Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων
Πανεπιστημιο Πειραιως
Σχολη Ναυτιλιας και Βιομηχανιας
Τμημα Βιομηχανικης Διοικησης και Τεχνολογιας
Διαπανεπιστημιακο Προγραμμα Μεταπτυχικων Σπουδων
Τεχνοοικονομικα Συστηματα

Αποκεντρωμένο σύστημα διαχείρισης εγγράφων υλοποιημένο στο Ethereum Blockchain

Διπλωματικη Εργασια

του

ΓΕΩΡΓΙΟΥ ΤΣΑΤΣΑΝΙΦΟΥ

Επιβλέπουσα : Δρ. Αδαμοπούλου Ευγενία
Ε.Μ.Π.

Αθήνα, Οκτώβριος 2018

© CopyLeft Notice

Περίληψη

Στη παρούσα εργασία σχεδιάζουμε κι υλοποιούμε πάνω στο Ethereum blockchain ένα smart-contract για τη διαχείριση εγγράφων, χρηστών και δικαιωμάτων πρόσβασης στο περιεχόμενο. Ξεκινώντας με τις προδιαγραφές των απαραίτητων μεθόδων, υλοποιούμε τις μεθόδους που θα αποτελέσουν το smart-contract μαζί με τις απαραίτητες μεταβλητές κατάστασης στις προγραμματιστικές γλώσσες Solidity και Serpent για smart-contracts στο Ethereum blockchain. Αμφότερες υλοποιήσεις είναι κατάλληλες για deployment και χρήση από το Ethereum Virtual Machine (EVM) και περιλαμβάνουν μεθόδους για την προσθαφαίρεση χρηστών κι εγγράφων, διαχείριση δικαιωμάτων καθώς και μεθόδους για την προβολή των διαθέσιμων εγγράφων σύμφωνα με τα δικαιώματα των χρηστών αλλά και του περιεχόμενού τους, ενώ παράλληλα η ενδεχόμενη τροποποίηση τους ελέγχεται κατάλληλα.

Επίσης, στοχεύοντας στην αποτελεσματική UI/UX διασύνδεση, κατασκευάζουμε μία γραφική διεπαφή σε JavaScript χρησιμοποιώντας το MVVM framework της AngularJS για τη φιλική πρόσβαση των χρηστών στο blockchain, την ανάρτηση εγγράφων, την ανάγνωση και τροποποίηση του περιεχομένου των εγγράφων, καθώς και τη διαχείριση των χρηστών από τον υπεύθυνο του smart-contract. Αρχικά, εξηγούμε την αρχιτεκτονική των εφαρμογών αυτού του είδους κι έπειτα περιγράφουμε τη δομή που υλοποιήσαμε βασισμένη στην επικοινωνία με το Ethereum blockchain, γεγονός που εσωκλείει ιδιαίτερες ιδιομορφίες, όπως η εξαιρετικά συχνή χρήση callbacks κι η μεταφορά της λογικής της εφαρμογής, διαχείρισης σφαλμάτων και παρουσίασης των κατατοπιστικών μηνυμάτων προς τον χρήστη εντός των callbacks και των μεθόδων διαχείρισης events που προέρχονται από το blockchain. Τέλος, παρουσιάζουμε σε εικόνες ένα εκτενές σενάριο που περικλύει όλα τα workflows που υποστηρίζει η εφαρμογή μας end-to-end.

Abstract

In this work, we design and implement on top of Ethereum blockchain a smart-contract for document manipulation, users and their access rights. We start by giving a specification and an interface, which is then implemented in Solidity and Serpent programming languages, used for developing smart-contracts in Ethereum blockchain. Both implementations are suitable for deployment on the Ethereum Virtual Machine (EVM) and include methods for adding and removing users, documents, granting and revoking privileges, obtaining and altering their content according to users' rights.

We also give a user-friendly web-interface, developed in JavaScript and built using the AngularJS MVVM framework. This way, users can use the elements of the HTML DOM with the Angular-related directives so as to upload documents, read and modify their content, while the owner of the contract is responsible for the users of the smart-contract. We first explain the architecture of such applications, and then we describe the structure of our implementation which is based on interfacing with Ethereum blockchain via a third party library. And this involves a lot of application-specific peculiarities, such as the exceedingly frequest usage of callback which requires transfering almost all logic, handling of errors and messages within callbacks and methods for handling events originated from blockchain. Last but not least, we illustrate an extensive scenario which includes all workflows that our paradigm supports, end-to-end.

Λέξεις κλειδιά

Διαχείριση εγγράφων

Κατανεμημένο λογιστικό κατάστιχο

Distributed ledger

Blockchain

Ethereum

Smart-contracts

Solidity

Serpent

JavaScript

AngularJS

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Κεφάλαιο 1

Εισαγωγή

Pecunia non olet. (Money does not stink.)

Roman emperor Vespasian (ruled AD 69–79)

Bad money drives out good.

Thomas Gresham (1519–1579), Gresham's Law

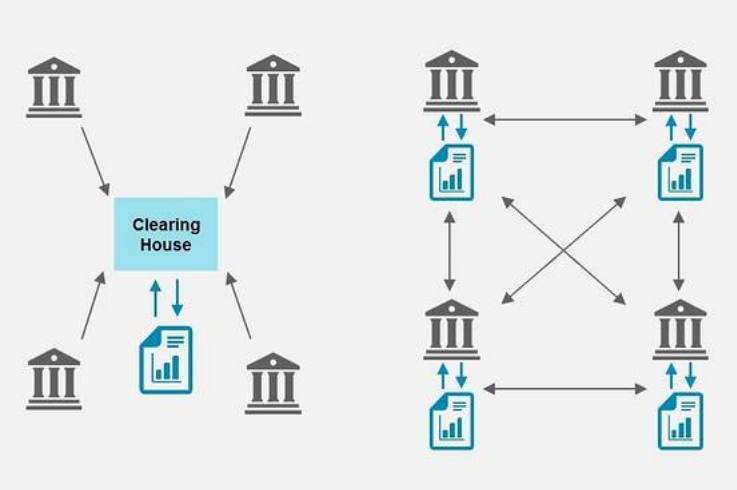
Στο κεφάλαιο αυτό κάνουμε μία συνοπτική περιγραφή της τεχνολογίας την οποία αφορά η εργασία αυτή, τις αρχές της, εφαρμογές από τον πραγματικό κόσμο και παραδείγματα.

1.1 Βασικές αρχές

Στον πυρήνα του το blockchain είναι μία κατανεμημένη βάση (distributed database) πάνω σε ένα δίκτυο ομοτίμων (peer-to-peer network) το οποίο απαρτίζεται από τους υπολογιστές των χρηστών που έχουν συνδεθεί σε αυτό. Το περιεχόμενο της βάσης σχετίζεται κυρίως με τον σκοπό της εφαρμογής την οποία εξυπηρετεί. Όσον αφορά το bitcoin[20] για παράδειγμα, το blockchain λειτουργεί ως ένα λογιστικό σύστημα αρχειοθέτησης στο οποίο καταγράφεται το υπόλοιπο κάθε χρήστη σε μονάδες κρυπτονομίσματος, ενώ παράλληλα επιτρέπεται η ανώνυμη μεταφορά μονάδων κρυπτονομίσματος από έναν χρήστη σε έναν άλλον χωρίς τη διαμεσολάβηση κάποιου τρίτου φορέα, π.χ. τραπεζικό ίδρυμα. Λόγω της αποκεντρωμένης (decentralized) δομής του blockchain το περιεχόμενο δεν μπορεί να ελεγχθεί από μία και μόνον οντότητα, ενώ επιπλέον, σε περίπτωση απώλειας ενός κόμβου του δικτύου, το σύστημα είναι σε θέση να συνεχίσει τη λειτουργία του απρόσκοπτα.

1.1.1 Τα συστατικά κι η συνταγή της επιτυχίας

Αλλά ας εξετάσουμε όμως εξ αρχής τι χρειάζεται ένα λογιστικό σύστημα ιντερνετικών διαστάσεων στο οποίο είναι σε θέση να συμμετάσχει ο οποιοσδήποτε ανώνυμος χρήστης του διαδικτύου, κι άρα μη πιστοποιημένος ή αξιόπιστος απαραίτητα. Κατ' ανάγκη οι συναλλαγές θα πρέπει να παραμένουν αναλλοίωτες και να μην είναι δυνατή η μετέπειτα "διόρθωσή" τους εφόσον αυτές λάβουν χώρα. Δηλαδή, είναι δυνατή μονον η προσθήκη νέων συναλλαγών κι όχι η διαγραφή, αναδιάταξη ή αλλαγή υπαρχόντων συναλλαγών. Θα πρέπει να υπάρχει ένας τρόπος διετέλεσης ασφαλών συναλλαγών ως φυσική απόρροια της αποτελεσματικής πιστοποίησης των λογιστικών εγγραφών καθώς αυτές λαμβάνουν χώρα με δυναμικό τρόπο, προερχόμενες ταυτόχρονα από αυθαίρετα κι απομακρυσμένα σημεία του δικτύου. Ως συνέπεια αυτής της αρχής είναι ο άμεσος εντοπισμός των πλαστών συναλλαγών, ο οποίος θα πρέπει να γίνεται με γρήγορο και "φτηνό" υπολογιστικά τρόπο ώστε η συνολική λειτουργία



Σχήμα 1.1: Συναλλασσόμενα ιδρύματα σε μία peer-to-peer οργάνωση.

του συστήματος, το οποίο απαρτίζεται από αυθαίρετο αριθμό από κόμβους, να συντελείται ομαλά, αποτελεσματικά κι απρόσκοπτα[21, 22].

Ακόμα, η ολική διάταξη των συναλλαγών ως προς τον χρόνο είναι απόλυτη και μη επιρρεπής σε μεταβολές. Κάθε νέα συναλλαγή που πιστοποιείται προστίθεται σε μία λίστα συνδεόμενων συναλλαγών σύμφωνα με μία χρονική ακολουθία. Η νέα πιστοποίηση που συμπεριλαμβάνει την καινούρια συναλλαγή λαμβάνει υπόψην την πιστοποίηση της προηγούμενης, κι άρα η αλλοίωση σε οποιοδήποτε σημείο της αλυσίδας, π.χ. η αλλαγή στη σειρά ή στα ποσά μίας ή περισσοτέρων συναλλαγών, είναι άμεσα ανιχνεύσιμη κι έχει ως αποτέλεσμα την απόρριψη της συναλλαγής ως μη αξιόπιστη. Οπότε, ως έγκυρες προσλαμβάνονται μόνον οι συναλλαγές οι οποίες είναι επικυρωμένες σύμφωνα με ένα Proof-of-Work (PoW) πρωτόκολλο, παραβίαση του οποίου απαιτεί ένα ασύμφωρο υπολογιστικό κι ενεργειακό κόστος ως προς τα οφέλη τα οποία θα επέφερε μία τέτοια κίνηση. Αυτό επιτυγχάνεται με τη χρήση συναρτήσεων hashing των οποίων το αποτέλεσμα είναι τυχαίο για ορισμένες πρακτικές περιπτώσεις. Αυτό σημαίνει ότι προκειμένου να βρεθεί (δεδομένου της εξόδου) η είσοδος στη hashing συνάρτηση θα πρέπει να δοκιμαστούν μία προς μία απαγορευτικά πολυάριθμες πιθανές τέτοιες λύσεις. Για παράδειγμα, ένας κακόβουλος χρήστης που προσπαθεί να βρεί το κλειδί για μία συνάρτηση που παράγει μία έξοδο που ξεκινάει με δέκα μηδενικά στο δυαδικό σύστημα, θα πρέπει να εξετάσει τουλάχιστον 2^{10} ενδεχόμενα. Ειδικότερα, σε κάθε block ο κόμβος που δημιουργεί την πιστοποίηση (miner) συμπληρώνει το πεδίο ονόματι **nonce** για το οποίο δοκιμάζει τιμές από το 0 έως το δεκαεξαδικό `0xFFFFFFFFFFFFFFFF`, έως ότου να επαληθευτεί η έξοδος της συνάρτησης hashing. Οπότε, μπορούμε τώρα να συνοψίσουμε όλα τα προηγούμενα βήματα λειτουργίας του blockchain στα παρακάτω σημεία ως εξής,

1. Καινούριες συναλλαγές γνωστοποιούνται (broadcasting) σε όλους τους κόμβους του δικτύου προερχόμενες από τον αιτούντα (π.χ. ο κόμβος που ξεκινάει την μεταφορά κρυπτονομισμάτων από το υπόλοιπό του προς έναν άλλον χρήστη).

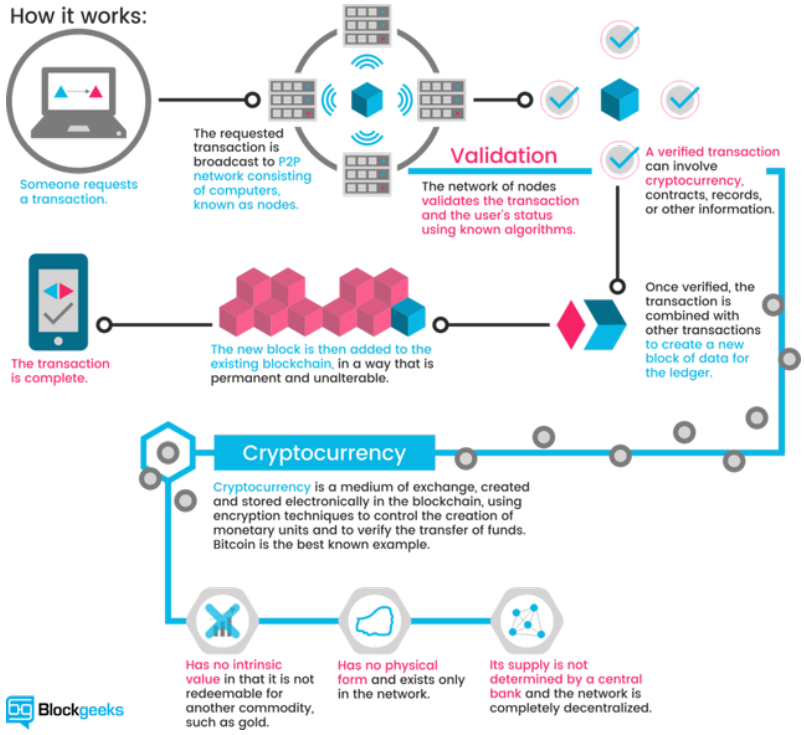
2. Κάθε κόμβος που λαμβάνει μία νέα συναλλαγή διατηρεί μία χρονολογημένη λίστα με τις ανεπιβεβαίωτες συναλλαγές.

3. Λειτουργεί ως miner προσπαθώντας να επιβεβαιώσει την αμέσως επόμενη συναλλαγή που έπεται της τελευταίας επιβεβαιωμένης του blockchain.

4. Ο miner ανακοινώνει σε όλο το δίκτυο την επιβεβαίωση.

5. Οι αποδέκτες προσθέτουν το block στην αλυσίδα τους εφόσον,

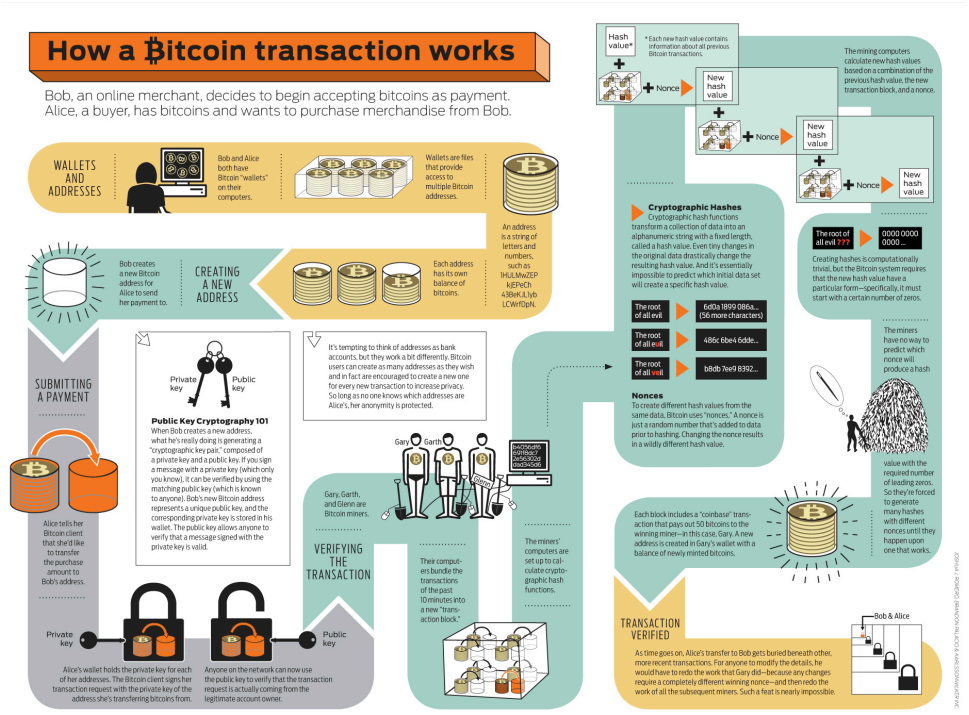
   - επιβεβαιωνεται το hash σύμφωνα με τα προηγούμενα,
   - η ίδια μονάδα κρυπτονομίσματος δεν έχει ξαναχρησιμοποιηθεί από τον χρήστη.



Σχήμα 1.2: Σχηματική απεικόνιση κύκλου ολοκλήρωσης συναλλαγής στο blockchain.

1.1.2 Υποκείμενοι κίνδυνοι

Όμως, ακόμα κι όταν ικανοποιούνται όλα αυτά τα κριτήρια, ελλοχεύουν συμπληρωματικοί κίνδυνοι λόγω της μεγάλης κλιμάκωσης του συστήματος. Ένα τέτοιο παράδειγμα προκύπτει από την ταυτόχρονη απόπειρα προσθήκης δύο έγκυρων συναλλαγών, οι οποίες πιστοποιήθηκαν αμφότερες με την έως εκείνη τη στιγμή ακολουθία συναλλαγών, χωρίς όμως ποτέ η μια να λάβει υπόψην την ύπαρξη της άλλης. Με τον τρόπο αυτό δημιουργούνται δύο διαφορετικές πραγματικότητες οι οποίες πρέπει να συμβιβαστούν στα πλαίσια του blockchain, ώστε όλοι οι κόμβοι που συμμετέχουν να αποκτήσουν μία επικαιρωποιημένη εικόνα των διεκπεραιωμένων συναλλαγών. Η συγκεκριμένη τεχνολογική πρόκληση ονομάζεται συγχρονισμός κι αποτελεί έμφυτο πρόβλημα στα κατανεμημένα συστήματα, τόσο σε θεωρητικό επίπεδο (βλέπε ρολόγια Lamport, Vector Clocks), όσο και σε πρακτικό, όπως η περίπτωση που εξετάζουμε στη συγκεκριμένη ενότητα. Η λύση που έχει δοθεί στο συγκεκριμένο πρόβλημα για το blockchain ίσως είναι η κορωνίδα και το πιο αντιπροσωπευτικό χαρακτηριστικό το οποίο πάντα θα μας υπενθυμίζει ότι μιλάμε για μία αποκεντρωμένη τεχνολογία συνδεόμενων κόμβων σε μία άναρχη οριζόντια δομή κανάβου (mesh). Ειδικότερα, το εάν το ένα από τα δύο blocks θα προηγείται εν τέλει του άλλου, εξαρτάται μόνο από το ποιά από τις δύο πραγματικότητες θα καταλήξει μακρύτερη με περισσότερα block να έπονται από το γεγονός αυτό και μετά, σε σχέση με την άλλη αλυσίδα με τη διαφορετική σειρά φυσικά. Συνεπώς, το σύστημα είναι ευέλικτο, κλιμακώσιμο και αρκετά ανεκτικό σε σφάλματα ώστε να μην χρειάζεται όλες οι συναλλαγές να φτάνουν σε όλους τους miners, ή ακόμα κι όταν το κάνουν, δεν χρειάζεται να έχουν την ίδια σειρά. Ιδιότητες και χαρακτηριστικά που είναι απαραίτητο να υποστηρίζονται για τόσο κατακερματισμένα συστήματα ιντερνετικής κλίμακας απομακρυσμένων κόμβων. Εδώ όμως κρύβεται μία πολύ μεγάλη αδυναμία του blockchain. Και το πρόβλημα αυτό, όπως και το προηγουμενο, έχει θεωρητικές προεκτάσεις κι ονομάζεται "**Βυζαντινή Συμφωνία**" (Byzantine Failure). Η ονομασία, αν και παράξενη, προέρχεται από την ιστορία κι αποτελεί παραβολή με τους στρατηγούς που συνωμοτούν για να κατατροπώσουν και να ανατρέψουν τον βυζαντινό αυτοκράτορα, π.χ. στάση του Νίκα, κοκ. Ας υποθέσουμε ότι υπεισέρχεται στο δίκτυο ένας πολύ υπολογιστικά ισχυρός κόμβος, του οποίου η ισχή υπερβαίνει αθροιστικά

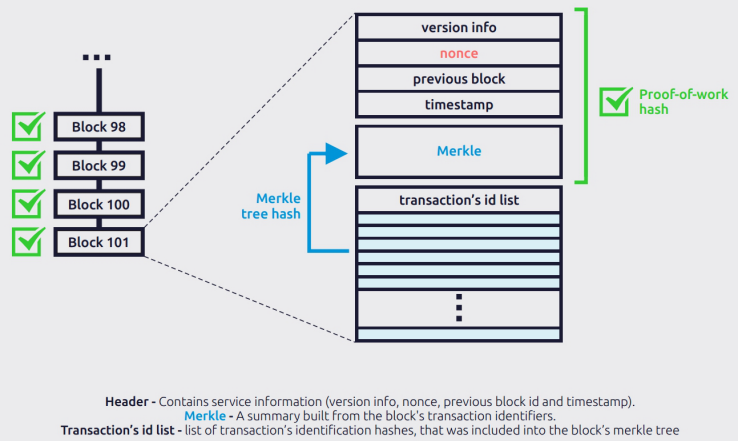


Σχήμα 1.3: Σχηματική απεικόνιση κύκλου ολοκλήρωσης συναλλαγής στο bitcoin.

τους δύο πιο "δυνατούς" κόμβους του blockchain, ο οποίος μάλιστα έχει εξασφαλίσει μία πολύ καλή κεντρική θέση στο δίκτυο και βρίσκεται λίγα μόνο hops μακρυά από οποιονδήποτε άλλο κόμβο στο blockchain (βλεπε network centrality), ώστε να αφουγκράζεται εύκολα τις συναλλαγές από όλες τις "γωνιές". Τότε, θα είναι σε θέση να δημιουργήσει τη δική του πραγματικότητα εντός του blockchain αλλά και να την επιβάλλει με το να επικυρώνει τις δικές του τεχνητές συναλλαγές μαζί με τις υπόλοιπες πιο γρήγορα από κάθε άλλον κόμβο. Συνεπώς, οι υπόλοιποι κόμβοι δε θα έχουν άλλη επιλογή πέρα από το να υιοθετούν και να επικαιρωποιούν τη δική τους πραγματικότητα με αυτήν που τους "σερβίρει" ο βυζαντινός στρατηγός που έχρισε τον εαυτό του πλούσιο με το να χρησιμοποιεί ξανά και ξανά τις ίδιες μονάδες κρυπτονομίσματος που του ανήκουν ή να εκδίδει για τον εαυτό του όσο κρυπτονόμισμα του επιτρέπει η ανώτερη υπολογιστική ισχύς του. Από εκεί και πέρα, θα του μένει μόνο να αναλύσει το νεκρό σημείο (break-even point) καθώς και το χρονικό διάστημα (payback period) που θα χρειαστεί προκειμένου να αποσβέσει την αγορά του hardware (σταθερό κόστος) και τους ενεργειακούς πόρους (variable cost)!

1.1.3 Πραγματικές αγορές εικονικών νομισμάτων

Οι διάφορες εκφάνσεις της blockchain τεχνολογίας που έχουν κατα καιρούς προταθεί δίνουν παρόμοιες λύσεις στα θεμελιώδη προβλήματα που προαναφέραμε. Προσφέρουν όμως διαφορετικά χαρακτηριστικά που κάνουν έτσι τη μία ή την άλλη λύση πιο θελκτική ή κατάλληλη στα μάτια του εκάστοτε χρήστη, προγραμματιστή ή επαγγελματία. Όσον αφορά τους εθελοντές που συμμετέχουν προσφέροντας υπολογιστική ισχύ στην υποδομή, γνωστοί κι ως miners, αυτοί ελκύονται από τη δύναμη που κρύβει το κρυπτονόμισμα που συνοδεύει την τεχνολογία την οποία πρεσβεύουν. Δεδομένου ότι τα κρυπτονομίσματα δεν έχουν εσωτερική αξία (intrinsic value) όπως ο χρυσός ή οι πολύτιμοι λίθοι, η εδραίωση της μίας τεχνολογίας αντί της άλλης σε μεγάλο βαθμό επαφίεται σε ακριβώς αυτό το χαρακτηριστικό, το πόσο υποσχόμενο είναι το κρυπτονόμισμα, τη δυναμική που κρύβει, και την ευελιξία και δυνατότητες της τεχνολογίας στην οποία βασίζεται. Αν τα χαρακτηριστικά αυτά δεν είναι αρκετά πειστικά στο ευρύ κοινό, τότε το όλο εγχείρημα κινδυνεύει, αφού τότε εύκολα κάποιος κακοβουλος

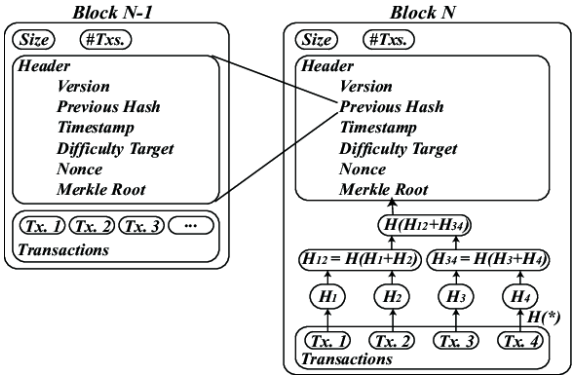


Σχήμα 1.4: Η ανατομία ενός block.

χρήστης θα μπορούσε πιο εύκολα να συγκεντρώσει αρκετό ποσοστό της συνολικής επεξεργαστικής ισχύος προκειμένου να είναι σε θέση να ελέγχει ποιές συναλλαγές θα προστίθονται στο blockchain, αλλά και να προσθέτει και τεχνητές συναλλαγές που θα του επιτρέπουν να ξοδεύει ξανά και ξανά την ίδια ακριβώς μονάδα κρυπτονομίσματος, πρόβλημα γνωστό κι ως double-spending. Ο κίνδυνος αυτός που υποβόσκει σε όλες τις blockchain πλατφόρμες που έχουν κατά καιρούς προταθεί κάνει τη διαφορά μεταξύ φθοράς κι αφθαρσίας της προτεινόμενης τεχνολογίας. Ίσως εκεί ακριβώς βρίσκεται το μυστικό της επιτυχίας του bitcoin[20], όπου μόλις τον περασμένο Δεκέμβρη (12/2017) η αξία της μονάδας του κρυπτονομίσματος προσέγγισε τα 20,000 USD. Δηλαδή, έπεισε το ευρύ κοινό ως υποσχόμενη τεχνολογία πέρα από την καινοτομία την οποία προσέφερε και κατάφερε να εδραιωθεί και να εγκαθιδρυθεί με τέτοιο τρόπο παράλληλα με την πραγματική οικονομία και να αποκτήσει ανταλλακτική αξία για καταναλωτικά αγαθά σαν να πρόκειται για τραπεζογραμμάτια (fiat money). Ακόμα, η μεταβλητότητα της τιμής του bitcoin έχει περιοριστεί πάρα πολύ σε σχέση με το παρελθόν, με το 2015 να σημειώνεται ως η χρονιά με τη μικρότερη μεταβλητότητα στην ιστορία του νομίσματος. Το γεγονός αυτό το κάνει ακόμα πιο ελκυστικό ως μέσο αποθήκευσης αξίας σε σχέση με άλλα, πολύ μεταβλητά κρυπτονομίσματα και μη.

Όμως, πέρα από τους νόμους της αγοράς που άγεται και φέρεται συνήθως με τη ψυχολογία της μάζας που τη διέπει, είχαν την προνοητικότητα οι σχεδιαστές του blockchain να προσδώσουν μηχανισμούς κινήτρων για να προσελκύσουν περισσότερους συμμέτοχους στην τεχνολογία τους. Έτσι, έχουμε την ανταμοιβή των miners με μονάδες ενός υποσχόμενου κρυπτονομίσματος για το έργο τους. Αυτό έγινε αρχικά πριν δέκα χρόνια περίπου με μία διανομή πληθωριστικών κρυπτονομισμάτων χωρίς πραγματικό αντίκρισμα, ενώ πλέον γίνεται με τη μορφή transaction fees. Ως ιδέα προσομοιάζει την έννοια του **seignorage** ή **seigneurage**[17], τη διαφορά δηλαδή μεταξύ της ονομαστικής αξίας του νομίσματος και του κόστους παραγωγής και διανομής του ως φυσικό αντικείμενο. Διατηρώντας τεχνηέντως μία διαφορά μεταξύ των ποσοτήτων κρυπτονομίσματος που διαχειρίζεται η κάθε συναλλαγή στην είσοδο και στην έξοδό της, γίνεται το ένα block πιο ελκυστικό προς mining εις βάρος του άλλου, κάτι βέβαια που μπορεί να θέσει έτσι τις μη γενναιόδωρες συναλλαγές εκτός blockchain και να οδηγήσει σε μία ελεγχόμενη, ολιγαρχική κι "ελιτίστικη" αγορά αποτελούμενη από τις προνομιακές συναλλαγές και μόνον. Προφανώς, κάτι τέτοιο θα έδινε έναυσμα και θα δημιουργούσε εφαλτήριο για ξέπλυμα μαύρου χρήματος και τον ανάλογο χαρακτηρισμό του blockchain με διάφορους συνειρμούς. Συνεπώς, λόγω της δημόσιας διαθεσιμότητας όλων των συναλλαγών, εγκληματολογικές έρευνες επιβάλλεται να διεξάγονται εντατικά και όταν χρειάζεται να επιχειρείται να συσχετιστεί ο ανώνυμος χρήστης με το πραγματικό πρόσωπο (π.χ. IP address).

Ιδιαιτερότητες σε κάποιες οικονομίες αλλά και πρόσκαιρες συνθήκες έχουν διευκολύνει την διείσδυση των κρυπτονομισμάτων κι έχουν καταφέρει να ενσωματώσουν στην παγκόσμια οικονομία διάφορους "μικρόκοσμους". Για παράδειγμα, είναι γεγονός ότι 2.5 δισεκατομμύρια ενήλικες [34] δεν έχουν τραπεζικό λογαριασμό. Συνεπώς, γύρω στα 5 δισεκατομμύρια άνθρω-



Σχήμα 1.5: Merkle-Tree ιεραρχία από hashes.

ποι ανήκουν σε νοικοκυριά τα οποία είναι τελείως αποκομμένα από το χρηματοπιστωτικό σύστημα το οποίο εμάς μας θεωρεί δεδομένους. Οι άνθρωποι αυτοί δεν μπορούν να αρχίσουν έναν καταθετικό λογαριασμό. Δεν έχουν τρεχούμενο λογαριασμό. Μένουν σε μέρη όπου οι τράπεζες δεν θέλουν να πάνε, και γι' αυτό δεν μπορούν να συμμετάσχουν σε αυτό που ονομάζουμε παγκόσμια αγορά. Οι τράπεζες δεν πάνε εκεί, είτε επειδή δεν υπάρχουν οι υποδομές ή η ασφάλεια που χρειάζεται για να ανοίξει κάποιο φυσικό κατάστημα, είτε επειδή τα κέρδη που θα αποφέρουν θα είναι πολύ μικρά για να δικαιολογήσουν το κόστος, αλλά κυρίως, δεν πάνε εκεί λόγω της έλλειψης, ανυπαρξίας ή κατάρευσης νομικών θεσμών και μη ανεπτυγμένων ιδιοκτησιακών τίτλων. Εν έτει 2018, τέτοια σενάρια είναι πραγματικότητα σε εμπόλεμες ζώνες στη Μέση Ανατολή και τη Συρία ειδικότερα, οι αρκετές εμφύλιες διαμάχες σε πολλές χώρες της Αφρικής, αλλά κι εξαιρετικές περιπτώσεις όπως η Βενεζουέλα όπου κρυπτονομίσματα όπως το bitcoin έχουν κατά πολύ ανώτερα χαρακτηριστικά σε σχέση με το εθνικό νόμισμα! Στην Ελλάδα, το ενδιαφέρον για το bitcoin και τα κρυπτονομίσματα εκδηλώθηκε ιδιαίτερα το 2015, την περίοδο που η κυβέρνηση ανήγγειλε τους περιορισμούς κεφαλαίου και το κλείσιμο των τραπεζών.

1.2 Το "φαινόμενο" Ethereum

Υπό το πρίσμα αυτό, το Ethereum[11, 32] είναι το πιο γρήγορα αναπτυσσόμενο κρυπτονόμισμα τον τελευταίο καιρό και σίγουρα ότι πιο ενδιαφέρον έχουμε δει από τη δημιουργία του bitcoin και μετά. Πρόκειται περί μιας προγραμματιστικής πλατφόρμας η οποία συμπεριλαμβάνει μέσα της και το κρυπτονόμισμα **ether**. Η πλατφόρμα έχει την δική της γλώσσα προγραμματισμού και επιτρέπει ακόμα και σε απλούς χρήστες να φτιάξουν εφαρμογές πάνω στο blockchain και να είναι προσβάσιμες σε όλους μέσω διαδικτύου. Πρόκειται για την πιο αξιόλογη κι αξιέπαινη προσπάθεια που προσπαθεί να πάρει τα σκήπτρα από το bitcoin που συνεχίζει να είναι εξαιρετικά δημοφιλές και να επικρατεί στον τομέα, κυρίως χάρη στο αρκετά διαδεδομένο και γνωστό πλέον όνομα που έχει κάνει και την αξιοπιστία που έχει δείξει σαν λογισμικό και σαν δίκτυο. Μια αξιόπιστη πλέον υποδομή και τεχνολογία που λειτουργεί ανελειπώς από το 2008, όταν πρωτολανσαρίστηκε από την ομάδα που υπογράφει με το ψευδόνυμο Satoshi Nakamoto. Το "φαινόμενο" Ethereum όμως καλύπτει πλήρως τη λειτουργικότητα που προσφέρει το bitcoin και καταφέρνει να την υπερκεράσει! Οι δημιουργοί του είχαν τη σοφία να προσδώσουν προγραμματιστικά χαρακτηριστικά στην πλατφόρμα του ώστε να επιτρέψει την ανάπτυξη πολύπλοκων εφαρμογών και τη διασύνδεση ρόλων και χρηστών που διαφορετικά δε θα είχαν αυτήν τη δυνατότητα. Προβλέπουμε ότι με τα ανταγωνιστικά πλεονεκτήματα αυτά το Ethereum σύντομα θα λάβει την πιο περίοπτη θέση και θα παραμείνει εκεί για όσο διαρκέσει η ανάγκη και ζήτηση για blockchain πλατφόρμες.

Το υποκείμενο κρυπτονόμισμα **ether** βασίζεται στην τεχνολογία του blockchain, το οποίο επιτρέπει στους χρήστες να επικοινωνούν καθώς και να διενεργούν συναλλαγές στη μονάδα νομίσματος που υποστηρίζεται. Η διασφάλιση των δεδομένων βασίζεται σε συγκεκριμένες

Denomination	Amount (in Ether)
Wei	10^{-18}
Kwei	10^{-15}
Mwei	10^{-12}
Gwei	10^{-9}
Szabo	10^{-6}
Finney	10^{-3}
Ether	1
Kether	10^{3}
Mether	10^{6}
Gether	10^{9}
Tether	10^{12}

Πίνακας 1.1: Υποδιαιρέσεις του κρυπτονομίσματος Ether.

θεσπισμένες εγγυήσεις κι αρχές όπως οι διπλές χρεώσεις ή διπλές χρήσεις του ίδιου "κρυπτογραμματίου".

Η κατανεμημένη βάση στην οποία βασίζεται η εφαρμογή διατηρεί ενημερωμένη μία κατάσταση που συσχετίζει κάθε μονάδα νομίσματος στην κυκλοφορία με τον ιδιοκτήτη της. Στην πραγματικότητα, ο κάθε χρήστης χαρακτηρίζεται από μία διεύθυνση που λειτουργεί ως αναγνωριστικό. Για όσους έχουν εντρυφίσει στις αρχές της κρυπτογραφίας, η διεύθυνση αυτή λειτουργεί ως δημόσιο κλειδί το οποίο γνωστοποιείται στους συναλλασσόμενους, ενώ το ιδιωτικό κλειδί, εν είδει μυστικού κωδικού, είναι γνωστό μόνο στον κάτοχο του λογαριασμού ο οποίος είναι σε θέση να κάνει χρήση των μονάδων κρυπτονομίσματος που συσχετίζονται με το συγκεκριμένο λογαριασμό. Προφανώς στην κατοχή του κάθε χρήστη μπορούν να εμπεριέχονται άνω του ενός λογαριασμοί τους οποίους μπορεί να χρησιμοποιήσει προκειμένου να διενεργεί συναλλαγές με άλλους χρήστες ή να χρησιμοποιεί τις υπηρεσίες των άλλων χρηστών αλλά και του ιδίου που έχουν χτιστεί πάνω στην υποδομή του blockchain συστήματος.

Πιο συγκεκριμένα, οι υπηρεσίες τις οποίες δημιουργούν οι χρήστες του blockchain αποτελούν προγραμματιστικές εφαρμογές με δικό τους αποθηκευτικό χώρο που προσπελαύνουν ως υπαγορεύει η εφαρμογή, καθώς και το δικό τους υπόλοιπο σε μονάδες κρυπτονομίσματος το οποίο αυξάνεται ή μειώνεται με την αντίστοιχη αφαίρεση ή πρόσθεση αντίστοιχα, ισάξιας αξίας μονάδων κρυπτονομίσματος από τους λογαριασμούς των χρηστών που κάνουν χρήση της εφαρμογής. Δεν υπάρχουν περιορισμοί στους χρήστες που μπορούν να δημιουργούν εφαρμογές στην υποδομή του blockchain καθώς αυτή διατηρεί ένα ιστορικό που παραμένει αναλλοίωτο και δεν υπόκειται σε μετέπειτα περαιτέρω αλλαγές, ενώ ταυτόχρονα εμπλουτίζεται με νέα λειτουργικότητα που προσδίδουν ευελιξία και δυναμισμό. Πρόκειται για ιδιαίτερα σημαντικά χαρακτηριστικά των επονομαζόμενων smart-contracts, αφού συνεπάγεται έτσι την αμέριστη σοβαρότητα και προσοχή από την πλευρά των προγραμματιστών για πολλούς λόγους, μεταξύ των οποίων αναφέρουμε την κλοπή από επιτήδειους ή ακόμα κι ακούσια απώλεια από το υπόλοιπο της εφαρμογής μέρους των μονάδων κρυπτονομίσματος το οποίο αυτή διαχειρίζεται.

Μία έννοια η οποία πρωτοεισήχθει με το Ethereum είναι αυτή του gas. Σκοπός αυτής της παραμέτρου που συνοδεύει κάθε smart-contract που δημιουργείται στο blockchain είναι να αποτρέψει την υπερκατανάλωση πόρων του δικτύου εις βάρος των υπολοίπων αφού όλα μοιράζονται την ίδια υποδομή. Στην πραγματικότητα, αποτελεί το κόστος του δικαιώματος χρήσης της υποδομής (hardware, servers, mining) και θα μπορούσε να συγκριθεί με τα έξοδα εντολής, νομικά έξοδα, κι άλλες μορφές κοστολόγησης των παραδοσιακών οικονομικών ιδρυμάτων. Ο χρήστης που δημιουργεί μία συναλλαγή πρέπει να την αρχικοποιήσει με μία ποσότητα κρυπτονομίσματος που να αφιερωθεί για την παράμετρο αυτή. Κατά την εκτέλεση της συναλλαγής, η κάθε εντολή που εκτελείται καταναλώνει ένα ποσοστό αυτής της παραμέτρου. Εάν όμως η ποσότητα που έχει αφιερωθεί για αυτόν τον σκοπό δεν επαρκεί έως ότου η συναλλαγή ολοκληρωθεί, τότε η κατάσταση του smart-contract επαναφέρεται στην αρχική κι αναιρούνται όσες αλλαγές επέφερε η συναλλαγή πάνω στις μεταβλητές του

προγράμματος. Όμως, οι μονάδες κρυπτονομίσματος που ξοδεύτηκαν από τον χρήστη για το σκοπό αυτό δεν επιστρέφονται ποτέ! Όταν ένα smart-contract χρησιμοποιεί κάποια μέθοδο ενός άλλου smart-contract του blockchain, τότε ένα ποσοστό του διαθέσιμου gas περνάει με την κλήση στο άλλο smart-contract, ενώ η υπολοιπόμενη ποσότητα παραμένει για την μετέπειτα επεξεργασία. Εάν με την δευτερεύουσα κλήση δεν περάσει αρκετή ποσότητα τότε ούτε η αρχική μέθοδος θα ολοκληρωθεί. Εν γένει όμως, κατά την αποστολή κρυπτονομισμάτων από τη μία εφαρμογή στην άλλη, υπάρχει η δυνατότητα ορισμού συγκεκριμένων ποσών τόσο για την μεταφορά καθεαυτή, αλλά και για το διατιθέμενο gas που περνάει με την κλήση. Κατά σύμβαση, εάν αυτή η ποσότητα δεν ορισθεί ρητά (explicitly) τότε μεταφέρεται ολόκληρο το διατιθέμενο gas, πλην 25 wei.

1.3 Επικαιρότητα

1.3.1 Proof-of-Stake Consensus

Πρόκειται για το νέο μηχανισμό ο οποίος θα χρησιμοποιείται για την πιστοποίηση συναλλαγών με τη νέα γενιά blockchain, το **Ethereum 2.0 Serenity**. Η σχεδιαστική ομάδα του Ethereum σκοπεύει να το λανσάρει εντός του 2019 δημιουργώντας όχι απλά ένα hard fork του blockchain, αλλά επιπλέον σκοπεύουν να αποτρέψουν τους miners από το να συνεχίσουν να πιστοποιούν συναλλαγές για το Ethereum Classic και το Ethereum, τα ήδη γνωστά μας blockchains που χρησιμοποιούν το **ETHhash** PoW πρωτόκολλο, προκειμένου να επιταχύνουν την εδραίωση του **Casper** PoS μηχανισμού[10]. Τα προβλήματα τα οποία σκοπεύει να λύσει είναι τα εξής:

κλιμάκωσης όταν αυτή τη στιγμή υποστηρίζεται throughput που δεν ξεπερνάει τα 20 transactions/sec,

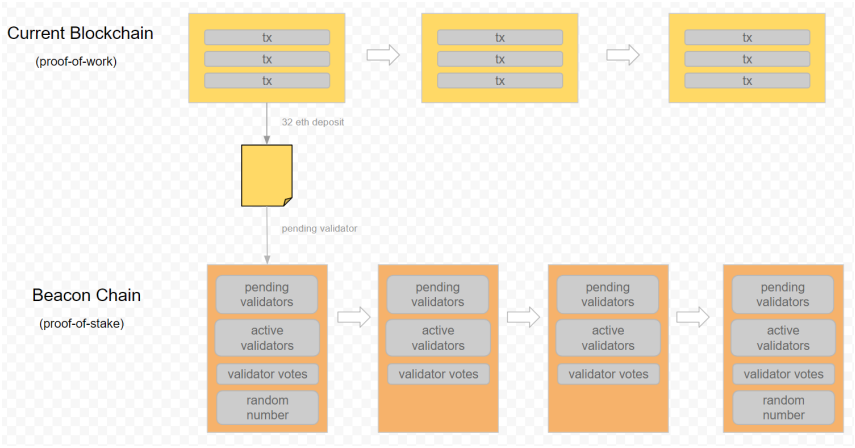
οικολογίας ως ένας μη ενεργοβόρος μηχανισμός,

ασφάλειας η απειλή του 51% hi-jack εξαφανίζεται.

Proof-of-Stake[29, 3, 7] πρωτόκολλα χρησιμοποιούνται ήδη σε λιγότερο δημοφιλή blockchains όπως το NXT, το PeerCoin, το BlackCoin και το ShadowCoin, τα οποία διαφημίζονται ως φιλικά προς το περιβάλλον λόγω του ότι απαιτούν mining, κι άρα δεν έχουν υπολογιστικό κόστος. Ενδεικτικό το ότι το ετήσιο mining κόστος για το bitcoin ανέρχεται στα $1,423,794,674$ USD. Ειδικότερα, ο μηχανισμός αυτός που σκοπεύει σύντομα να υιοθετήσει το Ethereum 2.0, πιστοποιεί την εγκυρότητα μίας συναλλαγής απλά ελέγχοντας την εγκυρότητά της (π.χ. double-spending) όπου ο **forger**[1] θα παρέχει οικονομική εγγύηση μία ποσότητα κρυπτονομίσματος ως εχέγγυο, της οποίας το μέγεθος θα καθορίζεται από τον πιο "γενναιόδωρο" εγγυητή, το επονομαζόμενο **stake**. Βέβαια, αν ο έλεγχος αυτός δεν είναι ακριβής ή ο εγγυητής επιλέξει να ψεύδεται εξυπηρετώντας δικούς του σκοπούς, τότε η εγγύηση που έχει ήδη δεσμευτεί από τους πόρους του αφαιρείται από το λογαριασμό του και χάνεται, κι άρα το αντικίνητρο γίνεται άμεσα σαφές και κατανοητό! Κάτι πολύ παρόμοιο συμβαίνει παραδοσιακά άλλωστε με τον εγγυητή ενός δανείου όταν ο δανειολήπτης δεν είναι σε θέση να αποπληρώσει το δάνειό του. Τότε ο εγγυητής καλείται να καλύψει το διαφιλουνικόμενο ποσό με δικούς του πόρους, αν και το δάνειο (ether transactions στην περίπτωσή μας) δεν τον αφορούσε.

Εκτός από το τεράστιο αντικίνητρο που εμποδίζει την οποιαδήποτε ροπή προς την απάτη, με κάθε επιτυχή επιβεβαίωση υπάρχει οικονομική επιβράβευση σε μορφή κρυπτονομίσματος, όπως άλλωστε γίνεται με το PoW. Όμως, εκεί οι miners διαγωνίζονταν αναμεταξύ τους για το ποιος θα καταφέρει πρώτος να βρεί το σωστό nonce πεδίο που επιβεβαιώνει το block. Εδώ, δεδομένου ότι δεν υπάρχει τέτοιου είδους ανταγωνισμός, γίνεται επιλογή του forger με συγκεκριμένα κριτήρια. Τα χαρακτηριστικά αυτά ακριβώς είναι αυτά που καθορίζουν και διαφοροποιούν το ένα PoS μηχανισμό έναντι του άλλου! Όπως ήδη προαναφέραμε, ένα σημαντικό κριτήριο βάσει του οποίου γίνεται η επιλογή αυτή είναι από το ύψος του stake. Κάτι πολύ λογικό αφού όσο υψηλότερο είναι το υπό εγγύηση ποσό, τόσα πιο πολλά

[1] ο νέος όρος για τον κόμβο του EVM που θα παίζει το ρόλο του miner.



Σχήμα 1.6: Συνεργασία PoS-PoW μηχανισμών στο Ethereum 2.0.

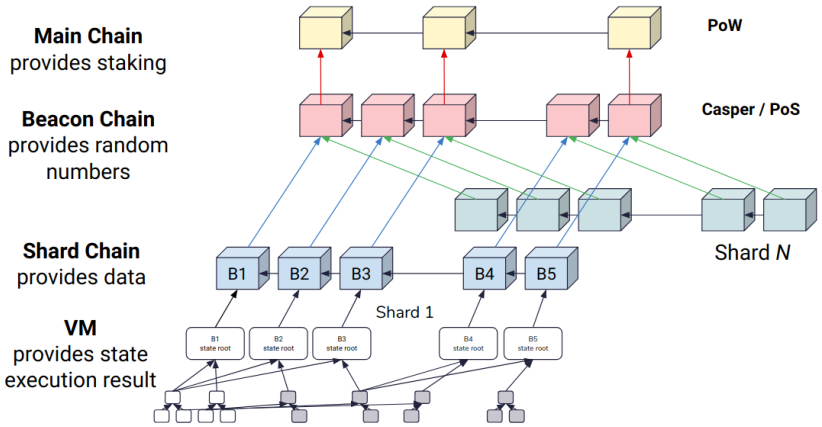
κρυπτονομίσματα θα χαθούν αν η πιστοποίηση κριθεί ως ψευδής κι ακυρωθεί, κι άρα εξαιρετικά ασύμφωρη μία τέτοια κίνηση. Βέβαια, αν αυτό ήταν το μοναδικό κριτήριο επιλογής, τότε θα δημιουργούνταν ένας φαύλος κύκλος σύμφωνα με τον οποίον μονόν οι πλούσιοι forgers θα πιστοποιούν συναλλαγές για να συλλέγουν νεοεκδοθείσες μονάδες κρυπτονομισμάτων και transaction fees.

Κι έτσι γίνεται προφανής η κρισημότητα των δευτερευόντων κριτηρίων επιλογής. Ο **Randomized Block Selection** μηχανισμός προμοτάρει μεταξύ των υψηλότερων εγγυήσεων τους χρήστες με συγκεκριμένα κριτήρια βάσει της διεύθυνσής τους, π.χ. το χαμηλότερο αριθμό. Κάτι παρόμοιο συμβαίνει με τα αυτοκίνητα που επιτρέπεται να κυκλοφορούν στο κέντρο της Αθήνας βάσει του λήγοντος ψηφίου, μονό ή ζυγό. Κι άρα ένα ψευδο-τυχαίος τρόπος επιλογής. Ο **Coin Age Selection** μηχανισμός επιλέγει ως forgers τους υποψηφίους που προσφέρουν τις λιγότερο πρόσφατες εγγυήσεις. Υπάρχει ακόμα η δυνατότητα χρήσης συνδυασμού των ανωτέρω μηχανισμών με υβριδικό τρόπο. Ένα επιθυμητό χαρακτηριστικό των PoS μηχανισμών είναι η γραμμικότητα στην πιθανότητα να εισάγεις το επόμενο block βάσει του πόσο έχεις επενδύσει σε αυτήν την πράξη. Κάτι τέτοιο δεν ισχύει για τους PoW μηχανισμούς, αφού ένας ισχυρότερος υπολογιστικά κόμβος θα είναι πάντα ισχυρότερος και θα είναι εκθετικά πιθανότερο να εξάγει το σωστό nonce που πιστοποιεί τη συναλλαγή.

Ως αντιστάθμισμα σε όλα αυτά τα οφέλη, αξίζει να αναφέρουμε ότι γίνεται πιο δύσκολη η σύγκλιση σε μία και μοναδική πραγματικότητα blockchain. Όπως είδαμε ήδη στην Ενότητα §1.1.2, λόγω της μεγάλης κλιμάκωσης του blockchain, υπάρχει η πιθανότητα να δημιουργηθούν άνω της μία πραγματικότητες λόγω της διαφορετικής σειράς με την οποία υπεισέρχονται δύο ή και περισσότερα φρεσκο-πιστοποιημένα blocks. Η όποια αμφισημία όμως αντιπαρέρχεται όταν η μία αλυσίδα γίνεται μακρύτερη από τις άλλες, κι οπότε κυριαρχεί ως προς τις άλλες εκτοπίζοντάς τες, αφού ο κάθε κόμβος επιλέγει να κάνει mining αποκλειστικά για την μακρύτερη αλυσίδα που γνωρίζει, διαφορετικά για να υποστηρίξει και τις υπόλοιπες πραγματικότητες θα πρέπει να μοιράσει ανάλογα την επεξεργαστική του ισχύ, κάτι που δεν τον συμφέρει αφού έτσι θα προλαβαίνουν να εισπράτουν την ανταμοιβή άλλοι miners. Όταν όμως δεν υπεισέρχεται το λεπτό αυτό σημείο, υπάρχει η δυνατότητα προκειμένου να πολλαπλασιάσουν τα κέρδη τους οι forgers να δημιουργούν πολλαπλούς συνδυασμούς από τα νεοεισερχόμενα blocks. Το πρόβλημα αυτό ονομάζεται **Nothing at Stake** και δεν επιτρέπεται για την PoS έκδοση του Ethereum 2.0, το επονομαζόμενο **Casper** πρωτόκολλο, συνοδευόμενο με το ίδιο αντικίνητρο της ακύρωσης της πιστοποίησης.

1.3.2 ETheist

Το ρητό "μία αλυσίδα είναι τόσο δυνατή, όσο ο πιο αδύνατος κρίκος" είναι γνωστό σε όλους. Όμως, ακριβώς στο σημείο αυτό στοχεύουν εξειδικευμένοι επιτήδειοι προκειμένου να υπεξαιρέσουν τεράστια ποσά κεφαλαίων από λογαριασμούς χρηστών κι εφαρμογών στο



Σχήμα 1.7: Μηχανισμός Sharding στο Ethereum 2.0.

blockchain. Οι επιπτώσεις τεράστιες και με σημασία πέρα από της οικονομικής διαστάσης της ληστείας. Όπως θα δούμε και παρακάτω, από το 2015 όπου πρωτολανσαρίστηκε το Ethereum λαμβάνει χώρα περίπου ένα σοβαρό περιστατικό ανά χρόνο. Αξιοσημείωτο το ότι τα κενά ασφαλείας βρίσκονταν πάντα από την πλευρά των smart-contracts κι όχι στην υποδομή του blockchain. Σχεδιαστικές οδηγίες προς ναυτηλομένους έχουν κατά καιρούς προταθεί, όπως στο [15], όμως δε φαίνονται να τηρούνται πάντα.

Το CrowdFunding ως HackerFunding

Το Decentralized Autonomous Organization (DAO) λειτουργούσε ως venture capital με σκοπό την χρηματοδότηση κινήσεων που αφορούσαν τεχνολογίες blockchain και κρυπτονομισμάτων εν γένει. Η απουσία κεντρικής αρχής και παραδοσιακών ιδρυμάτων θα παρείχε περισσότερο έλεγχο κι άμεση πρόσβαση στα κεφάλαια τόσο από τους επενδυτές, όσο κι από τους επωφελούμενους. Αλλά όλα αυτά στη θεωρία!

Μόλις το Μάη του 2016, η κοινότητα του Ethereum έχοντας κλείσει ένα χρόνο ζωής ήδη, ανακοινώνει την έναρξη του **Genesis DAO**, το οποίο υποστηριζόταν από ένα smart-contract που λειτουργούσε πάνω στο Ethereum. Η διαδικασία είχε ως εξής: για ένα συγκεκριμένο διάστημα θα μπορούσε να δεχτεί συμμετοχές από οποιονδήποτε χρήστη προκειμένου να ανταλλάξει ETH προς DAO tokens σε μία αναλογία 1-100! Η επιτυχία ήταν άνευ προηγουμένου με το συνολικό ποσό να αντιστοιχεί στο ιλιγγιώδες 150,000,000 USD (12,700,000 ETH).

Όλα αυτά μέχρι την 17η Ιουνίου του 2016, όπου ένας άγνωστος μέχρι στιγμής black hat hacker (συμπληρώνονται δύο χρόνια ήδη) βρίσκει τον τρόπο να αντλήσει κεφάλαια από το fund αυτό του DAO[28]. Μόλις τις λίγες πρώτες ώρες της κυβερνο-επίθεσης καταφέρνει να συλλέξει ποσό που αντιστοιχούσε τότε σε 70,000,000 USD (3,600,000 ETH)! Ο άγνωστος cracker κατάφερε χρησιμοποιώντας τον ίδιο τον πηγαίο κώδικα του smart-contract που είχε γράψει κι ανακοινώσει το Stock.it να στείλει μεν τη συνεισφορά του, κι έπειτα να την πάρει πάλι πίσω, όχι όμως μόνο μία φορά, αλλά όσες ήθελε μέσω μίας αναδρομικής κλήσης που δεν είχε προβλεφθεί.

Όταν έγινε αντιληπτή η κλοπή, το υπόλοιπο του fund κλειδώθηκε για 28 ημέρες για προστασία. Επιπλέον, προκειμένου να αποζημιωθεί η απώλεια, δημιουργήθηκε ένα νέο hard fork του blockchain χωρίς τις επιζήμιες συναλλαγές, ενώ το παλιό blockchain συνεχίζει να υπάρχει ακόμα με την ονομασία Ethereum Classic. Και κάπως έτσι ξεκίνησε το τέλος του DAO...

Parity wallet pwned

Ένα κενό ασφαλείας[24] που βρέθηκε στην εφαρμογή του Parity Multisig Wallet έκδοση 1.5+, επέτρεψε σε έναν ακόμα cracker να οικειοποιηθεί πάνω από 153,037 ETH ($> 30,000,000$

USD) πριν περίπου έναν χρόνο, την 19η Ιουλίου. Το ποσό αυτό υπεξαιρέθηκε από τρεις multi-signature λογαριασμούς που χρησιμοποιόντουσαν για κεφάλαια από πωλήσεις tokens. Ο cracker αρχικά απέκτησε αποκλειστική πρόσβαση στον MultiSig λογαριασμό χρησιμοποιώντας ένα παραθυράκι σαν αυτό της γραμμής 5, του Σχήματος 1.8, που του επέτρεψε να χρίσει τον εαυτό του κυρίαρχο της εφαρμογής και να μεταφέρει τα κεφάλαια σε ανώνυμο λογαριασμό του ιδίου με μία δεύτερη κλήση. Το κενό ασφαλείας του επέτρεπε να καλεί οποιαδήποτε μέθοδο της βιβλιοθήκης της πλατφόρμας επιθυμούσε, αντί να την είχαν θωρακίσει κατάλληλα.

```
1:     function() payable {
2:         if (msg.value > 0)
3:             Deposit(msg.sender, msg.value);
4:         else if (msg.data.length > 0)
5:             _walletLibrary.delegatecall(msg.data);
6:     }
```

Σχήμα 1.8: Ευκολίες στον πηγαίο κώδικα που αποβαίνουν μοιραίες.

Μετά την επίθεση αυτή ελήφθησαν κάποια ανεπαρκή μέτρα τα οποία προσπεράστηκαν με αστείο τρόπο[25] κι οδήγησαν σε μία ακόμα πιο επιτυχή επίθεση το Νοέμβρη του ιδίου χρόνου με το ποσό που εκλάπη να κυμαίνεται περί τα 150,000,000 USD.

Υπόθεση EOSBet

Πολύ παρόμοια με την προηγούμενη η μόλις λίγων εβδομάδων κλοπή αξίας 200,000 USD από τη στοιχηματική πλατφόρμα EOSBet του EOS blockchain (δεν πρόκειται για το Ethereum αυτήν τη φορά). Οι crackers αυτοί τη φορά δημιούργησαν ένα λογαριασμό που έμοιαζε με της πλατφόρμας και χρησιμοποίησαν ένα ψεύτικο hash. Συνέλλεξαν μικρά ποσά από πολυάριθμους λογαριασμούς σε συναλλαγές που εμπεριείχαν αποπροσανατολιστικά μηνύματα για την ταυτότητα του επωφελούμενου[13].

Σχήμα 1.9: Ponzi scheme DApp στο Ethereum.

333 ETH: Oldest trick in the book

Πρόκειται για την τρίτη πιο δημοφιλή εφαρμογή στο Ethereum αμέσως μετά τα CryptoKitties και λειτουργεί ως εξής[1, 19]: από τη συμμετοχή του κάθε χρήστη χρησιμοποιεί το 17% των κρυπτονομισμάτων του χρήστη για τους παρακάτω σκοπούς, 1% για τους επενδυτές, 2% έξοδα εντολής, 3% τεχνική υποστήριξη, και 11% marketing. Για το υπόλοιπο 83% θα λαμβάνει ες αεί ο χρήστης κάθε 24 ώρες 3.33% αποζημίωση-επιβράβευση. Όπως και σε όλα τα

συστήματα πυραμίδας, από τον καιρό του Ponzi στην Αμερική, για όσο διάστημα συμμετέχουν ολοένα και περισσότεροι νέοι χρήστες και προσφέρουν φρέσκα κεφάλαια δε θα υπάρχει κανένα πρόβλημα αποπληρωμής. Μέχρις στιγμής τα θύματα(;) αυτής της απάτης που δεν έχει φτάσει ακόμα στο κραχ απαριθμούν τους 1,600.

1.3.3 StableCoins

Τα stablecoins[18] είναι κρυπτονομίσματα των οποίων οι τιμές συνδέονται είτε με σταθερή αναλογία με αξίες της πραγματικής οικονομίας, όπως ο χρυσός ή κάποιο εθνικό νόμισμα, π.χ. USD, κι εκδίδονται από κάποια κεντρική αρχή, είτε από Decentralized Autonomous Organization (DAO) το οποίο ελέγχει την έκδοσή του και την τιμή του.

Η κεντρική αρχή εκδίδει νομίσματα σύμφωνα με την παρεχόμενη εγγύηση (collateral) που παρέχεται, επιτρέποντας έτσι την αύξηση της κυκλοφορίας του κρυπτονομίσματος καθώς αυξάνεται το απόθεμα. Παράδειγμα τέτοιου κρυπτονομίσματος είναι το Tether[31].

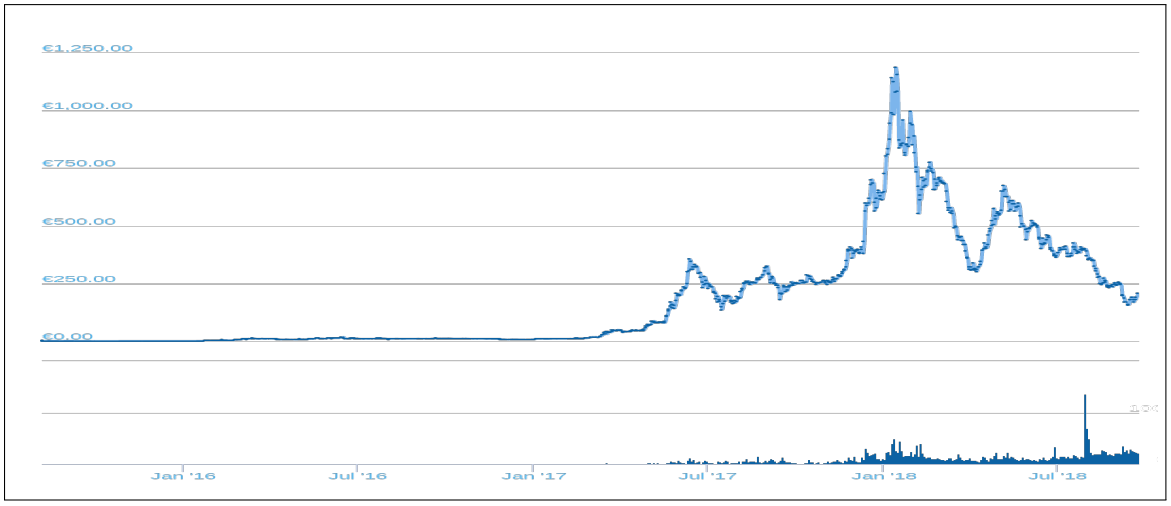
Τα οφέλη που προέρχονται με τη διαχείριση από το DAO έχουν να κάνουν κυρίως με τη διαφάνεια, όπως για παράδειγμα το Dai που χρησιμοποιεί την πλατφόρμα του Ethereum, ή το NuBits το οποίο είναι υβριδικό αφού αν και βασίζεται σε DAO, ελέγχεται επιπλέον από μία κεντρική αρχή. Τα stablecoins μπορεί να αποβούν προβληματικά εξαιτίας απορυθμιστικών παραγόντων ή λόγω ελλείψεων στο απαραίτητο απόθεμα που λειτουργεί ως εχέγγυο. Κάπως έτσι κατέρευσε το NuBit που δεν κατάφερε να κρατήσει την ένα-προς-ένα αναλογία με το δολάριο κι έχει φθάσει να υποτιμηθεί πλέον στο $\frac{1}{10}$ της αρχικής ισοτιμίας.

Το Tether[31] χρησιμοποιείται ευρέως σε crypto-to-crypto ανταλλακτήρια καθώς επιτρέπει την τιμολόγηση κρυπτονομισμάτων σε USD χωρίς την άμεση συνδιαλλαγή με παραδοσιακά τραπεζικά ιδρύματα, όταν πολλές φορές η ρευστοποίηση κρυπτονομισμάτων δεν είναι εύκολη. Προβλήματα στην αγορά του Tether θα μπορούσαν να προκαλέσουν domino effect και να διαταράξουν την ομαλότητα σε πολύ μεγάλο μέρος της κοινότητας λόγω του κεντρικού αυτού ρόλου στη διασύνδεση με την πραγματική οικονομία, ή ακόμα και να διαρρήξει εντελώς αυτή τη σχέση σε μία ενδεχόμενη κατάρευση της ισοτιμίας του.

Όσον αφορά νέες παρόμοιες κινήσεις[12], η πολιτεία της Νέας Υόρκης ενέκρινε μόλις πριν από ένα μήνα (17 Σεπτεμβρίου) τα κρυπτονομίσματα των ιδρυμάτων Gemini Trust και Paxos Trust. Πρόκειται για τα πρώτα κρυπτονομίσματα τα οποία εγκρίνει τοπική ρυθμιστική αρχή. Πίσω από το Gemini dollar βρίσκονται οι Cameron και Tyler Winklevoss, γνωστοί από τη διαμάχη πατρότητας του Facebook. Πρόκειται για ένα stablecoin το οποίο θα επιτρέψει στους χρήστες του Ethereum να αποστέλουν και να λαμβάνουν αμερικάνικά δολάρια, καθώς θα έχει μία αυστηρή ένα-προς-ένα ισοτιμία. Το καταπίστευμα θα διατηρείται σε αμερικανική τράπεζα και θα υπόκειται σε ελέγχους από τις ομοσπονδιακές αρχές των Η.Π.Α. Οι Winklevoss σκοπεύουν με τον τρόπο αυτόν να τραβήξουν ακόμα και τους πιο δύσπιστους επενδυτές παρέχοντας τέτοιου τύπου εχέγγυα!

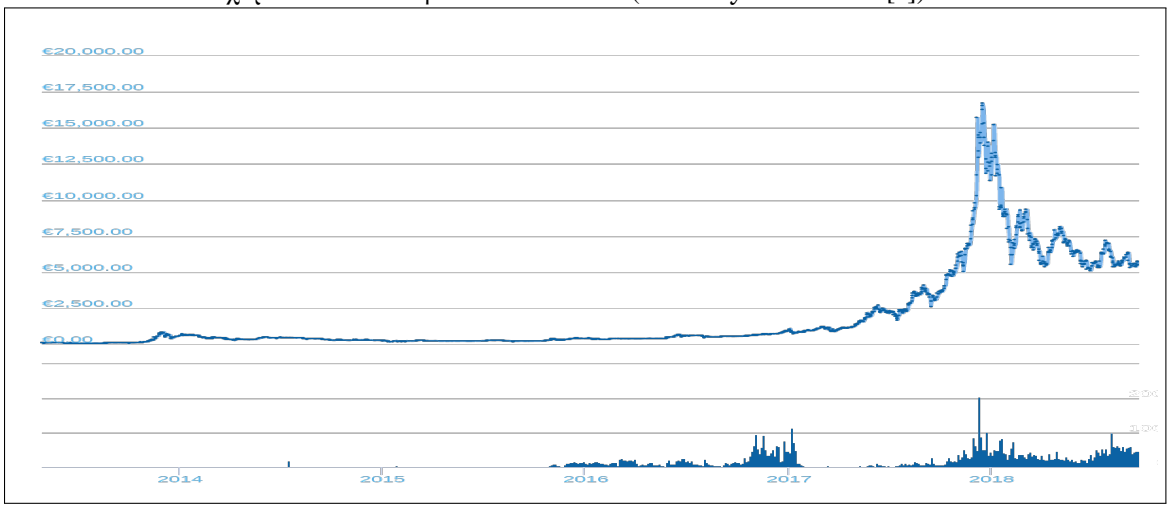
1.4 Κίνητρα και σκοποί

Έχοντας συζητήσει εκτενώς και σε κάποιο βάθος την περιοχή που πραγματεύεται η εργασία αυτή, θα περιγράψουμε εν συντομία για το υπόλοιπο του κεφαλαίου το τεχνικό αντικείμενο της εργασίας αυτής. Σκοπός μας είναι ο σχεδιασμός κι η υλοποίηση ενός ηλεκτρονικού αποθετηρίου εγγράφων τα οποία θα είναι διαθέσιμα είτε για ανάγνωση, είτε και για τροποποίηση σύμφωνα με τα δικαιώματα χρήσης που προδιαγράφονται ξεχωριστά για κάθε κατηγορία χρηστών με κάθε έγγραφο. Με τον τρόπο αυτό ο κάθε χρήστης είναι σε θέση να εμπιστευτεί όχι μόνον το έγγραφο καθεαυτό, αλλά και τις επεμβάσεις τις οποίες έχει υποστεί, καθώς προέρχονται από πηγές οι οποίες είναι εξουσιοδοτημένες να μεταβάλλουν το περιεχόμενό του με τρόπο ο οποίος ενδεχομένως και να διαφέρει κι από έγγραφο σε έγγραφο. Προκειμένου να επιτύχουμε την επιδιωκόμενη ευελιξία που περιγράφουμε ξεχωρίζουμε τους χρήστες σε τρεις βασικές κατηγορίες: (i) τον απλό χρήστη που μπορεί να έχει πρόσβαση στο blockchain, (ii) μία ειδική ομάδα χρηστών των οποίων οι διευθύνσεις υπάρχουν αποθηκευμένες με κάποιο τρόπο στο αποθετήριο ώστε να επεξεργάζεται τα ειδικά τους προνόμια



Σχήμα 1.10: Ether price in euros (credit by CoinGecko[2]).

Σχήμα 1.11: Ether price in bitcoins (credit by CoinGecko[2]).



Σχήμα 1.12: Bitcoin price in euros (credit by CoinGecko[2]).

και τα ανώτερα προνόμια που έχουν πάνω στο περιεχόμενο των εγγράφων, π.χ. τροποποίησης, και τέλος, (iii) τον διαχειριστή του αποθετηρίου που έχει διαφορετικές αρμοδιότητες κι ευθύνες κι είναι σε θέση να αλλάζει δυναμικά τη σύσταση της ειδικής ομάδας χρηστών, κι άρα απαιτείται η πιστοποιημένη προσθαφαίρεση χρηστών στην ειδική ομάδα, π.χ. ένας διαγραφέντας να προσπαθήσει να επαναφέρει τα παλιά του προνόμια που έχει απωλέσει, ή να αλλάξει τον διαχειριστή χωρίς να έχει τέτοια αρμοδιότητα. Οπότε το distributed ledger του blockchain παρέχει πιστοποιήσεις όχι μόνο για τα έγγραφα τα οποία αποκτούν το δικό τους lifecycle από την ανάρτησή τους κι έπειτα, αλλά και για τους χρήστες και τις ενδεχόμενες υπερεξουσίες ισχυριστούν προκειμένου να αποκτήσουν πρόσβαση και να μεταβάλλουν το περιεχόμενο ευαίσθητων, διαβαθμισμένων ή και όχι εγγράφων. Στα αυτονόητα θεωρούμε την αποτελεσματική θωράκιση της εφαρμογής από hackers, crackers και κάθε άλλης λογής επιτήδειους καθώς το smart-contract κοινοποιεί μία διεπαφή αποτελούμενη από ένα σύνολο μεθόδων και κοινών μεταβλητών που είναι σε θέση να καταστήσουν την εφαρμογή ευάλωτη σε διαφόρων ειδών επιθέσεις και να αχρηστεύσουν τις πιστοιποιήσεις χρηστών, εγγράφων και των μεταβολών τους με ποικίλους τρόπους.

Στα προνόμια της κοινότητας του Ethereum blockchain θεωρούμε βιβλιοθήκες όπως την Web3.js και EthereumJS οι οποίες είναι σε θέση να δώσουν τρομερή ώθηση στις εφαρμογές που γίνονται διαθέσιμες στο Ethereum. Έτσι, οι προγραμματιστές μπορούν για το ίδιο smart-contract να κατασκευάσουν και να κοινοποιήσουν διαφορετικές γραφικές διεπαφές σαν να επρόκειται για ξένες μεταξύ τους δικτυακές εφαρμογές. Το διαφορετικό κοινό που αυτές απευθύνονται θα ήταν ένα τέτοιο κίνητρο φυσικά. Εμείς, γοητευμένοι από το full-stack μοντέλο που προσφέρει η έμπειρη κοινότητα του Ethereum θα κάνουμε χρήση της Web3.js βιβλιοθήκης μέσα από την client-side εφαρμογή που αναπτύσσουμε με την MVVM πλατφόρμα του AngularJS framework για το χτίσιμο της γραφικής διεπαφής μας, δίνοντας έτσι άμεσα νόημα σε όλο το προηγούμενο μέρος της εργασίας που αφορούσε την κατασκευή του backend κομματιού.

Τέλος, το υπόλοιπο του βιβλίου που κρατάτε στα χέρια σας είναι οργανωμένο ως εξής: στο Κεφάλαιο 2 παρουσιάζουμε τις προδιαγραφές και τη διεπαφή του smart-contract που αναπτύσσουμε ώστε να λειτουργήσει ως ένα ηλεκτρονικό αποθετήριο, στο Κεφάλαιο 3 δίνουμε υλοποιήσεις του smart-contract που περιγράψαμε προηγουμένως σε δύο προγραμματιστικές γλώσσες, (i) Serpent, και (ii) Solidity. Στο Κεφάλαιο 4 περιγράφουμε τη γραφική διεπαφή που κατασκευάσαμε σε HTML και JavaScript με το MVVM AngularJS framework, στο Κεφάλαιο 5 κάνουμε μία σύντομη ανασκόπηση της δουλειάς μας και της συνεισφοράς μας και συζητάμε τους άξονες γύρω από τους οποίους θα εκτυλιχθούν τα μελλοντικά μας βήματα. Ακόμα, παραθέτουμε για αυτούς που επιθυμούν να εντρυφίσουν περισσότερο στην τεχνολογία του Ethereum το Παράρτημα .1 με μία σύντομη περιγραφή του μοντέλου και της αρχιτεκτονικής του, ενώ στο Παράρτημα .3 θα βρείτε μία σύντομη τεχνική περιγραφή της προγραμματιστικής γλώσσας Solidity, και κατ' αντιστοιχία για Serpent στο Παράρτημα .11.

Κεφάλαιο 2

Προδιαγραφές προγραμματιστικής διεπαφής

The blockchain is an incorruptible digital ledger
of economic transactions that can be programmed
to record not just financial transactions
but virtually everything of value.

Don & Alex Tapscott, Blockchain Revolution (2016)

Στο κεφάλαιο αυτό σχεδιάζουμε και περιγράφουμε τη διεπαφή που επιθυμούμε η εφαρμογή να υποστηρίζει βασιζόμενοι στις ανάγκες που θέλουμε να καλύπτει. Η διεπαφή του smart-contract που γίνεται διαθέσιμη από την εφαρμογή (Σχήμα 2.1) απαρτίζεται από δώδεκα μεθόδους εν συνόλω. Συνοπτικά έχουμε,

- μία μέθοδο για την αλλαγή διαχειριστή του smart-contract,
- δύο για τη διαχείριση των χρηστών από το διαχειριστή του contract,
- δύο για την ανάκτηση της λίστας των αναγνωριστικών των εγγράφων σύμφωνα με τα δικαιώματα προσπέλασης και τροποποίησης του χρήστη,
- μία για την προσθήκη νέων εγγράφων από το διαχειριστή του contract,
- μία για τη μεταβολή των δικαιωμάτων των χρηστών ανά έγγραφο από το διαχειριστή του contract, και
- πέντε ακόμα για τη διαχείριση των αναρτημένων εγγράφων από τους χρήστες του contract, σύμφωνα πάντα με τα δικαιώματα που έχει ορίσει εκ των προτέρων ο δημιουργός του contract,
  - για την αφαίρεση ενός εγγράφου εφόσον ο χρήστης έχει δικαιώματα τροποποίησής του,
  - για τη μετονομασία ενός εγγράφου εφόσον ο χρήστης έχει δικαιώματα τροποποίησής του,
  - για την μεταβολή περιεχομένου (προσθήκη/διαγραφή) σε ένα έγγραφο εφόσον ο χρήστης έχει δικαίωμα τροποποίησής του,
  - για την προβολή περιεχομένου εφόσον ο χρήστης έχει δικαίωμα ανάγνωσης.

```
 0. chown (string user_id)

 1. add_user (string user_id)
 2. del_user (string user_id)
 3. add_doc (string doc_id, string content, int flags)
 4. chmod (string doc_id, int new_flags)

 5. get_readable_docs ()
 6. get_writable_docs ()

 7. rmv_doc (string doc_id)
 8. rnm_doc (string doc_id_old, string doc_id_new)
 9. edit_insert (string doc_id, int start, string new_content)
10. edit_delete (string doc_id, int start, int length)
11. get_content (string doc_id)
```

Σχήμα 2.1: Μέθοδοι διεπαφής εφαρμογής.

2.1 Παραχώρηση-εκχώρηση ιδιότητας διαχειριστή

Ο αρχικός χρήστης, ο οποίος κάνει deploy την εφαρμογή στην blockchain πλατφόρμα του Ethereum, αρχικοποιεί το smart-contract με τη διεύθυνσή του κι έχει περισσότερα προνόμια ως διαχειριστής. Αυτά του επιτρέπουν να ορίσει τους χρήστες που θα απαρτίζουν την ειδική ομάδα χρηστών, να μεταβάλλει την ομάδα αυτή με προσαφθαίρεση χρηστών, να αναρτήσει καινούρια έγγραφα με νέο περιεχόμενο τα οποία από εκεί και πέρα ακολουθούν τον δικό τους κύκλο ζωής μετά από αλληλεπίδραση κι editing από τους χρήστες οι οποίοι έχουν δικαιώματα προσπέλασης και τροποποίησης σε αυτά. Επιπλέον, ο διαχειριστής, λόγω του ρόλου του και των αρμοδιοτήτων του, επιβαρύνεται με περαιτέρω ευθύνες αφού θα πρέπει να παρακολουθεί και να εντοπίζει πιθανές κακόβουλες πράξεις και να τις αντιμετωπίζει αναλόγως.

Οι αρμοδιότητες, τα προνόμια αλλά κι οι ευθύνες που συνδέονται με τον συγκεκριμένο ρόλο μπορούν να περαστούν από τον διαχειριστή σε άλλον χρήστη, που θα χρίσει ως νέο διαχειριστή ο τωρινός, με την χρήση μίας και μόνο μεθόδου που δέχεται ως όρισμα τη διεύθυνση του νέου διαχειριστή. Η κλήση της επιτρέπεται από τον τωρινό διαχειριστή και μόνον ενώ το αποτέλεσμά της δεν μπορεί να αναστραφεί παρά μόνον με νέα κλήση της μεθόδου από τον νέο διαχειριστή για επαναφορά του παλαιού. Επομένως, η αποποίηση της ιδιότητας αυτής είναι μή αναστρέψιμη διαδικασία.

2.2 Προσθήκη νέου χρήστη

Η μέθοδος αυτή χρησιμοποιείται για τον σχηματισμό της ειδικής ομάδας χρηστών του smart-contract. Η χρήση της περιορίζεται στο διαχειριστή του contract και προσθέτει σε ειδικές δομές αφιερωμένες για το σκοπό αυτόν την διεύθυνση του νέου χρήστη, η οποία δίνεται ως μοναδικό όρισμα, για μελλοντική χρήση. Απαραίτητη η συνέπεια και το μόνιμο αποτέλεσμά της, δηλαδή εφόσον ένας χρήστης προστίθεται στην ομάδα δεν αφαιρείται παρά μόνον μετά την κλήση της αντίστροφης μεθόδου αφαίρεσης χρήστη που περιγράφουμε εν συνεχεία.

Σε περίπτωση αστοχίας της μεθόδου (π.χ. το υπόλοιπο gas δεν επαρκεί) ή μη εξουσιοδοτημένος χρήστης δοκιμάζει να την εκτελέσει, θα επιστρέφεται σχετικό μήνυμα και παράλληλα θα "λογκάρεται" η αποτυχημένη ενέργεια προκειμένου αφενός ο αφελής χρήστης να ενημερωθεί για το ότι δεν είναι αρμόδιος, αφετέρου ο διαχειριστής να ενημερωθεί για τις κακόβουλες

ενέργειες και να δράσει κατάλληλα. Άλλωστε είναι εξαιρετικά αναμενόμενο από απλούς χρήστες του blockchain που δεν έχουν πρόσβαση σε συγκεκριμένα αποθηκευμένα έγγραφα βάσει των δικαιωμάτων τους, να δοκιμάσουν να αναβαθμίσουν τους εαυτούς τους προκειμένου να επιτύχουν τους σκοπούς τους.

2.3 Αφαίρεση υπάρχοντος χρήστη

Και η μέθοδος αυτή επιτρέπεται να εκτελεστεί μόνον από τον διαχειριστή κι αφαιρεί (εφόσον υπάρχει) τον χρήστη του οποίου η διεύθυνση δίνεται ως μοναδικό όρισμα από την ειδική ομάδα χρηστών του smart-contract, καθώς αναλαμβάνει την εκκαθάριση των σχετικών δομών από τη διεύθυνση του συγκεκριμένου χρήστη. Και πάλι απαραίτητη η συνέπεια και το μόνιμο αποτέλεσμά της.

Όσον αφορά αποτυχημένες ενέργειες, αυτές θα επιστρέφουν ένα σχετικό μήνυμα και θα "λογκάρουν" την ενέργεια προκειμένου να προσανατολιστεί καλύτερα ο χρήστης ως το προς το τι του επιτρέπεται να κάνει και ποιες μεθόδους είναι σε θέση να εκτελέσει, αλλά και να μπορεί να μελετηθεί από το διαχειριστή το pattern ενός κακόβουλου χρήστη και να τον αποκλείσει ή να το διευθετήσει με κάποιον άλλον κατάλληλο τρόπο. Η μέθοδος αυτή θα μπορούσε να χρησιμοποιηθεί από κάποιον κακόβουλο χρήστη που θέλει να διαταράξει τη λειτουργικότητα της εφαρμογής και να διαγράψει κάποια μέλη της ειδικής ομάδας χρηστών.

2.4 Προσθήκη νέου εγγράφου

Προφανώς μία από τις πιο αναγκαίες μεθόδους είναι αυτή της ανάρτησης εγγράφων στο contract. Αυτή παίρνει ως ορίσματα το αναγνωριστικό του εγγράφου, το περιεχόμενο, καθώς και τα δικαιώματα χρήσης του περιεχομένου ως mode bits. Τα δικαιώματα χρήσης θα επιτρέπουν τον διαχωρισμό στον τρόπο χρήσης του εγγράφου σύμφωνα με την κατηγορία στην οποία ανήκει ο χρήστης και παρέχουν μία ιδιαίτερη ευέλικτη μορφή διαχείρισης αυτής της δυνατότητας εμπνευσμένη από τα Unix-like υπολογιστικά περιβάλλοντα. Για παράδειγμα, θα μπορούσαμε να επιτρέπουμε σε μία ειδική ομάδα χρηστών για το smart-contract να είναι σε θέση να προσπελαύνει το περιεχόμενο του εγγράφου αλλά θα επιθυμούσαμε τον αποκλεισμό όλων των υπόλοιπων χρηστών του blockchain. Επιπλέον, επιθυμούμε η ανάρτηση νέων εγγράφων να επιτρέπεται αποκλειστικά από τον διαχειριστή του contract. Ο λόγος έχει σχέση κυρίως με τη φύση της πλατφόρμας που χρησιμοποιούμε. Δεδομένου ότι το περιεχόμενο είναι persistent στην μόνιμη μνήμη του blockchain κι είναι κοστοβόρα, πρέπει να είμαστε ιδιαίτερα προσεκτικοί στο πως αυτή χρησιμοποιείται. Συνεπώς, προκειμένου να αποφύγουμε πιθανά exploits του συστήματος, π.χ. κάποιος κακόβουλος χρήστης σκόπιμα να γεμίζει την μνήμη του smart-contract με αδιάφορα έγγραφα προκειμένου να εξαντλήσει το gas που έχει διατεθεί για την εφαρμογή.

Σχετικά με την επισήμανση των δικαιωμάτων χρήσης του περιεχομένου και τον τρόπο με τον οποίον αυτά συσχετίζονται με τις διάφορες κατηγορίες χρηστών θα υιοθετήσουμε τη μεθοδολογία που ακολουθείται σε Unix-like υπολογιστικά περιβάλλοντα. Δηλαδή, το όρισμα της μεθόδου το οποίο αναφέρεται στα δικαιώματα χρήσης θα πρέπει να συμμορφώνεται με τις εξής συμβάσεις:

- Να είναι τριψήφιος αριθμός.
- Κάθε ψηφίο του να παίρνει τιμές από το σύνολο $\{0,2,4,6\}$.
- Το πρώτο ψηφίο να υποδηλώνει τα δικαιώματα χρήσης του διαχειριστή του smart-contract.
- Το δεύτερο ψηφίο να αντιστοιχεί στα δικαιώματα της ειδικής ομάδας χρηστών του smart-contract.
- Το τρίτο ψηφίο στα δικαιώματα χρήσης οποιουδήποτε άλλου χρήστη έχει πρόσβαση στο Ethereum.

- Με 6 αντιστοιχούμε δικαιώματα ανάγνωσης **και** τροποποίησης.
- Με 4 αντιστοιχούμε **μόνον** δικαιώματα ανάγνωσης.
- Με 2 αντιστοιχούμε **μόνον** δικαιώματα τροποποίησης.
- Με 0 υποδηλώνουμε την απαγόρευση χρήσης του συγκεκριμένου περιεχομένου από την αντίστοιχη ομάδα χρηστών με οποιονδήποτε τρόπο.

2.5 Αφαίρεση υπαρχοντος εγγράφου

Η διαδικασία αυτή είναι η ακριβώς αντίστροφη της προηγούμενης. Η προδιαγραφή της μεθόδου αυτής αφορά: (i) πρώτον, ως προς τη λειτουργικότητα την απελευθέρωση όλων των πόρων που έχουν διατεθεί για την αποθήκευση κι ευρητηρίαση του σχετικού εγγράφου, και (ii) δεύτερον, το ποιοί χρήστες είναι σε θέση να φέρουν εις πέρας με επιτυχία αυτήν τη λειτουργία ως περιγράφεται. Πράγματι, αμφότερα σημεία είναι εξίσου σημαντικά και χρήζουν προσοχής σε μία πλατφόρμα όπως το blockchain. Για παράδειγμα, για την προηγούμενη μέθοδο που περιγράψαμε, απαιτούμε μόνον ο διαχειριστής του smart-contract να είναι σε θέση να αναρτά καινούριο περιεχόμενο με επιτυχία για λόγους που εξηγήσαμε προηγουμένως. Οι λόγοι αυτοί όμως δε συντρέχουν στην περίπτωση αυτή όμως. Συνεπώς, μπορούμε να επιτρέψουμε σε όσους χρήστες έχουν δικαιώματα τροποποίησης του εγγράφου να είναι σε θέση να το αφαιρέσουν από την μόνιμη μνήμη του contract. Υπάρχει όμως ένα σημαντικό μειονέκτημα με αυτήν την πολιτική που εφαρμόζουμε. Όσο μεγαλύτερο είναι το κοινό το οποίο έχει δικαιώματα τροποίησης, τόσο πιθανότερο είναι το ενδεχόμενο να αφαιρεθεί κατά λάθος, ή η αφαίρεσή του να βρίσκει άκρως αντίθετους κάποιους άλλους χρήστες. Τα ενδεχόμενα αυτά όμως μπορούν να αποτραπούν με την προσεκτική ανάθεση δικαιωμάτων τροποποίησης σε συνδυασμό με τον υπεύθυνο σχηματισμό της ειδικής ομάδας χρηστών του smart-contract, κι εδώ έγκειται η σημαντικότητα κι η εκμετάλλευση της ευελιξίας της μεθοδολογίας αυτής. Έτσι, για έγγραφα ευαίσθητου και σημαντικού περιεχομένου μπορούμε να δίνουμε δικαιώματα τροποίησης μόνον στους χρήστες της ειδικής ομάδας, ενώ οι υπόλοιποι χρήστες έχουν μόνον δικαιώματα προσπέλασης ή και καθόλου. Εάν τώρα όμως ένας χρήστης της ειδικής ομάδας είναι είτε επιρρεπής σε λάθη, είτε τήνει να δημιουργεί προβληματικές καταστάσεις, τότε είναι προτιμότερο να αφαιρεθεί από την ειδική ομάδα χρηστών ώστε αυτή να διατηρηθεί συμπαγής κι αποτελούμενη από έντιμους χρήστες εμπιστοσύνης.

2.6 Μετονομασία υπάρχοντος εγγράφου

Η συγκεκριμένη μέθοδος είναι πολύ παρόμοια με την προηγούμενη αλλά απαλλαγμένη από την κρισημότητα που την διέπει, αφού εδώ με τη λειτουργία της μετονομασίας, έστω και με μία λανθασμένη ενέργεια δε μπορούμε να χάσουμε περιεχόμενο, κι ακόμα, μπορούμε να αναιρέσουμε ένα ενδεχόμενο λάθος. Παρ' όλα αυτά, προτιμούμε να περιορίζουμε την χρήση της στους χρήστες που έχουν δικαιώματα τροποίησης πάνω στο έγγραφο ώστε να αποφύγουμε ακόμα την πιθανή όχληση των άλλων χρηστών από το λάθος ενός μη εξουσιοδοτημένου χρήστη, αλλά κι από πλευράς συνέπειας με άλλα συστήματα αρχείων που χρησιμοποιούν mode bits για τα δικαιώματα χρηστών.

2.7 Μεταβολή δικαιωμάτων χρήσης

Το κάθε έγγραφο το οποίο αποθηκεύεται στην μόνιμη μνήμη (persistent storage) του smart-contract συνοδεύεται από ένα τριψήφιο αριθμό αποτελούμενο από τα mode bits τα οποία αντιπροσωπεύουν τα δικαιώματα χρήσης (προσπέλασης και τροποίησης) του εγγράφου κι εκχωρείται κατά την εισαγωγή του εγγράφου. Συνεπώς, είναι απαραίτητη μία ακόμα μέθοδος για τη μεταβολή των δικαιωμάτων χρήσης του εγγράφου σύμφωνα με τις εκάστοτε ανάγκες οι οποίες προφανώς μπορεί να αλλάζουν με το πέρασμα του χρόνου. Ως ορίσματα χρειάζονται

μόνον το αναγνωριστικό του εγγράφου το οποίο αφορά η αλλαγή και το νέο τριψήφιο που αντιπροσωπεύει τα νέα δικαιώματα χρήσης ενω ελέγχεται το εάν ο καλών έχει δικαιώματα τουλάχιστον τροποίησης του εγγράφου. Διαφορετικά, η μέθοδος πρέπει να τερματίζει πρόωρα με ένα κατατοπιστικό μηνυμα σφάλματος που προϊδεάζει τον χρήστη για το τι έκανε λάθος.

2.8 Ανάκτηση αναγνωριστικών προσπελάσιμων ή τροποιήσιμων εγγράφων

Επιπλέον, χρειαζόμαστε ακόμα μεθόδους οι οποίες θα μπορούν να πληροφορούν τον χρήστη για τα έγγραφα τα οποία είναι διαθέσιμα για αυτόν καθώς και με τι δικαιώματα, ούτως ώστε σε δεύτερο χρόνο να επιλέγει κάποια και να επενεργεί σε αυτά. Αυτές δεν θα δέχονται ορίσματα, αλλά θα ταυτοποιούν την κατηγορία χρήστη στην οποία ανήκει ο καλών, και στη συνέχεια θα αντιστοιχούν τα έγγραφα στα οποία μπορεί να έχει πρόσβαση μετά από εξέταση αυτών κι εισαγωγή τους ή μη στη λίστα που επιστρέφεται στον καλώντα. Πρόκειται για ασφαλείς μεθόδους οι οποίες δεν έχουν κάποιο αντίκτυπο στην κατάσταση και την μνήμη του smart-contract. Χρήστες στους οποίους δεν αντιστοιχούν καθόλου προνόμια χρήσης λαμβάνουν μία άδεια λίστα που αντιπροσωπεύει αυτό ακριβώς το γεγονός.

2.9 Προσθήκη περιεχομένου σε έγγραφο

Η συγκεκριμένη μέθοδος επιθυμούμε να επιτρέπει στους χρήστες που έχουν δικαιώματα τροποίησης του εγγράφου, του οποίου το αναγνωριστικό δίνεται ως όρισμα στη μέθοδο, να προσθέτουν περιεχόμενο μέσα σε αυτό. Οπότε, εκτός από το αναγνωριστικό, χρειάζεται ακόμα η θέση που θα προστεθεί το νέο περιεχόμενο, δηλαδή μετά τον χαρακτήρα στη θέση v, καθώς και η νέα φράση, πρόταση ή παράγραφος, σε μορφή αλφαριθμητικού. Άρα, το v παίρνει τιμές που δεν πρέπει να υπερβαίνουν το μήκος του περιεχομένου ως έχει τη στιγμή της εκτέλεσης της μεθόδου, διαφορετικά θα πρέπει να τερματίζει πρόωρα και να ενημερώνει κατάλληλα τον χρήστη. Ομοίως, περιπτώσεις απόπειρας αλλοίωσης του εγγράφου από χρήστες που δεν έχουν τέτοια δικαιώματα θα πρέπει να μπορούν να εντοπιστούν και να γίνονται track ώστε να διευθετούνται κατάλληλα από το διαχειριστή.

2.10 Διαγραφή περιεχομένου από έγγραφο

Η μέθοδος αυτή αφορά επίσης τη μεταβολή περιεχομένου αλλά είναι πιο ευαίσθητη από την προηγούμενη αφού το αποτέλεσμά της δεν αναιρείται. Δηλαδή, αφού διαγραφεί κάποιο τμήμα του εγγράφου, έστω από λάθος, δεν υπάρχει η δυνατότητα να επαναφερθεί αυτό κι είναι χαμένο για πάντα. Αποτροπή τέτοιων σεναρίων γίνονται μόνο με τον προσεκτικό ορισμό δικαιωμάτων χρήσης αλλά και τη δημιουργία ειδικής ομάδας που απαρτίζεται αποκλειστικά από υπεύθυνους χρήστες.

Ως ορίσματα η μέθοδος αυτή δέχεται το αναγνωριστικό του εγγράφου που αφορά η διαγραφή, το σημείο που ξεκινάει η αφαίρεση του περιεχομένου, καθώς και το μήκος της διαγραφής. Προφανώς, το μήκος του περιεχομένου πρέπει να είναι τουλάχιστον ίσο ή μεγαλύτερο από τη θέση που ξεκινάει η διαγραφή συν το μήκος της διαγραφής. Διαφορετικά, δεν επιτρέπεται η ενέργεια αυτή κι επιστρέφεται ένα κατατοπιστικό μήνυμα σφάλματος.

2.11 Προβολή περιεχομένου

Η μέθοδος αυτή αιτείται του περιεχομένου ενός εγγράφου βάσει του αναγνωριστικού του, το οποίο δίνεται κι ως μοναδικό όρισμα. Θα πρέπει να επαληθευτεί εκ των προτέρων όμως το εάν ο καλών έχει δικαιώματα προσπέλασης του εγγράφου, ώστε αυτό να ανακτηθεί από την μόνιμη μνήμη (persistent storage). Διαφορετικά θα πρέπει να επιστραφεί ένα σχετικό μήνυμα ώστε να ενημερώνεται ο χρήστης κατάλληλα για την αστοχία της κλήσης του. Πρόκειται για

βασική μέθοδο του smart-contract η οποία απλά ανακτά περιεχόμενο χωρίς να το τροποποιεί, κι άρα εκτός από την επαλήθευση των δικαιωμάτων προσπέλασης του καλώντος δεν χρήζει κάποιας άλλης ιδιαίτερης αντιμετώπισης.

Κεφάλαιο 3

Τεχνικά θέματα υλοποίησης

This chain shall set you free, my friend.
But just after it's been verified!

Στην ενότητα αυτή περιγράφουμε την υλοποίηση του smart-contract για τη διαχείριση εγγράφων.

3.1 Solidity DockChain smart-contract

Στο Σχήμα 3.1 δείχνουμε τη δήλωση του smart-contract, τις μεταβλητές κατάστασης του contract, καθώς και τη μέθοδο που χρησιμοποιείται για την αρχικοποίησή του. Πιο συγκεκριμένα, Στη γραμμή 1 δείχνουμε τη δήλωση pragma προκειμένου να γνωρίζει το προγραμματιστικό περιβάλλον την έκδοση του compiler που θα πρέπει να χρησιμοποιήσει. Στη γραμμή 3 βρίσκεται η δήλωση κι ακολουθεί το περιεχόμενο του smart-contract, αρχικά οι μεταβλητές κατάστασης στις γραμμές 5–14, κι έπειτα οι συναρτήσεις του contract, που θα εξηγήσουμε μία προς μία για το υπόλοιπο της ενότητας αυτής.

Η πρώτη συνάρτηση που παρουσιάζουμε στο Σχήμα 3.2 αποτελεί τη μέθοδο τερματισμού του contract όπου γίνεται κλήση της destructor μεθόδου για την καταστροφή του και τη διοχέτευση των κρυπτονομισμάτων που έχουν αφιερωθεί αρχικά για το gas του contract αλλά και στη συνέχεια μεταφερθεί στο balance του, προς το λογαριασμό του χρήστη που έχει επιφορτιστεί με το ρόλο του διαχειριστή.

Στο Σχήμα 3.3 παρουσιάζουμε τον modifier που εκτελείται πριν την μεθοδο με την οποία έχει συσχετιστεί μέσω της κατάλληλης δήλωσης στην κεφαλίδα της συνάρτησης, καθώς και στο σώμα του modifier[1]. Σκοπός του συγκεκριμένου modifier είναι ο περιορισμός χρήσης των σχετιζόμενων μεθόδων μόνο από τον χρήστη που έχει ορισθεί ως διαχειριστής του smart-contract. Πρόκειται για μία "βοηθητική" μέθοδο η εκτέλεση της οποίας προηγείται κι εκτελεί ελέγχους με προτάσεις αληθείας μέσα σε κατάλληλες δηλώσεις, όπως στη γραμμή 3 του Σχήματος 3.3. Αρχικά ο διαχειριστής είναι ο χρήστης του blockchain που έχει κάνει deploy το smart-contract, ενώ δίνεται η δυνατότητα να αλλαχθεί και να οριστεί νέος διαχειριστής με τη μέθοδο που παρουσιάζουμε στο Σχήμα 3.4, η οποία φυσικά δύναται να εκτελεσθεί μόνον από τον τρέχοντα διαχειριστή που έχει ορισθεί μέχρι κι εκείνη τη στιγμή πριν λάβει χώρα η αλλαγή αυτή, όπου δε γίνεται τίποτε άλλο εκτός από την ανάθεση της μεταβλητής που διατηρεί τη διεύθυνση του διαχειριστή με την τιμή που εισάγει ο προηγούμενος διαχειριστής ως όρισμα (γραμμή 2). Το συγκεκριμένο modifier, ονόματι **onlyByOwner**, τον έχουμε συσχετίσει με τις ακόλουθες μεθόδους για τις οποίες αποκαλύπτουμε τα κίνητρά μας εν συντομία κατώθι:

kill για την καταστροφή του smart-contract και τη διοχέτευση των wei/ethers που έχουν δεσμευτεί για gas και balance στο λογαριασμό του διαχειριστή (Σχήμα 3.2),

[1]Στη γραμμή 5, Σχήμα 3.3 δηλώνεται ότι οποιαδήποτε μέθοδος μπορεί να κάνει χρήση του συγκεκριμένου modifier.

```
pragma solidity ^0.4.25;

contract DockChain {

    address owner;

    mapping(address => uint) users;

    mapping(string => uint16) flags;
    mapping(string => string) contents;

    string[] docs;
    address[] addresses;
    uint docIdCharsCounter;

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

    ...
    ...
}
```

Σχήμα 3.1: Solidity smart-contract for Ethereum.

chown για την αλλαγή διαχειριστή καθώς πρόκειται για μία σημαντική ιδιότητα χρήστη, κι η ανάθεσή της γίνεται με υπεύθυνο τρόπο καθώς οι συνέπειές της είναι μη αναστρέψιμες, π.χ. αν εσείς ως διαχειριστής περάσετε αυτήν την ιδιότητα σε έναν άλλον χρήστη, δεν είστε σε θέση να ξαναχρίσετε τον εαυτό σας ως διαχειριστή (Σχήμα 3.4),

addUser για την προσθήκη διευθύνσεων στην ειδική ομάδα χρηστών με διαφορετικά δικαιώματα από τους υπόλοιπους χρήστες που έχουν πρόσβαση στο blockchain (Σχήμα 3.5),

delUser για την αφαίρεση διευθύνσεων χρηστών από την ειδική αυτή ομάδα που προαναφέραμε (Σχήμα 3.6),

addDoc για την ανάρτηση εγγράφων με συγκεκριμένα δικαιώματα χρήσης, ήτοι προσπέλασης και τροποποίησης (Σχήμα 3.9).

```
function kill () public onlyByOwner() {
    selfdestruct (owner);
}
```

Σχήμα 3.2: Μέθοδος αυτοκαταστροφής του solidity smart-contract.

Εν συνεχεία, στο Σχήμα 3.5 δείχνουμε τη μέθοδο προσθήκης χρηστών στην ειδική ομάδα που έχει θεσπιστεί με τη δημιουργία ενός στιγμιοτύπου του προτεινόμενου smart-contract. Πρόκειται για μία εξαιρετικά απλή υλοποίηση στην οποία απλά ελέγχουμε αρχικά εάν η συγκεκριμένη διεύθυνση είναι ήδη γνωστή, κι αν δεν είναι, τότε τη συσχετίζουμε με την χρονική στιγμή όπου εκτελέστηκε η μέθοδος σε κάποιον κόμβο του Ethereum στον οποίον έχουν ανατεθεί καθήκοντα mining. Το Proof-of-Work (PoW) πρωτόκολλο είναι αναπόσπαστο κομμάτι του blockchain και είναι η μεθοδολογία η οποία έχει υιοθετηθεί προκειμένου να διασφαλιστεί με ανώνυμο τρόπο η εγκυρότητα της συναλλαγής. Ο πρώτος miner που

```
1:    modifier onlyByOwner() {
2:        require(
3:            msg.sender == owner,
4:            "Sender not owner to be authorized.");
5:        _;
6:    }
```

Σχήμα 3.3: Φίλτρο χρηστών της μεθόδου αυτοκαταστροφής του smart-contract.

```
1:    function chown (address newOwner) public onlyByOwner() {
2:        owner = newOwner;
3:    }
```

Σχήμα 3.4: Μέθοδος αλλαγής διαχειριστή του solidity smart-contract.

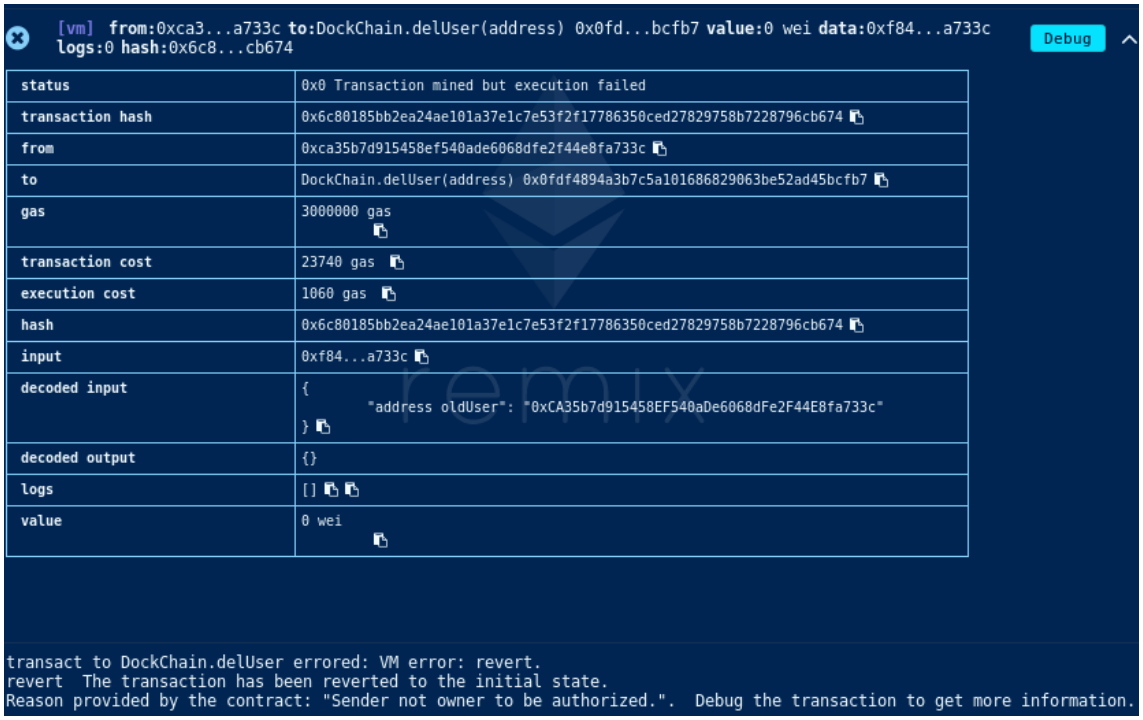
```
1:    function addUser (address newUser) public onlyByOwner() {
2:        if (users[newUser] <= 0) {
3:            users[newUser] = block.timestamp;
4:        }
5:    }
```

Σχήμα 3.5: Μέθοδος προσθήκης διεύθυνσης στην ομάδα χρηστών.

```
 1:    function delUser (address user)
              public onlyByOwner() existingUser(user) {
 2:           if (users[user] > 0) {
 3:             delete users[user];
 4:             for (uint i=0; i<addresses.length; ++i) {
 5:               if (addresses[i] == oldUser) {
 6:                 delete addresses[i];
 7:                 if (i != addresses.length-1) {
 8:                   addresses[i] = addresses[addresses.length-1];
 9:                   delete addresses[addresses.length-1];
10:                 }
11:                 break;
12:               }
13:             }
14:           }
15:    }
```

Σχήμα 3.6: Μέθοδος διαγραφής διεύθυνσης χρήστη.

εκτελεί τη μέθοδο και κατορθώνει να εξάγει το hash που πιστοποιεί τη συναλλαγή, την προσθέτει στο blockchain κι επιβραβεύεται για αυτήν την κίνηση με νεοεκδοθείσες μονάδες κρυπτονομίσματος που αντιστοιχούν στο Ethereum, ενώ όσοι miners έπονται πιστοποιούν τη συναλλαγή για μικρότερο τίμημα ως αποζημίωση για τη χρήση των πόρων που έχουν διαθέσει για τους απαραίτητους υπολογισμούς, τόσο υπολογιστικούς όσο κι ενεργειακούς, καθώς απαιτείται η κατανάλωση πολλών ενεργειακών μονάδων για την διετέλεση μίας τέτοιας εξαιρετικά ενεργοβόρας διαδικασίας.



Σχήμα 3.7: Αποτέλεσμα προσομοίωσης εκτέλεσης στο solidity προγραμματιστικό περιβάλλον του *http://remix.ethereum.org*.

Η ακριβώς ανάστροφη διαδικασία παρουσιάζεται στο Σχήμα 3.6 όπου γίνεται αφαίρεση του χρήστη που επιλέγει ο διαχειριστής, εφόσον φυσικά αυτός υπάρχει. Ο περιορισμός χρήσης από τον διαχειριστή επιβάλλεται με τη δήλωση του modifier **onlyByOwner** στον οποίον έχουμε ήδη αναφερθεί. Όμως παρατηρούμε την ύπαρξη ενός ακόμα περιορισμού που έχει θεσπιστεί με την χρήση του modifier **existingUser**, ο οποίος παρουσιάζεται στο Σχήμα 3.8 κι απαιτεί το όρισμα που δέχεται η μέθοδος να αντιστοιχεί σε διεύθυνση χρήστη η οποία έχει ήδη προστεθεί στην ομάδα χρηστών του smart-contract. Η τήρηση των περιορισμών αυτών μέσω της τεχνικής της χρήσης modifiers είναι απαρέκλητη κι εμποδίζουν την φυσιολογική εκτέλεση της μεθόδου όταν δεν ικανοποιούνται τα κριτήρια που ορίζονται. Για παράδειγμα, στο Σχήμα 3.7 δείχνουμε το αποτέλεσμα που προέρχεται από την κλήση της μεθόδου διαγραφής χρηστών από την ομάδα του smart-contract που προκάλεσε χρήστης του οποίου η διεύθυνση δεν ταυτιζόταν με αυτή του διαχειριστή, κι άρα η εκτέλεση τερματίστηκε πρόωρα στον ανάλογο modifier χωρίς όμως ποτέ να προχωρήσει στο κυρίως σώμα της μεθόδου. Ενδεικτικό είναι το μήνυμα που επιστρέφει ο remix.ethereum simulator: "**Transact to DockChain.delUser errored. VM error: Revert. The transaction has been reverted to the initial state. Reason provided by the contract: Sender not owner to be authorized.**" Στην πράξη αυτό σημαίνει ότι η μη αποδοχή της εκτέλεσης της συνάρτησης από τον modifier επαναφέρει την κατάσταση του smart-contract στην αρχική, πριν από την κλήση της μεθόδου, κι όσες μεταβολές είχαν λάβει χώρα έως εκείνη τη στιγμή αναιρούνται. Ακόμα, στις γραμμές 4–13 αφαιρούμε τη διεύθυνση του διαγραφέντος χρήστη από τη βοηθητική δομή `addresses`.

```
1:    modifier existingUser (address user) {
2:        require (
3:            owner == user || users[user] > 0,
4:            "No such user exists.");
5:        _;
6:    }
```

Σχήμα 3.8: Φίλτρο ορισμάτων εισόδου της μεθόδου διαγραφής διευθύνσεων χρηστών.

```
1:    function addDoc (string docId, string content, uint16 flag)
              public onlyByOwner() {
2:        assert (flag > 0);
3:        docs.push (docId);
4:        contents[docId] = content;
5:        flags[docId] = flag;
6:        docIdCharsCounter += bytes(docId).length;
7:    }
```

Σχήμα 3.9: Μέθοδος εισαγωγής περιεχομένου βάσει αλφαριθμητικού αναγνωριστικού.

```
1:    function chmod (string docId, uint16 flag)
              public existingId(docId) writingEligibility(docId) {
2:        assert (flag > 0);
3:        if (flags[docId] > 0) {
4:            flags[docId] = flag;
5:        }
6:    }
```

Σχήμα 3.10: Μέθοδος τροποποίησης δικαιωμάτων χρήσης περιεχομένου.

```
1:    function getContent (string docId) public
              constant readingEligibility(docId) returns (string) {
2:        return contents[docId];
3:    }
```

Σχήμα 3.11: Μέθοδος ανάκτησης περιεχομένου.

```
 1:    modifier readingEligibility (string docId) {
 2:        uint16 flag = flags[docId];
 3:        require (
 4:            flag > 0,
 5:            "No such document exists.");
 6:
 7:        uint16 ownerRight = (flag%1000)/100;
 8:        uint16 groupRight = (flag%100)/10;
 9:        uint16 restRight = flag%10;
10:
11:        require (
12:        (msg.sender==owner && (ownerRight==4 || ownerRight==6))
13:        || (users[msg.sender]>0 && (groupRight==4 || groupRight==6))
14:        || restRight==4 || restRight==6,
15:            "Ineligible user to read document");
16:        _;
17:    }
```

Σχήμα 3.12: Φίλτρο χρηστών βάσει δικαιωμάτων πρόσβασης περιεχομένου.

Ομοίως, η μέθοδος προσθήκης περιεχομένου που παρουσιάζεται στο Σχήμα 3.9 ελέγχει αρχικά για το εάν εκτελείται από το διαχειριστή του smart-contract μέσω του προαναφερθέντος modifier, και στη συνέχεια εντός του σώματος της μεθόδου αποθηκεύει εντός συγκεκριμένων persistent[2] δομών τα εξής:

1. Το αναγνωριστικό που θα χρησιμοποιείται για την ταυτοποίηση του περιεχομένου στη λίστα **docs**.
2. Το περιεχόμενο καθεαυτό στο mapping με το όνομα **contents**, από όπου θα γίνεται κι ανάκτησή του βάσει του αναγνωριστικού.
3. Τα δικαιώματα χρήσης του περιεχομένου αυτού σε mode bits format στο mapping με το όνομα **flags**.

Οπότε, με τη μέθοδο **chmod** του Σχήματος 3.10 καθιστούμε δυνατή σε δεύτερο χρόνο από την εισαγωγή του εγγράφου, την τροποποίηση των δικαιωμάτων αυτών από τους χρήστες στους οποίους έχει επισημανθεί το αντίστοιχο δικαίωμα, ήτοι με το ψηφίο $0, 2, 4$ ή 6 στην κατηγορία χρηστών που τους αντιστοιχεί.

Με τη μέθοδο **getContent** του Σχήματος 3.11 οι χρήστες είναι σε θέση να ανακτούν το αιτούμενο περιεχόμενο βάσει του αναγνωριστικού που έχει οριστεί κατά την εισαγωγή. Επιπλέον, παρατηρούμε το keyword constant στη κεφαλίδα της δήλωσης της μεθόδου που υποδηλώνει ότι η συγκεκριμένη μέθοδος δεν τροποποιεί κάποια μεταβλητή του smart-contract και προσφέρει κάποια οφέλη στην απόδοση εφόσον χειρίζεται ανάλογα. Πρόκειται για τεχνική δανεισμένη από τη γλώσσα προγραμματισμού C++. Για το blockchain όμως το γεγονός ότι η εκτέλεση μίας μεθόδου δεν αλλάζει την κατάσταση του smart-contract σημαίνει συγκεκριμένα οφέλη για τον τρόπο που αντιμετωπίζεται ως συναλλαγή, το πόσο σημαντική θα είναι η εκτέλεση του PoW πρωτοκόλλου αλλά κι η προσθήκη ενός ακόμα block στην ακολουθία των συναλλαγών. Ακόμα, η δήλωση του modifier **readingEligibility** περιορίζει την εκτέλεση της μεθόδου **getContent** στους χρήστες οι οποίοι έχουν τουλάχιστον δικαιώματα προσπέλασης του περιεχομένου που αιτείται ο χρήστης. Όπως δείχνουμε άλλωστε και στο Σχήμα 3.12, ο συγκεκριμένος modifier ελέγχει σε ποιά κατηγορία ανήκει ο χρήστης κι εν συνεχεία εάν τα δικαιώματα που έχει για την κατηγορία αυτή (γραμμές 7–9) του επιτρέπουν να προσπελάσει το περιεχόμενο. Διαφορετικά, εξετάζουμε εάν τα δικαιώματα της επόμενης πιο γενικής κατηγορίας χρηστών του εξασφαλίζουν την ανάγνωση του περιεχομένου. Δηλαδή

[2] το περιεχόμενο των οποίων συνοδεύει κάθε DockChain block του συγκεκριμένου smart-contract.

```
 1:    function rmvDoc (string docId) public writingEligibility(docId) {
 2:        for (uint i=0; i<docs.length; ++i) {
 3:            if (keccak256(bytes(docs[i]))==keccak256(bytes(docId))) {
 4:                delete docs[i];
 5:                if (i != docs.length-1) {
 6:                    docs[i] = docs[docs.length-1];
 7:                    delete docs[docs.length-1];
 8:                }
 9:                break;
10:            }
11:        }
12:        delete contents[docId];
13:        delete flags[docId];
14:
15:        uint charsCounter = bytes(docId).length;
16:        assert (docIdCharsCounter >= charsCounter);
17:        docIdCharsCounter -= charsCounter;
18:    }
```

Σχήμα 3.13: Μέθοδος αφαίρεσης περιεχομένου βάσει αναγνωριστικού.

αν δεν είναι ο διαχειριστής, εξετάζεται εάν ανήκει στην ειδική ομάδα χρηστών του smart-contract, κι αν δεν ανήκει ούτε εκεί, τότε εξετάζονται τα δικαιώματα προσπέλασης για τους υπόλοιπους χρήστες. Αλλά ακόμα κι αν ήταν ο διαχειριστής αλλά δεν είχε δικαιώματα ανάγνωσης ως διαχειριστής, δοκιμάζουμε εάν θα είχε τέτοια δικαιώματα ως μέλος της ειδικής ομάδας του smart-contract, ή ακόμα κι εν τη απουσία τέτοιων δικαιωμάτων αν θα μπορούσε να τα εξασφαλίσει αυτά ως απλός χρήστης του blockchain (γραμμές 12–14).

```
 1:    modifier writingEligibility (string docId) {
 2:        uint16 flag = flags[docId];
 3:        require (
 4:            flag > 0,
 5:            "No such document exists.");
 6:
 7:        uint16 ownerRight = (flag%1000)/100;
 8:        uint16 groupRight = (flag%100)/10;
 9:        uint16 restRight = flag%10;
10:
11:        require (
12:        (msg.sender==owner && (ownerRight==2 || ownerRight==6))
13:        || (users[msg.sender]>0 && (groupRight==2 || groupRight==6))
14:        || restRight==2 || restRight==6,
15:            "Ineligible user to read document");
16:        _;
17:    }
```

Σχήμα 3.14: Φίλτρο χρηστών βάσει δικαιωμάτων τροποποίησης περιεχομένου.

Στο Σχήμα 3.14 δείχνουμε τον αντίστοιχο **writingEligibility** modifier που εξασφαλίζει την εκτέλεση των μεθόδων βάσει δικαιωμάτων τροποποίησης τουλάχιστον. Μεταξύ άλλων γίνεται χρήση του γίνεται στη μέθοδο **rmvDoc** του Σχήματος 3.13 για την αφαίρεση περιεχομένου από το smart-contract. Αποτελείται από τέσσερα απλά βήματα προκειμένου να αφαιρεθεί τόσο το περιεχόμενο καθεαυτό, αλλά και το αναγνωριστικό από όλες τις δομές του smart-

contract στις οποίες έχει προστεθεί, ήτοι (1) η λίστα **docs** που διατηρεί όλα τα αναγνωριστικά, (2) το mapping **contents**, (3) το mapping **flags** που διατηρεί τα δικαιώματα χρήσης που συσχετίζονται με το συγκεκριμένο περιεχόμενο, και τέλος, (4) προσαρμόζονται κατάλληλα κάποιες βοηθητικές μεταβλητές των οποίων γίνεται χρήση στις μεθόδους **getReadableDocs** και **getWritableDocs** που εξετάζουμε εν συνεχεία. Πολύ παρόμοια με τη μέθοδο **rmvDoc** είναι η μέθοδος **rnmDoc**, με τη διαφορά όμως ότι αντί να αφαιρείται εντελώς το περιεχόμενο, συσχετίζεται με το νέο αναγνωριστικό που δίνεται ως όρισμα στη μέθοδο, κι επιπλέον το παλιό αναγνωριστικό αντικαθίσταται με το καινούριο στις αντίστοιχες δομές.

```
 1:    function rnmDoc (string docId1, string docId2)
                public writingEligibility(docId1) {
 2:        for (uint i=0; i<docs.length; ++i) {
 3:            if (keccak256(bytes(docs[i]))==keccak256(bytes(docId1))) {
 4:                delete docs[i];
 5:                docs[i] = docId2;
 6:                break;
 7:            }
 8:        }
 9:        contents[docId2] = contents[docId1];
10:        flags[docId2] = flags[docId1];
11:        delete contents[docId1];
12:        delete flags[docId1];
13:    }
```

Σχήμα 3.15: Μέθοδος μετονομασίας αναγνωριστικού.

Εν συνεχεία θα μελετήσουμε τις μεθόδους **getReadableDocs** και **getWritableDocs** στα Σχήματα 3.16 και 3.17 αντίστοιχα. Πρόκειται για μεθόδους οι οποίες δε διαθέτουν περιορισμούς ως προς την χρήση τους, κι επιπλέον, δεν μεταβάλλουν την κατάσταση του smart-contract, όπως άλλωστε υποδηλώνει το keyword constant στις κεφαλίδες των δηλώσεων των μεθόδων. Σκοπός τους δεν είναι η ανάκτηση περιεχομένου, αλλά μίας λίστας αναγνωριστικών στο περιεχόμενο των οποίων ο χρήστης έχει δικαιώματα είτε προσπέλασης, είτε τροποποίησης αντιστοιχα, ούτως ώστε να είναι σε θέση εν συνεχεία να επεξεργαστεί το περιεχόμενο που αντιστοιχεί με κατάλληλο τρόπο εφόσον επιθυμεί. Η επιλογή των αναγνωριστικών βάσει των δικαιωμάτων του χρήστη που καλεί τη συνάρτηση γίνεται σειριακά. Δηλαδή, ένα προς ένα τα διαθέσιμα αναγνωριστικά εξετάζονται και συγκρίνουμε τα δικαιώματα του χρήστη για το περιεχόμενο που αντιστοιχεί σε αυτά. Εάν αυτά συμφωνούν, π.χ. ο χρήστης ανήκει σε κατηγορία που έχει επισημανθεί με ένα 6, τότε το αναγνωριστικό αυτό προστίθεται στη λίστα και συνεχίζουμε με την εξέταση του επόμενου αναγνωριστικού.

Οι δύο επόμενες μέθοδοι που θα περιγράψουμε αποτελούν και τις τελευταίες του smart-contract. Πρόκειται για τις μεθόδους **editDelete** (Σχήμα 3.18) και **editInsert** (Σχήμα 3.19) που χρησιμοποιούνται προκειμένου ο χρήστης να επιφέρει αλλαγές στο περιεχόμενο καθεαυτό! Έτσι έχουμε για την μέθοδο **editDelete** τον χρήστη να εισάγει το αναγνωριστικό του περιεχομένου που επιθυμεί να τροποποιήσει, το σημείο από το οποίο ξεκινάει η διαγραφή μέρους του περιεχομένου, καθώς και το μήκος της διαγραφής. Προφανώς, για να δύναται ο χρήστης να προβεί σε μία τέτοια μεταβολή του περιεχομένου υποχρεούται να έχει τέτοια δικαιώματα, όπως υποδηλώνει άλλωστε κι ο **writingEligibility** modifier στην κεφαλίδα της μεθόδου. Σχετικά με τις λεπτομέρειες της τροποποίησης του περιεχομένου και τη διαχείριση των αλφαριθμητικών, μπορούμε να συνοψίσουμε ότι είναι κάπως "δύστροπη" στο blockchain, τόσο στη προγραμματιστική γλώσσα Solidity, όσο και σε Serpent, όπως θα δούμε στην αμέσως επόμενη ενότητα. Πιο συγκεκριμένα, απαιτείται η φόρτωση του περιεχομένου με ρητές (explicit) εντολές, η μετατροπή του αλφαριθμητικού σε array από bytes πάνω στα οποία γίνεται η όποια μεταβολή, και εν συνεχεία, αφού ληφθούν υπόψην οι αλλαγές στο μήκος του αποτελέσματος, η εκ νέου μετατροπή της ακολουθίας από bytes ξανά σε αλφαριθμητικό προκειμένου να αποθηκευθεί κατάλληλα και να ενημερωθεί η κατάσταση στο νέο block που

```
 1:    function getReadableDocs () public constant
                 returns (string docList) {
 2:        docList = new string(docIdCharsCounter+docs.length);
 3:        bytes memory docListBytes = bytes(docList);
 4:        string memory delimiter = ",";
 5:        bytes memory delimiterBytes = bytes(delimiter);
 6:        uint counter = 0;
 7:        for (uint i=0; i<docs.length; ++i) {
 8:            uint16 flag = flags[docs[i]];
 9:            if (flag > 0) {
10:                uint16 ownerRight = (flag%1000)/100;
11:                uint16 groupRight = (flag%100)/10;
12:                uint16 restRight = flag%10;
13:
14:                if ((msg.sender==owner
                            && (ownerRight==4 || ownerRight==6))
15:                    || (users[msg.sender]>0
                            && (groupRight==4 || groupRight==6))
16:                    || restRight==4 || restRight==6) {
17:                        bytes memory docBytes = bytes(docs[i]);
18:                        for (uint j=0; j<docBytes.length; ++j) {
19:                            docListBytes[counter++] = docBytes[j];
20:                        }
21:                        docListBytes[counter++] = delimiterBytes[0];
22:                }
23:            }
24:        }
25:        if (counter == 0) {
26:            docList = "";
27:        }
28:        return docList;
29:    }
```

Σχήμα 3.16: Μέθοδος ανάκτησης αναγνωριστικών προσπελάσιμου περιεχομένου προς ανάγνωση.

```
 1:    function getWritableDocs () public constant
               returns (string docList) {
 2:        docList = new string(docIdCharsCounter+docs.length);
 3:        bytes memory docListBytes = bytes(docList);
 4:        string memory delimiter = ",";
 5:        bytes memory delimiterBytes = bytes(delimiter);
 6:        uint counter = 0;
 7:        for (uint i=0; i<docs.length; ++i) {
 8:            uint16 flag = flags[docs[i]];
 9:            if (flag > 0) {
10:                uint16 ownerRight = (flag%1000)/100;
11:                uint16 groupRight = (flag%100)/10;
12:                uint16 restRight = flag%10;
13:
14:                if ((msg.sender==owner
                           && (ownerRight==2 || ownerRight==6))
15:                    || (users[msg.sender]>0
                           && (groupRight==2 || groupRight==6))
16:                    || restRight==2 || restRight==6) {
17:                        bytes memory docBytes = bytes(docs[i]);
18:                        for (uint j=0; j<docBytes.length; ++j) {
19:                            docListBytes[counter++] = docBytes[j];
20:                        }
21:                        docListBytes[counter++] = delimiterBytes[0];
22:                }
23:            }
24:        }
25:        if (counter == 0) {
26:            docList = "";
27:        }
28:        return docList;
29:    }
```

Σχήμα 3.17: Μέθοδος ανάκτησης αναγνωριστικών προσπελάσιμου περιεχομένου προς τροποποίηση.

```
 1:    function editDelete (string docId, uint8 pos, uint8 length)
                public writingEligibility (docId) {
 2:        bytes memory contentBytes = bytes(contents[docId]);
 3:        uint contentLength = contentBytes.length;
 4:
 5:        string memory edited = new string(contentLength-length);
 6:        bytes memory editedBytes = bytes(edited);
 7:        for (uint i=0;i<pos; ++i) {
 8:            editedBytes[i] = contentBytes[i];
 9:        }
10:        for (i=pos+length; i<contentLength; ++i) {
11:            editedBytes [i-length] = contentBytes[i];
12:        }
13:        contents[docId] = string(editedBytes);
14:    }
```

Σχήμα 3.18: Μέθοδος τροποποίησης περιεχομένου.

```
 1:    function editInsert (string docId, uint8 pos, string newContent)
              public writingEligibility (docId) {
 2:        bytes memory contentBytes = bytes(contents[docId]);
 3:        bytes memory newBytes = bytes(newContent);
 4:
 5:        uint contentLength = contentBytes.length;
 6:        uint newBytesLength = newBytes.length;
 7:
 8:        string memory edited = new string(contentLength+newBytesLength);
 9:        bytes memory editedBytes = bytes(edited);
10:
11:        for (uint i=0; i<pos; ++i) {
12:            editedBytes[i] = contentBytes[i];
13:        }
14:        for (i=0; i<newBytesLength; ++i) {
15:            editedBytes[pos+i] = newBytes[i];
16:        }
17:        for (i=pos; i<contentLength; ++i) {
18:            editedBytes[newBytesLength+i] = contentBytes[i];
19:        }
20:        contents[docId] = string(editedBytes);
21:    }
```

Σχήμα 3.19: Μέθοδος εμπλουτισμού περιεχομένου βάσει αναγνωριστικού.

θα προσαρτηθεί στην αλυσίδα.

```
 1:    import bitcoin, pytest, serpent
 2:    from ethereum.utils import decode_hex
 3:
 4:    from ethereum import utils, abi
 5:    from ethereum.tools import tester
 6:
 7:    from ethereum.slogging import get_logger, configure_logging
 8:
 9:    logger = get_logger()
10:    configure_logging(':info')
11:
12:    dockchain_serpent = """
13:    data owner
14:    data users[]
15:    data flags[]
16:    data contents[](content[4000],length)
17:    data docids[]
18:    data group[]
19:    data repossize
20:    data groupsize
21:
22:    def init ():
23:        log("** User '"+msg.sender+"' creates a new smart-contract.")
24:        self.owner = msg.sender
25:        self.repossize = 0
26:        self.groupsize = 0

       ...
       ...

27:    """
28:
29:    def test_dockchain():
30:        c = tester.Chain()
31:        x = c.contract(dockchain_serpent,sender=tester.k1,
                                                  language='serpent')
```

Σχήμα 3.20: Serpent/Python smart-contract set-up for Ethereum.

3.2 Serpent DockChain smart-contract

Στην ενότητα αυτή εξετάζουμε το smart-contract που είναι γραμμένο σε Serpent[14] για testing και deployment μέσα από python περιβάλλον, απαριθμώντας κι εξηγώντας πρώτα τις απαραίτητες μεταβλητές, οι τιμές των οποίων απαρτίζουν την κατάσταση του instance το οποίο έχει γίνει deploy εντός του blockchain με ένα αρχικό block ως γεννήτορα. Στη γραμμή 13 του Σχήματος 3.20 έχουμε τη μεταβλητή **owner** που ανά πάσα στιγμή έχει αποθηκευμένη την τιμή του διαχειριστή του smart-contract. Η ιδιότητα του διαχειριστή είναι μία σημαντική ιδιότητα χρήστη για τις πιο βασικές λειτουργίες, όπως ο σχηματισμός της ειδικής ομάδας χρηστών του smart-contract, η ανάρτηση νέου περιεχομένου, κ.ο.κ. Οι χρήστες που απαρτίζουν την ειδική αυτή ομάδα αποθηκεύονται στη mapping δομή ονόματι **users** της γραμμής 14. Η προσπέλαση της ειδικής αυτής δομής γίνεται με χρήση της διεύθυνσης του χρήστη κι επιστέφει την χρονική στιγμή όπου εισήχθει ο χρήστης στην δομή από κάποιον κόμβο miner του blockchain όταν για πρώτη φορά εκτελέστηκε εις πέρας το proof-of-work πρωτόκολλο

πιστοποίησης συναλλαγής του blockchain για τη συγκεκριμένη κλήση. Επιπλέον κάνουμε χρήση μίας βοηθητικής λίστας διευθύνσεων χρηστών (γραμμή 18) για τη σειριακή προσπέλαση των διευθύνσεων των χρηστών. Η βοηθητική μεταβλητή της γραμμής 20 φανερώνει το μέγεθος της συλλογής των κειμένων.

Όσον αφορά το περιεχόμενο το οποίο έχει αναρτηθεί, κι ενδεχομένως τροποποιηθεί μέσω των ειδικών συναρτήσεων που εξετάζουμε στη συνέχεια, αποθηκεύεται στη δομή της γραμμής 16, ονόματι **contents**, ενώ τα δικαιώματα χρήσης του κάθε εγγράφου βρίσκονται στη δομή της γραμμής 15 ως mode bits, ονόματι **flags**. Η προσπέλαση αμφότερων δομών γίνεται βάσει του αναγνωριστικού του περιεχομένου που έχει αναρτηθεί, ενώ το σύνολο των αναγνωριστικών των εγγράφων που είναι διαθέσιμα ανά πάσα χρονική στιγμή βρίσκεται στη δομή **docids** της γραμμής 17. Στη γραμμή 19 βρίσκουμε μία βοηθητική μεταβλητή η οποία αντιπροσωπεύει τον αριθμό των διαθέσιμων εγγράφων και χρησιμοποιούμε προκειμένου να προσπελαύνουμε τα αναγνωριστικά της **docids** δομής.

```
1:    def chown (user):
2:        if user and self.owner == msg.sender:
3:            log("** User '"+msg.sender+"' changes owner to "+user+".")
4:            self.owner = user
5:            return(1)
6:        log("** User '"+msg.sender+"' unsuccessfully tried
                                      to change owner to "+user+".")
7:        return(0)
```

Σχήμα 3.21: Μέθοδος αλλαγής διαχειριστή του serpent smart-contract.

```
1:    def add_user (user):
2:        if user and self.owner == msg.sender:
3:            log("** Owner '"+msg.sender+"' adds
                                    user '"+user+"' to the group.")
4:            if not self.users[user] or self.users[user]<0:
5:                self.users[user] = block.timestamp
6:                self.group[self.groupsize] = user
7:                self.groupsize += 1
8:            return(1)
9:        log("** User '"+msg.sender+"' is unable to add
                                    user '"+user+"' to the group.")
10:        return(0)
```

Σχήμα 3.22: Μέθοδος προσθήκης χρήστη στην ειδική ομάδα.

Επιπλέον, στο Σχήμα 3.20 παρουσιάζουμε την μέθοδο **init** για την αρχικοποίηση του smart-contract, η οποία εκτελείται κατά τη δημιουργία ενός στιγμιοτύπου (ή αλλιώς constructor) κι ορίζει τον διαχειριστή ως τον χρήστη που το δημιούργησε. Στο Σχήμα 3.21 ορίζουμε τη συνάρτηση **chown** που μπορεί να χρησιμοποιήσει ο διαχειριστής προκειμένου να περάσει τα προνόμιά του σε κάποιον άλλον χρήστη. Ενδεικτικό το ότι δεν αφήνουμε κάποιον άλλον χρήστη να εκτελεσει τη συνάρτηση αυτή με επιτυχία μέσω κατάλληλων ελέγχων, όπως δείχνουμε στις γραμμή 2 του Σχήματος 3.21, ενώ στη γραμμή 3 "λογκάρουμε" το τι συμβαίνει για πιθανή χρήση στο μέλλον. Στη γραμμή 4 λαμβάνει χώρα η ανάθεση της μεταβλητής **owner** στη νέα τιμή της, και τέλος, στη γραμμή 5 τερματίζει το επιτυχές workflow με την κατάλληλη τιμή επιστροφής.

Στα Σχήματα 3.22 και 3.23 δείχνουμε τις συναρτήσεις **add_user** και **del_user** που μπορεί να χρησιμοποιήσει ο διαχειριστής προκειμένου είτε να προσθέσει, είτε να αφαιρέσει αντί-

```
 1:    def del_user (user):
 2:        if user and msg.sender == self.owner:
 3:            log("** Owner '"+msg.sender+"' removes
                                    user '"+user+"' from the group.")
 4:            if self.users[user] and self.users[user]>=0:
 5:                self.users[user] = -1
 6:                i = 0
 7:                while i < self.groupsize:
 8:                    if self.group[i] == user:
 9:                        if i < self.groupsize - 1:
10:                            self.group[i] = self.group[self.groupsize-1]
11:                        self.group[self.groupsize-1] = 0
12:                        self.groupsize -= 1
13:                        break
14:                    i += 1
15:                return(1)
16:        log("** User '"+msg.sender+"' is unable
                                     to delete user '"+user+"'.")
17:        return(0)
```

Σχήμα 3.23: Μέθοδος διαγραφής χρήστη από την ειδική ομάδα.

στοιχα, χρήστες από την ειδική ομάδα του smart-contract. Ο έλεγχος για το ποιός εκτελεί την συνάρτηση φαίνεται στη γραμμή 2 σε αμφότερες μεθόδους, ενώ για την μέθοδο **add_user** στη γραμμή 3 ελέγχουμε αν η διεύθυνση του χρήστη εμφανίζεται για πρώτη φορά. Εάν ισχύει η συνθήκη αυτή, τότε κάνουμε την εισαγωγή της διεύθυνσης στην ειδική mapping δομή **users** στη γραμμή 5 και στη λίστα **group** στη γραμμή 6. Αντίστοιχα, στη μέθοδο **del_user** ελέγχουμε ότι ο χρήστης προυπάρχει προκειμένου να τον διαγράψουμε από το mapping στις γραμμές 4 και 5, ενώ στις γραμμές 6–14 αφαιρούμε τη διεύθυνση του χρήστη κι από τις βοηθητικές δομές. Σε ενδεχόμενο εκτέλεσης των συγκεκριμένων συναρτήσεων από οιονδήποτε άλλο χρήστη επιστρέφουμε στο σύστημα τον ανάλογο κωδικό σφάλματος, 0 ως **false**.

```
 1:    def add_doc (docid, content:str, flag):
 2:        if docid and content and flag and msg.sender == self.owner:
 3:            if not flag:
 4:                flag = 640
 5:            save(self.contents[docid].content[0],content,chars=len(content))
 6:            self.contents[docid].length = len(content)
 7:            self.flags[docid] = flag
 8:
 9:            self.docids[self.repossize] = docid
10:            self.repossize += 1
11:
12:            log("** User '"+msg.sender+"' added document
                            with id '"+docid+"' and "+flag+" rights.")
13:            return(1)
14:        log("** User '"+msg.sender+"' failed to add document
                            with id '"+docid+"' and "+flag+" rights.")
15:        return(0)
```

Σχήμα 3.24: Μέθοδος προσθήκης περιεχομένου.

Στο Σχήμα 3.24 παρουσιάζουμε την μέθοδο εισαγωγής περιεχομένου **add_doc**, η οποία

```
 1:    def chmod (docid, newflag):
 2:        if newflag and self.contents[docid].length:
 3:            flag = self.flags[docid]
 4:            if not flag:
 5:                flag = 640
 6:            owner_right = (flag % 1000) / 100
 7:            group_right = (flag % 100) / 10
 8:            rest_right = (flag % 10)
 9:            if ((msg.sender == self.owner
                   and (owner_right == 2 or owner_right == 6)) \
10:            or (self.users[msg.sender] and self.users[msg.sender]>=0
                   and (group_right == 2 or group_right == 6)) \
11:            or (rest_right == 2 or rest_right == 6)):
12:                self.flags[docid] = newflag
13:                log("** User '"+msg.sender+"' changed document's '"
                       +docid+"' rights to "+newflag+" from "+flag+".")
14:                return(1)
15:        log("** User '"+msg.sender+"' is unable to change document's '"
               +docid+"' rights to "+newflag+".")
16:        return(0)
```

Σχήμα 3.25: Μέθοδος αλλαγής δικαιωμάτων χρήσης περιεχομένου.

επίσης εκτελείται από τον διαχειριστή του smart-contract, όπως δείχνουμε άλλωστε στη γραμμή 2. Αξίζει να τονίσουμε την ιδιαιτερότητα που χρήζει η αντιμετώπιση των αλφαριθμητικών, τόσο για την αποθήκευση στις δομές του smart-contract, όσο και για την ανάκτηση αυτών. Για παράδειγμα, στη γραμμή 5 γίνεται κλήση ενός blockchain system-call προκειμένου να αποθηκευθεί το νέο περιεχόμενο στην επιθυμητή θέση της δομής **contents**, αλλάζοντας έτσι την κατάσταση (state) με το επόμενο block που προσαρτάται, ενώ παράλληλα δίνεται ρητά (explicitly) ως όρισμα και το μήκος της συμβολοσειράς. Η "δυστροπία" αυτή ως προκύπτει, πηγάζει από το γεγονός ότι τα αλφαριθμητικά από μόνα τους δεν είναι ένας πρωτεύων (primitive) τύπος δεδομένων στο blockchain, αλλά κατ' ουσίαν πρόκειται για έναν πίνακα (array) ξεχωριστών χαρακτήρων, και για αυτό απαιτείται η χρήση ειδικών συναρτήσεων για τη διαχείρισή τους. Η ιδιαιτερότητα αυτή γίνεται ιδιαίτερα εμφανής στις ακόλουθες γραμμές όπου η ανάθεση δικαιωμάτων χρήσης και μήκους περιεχομένου γίνονται απευθείας με μία απλή ανάθεση μεταβλητής, χωρίς βέβαια τη χρήση ειδικών μεθόδων.

Επιπλέον, δίνεται η δυνατότητα αλλαγής των δικαιωμάτων χρήσης του εισαγώμενου περιεχομένου μέσω της μεθόδου **chmod** του Σχήματος 3.25. Προυπόθεση για να γίνει με επιτυχία η μεταβολή των δικαιωμάτων χρήσης είναι ο καλών να έχει ήδη δικαιώματα τροποποίησης του συγκεκριμένου εγγράφου. Έτσι όμως έχει και την ευχέρεια να αποκλείσει τον εαυτό του ή ακόμα και τον διαχειριστή από συγκεκριμένα προνόμια χρήσης περιεχομένου, είτε εκούσια, είτε ακούσια. Ειδικότερα για το πως υπολογίζουμε τα δικαιώματα χρήσης ενός εγγράφου για κάθε μία κατηγορία χρηστών ξεχωριστά στις γραμμές 6–8, αφού ανακτήσουμε από την ανάλογη δομή τα δικαιώματα του συγκεκριμένου εγγράφου, θα χρησιμοποιήσουμε μόνο τα τρία/δύο/ένα τελευταία ψηφία για κάθε μία από τις εξής κατηγορίες διαχειριστής/ειδική ομάδα/λοιποί χρήστες υπολογίζοντας το modulo με το 1000/100/10 αντίστοιχα, και στη συνέχεια διαιρώντας με το 100/10/1 παράγουμε το μονοψήφιο νούμερο που αφορά την κάθε κατηγορία χρηστών ξεχωριστά. Αυτό συγκρίνεται με το 2, το 4, ή το 6, προκειμένου να διαπιστωθεί εάν έχουν παραχωρηθεί είτε δικαιώματα τροποποίησης, είτε προσπέλασης, είτε αμφότερα. Εάν δεν υπάρχει ταύτιση με κανένα από τα τρία προαναφερθέντα ψηφία τότε θεωρούμε ότι στη συγκεκριμένη κατηγορία χρηστών δεν έχει παραχωρηθεί κανένα παντελώς προνόμιο χρήσης.

Με την κλήση της μεθόδου **get_content** του Σχήματος 3.26 επιστρέφεται στον χρήστη το περιεχόμενο για το αναγνωριστικό που αιτείται, δεδομένου βέβαια ότι έχει τουλάχιστον

```
 1:    def get_content (docid):
 2:        if docid and self.contents[docid].length:
 3:            flag = self.flags[docid]
 4:            if not flag:
 5:                flag = 640
 6:            owner_right = (flag % 1000) / 100
 7:            group_right = (flag % 100) / 10
 8:            rest_right = (flag % 10)
 9:            if ((msg.sender == self.owner
                   and (owner_right == 4 or owner_right == 6)) \
10:            or (self.users[msg.sender] and self.users[msg.sender]>=0
                   and (group_right == 4 or group_right == 6)) \
11:            or (rest_right == 4 or rest_right == 6)):
12:                length = self.contents[docid].length
13:                content = load(self.contents[docid].content[0],chars=length)
14:                log("** User '"+msg.sender+"' retrieved document '"+docid
                       +"' of "+length+" characters.")
15:                return (content:str)
16:        log("** User '"+msg.sender+"' is unable to retrieve
               document '"+docid+"'.")
17:        return(text(""):str)
```

Σχήμα 3.26: Μέθοδος ανάκτησης περιεχομένου.

δικαιώματα, ως φαίνεται στις γραμμές 3–11. Σε αυτήν την περίπτωση βρίσκουμε αρχικά το μήκος του περιεχομένου που μας ενδιαφέρει (γραμμή 12), κι εν συνεχεία, εξάγουμε το περιεχόμενο με το κατάλληλο system-call στη γραμμή 13.

Κατ' αντιστοιχία με το Σχήμα 3.24, όπου παρουσιάσαμε την μέθοδο εισαγωγής περιεχομένου, στη μέθοδο του Σχήματος 3.27 παρουσιάζουμε τη μέθοδο αφαίρεσης περιεχομένου από τις δομές και τα ευρετήρια του smart-contract, ονόματι **rmv_doc**. Εν αρχή ελέγχουμε στις γραμμές 3–11 ότι ο χρήστης έχει τουλάχιστον δικαιώματα τροποποίησης του εγγράφου. Εφόσον αυτό ισχύει, "μηδενίζουμε" τις τιμές που αντιστοιχούν στο συγκεκριμένο έγγραφο για όλες τις ειδικές δομές του smart-contract, ήτοι (i) **flags** για τα δικαιώματα χρήσης, (ii) **contents** για το μέγεθος του εγγράφου κι επακολούθως με μηδενικού μήκους περιεχόμενο στην ίδια δομή, και τέλος, (iii) αφαίρεση του αναγνωριστικού από τη λίστα **docsids**. Η μέθοδος μετονομασίας περιεχομένου **rnm_doc**, που παρουσιάζουμε στο Σχήμα 3.28, έχει πολύ παρόμοια λειτουργία με την προαναφερθείσα μέθοδο με τη διαφορά όμως ότι αντί να "μηδενίζονται" και να αφαιρούνται το περιεχόμενο και τα στοιχεία που σχετίζονται με αυτό, αντ' αυτού τα στοιχεία αυτά αποθηκεύονται ξανά με νέο και διαφορετικό αναγνωριστικό (γραμμές 12–17, 23–28), ενώ όλες οι εγγραφές που είχαν δεικτοδοτηθεί με το παλιό αναγνωριστικό αφαιρούνται στις γραμμές 19–21 με τον τρόπο που περιγράψαμε για την προηγούμενη μέθοδο. Ένας απλός τρόπος να περιγράψει κανείς αυτήν την μέθοδο θα ήταν ότι αποτελείται από την προαναφερθείσα μέθοδο συν ένα επιπλέον βήμα αποθήκευσης των στοιχείων με νέο αναγνωριστικό πριν τη διαγραφή του παλιού αναγνωριστικού.

Εν συνεχεία, περιγράφουμε τις μεθόδους **get_readable_docs** και **get_writable_docs** στα Σχήματα 3.29 και 3.30 αντίστοιχα, σύμφωνα με τις οποίες ο χρήστης ανακτά μία λίστα με τα αναγνωριστικά των εγγράφων, τα οποία είναι είτε τουλάχιστον προσπελάσιμα για την πρώτη μέθοδο, είτε τουλάχιστον τροποιήσιμα για τη δεύτερη. Η διαδικασία που ακολουθούμε είναι η εξής: για κάθε ένα από τα αναγνωριστικά της λίστας **docids**, εξετάζουμε τα δικαιώματα χρήσης που του αντιστοιχούν και τα συγκρίνουμε με την κατηγορία χρήστη στην οποία ανήκει ο καλών (γραμμές 4–18). Εάν τα δικαιώματα χρήσης του διαχειριστή δίνουν τέτοια πρόσβαση και ταιριάζουν με την ιδιότητα του καλώντος (γραμμές 7–15), τότε το αναγνωριστικό προστίθεται στα αποτελέσματα στη γραμμή 16. Ομοίως, αν τα δικαιώματα χρήσης της ειδικής ομάδας είναι προσβάσιμα κι ο χρήστης ανήκει στην ομάδα χρηστών. Διαφορετικά, προκειμένου να

```
 1:    def rmv_doc (docid):
 2:        if docid and self.contents[docid].length:
 3:            flag = self.flags[docid]
 4:            if not flag:
 5:                flag = 640
 6:            owner_right = (flag % 1000) / 100
 7:            group_right = (flag % 100) / 10
 8:            rest_right = (flag % 10)
 9:            if ((msg.sender == self.owner
                   and (owner_right == 2 or owner_right == 6)) \
10:            or (self.users[msg.sender] and self.users[msg.sender]>=0
                   and (group_right == 2 or group_right == 6)) \
11:            or (rest_right == 2 or rest_right == 6)):
12:                self.flags[docid] = 0
13:                self.contents[docid].length = 0
14:                save(self.contents[docid].content[0],"",chars=0)
15:
16:                i = 0
17:                while i<self.repossize:
18:                    if docid == self.docids[i]:
19:                        if i < self.repossize-1:
20:                            self.docids[i] = self.docids[self.repossize-1]
21:                        self.docids[self.repossize-1] = 0
22:                        self.repossize -= 1
23:                        break
24:                    i += 1
25:
26:                log("** User '"+msg.sender+"' removed
                       document '"+docid+"'.")
27:                return (1)
28:        log("** User '"+msg.sender+"' is unable to remove
               document '"+docid+"'.")
29:        return(0)
```

Σχήμα 3.27: Μέθοδος διαγραφής περιεχομένου.

```
 1:    def rnm_doc (docid0,docid1):
 2:        if docid0 and docid1 and self.contents[docid0].length:
 3:            flag = self.flags[docid0]
 4:            if not flag:
 5:                flag = 640
 6:            owner_right = (flag % 1000) / 100
 7:            group_right = (flag % 100) / 10
 8:            rest_right = (flag % 10)
 9:            if ((msg.sender == self.owner
                   and (owner_right == 2 or owner_right == 6)) \
10:            or (self.users[msg.sender] and self.users[msg.sender]>=0
                   and (group_right == 2 or group_right == 6)) \
11:            or (rest_right == 2 or rest_right == 6)):
12:                self.flags[docid1] = self.flags[docid0]
13:
14:                length = self.contents[docid0].length
15:                content = load(self.contents[docid0].content[0],chars=length)
16:                save(self.contents[docid1].content[0],content,chars=length)
17:                self.contents[docid1].length = length
18:
19:                self.flags[docid0] = 0
20:                self.contents[docid0].length = 0
21:                save(self.contents[docid0].content[0],"",chars=0)
22:
23:                i = 0
24:                while i<self.repossize:
25:                    if docid0 == self.docids[i]:
26:                        self.docids[i] = docid1
27:                        break
28:                    i += 1
29:
30:                log("** User '"+msg.sender+"' is renamed document '"
                       +docid0+"' to '"+docid1+"'.")
31:                return (1)
32:        log("** User '"+msg.sender+"' is unable to rename document '"
               +docid0+"' to '"+docid1+"'.")
33:        return(0)
```

Σχήμα 3.28: Μέθοδος μετονομασίας περιεχομένου.

```
 1:    def get_readable_docs ():
 2:        docList = array(self.repossize)
 3:        counter = 0
 4:        i = 0
 5:        while i<self.repossize:
 6:            docid = self.docids[i]
 7:            flag = self.flags[docid]
 8:            if not flag:
 9:                flag = 640
10:            owner_right = (flag % 1000) / 100
11:            group_right = (flag % 100) / 10
12:            rest_right = (flag % 10)
13:            if ((msg.sender == self.owner
                   and (owner_right == 4 or owner_right == 6)) \
14:            or (self.users[msg.sender] and self.users[msg.sender]>=0
                   and (group_right == 4 or group_right == 6)) \
15:            or (rest_right == 4 or rest_right == 6)):
16:                docList [counter] = docid
17:                counter += 1
18:            i += 1
19:        log("** User '"+msg.sender+"' retrieved the ids of "
               +counter+" readable documents.")
20:        return(slice(docList,items=0,items=counter):arr)
```

Σχήμα 3.29: Μέθοδος ανάκτησης αναγνωριστικών προσβάσιμου περιεχομένου.

```
 1:    def get_writable_docs ():
 2:        docList = array(self.repossize)
 3:        counter = 0
 4:        i = 0
 5:        while i<self.repossize:
 6:            docid = self.docids[i]
 7:            flag = self.flags[docid]
 8:            if not flag:
 9:                flag = 640
10:            owner_right = (flag % 1000) / 100
11:            group_right = (flag % 100) / 10
12:            rest_right = (flag % 10)
13:            if ((msg.sender == self.owner
                   and (owner_right == 2 or owner_right == 6)) \
14:            or (self.users[msg.sender] and self.users[msg.sender]>=0
                   and (group_right == 2 or group_right == 6)) \
15:            or (rest_right == 2 or rest_right == 6)):
16:                docList [counter] = docid
17:                counter += 1
18:            i += 1
19:        log("** User '"+msg.sender+"' retrieved the ids of "
               +counter+" writable documents.")
20:        return(slice(docList,items=0,items=counter):arr)
```

Σχήμα 3.30: Μέθοδος ανάκτησης αναγνωριστικών τροποποιήσιμου περιεχομένου.

επιλεγεί το εξεταζόμενο έγγραφο, θα πρέπει να δίνουν πρόσβαση τα δικαιώματα χρήσης που σχετίζονται με τους υπόλοιπους χρήστης. Αν ούτε αυτά εγγυώνται την πρόσβαση, τότε δυστυχώς, το αναγνωριστικό δεν προστίθεται στα αποτελέσματα κι εξετάζουμε το επόμενο διαθέσιμο αναγνωριστικό, μέχρις ότου αυτά να τελειώσουν κι η μέθοδος να επιστρέψει τη λίστα που έχει κατασκευάσει συγκεντρώνοντας **counter** τόσα αναγνωριστικά (γραμμή 20).

Οι τελευταίες δύο μέθοδοι που θα παρουσιάσουμε στα Σχήματα 3.31 και 3.32, ονόματι **edit_delete** κι **edit_insert** αντίστοιχα, αφορούν την τροποποίηση του περιεχομένου καθεαυτού. Προφανώς, αμφότερες απαιτούν από τον χρήστη που τις χρησιμοποιεί να διαθέτει τουλάχιστον δικαιώματα τροποποίησης για το συγκεκριμένο έγγραφο. Η σύγκριση αυτή λαμβάνει χώρα στις γραμμές 3–11, κι εφόσον είναι επιτυχής το περιεχόμενο φορτώνεται αρχικά από τον αποθηκευτικό χώρο που έχει δεσμευτεί στο blockchain. Από το σημείο αυτό όμως και πέρα οι δύο μέθοδοι διαφοροποιούνται αρκετά. Πιο συγκεκριμένα, για τη μέθοδο **edit_delete** αφού ελέγχουμε ότι το μήκος της διαγραφής δεν υπερβαίνει το τέλος του εγγράφου στη γραμμή 13, πανωγράφουμε στις γραμμές 16–20 το array χαρακτήρων που αντιστοιχεί στο περιεχόμενο από τη θέση που ξεκινάει η διαγραφή, με τους χαρακτήρες που αντιστοιχούν μετά το τέλος της διαγραφής. Αφού υπολογίσουμε το νέο μήκος της συμβολοσειράς, την αποθηκεύουμε στο ίδιο σημείο όπου υπήρχε το έγγραφο στην προηγούμενή του μορφή (γραμμή 22), μαζί με το νέο μήκος (γραμμή 23). Όμως, η μέθοδος **edit_insert** είναι πιο πολύπλοκη αφού πρέπει να δημιουργήσει πρώτα το χώρο που θα προστεθεί το νέο περιεχόμενο, καθώς και να ολισθήσει κατάλληλα τους χαρακτήρες που αντιστοιχούν στις θέσεις αυτές στις γραμμές 18–22 για να γεμίσει τις σωστές θέσεις του array στις γραμμές 23–27 με το νέο περιεχόμενο που έχει δώσει ο χρήστης πρωτού αυτό αποθηκευτεί κατάλληλα στη γραμμή 28 στο σωστό μήκος.

```
 1:    def edit_delete (docid, pos, edit_length):
 2:        if docid and pos and edit_length and self.contents[docid].length:
 3:            flag = self.flags[docid]
 4:            if not flag:
 5:                flag = 640
 6:            owner_right = (flag % 1000) / 100
 7:            group_right = (flag % 100) / 10
 8:            rest_right = (flag % 10)
 9:            if ((msg.sender == self.owner
                   and (owner_right == 2 or owner_right == 6)) \
10:            or (self.users[msg.sender] and self.users[msg.sender]>=0
                   and (group_right == 2 or group_right == 6)) \
11:            or (rest_right == 2 or rest_right == 6)):
12:                length = self.contents[docid].length
13:                if pos+edit_length < length:
14:                    content = load(self.contents[docid].content[0],
                           chars=length)
15:
16:                    i = pos + edit_length
17:                    while i < length:
18:                        c = getch(content,i)
19:                        setch(content,i-edit_length,c)
20:                        i = i + 1
21:                    new_length = length - edit_length
22:                    save(self.contents[docid].content[0],
                           content,chars=new_length)
23:                    self.contents[docid].length = new_length
24:                    log("** User '"+msg.sender+"' changed document '"
                           +docid+"' to "+new_length+" characters
                           from "+length+".")
25:                    return(content:str)
26:        log("** User '"+msg.sender+"' is unable to remove content
               from document '"+docid+"'.")
27:        return(text(""):str)
```

Σχήμα 3.31: Μέθοδος μεταβολής περιεχομένου μετ' αφαιρέσεως.

```
 1:    def edit_insert (docid, pos, addendum:str):
 2:        if docid and pos and addendum and self.contents[docid].length:
 3:            flag = self.flags[docid]
 4:            if not flag:
 5:                flag = 640
 6:            owner_right = (flag % 1000) / 100
 7:            group_right = (flag % 100) / 10
 8:            rest_right = (flag % 10)
 9:            if ((msg.sender == self.owner
                   and (owner_right == 2 or owner_right == 6)) \
10:            or (self.users[msg.sender] and self.users[msg.sender]>=0
                   and (group_right == 2 or group_right == 6)) \
11:            or (rest_right == 2 or rest_right == 6)):
12:                length = self.contents[docid].length
13:                edit_length = len(addendum)
14:                new_length = length + edit_length
15:
16:                content = load(self.contents[docid].content[0],
                       chars=new_length)
17:
18:                i = length - 1
19:                while i >= pos:
20:                    c = getch(content,i)
21:                    setch(content,i+edit_length,c)
22:                    i = i - 1
23:                i = 0
24:                while i < edit_length:
25:                    c = getch(addendum,i)
26:                    setch(content,pos+i,c)
27:                    i = i + 1
28:                save(self.contents[docid].content[0],
                       content,chars=new_length)
29:                self.contents[docid].length = new_length
30:                log("** User '"+msg.sender+"' changed document '"
                       +docid+"' to "+new_length+" characters from "+length+".")
31:                return(content:str)
32:        log("** User '"+msg.sender+"' is unable to add content
               to document '"+docid+"'.")
33:        return(text(""):str)
```

Σχήμα 3.32: Μέθοδος μεταβολής περιεχομένου με προσθήκη.

Κεφάλαιο 4

Διασύνδεση με web περιβάλλοντα

Technology...the knack of so arranging the world that we don't have to experience it.

Max Frisch (1911–1991)

Στο παρόν κεφάλαιο θα συζητήσουμε την ανάπτυξη της εφαρμογής από το γραφικό περιβάλλον προς το blockchain backend αφού επεξηγήσουμε αρχικά κάποια θεωρητικά θέματα για την οργάνωση και τη δομή μίας σύγχρονης εφαρμογής διεπαφής, αλλά και πρακτικά θέματα τεχνολογίας. Αρχικά, θεωρούμε ότι το smart-contract που έχει σχεδιαστεί με τη μία ή την άλλη γλώσσα υπάρχει ήδη στο blockchain [33] και πλέον στοχεύοντας σε μία full-stack λύση αποσκοπούμε σε αυτό το κεφάλαιο το σχεδιασμό μίας καλά δομημένης client-side εφαρμογής που να προσφέρει μία αποτελεσματική γραφική διεπαφή, παράθυρο προς την υποδομή του Ethereum.

4.1 Πρότυπα αρχιτεκτονικής λογισμικού

4.1.1 MVC και MVVM αρχιτεκτονικές

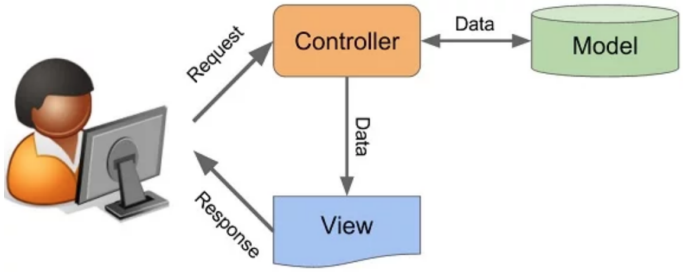
Το Model-View-Controller (MVC) είναι ένα μοντέλο αρχιτεκτονικής λογισμικού, το οποίο χρησιμοποιείται για την δημιουργία περιβαλλόντων αλληλεπίδρασης χρήστη. Σύμφωνα με αυτό, η εφαρμογή διαιρείται σε τρία διασυνδεδεμένα μέρη ώστε να διαχωριστεί το μέρος της εφαρμογής υπεύθυνο για την παρουσίαση της πληροφορίας στον χρήση από την μορφή που έχει αποθηκευτεί στο σύστημα, αλλά κι από τη λογική της εφαρμογής. Επίσης, το κύριο μέρος του προτύπου είναι το **model**, το οποίο διαχειρίζεται την ανάκτηση/αποθήκευση των δεδομένων στο σύστημα ανεξάρτητα της όποιας διεπαφής. Δεν πρόκειται απαραίτητα για μία βάση γνώσης ή δεδομένων. Μπορεί προκειμένου να ανακτήσει τα απαραίτητα δεδομένα για το view να καταναλώσει από κάποιο web-service. Το **view** χρησιμοποιείται μόνο για να παρουσιάζεται η πληροφορία στον χρήστη. Ακόμα, η ευελιξία της αρχιτεκτονικής επιτρέπει τη συνύπαρξη πολλαπλών views πάνω στο ίδιο μοντέλο, π.χ. διαφορετικές αναπαραστάσεις των ίδιων δεδομένων για διαφορετικούς σκοπούς και τύπους χρηστών. Ο **controller** δέχεται την είσοδο και στέλνει εντολές σε model και view (Σχήμα 4.1). Εκτός από το να διαιρείται η εφαρμογή σε τρία μέρη, το αρχιτεκτονικό μοντέλο ορίζει και τις αναμεταξύ τους αλληλεπιδράσεις:

controller στέλνει εντολές και ενημερώνει την κατάσταση του μοντέλου. Επίσης στέλνει εντολές ώστε να γίνει η αντίστοιχη αναπαράσταση των δεδομένων του μοντέλου μέσω του view.

model ενημερώνει τα αντίστοιχες views και τους controllers όταν υπάρχει αλλαγή στα δεδομένα. Αυτή η ενημέρωση επιτρέπει στα views να ανανεώνουν ανάλογα την γραφική απεικόνιση.

view αναπαριστά με γραφικό τρόπο την πληροφορία που περιέχει το model δημιουργώντας μία γραφική παρουσίαση στο χρήστη.

Με το να αποσυνδέονται τα μέρη της εφαρμογής σε ξεχωριστές οντότητες επιτρέπεται η παράλληλη ανάπτυξη του πηγαίου κώδικα κι η επαναληπτική χρήση διαφόρων modules όταν χρειάζεται. Επίσης, επιτρέπει την προσαρμογή της εφαρμογής για διαφορετικά συστήματα κάθε φορά.



Σχήμα 4.1: Diagram of interactions within the MVC pattern.

Σχετικά με την Model-View-ViewModel (MVVM) αρχιτεκτονική, αυτή υλοποιεί το sofware design pattern του **mediator**[16], οργανώνοντας την πρόσβαση από το view προς το model σύμφωνα με την υποστηριζόμενη λειτουργικότητα της εφαρμογής. Το module που έχει αναλάβει αυτή τη διασύνδεση είναι το **view model**, που αποτελεί μία μοντελοποίηση των χαρακτηριστικών και λειτουργιών που χρειάζεται το view, ήτοι δημόσια πεδία κι εντολές. Με άλλα λόγια, στην αρχιτεκτονική αυτή αντί για το component του controller του MVC, ή τον presenter του MVP, το MVVM διατηρεί αυτού του είδους τη διασύνδεση μεταξύ view και model με ένα ξεχωριστό στοιχείο ώστε να αυτοματοποιήσει την επικοινωνία απαραίτητη για την επίτευξη της λειτουργικότητας που υποστηρίζει και διαθέτει η εφαρμογή προς τον χρήστη.

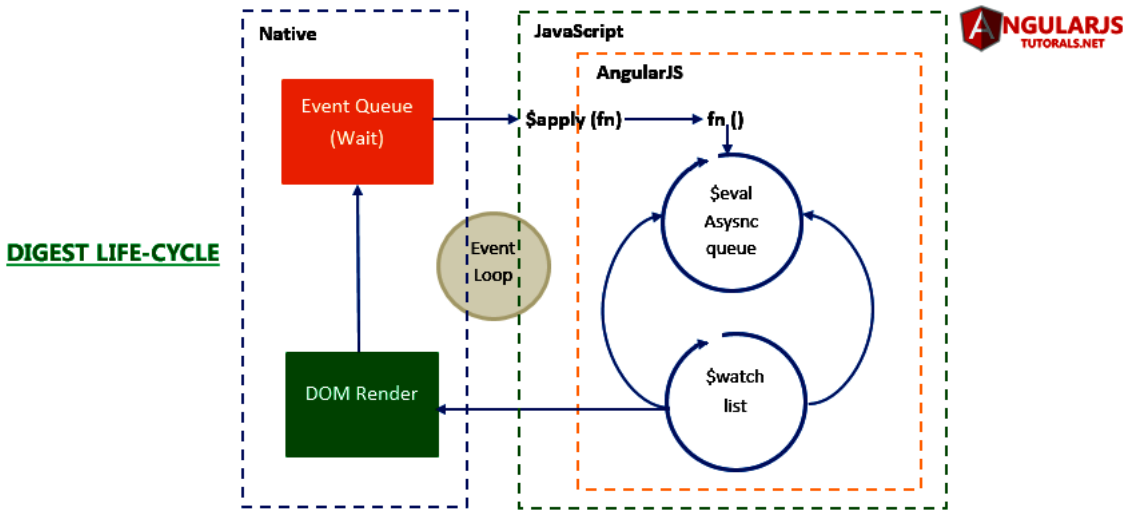
Η ειδοποιός διαφορά μεταξύ του view model component και του presenter της MVP αρχιτεκτονικής, είναι ότι δεν διατηρείται από το view model αναφορά προς το view. Εν αντιθέσει, το view διασυνδέεται με συγκεκριμένα στοιχεία του view model προκειμένου να τα ενημερώνει και να ενημερώνεται. Για να επιτευχθεί αυτή η επικοινωνία είναι απαραίτητη η χρήση μίας τέτοιου σκοπού τεχνολογίας, ή η παραγωγή του boilerplate κώδικα που θα αναλάβει το σκοπό αυτόν. Κι άρα ο προγραμματιστής δεν επιβαρύνεται με την ανάπτυξη της λογικής που θα συγχρονίζει το view model με το view, και μπορεί έτσι να συγκεντρωθεί σε πιο κρίσημα σημεία των υπόλοιπων modules. Κάτι που δεν ισχύει για τις άλλες αρχιτεκτονικές που αναφέραμε προηγουμένως, MVP και MVC.

Εν κατακλείδι, η MVVM αρχιτεκτονική προσφέρει "**loose coupling**, **high cohesion**" ωφέλη με το διαχωρισμό στα επί μέρους components, σε συνδυασμό με την πρωτοποριακά άμεση κι αμφίδρομη διασύνδεση των δεδομένων και της απεικόνισής τους μέσω της λογικής της εφαρμογής και του boilerplate κώδικα που προσφέρουν frameworks όπως τα AngularJS, EmberJS, KnockoutJS και ReactJS.

4.1.2 AngularJS

Πλέον, το πιο δημοφιλές MVVM framework για διαδικτυακές εφαρμογές ονομάζεται AngularJS[27], κι όπως άλλωστε υποδηλώνει κι η ονομασία του βασίζεται στη γλώσσα προγραμματισμού JavaScript. Πρόκειται για ένα περιβάλλον ανάπτυξης client-side εφαρμογών με συγκεκριμένα χαρακτηριστικά που ενδεχομένως να το καθιστούν και να το αναδεικνύουν ως εξαιρετική λύση για συγκεκριμένου τύπου προβλήματα εφαρμογών, π.χ. λόγω υπερβολικά μεγάλου latency με έναν υπεραντλαντικό server, ή η σχεδιαστική προδιαγραφή ο server να έχει την ελάχιστη επικοινωνία με τους clients προκειμένου να μεγιστοποιήσει τον πιθανό αριθμό χρηστών που εξυπηρετεί για περιεχόμενο που δεν ενημερώνεται ζωντανά. Για τη δική μας περίπτωση, μεγιστοποιούμε το UI/UX και responsiveness της εφαρμογής με το να

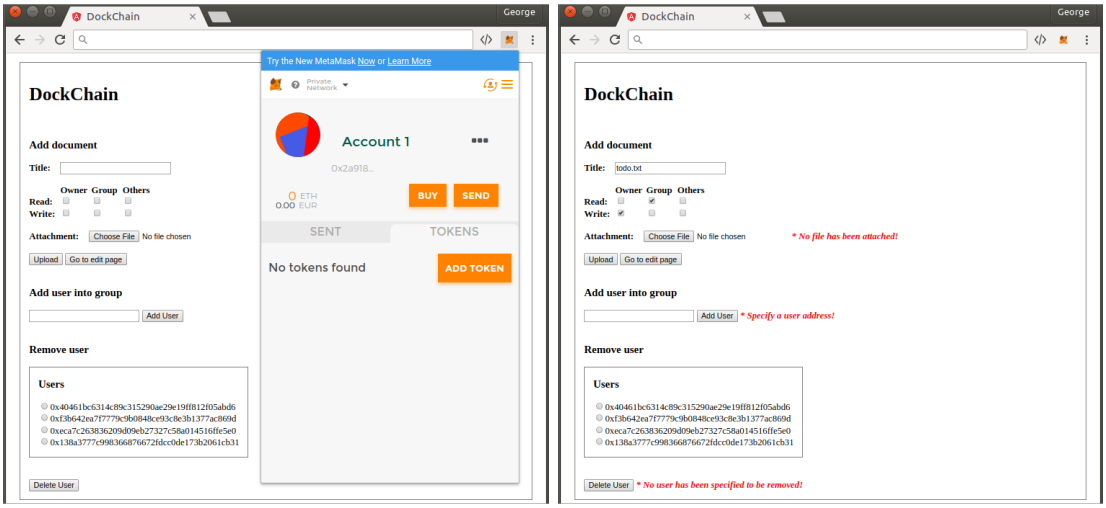
επιτρέπουμε στον browser του χρήστη να επικοινωνεί άμεσα με το blockchain χωρίς τη διαμεσολάβηση κάποιου server ως ενδεχομένο bottleneck, αλλά και λόγους ευελιξίας, κλιμάκωσης κι ανοχής σε σφάλματα, καθώς το peer-to-peer δίκτυο μπορεί να αντισταθμίσει την απώλεια αριθμού από κόμβους, π.χ. DDoS.



Σχήμα 4.2: AngularJS digest life-cycle event loop.

Σε πιο τεχνικό επίπεδο, μία όψη (view) για την AngularJS αποτελεί μία HTML σελίδα αφού μεταγλωττιστεί από το framework. Η όψη εμπεριέχει τις κατάλληλες δηλώσεις, ειδικά tags που ακολουθούν το συνακτικό της Angular (directives), ενώ δίνεται κι η δυνατότητα στον προγραμματιστή να δημιουργήσει custom tags και elements ως syntactic sugar προκειμένου να μεγιστοποιήσει την ευελιξία και τις δυνατότητες της εφαρμογής. Τα στοιχεία αυτά έχουν αντίκτυπο στον controller όπου επεξεργάζεται τις αλλαγές στην κατάστασή τους σε συνδυασμό με την είσοδο του χρήστη προκειμένου να επιτύχει την επιθυμητή συμπεριφορά της εφαρμογής. Στον πυρήνα του όμως ο controller είναι ένα JavaScript αντικείμενο αποτελούμενο από μεθόδους και πεδία. Όπως εξηγήσαμε προηγουμένως στη περιγραφή του MVVM αρχιτεκτονικού μοντέλου, view και controller είναι ξεχωριστά modules που δεν διασυνδέονται άμεσα σε κανένα επίπεδο. Το ρόλο του συνδετικού κρίκου μεταξύ των δύο αυτών ανεξάρτητων οντοτήτων παίζει το scope. Επί τη προκειμένη, ο controller κάνει διαθέσιμες συγκεκριμένες μεθόδους και πεδία προς το view με το να τα αναθέτει στο scope ή `$scope`. Δηλαδή, για τον controller υπάρχουν δύο κατηγορίες στοιχείων, αυτά που είναι τοπικά προς ιδίαν χρήση, κι αυτά που είναι διαθέσιμα προς το view, τα οποία ενδεχομένως να χρησιμοποιούν και τα στοιχεία της άλλης κατηγορίας, μεταβλητές και συναρτήσεις (κι αντίστροφα). Το scope με τους επονομαζόμενους Angular controllers απαρτίζουν το **view model**, ενώ όπως θα δούμε και στη συνέχεια, το model υλοποιείται συνήθως με τη μορφή Angular services και factories.

Ο κύκλος μεταγλώττισης λαμβάνει χώρα σε δύο φάσεις: (i) την επεξεργασία του DOM με τις AngularJS δηλώσεις, και (ii) την διασύνδεση των στοιχείων που εξήχθησαν κατά το προηγούμενο στάδιο με τα ομόλογα στοιχεία του scope. Με το πέρας της μεταγλώττισης είναι δυνατή η επικοινωνία των στοιχείων της εφαρμογής μέσω της δημιουργίας και της επεξεργασίας ειδικών γεγονότων από JavaScript αντικείμενα που παράγουν και καταναλώνουν events προκειμένου να συγχρονίζονται view και controller. Αν και η διαδικασία της μεταγλώττισης και της διασύνδεσης είναι αυτόματες, δίνεται στον προγραμματιστή η δυνατότητα να επέμβει προκειμένου να επιτύχει ένα διαφορετικό αποτέλεσμα με το να αναλάβει την περάτωση της διαδικασίας με "χειροκίνητο" τρόπο. Κάτι τέτοιο είθισται με την χρήση APIs και εξωτερικών βιβλιοθηκών, όπως η Web3.js στην περίπτωσή μας.



Σχήμα 4.3: Web interface για τον διαχειριστή του smart-contract.

4.2 Βάζοντας το Web3.js στην εξίσωση

Για τους σκοπούς της εφαρμογής μας σχεδιάσαμε δύο views:

admin.html για τη διαχείριση των χρηστών που προορίζεται για τον owner του contract,

edit.html για τη διαχείριση των εγγράφων που αν κι είναι ανοικτό σε όλους τους χρήστες, η αλληλεπίδραση με το περιεχόμενο επιτρέπεται σύμφωνα με τα δικαιώματα χρήσης που έχουν οριστεί κατά την εισαγωγή.

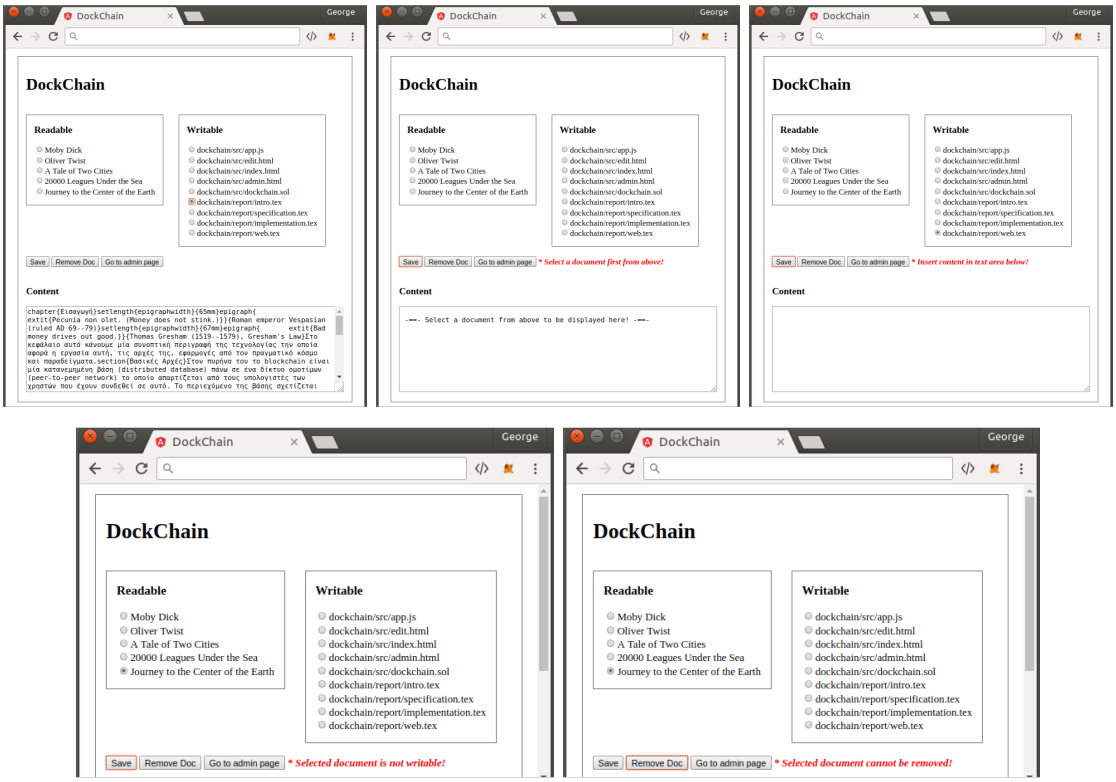
Κάθε ένα από τα views αλληλεπιδρά με έναν αλλά διαφορετικό controller ανά view, ήτοι `admin` και `edit` αντίστοιχα, με ξεχωριστό `$scope` έκαστο, τα οποία διαδραματίζουν σημαντικό ρόλο λειτουργώντας ως ενδιάμεσος της γραφικής διεπαφής με το singleton αντικείμενο που επιστρέφεται από το factory function για την επικοινωνία των controllers με το blockchain. Επιπλέον, το module διαθέτει και δύο σταθερές, (i) `SERVICE_URL` ως το endpoint για την επικοινωνία με το EVM, και (ii) `CONTRACT_ADDRESS` για την κλήση μεθόδων του deployed smart-contract στη συγκεκριμένη διεύθυνση του blockchain. Επίσης, έχουμε δύο view states της `ui.router` βιβλιοθήκης, ένα για κάθε view για την πλοήγηση από το ένα view στο άλλο κι αντίστροφα. Κι έτσι ολοκληρώνουμε την καταμέτρηση των στοιχείων της εφαρμογής της γραφικής διεπαφής τα οποία θα εξετάσουμε σε περισσότερο βάθος εν συνεχεία.

Στο Σχήμα 4.4 δείχνουμε τη δήλωση και τη δομή του module της εφαρμογής υπό τη μορφή Immediate Function Invocation[1] (IFI). Δεδομένου ότι έχουμε μόλις δύο views με έναν controller έκαστη, όπου αμφότεροι controllers απαιτούν πρόσβαση στο blockchain μέσω του ιδίου service, δεν χρειάζονται άνω του ενός module (γραμμή 4). Στις γραμμές 5–7 δηλώνουμε τις σταθερές του module οι οποίες γίνονται inject στα services και τους controllers όπου χρειάζεται, π.χ. στην γραμμή 29. Πιο συγκεκριμένα, το πρώτο όρισμα της δήλωσης αντιστοιχεί στο όνομα με το οποίο γίνεται συσχέτιση και χρήση στο view, ενώ εν συνεχεία δηλώνονται τα ονόματα των στοιχείων που γίνονται inject, και τέλος, η συνάρτηση που εμπεριέχει τη λογική του controller, service ή factory. Ακόμα, αξίζει να αναφέρουμε ότι το συντακτικό της Angular επιτρέπει άνω του ενός τρόπου προκειμένου να οριστεί ένα module κι η λειτουργικότητά του. Για παράδειγμα, αντί της δήλωσης του `$scope` θα μπορούσε να χρησιμοποιηθεί το keyword this. Κάτι τέτοιο ενδεχομένως να ήταν απαραίτητο σε περιπτώσεις όπου στο view λειτουργούν με αλληλοεπικαλυπτόμενο τρόπο και παράλληλα άνω του ενός controllers (π.χ. συσχετιζόμενοι με ιεραρχημένα `div` containers) όπου ενδεχομένως να δημιουργείται αμφισημία

[1] απαιτώντας την άμεση εκτέλεση της δηλωθείσας συνάρτησης κι όχι την ανάθεσή της.

```
 1:(function () {
 2:    'use strict';
 3:
 4:    angular.module('app',['ui.router'])
 5:        .constant('SERVICE_URL','http://localhost:8545')
 6:        .constant('CONTRACT_ADDRESS','0x50c72fbb28d10181cfb1acb8729c2...')
 7:        .constant('ABI',[...])
 8:        .service('blockchain',['CONTRACT_ADDRESS','SERVICE_URL','ABI',blockchain])
 9:        .controller('edit',['$scope','$state','blockchain',edit])
10:        .controller('admin',['$scope','$state','blockchain',admin])
11:        .config(function($stateProvider,$urlRouterProvider) {
12:            $urlRouterProvider.otherwise('/edit');
13:            var adminState = {
14:                name: 'admin',
15:                url: '/admin',
16:                templateUrl: 'admin.html',
17:                controller: admin
18:            };
19:            var editState = {
20:                name: 'edit',
21:                url: '/edit',
22:                templateUrl: 'edit.html',
23:                controller: edit
24:            };
25:            $stateProvider.state(adminState);
26:            $stateProvider.state(editState);
27:        });
28:
29:    function blockchain (CONTRACT_ADDRESS,SERVICE_URL,ABI) {
           ...
       }
       function admin ($scope,$state,blockchain) {
           ...
       }
       function edit ($scope,$state,blockchain) {
           ...
       }
   })();
```

Σχήμα 4.4: Angular module structure and declarations.



Σχήμα 4.5: Web interface για τη διαχείριση προσπελάσιμων και τροποιήσιμων εγγράφων.

για κάποια από τα στοιχεία τους. Επίσης κάτι τέτοιο θα ήταν ένας έμμεσος τρόπος να υλοποιηθεί η κληρονομικότητα μεταξύ των controllers, αλλά αυτό υπεισέρχεται σε θέματα πέρα από τους σκοπούς του παρόντος και δε θα μας απασχολήσει περισσότερο εδώ για λόγους απλότητας καθώς δε χρειάζεται να προσδόσουμε τέτοια χαρακτηριστικά σε μία τέτοια απλή εφαρμογή.

Στις γραμμές 11–27 δηλώνουμε τα view states στα οποία μπορούμε να πλοηγηθούμε μέσα από τη λογική που κατασκευάζουμε εντός των controllers με εντολές της μορφής `$state.go('edit');` για παράδειγμα. Όπως δείχνουμε, το κάθε view state έχει ένα όνομα εν είδει αναγνωριστικού, ένα URL το οποίο δεν είναι απαραίτητα ίδιο ή όμοιο με το όνομα του html αρχείου που εμπεριέχει, ένα html template με Angular directives για τη γραφική απεικόνιση του view, καθώς κι έναν controller του οποίου το scope συσχετίζεται με το προαναφερθέν view. Δεν χρειαζόμαστε και δεν είναι σκόπιμη περισσότερη πολυπλοκότητα για τη συγκεκριμένη εφαρμογή, π.χ. δευτερεύουσα view, κ.ο.κ. Στη γραμμή 12 δηλώνουμε το προεπιλεγμένο url σε περίπτωση όπου η διεύθυνση που πληκτρολογεί ο χρήστης δεν είναι πλήρης ή σωστή. Τέλος, με τις γραμμές 25 και 26 κάνουμε διαθέσιμα τα δηλωθέντα states στη λογική του module. Οι δηλώσεις αυτές γίνονται εντός του configuration του module και επενεργούν πολύ νωρίς, μόλις κατά τη διάρκεια κατασκευής του πριν αρχικοποιηθεί με οτιδήποτε άλλο. Από τη γραμμή 29 κι έπειτα υλοποιούμε τη λογική της client-side, όπως θα δούμε αναλυτικά για κάθε ένα από τα τρία αυτά στοιχεία παρακάτω.

Στο Σχήμα 4.6 παρουσιάζουμε την υλοποίηση του service το οποίο προσφέρει τη λειτουργικότητα του blockchain σαν το backend της εφαρμογής μας μέσω της third party βιβλιοθήκης Web3.js [5, 26]. Αλλά θα πρέπει φυσικά να αρχικοποιήσουμε τη διεπαφή μας με έναν provider που συνδέεται με τον EVM του οποίου η διεύθυνση δίνεται ως μοναδικό όρισμα στη γραμμή 2. Το URL αυτό μπορεί να είναι είτε κάποιου κόμβου του Ethereum blockchain, είτε κάποιου blockchain που έχει γίνει deploy ανεξάρτητα [8, 9], ή ακόμα και τοπικού κόμβου [23, 30] για όσο είμαστε ακόμα στο στάδιο της ανάπτυξης της εφαρμογής. Αφού έχουμε έτοιμο τον provider, θα δημιουργήσουμε το reflection του smart-contract στη γραμμή 3 δίνοντας σε JSON

format το σύνολο των public συναρτήσεων, τον τύπο τους (π.χ. constant, payable), τα αναμενόμενα ορίσματά τους, καθώς και τον τύπο της τιμής επιστροφής κάθε μίας από αυτές, γνωστό κι ως ABI (Application Binary Interface). Έτσι θα μπορεί η βιβλιοθήκη να τροφοδοτεί τη διεπαφή με τις ανάλογες δομές εισόδου, αλλά και να επεξεργάζεται κατάλληλα την τιμή της εξόδου ώστε αυτή να είναι σε θέση να ανατεθεί σε κάποια μεταβλητή. Στη γραμμή 4 αρχικοποιούμε με τη διεύθυνση του smart-contract στην οποία θα συνδεθεί η διεπαφή μας ώστε να είναι δυνατή η κλήση των δημοσίων μεθόδων από ένα συγκεκριμένο instance του smart-contract. Τέλος, στη γραμμή 5 επιστρέφει το service μία συνάρτηση, υποδηλώνοντας έτσι ότι πρόκειται για factory με τη διαφορά ότι η τιμή του πεδίου `dockchain` της μεταβλητής επιστροφής είναι singleton[2]. Πραγματικά πολύ λίγος κώδικας, αλλά χτισμένος με δομημένο τρόπο σύμφωνα με το software design pattern που υπακούει το framework.

```
1:    function blockchain (CONTRACT_ADDRESS,SERVICE_URL,ABI) {
2:        var web3 = new Web3(new Web3.providers.HttpProvider("http://...:8545"));
3:        var dockchainContract = web3.eth.contract(ABI);
4:        var dockchain = dockchainContract.at(CONTRACT_ADDRESS);
5:        return { dockchain: function() {return dockchain;} };
6:    }
```

Σχήμα 4.6: Angular service για την επικοινωνία με το blockchain.

Στα Σχήματα 4.7, 4.8 παρουσιάζουμε την υλοποίηση του controller που χρησιμοποιεί το view (Σχήμα 4.5, 4.9) που βαρύνεται με τη διαχείριση των εγγράφων. Αποτελείται από μόλις τέσσερις μεθόδους,

getDocs για την ανάκτηση των αναγνωριστικών των προσπελάσιμων και τροποιήσιμων εγγράφων,

getContent για την ανάκτηση του περιεχομένου του εγγράφου που επιθυμεί ο χρήστης,

remove για τη διαγραφή του επιλεγμένου από τον χρήστη εγγράφου,

save για την αποθήκευση του εγγράφου στο οποίο έχει προβεί σε αλλαγές ο χρήστης,

αλλά ας τις εξετάσουμε μία προς μία παρακάτω. Η `getDocs` αν και δεν είναι ορατή από το view, καθώς δεν έχει γίνει μέρος του `$scope` όπως οι άλλες δύο μέθοδοι, έχει αντίκτυπο σε στοιχεία τα οποία είναι συσχετισμένα με one-way data bindings[3] Κι έτσι ο χρήστης μπορεί μόνο να δει τις αλλαγές στις οποίες υπόκεινται οι μεταβλητές στις γραμμές 7–9 και 14–16, ενώ λόγω του ότι οι αναθέσεις αυτές γίνονται εντός function callbacks σε κλήσεις του third party library είναι απαραίτητη η επανεκκίνηση του digest cycle manager προκειμένου να εντοπίσει τις αλλαγές αυτές (με κλήσεις της συνάρτησης `watch(oldVal,newVal)` πάνω σε κάθε στοιχείο του scope) και να επέμβει στην παρουσίαση των αλλαγμένων στοιχείων του view αναλόγως. Έτσι έχουμε την ανάθεση μηνυμάτων για σφάλματα καθώς και το toggle του διακόπτη για την εμφάνισή τους, αλλά και την ανάθεση στις μεταβλητές `readableDocs` και `writableDocs` των πινάκων με τα αναγνωριστικά των διαθέσιμων για τον χρήστη εγγράφων της κάθε κατηγορίας ξεχωριστά. Η εμφάνισή τους στα radio button boxes του view του Σχήματος 4.5 γίνεται εξαιρετικά εύκολα με επαναληπτικό τρόπο με την χρήση της εντολής `ng-repeat`.

Με πολύ παρόμοιο τρόπο λειτουργεί κι η μέθοδος `getContent(x,y)`, στην οποία δίνεται ως είσοδος ένα αναγνωριστικό εγγράφου στο οποίο κλικάρει ο χρήστης, και στο σώμα της μεθόδου γίνεται πρώτα ανάκτηση του περιεχομένου του με το ανάλογο callback στις γραμμές 24–30, κι έπειτα ανάθεση του ανακτηθέντος περιεχομένου του στο ανάλογο πεδίο

[2]με κάθε κλήση επιστρέφει αναφορά στο ίδιο ακριβώς αντικείμενο χωρίς να δημιουργείται ποτέ καινούριο instance παρά μόνο μία φορά, κατά τη διάρκεια της αρχικοποίησης, ή ακόμα καλύτερα στην πρώτη κλήση.

[3]μεταβλητές του scope των οποίων η χρήση στο view δηλώνεται ως `{{someVar}}`, υπονοώντας έτσι ότι δεν υπόκεινται σε αλλαγές από τον χρήστη.

```
 1:  function edit ($scope,$state,blockchain) {
 2:    var dockchain = blockchain.dockchain();
 3:
 4:    function getDocs () {
 5:      dockchain.getReadableDocs(function(error,result){
 6:        if (error) {
 7:          $scope.generalErrorMessage = "* " + error;
 8:          $scope.generalError = true;
 9:        }else $scope.readableDocs = result.slice(0,-1).split(',');
10:        $scope.$apply();
11:      });
12:      dockchain.getWritableDocs(function(error,result){
13:        if (error) {
14:          $scope.generalErrorMessage = "* " + error;
15:          $scope.generalError = true;
16:        }else $scope.writableDocs = result.slice(0,-1).split(',');
17:        $scope.$apply();
18:      });
29:    };
20:
21:    $scope.getContent = function (title,editable) {
22:      $scope.selectedTitle = title;
23:      $scope.editable = editable;
24:      dockchain.getContent(title, function(error,result){
25:        if (error) {
26:          $scope.generalErrorMessage = "* " + error;
27:          $scope.generalError = true;
28:        }else $scope.textarea = result;
29:        $scope.$apply();
30:      });
31:    };
32:
34:    $scope.remove = function (title) {
35:      if (!$scope.selectedTitle) {
36:        $scope.removalErrorMessage = "* You need to select a document first!"
37:        $scope.removalError = true;
38:      }else if (!$scope.editable) {
39:        $scope.removalErrorMessage = "* Selected document cannot be removed!"
40:        $scope.removalError = true;
41:      }else{
42:        dockchain.rmvDoc ($scope.selectedTitle, function(error,result){
43:          if (error) {
44:            $scope.generalErrorMessage = "* " + error;
45:            $scope.generalError = true;
46:          }else{
47:            if ($scope.writableDocs)
48:              for(var i=0; i<$scope.writableDocs.length; i++)
49:                if ($scope.writableDocs[i] === $scope.selectedTitle)
50:                  $scope.writableDocs.splice(i,1);
51:          }
52:          $scope.$apply();
53:        });
54:      }
55:    };
```

Σχήμα 4.7: Angular edit controller για το view διαχείρισης εγγράφων (μέρος α').

(γραμμή 28) για την εμφάνισή του στην textarea με την οποία είναι διασυνδεδεμένο με two-way data binding (άλλωστε ο χρήστης μπορεί να προσθέσει και να αφαιρέσει περιεχόμενο στο κείμενο). Το δεύτερο όρισμα της μεθόδου είναι βοηθητικό κι υποδηλώνει το εάν το προς ανάκτηση έγγραφο θα είναι διαθέσιμο προς τροποποίηση. Από το σημείο επιλογής του προς εμφάνιση εγγράφου και την ανάκτησή του, εξωτερικεύεται ξανά προς το view όταν διαχειριζόμαστε το περιεχόμενο του εγγράφου μέσω της `$scope.textarea` μεταβλητής (Σχήμα 4.9, γραμμή 23).

Η λειτουργία της μεθόδου `remove` γίνεται πρόδηλη από το όνομά της. Ελέγχει στις γραμμές 35–40 ότι ο χρήστης έχει όντως επιλέξει ένα έγγραφο προς διαγραφή καθώς κι ότι στο συγκεκριμένο περιεχόμενο έχει δικαιώματα τροποποίησης, κι εφόσον αυτό ισχύει καλείται η αντίστοιχη μέθοδος `rmvDoc` του smart-contract για την απομάκρυνση του εγγράφου από το repository. Σε αντίθετη περίπτωση ενημερώνουμε τον χρήστη για το λόγο για τον οποίο δεν μπορούμε να φέρουμε εις πέρας τη διαγραφή με το ανάλογο μήνυμα που ανατίθεται στη μεταβλητή `removalErrorMessage`, ενώ εκβιάζουμε την άμεση εμφάνιση των αλλαγών με χρήση της εντολής `$scope.$apply();` στη γραμμή 52. Η τελευταία μέθοδος του controller `save` του Σχήματος 4.8 χρησιμοποιείται για να σώσει ο χρήστης τις όποιες αλλαγές στις οποίες έχει επιβάλλει το περιεχόμενο του εγγράφου. Σε αυτήν την πρώτη έκδοση απλά φορτώνει στο blockchain το νέο κείμενο πανωγράφωντας το παλαιό περιεχόμενο. Σε επόμενη έκδοση θα περνάνε μόνο οι αλλαγές του χρήστη με κλήσεις των `editInsert` και `editDelete` μεθόδων του smart-contract.

```
 1:    $scope.save = function () {
 2:      if ($scope.selectedTitle) {
 3:        if ($scope.editable) {
 4:          if ($scope.textarea) {
 5:            dockchain.updateContent($scope.selectedTitle,$scope.textarea,
                 function(error,result){
 6:               if (error) {
 7:                 $scope.generalErrorMessage = "* " + error;
 8:                 $scope.generalError = true;
 9:               }else console.log (result);
10:               $scope.$apply();
11:             });
12:          }else{
13:            $scope.documentErrorMessage = "* Insert content in text area below!";
14:            $scope.documentError = true;
15:          }
16:        }else{
17:          $scope.documentErrorMessage = "* Selected document is not writable!";
18:          $scope.documentError = true;
19:        }
20:      }else{
21:        $scope.documentErrorMessage = "* Select a document first from above!";
22:        $scope.documentError = true;
23:      }
24:    };
25:  };
```

Σχήμα 4.8: Angular edit controller για το view διαχείρισης εγγράφων (μέρος β').

Στα Σχήματα 4.10 και 4.11 παρουσιάζουμε τον controller που αντιστοιχεί στο view του Σχήματος 4.3 για τον owner του smart-contract. Αποτελείται από τις εξής τέσσερις μεθόδους:

getUsers για την ανάκτηση της λίστας των διευθύνσεων των χρηστών της ειδικής ομάδας του smart-contract,

addUser για την προσθήκη ενός νέου χρήστη στην ειδική ομάδα,

delUser για τη διαγραφή κάποιου χρήστη από την ειδική ομάδα,

upload για την ανάρτηση νέων εγγράφων και περιεχομένου στο αποθετήριο του contract.

Η πρώτη μέθοδος δεν κάνει τίποτα παραπάνω από ένα remote call στην ομώνυμη μέθοδο του contract `getUsers` για να προωθήσει το αποτέλεσμά της στο view μέσω της μεταβλητής του scope `users` (γραμμή 9), ή να διαχειριστεί εντός του callback (γραμμές 6–8) το σφάλμα ανάλογα, με το να ενημερώνει τον χρήστη με ένα ενδεικτικό μήνυμα λάθους. Ομοίως πράττει κι η μέθοδος `addUser`, αφού ελέγξει πρώτα ότι ο χρήστης έχει όντως συμπληρώσει κάποια είσοδο (γραμμή 15). Διαφορετικά, σεττάρει την ανάλογη μεταβλητή με το σχετικό μήνυμα λάθους και του την εμφανίζει προκειμένου να συμπληρώσει το πεδίο της φόρμας (γραμμές 23–26). Παρόμοια κι η λειτουργία της `delUser` με τον έλεγχο στη γραμμή 30 ότι ο χρήστης έχει ήδη κλικάρει σε τουλάχιστον ένα εκ των radio buttons για να επιλέξει τον χρήστη που θα διαγραφεί από την ειδική ομάδα.

```
 1:  <span class="error" ng-show="generalError">{{generalErrorMessage}}</span><br/>
 2:  <div style="display:block">
 3:    <div style="float:left; margin: 0 15px 0 0;
            padding: 0 15px 15px 15px; border:1px solid gray;">
 4:      <h3>Readable</h3>
 5:      <label ng-repeat="title in readableDocs">
 6:      <input type="radio" name="radio" ng-value="title"
            ng-click="getContent(title,false)">{{title}}<br/>
 7:      </label>
 8:    </div>
 9:    <div style="float:left; margin: 0 0 0 15px;
            padding: 0 15px 15px 15px; border:1px solid gray;">
10:      <h3>Writable</h3>
11:      <label ng-repeat="title in writableDocs">
12:      <input type="radio" name="radio" ng-value="title"
            ng-click="getContent(title,true)">{{title}}<br/>
13:      </label>
14:    </div>
15:  </div>
16:  <div style="clear:both;display:block"><br/>
17:    <input type="button" value="Save" ng-click="save()">
18:    <input type="button" value="Go to admin page" ng-click="gotoAdminPage()"/>
19:    <span class="error" ng-show="documentError">{{documentErrorMessage}}</span>
20:  </div>
21:  <div style="clear:both;display:block"><br/>
22:    <h3>Content</h3>
23:    <textarea rows="10" cols="80" ng-model="textarea" ng-readonly="!editable"/>
24:  </div>
```

Σχήμα 4.9: HTML DOM για το view διαχείρισης εγγράφων.

Τέλος, στο Σχήμα 4.11 δείχνουμε την τελευταία μέθοδο του admin controller για την ανάρτηση νέου περιεχομένου στο repository. Δεδομένου ότι όντως ο χρήστης έχει συμπληρώσει τον τίτλο του εγγράφου που θα εξυπηρετήσει ως αναγνωριστικό (γραμμή 2), καθώς κι επιλέξει ήδη ένα αρχείο από το σκληρό του δίσκο ως περιεχόμενο προς ανάρτηση (γραμμή 3) θα πρέπει να υπολογίσουμε στις γραμμές 6–7 το τριψήφιο νούμερο που αντιστοιχεί στα mode bits που επιθυμεί ο διαχειριστής, δεδομένου της εισόδου του χρήστη από τα check-boxes του Σχήματος 4.3. Αν τουλάχιστον ένα check-box έχει επιλεγεί, μπορούμε τότε να προχωρήσουμε μετά τον έλεγχο της γραμμής 8 με την ανάγνωση του αρχείου από τον σκληρό δίσκο (γραμμές 9–21) και στη συνέχεια να γίνει η κλήση της `addDoc` μεθόδου του contract για την ανάρτηση του εγγράφου στο blockchain και τη διαχείριση του αποτελέσματος της απομακρυσμένης κλήσης στις γραμμές 13–19 εντός του callback.

```
 1:  function admin ($scope,$state,blockchain) {
 2:    var dockchain = blockchain.dockchain();
 3:
 4:    function getUsers () {
 5:      dockchain.getUsers(function(error,result){
 6:        if (error){
 7:            $scope.generalErrorMessage = "* " + error;
 8:            $scope.generalError = true;
 9:        }else $scope.users = result;
10:        $scope.$apply();
11:      });
12:    };
13:
14:    $scope.addUser = function () {
15:      if ($scope.newUser) {
16:        dockchain.addUser($scope.newUser, function(error,result){
17:          if (error){
18:            $scope.addUserErrorMessage = "* " + error;
19:            $scope.addUserError = true;
20:            $scope.$apply();
21:          }else getUsers();
22:        });
23:      }else{
24:        $scope.addUserErrorMessage = "* Specify a user address!";
25:        $scope.addUserError = true;
26:      }
27:    };
28:
29:    $scope.delUser = function () {
30:      if ($scope.selectedUser) {
31:        dockchain.delUser($scope.selectedUser, function(error,result){
32:          if (error){
33:            $scope.delUserErrorMessage = "* " + error;
34:            $scope.delUserError = true;
35:            $scope.$apply();
36:          }else getUsers();
37:        });
38:      }else{
39:        $scope.delUserErrorMessage = "* No user has been specified to be removed!";
40:        $scope.delUserError = true;
41:      }
42:    };
```

Σχήμα 4.10: Angular controller για τις λειτουργίες του διαχειριστή (μέρος α').

```
 1:    $scope.upload = function () {
 2:      if ($scope.newTitle) {
 3:        if ($scope.newFile) {
 4:            var rights = 0;
 5:            if($scope.owner.read)rights+=400;if($scope.owner.write)rights+=200;
 6:            if($scope.group.read)rights+=40;if($scope.group.write)rights+=20;
 7:            if($scope.others.read)rights+=4;if($scope.others.write)rights+=2;
 8:            if (rights > 0) {
 9:              var file = $scope.newFile.files[0];
10:              var fileReader = new FileReader();
11:              fileReader.onload = function(fileLoadedEvent){
12:                var content = fileLoadedEvent.target.result;
13:                dockchain.addDoc ($scope.newTitle,content,rights,
                     function(error,result){
14:                    if (error){
15:                      $scope.noTitleErrorMessage = "* " + error;
16:                      $scope.noTitleError = true;
17:                      $scope.$apply();
18:                    }else $state.goto('edit');
19:                  });
20:              };
21:              fileReader.readAsText(file,'UTF-8');
22:            }else{
23:              $scope.noRightsErrorMessage = "* No rights have been specified!";
24:              $scope.noRightsError = true;
25:            }
26:          }else{
27:            $scope.noFileErrorMessage = "* No file has been attached!";
28:            $scope.noFileError = true;
29:          }
30:        }else{
31:          $scope.noTitleErrorMessage = "* Specify a title!";
32:          $scope.noTitleError = true;
33:        }
34:    };
35:  }
```

Σχήμα 4.11: Angular controller για τις λειτουργίες του διαχειριστή (μέρος β').

4.3 On-line ενημερώσεις αποθετηρίου

Τα Ethereum events επιτρέπουν την χρήση των logging δυνατοτήτων του EVM και την αξιοποίησή τους προκειμένου να πυροδοτήσουν JavaScript callbacks σε ένα DApp που παρακολουθεί τέτοιου είδους events. Τα events ενός contract μπορούν επίσης να κληρονομηθούν από τις υπερκλάσεις τους contracts. Με τη δημιουργία ενός event κατά την εκτέλεση μίας μεθόδου αποθηκεύονται στα logs της συναλλαγής τα ορίσματα της μεθόδου κατασκευής του και θα παραμείνουν στο block για όσο αυτό είναι συνδεδεμένο στο blockchain. Τα events είναι αναγκαία για την live ενημέρωση των DApps και την επικαιρωποίηση των views βάσει των αλλαγών που συμβαίνουν στο blockchain. Για την περίπτωσή μας, επιθυμούμε όταν ο owner του contract είτε προσθέσει, είτε αφαιρέσει έναν χρήστη από την admin σελίδα, τα radio button boxes να ανανεωθούν αυτόματα προσθέτοντας ή αφαιρώντας από τη λίστα τον αντίστοιχο χρήστη ανάλογα. Για να το πετύχουμε αυτό, θα πρέπει αρχικά να δηλώσουμε τους τύπους των events εντός του dockchain smart-contract κι έξω από το σώμα οποιασδήποτε συνάρτησης ως,

```
event addUserEvent (address user);
event delUserEvent (address user);
```

ενώ εντός του σώματος των μεθόδων `addUser` και `delUser`, αμέσως μετά της επιτυχούς εισαγωγής ή διαγραφής του χρήστη αντίστοιχα, θα πρέπει να γίνει η εκπομπή του ανάλογου event ως,

```
emit addUserEvent (newUser); // inside addUser(), upon success.
emit delUserEvent (oldUser); // inside delUser(), upon success.
```

Επιπλέον, θα πρέπει να επεκτείνουμε τη λειτουργικότητα του admin controller με τον τρόπο που δείχνουμε στο Σχήμα 4.12 προκειμένου να είναι σε θέση να ανταποκρίνεται σε αυτήν την αλλαγή της κατάστασης του μοντέλου, και να προωθεί έπειτα την αλλαγή στην γραφική απεικόνιση των δεδομένων στο view. Οπότε, στις γραμμές 1–13 έχουμε τη διαχείριση των events από τις προσθήκες χρηστών, ενώ στις γραμμές 15–31 τη διαχείριση των διαγραφών. Αμφότερες δηλώνουν την παρακολούθηση των συγκεκριμένου τύπου events (γραμμές 1, 2, 15, 16) ενώ εντός του callback στη γραμμή 9 προσθέτουμε τον νέο χρήστη και με την εντολή `$scope.$apply();` της γραμμής 11 προκαλούμε την ανανέωση του digest cycle της εφαρμογής για το "φρεσκάρισμα" του box. Ομοίως, ο απομακρυσμένος χρήστης αφαιρείται από την αντίστοιχη λίστα του scope στις γραμμές 23–27, για να ανανεωθεί το view με την εντολή της γραμμής 29.

Με παρόμοιο τρόπο θα διαχειριστούμε τα αναγνωριστικά των νέων ή διαγραμμένων εγγράφων, αφού δηλώσουμε προηγουμένως τα events των κατώθι τύπων:

```
event rmvDocEvent (string title);
event addDocEvent (string title, uint16 flag);
```

ενώ εντός του σώματος των μεθόδων `rmvDoc` και `addDoc`, θα πρέπει να γίνει η εκπομπή των ανάλογων events ως,

```
emit rmvDocEvent (docId);       // inside rmvDoc(), upon success.
emit addDocEvent (docId,flag);  // inside addDoc(), upon success.
```

Εν συνεχεία, θα πρέπει να προσθέσουμε στον edit controller επιπλέον λογική προκειμένου να επιτευχθεί η διαχείριση των αντιστοίχων events. Πιο συγκεκριμένα, επιθυμούμε καθώς προσαφθαιρούνται δυναμικά έγγραφα στο repository του blockchain, να αντικατοπτρίζονται οι αλλαγές αυτές στις λίστες που βλέπουν οι χρήστες, σύμφωνα βέβαια πάντα με τα δικαιώματα χρήσης που έχει έκαστος. Αυτό το ρόλο παίζουν οι event handlers στα Σχήματα 4.13 και 4.15. Η διαδικασία του Σχήματος 4.13 είναι απλή και με τη σύλληψη του event διαγραφής διασχίζει αμφότερες λίστες αναγνωριστικών εγγράφων (γραμμές 8–24) κι όταν εντοπίσει την παρουσία του διεγραμμένου εγγράφου τότε το αφαιρεί κι εκβιάζει την επανεμφάνιση των ανανεωμένων λιστών με την εντολή της γραμμής 30. Ως επιπλέον βήμα ελέγχουμε και το εάν το απομακρυσμένο έγγραφο παρουσιάζεται ήδη στο textarea του view, κι αν αυτό είναι

```
 1:      var addUserEvent = dockchain.addUserEvent();
 2:      addUserEvent.watch (function (error,result) {
 3:       if (error) {
 4:         $scope.generalErrorMessage = "* " + error;
 5:         $scope.generalError = true;
 6:       }else{
 7:         $scope.generalError = false;
 8:          if ($scope.users) {
 9:            $scope.users.push(result.args.user);
10:          }else getUsers();
11:          $scope.$apply();
12:        }
13:      });
14:
15:      var delUserEvent = dockchain.delUserEvent();
16:      delUserEvent.watch (function (error,result) {
17:        if (error) {
18:          $scope.generalErrorMessage = "* " + error;
19:          $scope.generalError = true;
20:        }else{
21:          if ($scope.users) {
22:            $scope.generalError = false;
23:            for(var i=0; i<$scope.users.length; i++)
24:              if ($scope.users[i] === result.args.user)
25:                $scope.users.splice(i,1);
26:          }else getUsers();
27:          $scope.$apply();
28:        }
29:      });
```

Σχήμα 4.12: Διαχείριση blockchain events εντός του admin controller.

```
 1:  var rmvDocEvent = dockchain.rmvDocEvent();
 2:  rmvDocEvent.watch (function (error,result) {
 3:    if (error) {
 4:      $scope.generalErrorMessage = "* " + error;
 5:      $scope.generalError = true;
 6:    }else{
 7:      var runGetDocs = false;
 8:      if ($scope.readableDocs) {
 9:        for(var i=0; i<$scope.readableDocs.length; i++){
10:          if ($scope.readableDocs[i] === result.args.title) {
11:            $scope.readableDocs.splice(i,1);
12:            $scope.generalError = false;
13:          }
14:        }
15:      }else runGetDocs = true;
16;
17:      if ($scope.writableDocs) {
18:        for(var i=0; i<$scope.writableDocs.length; i++){
19:          if ($scope.writableDocs[i] === result.args.title) {
20:            $scope.writableDocs.splice(i,1);
21:            $scope.generalError = false;
22:          }
23:        }
24:      }else runGetDocs = true;
25:
26:      if ($scope.selectedTitle && $scope.selectedTitle === result.args.title) {
27:        $scope.textarea = " -==- The document has been deleted by another user! -==-";
28:      }
29:      if (runGetDocs) getDocs();
30:      else $scope.$apply();
31:    }
32:  });
```

Σχήμα 4.13: Διαχείριση events διαγραφής εγγράφων εντός του edit controller.

αληθές, τότε αντικαθιστούμε στην οθόνη του χρήστη το περιεχόμενο του εγγράφου με ένα κατατοπιστικό κείμενο ότι το έγγραφο αφαιρέθηκε από κάποιον άλλον χρήστη κι ότι δεν είναι πια σε θέση να δουλέψει σε αυτό.

Ομοίως, στο Σχήμα 4.15 παρουσιάζουμε τον event handler για τη διαχείριση events τύπου `addDocEvent`. Η διαχείριση αυτού του τύπου εμπεριέχει σαφώς περισσότερη λογική από προηγουμένως κι αυτό πηγάζει από το γεγονός ότι θα πρέπει να εξεταστεί κατά πόσο ο χρήστης έχει πρόσβαση στο νέο έγγραφο (γραμμές 8–11, 20–26 και 35–51) κι εφόσον αυτό ισχύει προστίθεται το αναγνωριστικό του εγγράφου στη μεταβλητή που γίνεται προσβάσιμη από το view. Τέλος, η γραμμή 53 πυροδοτεί την εμφάνιση των όποιων αλλαγών στην οθόνη του χρήστη. Σημειωτέον βέβαια ότι ακόμα κι αν για κάποιο λόγο κάποιος κακόβουλος χρήστης υποκλέψει το event προσθήκης εγγράφου και το διαχειριστεί έστω με δικό του τρόπο χωρίς να έχει δικαιώματα είτε προσπέλασης, είτε τροποποίησης, δεν θα μπορεί να συνεχίσει με το να προβάλει στην οθόνη του το περιεχόμενο ή να το τροποιήσει αφού οι μέθοδοι του smart-contract συμπεριλαμβάνουν τη λογική που θα του απαγορέψει να προβεί σε οποιασδήποτε περαιτέρω κίνηση.

```
 1:  var updateContentEvent = dockchain.updateContentEvent();
 2:  updateContentEvent.watch (function (error,result) {
 3:    if (error) {
 4:      $scope.generalErrorMessage = "* " + error;
 5:      $scope.generalError = true;
 6:    }else{
 7:      if ($scope.editable === false && $scope.selectedTitle
             && $scope.selectedTitle === result.args.title) {
 8:        $scope.textarea = result.args.content;
 9:        $scope.generalError = false;
10:      }
11:    }
12:  });
```

Σχήμα 4.14: Διαχείριση events μεταβολής περιεχομένου εντός του edit controller.

Για το υπόλοιπο της ενότητας θα μιλήσουμε για τους μηχανισμούς ανανέωσης του περιεχομένου του εγγράφου στο view. Η πρώτη υλοποίηση που περιγράφουμε στο παρόν αποτελεί μία απλουστευτική έκδοση, η οποία σύντομα θα μετεξελιχθεί να καλύπτει την πλήρη λειτουργικότητα που μπορεί να υποστηρίξει το smart-contract ανανεώνοντας έτσι live το view με κάθε κλήση της `editInsert` και `editDelete`. Έτσι λοιπόν, επιτρέπουμε στον χρήστη να πατήσει στο κουμπί `Save` προκειμένου να αποθηκεύσει τις μεταβολές που έχει προκαλέσει στο περιεχόμενο των εγγράφων στα οποία διαθέτει δικαιώματα τροποποίησης. Έτσι έχουμε τη δήλωση του event όπως δείχνουμε παρακάτω,

```
event updateContentEvent (string title, string content);
```

ενώ στις αντίστοιχες μεθόδους εντός του smart-contract όπου γίνεται η μεταβολή του περιεχομένου του εγγράφου στη mapping δομή `contents[docId]` έχουμε την εκπομπή του event ως,

```
emit updateContentEvent (docId,newContent);
```

Η διαχείριση του συγκεκριμένου τύπου event γίνεται από τον edit controller με τον handler του Σχήματος 4.14 κι ελέγχει στις γραμμές 7–11 ότι το επιλεγμένο έγγραφο είναι αυτό το οποίο έχει υποστεί τη μεταβολή στο περιεχόμενό του, κι οπότε ανανεώνει το textarea του view σύμφωνα με το νέο περιεχόμενο.

```
 1:  var addDocEvent = dockchain.addDocEvent();
 2:  addDocEvent.watch (function (error,result) {
 3:    if (error) {
 4:      $scope.generalErrorMessage = "* " + error;
 5:      $scope.generalError = true;
 6:    }else{
 7:      var title = result.args.title;
 8:      var flag = result.args.flag;
 9:      var ownerRight = (flag%1000)/100;
10:      var groupRight = (flag%100)/10;
11:      var restRight = flag%10;
12:      dockchain.isOwner (function (error,result) {
13:        if (error) {
14:          $scope.generalErrorMessage = "* " + error;
15:          $scope.generalError = true;
16:          $scope.$apply();
17:        }else{
18:          if (result === true) {
19:            $scope.generalError = false;
20:            if (ownerRight == 6) {
21:              $scope.readableDocs.push (title);
22:              $scope.writableDocs.push (title);
23:            }else if (ownerRight == 4) {
24:              $scope.readableDocs.push (title);
25:            }else if (ownerRight == 2)
26:              $scope.writableDocs.push (title);
27:            $scope.$apply();
28:          }else{
29:            dockchain.isInGroup (function (error,result) {
30:              if (error) {
31:                $scope.generalErrorMessage = "* " + error;
32:                $scope.generalError = true;
33:              }else{
34:                $scope.generalError = false;
35:                if (result === true) {
36:                  if (groupRight == 6) {
37:                    $scope.readableDocs.push (title);
38:                    $scope.writableDocs.push (title);
39:                  }else if (groupRight == 4) {
40:                    $scope.readableDocs.push (title);
41:                  }else if (groupRight == 2)
42:                    $scope.writableDocs.push (title);
43:                }else{
44:                  if (restRight == 6) {
45:                    $scope.readableDocs.push (title);
46:                    $scope.writableDocs.push (title);
47:                  }else if (restRight == 4) {
48:                    $scope.readableDocs.push (title);
49:                  }else if (restRight == 2)
50:                    $scope.writableDocs.push (title);
51:                }
52:              }
53:              $scope.$apply();
54:            });
55:          }
56:        }
57:      });
58:    }
59:  });
```

Σχήμα 4.15: Διαχείριση events προσθήκης εγγράφων εντός του edit controller.

4.4 Άσκηση επι χάρτου

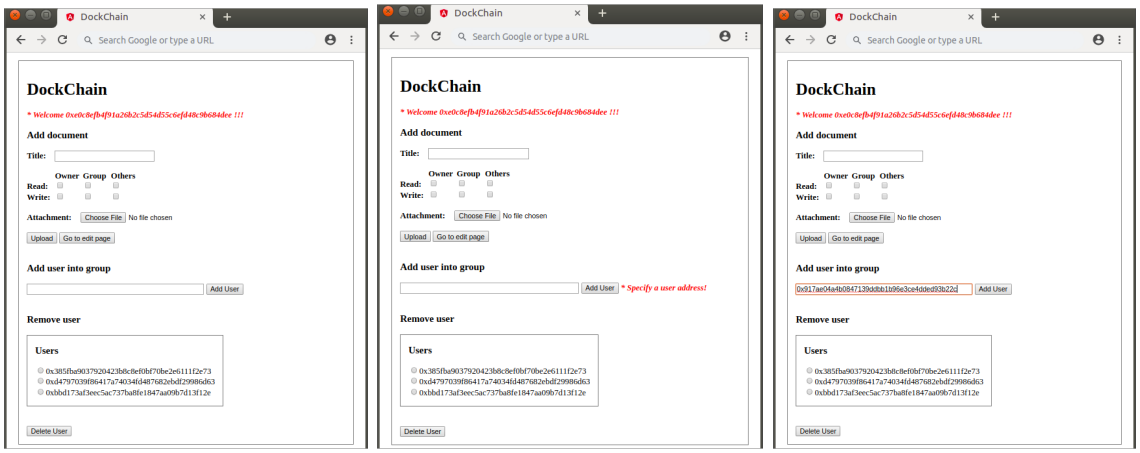
Έχοντας συνοψίσει τη λειτουργικότητα της εφαρμογής όσον αφορά το smart-contract που αφορά το blockchain, αλλά και το client-side κομμάτι που τρέχει στον browser του χρήστη, θα δώσουμε στην ενότητα αυτή ένα πραγματικό σενάριο λειτουργίας που εμπεριέχει τα workflows που υποστηρίζει η εφαρμογή. Πιο συγκεκριμένα, ξεκινάμε (i) εισάγωντας έναν νεό χρήστη στην ειδική ομάδα του smart-contract, (ii) έπειτα θα τον διαγράψουμε ώστε να εμφανιστούν τα ειδικά μηνύματα επιτυχίας και σφαλμάτων προς το χρήστη όταν συμβαίνει κάποια αστοχία στην αλυσίδα των απομακρυσμένων κόμβων που αποτελούν το σύστημα ή αποπειράται να προβεί σε κάποια μη επιτρεπτή ενέργεια, (iii) έπειτα θα αναρτήσουμε ένα νέο έγγραφο στο αποθετήριο του smart-contract συμπληρώνοντας τα στοιχεία ένα προς ένα με κάθε δοκιμή σύμφωνα με τα μηνύματα του συστήματος, (iv) κι ύστερα της επιτυχούς ανάρτησης του εγγράφου και της αυτόματης παραπομπής μας στη σελίδα διαχείρισης των εγγράφων προβάλλουμε το έγγραφο σε mode ανάγνωσης και δοκιμάζουμε να το σώσουμε και να το διαγράψουμε, κάτι που δεν επιτρέπεται με αυτή τη ρύθμιση, (v) σε write mode όμως αυτό καθίσταται δυνατόν, όπως θα δούμε στη συνέχεια. Επιπλέον, (vi) τροποποιούμε το περιεχόμενο του εγγράφου, (vii) αποθηκεύουμε τις αλλαγές, και τέλος, (iix) το διαγράφουμε από το αποθετήριο του smart-contract.

Στο Σχήμα 4.16 δείχνουμε την πρώτη οθόνη που βλέπει ο διαχειριστής του smart-contract από το admin view που περιγράψαμε σε προηγούμενη ενότητα. Υπάρχει ένα μήνυμα καλωσορίσματος που ενημερώνει τον χρήστη για τη blockchain διεύθυνση που χρησιμοποιεί. Εν αντιθέσει, εάν άλλος χρήστης έβλεπε το ίδιο view, τότε αντί για μήνυμα καλωσορίσματος θα έβλεπε το μήνυμα *"* You are not allowed to use this page!"*.

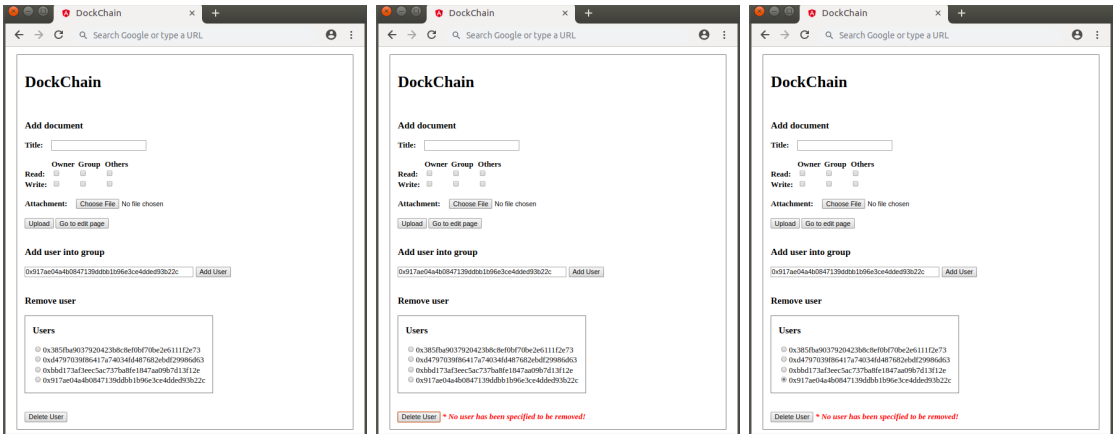
Στη συνέχεια, πατάμε το κουμπί με την επισήμανση "Add user" χωρίς όμως να έχουμε προσθέσει πιο πριν στο διπλανό textbox κάποια διεύθυνση. Οπότε, η γραφική διεπαφή αποκρίνεται κατάλληλα κι ενημερώνει τον χρήστη σχετικά με την αμέλειά του με το σχετικό μήνυμα *"* Specify a user-address"* ώστε να πράξει ανάλογα. Συμμορφούμενοι λοιπόν σε αυτήν την υπόδειξη, συμπληρώνουμε το textbox με την ακόλουθη διεύθυνση blockchain χρήστη `0x917ae04a4b0847139ddbb1b96e3ce4dded93b22c` για προσθήκη και ξαναδοκιμάζουμε. Αυτόματα βλέπουμε τη λίστα με τις διευθύνσεις των χρηστών της ειδικής ομάδας του smart-contract να αυξάνεται κατά ένα στοιχείο, την ανωτέρω διεύθυνση, μετά την ενημέρωση του αντίστοιχου στοιχείου ύστερα από τη σύλληψη του ειδικού event τύπου `addUserEvent`.

Στο Σχήμα 4.17 διαγράφουμε τον ίδιο χρήστη που μόλις προσθέσαμε. Αρχικά πυροδοτούμε τη σχετική διαδικασία χωρίς όμως να έχουμε επιλέξει εκ των προτέρων κάποιον χρήστη από τη λίστα. Προφανώς το σύστημα μας σταματάει και αποκρίνεται με το σχετικό μήνυμα *"* No user has been specified to be removed!"*. Έτσι επιλέγουμε το τελευταίο στοιχείο της λίστας που αντιστοιχεί στον χρήστη που μόλις προσθέσαμε και πατάμε το κουμπί της διαγραφής με την επισήμανση `Delete User`. Το αποτέλεσμά είναι άμεσο και φαίνεται στο Σχήμα 4.18 όπου εξαφανίζεται ο διαγραφέντας χρήστης ύστερα από τη σύλληψη του ειδικού event τύπου `delUserEvent`. Μετά τη διαγραφή του χρήστη ξεκινάμε τη διαδικασία εισαγωγής αρχείου. Αυτή δεν προχωράει αν δεν εισάγουμε πρώτα έναν τίτλο εν ήδη αναγνωριστικού, επιλέξουμε ένα αρχείο από τον σκληρό δίσκο του οποίου το περιεχόμενο θα αναρτηθεί, καθώς και τα δικαιώματα χρήσης για τις τρεις διαθέσιμες κατηγορίες χρηστών. Αν έστω κι ένα από αυτά τα στοιχεία λοίπον από την είσοδο πριν πατηθεί το κουμπί `Upload`, εμφανίζεται το ανάλογο μήνυμα, *"* Specify a title!"*, *"* No file has been attached!"*, και *"* No rights have been specified!"* αντίστοιχα, δίπλα από το αντίστοιχο στοιχείο της γραφικής διεπαφής, textbox, file input, και checkboxes, αντίστοιχα. Έτσι ο χρήστης θα γνωρίζει πως να αντιδράσει και τι να συμπληρώσει. Εφόσον τώρα η είσοδος είναι πλήρης, όπως δείχνουμε στην Εικόνα 4.19, παραπεμπόμαστε στο view edit του Σχήματος 4.20.

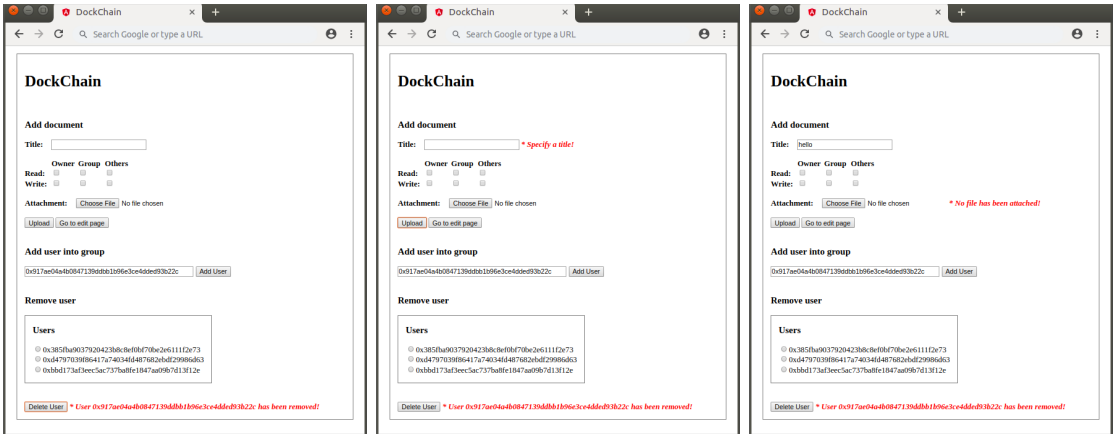
Στο view edit διακρίνουμε δύο ξεχωριστές λίστες με radio boxes, στο αριστερό έχουμε τα έγγραφα στα οποία ο χρήστης έχει δικαιώματα ανάγνωσης, ενώ στο δεξί δικαιώματα τροποποίησης. Επιλέγοντας ένα έγγραφο από οποιαδήποτε λίστα, το περιεχόμενό του εμφανίζεται στην textarea από κάτω. Όπως δείχνουμε και στην Εικόνα 4.21, έχουμε τοποθετήσει δύο κουμπιά, `Save` και `Remove`, των οποίων η λειτουργία πάνω στα έγγραφα είναι η προφανής. Όμως για να επενεργήσουν θα πρέπει πρώτα να επιλεχθεί κάποιο έγγραφο, διαφορετικά ο χρή-



Σχήμα 4.16: Αρχική οθόνη για τον διαχειριστή του smart-contract.



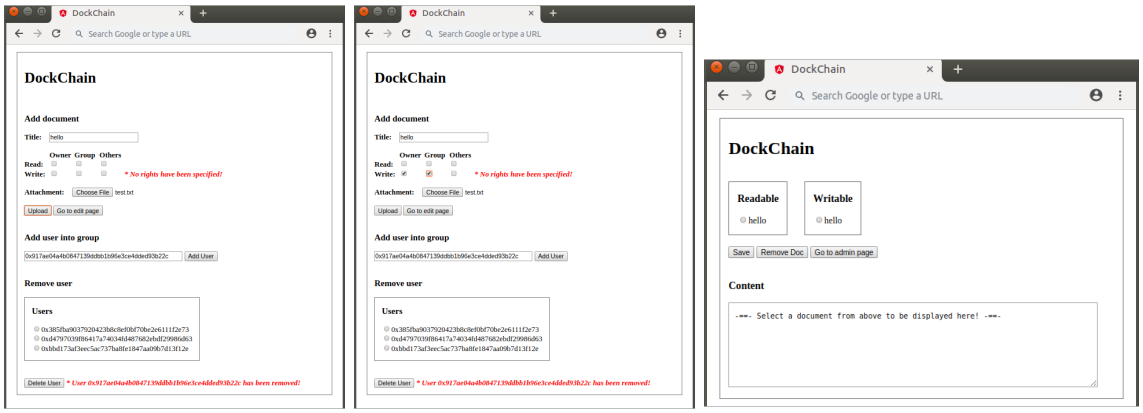
Σχήμα 4.17: Εισαγωγή χρήστη στην ειδική ομάδα του smart-contract.



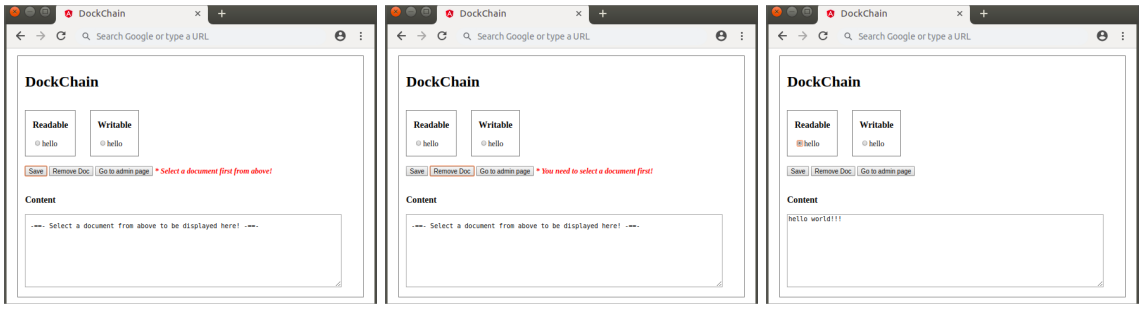
Σχήμα 4.18: Διαγραφή χρήστη από την ειδική ομάδα του smart-contract.

στης θα ενημερωθεί ώστε να επιλέξει κάποιο έγγραφο με το μήνυμα *"* You need to select a document first!"*. Αλλά ακόμα κι όταν γίνεται επιλογή κάποιου εγγράφου, σε αυτό θα πρέπει ο χρήστης να έχει δικαιώματα τροποποίησης. Διαφορετικά, ανάλογα με τη λειτουργία στην οποία επιλέγει να προβεί εμφανίζεται είτε το μήνυμα *"* Selected document is not writable!"*, είτε το *"* Selected document cannot be removed!"*.

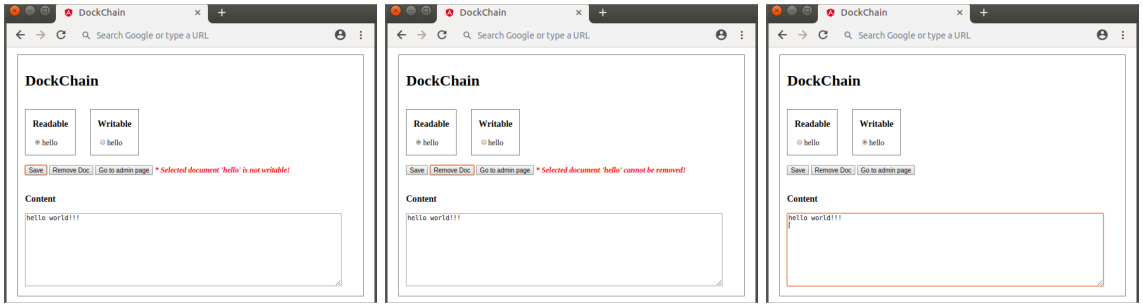
Έχοντας επιλέξει ένα έγγραφο σε mode τροποποίησης στο Σχήμα 4.22, βλέπουμε τον κέρσορα να περμένει στην textarea για είσοδο από τον χρήστη. Όπως δείχνουμε στα τελευταία βήματα του σεναρίου, τροποποιούμε το περιεχόμενο ελαφρά, κι έπειτα, πατώντας το κουμπί `Save` ολοκληρώνεται η διαδικασία μεταβολής κι αποθήκευσης του εγγράφου με ένα μήνυμα επιβεβαίωσης *"* Submitted changes!"*. Για το τέλος αφήσαμε τη διαγραφή του εγγράφου. Πατώντας το κουμπί *Remove Doc* απομακρύνεται το επιλεγμένο έγγραφο από το αποθετήριο του smart-contract, και με τη σύλληψη του αντίστοιχου event `rmvDocEvent` αφαιρούμε το αναγνωριστικό του εγγράφου από τις στήλες που επιτρέπουν την επιλογή του για ανάγνωση ή τροποποίηση.



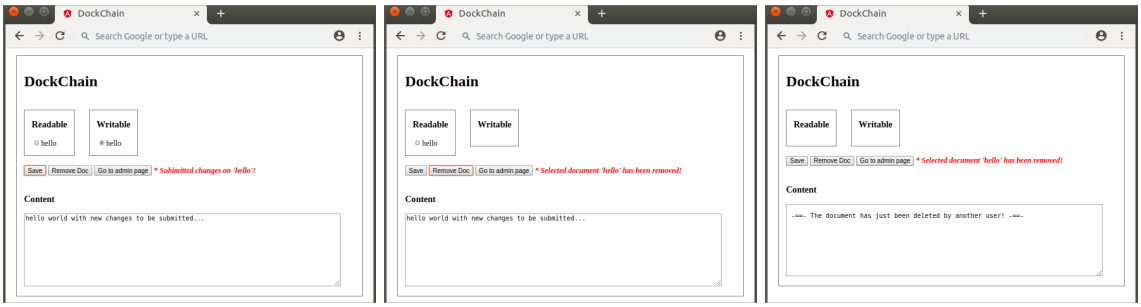
Σχήμα 4.19: Εισαγωγή εγγράφου στο αποθετήριο του smart-contract.



Σχήμα 4.20: Ανάγνωση περιεχομένου εγγράφου.



Σχήμα 4.21: Λειτουργίες μεταβολής εγγράφου.



Σχήμα 4.22: Τροποποίηση, αποθήκευση και διαγραφή περιεχομένου εγγράφου.

Κεφάλαιο 5

Επίλογος

A conclusion is the place where you get tired of thinking.

Arthur Bloch (1948–)

5.1 Σύνοψη και συμπεράσματα

Στην εργασία αυτή μελετήσαμε το blockchain μοντέλο μέσα από συγκεκριμένα ενδεικτικά και δημοφιλή παραδείγματα όπως το BitCoin, ενώ στη συνέχεια επικεντρωθήκαμε στο Ethereum, την αρχιτεκτονική του, το προγραμματιστικό του μοντέλο, καθώς και τις διαδικασίες ανάπτυξης λογισμικού γύρω από αυτό. Δεδομένου ότι ο σκοπός της εργασίας αυτής ήταν η end-to-end ανάπτυξη μίας εφαρμογής για τη διαχειριση εγγράφων πάνω στο blockchain, αφιερώσαμε μία πλήρη ενότητα για την ανάλυση των προδιαγραφών και των υπόλοιπων απαραίτητων στοιχείων που θα απαρτίζουν το smart-contract το οποίο θα αποτελεί το backend κομμάτι της εφαρμογής αφού γίνει deploy στο blockchain. Αναπτύξαμε δύο υλοποιήσεις του smart-contract σε διαφορετικές γλώσσες προγραμματισμού, Serpent και Solidity, για το EVM τις οποίες εξετάσαμε με διαφορετικά εργαλεία. Έπειτα κατασκευάσαμε το frontend κομμάτι σε HTML και JavaScript πάνω στην MVVM πλατφόρμα που προσφέρει το εξαιρετικά πετυχημένο και δημοφιλές AngularJS framework. Τέλος, σχεδιάσαμε μία κομψή και φιλική προς τον χρήστη διεπαφή την οποία μάλιστα χρησιμοποιούμε για να κάνουμε μία εκτεταμένη παρουσίαση ενός σεναρίου που περιλαμβάνει την πλήρη λειτουργικότητα της εφαρμογής end-to-end.

Αν και καθ' όλη τη διάρκεια σχεδιασμού κι ανάπτυξης της εφαρμογής υπήρχε πρόνοια για μία μεγάλης κλίμακας εφαρμογή πραγματικού χρόνου, στην πραγματικότητα οι παράγοντες κλιμάκωση και αποκρισιμότητα καθορίζονται σε αποκλειστικότητα από τις δυνατότητες του Ethereum blockchain που υπηρετεί ως δικτυακή υποδομή κι επιτρέπει την ενεργή συμμετοχή αυθαίρετου αριθμού χρηστών. Με το τρέχον μοντέλο που έχει υιοθετήσει το Ethereum οι συναλλαγές που διεκπαιρεώνονται ανά δευτερόλεπτο δεν ξεπερνάνε τις 25 ενώ το νούμερο αυτό σκοπεύει να απογειωθεί εντός του 2019 με την υιοθέτηση νέας τεχνολογίας για την επαλήθευση των συναλλαγών και την προσάρτησή τους στο blockchain. Σε αντιπαραβολή με τους δικούς μας σκοπούς, οι περιορισμοί της τρέχουσας τεχνολογίας δε θα αποτελούσαν εμπόδιο εφόσον η εφαρμογή δεν είχε δυναμική φύση. Κάτι τέτοιο όμως δεν ισχύει καθώς τα έγγραφα τα οποία αναρτώνται στο αποθετήριο του smart-contract μπορούν κι υπόκεινται σε αλλαγές σύμφωνα με τα δικαιώματα πρόσβασης και χρήσης που έχουν οριστεί. Συνεπώς, ο ρυθμός με τον οποίον αφομοιώνονται και κοινοποιούνται οι αλλαγές αυτές προς του χρήστες που συμμετέχουν είναι παράγοντας ζωτικής σημασίας για την εκπλήρωση των σκοπών, των κινήτρων και του οράματος του πλάνου της εφαρμογής. Ο λόγος έγκειται κυρίως γύρω από τις ενημερώσεις του περιεχομένου των εγγράφων όπου οι μεταβολλές υπόκεινται σε ενδεχόμενη αλληλουχία, έστω από διαφορετικούς χρήστες πάνω στο ίδιο έγγραφο, επηρεάζοντας με τον τρόπο αυτόν την γενικότερη εμπειρία τον χρηστών από το σύστημα, αλλά κι αποτελώντας

ταυτόχρονα την πιθανή γενεσιουργό αιτία για αναντιστοιχίες στο παράγωγο περιεχόμενο από του αναμενόμενου καθώς αυτό προέρχεται από διαφορετικές πηγές. Αντιλαμβανόμενοι τις απαιτήσεις των χρηστών σε αντιπαραβολή με την ευαισθησία της εφαρμογής και των θεμελίων στα οποία στηρίζεται, επικεντρώνουμε τις σκέψεις μας κι αφιερώνουμε τη μελλοντική μας προσπάθεια στην άμβλυνση του αντίκτυπου των φαινομένων του ταυτοχρονισμού ως προς τους πόρους και τα έγγραφα του smart-contract για τα οποία ανταγωνίζονται πολλαπλοί χρήστες, όπως θα δούμε σε πιο ουσιαστικό και τεχνικό επίπεδο παρακάτω για το υπόλοιπο αυτού του κεφαλαίου.

5.2 Μελλοντικές προεκτάσεις

Οι άξονες στους οποίους θα βασιστεί η επέκταση της εφαρμογής μας έχουν να κάνουν σε μεγάλο βαθμό με τις live ενημερώσεις των μεταβολών στο περιεχόμενο των εγγράφων. Προς το παρόν, η παρεχόμενη υπηρεσία μπορεί να ενημερώσει με επαρκή τρόπο το view των χρηστών που προβάλλουν ένα έγγραφο όταν ένας άλλος χρήστης μεταβάλλει το περιεχόμενό του. Τι συμβαίνει όμως όταν ένας αυθαίρετος αριθμός χρηστών μεταβάλλουν ταυτόχρονα το περιεχόμενο ενός συγκεκριμένου εγγράφου και περιμένουν το σύνολο των αλλαγών να μπορέσουν να συγχωνευθούν δίχως απώλια ή βλάβη; Τότε θα ήταν συνετό να αποθηκεύονταν κάθε φορά μόνον οι αλλαγές στις οποίες υποβάλλει ο χρήστης το έγγραφο, δηλαδή όχι ολόκληρο το νέο περιεχόμενο κι εν συνεχεία όλοι οι υπόλοιποι χρήστες να ενημερώνονται με το ανάλογο event για τη μεταβολή, ώστε όταν αυτοί με τη σειρά τους υποβάλλουν τις αλλαγές τους, να γίνει αυτό με σεβασμό στις προηγούμενες αλλαγές των υπολοίπων χρηστών. Λαμβάνοντας υπόψην τις live αλλαγές των events καθώς κάνει edit ο χρήστης ένα έγγραφο γίνεται εφικτή η συγχώνευση με τις δικές του αλλαγές. Ως τώρα όλα καλά εφόσον οι χρήστες κάνουν αλλαγές σε διαφορετικά σημεία του ίδιου εγγράφου. Τι γίνεται όμως όταν οι αλλαγές τους αφορούν την ίδια πρόταση, ή ακόμα και την ίδια φράση; Εκεί τα πράγματα για το ποιού χρήστη οι αλλαγές θα υπερισχύσουν γίνονται δυσδιάκριτα και θα πρέπει να επανεξετάσουν το αποτέλεσμα οι ίδιοι οι χρήστες που θα πρέπει να επιληφθούν της συγχώνευσης.

Η εναλλακτική λύση στο πρόβλημα αυτό θα ήταν όταν ο χρήστης υποβάλλει ένα έγγραφο σε αλλαγές, να μην αφήνουμε τους υπόλοιπους χρήστες να αγγίζουν τα συμφραζόμενα των αλλαγών μέχρι αυτές να γίνουν commit, ή έστω ο χρήστης να κλείσει τη συνεδρία του. Ένα είδος κλειδώματος δηλαδή αλλά όχι για ολόκληρο το έγγραφο, μόνο για τα μέρη τα οποία υπόκεινται σε αλλαγές. Για να το επιτύχουμε αυτό θα πρέπει να παρακολουθούμε εντός του edit controller για events τα οποία προέρχονται από τον χρήστη αυτήν τη φορά αντί του blockchain, με τον τρόπο που δείχνουμε παρακάτω:

```
var events = [];
document.getElementById('textarea').addEventListener('keydown',
  function (event) { events.push (event); });
```

και με το πάτημα του `Save` να επεξεργάζεται άλλη μέθοδος την είσοδο του χρήστη προκειμένου να την ομαδοποιηθεί σε insertions και deletions προκειμένου να ενημερώνεται το state του contract στο blockchain κατάλληλα.

Παράρτημα α'
Η δομή του Ethereum

.1 Ένα οικοσύστημα χρηστών και smart-contracts

Ο κόσμος του Ethereum απαρτίζεται κατά κύριο λόγο από 160-μπιτες διευθύνσεις που συσχετίζονται με τους λογαριασμούς των χρηστών μέσω της RLP δομής. Κάθε λογαριασμός αποτελείται από τα εξής πεδία:

nonce ο αριθμός των συναλλαγών που έχουν ξεκινήσει από τη συγκεκριμένη διεύθυνση, ή ο αριθμός των smart-contracts που έχει δημιουργήσει,

balance η ποσότητα κρυπτονομίσματος που διαχειρίζεται η συγκεκριμένη διεύθυνση,

storageRoot το Keccak 256-bit hash που αντιστοιχεί στο ιεραρχικό aggregation του Merkle Tree με το κωδικοποιημένο περιεχόμενο που αντιστοιχεί στη διεύθυνση.

codeHash Ο αμετάβλητος κωδικοποιημένος πηγαίος κώδικας στο Ethereum Virtual Machine (EVM) που εκτελείται όταν δέχεται το αντίστοιχο μήνυμα.

.2 Συναλλαγές

Η συναλλαγή είναι μία κρυπτογραφημένη εντολή που δημιουργείται από έναν εξωτερικό ρόλο, π.χ. χρήστες του blockchain, εξωτερικές εφαρμογές, κτλ. Υπάρχουν δύο τύποι συναλλαγών: (i) αυτές που καταλήγουν στη δημιουργία μηνυμάτων και κλήσεις μεθόδων, και (ii) αυτές που προκαλούν τη δημιουργία νέων διευθύνσεων σχετιζόμενων με πηγαίο κώδικα για τη δημιουργία νέων smart-contracts. Αμφότεροι τύποι εμπεριέχουν έναν αριθμό κοινών πεδίων, όπως δείχνουμε παρακάτω:

nonce ο αριθμός των συναλλαγών που έχουν προκληθεί από τον ίδιο αποστολέα,

gasPrice οι μονάδες κρυπτονομίσματος που αντιστοιχούν ανά μονάδα gas για την ολοκλήρωση της συναλλαγής κι αντιστοιχούν στα υπολογιστικά κόστη της συναλλαγής,

gasLimit η μέγιστη ποσότητα gas που θα μπορούσε να χρειαστεί για την εκτέλεση της συναλλαγής. Αυτή μεταφέρεται πριν την εκτέλεση οποιασδήποτε εντολής της συναλλαγής κι είναι αμετάβλητη,

to η 160-μπιτη διεύθυνση του παραλήπτη, είτε χρήστη, είτε ενός smart-contract. Το πεδίο αυτό μένει κενό μόνο στη δημιουργία ενός smart-contract.

value οι μονάδες κρυπτονομίσματος που αποστέλονται με την συγκεκριμένη συναλλαγή στον παραλήπτη, ή στην περίπτωση δημιουργίας ενός contract, αρχικοποιεί το balance του νέου smart-contract.

.3 Blocks

Το κάθε block στο Ethereum αποτελεί συλλογή πληροφορίας που μαζί με πληροφορία για τις συνδεόμενες συναλλαγές και ένα σετ από τις κεφαλίδες των blocks που έπονται του block που έπεται του block που προηγείται το τρέχον, γνωστά κι ως ommers, συνοψίζεται στα εξής πεδία:

parentHash το Keccak 256-bit hash ολόκληρου της κεφαλίδας του γονικού block,

ommersHash το Keccak 256-bit hash της ommer λίστας του block,

beneficiary η 160-μπιτη διεύθυνση στην οποία μεταφέρονται όλες οι ανταμοιβές από τη διαδικασία mining,

stateRoot το Keccak 256-bit hash του κόμβου στη ρίζα αφού ολοκληρωθούν όλες οι συναλλαγές,

transactionsRoot το Keccak 256-bit hash που αντιστοιχεί στο ιεραρχικό aggregation του Merkle Tree με τα hashes των συναλλαγών,

receiptsRoot το Keccak 256-bit hash που αντιστοιχεί στο ιεραρχικό aggregation του Merkle Tree με τα hashes των receipts,

logsBloom το φίλτρο Bloom αποτελούμενο απο indexable πληροφορία (π.χ. διεύθυνση logger) που εμπεριέχεται σε κάθε log εγγραφή από κάθε receipt συναλλαγής,

timestamp βαθμωτό μέγεθος που αντιστοιχεί στο epoch time που εισήχθει το block στην αλυσίδα,

difficulty βαθμωτό μέγεθος που υποδηλώνει το επίπεδο δυσκολίας του block κι υπολογίζεται βάσει της αντίστοιχης μεταβλητής του προηγούμενου block και του timestamp,

number βαθμωτό μέγεθος που αντιστοιχεί στον αριθμό των blocks που προηγούνται. Το genesis block που αρχικοποιεί το blockchain έχει την τιμή 0 για το συγκεκριμένο πεδίο,

gasLimit η ανώτατη τιμή gas που μπορεί να καταναλωθεί ανά block,

gasUsed η συνολική ποσότητα gas που έχει χρησιμοποιηθεί έως τώρα από όλα τα blocks του contract,

extraData η δυαδική αναπαράσταση των δεδομένων του block με μέγιστο επιτρεπτό μέγεθος ως 32 bytes.

nonce 64-μπιτη μεταβλητή που χρησιμοποιείται για την πιστοποίηση του block και συμπληρώνεται από τον miner κόμβο.

mixHash 256-μπιτο hash που σε συνδυασμό με το nonce αποδεικνύει την εγκυρότητα του block.

Παράρτημα β'
Solidity

Σε μεγάλο βαθμό (εξ' ολοκλήρου όσον αφορά εμάς) η γλώσσα προγραμματισμού Solidity[4, 6] βασίζεται σε ένα static strong typing μοντέλο. Αυτό σημαίνει πρακτικά ότι οι τύποι των μεταβλητών είναι ήδη διαθέσιμοι κατά τη διαδικασία της μεταγλώττισης, δεν αλλάζουν κατά τη διάρκεια της εκτέλεσης, και τέλος γίνεται ανάθεση από τιμές όμοιου τύπου ή μετά από casting.

.1 Προσβασιμότητα Μεταβλητών και Συναρτήσεων

Σχετικά με τους όρους χρήσης των μεταβλητών και των συναρτήσεων ενός smart-contract έχουμε ορισμένες διαφοροποιήσεις συγκριτικά με τις παραδοσιακές γλώσσες προγραμματισμού. Αυτές επαφίονται στο γεγονός ότι η γλώσσα αυτή έχει αναπτυχθεί αποκλειστικά για χρήση σε ένα μεγάλης κλίμακας κατανεμημένο σύστημα. Για αυτόν το λόγο τη διακατέχει σημειολογία που είναι στενά συνδεδεμένη με τη μετάδοση μηνυμάτων για απομακρυσμένες κλήσεις, μεταξύ άλλων φυσικά. Συνεπώς, διακρίνουμε εξής κύριες κατηγορίες πρόσβασης μεταβλητών και συναρτήσεων:

external αποτελούν μέρος της διεπαφής του contract, που σημαίνει στην πράξη ότι είναι δυνατή η κλήση τους από άλλες συναρτήσεις και συναλλαγές. Δεν είναι δυνατή η απευθείας εσωτερική τους κλήση κι είναι πιο αποδοτικές όσον αφορά την επεξεργασία μεγάλων πηγών δεδομένων που δέχονται από εξωτερικές πηγές.

public επίσης μέρος της διεπαφής του contract, οι οποίες όμως είναι προσβασιμες τόσο από τον έξω κόσμο, όσο κι από εσωτερικές συναρτήσεις μέσω `external` συναρτήσεων που δημιουργούνται αυτόματα με τα αντίστοιχα ονόματα.

internal για μεταβλητές και συναρτήσεις των οποίων η χρήση περιορίζεται εντός του contract κι αυτών που προέρχονται από αυτό μέσω του μηχανισμού της κληρονομικότητας.

private για μεταβλητές και συναρτήσεις των οποίων η χρήση περιορίζεται εντός του contract και δεν είναι προσβάσιμες από contracts τα οποία προέρχονται από αυτό μέσω του μηχανισμού της κληρονομικότητας.

Για παράδειγμα, έστω η παρακάτω δήλωση:

```
1:    contract complex {
2:        struct Data { uint a; bytes3 b; mapping(uint => uint) map; }
3:        mapping(uint => mapping(bool => Data[])) public data;
4:    }
```

που σημαίνει ότι θα αντιστοιχηθεί συνάρτηση πρόσβασης στη `data` δομή ως,

```
1:    function data(uint arg1,bool arg2,uint arg3)returns(uint a,bytes3 b){
2:        a=data[arg1][arg2][arg3].a;
3:        b=data[arg1][arg2][arg3].b;
4:    }
```

.2 Booleans

Το πεδίο τιμών των αντίστοιχων μεταβλητών `bool` περιορίζεται στο σύνολο {`true`, `false`}.

.2.1 Τελεστές

- ! (λογική άρνηση)
- && (λογική σύζευξη)
- || (λογική διάζευξη)
- == (έλεγχος ισότητας)
- != (έλεγχος ανισότητας)

Για τον τελεστή || εάν έστω μία από τις συνθήκες ικανοποιείται, τότε δεν εξετάζονται οι υπόλοιπες, έστω κι αν αυτό θα εμπεριείχε συνέπειες, π.χ. κλήση συναρτήσεων. Ομοίως για τον τελεστή &&, έστω μία από τις συνθήκες αν δεν ικανοποιείται, η πρόταση θεωρείται απευθείας αναληθής χωρίς την εξέταση των υπολοίπων συνθηκών.

.3 Integers

Προσημασμένοι και μη προσημασμένοι ακέραιοι μεγέθους από 8 bits (`int8`, `uint8`), έως 256 bits (`int256`, `uint256`). Όταν δεν επισημαίνεται ρητά το μέγεθος, τότε υπονοείται η μεγαλύτερη δυνατή ακρίβεια.

.3.1 Τελεστές

Συγκρίσεις <=, <, ==, !=, >=, >

Δυαδικοί &, |, ^(xor), ~(not)

Αριθμητικοί +, -, (μοναδιαίος) -, (μοναδιαίος) +, *, /, % (υπόλοιπο), ** (εκθετικό)

.4 Διευθύνσεις

Οι μεταβλητές τύπου `address` έχουν μέγεθος 20 bytes. Εμπεριέχουν μέλη και λειτουργούν ως βάση για όλα τα smart-contracts.

.4.1 Τελεστές

Συγκριτικοί <=, <, ==, !=, >= και >

.4.2 Μέλη

balance φανερώνει το υπόλοιπο ενός χρήστη σε wei, ή του λογαριασμού που διαχειρίζεται ένα smart-contract,

send για την αποστολή Ether (σε wei μονάδες). Για contracts θα εκτελεστεί μαζί και η fallback[1] μεθοδος που αντιστοιχεί στο contract, ενώ σε περίπτωση αποτυχίας της κλήσης (π.χ. μη επαρκές gas) οι αλλαγές αναιρούνται κι επιστρέφει τιμή false,

[1] η μία και μοναδική ανώνυμη μέθοδος που επιτρέπεται ανά contract.

call επιστρέφει μία boolean τιμή που υποδεικνύει το εάν η κλήση τερμάτισε με επιτυχία ή όχι. Δεν είναι δυνατή η ανάκτηση της τιμής επιστροφής της κληθείσας συναρτησης καθώς αυτό θα προϋπέθετε τη χρήση του μεγέθους και της κωδικοποίησης των μεταβλητών εκ των προτέρων,

callcode παρομοίως με την call, αλλά με τη διαφορά ότι αν και χρησιμοποιείται ο κώδικας του contract της διεύθυνσης, για όλα τα υπόλοιπα στοιχεία (storage, balance, ...) χρησιμοποιείται το καλών contract. Αμφότερα contracts θα πρέπει να έχουν συμβατό storage layout και κατάλληλο για την κλήση της callcode.

.5 Δυαδικές μεταβλητές σταθερού μεγέθους

Μεταβλητές μεγέθους από 8 bits (`bytes1`) έως 256 bits για τον τύπο `bytes32`. Όταν δεν προσδιορίζεται το μέγεθος στον τύπο της μεταβλητής `byte` υπονοείται ο τύπος `bytes1`.

.5.1 Τελεστές

Συγκρίσεις <=, <, ==, !=, >=, > (evaluate to bool)

Δυαδικοί &, |, ^(xor), ~(not)

Index access $x[k]$ επιστρέφει το k-οστό byte εφόσον το k είναι μικρότερο από το μήκος του τύπου.

.5.2 Μέλη

.length φανερώνει το μέγεθος της μεταβλητής και δεν είναι τροποποιήσιμο.

.6 Μεταβλητές δυναμικού μεγέθους

Για αδιευκρίνιστου μεγέθους δυαδικά δεδομένα ο τύπος `bytes`, και για αλφαριθμητικά κωδικοποίησης UTF-8 ο τύπος `string`.

.7 Σταθερές

Οι αριθμητικές σταθερές είναι ακέραιοι με αυθαίρετη ακρίβεια έως ότου χρησιμοποιηθούν με μεταβλητές, όπου τότε λαμβάνουν τον μικρότερου μεγέθους τύπο που ταιριάζει και χωράει το αποτέλεσμα της αριθμητικής πράξης. Ακόμα είναι δυνατόν το να ξεπεραστεί το μέγεθος των 256 bits, εφόσον μόνο ακέραιες σταθερές χρησιμοποιούνται για τον υπολογισμό.

Οι αλφαριθμητικές σταθερές υποδηλώνονται με διπλά εισαγωγικά (“abc”). Ο τύπος τους μπορεί να ποικίλει μεταξύ `bytes` και `string`.

.8 Απαριθμήσεις

Πρόκειται για τύπους δεδομένων που ορίζονται από τον προγραμματιστή. Μετατρέπονται προς και από όλους τους ακέραιους τύπους ενώ δεν επιτρέπεται η χωρίς ρητή μετατροπή από τον χρήστη πράξη με άλλους τύπους πλυν της συγκεκριμένης απαρίθμησης.

```
enum ActionChoices { GoLeft, GoRight, GoStraight, SitStill }

ActionChoices constant choice = ActionChoices.GoStraight;

function getChoice() returns (ActionChoices) {
```

```
    return choice;
}
```

Στο ανωτέρω παράδειγμα ο τύπος επιστροφής της συνάρτησης `getChoice()` μετατρέπεται σε `uint8`, αφού η απαρίθμηση δεν είναι μέρος του Application Binary Interface (ABI).

.9 Πίνακες

Μπορούν να έχουν σταθερό ή δυναμικό μέγεθος. `Memory arrays` δεν μπορούν να έχουν στοιχεία τύπου `mapping` ενώ αν πρόκειται για όρισμα `public` μεθόδου θα πρέπει να είναι ενός εκ των ABI τύπων.

Ένας πίνακας τύπου T και σταθερού μήκους k δηλώνεται ως $T[k]$, ενώ ο αντίστοιχος δυναμικός πίνακας ως $T[\,]$. Ένας πίνακας αποτελούμενος από 5 δυναμικούς υποπίνακες ακεραίων δηλωνεται ως $uint[\,][5]$. Η προσπέλαση του δεύτερου ακεραίου από τον τρίτο υποπίνακα γίνεται ως $x[2][1]$.

Οι μεταβλητές τύπου bytes και string αποτελούν ειδικές περιπτώσεις πινάκων. Μία μεταβλητή τύπου `bytes` δεν ταυτίζεται με έναν πίνακα `byte[ ]`. Μία μεταβλητή τύπου `string` ισοδυναμεί με μία `bytes`, αν και δεν επιτρέπει index access και δεν εμπεριέχει μέλος που να υποδηλώνει το μήκος.

length υποδηλώνει το μήκος του πίνακα και μπορεί να αυξομειωθεί για δυναμικούς πίνακας ώστε να προσαρμοστεί κατάλληλα το μέγεθός του. Τέτοιου είδους αναπροσαρμογές δεν μπορούν να γίνουν όμως με ανάθεση σε θέση πέρα από το μήκος του πίνακα, εναλλακτικά με τη χρήση της `push`.

push σε δυναμικού μεγέθους πίνακες κι μεταβλητές τύπου `bytes` (αλλά όχι τύπου `string`) η συνάρτηση αυτή προσθέτει ένα στοιχείο στο τέλος του πίνακα κι αυξάνει το μέγεθός του κατά ένα. Το νέο μήκος αποτελεί την τιμή επιστροφής της συνάρτησης.

.10 Ευρετήρια

Δηλώνονται ως `mapping (_KeyType => _ValueType)`, όπου το `_KeyType` μπορεί να είναι οποιοσδήποτε τύπος εκτός από `mapping`. Το `_ValueType` μπορεί να είναι οποιοσδήποτε τύπος, συμπεριλαμβανομένου και του `mapping`. Όταν δεν υπάρχει το αιτούμενο στοιχείο στο `mapping` τότε επιστρέφεται ένα στοιχείο που αντιστοιχεί στη δυαδική αναπαράσταση του `_ValueType` τύπου με όλα τα δυφία 0.

Τα ζεύγη κλειδιών-τιμών δεν αποθηκεύονται στο `mapping` καθεαυτό, αντ' αυτού η τιμή που αντιστοιχεί στο `sha3` hash τους. Για αυτόν το λόγο δεν έχουν ακριβές μέγεθος που να επιστρέφεται μέσω κάποιου πεδίου στον προγραμματιστή, ή κάποιο τρόπο προκειμένου να είναι δυνατή η ανάκτηση του συνόλου των διαθέσιμων κλειδιών που υπάρχουν στο ευρετήριο. Τέλος, μπορούν να χρησιμοποιηθούν μόνο ως μεταβλητές κατάστασης του contract ή ως `storage` μεταβλητές εντός των συναρτήσεων.

.11 Ειδικές Μεταβλητές

- block.coinbase (address): current block miner's address
- block.difficulty (uint): current block difficulty
- block.gaslimit (uint): current block gaslimit
- block.number (uint): current block number
- block.blockhash (function(uint) returns (bytes32)): hash of the given block - only for 256 most recent blocks

- block.timestamp (uint): current block timestamp
- msg.data (bytes): complete calldata
- msg.gas (uint): remaining gas
- msg.sender (address): sender of the message (current call)
- msg.sig (bytes4): first four bytes of the calldata (i.e. function identifier)
- msg.value (uint): number of wei sent with the message
- now (uint): current block timestamp (alias for block.timestamp)
- tx.gasprice (uint): gas price of the transaction
- tx.origin (address): sender of the transaction (full call chain)

Παράρτημα γ' Serpent

Το συνακτικό της προγραμματιστικής γλώσσας Serpent [14] είναι πολύ παρόμοιο με της Python. Πρόκειται δηλαδή για weak dynamic typing γλώσσα όπου τα κενά, tabs και spaces εν γένει είναι πολύ σημαντικά κι αποτελούν μέρος του συντακτικού. Αυτό σημαίνει στην πράξη ότι η εσωτερική στοίχιση δημιουργεί ένα ξεχωριστό block πηγαίου κώδικα και θα εκτελεστεί εφόσον η συνθήκη που το προδιαγράφει (if/elif/else-branching, while-loop) ισχύει. Ομοίως για τους ορισμούς συναρτήσεων, όπου το σώμα της συνάρτησης είναι στοιχισμένο εσωτερικά της δήλωσης της συνάρτησης.

.1 Μεταβλητές

Οι τοπικές μεταβλητές των συναρτήσεων δηλώνονται χωρίς τύπο αφού αυτός εξάγεται από την τιμή της ανάθεσης, ενώ επιπλέον μπορεί να αλλάξει σε δεύτερο χρόνο κατά την εκτέλεση του προγράμματος. Για παράδειγμα, οι παρακάτω έγκυρες δηλώσεις:

```
a = 5
b = "hello"
if a > 0:
    a = b
```

.2 Βρόχοι

Ο μόνος τρόπος επανάληψης με βρόχο γίνεται σε while-loop. Σε αντιπαραβολή με όλες τις γνωστές γλώσσες προγραμματισμού, η συνθήκη έπεται του `while` keyword κι ο βρόχος επανάληψης κείται εμφωλευμένος ως προς τη στοίχιση. Τοπικές μεταβλητές είθιστε να ορίζονται πριν κι εντός του σώματος του βρόχου.

.3 Πίνακες

Οι πίνακες που χρησιμοποιούνται εντός των συναρτήσεων έχουν σταθερό μέγεθος, αλλά υπάρχει η δυνατότητα παραγωγής "υπο-πινάκων" τους με system-calls όπως η `slice()`. Στα θετικά της γλώσσας το ότι οι συναρτήσεις έχουν τη δυνατότητα να επιστρέφουν πίνακες μέσω της ανάλογης δήλωσης `:arr`.

.4 Συμβολοσειρές

Στο Serpent υπάρχει η δυνατότητα δύο διακριτών τύπων συμβολοσειρών με διαφορετικά χαρακτηριστικά έκαστη και να χρήζει διαφορετικής αντιμετώπισης από τον προγραμματιστή.

Μεταβλητή	Χρήση
tx.origin	Αποθηκεύει τη διεύθυνση από την οποία ξεκίνησε η συναλλαγή.
tx.gasprice	Αποθηκεύει το κόστος σε gas της τρέχουσας συναλλαγής.
tx.gas	Αποθηκεύει την υπολοιπόμενη ποσότητα gas.
msg.sender	Αποθηκεύει τη διεύθυνση του χρήστη που κάνει την κλήση της μεθόδου του contract.
msg.value	Αποθηκεύει την ποσότητα κρυπτονομίσματος σε Wei που αποστέλεται με την κλήση.
self	Η διεύθυνσή του smart-contract.
self.balance	Η ποσότητα κρυπτονομίσματος που ελέγχει κι αξιοποιεί το contract.
x.balance	Όπου x οποιαδήποτε διεύθυνση χρήστη ή contract. Η ποσότητα κρυπτονομίσματος που διαθέτει κι ελέγχει.
block.coinbase	Αποθηκεύει τη διεύθυνση του miner κόμβου που επεξεργάζεται τη συναλλαγή.
block.timestamp	Αποθηκεύει τη χρονική στιγμή που πιστοποιείται η συναλλαγή και προστίθεται το block στην αλυσίδα.
block.prevhash	Αποθηκεύει τη διεύθυνση του block που προηγείται.
block.difficulty	Αποθηκεύει τη δυσκολία του block προς mining.
block.number	Αποθηκεύει το αριθμητικό αναγνωριστικό του block.
block.gaslimit	Αποθηκεύει το μέγιστο όριο gas που μπορεί να μεταφερθεί με την κλήση της μεθόδου.

Πίνακας 1: Ειδικές μεταβλητές για Ethereum εφαρμογές σε περιβάλλον Serpent.

Η πρώτη κατηγορία ονομάζεται short strings κι οι μεταβλητές του τύπου αυτού συμπεριφέρονται σαν πρωτεύων τύπος (primitive). Εν αντιθέση, η δεύτερη κατηγορία ονόματι long strings αντιμετωπίζεται ως πίνακας χαρακτήρων κι η διαχείρισή του γίνεται με τη χρήση system-calls, ακόμα και για πρόσβαση στα στοιχεία του πίνακα ξεχωριστά, π.χ. `setch()`, `getch()`.

.5 Συναρτήσεις

Οι εσωτερικές κλήσεις μεθόδων του smart-contract γίνονται με αναφορά στο τρέχον contract με τη χρήση του keyword `self`. Σημειωτέον ότι όλες οι συναρτήσεις είναι προσβάσιμες στον έξω κόσμο κι αυτό διαφοροποιεί αρκετά τον τρόπο ανάπτυξης του κώδικα. Επιπλέον, υπάρχουν τρεις ειδικές κατηγορίες συναρτήσεων που προηγούνται των μεθόδων της διεπαφής του contract, ως δείχνουμε παρακάτω.

init για την αρχικοποίηση του smart-contract και των μεταβλητών κατάστασης.

shared εκτελείται πριν από οποιαδήποτε άλλη συνάρτηση, ακόμα κι από την `init`. Είθιστε να χρησιμοποιείται για την αρχικοποίηση σταθερών.

any η εκτέλεσή της προηγείται όλων των συναρτήσεων πλην της `init`.

.6 Μεταβλητές Κατάστασης

Οι μεταβλητές κατάστασης δηλώνονται με τη χρήση του keyword `data` έξω από οποιαδήποτε συνάρτηση. Είναι προσβάσιμες από όλες τις συναρτήσεις του smart-contract. Ακόμα, μπορούν να αφορούν πίνακες καθώς και πλειάδες (tuples), όπως για παράδειγμα:

```
data twoDimArray[][]
data arrayWithTuples[](item1, item2)
```

όπου η παρακάτω ανάθεση διαβάζει το δεύτερο στοιχείο της πλειάδας από το τρίτο στοιχείο του πίνακα,

```
x = self.arrayWithTuples[2].item1
```

Τέλος, το Ethereum παρέχει σε κάθε contract ένα ευρετήριο ονόματι `self.storage[]`, αν και η χρήση του θεωρείται ελαφρώς παρωχημένη.

Βιβλιογραφία

[1] 333 eth. https://333eth.io/.

[2] Coingecko. www.coingecko.com/en.

[3] Proof of work vs proof of stake: Basic mining guide. https://blockgeeks.com/guides/proof-of-work-vs-proof-of-stake/.

[4] Solidity project. https://solidity.readthedocs.io/en/v0.4.25/.

[5] Web3.js github page. https://github.com/ethereum/web3.js/.

[6] Solidity documentation release 0.4.26. https://media.readthedocs.org/pdf/solidity/develop/solidity.pdf, 2018.

[7] Jimmy Aki. Ethereum developers reduce ether rewards to 2 eth, delay "difficulty bomb". https://coinjournal.net/ethereum-developers-reduce-ether-rewards-to-2-eth-delay-difficulty-bomb/, 2018-09-03.

[8] BlockGeeks. Two-node setup of a private ethereum on aws with contract deployment (part 1). https://blockgeeks.com/two-node-setup-of-a-private-ethereum/, 2018.

[9] BlockGeeks. Two-node setup of a private ethereum on aws with contract deployment (part 2). https://blockgeeks.com/two-node-setup-of-a-private-ethereum-on-aws-with-contract-deployment-part-2/, 2018.

[10] BlockGeeks. What is ethereum casper protocol? crash course. https://blockgeeks.com/guides/ethereum-casper/, 2018.

[11] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper, 2013.

[12] Gertrude Chavez-Dreyfuss. New york regulator approves winklevoss, paxos dollar-linked tokens. www.reuters.com/article/us-cryptocurrency-dollar-paxos-winklevos/new-york-regulator-approves-winklevoss-paxos-dollar-linked-tokens-idUSKCN1LQ1O5, 2018-09-10.

[13] Rick D. Hackers steal $200,000 worth of eos, dapp had smart contract flaw. www.newsbtc.com/2018/09/15/hackers-steal-200000-worth-of-eos-dapp-had-smart-contract-flaw/, 2018-09-15.

[14] Kevin Delmolino, Mitchell Arnett, Ahmed Kosba, Andrew Miller, and Elaine Shi. A programmer's guide to ethereum and serpent. Technical report, University of Maryland, 2015.

[15] Kevin Delmolino, Mitchell Arnett, Ahmed Kosba, Andrew Miller, and Elaine Shi. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. Technical report, University of Maryland, 2015.

[16] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[17] F. A. Hayek. Denationalisation of money - the argument refined, 1978.

[18] Sherman Lee. Explaining stable coins, the holy grail of cryptocurrency. www.forbes.com/sites/shermanlee/2018/03/12/explaining-stable-coins-the-holy-grail-of-crytpocurrency/#aad51b34fc64, 2018-03-12.

[19] Jack Morse. One of the most popular ethereum apps sure looks like a ponzi scheme. https://mashable.com/article/ethereum-dapp-ponzi-scheme/?europe=true#O9ocg8rNmZq6, 2018-09-15.

[20] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. www.bitcoin.org, 2012.

[21] Arvind Narayanan and Jeremy Clark. Bitcoin's academic pedigree. https://queue.acm.org/detail.cfm?id=3136559, 2017.

[22] Arvind Narayanan and Andrew Miller. Research for practice: Cryptocurrencies, blockchains, and smart contracts. http://randomwalker.info/publications/research-for-practice-cryptocurrencies.pdf, 2017.

[23] Nethereum. Test rpc configuration and usage. https://nethereum.readthedocs.io/en/latest/ethereum-and-clients/test-rpc/, 2018.

[24] Santiago Palladino. The parity wallet hack explained. https://blog.zeppelin.solutions/on-the-parity-wallet-multisig-hack-405a8c12e8f7, 2017-06-20.

[25] Santiago Palladino. The parity wallet hack reloaded. https://blog.zeppelinos.org/parity-wallet-hack-reloaded/, 2017-11-07.

[26] Santiago Palladino. Designing the architecture for your ethereum application. https://blog.zeppelin.solutions/designing-the-architecture-for-your-ethereum-application-9cec086f8317, 2017-11-21.

[27] Lukas Ruebbelke. *AngularJS in Action*. Manning, 2015.

[28] David Siegel. Understanding the dao attack. www.coindesk.com/understanding-dao-hack-journalists/, 2016-06-25.

[29] Max Thake. What is proof of stake? (pos). https://medium.com/nakamo-to/what-is-proof-of-stake-pos-479a04581f3a, 2018-07-08.

[30] Truffle. Fast ethereum rpc client for testing and development. https://github.com/trufflesuite/ganache-cli, 2017.

[31] Oscar Williams-Grut. Everything you need to know about tether, the cryptocurrency academics claim was used to manipulate bitcoin. www.businessinsider.com/tether-explained-bitcoin-cryptocurrency-why-people-worried-2018-1, 2018-06-13.

[32] Gavin Wood. Ethereum: A secure decentralised deneralised transaction ledger. Technical report, Ethereum and Ethcore, 2014.

[33] Nicole Zhu. 5 minute guide to deploying smart contracts with truffle and ropsten. https://medium.com/coinmonks/5-minute-guide-to-deploying-smart-contracts-with-truffle-and-ropsten-b3e30d5ee1e, 2018.

[34] Π. Δημητράκος. Κρυπτονόμισμα : Ένα αναδυόμενο εργαλείο οικονομικής ελευθερίας. www.acg.edu/ckeditor_assets/attachments/2063/kriptonomismata.pdf.