχει ένα έξυπνο συμβόλαιο αποθηκευμένο στο Ethereum δίκτυο που αποθηκεύει ether. Σε αυτό το συμβόλαιο υπάρχει μία μέθοδος με την οποία ένας χρήστης μπορεί να ανακτήσει το ether που είναι αποθηκευμένο στο συμβόλαιο. Εάν δεν υπάρχουν οι κατάλληλοι περιορισμοί για τον έλεγχο των λογαριασμών που μπορούν να χρησιμοποιήσουν την παραπάνω μέθοδο ή εάν αυτή η μέθοδος είναι δημόσια τότε ακόμα και αν η υλοποίηση της εφαρμογής είναι ανεπτυγμένη με τέτοιο τρόπο ώστε να προσδίδει την μέγιστη ασφάλεια εκτός του ethereum δικτύου, κάποιος που έχει πρόσβαση στο δίκτυο μπορεί να ανακτήσει την διεύθυνση του έξυπνου συμβολαίου στο δίκτυο και να χρησιμοποιήσει την μέθοδο αυτή ώστε να πάρει ο ίδιος το αποθηκευμένο ether παρακάμπτοντας οποιαδήποτε μορφή ασφάλειας εκτός του δικτύου.

Μία πιο απλουστευμένη μορφή του παραπάνω παραδείγματος αναπτύσσεται στη συγκεκριμένη εφαρμογή. Στο έξυπνο συμβόλαιο που αναπτύχθηκε στην προηγούμενη εφαρμογή υπάρχει μία μέθοδος η οποία δίνει τη δυνατότητα σε ένα χρήστη να δει, δηλαδή να «διαβάσει» από το συμβόλαιο, οποιοδήποτε τεκμήριο θελήσει. Όπως παρατηρήθηκε παραπάνω, κατά την παρουσίαση της εφαρμογής, ένας χρήστης δεν έχει τη δυνατότητα να δει όλα τα τεκμήρια που υπάρχουν στο συμβόλαιο παρά μόνο αυτά που κατέχει ο ίδιος και όσα βρίσκονται προς πώληση. Η προαναφερθείσα μέθοδος εσκεμμένα δημιουργήθηκε χωρίς κάποιον περιορισμό έτσι ώστε να μπορεί να χρησιμοποιηθεί για το συγκεκριμένο παράδειγμα. Προφανώς σε αντίθεση με το παράδειγμα που δόθηκε προηγμένως, δεν υπάρχει μεγάλη σημασία στην εκμετάλλευση της συγκεκριμένης μεθόδου διότι δεν επηρεάζει την εφαρμογή άμεσα «γράφοντας» στο δίκτυο ή εάν υπήρχε κάποια αποθηκευμένη πληροφορία την οποία θα έπρεπε να γνωρίζουν συγκεκριμένοι χρήστες. Ωστόσο από την στιγμή που η κατανεμημένη εφαρμογή δεν δίνει αυτήν την δυνατότητα σε έναν απλό χρήστη αν κάποιος εκμεταλλευτεί αυτό το «κενό ασφαλείας» κερδίζει μια μορφή πλεονεκτήματος.

3.3.1 Ανάπτυξη έξυπνου συμβολαίου

Έστω ότι ο χρήστης Cli-1, που έχει πρόσβαση στο δίκτυο, ανακτά την διεύθυνση του έξυπνου συμβολαίου που δημιούργησε και αποθήκευσε στο δίκτυο ο Cli-0. Τότε ο Cli-1 δημιουργεί ένα έξυπνο συμβόλαιο που επικοινωνεί με το έξυπνο συμβόλαιο του Cli-0.

```
pragma solidity >=0.4.0<0.6.0;

contract CardMarketInterface{

functiongetTotalCards() public view returns(uint);
functiongetCard(uint _cardId) public view returns(uint,uint16,uint16, uint16, uint16, uint);
}

contract NewCardMarket is CardMarketInterface{

  address admin;

  modifier onlyAdmin(address _owner){
    require(_owner == admin);
    _;
```

```
  }

  constructor() public {
    admin = msg.sender;
  }

CardMarketInterface CMI = CardMarketInterface(0x8f11b9d35CC5E930E721CA207F5927D0304E86B7);

functiongetTotalCards() public view onlyAdmin(msg.sender) returns(uint){
return(CMI.getTotalCards());
  }


functiongetCard(uint _cardId) public view onlyAdmin(msg.sender) returns(uint,uint16,uint16, uint16, uint16,
uint){
uintcardDna;
      uint16 cardShape;
      uint16 cardColor;
      uint16 cardUses;
      uint16 cardLimit;
uintcardId;
      (cardDna, cardShape, cardColor, cardUses, cardLimit, cardId) = CMI.getCard(_cardId);
return (cardDna, cardShape, cardColor, cardUses, cardLimit, cardId);

  }
}
```

Στην αρχή το έξυπνου συμβολαίου δημιουργείται ένα interface το οποίο προσομοιάζει το έξυπνο συμβόλαιο που αναπτύχθηκε προηγμένως και συγκεκριμένα τις μεθόδους που θα χρησιμοποιηθούν. Οι δύο μέθοδοι που χρησιμοποιούνται είναι η getTotalCards και η getCard. Η πρώτη μέθοδος επιστρέφει τον συνολικό αριθμό των τεκμηρίων που βρίσκονται αποθηκευμένα στο έξυπνο συμβόλαιο και η δεύτερη επιστρέφει τα χαρακτηριστικά τους. Παρατηρείται ότι και οι δύο μέθοδοι είναι δημόσιες και στον κώδικα στο κεφάλαιο 4.1 φαίνεται ότι δεν υπάρχει κάποια μορφή περιορισμού.

```
contract CardMarketInterface{

functiongetTotalCards() public view returns(uint);
functiongetCard(uint _cardId) public view returns(uint,uint16,uint16, uint16, uint16, uint);
}
```

Έπειτα δημιουργείται μία αναφορά του interface και της προσδίδεται η διεύθυνση του έξυπνου συμβολαίου με το οποίο θα υπάρξει επικοινωνία.

```
CardMarketInterface CMI = CardMarketInterface(0x8f11b9d35CC5E930E721CA207F5927D0304E86B7);
```

Στη συνέχεια δημιουργείται ένας modifier που περιορίζει τις μεθόδους του έξυπνου συμβολαίου να μπορούν να χρησιμοποιηθούν μόνο από αυτόν που αποθήκευσε το έξυπνο στο δίκτυο, δηλαδή τον Cli-1. Ταυτόχρονα αναπτύσσεται και ένας constructor που δηλώνει ως admin αυτόν που έκανε την αποθήκευση στο δίκτυο.

```
address admin;

  modifier onlyAdmin(address _owner){
    require(_owner == admin);
    _;
  }
  constructor() public {
    admin = msg.sender;
  }
```

Τέλος δημιουργούνται οι μέθοδοι που επικοινωνούν με προηγούμενο έξυπνο συμβόλαιο και προσομοιάζουν τα δεδομένα που επιστρέφουν.

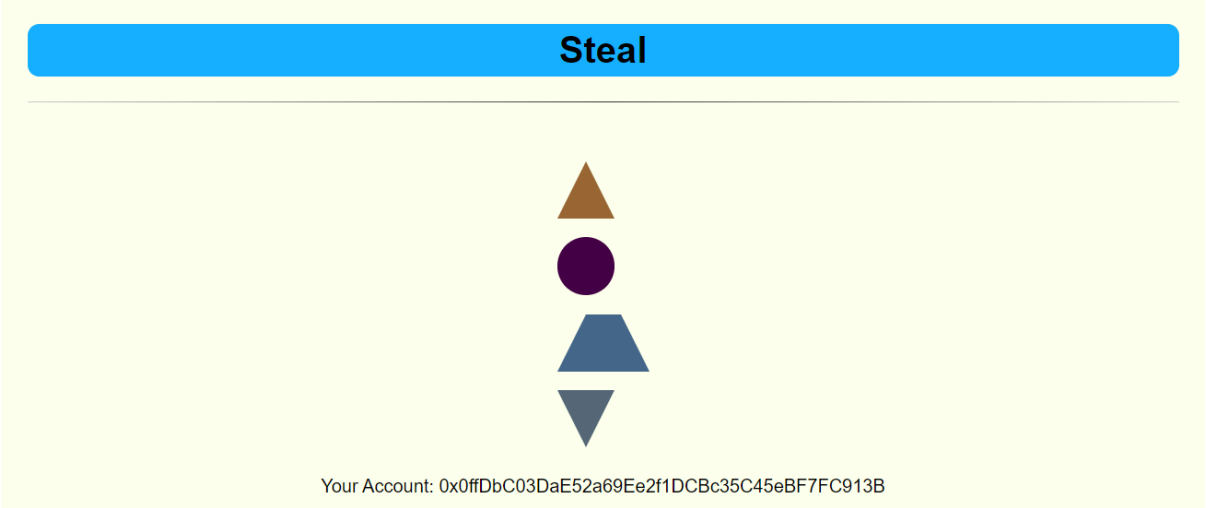
```
functiongetTotalCards() public view onlyAdmin(msg.sender) returns(uint){
return(CMI.getTotalCards());
  }

functiongetCard(uint _cardId) public view onlyAdmin(msg.sender) returns(uint,uint16,uint16, uint16, uint16,
uint){
uintcardDna;
     uint16 cardShape;
     uint16 cardColor;
     uint16 cardUses;
     uint16 cardLimit;
uintcardId;
     (cardDna, cardShape, cardColor, cardUses, cardLimit, cardId) = CMI.getCard(_cardId);
return (cardDna, cardShape, cardColor, cardUses, cardLimit, cardId);
  }
```

3.3.2 Ανάπτυξη εφαρμογής

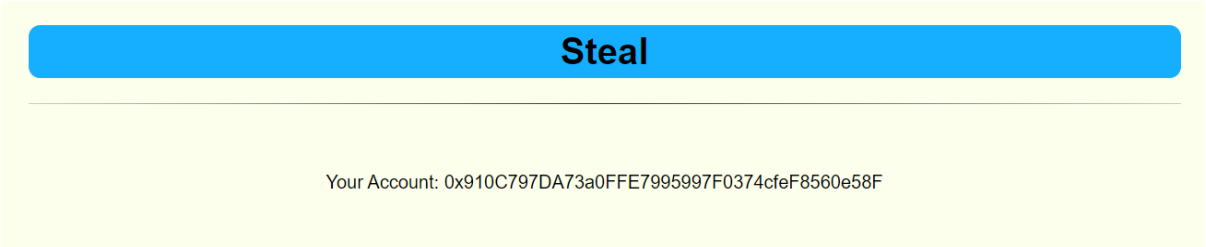
Για την ανάπτυξη της εφαρμογής ακολουθείται η ίδια διαδικασία που ακολουθήθηκε στο κεφάλαιο 4.2 με την διαφορά ότι στο truffle-config.js δηλώνεται η παράμετρος from που έχει να κάνει με το ποιος λογαριασμός αποθηκεύει την εφαρμογή στο δίκτυο. Ο λόγος που συμβαίνει αυτό είναι διότι στο Ganache-Cli χρησιμοποιείται από προεπιλογή ο πρώτος λογαριασμός για την αποθήκευση των έξυπνων συμβολαίων στο δίκτυο, δηλαδή ο Cli-0. Άρα για να κάνει την αποθήκευση ο Cli-1 ισχύει το παρακάτω truffle-config.js αρχείο.

```
module.exports = {

  networks: {
    development: {
      host: "127.0.0.1",
      port: 8545,
      from: "0x0ffDbC03DaE52a69Ee2f1DCBc35C45eBF7FC913B",
network_id: "*"
    },
  compilers: {
solc: {
      version: '0.4.25',
      optimizer: {
        enabled: true,
        runs: 200
      }
    }
  }
};
```

Στην εικόνα 3.19 φαίνεται πως ο χρήστης Cli-1 χρησιμοποιεί την εφαρμογή ενώ στην εικόνα 3.20 φαίνεται πώς ένας άλλος λογαριασμός χρησιμοποιεί την εφαρμογή.



Εικόνα 3.19 Εφαρμογή του Cli-1 όπως φαίνεται και από τον λογαριασμό στο κάτω μέρος της εικόνας. Μία παρατήρηση είναι πως τα τεκμήρια που εμφανίζονται είναι λιγότερα από αυτά που εμφανίζονταν στην προηγούμενη εφαρμογή διότι το στιγμιότυπο πάρθηκε σε προηγούμενο χρόνο.



Εικόνα 3.20 Ο χρήστης Cli-2 χρησιμοποιεί την εφαρμογή και εφόσον δεν έχει ορισθεί ως admin δεν μπορεί να δει τα τεκμήρια.

Γ. Συμπεράσματα

Το blockchain αρχικά χαρακτηρίστηκε ως μία αποκεντρωμένη τεχνολογία για ψηφιακές συναλλαγές. Με την ανάπτυξη του ethereum το blockchain έχει εξελιχθεί σε μια πλατφόρμα που μπορεί να χρησιμοποιηθεί για την υλοποίηση ενός τεράστιου εύρους εφαρμογών. Η συγκεκριμένη διπλωματική δημιουργήθηκε με σκοπό να αναδείξει τις δυνατότητες των έξυπνων συμβολαίων καθώς και τους περιορισμούς τους. Όπως φάνηκε κατά την υλοποίηση των εφαρμογών τα έξυπνα συμβόλαια πρέπει να αναπτύσσονται με ιδιαίτερη προσοχή διότι από τη στιγμή που αποθηκευτούν στο ethereum δίκτυο δεν μπορούν να διαγραφούν ή να αλλάξουν. Ένα λάθος ή μία παράβλεψη σε μια λειτουργία ενός έξυπνου συμβολαίου μπορεί να οδηγήσει σε προβλήματα διαφόρων μεγεθών από μια διαρροή δεδομένων μέχρι και απώλειες χρηματικών ποσών. Ωστόσο με τη σωστή δόμηση ενός συμβολαίου μπορούν να δημιουργηθούν εφαρμογές που να αλλάξουν την οικονομία, το εμπόριο, τις συναλλαγές, τη βιομηχανία, τη δικαιοσύνη και γενικά τον τρόπο ζωής των ανθρώπων επειδή προσφέρουν ασφάλεια, σιγουριά και απολυτότητα, ιδιότητες που μέχρι σήμερα ήταν δύσκολο να υπάρξουν στο μέγιστο βαθμό τους και σχεδόν πάντα υπήρχε μεσολάβηση τρίτων ατόμων ή εταιριών όπου οποιαδήποτε στιγμή μια λανθασμένη ή κακόβουλη απόφαση θα μπορούσε να αποβεί μοιραία. Το έξυπνο συμβόλαιο που αναπτύχθηκε κατά την εκπόνηση αυτής της διπλωματικής εργασίας αποτελεί μια βάση για την δημιουργία εφαρμογών που χρησιμοποιούν τεκμήρια. Τα τεκμήρια μπορούν να είναι μοναδικά, δηλαδή να αντιπροσωπεύουν κάτι ξεχωριστό ή μπορούν να είναι όμοια και να χρησιμοποιούνται ως κρυπτοοικονομία. Σε κάθε περίπτωση το βασικό κριτήριο για την δημιουργία τους ήταν η «ιδιότητα της ιδιοκτησίας», η οποία μπορεί να αποδοθεί λόγω της τεχνολογίας του blockchain σε ψηφιακό επίπεδο. Επειδή τα τεκμήρια ακολουθούν συγκεκριμένα standards μπορούν να χρησιμοποιηθούν σε άλλα είδη εφαρμογών ως βάση για την ανάπτυξη των έξυπνων συμβολαίων τους. Για την συγκεκριμένη εφαρμογή υπάρχουν τεράστια περιθώρια βελτίωσης όπως δημιουργία καλύτερης σελίδας διεπαφής χρήστη, ολοκλήρωση της σελίδας admin όπου στο έξυπνο συμβόλαιο υπάρχει η αντίστοιχη λειτουργικότητα αλλά δεν χρησιμοποιήθηκε στην σελίδα και μεταφορά εφαρμογής στο κύριο ethereum δίκτυο.

Δ. Βιβλιογραφικές αναφορές

[1] Hooda, P. (2019, Δεκέμβριος 12). *practical Byzantine Fault Tolerance(pBFT)*. Retrieved Μάιος 2020, from geeksforgeeks.org: https://www.geeksforgeeks.org/practical-byzantine-fault-tolerancepbft/

[2] Konstantopoulos, G. (2017, Δεκέμβριος 1). *Understanding Blockchain Fundamentals, Part 1: Byzantine Fault Tolerance*. Retrieved Μάιος 2020, from medium.com: https://medium.com/loom-network/understanding-blockchain-fundamentals-part-1-byzantine-fault-tolerance-245f46fe8419

[3] Eduardo A. P. Alchieri, A. N. (n.d.). *Byzantine Consensus with Unknown Participants.* Retrieved Μάιος 2020, from https://link.springer.com/chapter/10.1007/978-3-540-92221-6_4

[4] Rouse, M. (n.d.). *asymmetric cryptography (public key cryptography).* Retrieved Μάιος 2020, from SearchSecurity: https://searchsecurity.techtarget.com/definition/asymmetric-cryptography

[5] Selvamanikkam, M. (2018, Φεβρουάριος 1). *Digital Signature Generation.* Retrieved Μάιος 2020, from medium.com: https://medium.com/@meruja/digital-signature-generation-75cc63b7e1b4

[6] Tutorials points. (n.d.). *Cryptography Hash functions.* Retrieved Μάιος 2020, from www.tutorialspoint.com: https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm

[7] Preneel, B. (2003, Φεβρουάριος). *Analysis and Design ofCryptographic Hash Functions.* Retrieved Μάιος 2020, from http://www.e-reading-lib.com/bookreader.php/141503/Preneel_-_Analysis_and_Design_of_Cryptographic_Hash_Functions.pdf

[8] *Introdu*(Schollmeier, 2001)*ction to Merkle Tree.* (2018, Δεκέμβριος 6). Retrieved Μάιος 2020, from www.geeksforgeeks.org:https://www.geeksforgeeks.org/introduction-to-merkle-tree/

[9] Schollmeier, R. (2001, Σεπτεμβρίος). *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications.* Retrieved Μάιος 2020, from www.researchgate.net/profile/Ruediger_Schollmeier/publication/3940901_A_Definition_ohttps://www.researchgate.net/publication/3940901_A_Definition_of_Peer-to-Peer_Networking_for_the_Classification_of_Peer-to-Peer_Architectures_and_Applications

[10] Wattana Viriyasitavat, D. H. (2019, Μάρτιος). *Blockchain Characteristicsand Consensus in Modern Business Processes.* Retrieved Μάιος 2020, from www.researchgate.net: https://www.researchgate.net/profile/Wattana_Viriyasitavat/publication/326680277_Blockchain_Characteristics_and_Consensus_in_Modern_Business_Processes/links/5ee9c67a299bf1faac5c8a37/Blockchain-Characteristics-and-Consensus-in-Modern-Business-Processes.pdf

[11] Rosic, A. (n.d.). *What is Blockchain Technology? A Step-by-Step Guide For Beginners.* Retrieved Μάιος 2020, from blockgeeks.com: https://blockgeeks.com/guides/what-is-blockchain-technology/

[12] Gupta, A. (2019, Ιανουάριος 7). *Introduction to Blockchain technology.* Retrieved Μάιος 2020, from www.geeksforgeeks.org:

[13] https://www.geeksforgeeks.org/blockchain-technology-introduction/

[14] joshi, t. (2020, Φεβρουάριος 5). *Features of Blockchain.* Retrieved Μάιος 2020, from geeksforgeeks.org: https://www.geeksforgeeks.org/features-of-blockchain/

[15] IREDALE, G. (2020, Μάιος 24). *6 Key Blockch ain Features You Need to Know Now.* Retrieved Μάιος 2020, from 101blockchains.com: https://101blockchains.com/introduction-to-blockchain-features/

[16] Kangas, E. (2017, Νοέμβριος 10). *UNDERSTANDING BLOCKCHAINS (AND BITCOIN) – PART 2: TECHNOLOGY.* Retrieved Μάιος 2020, from luxsci.com: https://luxsci.com/blog/understanding-blockchains-and-bitcoin-technology.html

[17] *Genesis block.* (2020, Ιούλιος 13). Retrieved Μάιος 2020, from en.bitcoin.it: https://en.bitcoin.it/wiki/Genesis_block

[18] Huskanović, A. (n.d.). *Params in Ethereum Genesis Block Explained.* Retrieved Μάιος 2020, from asynclabs.co: https://www.asynclabs.co/blog/params-in-ethereum-genesis-block-explained/

[19] *Consensus Algorithms in Blockchain.* (2020, 4 13). Retrieved Ιούλιος 2020, from geeksforgeeks.org: https://www.geeksforgeeks.org/consensus-algorithms-in-blockchain/

[20] Hooda, P. (2019, Ιανουάριος 9). *Proof of Work (PoW) Consensus.* Retrieved Ιούλιος 2020, from geeksforgeeks.org/: https://www.geeksforgeeks.org/proof-of-work-pow-consensus/

[21] *Bitcoin*. (2018, Σεπτέμβριος 19). Retrieved Ιούλιος 2020, from bitcoin.it: https://en.bitcoin.it/wiki/Bitcoin

[22] Nakamoto, S. (2008, Οκτώβριος 31). *Bitcoin: A Peer-to-Peer Electronic Cash System.* Retrieved Ιούλιος 2020, from bitcoin.org: https://bitcoin.org/bitcoin.pdf

[23] Rosic, A. (2019). *What is Ethereum? [The Most Updated Step-by-Step-Guide!].* Retrieved Ιούλιος 2020, from blockgeeks.com: https://blockgeeks.com/guides/ethereum/

[24] Buterin, V. (2020, Ιούλιος 9). *Ethereum Whitepaper*. Retrieved Ιούλιος 2020, from ethereum.org: https://ethereum.org/en/whitepaper/

[25] *What is DAO.* (n.d.). Retrieved Ιούλιος 20202, from cointelegraph.com: https://cointelegraph.com/ethereum-for-beginners/what-is-dao

[26] Fabian Vogelsteller, V. B. (2015, Νοέμβριος). *EIP-20: ERC-20 Token Standard.* Retrieved Απρίλιος 2019, from eips.ethereum.org: https://eips.ethereum.org/EIPS/eip-20

[27] *ERC-721.* (n.d.). Retrieved Μάιος 2019, from http://erc721.or g/: http://erc721.org/

[28] https://berty.tech/blog/decentralized-distributed-centralized