ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Know Your Customer Σύστημα με IPFS διεπαφή σε Quorum Permissioned Blockchain

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Γ. Μπαξόπουλος

Επιβλέπουσα: Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2020

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Know Your Customer Σύστημα με IPFS διεπαφή σε Quorum Permissioned Blockchain

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Γ. Μπαξόπουλος

Επιβλέπουσα: Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 28η Φεβρουαρίου 2020

.....	.....	.....
Θεοδώρα Βαρβαρίγου Καθηγήτρια Ε.Μ.Π.	Δημήτριος Ασκούνης Καθηγητής Ε.Μ.Π.	Συμεών Παπαβασιλείου Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2020

.....

Γεώργιος Γ. Μπαξόπουλος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Το τέλος του 20ού αιώνα χαρακτηρίστηκε από ραγδαία ανάπτυξη της τεχνολογίας, με σημαντικότερη ίσως εξέλιξη την εμφάνιση του Διαδικτύου. Με γρήγορους ρυθμούς έγινε κομμάτι της καθημερινότητας ολοένα και μεγαλύτερου πληθυσμού, προσφέροντας πρόσβαση σε ασύλληπτο όγκο γνώσης και πληροφοριών, δυνατότητες άμεσης επικοινωνίας και υπηρεσίες που έκαναν ευκολότερη την καθημερινή ζωή. Οι υπηρεσίες αυτές, μέχρι σήμερα βασίζονται κυρίως σε μια κεντροποιημένη λογική. Οι χρήστες βασίζονται σε τεχνολογίες που ακολουθούν την αρχιτεκτονική του πελάτη-εξυπηρετητή, μια αρχιτεκτονική που απαιτεί εμπιστοσύνη από την πλευρά του χρήστη προς τον εξυπηρετητή και έχει αποδειχτεί ευάλωτη σε κακόβουλες επιθέσεις.

Το Blockchain ήρθε με σκοπό να καλύψει αυτή την ανάγκη. Ξεκινώντας το 2009 ως βάση για το πρώτο «κρυπτονόμισμα», το Bitcoin, υποσχέθηκε μια νέα εποχή στις συναλλαγές μεταξύ χρηστών, με ένα επαναστατικό μοντέλο επικύρωσης συναλλαγών. Το Blockchain έχει εξ' ορισμού αποκεντρωμένη λειτουργία, αφού αποτελεί ένα ψηφιακό, κατανεμημένο, δημόσιο καθολικό (ledger), οι εγγραφές του οποίου είναι αμετάβλητες, αρχή που επιβεβαιώνεται από ένα δίκτυο ομότιμων κόμβων. Χάρη στα πλεονεκτήματα που προσφέρει αυτή η δομή, το Blockchain επεκτάθηκε πέρα από το Bitcoin. Η υλοποίηση του Ethereum Blockchain, συγκεκριμένα, έδωσε την ευκαιρία να χρησιμοποιηθεί ως πλατφόρμα για αποκεντρωμένες εφαρμογές (Decentralized Apps), που δε βασίζονται ούτε στο ελάχιστο στο κλασικό μοντέλο πελάτη-εξυπηρετητή.

Βασισμένο στο Ethereum, και χρησιμοποιώντας το Ethereum Virtual Machine για την εκτέλεση κώδικα που αποθηκεύεται σε μορφή έξυπνων συμβολαίων στο Blockchain, έχει κάνει την εμφάνισή του το Quorum Blockchain. Αποτελεί επέκταση του Ethereum που απευθύνεται περισσότερο σε επιχειρήσεις και προσφέρει δυνατότητες ιδιωτικότητας συναλλαγών και πιο δομημένη οργάνωση ενός ιδιωτικού Blockchain. Καλύπτει έτσι μια βασική έλλειψη όλων των υπαρχόντων σχετικών τεχνολογιών, υποστηρίζοντας βέβαια και τη λειτουργία αποκεντρωμένων εφαρμογών.

Μια τέτοια εφαρμογή είναι το αντικείμενο της παρούσας διπλωματικής. Μετά από μελέτη των εργαλείων που προσφέρονται από την τεχνολογική στοίβα του Quorum και του Ethereum Blockchain, δημιουργήθηκε μια αποκεντρωμένη εφαρμογή, το Know Your Customer. Πρόκειται για μια εφαρμογή που τρέχει στο Quorum μέσω ενός έξυπνου συμβολαίου και συνδέεται με το IPFS, μια τεχνολογία για αποκεντρωμένη αποθήκευση δεδομένων.

Για την ολοκλήρωσή της έγινε έρευνα σε σχετικές τεχνολογίες, το θεωρητικό υπόβαθρο των οποίων παρουσιάζεται στην αρχή της εργασίας. Στη συνέχεια αναλύονται τα εργαλεία που χρησιμοποιήθηκαν και η διαδικασία ανάπτυξης της εφαρμογής, οι δυσκολίες που παρουσιάστηκαν στην πορεία, και τέλος αναλύεται η λειτουργία της ίδιας της εφαρμογής.

Λέξεις κλειδιά

Quorum, Blockchain, Ethereum, Έξυπνα συμβόλαια, IPFS, Αποκεντρωμένες εφαρμογές

Abstract

The end of the 20th century was marked by rapid technological developments, of which the emergence of the Internet was probably the most important. It has quickly become part of the daily life of an increasing number of people, providing access to an unrivaled amount of knowledge and information, instant communication capabilities and services that have made daily life easier. To date, these services are mostly based on a centralized logic. Users rely on technologies that follow the client-server architecture, an architecture that requires trust from the server-side and is vulnerable to malicious attacks.

Blockchain made its appearance aiming to solve the aforementioned problem. Created in 2009 as the supporting techonology of the first “cryptocurrency”, Bitcoin, it paved the way for a new era for the transactions among users, introducing a revolutionary transaction validation model. Blockchain works by default in a completely decentralized manner, as it is a digital, distributed, public ledger whose records are unchanged, a principle constantly confirmed by a peer-to-peer network. Thanks to the advantages of this structure, Blockchain has expanded beyond Bitcoin. The implementation of Ethereum Blockchain, in particular, provided the opportunity to be used as a platform for Decentralized Apps, which is not at all based on the classic client-server model.

Based on Ethereum, and using the Ethereum Virtual Machine to execute code stored in smart contract form in the Blockchain, Quorum Blockchain emerged. It is an extension of Ethereum that mainly caters to enterprise needs and offers transaction privacy capabilities and a more structured organization of a private Blockchain. By supporting transaction privacy it covers a fundamental shortcoming of all existing relevant technologies, while also supporting decentralized applications.

Such an application is the subject of this diploma thesis. After studying the tools offered by the Quorum and Ethereum Blockchain technology stack, a decentralized application, Know Your Customer, was developed. It is an application running on Quorum through a smart contract and linked to IPFS, a technology for decentralized data storage.

For its completion, research was conducted on related technologies, the theoretical background of which is presented at the beginning of the work. Subsequently, this thesis presents the used tools, the application development process, the difficulties encountered along the way are analyzed, and finally the operation of the application itself.

Key words

Quorum, Blockchain, Ethereum, Smart contracts, IPFS, Decentralized Applications

Ευχαριστίες

Ευχαριστώ όλους όσους με βοήθησαν σε ό,τι έχω καταφέρει μέχρι τώρα, με στήριξαν και διαρκώς μου έδιναν ώθηση για το κάτι παραπάνω: την οικογένεια μου, τους φίλους μου, και ανθρώπους από το στενό μου κύκλο χωρίς τους οποίους δε θα έφτανα μέχρι εδώ.

Θα ήθελα επίσης να ευχαριστήσω την κα. Βαρβαρίγου για την ενδιαφέρουσα διπλωματική που μου ανέθεσε, τον υποψήφιο διδάκτορα Γεώργιο Παλαιοκρασσά και τον δόκτορα Αντώνη Λίτκε για την καθοδήγηση και τη βοήθεια που μου προσέφεραν, και όλους όσους συντελούν στην ομαλή λειτουργία του ιδρύματος.

Γεώργιος Γ. Μπαξόπουλος
Αθήνα, Φεβρουάριος 2020

Περιεχόμενα

Κατάλογος Σχημάτων

1

Εισαγωγή

Η ανάπτυξη του Διαδικτύου έχει φέρει μια νέα εποχή στις καθημερινές ζωές όλων. Η ευκολία που προσφέρει στην επικοινωνία, στην πρόσβαση σε γιγαντιαίο όγκο πληροφοριών, και οι υπηρεσίες που κάνουν την καθημερινότητα γρηγορότερη το έχουν καταστήσει απαραίτητο εργαλείο. Ο κόσμος χρησιμοποιεί το Διαδίκτυο για να επικοινωνήσει, να προμηθευτεί προϊόντα και αγαθά, να ψυχαγωγηθεί, ακόμα και να κάνει τραπεζικές συναλλαγές. Όσο και αν το χρήμα γίνεται ηλεκτρονικό, όμως, ακόμα χρειάζεται εμπιστοσύνη στο τραπεζικό σύστημα.

Το Bitcoin, ωστόσο, εισήγαγε ένα νέο τρόπο για να σκεφτεί κανείς το χρήμα. Γνώρισε στον κόσμο το πρώτο κρυπτονόμισμα· ένα νόμισμα η δημιουργία και η διακίνηση του οποίου δεν ελέγχεται από κανέναν οργανισμό ή τράπεζα. Η αξία του ελέγχεται μόνο από το κρυπτογραφικό πρωτόκολλο (και κατ' επέκταση την ενέργεια που δαπανάται) για τη δημιουργία καινούριων νομισμάτων ή την επικύρωση συναλλαγών. Το Bitcoin, καθώς και τα κρυπτονομίσματα που δημιουργήθηκαν μετά από αυτό, βασίζονται όλα στην τεχνολογία του Blockchain. Το Blockchain, με την έλευση του Ethereum, αποτέλεσε πλατφόρμα για τη δημιουργία και την εκτέλεση αποκεντρωμένων εφαρμογών (Decentralized Apps).

Ξεπερνώντας τα όρια του εργαλείου για την ύπαρξη κρυπτονομισμάτων, το Blockchain ως δομή δεδομένων φάνηκε χρήσιμη σε εταιρίες, οι οποίες έστρεψαν την προσοχή τους σε αυτό για τις δικές τους ανάγκες. Έτσι δημιουργήθηκε το Quorum, μια Blockchain λύση, επέκταση του Ethereum, που ήρθε για να λύσει το πρόβλημα της ιδιωτικότητας των συναλλαγών σε ένα Blockchain δίκτυο.

1.1 Αντικείμενο της Διπλωματικής Εργασίας

Αντικείμενο της διπλωματικής αυτής αποτελεί η δημιουργία μιας εφαρμογής που θα τρέχει στο προαναφερθέν δίκτυο και θα ελέγχει τη σύνδεση ενός χρήστη σε ένα ιδιωτικό δίκτυο, στο οποίο τρέχει ένα Quorum Blockchain. Η εφαρμογή αυτή δε θα βασίζεται καθόλου στη λογική του πελάτη-εξυπηρετητή, αλλά θα «ζει» εξ' ολοκλήρου στο blockchain. Ο κώδικας που αφορά το blockchain θα αποθηκεύεται σε ένα έξυπνο συμβόλαιο (smart contract), θα μεταφράζεται σε bytecode και θα εκτελείται στο Ethereum Virtual Machine.

Για την ανάγκη αυτή χρειάζεται η συνεργασία ορισμένων διαφορετικών τεχνολογιών. Χρειάζεται αρχικά να υλοποιηθεί ο κώδικας του έξυπνου συμβολαίου, η «ρα-

χοκοκαλιά» δηλαδή της εφαρμογής, ο κώδικας που διαχειρίζεται ό,τι έχει σχέση με το blockchain. Στη συνέχεια πρέπει να γίνει το setup ενός ιδιωτικού IPFS δικτύου, στο οποίο θα αποθηκεύεται το έγγραφο πιστοποίησης που θα υποβάλλει κάθε χρήστης της εφαρμογής. Τέλος, πρέπει να σχεδιαστεί το UI, μια φιλική γραφική διεπαφή ανάμεσα στο χρήστη και το blockchain.

1.2 Οργάνωση κειμένου

Το κείμενο της διπλωματικής αποτελείται από 8 Κεφάλαια, το τελευταίο εκ των οποίων είναι η Βιβλιογραφία, και 2 Παραρτήματα. Το πρώτο κεφάλαιο είναι εισαγωγικό.

Στο δεύτερο παρουσιάζεται το θεωρητικό υπόβαθρο της εργασίας, γίνεται δηλαδή παρουσίαση των τεχνολογιών που σχετίζονται με αυτήν. Στην αρχή γίνεται αναφορά στο Web 3.0, τη νέα μορφή δηλαδή του Διαδικτύου στην οποία έχει βρει άνθιση το Blockchain. Ύστερα παρουσιάζονται οι τεχνολογικοί θεμέλιοι λίθοι του Blockchain, το Ethereum και ο τρόπος λειτουργίας του αναλυτικά, και στη συνέχεια το Quorum, η μελέτη του οποίου ήταν βασικό κομμάτι της παρούσας διπλωματικής. Στο τέλος του γίνεται αναφορά και σε σχετικές εργασίες που αποτέλεσαν πηγή έμπνευσης ή/και χρησιμοποιήθηκαν ως εργαλεία για την ανάπτυξη και το testing.

Στο Κεφάλαιο 3 παρουσιάζεται το τεχνικό υπόβαθρο της διπλωματικής, εργαλεία και τεχνολογίες δηλαδή που τη συνέθεσαν στην τελική της μορφή. Συγκεκριμένα αναλύονται όλα τα κομμάτια της σουίτας του Truffle για ανάπτυξη αποκεντρωμένων εφαρμογών στο Ethereum blockchain και, κατ' επέκταση στο Quorum, το IPFS, το Docker και τα στοιχεία από το JavaScript (Node) οικοσύστημα που χρησιμοποιήθηκαν για το web κομμάτι.

Στο τέταρτο κεφάλαιο γίνεται ανάλυση των λειτουργικών και των μη λειτουργικών απαιτήσεων του συστήματος, των κατηγοριών των χρηστών, και των παραδοχών που έγιναν στην πορεία της εκπόνησης.

Στο πέμπτο κεφάλαιο γίνεται εξονυχιστική παρουσίαση του έξυπνου συμβολαίου πάνω στο οποίο βασίζεται η εφαρμογή. Εξηγείται η λειτουργία κάθε μεθόδου του κώδικά του και παρουσιάζονται οι αποφάσεις που πάρθηκαν για τη δομή του συμβολαίου. Γίνεται επίσης ανάλυση της επικοινωνίας του συμβολαίου με το web κομμάτι της εφαρμογής.

Στο έκτο κεφάλαιο ακολουθεί η επίδειξη της λειτουργικότητας της εφαρμογής, με λεκτική περιγραφή της ροής λειτουργίας και εικόνες για τη γραφική διεπαφή της με το χρήστη.

Στον επίλογο που αποτελεί το έβδομο κεφάλαιο παρατίθενται τα συμπεράσματα που πάρθηκαν κατά τη διαδικασία μελέτης των διαφόρων τεχνολογιών και την ανά-

πτυξη της εφαρμογής, καθώς και προτάσεις υλοποιήσεις που θα μπορούσαν να γίνουν για επέκταση της λειτουργικότητας.

Μετά τη Βιβλιογραφία (Κεφάλαιο 8) ακολουθούν τα Παραρτήματα, το πρώτο εκ των οποίων περιέχει οδηγίες για την εγκατάσταση και τη χρήση της εφαρμογής. Στο δεύτερο Παράρτημα, τέλος, υπάρχουν οι κώδικες της εφαρμογής.

2

Θεωρητικό υπόβαθρο και σχετικές εργασίες

Στο ακόλουθο κεφάλαιο θα διατυπωθεί το θεωρητικό υπόβαθρο, η μελέτη του οποίου προηγήθηκε της υλοποίησης αυτής της εργασίας. Αρχικά γίνεται παρουσίαση της νέας μορφής που έχει πάρει το Διαδίκτυο, λόγω της τάσης για «αποκεντροποίηση» των προσφερόμενων υπηρεσιών. Στη συνέχεια αναλύονται θεωρητικά στοιχεία που συνθέτουν την τεχνολογία του Blockchain, και τέλος παρατίθενται στοιχεία για το Ethereum Blockchain, το οποίο και χρησιμοποιήθηκε για την ανάπτυξη της παρούσας διπλωματικής και αποτελεί βάση του Quorum Blockchain.

2.1 Web 3.0

Ο παγκόσμιος ιστός από την ευρεία εξάπλωσή του στα τέλη του 20^{ού} αιώνα έχει περάσει από πολλές αλλαγές, συγκεκριμένες όμως από αυτές ήταν τόσο σημαντικές, που θεωρήθηκαν αρκετά σημαντικές ώστε να τμηματοποιήσουν την πορεία της ιστορίας του σε τρία βασικά στάδια.

- **Web 1.0**

Ο όρος Web 1.0 αναφέρεται στην πρώτη φάση της ύπαρξης του Παγκόσμιου Ιστού. Παρότι δεν υπάρχει και πιθανώς δε θα υπάρξει σαφής ορισμός, οι περισσότεροι συμφωνούν ότι ο όρος χαρακτηρίζει το Διαδίκτυο, όταν οι διαθέσιμες ιστοσελίδες ήταν στην πλειοψηφία τους στατικές ως προς το περιεχόμενό τους.

Εικόνα 2-1. Η αρχική σελίδα του Google το 1996

- **Web 2.0**

Ο όρος Web 2.0 έχει αρχίσει να χρησιμοποιείται περίπου από το 2004 για να περιγράψει το "social web", το Διαδίκτυο δηλαδή που τότε άρχισε να αποκτά χαρακτήρα κοινωνικής δικτύωσης. Ο νέος τίτλος δε χρησιμοποιήθηκε για να αναφερθεί σε κάποια ριζική αλλαγή στις τεχνολογίες που χρησιμοποιούνται για τη δημιουργία και τη χρήση ιστοσελίδων, αλλά για να αναφερθεί σε μια αλλαγή στο σχεδιασμό των σελίδων και στον τρόπο με τον οποίο αυτές χρησιμοποιούνταν. Κατά τη διάρκεια της ανόδου των κοινωνικών μέσων, το Web 2.0 έγινε ένα ουσιαστικό εργαλείο για κάθε άτομο με τους περισσότερους χρήστες του διαδικτύου να αρχίζουν να αλληλεπιδρούν μεταξύ τους και να συνεργάζονται σε εικονικές κοινότητες. Αυτό οδήγησε στη δημοτικότητα των δικτυακών τόπων που τόνισαν το περιεχόμενο που δημιουργήθηκε από τους χρήστες (user-generated content) μέσω των κοινωνικών μέσων όπως το Facebook και το Twitter ή οι ιστοσελίδες crowdsourcing όπως η Wikipedia και το Trip Advisor. Τώρα, λόγω αυτής της δέσμευσης του χρήστη, ο ιστός έχει γίνει πιο ανοικτός, πιο συνδεδεμένος και πιο έξυπνος, οδηγώντας μας στην τρέχουσα κατάσταση, το Web 3.0.

Εικόνα 2-2 Βασικές υπηρεσίες του Web 2.0

- **Web 3.0**

Το Web 3.0 είναι ένας όρος που δημιουργήθηκε το 2004 από τον John Markoff, δημοσιογράφο της New York Times. Ένας ακόμα χαρακτηρισμός του Διαδικτύου σήμερα είναι ο «Σημασιολογικός Ιστός». Το Web 3.0 περιγράφει την τρίτη γενιά του Διαδικτύου, την οποία βιώνουμε σήμερα στα πρώτα στάδια του. Πρόκειται για το "έ-ξυπνο δίκτυο" και έχει χαρακτηριστεί ως μία από τις μεγαλύτερες επαναστάσεις της εποχής μας.

Το Web 3.0 είναι μια επέκταση του Web 2.0. Τονίζει μια «κατανόηση των πλη-ροφοριών από το μηχάνημα ώστε να παρέχει μια πιο παραγωγική και διαισθητική ε-μπειρία χρήστη». Έχει δημιουργήσει μια προσαρμόσιμη εμπειρία ιστού με έξυπνες διαφημίσεις αναζήτησης και συμπεριφοράς, όπου το περιεχόμενο δημιουργείται από μηχανές αντί για ανθρώπους. Για παράδειγμα, αντί για έναν επαγγελματία που χρησι-μοποιεί έρευνα για να καταλάβει σε ποια άτομα θα απευθυνθεί η εκάστοτε διαφήμιση ή περιεχόμενο, το Web 3.0 χρησιμοποιεί μηχανές και αλγόριθμους για να στοχεύσει συγκεκριμένα άτομα με βάση το ιστορικό ιστού και τα δεδομένα χρήστη.

Μπορεί ο όρος να ακούγεται ξένος, όμως οι περισσότεροι άνθρωποι συνεργάζο-νται καθημερινά με το Web 3.0, κυρίως μέσω των smartphone. Πράγματα όπως το cloud computing και το δορυφορικό Internet υψηλής ταχύτητας, που έχουν βελτιώσει δραστικά την εμπειρία μας στο διαδίκτυο, είναι όλα προϊόντα του Web 3.0. [1]

Στις περισσότερες από τις αλληλεπιδράσεις με το Διαδίκτυο, ωστόσο, οι χρήστες αλληλεπιδρούν με μια κεντρική οντότητα, που κατέχει τον πλήρη έλεγχο της προ-σφερόμενης υπηρεσίας. Μια τέτοια οντότητα υφίσταται ακόμα και στις peer-to-peer (P2P) εφαρμογές. Αυτό, λοιπόν, δημιουργεί τον προβληματισμό ότι οι χρήστες είναι υποχρεωμένοι να έχουν εμπιστοσύνη σε μια –άγνωστη, συχνά- οντότητα για να χρη-σιμοποιήσουν πολλές υπηρεσίες που χρειάζονται καθημερινά, προβληματισμός που εντείνεται όλο και περισσότερο με κάθε παραβίαση της ιδιωτικότητας των χρηστών μέσω κακόβουλων ενεργειών.

Λύση στο πρόβλημα αυτό φαίνεται να μπορεί να δώσει το Blockchain, τεχνολογία πάνω στην οποία στηρίζεται όλη η έννοια του «αποκεντρωμένου ιστού». Ειδοποιός διαφορά του Blockchain από τις υπάρχουσες τεχνολογίες είναι ότι μπορεί να προσφέ-ρει P2P επικοινωνίες (και συναλλαγές) εξαλείφοντας εντελώς το μεσάζοντα, ο οποίος παραδοσιακά (στο Web 2.0) ενεργεί και ως ελεγκτικός μηχανισμός. Κάποιοι μάλιστα, εν προκειμένω το “Web3 Foundation” χαρακτηρίζουν το Blockchain, μαζί με όλες τις αποκεντρωμένες εφαρμογές που βασίζονται σε αυτό, ως την κινητήριο δύναμη και το βασικό χαρακτηριστικό του Web 3.0, ότι δηλαδή η $3^{η}$ γενιά του Διαδικτύου είναι το “Decentralized Web”, ή αλλιώς «Αποκεντρωμένο Διαδίκτυο». [2]

Web 1.0	Web 2.0	Web 3.0
The Web	The Social Web	The Semantic Web
Read-only Web	Read and Write Web	Read, Write and Execute Web
Information sharing	Interaction	Immersion
Connect Information	Connect People	Connect context, people and knowledge
All about static content (one-way interaction)	Two-way communication through social networking, blogging etc.	Visualization
Owning content	Sharing content	Consolidating content
Web Forms	Web Applications	Smart Applications
HTML Portals	XML/RSS	RDF/RDFS/OWL
Banner Advertising	Interactive advertising	Behavioral Advertising
Britannica Online	Wikipedia	Semantic Web

Εικόνα 2-3 Σύγκριση των τριών γενεών του Διαδικτύου

Το Διαδίκτυο που χρησιμοποιούμε σήμερα βασίζεται σε αρχιτεκτονική πελάτη-εξυπηρετητή. Από την αρχή του, όταν δηλαδή οι χρήστες κατά κανόνα έβλεπαν περιεχόμενο και δεν μπορούσαν να δημιουργήσουν οι ίδιοι, η βασική ιδέα ήταν ότι κάποιος οργανισμός δημιουργούσε αυτό το περιεχόμενο και ο χρήστης τον εμπιστευόταν για να επικοινωνήσει μαζί του. Όταν, στη συνέχεια, οι χρήστες άρχισαν να επικοινωνούν μεταξύ τους μέσω P2P τεχνολογιών, πάλι υπάρχει μεσάζοντας, ο οποίος είχε εξουσία πάνω στις μεταξύ των χρηστών συναλλαγές και έθετε τους κανόνες σύμφωνα με τους οποίους αυτές πραγματοποιούνταν.

Το Blockchain, πρώτη εφαρμογή του οποίου ήταν το Bitcoin το 2009, υπόσχεται αλλαγή στον τρόπο με τον οποίο γίνονται οι επικοινωνίες μεταξύ των χρηστών, και κατ' επέκταση στον τρόπο χρήσης του Διαδικτύου. Η μετάβαση από Web 1.0 σε Web 2.0 αφορούσε ιδιαίτερα το front end των εφαρμογών, καθώς και τον τρόπο και τους λόγους χρήσης του Διαδικτύου. Στη μετάβαση όμως από Web 2.0 σε Web 3.0, αυτό που θα αλλάξει είναι το back end, και όχι η εμπειρία χρήσης. [3, σ. 3]

2.2 Δίκτυα ομότιμων κόμβων

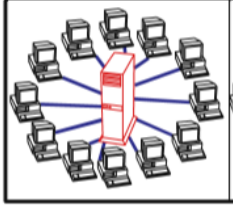
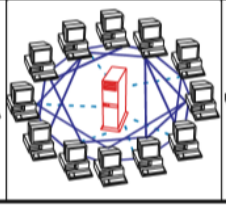
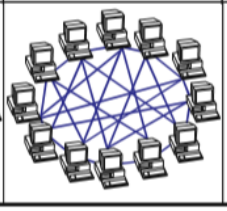
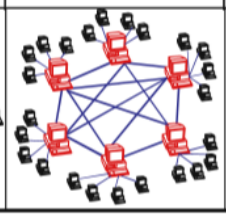
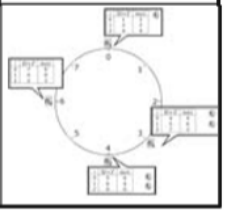
Η ενότητα αυτή παρουσιάζει την αρχιτεκτονική ομότιμων κόμβων, ή peer-to-peer, όπως πιο συνηθισμένα συναντάται, βασικό πυλώνα της τεχνολογίας του Blockchain.

Σήμερα δύο είναι τα βασικά μοντέλα δικτυακών εφαρμογών: η αρχιτεκτονική πελάτη-εξυπηρετητή (client-server) και η αρχιτεκτονική ομότιμων κόμβων (peer-to-peer), για συντομία P2P. Στην πρώτη περίπτωση υπάρχει πάντα ένας ενεργός υπολογιστής, ο εξυπηρετητής, ο οποίος προσφέρει υπηρεσίες «ακούγοντας» αιτήσεις από ένα σύνολο πελατών και απαντώντας σε αυτές. Δίκτυα τέτοιου τύπου δηλαδή εξαρτώνται άμεσα από τον κεντρικό εξυπηρετητή. Στην αρχιτεκτονική ομότιμων κόμβων,

ωστόσο, η στήριξη σε αποκλειστικούς εξυπηρετητές σε κέντρα δεδομένων είναι από μικρή έως και μηδενική.

Η αρχή λειτουργίας των δικτύων ομότιμων κόμβων είναι η απευθείας επικοινωνία ανάμεσα σε ζεύγη υπολογιστών. Οι υπολογιστές αυτοί καλούνται peers (ομότιμοι) και κατά κανόνα ελέγχονται από χρήστες, δεν αποτελούν δηλαδή μέρος κάποιας διαχειριστικής οντότητας του δικτύου. Επίσης, στην πλειοψηφία των P2P εφαρμογών, η μεταξύ τους επικοινωνία μπορεί να είναι και διακοπτόμενη, δεν είναι δηλαδή υποχρεωτικό να είναι πάντα online. Τα μέλη ενός τέτοιου δικτύου έχουν προφανώς τα ίδια προνόμια, τις ίδιες δυνατότητες και τον ίδιο ρόλο. Διαθέτουν ένα μέρος των υπολογιστικών τους πόρων για την λειτουργία του δικτύου, επιτελώντας δηλαδή το ρόλο για τον οποίο θα χρειαζόταν παραδοσιακά ο κεντρικός εξυπηρετητής. Σε πολλές περιπτώσεις, βέβαια, οι δύο προαναφερθείσες αρχιτεκτονικές συνδυάζονται. [4]

<table>
<tr><th>Client-Server</th><th colspan="4">Peer-to-Peer</th></tr>
<tr><td></td><td colspan="4">1. Resources are shared between the peers
2. Resources can be accessed directly from other peers
3. Peer is provider and requestor (Servent concept)</td></tr>
<tr><td></td><td colspan="3">Unstructured P2P</td><td>Structured P2P</td></tr>
<tr><td></td><td colspan="2">1st Generation</td><td>2nd Generation</td><td></td></tr>
<tr><td>1. Server is the central entity and only provider of service and content.
→ Network managed by the Server
2. Server as the higher performance system.
3. Clients as the lower performance system

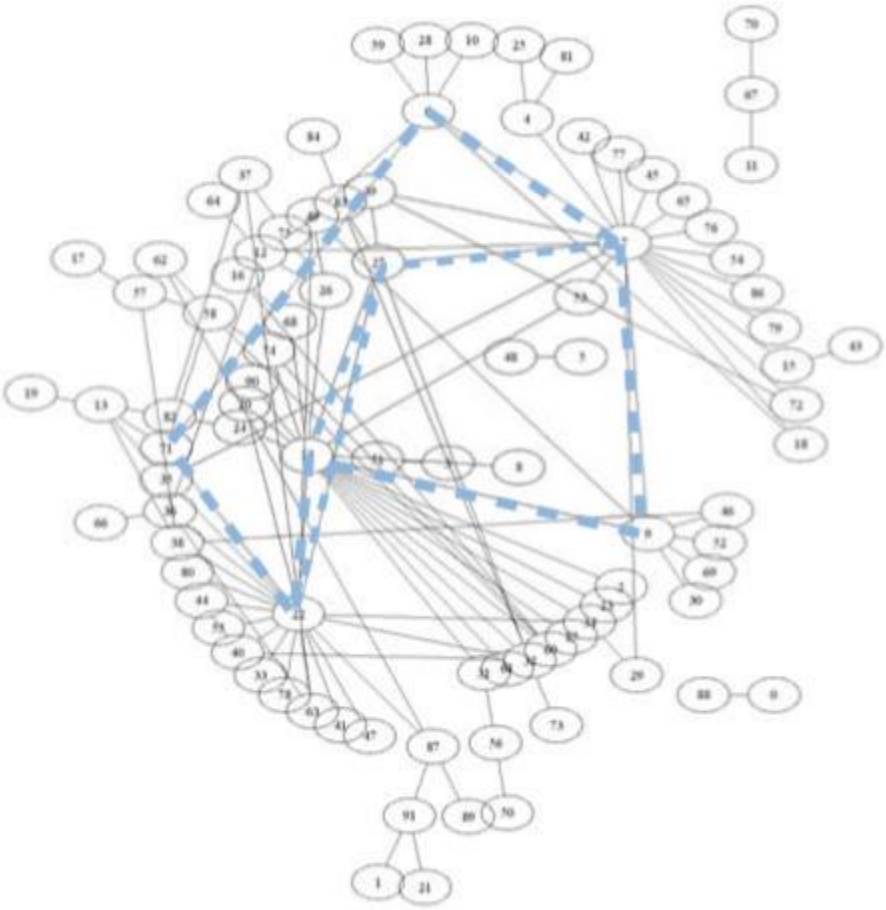
Example: WWW</td><td>Centralized P2P</td><td>Pure P2P</td><td>Hybrid P2P</td><td>DHT-Based</td></tr>
<tr><td></td><td>1. All features of Peer-to-Peer included
2. Central entity is necessary to provide the service
3. Central entity is some kind of index/group database
Example: Napster</td><td>1. All features of Peer-to-Peer included
2. Any terminal entity can be removed without loss of functionality
3. → No central entities
Examples: Gnutella 0.4, Freenet</td><td>1. All features of Peer-to-Peer included
2. Any terminal entity can be removed without loss of functionality
3. → dynamic central entities
Example: Gnutella 0.6, JXTA</td><td>1. All features of Peer-to-Peer included
2. Any terminal entity can be removed without loss of functionality
3. → No central entities
4. Connections in the overlay are “fixed”
Examples: Chord, CAN</td></tr>
<tr><td>  </td><td>  </td><td>  </td><td>  </td><td>  </td></tr>
</table>

Εικόνα 2-4 Κατηγορίες δικτύων ομότιμων κόμβων

Ένα δίκτυο P2P μπορεί να χαρακτηριστεί ως «καθαρό» P2P (δηλαδή χωρίς την ύπαρξη κεντρικής οντότητας), εάν ένας τυχαίος κόμβος του μπορεί να αφαιρεθεί χωρίς να προκληθεί πρόβλημα λειτουργίας/συνδεσιμότητας στο δίκτυο. Στην περίπτωση του συνδυασμού μπορεί να υπάρξει το “Centralized P2P”, κεντροποιημένο δηλαδή, όπου υπάρχει μια μοναδική κεντρική οντότητα που προσφέρει υπηρεσίες επιπρόσθετες αυτών που παρέχονται μέσω της επικοινωνίας των ομότιμων κόμβων, αλλά και το “Hybrid P2P”, υβριδικό μοντέλο δηλαδή, το οποίο από μόνο του χαρακτηρίζει τη δεύτερη γενιά των δικτύων ομότιμων κόμβων.

Το υβριδικό P2P μοντέλο εισάγει ένα «ενδιάμεσο» ιεραρχικό επίπεδο που περιλαμβάνει υπερκόμβους (supernodes), οι οποίοι συνδέονται με ένα πλήθος «κανονι-

κών» κόμβων (peers). Σε ένα τέτοιο δίκτυο (για παράδειγμα στην αρχιτεκτονική Gnutella 0.6), οι συνδέσεις των κόμβων σχηματίζουν μια δεντρική μορφή, όπως στην παρακάτω εικόνα. Σκοπός τέτοιων δικτύων είναι να οργανώσουν τους κόμβους σε ομάδες, στα πλαίσια των οποίων η επικοινωνία είναι πιο γρήγορη και αξιόπιστη, διατηρώντας όμως τα χαρακτηριστικά ενός “pure” P2P δικτύου. [5]



Εικόνα 2-5 Υβριδικό δίκτυο P2P σε αρχιτεκτονική Gnutella

2.3 Κρυπτογραφία ελλειπτικών καμπυλών

Το Ethereum χρησιμοποιεί την κρυπτογραφία ελλειπτικών καμπυλών για την υπογραφή των συναλλαγών, συμβάλλοντας έτσι στην ασφάλεια των συναλλαγών των χρηστών. Αυτή είναι η τεχνολογία που χρησιμοποιεί και το Bitcoin.

Η θεωρία της κρυπτογραφίας ελλειπτικών καμπυλών (Elliptic Curve Cryptography – ECC) προτάθηκε πρώτη φορά το 1985 από τους Victor Miller (IBM) and Neil Koblitz (University of Washington) ως ένας εναλλακτικός μηχανισμός για την υλοποίηση της κρυπτογραφίας δημόσιου κλειδιού. Οι αλγόριθμοι δημόσιου κλειδιού δημιουργούν ένα μηχανισμό για την κοινή χρήση κλειδιών μεταξύ μεγάλου αριθμού

συμμετεχόντων ή οντοτήτων σε ένα περίπλοκο σύστημα πληροφοριών. Σε αντίθεση με άλλους δημοφιλείς αλγορίθμους όπως το RSA, το ECC βασίζεται σε διακριτούς λογαρίθμους που είναι πολύ πιο δύσκολο να αμφισβητηθεί σε ισοδύναμα μήκη κλειδιών. Το μεγάλο πλεονέκτημα της ECC είναι το γεγονός ότι βασίζεται σε διακριτούς λογαρίθμους και συνεπώς είναι πολύ δυσκολότερο να παραβιαστεί σε σχέση με άλλους γνωστούς αλγορίθμους δημόσιων κλειδιών όπως ο RSA. [6]

Αντίστοιχα, ένα μεγάλο πλεονέκτημα του ECC απέναντι στον RSA, είναι η οικονομία στο μήκος των κλειδιών. Για το ίδιο αποτέλεσμα δηλαδή, για το επίπεδο ασφάλειας που προσφέρει ένα 256 bit ECC κλειδί, θα χρειαζόταν ένα κλειδί 3072 bits σε RSA κρυπτογραφία. Έτσι φαίνεται πόσο πιο οικονομική είναι η κρυπτογραφία ελλειπτικών καμπυλών από άποψη χώρου. Άλλο ένα χαρακτηριστικό παράδειγμα που αποδεικνύει τη σημασία αυτής της μεθόδου είναι ότι για να «σπάσει» ένα RSA κλειδί 228 bits θα χρειαζόταν υπολογιστική ισχύς που αντιστοιχεί σε ενέργεια μικρότερη από αυτή που χρειάζεται για να έρθει σε σημείο βρασμού ένα κουταλάκι του γλυκού νερό. Για να σπάσει ένα 228 bit ECC κλειδί όμως, θα χρειαζόταν ενέργεια αντίστοιχη αυτής για να βράσει όλο το νερό στον Πλανήτη. [6]

Ο αναλυτικός τρόπος λειτουργίας και το θεωρητικό υπόβαθρο του αλγορίθμου κρυπτογραφίας ελλειπτικών καμπυλών ξεφεύγουν από το αντικείμενο και το εύρος της παρούσας διπλωματικής. Περισσότερες πληροφορίες μπορεί κανείς να βρει στις πηγές [7] και [8] της βιβλιογραφίας.

2.4 Ethereum Blockchain

Το Blockchain σαν ιδέα έρχεται για να λύσει το πρόβλημα της εμπιστοσύνης μεταξύ χρηστών και κεντρικών οντοτήτων που περιγράφηκε προηγουμένως. Μπορεί να οριστεί σαν ένα είδος βάσης δεδομένων στην οποία τα δεδομένα διατάσσονται σε blocks, κάθε ένα από τα οποία συνδέεται με το επόμενο με μια κρυπτογραφημένη υπογραφή. Βασική διαφορά με μια παραδοσιακή βάση δεδομένων είναι ωστόσο ότι δεν υπάρχει δυνατότητα μεταβολής των δεδομένων που είναι γραμμένα στο blockchain. Για το λόγο αυτό χαρακτηρίζεται και ως ένα δημόσιο καθολικό (ledger), το οποίο είναι προφανώς ψηφιακό, και κατανεμημένο σε οποιονδήποτε έχει δικαιώματα να το δει.

Η επαλήθευση των δεδομένων που περιέχονται στο blockchain γίνεται με μηχανισμούς που ονομάζονται μηχανισμοί consensus. Βασικά παραδείγματα είναι το Proof of Work, που χρησιμοποιείται στο Bitcoin και στο Ethereum [9], και το Proof of Stake. Οι μηχανισμοί αυτοί βασίζονται στη δυνατότητα του Blockchain να επιβάλλει business logic σε επίπεδο συναλλαγών (transaction level), σε αντίθεση με συμβατικές βάσεις δεδομένων που το κάνουν σε επίπεδο εφαρμογής ή ολόκληρης της βάσης.

Το blockchain βασίζεται σε ένα δίκτυο ομότιμων (P2P) κάθε ένας από τους οποίους διατηρεί ένα πιστό αντίγραφο του καθολικού (ledger). Αυτό είναι ένα αρχείο

που περιλαμβάνει το σύνολο όλων των συναλλαγών που έχουν πραγματοποιηθεί. Έτσι μπορούν να γίνονται συναλλαγές χωρίς την παρουσία κάποιας κεντρικής οντότητας (που θα απαιτούσε την εμπιστοσύνη των χρηστών), αλλά μόνο με τη συνεννόηση με τους υπόλοιπους κόμβους του δικτύου.

Το πρώτο Blockchain που εμφανίστηκε ήταν το Bitcoin το 2009, δημιούργημα ενός ατόμου ή μιας ομάδας με το ψευδώνυμο Satoshi Nakamoto. Η βασική και μοναδική λειτουργία του Bitcoin blockchain είναι η εκτέλεση συναλλαγών μεταξύ των χρηστών του και η επαλήθευση αυτών των συναλλαγών με τον Proof of Work αλγόριθμο. Ο αλγόριθμος αυτός διασφαλίζει την ασφάλεια των συναλλαγών, απαιτώντας τέτοια υπολογιστική ισχύ για την επαλήθευση των συναλλαγών και τη δημιουργία blocks, ώστε να μην είναι ευάλωτο το δίκτυο σε κάποια μαζική κακόβουλη επίθεση. Η φύση του αλγόριθμου όμως απαιτεί πολλοί χρήστες να ανταγωνιστούν μεταξύ τους για τον υπολογισμό ενός hash, με αποτέλεσμα την υπερβολικά μεγάλη κατανάλωση ισχύος για τη διαδικασία αυτή (που ονομάζεται mining). Για το λόγο αυτό, το Ethereum βρίσκεται ήδη στη διαδικασία μετάβασης στον consensus μηχανισμό Proof of Stake, που έχει αποδειχτεί ότι απαιτεί πολύ λιγότερη ισχύ. [10]

Βασικό χαρακτηριστικό του Ethereum είναι πως αποτελεί μια Turing complete υπολογιστική αρχιτεκτονική, μια ολόκληρη πλατφόρμα δηλαδή πάνω στην οποία μπορεί να τρέχει κώδικας που είναι αποθηκευμένος, όπως θα αναλυθεί παρακάτω, σε έξυπνα συμβόλαια (smart contracts). Ενώ το Bitcoin δηλαδή έχει μία μόνο λειτουργία, την εκτέλεση συναλλαγών μεταξύ των χρηστών, το Ethereum επιτρέπει τη δημιουργία και λειτουργία αποκεντρωμένων εφαρμογών. Αυτό επιτυγχάνεται με εκτέλεση κώδικα στο Ethereum Virtual Machine. [11]

2.4.1 Ethereum Virtual Machine

Ο κώδικας στα συμβόλαια του Ethereum γράφεται σε γλώσσα χαμηλού επιπέδου, βασισμένη σε στοίβα (stack-based bytecode), που αναφέρεται ως «κώδικας εικονικής μηχανής Ethereum» ή «κώδικας EVM». Ο κώδικας αποτελείται από μια σειρά από bytes, όπου κάθε byte αντιπροσωπεύει μια λειτουργία. Γενικά, η εκτέλεση κώδικα είναι ένας ατέρμων βρόχος που αποτελείται από την επανειλημμένη διεξαγωγή της λειτουργίας στον τρέχοντα μετρητή προγράμματος (ο οποίος ξεκινά από το μηδέν) και στη συνέχεια την αύξηση του μετρητή προγράμματος κατά ένα, μέχρι να φτάσει το τέλος του κώδικα ή ένα σφάλμα ή STOP ή εντολή RETURN. Οι λειτουργίες έχουν πρόσβαση σε τρεις τύπους χώρου στον οποίο αποθηκεύονται τα δεδομένα:

- τη στοίβα, μια Last In First Out (LIFO) στοίβα στην οποία τιμές μπορούν να εισαχθούν και να εξαχθούν (push/pop),
- τη μνήμη, έναν απείρως επεκτάσιμο πίνακα από bytes,
- τη μακροχρόνια αποθήκευση του έξυπνου συμβολαίου, αποθηκευτικό χώρο τύπου key/value. Σε αντίθεση με τη στοίβα και τη μνήμη, η οποία επαναφέρεται στην αρχική κατάσταση μετά το τέλος του υπολογισμού, η μακροχρόνια αποθήκευση διατηρεί τα δεδομένα της.

Ο κώδικας μπορεί επίσης να αποκτήσει πρόσβαση στην τιμή, τον αποστολέα και τα δεδομένα του εισερχόμενου μηνύματος, καθώς και τα δεδομένα κεφαλίδας του μπλοκ (block header), και μπορεί επίσης να επιστρέψει μια σειρά byte δεδομένων ως έξοδο.

Το τυπικό μοντέλο εκτέλεσης του κώδικα EVM είναι απλό. Όσο η εικονική μηχανή Ethereum εκτελείται, η πλήρης υπολογιστική της κατάσταση μπορεί να οριστεί από την πλειάδα (block_state, transaction, message, code, memory, stack, pc, gas) όπου το block_state είναι η παγκόσμια κατάσταση που περιέχει όλους τους λογαριασμούς και περιλαμβάνει τα υπόλοιπα και τα αποθηκευμένα δεδομένα. Στην αρχή κάθε κύκλου εκτέλεσης, η τρέχουσα εντολή βρίσκεται με το pc-οστό byte του code (ή 0 αν pc >= length(code)), όπου pc ο μετρητής προγράμματος (program counter), και κάθε εντολή έχει τον δικό της ορισμό ως προς τον τρόπο με τον οποίο επηρεάζει την πλειάδα. Για παράδειγμα, το ADD εξάγει δύο στοιχεία από τη στοίβα (pop) και εισάγει το άθροισμά τους (push), μειώνει το gas κατά 1 και αυξάνει το pc κατά 1. Παρόλο που υπάρχουν πολλοί τρόποι βελτιστοποίησης της εκτέλεσης της εικονικής μηχανής Ethereum μέσω της σύνταξης μόλις-χρόνου, μια βασική υλοποίηση του Ethereum μπορεί να γίνει σε μερικές εκατοντάδες γραμμές κώδικα.

Το Ethereum Virtual Machine αποτελεί λοιπόν τη βάση του Ethereum. Πάνω στο EVM μπορεί κάθε κόμβος του Ethereum να τρέξει εντολές, και μάλιστα να εκτελέσει οποιαδήποτε λειτουργία, αφού το EVM είναι μια Turing Complete μηχανή. Το Ethereum σαν πλατφόρμα εμπλουτίζεται από διάφορα άλλα στοιχεία που καθορίζουν τον τρόπο λειτουργίας του. [12]

2.4.2 Λογαριασμοί

Στο Ethereum, το state (η κατάσταση) αποτελείται από αντικείμενα που ονομάζονται «λογαριασμοί» (Ethereum accounts). Κάθε λογαριασμός είναι μια διεύθυνση μήκους 20 bytes. Οι μεταβάσεις της κατάστασης (state transitions) του Ethereum αποτελούν άμεσες μεταφορές αξίας και πληροφοριών μεταξύ λογαριασμών. Ένας λογαριασμός του Ethereum περιλαμβάνει τέσσερα πεδία:

- Το nonce, ένα μετρητή που χρησιμοποιείται για να επαληθευτεί ότι κάθε συναλλαγή μπορεί να επεξεργαστεί μόνο μία φορά
- Το τρέχον ισοζύγιο του λογαριασμού (σε ethers)
- Ο κώδικας του έξυπνου συμβολαίου του λογαριασμού, εφόσον υπάρχει
- Τον αποθηκευτικό χώρο του λογαριασμού (κενός αρχικά)

Το Ether είναι το κύριο εσωτερικό κρυπτονόμισμα του Ethereum και χρησιμοποιείται για την πληρωμή των τελών συναλλαγής, αποτελεί δηλαδή και καύσιμο για το Ethereum Virtual Machine. Κάθε λειτουργία, κάθε εντολή που εκτελείται στο EVM, εφόσον καταναλώνει υπολογιστική ισχύ, απαιτεί Ether για να εκτελεστεί. Το Ether που καταναλώνεται για να γίνει μια λειτουργία καλείται gas (χάριν ευκολίας), αφού μπορεί να θεωρηθεί και καύσιμο του EVM. Λόγω της μεγάλης αξίας της μονά-

δας του νομίσματος Ether, για το σκοπό του υπολογισμού του gas χρησιμοποιούνται κυρίως οι υποδιαιρέσεις του, συγκεκριμένα το wei (1 ETH = 10^{18} wei) και το Gwei (Gigawei, 1gwei = 10^{9} wei). [13]

Σε γενικές γραμμές, υπάρχουν δύο τύποι λογαριασμών:

- Externally Owned λογαριασμοί (EOA), ελεγχόμενοι από ιδιωτικά κλειδιά, και
- Λογαριασμοί συμβολαίων, οι οποίοι ελέγχονται από τον κώδικα κάποιου έξυπνου συμβολαίου που τους διαχειρίζεται. Ένας εξωτερικός ιδιόκτητος λογαριασμός δεν έχει κώδικα και μπορεί κανείς να στείλει μηνύματα από εξωτερικό ιδιόκτητο λογαριασμό δημιουργώντας και υπογράφοντας μια συναλλαγή. Σε ένα λογαριασμό συμβολαίου, κάθε φορά που ο λογαριασμός του συμβολαίου λαμβάνει ένα μήνυμα ενεργοποιεί και εκτελεί τον κώδικα του, επιτρέποντάς του να διαβάζει και να γράφει στην εσωτερική αποθήκευση και να στέλνει άλλα μηνύματα ή να δημιουργεί έξυπνα συμβόλαια με τη σειρά του.

Τα έξυπνα συμβόλαια στο Ethereum δεν αποτελούν παραδοσιακά συμβόλαια, δεν είναι δηλαδή κάτι που πρέπει να «εκπληρωθεί». Είναι περισσότερο «αυτόνομοι πράκτορες» που ζουν μέσα στο περιβάλλον εκτέλεσης του Ethereum, εκτελώντας πάντοτε ένα συγκεκριμένο κομμάτι κώδικα όταν καλούνται από ένα μήνυμα ή συναλλαγή και έχοντας άμεσο έλεγχο πάνω στο δικό τους ισοζύγιο ether και το δικό τους αποθηκευτικό χώρο που περιλαμβάνει ό,τι μεταβλητή έχει δηλωθεί.[12]

2.4.3 Μηνύματα

Τα έξυπνα συμβόλαια έχουν τη δυνατότητα να στέλνουν μηνύματα σε άλλα συμβόλαια. Τα μηνύματα είναι εικονικά αντικείμενα που δεν έχουν σειριοποιηθεί και υπάρχουν μόνο στο περιβάλλον εκτέλεσης του Ethereum. Ένα μήνυμα περιέχει:

- Τον αποστολέα του μηνύματος
- Τον παραλήπτη του μηνύματος
- Την ποσότητα ether που μεταφέρεται μαζί με το μήνυμα
- Ένα προαιρετικό πεδίο δεδομένων
- Μια τιμή STARTGAS

Ουσιαστικά, ένα μήνυμα είναι σαν μια συναλλαγή, εκτός από το γεγονός ότι παράγεται από σύμβαση και όχι από εξωτερικό παράγοντα (EOA). Ένα μήνυμα εμφανίζεται όταν ένα συμβόλαιο που εκτελεί τον τρέχοντα κώδικα εκτελεί τον CALL κωδικό εκτέλεσης (opcode), ο οποίος παράγει και εκτελεί ένα μήνυμα. Όπως μια συναλλαγή, ένα μήνυμα οδηγεί στον λογαριασμό παραλήπτη και εκτελεί τον κώδικα του. Έτσι, οι συμβάσεις μπορούν να έχουν σχέσεις με άλλες συμβάσεις με τον ίδιο ακριβώς τρόπο που μπορούν οι εξωτερικοί φορείς.[12]

2.4.4 Ασφάλεια Συναλλαγών

Κάθε φορά που ένας λογαριασμός στέλνει μια συναλλαγή σε άλλο λογαριασμό στο δίκτυο, την υπογράφει με την προσωπική του κρυφή υπογραφή, χρησιμοποιώντας την κρυπτογραφία ελλειπτικών καμπυλών (Elliptic Curve Digital Signature Algorithm). Από αυτό το σημείο είναι υποχρέωση του δικτύου να επαληθεύσει την ταυτότητα του αποστολέα της συναλλαγής.

Όπως αναφέρθηκε προηγουμένως, για κάθε κίνηση που γίνεται στο Ethereum και επιφέρει αλλαγή στο blockchain, πρέπει να πληρωθεί ένα τέλος (gas) στο δίκτυο, το οποίο είναι ανάλογο των υπολογιστικών πόρων του EVM που θα χρησιμοποιηθούν για την περάτωσή της. Το gas αυτό αποτελεί εγγύηση ότι κανένας χρήστης δε θα προβεί σε αχρείαστες υπολογιστικές διεργασίες που θα μπορούσαν κακόβουλα να υπερφορτώσουν το δίκτυο (αφού δε θα τον συμφέρει να το κάνει). Το gas που πληρώνεται, λοιπόν, καταλήγει σε μορφή Ether στους miners του δικτύου, τους λογαριασμούς δηλαδή που σπαταλούν υπολογιστική ισχύ για να επικυρώσουν κάθε συναλλαγή.

Η επικύρωση των συναλλαγών γίνεται μέσω της λύσης ενός δύσκολου μαθηματικού προβλήματος που έγκειται στον υπολογισμό του nonce της κάθε συναλλαγής. Ο consensus μηχανισμός του Ethereum είναι το Proof of Work (PoW), διαφοροποιημένος όμως ως προς τους υπολογιστικούς πόρους που απαιτούνται. Συγκεκριμένα, σε αντίθεση με το Bitcoin, στο Ethereum το mining απαιτεί και μνήμη, αποθαρρύνοντας έτσι τη χρήση ειδικού hardware αποκλειστικά για mining. Έτσι γίνεται σημαντικά πιο ασφαλές το δίκτυο. [12]

2.5 Quorum

Το Quorum είναι ένα blockchain πρωτόκολλο που αποτελεί προέκταση του Ethereum. Προσφέρει ιδιωτικότητα στην επικοινωνία των έξυπνων συμβολαίων και τις κινήσεις καθώς και ιδιωτικότητα σε επίπεδο κόμβων, ενώ περιλαμβάνει και δικούς του consensus μηχανισμούς.

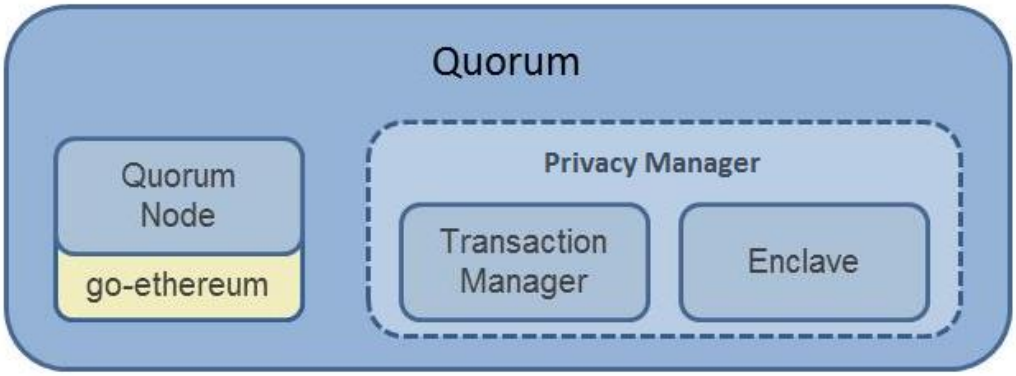
Είναι fork του go-ethereum και ανανεώνεται παράλληλα με αυτό. Βασικές του βελτιώσεις είναι οι εξής:

- Ιδιωτικότητα – Το Quorum υποστηρίζει ιδιωτικές συναλλαγές και ιδιωτικά έξυπνα συμβόλαια μέσω διαχωρισμού του public από το private state και χρησιμοποιεί κρυπτογραφία κατά την peer-to-peer ανταλλαγή μηνυμάτων για κατευθυνόμενη μεταφορά ιδιωτικών δεδομένων στους κόμβους του δικτύου.
- Εναλλακτικοί consensus μηχανισμοί: ένα δίκτυο με permissioning μηχανισμό συμμετοχής σε αυτό, απαλλάσσεται από τους κλασικούς Proof of Work και Proof of Stake μηχανισμούς που χρησιμοποιούν τα παραδοσιακά blockchains όπως το Ethereum και το Bitcoin. Οι δύο βασικοί αλγόριθμοι που προσφέρει το Quorum είναι οι:

    - Raft-based Consensus: consensus μηχανισμούς για μικρότερους χρόνους κατασκευής blocks, transaction mining, και on-demand κατασκευή blocks.
    - Istanbul BFT - consensus αλγόριθμος βασισμένος στο pBFT (Practical Byzantine Fault Tolerance).
- Network Peer Permissioning: μηχανισμός ελέγχου των κόμβων που μπορούν να συνδεθούν στο δίκτυο ανά πάσα στιγμή. Ο μηχανισμός αυτός είναι smart contract – based, και παρουσιάζεται λεπτομερώς στη συνέχεια
- Υψηλότερη Απόδοση

Το Quorum αποτελείται από τα παρακάτω βασικά στοιχεία:

- Το Quorum node: Ο κόμβος του Quorum δικτύου βασίζεται στο (public) Geth client αλλά με τροποποιήσεις που έχουν γίνει από τους developers του Quorum.
- Τον Privacy Manager (Διαχειριστή Ιδιωτικότητας), ο οποίος αποτελείται από:
    - Transaction Manager (Διαχειριστή Συναλλαγών)
    - Enclave: Δουλεύει σε συνεργασία με τον Transaction Manager και διαχειρίζεται την κωδικοποίηση/αποκωδικοποίηση των συναλλαγών με απομονωμένο τρόπο για αυξημένη ασφάλεια. Διατηρεί όλα τα private keys και είναι ουσιαστικά ένα vHSM (virtual Hardware Security Module) που δουλεύει σε απομόνωση από τα υπόλοιπα components.



Εικόνα 2-6 Τα δομικά στοιχεία του Quorum

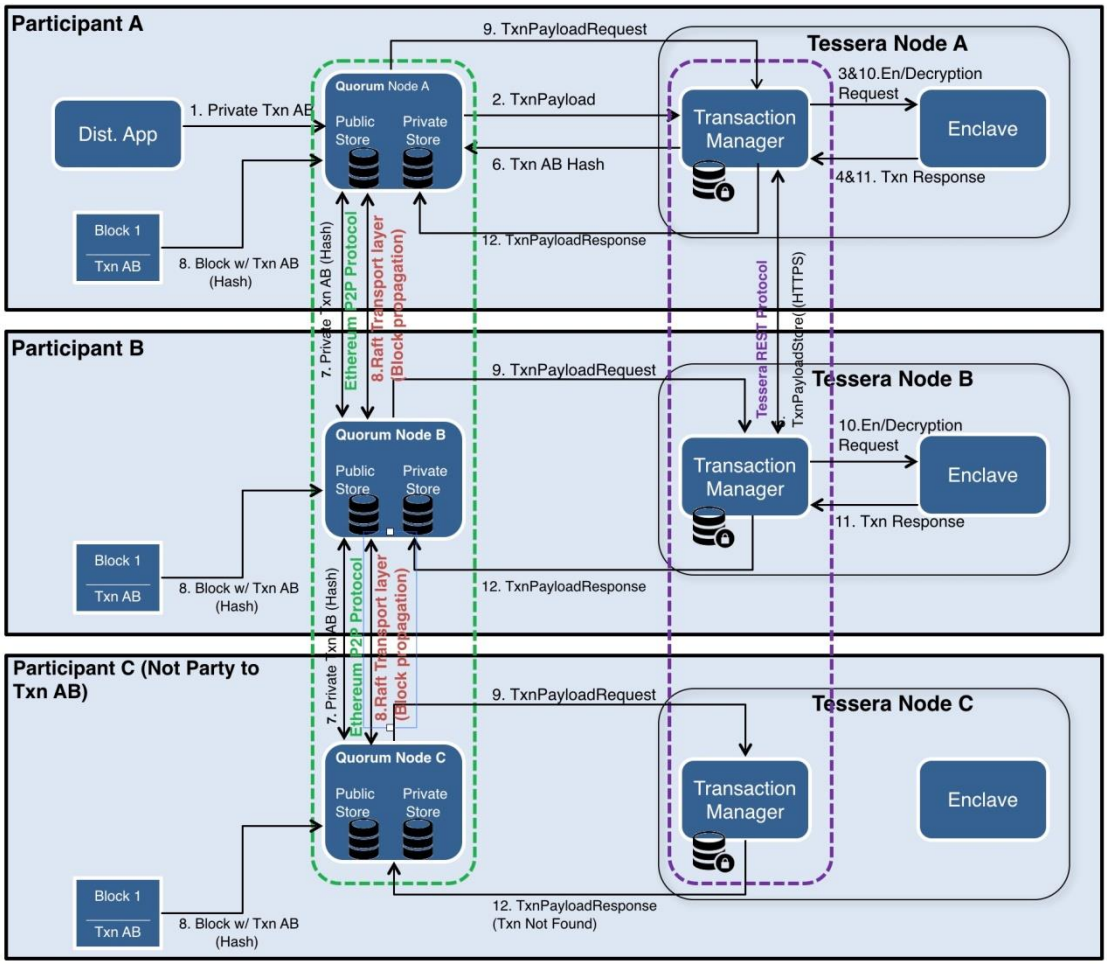
2.5.1 Ιδιωτικότητα

Το Quorum επιτυγχάνει την παραπάνω περιγραφόμενη ιδιωτικότητα κυρίως με 2 βασικά του εργαλεία. Αυτά είναι τα εξής:

Tessera: Το Tessera είναι ένα stateless Java σύστημα που χρησιμοποιείται για την κωδικοποίηση, την αποκωδικοποίηση και το διαμοιρασμό όλων των συναλλαγών για το Quorum.

Κάθε κόμβος του Tessera (δηλαδή κατ'επέκταση κάθε κόμβος του Quorum network):

- Παράγει και συντηρεί ένα πλήθος από ζευγάρια public/private κλειδιών.
- Μπορεί να ανακαλύψει και να διαχειριστεί όλους του κόμβους του δικτύου με σύνδεση μόνο σε ένα άλλο Tessera node.
- Προσφέρει public και private API για επικοινωνία:
  - Private API – Χρησιμοποιείται για επικοινωνία με το Quorum
  - Public API - Χρησιμοποιείται για επικοινωνία μεταξύ των υπολοίπων Tessera κόμβων
- Προσφέρει αμφίδρομη SSL ασφάλεια χρησιμοποιώντας TLS certificates και διάφορα μοντέλα εμπιστοσύνης όπως Trust On First Use (TOFU), whitelist, certificate authority, κλπ.
- Υποστηρίζει IP whitelisting.
- Μπορεί να συνδεθεί σε οποιαδήποτε SQL βάση δεδομένων που υποστηρίζει JDBC client.



Εικόνα 2-7 Διάγραμμα απεικόνισης της επικοινωνίας μεταξύ των στοιχείων του Quorum κατά την εκτέλεση συναλλαγής

Τρόπος λειτουργίας

Στο παράδειγμα της παραπάνω εικόνας, το συμβαλλόμενο μέρος (κόμβος) Α και το μέρος Β είναι συμβαλλόμενα μέρη της συναλλαγής ΑΒ, ενώ το συμβαλλόμενο μέρος C όχι. Η διαδικασία που ακολουθείται είναι αναλυτικά η ακόλουθη:

1. Το συμβαλλόμενο μέρος Α αποστέλλει μια συναλλαγή στον Κόμβο Α, προσδιορίζοντας το payload (ωφέλιμο φορτίο) της συναλλαγής και ορίζοντας στο privateFor τα δημόσια κλειδιά για τα συμβαλλόμενα μέρη Α και Β (το μέρος Α είναι προαιρετικό, πρόκειται για τον ίδιο κόμβο)
2. Ο Quorum κόμβος του Α διαβιβάζει τη συναλλαγή στο δικό του διαχειριστή συναλλαγών (Transaction Manager, Tessera εν προκειμένω), ζητώντας του να αποθηκεύσει το payload της συναλλαγής
3. Ο διαχειριστής συναλλαγών του Α κάνει μια κλήση στο σχετικό Enclave για να επικυρώσει τον αποστολέα και να κρυπτογραφήσει το φορτίο.
4. Το Enclave του Α ελέγχει το ιδιωτικό κλειδί του Α και, αφού επικυρωθεί, εκτελεί τη μετατροπή συναλλαγών. Αυτό συνεπάγεται:
    a. Δημιουργία ενός τυχαίου κλειδιού (RMK) και ενός nonce.
    b. Κρυπτογράφηση του ωφέλιμου φορτίου της συναλλαγής με το nonce και το RMK από το προηγούμενο βήμα.
    c. Προσπέλαση του καταλόγου των μερών της συναλλαγής, στην προκειμένη περίπτωση των Α και Β, και κρυπτογραφώντας το ίδιο το RMK από το πρώτο βήμα, χρησιμοποιώντας το κλειδί που προέρχεται από το ιδιωτικό κλειδί του Α και το δημόσιο κλειδί του Β, μαζί με ένα άλλο τυχαία παραγόμενο nonce. Έτσι, κάθε κρυπτογραφημένο RMK είναι μοναδικό για κάθε παραλήπτη και θα μοιραστεί μόνο με τον αντίστοιχο παραλήπτη μαζί με το κρυπτογραφημένο ωφέλιμο φορτίο.
    d. Επιστροφή του κρυπτογραφημένου ωφέλιμου φορτίου από το δεύτερο βήμα και όλων των κρυπτογραφημένων RMK από το τρίτο βήμα στο διαχειριστή συναλλαγών (Transaction Manager).
    e. Ο διαχειριστής συναλλαγών του Α υπολογίζει το SHA3-512 hash του κρυπτογραφημένου ωφέλιμου φορτίου και στη συνέχεια αποθηκεύει το κρυπτογραφημένο ωφέλιμο φορτίο και τα κρυπτογραφημένα RMKs από το hash στη βάση δεδομένων
    f. Στη συνέχεια, ο Διαχειριστής Συναλλαγών του Διαδικτυακού Κόμματος Α μεταφέρει με ασφαλή τρόπο (μέσω HTTPS) το κρυπτογραφημένο ωφέλιμο φορτίο και το RMK που έχει κρυπτογραφηθεί με κοινόχρηστο κλειδί σε προηγούμενο βήμα και τα nonce στο διαχειριστή συναλλαγών του Β. Ο διαχειριστής συναλλαγών του Β απαντά με μια απάντηση Ack/Nack. Εάν το Κόμμα Α δεν λάβει απάντηση / λάβει Nack από το Β τότε η συναλλαγή δεν θα μεταδοθεί καν στο δίκτυο. Αποτελεί προϋπόθεση για τους αποδέκτες για να αποθηκεύσουν το payload.
    g. Εφόσον η διαβίβαση δεδομένων στο διαχειριστή συναλλαγών του Β ήταν επιτυχής, ο διαχειριστής συναλλαγών του Α επιστρέφει το hash στον Quorum κόμβο, ο οποίος στη συνέχεια αντικαθιστά το αρχικό payload της συναλλαγής με αυτό το hash και αλλάζει την τιμή V της συναλλαγής σε 37 ή 38, κάτι που δείχνει σε άλλους κόμβους ότι αυτό το hash αντιπροσωπεύει μια ιδιωτική συναλλαγή με ένα κρυπτογρα-

φημένο payload σε αντίθεση με μια δημόσια συναλλαγή με bytecode που δε θα μπορούσαν να καταλάβουν οι υπόλοιποι κόμβοι.

5. Στη συνέχεια, η συναλλαγή μεταδίδεται στο υπόλοιπο δίκτυο χρησιμοποιώντας το παραδοσιακό Ethereum peer-to-peer πρωτόκολλο.
6. Ένα μπλοκ που περιέχει τη συναλλαγή AB δημιουργείται και διανέμεται σε κάθε μέρος στο δίκτυο.
7. Κατά την επεξεργασία του μπλοκ, όλοι οι κόμβοι θα προσπαθήσουν να επεξεργαστούν τη συναλλαγή. Κάθε κόμβος του Quorum ωστόσο θα αναγνωρίσει μια τιμή V 37 ή 38, καταλαβαίνοντας ότι πρόκειται για ιδιωτική συναλλαγή με κρυπτογραφημένο payload, και θα καλέσει τον τοπικό διαχειριστή συναλλαγών του για να προσδιορίσει εάν ο ίδιος ο κόμβος είναι μέρος της συναλλαγής (χρησιμοποιώντας το hash ως το ευρετήριο για αναζήτηση) .
8. Αφού ο κόμβος C δεν κατέχει τη Συναλλαγή, θα λάβει ένα μήνυμα NotARecipient και θα παραλείψει τη συναλλαγή, δηλαδή δεν θα ενημερώσει το private state του. Οι κόμβοι A και B θα αναζητήσουν το hash στους τοπικούς διαχειριστές συναλλαγών τους και θα αποφανθούν ότι αποτελούν μέρη της συναλλαγής. Καθένας θα κάνει μια κλήση στο Enclave του, διαβιβάζοντας το κρυπτογραφημένο payload, το κρυπτογραφημένο συμμετρικό κλειδί και την υπογραφή του.
9. Το Enclave επικυρώνει την υπογραφή και στη συνέχεια αποκρυπτογραφεί το συμμετρικό κλειδί χρησιμοποιώντας το ιδιωτικό κλειδί του κόμβου που κρατείται στο Enclave, αποκρυπτογραφεί το payload χρησιμοποιώντας το συμμετρικό κλειδί που τώρα είναι φανερό σε αυτό και επιστρέφει το αποκρυπτογραφημένο payload στο διαχειριστή συναλλαγών.
10. Οι διαχειριστές συναλλαγών των A και B στέλνουν στη συνέχεια το αποκρυπτογραφημένο payload στο EVM για την εκτέλεση του κώδικα του έξυπνου συμβολαίου. Αυτή η εκτέλεση θα ενημερώσει την κατάσταση στο Private State DB του κόμβου Quorum μόνο. Μόλις ο κώδικας εκτελεστεί απορρίπτεται, οπότε δεν είναι ποτέ διαθέσιμος για ανάγνωση χωρίς πέρασμα ξανά από την παραπάνω διαδικασία.

Constellation: αποτελεί την άλλη επιλογή που προσφέρει το Quorum για transaction management. Τα χαρακτηριστικά του είναι τα εξής:

- Διαχειρίζεται έναν αριθμό ζευγών δημόσιου / ιδιωτικού κλειδιού NaCl (Curve25519).
- Ανιχνεύει αυτόματα άλλους κόμβους στο δίκτυο μετά από συγχρονισμό με μόλις έναν άλλον κόμβο.
- Συγχρονίζει έναν κατάλογο δημόσιων κλειδιών που αντιστοιχίζεται στους κόμβους του δικτύου.
- Εκθέτει ένα δημόσιο API το οποίο επιτρέπει σε άλλους κόμβους να στέλνουν κρυπτογραφημένα bytestrings στον εκάστοτε κόμβο και να συγχρονίζουν, ανακτώντας πληροφορίες σχετικά με τους κόμβους που γνωρίζει ο κόμβος κόμβος με τον οποίο έχει γίνει επικοινωνία.
- Εκθέτει ένα ιδιωτικό API το οποίο:
    - Σας επιτρέπει να στείλετε ένα bytestring σε ένα ή περισσότερα δημόσια κλειδιά, επιστρέφοντας ένα αναγνωριστικό διευθυνσιοδοτούμενο από το περιεχόμενο (content-adressable). Αυτό το bytestring είναι

κρυπτογραφημένο με τρόπο διαφανή και αποτελεσματικό (σε συμμετρικές ταχύτητες κρυπτογράφησης) προτού μεταδοθεί στους σωστούς κόμβους (και μόνο σε αυτούς τους κόμβους.) Το αναγνωριστικό είναι ένα hash digest του κρυπτογραφημένου payload που λαμβάνει κάθε κόμβος παραλήπτη. Κάθε κόμβος παραλήπτη λαμβάνει επίσης ένα μικρό blob κρυπτογραφημένο για το δημόσιο κλειδί του το οποίο περιέχει το κύριο κλειδί για το κρυπτογραφημένο payload.
    - Επιτρέπει την παραλαβή ενός αποκρυπτογραφημένου bytestring βάσει ενός αναγνωριστικού. Τα payloads που έχει αποστείλει ή λάβει ο εκάστοτε κόμβος μπορούν να αποκρυπτογραφηθούν και να ανακτηθούν με αυτόν τον τρόπο.
    - Εκθέτει μεθόδους για διαγραφή, επανασυγχρονισμό και άλλες λειτουργίες διαχείρισης.
- Υποστηρίζει έναν αριθμό αποθηκευτικών backends, συμπεριλαμβανομένων των LevelDB, BerkeleyDB, SQLite και Directory / Maildir, κατάλληλα για χρήση με οποιονδήποτε προσαρμογέα FUSE, π.χ. για το AWS S3.
- Χρησιμοποιεί αμφίδρομα πιστοποιημένο TLS με σύγχρονες ρυθμίσεις και διάφορα μοντέλα εμπιστοσύνης, συμπεριλαμβανομένων των υβριδικών CA / tofu (προεπιλογή), tofu (OpenSSH) και whitelist (ώστε να επιτρέπεται η σύνδεση σε ένα σύνολο μόνο από δημόσια κλειδιά).
- Υποστηρίζει IP whitelisting.

Σύγκριση

Για τις ανάγκες της παρούσας διπλωματικής χρησιμοποιήθηκε το Tessera, λόγω της αυξημένης σταθερότητας σε σχέση με το Constellation, της αυξημένης ταχύτητας, καθώς και λόγω του ότι το Constellation δε λαμβάνει πλέον ενημερώσεις. Αποτελεί δηλαδή τον προεπιλεγμένο διαχειριστή συναλλαγών του Quorum κατά τη στιγμή της συγγραφής.

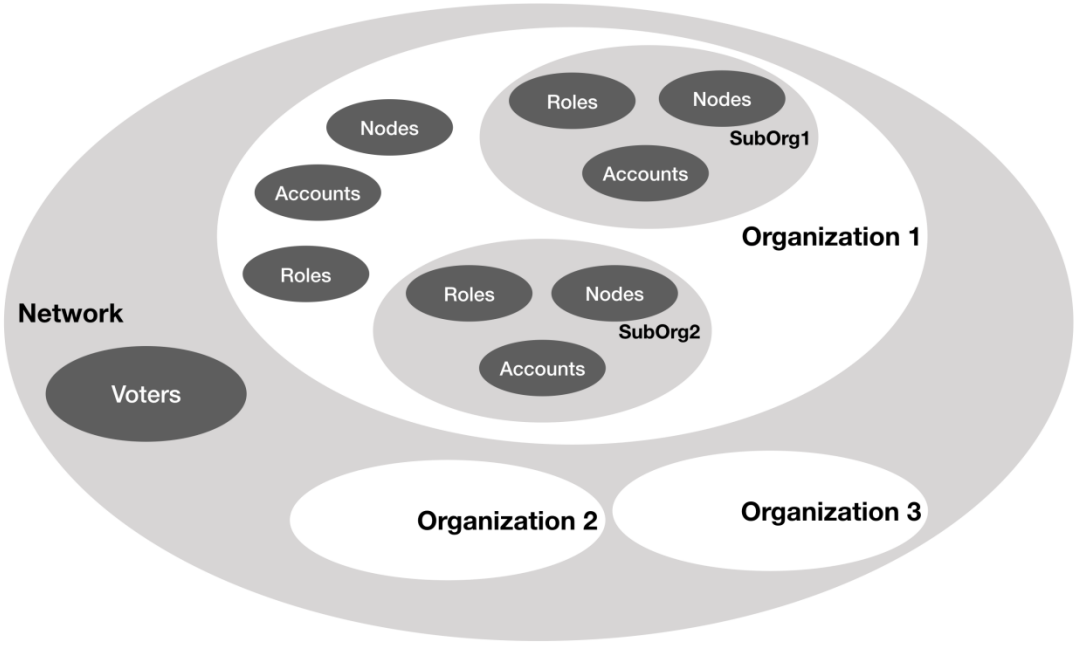
2.5.2 Μοντέλο ελέγχου δικαιωμάτων

Το τρέχον μοντέλο δικαιωμάτων στο Quorum περιορίζεται μόνο σε δικαιώματα επιπέδου κόμβου και επιτρέπει τη σύνδεση ενός δικτύου κόμβων που είναι μέρος του permissioned-nodes.json. Το μοντέλο έχει ενισχυθεί για να ανταποκριθεί στις ανάγκες των επιχειρήσεων να έχουν ένα μοντέλο δικαιωμάτων βασισμένο σε έξυπνα συμβόλαια. Αυτό έχει την ευελιξία να διαχειρίζεται τους κόμβους, τους λογαριασμούς και τους ελέγχους πρόσβασης σε επίπεδο λογαριασμού. Μια απεικόνιση του μοντέλου αυτού είναι η παραπάνω.

Βασικοί Ορισμοί :

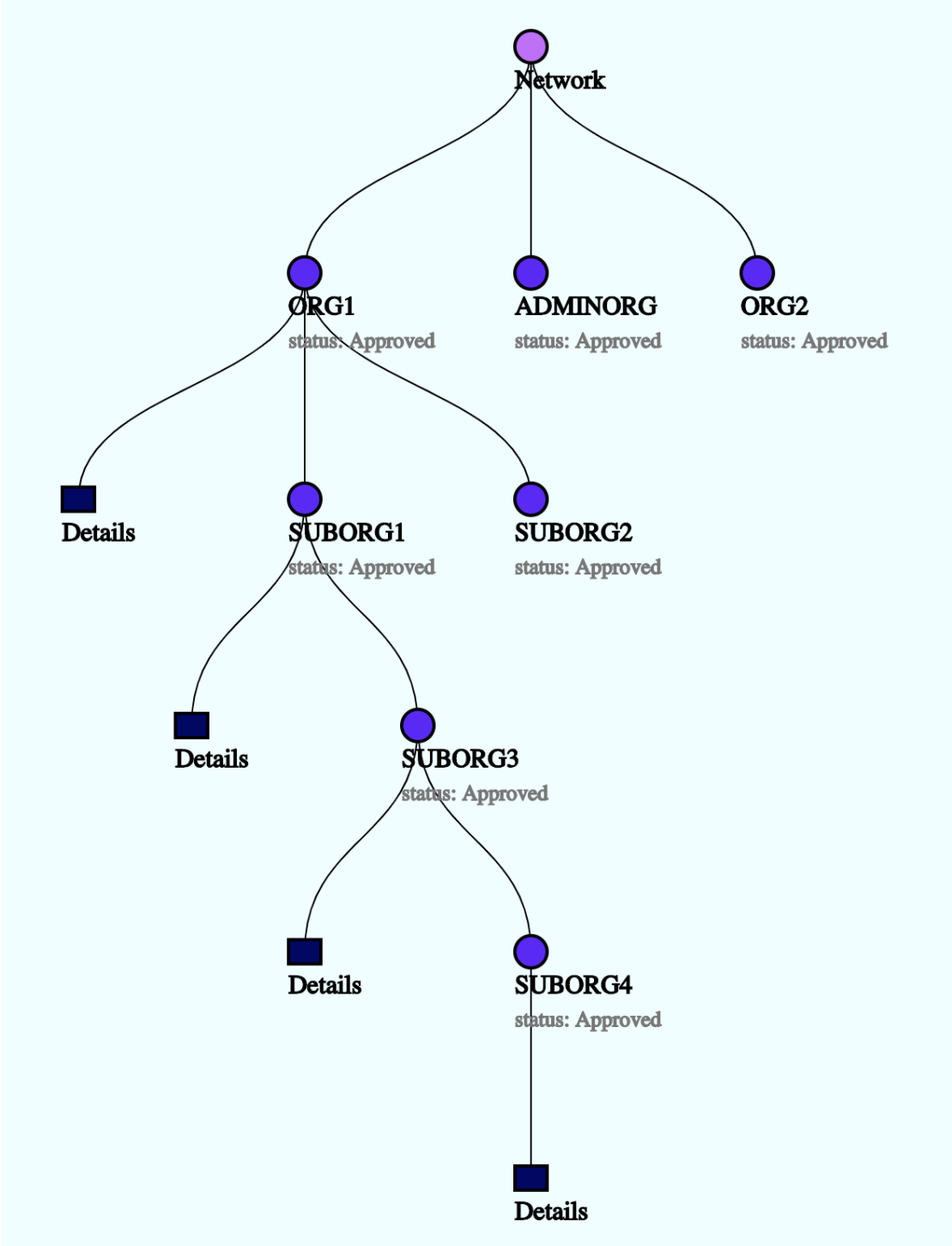
- Δίκτυο - Ένα σύνολο διασυνδεδεμένων κόμβων που αντιπροσωπεύουν ένα blockchain επιχείρησης που περιέχει οργανισμούς
- Οργανισμός - Ένα σύνολο ρόλων, Ethereum λογαριασμών και κόμβων που έχουν ποικίλα δικαιώματα για αλληλεπίδραση με το δίκτυο
- Υποοργανισμός - Περαιτέρω υποομάδα εντός του Οργανισμού. Ομαδοποίηση σύμφωνα με τις ανάγκες της επιχείρησης για το blockchain.

- Λογαριασμός - Λογαριασμός Ethereum, ο οποίος είναι externally owned λογαριασμός (EOA)
- Εκλογέας - Λογαριασμός που μπορεί να ψηφίσει για μια συγκεκριμένη ενέργεια
- Ρόλος - Μια ονομαστική εργασία σε έναν οργανισμό
- Κόμβος - Ένας κόμβος που είναι μέρος του δικτύου και ανήκει σε έναν οργανισμό ή μια δευτερεύουσα οργάνωση



Εικόνα 2-8 Σύνολα οργανισμών και χρηστών σε ένα τυπικό Quorum δίκτυο

Όπως απεικονίζεται παραπάνω, στο μοντέλο των δικαιωμάτων, το δίκτυο περιλαμβάνει μια ομάδα οργανισμών. Οι λογαριασμοί διαχειριστή δικτύου που ορίζονται σε επίπεδο δικτύου μπορούν να προτείνουν και να εγκρίνουν νέους οργανισμούς για να συμμετάσχουν στο δίκτυο και μπορούν να αντιστοιχίσουν έναν λογαριασμό ως λογαριασμό διαχείρισης της εταιρείας. Ο λογαριασμός διαχείρισης οργανισμού μπορεί να δημιουργήσει ρόλους, να δημιουργήσει υποοργανισμούς, να αναθέσει ρόλους στους λογαριασμούς του και να προσθέσει οποιονδήποτε άλλο κόμβο που είναι μέρος του οργανισμού. Ένας υποοργανισμός μπορεί να έχει δικό του σύνολο ρόλων, λογαριασμών και υποοργανισμών. Ο λογαριασμός διαχείρισης οργανισμού διαχειρίζεται και ελέγχει όλες τις δραστηριότητες σε επίπεδο οργανισμού. Ο διαχειριστής του οργανισμού μπορεί να δημιουργήσει έναν ρόλο διαχειριστή και να τον αναθέσει σε έναν διαφορετικό λογαριασμό για να διαχειριστεί τη διοίκηση ενός δευτερεύοντος οργανισμού. Τα δικαιώματα πρόσβασης ενός λογαριασμού προκύπτουν βάσει του ρόλου που του έχει ανατεθεί. Ο λογαριασμός θα είναι σε θέση να πραγματοποιεί συναλλαγές μέσω οποιουδήποτε κόμβου που συνδέεται με τον υποοργανισμό ή σε επίπεδο συνολικών οργανισμών.



Εικόνα 2-9 Παράδειγμα κληρονομικότητας ρόλων σε ένα Quorum δίκτυο

2.6 Σχετικές εργασίες

Στην ενότητα αυτή παρατίθενται εργασίες που αφορούν τεχνολογίες παρόμοιες με της παρούσας εργασίας και βοήθησαν στην ανάπτυξη, το setup και το testing της εφαρμογής.

2.6.1 Quorum-examples [14]

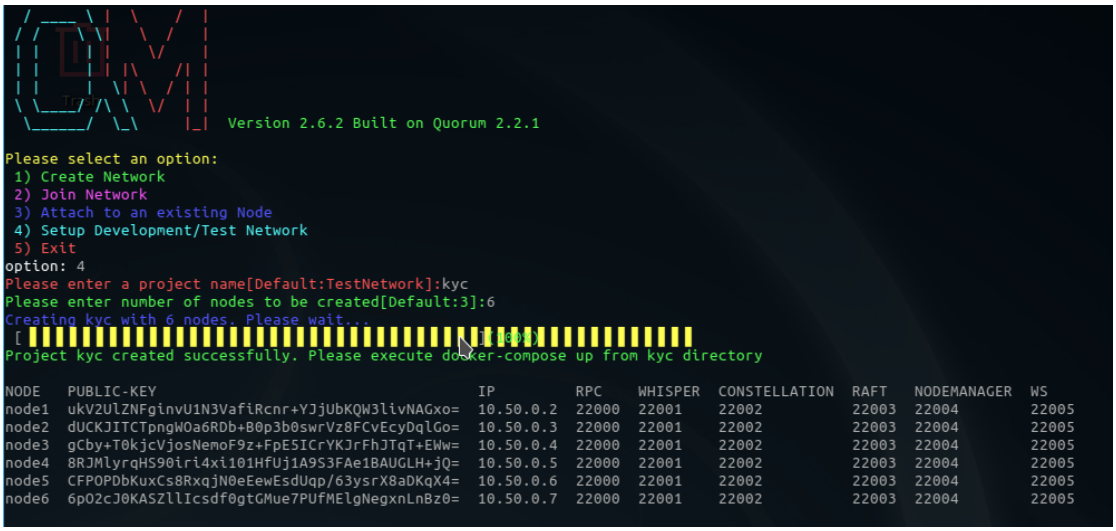
Το quorum-examples είναι ένα repository δημιουργημένο από τη JP Morgan Chase (δημιουργός του Quorum) που χρησιμοποιήθηκε στα πλαίσια της εργασίας αυτής. Περιλαμβάνει παραδείγματα που τονίζουν τα features του Quorum απέναντι σε ένα απλό Ethereum δίκτυο, scripts για γρήγορη δημιουργία δικτύου με τοπικούς κόμβους που αποτελούν docker instances, και το cakeshop, ένα UI για παρακολούθηση της κατάστασης των κόμβων.

Στην προκειμένη χρησιμοποιήθηκε το 7nodes project για να δημιουργηθεί τοπικό Quorum δίκτυο, και το cakeshop για την παρακολούθηση των κόμβων του δικτύου. Λεπτομέρειες για τον τρόπο με τον οποίο δημιουργεί το τοπικό δίκτυο δίνονται στο Παράρτημα Ι, στις οδηγίες εγκατάστασης.

2.6.2 Quorum maker [15]

Πρόκειται για εργαλείο για δημιουργία Quorum δικτύου το οποίο προσφέρει και γραφική διεπαφή για παρακολούθηση της κατάστασης των κόμβων του δικτύου, αλλά και το compilation και deployment των smart contracts στο δίκτυο.

Με το Quorum maker υποστηρίζεται δημιουργία τόσο τοπικού (development) network, όσο και «κανονικού» δικτύου με απομακρυσμένους κόμβους, αλλά και σύνδεση σε υπάρχον δίκτυο.



Εικόνα 2-10 Δημιουργία τοπικού δικτύου μέσω της διεπαφής γραμμής εντολών

2.6.3 Quorum raft cluster [16]

Άλλο ένα project με υλικό για δημιουργία Quorum δικτύου. Πρόκειται για πιο “low-level” λύση σε σχέση με τα 2 προαναφερθέντα, αφού δίνει τη δυνατότητα για (και βασίζεται στην) παραμετροποίηση του raft consensus του κάθε κόμβου. Δε χρησιμοποιήθηκε για τη δημιουργία κόμβων, αλλά είχε καλύτερο documentation και πιο

καθαρό κώδικα που βοήθησε στην κατανόηση της διαδικασίας για το στήσιμο ενός Quorum κόμβου.

2.6.4 Truffle suite on Docker [17]

Η εργασία αυτή (άρθρο μαζί με αντίστοιχο κώδικα) βοήθησε στη μετάβαση της εφαρμογής σε “dockerized” μορφή. Χρησιμοποιεί το truffle σε ένα docker image και ενθυλακώνει το ganache-cli σε ένα άλλο docker image. Χρησιμοποιώντας τη βασική δομή που προτείνει, το KYC σύστημα μπορούσε πλέον να ξεκινήσει με μία μόνο εντολή, σε οποιοδήποτε υπολογιστή που τρέχει το Docker engine, και να ανεξαρτητοποιηθεί από τα ομολογουμένως πολλά dependencies της εφαρμογής.

3

Εργαλεία και τεχνολογίες

Στο κεφάλαιο αυτό παρουσιάζονται τα κυριότερα εργαλεία και τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής της παρούσας διπλωματικής εργασίας. Πιο συγκεκριμένα παρουσιάζονται ορισμένα εργαλεία που προσφέρει το Ethereum για την ανάπτυξη, λειτουργία και πρόσβαση στις αποκεντρωμένες εφαρμογές (DApps), μεταξύ άλλων τα Geth, Ganache CLI, Ethereumjs, Web3, Solidity, Metamask.

3.1 Ethereum

Το οικοσύστημα του Ethereum παρέχει στους προγραμματιστές πολλά εργαλεία για την ανάπτυξη αποκεντρωμένων εφαρμογών (DApps), στην συνέχεια παρουσιάζονται μερικά από τα βασικότερα εργαλεία.

3.1.1 Geth

Το Geth, συντόμευση για το Go Ethereum, είναι μία standalone διεπαφή γραμμής εντολών, ανοιχτού κώδικα και υλοποιημένη στην γλώσσα προγραμματισμού Go, για την εκτέλεση του πλήρους Ethereum κόμβου[18]. Προσφέρει στο χρήστη λειτουργίες, μεταξύ άλλων, για να:

- Συνδεθεί στο Ethereum δίκτυο.
- Εξορύξει ether (mining).
- Στείλει κινήσεις μεταξύ διευθύνσεων.
- Να δημιουργήσει συμβόλαια.
- Ξεκινήσει ένα ιδιωτικό Ethereum blockchain.

3.1.2 Web3.js

Το πακέτο web3.js είναι μία συλλογή βιβλιοθηκών που επιτρέπουν την αλληλεπίδραση με έναν τοπικό ή απομακρυσμένο κόμβο του Ethereum blockchain, χρησιμοποιώντας HTTP ή IPC σύνδεση. Είναι μία διεπαφή επικοινωνίας μεταξύ της JavaScript και του Ethereum blockchain, δηλαδή τρόπος για να αναφέρεται η JavaScript (server-side ή/και front-end) σε αντικείμενα που «ζουν» μέσα στο blockchain, όπως τα έξυπνα συμβόλαια (solidity smart contracts), τα δεδομένα τους, τις συναρτήσεις τους, τις διευθύνσεις και τα υπόλοιπα λογαριασμών, όπως και πολλά άλλα. [19].

Το web3 προσφέρεται και ως npm πακέτο για την JavaScript, και στην παρούσα εργασία χρησιμοποίησα την έκδοση 1.2.2.

3.1.3 Solidity

Η Solidity είναι μία αντικειμενοστρεφής, υψηλού επιπέδου γλώσσα προγραμματισμού για την υλοποίηση smart contracts που εκτελούνται στο Ethereum Virtual Machine (EVM) και έχει δημιουργηθεί για να μεγεθύνει τις δυνατότητες της ιδεατής αυτής μηχανής. Για το λόγο αυτό η γλώσσα καλείται και «συμβολαιοστρεφής». Ο ίδιος ο πηγαίος κώδικας της γλώσσας Ethereum μάλιστα είναι γραμμένος σε Solidity.

Η γλώσσα είναι επηρεασμένη από τη C++, την Python και τη JavaScript σύμφωνα με τους δημιουργούς της. Έχει παρόμοιο συντακτικό με αυτό της JavaScript, οπότε είναι εύκολα κατανοήσιμη από έναν πολύ μεγάλο αριθμό προγραμματιστών. Είναι στατικού τύπου, υποστηρίζει κληρονομικότητα, βιβλιοθήκες και περίπλοκους τύπους ορισμένους από τον προγραμματιστή μεταξύ άλλων. [20]

Χρησιμοποιήθηκε η έκδοση 0.5.12.

3.1.4 Truffle

Το truffle είναι μια πλατφόρμα που προσφέρει πλήθος εργαλείων για ανάπτυξη, testing και deployment στο Ethereum blockchain και αποτελεί npm package. Υπόσχεται γρήγορη ανάπτυξη αποκεντρωμένων εφαρμογών, παρακολούθηση της λειτουργικότητάς τους σε μήκος όλου του stack ενός DApp που δουλεύει στο Ethereum Virtual Machine, και debugging. Συγκεκριμένα, το truffle περιλαμβάνει:

- Δυνατότητα για compilation, linking, deployment και binary management των έξυπνων συμβολαίων.
- Αυτοματοποιημένο testing των έξυπνων συμβολαίων με Mocha και Chai.
- Παραμετροποιήσιμο build pipeline με υποστήριξη για προσαρμοσμένα στις ανάγκες του χρήστη build processes.
- Δυνατότητα αυτοματοποίησης του deployment και του migration των έξυπνων συμβολαίων με scripting.
- Δυνατότητα διαχείρισης δικτύου για deployment σε πλήθος δημόσιων ή ιδιωτικών δικτύων.
- Truffle console: command line διεπαφή για επικοινωνία με τα έξυπνα συμβόλαια.
- Άμεση ανανέωση των assets κατά το deployment.

[21]

Άξια ξεχωριστής αναφοράς είναι η ευκολία που προσφέρει το truffle στην οργάνωση της δομής μιας αποκεντρωμένης εφαρμογής. Η σουίτα του truffle περιλαμβάνει έτοιμα πακέτα, τα οποία ονομάζει truffle boxes, τα οποία με μία εντολή δημιουργούν dummy projects με δομή κατάλληλη για χρήση της τεχνολογίας που χρησιμοποιεί το κάθε πακέτο. Για παράδειγμα, στην παρούσα εφαρμογή, η διαδικασία

ανάπτυξης του DApp ξεκίνησε με την εντολή `truffle init`, η οποία δημιουργεί την απλή δομή ενός DApp που βασίζεται στο truffle και στη συνέχεια εγκατέστησα και χρησιμοποίησα το React JS framework, αλλά το truffle θα μπορούσε να γλιτώσει αρκετό χρόνο με την εντολή `truffle unbox react`, όπου το react αποτελεί truffle box.[22]

Φυσικά στην προκειμένη περίπτωση, το truffle χρησιμοποιήθηκε για το deployment της εφαρμογής στο Quorum blockchain, αλλά το μεγαλύτερο μέρος της ανάπτυξης και του testing έγινε σε Ethereum blockchain, με τη βοήθεια του Ganache, όπως περιγράφεται παρακάτω.

3.1.5 Ganache

Το Ganache είναι εργαλείο που τυπικά περιλαμβάνεται στη σουίτα του Truffle [21]. Η περιγραφή του είναι “One click blockchain” και δε θα μπορούσε να αποτυπώνει καλύτερα τη λειτουργία του και τη σπουδαιότητά του για την εκπόνηση της παρούσας εργασίας. Αποτελεί npm πακέτο, είναι δηλαδή cross platform και εύκολο στην εγκατάσταση μέσω του node package manager. Προσφέρεται τόσο ως CLI (ganache-cli), δηλαδή ως διεπαφή γραμμής εντολών, όσο και σε μορφή με GUI, η οποία και χρησιμοποιήθηκε στην προκειμένη.

Πρόκειται δηλαδή για ένα Ethereum client που προσομοιώνει έναν πλήρη Ethereum κόμβο χρησιμοποιώντας τις ethereumjs βιβλιοθήκες.[23] Απαλλάσσει τον προγραμματιστή από το βάρος της εκτέλεσης ενός ολόκληρου Ethereum κόμβου δημιουργώντας ένα private blockchain, κάνοντας πολύ πιο απλή και γρήγορη την ανάπτυξη ενός DApp. Όπως φαίνονται και από τις παρακάτω εικόνες, προσφέρει επίσης δυνατότητα εποπτείας επί των smart contracts που έχουν γίνει deploy στο δίκτυο, καθώς και επί των accounts που έχει δημιουργήσει το ίδιο το Ganache. Με την εκκίνηση του Ganache, δημιουργείται ένα δίκτυο με 10 EOA (Externally Owned Ethereum Accounts). Τα accounts αυτά είναι που χρησιμοποιήθηκαν για το testing της εφαρμογής, σε συνδυασμό βέβαια με το Metamask.

3.1.6 MetaMask

Το Metamask είναι ένα από τα σημαντικότερα εργαλεία για την ανάπτυξη και το testing οποιασδήποτε αποκεντρωμένης εφαρμογής. Αποτελεί ένα add-on για όλους τους σύγχρονους browsers, και χαρακτηρίζεται ως μια γέφυρα για το χρήστη που θέλει να τρέξει μια αποκεντρωμένη εφαρμογή, χωρίς την ανάγκη να τρέχει έναν ολόκληρο Ethereum κόμβο τοπικά στον υπολογιστή του. Το βασικό του χαρακτηριστικό είναι ότι λειτουργεί σαν ένα ασφαλές ψηφιακό πορτοφόλι. [24]

Είναι ένα από τα επίσημα υποστηριζόμενα ψηφιακά πορτοφόλια για χρήση με αποκεντρωμένες εφαρμογές, μαζί με το Coinbase Wallet και το Trust Wallet για παράδειγμα. [25] Στο Metamask δηλαδή, για να χρησιμοποιήσει ο χρήστης μια εφαρμογή στο public Ethereum blockchain, θα έπρεπε να βάλει ethers στο λογαριασμό του,

και να τον συνδέσει με το Metamask, δημιουργώντας έναν ασφαλή κωδικό, για τον οποίο έχει προσωπική ευθύνη. Έπειτα, το Metamask προσφέρει μια διεπαφή διαχείρισης των διαφορετικών λογαριασμών του χρήστη, αλλά δεν έχει έλεγχο πάνω σε κανέναν από αυτούς.

Η έκδοση που χρησιμοποιήθηκε στην εργασία αυτή είναι η 7.7.4 για Mozilla Firefox.

3.2 IPFS

Το IPFS (InterPlanetary File System – Διαπλανητικό Σύστημα Αρχείων) είναι ένα πρωτόκολλο και δίκτυο που υλοποιεί Peer-to-Peer (P2P) μέθοδο για διαμοιρασμό, αποθήκευση και κοινή χρήση αρχείων και πολυμέσων.

Αποτελεί διανεμημένο σύστημα αρχείων peer-to-peer που επιδιώκει να συνδέσει όλες τις υπολογιστικές συσκευές με το ίδιο σύστημα αρχείων. Κατά κάποιο τρόπο, το IPFS είναι παρόμοιο με το World Wide Web, αλλά το IPFS, θα μπορούσε να θεωρηθεί ως ενιαίο σμήνος BitTorrent, ανταλλάσσοντας αντικείμενα μέσα σε αποθετήριο Git. Με άλλα λόγια το IPFS παρέχει υψηλής απόδοσης, διευθετημένο σε περιεχόμενο μοντέλο μπλοκ αποθήκευσης, με διευθετημένες σε περιεχόμενο υπερσυνδέσεις. Αυτό αποτελεί ένα γενικευμένο κατευθυνόμενο άκυκλο γράφο Μερκλ. Το IPFS, συνδυάζει ένα κατανεμημένο πίνακα κατακερματισμού, ένα μεγάλο μπλοκ ανταλλαγής και ονοματοχώρο αυτοπιστοποίησης. Το IPFS δεν έχει ενιαίο σημείο αποτυχίας (single point of failure), και οι κόμβοι δεν πρέπει να εμπιστεύονται ο ένας τον άλλον για να μην πειραχτούν τα δεδομένα κατά τη μεταφορά. Η Διανεμημένη Παράδοση Περιεχόμενου εξοικονομεί εύρος ζώνης και αποτρέπει επιθέσεις DDoS, με τις οποίες αγωνίζεται το HTTP.

Το σύστημα αρχείων μπορεί να έχει πρόσβαση σε μια ποικιλία από τρόπους, όπως μέσω FUSE και μέσω HTTP. Ένα τοπικό αρχείο μπορεί να προστεθεί στο σύστημα αρχείων του IPFS, καθιστώντας το διαθέσιμο στον κόσμο. Διανέμονται χρησιμοποιώντας πρωτόκολλο βασισμένο στο BitTorrent. Άλλοι χρήστες που βλέπουν το περιεχόμενο βοηθούν στην εξυπηρέτηση του περιεχομένου σε άλλους στο δίκτυο, κατά το γνωστό P2P πρότυπο. Το IPFS διαθέτει υπηρεσία ονόματος, την IPNS, ένα παγκόσμιο χώρο ονομάτων που βασίζεται στη PKI. Χρησιμεύει για να οικοδομήσει αλυσίδες εμπιστοσύνης και είναι συμβατό με άλλα NS και μπορεί να χαρτογραφήσει DNS, .onion, .bit, κλπ. στο IPNS.

Κάθε δέντρο Μερκλ αποτελεί κατευθυνόμενο άκυκλο γράφο επειδή κάθε κόμβος είναι προσβάσιμος μέσω του ονόματός του. Κάθε παράρτημα του Μερκλ είναι το κλειδί κατακερματισμού του τοπικού περιεχομένου. Έτσι, μετά τη δημιουργία δεν υπάρχει κανένας τρόπος για την επεξεργασία ενός μόνο κόμβου. Αυτό αποτρέπει κύκλους (υποθέτοντας ότι δεν υπάρχουν συγκρούσεις κατακερματισμού), εφόσον ένας δεν μπορεί να συνδέσει τον πρωτοδημιουργημένο κόμβο στον τελευταίο κόμβο για να δημιουργήσει αναφορά.

Σε γενικές γραμμές, για κάθε Μερκλ για να δημιουργηθεί ένα νέο παράρτημα ή την επαλήθευση υφιστάμενου κλάδου, ένας αλγόριθμος κατακερματισμού χρησιμοποιείται σε συνδυασμό με το τοπικό περιεχόμενο. Η εισαγωγή δεδομένων σε οποιαδήποτε από αυτούς τους αλγόριθμους κατακερματισμού είναι τεκμηριωμένη.

Από το 2015 και την αρχική του stable υλοποίηση έχει χρησιμοποιηθεί σε αρκετές πραγματικές εφαρμογές [26] όπου χρειάζεται διαφάνεια στο διαμοιρασμό αρχείων και κατάργηση του single point of failure, ενώ έχει αποδειχτεί χρήσιμο για την παράκαμψη απαγορεύσεων περιεχομένου. Παραδείγματα αποτελούν το δημοψήφισμα για την ανεξαρτησία της Καταλονίας, το οποίο το Καταλανικό Πειρατικό ανέβασε στο IPFS μετά το χαρακτηρισμό του ως αντισυνταγματικό από το Συνταγματικό Δικαστήριο της Ισπανίας, η Wikipedia, αντίγραφο της οποίας είναι ανεβασμένο στο IPFS για να επιτρέπει την πρόσβαση σε περιοχές όπου έχει απαγορευτεί, καθώς και το UltraNote, κρυπτονόμισμα που υποστηρίζει κρυπτογραφημένα μηνύματα με μεταφορές αρχείων έως και 100MB. [27]

Βασική χρήση, ωστόσο, έχει βρει σε DApps που δουλεύουν στο Ethereum Blockchain, αφού επιτρέπει την αποθήκευση αρχείων off-chain. Έτσι, τα smart contracts μιας τέτοιας εφαρμογής μπορούν να χρησιμοποιούν ένα μόνο hash για την ανάκτηση αρχείων από το IPFS, αποφεύγοντας έτσι την αποθήκευση μεγάλου όγκου πληροφοριών στο Blockchain. Για το λόγο αυτό αποδείχθηκε απαραίτητο για την παρούσα εργασία.

Το IPFS μέχρι σήμερα έχει υλοποιηθεί σε Go(lang), JavaScript, C, ενώ σε εξέλιξη βρίσκεται η υλοποίηση σε Python. Η υλοποίηση σε Go είναι η πρώτη και προτείνεται ως η πιο σταθερή. Αυτή χρησιμοποιήθηκε και για τις ανάγκες αυτής της διπλωματικής.

3.2.1 go-ipfs

Τόσο για το testing, όσο και για ενδεχόμενη χρήση της DApp εφαρμογής σε περιβάλλον παραγωγής, προτείνεται η υλοποίηση του ipfs σε Go, τόσο για τη σταθερότητά της στο διαμοιρασμό αρχείων, όσο και για την ασυμβατότητα των υπόλοιπων, λιγότερο συνηθισμένων υλοποιήσεων, στην επικοινωνία μεταξύ των κόμβων. [28]

Παρότι το μεγαλύτερο μέρος του testing έγινε με το default setup του IPFS, δηλαδή ανέβασμα των αρχείων στο public IPFS δίκτυο, προφανώς μια Know Your Customer εφαρμογή έχει ανάγκη για περισσότερη ιδιωτικότητα. Έτσι κρίνεται απαραίτητη η δημιουργία private IPFS network, δικτύου δηλαδή ιδιωτικού και αόρατου στους κόμβους του global IPFS network. Αυτό, προς το παρόν, δεν υποστηρίζεται out of the box από το IPFS, αλλά απαιτεί το setup που περιγράφεται στο Κεφάλαιο 7.

Χρησιμοποιήθηκε η έκδοση 1.13.3 της Go και η έκδοση 0.4.22 του go-ipfs.

3.2.2 js-ipfs-http-client

Το js-ipfs-http-client είναι το κατεξοχήν εργαλείο για επικοινωνία JavaScript κώδικα με IPFS δίκτυο. Αποτελεί βιβλιοθήκη με ένα σύνολο λειτουργιών για διαχείριση αρχείων και κόμβων του δικτύου και είναι node module, οπότε υποστηρίζεται από το npm και επομένως είναι ιδιαίτερα εύκολη η ενσωμάτωσή του στην εφαρμογή. Στα πλαίσια αυτής της εργασίας, χρειάστηκαν ωστόσο δύο μόνο λειτουργίες της προσφερόμενης σουίτας, συγκεκριμένα η αποθήκευση αρχείου και η ανάκτησή του. [29]

Χρησιμοποιήθηκε η έκδοση 39.0.2.

3.3 NPM

Το npm, ολογράφως Node Package Manager, είναι αφενός ένα online αποθετήριο (repository) για τη δημοσίευση Node.js projects ανοιχτού κώδικα. Δεύτερον, είναι ένα βοηθητικό πρόγραμμα γραμμής εντολών για αλληλεπίδραση με το εν λόγω αποθετήριο που βοηθά στην εγκατάσταση πακέτων, στη διαχείριση εκδόσεων και στη διαχείριση των εξαρτήσεων (dependencies) ενός project. Μια πληθώρα βιβλιοθηκών και εφαρμογών Node.js δημοσιεύονται στο npm, και πολλά προστίθενται κάθε μέρα. Αυτές οι εφαρμογές μπορούν να αναζητηθούν στο http://npmjs.org/. Κάθε πακέτο μπορεί να εγκατασταθεί με μία μόνο εντολή. [30]

Το npm αποτελεί με διαφορά το μεγαλύτερο αποθετήριο πακέτων, αριθμώντας 350.000 πακέτα, περισσότερα από τα διπλάσια από αυτά του Apache Maven Repository που βρίσκεται στη δεύτερη θέση.[31] Κάθε node πακέτο περιγράφεται από ένα package.json, και εγκαθίσταται ως ένα module, δηλαδή ένα αρχείο ή ένα directory που φορτώνεται από το Node.js μέσω της εντολής require(), στο JavaScript κώδικα.

Για τις ανάγκες της συγκεκριμένης εργασίας χρησιμοποίησα κατά κόρον το npm, για πολύ σημαντικά εργαλεία που ήταν ολόκληρες σουίτες, όπως το truffle, αλλά και για μικρά utilities όπως το timestamp-to-date.

Χρησιμοποιήθηκε η έκδοση 6.12.0 του NPM.

3.4 Node.js

Το Node.js [32] είναι ένα JavaScript runtime περιβάλλον και server-side πλατφόρμα, η οποία έχει αναπτυχθεί πάνω στο V8 Engine, τη μηχανή της Google για JavaScript και WebAssembly, γραμμένη σε C++. Είναι open source (ανοιχτού κώδικα) και έχει φτιαχτεί με γνώμονα την ταχύτητα και την επεκτασιμότητα των JavaScript εφαρμογών.

Βασικός λόγος που επελέγη είναι το μέγεθος του community πίσω από αυτό, οι αναρίθμητοι δηλαδή προγραμματιστές που καθημερινά το χρησιμοποιούν. Σε συνεργασία με το προαναφερθέν npm, αποτελεί μια τρομερά δυνατή πλατφόρμα που διευ-

κολύνει την ανάπτυξη μικρών και μεγάλων εφαρμογών. Πρόκειται για μια πλατφόρμα τόσο βελτιστοποιημένη, που ακόμα και το βασικό της μειονέκτημα, δηλαδή η μονονηματική αρχιτεκτονική, δε δημιουργεί κανένα πρόβλημα στην απόδοση. Συγκεκριμένα, χρησιμοποιώντας event looping και μονονηματική (single-threaded) αρχιτεκτονική, επιτρέπει στον εξυπηρετητή να απαντάει σε αιτήματα με non-blocking τρόπο και δίνει δυνατότητα για μεγάλη κλιμακωσιμότητα της εφαρμογής.

Χρησιμοποιήθηκε η έκδοση 12.6.0.

3.5 Docker

Το Docker είναι μια σουίτα από Platform as a Service (PaaS) προϊόντα που χρησιμοποιούν Εικονικοποίηση σε επίπεδο λειτουργικού συστήματος (OS-level virtualization). Τα containers είναι αυτόνομα, περιλαμβάνουν τις δικές τους βιβλιοθήκες, λογισμικό και παραμετροποιήσεις και μπορούν να επικοινωνήσουν μεταξύ τους. Τρέχουν ωστόσο από τον υπάρχοντα πυρήνα του λειτουργικού συστήματος (kernel) και γι'αυτό είναι πολύ ελαφρύτερα και γρήγορα από ένα Virtual Machine, το οποίο περιλαμβάνει το δικό του λειτουργικό σύστημα.

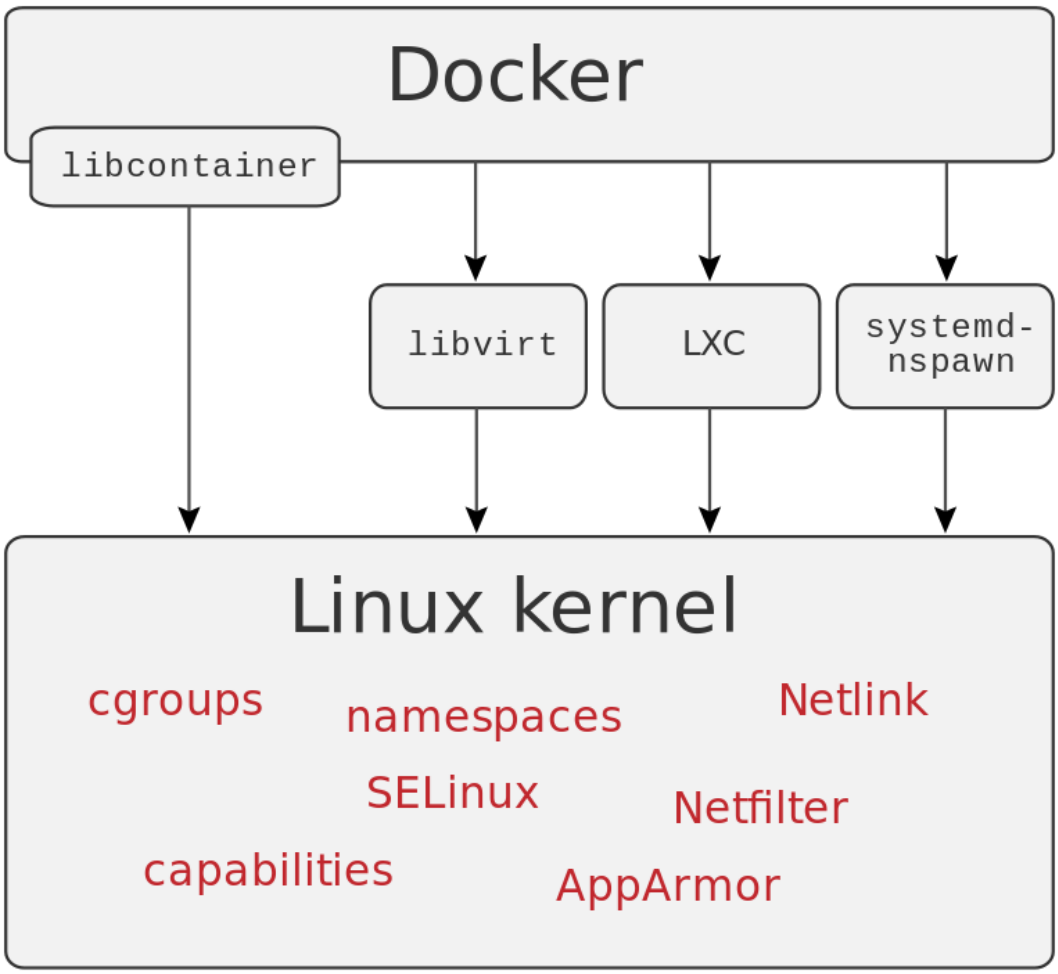
Τα containers τρέχουν μέσα από τη μηχανή του Docker (Docker Engine), η οποία είναι υπεύθυνη για το "δέσιμό" τους με το εκάστοτε λειτουργικό σύστημα / περιβάλλον. Το Docker μπορεί να πακετάρει μια εφαρμογή και τις εξαρτημένες βιβλιοθήκες της σε ένα εικονικό container που μπορεί να τρέξει σε οποιοδήποτε υποστηριζόμενο σύστημα. Προσφέρει δηλαδή ευελιξία και φορητότητα αφού η ίδια εφαρμογή μπορεί να τρέξει σε οποιοδήποτε περιβάλλον, έχοντας την ίδια συμπεριφορά, είτε τρέξει στο cloud, είτε σε Linux server, είτε και on-premises.

Για να πετύχει τα παραπάνω, χρησιμοποιεί συγκεκριμένα χαρακτηριστικά του linux kernel που σχετίζονται με απομόνωση resources, όπως τα cgroups και τα kernel namespaces. Χρησιμοποιεί επίσης ένα δικό του σύστημα οργάνωσης αρχείων (το OverlayFS) για να επιτρέψει σε περισσότερα του ενός containers να τρέξουν στο ίδιο μηχάνημα. Καταφέρνει δηλαδή να καταργήσει το βασικό μειονέκτημα των virtual machines, που είναι οι αυξημένες απαιτήσεις σε υπολογιστικούς πόρους, και να κρατήσει το βασικό πλεονέκτημά τους, τη φορητότητα. [33]

Το Docker περιβάλλον αποτελείται από 3 βασικά εργαλεία:

- Λογισμικό: Ο Docker daemon, με ονομασία dockerd, είναι ένα process που διαχειρίζεται τα containers και τα objects τους (βλέπε παρακάτω). Ο daemon ακούει για αιτήματα που γίνονται μέσω του Docker Engine API, και το Docker client program, με ονομασία docker, προσφέρει διεπαφή μέσω κονσόλας (Command line interface) για αλληλεπίδραση με το daemon μέσω του API.
- Objects: Τα Docker objects είναι διάφορα αντικείμενα που χρησιμοποιούνται για να "συναρμολογηθεί" μια εφαρμογή στο Docker. Αυτά είναι:

    - Containers: είναι τυποποιημένα, ενθυλακωμένα περιβάλλοντα που τρέχουν Docker εφαρμογές.
    - Images: είναι read-only templates που χρησιμοποιεί το Docker engine για να χτίσει containers. Χρησιμοποιούνται δηλαδή για να αποθηκεύονται και να μεταφέρονται με αυτές ολόκληρες Docker εφαρμογές.
    - Services: επιτρέπουν σε containers να επεκτείνονται σε περισσότερα του ενός Docker daemons. Το αποτέλεσμα είναι γνωστό ως Docker swarm, ένα σύνολο από συνεργαζόμενους daemons που επικοινωνούν μέσω του Docker API.
- Registries: Τα Docker registries είναι αποθήκες (repositories) για Docker images, στο οποίο συνδέονται οι Docker clients για να κατεβάσουν ή να ανεβάσουν Docker images. Μπορούν να είναι public ή private. Οι δύο πιο δημοφιλείς public registries είναι το Docker Hub και το Docker Cloud.



Εικόνα 3-1 Απεικόνιση της επικοινωνίας του Docker με το linux kernel

3.5.1 Docker Compose

Σημαντικό ρόλο επιτέλεσε το εργαλείο Docker Compose. Αποτελεί εργαλείο για τον ορισμό και την εκτέλεση εφαρμογών που περιλαμβάνουν περισσότερα του ενός Docker containers. Χρησιμοποιεί YAML αρχεία για να ορίσει και να παραμετροποιήσει τα services της εφαρμογής και εκτελεί τις start-up διεργασίες όλων των περιεχόμενων containers με μία εντολή. Η CLI διεπαφή docker-compose επιτρέπει επίσης στους χρήστες να τρέξουν εντολές σε περισσότερα του ενός containers με τη μία, για παράδειγμα να χτίσει images, να επεκτείνει containers σε περισσότερους daemons και να τρέξει containers που έχουν σταματήσει. Το docker-compose.yml αρχείο χρησιμοποιείται για να ορίσει τα services της εφαρμογής και περιλαμβάνει δεδομένα παραμετροποίησης της εφαρμογής.

Στην προκειμένη περίπτωση, το Docker Compose χρησιμοποιήθηκε τόσο για το testing της εφαρμογής (στο quorum-examples/7nodes ή στο quorum-maker) όσο και για το εύκολο deployment αυτής. Χωρίς το docker compose, η σωστή λειτουργία της εφαρμογής ήταν πολύ εξαρτημένη από τις εκδόσεις ενός πλήθους εργαλείων, και λόγω της αστάθειας των εργαλείων αυτών (beta εκδόσεις, ασυμβατότητες με διαφορετι-

κές εκδόσεις του node, npm, Solidity compiler κλπ), πολύς χρόνος καταναλώθηκε σε αυτό το κομμάτι του debugging. Με το Docker Compose όμως, η σωστή λειτουργία σε ένα μηχάνημα αποτελεί εγγύηση της σωστής λειτουργίας σε ένα δεύτερο μηχάνημα.[34]

3.6 Ανάπτυξη ιστοσελίδων

Για το web κομμάτι της εφαρμογής χρησιμοποιήθηκαν οι εξής τεχνολογίες. Χρησιμοποιήθηκε προφανώς η HTML (Hypertext Markup Language), CSS για το styling της σελίδας και JavaScript για το server-side κομμάτι και το front-end.

3.6.1 JavaScript

Η JavaScript είναι μία γλώσσες προγραμματισμού για τις οποίες δε χρειάζονται συστάσεις. Αποτελεί την τρίτη πιο δημοφιλή γλώσσα προγραμματισμού, μετά την Python και τη Java. [35] Χρησιμοποιείται κατά κόρον σε διαδικτυακές εφαρμογές (και όχι μόνο) και υποστηρίζεται από όλους τους μοντέρνους φυλλομετρητές. Είναι ο καλύτερος (αν όχι ο μόνος ενδεδειγμένος) τρόπος να δώσουμε δυναμικό περιεχόμενο σε μία ιστοσελίδα κάνοντάς την να επιτελεί σύνθετες λειτουργίες. Μπορεί επίσης να χρησιμοποιηθεί και για την ανάπτυξη προγραμμάτων που δεν εκτελούνται σε φυλλομετρητές, όπως για παράδειγμα σε εξυπηρετητές, βάσεις δεδομένων, επεξεργαστές PDF εγγράφων, Desktop εφαρμογές, αλλά και εφαρμογές κινητών.

Είναι υψηλού επιπέδου, δυναμική, weakly typed, prototype-based, multi-paradigm, interpreted γλώσσα προγραμματισμού. Ως multi-paradigm γλώσσα, η JavaScript υποστηρίζει event-driven, functional, and imperative (including object-oriented and prototype-based) προγραμματιστικά στυλ. Έχει έτοιμες προγραμματιστικές διεπαφές (APIs) για την επεξεργασία κειμένου, πινάκων, ημερομηνιών, καθώς και τον χειρισμό DOM (Document Object Model), όμως δεν υποστηρίζει λειτουργίες εισόδου εξόδου (I/O), όπως δικτύωση, αποθήκευση, γραφικά, παρά βασίζει την υλοποίηση των λειτουργιών αυτών στο περιβάλλον εκτέλεσής της.[36]

Για την εκτέλεσή της είναι απαραίτητη κάποια μηχανή JavaScript (JavaScript engine). Στην περίπτωση του Node.js που χρησιμοποιήθηκε εδώ, η μηχανή αυτή ήταν η Google V8.

Βασικά πλεονεκτήματα της JavaScript για διαδικτυακές εφαρμογές είναι [37]:

- Λιγότερη αλληλεπίδραση με τον εξυπηρετητή. Προσφέρεται η δυνατότητα προεπεξεργασίας των δεδομένων τοπικά, στον φυλλομετρητή του πελάτη πριν αυτά σταλούν στον εξυπηρετητή με αποτέλεσμα τη μείωση του φόρτου του εξυπηρετητή, αλλά και ολόκληρου του δικτύου.
- Άμεση ανατροφοδότηση στο χρήστη. Υπάρχει η δυνατότητα εμφάνισης μηνυμάτων στο χρήστη σχετικά με την κατάσταση διεργασιών που απαιτούν πολύ χρόνο κτλ.

- Αυξημένη διαδραστικότητα ιστοσελίδας. Μπορούν να δημιουργηθούν διεπαφές, οι οποίες ανταποκρίνονται στις κινήσεις του χρήστη, χωρίς να χρειάζεται να ανανεωθεί ολόκληρη η ιστοσελίδα (client-side διεργασίες).
- Πλουσιότερες διεπαφές. Η JavaScript εμπλουτίζει την HTML και τα CSS, δίνοντας τη δυνατότητα δημιουργίας εφέ που βελτιώνουν την εμφάνιση μιας σελίδας και την απομακρύνουν από το στατικό χαρακτήρα του Διαδικτύου πριν την έλευση της JavaScript.

Η JavaScript βέβαια έρχεται και με ορισμένους περιορισμούς:

- Η JavaScript στην πλευρά του χρήστη (client-side) δεν επιτρέπει το γράψιμο και το διάβασμα αρχείων, κάτι που έχει γίνει στάνταρ για λόγους ασφαλείας.
- Δε μπορεί να χρησιμοποιηθεί για εφαρμογές που απαιτούν πολυνηματικές διεργασίες, ή χρειάζεται να εκμεταλλευτούν περισσότερους του ενός επεξεργαστές.

3.6.2 ReactJS

Η ReactJS αποτελεί μια Javascript βιβλιοθήκη για τη δημιουργία user interfaces. Τα βασικά χαρακτηριστικά της είναι τα εξής:

- Declarative: Η React κάνει εύκολη τη δημιουργία διαδραστικών UIs. Δίνει τη δυνατότητα σχεδιασμού απλών views για κάθε state της εφαρμογής, και αναλαμβάνει την ενημέρωση και την παρουσίαση των σωστών components όταν τα δεδομένα αλλάξουν. Τα declarative views κάνουν τον κώδικα πιο οργανωμένο, πιο προβλέψιμο και διευκολύνει τον εντοπισμό λαθών.
- Component-based: Με τη React μπορεί κανείς να δημιουργήσει ενθυλακωμένα components που διαχειρίζονται το δικό τους state, και έπειτα να τα συνθέσει για να φτιάξει περίπλοκα UIs, τα οποία όμως έχουν οργανωμένο κώδικα. Εφόσον η λογική των components είναι γραμμένη στη JavaScript και όχι σε templates, το state μπορεί να παραμείνει έξω από το DOM κατά τον την ανανέωση και τον εμπλουτισμό της εφαρμογής με δεδομένα.
- Learn Once, Write Anywhere: Η React είναι αγνωστική ως προς το υπόλοιπο stack της εκάστοτε εφαρμογής. Αυτό διευκολύνει τον προγραμματισμό του front end, χωρίς να υπάρχει ανησυχία για τον υπόλοιπο κώδικα της εφαρμογής. Σε συνεργασία με το Node, ακόμα, μπορεί να κάνει render στην πλευρά του διακομιστή, ενώ χρησιμοποιείται και για mobile εφαρμογές με χρήση του React Native framework. [38]

Στην παρούσα εφαρμογή, το React επιλέχτηκε λόγω της επεκτασιμότητας που προσφέρει στο σχεδιασμό του UI. Έτσι, εκτός από την ευκολία της δημιουργίας ενός απλού UI όπως αυτό του KYC, επιτρέπει την επέκταση της εφαρμογής, απλά φτιάχνοντας κι άλλα components, χωρίς να χρειαστεί να αλλαχτεί υπάρχων κώδικας. [39]

Χρησιμοποιήθηκε η έκδοση 16.6.3.

3.6.3 Bootstrap

Το Bootstrap είναι ένα ελεύθερο, ανοιχτού κώδικα πλαίσιο ανάπτυξης ιστοσελίδων και διαδικτυακών εφαρμογών (front-end web framework). Απαρτίζεται από μία σειρά από Sass stylesheets, με σκοπό να κάνει ευκολότερη την ανάπτυξη σελίδων που είναι όμορφες, λειτουργικές και προσαρμοστικές στις διάφορες διαστάσεις των συσκευών από τις οποίες μπορεί κανείς να περιηγηθεί σε αυτές. [40]

Το Bootstrap προσφέρεται ως npm πακέτο, και στην εργασία χρησιμοποιήθηκε η έκδοση 4.4.1.

4

Ανάλυση Απαιτήσεων Συστήματος

Σε αυτό το κεφάλαιο θα αναλυθούν οι λεπτομέρειες για τις ανάγκες που οδήγησαν στην υλοποίηση αυτού του project, τους πιθανούς πελάτες στους οποίους απευθύνεται η εφαρμογή (ή πιθανή επέκταση αυτής), τα είδη χρηστών, καθώς και τις λειτουργικές ή μη απαιτήσεις. Θα παρουσιαστούν παράλληλα και οι παραδοχές που έγιναν κατά την ανάπτυξη της εφαρμογής.

Σκοπός του συστήματος

Σκοπός του συστήματος είναι η κάλυψη της ανάγκης ενός Know Your Customer (KYC) συστήματος που θα τρέχει πάνω στο blockchain, δηλαδή on-chain, θα συνεργάζεται με το Quorum blockchain, και θα αποτελεί και διεπαφή με τις off-chain οντότητες που θα έχουν πρόσβαση στα δεδομένα που ανεβάζει ο χρήστης για ταυτοποίηση. Στην περίπτωση, συγκεκριμένα, ενός marketplace όπου διακινείται ψηφιακό υλικό, ένα blockchain περιβάλλον είναι κατάλληλο για την υλοποίησή του. Χρειάζεται, ωστόσο, και έλεγχος, τόσο για τη δυνατότητα ενός χρήστη να χρησιμοποιήσει το σύστημα, όσο και για το χρονικό διάστημα για το οποίο θα μπορεί να είναι εγγεγραμμένος.

Την ανάγκη αυτή καλύπτει η παρούσα εφαρμογή, η λειτουργικότητα της οποίας υπεισέρχεται στο κομμάτι της εισόδου του χρήστη και όχι στο κομμάτι των αγοραπωλησιών ή οποιωνδήποτε άλλων συναλλαγών θα εκτελέσει μέσα σε αυτό. Η χρήση, ωστόσο, του IPFS, είναι κατάλληλη και για την αποθήκευση πολυμέσων ή οποιουδήποτε άλλου τύπου αρχείων χρειάζεται, οπότε όσον αφορά το κομμάτι των εργαλείων, καλύπτει και τις ανάγκες του DApp συστήματος που θα έρθει για να καλύψει όποιες άλλες ανάγκες προκύψουν.

Κατηγορίες χρηστών

1. Χρήστες της πλατφόρμας

Χρήστες της πλατφόρμας αποτελούν οι οντότητες που επιχειρούν είσοδο στο Quorum blockchain μέσω της KYC εφαρμογής. Με τη σύνδεσή τους στο προσωπικό τους Quorum account, είτε μέσω Metamask είτε μέσω κάποιου άλλου blockchain wallet, φτιάχνουν λογαριασμό στο blockchain μέσω της ροής του KYC που θα παρουσιαστεί παρακάτω.

Το Quorum address τους, δηλαδή, αποτελεί το μοναδικό κλειδί για τη σύνδεσή τους στο blockchain. Μετά την εγγραφή τους θεωρούνται approved, μόνο όμως εφόσον ανεβάσουν έγγραφο ταυτοποίησης και η εξωτερική οντότητα το εγκρίνει.

2. Εξωτερική οντότητα

Η εξωτερική οντότητα είναι αγνωστική από την πλευρά του συστήματος. Τέτοια μπορεί να είναι ένας οποιοσδήποτε χρήστης ο οποίος έχει δικαίωμα να εγκρίνει την ταυτότητα του χρήστη.

Ο σχεδιασμός της εξωτερικής οντότητας δεν εμπίπτει στο πλαίσιο της εργασίας αυτής. Μια τέτοια οντότητα θα μπορούσε να είναι μια δημόσια υπηρεσία που μπορεί να επιβεβαιώσει την ταυτότητα ενός χρήστη. Στο πλαίσιο της εργασίας ωστόσο εμπίπτει ο σχεδιασμός της προδιαγραφής της μορφής του HTTP Request που αποστέλλεται σε αυτήν, και του HTTP Response που επιστρέφει στην εφαρμογή.

4.1 Λειτουργικές απαιτήσεις

Οι λειτουργικές απαιτήσεις της εφαρμογής είναι οι παρακάτω:

- Να λειτουργεί Quorum Blockchain με τουλάχιστον 2 κόμβους.
- Να γίνονται deploy σε αυτό τα απαραίτητα smart contracts της εφαρμογής.
- Οι χρήστες να έχουν πρόσβαση στα δεδομένα που έχουν καταχωρήσει για την εγγραφή τους στην υπηρεσία, όπως όνομα και ημερομηνία λήξης σύνδεσης
- Να μην απαιτείται συμβατικός login μηχανισμός για την εφαρμογή, αλλά να συνδέεται με το Quorum account του (EOA).

Το έξυπνο συμβόλαιο UserStorage θα πρέπει:

- Να διατηρεί το σύνολο των EOAs των χρηστών.
- Να μπορεί να ανανεώσει το χρόνο σύνδεσης του χρήστη.
- Να μπορεί να τον καταργήσει.
- Να μπορεί να επιστρέψει τις πληροφορίες του.
- Να κάνει τους απαραίτητους ελέγχους σε περίπτωση αιτήματος νέου χρήστη.

Η εξωτερική οντότητα θα πρέπει:

- Να μπορεί να αποδεχτεί ή να απορρίψει το αίτημα.

4.2 Μη λειτουργικές απαιτήσεις

Ως μη λειτουργικές χαρακτηρίζονται οι απαιτήσεις που δεν έχουν να κάνουν με τα ποσοτικά στοιχεία της εφαρμογής, αλλά με τα ποιοτικά.

Επεκτασιμότητα

- Με δεδομένο ότι η παρούσα αποκεντρωμένη εφαρμογή δεν είναι production ready, πρέπει να μπορεί να επεκταθεί εύκολα από πιθανό μελλοντικό διαχειριστή.
- Πρέπει να μπορεί να παραμετροποιηθεί για τις ανάγκες του εκάστοτε πελάτη, όπου πελάτης κάποιος διαχειριστής ενός Quorum blockchain που χρειάζεται KYC.

Ασφάλεια – Απόδοση

- Οι συναλλαγές μεταξύ των χρηστών/κόμβων πρέπει να είναι ασφαλείς και ιδιωτικές (εξασφαλίζεται από το Quorum).
- Τα έξυπνα συμβόλαια θα πρέπει να είναι ασφαλή, με κώδικα που τρέχει με όσο το δυνατόν μικρότερο υπολογιστικό κόστος (το gas δεν είναι πρόβλημα στο Quorum λόγω μηδενικού gas price).
- Αν δεν είναι απαραίτητο να ανανεωθεί το public/private state του Blockchain ή να φτιαχτεί καινούριο block, να μη γίνεται, για μείωση του κόστους

4.3 Παραδοχές

Στην παρούσα διπλωματική εργασία, έχουν γίνει ορισμένες παραδοχές, αφενός στη βάση ότι δεν αποτελεί production ready εφαρμογή, αλλά ένα proof of concept για ένα ολοκληρωμένο σύστημα αγοραπωλησιών ή ιδιωτικών συναλλαγών γενικότερα στο Quorum blockchain, και αφετέρου λόγω περιορισμών από τις τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν:

- Πρώτη και σημαντικότερη παραδοχή αποτελεί η “1-1” σχέση του Quorum account και του Quorum blockchain peer/node. Συγκεκριμένα, ένας χρήστης (δηλαδή ο κάτοχος των δικαιωμάτων διαχείρισης ενός account), έχει δικαίωμα εισόδου μόνο από ένα συγκεκριμένο node, δηλαδή peer του Quorum blockchain. Αυτό προέκυψε από το ότι (ορθώς) το permissioning API του Quorum, όπως αναλύθηκε παραπάνω, εκτελεί ελέγχους σε επίπεδο Node και όχι σε επίπεδο account. Επομένως, δεν υπάρχει τρόπος να περιοριστεί η ορατότητα της εκάστοτε συναλλαγής σε επίπεδο λογαριασμών από την πλευρά του Quorum. Εννοείται, βέβαια, πως η ιδιωτικότητα του account εξασφαλίζεται από την πλευρά του χρήστη και της διαχείρισης του προσωπικού του account με wallet/private key ή οποιουδήποτε άλλου software ή hardware τρόπου επιλέξει.
- Ο χρήστης της εφαρμογής ήδη έχει Quorum account (EOA) και το χρησιμοποιεί από μοναδικό κόμβο του δικτύου. Λόγω της παραπάνω παραδοχής, θα πρέπει να συνδέεται από συγκεκριμένο κόμβο για να μπορεί το Quorum να διαχειρίζεται την ιδιωτικότητα των συναλλαγών του, εφόσον χρειαστεί.

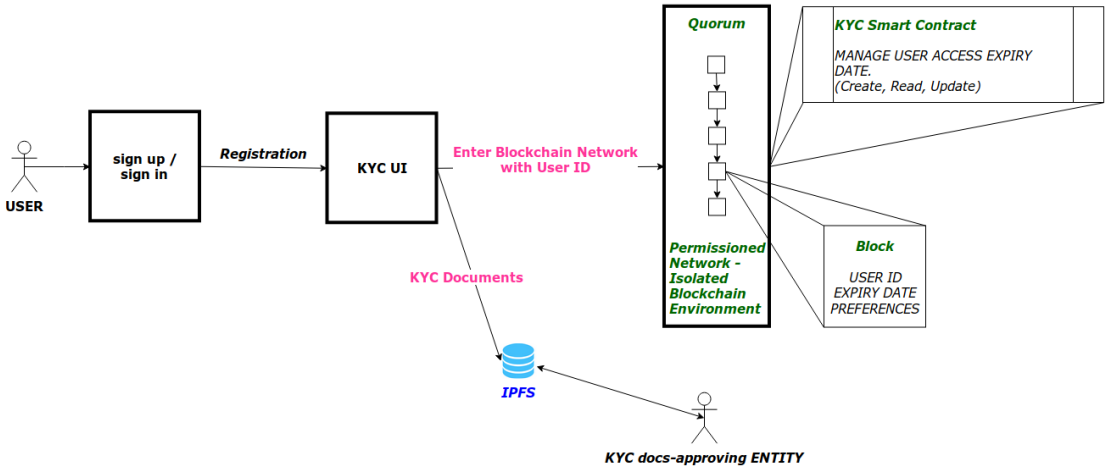
- Παραδοχή είναι, ακόμη, η “εμπιστοσύνη” στην ταυτότητα της εξωτερικής οντότητας. Όσον αφορά το απλό REST API που χρησιμοποιείται για την έγκριση ή απόρριψη του αιτήματος του κάθε χρήστη, δεν πραγματοποιείται έλεγχος για το ποιος στέλνει request εμφάνισης εγγράφου, έγκρισης ή απόρριψης.
- Η εξωτερική οντότητα έχει πρόσβαση στο ιδιωτικό IPFS δίκτυο, στο οποίο είναι ανεβασμένο κάθε έγγραφο.
- Τέλος, στην πορεία της εργασίας κρίθηκε αδύνατη η δημιουργία αυτοματοποιημένου μηχανισμού διαγραφής (εφόσον χρειάζεται) χρήστη από το Quorum δίκτυο. Συγκεκριμένα, όπως θα περιγραφεί πιο αναλυτικά στο Κεφάλαιο 6, ούτε το web3js πακέτο ούτε και το quorum.js (το οποίο αποτελεί επέκταση του πρώτου) δεν έχει τρόπο επικοινωνίας με την κονσόλα του Quorum, από την οποία θα μπορούσε να δοθεί εντολή να διαγραφεί ένας χρήστης του οποίου ο λογαριασμός έχει λήξει.

5

Σχεδιασμός και υλοποίηση Συστήματος

Στο κεφάλαιο αυτό θα αναλυθεί ο σχεδιασμός και η υλοποίησης της αποκεντρωμένης εφαρμογής, οι αποφάσεις που πάρθηκαν στα διάφορα στάδια της ανάπτυξης, καθώς και τα προβλήματα που τελικά οδήγησαν στις παραδοχές του προηγούμενου κεφαλαίου. Στόχος του κεφαλαίου αυτού, λοιπόν, είναι να γίνει κατανοητή η διαδικασία ανάπτυξης που ακολουθήθηκε.

5.1 Σχεδιασμός



Εικόνα 5-1 Η αρχιτεκτονική ενός συστήματος που χρησιμοποιεί το KYC

Στο παραπάνω διάγραμμα παρουσιάζεται η αρχιτεκτονική ενός συστήματος που χρησιμοποιεί Quorum blockchain, KYC για ταυτοποίηση χρήστη, IPFS για ανέβασμα εγγράφων, και μια εξωτερική οντότητα για την επιβεβαίωση των αιτημάτων του χρήστη.

Η διαδικασία ξεκίνησε με αντικειμενικό στόχο τη δημιουργία ενός συστήματος που θα λειτουργήσει σαν πύλη εισόδου σε κάποιο Quorum blockchain. Η επιλογή των επιμέρους εργαλείων οφείλεται στους λόγους που αναφέρονται στα αντίστοιχα κεφάλαια και δε θα αναλυθούν σε αυτό το σημείο.

Πρώτη συνιστώσα που επιτέλεσε σημαντικό ρόλο είναι η κατάσταση στην οποία βρίσκεται αυτή τη στιγμή όλη η τεχνολογική στοίβα που αφορά ανάπτυξη DApps στο Ethereum blockchain. Βρίσκεται, συγκεκριμένα, σε ένα πολύ πρώιμο

στάδιο, όπως έχει συμβεί άλλωστε με κάθε νέα τεχνολογία. Το blockchain development, για παράδειγμα, δεν έχει αποκτήσει ακόμα δομημένα πρότυπα ανάπτυξης ή οργάνωσης του κώδικα. Παράλληλα, το μέγεθος της κοινότητας της ανάπτυξης DApps δε συγκρίνεται με κοινότητες «παραδοσιακού» προγραμματισμού. Το Ethereum έχει βρει φυσικά εφαρμογή σε πολύ μεγαλύτερο βαθμό, σε σύγκριση με το Quorum, αλλά χρειάζεται ακόμα χρόνια ωρίμανσης. Πιθανώς η στροφή των μεγάλων εταιριών σε αυτό, λόγω των πλεονεκτημάτων που προσφέρει στην ασφάλεια και τη διαφάνεια, να είναι αυτό που θα το εκτοξεύσει τα επόμενα χρόνια.

Δεύτερη συνιστώσα θα μπορούσε να χαρακτηριστεί ο «εκπαιδευτικός» χαρακτήρας της εφαρμογής. Με δεδομένο ότι αναπτύχθηκε στα πλαίσια διπλωματικής εργασίας, είχε πιο πολλή σημασία η μελέτη τεχνολογιών όπως το Quorum και το IPFS, παρά η ανάπτυξή της για επαγγελματική χρήση. Στην αρχή της εργασίας υπήρχε η υπόθεση ότι το μοντέλο ελέγχου δικαιωμάτων του Quorum θα μπορούσε να συνδεθεί μέσω web3.js ή κάποιου άλλου πακέτου (όπως το quorum.js [41]). Αυτό όμως, στην παρούσα φάση του Quorum τουλάχιστον, δεν υποστηρίζεται.

Τα παραπάνω οδήγησαν στην εφαρμογή στη φάση που βρίσκεται αυτή τη στιγμή. Προσφέρει διεπαφή ανάμεσα στο χρήστη και την (άγνωστη σε αυτόν) εξωτερική οντότητα, με ενδιάμεσο «κόμβο» το IPFS, έχει όμως αφαιρεθεί η αρχική λειτουργικότητα της αυτοματοποιημένης διαχείρισης των κόμβων του IPFS. Στην αρχική σχεδίαση είχε συμπεριληφθεί, ακόμα, ο αυτόματος (ασύγχρονος) έλεγχος της λήξης ενός λογαριασμού· να γίνεται πυροδότηση συγκεκριμένου event δηλαδή, τη στιγμή που ο λογαριασμός λήγει, μέσα από το έξυπνο συμβόλαιο. Κάτι τέτοιο όμως δεν υποστηρίζεται από τη Solidity, στη βάση της λογικής ότι τα έξυπνα συμβόλαια δε μπορούν να στείλουν συναλλαγές από μόνα τους, αλλά μόνο μετά από παρέμβαση εξωτερικού λογαριασμού.

Έτσι, αποφασίστηκε ότι η λειτουργικότητα που θα δίνεται για τον έλεγχο της λήξης του λογαριασμού θα είναι η εξής: Ο χρήστης συνδέεται στο KYC, εγγράφεται με τα στοιχεία του και δηλώνει την επιθυμητή διάρκεια ισχύος του λογαριασμού του. Εφόσον ο λογαριασμός του έχει λήξει, το σύστημα τον ενημερώνει κατά τη σύνδεσή του. Παράλληλα, προσφέρεται δυνατότητα ανάκτησης όλων των ληγμένων λογαριασμών (και όλων των λογαριασμών) μέσω του έξυπνου συμβολαίου.

Όπως έχει ήδη περιγραφεί στις Παραδοχές, κάθε χρήστης (δηλαδή κάθε λογαριασμός) μπορεί να συνδεθεί μόνο από συγκεκριμένο κόμβο του δικτύου, ώστε να διατηρείται η «1-1» σχέση λογαριασμών-κόμβων. Έτσι, οι ληγμένοι λογαριασμοί που επιστρέφονται από το συμβόλαιο, μπορούν να χρησιμοποιηθούν για να αφαιρεθούν από το δίκτυο, εφόσον αυτό ορίζουν οι προδιαγραφές του διαχειριστή. Αυτό μπορεί να γίνει στη geth κονσόλα, μέσω του `raft.removePeer(nodeId)`, όπου nodeId το enode id του κόμβου του δικτύου που υπάρχει στο static-nodes.json αρχείο κάθε κόμβου. Σε αυτό θεωρείται δεδομένο ότι έχει γίνει μέριμνα από το διαχειριστή του λογαριασμού για την αντιστοίχιση κάθε λογαριασμού σε συγκεκριμένο nodeId (κόμβο).

5.2 Έξυπνα συμβόλαια

Τα έξυπνα συμβόλαια είναι η βάση μιας αποκεντρωμένης εφαρμογής που δουλεύει στο Ethereum Network ή κάποια επέκταση αυτού. Ο κώδικάς τους μεταφράζεται μέσω του Solidity compiler σε bytecode που μπορεί να εκτελεστεί στο Ethereum Virtual Machine.

Στην εφαρμογή αυτή, χρησιμοποιήθηκε ένα μόνο συμβόλαιο, το UserStorage, η παρουσίαση του οποίου ακολουθεί.

5.2.1 Αναλυτική παρουσίαση συμβολαίου UserStorage

Στην συνέχεια γίνεται αναλυτική παρουσίαση του κώδικα του συμβολαίου `UserStorage`.

```
pragma solidity ^0.5.3;
```

Δήλωση της έκδοσης της Solidity.

```
uint totalUsers;
```

Μεταβλητή χρήσιμη για την ανάκτηση των λογαριασμών του συμβολαίου.

```
mapping (uint => address) addressIndexing;
mapping (address => User) users;
```

Τα 2 mappings για τις διευθύνσεις και τους χρήστες. Το mapping αποτελεί μια δομή δεδομένων στη Solidity που μπορεί κανείς να τη σκεφτεί σαν το HashMap, της Java για παράδειγμα. Το addressIndexing δηλαδή για έναν δοθέντα ακέραιο, επιστρέφει μια διεύθυνση, και το users για μια διεύθυνση επιστρέφει έναν User.

```
event UserCreated(address indexed user, uint accountDuration);
event UserAccessed(address indexed user);
event UserExists(address indexed user);
event DocumentStored(address indexed user);
event UserApproved(address indexed user, uint accounDuration);
event UserExpired(address indexed user);
```

Τα events του συμβολαίου που στέλνονται από τις διάφορες μεθόδους του. Είναι χρήσιμα τόσο για το debugging της εφαρμογής όσο και για την παρακολούθησή της από τη μεριά της JavaScript, μέσω του web3.js. Αποτελούν στην πραγματικότητα έναν από τους ελάχιστους τρόπους να γίνει debugging του Solidity κώδικα. Μπορούν να σταλούν μόνο από μεθόδους που είναι payable (default) και όχι view, επειδή απαιτούν gas για να γίνει αυτό.

```
struct User {
```

```
  address addr;
  uint amount;
  uint accountExpiration; // EPOCH IN MS
  uint accountDuration;   // IN DAYS
  string userId;
  string docHash; // ipfs hash of the user's document
  bool hasDocument;
  bool approved;
  bool expired;
  bool stored;
}
```

Το struct στη Solidity μπορεί κανείς να το σκεφτεί σαν το struct της C. Κρατάει τις πληροφορίες για το χρήστη της εφαρμογής.

```
function createUser(address _address, uint accountDuration,
uint callTimestamp, string memory userId) public {
  require(!userExists(_address), "User already exists");

  addressIndexing[totalUsers] = _address;
  totalUsers++;

  users[_address].addr = msg.sender;
  users[_address].accountDuration = accountDuration;
  users[_address].accountExpiration = callTimestamp + ac-
countDuration * 24 * 3600 * 1000; // converted days to mili-
seconds
  users[_address].userId = userId;
  users[_address].hasDocument = false;
  users[_address].approved = false;
  users[_address].stored = true;
  emit UserCreated(_address, accountDuration);
}
```

Δημιουργεί το χρήστη της εφαρμογής, εφόσον δεν υπάρχει ήδη, και στέλνει ένα UserCreated event εάν επιτύχει.

```
function getUserDuration(address _address) public view
returns(uint) {
  return users[_address].accountDuration;
}
```

Επιστρέφει τη διάρκεια του χρήστη. Η λειτουργικότητά της καλύπτεται από την getUserInfo παρακάτω αλλά υπάρχει ξεχωριστά γιατί απαιτεί λιγότερους υπολογιστικούς πόρους. Σε εφαρμογές που τρέχουν στο EVM, σε μεγάλη έκταση οι μικρές διαφορές κάνουν τη διαφορά στην απόδοση.

```
function storeDocument(address _address, string memory ipf-
sHash) public {
  users[_address].docHash = ipfsHash;
  users[_address].hasDocument = true;
  users[_address].approved = false; // new ID, new request
  emit DocumentStored(_address);
}
```

Αποθηκεύει το ipfsHash του ανεβασμένου εγγράφου στο struct του χρήστη.

```
function getUserExpiration(address _address) public re-
turns(uint) {
  require(userExists(_address));
  uint userExpiration = users[_address].accountExpiration;
  emit UserAccessed(_address);
  return userExpiration;
}
```

Ομοίως με το getUserDuration παραπάνω.

```
function isUserApproved(address _address) public view returns
(bool) {
  return users[_address].approved;
}

function isExpired(address _address) public view returns
(bool) {
  return users[_address].expired;
}

function userExists(address _address) public view returns
(bool) {
  return users[_address].stored;
}

function userHasDocument(address _address) public view returns
(bool) {
  return users[_address].hasDocument;
}

function verifyDocument(address _address) public {
  users[_address].approved = true;
}

function setExpired(address _address) public {
```

```
  users[_address].expired = true;
  emit UserExpired(_address);
}
```

Σειρά μεθόδων που καλύπτονται και από την getUserInfo αλλά υπάρχουν για το λόγο που περιγράφηκε παραπάνω.

```
function approveUser(address _address, uint callTimestamp,
uint duration) public {
  require(userExists(_address) && userHasDocument(_address));
  users[_address].approved = true;
  users[_address].accountDuration = duration; //renewed
  users[_address].accountExpiration = convertDura-
tion(callTimestamp, duration);
  emit UserApproved(_address, duration);
}
```

Σε περίπτωση επιβεβαίωσης του χρήστη από τη μεριά της JavaScript, αποθηκεύεται εδώ η αντίστοιχη ένδειξη στο struct.

```
function getUserInfo(address _address) public view re-
turns(string memory, uint, bool, string memory, bool, uint) {
  // emit UserAccessed(_address);
  return (users[_address].userId,
          users[_address].accountExpiration,
          users[_address].approved,
          users[_address].docHash,
          users[_address].hasDocument,
          users[_address].accountDuration);
}
```

Η getUserInfo επιστρέφει τα περισσότερα στοιχεία του χρήστη, όταν η εφαρμογή τα χρειάζεται όλα μαζί.

```
function getExpiredUsers() public returns (address[] memory) {
  address[] memory expiredUsers = new address[](totalUsers);
// max possible size
  for (uint i = 0; i < totalUsers; i++) {
    if (users[addressIndexing[i]].expired) {
      expiredUsers[i] = users[addressIndexing[i]].addr;
    }
  }
  return expiredUsers;
}

function getAllUsers() public returns (address[] memory) {
```

```
  address[] memory returned = new address[](totalUsers);
// memory array cannot be dynamic in solidity
  for (uint i = 0; i < totalUsers; i++) {
    returned[i] = users[addressIndexing[i]].addr;
  }
  return returned;
}
```

Οι δύο αυτές μέθοδοι επιστρέφουν το σύνολο των χρηστών και τους ληγμένους λογαριασμούς αντίστοιχα. Εδώ χρησιμοποιείται μια chaining τεχνική όσον αφορά τα 2 mappings, όπου η επιστρεφόμενη τιμή του ενός (address) αποτελεί index του άλλου (user). Επειδή η Solidity δεν υποστηρίζει επιστροφή δυναμικών τύπων δεδομένων, αναγκαστικά δηλώνονται πίνακες από διευθύνσεις με μέγεθος όσο το πλήθος όλων των λογαριασμών. Στη δεύτερη περίπτωση μάλιστα, όπου το σύνολο των ληγμένων λογαριασμών είναι κατά κανόνα μικρότερο από το σύνολο των λογαριασμών, επιστρέφεται ένας πίνακας με λογαριασμούς μορφής 0x000… και 0x123…, και είναι ευθύνη του JavaScript κώδικα να τους διαχωρίσει.

```
function convertDuration(uint callTimestamp, uint duration-
InDays) private view returns (uint)  {
  return callTimestamp + durationInDays * 24 * 3600 * 1000;
}
```

Η διάρκεια του λογαριασμού δηλώνεται σε μέρες αλλά το callTimestamp (timestamp της κλήσης της μεθόδου) σε δευτερόλεπτα, και χρειάζεται μετατροπή σε milliseconds (unix epoch) για ανανέωση του expirationDate.

```
function () external payable {
  // empty
}
```

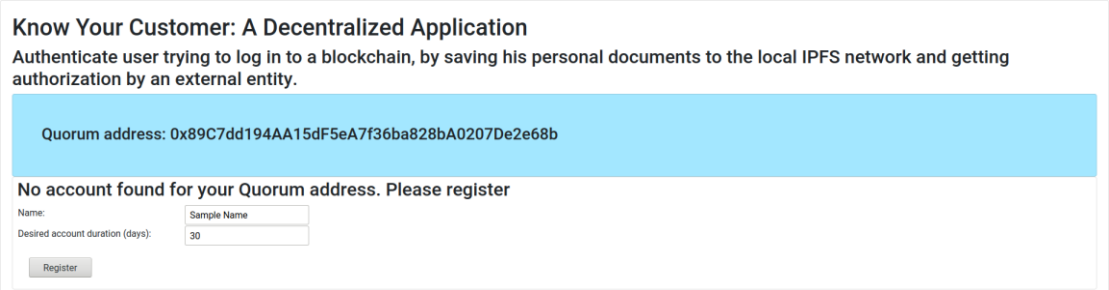
Η fallback συνάρτηση του συμβολαίου. Κάθε συμβόλαιο πρέπει να έχει μια τέτοια για να μπορεί να απαντάει σε κλήσεις που δεν περιλαμβάνουν δεδομένα πέρα του πλήθους από tokens που αποστέλλονται. Χρειάζεται δηλαδή για να μπορεί να δεχτεί tokens, π.χ. Ether.

6

Επίδειξη λειτουργικότητας εφαρμογής

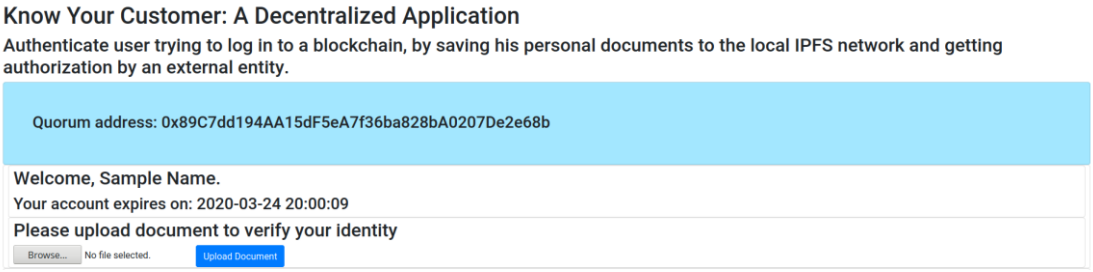
Την παρακάτω λειτουργικότητα μπορεί κανείς να τη δει και σε βίντεο στο https://www.youtube.com/watch?v=qMGky1kyhSQ.

Αφού η εφαρμογή ξεκινήσει, με οποιονδήποτε από τους τρόπους που περιγράφονται στο Παράρτημα Ι, και ο χρήστης έχει συνδεθεί μέσω του Metamask με το τοπικό blockchain μέσω RPC, μεταβαίνει στο http://localhost:3000 για να δει τη σελίδα του Know Your Customer (KYC).



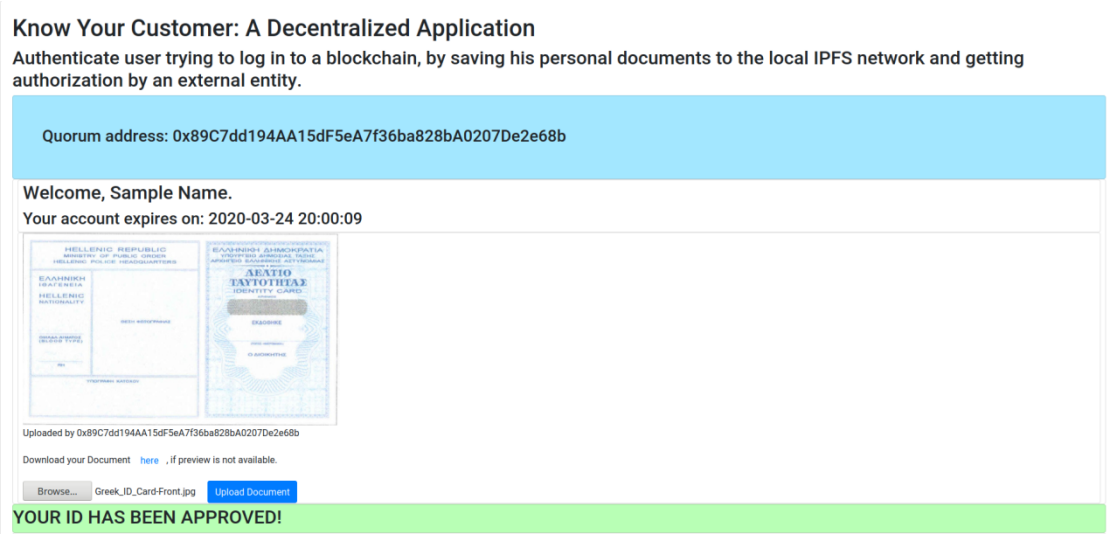
Εικόνα 6-1 Οθόνη πριν την εγγραφή στην πλατφόρμα

Η αναμενόμενη εικόνα της πρώτης σύνδεσης με το KYC είναι η παραπάνω, στην οποία ο χρήστης δεν έχει δημιουργήσει account στο KYC. Για να δημιουργήσει account πρέπει να δηλώσει ένα username και χρονικό διάστημα (σε μέρες) για το οποίο θα επιθυμούσε πρόσβαση στο (Quorum) blockchain μέσω του KYC.



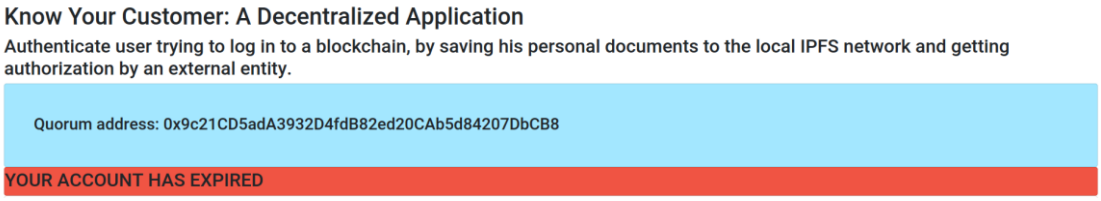
Εικόνα 6-2 Οθόνη μετά την εισαγωγή των στοιχείων και πριν την υποβολή εγγράφου

Μετά την εισαγωγή των στοιχείων του, μεταβαίνει στην παραπάνω οθόνη, στην οποία βλέπει μήνυμα για να ανεβάσει έγγραφο ταυτοποίησης και να επιβεβαιωθεί η σύνδεσή του. Με το ανέβασμα εγγράφου στο IPFS, στέλνεται HTTP POST request στο external authority για να λάβει επιβεβαίωση. Το external authority δέχεται ένα request με το όνομα του χρήστη που θέλει να συνδεθεί, το ipfs hash του εγγράφου του, την επιθυμητή διάρκεια σύνδεσης και τη διεύθυνσή του στο blockchain. Όταν στείλει response (το οποίο περιλαμβάνει ένδειξη για το εάν ταυτοποιήθηκε ο χρήστης και έγινε δεκτό το αίτημά του), ανανεώνεται η σελίδα για να δείξει ότι ο χρήστης επιβεβαιώθηκε, καθώς και πότε λήγει ο λογαριασμός του.



Εικόνα 6-3 Οθόνη επιβεβαιωμένης σύνδεσης χρήστη

Όταν και εάν ο χρήστης προσπαθήσει να συνδεθεί στο KYC μετά το πέρας της λήξης της σύνδεσής του, η σελίδα τον ειδοποιεί ότι η σύνδεσή του έχει λήξει και ο λογαριασμός του μπαίνει σε λίστα με expired λογαριασμούς. Τη λίστα αυτή μπορεί να τη δει κάποιος διαχειριστής του Quorum δικτύου μέσω της αντίστοιχης μεθόδου του έξυπνου συμβολαίου και να πράξει αναλόγως για τους λογαριασμούς αυτούς.



Εικόνα 6-4 Οθόνη ληγμένης σύνδεσης

Λόγω της παραδοχής που έγινε ότι κάθε λογαριασμός πρέπει να μπαίνει στο blockchain από έναν και μόνο κόμβο, υπάρχει μια «1-1» σχέση ανάμεσα σε λογαριασμούς και κόμβους. Επομένως, μπορεί να χρησιμοποιηθεί το permissioning model του Quorum, και συγκεκριμένα να διαγράψει τους λογαριασμούς που έχουν λήξει, ή οτιδήποτε άλλο κριθεί απαραίτητο σε πραγματική εφαρμογή της εργασίας αυτής.

Αυτό μπορεί να γίνει στη geth κονσόλα, μέσω του `raft.removePeer(nodeId)`, όπου nodeId το enode id του κόμβου του δικτύου που υπάρχει στο static-nodes.json αρχείο κάθε κόμβου.

Σε όσα σημεία ο χρήστης βλέπει prompt από το metamask, χρειάζεται να επικυρώσει συναλλαγή η οποία θα αλλάξει το state του blockchain.

7

Επίλογος

7.1 Σύνοψη και συμπεράσματα

Κατά την ανάπτυξη της παρούσας διπλωματικής, μελετήθηκε εκτενώς ο τρόπος λειτουργίας τεχνολογιών οι οποίες τα επόμενα χρόνια αναμένεται να βρεθούν σε άνθηση και να υπεισέλθουν στην υλοποίηση πιο mainstream εφαρμογών. Από τις τεχνολογίες αυτές, το Ethereum έχει δείξει τη χρησιμότητά του και έχει βρει εφαρμογή, χάρη στα έξυπνα συμβόλαιά του σε αρκετούς τομείς. Παραδείγματα αποτελούν η αποκεντρωμένη Οικονομία, εφαρμογές στην πολιτική για εκλογές, στην υγεία και όπου χρειάζεται διαφάνεια και ασφάλεια. [42]

Σκοπός ήταν λοιπόν να εξακριβωθεί κατά πόσο το Quorum μπορεί να συμπληρώσει τα κενά του Ethereum προσφέροντας ένα παραπάνω επίπεδο ασφάλειας. Το Quorum, όπως έχει αναλυθεί στο Κεφάλαιο 2, προσφέρει ασφάλεια σε επίπεδο συναλλαγών και σε επίπεδο κόμβου. Η ασφάλεια σε επίπεδο κόμβου είναι αυτή που αφορά τη δημιουργία ενός KYC συστήματος, το οποίο δηλαδή θα ελέγχει την είσοδο του χρήστη και τα δικαιώματά του στο blockchain.

Όπως αναλύθηκε εκτενώς στο Κεφάλαιο 5, προσπάθεια έγινε να γίνει όσο το δυνατόν πιο δυναμικός ο έλεγχος των δικαιωμάτων του χρήστη· να συνδεθεί δηλαδή το web κομμάτι της εφαρμογής, η –γραμμένη σε JavaScript- σελίδα, με το μοντέλο ελέγχου δικαιωμάτων (Permissioning Model) του Quorum. Τα παραπάνω οδηγούν στο συμπέρασμα ότι η διαχείριση των λογαριασμών τελικά εναπόκειται στην παραμετροποίηση του συστήματος με τέτοιο τρόπο ώστε να ικανοποιηθούν οι παραδοχές που έχουν γίνει και να βρεθεί “workaround” στα προβλήματα που αναλύθηκαν.

Σε τελική ανάλυση, η διπλωματική αυτή είναι μία από τις πρώτες προσπάθειες σύνδεσης του Quorum blockchain με το IPFS σε μια JavaScript εφαρμογή. Στο τεχνικό κομμάτι προσφέρει λειτουργικότητες που μπορούν να φανούν χρήσιμες σε μια επαγγελματική υλοποίηση ενός Quorum blockchain, ενώ στο ερευνητικό κομμάτι παρουσιάζει τις ελλείψεις και τις δυσκολίες στη σύνδεση των παραπάνω τεχνολογιών. Στην τωρινή της φάση, ωστόσο, αποτελεί περισσότερο ένα proof of concept για ένα Know Your Customer σύστημα που θα δουλεύει αποκλειστικά on chain, και όχι μια ολοκληρωμένη λύση για περιβάλλον παραγωγής.

7.2 Μελλοντικές επεκτάσεις

Το βασικό κομμάτι που λείπει από την εφαρμογή είναι η σύνδεση σε κάποιο πραγματικό περιβάλλον παραγωγής που θα λειτουργεί σε Quorum blockchain. Σε ένα τέτοιο περιβάλλον θα φανεί το βασικό πρόβλημα της στοίβας τεχνολογιών που χρησιμοποιήθηκαν: η αδυναμία σύνδεσης του web κομματιού με την κονσόλα διαχείρισης του Quorum blockchain.

Η επέκταση της εφαρμογής στην περίπτωση αυτή θα περιλάμβανε το απαραίτητο setup του blockchain. Συγκεκριμένα, χρειάζεται κάποιος διαχειριστής:

1. Να δημιουργήσει το δίκτυο, δηλαδή τους κόμβους του δικτύου
2. Να δημιουργήσει αντιστοιχία μεταξύ λογαριασμών και κόμβων
3. Να στήσει μηχανισμό ανάκτησης των λογαριασμών και των ληγμένων λογαριασμών της εφαρμογής, πιθανώς μέσω ενός άλλου έξυπνου συμβολαίου, η υλοποίηση του οποίου θα εξαρτιόταν από τις ανάγκες της εκάστοτε εφαρμογής.

Όσον αφορά την εφαρμογή, προσωπική μου άποψη είναι ότι πιθανή βελτίωση θα ήταν να χρησιμοποιηθεί πιο component-based φιλοσοφία, κάτι που υποστηρίζει (και ενθαρρύνει) η ReactJS. Αυτό θα βελτίωνε την επεκτασιμότητά της και θα διευκόλυνε τον επόμενο developer. Ειδικά σε περίπτωση επέκτασής της για να περιλαμβάνει περισσότερες οθόνες, κάποιο πιθανό dashboard για τη διαχείριση των στοιχείων του χρήστη, ή ό,τι άλλο θα πρόσταζε μια «πραγματική» εφαρμογή της, η παραπάνω πρόταση θα ήταν απαραίτητη.

8

Βιβλιογραφία

[1] ‘Web 3.0 - The Current Web Revolution’, *digitales*. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://www.digitales-blog.com/blog/web-30-the-current-web-revolution. [Ημερομηνία πρόσβασης: 01-Φεβρουαρίου-2020].

[2] ‘The Journey to Web 3 - Web3 Foundation - Medium’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://medium.com/web3foundation/the-journey-to-web-3-3ead84261000.

[3] ‘Web3 - The Decentralized Web’, *BlockchainHub*. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://blockchainhub.net/web3-decentralized-web/.

[4] J. Eberspächer και Rüdiger. Schollmeier, ‘First and Second Generation of Peer-to-Peer Systems’, *Lect. Notes Comput. Sci.*, τ. 3485, σσ. 35–56, Ιανουαρίου 2005.

[5] R. Schollmeier, ‘A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications’, στο *Proc. of the First International Conference on Peer-to-Peer Computing*, 2001.

[6] ‘Elliptic Curve Cryptography (ECC)’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.certicom.com/content/certicom/en/ecc.html. [Ημερομηνία πρόσβασης: 01-Φεβρουαρίου-2020].

[7] ‘What is the math behind elliptic curve cryptography?’ [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://hackernoon.com/what-is-the-math-behind-elliptic-curve-cryptography-f61b25253da3. [Ημερομηνία πρόσβασης: 01-Φεβρουαρίου-2020].

[8] ‘A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography’, *The Cloudflare Blog*, 24-Οκτωβρίου-2013. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/. [Ημερομηνία πρόσβασης: 01-Φεβρουαρίου-2020].

[9] S. Nakamoto, ‘Bitcoin: A Peer-to-Peer Electronic Cash System’.

[10] J. Frankenfield, ‘Proof of Stake (PoS)’, *Investopedia*. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.investopedia.com/terms/p/proof-stake-pos.asp. [Ημερομηνία πρόσβασης: 02-Φεβρουαρίου-2020].

[11] D. G. Wood, ‘ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER’.

[12] ‘Ethereum White Paper’, *GitHub*. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://github.com/ethereum/wiki/wiki/white-paper. [Ημερομηνία πρόσβασης: 02-Φεβρουαρίου-2020].

[13] C. Tardi, ‘Gwei Definition’, *Investopedia*. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.investopedia.com/terms/g/gwei-ethereum.asp. [Ημερομηνία πρόσβασης: 02-Φεβρουαρίου-2020].

[14] ‘jpmorganchase/quorum-examples’, *GitHub*. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://github.com/jpmorganchase/quorum-examples. [Ημερομηνία πρόσβασης: 15-Φεβρουαρίου-2020].
[15] ‘Home · synechron-finlabs/quorum-maker Wiki · GitHub’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://github.com/synechron-finlabs/quorum-maker/wiki. [Ημερομηνία πρόσβασης: 15-Φεβρουαρίου-2020].
[16] Z. Shi, *Szkered/quorum-raft-cluster*. 2020.
[17] L. Zhou, ‘The complete Truffle suite on Docker (Truffle + Ganache + Drizzle)’, *Medium*, 28-Ιουνίου-2019. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://medium.com/@lzhou1110/the-complete-truffle-suite-on-docker-truffle-ganache-drizzle-47ab18b1ec83. [Ημερομηνία πρόσβασης: 16-Φεβρουαρίου-2020].
[18] ‘Go Ethereum’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://geth.ethereum.org/. [Ημερομηνία πρόσβασης: 09-Φεβρουαρίου-2020].
[19] ‘web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://web3js.readthedocs.io/en/v1.2.6/. [Ημερομηνία πρόσβασης: 09-Φεβρουαρίου-2020].
[20] ‘Solidity — Solidity 0.5.12 documentation’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://solidity.readthedocs.io/en/v0.5.12/. [Ημερομηνία πρόσβασης: 09-Φεβρουαρίου-2020].
[21] ‘Truffle Ethereum Suite’, *npm*. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.npmjs.com/package/truffle.
[22] ‘react | Boxes’, *Truffle Suite*. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://trufflesuite.com/boxes/react. [Ημερομηνία πρόσβασης: 08-Φεβρουαρίου-2020].
[23] ‘EthereumJS’, *EthereumJS*. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://ethereumjs.github.io/. [Ημερομηνία πρόσβασης: 08-Φεβρουαρίου-2020].
[24] ‘MetaMask’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://metamask.io/. [Ημερομηνία πρόσβασης: 08-Φεβρουαρίου-2020].
[25] ‘State of the DApps — Blockchain Apps for Ethereum, Steem, EOS, and More’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.stateofthedapps.com/help/metamask. [Ημερομηνία πρόσβασης: 08-Φεβρουαρίου-2020].
[26] ‘Latest Ecosystem/Use Cases and Applications topics - discuss.ipfs.io’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://discuss.ipfs.io/c/ecosystem/applications-of-ipfs. [Ημερομηνία πρόσβασης: 22-Φεβρουαρίου-2020].
[27] ‘IPFS Documentation’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://docs.ipfs.io/. [Ημερομηνία πρόσβασης: 09-Φεβρουαρίου-2020].
[28] ‘IPFS Distributions’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://dist.ipfs.io/#go-ipfs. [Ημερομηνία πρόσβασης: 09-Φεβρουαρίου-2020].
[29] ‘GitHub - ipfs/js-ipfs-http-client: A client library for the IPFS HTTP API, implemented in JavaScript.’ [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://github.com/ipfs/js-ipfs-http-client. [Ημερομηνία πρόσβασης: 09-Φεβρουαρίου-2020].
[30] Node.js, ‘What is npm?’, *Node.js*. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/. [Ημερομηνία πρόσβασης: 08-Φεβρουαρίου-2020].
[31] ‘Node.js’s npm Is Now The Largest Package Registry in the World - Slashdot’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο:

https://developers.slashdot.org/story/17/01/14/0222245/nodejss-npm-is-now-the-largest-package-registry-in-the-world. [Ημερομηνία πρόσβασης: 08-Φεβρουαρίου-2020].
[32] Node.js, 'Node.js', *Node.js*. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://nodejs.org/en/. [Ημερομηνία πρόσβασης: 08-Φεβρουαρίου-2020].
[33] 'Docker overview', *Docker Documentation*, 14-Φεβρουαρίου-2020. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://docs.docker.com/engine/docker-overview/. [Ημερομηνία πρόσβασης: 16-Φεβρουαρίου-2020].
[34] 'Overview of docker-compose CLI', *Docker Documentation*, 14-Φεβρουαρίου-2020. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://docs.docker.com/compose/reference/overview/. [Ημερομηνία πρόσβασης: 16-Φεβρουαρίου-2020].
[35] 'PYPL PopularitY of Programming Language index'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://pypl.github.io/PYPL.html. [Ημερομηνία πρόσβασης: 09-Φεβρουαρίου-2020].
[36] 'How JavaScript Works'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://fireship.io/courses/javascript/intro-how-js-works/. [Ημερομηνία πρόσβασης: 09-Φεβρουαρίου-2020].
[37] 'JavaScript - Overview - Tutorialspoint'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.tutorialspoint.com/javascript/javascript_overview.htm. [Ημερομηνία πρόσβασης: 09-Φεβρουαρίου-2020].
[38] 'Getting Started · React Native'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://facebook.github.io/react-native/. [Ημερομηνία πρόσβασης: 16-Φεβρουαρίου-2020].
[39] 'Getting Started – React'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://reactjs.org/docs/getting-started.html. [Ημερομηνία πρόσβασης: 09-Φεβρουαρίου-2020].
[40] Mark Otto, Jacob Thornton, and Bootstrap, 'Bootstrap'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://getbootstrap.com/docs/4.0/getting-started/introduction/. [Ημερομηνία πρόσβασης: 09-Φεβρουαρίου-2020].
[41] *jpmorganchase/quorum.js*. J.P. Morgan, 2020.
[42] N. Dominguez, 'These are 11 Real World Use Cases for Ethereum', *CoinDiligent*, 27-Ιανουαρίου-2020. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://coindiligent.com/ethereum-real-world-use. [Ημερομηνία πρόσβασης: 16-Φεβρουαρίου-2020].
[43] 'Cross-Origin Resource Sharing (CORS)', *MDN Web Docs*. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. [Ημερομηνία πρόσβασης: 15-Φεβρουαρίου-2020].

Παράρτημα Ι: Εγχειρίδιο χρήσης

Α Εγκατάσταση

Λόγω του μεγάλου πλήθους των διαφορετικών εργαλείων και τεχνολογιών που χρησιμοποιούνται και τα οποία πρέπει να μπορού να συνεργάζονται μεταξύ τους, η εγκατάστασή τους έχει αρκετά βήματα.

1) Εγκατάσταση ιδιωτικού δικτύου IPFS

Ακολουθούν οδηγίες για το στήσιμο private IPFS network με libp2p πρωτόκολλο επικοινωνίας. Δοκιμασμένο σε linux με:

```
4.19.0-041900-generic kernel
go version go1.13.3 linux/amd64
ipfs version 0.4.22
```

Σε πρώτη φάση πρέπει να δημιουργηθεί ένα swarm key, τρέχοντας το παρακάτω στον bootstrap (κύριο) κόμβο του δικτύου:

```
go get -u github.com/Kubuxu/go-ipfs-swarm-key-gen/ipfs-swarm-key-gen
ipfs-swarm-key-gen & > ~/.ipfs/swarm.key
```

Το παραπάνω ipfs-swarm-key-gen module αποθηκεύεται στο home path της go, οπότε πρέπει να μπει το directory του στο PATH.

Το παραγόμενο κλειδί πρέπει να περαστεί και σε όλους του υπόλοιπους κόμβους, ως `~/.ipfs/swarm.key`, ή ως `$IPFS_PATH/swarm.key` εάν αυτό είναι custom.

Το παρακάτω σβήνει τα default public nodes από τη λίστα του IPFS. Απαραίτητο για να γίνει private το δίκτυο.

```
ipfs bootstrap rm --all
```

Στη συνέχεια προσθέτουμε *σε κάθε* κόμβο του δικτύου τη *local* IP διεύθυνση του bootstrap node του δικτύου με την εξής εντολή:

```
ipfs bootstrap add \
/ip4/<IP_ADDRESS>/tcp/4001/ipfs/<IPFS_ID>
```

όπου IP_ADDRESS η _local_ IP και το IPFS_ID προκύπτει από το αντίστοιχο value του JSON που επιστρέφει η `ipfs id`.

Στη συνέχεια χρειάζεται να τεθεί η μεταβλητή συστήματος LIBP2P_FORCE_PNET σε 1 για να ενεργοποιηθεί το private mode, το οποίο είναι προς το παρόν experimental feature του go-ipfs.

```
export LIBP2P_FORCE_PNET=1;
```

Για το testing του δικτύου, μένει μόνο να ξεκινήσει ο ipfs daemon σε όλους τους κόμβους. Αυτό γίνεται με `ipfs daemon` (βλ. παρακάτω για το startup ως service). Έπειτα, η `ipfs swarm peers` σε οποιονδήποτε κόμβο θα πρέπει να δείχνει τις διευθύνσεις όλων των υπολοίπων (και καμία παραπάνω).

Για δοκιμή του παρόντος setup, φτιάξτε τέλος ένα αρχείο, ανεβάστε το στο ipfs και ελέγξτε εάν είναι διαθέσιμο στους υπόλοιπους κόμβους.

```
touch /tmp/test.txt
echo "some text" > /tmp/test.txt
ipfs add /tmp/test.txt
```

Η τελευταία εντολή θα επιστρέψει ένα HASH, το οποίο μπορεί να χρησιμοποιηθεί για ανάκτηση του αρχείου από άλλο κόμβο με `ipfs cat HASH`.

Τρέξτε την τελευταία εντολή και από κόμβο εξωτερικό του τοπικού δικτύου για απόδειξη ότι δεν ανέβηκε στο (global) δίκτυο gateway.ipfs.io/ipfs/HASH (=ότι το δίκτυο είναι όντως private). Στο URL αυτό βέβαια δε θα έπρεπε να φαίνεται το αρχείο ούτε από κόμβο του εσωτερικού δικτύου, αφού by default θα ψάξει στο global IPFS network.

Όσον αφορά τη σχετική αυτοματοποίηση της παραπάνω διαδικασίας, για να τρέξει ο daemon ως background service, αρκεί να δημιουργηθεί ένα αρχείο `/etc/systemd/system/ipfs_daemon.service` ως εξής:

```
[Unit]
Description=IPFS Daemon
After=syslog.target network.target remote-fs.target nss-
lookup.target
[Service]
Type=simple
ExecStart=/usr/local/bin/ipfs daemon --enable-namesys-
pubsub
User=USER
[Install]
WantedBy=multi-user.target
```

όπου USER ο χρήστης στο homedir του οποίου βρίσκεται το .ipfs directory.
Για το χειρισμό του νέου service:

```
sudo systemctl daemon-reload
sudo systemctl enable ipfs_daemon
sudo systemctl start ipfs_daemon
sudo systemctl status ipfs_daemon
```

Σημείωση: οι σύγχρονοι browsers έχουν από προεπιλογή απενεργοποιημένη τη δυνατότητα αποστολής CORS requests [43], δηλαδή την αποστολή request στο ίδιο host. Αυτό αποτελεί πρόβλημα τόσο στην περίπτωση επικοινωνίας με το IPFS (όταν ο localhost είναι ο bootstrap κόμβος), όσο και στην περίπτωση επικοινωνίας με την «εξωτερική οντότητα», που στην παρούσα φάση είναι άλλη μία εφαρμογή που ακούει σε άλλο port του localhost. Η πιο σίγουρη λύση αυτού του προβλήματος είναι η εγκατάσταση κάποιου browser addon που απενεργοποιεί την προαναφερθείσα προστασία.

2) Δημιουργία ιδιωτικού Quorum δικτύου

Η δημιουργία ενός Quorum δικτύου δεν είναι απλή, και απαιτεί γνώση των εργαλείων που απαρτίζουν το Quorum, όπως το raft, το tessera κλπ. Εφόσον στην παρούσα φάση πρόκειται για μια εκπαιδευτική εφαρμογή, χωρίς πολλές απαιτήσεις για ιδιωτικότητα, μπορεί να χρησιμοποιηθεί το quorum-maker εργαλείο που παρουσιάζεται στο 2ο Κεφάλαιο ή κατευθείαν το docker deployment του 7 nodes example.

Σε περίπτωση που απαιτείται η δημιουργία ενός δικτύου από εικονικές μηχανές ή Docker instances, και τα 2 παραπάνω εργαλεία καλύπτουν την ανάγκη αυτή. Είτε με την πρώτη επιλογή στο setup του quorum-maker, είτε με το ανέβασμα του docker-compose του 7nodes, μέσα σε λίγα λεπτά μπορεί να δημιουργηθεί ένα τοπικό δίκτυο που ακούει στην πόρτα 22000 (προεπιλογή) μέσω RPC.

Σε περίπτωση που θέλει κανείς να ελέγξει την εφαρμογή σε ένα πιο «πραγματικό» δίκτυο, καλύπτεται αρκετά εύκολα από το quorum-maker. Όπως περιγράφεται και στο documentation του εργαλείου αυτού [15], το μόνο που χρειάζεται είναι η IP διεύθυνση κάθε κόμβου του δικτύου και το script αναλαμβάνει όλα τα υπόλοιπα, όπως για παράδειγμα την παραγωγή του genesis.json αρχείου που περιγράφει κάθε κόμβο πριν την εκκίνησή του, ή την εισαγωγή του στο static-nodes.json και στο permissioned-nodes.json.

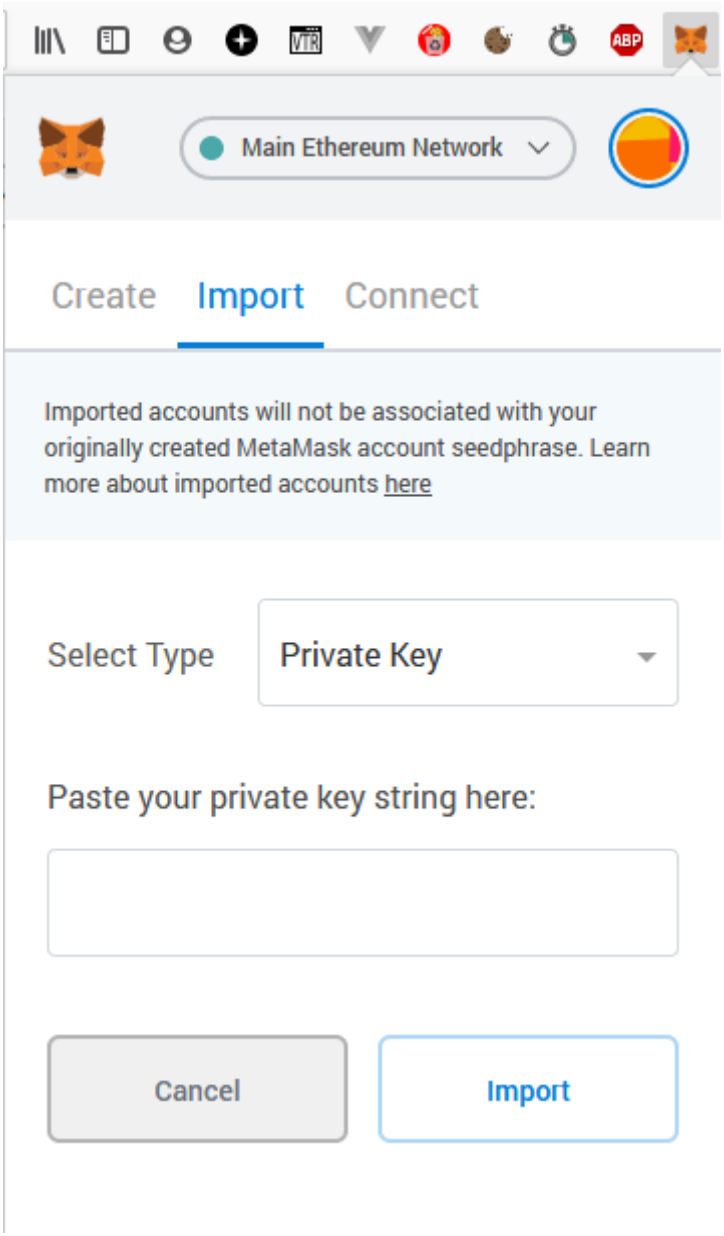
Εικόνα 0-1 Χρήση του Quorum-maker για νέο δίκτυο

3) Metamask Setup

Το Metamask είναι ένα addon τόσο για τον Firefox όσο και για όλους τους Chromium browsers (Opera, Brave, Edge, κλπ) και η σημαντικότητά του έχει εξηγηθεί σε προηγούμενο κεφάλαιο. Η εγκατάστασή του γίνεται από το αντίστοιχο addon page του browser. Η έκδοσή του που χρησιμοποιήθηκε για για την παρούσα εργασία είναι η 7.7.4.

Το Metamask κρατάει ένα σύνολο από Ethereum/Quorum λογαριασμούς, οι οποίοι προστατεύονται από ένα password που ο χρήστης θέτει. Για τη χρήση της εφαρμογής απαιτείται να γίνει εισαγωγή τουλάχιστον ενός λογαριασμού που ο χρήστης θα χρησιμοποιήσει για το KYC σύστημα. Χάριν ευκολίας ο λογαριασμός αυτός μπορεί να είναι (στην παρακάτω αναλυόμενη περίπτωση του Docker deployment), ένας από αυτούς τα κλειδιά των οποίων φαίνονται στο ganache_with_accounts.sh script, ή ένας από τους λογαριασμούς του Ganache σε περίπτωση deployment σε Ganache δίκτυο.

Σε περίπτωση που η εφαρμογή τρέχει σε Quorum δίκτυο, η εισαγωγή του λογαριασμού μπορεί να γίνει είτε με το private key του λογαριασμού (εφόσον είναι γνωστό) είτε με την εισαγωγή του JSON που «περιγράφει» το λογαριασμό. Συγκεκριμένα στο παράδειγμα που κάνει χρήση του quorum-examples/7nodes (βλ. παρακάτω), εισαγωγή του N-οστού account γίνεται με το keyN από τα διαθέσιμα κλειδιά. Πρώτα πρέπει να ανοίξει το Metamask σε δικό του tab (επιλογή “Expand View”), στη συνέχεια να γίνει επιλογή του «JSON File», και τέλος επιλογή του keyN (που αποτελεί JSON File με τις πληροφορίες του λογαριασμού).



Εικόνα 0-2 Το μενού εισαγωγής λογαριασμού

Στο εικονιζόμενο μενού γίνεται εισαγωγή του private key του Ethereum account. Στην περίπτωση Ethereum blockchain που τρέχει μέσω του Ganache, το private key είναι γνωστό μέσω του GUI του Ganache. Στην περίπτωση Quorum blockchain όπου δεν είναι γνωστό το private key (όπως στην περίπτωση του 7nodes quorum example), μπορεί να γίνει εισαγωγή του account μέσω του json αρχείου που είναι αποθηκευμένο στο εικονικό μηχάνημα όπου τρέχει το Quorum.

Στη συνέχεια μένει η σύνδεση με οποιοδήποτε blockchain τρέχει η εφαρμογή. Το μενού είναι αυτό που εικονίζεται παρακάτω, και το port μέσω του οποίου θα γίνει η σύνδεση είναι κάποιο από αυτά που γράφονται παρακάτω, ανάλογα με τη χρήση της εφαρμογής.

Β Χρήση εφαρμογής

Για τη χρήση της εφαρμογής υποστηρίζονται οι εξής 2 τρόποι:

1) Docker deployment

Προαπαιτούμενα:

- unix λειτουργικό σύστημα (δοκιμασμένο σε Ubuntu 18.04 LTS)
- εγκατεστημένο git
- εγκατεστημένο docker και docker-compose.

Πρόκειται για τον πιο εύκολο τρόπο για να γίνει deploy η εφαρμογή. Στην περίπτωση αυτή επικοινωνεί με ένα private Ethereum network (και όχι Quorum), το οποίο το διαχειρίζεται ένα custom docker image που περιλαμβάνει το ganache-cli tool και ένα script που δίνει τη δυνατότητα δημιουργίας συγκεκριμένων accounts στο Ethereum network.

Με δεδομένο ότι ο χρήστης έχει κάνει import κάποιο προσωπικό του (test) λογαριασμό στο Metamask χρησιμοποιώντας το private key του, η διαδικασία που ακολουθεί είναι η εξής:

i) Κατέβασμα του κώδικα της εφαρμογής από το github:
```
git clone --single-branch --branch docker_exp
https://github.com/georgebax/kyc_thesis.git
```
ii) Δημιουργία καινούριας γραμμής (ή αντικατάσταση κάποιας υπάρχουσας) στο ganache_with_accounts.sh script, στην οποία θα περιλαμβάνεται το private key του λογαριασμού του. Πρόκειται για ιδιωτικό δίκτυο που τρέχει τοπικά, οπότε είναι ασφαλή η χρήση οποιουδήποτε κλειδιού. Το δίκτυο αυτό ακούει μέσω RPC πρωτοκόλλου στη διεύθυνση localhost:**8545**.

iii) Μετάβαση (σε κάποιο unix shell) στην τοποθεσία kyc-thesis που προέκυψε από την προηγούμενη εντολή και εκτέλεση του docker-compose:
```
docker-compose up --build
```
iv) Έπειτα από 1-2 λεπτά, ανάλογα με την ταχύτητα της σύνδεσης για το κατέβασμα των docker images και το installation των node πακέτων, η εφαρμογή θα τρέξει, by default στην πόρτα 3000. Όπως αναφέρεται και παραπάνω, δε χρειάζεται κάτι εγκατεστημένο πέρα από το docker.

2) Normal deployment

Προαπαιτούμενα:

- unix λειτουργικό σύστημα (δοκιμασμένο σε Ubuntu 18.04 LTS)
- εγκατεστημένο git
- εγκατεστημένο node και npm (δοκιμασμένο με 12.6.0 και 6.12.0 αντίστοιχα)
- εγκατεστημένο docker και docker-compose

- εγκαστεστημένο truffle (`npm install –g truffle`)

Πρόκειται για τον τρόπο που επιτρέπει τη χρήση της εφαρμογής με Quorum δίκτυο. Στο testing της εφαρμογής χρησιμοποιήθηκε δίκτυο που έγινε deploy τοπικά σε εικονικά μηχανήματα, τόσο με το επίσημο quorum-examples project των δημιουργών του Quorum [14], όσο και με τη χρήση του Quorum maker [15] που έχουν αναφερθεί στο δεύτερο κεφάλαιο.

Η διαδικασία είναι η εξής:

i. Κατέβασμα του κώδικα της εφαρμογής από το github: `git clone https://github.com/georgebax/kyc-thesis.git`
ii. Κατέβασμα του quorum-examples project ή του quorum maker (χάριν ευκολίας το πρώτο) :

```
git clone https://github.com/jpmorganchase/quorum-examples/tree/master/examples/7nodes
```

iii. Deployment ιδιωτικού Quorum δικτύου με χρήση του παραπάνω εργαλείου. Μετάβαση από shell στο directory του 7nodes example και εκτέλεση του

```
docker-compose up –d
```

Το δίκτυο αυτό ακούει μέσω RPC πρωτοκόλλου στη διεύθυνση localhost:**22000**.
iv. Deployment των smart contracts: Μετάβαση στο kyc-thesis directory και εκτέλεση του

```
truffle migrate {--network <network_name>} --reset --compile-all.
```

όπου network_name το όνομα ενός από τα δίκτυα που δηλώνονται στο truffle-config.js.
v. Μετάβαση στο client/ directory (home της ReactJS εφαρμογής) και εκτέλεση του startup script:

```
npm run start
```

vi. Μετάβαση στο ext_auth/ directory (home της υποτυπώδους external authority εφαρμογής) και εκτέλεση του startup script:

```
node app.js
```

vii. Έναλλακτικά, μπορεί να σηκωθεί και ιδιωτικό (Ethereum) δίκτυο μέσω του GUI του Ganache (ή του ganache-cli), το οποίο by default θα ακούει στη διεύθυνση localhost:**7545.**

Έπειτα από αυτά τα στάδια, η εφαρμογή είναι σηκωμένη στο localhost:3000.

3) DApp setup

Το setup της εφαρμογής αφορά τις συνδέσεις της με το chain (δηλαδή το block-chain με το οποίο θα συνδεθεί) και τις off-chain οντότητες, δηλαδή το IPFS και την εξωτερική οντότητα.

Το blockchain με το οποίο θα συνδεθεί καθορίζεται στο truffle-config.js αρχείο, όπου έχουν ήδη φτιαχτεί επιλογές για σύνδεση με τοπικό Ethereum που «ακούει» μέ-σω RPC στην πόρτα 7545, και τοπικό Quorum στην 22000.

Το .env αρχείο της εφαρμογής (στο client/ directory) περιέχει επιλογή για τη δι-εύθυνση του bootstrap κόμβου του IPFS δικτύου, και το .env της εφαρμογής για το external authority έχει επιλογή για το εάν θα εγκρίνονται οι αιτήσεις ή όχι. Η έγκριση (ή η απόρριψη) είναι αυτόματη αφού πρόκειται για υποτυπώδη υλοποίηση.

Παράρτημα ΙΙ: Κώδικες

A Solidity

```
pragma solidity ^0.5.3;

contract UserStorage {

  string public message;

  uint totalUsers;

  mapping (uint => address) addressIndexing;
  mapping (address => User) users;

  event UserCreated(address indexed user, uint accountDura-
tion);
  event UserAccessed(address indexed user);
  event UserExists(address indexed user);
  event DocumentStored(address indexed user);
  event UserApproved(address indexed user, uint accounDura-
tion);
  event UserExpired(address indexed user);

  struct User {
    address addr;
    uint amount;
    uint accountExpiration; // EPOCH IN MS
    uint accountDuration;   // IN DAYS
    string userId;
    string docHash; // ipfs hash of the user's document
    bool hasDocument;
    bool approved;
    bool expired;
    bool stored;
  }

  constructor() public {
    totalUsers = 0;
  }

  // accounDuration in days
  // callTimestamp  in miliseconds
```

```
  function createUser(address _address, uint accountDuration,
uint callTimestamp, string memory userId) public {
     require(!userExists(_address), "User already exists");

     addressIndexing[totalUsers] = _address;
     totalUsers++;

     users[_address].addr = msg.sender;
     users[_address].accountDuration = accountDuration;
     users[_address].accountExpiration = callTimestamp + ac-
countDuration * 24 * 3600 * 1000; // converted days to mili-
seconds
     users[_address].userId = userId;
     users[_address].hasDocument = false;
     users[_address].approved = false;
     users[_address].stored = true;

     emit UserCreated(_address, accountDuration);
  }

  function getUserDuration(address _address) public view re-
turns(uint) {
    // emit UserAccessed(msg.sender);
    return users[_address].accountDuration;
  }

  function storeDocument(address _address, string memory ipf-
sHash) public {
    users[_address].docHash = ipfsHash;
    users[_address].hasDocument = true;
    users[_address].approved = false; // new ID, new request
    emit DocumentStored(_address);
  }

  function getUserExpiration(address _address) public re-
turns(uint) {
    require(userExists(_address));
    uint userExpiration = users[_address].accountExpiration;
    emit UserAccessed(_address);
    return userExpiration;
  }

  function isUserApproved(address _address) public view re-
turns (bool) {
    return users[_address].approved;
  }
```

```
  function isExpired(address _address) public view returns
(bool) {
    return users[_address].expired;
  }

  function userExists(address _address) public view returns
(bool) {
    return users[_address].stored;
  }

  function userHasDocument(address _address) public view re-
turns (bool) {
    return users[_address].hasDocument;
  }

  function approveUser(address _address, uint callTimestamp,
uint duration) public {
    require(userExists(_address)        &&        userHasDocu-
ment(_address));
    users[_address].approved = true;
    users[_address].accountDuration = duration; //renewed
    users[_address].accountExpiration      =      convertDura-
tion(callTimestamp, duration);
    emit UserApproved(_address, duration);
  }

  function verifyDocument(address _address) public {
    users[_address].approved = true;
  }

  function setExpired(address _address) public {
    users[_address].expired = true;
    emit UserExpired(_address);
  }

  function getUserInfo(address _address) public view re-
turns(string memory, uint, bool, string memory, bool, uint) {
    return (users[_address].userId,
            users[_address].accountExpiration,
            users[_address].approved,
            users[_address].docHash,
            users[_address].hasDocument,
            users[_address].accountDuration);
  }

  function getExpiredUsers() public returns (address[] memory)
{
```

```
    address[] memory expiredUsers = new address[](totalUsers);
// max possible size
    for (uint i = 0; i < totalUsers; i++) {
      if (users[addressIndexing[i]].expired) {
        expiredUsers[i] = users[addressIndexing[i]].addr;
      }
    }
    return expiredUsers;
  }

  function getAllUsers() public returns (address[] memory) {
    address[] memory returned = new address[](totalUsers); //
memory array cannot be dynamic
    for (uint i = 0; i < totalUsers; i++) {
      returned[i] = users[addressIndexing[i]].addr;
    }
    return returned;
  }

  function convertDuration(uint callTimestamp, uint duration-
InDays) private view returns (uint)  {
    return callTimestamp + durationInDays * 24 * 3600 * 1000;
  }

  function () external payable {
    // empty
  }

}
```

B JavaScript

App.js (Main React App code)

```
import React, { Component } from "react";
import UserStorage from "./contracts/UserStorage.json";
import getWeb3 from "./utils/getWeb3";
import { ipfs } from './ipfs';
import 'bootstrap/dist/css/bootstrap.css';


class App extends Component {
  state = {
    web3: null,
    contract: null,
```

```
    buffer: null,
    account: null,
    ipfsHash: null,
    expirationDate: null,
    isApproved: false,
    isExpired: false,
    userExists: false,
    documentUploaded: false,
    newUserName: '',
    newUserDuration: 0,
    currentUserName: ''
  };

  componentDidMount = async () => {
    try {
      // Get network provider and web3 instance.
      const web3 = await getWeb3();
      //console.log(web3);
      // Use web3 to get the user's accounts.
      const accounts = await web3.eth.getAccounts();
      // Get the contract instance.
      const networkId = await web3.eth.net.getId();
      //console.log('network id: ' + networkId);
      const deployedNetwork = UserStorage.networks[networkId];
      const instance = new web3.eth.Contract(
        UserStorage.abi,
        deployedNetwork && deployedNetwork.address
        );

      //event listener for approval
      instance.events.UserApproved({},  function(error,  event)
{
        console.log(event);
      });

      this.setState({  web3,  contract:  instance,  account:  ac-
counts[0] }, this.runExample);
    } catch (error) {
        alert(`Failed  to  load  web3,  accounts,  or  contract.
Check console for details.`);
        console.error(error);
      }
    };

    runExample = async () => {
      const { account, contract } = this.state;
```

```
        contract.methods.userExists(account).call( {from: ac-
count}, (error, result) => {
          if (error) {
            console.error(error);
            return;
          }
          if (result) {
            contract.methods.getUserInfo(account).call( {from:
account}, (error, result) => {
              if (error) {
                console.error(error);
                return;
              }

              let expiredFlag = false;
              if (result[1] < new Date().getTime()) {
                expiredFlag = true;
                contract.methods.isExpired(account).call(
{from: account}, (error, result) => {
                  if (!result) {
                    contract.methods.setExpired(account).send(
{from: account}, (error, result) => {
                      if (error) {
                        console.error(error);
                        return;
                      }
                    })
                  }
                })
              }

              this.setState({
                userExists        : true,
                currentUserName   : result[0],
                expirationDate                                  :
this.convertTimestamp(result[1]),
                isApproved        : result[2],
                isExpired         : expiredFlag,
                ipfsHash          : result[3],
                documentUploaded  : result[4],
                newUserDuration   : result[5] // in days
              });

            });
          }
        });
```

```
      }

      convertTimestamp(timeStamp) {
        const timestampToDate = require('timestamp-to-date');
        let expiration = timestampToDate(timeStamp, 'yyyy-MM-
dd HH:mm:ss'); // expirationDate is in seconds, not ms.
        return expiration;
      }

      captureFile(event) {
        event.preventDefault();
        const file = event.target.files[0];
        const reader = new window.FileReader();
        reader.readAsArrayBuffer(file);
        reader.onloadend = () => {
          this.setState( {buffer: Buffer(reader.result)} );
        }
      }

      onSubmit(event) {
        event.preventDefault();

        ipfs.add(this.state.buffer, (error, result) => {
          if(error) {
            console.error(error);
            return;
          }
          const ipfsHash = result[0].hash;
          this.setState({ ipfsHash });

          const { account, contract } = this.state;

          contract.methods.storeDocument(account,          ipf-
sHash).send( {from: account}, (error, result) => {
            if (error) {
              console.error(error);
              return;
            }
            this.setState( {documentUploaded: true} );

            this.notifyAuthority(event);
          });
        });

      }
```

```
      notifyAuthority(event) {
        event.preventDefault();

        const options = {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({
            name: this.state.currentUserName,
            duration: this.state.newUserDuration,
            ipfsHash: this.state.ipfsHash,
            quorumAddress: this.state.account
          })
        }

        fetch(process.env.REACT_APP_EXT_AUTH_ADDR, options)
          .then(response => response.json())
          .then(json => {
            if (json.approved) {
              this.approveUser(json.quorumAddress,
json.duration);
            }
        });

      }


      handleNewUserRequest(event) {
        event.preventDefault();

        const { account, contract } = this.state;
        contract.methods.createUser(account,
this.state.newUserDuration,        new        Date().getTime(),
this.state.newUserName).send( {from: account}, (error, result)
=> {
            if (error) {
              console.error(error);
              return;
            }

            // following line is to update the page's state
and enable the panel showing the existing user's info without
reloading
            contract.methods.getUserExpiration(account).send(
{from: account}, (error, result) => {
              if (error) {
                console.error(error);
                return;
```

```
            }
            this.setState(                    {expirationDate:
this.convertTimestamp(result)} );

          })
          this.setState(                    {currentUserName:
this.state.newUserName} );
          this.setState(  {newUserName:  null,  newUserDura-
tion: null, userExists: true, isApproved: false} );
        });

    }

    getUsers(event) {
      event.preventDefault();
      this.state.contract.methods.getAllUsers().call(  {from:
this.state.account}, (error, result) =>  {
        if (error) {
          console.error(error);
          return;
        }
        console.log('fetching the users');
        console.log(result);
      });
    }

    getExpiredUsers(event) {
      event.preventDefault();
      this.state.contract.methods.getExpiredUsers().call(
{from: this.state.account}, (error, result) =>  {
        if (error) {
          console.error(error);
          return;
        }
        console.log('fetching expired users');
        console.log(result);
      });
    }

    approveUser(address, duration) {
      const { account, contract } = this.state;
      contract.methods.approveUser(address,               new
Date().getTime(), duration).send( {from: account}, (error, re-
sult) => {
        if (error) {
          console.error(error);
          return;
```

```
          }
          if (account === address) {
            this.setState( {isApproved : true} );
          }
        });

      }

      handleUserName(event) {
        this.setState( {newUserName: event.target.value} );
      }

      handleUserDuration(event) {
        this.setState(                          {newUserDuration:
event.target.value.replace(/\D/,'')} );
      }

      render() {
        if (!this.state.web3) {
          return (
            <div>Loading Web3, accounts, and contract...</div>
            );
        }

        // set conditionals here

        let statusBanner;
        if (this.state.userExists) {
          if (this.state.isApproved) {
            statusBanner =
              <div  class="card"  style={{'background-color'  :
'#b8ffb5'}}>
                <h2>YOUR ID HAS BEEN APPROVED!</h2>
              </div>;
          } else if (this.state.documentUploaded) {
            statusBanner =
              <div   class="card   px-2"   style={{'background-
color' : '#fcf45d'}}>
                <h2>YOUR   DOCUMENT   IS   AWAITING   VERIFICA-
TION</h2>
                <form                                       onSub-
mit={this.notifyAuthority.bind(this)}>
                  <input class="btn btn-primary" type='submit'
value='Request Approval'/>
                </form>
              </div>;
          }
```

```
        }

        if (this.state.isExpired) {
          statusBanner =
            <div   class="card"   style={{'background-color'   :
'#f05443'}}>
              <h2>YOUR ACCOUNT HAS EXPIRED</h2>
            </div>
        }

        let newUserForm;
        let existingUserInfo;
        let uploadedBy;
        let uploadPrompt = <h2>Please upload document to veri-
fy your identity</h2>;
        let ipfsHost = process.env.REACT_APP_IPFS_HOST;
        let adminSection;

        if (this.state.userExists && !this.state.isExpired) {

          if (this.state.documentUploaded) {
            uploadedBy =
              <div>
                <img
src={`${ipfsHost}/ipfs/${this.state.ipfsHash}`} alt=""/>
                <p>Uploaded by {this.state.account}</p>
                <p>Download  your  Document  <a  class="btn  btn-
link"
href={`${ipfsHost}/ipfs/${this.state.ipfsHash}`}>here</a>,   if
preview is not available.</p>
              </div>;
            uploadPrompt = <br></br>;
          }

          existingUserInfo =
            <div class="card px-2">
              <div class="card px-2">
                <h2>Welcome,
{this.state.currentUserName}.</h2>
                <h3>Your       account       expires       on:
{this.state.expirationDate}</h3>
              </div>
              <div class="card px-2">
                {uploadedBy}
                {uploadPrompt}
                <form onSubmit={this.onSubmit.bind(this)} >
```

```
                  <input            type='file'            on-
Change={this.captureFile.bind(this)} />
                  <input class="btn btn-primary" type='submit'
value="Upload Document"/>
                </form>
                <br/>
              </div>
            </div>

        } else if (!this.state.isExpired) {
          newUserForm =
            <div class="card px-2 form-group">
              <h2>No account found for your Quorum address.
Please register</h2>
              <form                                          onSub-
mit={this.handleNewUserRequest.bind(this)}>
                <table style={{'width' : '30%'}}>
                  <tbody>
                    <tr>
                      <td><label>Name:</label></td>
                      <td><input        type="text"        val-
ue={this.state.newUserName}                                   on-
Change={this.handleUserName.bind(this)} /></td>
                    </tr>
                    <tr>
                      <td><label>Desired    account    duration
(days):</label></td>
                      <td><input        type="text"        val-
ue={this.state.newUserDuration}                               on-
Change={this.handleUserDuration.bind(this)} /></td>
                    </tr>
                  </tbody>
                </table>
                <input      type="submit"      value="Register"
style={{'margin' : '20px'}}/>
              </form>
              <br/>
            </div>
        }

        if (process.env.REACT_APP_DEPLOYMENT==='test') {
          adminSection =
            <div class="card">
              <h2>Admin Section.</h2>
              <h3>For demonstration purposes only</h3>
              <div class="form-group">
                <form onSubmit={this.getUsers.bind(this)} >
```

```
                  <input class="btn btn-primary" type='submit'
value="Get All Users"/>
                </form>
                <br/>
                <form                                        onSub-
mit={this.getExpiredUsers.bind(this)} >
                  <input class="btn btn-primary" type='submit'
value="Get Expired Users"/>
                </form>
              </div>
            </div>

        }

        return (
          <div class="card" className="App">
            <div class="card card-body">
              <h1>Know Your Customer: A Decentralized Applica-
tion</h1>
              <h2>Authenticate user trying to log in to a
blockchain, by saving his personal documents to the local IPFS
network and getting authorization by an external entity.</h2>
              <div  class="card"  style={{'background-color'  :
'#a3e7ff', padding : '50px'}}>
                <h3>Quorum address: {this.state.account}</h3>
              </div>

              {existingUserInfo}
              {statusBanner}
              {newUserForm}
              {adminSection}
            </div>
          </div>
        );
      }
    }

    export default App;
```

ext_auth/app.js (Dummy external authority server)

```
const express = require('express');
const app = express();
const dotenv = require('dotenv');
const port = 5000;
```

```
dotenv.config();
app.use(express.json());

app.post('/', (req, res) => {
      console.log(req.body);
      let approved = false;
      if (process.env.APPROVE === 'true') {
            approved = true;
      }
     res.json({
      quorumAddress: req.body.quorumAddress,
      duration: req.body.duration,
      approved: approved
     });
})

app.listen(port, () => console.log(`Dummy app for the external
authority listening on ${port}! Approves or rejects requests
based on an  environment variable`))
```