ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

Τίτλος Εργασίας

ΥΛΟΠΟΙΗΣΗ ΜΙΑΣ WEB ΕΦΑΡΜΟΓΗΣ ΜΕ ΤΗ ΧΡΗΣΗ ΤΗΣ ΤΕΧΝΟΛΟΓΙΑΣ BLOCKCHAIN

Όνομα φοιτητή

ΑΝΤΩΝΙΟΥ ΑΣΗΜΑΚΗΣ

Αθήνα, 2020

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

Τριμελής Εξεταστική Επιτροπή

ΤΣΑΔΗΜΑΣ ΑΝΑΡΓΥΡΟΣ
Μέλος Ε.ΔΙ.Π, Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

ΑΝΑΓΝΩΣΤΟΠΟΥΛΟΣ ΔΗΜΟΣΘΕΝΗΣ
Καθηγητής, Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

ΝΙΚΟΛΑΪΔΟΥ ΜΑΡΑ
Καθηγήτρια, Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

Ο ΑΝΤΩΝΙΟΥ ΑΣΗΜΑΚΗΣ

δηλώνω υπεύθυνα ότι:

1) Είμαι ο κάτοχος των πνευματικών δικαιωμάτων της πρωτότυπης αυτής εργασίας και από όσο γνωρίζω η εργασία μου δε συκοφαντεί πρόσωπα, ούτε προσβάλει τα πνευματικά δικαιώματα τρίτων.

2) Αποδέχομαι ότι η ΒΚΠ μπορεί, χωρίς να αλλάξει το περιεχόμενο της εργασίας μου, να τη διαθέσει σε ηλεκτρονική μορφή μέσα από τη ψηφιακή Βιβλιοθήκη της, να την αντιγράψει σε οποιοδήποτε μέσο ή/και σε οποιοδήποτε μορφότυπο καθώς και να κρατά περισσότερα από ένα αντίγραφα για λόγους συντήρησης και ασφάλειας.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Τσαδήμα Ανάργυρο για όλη τη βοήθεια και συνεχή καθοδήγηση καθόλη τη διάρκεια προετοιμασίας της πτυχιακής μου εργασίας με θέμα «Ανάπτυξη ενός Web Service με την χρήση τεχνολογίας του Blockchain».

Επιπλέον, θα ήθελα να ευχαριστήσω την οικογένεια και τους φίλους μου που με στήριξαν σε όλο αυτό το διάστημα της φοίτησης μου για να μπορώ να επικεντρωθώ στους στόχους μου και στην σχολή μου.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Contents

ΠΕΡΙΛΗΨΗ

Σκοπός αυτής της εργασίας είναι η περιγραφή, η κατανόηση και η ανάλυση της τεχνολογίας Blockchain και κυρίως του framework που χρησιμοποιήθηκε που είναι το Hyperledger Fabric καθώς και η σύνδεσή του με τη διαδικτυακή εφαρμογή που δημιουργήθηκε. Αρχικά, γίνεται μια εκτενέστερη ανάλυση γύρω από την τεχνολογία του Blockchain και πως αυτή δουλεύει. Γίνεται αναφορά στις υπόλοιπες υλοποιήσεις που υπάρχουν και εξηγείται γιατί επικράτησε το Hyperledger Fabric. Επίσης, παρουσιάζονται τα χαρακτηριστικά του και η διαφορές που έχουν με τις απλές βάσεις δεδομένων και στη συνέχεια γίνεται μια περιγραφή τόσο τον προ απαιτούμενων που χρειάζονται, όσο και την εγκατάσταση του Hyperledger Fabric. Επιπρόσθετα, γίνεται μια αναλυτική περιγραφή του σεναρίου που ακολουθήθηκε ώστε να δημιουργηθεί η διαδικτυακή εφαρμογή που ασχολείται με το κλείσιμο ραντεβού σε ένα ΚΤΕΟ και η αλληλεπίδραση που έχει αυτή με το Blockchain. Τέλος, δείχνεται η υλοποίηση της εφαρμογής και τι δυνατότητες παρέχονται σε κάθε χρήστη.

Λέξεις κλειδιά: Hyperledger Fabric, Blockchain, Διαδικτυακή εφαρμογή, ΚΤΕΟ

ABSTRACT

The aim of this project is to describe, understand and analyze the Blockchain technology and mainly of the framework that is used, which is the Hyperledger Fabric and its connection with the web service that is created. Firstly, a more detailed analysis is made a bit Blockchain technology and how it works. A reference has been made to the other implementations that exist and explain why the Hyperledger Fabric has dominated. Also presented are its features and the differences they have with the simple databases and then a description of both of the prerequisited that are needed as well as the installation of Hyperledger Fabric. Additionally, there is a detailed description of the scenario that has been followed to create the web application that deals with closing an appointment at a KTEO and the interaction that it has with the Blockchain. Finally, the implementation of the application is shown and what features are available to each user.

Keywords: Hyperledger Fabric, Blockchain, Web Service

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ

DLT	Distributed Ledger Technology
LTS	Long Term Support
VIM	VI iMproved
CA	Certificate Authority
PoET	Proof of elapsed Time
MSP	Membership Service Provider

1 ΕΙΣΑΓΩΓΗ

1.1 ΑΝΤΙΚΕΙΜΕΝΟ ΕΡΓΑΣΙΑΣ

Αντικείμενο της εργασίας αποτελεί η δημιουργία μιας διαδικτυακής (Web) εφαρμογής, η οποία κάνει χρήση της τεχνολογίας του Blockchain. Η ιδέα της εφαρμογής είναι να αναπτυχθεί ένα Web service το οποίο θα έχει το ρόλο της διαχείρισης ραντεβού σε ένα ΚΤΕΟ. Παράλληλα, θα έχει την ικανότητα να επικοινωνεί με το Blockchain, που θα αναπτυχθεί στη συνέχεια και θα αλληλεπιδρά με αυτό μέσα από την διαδικτυακή εφαρμογή. Θα μπορεί να αναζητήσει καταχωρήσεις, να κάνει καινούργιες καταχωρήσεις και να τροποποιεί ορισμένα στοιχεία αυτών των καταχωρήσεων. Στη συνέχεια, παρουσιάζεται η ανασκόπηση της πορείας του Blockchain αλλά και των κρυπτονομισμάτων, καθώς και το μέλλον της τεχνολογίας αυτής.

Το Bitcoin είναι μια λύση που αναπτύχθηκε με αφορμή την οικονομική κρίση του 2008, η οποία καθόρισε τις εξελίξεις στην οικονομία έως και σήμερα. Η ιδέα αυτή θεμελιώθηκε λόγω της ραγδαίας ανάπτυξης μιας τεχνολογίας που θα απελευθέρωνε τις συναλλαγές από τον έλεγχο τρίτων, δηλαδή των τραπεζών και των κυβερνήσεων, δίνοντας έτσι τον έλεγχο πίσω στους πολίτες. Η δυσκολία της τεχνολογίας αυτής είναι η προσέγγιση του κοινού πέρα από τις νεαρές ηλικίες, που δεν έχει εμπειρία από επενδύσεις.

Συνοπτικά σκοπός της τεχνολογίας αυτής είναι η τήρηση του μητρώου των δεδομένων, καθώς επίσης και η εμπιστοσύνη που θα υπάρχει σε αυτά, διότι όλες οι συναλλαγές είναι καταγεγραμμένες και είναι εξαιρετικά δύσκολο να τροποποιηθούν. Ένα απλό σενάριο χρήσης της τεχνολογίας αυτής είναι η αποθήκευση όλων των εγγράφων μιας εταιρείας στο μητρώο του Blockchain. Με αυτό συνεπάγεται πως ο έλεγχος της εταιρείας θα μπορεί να πραγματοποιηθεί πλήρως αυτοματοποιημένα, με ακρίβεια και ασφάλεια.

1.2 ΔΙΑΡΘΡΩΣΗ ΕΡΓΑΣΙΑΣ

Ο τρόπος διάρθρωσης της εργασίας, πραγματοποιείται με τον ακόλουθο τρόπο: Αρχικά, γίνεται μια παρουσίαση του απαραίτητου θεωρητικού

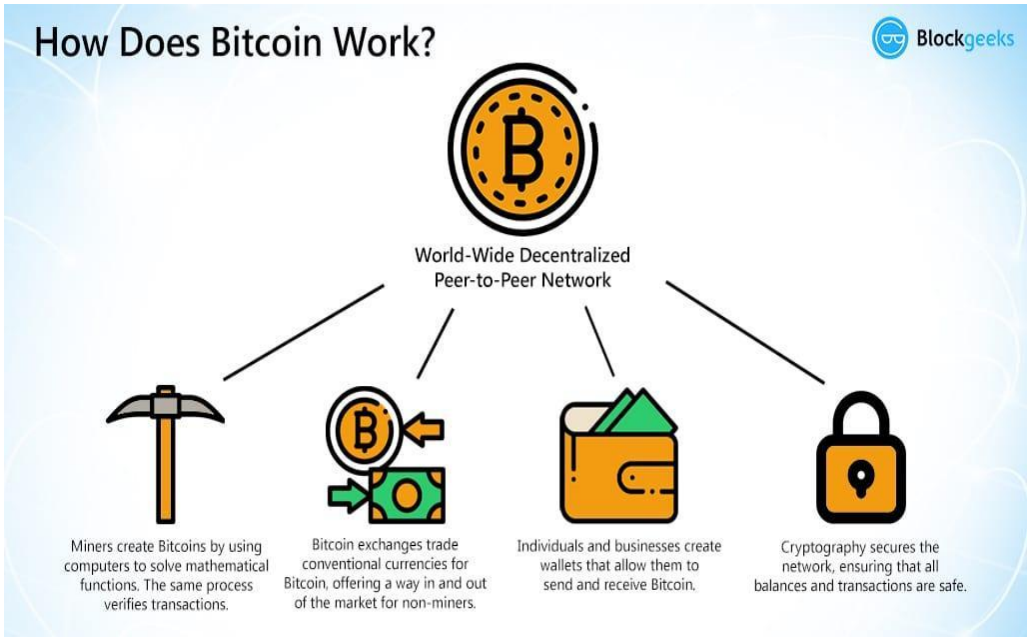
υποβάθρου για τη τεχνολογία του Blockchain. Στο τρίτο κεφάλαιο, πραγματοποιείται μια πιο αναλυτική προσέγγιση για το λογισμικό που χρησιμοποιήθηκε, Hyperledger Fabric, καθώς αναλύονται τα χαρακτηριστικά του, η αρχιτεκτονική και η διαφορά με τις βάσεις δεδομένων. Στο τέταρτο κεφάλαιο γίνεται η ανάλυση του σεναρίου που πραγματοποιήθηκε για να δημιουργηθεί μια εφαρμογή όπου θα κάνει χρήση της τεχνολογίας του Hyperledger Fabric. Στο πέμπτο κεφάλαιο παρουσιάζεται αναλυτική εγκατάσταση των προ απαιτούμενων λογισμικών ώστε να μπορεί να αναπτυχθεί η εφαρμογή. Τέλος, στο έκτο κεφάλαιο, παρουσιάζεται η υλοποίηση της εφαρμογής τόσο σε επίπεδο frontend όσο και στο backend, δείχνοντας αναλυτικά πως δημιουργήθηκε η εφαρμογή και πως έγινε η σύνδεση της εφαρμογής με το Blockchain.

2 ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

2.1 BLOCKCHAIN

Λόγω της ραγδαίας εξέλιξης της τεχνολογίας, της ανοδικής πορείας του Bitcoin και άλλων κρυπτονομισμάτων, όπως για παράδειγμα το Ethereum και έχουν πυροδοτήσει πολλές συζητήσεις πάνω στην οικονομία και την πληροφορική γύρω από αυτά τα θέματα.

Το Bitcoin είναι από τα πρώτα κρυπτονομίσματα που χρησιμοποίησαν την τεχνολογία του Blockchain. Συχνά περιγράφεται και σαν εικονικό νόμισμα ή ψηφιακό, πλέον είναι ένας τύπος νομίσματος πλήρως ψηφιακός. Πρόκειται για έναν τύπο διαδικτυακού νομίσματος που με αυτό μπορείς να αγοράσεις προϊόντα και υπηρεσίες από όσα καταστήματα δέχονται πληρωμή με Bitcoin. Για να αποκτηθεί ένα bitcoin μπορεί να αγοραστεί με κανονικά χρήματα ή αλλιώς μπορεί κάποιος να αναθέσει στον υπολογιστή του να δουλέψει για την επίλυση πολύ δύσκολων και πολύπλοκων πράξεων. Έτσι, ανταμείβονται με bitcoin. Αυτός ο τρόπος ονομάζεται mining.

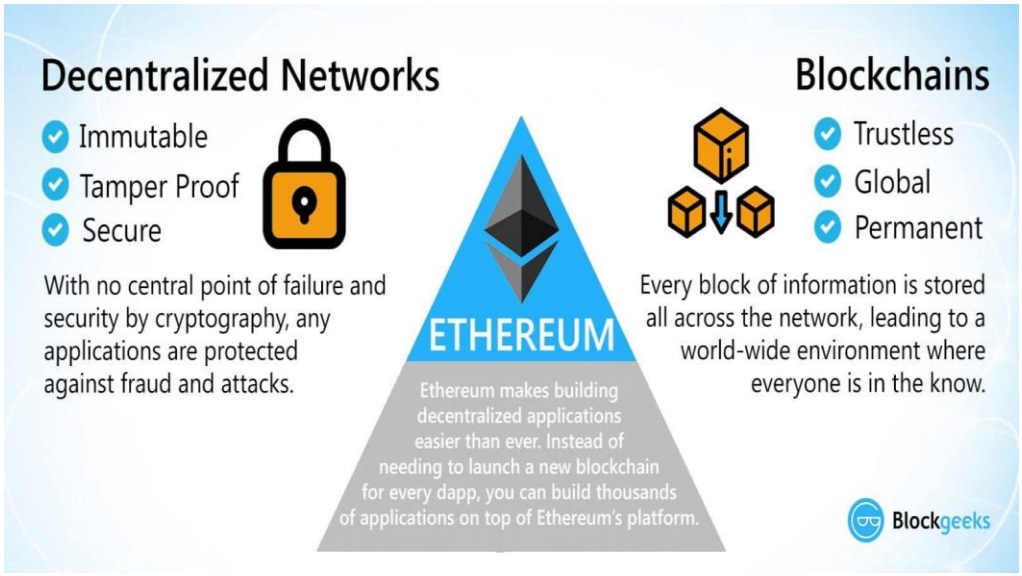


Εικόνα 1: How Bitcoin Works.

Το bitcoin προσφέρει μόνο μια συγκεκριμένη εφαρμογή της τεχνολογίας του Blockchain, ένα σύστημα ηλεκτρονικού χρήματος που βασίζεται στο peer to

peer τεχνολογία και επιτρέπει αγορές στο διαδίκτυο. Από την άλλη, το Ethereum εστιάζει στην ανάπτυξη οποιαδήποτε αποκεντρωμένης εφαρμογής.

Το Ethereum είναι ένα ανοιχτού κώδικα λογισμικό που βασίζεται στην τεχνολογία του Blockchain και επιτρέπει στους προγραμματιστές να δημιουργήσουν και να αναπτύσσουν αποκεντρωμένες εφαρμογές διαφόρων τύπων. Το Ethereum δίνει τη δυνατότητα στους προγραμματιστές να φτιάξουν τις εφαρμογές τους στην κορυφή της Ethereum πλατφόρμας. Δηλαδή, δεν χρειάζεται να «δημιουργήσουν» ο καθένας το δικό του blockchain με τον κώδικα που δίνει η εταιρία. Στην ουσία δίνει τη δυνατότητα στον καθένα να εντάξει στο project του ή οπουδήποτε την δυνατότητα των συναλλαγών μέσω smart contract. Στην ουσία το Ethereum θεμελίωσε την έννοια μιας smart contract εφαρμογής που σου προσφέρει είτε την εξαργύρωση χρημάτων ή οτιδήποτε έχει αξία προσφέροντας διαφάνεια χωρίς την παρουσία διαμεσολαβητή.



Εικόνα 2: How Ethereum Works.

Ένα καίριο ερώτημα, το οποίο οφείλει να απαντηθεί είναι το πως ακριβώς λειτουργεί η τεχνολογία του Blockchain. Blockchain είναι ένας αποκεντρωμένος, κατανεμημένος και δημόσιος ψηφιακός κατάλογος (ledger) που χρησιμοποιείται για την καταγραφή συναλλαγών σε πολλούς υπολογιστές έτσι ώστε κάθε εγγραφή που εμπλέκεται να μην μπορεί να

τροποποιηθεί αναδρομικά χωρίς την αλλαγή όλων των επόμενων block. Αυτό επιτρέπει στους συμμετέχοντες να επαληθεύουν και να ελέγχουν τις συναλλαγές ανεξάρτητα και σχετικά ανέξοδα. Με λίγα λόγια θα μπορούσαμε να πούμε ότι η τεχνολογία Blockchain βασίζεται σε τρεις θεμελιώδεις πυλώνες που την βοηθούν να κερδίσει ευρεία αναγνώριση έναντι των υπόλοιπων συστημάτων και αυτοί είναι η αποκέντρωση, η διαφάνεια και η αμεταβλητότητα του συστήματος. (Παναγόπουλος, 2019)

Πρόκειται ουσιαστικά για μια σειρά καταχωρίσεων που αφορούν συναλλαγές σε ένα δημόσιο ledger (κατάστιχο). Κάθε καινούργια ομάδα καταχωρήσεων – ένα block δηλαδή – συνδέεται με τα προηγούμενα, δημιουργώντας μια αλυσίδα καταχωρίσεων, δηλαδή ένα Blockchain. Τα blocks συνδέονται μονοσήμαντα μεταξύ τους και κατ'αυτόν τον τρόπο, το blockchain λειτουργεί ως ένα αποκεντρωμένο (decentralized) λογιστικό, το οποίο είναι κοινό για όλους τους συμμετέχοντες, μιας και όλοι όσοι συμμετέχουν σε αυτό αποθηκεύουν ένα αντίγραφο του. Έτσι, εξασφαλίζεται η ασφάλεια και η διαφάνεια των συναλλαγών. Η διαφορά με τις μέχρι πρότινος συναλλαγές είναι ότι πλέον δεν είναι απαραίτητη η ύπαρξη μιας ενδιάμεσης έμπιστης αρχής – όπως αποτελεί για παράδειγμα η τράπεζα – ενώ η εμπιστοσύνη των συναλλασσόμενων μερών βασίζεται σε αλγοριθμική επιβεβαίωση.

Η τεχνολογία σύμφωνα με τον Παναγόπουλο (2019), βρίσκεται ακόμα σε πρώιμο στάδιο της ανάπτυξης του. Η πρώτη φάση είναι η φάση που δημιουργούνται τα κρυπτονομίσματα και το Bitcoin είναι το πιο σημαντικό από αυτά.

Στη δεύτερη φάση του Blockchain, υποστηρίζεται πια και η δημιουργία προηγμένων έξυπνων συμβολαίων (Smart Contracts) για επιτεύξιμα προγράμματα και εντολές που επεκτείνουν σταδιακά την περιοχή και το πεδίο εφαρμογής του. Αυτή η φάση επεκτείνει την εφαρμογή Blockchain σε διαφορετικές βιομηχανίες και κάνει εφικτή τη συνεργασία μεταξύ τους. Η

υιοθέτηση τεχνολογίας Blockchain όχι μόνο επιλύει το πρόβλημα της εμπιστοσύνης, αλλά και επιτρέπει την όλο και πιο αυτοματοποιημένη κατανομή πόρων σε παγκόσμια κλίμακα. Σε αυτή τη φάση, το έξυπνο συμβόλαιο έχει χρησιμοποιηθεί και ενσωματωθεί στο σύστημα Blockchain για την αντιμετώπιση των ζητημάτων αμοιβαίας εμπιστοσύνης και ταυτότητας μεταξύ των κόμβων του δικτύου. Συγκεκριμένα, το έργο Hyperledger είναι μία από τις δημοφιλείς υποδομές Blockchain που συνδέονται με την έξυπνη σύμβαση και την εξουσιοδοτημένη αρχή.

Στην επόμενη γενιά οι πτυχές που σχετίζονται με το Blockchain δεν θα επηρεάσουν μόνο την ανθρώπινη ιδεολογία αλλά εν γένει την κοινωνία. Οι κατανεμημένες εφαρμογές συστημάτων τεχνητής νοημοσύνης, όπως η Αποκεντρωμένη Εφαρμογή (Dapp- Decentralized Application) η Αποκεντρωμένη Αυτόνομη Οργάνωση (DAODecentralized Autonomous Organization), η Αποκεντρωμένη Αυτόνομη Εταιρεία (DAC-Decentralized Autonomous Corporation), αρχίζουν να εμφανίζονται. Επίσης τα τελευταία χρόνια έχουν διατυπωθεί πολύ ενδιαφέρουσες ιδέες για εφαρμογή της Blockchain τεχνολογίας σε ποικίλα επιστημονικά πεδία αλλά και στην εξέλιξη “αμφιλεγόμενων” διαδικασιών με τις οποίες ερχόμαστε αντιμέτωποι στην καθημερινότητά μας, όπως για παράδειγμα στη διεξαγωγή μιας ψηφοφορίας χωρίς νοθεία και με αποδοχή του αποτελέσματος από όλους τους συμμετέχοντες.

2.2 FRAMEWORKS

Υπάρχουν ποικίλες υλοποιήσεις για την ανάπτυξη ενός Blockchain. Οι επικρατέστερες ήταν το Framework του Ethereum, το Hyperledger Sawtooth και το Hyperledger Fabric. Τα βασικά στοιχεία αυτών είναι:

Η βασική ιδέα του Ethereum είναι η δημιουργία ενός γενικού σκοπού blockchain στο οποίο οι χρήστες μπορούν να αναπτύξουν τις δικές τους επιχειρηματικές εφαρμογές και να μην χρειάζεται να περιοριστούν μόνο στην μεταφορά κρυπτονομισμάτων. Ο προγραμματιστής μπορεί να υλοποιήσει

έξυπνα συμβόλαια (Smart Contracts) που γράφονται στη γλώσσα προγραμματισμού Solidity και να τα τρέξει στο blockchain που του παρέχεται. Μειονέκτημα του Ethereum είναι πως παρέχεται ήδη το blockchain, οπότε η ευελιξία του προγραμματιστή στη δομή του blockchain είναι περιορισμένη. Επίσης, πρόκειται για ένα permissionless blockchain και η επίτευξη των smart contracts βασίζονται στο ether που είναι το κρυπτονόμισμα του Ethereum και αποκτείτε με την επίλυση αλγορίθμων.

Από την άλλη υπάρχει το Hyperledger Sawtooth που πρόκειται για μια modular πλατφόρμα για την κατασκευή, την ανάπτυξη και την λειτουργία κατανεμημένων καταλόγων (Distributed Ledgers). Το Sawtooth διαθέτει καινούργια χαρακτηριστικά από το Ethereum, τόσο στην συναίνεση των μελών όταν επιτυγχάνεται μια συμφωνία (transaction), όσο και στους αλγόριθμους που χρησιμοποιεί. Οι οποίοι είναι ο proof of elapsed time (PoET) ο οποίος εμποδίζει την χρήση υψηλών πόρων και την υψηλή κατανάλωση ενέργειας και διατηρεί τη διαδικασία πιο αποτελεσματική.

Τέλος, υπάρχει και το Hyperledger Fabric το οποίο είναι μια πλατφόρμα κατασκευής κατανεμημένων καταλόγων (Distributed Ledger) και παρέχει μια modular αρχιτεκτονική η οποία προσφέρει υψηλό βαθμό εμπιστευτικότητας, ευελιξίας και δυνατότητα κλιμάκωσης. Επίσης, ο χρήστης έχει τη δυνατότητα να προγραμματίσει ακριβώς όπως θέλει πολλά σημεία, όπως σε επίπεδο ασφαλείας ή σε επίπεδο αλγορίθμων συναίνεσης. Έτσι, το Hyperledger Fabric μπορεί να προσαρμοστεί πιο εύκολα σε κάθε επιχειρησιακό δίκτυο.

Λαμβάνοντας υπόψη όλα τα παραπάνω η υλοποίηση που χρησιμοποιήθηκε για την ανάπτυξη του blockchain είναι το Hyperledger Fabric.

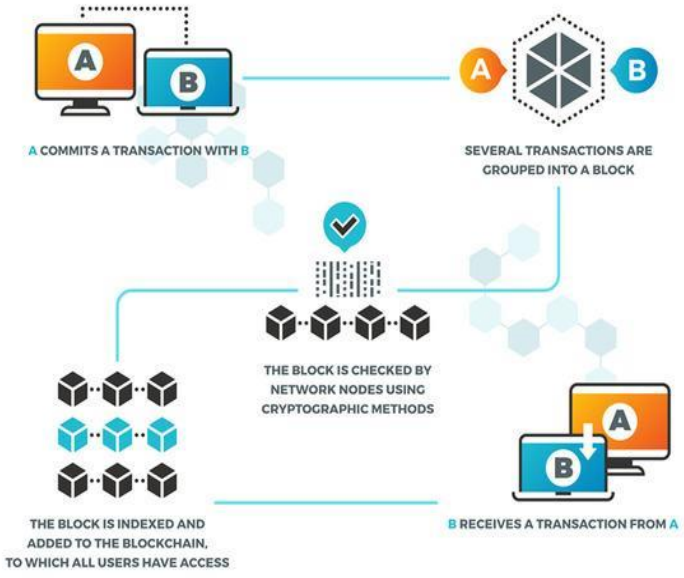
3 Hyperledger Fabric

3.1 Εισαγωγη

Το Hyperledger Fabric αποτελεί μια πλατφόρμα τεχνολογίας κατανεμημένου λογιστικού διαύλου (DLT), ανοικτού κώδικα. Η παρούσα τεχνολογία έχει σχεδιαστεί για χρήση σε επιχειρηματικά περιβάλλοντα και προσφέρει ορισμένες βασικές δυνατότητες διαφοροποίησης σε σχέση με άλλες δημοφιλείς πλατφόρμες κατανεμημένων καταλόγων ή Blockchain.

Βασικό σημείο διαφοροποίησης αποτελεί το γεγονός ότι το Hyperledger Fabric ιδρύθηκε και συστεγάστηκε στο πλαίσιο των βασικών αρχών του Linux Foundation μέχρι το 2015. Το Hyperledger Fabric κάνει τα επαγγελματικά δίκτυα (Business Networks) και τις συναλλαγές (Transactions) πιο αποδοτικές.

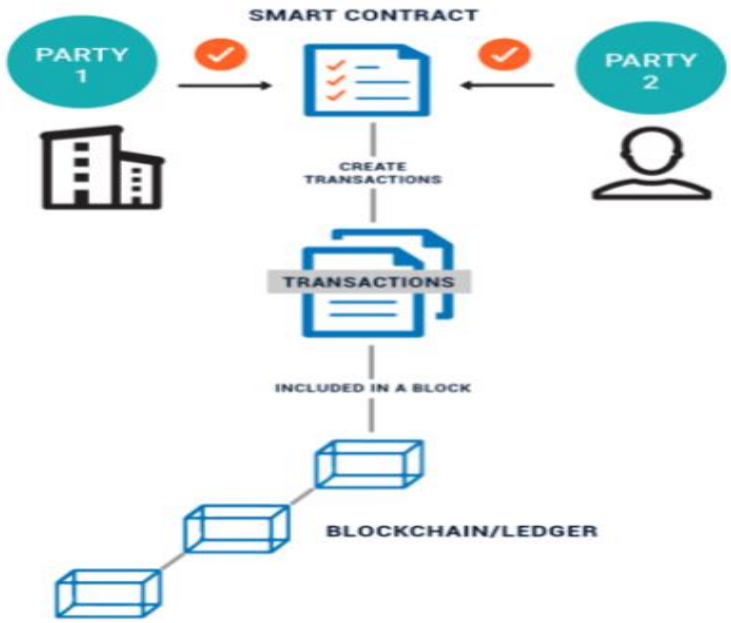
Με την έννοια «συναλλαγή» (Transaction) γίνεται αναφορά στην καταγραφή ενός γεγονότος, που είναι ασφαλισμένο κρυπτογραφημένα με μια ψηφιακή υπογραφή, η οποία είναι πιστοποιημένη, διατεταγμένη και δεσμευμένη μέσα σε ένα τμήμα – block.



Εικόνα 3: How Transaction Works.

Επιπροσθέτως, το Fabric αποτελεί μια κατανεμημένη πλατφόρμα, η οποία υποστηρίζει Smart Contracts. Ένα Smart Contract είναι ένα πρωτόκολλο, του οποίου σκοπός του είναι να διευκολύνει ψηφιακά την επαλήθευση ή την επιβολή της τήρησης των συμφωνηθέντων όρων σε μια σύμβαση. Αυτό σημαίνει πως επιτρέπουν την εκτέλεση αξιόπιστων συμβάσεων χωρίς να εμπλέκεται κάποιος τρίτος που να επιτηρεί. Οι συναλλαγές αυτές είναι παρακολουθήσιμες και μη αναστρέψιμες.

Πρόκειται για δίκτυο στο οποίο ο χρήστης χρειάζεται άδεια προκειμένου να ενταχθεί σε αυτό (permission). Δηλαδή, σε σχέση με τα κοινά δημόσια δίκτυα, οι συμμετέχοντες γνωρίζονται – ψηφιακά μεταξύ τους, από το τελείως ανώνυμο και μη αξιόπιστο δίκτυο. Αυτό γίνεται καθώς το Hyperledger Fabric παρέχει μια υπηρεσία η οποία διαχειρίζεται όλες τις ταυτότητες (ID's) των μελών και επαληθεύει όλους τους συμμετέχοντες στο δίκτυο(Membership Service Provider, MSP).



Εικόνα 4: Blockchain and Smart Contract Diagram.

3.2 Αρχιτεκτονική

Το Hyperledger Fabric διαθέτει υψηλά ευέλικτη και ρυθμιζόμενη αρχιτεκτονική, η οποία επιτρέπει στο χρήστη, καινοτομία, ευελιξία και

βελτιστοποίηση, για ένα μεγάλο φάσμα χρήσεων πάνω στη βιομηχανία. Ενδεικτικό παράδειγμα αποτελεί, ένα τραπεζικό, χρηματοοικονομικό, ασφαλιστικό, υγειονομικής περίθαλψης, ανθρώπινου δυναμικού συστήματος ακόμη και ψηφιακής μουσικής.

Επιπροσθέτως, το Hyperledger Fabric έχει σχεδιαστεί ώστε να έχει μια αρθρωτή αρχιτεκτονική και διαθέτει ποικιλία εργαλείων που συνδέονται εύκολα όπως πρωτόκολλα για την διαχείριση ταυτοτήτων (identity management), πρωτόκολλα για την διαχείριση κλειδιών (key management protocols) όπως επίσης και διάφορες βιβλιοθήκες (cryptographic libraries). Η πλατφόρμα είναι σχεδιασμένη για να ικανοποιεί μια μεγάλη γκάμα των επιχειρησιακών απαιτήσεων ανάλογα τη χρήση.

Αξίζει να τονιστεί πως το Fabric μπορεί να ρυθμιστεί ώστε να έχει:

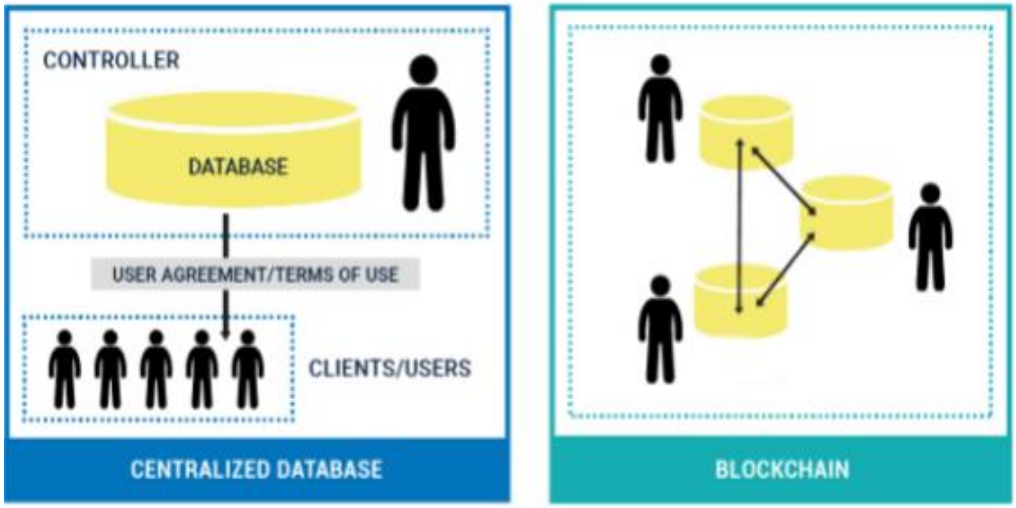
- Εφαρμογή επικύρωσης και επιβολής των κανονισμών η οποία μπορεί να σχεδιαστεί διαφορετικά σε κάθε είδος εφαρμογής.
- Οι Smart Contracts (στο Fabric ονομάζονται “chaincode”) τρέχουν μέσω ενός container όπως είναι το Docker και μπορούν να γραφτούν σε βασικές γλώσσες προγραμματισμού αλλά δεν έχουν απευθείας πρόσβαση στον πυρήνα (ledger).

Εφαρμογή συσχέτισης των “οντοτήτων” μέσα στο δίκτυο με κρυπτογραφημένες ταυτότητες (membership service provider).

3.3 BLOCKCHAIN VS DATABASES

Η αρχιτεκτονική του Blockchain διαθέτει εργαλεία, με τα οποία ο χρήστης έχει τη δυνατότητα μόνο να γράφει δεδομένα (write-only data structure), όπου καινούργιες οντότητες προσθέτονται στο τέλος του κατάστιχου (ledger). Κάθε καινούργιο block προσάπτεται στο blockchain συνδέοντας το με το Hash του προηγούμενη block. Σε αυτό το σημείο αξίζει να τονιστεί πως δεν υπάρχει κάποιος διαχειριστής στο blockchain ο οποίος έχει δικαιώματα να επεξεργάζεται ή να διαγράφει δεδομένα.

Στις κοινές βάσεις δεδομένων, τα δεδομένα μπορούν εύκολα να επεξεργαστούν ή να διαγραφούν. Σε κάθε βάση δεδομένων υπάρχει ο διαχειριστής που μπορεί εύκολα να κάνει αλλαγές σε κάθε κομμάτι δεδομένων ή στην δομή της βάσης. Επιπροσθέτως, το blockchain έχει σχεδιαστεί ώστε να χρησιμοποιείται σε αποκεντρωμένες εφαρμογές, ενώ οι σχεσιακές βάσεις δεδομένων χρησιμοποιούνται για συγκεντρωμένες εφαρμογές, όπου μια οντότητα – ενδεικτικά παραδείγματα αποτελούν ο server και ο κεντρικός υπολογιστής – ελέγχει τα δεδομένα**.**



Εικόνα 5: Centralized Databases VS Blockchain.

3.4 Ειδη Blockchain

Σε αυτό το σημείο, αξίζει να γίνει μια αναφορά στα διάφορα είδη του Blockchain. Πιο συγκεκριμένα:

- **Permissionless BlockChain**: Παρόμοιο με το bitcoin ή το Ethereum. Υπάρχουν ορισμένα Blockchain, τα οποία δεν έχουν κάποιες συγκεκριμένες άδειες στην χρήση. Με αυτό τον τρόπο, ο εκάστοτε χρήστης τους μπορεί να τρέξει έναν κόμβο, να εξορύξει (mining), να έχει πρόσβαση στο προσωπικό του πορτοφόλι (wallet) και να γράφει δεδομένα, αν αυτά πληρούν τους κανόνες του εκάστοτε blockchain. Ένα permissionless blockchain είναι ένα δημόσιο blockchain, στο οποίο ο καθένας μπορεί να εισέλθει.
- **Permissioned Blockchain:** Παρόμοιο με το Hyperledger Fabric. Ένα Blockchain μπορεί να δημιουργηθεί με τέτοιο τρόπο ώστε όταν ένας

χρήστης θέλει να εισέλθει σε αυτό θα πρέπει να του καταχωρηθούν και τα κατάλληλα δικαιώματα. Όπως επίσης μπορεί να χρειάζονται και αντίστοιχα δικαιώματα για να γράψει ή να διαβάσει μέσα σε αυτό. . Επίσης, στα permissioned blockchain συνήθως οι χρήστες «γνωρίζονται» μεταξύ τους εφόσον είναι «κλειστό» δίκτυο και χρειάζεται πρόσβαση για να εισέλθει.

4 ΑΝΑΛΥΣΗ ΣΕΝΑΡΙΟΥ

4.1 ΕΙΣΑΓΩΓΗ

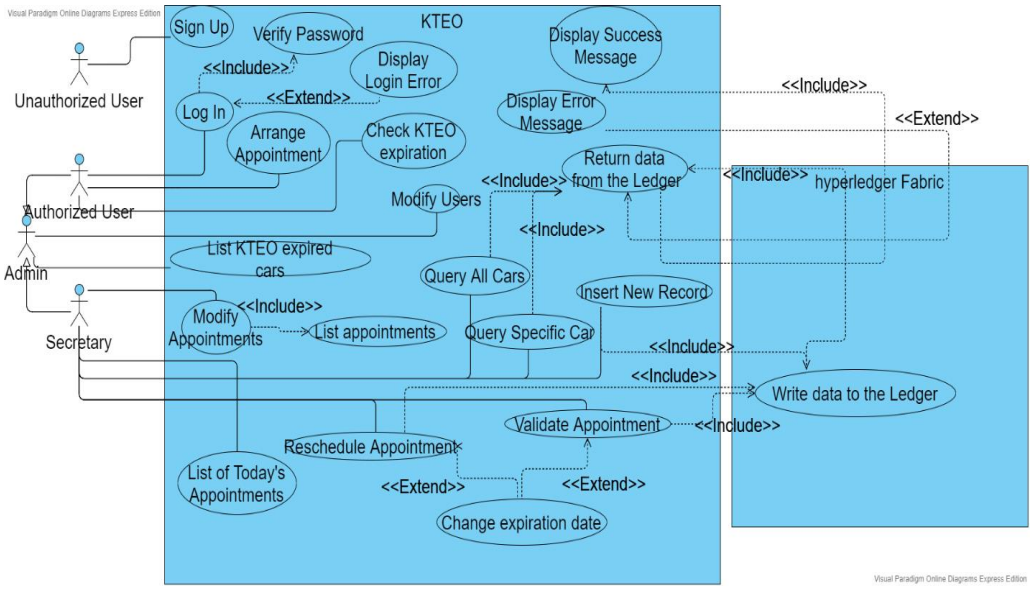
Στο παρόν κεφάλαιο θα πραγματοποιηθεί μια αναλυτική περιγραφή του σεναρίου που ακολουθήθηκε, προκειμένου να πραγματοποιηθεί η υλοποίηση της εφαρμογής. Παρακάτω καταγράφονται τα UML διαγράμματα όπως Use Case Diagram, Activity Diagram καθώς και Class Diagram που σχεδιάστηκαν κατά τη φάση της ανάλυσης και σχεδίασης της εφαρμογής.

Η εφαρμογή που θα πραγματοποιηθεί στη συνέχεια έχει ως αρχική ιδέα να αναπτυχθεί ένα σύστημα διαχείρισης των ραντεβού για μια ιδιωτική εταιρεία που ασχολείται με την πιστοποίηση ΚΤΕΟ οχημάτων. Επίσης, αναπτύσσεται η τεχνολογία του Block Chain όπου στεγάζεται και αλληλεπιδρούν οι χρήστες με αυτό μέσα στο σύστημα. Πρόκειται για ένα Permissioned Block Chain όπου αλλαγές στο περιεχόμενο και όχι στην δομή του θα μπορούν να κάνουν μόνο ο διαχειριστής και η γραμματεία

4.2 ΑΝΑΛΥΣΗ ΚΑΙ ΣΧΕΔΙΑΣΗ

4.2.1 USE CASE DIAGRAM

Κάθε χρήστης που εισέρχεται στο περιβάλλον της εφαρμογής μπορεί να πραγματοποιήσει διαφορετικές λειτουργίες ανάλογα με τον ρόλο του. Προκειμένου να παρουσιαστούν οι λειτουργίες αυτές δημιουργήθηκε το Use Case Diagram.



Εικόνα 6: Use Case Diagram

Το συγκεκριμένο διάγραμμα δείχνει όλες τις περιπτώσεις χρήσης του συστήματος από τους χρήστες. Το σύστημα μπορεί να χρησιμοποιηθεί από εξουσιοδοτημένους χρήστες, τον διαχειριστή (Administrator) καθώς και τη γραμματεία της εταιρίας.

Αρχικά, ένας μη εξουσιοδοτημένος χρήστης έχει τη δυνατότητα να αποκτήσει πρόσβαση στο σύστημα κατόπιν εγγραφής. Πιο συγκεκριμένα, υπάρχει η λειτουργία Sing Up. Εφόσον ο χρήστης αποκτήσει πρόσβαση στο σύστημα (Authorized User), έχει τη δυνατότητα να πραγματοποιήσει μια από τις ακόλουθες λειτουργίες: να συνδεθεί στο σύστημα (Login), να κανονίσει ένα ραντεβού (Arrange Appointment) καθώς και να ελέγξει την ημερομηνία λήξης του ΚΤΕΟ του αυτοκινήτου (Check KTEO expiration). Αξίζει να τονιστεί, πως προκειμένου ο χρήστης να πραγματοποιήσει έλεγχο της ημερομηνίας λήξης του ΚΤΕΟ, οφείλει να έχει επισκεφτεί τουλάχιστον μια φορά το χώρο της εταιρίας.

Η λειτουργία Login συσχετίζεται με τις λειτουργίες επαλήθευσης του κωδικού πρόσβασης (Verify Password) και εμφάνιση σφάλματος σύνδεσης (Display Login Error). Η σύνδεση της λειτουργίας Login και Verify Password πραγματοποιείται με Include, διότι προκειμένου να συνδεθεί κανείς στο

σύστημα οφείλει πρώτα να επαληθευτεί με επιτυχία ο κωδικός πρόσβασης. Αντίθετα, η σύνδεση της λειτουργίας Login και Display Login Error πραγματοποιείται με Extend. Η συσχέτιση αυτή αποτυπώνεται με αυτόν τον τρόπο καθώς η λειτουργία Display Login Error δεν πραγματοποιείται πάντα καθώς προϋποθέτει σφάλμα εισαγωγής του κωδικού πρόσβασης.

Η γραμματεία έχει τη λειτουργία να βλέπει μια λίστα με τα εκάστοτε ραντεβού της παρούσας ημέρας (List of Today's Appointments), καθώς και να πραγματοποιεί τροποποίηση αυτών (Modify Appointment). Αξίζει να σημειωθεί πως η λειτουργία Modify Appointments συσχετίζεται με τη λειτουργία List Appointments με Include. Με την παραπάνω λειτουργία η γραμματεία μπορεί να εγκρίνει ή να διαγράψει ένα ραντεβού.

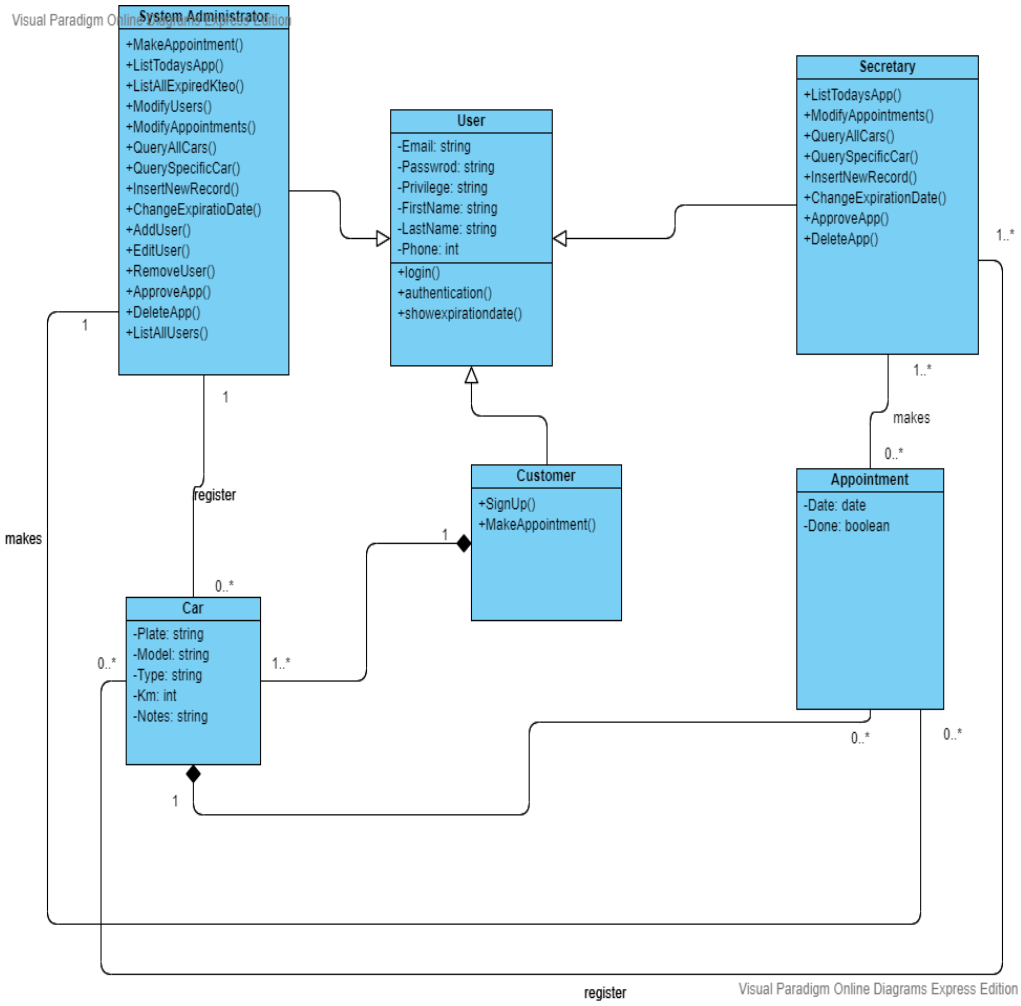
Στη συνέχεια η γραμματεία διαθέτει επιπλέον λειτουργίες η οποίες αλληλεπιδρούν με το σύστημα καθώς και με το Block Chain. Μια από τις λειτουργίες αυτές είναι η εμφάνιση όλων των δεδομένων που υπάρχουν στον πυρήνα (Ledger) του Blockchain (Query All Cars) και η εμφάνιση ενός συγκεκριμένου αυτοκινήτου που πραγματοποιείται με βάση την πινακίδα του οχήματος (Query Specific Car). Αξίζει να σημειωθεί πως οι δυο αυτές λειτουργίες συνδέονται μέσω include με τη λειτουργία Return data from the Ledger. Αντίστοιχα η προαναφερθείσα συσχετίζεται με include αν τα δεδομένα που εμφανιστούν είναι σωστά, αλλιώς συσχετίζεται με extend το οποίο θα εμφανίσει μήνυμα σφάλματος. Ακόμη η γραμματεία πραγματοποιεί την καταγραφή ενός χρήστη, ο οποίος πραγματοποιεί για πρώτη φορά επίσκεψη στο χώρο. Αυτό έχει σαν αποτέλεσμα την εισαγωγή μιας εγγραφής - record - στο Blockchain (Insert New Record).Επίσης η λειτουργία Insert New Record συσχετίζεται με include με τη λειτουργία Write data to the Ledger όπου αυτή αντίστοιχα συνδέεται με include με την λειτουργία Return data from the Ledger καθώς επιστρέφει το ID της αλλαγής που εκτελέστηκε ή ένα μήνυμα σφάλματος που συσχετίζεται με extend αν γίνει κάτι λάθος κατά της διαδικασία . Μια ακόμη λειτουργία αποτελεί η μεταβολή της ημερομηνίας

λήξης ΚΤΕΟ ενός οχήματος μετά την ολοκλήρωση του ραντεβού. Η ημερομηνία μεταβάλλεται για έναν χρόνο αργότερα (Validate Appointment) στην περίπτωση που το όχημα είναι πλήρως λειτουργικό και για έναν μήνα αργότερα (Reschedule Appointment) αν στο αυτοκίνητο έχει εντοπιστεί βλάβη. Έτσι, δίνεται η δυνατότητα στον πελάτη να επισκευάσει την βλάβη μέσα στο χρονικό περιθώριο και να επισκεφτεί ξανά το χώρο για έλεγχο. Η παραπάνω λειτουργία σχετίζεται με include με τη λειτουργία Write data to the Ledger όπου αυτό με τη σειρά του σχετίζεται με include με τη λειτουργία Return data from the Ledger. Οι παραπάνω λειτουργίες ενημερώνουν αντίστοιχα τα πεδία που αλλάζουν στον πυρήνα (Ledger) του Blockchain.

Ο Administrator έχει τη δυνατότητα να πραγματοποιεί όλες τις λειτουργίες που έχουν περιγραφεί παραπάνω, τόσο του εξουσιοδοτημένου χρήστη όσο και της γραμματείας. Επιπροσθέτως, ο Admin έχει τη λειτουργία να πραγματοποιεί τροποποιήσεις στους χρήστες (Modify Users) καθώς και να εμφανίζει μια λίστα με τα οχήματα των οποίων το ΚΤΕΟ έχει λήξει (List KTEO expired cars). Αξίζει να τονιστεί πως η λίστα αυτή ταξινομείται με βάση την ημερομηνία, σε αύξουσα σειρά.

4.2.2 CLASS DIAGRAM

Σε ένα Class Diagram αναπαρίστανται τα χαρακτηριστικά για τις πιο σημαντικές οντότητες του συστήματος. Στη συγκεκριμένη περίπτωση, αυτές είναι: ο πελάτης (Customer), ο System Administrator, η γραμματεία (Secretary), το αυτοκίνητο και τα ραντεβού.



Εικόνα 7: Class Diagram

Αρχικά, για να αναπαραστήσουμε τον διαχειριστή, τη γραμματεία και τον πελάτη, δημιουργήσαμε τη γενική κλάση User. Η συγκεκριμένη κλάση έχει ως attributes το email και το password τα οποία είναι τύπου string και παρέχει μεθόδους ώστε να επιτρέπει την είσοδο του κάθε χρήστη στην ιστοσελίδα της εταιρίας. Ακόμη, υπάρχει το privilege τύπου string που δείχνει τα δικαιώματα του κάθε χρήστη μέσα στο σύστημα και τον ανακατευθύνει στο αντίστοιχο Home Page. Τα first name, last name τύπου string καθώς και το phone το οποίο είναι τύπου int και δείχνουν πληροφορίες σχετικά με τα στοιχεία του χρήστη.

Επίσης, έχει τις μεθόδους Login για τη σύνδεση του διαχειριστή, της γραμματείας ή του πελάτη στο σύστημα η οποία αφού πραγματοποιηθεί με επιτυχία με τη βοήθεια της μεθόδου authentication τον ανακατευθύνει στην αντίστοιχη Αρχική Σελίδα. Άλλη μια μέθοδο που παρέχεται από το σύστημα είναι η showexpirationdate, η οποία παρέχει την δυνατότητα να ελεγχθεί η

ημερομηνία λήξης ΚΤΕΟ του οχήματος. Βασική προϋπόθεση να περάσει από έλεγχο το όχημα για τουλάχιστον μια φορά στο χώρο της εταιρείας.

Αξίζει να γίνει αναφορά για τις κλάσεις System Administrator, Secretary και Customer οι οποίες κάνουν extend την κλάση User και έτσι κληρονομούν τα attributes και τις μεθόδους που αναφέρθηκαν παραπάνω.

Ο διαχειριστής έχει τη δυνατότητα να πραγματοποιήσει ένα ραντεβού παίρνοντας ως παράμετρο την πινακίδα του οχήματος μέσω της μεθόδου MakeAppointment. Παράλληλα, μπορεί να ενημερωθεί για όλα τα ραντεβού που έχουν οριστεί για τη σημερινή ημέρα με την μέθοδο ListTodaysApp, με τη μέθοδο ListAllExpiredKteo εμφανίζεται μια λίστα με όλα τα οχήματα που έχει λήξει το ΚΤΕΟ τους καθώς επίσης και με τη μέθοδο ModifyAppointments μπορεί να διαχειριστεί όλα τα ραντεβού που είναι σε αναμονή με τη βοήθεια των μεθόδων ApproveApp και DeleteApp. Επιπροσθέτως, η μέθοδος ModifyUsers δίνει την δυνατότητα στον administrator να εμφανίσει μια λίστα με όλους τους ενεργούς χρήστες με τη μέθοδο ListAllUsers. Σε αυτή τη λίστα ο διαχειριστής έχει τη δυνατότητα να προσθέσει έναν καινούργιο χρήστη, να αφαιρέσει ή να τροποποιήσει τα στοιχεία ενός υπάρχον χρήστη με την βοήθεια των μεθόδων AddUser, RemoveUser και EditUser αντίστοιχα. Του δίνεται η δυνατότητα να εμφανίσει μια λίστα με τα οχήματα τα οποία έχουν περαστεί στο Block Chain με τη βοήθεια της μεθόδου QueryAllCars αλλά παράλληλα να εμφανίσει και ένα συγκεκριμένο όχημα με τη μέθοδο QuerySpecificCar παίρνοντας ως παράμετρο την πινακίδα του οχήματος. Αν κάποιος πελάτης έρθει για πρώτη φορά στον χώρο της εταιρίας και περάσει από τον έλεγχο, ο διαχειριστής τον προσθέτει στο Block Chain με τη βοήθεια της μεθόδου InsertNewRecord. Από την άλλη πλευρά αν ένας πελάτης έχει επισκεφτεί ξανά στο παρελθόν το χώρο της εταιρείας, στην ολοκλήρωση του διαγνωστικού ελέγχου μεταβάλλεται η ημερομηνία ΚΤΕΟ του οχήματος σε έναν μήνα ή έναν χρόνο αν υπάρχει κάποιο ελάττωμα ή όχι αντίστοιχα με τη

βοήθεια της μεθόδου ChangeExpirationDate που παίρνει ως παράμετρο την πινακίδα του εκάστοτε οχήματος.

Η γραμματεία διαθέτει κάποιες από τις δυνατότητες που αναφέρθηκαν παραπάνω όπως την εμφάνιση των σημερινών ραντεβού με τη μέθοδο ListTodayApp καθώς επίσης και την διαχείριση των ραντεβού που γίνεται με την μέθοδο ModifyAppointments και με τις μεθόδους ApproveApp γίνεται η έγκριση αυτών ή DeleteApp που γίνεται η απόρριψη αυτών. Αντίστοιχα, διαθέτει τις δυνατότητες που έχει και ο διαχειριστής όσο αναφορά το Block Chain. Μπορεί να εμφανίσει λίστα με όσα οχήματα υπάρχουν σε αυτό ή ένα συγκεκριμένο με το αναγνωριστικό του, δηλαδή την πινακίδα του οχήματος, με τις μεθόδους QueryAllCars και QuerySpecificCar. Επιπροσθέτως, μπορεί να εγγράψει έναν πελάτη στο σύστημα που έρχεται για πρώτη φορά (InsertNewRecord) και να μεταβάλλει την ημερομηνία λήξης ΚΤΕΟ σε περίπτωση που υπάρχει κάποια βλάβη σε διάστημα ενός μήνα ή ενός χρόνου αν όλα ολοκληρωθούν με επιτυχία με τη μέθοδο ChangeExpirationDate.

Ο πελάτης από την άλλη μπορεί να κάνει εγγραφή στο σύστημα αν δεν είναι ήδη εγγεγραμμένος με τη μέθοδο SignUp που παίρνει σαν παραμέτρους όλα τα στοιχεία που είναι απαραίτητα όπως email, first name , last name ,phone plate,model. Άλλη μια δυνατότητα που διαθέτει ο πελάτης είναι ότι μπορεί να ορίσει ένα ραντεβού ώστε να παρευρεθεί στον χώρο με τη μέθοδο MakeAppointment.

Στη συνέχεια, δημιουργήθηκε η κλάση Car η οποία αναπαριστά το όχημα που προορίζεται για έλεγχο ΚΤΕΟ. Το Car διαθέτει τα εξής attributes: τον αριθμό κυκλοφορίας του αυτοκινήτου (Plate) τύπου string, ο οποίος είναι μοναδικός για κάθε όχημα και το μοντέλο (Model) του αυτοκινήτου τύπου string. Ακόμη, η κλάση διαθέτει τον τύπο του οχήματος (Type) τύπου string, τα χιλιόμετρα που έχει διανύσει (Km) τύπου int καθώς επίσης και σημειώσεις για το συγκεκριμένο όχημα (Notes) τύπου string.

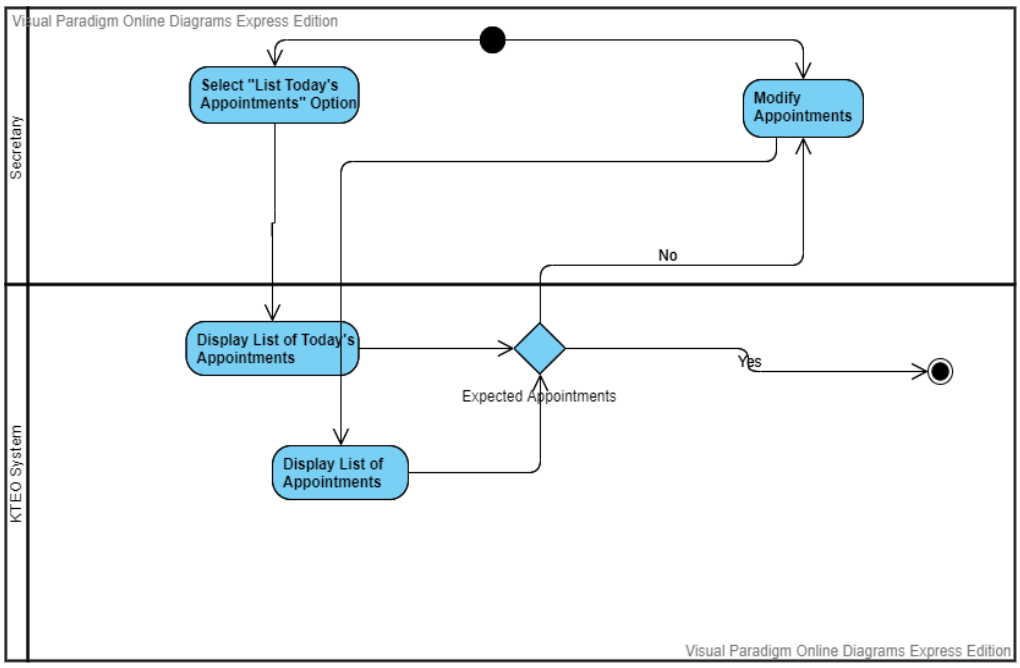
Η κλάση Car συσχετίζεται με τις κλάσεις System Administrator, Secretary και Customer. Ο System Administrator καταχωρεί (register) από μηδέν και πάνω οχήματα, όπως επίσης και κάθε υπάλληλος της γραμματείας (Secretary) καταχωρεί (register) από μηδέν και πάνω οχήματα. Από την άλλη, κάθε πελάτης διαθέτει από ένα και παραπάνω οχήματα. Επίσης, η κλάση Car συσχετίζεται με την κλάση Appointment καθώς ένα όχημα μπορεί να έχει από μηδέν και πάνω ραντεβού.

Η κλάση Appointment διαθέτει τα attributes τα οποία είναι η ημερομηνία που ορίζεται το ραντεβού (Date) τύπου string, το Done το οποίο δείχνει αν έγινε το προηγούμενο ραντεβού του οχήματος που το «έκλεισε», που είναι τύπου Boolean.

Αξίζει να σημειωθεί πως η κλάση Appointment συσχετίζεται με την κλάση System Administrator και Secretary καθώς κάθε διαχειριστής έχει τη δυνατότητα να κανονίσει (makes) ένα ραντεβού όπως αντίστοιχα και κάθε υπάλληλος της γραμματείας.

4.2.3 Activity Diagram

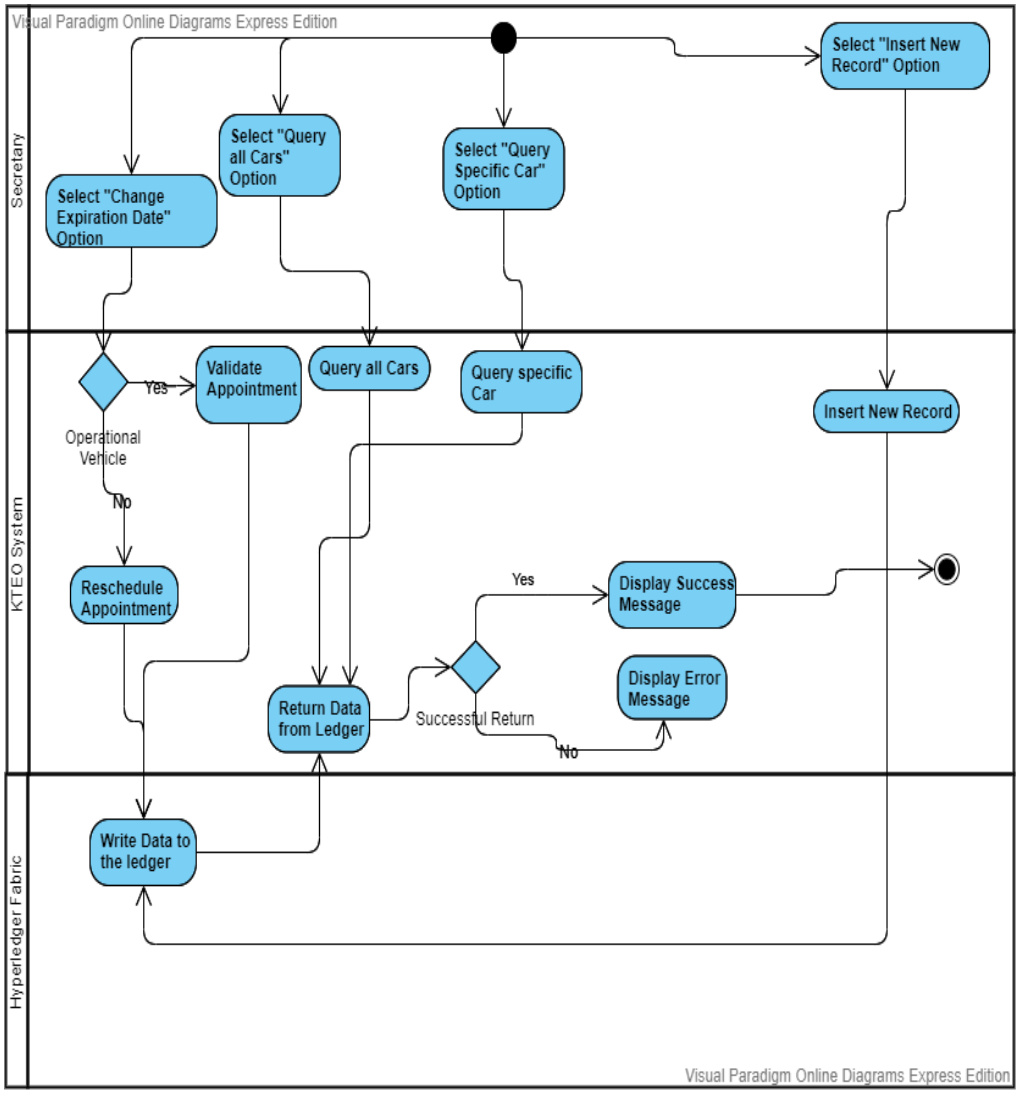
4.2.3.1 Appointments Activity Diagram



Εικόνα 8: Appointments Activity Diagram

Η λειτουργία διαχείρισης ραντεβού (Appointments) συσχετίστηκε με ένα ολόκληρο διάγραμμα - Activity Diagram – προκειμένου να αναλυθεί περαιτέρω το συγκεκριμένο Use Case. Ο υπάλληλος της γραμματείας θα έχει τη δυνατότητα να επιλέξει να δει μια λίστα με τα σημερινά ραντεβού (Select “List Today’s Appointments” Option), είτε να τροποποιήσει ορισμένα ραντεβού (Modify Appointments). Το σύστημα του ΚΤΕΟ παρουσιάζει αντίστοιχα μια λίστα με τα εκάστοτε ραντεβού της παρούσας ημέρας και μια λίστα με τα ραντεβού που περιμένουν έγκριση. Αυτά τα δύο Activities (Display List of Today's Appointments και Display List of Appointments) καταλήγουν στο decision node Expected Appointments. Εκεί, υπάρχουν δύο περιπτώσεις αν τα ραντεβού είναι τα αναμενόμενα πραγματοποιείται ολοκλήρωση της διαδικασίας και οποιαδήποτε αλλαγή πραγματοποιήθηκε από το χρήστη αποθηκεύεται. Αντίθετα, αν τα ραντεβού δεν είναι τα αναμενόμενα, ο χρήστης οδηγείται ξανά στην επιλογή Modify Appointments.

4.2.3.2 HYPERLEDGER FABRIC ACTIVITY DIAGRAM



Εικόνα 9: Hyperledger Fabric Activity Diagram

Η λειτουργία Hyperledger Fabric συσχετίστηκε με ένα ολόκληρο διάγραμμα – Activity Diagram – προκειμένου να αναλυθεί περισσότερο το συγκεκριμένο Use Case. Αρχικά ο χρήστης έχει τη δυνατότητα να επιλέξει να εμφανιστούν όλα τα δεδομένα που υπάρχουν στον πυρήνα (Ledger) του Blockchain (Query all Cars) και το σύστημα τα εμφανίζει επιστρέφοντας δεδομένα από το Ledger. Στη συνέχεια, το συγκεκριμένο Activity καταλήγει στο decision node Successful Return σύμφωνα με το οποίο αν πραγματοποιήθηκε επιτυχής επιστροφή των δεδομένων εμφανίζεται ένα μήνυμα επιτυχίας και η διεργασία ολοκληρώνεται. Σε αντίθετη περίπτωση, πραγματοποιείται εμφάνιση μηνύματος λάθους. Παρόμοια λειτουργικότητα έχει και το Query Specific Car

όπου ο χρήστης επιλέγει την εμφάνιση ενός συγκεκριμένου αυτοκινήτου που πραγματοποιείται με βάση την πινακίδα του οχήματος.

Επιπροσθέτως, ο υπάλληλος έχει τη δυνατότητα να εισάγει ένα χρήστη ο οποίος πραγματοποιεί για πρώτη φορά επίσκεψη στο χώρο. Αφού ο υπάλληλος εισάγει τα απαραίτητα στοιχεία στο σύστημα του ΚΤΕΟ και πατήσει create τα δεδομένα γράφονται στο Ledger (Write Data to the ledger). Μετά το συγκεκριμένο Activity οδηγείται κανείς στο Return Data from Ledger Activity όπου ακολουθείται η διαδικασία όπως περιγράφηκε παραπάνω.

Τέλος, ο υπάλληλος της γραμματείας έχει τη δυνατότητα να επιλέξει τη μεταβολή της ημερομηνίας λήξης ΚΤΕΟ ενός οχήματος μετά την ολοκλήρωση του ραντεβού. Με αυτό τον τρόπο ο χρήστης-υπάλληλος οδηγείται στο decision node Operational Vehicle σύμφωνα με το οποίο στην περίπτωση που το όχημα είναι πλήρως λειτουργικό η ημερομηνία μεταβάλλεται για ένα χρόνο αργότερα (Validate Appointment) ενώ αν στο όχημα έχει εντοπιστεί βλάβη επαναπρογραμματίζεται για τον επόμενο μήνα (Reschedule Appointment). Τόσο με το Validate Appointment όσο και με το Reschedule Appointment γράφονται δεδομένα στο Ledger (Write Data to the Ledger) και επαναλαμβάνεται η διαδικασία που περιγράφηκε παραπάνω.

5 ΕΓΚΑΤΑΣΤΑΣΗ

5.1 Εισαγωγή

Στο παρόν κεφάλαιο περιγράφεται αναλυτικά η εγκατάσταση των προαπαιτούμενων, προκειμένου να εκτελεστεί επιτυχώς το Hyperledger Fabric.

Έχοντας ως λειτουργικό σύστημα το Ubuntu 16.04.6 LTS και χρησιμοποιώντας για εικονική μηχανή (Virtual Machine) το λογισμικό VMware Workstation, κρίνεται απαραίτητο να εγκατασταθούν κάποια άλλα λογισμικά μέσα στο λειτουργικό σύστημα των Ubuntu, ώστε να τρέξει επιτυχώς το Hyperledger Fabric.

5.2 ΠΡΟΑΠΑΙΤΟΥΜΕΝΑ

1. Σε πρώτο στάδιο, πραγματοποιείται εγκατάσταση του cURL.

Πρόκειται για μια εντολή που παρέχει τη δυνατότητα στο χρήστη να μεταφέρει δεδομένα από ή προς ένα διακομιστή, χωρίς ο χρήστης να έχει κάποια αλληλεπίδραση.

Αρχικά, ο χρήστης οφείλει ως διαχειριστής, να εκτελέσει στο terminal την ακόλουθη εντολή:

$ sudo apt-get update

Στη συνέχεια, εκτελείται η εντολή:

$ sudo apt-get install curl

Τέλος, προκειμένου ο χρήστης να βεβαιωθεί για την επιτυχή εγκατάσταση του cURL, έχει τη δυνατότητα να εκτελέσει την εντολή:

$ curl –version

2. Σε δεύτερο στάδιο, ο χρήστης πραγματοποιεί εγκατάσταση του Docker και Docker Compose:

Το Docker για ένα εργαλείο που επιτρέπει την έναρξη εφαρμογών σε ένα απομονωμένο εικονικό περιβάλλον, σαν Virtual Machine.

Προκειμένου κανείς να προβεί στην εγκατάσταση του Docker, ακολουθώντας τον σύνδεσμο:

ο χρήστης οφείλει να κατεβάσει το παρακάτω αναφερόμενο αρχείο:

docker-ce_17.12.1~ce-0~ubuntu_amd64.deb

Κατόπιν από το terminal ο χρήστης πάει στη θέση που είναι αποθηκευμένο το αρχείο και εκτελεί την εντολή:

$ sudo dpkg -i "package_path/name"

Προκειμένου να ελέγξει ο χρήστης πως η εγκατάσταση του προγράμματος, πραγματοποιήθηκε με επιτυχία, μπορεί να τρέξει την ακόλουθη εντολή:

$ docker --version

Στη συνέχεια, προκειμένου να πραγματοποιηθεί εγκατάσταση του Docker Compose, εκτελείται η εντολή: Για την εγκατάσταση του docker compose και την επαλήθευση:

$ sudo apt install docker-compose

Τέλος, για την επαλήθευση της ορθής εγκαταστασης, ο χρήστης εκτελεί την εντολή:

$ docker-compose --version

3. Σε επόμενο στάδιο, ο χρήστης οφείλει να πραγματοποιήσει εγκατάσταση της γλώσσας προγραμματισμού **GO**. Πιο συγκεκριμένα, η γλώσσα προγραμματισμού GO χρησιμοποιείται από το Hyperledger Fabric κυρίως για το προγραμματισμό των συναλλαγών (chaincode).

Στο terminal, ο χρήστης πληκτρολογεί κι εκτελεί αρχικά την εντολή:

$ sudo apt install golang-go

και κατόπιν την εντολή:

$ go version

Για πιο ορθή προγραμματιστική διαχείριση, ο χρήστης θα ήταν συνετό να δημιουργήσει ένα φάκελο. Ο φάκελος καλό θα ήταν να ονομαστεί “go”, προκειμένου να αποτελεί αναγνωριστικό πως μέσα σε αυτόν, βρίσκονται αποθηκευμένα όλα τα αρχεία που αφορούν την συγκεκριμένη γλώσσα προγραμματισμού. Προκειμένου να δημιουργηθεί ο φάκελος αυτός, εκτελείται η εντολή:

$ mkdir $HOME/go

Στη συνέχεια, πραγματοποιείται εγκατάσταση του vim. Το vim, αποτελεί ένα πρόγραμμα επεξεργασίας αρχείων ώστε να παρέχεται η δυνατότητα επεξεργασίας του αρχείο **~/.bashrc** . Πρόκειται για ένα αρχείο των Linux που εκτελείται κάθε φορά που ανοίγει ένα terminal και περιέχει μια ρυθμίσεις όπως ιστορικό του terminal, χρώμα, συντομογραφίες εντολών και άλλα. Προκειμένου να πραγματοποιηθεί η εγκατάσταση του vim, πραγματοποιείται η εκτέλεση της ακόλουθης εντολής:

$ sudo apt install vim

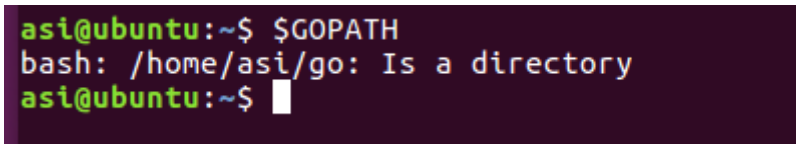
Και στο τέλος αυτού του αρχείου προσθέτω τις δύο παρακάτω εντολές με την σειρά που καταγράφονται:

export GOPATH=$HOME/go

export PATH=$PATH:$GOPATH/bin

με αυτό δείχνουμε στο κατάλληλο αρχείο που ξεκινάει μαζί με τα linux που βρίσκονται όλα τα αρχεία και οι βιβλιοθήκες της go. Στη συνέχεια με τη δεύτερη εντολή τοποθετούμε τα αρχεία της go στο ευρετήριο **bin.** Μετά κλείνουμε και ξανά ανοίγουμε το terminal και δοκιμάζουμε αν όλα έγιναν σωστά πληκτρολογώντας την ακόλουθη εντολή:

$GOPATH



Εικόνα 10: Έλεγχος Εγκατάστασης Go

4. Στη συνέχεια, πραγματοποιήθηκε εγκατάσταση της node.js. Λόγω του γεγονότος ότι το Hyperledger fabric που χρησιμοποιεί Node.js υποστηρίζει εκδόσεις του node πάνω από την version 8, εγκαταστάθηκε η έκδοση v8.16.1 (χρησιμοποιήθηκε το ακόλουθο αρχείο) με τις παρακάτω εντολές. Να σημειωθεί πως οι εντολές εκτελέστηκαν με την σειρά καταγραφής τους.

   $ curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -

   $ sudo apt-get install -y nodejs

   $ node –version

   Μαζί με την Node.js εγκαθίσταται επίσης και το NPM το οποίο παρέχει πακέτα με κώδικα ώστε να μπορεί να τρέξει η γλώσσα προγραμματισμού Node.js, παρόλα αυτά προτείνεται να εγκατασταθεί/αναβαθμιστεί χειροκίνητα (manually),

   $ sudo npm install npm –g

   Στη συνέχεια προστίθεται το χρήστη στην ομάδα του Docker ώστε να μπορούμε να το “τρέξουμε”.

   $ sudo usermod -aG docker "user_name"

   Ο χρήστης πραγματοποιεί αποσύνδεση (log out) από το σύστημα και συνδέεται ξανά, προκειμένου να ελέγξει αν όλα πραγματοποιήθηκαν επιτυχώς και πληκτρολογεί την εντολή:

   $ docker ps

Εικόνα 11: Έλεγχος Πρόσβασης Docker

5.3 Εγκατασταση Hyperledger Fabric

Προκειμένου να πραγματοποιηθεί εγκατάσταση του Hyperledger Fabric, ο χρήστης οφείλει να εκτελέσει στο Terminal την εντολή:

$ curl -sSL http://bit.ly/2ysbOFE | bash -s -- <fabric_version> <fabric-ca_version> <thirdparty_version>

Ενδεικτικό παράδειγμα αποτελεί:

$ curl -sSL http://bit.ly/2ysbOFE | bash -s -- 1.4.3 1.4.3 0.4.15

Στη συνέχεια, αφού τα αρχεία βρίσκονται στην κατοχή του χρήστη, εκείνος είναι αναγκαίο να δηλώσει στο αρχείο **~/.bashrc** που υπάρχουν τα αρχεία του fabric.

Το αρχείο ανοίγεται με το Vim και στο τέλος προστίθεται:

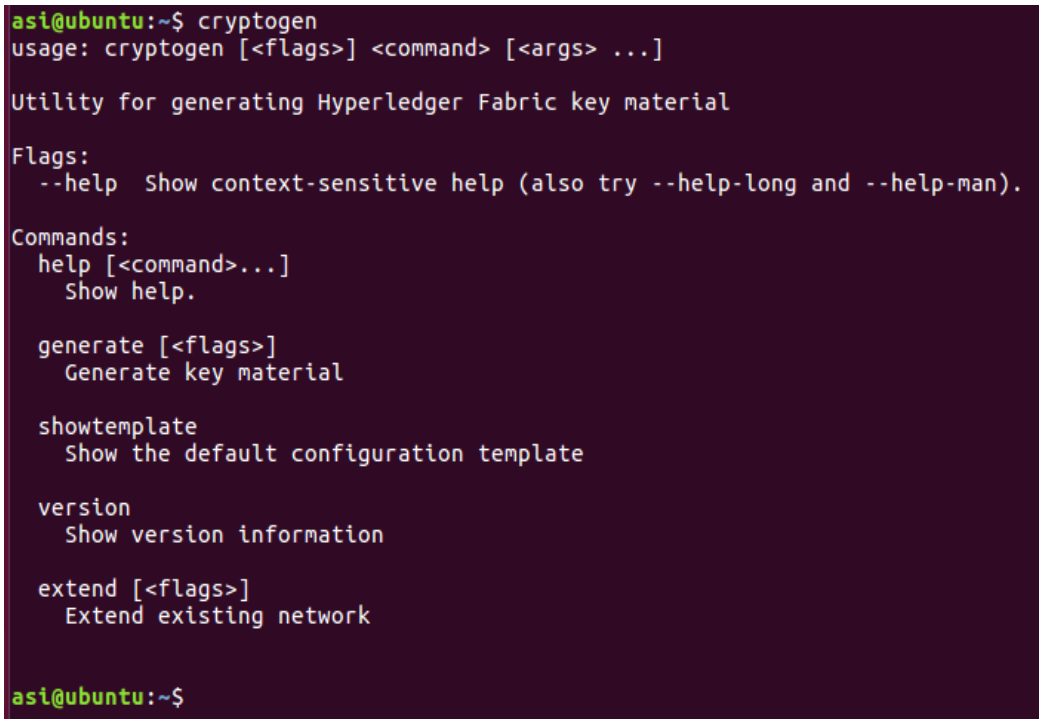
export PATH=/usr/local/hyperledger/fabric-samples/bin:$PATH

Το **/usr/local/hyperledger/fabric-samples/** αλλάζει ανάλογα με το που θα επιλέξει κάποιος να κατεβάσει τα αρχεία.

Μετά το πέρας όλων των παραπάνω, ο χρήστης κλείνει και ανοίγει ξανά το Terminal και πληκτρολογεί την εντολή:

$ cryptogen

όπως αυτή φαίνεται στην εικόνα που ακολουθεί, προκειμένου να πραγματοποιήσει έλεγχο για την επιτυχία της εγκατάστασης του Hyperledger Fabric:



Εικόνα 12: Έλεγχος Εγκατάσταση Hyperledger Fabric

6 ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΣΕΝΑΡΙΑ ΧΡΗΣΗΣ

6.1 ΥΛΟΠΟΙΗΣΗ

Προαπαιτούμενα για την δημιουργία του Web Service είναι η ύπαρξη μιας βάσης δεδομένων. Στην προκειμένη περίπτωση έχει δημιουργεί μια βάση δεδομένων με τη βοήθεια του MySQL Workbench. Ο λόγος που δημιουργήθηκε είναι για να μπορούμε να κρατάμε αρχείο με τους χρήστες που θα υπάρχουν στο σύστημα καθώς και άλλες πληροφορίες.

Στη συνέχεια δημιουργούμε ένα αρχείο με το όνομα server.js που μέσα από αυτό το αρχείο θα μπορούμε να «στήσουμε» το διακομιστή. Η Node.js γλώσσα έχει ένα σύνολο από συναρτήσεις που μπορούν περιληφθούν σε κάθε εφαρμογή, λέγονται modules. Για να κάνεις include ένα module και να χρησιμοποιήσεις ότι περιέχει αυτό απλά συμπεριλαμβάνεις μια require() συνάρτηση μαζι με το όνομα του module.

```
var http        = require('http');
var express     = require('express');      // call express
 var app        = express();              // define our app using express
 var bodyParser = require('body-parser');
 var http       = require('http');
```

Με αυτό η εφαρμογή έχει πρόσβαση στο HTTP Module και έχει τη δυνατότητα να ξεκινήσει ένα διακομιστή. Δηλώνουμε τη βιβλιοθήκη express και στη συνέχεια ορίζουμε την εφαρμογή να τη χρησιμοποιεί. Express είναι ένα μικρό και ευλύγιστο framework με εφαρμογή στη Nodejs, το οποίο παρέχει ένα σύνολο χαρακτηριστικών για τη δημιουργία διαδικτυακών εφαρμογών. Στη συνέχεια δηλώνουμε όλα τα ενδιάμεσα λογισμικά μας και ρυθμίζουμε την εφαρμογή να χρησιμοποιεί bodyParser() ώστε στην πορεία να μπορούμε να παίρνουμε δεδομένα από ένα POST.

```
app.use(bodyParser.urlencoded({ extended: true }));
```

```
app.use(bodyParser.json());
// Save our port
var port = process.env.PORT || 8000;

// Start the server and listen on port
app.listen(port,function(){
  console.log("Live on port: " + port);
});
```

Με το παραπάνω κομμάτι κώδικα έχουμε εύκολα και γρήγορα ένα διακομιστή να «τρέχει».

Στη συνέχεια, προσθέτουμε:

```
app.get('/', function (request, response) {
        response.end('hello word');
});
```

Έτσι ώστε κάθε φορά που πληκτρολογούμε στο browser την διεύθυνση localhost:8000 θα μας εμφανίζεται ένα hello word. Επειδή όμως χρειαζόμαστε ένα Login Page στο σύστημα χρειάζεται να τον ανακατευθύνουμε σε ένα άλλο αρχείο που εκεί ο χρήστης θα κάνει Login. Αυτό επιτυγχάνεται με την εντολή response.render(“FileName”). Οπότε για να ανατατευθύνουμε το χρήστη στο Login Page δημιουργείται ένα αρχείο που θα λέγεται login.ejs και θα περιέχει:

```
<div class="login-form">
        <h1>Login Form</h1>
        <form action="/auth" method="POST">
                <input type="text" name="email" placeholder="Email" required>
                <input type="password" name="password" placeholder="Password" required>
                <input type="submit">
```

```
</form>
</div>
```

Με αυτή τη διαδικασία αλληλοεπιδράει το frontend με το backend καθώς στη συνεχεία στο αρχείο του server.js θα φτιάξουμε ένα app.get το οποίο θα έχει το όνομα /auth και θα εκτελείται όταν ο χρήστης να κάνει Login.

Επόμενο βήμα είναι να μπορούμε να ελέγχουμε ποιοι είναι χρήστες. Έτσι δημιουργούμε μια βάση δεδομένων (MySQL). Στο αρχείο server.js δηλώνουμε το module για τη χρήση της mysql γλώσσας και στη συνέχεια κάνουμε τη σύνδεση με τη βάση.

```
var mysql = require('mysql');

var connection = mysql.createConnection({
host: 'localhost',
user: 'Username',
password: 'Password',
database: 'Database_Name'
});
```

Παραπάνω ορίζουμε ότι η μεταβλητή connection θα έχει τα παραπάνω στοιχεία για το κάθε πεδίο. Οπότε αν θέλουμε να εκτελέσουμε ένα query από το backend της εφαρμογής, όπως για παράδειγμα το query του login form ώστε να ελεγχθεί αν ο χρήστης υπάρχει στο σύστημα.

Αρχικά, ελέγχουμε αν πήραμε τα δεδομένα από τη φόρμα του Login.

```
var email = request.body.email;
var pass =request.body.pass;
```

Αφού γίνει ο έλεγχος στη συνέχεια:

```
connection.query('SELECT * FROM users WHERE email = ? AND pass = ?',
[email, password], function (error, results, fields) {
If(results.length > 0){
 …Ανακατευθύνουμε το χρήστη στο αντίστοιχο HomePage με βάση τα
αποτελέσματα που πήραμε…
        If(results[0].privilege == ‘s’){
                ….go to Secretary Home Page
          } else if (results[0].privilege == ‘a’){
                ….go to Admin Home Page
} else {
        ….go to User Home Page
}
} else {
          response.send('Incorrect Username and/or Password!');
          }
          response.end();
});
```

Με τον παραπάνω κώδικα ζητάμε από τη βάση να μας επιστρέψει όλες τις καταχωρήσεις με το email και pass που δόθηκε από το χρήστη. Αν υπάρχουν αποτελέσματα τότε το results.length θα είναι θετικό αντιθέτως δεν θα υπάρχει καταχώρηση στη βάση με τα συγκεκριμένα αναγνωριστικά. Έγινε δημιουργία μιας στήλης στη βάση δεδομένων που ονομάζεται Privilege και μπορεί να πάρει τιμές A, U, S για τον διαχειριστή, τη γραμματεία και τον απλό χρήστη αντίστοιχα. Επομένως, η ανακατεύθυνση γίνεται με την χρήση πολλαπλών if conditions.

Αντίστοιχα, στο Login Page δίνεται και η δυνατότητα σε κάποιον επισκέπτη να κάνει εγγραφή στο σύστημα (Sign Up button)ώστε να του παρέχονται κάποιες δυνατότητες όπως να ελέγξει την ημερομηνία του ΚΤΕΟ του ή να κλείσει ένα

ραντεβού για να επίσκεψη στο χώρο της εταιρείας. Πάλι με τον ίδιο τρόπο δημιουργείται ένα <form action=”/signup” method=”POST”></form> πεδίο στο login.ejs αρχείο όπου επικοινωνεί με το αντίστοιχο app.get(“/signup”) του αρχείου server.js και μέσα σε αυτό αρχικά ελέγχεται αν ο χρήστης συμπλήρωσε όλα τα πεδία που είναι υποχρεωτικά και στη συνέχεια γίνεται η καταχώρηση των στοιχείων του στη βάση δεδομένων, με αποτέλεσμα να μπορεί αν όλα γίνουν σωστά να κάνει Login στο σύστημα.

Αφού γίνουν οι διαδικασίες που περιγράφηκαν παραπάνω γίνεται η ανακατεύθυνση του χρήστη (Admin, Secretary, User) με τη βοήθεια της εντολής response.render(“HomePageName”) στο αντίστοιχο Home Page. Οπότε χρειάστηκε να δημιουργηθούν τρία αντίστοιχα Home Page αρχεία που κάθε ένα από αυτά θα διαθέτει ένα app.get(“/HomePageName”) ώστε να γίνεται η ανακατεύθυνση και να μπορεί ο χρήστης να έχει οπτική επαφή. Αντίστοιχα, χρειαζόμαστε και app.post ώστε να επικοινωνεί με το server το αρχείο μας.

Αν λοιπόν ο χρήστης που κάνει Login έχει για privilege = ‘U’, δηλαδή είναι απλός χρήστης τότε θα τον ανακατευθύνει στην αρχική σελίδα που βλέπει κάθε χρήστης.

Το αρχείο (HomeUser.ejs) περιέχει βιβλιοθήκες που αφορούν το Bootstrap, Ajax, Angular, Jquery. Για να δημιουργηθεί η μπάρα ή το μενού επιλογών που έχει ο χρήστης χρησιμοποιήθηκε μια κλάση από τη βιβλιοθήκη του Bootstrap που ονομάζεται navbar.

```
<nav class="navbar navbar-inverse">
  <div class="container-fluid">
   <div class="navbar-header">
    <a class="navbar-brand" href="/homeU">KTEO</a>
   </div>
```

```
    <ul class="nav navbar-nav">
      <li class="active"><a href="/homeU">Home</a></li>
      <li><a href="/showExDateU">Show Expired Date</a></li>
      <li><a href="/makeAppointment">Make Appointment</a></li>


    </ul>
    <ul class="nav navbar-nav navbar-right">
      <li>
        <a href="/logout"><span class="glyphicon glyphicon-log-
out"></span></a>
      </li>
    </ul>
   </div>
  </nav>
```

Παραπάνω διακρίνεται πως σε κάθε υπάρχει η αντίστοιχη σελίδα-αρχείο που φτιάχτηκε για να μπορεί ο χρήστης να εκτελέσει την οποιαδήποτε λειτουργία που έχει δημιουργηθεί μέσα στο αρχείο. Τέλος, υπάρχει το του Logout που φαίνεται στην παραπάνω εικόνα σαν εικονίδιο εξόδου, που συνδέεται με το αρχείο του server μέσω app.get("/logout") και από εκεί γίνεται η διαδικασία αποσύνδεσης του χρήστη από το σύστημα.

```
app.get('/logout', function (request, response, next) {
        if (request.session) {
                request.session.destroy(function (err) {
                        if (err) {
                                return next(err);
                        } else {
                                return response.redirect('/');
                        }
```

```
            });
        }

});
```

Αναλυτικότερα αν ο χρήστης μετακινήσει τον κέρσορα πάνω στο σημείο του φαίνεται το ΚΤΕΟ καθώς και το Home το σύστημα θα τον μεταφέρει το home page, δηλαδή στην παραπάνω εικόνα. Αν μετακινήσει τον κέρσορα και πατήσει το Show Expired Date θα τον μεταφέρει σε μια καινούργια οθόνη που ο κώδικας της σελίδας υπάρχει στο αρχείο showExDate.ejs.

Ο χρήστης μπορεί να πληκτρολογήσει την πινακίδα του οχήματος του και να δει πότε λήγει το ΚΤΕΟ του οχήματός του. Στο server αρχείο μεταφέρεται η πινακίδα και από εκεί εκτελείται ένα Query στη βάση όπου επιστρέφει το αποτέλεσμα της στήλης ExpiredDate. Αν αυτό είναι κενό (NULL) σημαίνει πως ο χρήστης δεν έχει επισκεφθεί το χώρο, οπότε δεν έχει περαστεί στη βάση η ημερομηνία λήξης ΚΤΕΟ του οχήματός του. Επομένως, θα εμφανιστεί μήνυμα σφάλματος. Θα τον ανακατευθύνει με τη βοήθεια της response.render σε καινούργιο αρχείο (ShowExDate1.ejs) όπου εκεί θα παρουσιάζονται η ημερομηνία ή αλλιώς το μήνυμα σφάλματος, όπως μπορεί κάποιος να δει παρακάτω.

Άλλη μια δυνατότητα που έχει ο χρήστης είναι το Make Appointment. Όταν λοιπόν πατήσει σε αυτό μεταφέρεται σε καινούργια οθόνη όπου εκεί θα του ζητηθεί να ορίσει την ημερομηνία που επιθυμεί για να παρευρεθεί στο χώρο και να πληκτρολογήσει την πινακίδα του οχήματός του για επαλήθευση ή και σε περίπτωση που διαθέτει παραπάνω από ένα οχήματα.

Εκτός από τις βιβλιοθήκες που υπάρχουν σε όλα τα .ejs αρχεία η συγκεκριμένη διαθέτει μια παραπάνω βιβλιοθήκη η οποία μας βοηθάει να εντάξουμε το ημερολόγιο και είναι η Bootstrap-datepicker. Η δομή του αρχείου όσον αφορά το ημερολόγιο είναι η παρακάτω.

```
<div class="bootstrap-iso">
                <div class="container-fluid">
                 <div class="row">
                  <div class="col-md-6 col-sm-6 col-xs-12">

                   <!-- Form code begins -->
                   <form method="post">
                    <div class="form-group"> <!-- Date input -->
                     <label class="control-label" for="date">Date</label>
                     <input class="form-control" id="date" name="date" placeholder="MM/DD/YYY" type="text"/>
                    </div>
                    <input type="text" name="carid" id="carid" placeholder="plate" required>
                    <div class="form-group"> <!-- Submit button -->
                     <button class="btn btn-primary " name="submit" type="submit">Submit</button>
                    </div>
                   </form>
                   <!-- Form code ends -->

                  </div>
                 </div>
                </div>
               </div>
```

Και στη συνέχεια χρειάζεται να δημιουργηθεί ένα script ώστε να παρθούν τα δεδομένα από το ημερολόγιο.

```
<script>
            $(document).ready(function(){
              var carid=$('#carid').val();
             var date_input=$('input[name="date"]'); //our date input has the name "date"
             var container=$('.bootstrap-iso form').length>0 ? $('.bootstrap-iso form').parent() : "body";
             var options={
              format: 'yyyy-mm-dd',
              container: container,
              todayHighlight: true,
              autoclose: true,
             };
             date_input.datepicker(options);

            })
            $.post('/makeAppointment', { date: date_input.datepicker(options), carid: carid});
        </script>
```

Με τον παραπάνω κώδικα αντλείται η ημερομηνία που έδωσε ο χρήστης και στη συνέχεια διαμορφώνεται σύμφωνα με κάποια πρότυπα που βολεύουν τόσο στην ανάγνωση όσο και στην αποθήκευση αυτής στη βάση. Στη συνέχεια, με τη βοήθεια του jQuery ($.post) στέλνονται τα δεδομένα (ημερομηνία, πινακίδα οχήματος) στο server αρχείο. Αφού πατήσει το κουμπί του Submit πραγματοποιούνται έλεγχοι για το αν η ημερομηνία είναι σωστή καθώς

επίσης και αν η υπάρχει η πινακίδα στη βάση του συστήματος. Στη συνέχεια, μεταφέρεται σε άλλη οθόνη που του δείχνει αν όλα έγιναν σωστά και πως το ραντεβού έχει οριστεί για την ημερομηνία που επέλεξε.

Στην περίπτωση που κάνει Login ένας υπάλληλος του ΚΤΕΟ με privilege S (Γραμματεία) θα εμφανιστεί διαφορετική αρχική οθόνη από αυτή του απλού χρήστη.

Το αρχείο που αντικατοπτρίζει την παραπάνω αρχική οθόνη έχει δομηθεί με το ίδιο μοτίβο καθώς επίσης και με τις ίδιες βιβλιοθήκες που δομήθηκε και η αρχική οθόνη του user. Επίσης, η λειτουργία του Show Expired Date έχει φτιαχτεί με τον ίδιο τρόπο. Δηλαδή, στο .ejs αρχείο υπάρχει ένα <form></form> που συνδέεται με το server αρχείο μέσω ενός app.post(/exdate). Στη συνέχεια, μέσα στο POST ορίζονται οι μεταβλητές που θα χρειαστουν η οποίες είναι η πινακίδα (Plate) και τα δικαιώματα (Privilege). Στην πορεία με βάση την μεταβλητή privilege εκτελείται ένα Switch που έχει για περιπτώσεις αν το privilege είναι A , S και U. Ανάλογα με το δικαίωμα του χρήστη εκτελείται και η αντίστοιχη περίπτωση που αναζητά στη βάση την ημερομηνία λήξης ΚΤΕΟ που πληκτρολογήθηκε και τον ανακατευθύνει στην αντίστοιχη σελίδα.

Μετά υπάρχει και η ενότητα Appointments που πρόκειται για ένα Dropdown Menu το οποίο εχει τις ενότητες Today’s Appointments και Modify. Στο πρώτο ο υπάλληλος της γραμματείας μπορεί να δει όλα τα ραντεβού που έχουν προγραμματιστεί για σήμερα, ενώ στο δεύτερο βλέπει όλες τις αιτήσεις που έχουν γίνει από τους χρήστες για ραντεβού και τις εγκρίνει ή της απορρίπτει ανάλογα με το φόρτο εργασίας.

Για να φτιαχτεί το menu χρησιμοποιήθηκε το Bootstrap.

<!-- Dropdown -->

```
<li class="nav-item dropdown">
                    <a class="nav-link dropdown-toggle" href="#"
id="navbardrop" data-toggle="dropdown">
                    Appointments
                    </a>
                    <div class="dropdown-menu">
                      <a class="dropdown-item" href="/today">Today's
Appointments</a>
                      <a class="dropdown-item"
href="/findAppointment">Modify</a>

                    </div>
                  </li>
```

Αν πατήσει κάποιος το Today's Appointment στο αρχείο του server εκτελείται ένα query στη βάση όπου επιστρέφει όλα τα αποτελέσματα που έχουν εγκριθεί και έχουν τη σημερινή ημερομηνία. Στη συνέχεια ανακατευθύνει σε καινούργια σελίδα όπου εκτυπώνονται τα αποτελέσματα σε ένα πίνακα ώστε να μπορούν να διαβαστούν πιο εύκολα από το χρήστη.

Η άλλη επιλογή είναι το Modify, ο χρήστη κατευθύνεται σε καινούργια σελίδα όπου του ζητείται να επιλέξει ημερομηνία ώστε να του εμφανιστούν όλες οι αιτήσεις που έχουν γίνει για τη συγκεκριμένη μέρα. Η δημιουργία του ημερολογίου πραγματοποιήθηκε με τον ίδιο τρόπο όπως έγινε παραπάνω. Εκεί εμφανίζεται ένας πίνακας ο οποίος δείχνει τις αιτήσεις που έχουν κάνει οι χρήστες για ραντεβού. Στον πίνακα εκτός από τις στήλες όνομα, επίθετο και άλλα στο τέλος υπάρχουν δύο κουμπιά της Έγκρισης και της Απόρριψης του ραντεβού καθώς με αυτόν τον τρόπο γίνεται πιο εύκολη και γρήγορη η διαδικασία έγκρισης των ραντεβού. Ο server στέλνει στο .ejs αρχείο τα δεδομένα από τη βάση οπότε ο κώδικας θα είναι ο παρακάτω.

```
<tbody>
```

```
<% for(var i=0; i<results.length; i++) {%>
<tr>
<td><%= results[i]['fname'] %></td>
<td><%= results[i]['lname'] %></td>
<td><%= results[i]['phone'] %></td>
<td><%= results[i]['plate'] %></td>
<td><button type="button" onclick="approveAp('<%= results[i]['plate']%>')" class="btn btn-default btn-sm"><span class="glyphicon glyphicon-ok"></span></button></td>
<td><button type="button" onclick="deleteAp('<%= results[i]['plate']%>')" class="btn btn-default btn-sm"><span class="glyphicon glyphicon-trash"></span></button></td>
</tr>
<% } %>
</tbody>
```

Τα δυο τελευταία <td></td> είναι τα κουμπιά στον πίνακα που στέλνου με την βοήθεια ενός script ($.post) τα δεδομένα (Ημερομηνία, Πινακίδα οχήματος) στο server και αυτός με τη σειρά του εγκρίνει ή απορρίπτει ένα ραντεβού ανάλογα με το κουμπί που επιλέχτηκε.

Τέλος ο τελευταίος χρήστης που κάνει Login και έχει διαφορετική αρχική οθόνη από τους παραπάνω χρήστες (Secretary, User) είναι ο διαχειριστής (Admin). Ο διαχειριστής έχει παραπάνω λειτουργίες στην σελίδα Home οι οποίες του δίνουν τη δυνατότητα να δει αν το ΚΤΕΟ ενός οχήματος έχει λήξη, να ορίσει ημερομηνία για ραντεβού, να δει τα σημερινά ραντεβού καθώς επίσης να δει και όλα τα οχήματα που έχει τελειώσει το ΚΤΕΟ τους με αύξουσα σειρά παλαιότητας, δηλαδή το πιο παλιό εμφανίζεται πρώτο. Όλες αυτές οι λειτουργίες φτιαχτηκαν με τη βοήθεια του jQuery και το AJAX. Αναλυτικότερα στο αρχείο HomePageAdmin.ejs εκτός από τα scripts των βιβλιοθηκών

υπάρχουν και scripts που παίρνουν τα δεδομένα από τις αντίστοιχη λειτουργία που θα επιλέξει ο διαχειριστής να κάνει, επικοινωνούν με το αντίστοιχο app.post του server και επιστρέφουν τα αποτελέσματα χωρίς να κατευθυνθεί σε διαφορετικό .ejs αρχείο.

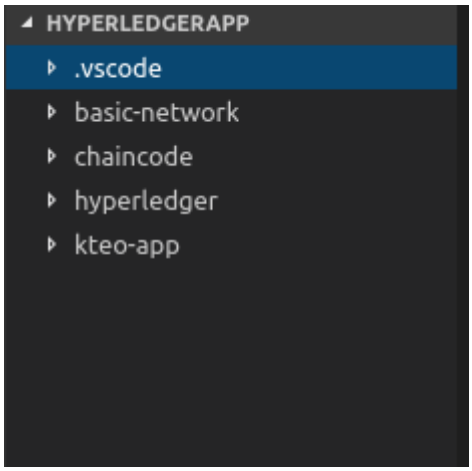
Στη μπάρα πλοήγησης υπάρχει η σελίδα που ο διαχειριστής μπορεί να τροποποιήσει τους χρήστες του συστήματος, να προσθέσει έναν καινούργιο χρήστη, να διαγράψει ή να αλλάξει τα στοιχεία των χρηστών.

Μετά υπάρχει το Modify Appointments που έχει την ίδια λειτουργικότητα με αυτή της γραμματείας.

Τέλος, στο διαχειριστή και στη γραμματεία υπάρχει η σελίδα που ονομάζεται Validate. Αυτή χρησιμοποιείται για την εκτύπωση όλων των δεδομένων στο Blockchain, αναζήτηση συγκεκριμένου στοιχείου με βάση την πινακίδα ενός οχήματος. Αν ο πελάτης έρχεται για πρώτη φορά στο χώρο της εταιρίας προστίθεται από τη γραμματεία τόσο στο Blockchain όσο και στη βάση δεδομένων του ΚΤΕΟ. Τέλος, αν ολοκληρωθεί το ραντεβού και γίνει ο έλεγχος η γραμματεία αλλάζει την ημερομηνία λήξης του ΚΤΕΟ είτε σε ένα χρόνο αν όλα είναι εντάξει ή δίνει διορία ενός μήνα στον πελάτη να φτιάξει τα πράγματα που διαγνώστηκαν από τον έλεγχο ότι δεν πληρούν τις προϋποθέσεις.

Για να είναι πιο εύκολο φτιάχνουμε τους φακέλους που είναι μέσα το project. Ο πρώτος φάκελος θα ονομαστεί HyperledgerApp μέσα σε αυτόν χρειαζόμαστε άλλους τρεις υπό φακέλους που θα λέγονται basic-network, chaincode και kteo-app. Μέσα στο φάκελο kteo-app υπάρχουν όλα τα αρχεία που αφορούν την ιστοσελίδα (server.js και όλα τα .ejs αρχεία) επίσης και το αρχείο package.json το οποίο είναι απαραίτητο. Στη συνέχεια στο φάκελο basic-network πρέπει να αντιγραφούν όλα τα αρχεία που βρίσκονται στο φάκελο (./fabric-samples/basic-network) που κατέβηκε από το σύνδεσμο του

Hyperledger στην ενότητα με τα προαπαιτούμενα. Η δομή πρέπει να είναι όπως φαίνεται παρακάτω.



Εικόνα 13: Directory

Το επόμενο βήμα εφόσον υπάρχουν τα αρχεία του Hyperledger μέσα στο φάκελο του basic-network, είναι η δημιουργία τριών αρχείων που αφορούν το hyperledger, starFabric.sh το οποίο ξεκινάει το δίκτυο και δημιουργεί το αρχικο block (genesis.block) και δημιουργεί το μέσο για να μπορούν να επικοινωνούν τα μέλη του blockchain με διαφάνεια (channel.tx). Το επόμενο αρχείο που χρειαζόμαστε είναι το registerAdmin.js το οποίο εγγράφει το χρήστη στο πιστοποιητικό ασφαλείας του blockchain (certification-authority, CA) και δίνει στο χρήστη ένα κλειδί ασφαλείας που αποθηκεύεται σε έναν φάκελο του διαχειριστή. Στη συνέχεια χρειαζόμαστε το registerUser.js με το οποίο ο admin κάνει εγγραφή του user1 στο CA μαζί με τις απαραίτητες πληροφορίες. Τέλος, στο φάκελο chaincode χρειάζεται να δημιουργηθεί ένα αρχείο που διαχειρίζεται τον πυρήνα του blockchain και επιτρέπει να πραγματοποιούνται οι συναλλαγές από την εφαρμογή.

Αναλυτικότερα, για να ξεκινήσει το δίκτυο του Hyperledger Fabric πρέπει από το τερματικό (terminal) να βρισκόμαστε μέσα στο φάκελο του kteo-app. Από εκεί τρέχουμε την εντολή

$ ./startFabric.sh

Με την παραπάνω εντολή αφαιρούνται από τον Docker όλα τα υπάρχοντα δίκτυα και δημιουργείται ένα καινούργιο με 1 peer & couch DB μαζί με έναν orderer.

```
set -e

# don't rewrite paths for Windows Git Bash users
export MSYS_NO_PATHCONV=1

starttime=$(date +%s)

if [ ! -d ~/.hfc-key-store/ ]; then
        mkdir ~/.hfc-key-store/
fi

# launch network; create channel and join peer to channel
cd ../basic-network
./start.sh
```

Δημιουργείται ένα wallet (hfc-key-store) και στη συνέχεια καλείται το αρχείο **./start.sh** όπου αυτό με τη σειρά του κάνει clean-up and re-populate τον Docker. Επίσης, δημιουργείται το channel για την επικοινωνία και ένας peer προστίθεται. Στη συνέχεια εφόσον το δίκτυο είναι έτοιμο, πρέπει να κάνουμε εγγραφή του admin στο δίκτυο και αυτό γίνεται με το να τρέξουμε το αρχείο registerAdmin.js.

```
$ node registerAdmin.js
```

Μέσα στον κώδικα του αρχείου δημιουργείται ο Admin μαζί με το απαραίτητο κλειδί. Αφού δημιουργηθεί ο Admin, τρέχουμε το αρχείο registerUser.js και

στην ουσία γίνεται εγγραφή του user1. Τελευταίο βήμα είναι να τρέξουμε το διακομιστή που έχει φτιαχτεί, server.js.

Για να μπορούμε να έχουμε αλληλεπίδραση μεταξύ του web service και του blockchain πρέπει πρώτα να φτιάξουμε το αρχείο που αφορά τα chaincode ή αλλιώς smart contracts. Το αρχείο το ονομάζουμε kteo-app.go και το τοποθετούμε μέσα στο φάκελο του chaincode. Για αρχή πρέπει να κάνουμε import τα απαραίτητα εργαλεία-βιβλιοθήκες που βοηθούν στη διαχείριση των bytes, την ανάγνωση και τη γραφή JSON δεδομένων και string.

```
import (
"bytes"
"encoding/json"
"fmt"
"strconv"


"github.com/hyperledger/fabric/core/chaincode/shim"
sc "github.com/hyperledger/fabric/protos/peer"
)
```

Και στη συνέχεια ορίζουμε τη δομή του αντικειμένου και τη δομή του smart contract.

```
// Define the Smart Contract structure
type SmartContract struct {
}

/* Define Car structure,
Structure tags are used by encoding/json library
```

```
*/
type Car struct {
	Fname string `json:"fname"`
	Lname string `json:"lname"`
	Phone string `json:"phone"`
	Model string `json:"model"`
	Type  string `json:"type"`
	Km    string `json:"km"`
	Notes string `json:"notes"`
	ExDate string `json:"exdate"`
}
```

Χρειαζόμαστε μια μέθοδο η οποία θα εγκαθιστά το smart contract κατά τη δημιουργία του δικτύου. Αυτή η μέθοδος λέγεται Init. Επιπλέον, χρειαζόμαστε μια μέθοδο που θα καλείται όταν η εφαρμογή μας κάνει request για την εκτέλεση ενός Smart contract “chaincode”.

```
func (s *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {

	// Retrieve the requested Smart Contract function and arguments
	function, args := APIstub.GetFunctionAndParameters()
	// Route to the appropriate handler function to interact with the ledger
	if function == "queryCar" {
		return s.queryCar(APIstub, args)
	} else if function == "initLedger" {
		return s.initLedger(APIstub)
	} else if function == "queryAllCars" {
		return s.queryAllCars(APIstub)
```

```
        } else if function == "recordCar" {
                return s.recordCar(APIstub, args)
        } else if function == "nextYear" {
                return s.nextYear(APIstub, args)
        } else if function == "nextMonth" {
                return s.nextMonth(APIstub, args)
        }

        return shim.Error("Invalid Smart Contract function aname.")
}
```

Παρακάτω σε αυτό το αρχείο συντάσσουμε τις μεθόδους σύμφωνα με τις απαιτήσεις. Η μέθοδος queryCar παίρνει σαν όρισμα μια μεταβλητή (πινακίδα)που στην προκειμένη περίπτωση δίνεται από το χρήστη και αυτή αναζητάει στο ledger αν υπάρχει η συγκεκριμένη καταχώρηση με αυτό το αναγνωριστικό. Στη συνέχεια υπάρχει η μέθοδος initLedger που καλείται από το αρχείο startFabric και «γεμίζει» το ledger με δεδομένα. Η μέθοδος queryAllCars επιστρέφει όλες τις καταχωρήσεις που υπάρχουν στο ledger και δεν παίρνει ορίσματα, παρά μόνο επιστρέφει ένα JSON string με τα δεδομένα. Οι μέθοδοι nextYear, nextMonth παίρνουν σαν ορίσματα την πινακίδα του οχήματος και την ημερομηνία και πραγματοποιούν τις αντίστοιχες αλλαγές στο ledger. Επίσης, η μέθοδος recordCar παίρνει τα ορίσματα που δίνονται από το χρήστη και κάνει μια καινούργια καταχώρηση στο ledger. Τέλος, υπάρχει η συνάρτηση main η οποία ξεκινάει το chaincode σε ένα container του Docker κατά τη διαδικασία την εγκατάστασης.

```
/*
 * main function *
calls the Start function
The main function starts the chaincode in the container during instantiation.
 */
```

```
func main() {

	// Create a new Smart Contract
	err := shim.Start(new(SmartContract))
	if err != nil {
		fmt.Printf("Error creating new Smart Contract: %s", err)
	}
}
```

Αφού λοιπόν έχουμε το δίκτυο του blockchain ενεργό, έχει προγραμματιστεί το αρχείο που εκτελεί τα chaincode μπορούμε χειροκίνητα να κάνουμε invoke στον ledger του blockchain και για παράδειγμα να αναζητήσουμε ένα όχημα ή να εκτυπώσουμε όλα όσα υπάρχουν αποθηκευμένα μέσα σε αυτό. Αυτό μπορεί να γίνει εύκολα από το terminal εκτελώντας την παρακάτω εντολή.

```
docker exec -e "CORE_PEER_LOCALMSPID=Org1MSP" -e "CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp" cli peer chaincode invoke -o orderer.example.com:7050 -C mychannel -n kteo-app -c '{"function":"queryAllCars","Args":[""]}'
```

Με αυτόν τον τρόπο θα εμφανίσουμε ότι υπάρχει με πυρήνα του ledger. Μπορούμε επίσης να καλέσουμε και διαφορετικές μεθόδους αρκεί να αλλάξουμε το όνομα του function και τις τιμές στο Args.

Παρακάτω θα αναλυθεί ο τρόπος που θα συνδέσουμε το blockchain με το σύστημα του ΚΤΕΟ έτσι ώστε να μπορεί ένας χρήστης να κάνει invoke σε αυτό από το web service και όχι από το terminal καθώς αυτό θα προκαλούσε προβλήματα και θα δυσκόλευε τους χρήστες. Για την σύνδεση χρειάζεται να

τροποποιήσουμε το αρχείο index.html, να προσθέσουμε κομμάτια κώδικα στο server αρχείο και να φτιάξουμε και ένα καινούργιο αρχείο που θα είναι ο μεσάζων μεταξύ του server, της οθόνης που αλληλοεπιδράει ο χρήστης και του αρχείου που εκτελεί τα smart contracts και θα το ονομάσουμε controller.js

Για να εκτυπώσουμε στο html αρχείο, όλα τα οχήματα που υπάρχουν στο ledger πρέπει πρώτα να φτιάξουμε τον πίνακα που θα φιλοξενήσει τα δεδομένα που θα σταλούν από αυτόν. Αρχικά, χρησιμοποιούνται οι βιβλιοθήκες του jQuery, Bootstrap καθώς επίσης και Angular. Ορίζουμε στο <body> της html ότι χρησιμοποιούμε ng-app και το ονομάζουμε application, αυτό γίνεται για να δείξουμε στο html αρχείο ότι τα δεδομένα μέσα στο body μπορούν να έχουν AngularJS κώδικα. Επίσης ορίζουμε ng-controller και το ονομάζουμε appController, αυτό βοηθάει να διαχειρίζεται κάποιος τα element μέσα στον html κώδικα σαν αντικείμενα.

```
<div id="body">
    <div class="form-group">
     <label>Query All Cars Catches</label>
     <p><input id="queryAllCars" type="submit" value="Query" class="btn btn-primary" ng-click="queryAllCars()"></p>
    </div>
```

Δημιουργείται το κουμπί που αυτό όταν πατηθεί καλείται η αντίστοιχη συνάρτηση που έχει οριστεί στο ng-click.

```
<table id="all_cars" class="table" align="center">

      <tr>
       <th>ID</th>
       <th>FistName</th>
       <th>LastName</th>
```

```
  <th>Model</th>
  <th>Type</th>
  <th>ExDate</th>
</tr>

<tr ng-repeat="car in all_cars">
  <td>{{car.plate}}</td>
  <td>{{car.fname}}</td>
  <td>{{car.lname}}</td>
  <td>{{car.model}}</td>
  <td>{{car.type}}</td>
  <td>{{car.exdate}}</td>
</tr>
    </table>
```

Φτιάχνουμε τον πίνακα που θα φιλοξενήσει τα δεδομένα. Επειδή αυτά θα είναι περισσότερα από ένα χρησιμοποιούμε το ng-repeat που εκτυπώνει όλα τα δεδομένα που υπάρχουν στην μεταβλητή all_cars. Σειρά τώρα έχει η δημιουργία των scripts που θα κάνουν request στο server και στη συνέχεια θα μεταφέρουν τα δεδομένα στον πίνακα.

```
    var app = angular.module('application', []);
    // Angular Controller
    app.controller('appController', function ($scope, appFactory) {

    $scope.queryAllCars = function () {

        appFactory.queryAllCars(function (data) {
          var array = [];
          console.log(data.data[1].Record);
```

```
                for (var i = 0; i < data.data.length; i++) {
                  parseInt(data.data[i].plate);
                  data.data[i].Record.plate = parseInt(data.data[i].plate);
                  array.push(data.data[i].Record);
                }
                array.sort(function (a, b) {
                  return parseFloat(a.Key) - parseFloat(b.Key);
                });
                // console.log(array);
                $scope.all_cars = array;
              });
            }
          });

          //Δεύτερο script
          // Angular Factory
          app.factory('appFactory', function ($http) {

          var factory = {};
         factory.queryAllCars = function (callback) {
          $http({
        method: 'GET',
        url: '/get_all_cars/'
}).then(function (output) {
        callback(output)
}, function (error) {

});
}
return factory;
```

```
});
```

Το $scope.queryAllCars διαχειρίζεται τον πίνακα που φτιάχτηκε παραπάνω σαν αντικείμενο. Στη συνέχεια, καλείται το δεύτερο script πού στέλνει ένα request στο server αρχείο, όταν ο server στείλει τα δεδομένα ως JSON data από το ledger, δίνονται πάλι στο πρώτο script και αυτό τα μετατρέπει σε Array ώστε να μπορούν να εκτυπωθούν στον πίνακα με τη βοήθεια του $scope.

Στη συνέχεια προσθέτουμε στο server αρχείο τον εξής κώδικα ώστε να λάβει τα δεδομένα που αποστέλλονται από τα scripts και να τα προωθήσει στο controller για να επικοινωνήσει με τη σειρά του με το blockchain. Για να μπορεί να διαβαστεί το αρχείο controller.js πρέπει να δηλώσουμε την ύπαρξη του στο αρχείο του server.

```
//Γίνεται import το controller
var car = require('../kteo-app/controller.js');
//Μέσω το get γίνεται το request από το frontend στο blockchain
app.get('/get_car/:id', function (req, res) {
        car.get_car(req, res);
});
```

Τελευταίο βήμα είναι η σύνταξη του controller αρχείου ώστε να μπορέσει να επικοινωνήσει με το chaincode. Σε αυτό για αρχή δηλώνουμε όλα τα πακέτα που θα μας χρειαστούν.

```
// call the packages we need
var express = require('express');       // call express
var app = express();                    // define our app using express
var bodyParser = require('body-parser');
var http = require('http')
```

```
var fs = require('fs');
var Fabric_Client = require('fabric-client');
var path = require('path');
var util = require('util');
var os = require('os');
```

Μετά από τα import των πακέτων συντάσσουμε τον κώδικα ώστε να επιστρέψει τα δεδομένα που θα λάβει από το blockchain πίσω στο server.

```
module.exports = (function () {
	return {
	//εδώ καλούνται οι μέθοδοι που έχουν αναπτυχθεί μέσα στο αρχείο GO
		get_all_cars: function (req, res) {}
	}
})();
```

Μέσα στο function αρχικά γίνονται οι προετοιμασίες ώστε να συνδεθεί το σύστημα με το blockchain, το channel, δηλώνεται η peer και το port που έχει και αποθηκεύεται με μία μεταβλητή το μονοπάτι που έχουν αποθηκευτεί τα κλειδιά του blockchain.

```
var fabric_client = new Fabric_Client();

// setup the fabric network
var channel = fabric_client.newChannel('mychannel');
var peer = fabric_client.newPeer('grpc://localhost:7051');
channel.addPeer(peer);
```

```
var member_user = null;
var store_path = path.join(os.homedir(), '.hfc-key-store');
console.log('Store path:' + store_path);
var tx_id = null;
```

Στη συνέχεια δημιουργείται ο αποθηκευτικός χώρος που το κανάλι (Channel) χρησιμοποιεί για να αποθηκεύσει ευαίσθητες πληροφορίες, όπως το πιστοποιημένο, ιδιωτικό κλειδί του χρήστη. Δημιουργείται η κλάση newCryptoSuite που βοηθάει στην κρυπτογράφηση αλγορίθμων και κλειδιών και γίνεται η εγγραφή του χρήστη που θα κάνει όλα τα αιτήματα.

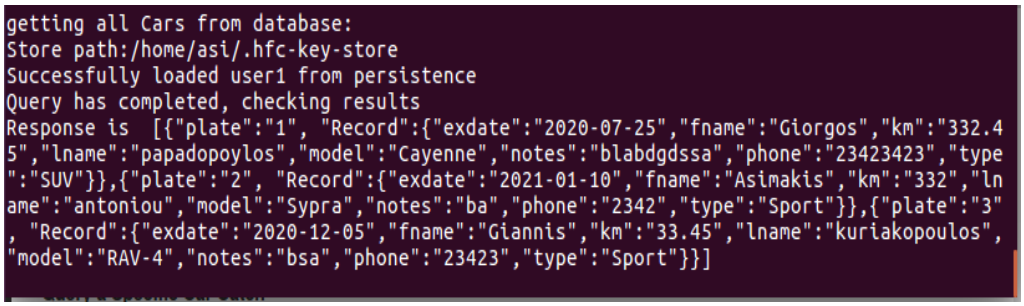
```
// create the key value store as defined in the fabric-client/config/default.json 'key-value-store' setting
                    Fabric_Client.newDefaultKeyValueStore({
                            path: store_path
                    }).then((state_store) => {
                    // assign the store to the fabric client
                    fabric_client.setStateStore(state_store);
                    var crypto_suite = Fabric_Client.newCryptoSuite();
// use the same title for the state store (where the users' certificate are kept)
                    // and the crypto store (where the users' keys are kept)
        var crypto_store = Fabric_Client.newCryptoKeyStore({ path: store_path });
                    crypto_suite.setCryptoKeyStore(crypto_store);
                    fabric_client.setCryptoSuite(crypto_suite);

        // get the enrolled user from persistence, this user will sign all requests
                    return fabric_client.getUserContext('user1', true);
```

```
                }).then((user_from_store) => {
                        if (user_from_store && user_from_store.isEnrolled()) {
                                console.log('Successfully loaded user1 from persistence');
                                member_user = user_from_store;
                        } else {
                                throw new Error('Failed to get user1.... run registerUser.js');
                        }

                        // queryAllCars - requires no arguments , ex: args: [''],
                        const request = {
                                chaincodeId: 'kteo-app',
                                txId: tx_id,
                                fcn: 'queryAllCars',
                                args: ['']
                        };

                        // send the query proposal to the peer
                        return channel.queryByChaincode(request);

                }).then((query_responses) => {
                        console.log("Query has completed, checking results");
                        // query_responses could have more than one results if there multiple peers were used as targets
                        if (query_responses && query_responses.length == 1) {
```

```
                                        if (query_responses[0] instanceof Error) {
                                                console.error("error from query =
", query_responses[0]);
                                        } else {
                                                console.log("Response is ",
query_responses[0].toString());

        res.json(JSON.parse(query_responses[0].toString()));
                                        }
                                } else {
                                        console.log("No payloads were returned
from query");
                                }
                        }).catch((err) => {
                                console.error('Failed to query successfully :: ' +
err);
                        });
                }
```

Καλείται η μέθοδος από το .GO αρχείο που θα πραγματοποιήσει το smart contract και στη συνέχεια το .GO αρχείο επιστρέφει μια κατάσταση που ελέγχεται από το controller αν όλα πήγαν σωστά. Μετά αν όλα είναι σωστά στέλνεται ένα JSON αρχείο από τον controller πίσω στο html αρχείο στο script του app.Factory και εκεί με τη βοήθεια της callback συνάρτησης στέλνονται τα δεδομένα στο πρώτο script και από εκεί «τυπώνονται» στον πίνακα.

Αν όλα πραγματοποιηθούν σωστά το αποτέλεσμα που εμφανιστεί όταν ο χρήστης πάει στο Validate και πατήσει να εκτυπώσει όλα τα αυτοκίνητα που υπάρχουν στο blockchain θα δει το παρακάτω αποτέλεσμα στο terminal.



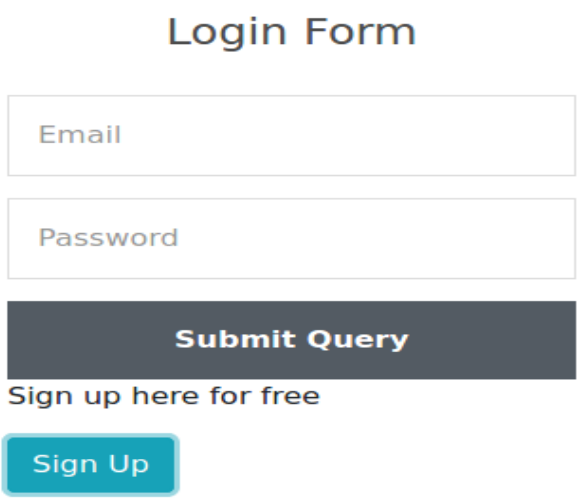
Εικόνα 14: Results From Querying All Cars in Terminal

Με παρόμοιο τρόπο φτιάχνονται και καλούνται και οι υπόλοιπες συναρτήσεις και με αυτόν τον τρόπο δίνουν λειτουργικότητα στο frontend που αλληλοεπιδράει ο χρήστης με το blockchain. Όσες συναρτήσεις παίρνουν ορίσματα αλλάζει ο τρόπος που καλείται όπως φαίνεται παρακάτω.

```
var request = {
                    chaincodeId: 'kteo-app',
                    fcn: 'nextYear',
                    args: [key, date],
                    chainId: 'mychannel',
                    txId: tx_id
                };
```

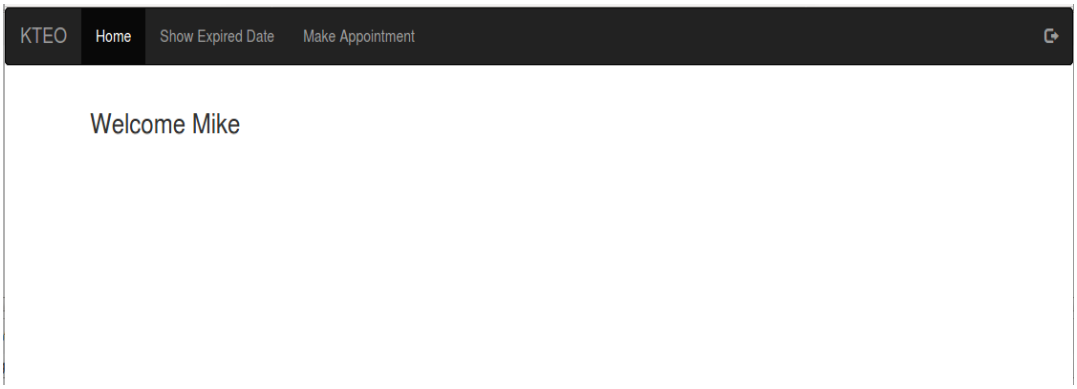
6.2 ΣΕΝΑΡΙΑ ΧΡΗΣΗΣ

Όταν κάποιος πατήσει την διεύθυνση της σελίδας θα του εμφανιστεί:



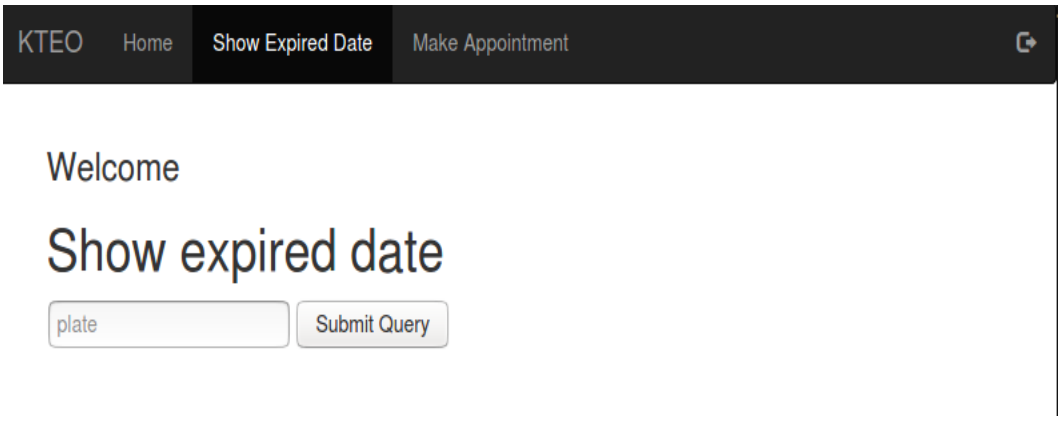
Εικόνα 15: Login Page

Αν λοιπόν ο χρήστης που κάνει Login έχει για privilege = 'U', δηλαδή είναι απλός χρήστης τότε θα τον ανακατευθύνει στην αρχική σελίδα που βλέπει κάθε χρήστης.



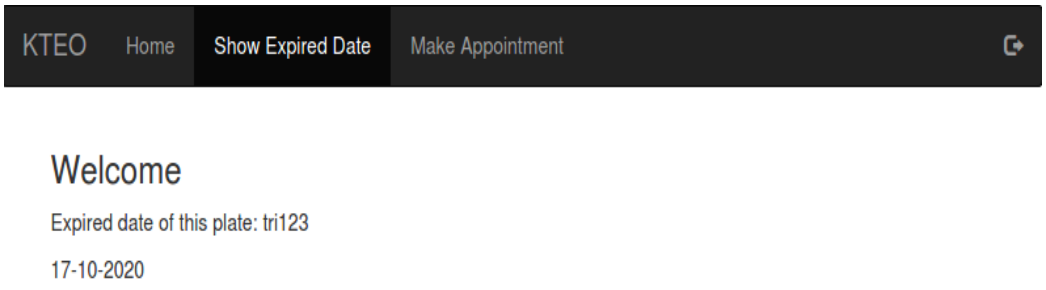
Εικόνα 16: User Home Page

Από εκεί ο χρήστης έχει τη δυνατότητα να δει πότε λήγει το ΚΤΕΟ του οχήματός του αν μετακινήσει τον κέρσορα και στην μπάρα πλοήγησες πατήσει το Show Expired Date.



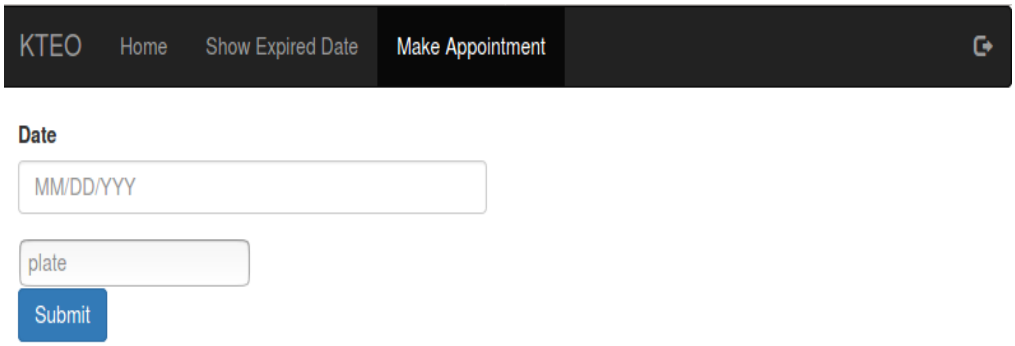
Εικόνα 17: Show Expiration Date

Του ζητείται να πληκτρολογήσει την πινακίδα τόσο για επιβεβαίωση όσο και στην περίπτωση που διαθέτει παραπάνω από ένα όχημα.



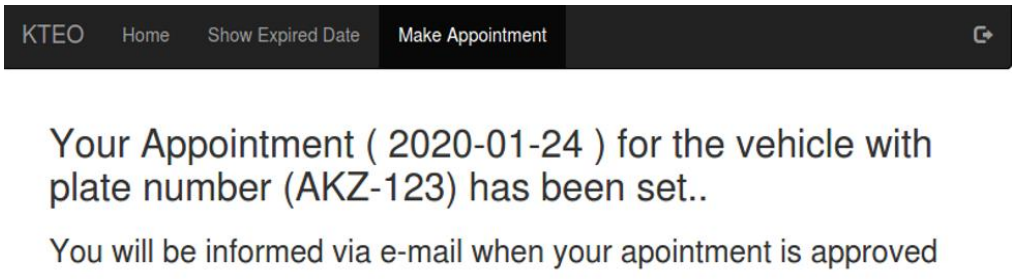
Εικόνα 18: Show Expiration Date Results

Πατώντας στην επιλογή του Make Appointment του εμφανίζεται η παρακάτω οθόνη όπου του ζητείται να διαλέξει την ημερομηνία που επιθυμεί για το ραντεβού, καθώς επίσης και την πινακίδα του οχήματος.



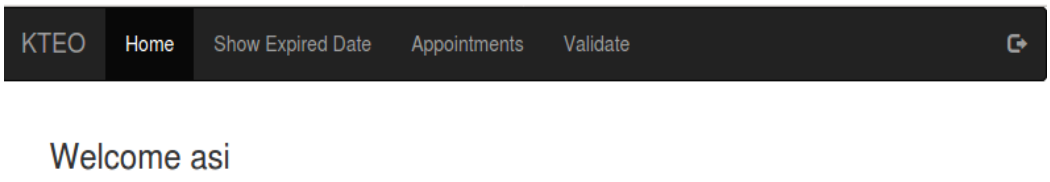
Εικόνα 19: Make Appointment

Αν όλα γίνουν με επιτυχία, παρακάτω φαίνεται τι θα εμφανιστεί στο χρήστη.



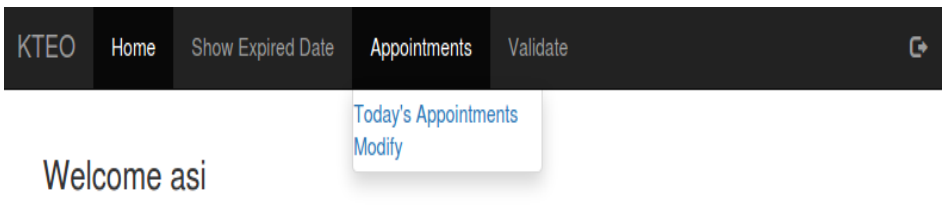
Εικόνα 20: Make Appointment Results

Στην περίπτωση που κάνει Login ένας υπάλληλος του ΚΤΕΟ με privilege S (Γραμματεία) θα εμφανιστεί διαφορετική αρχική οθόνη από αυτή του απλού χρήστη. Αυτή είναι η παρακάτω.



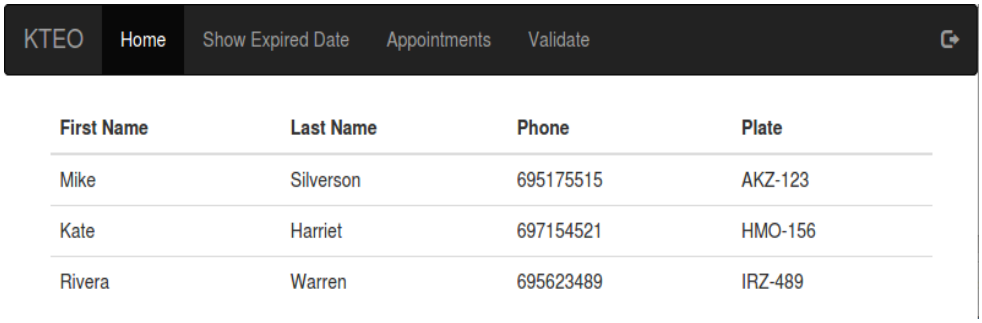
Εικόνα 21: Secretary Home Page

Η λειτουργία του Show Expired Date είναι η ίδια με αυτή που είδαμε παραπάνω. Η άλλη λειτουργία που έχει η γραμματεία είναι το Appointments που πρόκειται για ένα Drop down menu με τις επιλογές Today's Appointments και Modify.



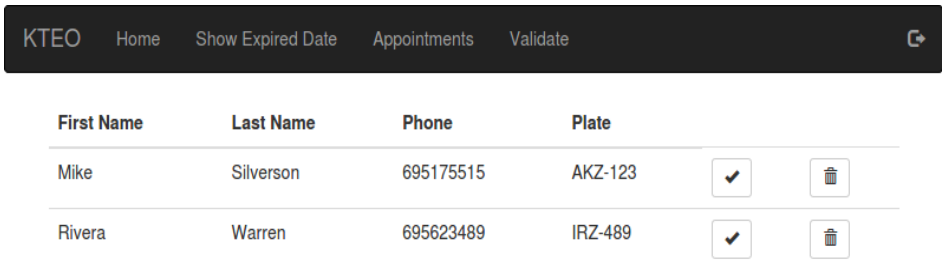
Εικόνα 22: Drop - Down Menu

Αν επιλεχθεί η επιλογή Today's Appointments εμφανίζεται το παρακάτω αποτέλεσμα.



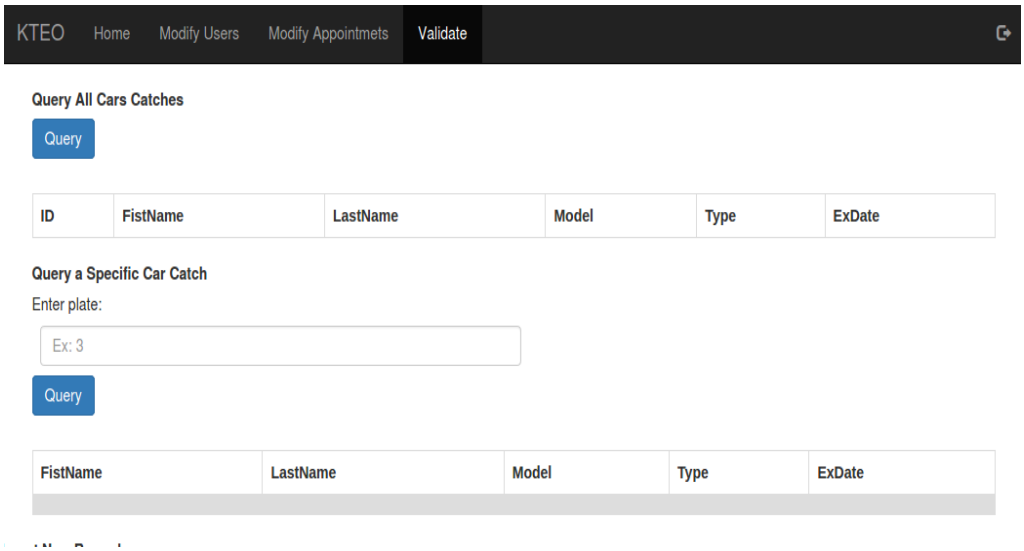
Εικόνα 23: Results of Today's Appointments

Αν επιλεχθεί η επιλογή Modify εμφανίζονται όλες οι αιτήσεις που έχουν κάνει οι χρηστές.



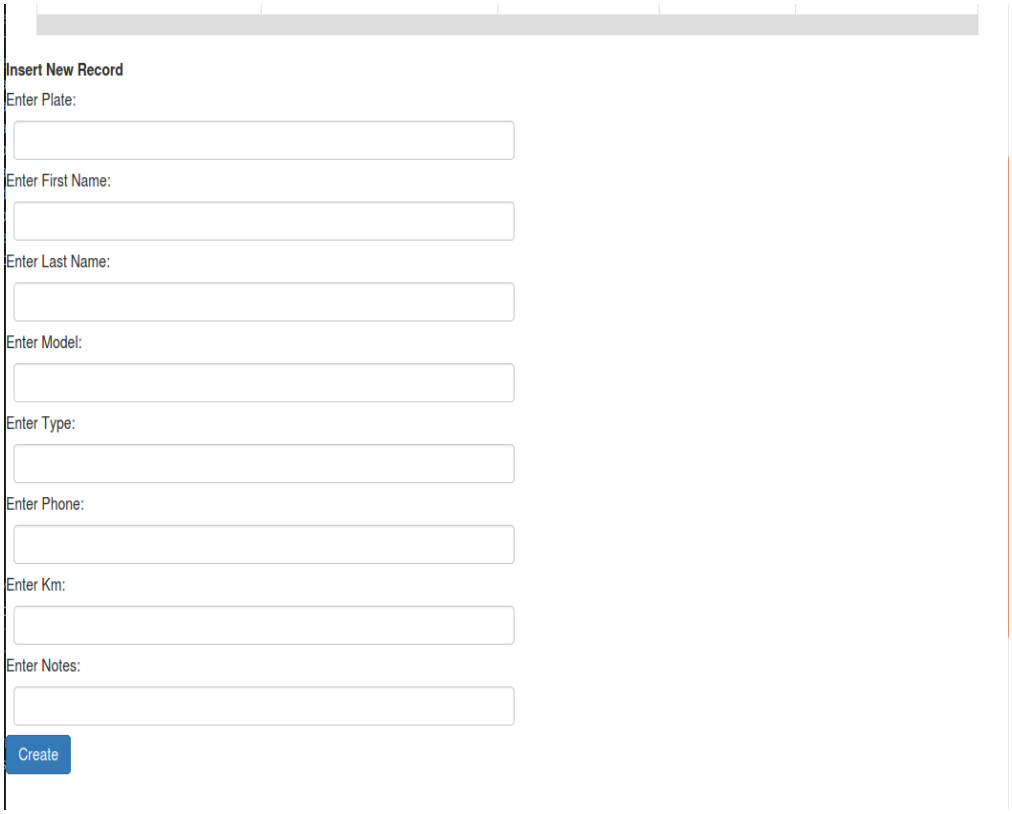
Εικόνα 24: Results of Modify Appointments

Παράλληλα υπάρχει και η λειτουργία Validate που αναφέρθηκε παραπάνω.



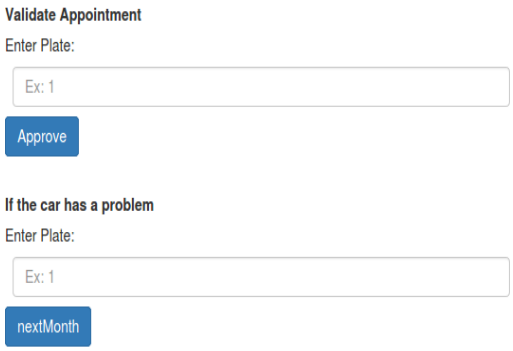
Εικόνα 25: Validation Page, Query All Cars and Specific Car

Εδώ φαίνονται οι λειτουργίες της εμφάνισης όλων των δεδομένων που υπάρχουν στο Blockchain και από κάτω η αναζήτηση ενός με βάση την πινακίδα. Παρακάτω φαίνεται η λειτουργία της προσθήκης ενός καινούργιου πελάτη.



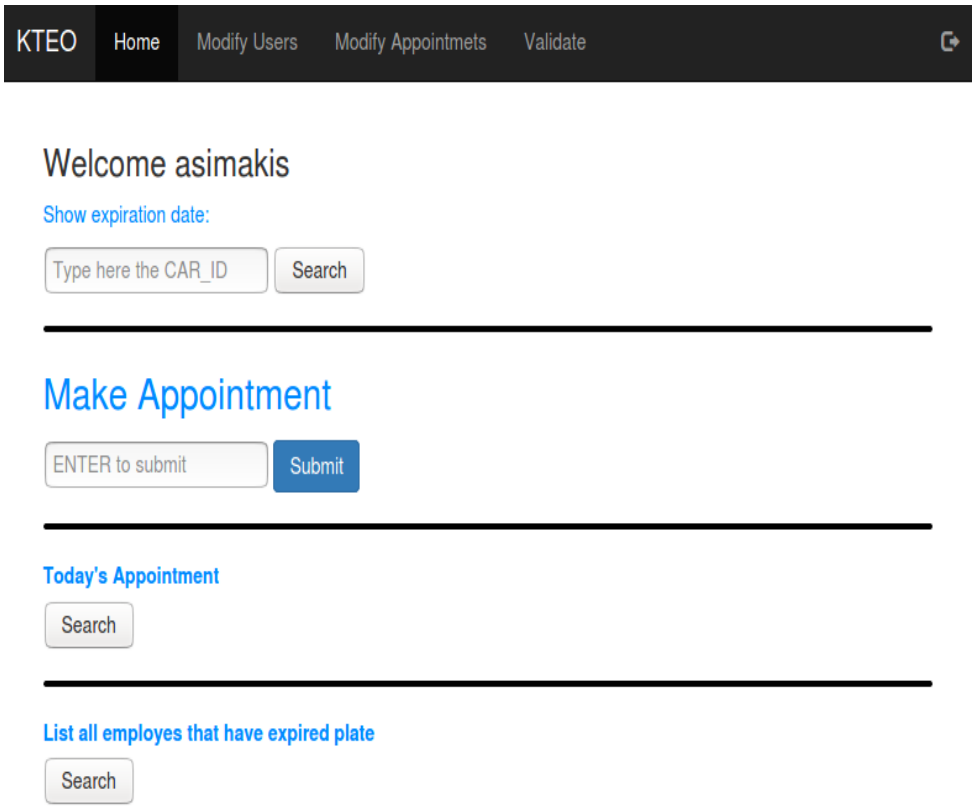
Εικόνα 26: Validation Page, Insert New Record

Και στη συνέχεια φαίνεται η τελευταία λειτουργία που αλλάζει την ημερομηνία ΚΤΕΟ ενός οχήματος με βάση την πινακίδα.



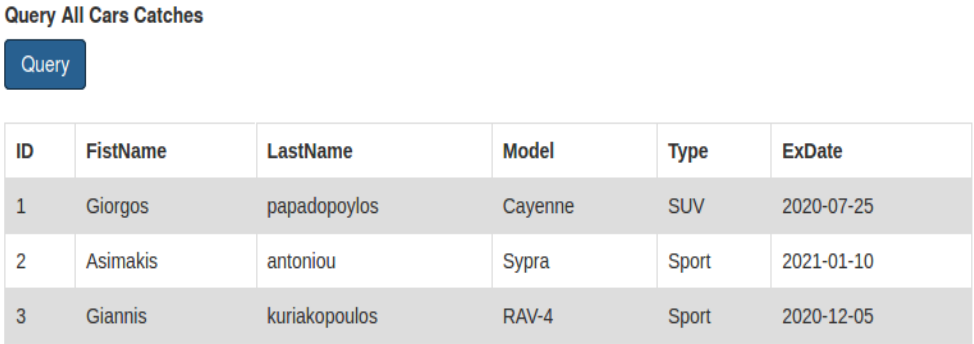
Εικόνα 27: Validation Page, Reschedule Appointment

Τέλος ο τελευταίος χρήστης που κάνει Login και έχει διαφορετική αρχική οθόνη από τους παραπάνω χρήστες (Secretary, User) είναι ο διαχειριστής (Admin). Τέλος ο τελευταίος χρήστης που κάνει Login και έχει διαφορετική αρχική οθόνη από τους παραπάνω χρήστες (Secretary, User) είναι ο διαχειριστής (Admin). Η λειτουργίες Modify Appointments και το Validate είναι ίδιες με αυτές που υπάρχουν στον υπάλληλο της γραμματείας.



Εικόνα 28: Admin Home Page

Αν όλα γίνουν όπως αναλύθηκαν πιο πάνω και αφού έχει δημιουργεί το δίκτυο του Blockchain και το έχουμε γεμίσει με δεδομένα, στην περίπτωση που πάμε στο validate και πατήσουμε να εμφανίσουμε όλα τα δεδομένα βλέπουμε αυτό το αποτέλεσμα.



ID	FistName	LastName	Model	Type	ExDate
1	Giorgos	papadopoylos	Cayenne	SUV	2020-07-25
2	Asimakis	antoniou	Sypra	Sport	2021-01-10
3	Giannis	kuriakopoulos	RAV-4	Sport	2020-12-05

Εικόνα 29: Results of Querying All Cars From Blockchain

7 Επιλογοσ

Είναι εύλογο επομένως πως η δημιουργία ενός δικτύου Blockchain είναι λίγο πολύπλοκη στην αρχή καθώς χρειάζεται την κατάλληλη εξοικείωση. Αυτό συμβαίνει επειδή η τεχνολογία αυτή είναι νέα και αναπτύσσεται ραγδαία. Στο μέλλον θα υπάρξουν πιο εξελιγμένα εργαλεία και κάποιες διαδικασίες ίσως να αυτοματοποιηθούν. Ο σκοπός της εργασίας είναι η διασφάλιση που προσφέρει το σύστημα ώστε η χρήστες να μην μπορούν να δηλώσουν ψευδή στοιχεία για τα οχήματά τους. Αυτό επιτυγχάνεται με την βοήθεια του Blockchain καθώς η αλλαγές που μπορούν να γίνουν είναι περιορισμένες. Επίσης, σε σχέση με τις κοινές βάσεις δεδομένων όταν γίνεται μια αλλαγή η προηγούμενη καταχώρηση διαγράφεται, αντιθέτως στο Blockhain τα προηγούμενα στοιχεία μένουν. Η δομή που προσφέρει το Blockchain μπορεί να θεωρηθεί απαράλλακτη καθώς υπάρχει κοινή καταγραφή όλων των σημαντικών γεγονότων από όλες τις εμπλεκόμενες οντότητες. Με τη χρήση της τεχνολογίας αυτή συνεπάγεται ότι η ανάγκη για μια έμπιστη οντότητα η οποία θα μεσολαβεί ώστε να πραγματοποιηθεί μια συναλλαγή, καταργείται. Όσο αναφορά το θεωρητικό κομμάτι της εργασίας, έμαθα τι είναι το Blockchain, τα κρυπτονομίσματα και τη λογική με την οποία αυτά λειτουργούν. Μέσο της εργασίας ενστερνίστηκα με την ιδέα για την μελλοντική χρήση του Blockchain τόσο σε επιχειρήσεις και στον ιδιωτικό τομέα, όσο και στο δημόσιο. Όσο το πρακτικό έμαθα ποικίλες γλώσσες προγραμματισμού όπως NodeJS, Go-lang, AngularJS. Οι δυσκολίες που συνάντησα ήταν κυρίως στο πρακτικό κομμάτι της εργασίας καθώς το Hyperledger Fabric είναι ακόμα σε πρώιμο στάδιο και η πληροφορίες για την ανάπτυξή του ήταν δυσεύρετες.

8 Πηγες Και Βιβλιογραφια

Βιβλιογραφια

[1] Desrosiers, L.(2019). Blockchain Development with Hyperledger. Packt

[2] Gaur, N(2018). Hands-On Blockchain with Hyperledger. Packt

[3] Παναγόπουλος, Ι.(2019). Ανάλυση τεχνολογίας Blockchain για την υλοποίηση πληροφοριακού συστήματος διαχείρισης ευαίσθητων ιατρικών δεδομένων.

[4] Cachin, C.(2016). Architecture of the Hyperledger Blockchain Fabric.

Δικτυογραφια

[1] Hyperledger, “An Introduction to Hyperledger” [Online]. Available: https://www.hyperledger.org/wp-content/uploads/2018/07/HL_Whitepaper_IntroductiontoHyperledger.pdf

[2] Hyperledger-fabric, “Introduction” [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatis.html#hyperledger-fabric

[3] Blocks Decoded, “What is Blockchain? A Simple Explanation in Easy Terms” [Online]. Available: https://blocksdecoded.com/what-is-blockchain/

[4] Medium, “Hyperledger-Chapter 1 | Blockchain Foundation” [Online]. Available: https://medium.com/swlh/hyperledger-chapter-1-foundation-7ad5bd94d452

[5] Medium, “Permissioned vs Permissionless blockchains:” [Online]. Available: https://medium.com/@dustindreifuerst/permissioned-vs-permissionless-blockchains-acb8661ee095

[6] Medium, “Exploring Fabric-CA: Registration and Enrollment” [Online]. Available:https://medium.com/@kctheservant/exploring-fabric-ca-registration-and-enrollment-1b9f4a1b3ace

[7] IBM, “Chaincode for GO developers, Part 1: Writing Blockchain chaincode in Go for Hyperledger Fabric v0.6” [Online]. Available: https://www.ibm.com/developerworks/cloud/library/cl-ibm-blockchain-chaincode-development-using-golang/cl-ibm-blockchain-chaincode-development-using-golang-pdf.pdf

[8] Medium, “Step by Step Towards Hyperledger Fabric-Part 1” [Online]. Available: https://medium.com/@kotsbtechcdac/step-by-step-towards-hyperledger-fabric-part-1-c867fc5fe18

[9] hyperledger-fabric, “Building Your First Network” [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/build_network.html

[10] hyperledger-fabric, “Writing Your First Application” [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/write_first_app.html

[11] Medium, “Understanding the Node-SDK of Hyperledger and Running the First Example” [Online]. Available: https://medium.com/mlg-blockchain-

consulting/understanding-the-node-sdk-of-hyperledger-and-running-the-first-example-5f6753393ec8

[12] How Bitcoin Works Image extracted from https://blockgeeks.com/guides/what-is-bitcoin/

[13]How Ethereum Works Image extracted from https://blockgeeks.com/guides/ethereum/#What_is_a_Ethereum_smart_contract

[14] How Transaction Works Image extracted from https://thehungryjpeg.com/product/3621996-smart-contract-how-smart-contracts-work-in-blockchain-with-cryptocurr

[15] Blockchain and Smart Contract Diagram Image extracted from https://medium.com/swlh/hyperledger-chapter-1-foundation-7ad5bd94d452

[16] Centralized Databases VS Blockchain. Image extracted from https://medium.com/swlh/hyperledger-chapter-1-foundation-7ad5bd94d452