



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ

Δ.Π.Μ.Σ ΕΠΙΣΤΗΜΗ ΔΕΔΟΜΕΝΩΝ & ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

Εξαμηνιαία Εργασία

Κατανεμημένα Συστήματα

“Noobchain: a Blockchain Implementation”

Νικήτας Νικόλαος 03400043

Ζωγραφάκης Δημήτριος 03400050

Λαμπράκης Δημήτριος 03400059

Αθήνα

03/04/2020

Περιεχόμενα

1	Εισαγωγή	2
1.1	Σημειώσεις	2
1.2	Restful Αρχιτεκτονική του API	3
2	Σχεδιασμός Του Συστήματος	4
2.1	Κύρια Εφαρμογή	4
2.1.1	Κλάση Node	4
2.1.2	Κλάση Block	7
2.1.3	Κλάση Blockchain	7
2.1.4	Κλάση Wallet	8
2.1.5	Κλάση Transaction	9
2.1.6	Αλληλεπιδράσεις Κλάσεων	10
2.2	Εφαρμογή Client	11
2.2.1	Σελίδα Home	11
2.2.2	Σελίδα Wallet	12
2.2.3	Σελίδα Blockchain	12
2.2.4	Σελίδα About	13
2.2.5	Σελίδα Help	13
2.3	Επιπλέον Υλοποιήσεις	14
3	Αξιολόγηση Συστήματος	15
3.1	Απόδοση	15
3.2	Κλιμακωσιμότητα	15
3.3	Γραφικές Παραστάσεις	16
3.4	Επεκτάσεις	18
4	Παράδειγμα Εκτέλεσης	19
	Κατάλογος πινάκων	21
	Κατάλογος σχημάτων	21

Εισαγωγή

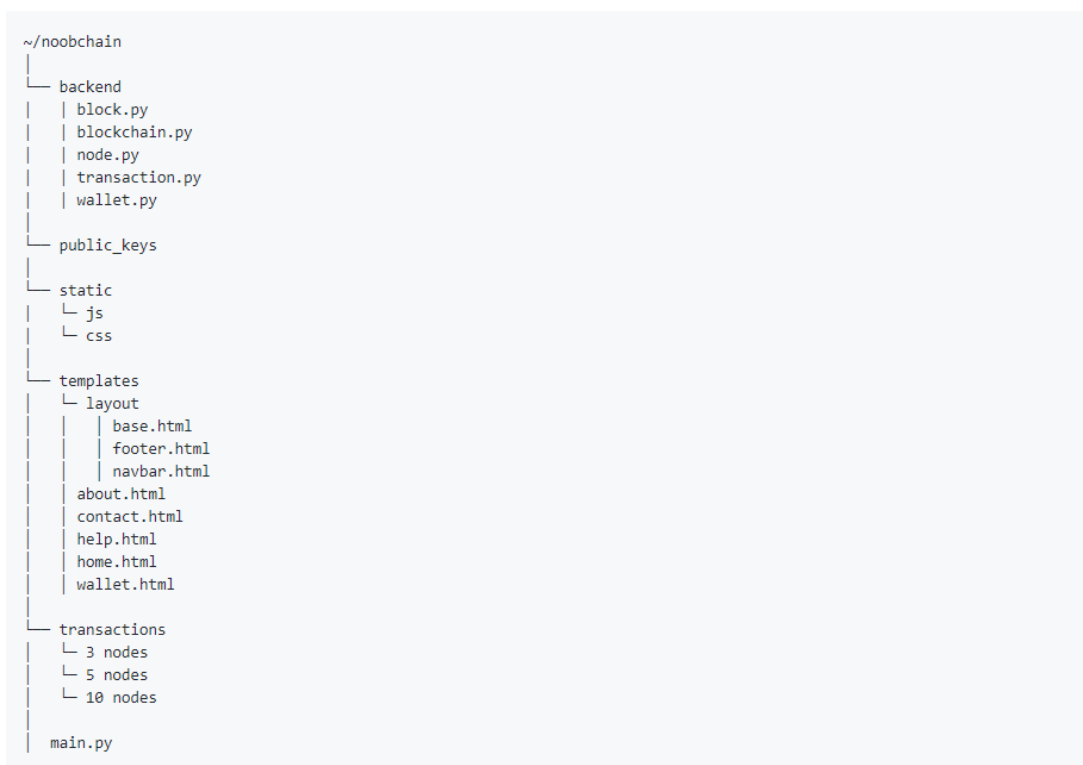
Οι διαδικασίες mining και validation ΓΙΝΟΝΤΑΙ ΠΑΡΑΛΛΗΛΑ, με χρήση νημάτων.

1.1 Σημειώσεις

Η εργασία που θα παρουσιαστεί, βρίσκεται ανεβασμένη σε ιδιωτικό αποθηκευτικό χώρο στο [Github](#). Η εργασία υλοποιήθηκε με γνώμονα επιθυμητές πρακτικές που προτείνονται από το Flask (1.0.1), ενώ κύρια γλώσσα προγραμματισμού αποτελεί η έκδοση 3.8 της Python. Για την αρτιμελή ενσωμάτωση επιπέδου διεπαφής με τον χρήστη έγινε χρήση τεχνολογιών HTML, css, Ajax, Jinja2. Επιπλέον, έγινε αξιοποίηση και του εργαλείου Docker για την εκτέλεση της εφαρμογής σε περιβάλλον Docker Container σαν δεύτερη υλοποίηση. Για την ευκολότερη ενσωμάτωση των απαιτούμενων βιβλιοθηκών Python, δίνεται αρχείο απαιτήσεων (requirements.txt) που τις εγκαθιστά με την ακόλουθη εντολή:

```
1 pip install -r requirements.txt
```

Η δομή της εργασίας είναι όπως παρουσιάζεται παρακάτω



Σχήμα 1.1: Δομή Εργασίας.

1.2 Restful Αρχιτεκτονική του API

Το τελικό αποτέλεσμα της εργασίας μπορεί να χαρακτηριστεί ως full stack. Αφού, υλοποιήθηκαν τόσο οι εφαρμογές του server, ενώ ακόμα προσφέρεται επίπεδο αλληλεπίδρασης με τον χρήστη μέσω μίας διαδικτυακής ιστοσελίδας. Η ιστοσελίδα μάλιστα, δεν παρουσιάζει έλλειψη περιεχομένου αφού εμπεριέχει τόσο βοηθητικές σελίδες, π.χ Βοήθεια, Επικοινωνία και Πληροφορίες, αλλά και προσφέρονται υπηρεσίες συναλλαγών, προβολή κατάστασης πορτοφολιού και αλυσίδας με πλήρη λειτουργικότητα. Ως αποτέλεσμα, το σύστημα αποτελεί μία RESTful αρχιτεκτονική.

Αρχικά για το frontend, δηλαδή το επίπεδο αλληλεπίδρασης με τον χρήστη μέσω διαδικτυακής εφαρμογής, γίνεται χρήση των παρακάτω διαδρομών

Πίνακας 1.1: Frontend.

Διαδρομή	Τύπος	Αποτέλεσμα
/	GET	Αρχική Σελίδα
/wallet	GET	Πορτοφόλι Χρήστη
/help	GET	Εγχειρίδιο Χρήστη
/blockchain	GET	Τωρινή Κατάσταση Αλυσίδας
/about	GET	Πληροφορίες Εφαρμογής
/contact	GET	Σελίδα Επικοινωνίας

Ενώ, οι διαδρομές του backend χρησιμοποιούνται ως επί το πλείστον από το λογισμικό, προκειμένου να ενημερώσουν δεδομένα είτε να επικοινωνήσουν για συγχρονισμό.

Πίνακας 1.2: Backend.

Διαδρομή	Τύπος	Αποτέλεσμα
/chain	GET	Get the blockchain from other nodes to resolve conflict
/transactions/view	GET	View transactions from last verified block
/transactions/money/sent	GET	Get money sent from verified transactions
/transactions/money/received	GET	Get money received from verified transactions
/transactions/create/browser	POST	Create a transaction via browser client
/transactions/create	POST	Create a transaction
/broadcast/transaction	POST	Broadcast a transaction to other nodes
/broadcast/ring	POST	Broadcast the node to other nodes (for Bootstrap node)
/broadcast/block	POST	Broadcast a block to other nodes
/nodes/register	POST	Register node to bootstrap ring

Κάποιες εκ των οποίων γίνονται διαθέσιμες μόνο σε εξειδικευμένους κόμβους, όπως για παράδειγμα τον αρχικό κόμβο (bootstrap), για την διαμοίραση του δακτυλίου των κόμβων, που αποσκοπεί στην ορθή επικοινωνία μεταξύ όλων των κόμβων του δικτύου.

Σχεδιασμός Του Συστήματος

Για την εκκίνηση της εφαρμογής, χρειάζονται κάποιες παράμετροι με την εντολή, αλλιώς θα επιλεγούν κάποιες τιμές που έχουν οριστεί ως default για το σύστημα. Οι παράμετροι αυτές που δύναται να επιλέξει ο χρήστης φαίνονται στη συνέχεια:

Πίνακας 2.1: Παράμετροι για την εκτέλεση της εφαρμογής.

Παράμετρος	Επεξήγηση
-ip	IP του κόμβου
-p	Port του κόμβου
-bootstrap	Boolean - εάν είναι ο κόμβος ο bootstrap
-ip_bootstrap	IP του bootstrap κόμβου
-port_bootstrap	Port του bootstrap κόμβου
-nodes	αριθμός των κόμβων στο δίκτυο
-cap	μέγεθος των transactions για κάθε μπλόκ
-dif	δυσκολία για το mining των blocks

Παραδείγματος χάριν, για την εκκίνηση ενός node με ορισμένες όλες τις παραμέτρους φαίνεται παρακάτω:

```
1 python3 noobchain/main.py -ip 0.0.0.0 -p 1000 -bootstrap True -ip_bootstrap 0.0.0.0  
-port_bootstrap 1000 -nodes 5 -cap 4 -dif 4
```

Για να γίνει πιο κατανοητή η δομή της εφαρμογής, θα παρουσιαστεί σε 2 τμήματα.

2.1 Κύρια Εφαρμογή

Η κύρια εφαρμογή είναι η υλοποίηση του Noobcash Client, που είναι ουσιαστικά το backend της εφαρμογής. Οι βασικές οντότητες με τα χαρακτηριστικά τους που απαρτίζουν την εφαρμογή παρουσιάζονται παρακάτω:

2.1.1 Κλάση Node

Αποτελεί την κύρια κλάση και τον συνδεδετικό κρίκο μεταξύ των υπολοίπων. Από αυτή την κλάση γίνεται στην πραγματικότητα η οργάνωση ολόκληρου του συστήματος. Τα βασικά στοιχεία του κατασκευαστή της κλάσης και τα χαρακτηριστικά των παραγόμενων αντικειμένων παρουσιάζονται στον κάτωθι πίνακα:

Πίνακας 2.2: Χαρακτηριστικά Αντικειμένου Node.

Παράμετρος	Τύπος Μεταβλητής	Επεξήγηση
ip	String	IP Address
port	Int	Listening Port
no_of_nodes	Int	Number Of Nodes In The Network
id	String	Node id
address	String	Browser Address
pending_transactions	List	Transactions To Be Mined
public	String	SHA256 Public Key
private	String	SHA256 Private Key
wallet	Wallet	Class Instance
ring	List	The Network Map
blockchain	Blockchain	Class Instance
new_block	Block	The Block To Be Mined
trans	Transaction	Item Of The Transaction Class
mining	Bool	The Mining State Of The Node
capacity	Int	Block Capacity
difficulty	Int	Mining Difficulty
is_bootstrap	Bool	True Only For Bootstrap

Η κλάση Node είναι η μόνη κλάση που διαφοροποιείται ελαφρώς μεταξύ των απλών nodes και του bootstrap. Συγκεκριμένα, αν στον κατασκευαστή της κλάσης το όρισμα “is_bootstrap” είναι **True** τότε δημιουργείται το Blockchain με το Genesis block και ο Bootstrap έχει στο Wallet του τα χρήματα που θα διαμοιράσει στους υπόλοιπους nodes μόλις συμπληρωθεί ο αριθμός τους ο οποίος ορίζεται απο το “no_of_nodes”. Στην συνέχεια θα περιγράψουμε τις συναρτήσεις που εμπεριέχονται εντός της κλάσης.

- **__str__**: Override του builtin αντικειμένου string της Python για να δηλώσουμε τον τρόπο με τον οποίο θα τυπώνεται το αντικείμενο αν κληθεί σε αυτό η print.
- **register_on_bootstrap**: Όλοι οι nodes πρέπει μόλις δημιουργηθούν να δηλώσουν την παρουσία τους στον Bootstrap και να του δώσουν την ip και το public key τους, η συνάρτηση πραγματοποιεί αυτή την δήλωση.
- **get_address**: Συνάρτηση μορφοποίησης η οποία επιστρέφει μια διεύθυνση δικτύου σε μορφή url.
- **first_time_keys**: Δημιουργεί στον κάθε node το ζεύγος private - public keys.
- **generate_wallet**: Δημιουργία πορτοφολιού στον κάθε node, το πορτοφόλι είναι αντικείμενο της κλάσης Wallet.
- **register_node_to_ring**: Συνάρτηση η οποία καλείται μόνο απο τον Bootstrap κάθε φορά που κάποιος κόμβος δηλώνει την παρουσία του στο δίκτυο. Η τελική λίστα ring που διαμορφώνεται απο τα καλέσματα αυτής της συνάρτησης κατα την εγγραφή κάθε νέου κόμβου είναι αυτή που θα διαμοιραστεί σε όλους και θα λειτουργεί σαν χάρτης του δικτύου.

-
- **broadcast_ring_to_nodes:** Όταν συμπληρωθεί ο αριθμός των προκαθορισμένων κόμβων στο δίκτυο ο Bootstrap στέλνει σε όλους τους συμμετέχοντες τον χάρτη του δικτύου για να μπορέσει να γίνει η σωστή επικοινωνία μεταξύ τους, και κυρίως σωστές συναλλαγές.
 - **respond_to_node:** Όταν συμπληρωθεί ο αριθμός των προκαθορισμένων κόμβων στο δίκτυο και έχει σταλεί ο χάρτης σε όλους, ο Bootstrap με αυτή την συνάρτηση ξεκινάει την διαδικασία δημιουργίας των πρώτων συναλλαγών κατά των οποίων όλοι οι κόμβοι θα λάβουν 100 νομίσματα.
 - **create_transaction:** Η συνάρτηση αυτή δημιουργεί μια προς εκτέλεση συναλλαγή που θέλει να πραγματοποιήσει ο κόμβος. Αρχικά υπολογίζει απο ποιιά UTXOs θα καλύψει το απαιτούμενο ποσό και στην συνέχεια δημιουργεί ένα αντικείμενο την κλάση Transaction.
 - **broadcast_transaction:** Η εκτέλεση ενός transaction στο σύστημα blockchain σημαίνει να επαληθευθεί απο όλους τους κόμβους και τλικά να μπει σε ένα block. Η συγκεκριμένη συνάρτηση αναλαμβάνει να ενημερώσει όλο το δίκτυο για την συναλλαγή που θέλει ο εκάστοτε κόμβος να εκτελέσει.
 - **validate_transaction:** Κατά την παραλαβή ενός νέου transaction από το δίκτυο πρέπει να το επαληθεύσουμε. Η διαδικασία επαλήθευσης συνοψίζεται σε δύο βασικούς ελέγχους: τον έλεγχο υπογραφής και τον έλεγχο UTXOs. Αν η συναλλαγή περάσει με επιτυχία τους ελέγχους, προστίθεται στην λίστα με τις συναλλαγές που αναμένουν να μπουόν σε block και αν έχουμε φτάσει το απαραίτητο capacity για ένα νέο block αρχίζει η διαδικασία του mine.
 - **verify_signature:** Η συνάρτηση πραγματοποιεί τον έλεγχο υπογραφής με χρήση του public key του αποστολέα στο προς εξέταση transaction.
 - **verify_value:** Πραγματοποιεί τον δεύτερο έλεγχο για ένα προς εξέταση transaction, ελέγχει αν ο αποστολέας έχει τα διαθέσιμα UTXOs που δηλώνει και αν αυτά καλύπτουν το ποσό της συναλλαγής. Αξίζει σε αυτό το σημείο να παρατηρήσουμε ότι σε ένα κανονικό σύστημα αυτός ο έλεγχος γίνεται με εξέταση ολόκληρου του blockchain, αλλά χάριν ευκολίας ο κάθε κόμβος διατηρεί μια δική του δομή δεδομένων όπου καταγράφονται όλα τα UTXOs των κόμβων. Έτσι ο έλεγχος πραγματοποιείται πολύ πιο εύκολα και γρήγορα.
 - **update_utxos:** Αν η συναλλαγή βρεθεί έγκυρη τότε πρέπει να ενημερώσουμε τα UTXOs (είτε τα δικά μας αν εμπλεκόμαστε στην συναλλαγή είτε άλλων κόμβων αν δεν εμπλεκόμαστε) για να είμαστε έτοιμοι να κρίνουμε με ορθότητα μια μελλοντική συναλλαγή.
 - **valid_block:** Άμα απο το δίκτυο κάποιος node τελειώσει το mine στέλνει το νέο block σε όλους προς επαλήθευση. Η συνάρτηση άπαξ και λάβει ένα νέο block προς εξέταση σταματάει την διαδικασία mining αν τρέχει, ελέγχει ποιες συναλλαγές εμπεριέχονται στο block και αν απο αυτές που περιέχονται στο block συμπίπτουν με κάποιες απο αυτές που έχουμε ως υπολειπόμενες στην ουρά και περιμένουν mine απο εμάς, αν το block δεν έχει τέτοιες συναλλαγές το απορρίπτουμε. Αντιθέτως, αν περάσει τους βασικούς ελέγχους για την σύνδεση με το δικό μας blockchain το προσθέτουμε ενώ αν δεν περάσει τους ελέγχους ελέγχουμε μήπως υπάρχουν conflicts στα blockchain των κόμβων και επιχειρούμε να τα λύσουμε.
 - **mine:** Την υλοποίηση της συνάρτησης αυτής αναλαμβάνει αποκλειστικά ένα thread, το οποίο τρέχει για πάντα. Το thread βρίσκεται σε κατάσταση ύπνου όσο δεν υπάρχουν αρκετές συναλλαγές για να γεμίσει ένα Block ($< capacity$). Όταν οι συναλλαγές που έχουν επικυρωθεί ξεπεράσουν το όριο, τότε το thread ξυπνάει και αναλαμβάνει να κάνει mine το εν λόγω μπλοκ. Όταν ολοκληρωθεί η διαδικασία, το μπλοκ γίνεται broadcast στους υπόλοιπους κόμβους.
 - **read_file:** Η συνάρτηση αυτή διβάζει απο κάποιο αρχείο, ένα για τον κάθε node, συναλλαγές (μια ανα γραμμή) και τις εκτελεί. Αποσκοπεί στην αυτοματοποίηση εκτέλεσης κάποιων τυχαίων δοκιμαστικών συναλλαγών για την μελέτη του συστήματος.
-

2.1.2 Κλάση Block

Η κλάση block, αποτελεί ένα instance του blockchain. Αναλυτικότερα, επρόκειτο για ένα κοντέινερ που εμπεριέχει το πλήθος συναλλαγών όπως αυτό ορίζεται από την χωρητικότητα (capacity). Συνοπτικά, τα εκ των έσω του Block παρουσιάζονται στον παρακάτω πίνακα.

Πίνακας 2.3: Χαρακτηριστικά Αντικειμένου Block.

Παράμετρος	Τύπος Μεταβλητής	Επεξήγηση
index	Int	Incremental Unique Identifier
timestamp	Datetime	UNIX Timestamp
transactions	List<Transaction>	List of Transaction Objects
nonce	Int	Number of tried Hashes
previous_hash	String	Previous Block Hash
current_hash	String	Current's Block Hash

- **to_od:** Μετατροπή του Block σε διατεταγμένο λεξικό.
- **to_json:** Μετατροπή του Block σε json μορφή για να μπορεί να γίνει εύκολη μεταφορά και επεξεργασίας στο δίκτυο.
- **get_hash:** Επιστρέφει το Hash του διατεταγμένου λεξικού σε μορφή συμβολοσειράς.
- **get_hash_obj:** Επιστρέφει το Hash του διατεταγμένου λεξικού σε μορφή αντικειμένου sha256.

2.1.3 Κλάση Blockchain

Η κλάση Blockchain αναλαμβάνει την υλοποίηση των μεθόδων που χρειάζονται για την εύρυθμη λειτουργία και συντήρηση της αλυσίδας του κάθε κόμβου.

Πίνακας 2.4: Χαρακτηριστικά Αντικειμένου Blockchain.

Παράμετρος	Τύπος Μεταβλητής	Επεξήγηση
ring	List<Dictionary>	Network Map
genesis	Block	Genesis Block
my_id	String	The Node Id on the Network
resolve	Bool	Update Erroneous Chain
blocks	List<Block>	List of Blocks

Ο κατασκευαστής της κλάσης πρέπει να αντιμετωπίσει μια βασική ιδιαιτερότητα, αυτή του Genesis block. Είναι ένα block το οποίο περιέχει μόνο μια συναλλαγή, αυτή της δημιουργίας του αρχικού ποσού με παραλήπτη τον Bootstrap και δεν επαληθεύεται απο κανέναν κόμβο. Προς αποφυγήν κακών συνεννοήσεων σε αυτό το βήμα ο κάθε κόμβος δημιουργεί το Genesis block με το που καλείται η δημιουργία του blockchain του με την ιδιαιτερότητα ότι το timestamp δημιουργίας του είναι κοινό και ίσο με μηδέν (0) για όλους. Η “καρδιά” της κλάσης εντοπίζεται στις εσωτερικές της μεθόδους τις οποίες θα περιγράψουμε ευθύς αμέσως.

- **__str__:** Override του builtin αντικειμένου string της Python για να δηλώσουμε τον τρόπο με τον οποίο θα τυπώνεται το αντικείμενο αν κληθεί σε αυτό η print.

- **add_block:** Προσθέτει ένα αντικείμενο Block στην αλυσίδα, την λίστα με τα ήδη υπάρχοντα blocks.
- **mine_block:** Η συνάρτηση αυτή υπολογίζει το nonce του block για το οποίο θα εκτελεστεί η διαδικασία του mine δοκιμάζοντας σειριακά έναν προς έναν αριθμούς για nonce μέχρι να καλυφθεί η απαίτηση για αρχή του hash από τόσα μηδενικά όσα ορίζει η παράμετρος “difficulty”.
- **broadcast_block:** Στην περίπτωση κατά την οποία έχουμε επιτυχία στην διαδικασία του mine, το νέο block πρέπει να ανακοινωθεί σε όλους τους κόμβους για να υιοθετηθεί στην αλυσίδα ολονών, μετά απο τους απαραίτητους ελέγχους φυσικά.
- **resolve_conflict:** Άμα ένα block προς επαλήθευση αποτύχει κάποιον έλεγχο και δεν μπορέσουμε να το προσαρτήσουμε στην τοπική αλυσίδα, υπάρχει περίπτωση να εργαζόμαστε σε ένα άλλο branch της αλυσίδας και ζητάμε από όλα τα nodes να μας στείλουν τις αλυσίδες τους να δούμε να υπάρχει μεγαλύτερη σε μήκος και αφού την επαληθεύσουμε να την υιοθετήσουμε.
- **to_od:** Μετατροπή του Blockchain σε διατεταγμένο λεξικό.
- **to_od_with_hash:** Μετατροπή του Blockchain σε διατεταγμένο λεξικό με στοιχεία λίστας tuples που εμπεριέχουν το block αλλά και το hash του.
- **to_json:** Μετατροπή του Blockchain σε json μορφή για να μπορεί να γίνει εύκολη μεταφορά και επεξεργασίας στο δίκτυο.
- **validate_block:** Όταν παραλάβουμε ένα block από το δίκτυο προς επαλήθευση πρέπει να του κάνουμε δύο βασικούς ελέγχους. Ο πρώτος και ο βασικότερος έλεγχος είναι να δούμε ότι όντως το nonce που συνοδεύει το block ικανοποιεί τον περιορισμό του “difficulty” και ο δεύτερος έλεγχος είναι να δούμε αν το block μπορεί να προσαρτηθεί στο blockchain μας.
- **validate_chain:** Όταν ζητήσουμε απο το δίκτυο ένα blockchain πρέπει να ελέγξουμε την ορθότητά του πριν αντικαταστήσουμε το δικό μας. Η ορθότητα ελέγχεται απο 2 παραμέτρους, αν τα hash των επιμέρους block είναι σωστά και αν η αλυσίδα είναι σωστά συνδεδεμένη.

2.1.4 Κλάση Wallet

Η κλάση Wallet υλοποιεί την ιδέα του πορτοφολιού στο δίκτυο noobchain, με τα χαρακτηριστικά της να συνοψίζονται στον ακόλουθο πίνακα.

Πίνακας 2.5: Χαρακτηριστικά Αντικειμένου Wallet.

Παράμετρος	Τύπος Μεταβλητής	Επεξήγηση
public_key	String	The wallet's address
private_key	String	The wallet's signing key
utxos	Dictionary	My unspend transactions
others_utxos	List	Unspent transactions from the other nodes
value	Float	The amount of noobcash I own

Κάθε κόμβος του δικτύου κατά την αρχικοποίησή του έχει δημιουργήσει ένα ζεύγος public/private keys. Το public_key λειτουργεί σε πλήρη αντιστοιχία σαν τον αριθμό λογαριασμού σε μια τράπεζα. Το private_key είναι το κλειδί που πρέπει να κρατείται κρυφό και είναι αυτό με το οποίο υπογράφονται οι συναλλαγές προς εκτέλεση για να ελεγχθεί η εγκυρότητα τους. Αν θέλουμε να διατηρήσουμε μια αναλογία με το κλασικό τραπεζικό σύστημα το private_key είναι το pin που εισάγουμε για την εκτέλεση νέων συναλλαγών. Η κλάση wallet προφανώς συνοδεύεται και απο τις βασικές συναρτήσεις που φροντίζουν για την σωστή συμπεριφορά του:

- **wallet_balance:** Υπολογίζει το άθροισμα των UTXOs που υπάρχουν στο πορτοφόλι για να έχουμε μια εικόνα των διαθέσιμων νομισμάτων.
- **get_public_key:** Επιστρέφει το δημόσιο κλειδί του πορτοφολιού.
- **get_private_key:** Επιστρέφει το ιδιωτικό κλειδί του πορτοφολιού.
- **get_transactions:** Επιστρέφει τις συναλλαγές που υπάρχουν μέσα στο πορτοφόλι.
- **sign_transaction:** Όταν δημιουργείται ένα νέο transaction πρέπει ο εντολέας να το υπογράψει με το ιδιωτικό του κλειδί, η διαδικασία της υπογραφής υλοποιείται με την συγκεκριμένη συνάρτηση.
- **to_od:** Μετατροπή του wallet σε διατεταγμένο λεξικό.
- **to_json:** Μετατροπή του wallet σε json μορφή για να μπορεί να γίνει εύκολη μεταφορά και επεξεργασίας στο δίκτυο.

2.1.5 Κλάση Transaction

Η κλάση που ολοκληρώνει την υλοποίηση του backend, είναι η κλάση Transactions και αναλαμβάνει την κατασκευή των προς εκτέλεση συναλλαγών. Τα χαρακτηριστικά της εμφανίζονται στον κάτωθι πίνακα:

Πίνακας 2.6: Χαρακτηριστικά Αντικειμένου Transaction.

Παράμετρος	Τύπος Μεταβλητής	Επεξήγηση
sender_address	String	Sender's Public Key
receiver_address	String	Receiver's Public Key
transaction_id	String	The unique identifier of the transaction
transaction_inputs	List	List of UTXOs to spend
transaction_outputs	List	List of UTXOs to be created
signature	String	Sender's Private Key
wallet	Wallet	Class Instance
change	Int	Coins to return
node_id	String	Node's unique identifier

Ο βασικός πυρήνας αυτής της κλάσης υλοποιείται στον κατασκευαστή της κλάσης και όχι από άλλες εσωτερικές συναρτήσεις. Ο κατασκευαστής αναλαμβάνει να υπολογίσει τα transaction_outputs, δηλαδή το νέο UTXO που θα δημιουργηθεί στον παραλήπτη και αν χρειάζεται ο αποστολέας να έχει ρέστα, τότε δημιουργεί και ένα δεύτερο transaction_output με UTXO που αφορά τον αποστολέα. Εκτός από το κατασκευαστή της κλάσης υπάρχουν και δύο εσωτερικές συναρτήσεις:

- **to_od:** Μετατροπή του transaction σε διατεταγμένο λεξικό.
- **to_json:** Μετατροπή του transaction σε json μορφή για να μπορεί να γίνει εύκολη μεταφορά και επεξεργασίας στο δίκτυο.

2.1.6 Αλληλεπιδράσεις Κλάσεων

Δεδομένης της παράλληλης φύσης του project είναι αρκετά δύσκολη η περιγραφή όλων των αλληλεπιδράσεων που συμβαίνουν στο σύστημα κάθε δεδομένη χρονική στιγμή. Για τον λόγο αυτό θα προσπαθήσουμε να αρκεστούμε σε μια κάπως πιο σειριοποιημένη περιγραφή του συστήματος, αλλά πάντα πρέπει να έχουμε σαν γενική ιδέα ότι όπου αναφέρεται η λειτουργία με “thread” το σύστημα εκτελεί σε ένα παράλληλο επίπεδο μια ξεχωριστή λειτουργία.

Η αρχικοποίηση του συστήματος ξεκινάει με την δήλωση του bootstrap node. Ο κόμβος αυτός είναι υπεύθυνος για τον αρχικό διαμοιρασμό των πρώτων και μοναδικών χρημάτων που θα υπάρχουν στο δίκτυο. Όλοι οι κόμβοι, συμπεριλαμβανομένου και του Bootstrap, δημιουργούν στην αλυσίδα τους το πρώτο block, το λεγόμενο Genesis, το οποίο περιέχει ένα και μοναδικό transaction, αυτό κατά το οποίο ο Bootstrap λαμβάνει το συνολικό αριθμό νομισμάτων του δικτύου (100 νομίσματα για τον κάθε κόμβο). Το Genesis είναι το μοναδικό block το οποίο δεν γίνεται mine. Κάθε κόμβος που εισέρχεται στο δίκτυο δηλώνει την παρουσία του στον Bootstrap ο οποίος περιμένει να συμπληρωθούν όλες οι προκαθορισμένες θέσεις. Μόλις γεμίσει το δίκτυο (καλυφθούν όλες οι θέσεις από κόμβους) ο Bootstrap, στέλνει σε όλους τους κόμβους τον χάρτη του δικτύου με στόχο όλοι να έχουν επίγνωση σε ποιά διεύθυνση βρίσκεται ο καθένας, ποίο είναι το δημοσίο κλειδί όλων των κόμβων και τι id έχει ο κάθε κόμβος. Αμέσως μετά ο Bootstrap αναλαμβάνει να “τροφοδοτήσει” τον κάθε συμμετέχοντα με το αρχικό ποσό που θα έχει στην διάθεση του για να πραγματοποιήσει συναλλαγές, 100 νομίσματα ο καθένας. Αυτές οι συναλλαγές είναι ανεξάρτητες από το Genesis και ανάλογα με την χωρητικότητα του κάθε block μπορεί να χρειαστεί να γίνουν mine.

Απο την στιγμή που ο κάθε κόμβος διαθέτει το αρχικό ποσό που του δόθηκε από τον Bootstrap το σύστημα είναι έτοιμο να αρχίσει την διαδικασία των συναλλαγών. Κάθε node αρχίζει 2 βασικά threads, ένα που αναλαμβάνει την ανάγνωση συναλλαγών που πρέπει να εκτελέσει από συγκεκριμένο αρχείο και ένα thread που αναμένει διαρκώς να γεμίσει το capacity ενός block για να ξεκινήσει η διαδικασία του mine. Οι εκτελούμενες συναλλαγές βασίζονται στην λογική των Unspent Transactions (UTXOs), δηλαδή αν ένας κόμβος έχει στην διάθεση του τα αρχικά 100 νομίσματα και θέλει να στείλει 10 σε κάποιον άλλον, πρέπει υποχρεωτικά να στείλει και τα 100 και από την συναλλαγή που θα προκύψει να πάρει ρέστα 90 νομίσματα. Τα 90 νομίσματα γίνονται το νέο UTXO του αποστολέα ενώ τα 10 γίνονται ένα επιπλέον UTXO του παραλήπτη. Για να θεωρηθεί μια συναλλαγή έγκυρη από θέμα αξίας νομισμάτων που μεταφέρεται πρέπει ο κόμβος που πραγματοποιεί την συναλλαγή να έχει στην διάθεση του τόσα UTXOs που να καλύπτουν το ποσό, αν ξεπερνάνε σε αξία το επιθυμητό ποσό θα πάρει σε “ρέστα” ένα νέο UTXO. Κανονικά η διαδικασία επαλήθευσης των UTXOs του αποστολέα σε μια συναλλαγή πρέπει να γίνει με σειριακή ανάγνωση της αλυσίδας, όμως για μεγαλύτερη ταχύτητα και ευκολία, κάθε κόμβος διατηρεί ένα τοπικό αντίγραφο με τα διαθέσιμα UTXOs όλων των κόμβων και τα συγκρίνει τοπικά. Προφανώς, η διαδικασία ανάγνωσης, δημιουργίας, αποστολής και επαλήθευσης μιας συναλλαγής συμβαίνει πιο συχνά από την δημιουργία και το mine ενός νέου block. Για τον λόγο αυτό κάθε node διατηρεί μια λίστα με συναλλαγές που έχει έτοιμες να μπουν στο νέο block. Τον έλεγχο για τον αν η λίστα έχει ικανό αριθμό συναλλαγών για να δημιουργηθεί νέο block τον πραγματοποιεί διαρκώς το thread του mine. Στην περίπτωση κατά την οποία υπάρχει επαρκής αριθμός συναλλαγών για την δημιουργία ενός νέου block, ξεκινάει η διαδικασία του mine.

Η διαδικασία του mine η οποία συμβαίνει σε ξεχωριστό thread για να μπορέσουμε να πραγματοποιούμε και συναλλαγές ταυτόχρονα, αποτελεί την προσπάθεια υπολογισμού ενός αριθμού (nonce) για κάθε block με στόχο το hash του block με το nonce να αρχίζει από τόσα μηδενικά όσα καθορίζει η σταθερά “difficulty”. Μόλις τελειώσει η διαδικασία του mine πρέπει το block να μοιραστεί σε όλους του κόμβους. Οι υπόλοιποι κόμβοι με την σειρά τους θα ελέγξουν την ορθότητα του block και το αν μπορεί να συνδεθεί με την δικιά τους αλυσίδα. Στην περίπτωση κατά την οποία επιτύχουν και οι δύο έλεγχοι το νέο block προσαρτάται στην αλυσίδα του, σταματάνε την δικιά τους διαδικασία mine για τις συγκεκριμένες συναλλαγές και συνεχίζουν σε επόμενες που αναμένουν αν αυτές υπάρχουν. Αρκετές φορές ο έλεγχος συμβατότητας του νέου block με την υπάρχουσα αλυσίδα θα αποτύχει. Αυτό είναι μια ισχυρή ένδειξη ότι είτε ο αποστολέας είτε ο παραλήπτης του block δουλεύουν σε ένα branch της αλυσίδας και έχουν μείνει πίσω, και χρειάζεται συγχρονισμός των αλυσίδων. Στο δίκτυο noobchain ισχύει η λογική ότι η πιο μεγάλη σε μήκος αλυσίδα είναι και αυτή που είναι η σωστή, έτσι σε περίπτωση που απαιτηθεί συγχρονισμός, ο κόμβος που νομίζει ότι έχει το πρόβλημα ζητάει από

όλους τους υπόλοιπους την αλυσίδα τους και υιοθετεί σαν δικιά του αυτή με το μεγαλύτερο μήκος αφού πρώτα ελέγξει την σωστή της σύνδεση. Ολόκληρη η διαδικασία συνεχίζει επ'αόριστον με τους κόμβους να περιμένουν νέες συναλλαγές για να εκτελέσουν.

2.2 Εφαρμογή Client

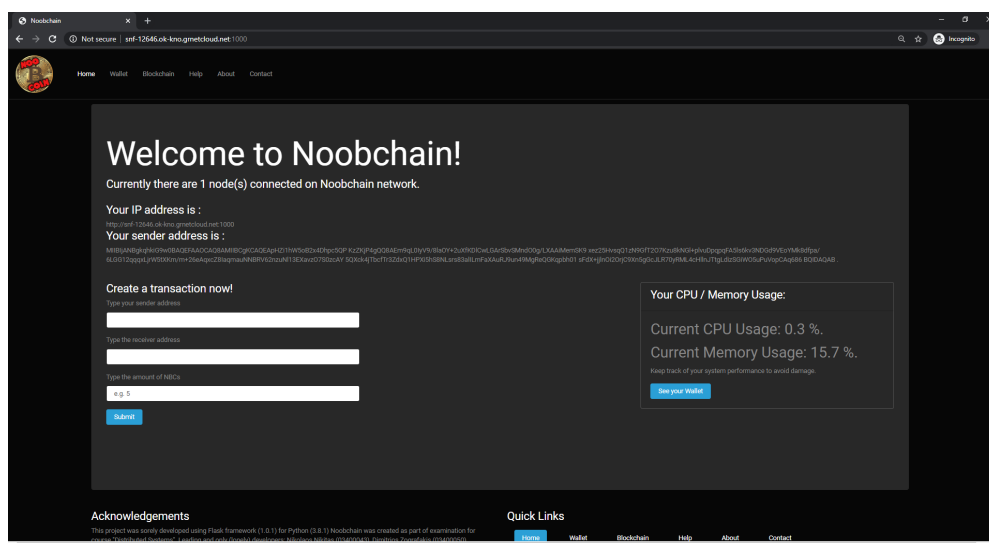
Η δημιουργία του Noobcash Client κατέστη δυνατή μέσω του κατάλληλου API της Flask που παρουσιάστηκε προηγουμένως. Για να είναι πιο ευχάριστη η εμπειρία για το χρήστη, αποφασίστηκε να δημιουργηθεί μία εφαρμογή που θα επισκέπτεται ο τελευταίος μέσω browser. Αναλυτικά, οι δυνατότητες που παρέχονται στο χρήστη είναι οι ακόλουθες:

Πίνακας 2.7: Λειτουργικότητα της Web εφαρμογής

Λειτουργικότητα	Σελίδα	Επεξήγηση
Create A Transaction	Home	Δημιουργία ενός transaction
View Transactions Made In Last Verified Block	Wallet	Εμφάνιση των transactions στο τελευταίο verified block της αλυσίδας
Balance Statistics	Wallet	Εμφάνιση στατιστικών του πορτοφολιού, όπως balance,
View Blockchain	Blockchain	Εμφάνιση πληροφοριών για κάθε μπλοκ από όλο το blockchain
CPU/RAM Usage	Home	Κατανάλωση CPU/RAM του συστήματος
Application's Info	About	Πληροφορίες για την Εφαρμογή
Help	Help	Επεξήγηση των λειτουργιών της Web εφαρμογής

Στη συνέχεια, παρουσιάζεται η διεπαφή της Web εφαρμογής

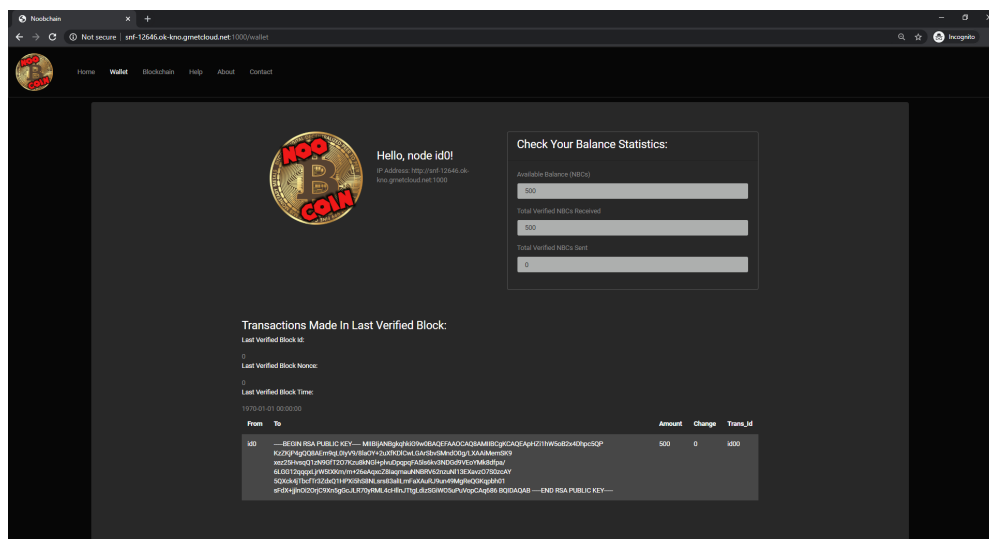
2.2.1 Σελίδα Home



Σχήμα 2.1: Σελίδα “Home”.

Η σελίδα “Home” αποτελεί την κεντρική σελίδα της διαδικτυακής εφαρμογής. Εδώ ο χρήστης, λαμβάνει σημαντικές πληροφορίες όπως το δημόσιο προφίλ του, την διεύθυνση στην οποία ακούει από τους άλλους κόμβους καθώς και μπορεί να δημιουργήσει συναλλαγές. Επίσης, είναι εφικτό να παρακολουθήσει χρήσιμες μετρικές της κατάστασης του υπολογιστή του, ιδιαίτερα χρήσιμο κατά την διαδικασία Mining.

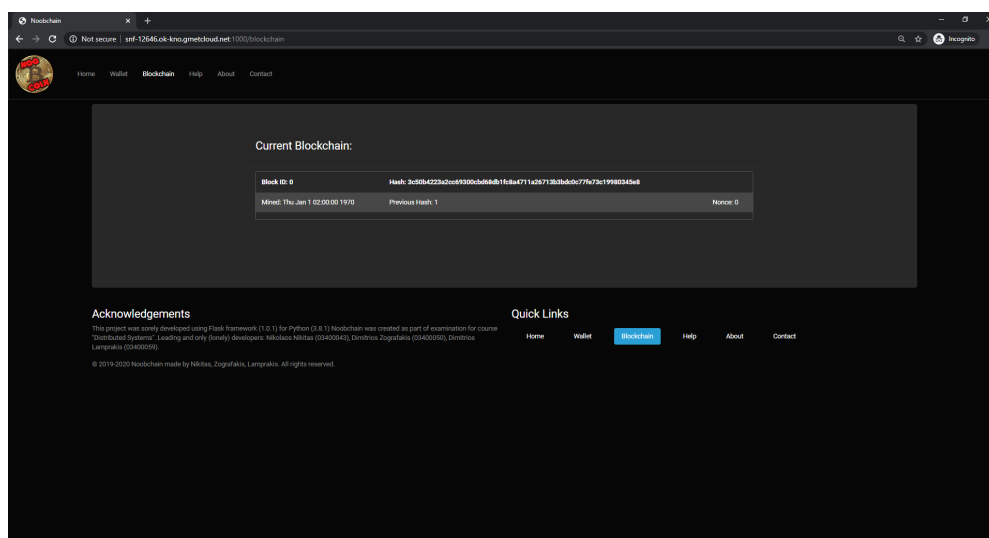
2.2.2 Σελίδα Wallet



Σχήμα 2.2: Σελίδα “Wallet”.

Η σελίδα “Wallet” εμφανίζει την τωρινή κατάσταση του πορτοφολιού του χρήστη. Αναλυτικότερα, προσφέρει γνώσεις ως προς το διαθέσιμο υπόλοιπο νομισμάτων, τα συνολικά νομίσματα που έχει μεταφέρει (από verified transactions) καθώς και αυτά που έχει λάβει (από verified transactions). Παρουσιάζονται, επιπλέον, το id του κόμβου, η διεύθυνση στην οποία ακούει από τους άλλους χρήστες, καθώς και υπάρχουν πληροφορίες όσον αφορά το τελευταίο verified Μπλοκ της αλυσίδας, το ποίο επαλήθευσε. Συγκεκριμένα, εμφανίζονται το πότε το επαλήθευσε, πόσα hashes χρειάστηκαν, καθώς και τα verified transactions που περιλαμβάνει.

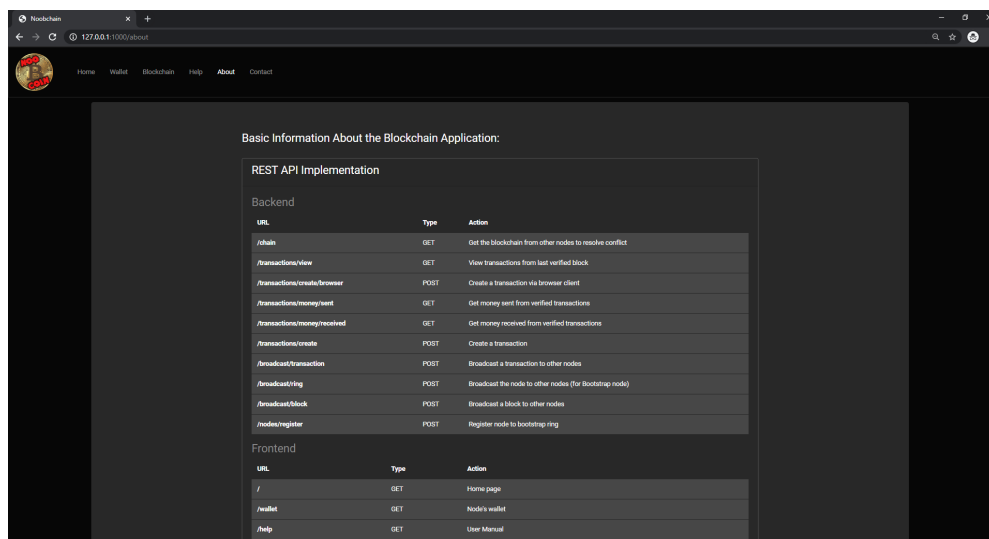
2.2.3 Σελίδα Blockchain



Σχήμα 2.3: Σελίδα “Blockchain”.

Η σελίδα “Blockchain” παραθέτει την τωρινή κατάσταση της αλυσίδας που έχει ο χρήστης. Συγκεκριμένα, κάθε μπλοκ παρουσιάζεται ως ένας πίνακας που περιέχει πληροφορίες σχετικά με τα πεδία του block. Επεξηγηματικά, εμφανίζεται ο αύξοντας αριθμός μπλοκ, το hash τωρινού μπλοκ, το hash προηγούμενου μπλοκ, η ημερομηνία δημιουργίας καθώς και ο αριθμός που δηλώνει το πλήθος των hashes που πραγματοποιήθηκαν για να προστεθεί το μπλοκ στην αλυσίδα.

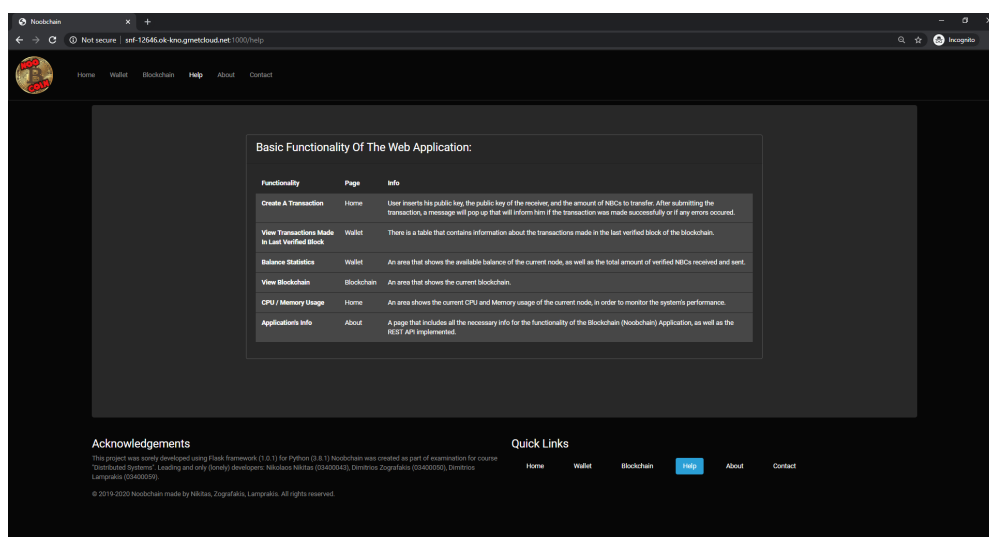
2.2.4 Σελίδα About



Σχήμα 2.4: Σελίδα “About”.

Στην σελίδα “About” εμφανίζονται σχόλια περί αρχιτεκτονικής της εφαρμογής, ενώ παρατίθενται όλες οι διαδρομές που αξιοποιεί η εφαρμογή με σύντομες περιγραφές.

2.2.5 Σελίδα Help



Σχήμα 2.5: Σελίδα “Help”.

Η σελίδα “Help”, ουσιαστικά, αποτελεί ένα μικρό τεχνικό εγχειρίδιο για τη χρήση της Web εφαρμογής.

2.3 Επιπλέον Υλοποιήσεις

Αποφασίστηκε να υλοποιηθεί και ένας διαφορετικός τρόπος ανάπτυξης και εκτέλεσης της συνολικής εφαρμογής, με στόχο την ύπαρξη αφαιρετικού επιπέδου όχι στο υλικό, που επιτυγχάνεται μέσω των Virtual Machines, αλλά στο επίπεδο του λειτουργικού συστήματος. Η υλοποίηση αυτή πραγματοποιείται μέσω της χρήσης των Docker Containers, παρέχοντας απομόνωση σε επίπεδο εφαρμογής, ταχύτερη ρύθμιση, χρήση λιγότερης μνήμης, μείωση της πολυπλοκότητας, καθώς και φορητότητα και κλιμακοσιμότητα. Με αυτόν τον τρόπο, έχοντας έτοιμη την βασική εφαρμογή, μέσω ελαχίστων εντολών μπορούμε να την εκκινήσουμε χωρίς καμία εξάρτηση σε υλικό, λειτουργικό, και βιβλιοθήκες εγκατεστημένες.

Παρέχεται λοιπόν η δυνατότητα για εκκίνηση ενός Noobcash Node σε περιβάλλον Docker Container μέσω του κατάλληλα ορισμένου Dockerfile αρχείου. Στο συγκεκριμένο αρχείο ορίζεται σαν βάση ένα λειτουργικό Debian Stretch με εγκατεστημένη Python, ορίζονται οι απαραίτητες μεταβλητές που αναφέρθηκαν προηγουμένως για την εκτέλεση του προγράμματος, μεταφέρονται τα αρχεία εντός του Docker Image, φορτώνονται η βιβλιοθήκη gcc καθώς και αυτές απ' το αρχείο requirements.txt, και τελικά αναμένεται η εκτέλεση του προγράμματος μέσω της τελικής εντολής Entrypoint.

Για να εκκινήσει το Docker Container (Bootstrap Node), απαιτούνται οι ακόλουθες εντολές:

```
1 # To run one node instance in docker container
2 docker build -t noobcash:latest .
3 sudo docker run -p 1000:1000 noobcash:latest
```

Μία άλλη δυνατή υλοποίηση που παρέχεται είναι η εκτέλεση 5 Noobcash Clients ταυτόχρονα σε διαφορετικά Docker Containers, με τις κατάλληλες παραμέτρους να έχουν οριστεί στο απαραίτητο αρχείο docker-compose.yml. Στο συγκεκριμένο αρχείο ορίζεται ένα υποδίκτυο μέσω του οποίου θα επικοινωνούν οι Containers, καθώς και τα 5 nodes σαν services. Για το καθένα από αυτά ορίζεται μία στατική IP απ' το υποδίκτυο, το port που θα μείνει ανοιχτό ώστε να βλέπει και ο εξωτερικός χρήστης την εφαρμογή, το μέγιστο μέγεθος μνήμης που θα μπορεί να αποκτήσει, καθώς και η τελική εντολή που θα ξεκινήσει την εφαρμογή.

Για να εκκινήσει το συνολικό σύστημα με τους Docker Containers (5 nodes για capacity 5 και difficulty 4), απαιτούνται οι ακόλουθες εντολές:

```
1 # To run 5 node instances in docker containers
2 docker-compose build
3 docker-compose up
```

Αξιολόγηση Συστήματος

Η αξιολόγηση του συστήματος επιτυγχάνεται με την πραγματοποίηση κάποιων πειραμάτων. Συγκεκριμένα, θα παρουσιαστούν αναλυτικότερα μετρικές για την απόδοση καθώς και την κλιμακωσιμότητα του συστήματος. Να το τονιστεί, προτού πραγματοποιηθεί οποιαδήποτε αξιολόγηση, ότι το σύστημα κάνει χρήση νημάτων (Threading) για να λειτουργήσει. Συνεπώς, σε μερικά σημεία, π.χ στην πραγματοποίηση των συναλλαγών (αυτών που δόθηκαν από εκφώνηση) το νήμα μπαίνει σε κατάσταση αδράνειας (sleep) με σκοπό να μην πλημμυρίζει το δίκτυο με συναλλαγές. Οι χρονικές στιγμές αδράνειας θα συμπεριληφθούν στον τελικό χρόνο διεκπεραίωσης των πειραμάτων, διότι έτσι κατασκευάστηκε το σύστημα, όμως είναι κάτι που πρέπει να αναφερθεί εδώ. Επιπλέον, αξίζει να αναφερθεί ότι, τελικά, όλες οι δοκιμές μαζί με τις μετρήσεις έλαβαν χώρα σε έναν υπολογιστή, και όχι σε VMs. Εφόσον υπάρχουν 2 threads που λειτουργούν παράλληλα (thread για mining και thread για διάβασμα και εκτέλεση των transactions), και δεν γίνεται η διαδικασία σειριακά (πρώτα να φορτώσουν όλα τα transactions και μετά να αρχίσει το mining), είναι λογικό τα αποτελέσματα να είναι ελαφρώς χειρότερα από τα ιδανικά.

3.1 Απόδοση

Για να μετρηθεί η απόδοση, εξετάζονται οι περιπτώσεις όπου 5 κόμβοι βρίσκονται συνδεδεμένοι στο δίκτυο και

- Η χωρητικότητα (capacity) του Block μεταβάλλεται στις τιμές 1, 5, 10.
- Η δυσκολία εύρεσης (difficulty) hash, λαμβάνει τιμές 4, 5.

Συσσωρεύοντας τις εν λόγω διατάξεις σε έναν πίνακα, και με μετρικές το throughput (transactions/sec) και Block time (χρόνος για mining ενός block (sec)) προκύπτει ότι:

Cients	Capacity	Difficulty	Total Time (min.)	Throughput	Block Time (sec.)
5	1	4	26	19.2	3.52
5	1	5	210	2.38	49.52
5	5	4	10	50	12.03
5	5	5	125	4	144.35
5	10	4	7	71.43	17.19
5	10	5	60	8.33	117.02

Πίνακας 3.1: Πειράματα (i).

3.2 Κλιμακωσιμότητα

Όσον αφορά την κλιμακωσιμότητα του συστήματος, θα επαναληφθούν τα ίδια πειράματα προκειμένου να δημιουργηθεί μία εικόνα, για το πως ανταποκρίνεται το σύστημα σε μεγαλύτερο φόρτο

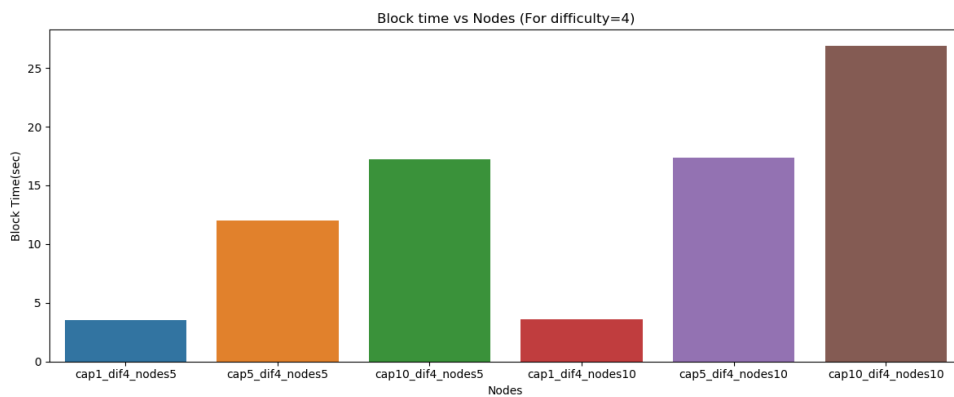
εργασίας. Έτσι, θέτοντας τους κόμβους ίσους με 10, για τις ίδιες τιμές των παραμέτρων Capacity, Difficulty, λαμβάνεται:

Cients	Capacity	Difficulty	Total Time (min.)	Throughput	Block Time (sec.)
10	1	4	34	29.41	3.59
10	1	5	701.4	1.43	91.347
10	5	4	26	38.46	17.32
10	5	5	320	3.13	237.19
10	10	4	20	50	26.91
10	10	5	300	3.33	260.56

Πίνακας 3.2: Πειράματα (ii).

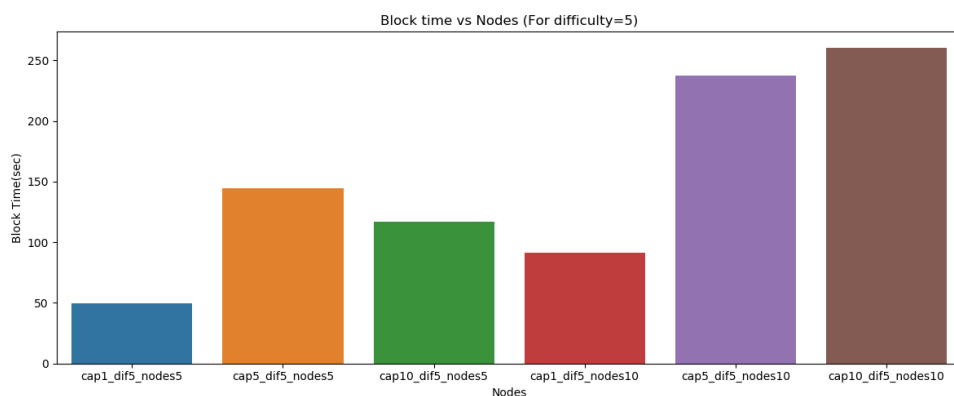
3.3 Γραφικές Παραστάσεις

Συνοπτικά, τα προηγούμενα αποτελέσματα απεικονίζονται στη συνέχεια. Όσον αφορά το block time, για difficulty=4 και μεταβαλλόμενο capacity για 5 και 10 nodes βλέπουμε τις παρακάτω γραφικές:



Σχήμα 3.1: Αξιολόγηση Συστήματος - Block Time για difficulty=4

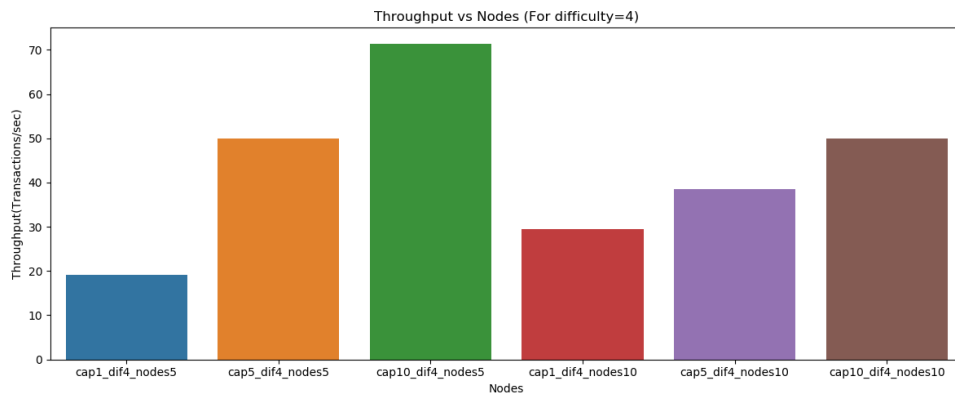
Για difficulty=5 και μεταβαλλόμενο capacity για 5 και 10 nodes βλέπουμε τις παρακάτω γραφικές:



Σχήμα 3.2: Αξιολόγηση Συστήματος - Block Time για difficulty=5

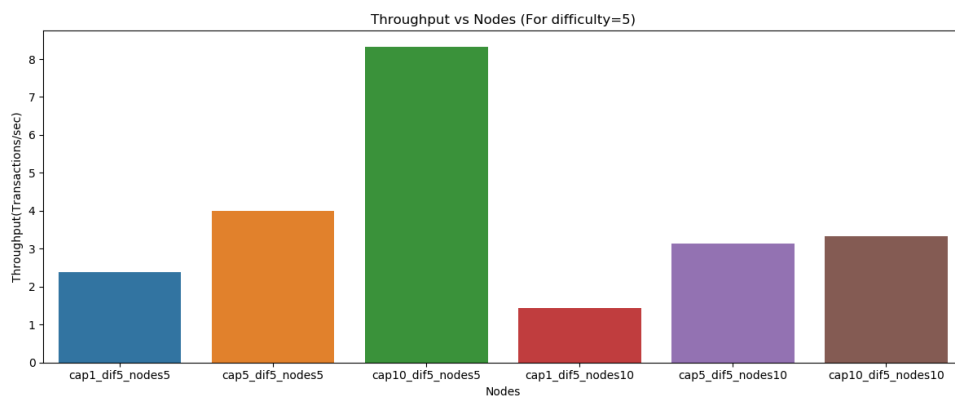
Παρατηρείται, λοιπόν, ότι φαίνεται να υπάρχει μεγαλύτερο latency για το mine ενός block με περισσότερο αριθμό nodes, κάτι που είναι λογικό λόγω μεγαλύτερης συμφώρησης του δικτύου, και άρα λιγότερης υπολογιστικής δύναμης διαθέσιμης για mining. Επιπλέον, με την αύξηση του capacity για κάθε block παρατηρείται ότι αυξάνεται σημαντικά το block time σχεδόν σε όλες τις περιπτώσεις, εκτός από την περίπτωση με 5 nodes και difficulty 5.

Όσον αφορά το throughput (transactions/sec), για difficulty=4 και μεταβαλλόμενο capacity για 5 και 10 nodes βλέπουμε τις παρακάτω γραφικές:



Σχήμα 3.3: Αξιολόγηση Συστήματος - Throughput για difficulty=4

Για difficulty=5 και μεταβαλλόμενο capacity για 5 και 10 nodes βλέπουμε τις παρακάτω γραφικές:



Σχήμα 3.4: Αξιολόγηση Συστήματος - Throughput για difficulty=5

Παρατηρείται, σε αυτήν την περίπτωση, να καθυστερεί λίγο περισσότερο το σύστημα για τα 5 nodes, σχετικά με την εξυπηρέτηση των transactions στη μονάδα του χρόνου, σε σχέση με τα 10 nodes. Επιπλέον, με την αύξηση του capacity για κάθε block παρατηρείται ότι αυξάνεται σημαντικά το throughput σε όλες τις περιπτώσεις.

3.4 Επεκτάσεις

Εφαρμογή Διαδικτύου

Όσον αφορά την διαδικτυακή εφαρμογή, υπάρχουν βλέψεις εγκατάστασης λειτουργίας για ενεργοποίηση και απενεργοποίηση του Mining με την χρήση κάποιου κουμπιού. Επιπρόσθετα, επιθυμητή αποτελεί η προβολή της χρήσης των πόρων του συστήματος με την μορφή Γραφικής Παράστασης στην κεντρική σελίδα. Τέλος, θα θέλαμε να συμπεριλάβουμε κάποια μορφή αποθήκευσης των δημοσίων κλειδιών των αποστολέων, προτάσεις συναλλαγών για τους κορυφαίους κόμβους που έχουν πραγματοποιήσει συναλλαγές, καθώς και αρχική διεπιφάνεια για διαδικασία ορθής σύνδεσης κάθε κόμβου, ώστε να είναι τα προσωπικά του δεδομένα ασφαλή.

Λειτουργικότητα

Από άποψη λειτουργικότητας, θα θέλαμε να δοκιμάσουμε σε δυνατότερα συστήματα την απόδοση της εφαρμογής, για μεγάλες τιμές difficulty, περισσότερους κόμβους, καθώς και μεγαλύτερες χωρητικότητες σε Blocks. Μια άλλη δοκιμή θα ήταν η υλοποίηση του συστήματος με χρήση TCP/UDP Sockets με στόχο την σύγκριση αυτών με την αρχιτεκτονική REST που εφαρμόστηκε, ως προς την απόδοσή τους. Τέλος, θα μπορούσαν να δοκιμαστούν τεχνικές με semaphores/locks και έλεγχο συστήματος για το εάν είναι πιο αποδοτικό.

Παράδειγμα Εκτέλεσης

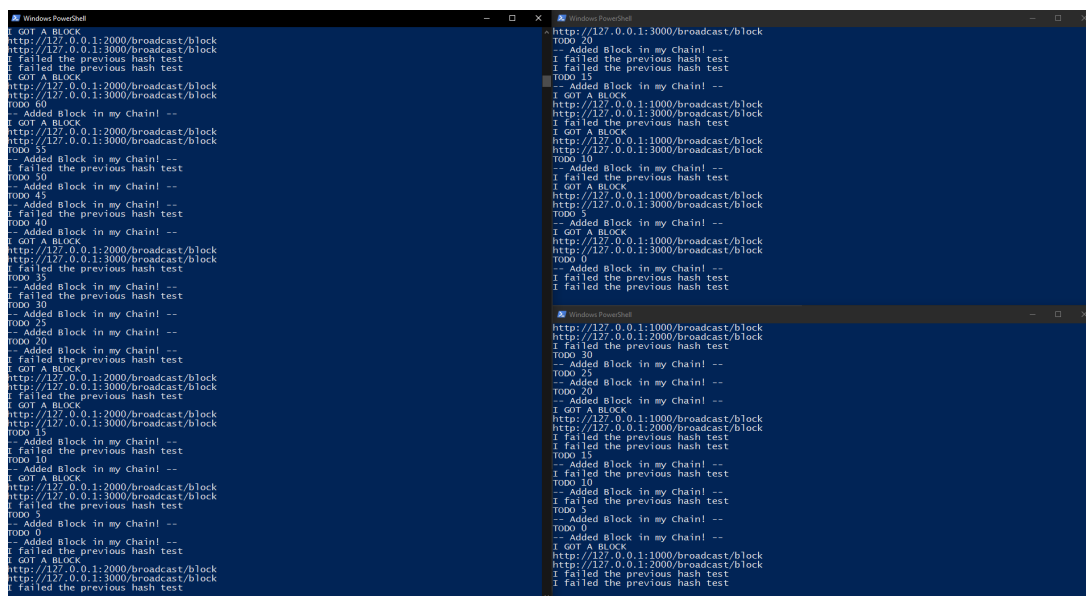
Παρατίθεται ένα παράδειγμα εκτέλεσης για 3 κόμβους, capacity 5 και difficulty 4.

Το παραπάνω εκτελείται εύκολα, ανοίγοντας 3 cmds στο directory του main αρχείου με τις ακόλουθες εντολές.

Τρέχουν από default στον localhost (127.0.0.1) και πόρτα 1000. Όπως έχει αναφερθεί, όλοι οι παράμετροι αυτοί μπορούν να κουρδιστούν, σε οποιαδήποτε τιμή επιθυμεί ο χρήστης.

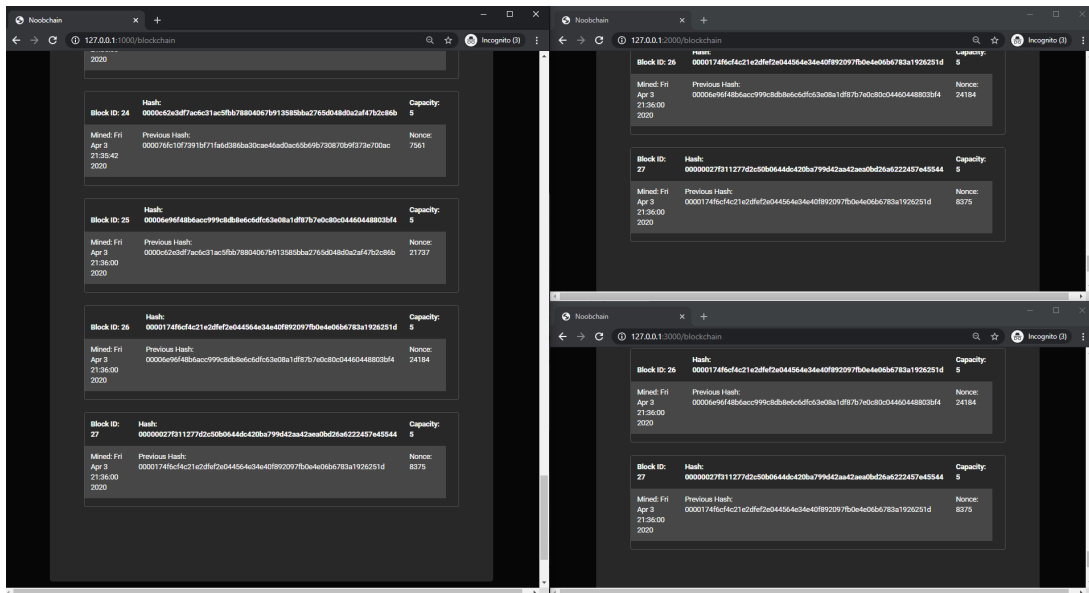
```
1 # Bootstrap Node
2 python main.py -bootstrap True -nodes 3 -cap 5 -dif 4
3
4 # 2nd Node
5 python main.py -p 2000 -nodes 3 -cap 5 -dif 4
6
7 # 3rd Node
8 python main.py -p 3000 -nodes 3 -cap 5 -dif 4
```

Μετά το πέρας της εκτέλεσης, λαμβάνουμε τα στιγμιότυπα από τα cmd.



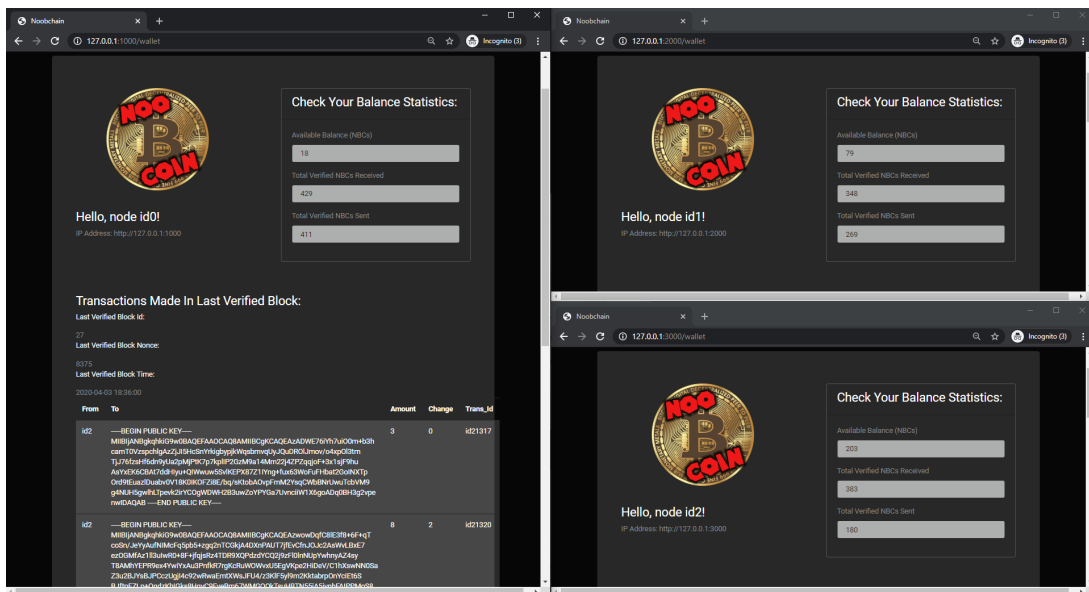
Σχήμα 4.1: Runtime (i): Στιγμιότυπο Terminals.

Ενώ από την σελίδα “/blockchain”, για κάθε ένα από τα ports, ώστε να επαληθεύσουμε ότι έχουν τις ίδιες αλυσίδες. Όπως φαίνεται από το παρακάτω στιγμιότυπο, όλοι οι κόμβοι έχουν συγχρονιστεί και έχουν την ίδια αλυσίδα. Οι κόμβοι περιμένουν, επιπλέον συναλλαγές για να συνεχίσουν το Mining. Οι συναλλαγές αυτές μπορούν να γίνουν είτε με μέθοδο post στην εφαρμογή είτε μέσω της Αρχικής Σελίδας.



Σχήμα 4.2: Runtime (ii): Στιγμιότυπο Σελίδας “Blockchain”.

Ενώ κάνοντας χρήση της σελίδας “/wallet”, προκύπτει ότι το Available Balance = Total Verified Received - Total Verified Sent, που είναι το επιθυμητό.



Σχήμα 4.3: Runtime (iii): Στιγμιότυπο Σελίδας “Wallet”.

Κατάλογος πινάκων

1.1	Frontend.	3
1.2	Backend.	3
2.1	Παράμετροι για την εκτέλεση της εφαρμογής.	4
2.2	Χαρακτηριστικά Αντικειμένου Node.	5
2.3	Χαρακτηριστικά Αντικειμένου Block.	7
2.4	Χαρακτηριστικά Αντικειμένου Blockchain.	7
2.5	Χαρακτηριστικά Αντικειμένου Wallet.	8
2.6	Χαρακτηριστικά Αντικειμένου Transaction.	9
2.7	Λειτουργικότητα της Web εφαρμογής	11
3.1	Πειράματα (i).	15
3.2	Πειράματα (ii).	16

Κατάλογος σχημάτων

1.1	Δομή Εργασίας.	2
2.1	Σελίδα “Home”.	11
2.2	Σελίδα “Wallet”.	12
2.3	Σελίδα “Blockchain”.	12
2.4	Σελίδα “About”.	13
2.5	Σελίδα “Help”.	13
3.1	Αξιολόγηση Συστήματος - Block Time για difficulty=4	16
3.2	Αξιολόγηση Συστήματος - Block Time για difficulty=5	16
3.3	Αξιολόγηση Συστήματος - Throughput για difficulty=4	17
3.4	Αξιολόγηση Συστήματος - Throughput για difficulty=5	17
4.1	Runtime (i): Στιγμιότυπο Terminals.	19
4.2	Runtime (ii): Στιγμιότυπο Σελίδας “Blockchain”.	20
4.3	Runtime (iii): Στιγμιότυπο Σελίδας “Wallet”.	20