ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Τίτλος Πτυχιακής Εργασίας	Δημιουργία πληροφοριακού συστήματος αξιοποιώντας την τεχνολογία Blockchain για την προστασία A Blockchain-based information system for managing privacy of data
Ονοματεπώνυμο Φοιτητή	ΚΩΝΣΤΑΝΤΙΝΟΣ ΒΑΛΛΑΣ
Πατρώνυμο	ΧΑΡΑΛΑΜΠΟΣ
Αριθμός Μητρώου	Π/ 19024
Επιβλέπων	Δημήτριος Αποστόλου

Ημερομηνία Παράδοσης Σεπτέμβριος 2024

Περίληψη

Σκοπός της παρούσας πτυχιακής εργασίας είναι η ανάπτυξη ενός κατάλληλου πληροφοριακού συστήματος το οποίο θα επικοινωνεί με blockchain network του οποίου ο ledger περιέχει στοιχεία περιουσιακών ακινήτων (real estate properties), συγκεκριμένα τον αριθμό υπνοδωματίων, τον αριθμό των μπάνιων και τον όροφο (σε περίπτωση όπου αναφερόμαστε σε διαμέρισμα). Με το σύστημα αυτό θα επιτυγχάνεται η πλήρης ασφάλεια και εμπιστευτικότητα χρησιμοποιώντας το hyperledger fabric framework και το blockchain δίκτυο
που θα πρέπει να στήσουμε με βάση τα real estate properties values που αναφέραμε πιο πάνω. Το σύστημα θα 'τροφοδοτεί' με δεδομένα το web application που θα δημιουργήσουμε για την παρουσίαση των στοιχείων με ωραιοποιημένο τρόπο από την οπτική του χρήστη.

Τις τιμές των μεταβλητών εισόδου τις παίρνουμε από ένα σύνολο δεδομένων (dataset) που περιλαμβάνει δεδομένα για 10 ακίνητα. Το σύνολο δεδομένων περιέχεται στο ledger του blockchain δικτύου..

Χρησιμοποιώντας το hyperledger fabric java sdk, δημιούργησα ένα API με το Spring Boot framework (σε eclipse), το οποιο επικοινωνεί με το blockchain δίκτυο μέσω του fabcar chaincode. Ανέπτυξα συγκεκριμένα requests για την εξαγωγή των κατάλληλων πληροφοριών από το ledger. Χρησιμοποιώ το API για να επιτυγχάνεται η επικοινωνία με το blockchain δίκτυο και το web application για την παρουσίαση των πληροφοριών με εμφανίσιμο τρόπο για τον τελικό χρήστη. Στις πληροφορίες αυτές περιλαμβάνονται στοιχεία για τη ακίνητη περιουσία, εικόνες του ακινήτου καθώς και το όνομα του ιδιοκτήτη.

Το απαιτούμενο πληροφοριακό σύστημα αναπτύχθηκε στο περιβάλλον ανάπτυξης προγραμματισμού Eclipse σε γλώσσα προγραμματισμού Java.

Λέξεις κλειδιά: Blockchain δίκτυο, Ακίνητες περιουσίες, Ledger, Hyperledger Fabric, chaincode, ανάπτυξη API.

Abstract

The purpose of this thesis is to develop an appropriate information system that will communicate with a blockchain network whose ledger contains data on real estate properties, specifically the number of bedrooms, the number of bathrooms, and the floor (in cases where we refer to an apartment). This system will ensure complete security and confidentiality by using the Hyperledger Fabric framework and the blockchain network that we need to set up based on the aforementioned real estate property values. This system will supply data to a web application that we will create to present the information in a user-friendly manner.

The input variable values are taken from a dataset that includes data for 10 properties. This dataset is contained within the ledger of the blockchain network. Using the Hyperledger Fabric Java SDK, I created an API with the Spring Boot framework (in Eclipse), which communicates with the blockchain network via the Fabcar chaincode. I developed specific requests to extract the appropriate information from the ledger. The API is used to facilitate communication with the blockchain network, while the web application presents the information in an attractive manner for the end user. This information includes details about the real estate properties, images of the properties, and the owner's name.

The required information system was developed in the Eclipse development environment using the Java programming language.

Keywords: Blockchain network, Real estate properties, Ledger, Hyperledger Fabric, chaincode, API development.

Περιεχόμενα

Εισαγωγή 6
Θεωρία Hyperledger Fabric 8
2.1 Επισκόπηση της τεχνολογίας Blockchain 8
2.2 Εισαγωγή στα Hyperledger και Hyperledger Fabric 9
2.3 Αρχιτεκτονική του Hyperledger Fabric 10
2.4 Chaincode και Smart Contracts 12
2.5 Transaction Flow στο Hyperledger Fabric 13
2.6 Consensus Mechanism 14
2.7 Λειτουργίες για Privacy και Confidentiality 14
2.8 Membership και Identity Management 15
2.9 Διαδικασία Development και Deployment 15
2.10 Governance και Management 16
2.11 Use Cases και Applications 17
2.12 Σύγκριση με άλλα Blockchain Frameworks 18
2.13 Δοκιμασίες και Μελλοντικές Κατευθύνσεις 19
2.14 Σύνοψη 19
Η Λογική του Hyperledger Fabric Πληροφοριακού Συστήματος 20
3.1 Αρχιτεκτονική του Συστήματος 20
3.2 Ροή Δεδομένων και Αλληλεπίδραση με το Blockchain δίκτυο 21
3.3 Έξυπνα Συμβόλαια (Smart Contracts) 21
3.4 Δεδομένα Ασφάλειας και Ακεραιότητα 22
3.5 Ανάπτυξη του API 22
3.6 Ανάπτυξη Διαδικτυακής Εφαρμογής 23
Ανάπτυξη Πληροφοριακού Συστήματος με τη τεχνολογία Hyperledger Fabric 24
4.1 Διατύπωση του Προβλήματος 24
4.2 Δεδομένα Περιουσιακών Ακινήτων 24
4.3 Διαδικασία Ανάπτυξης Πληροφοριακού Συστήματος με Hyperledger Fabric Framework 24
4.4 Ο Ρόλος του Chaincode στο Πληροφοριακό Σύστημα 26
4.5 Λειτουργίες του API και Επικοινωνία με το Blockchain δίκτυο 27
4.6 Ο Ρόλος του Web Application 27
4.7 Αποτελέσματα 27
Συμπεράσματα 30
Βιβλιογραφία 31
Παράρτημα Α 32

Κατάλογος Εικόνων

Εικόνα 1. Συνοπτική περιγραφή της τεχνολογίας blockchain
Εικόνα 2. Δομή του ledger 9
Εικόνα 3. Κύκλος ζωής του chaincode 11
Εικόνα 4. Transaction flow στο Hyperledger Fabric 13
Εικόνα 5. Επικοινωνία εφαρμογής και blockchain δικτύου μέσω των smart contracts 16
Εικόνα 6. Διαχωρισμός public,private, permissioned και permissionless blockchain 18
Εικόνα 7. Διάγραμμα ροής λογικής του πληροφοριακού συστήματος 23
Εικόνα 8. Αρχική σελίδα του web application (1/2) 25
Εικόνα 9. Αρχική σελίδα του web application (2/2) 26
Εικόνα 10. Δημιουργία καινούργιας ακίνητης περιουσίας 26
Εικόνα 11. Επεξεργασία δεδομένων ακίνητης περιουσίας 27

Κεφάλαιο 1ο

Εισαγωγή

Στη σημερινή ψηφιακή εποχή, η ασφάλεια των ευαίσθητων δεδομένων αποτελεί ένα πιεστικό ζήτημα σε όλους τους κλάδους, από τα ακίνητα μέχρι την υγειονομική περίθαλψη. Καθώς ο όγκος των δεδομένων συνεχίζει να αυξάνεται, αυξάνονται και οι κίνδυνοι παραβίασης δεδομένων και μη εξουσιοδοτημένης πρόσβασης. Μία από τις κορυφαίες τεχνολογικές καινοτομίες που έχουν σχεδιαστεί για να μετριάσουν αυτούς τους κινδύνους είναι η τεχνολογία blockchain. Αρχικά εισήχθη μέσω κρυπτονομισμάτων, όπως το Bitcoin, αλλά έκτοτε έχει επεκταθεί σε διάφορους τομείς, προσφέροντας αποκεντρωμένες, αμετάβλητες και διαφανείς λύσεις για τη διαχείριση των δεδομένων με ασφάλεια. Αυτή η στροφή προς τις λύσεις blockchain σηματοδοτεί την αυξανόμενη ανάγκη για τεχνολογίες που διασφαλίζουν την εμπιστοσύνη και την ακεραιότητα των δεδομένων χωρίς εξάρτηση από μια κεντρική αρχή.

Αυτή η διπλωματική εργασία εξετάζει την ανάπτυξη ενός συστήματος πληροφοριών βασισμένου σε blockchain, ειδικά σχεδιασμένου για τη διαχείριση ευαίσθητων δεδομένων που σχετίζονται με ακίνητα. Ο στόχος είναι η ασφαλής διαχείριση λεπτομερειών ακινήτων, όπως ο αριθμός των υπνοδωματίων, των μπάνιων και το επίπεδο του ορόφου, χρησιμοποιώντας το Hyperledger Fabric, ένα πλαίσιο blockchain σχεδιασμένο για επιχειρηματική χρήση. Αξιοποιώντας το Hyperledger Fabric, το αναπτυγμένο σύστημα ενισχύει την εμπιστευτικότητα, την αμετάβλητη φύση των δεδομένων και την αποκέντρωση, απαραίτητα για την προστασία πληροφοριών υψηλής αξίας.

Βασικοί στόχοι περιλαμβάνουν:

- Τη δημιουργία ενός δικτύου blockchain που αποθηκεύει και επεξεργάζεται με ασφάλεια δεδομένα ακινήτων.
- Την ανάπτυξη μιας διαδικτυακής εφαρμογής που αλληλεπιδρά με αυτό το δίκτυο blockchain για φιλική παρουσίαση δεδομένων προς τον χρήστη.
- Τη χρήση του Hyperledger Fabric για τη διασφάλιση της ασφάλειας και της εμπιστευτικότητας των δεδομένων μέσω των μηχανισμών συναίνεσης και των έξυπνων συμβολαίων του.

Οι συναλλαγές και η διαχείριση δεδομένων στον τομέα των ακινήτων βασίζονται παραδοσιακά σε κεντρικά συστήματα, τα οποία μπορεί να είναι ευάλωτα σε παραβιάσεις δεδομένων, σφάλματα και αναποτελεσματικότητα. Η τεχνολογία blockchain, ειδικά με πλαίσια όπως το Hyperledger Fabric, προσφέρει μια μετασχηματιστική προσέγγιση σε αυτές τις προκλήσεις. Με την αποκέντρωση της διαδικασίας διαχείρισης δεδομένων, το blockchain μειώνει την εξάρτηση από τρίτους και διασφαλίζει ότι τα δεδομένα ακινήτων καταγράφονται με ασφάλεια, με διαφάνεια και χωρίς δυνατότητα αλλοίωσης. Αυτό το σύστημα είναι κρίσιμο για την επίτευξη ασφαλέστερων, αποδοτικότερων και πιο αξιόπιστων συναλλαγών στον τομέα των ακινήτων.

Αυτή η διπλωματική εργασία χωρίζεται στα ακόλουθα κεφάλαια:
Το Κεφάλαιο 2 παρέχει μια ολοκληρωμένη ανασκόπηση της τεχνολογίας Hyperledger Fabric και των αρχιτεκτονικών της στοιχείων.

Το Κεφάλαιο 3 περιγράφει τη λογική πίσω από το πληροφοριακό σύστημα και της ενσωμάτωσης του Hyperledger Fabric πλαισίου σε αυτή.
Το Κεφάλαιο 4 παρουσιάζει τη διαδικασία ανάπτυξης του πληροφοριακού συστήματος, συμπεριλαμβανομένου του σχεδιασμού και της ανάπτυξης του δικτύου blockchain.
Το Κεφάλαιο 5 προσφέρει συμπεράσματα σχετικά με τον αντίκτυπο της τεχνολογίας blockchain στη διαχείριση ευαίσθητων δεδομένων και τις μελλοντικές κατευθύνσεις ανάπτυξης.
Μέχρι το τέλος αυτής της διπλωματικής εργασίας, ο αναγνώστης θα έχει αποκτήσει μια εικόνα για το πώς το blockchain, και συγκεκριμένα το Hyperledger Fabric, μπορεί να αξιοποιηθεί για την ανάπτυξη ενός ασφαλούς και αποδοτικού πληροφοριακού συστήματος, εφαρμόσιμου σε κλάδους που απαιτούν υψηλά πρότυπα ακεραιότητας και εμπιστευτικότητας δεδομένων.

Κεφάλαιο 2ο
Θεωρία Hyperledger Fabric

2.1 Επισκόπηση της Τεχνολογίας Blockchain

Η τεχνολογία blockchain, που πρωτοπαρουσιάστηκε μέσω του Bitcoin από τον ψευδώνυμο Satoshi Nakamoto το 2008, έχει φέρει επανάσταση στον τρόπο με τον οποίο διεξάγονται οι ψηφιακές συναλλαγές. Στην ουσία του, ένα blockchain είναι ένα αποκεντρωμένο και κατανεμημένο καθολικό (ledger) που καταγράφει συναλλαγές σε πολλούς υπολογιστές με τέτοιο τρόπο ώστε οι καταγεγραμμένες συναλλαγές να μην μπορούν να τροποποιηθούν αναδρομικά. Αυτό διασφαλίζει τη διαφάνεια, την ασφάλεια και την ακεραιότητα των δεδομένων χωρίς την ανάγκη για μια κεντρική αρχή.

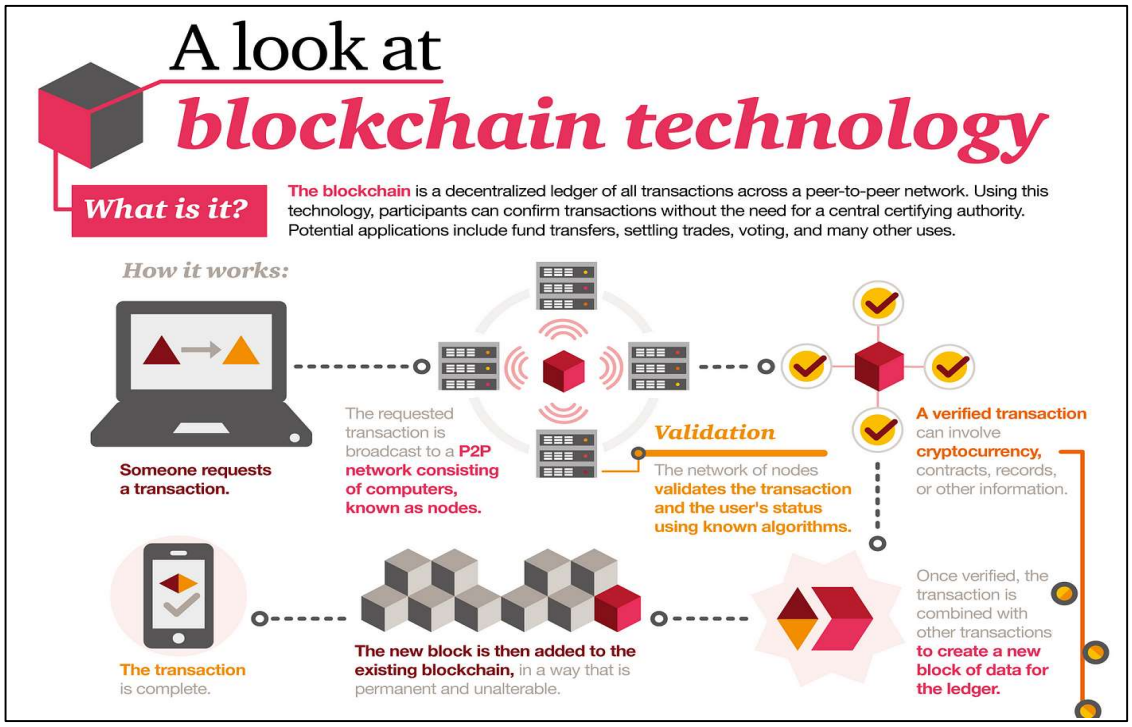
Τα βασικά στοιχεία του Blockchain περιλαμβάνουν:

Αποκέντρωση: Σε αντίθεση με τις παραδοσιακές κεντρικές βάσεις δεδομένων, τα blockchains λειτουργούν σε ένα δίκτυο peer-to-peer, εξαλείφοντας την ανάγκη για μεσάζοντες.

Αμεταβλητότητα: Μόλις τα δεδομένα καταγραφούν σε ένα blockchain, είναι εξαιρετικά δύσκολο να αλλάξουν, εξασφαλίζοντας μόνιμη καταγραφή των συναλλαγών.

Διαφάνεια: Όλοι οι συμμετέχοντες στο δίκτυο έχουν πρόσβαση στα ίδια δεδομένα, προάγοντας τη διαφάνεια και την εμπιστοσύνη.

Ασφάλεια: Προηγμένες κρυπτογραφικές τεχνικές ασφαλίζουν τα δεδομένα, διασφαλίζοντας ότι μόνο εξουσιοδοτημένα μέρη μπορούν να έχουν πρόσβαση ή να τροποποιούν τις πληροφορίες.



Εικόνα 1. Συνοπτική περιγραφή της τεχνολογίας blockchain

2.2 Εισαγωγή στα Hyperledger και Hyperledger Fabric

Το Hyperledger είναι ένα συνεργατικό έργο ανοιχτού κώδικα που φιλοξενείται από το Linux Foundation, με στόχο την προώθηση τεχνολογιών blockchain που αφορούν διάφορους κλάδους. Το Hyperledger, το οποίο ξεκίνησε τον Δεκέμβριο του 2015, συγκεντρώνει ένα ευρύ φάσμα βιομηχανιών, από τα χρηματοοικονομικά έως τη βιομηχανική παραγωγή, για την ανάπτυξη ισχυρών, διαλειτουργικών πλαισίων blockchain.

Το Hyperledger Fabric, ένα από τα πιο εξέχοντα έργα υπό την ομπρέλα του Hyperledger, είναι μια πλατφόρμα κατανεμημένου καθολικού επιχειρηματικής κλάσης, με άδεια χρήσης (permissioned). Σε αντίθεση με τα δημόσια blockchains όπως το Bitcoin και το Ethereum, το Hyperledger Fabric έχει σχεδιαστεί ειδικά για επιχειρηματική χρήση, προσφέροντας μια αρθρωτή αρχιτεκτονική που παρέχει υψηλά επίπεδα εμπιστευτικότητας, ανθεκτικότητας, κλιμακωσιμότητας και ευελιξίας. Η μοναδική προσέγγιση του Fabric στη συναίνεση, σε συνδυασμό με τη φύση του που απαιτεί άδειες, το καθιστά ιδιαίτερα κατάλληλο για επιχειρηματικά περιβάλλοντα όπου η ιδιωτικότητα και η απόδοση είναι κρίσιμες.

2.2.1 Σημασία του Hyperledger Fabric σε Enterprise Solutions

Στο επιχειρηματικό περιβάλλον, οι εταιρείες χρειάζονται μια πλατφόρμα blockchain που να υποστηρίζει την εμπιστευτικότητα, τη λογοδοσία και την κλιμακωσιμότητα. Το Hyperledger Fabric καλύπτει αυτές τις ανάγκες μέσω της καινοτόμου αρχιτεκτονικής και των χαρακτηριστικών του, επιτρέποντας στους οργανισμούς να βελτιστοποιούν τις λειτουργίες τους, να ενισχύουν την ασφάλεια και να οικοδομούν εμπιστοσύνη μεταξύ των εμπλεκόμενων μερών. Βιομηχανίες όπως τα χρηματοοικονομικά, η εφοδιαστική αλυσίδα, η υγειονομική περίθαλψη και τα ακίνητα έχουν υιοθετήσει το Hyperledger Fabric για να λύσουν πολύπλοκα προβλήματα που περιλαμβάνουν πολλαπλά μέρη και απαιτούν υψηλό επίπεδο ακεραιότητας και ιδιωτικότητας των δεδομένων.

2.3 Αρχιτεκτονική του Hyperledger Fabric

2.3.1 Αρθρωτή Σχεδίαση (Modular Design)

Η αρθρωτή σχεδίαση του Hyperledger Fabric είναι μία από τις πιο σημαντικές του δυνάμεις, επιτρέποντας την προσαρμογή και επέκταση των συστατικών του για να καλύπτουν συγκεκριμένες επιχειρηματικές ανάγκες. Η αρθρωτή αρχιτεκτονική αποτελείται από διάφορα εναλλάξιμα μέρη, συμπεριλαμβανομένων μηχανισμών συναίνεσης, υπηρεσιών μέλους και περιβάλλοντων εκτέλεσης chaincode. Αυτή η ευελιξία διασφαλίζει ότι οι οργανισμοί μπορούν να προσαρμόσουν το blockchain δίκτυο στις απαιτήσεις τους, χωρίς να περιορίζονται από λύσεις που ταιριάζουν σε όλες τις περιπτώσεις.

Βασικές ενότητες (key modules) περιλαμβάνουν:

- **Επίπεδο Συναίνεσης (Consensus Layer)**: Διαχειρίζεται την σειρά και την επιβεβαίωση των συναλλαγών.
- **Πάροχος Υπηρεσιών Μέλους (Membership Service Provider)**: Χειρίζεται τη διαχείριση ταυτοτήτων και τον έλεγχο πρόσβασης.
- **Έξυπνα συμβόλαια (Chaincode-Smart Contracts)**: Εκτελεί την επιχειρισιακή λογική (business logic).
- **Καθολικό (Ledger)**: Αποθηκεύει δεδομένα συναλλαγών με έναν επαληθεύσιμο και αμετάβλητο τρόπο.

2.3.2. Στοιχεία Δικτύου (Network Components)

Ομότιμοι (Peers): Peers είναι τα θεμελιώδη δομικά στοιχεία ενός δικτύου Hyperledger Fabric. Φιλοξενούν τα καθολικά και τον κώδικα αλυσίδας (chaincode) και έχουν διαφορετικούς ρόλους:

- **Ομότιμοι επικύρωσης (Endorsing Peers)**: Επικυρώνουν και εγκρίνουν συναλλαγές εκτελώντας το chaincode..
- **Ομότιμοι δέσμευσης (Committing Peers)**: Διατηρούν το καθολικό δεσμεύοντας τις επικυρωμένες συναλλαγές..
- **Ομότιμοι αγκύρωσης (Anchor Peers)**: Διευκολύνουν την επικοινωνία μεταξύ διαφορετικών οργανισμών σε ένα κανάλι.

Παραγγελιοδότες (Orderers): Orderers, επίσης γνωστοί ως ordering nodes, είναι υπεύθυνοι για τον καθορισμό της σειράς των συναλλαγών και τη δημιουργία των μπλοκ που θα διανεμηθούν στους κόμβους. Αυτή η διαδικασία διασφαλίζει ότι όλοι οι κόμβοι έχουν μια συνεπή άποψη του καθολικού. Η υπηρεσία διαταγής μπορεί να υλοποιηθεί χρησιμοποιώντας διαφορετικούς αλγόριθμους συναίνεσης, όπως ο Solo (για ανάπτυξη), ο Kafka ή ο Raft.

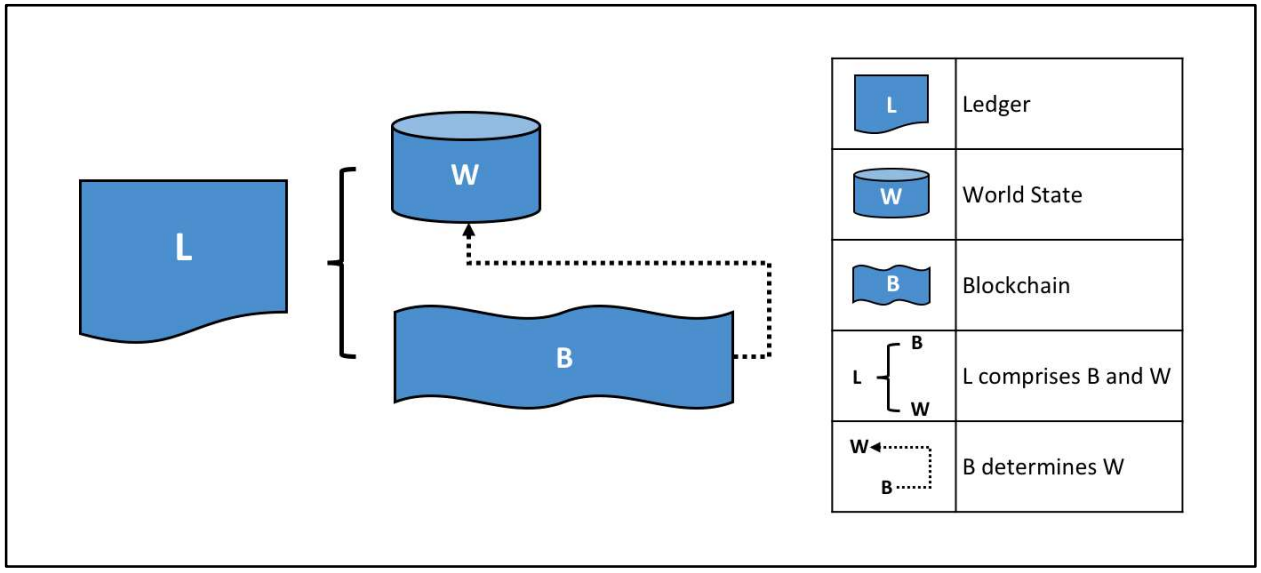
Πάροχος Υπηρεσιών Μέλους (Membership Service Provider ή MSP): Το MSP είναι ένα κρίσιμο συστατικό που διαχειρίζεται τις ταυτότητες των συμμετεχόντων στο δίκτυο. Εκδίδει και επαληθεύει ψηφιακά πιστοποιητικά χρησιμοποιώντας μια Υποδομή Δημοσίου Κλειδιού (PKI), διασφαλίζοντας ότι μόνο εξουσιοδοτημένες οντότητες μπορούν να συμμετέχουν στο δίκτυο. Το MSP επιβάλλει πολιτικές και παρέχει έναν μηχανισμό για την επαλήθευση ταυτότητας και την ανάθεση ρόλων.

2.3.3 Δομή του Καθολικού (Ledger)

Ο ledger του Hyperledger Fabric αποτελείται από δύο διακριτά μέρη: το world state και το transaction log.

World State: Το world state αντιπροσωπεύει την τρέχουσα κατάσταση του ledger και αποθηκεύεται ως βάση δεδομένων που μπορεί να αναζητηθεί αποδοτικά. Αντικατοπτρίζει την πιο πρόσφατη τιμή όλων των κλειδιών και ενημερώνεται με κάθε συναλλαγή. Το Hyperledger Fabric υποστηρίζει πολλαπλές επιλογές βάσεων δεδομένων για την αποθήκευση του world state, συμπεριλαμβανομένων των LevelDB και CouchDB.

Λίστα Συναλλαγών (Transaction Log): Η λίστα συναλλαγών είναι μια αμετάβλητη καταγραφή όλων των συναλλαγών που έχουν ποτέ πραγματοποιηθεί στο δίκτυο. Είναι μόνο προσθηκών, που σημαίνει ότι μπορούν να προστεθούν νέες συναλλαγές, αλλά οι υπάρχουσες δεν μπορούν να τροποποιηθούν ή να διαγραφούν. Αυτό το αρχείο διασφαλίζει μια πλήρη ιστορική καταγραφή των αλλαγών στην κατάσταση του καθολικού (ledger).



Εικόνα 2. Δομή του ledger

2.4 Chaincode και Smart Contracts

2.4.1 Ορισμός και Σκοπός

Το Chaincode, ο όρος του Hyperledger Fabric για τα έξυπνα συμβόλαια, περιέχει τη λογική της επιχείρησης που καθορίζει τη συμπεριφορά του δικτύου. Είναι ένα πρόγραμμα γραμμένο σε γλώσσα προγραμματισμού υψηλού επιπέδου που ορίζει τους κανόνες για τη διαχείριση των στοιχείων. Το Chaincode αναπτύσσεται στο δίκτυο και ενεργοποιείται από συναλλαγές που υποβάλλονται από πελάτες εφαρμογών.

2.4.2 Κύκλος Ζωής του Chaincode

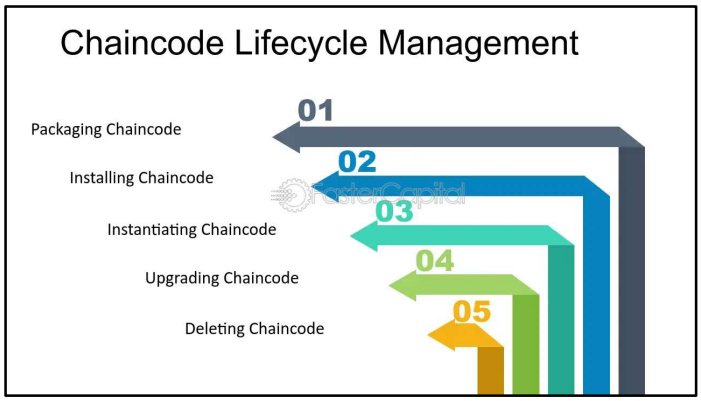
Ο κύκλος ζωής του chaincode περιλαμβάνει διάφορα στάδια: ανάπτυξη, ανάπτυξη στο δίκτυο και εκτέλεση.

Ανάπτυξη: Το Chaincode αναπτύσσεται χρησιμοποιώντας υποστηριζόμενες γλώσσες προγραμματισμού, όπως Go, Java ή Node.js. Οι προγραμματιστές γράφουν τη λογική της επιχείρησης για τη διαχείριση συναλλαγών, τη διαχείριση στοιχείων και την επιβολή κανόνων.

Παραμετροποίησης (Deployment): Η ανάπτυξη του chaincode περιλαμβάνει διάφορα βήματα:

- **Συσκευασία (Packaging)**: Το Chaincode συσκευάζεται σε μια μορφή έτοιμη για ανάπτυξη.
- **Εγκατάσταση (Installation)**: Το συσκευασμένο chaincode εγκαθίσταται στους κόμβους επικύρωσης.
- **Ενεργοποίηση (Instantiation)**: Το εγκατεστημένο chaincode ενσταλάσσεται σε ένα κανάλι, καθιστώντας το διαθέσιμο για εκτέλεση.

Εκτέλεση (Execution): Η εκτέλεση του chaincode ενεργοποιείται από προτάσεις συναλλαγών από πελάτες εφαρμογών. Οι κόμβοι επικύρωσης προσομοιώνουν τη συναλλαγή εκτελώντας το chaincode και δημιουργούν απαντήσεις προτάσεων, οι οποίες στη συνέχεια χρησιμοποιούνται για την επικύρωση και τη δέσμευση της συναλλαγής.



Εικόνα 3. Κύκλος ζωής του chaincode

2.5 Ροή Δεδομένων στο Hyperledger Fabric

The transaction flow in Hyperledger Fabric έχει σχεδιαστεί για να εξασφαλίζει την ασφάλεια, την ακεραιότητα και την αποδοτικότητα μέσω μιας διαδικασίας πολλαπλών φάσεων.

2.5.1 Προτασιακή φάση (Proposal Phase)

Το transaction flow ξεκινά με ένα client application που δημιουργεί μια πρόταση συναλλαγής. Αυτή η πρόταση περιλαμβάνει τη λειτουργία του chaincode που πρέπει να εκτελεστεί και τις απαραίτητες παραμέτρους. Ο πελάτης στέλνει αυτήν την πρόταση στους endorsing peers που ορίζονται από την endorsement policy.

2.5.2 Επικυρωτική φάση (Endorsement Phase)

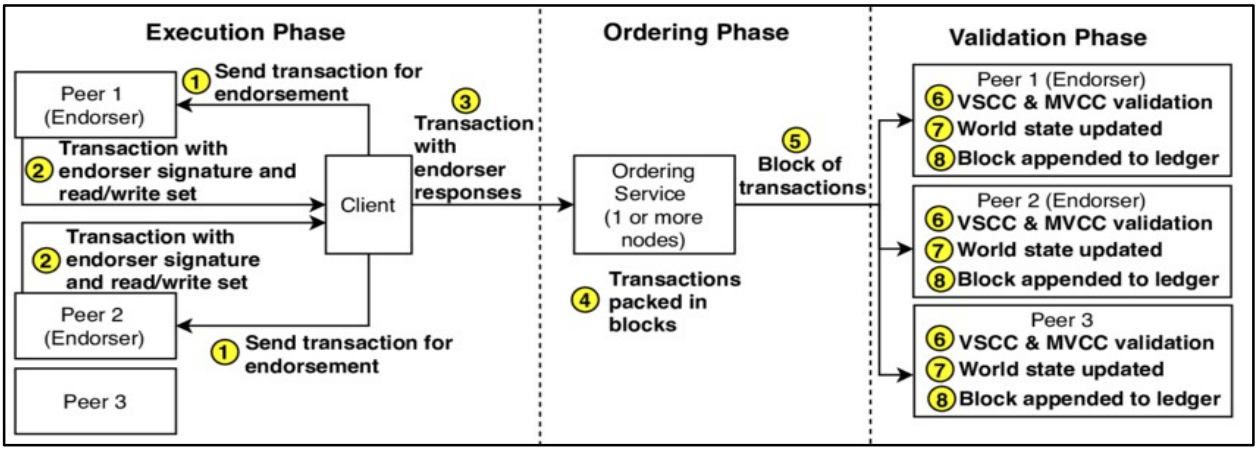
Οι endorsing peers λαμβάνουν την πρόταση συναλλαγής και προσομοιώνουν τη συναλλαγή εκτελώντας τον καθορισμένο chaincode. Δημιουργούν μια απόκριση πρότασης που περιλαμβάνει τα σετ ανάγνωσης/εγγραφής (τα δεδομένα που διαβάζονται από και γράφονται στο ledger) και μια ψηφιακή υπογραφή. Ο πελάτης συλλέγει αυτές τις επικυρώσεις και, εάν πληρούν την πολιτική επικύρωσης, η συναλλαγή θεωρείται έγκυρη.

2.5.3 Φάση παραγγελίας (Ordering Phase)

Ο πελάτης υποβάλλει την επικυρωμένη συναλλαγή στην υπηρεσία ταξινόμησης, η οποία διατάσσει τις συναλλαγές και τις συσκευάζει σε μπλοκ. Η υπηρεσία ταξινόμησης διασφαλίζει ότι όλες οι συναλλαγές παραδίδονται στους committing peers στη σωστή σειρά, διατηρώντας τη συνοχή σε όλο το δίκτυο.

2.5.4 Επιβεβαιωτική φάση (Validation Phase)

Οι committing peers λαμβάνουν τα μπλοκ συναλλαγών από την υπηρεσία ταξινόμησης. Κάθε κόμβος επικυρώνει τις συναλλαγές ελέγχοντας τις επικυρώσεις και διασφαλίζοντας ότι δεν υπάρχουν συγκρούσεις με άλλες συναλλαγές χρησιμοποιώντας τον έλεγχο ταυτόχρονης πρόσβασης πολλαπλών εκδόσεων (MVCC). Οι έγκυρες συναλλαγές δεσμεύονται στο ledger και η κατάσταση του κόσμου ενημερώνεται.



Εικόνα 4. Transaction flow στο Hyperledger Fabric

2.6 Μηχανισμός Συνάινεσης (Consensus Mechanism)

Το Hyperledger Fabric υποστηρίζει προσθέσιμους μηχανισμούς συναίνεσης, επιτρέποντας στους οργανισμούς να επιλέξουν το πρωτόκολλο συναίνεσης που ταιριάζει καλύτερα στις απαιτήσεις τους. Αυτή η ευελιξία είναι κρίσιμη για την εξισορρόπηση της απόδοσης, της ασφάλειας και της ανοχής σε σφάλματα.

Raft Consensus: Το Raft είναι ένας αλγόριθμος συναίνεσης ανθεκτικός σε σφάλματα διακοπής λειτουργίας, που εξασφαλίζει συνέπεια και αξιοπιστία στην υπηρεσία ταξινόμησης. Είναι βασισμένο σε ηγέτη, όπου ένας κόμβος ηγέτης συντονίζει την ταξινόμηση των συναλλαγών, ενώ οι ακόλουθοι αναπαράγουν το ταξινομημένο αρχείο καταγραφής.

Byzantine Fault Tolerant Consensus: Για περιβάλλοντα που απαιτούν υψηλότερη ασφάλεια και ανθεκτικότητα, μπορούν να χρησιμοποιηθούν Byzantine Fault Tolerant (BFT) αλγόριθμοι. Αυτοί οι αλγόριθμοι είναι σχεδιασμένοι να διαχειρίζονται κακόβουλους ή ελαττωματικούς κόμβους, εξασφαλίζοντας ότι το δίκτυο μπορεί να φτάσει σε συναίνεση ακόμα και αν κάποιοι κόμβοι δρουν ανέντιμα.

Performance and Scalability Considerations: Η επιλογή του μηχανισμού συναίνεσης επηρεάζει την απόδοση και την επεκτασιμότητα του δικτύου. Για παράδειγμα, το Raft είναι κατάλληλο για περιβάλλοντα που δίνουν προτεραιότητα στην ταχύτητα και την απλότητα, ενώ το BFT προσφέρει αυξημένη ασφάλεια με το κόστος μεγαλύτερης υπολογιστικής επιβάρυνσης.

2.7 Λειτουργίες για Ιδιωτικότητα και Εμπιστευτικότητα

Το Hyperledger Fabric προσφέρει ισχυρά χαρακτηριστικά ιδιωτικότητας και εμπιστευτικότητας για να διασφαλίσει ότι τα ευαίσθητα δεδομένα προστατεύονται και είναι προσβάσιμα μόνο από εξουσιοδοτημένους συμμετέχοντες.

Κανάλια (Channels): Τα channels είναι ιδιωτικά υποδίκτυα μέσα στο κύριο δίκτυο blockchain, επιτρέποντας σε συγκεκριμένες ομάδες συμμετεχόντων να διεξάγουν συναλλαγές ιδιωτικά. Κάθε channel έχει το δικό του ledger και μόνο τα μέλη του channel αυτού μπορούν να έχουν πρόσβαση στις συναλλαγές και στα δεδομένα που περιέχει.

Συλλογές Οδιωτικών δεδομένων (Private Data Collections): Οι συλλογές ιδιωτικών δεδομένων επιτρέπουν υποσύνολα δεδομένων να μοιράζονται μεταξύ συγκεκριμένων συμμετεχόντων χωρίς να εκτίθενται σε ολόκληρο το δίκτυο. Αυτό το χαρακτηριστικό είναι ιδιαίτερα χρήσιμο σε περιπτώσεις όπου ορισμένα δεδομένα πρέπει να παραμείνουν εμπιστευτικά, ενώ εξακολουθούν να είναι επαληθεύσιμα από άλλες πλευρές.

Παροδικά δεδομένα (Transient Data): Τα transient data είναι προσωρινές πληροφορίες που χρησιμοποιούνται κατά τη φάση της πρότασης συναλλαγής και δεν περιλαμβάνονται στο αρχείο καταγραφής συναλλαγών. Αυτό βοηθά στη διατήρηση της εμπιστευτικότητας για ευαίσθητα δεδομένα που δεν χρειάζεται να καταγράφονται μόνιμα στο ledger.

Μηχανισμοί Ελέγχου Πρόσβασης (Access Control Mechanisms): Το Hyperledger Fabric χρησιμοποιεί προηγμένους μηχανισμούς ελέγχου πρόσβασης για να διασφαλίσει ότι μόνο εξουσιοδοτημένοι χρήστες μπορούν να εκτελούν συγκεκριμένες ενέργειες. Μπορούν να οριστούν πολιτικές για τη διαχείριση του ποιος μπορεί να διαβάζει, να γράφει και να επικυρώνει συναλλαγές, παρέχοντας λεπτομερή έλεγχο των δραστηριοτήτων του δικτύου.

2.8 Διαχείριση Μέλους και Ταυτότητας (Membership και Identity Management)

Το Membership Service Provider (MSP) παίζει κρίσιμο ρόλο στη διαχείριση των ταυτοτήτων και στη διασφάλιση ότι μόνο εξουσιοδοτημένες οντότητες μπορούν να συμμετέχουν στο δίκτυο Hyperledger Fabric.

Role of Membership Service Provider (MSP): Ο MSP είναι υπεύθυνος για την έκδοση, την επικύρωση και τη διαχείριση ψηφιακών ταυτοτήτων. Χρησιμοποιεί ένα Public Key Infrastructure (PKI) για την παροχή κρυπτογραφικής ασφάλειας και την επιβολή πολιτικών ελέγχου πρόσβασης. Ο MSP διασφαλίζει ότι κάθε συμμετέχοντας έχει επαληθεύσιμη ταυτότητα και τις κατάλληλες άδειες.

Public Key Infrastructure (PKI): Η PKI χρησιμοποιείται για την έκδοση ψηφιακών πιστοποιητικών στους συμμετέχοντες, παρέχοντας έναν ασφαλή τρόπο για την επαλήθευση των ταυτοτήτων και τον έλεγχο ταυτότητας των συναλλαγών. Κάθε συμμετέχοντας διαθέτει ένα μοναδικό ψηφιακό πιστοποιητικό, το οποίο χρησιμοποιείται για την υπογραφή συναλλαγών και την αυθεντικοποίηση των δραστηριοτήτων στο δίκτυο.

Identity Issuance and Authentication: Οι ταυτότητες στο Hyperledger Fabric εκδίδονται από μια Αρχή Πιστοποίησης (CA), η οποία αποτελεί μέρος του MSP. Οι συμμετέχοντες χρησιμοποιούν τα ψηφιακά τους πιστοποιητικά για να πιστοποιούν την ταυτότητά τους όταν αλληλεπιδρούν με το δίκτυο, διασφαλίζοντας ότι μόνο εξουσιοδοτημένοι χρήστες μπορούν να έχουν πρόσβαση και να τροποποιούν το ledger.

2.9 Διαδικασία Ανάπτυξης και Εγκατάστασης

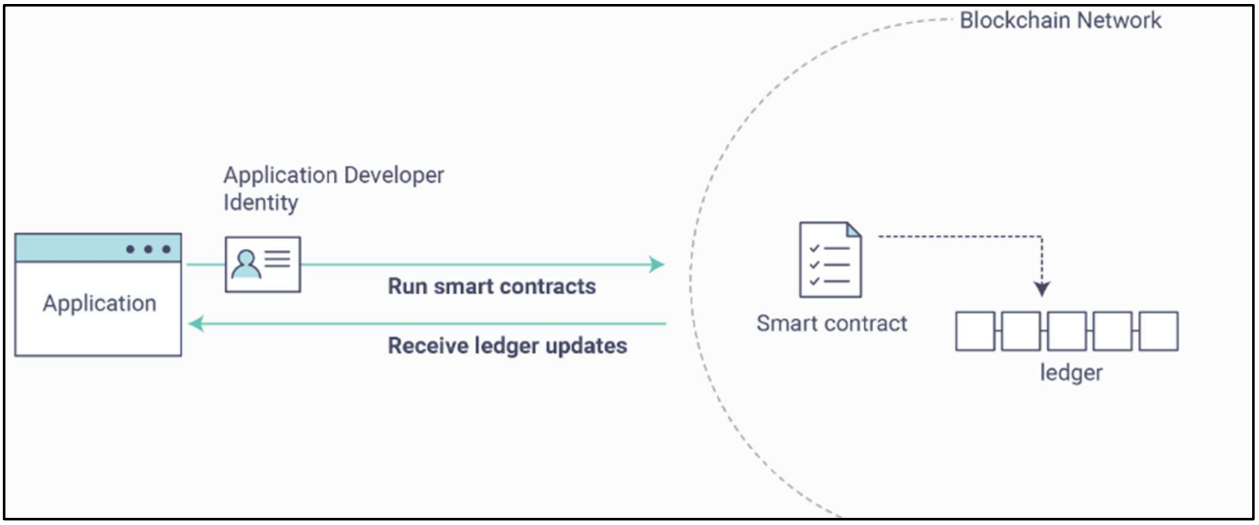
Η ανάπτυξη και η εγκατάσταση εφαρμογών στο Hyperledger Fabric περιλαμβάνει διάφορα βασικά βήματα, από τη ρύθμιση του δικτύου έως τη συγγραφή και την εγκατάσταση του chaincode.

Setting Up the Network: Το πρώτο βήμα στην εγκατάσταση ενός δικτύου Hyperledger Fabric είναι η διαμόρφωση των στοιχείων του δικτύου. Αυτό περιλαμβάνει τη ρύθμιση των peers, των orderers και των MSPs, καθώς και τον καθορισμό της τοπολογίας του δικτύου. Αρχεία διαμόρφωσης και scripts χρησιμοποιούνται για την αυτοματοποίηση της διαδικασίας ρύθμισης.

Writing Chaincode: Η ανάπτυξη chaincode περιλαμβάνει τη συγγραφή της επιχειρηματικής λογικής που θα διέπει τις συναλλαγές στο δίκτυο. Οι προγραμματιστές μπορούν να χρησιμοποιήσουν υποστηριζόμενες γλώσσες προγραμματισμού, όπως Go, Java ή Node.js, για να γράψουν chaincode, το οποίο καθορίζει πώς διαχειρίζονται τα assets και πώς επεξεργάζονται οι συναλλαγές.

Installing and Instantiating Chaincode: Μόλις γραφτεί το chaincode, πρέπει να πακεταριστεί, να εγκατασταθεί στους endorsing peers και να ενσταλαχθεί σε ένα κανάλι. Αυτή η διαδικασία περιλαμβάνει την ανάπτυξη του chaincode στο δίκτυο και τη διαθεσιμότητά του για εκτέλεση.

Developing Client Applications: Τα client applications αλληλεπιδρούν με το blockchain δίκτυο για να υποβάλλουν συναλλαγές και να κάνουν ερωτήματα στο ledger. Αυτές οι εφαρμογές χρησιμοποιούν το SDK του Hyperledger Fabric για να επικοινωνούν με το δίκτυο, να διαχειρίζονται ταυτότητες και να χειρίζονται τις ροές εργασιών των συναλλαγών.



Εικόνα 5. Επικοινωνία εφαρμογής και blockchain δικτύου μέσω των smart contracts

2.10 Κυβέρνηση και διαχείριση (Governance και Management)

Η διακυβέρνηση και η διαχείριση ενός δικτύου Hyperledger Fabric περιλαμβάνουν αρκετές βασικές πτυχές, όπως η διαμόρφωση του δικτύου, η επιβολή πολιτικών και η διαχείριση του chaincode.

Ρυθμίσεις Δικτύου (Network Configuration): Η αποτελεσματική διακυβέρνηση ξεκινά με μια καλά ορισμένη διαμόρφωση του δικτύου. Αυτό περιλαμβάνει την εγκατάσταση της τοπολογίας του δικτύου, τη διαμόρφωση των peers, των orderers και των καναλιών, και την εγκαθίδρυση πρωτοκόλλων επικοινωνίας. Τα αρχεία διαμόρφωσης και τα scripts βοηθούν στην αυτοματοποίηση και τη διατήρηση της συνέπειας σε όλο το δίκτυο.

Πολιτικές και Άδειες (Policies and Permissions): Οι πολιτικές στο Hyperledger Fabric ορίζουν τους κανόνες και τις άδειες για τις λειτουργίες του δικτύου. Αυτές οι πολιτικές καθορίζουν ποιοι μπορούν να εκτελούν συγκεκριμένες ενέργειες, όπως η επικύρωση συναλλαγών, η προσθήκη νέων μελών ή η ενημέρωση του chaincode. Οι πολιτικές επιβάλλονται μέσω του MSP και είναι κρίσιμες για τη διατήρηση της ασφάλειας και της ακεραιότητας του δικτύου.

Ενημέρωση και Διαχείριση Έξυπνων Συμβολαίων (Updating and Managing Chaincode): Η διαχείριση του chaincode είναι μια συνεχής διαδικασία που περιλαμβάνει την ενημέρωση και τη συντήρηση της επιχειρηματικής λογικής. Οι ενημερώσεις στο chaincode πρέπει να συντονίζονται προσεκτικά για να εξασφαλίζεται η συμβατότητα και να αποφεύγονται οι διακοπές. Το Hyperledger Fabric παρέχει μηχανισμούς για την αναβάθμιση των εκδόσεων του chaincode διατηρώντας ταυτόχρονα την ακεραιότητα του ledger.

2.11 Τρόποι Χρήσης και Εφαρμογές (Use Cases και Applications)

Η ευελιξία και το ισχυρό σύνολο χαρακτηριστικών του Hyperledger Fabric το καθιστούν κατάλληλο για ένα ευρύ φάσμα εφαρμογών σε διάφορους κλάδους.

Διαχείριση Τροφοδοτικής Αλυσίδας (Supply Chain Management): Το Hyperledger Fabric ενισχύει τη διαφάνεια και την αποδοτικότητα της εφοδιαστικής αλυσίδας παρέχοντας ένα αμετάβλητο βιβλίο για την παρακολούθηση των αγαθών. Κάθε συναλλαγή, από την παραγωγή μέχρι την παράδοση, καταγράφεται στην αλυσίδα των μπλοκ, διασφαλίζοντας την ανιχνευσιμότητα και μειώνοντας την απάτη.

Χρηματοοικονομικές Υπηρεσίες (Financial Services): Στον χρηματοοικονομικό τομέα, το Hyperledger Fabric διευκολύνει τις ασφαλείς και διαφανείς συναλλαγές. Επιτρέπει την αποδοτική διαχείριση διασυνοριακών πληρωμών, τη διαχείριση περιουσιακών στοιχείων και τη χρηματοδότηση εμπορίου, παρέχοντας μια αξιόπιστη πλατφόρμα για την καταγραφή και την επαλήθευση των συναλλαγών.

Τομέας Υγείας (Healthcare): Οι εφαρμογές του Hyperledger Fabric στην υγειονομική περίθαλψη περιλαμβάνουν την ασφαλή διαχείριση των ιατρικών φακέλων ασθενών, την ανιχνευσιμότητα φαρμάκων και τις κλινικές δοκιμές. Το blockchain διασφαλίζει την ακεραιότητα και την ιδιωτικότητα των δεδομένων, επιτρέποντας παράλληλα την απρόσκοπτη ανταλλαγή πληροφοριών μεταξύ εξουσιοδοτημένων μερών.

Περιουσιακά Ακίνητα (Real Estate): Στον τομέα των ακινήτων, το Hyperledger Fabric μπορεί να απλοποιήσει τις συναλλαγές ιδιοκτησίας, τις μεταβιβάσεις τίτλων και τις μισθώσεις. Το blockchain παρέχει μια διαφανή και αδιάβλητη καταγραφή της ιδιοκτησίας και του ιστορικού συναλλαγών, μειώνοντας τον κίνδυνο διαφορών και απάτης.

Κυβερνητικός και Δημόσιος Τομέας (Government and Public Sector): Οι κυβερνήσεις μπορούν να χρησιμοποιήσουν το Hyperledger Fabric για να βελτιώσουν τη διαφάνεια και την αποδοτικότητα στις δημόσιες υπηρεσίες. Οι εφαρμογές περιλαμβάνουν ασφαλή συστήματα ψηφοφορίας, κτηματολόγια και διανομή κοινωνικής πρόνοιας. Το blockchain διασφαλίζει τη λογοδοσία και μειώνει το διοικητικό φόρτο.

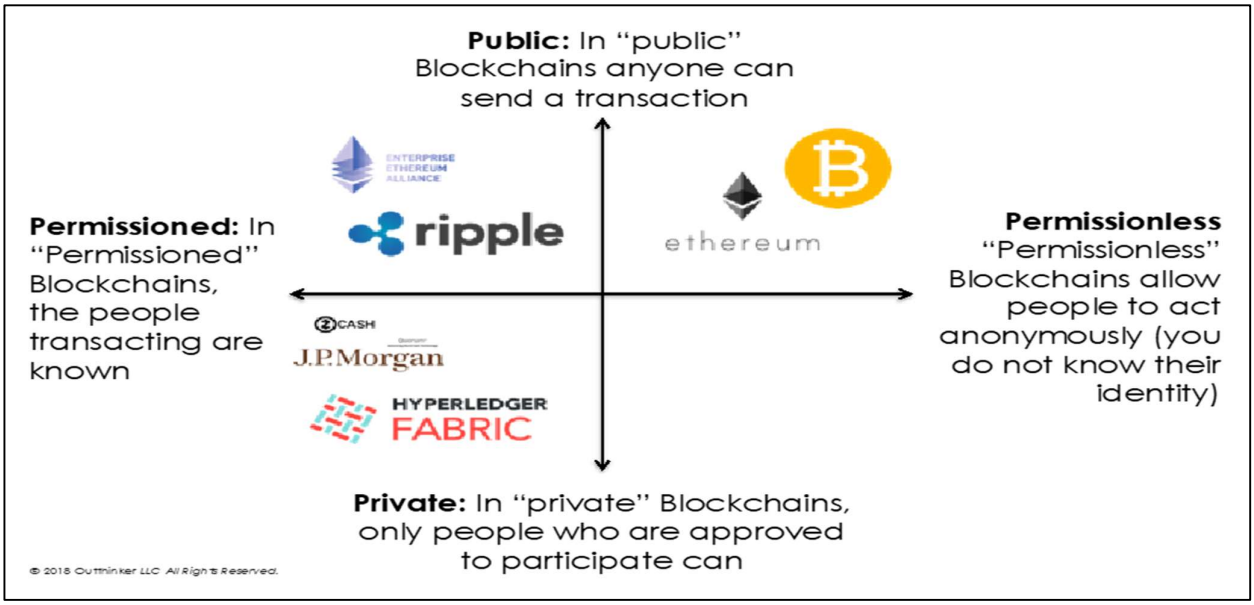
2.12 Σύγκριση με άλλα Blockchain Frameworks

Το Hyperledger Fabric μπορεί να συγκριθεί με άλλα blockchain frameworks για να αναδειχθούν τα μοναδικά του χαρακτηριστικά και πλεονεκτήματα.

Ethereum: Το Ethereum είναι ένα δημόσιο blockchain που υποστηρίζει smart contracts μέσω του Ethereum Virtual Machine (EVM). Σε αντίθεση με το Hyperledger Fabric, το Ethereum έχει σχεδιαστεί για ανοιχτές και αποκεντρωμένες εφαρμογές. Η αδειοδοτημένη φύση και ο αρθρωτός σχεδιασμός του Fabric το καθιστούν πιο κατάλληλο για επιχειρηματική χρήση, όπου ο έλεγχος και η ιδιωτικότητα είναι υψίστης σημασίας.

Corda: Το Corda, που αναπτύχθηκε από την R3, είναι μια άλλη blockchain πλατφόρμα που απευθύνεται στις χρηματοοικονομικές υπηρεσίες. Το Corda επικεντρώνεται στην ιδιωτικότητα και τις άμεσες συναλλαγές μεταξύ των μερών. Ενώ τόσο το Corda όσο και το Hyperledger Fabric προσφέρουν χαρακτηριστικά ιδιωτικότητας, η αρθρωτότητα του Fabric και οι ενσωματώσιμες μηχανισμοί συναίνεσης του παρέχουν μεγαλύτερη ευελιξία για ένα ευρύτερο φάσμα εφαρμογών.

Quorum: Το Quorum είναι μια έκδοση του Ethereum προσανατολισμένη στις επιχειρήσεις, που αναπτύχθηκε από την J.P. Morgan. Διατηρεί τις δυνατότητες των smart contracts του Ethereum, αλλά προσθέτει χαρακτηριστικά ιδιωτικότητας και αδειοδοτημένη πρόσβαση. Το Quorum και το Hyperledger Fabric μοιράζονται ομοιότητες ως προς τον επιχειρηματικό προσανατολισμό τους, αλλά η αρθρωτότητα και η αρχιτεκτονική καναλιών του Fabric προσφέρουν διακριτά πλεονεκτήματα όσον αφορά την επεκτασιμότητα και την εμπιστευτικότητα.



Εικόνα 6. Διαχωρισμός public,private, permissioned και permissionless blockchain

2.13 Δοκιμασίες και Μελλοντικές Κατευθύνσεις

Παρά τα πλεονεκτήματά του, το Hyperledger Fabric αντιμετωπίζει αρκετές προκλήσεις που η συνεχιζόμενη ανάπτυξη στοχεύει να αντιμετωπίσει.

Τρέχοντες Περιορισμοί:

- **Πολυπλοκότητα (Complexity)**: Η εγκατάσταση και η διαχείριση ενός δικτύου Hyperledger Fabric μπορεί να είναι περίπλοκη, απαιτώντας σημαντική εξειδίκευση.
- **Επεκτασιμότητα (Scalability)**: Αν και το Fabric έχει σχεδιαστεί για

επεκτασιμότητα, η διαχείριση μεγάλων δικτύων με υψηλούς όγκους συναλλαγών μπορεί να παραμένει προκλητική.
- **Διαλειτουργικότητα (Interoperability)**: Η ενσωμάτωση του Hyperledger Fabric με άλλα blockchain δίκτυα και παλαιά συστήματα απαιτεί ισχυρές λύσεις διαλειτουργικότητας.

Συνεχής Ανάπτυξη (Ongoing Development): Η κοινότητα του Hyperledger συνεχίζει να βελτιώνει το Fabric, εστιάζοντας στη βελτίωση της χρηστικότητας, της απόδοσης και της διαλειτουργικότητας. Καινοτομίες στους αλγόριθμους συναίνεσης, τη διαχείριση ταυτότητας και τα εργαλεία ανάπτυξης στοχεύουν στην αντιμετώπιση των τρεχόντων περιορισμών και στην επέκταση των δυνατοτήτων του Fabric.

Μελλοντικές Ενισχύσεις (Future Enhancements): Οι μελλοντικές κατευθύνσεις για το Hyperledger Fabric περιλαμβάνουν:

- **Improved Developer Experience**: Απλοποίηση της διαδικασίας ανάπτυξης και εγκατάστασης μέσω καλύτερων εργαλείων και τεκμηρίωσης.
- **Λειτουργίες Ενισχυμένης Ιδιωτικότητας (Enhanced Privacy Features)**: Επέκταση των επιλογών ιδιωτικότητας για την υποστήριξη πιο σύνθετων περιπτώσεων χρήσης.
- **Διαλειτουργικές Λύσεις (Interoperability Solutions)**: Ανάπτυξη προτύπων και πρωτοκόλλων για την απρόσκοπτη ενσωμάτωση με άλλα blockchain δίκτυα και υπάρχοντα συστήματα.
- **Βελτιστοποίηση Απόδοσης (Performance Optimizations)**: Συνεχής βελτιστοποίηση της απόδοσης του Fabric για την αντιμετώπιση μεγαλύτερων όγκων συναλλαγών και δικτύων.

2.14 Σύνοψη

Η Hyperledger Fabric είναι μια ισχυρή και ευέλικτη πλατφόρμα blockchain σχεδιασμένη για εφαρμογές επιχείρησης. Η αρθρωτή αρχιτεκτονική της, η φύση της ως επιτρεπόμενη και τα ισχυρά χαρακτηριστικά απορρήτου την καθιστούν ιδανική για ένα ευρύ φάσμα επιχειρηματικών περιπτώσεων χρήσης. Παρέχοντας έναν ασφαλή, διαφανή και αποδοτικό τρόπο διαχείρισης συναλλαγών, η Hyperledger Fabric έχει τη δυνατότητα να μετασχηματίσει τις βιομηχανίες και να επιτρέψει νέα επιχειρηματικά μοντέλα.

Συνοπτικά, η αρχιτεκτονική και τα χαρακτηριστικά της Hyperledger Fabric καλύπτουν τις ειδικές ανάγκες των περιβαλλόντων επιχείρησης, προσφέροντας απαράμιλλη ευελιξία και έλεγχο. Καθώς η τεχνολογία συνεχίζει να εξελίσσεται, θα διαδραματίσει έναν όλο και πιο σημαντικό ρόλο στην υιοθέτηση λύσεων blockchain σε διάφορους τομείς. Η επίδραση της Hyperledger Fabric στην υιοθέτηση του blockchain είναι βαθιά, παρέχοντας μια αξιόπιστη και αξιόπιστη πλατφόρμα για το μέλλον των ψηφιακών συναλλαγών και της διαχείρισης δεδομένων.

Κεφάλαιο 3ο
Η Λογική του Hyperledger Fabric Πληροφοριακού Συστήματος

Το πληροφοριακό σύστημα που αναπτύχθηκε σε αυτή τη διπλωματική εργασία αξιοποιεί την τεχνολογία blockchain, συγκεκριμένα το Hyperledger Fabric, για να διασφαλίσει την ασφάλεια και την εμπιστευτικότητα των δεδομένων ακινήτων. Η βασική λογική του συστήματος βασίζεται στη διαχείριση αποκεντρωμένων καθολικών, στα έξυπνα συμβόλαια και στην ασφαλή επικοινωνία δεδομένων μεταξύ μιας διαδικτυακής εφαρμογής και του δικτύου blockchain. Σε αυτό το κεφάλαιο θα αναλυθεί η υποκείμενη λογική του συστήματος, συμπεριλαμβανομένου του τρόπου επεξεργασίας των δεδομένων, της αλληλεπίδρασης του συστήματος με το δίκτυο blockchain και του ρόλου διαφόρων στοιχείων, όπως το chaincode, το API και η διαδικτυακή διεπαφή.

3.1 Αρχιτεκτονική του Συστήματος

Το σύστημα είναι χτισμένο σε μια πολυεπίπεδη αρχιτεκτονική, όπου κάθε επίπεδο εξυπηρετεί μια ξεχωριστή λειτουργία:

Δίκτυο Blockchain: Στον πυρήνα του συστήματος βρίσκεται το δίκτυο blockchain Hyperledger Fabric. Αυτό το δίκτυο λειτουργεί ως ένα αποκεντρωμένο, αδειοδοτημένο καθολικό που αποθηκεύει με ασφάλεια δεδομένα ακινήτων. Το καθολικό περιέχει βασικές πληροφορίες, όπως ο αριθμός των υπνοδωματίων, των μπάνιων, τα επίπεδα ορόφων και τα στοιχεία ιδιοκτησίας για κάθε ακίνητο.

Επίπεδο API: Η διεπαφή προγραμματισμού εφαρμογών (API), που αναπτύχθηκε χρησιμοποιώντας το πλαίσιο Spring Boot, διευκολύνει την επικοινωνία μεταξύ του δικτύου blockchain και της διαδικτυακής εφαρμογής. Το API στέλνει αιτήματα στο δίκτυο blockchain για να διαβάσει ή να γράψει δεδομένα στο καθολικό.

Διαδικτυακή Εφαρμογή: Μια φιλική προς τον χρήστη διαδικτυακή εφαρμογή επιτρέπει στους τελικούς χρήστες να αλληλεπιδρούν με το σύστημα. Παρουσιάζει δεδομένα και εικόνες ακινήτων που ανακτώνται από το δίκτυο blockchain και επιτρέπει στους χρήστες να ενημερώνουν ή να προσθέτουν λεπτομέρειες για τα ακίνητα με έναν ευνόητο και προσβάσιμο τρόπο.

3.2 Ροή Δεδομένων και Αλληλεπίδραση με το Blockchain δίκτυο

Στην καρδιά της λειτουργικότητας του πληροφοριακού συστήματος βρίσκεται το μοντέλο συναλλαγών, το οποίο περιστρέφεται γύρω από το δίκτυο Hyperledger Fabric. Κάθε συναλλαγή που εκτελείται εντός του συστήματος ακολουθεί μια συγκεκριμένη ακολουθία:

Εκκίνηση από Χρήστη: Ο τελικός χρήστης ξεκινά μια συναλλαγή μέσω της διαδικτυακής εφαρμογής, όπως η προσθήκη ενός νέου ακινήτου ή η ενημέρωση των λεπτομερειών ενός υπάρχοντος.

Επικοινωνία μέσω API: Το API λαμβάνει το αίτημα του χρήστη και το μετατρέπει σε πρόταση συναλλαγής. Αυτή η πρόταση αποστέλλεται στους υποστηρικτές εντός του δικτύου blockchain.

Επικύρωση: Οι υποστηρικτές εκτελούν το chaincode (έξυπνα συμβόλαια) που σχετίζεται με τη συναλλαγή. Σε αυτή την περίπτωση, το chaincode επεξεργάζεται λειτουργίες σχετικές με τη διαχείριση ακινήτων, όπως η προσθήκη νέων εγγραφών ή η τροποποίηση υφιστάμενων.

Επικύρωση Συναλλαγής: Μετά την επικύρωση, η συναλλαγή κατατάσσεται και επικυρώνεται από τους διαχειριστές του δικτύου, διασφαλίζοντας ότι συμμορφώνεται με τις πολιτικές επικύρωσης.

Καταγραφή στο Καθολικό (ledger): Μόλις επικυρωθεί, η συναλλαγή καταγράφεται στο καθολικό του blockchain. Αυτή η ενημέρωση αντικατοπτρίζεται στη διαδικτυακή εφαρμογή, όπου ο χρήστης μπορεί να δει τα νέα ή τροποποιημένα δεδομένα του ακινήτου.

3.3 Έξυπνα Συμβόλαια (Smart Contracts)

Τα έξυπνα συμβόλαια, ή chaincode σύμφωνα με την ορολογία του Hyperledger Fabric, είναι υπεύθυνα για τη λογική του συστήματος. Το chaincode που αναπτύχθηκε για αυτό το σύστημα περιέχει τους κανόνες για τη διαχείριση ακινήτων. Συγκεκριμένα, αναλαμβάνει τις εξής λειτουργίες:

Προσθήκη Ακινήτου: Προσθέτει ένα νέο ακίνητο στο καθολικό με πληροφορίες όπως ο αριθμός των υπνοδωματίων, των μπάνιων, το επίπεδο του ορόφου και το όνομα του ιδιοκτήτη. Κάθε ακίνητο συνδέεται με ένα μοναδικό κλειδί για να διασφαλιστεί η ακεραιότητα των δεδομένων.

Ενημέρωση Ακινήτου: Επιδρέπει τροποποιήσεις στις υπάρχουσες λεπτομέρειες του ακινήτου, όπως η αλλαγή του αριθμού των υπνοδωματίων ή η ενημέρωση των στοιχείων ιδιοκτησίας.

Αναζήτηση Ακινήτου: Ανακτά δεδομένα ακινήτων από το καθολικό blockchain βάσει αιτημάτων χρηστών, όπως η καταχώρηση όλων των διαθέσιμων ακινήτων ή η λήψη λεπτομερειών για ένα συγκεκριμένο ακίνητο.

3.4 Δεδομένα Ασφάλειας και Ακεραιότητα

Ένα από τα βασικά πλεονεκτήματα του συστήματος είναι η ικανότητά του να διατηρεί την ασφάλεια και την ακεραιότητα των δεδομένων:

Αποκέντρωση: Τα δεδομένα αποθηκεύονται σε διάφορους κόμβους του δικτύου, διασφαλίζοντας ότι κανένας μεμονωμένος φορέας δεν έχει πλήρη έλεγχο πάνω στις πληροφορίες. Αυτή η αποκέντρωση αποτρέπει την μη εξουσιοδοτημένη παρέμβαση ή διαγραφή των εγγραφών ακινήτων.

Αμεταβλητότητα: Μόλις μια συναλλαγή καταγραφεί στο καθολικό, δεν μπορεί να τροποποιηθεί. Αυτή η αμεταβλητότητα εγγυάται ότι όλες οι εγγραφές ακινήτων διατηρούνται στην αρχική τους μορφή, δημιουργώντας μια αξιόπιστη και προστατευμένη αλυσίδα δεδομένων.

Έλεγχος Πρόσβασης: Μόνο οι εξουσιοδοτημένοι χρήστες μπορούν να εκτελούν συναλλαγές εντός του συστήματος, διασφαλίζοντας ότι τα ευαίσθητα δεδομένα ακινήτων δεν είναι προσβάσιμα σε μη εξουσιοδοτημένα άτομα. Αυτό διαχειρίζεται μέσω του Membership Service Provider (MSP), ο οποίος εκδίδει ψηφιακά πιστοποιητικά στους εξουσιοδοτημένους συμμετέχοντες του δικτύου.

Το δίκτυο blockchain δημιουργήθηκε χρησιμοποιώντας το περιβάλλον δοκιμών του Hyperledger Fabric. Αυτό περιλάμβανε την εγκατάσταση δύο κόμβων (Org1 και Org2) και μιας αρχής πιστοποίησης για τη διαχείριση της ταυτότητας. Το chaincode αναπτύχθηκε στο δίκτυο για τη διαχείριση των συναλλαγών ακινήτων.

3.5 Ανάπτυξη του API

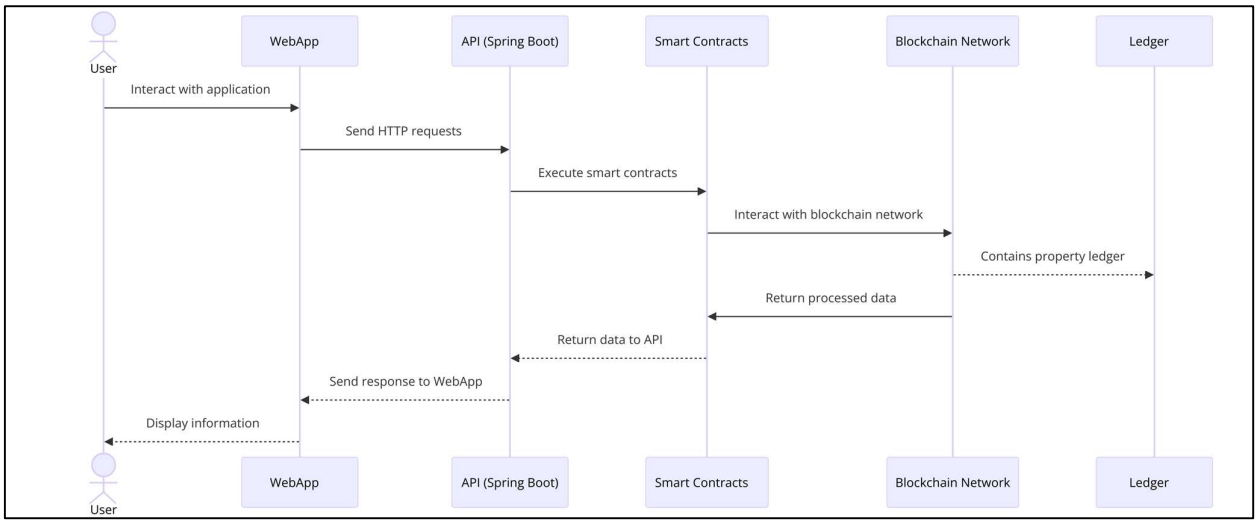
Το API, που αναπτύχθηκε χρησιμοποιώντας τη γλώσσα Java και το πλαίσιο Spring Boot, δημιουργήθηκε για να συνδέεται με το δίκτυο blockchain. Το API λειτουργεί ως το ενδιάμεσο επίπεδο μεταξύ της διαδικτυακής εφαρμογής και του blockchain, μετατρέποντας τις ενέργειες των χρηστών σε συναλλαγές blockchain. Το API επικοινωνεί με το δίκτυο μέσω του Hyperledger Fabric Java SDK, διευκολύνοντας τις εξής βασικές λειτουργίες:

- **Εγγραφή Διαχειριστή**: Επικυρώνει τον διαχειριστή με το δίκτυο.
- **Εγγραφή Χρήστη**: Εγγράφει νέους χρήστες που μπορούν να αλληλεπιδρούν με το σύστημα.
- **Λειτουργίες CRUD**: Διαχειρίζεται τις λειτουργίες Δημιουργίας, Ανάγνωσης, Ενημέρωσης και Διαγραφής (CRUD) για δεδομένα ακινήτων, οι οποίες μετατρέπονται σε συναλλαγές blockchain.

3.6 Ανάπτυξη Διαδικτυακής Εφαρμογής

Η διαδικτυακή εφαρμογή σχεδιάστηκε ως μια διαδικτυακή εφαρμογή Spring Boot. Παρουσιάζει δεδομένα ακινήτων με αισθητικά ευχάριστο και διαισθητικό τρόπο για τους τελικούς χρήστες. Οι χρήστες μπορούν να δουν λεπτομέρειες ακινήτων, συμπεριλαμβανομένων εικόνων, και να κάνουν ενημερώσεις στο καθολικό blockchain μέσω της διαδικτυακής διεπαφής. Η διαδικτυακή εφαρμογή συνδέεται με το API, το οποίο επεξεργάζεται αυτά τα αιτήματα και ενημερώνει το καθολικό blockchain αναλόγως.

Η λογική πίσω από το πληροφοριακό σύστημα βασίζεται στην αλληλεπίδραση μεταξύ των συστατικών του: του δικτύου blockchain, του chaincode, του API και της διαδικτυακής εφαρμογής. Η αποκεντρωμένη φύση του blockchain διασφαλίζει ασφαλείς, αμετάβλητες συναλλαγές, ενώ το chaincode καθορίζει τη λογική του συστήματος. Το API διευκολύνει την επικοινωνία μεταξύ του blockchain και της διαδικτυακής εφαρμογής, εξασφαλίζοντας ότι το σύστημα λειτουργεί ομαλά και με ασφάλεια. Τελικά, αυτό το σύστημα αντιπροσωπεύει μια ισχυρή και αποδοτική λύση για τη διαχείριση ευαίσθητων δεδομένων ακινήτων, προσφέροντας ενισχυμένη ασφάλεια και διαφάνεια μέσω της χρήσης προηγμένης τεχνολογίας blockchain.



Εικόνα 7. Διάγραμμα ροής λογικής του πληροφοριακού συστήματος

Κεφάλαιο 4ο
Ανάπτυξη Πληροφοριακού Συστήματος με τη Τεχνολογία Hyperledger Fabric

4.1 Διατύπωση του Προβλήματος

Σκοπός της παρούσας πτυχιακής εργασίας είναι η ανάπτυξη ενός κατάλληλου πληροφοριακού συστήματος το οποίο θα επικοινωνεί με blockchain δίκτυο και θα δέχεται στοιχεία περιουσιακών ακινήτων (real estate properties), συγκεκριμένα τον αριθμό υπνοδωματίων, μπάνιων, τον όροφο (σε περίπτωση που το ακίνητο είναι διαμέρισμα) και το όνομα του ιδιοκτήτη. Το σύστημα θα 'τροφοδοτεί' με δεδομένα ένα web application προκειμένου οι πληροφορίες να παρουσιάζονται στο τελικό χρήστη με ωραιοποιημένο τρόπο.

4.2 Δεδομένα Περιουσιακών Ακινήτων

Τα δεδομένα περιουσιακών ακινήτων που χρησιμοποιήθηκαν για τον έλεγχο λειτουργικότητας και τη παρουσίαση του πληροφοριακού συστήματος είναι ένα σύνολο δεδομένων για το κάθε ακίνητο (10 ακίνητα συνολικά). Το σύνολο δεδομένων περιέχει για κάθε ακίνητο τον υπνοδωματίων, μπάνιων, τον όροφο (σε περίπτωση που πρόκειται για διαμέρισμα, σύνδεσμο για φωτογραφία του εσωτερικού του ακινήτου, και το όνομα του ιδιοκτήτη. Επιπλέον, περιέχει μια μεταβλητή 'κλειδί΄, η οποία έχει μοναδική τιμή σε κάθε ακίνητο.

4.3 Διαδικασία Ανάπτυξης Πληροφοριακού Συστήματος με το Hyperledger Fabric Framework

Αρχικά στήνουμε το blockchain δίκτυο χρησιμοποιώντας το fabric testing network (περιέχεται στο fabric samples repository).

Μεταβαίνουμε στον υποκατάλογο test-network μέσα στον local clone του repository fabric-samples.
cd fabric-samples/test-network

Αν ήδη εκτελείται ένα δοκιμαστικό δίκτυο, το τερματίζουμε για να διασφαλίσουμε ότι το περιβάλλον είναι καθαρό.

./network.sh down

Εκκινούμε το δοκιμαστικό δίκτυο Fabric χρησιμοποιώντας το shell script network.sh.

./network.sh up createChannel -c mychannel -ca

Αυτή η εντολή θα αναπτύξει το δοκιμαστικό δίκτυο Fabric με δύο peers, μια υπηρεσία παραγγελίας (ordering service) και τρεις αρχές πιστοποίησης (Orderer, Org1, Org2). Αντί να χρησιμοποιήσουμε το εργαλείο cryptogen, εκκινούμε το δοκιμαστικό δίκτυο χρησιμοποιώντας αρχές πιστοποίησης, επομένως προσθέτουμε τη σημαία -ca. Επιπλέον, η εγγραφή του διαχειριστή της οργανισμού πραγματοποιείται αυτόματα όταν η αρχή πιστοποίησης ξεκινά.

Έπειτα κάνουμε deploy ένα smart contract στο κανάλι που δημιουργήσαμε. Μέσο του smart contract θα επιτευχθεί η δυνατότητα αλληλεπίδρασης μεταξύ της εφαρμογής μας και του blockchain δικτύου.

Στη συνέχεια, αναπτύσσουμε το πακέτο chaincode που περιέχει το smart contract καλώντας το script ./network.sh με τις επιλογές ονόματος και γλώσσας του chaincode.

./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-java/ -ccl java

Αυτό το script χρησιμοποιεί τον κύκλο ζωής του chaincode για να πακετάρει, εγκαταστήσει, κάνει query για το εγκατεστημένο chaincode, εγκρίνει το chaincode τόσο για το Org1 όσο και για το Org2, και τέλος να το κάνει commit.

Η εφαρμογή που αναπτύχθηκε για να επικοινωνεί με το blockchain δίκτυο είναι ένα API σε Spring Boot Framework και χρησιμοποιεί το Hyperledger fabric java sdk. Η επικοινωνία αυτή γίνεται μέσω του chaincode fabhouse. Το API υποστηρίζει request για να κάνουμε enroll έναν admin, να κάνουμε register ένα user, να εμφανίζουμε όλα τα περιουσιακά ακίνητα που βρίσκονται στο ledger, να δημιουργούμε καινούργιο ακίνητο το οποίο θα προστίθεται στο ledger, καθώς και για να αλλάζουμε τον ιδιοκτήτη ενός ακινήτου.

Για την παρουσίαση των δεδομένων που λαμβάνει το API με πιο εμφανίσιμο τρόπο,

αναπτύξαμε ένα Spring Boot Web Application. Το ίδιο έχει ως σκοπό την επικοινωνία με το API, ώστε ο τελικός χρήστης να έχει την δυνατότητα να προσθέσει ένα καινούργιο ακίνητο στο ledger, καθώς και να αλλάξει τον ιδιοκτήτη και την εικόνα του κάθε ακινήτου.

4.4 Ο Ρόλος του Chaincode στο Πληροφοριακό Σύστημα

Το chaincode είναι ζωτικής σημασίας για το σύστημα blockchain, καθώς κωδικοποιεί τη λογική της επιχείρησης για τη διαχείριση συναλλαγών ακινήτων. Καθορίζει ποιες ενέργειες μπορούν να εκτελεστούν στο blockchain, διασφαλίζοντας ότι όλες οι λειτουργίες που σχετίζονται με ακίνητα (όπως η δημιουργία ή η ενημέρωση ακινήτων) ακολουθούν τους προκαθορισμένους κανόνες.

Σε αυτό το σύστημα, το chaincode είναι γραμμένο σε Java και αναπτύσσεται στο blockchain για να χειριστεί βασικές λειτουργίες, όπως:

Δημιουργία Ακινήτου: Προσθέτει ένα νέο ακίνητο στο καθολικό. Το chaincode λαμβάνει παραμέτρους, όπως η μεταβλητή κλειδί, τον αριθμό των υπνοδωματίων, των μπάνιων, των ορόφων και το όνομα του ιδιοκτήτη. Μόλις υποβληθεί, η συναλλαγή επικυρώνεται και καταγράφεται στο blockchain.

Ενημέρωση Ακινήτου: Επιτρέπει την τροποποίηση των στοιχείων ενός υφιστάμενου ακινήτου. Το chaincode διασφαλίζει ότι μόνο έγκυρες ενημερώσεις (π.χ. αλλαγή ιδιοκτήτη ή λεπτομερειών του ακινήτου) εφαρμόζονται στο καθολικό, διατηρώντας την ακεραιότητα των προηγούμενων εγγραφών.

Αναζήτηση Ακινήτου: Ανακτά τις λεπτομέρειες ενός ακινήτου με την αναζήτηση στο καθολικό βάσει μίας μοναδικής μεταβλητής κλειδί. Αυτό είναι κρίσιμο για την παροχή πληροφοριών στους χρήστες χωρίς να αλλοιώνεται το blockchain.

CreateHouse():
Παράμετροι Εισόδου: μεταβλητή κλειδί, αριθμός υπνοδωματίων, αριθμός μπάνιων, όροφος και όνομα ιδιοκτήτη.
Λειτουργικότητα: Προσθέτει μια νέα εγγραφή στο καθολικό με τα παρεχόμενα στοιχεία του ακινήτου. Το chaincode πρώτα ελέγχει αν η τιμή της μεταβλητής κλειδί υπάρχει ήδη για να αποφευχθούν διπλές εγγραφές. Αν το ακίνητο είναι μοναδικό, καταχωρεί τα νέα δεδομένα στο καθολικό.

ChangeHouseImage() και ChangeHouseOwner():

Παράμετροι Εισόδου: μεταβλητή κλειδί, ενημερωμένα στοιχεία ακινήτου (π.χ. νέος ιδιοκτήτης ή ενημερωμένος αριθμός δωματίων).
Λειτουργικότητα: Ενημερώνει μια υπάρχουσα εγγραφή ακινήτου. Το chaincode διασφαλίζει ότι το ακίνητο υπάρχει και επαληθεύει την ακεραιότητα του αιτήματος ενημέρωσης πριν δεσμεύσει τις αλλαγές στο καθολικό.

queryHouse():
Παράμετρος Εισόδου: μεταβλητή κλειδί.
Λειτουργικότητα: Ανακτά και επιστρέφει τα στοιχεία του ακινήτου κάνοντας αναζήτηση στο καθολικό με βάση τη παρεχόμενη τιμή της μεταβλητής κλειδιού.

4.5 Λειτουργίες του API και Επικοινωνία με το Blockchain δίκτυο

Το API, κατασκευασμένο με χρήση του Spring Boot, λειτουργεί ως ενδιάμεσο επίπεδο μεταξύ της διαδικτυακής εφαρμογής και του blockchain. Χρησιμοποιεί το Hyperledger Fabric Java SDK για να επικοινωνεί με το δίκτυο και να εκτελεί λειτουργίες του chaincode. Το API παρέχει σημεία πρόσβασης (endpoints) για την εκτέλεση λειτουργιών CRUD (Δημιουργία, Ανάγνωση, Ενημέρωση, Διαγραφή) στα δεδομένα σπιτιών:

CreateNewHouse(): Υποβάλλει μια συναλλαγή στο blockchain για την προσθήκη ενός νέου ακινήτου.

UpdateHouseImage() και UpdateHouseOwner(): Στέλνουν μια συναλλαγή για την τροποποίηση ενός υφιστάμενου ακινήτου (ως προς την εικόνα του ακινήτου και τον ιδιοκτήτη του αντίστοιχα).

ClientAppTest(): Ανακτά δεδομένα ακινήτου από το blockchain.

4.6 Ο Ρόλος του Web Application

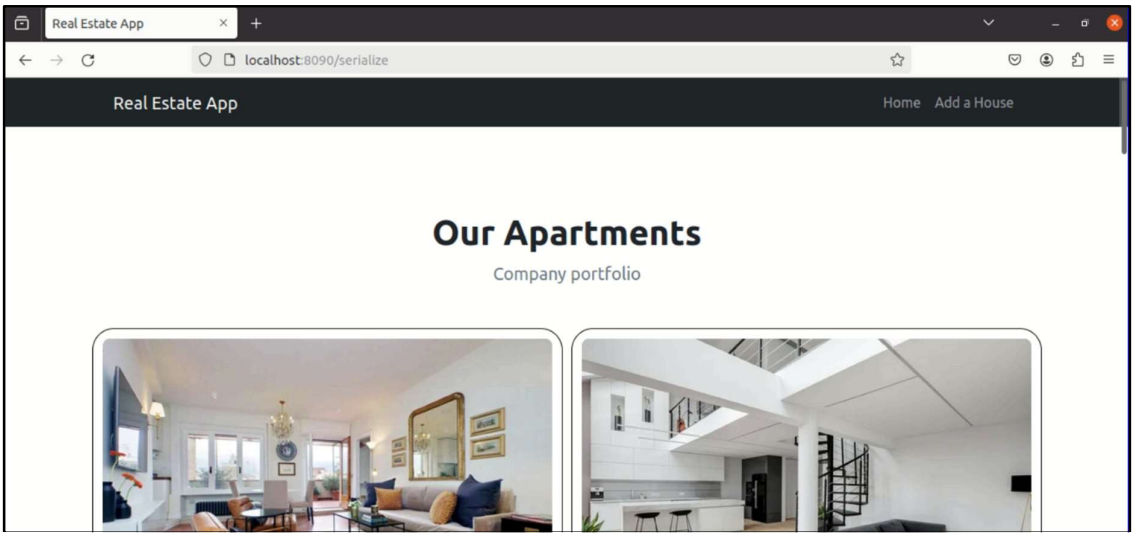
Η διαδικτυακή εφαρμογή είναι η διεπαφή που βλέπουν οι χρήστες και αλληλεπιδρά με το blockchain μέσω του API. Οι χρήστες μπορούν να:

Προβολή ακινήτων(Index()): Αναζητούν και εμφανίζουν λεπτομέρειες ακινήτων, συμπεριλαμβανομένων εικόνων και πληροφοριών ιδιοκτήτη, ανακτώντας δεδομένα από το blockchain.

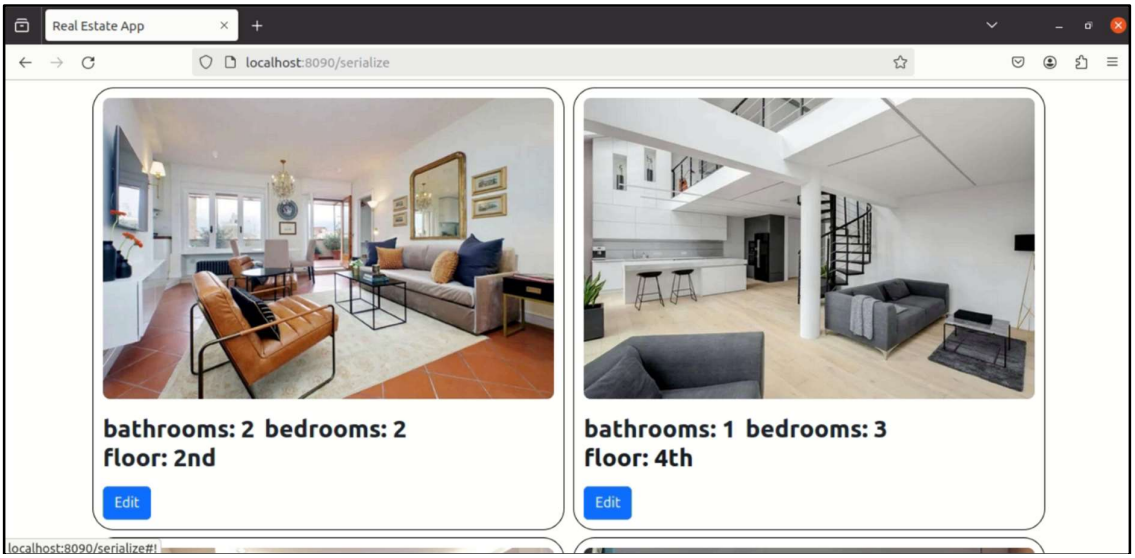
Προσθήκη ή ενημέρωση ακινήτων(CreateHouseForm(),EditHouseForm()):
Υποβάλλουν νέα δεδομένα ακινήτων ή ενημερώνουν υπάρχουσες εγγραφές, ενεργοποιώντας συναλλαγές που εκτελούνται μέσω του chaincode και αποθηκεύονται στο blockchain.

4.7 Αποτελέσματα

Δεδομένου ότι το δίκτυο blockchain και το API λειτουργούν κανονικά, ο χρήστης μπορεί να ανοίξει το web application για τη προβολή και την επεξεργασία των δεδομένων που περιέχονται στο ledger.

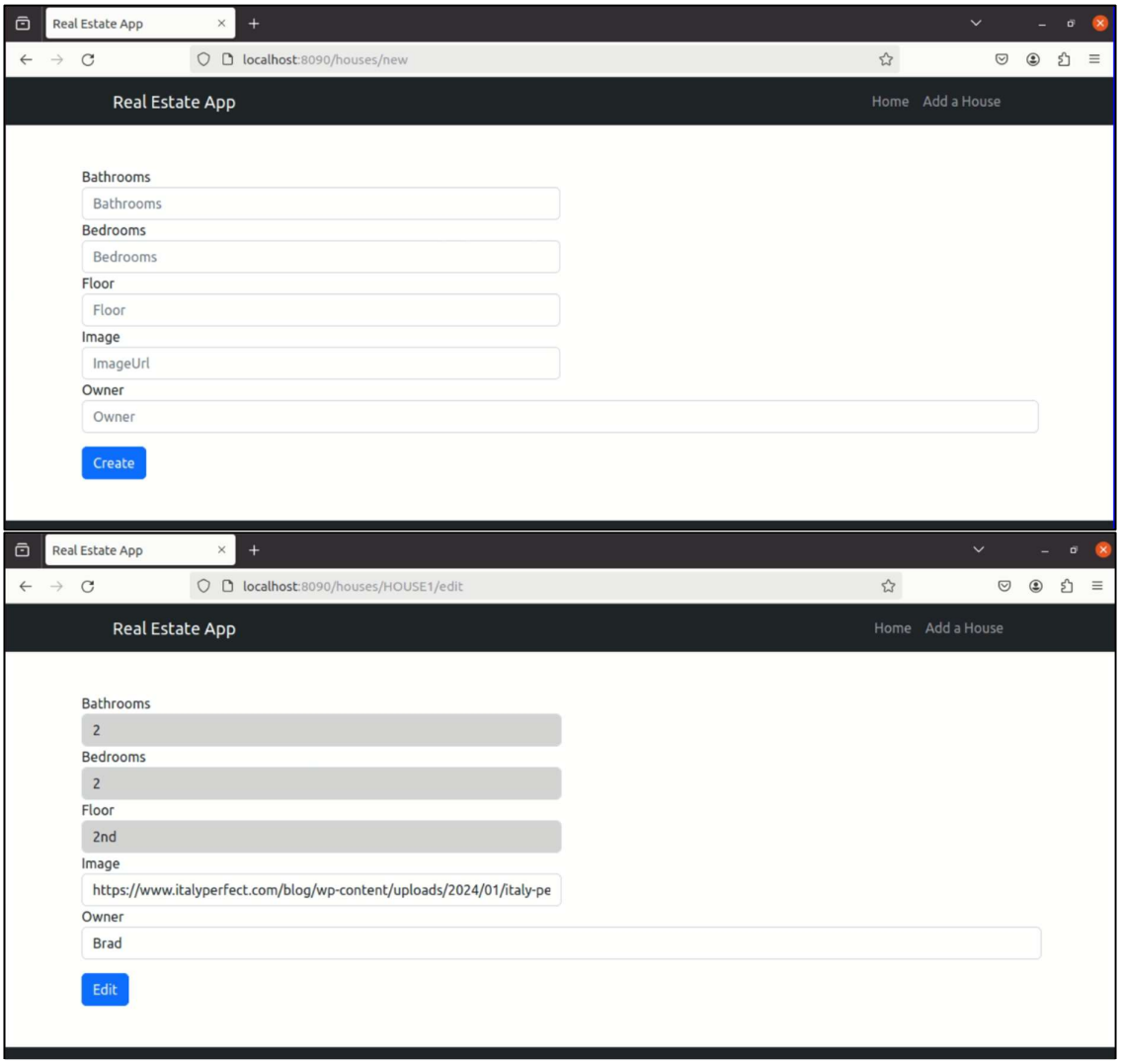


Εικόνα 8. Αρχική σελίδα του web application (1/2)



Εικόνα 9. Αρχική σελίδα του web application (2/2)

Εικόνα 10. Δημιουργία καινούργιας ακίνητης περιουσίας



Εικόνα 11. Επεξεργασία δεδομένων ακίνητης περιουσίας

Κεφάλαιο 5ο
Συμπεράσματα

Το Hyperledger Fabric είναι κρίσιμο για τις επιχειρηματικές λύσεις blockchain λόγω του δικτύου με άδειες (permissioned network), εξασφαλίζοντας ελεγχόμενη πρόσβαση και ιδιωτικότητα. Η αρθρωτή του αρχιτεκτονική επιτρέπει την προσαρμογή, υποστηρίζοντας διάφορες γλώσσες προγραμματισμού για smart contracts. Το Fabric επιτρέπει την ιδιωτικότητα μέσω ιδιωτικών καναλιών και διαχωρισμού δεδομένων, καθιστώντας το κατάλληλο για κλάδους όπως τα χρηματοοικονομικά και η υγειονομική περίθαλψη. Προσφέρει επεκτασιμότητα και βελτιώσεις απόδοσης με παράλληλη εκτέλεση συναλλαγών και αποδοτικούς μηχανισμούς συναίνεσης. Υποστηριζόμενο από το Ίδρυμα Linux, το Fabric διασφαλίζει ισχυρή διακυβέρνηση, υποστήριξη από την κοινότητα και ασφάλεια. Η δυνατότητά του να ενσωματώνεται με υπάρχοντα συστήματα και να εξυπηρετεί ένα ευρύ φάσμα χρήσεων το καθιστά ένα ευέλικτο και ισχυρό blockchain framework.

Το πληροφοριακό σύστημα που δημιούργησα περιλαμβάνει το blockchain δίκτυο, το Spring Boot API και το Spring Boot Web application. Το blockchain δίκτυο περιέχει τα δεδομένα των ακινήτων, όπως ο αριθμος των μπάνιων, των υπνοδωματίων, τον όροφο, εικόνα του ακινήτου και το όνομα του ιδιοκτήτη. Το API επικοινωνεί με το blockchain δίκτυο μέσω των smart contracts και τροφοδοτεί τα δεδομένα που λαμβανει στο web application για την παρουσίαση τους με περισσότερο εμφανίσιμο τρόπο στο τελικό χρήστη. . Μέσω του web app ο χρήστης μπορεί επίσης να προσθέσει ένα καινούργιο ακίνητο στο ledger, καθώς και να αλλάξει τον ιδιοκτήτη και την εικόνα του κάθε ακινήτου.

Η χρήση του Hyperledger Fabric για δεδομένα ακινήτων διασφαλίζει αποκέντρωση, εμπιστοσύνη, αμεταβλητότητα και ενισχυμένη ασφάλεια, επιτρέποντας διαφανείς, αδιάβλητες συναλλαγές και αυτοματοποιημένες διαδικασίες σε σύγκριση με μια κεντρικοποιημένη βάση δεδομένων.

Βιβλιογραφία

1. "The case of **HyperLedger Fabric** as a blockchain solution for healthcare **applications**" by A Adnane, F Ahmad, R Hussain
2. "**Hyperledger fabric**: a distributed operating system for permissioned blockchains" by E Androulaki, A Barger, V Bortnikov, C Cachin
3. "Configuration and Implementation of Novel and Safe Methodology for Processing **Real Estate** Transactions by Utilizing **Hyperledger Fabric**" by V Langaliya, JA Gohil
4. "Performance analysis of **hyperledger fabric** platforms" by Q Nasir, IA Qasse, M Abu Talib
5. "Performance characterization of **hyperledger fabric**" by A Baliga, N Solanki, S Verekar
6. "Architecture of the **hyperledger** blockchain **fabric**" by C Cachin
7. "Prototype of blockchain in dental care service **application** based on **hyperledger** composer in **hyperledger fabric** framework" by R Wutthikarn, YG Hui
8. "Potential risks of **hyperledger fabric** smart contracts" by K Yamashita, Y Nomura, E Zhou, B Pi
9. "A **hyper-ledger fabric** framework as a service for improved quality e-voting system" by PP Mukherjee, AA Boshra, MM Ashraf

Παράρτημα Α

Πιο κάτω παρουσιάζεται ο κώδικας που έγραψα σε γλώσσα προγραμματισμού Java.

Κλάση **House.java:** Χρησιμοποιείται ως μοντέλο και κάθε αντικείμενο αυτής της κλάσης αντιπροσωπεύει μια ακίνητη περιουσία. Περιέχει attributes και συναρτήσεις (functions) για την επίτευξη εμφάνισης και επεξεργασίας των δεδομένων των ακινήτων.

```java
/*
 * SPDX-License-Identifier: Apache-2.0
 */

package org.hyperledger.fabric.samples.fabcar;

import java.util.Objects;

import org.hyperledger.fabric.contract.annotation.DataType;
import org.hyperledger.fabric.contract.annotation.Property;

import com.owlike.genson.annotation.JsonProperty;

@DataType()
public final class House {

    @Property()
    private final String bathrooms;

    @Property()
    private final String bedrooms;

    @Property()
    private final String image;

    @Property()
    private final String floor;

    @Property()
    private final String owner;


    public String getBathrooms() {
        return bathrooms;
    }

    public String getBedrooms() {
        return bedrooms;
    }

    public String getImage() {
        return image;
    }

    public String getFloor() {
        return floor;
    }

    public String getOwner() {
        return owner;
    }

    public House(@JsonProperty("bathrooms") final String bathrooms, @JsonProperty("bedrooms") final String bedrooms,
  @JsonProperty("image") final String image, @JsonProperty("floor") final String floor, @JsonProperty("owner") final String owner) {
        this.bathrooms = bathrooms;
        this.bedrooms = bedrooms;
        this.image = image;
        this.floor = floor;
        this.owner = owner;
    }

```

```
60      @Override
61      public boolean equals(final Object obj) {
62          if (this == obj) {
63              return true;
64          }
65
66          if ((obj == null) || (getClass() != obj.getClass())) {
67              return false;
68          }
69
70          House other = (House) obj;
71
72          return Objects.deepEquals(new String[] {getBathrooms(), getBedrooms(), getImage(), getFloor(), getOwner()},
73                  new String[] {other.getBathrooms(), other.getBedrooms(), other.getImage(), other.getFloor(), other.getOwner()});
74      }
75
76      @Override
77      public int hashCode() {
78          return Objects.hash(getBathrooms(), getBedrooms(), getImage(), getFloor(), getOwner());
79      }
80
81      @Override
82      public String toString() {
83          return this.getClass().getSimpleName() + "@" + Integer.toHexString(hashCode()) + " [bathrooms=" + bathrooms + ", bedrooms="
84                  + bedrooms + ", image=" + image + ", floor=" + floor + ", owner=" + owner + "]";
85      }
86  }
```

Κλάση **HouseQueryResult.java:** Χρησιμοποιείται ως μοντέλο και κάθε αντικείμενο αυτής της κλάσης περιέχει μια ακίνητη περιουσία και μια μοναδική μεταβλητή κλειδί. Αντιπροσωπεύει τα αποτελέσματα αναζήτησης ενός ακινήτου για να ληφθούν τα δεδομένα του από το καθολικό (ledger).

```
1  /*
2   * SPDX-License-Identifier: Apache-2.0
3   */
4
5  package org.hyperledger.fabric.samples.fabcar;
6
7  import java.util.Objects;
8
9  import org.hyperledger.fabric.contract.annotation.DataType;
10 import org.hyperledger.fabric.contract.annotation.Property;
11
12 import com.owlike.genson.annotation.JsonProperty;
13
14 /**
15  * CarQueryResult structure used for handling result of query
16  *
17  */
18 @DataType()
19 public final class HouseQueryResult {
20      @Property()
21      private final String key;
22
23      @Property()
24      private final House record;
25
26      public HouseQueryResult(@JsonProperty("Key") final String key, @JsonProperty("Record") final House record) {
27          this.key = key;
28          this.record = record;
29      }
```

```
31      public String getKey() {
32          return key;
33      }
34
35      public House getRecord() {
36          return record;
37      }
38
39      @Override
40      public boolean equals(final Object obj) {
41          if (this == obj) {
42              return true;
43          }
44
45          if ((obj == null) || (getClass() != obj.getClass())) {
46              return false;
47          }
48
49          HouseQueryResult other = (HouseQueryResult) obj;
50
51          Boolean recordsAreEquals = this.getRecord().equals(other.getRecord());
52          Boolean keysAreEquals = this.getKey().equals(other.getKey());
53
54          return recordsAreEquals && keysAreEquals;
55      }
57      @Override
58      public int hashCode() {
59          return Objects.hash(this.getKey(), this.getRecord());
60      }
61
62      @Override
63      public String toString() {
64          return this.getClass().getSimpleName() + "@" + Integer.toHexString(hashCode()) + " [key=" + key + ", record="
65                  + record + "]";
66      }
67  }
```

Κλάση **FabHouse.java:** Αλληλεπιδρά με το καθολικό του blockchain χρησιμοποιώντας το Hyperledger Fabric. Παρέχει μεθόδους για την εκτέλεση διάφορων λειτουργιών που σχετίζονται με τα ακίνητα, όπως η δημιουργία, η ενημέρωση και η αναζήτηση ακινήτων. Αυτή η κλάση λειτουργεί ως γέφυρα μεταξύ της εφαρμογής και του έξυπνου συμβολαίου (chaincode) που έχει αναπτυχθεί στο blockchain.

```java
package org.hyperledger.fabric.samples.fabcar;

import java.util.ArrayList;
import java.util.List;

import org.hyperledger.fabric.contract.Context;
import org.hyperledger.fabric.contract.ContractInterface;
import org.hyperledger.fabric.contract.annotation.Contact;
import org.hyperledger.fabric.contract.annotation.Contract;
import org.hyperledger.fabric.contract.annotation.Default;
import org.hyperledger.fabric.contract.annotation.Info;
import org.hyperledger.fabric.contract.annotation.License;
import org.hyperledger.fabric.contract.annotation.Transaction;
import org.hyperledger.fabric.shim.ChaincodeException;
import org.hyperledger.fabric.shim.ChaincodeStub;
import org.hyperledger.fabric.shim.ledger.KeyValue;
import org.hyperledger.fabric.shim.ledger.QueryResultsIterator;

import com.owlike.genson.Genson;

@Contract(
        name = "FabHouse",
        info = @Info(
                title = "FabHouse contract",
                description = "The hyperlegendary house contract",
                version = "0.0.1-SNAPSHOT",
                license = @License(
                        name = "Apache 2.0 License",
                        url = "http://www.apache.org/licenses/LICENSE-2.0.html"),
                contact = @Contact(
                        email = "f.housee@example.com",
                        name = "F Housee",
                        url = "https://hyperledger.example.com")))
@Default
public final class FabHouse implements ContractInterface {

    private final Genson genson = new Genson();

    private enum FabHouseErrors {
        HOUSE_NOT_FOUND,
        HOUSE_ALREADY_EXISTS
    }


    @Transaction()
    public House queryHouse(final Context ctx, final String key) {
        ChaincodeStub stub = ctx.getStub();
        String houseState = stub.getStringState(key);

        if (houseState.isEmpty()) {
            String errorMessage = String.format("House %s does not exist", key);
            System.out.println(errorMessage);
            throw new ChaincodeException(errorMessage, FabHouseErrors.HOUSE_NOT_FOUND.toString());
        }

        House house = genson.deserialize(houseState, House.class);

        return house;
    }
```

```
    @Transaction()
    public void initLedger(final Context ctx) {
        ChaincodeStub stub = ctx.getStub();

        String[] houseData = {
                "{ \"bathrooms\": \"1\", \"bedrooms\": \"1\", \"image\": \"https://www.apartments.com/blog/sites/default/files/styles/-x_large_hq/public/image/2023-06/ParkLine-apartment-in-Miami-FL.jpg\", \"floor\": \"1st\", \"owner\": \"Tomoko\" }",
                "{ \"bathrooms\": \"2\", \"bedrooms\": \"2\", \"image\": \"https://www.italyperfect.com/blog/wp-content/uploads/2024/01/-italy-perfect-florence-vacation-rental-amore-two-bedroom-600x400.jpg\",  \"floor\": \"2nd\", \"owner\": \"Brad\" }",
                "{ \"bathrooms\": \"1\", \"bedrooms\": \"3\", \"image\": \"https://st4.depositphotos.com/6297298/40013/i/450/-depositphotos_400134112-stock-photo-elegant-kitchen-contemporary-living-room.jpg\",  \"floor\": \"4th\", \"owner\": \"Jin Soo\" }",
                "{ \"bathrooms\": \"2\", \"bedrooms\": \"2\", \"image\": \"https://dimitriosdoukas.gr/wp-content/uploads/-2022/11/1-600x400.jpg\",  \"floor\": \"1st\", \"owner\": \"Max\" }",
                "{ \"bathrooms\": \"2\", \"bedrooms\": \"2\", \"image\": \"https://www.openhouseathens.gr/wp-content/uploads/2024/03/-diamerisma-ignatiou-01-600x400.jpg\",  \"floor\": \"3rd\", \"owner\": \"Adrian\" }",
                "{ \"bathrooms\": \"1\", \"bedrooms\": \"3\", \"image\": \"https://barnes-greece.com/wp-content/uploads/-2100295469_BG_105747_2-600x400.jpg\",  \"floor\": \"3rd\", \"owner\": \"Michel\" }",
                "{ \"bathrooms\": \"1\", \"bedrooms\": \"1\", \"image\": \"https://static01.nyt.com/images/2020/11/13/t-magazine/13tmag-capetown-slide-NWOX/13tmag-capetown-slide-NWOX-articleLarge.jpg\",  \"floor\": \"2nd\", \"owner\": \"Aarav\" }",
                "{ \"bathrooms\": \"1\", \"bedrooms\": \"1\", \"image\": \"https://barnes-greece.com/wp-content/uploads/-6731189721_BG_105747_1-600x400.jpg\",  \"floor\": \"1st\", \"owner\": \"Pari\" }",
                "{ \"bathrooms\": \"2\", \"bedrooms\": \"1\", \"image\": \"https://barnes-greece.com/wp-content/uploads/-5853575304_BG_86523_7-600x400.jpg\",  \"floor\": \"1st\", \"owner\": \"Valeria\" }",
                "{ \"bathrooms\": \"1\", \"bedrooms\": \"2\", \"image\": \"https://static01.nyt.com/images/2021/02/17/t-magazine/17tmag-brule-slide-QUJX/17tmag-brule-slide-QUJX-articleLarge.jpg\",  \"floor\": \"4th\", \"owner\": \"Shotaro\" }"
        };
        for (int i = 0; i < houseData.length; i++) {
            String key = String.format("HOUSE%d", i);

            House house = genson.deserialize(houseData[i], House.class);
            String houseState = genson.serialize(house);
            stub.putStringState(key, houseState);
        }
    }

    @Transaction()
    public House createHouse(final Context ctx, final String key, final String bathrooms, final String bedrooms, final String image, final String floor, final String owner) {
        ChaincodeStub stub = ctx.getStub();

        String houseState = stub.getStringState(key);
        if (!houseState.isEmpty()) {
            String errorMessage = String.format("House %s already exists", key);
            System.out.println(errorMessage);
            throw new ChaincodeException(errorMessage, FabHouseErrors.HOUSE_ALREADY_EXISTS.toString());
        }

        House house = new House(bathrooms, bedrooms, image, floor, owner);
        houseState = genson.serialize(house);
        stub.putStringState(key, houseState);

        return house;
    }
    @Transaction()
    public String queryAllHouses(final Context ctx) {
        ChaincodeStub stub = ctx.getStub();

        final String startKey = "HOUSE1";
        final String endKey = "HOUSE99";
        List<HouseQueryResult> queryResults = new ArrayList<HouseQueryResult>();

        QueryResultsIterator<KeyValue> results = stub.getStateByRange(startKey, endKey);

        for (KeyValue result: results) {
            House house = genson.deserialize(result.getStringValue(), House.class);
            queryResults.add(new HouseQueryResult(result.getKey(), house));
        }

        final String response = genson.serialize(queryResults);

        return response;
    }
    @Transaction()
    public House changeHouseOwner(final Context ctx, final String key, final String newOwner) {
        ChaincodeStub stub = ctx.getStub();

        String houseState = stub.getStringState(key);

        if (houseState.isEmpty()) {
            String errorMessage = String.format("House %s does not exist", key);
            System.out.println(errorMessage);
            throw new ChaincodeException(errorMessage, FabHouseErrors.HOUSE_NOT_FOUND.toString());
        }

        House house = genson.deserialize(houseState, House.class);

        House newHouse = new House(house.getBathrooms(), house.getBedrooms(), house.getImage(), house.getFloor(), newOwner);
        String newHouseState = genson.serialize(newHouse);
        stub.putStringState(key, newHouseState);

        return newHouse;
    }
```

```
190     @Transaction()
191     public House changeHouseImage(final Context ctx, final String key, final String newImage) {
192         ChaincodeStub stub = ctx.getStub();
193
194         String houseState = stub.getStringState(key);
195
196         if (houseState.isEmpty()) {
197             String errorMessage = String.format("House %s does not exist", key);
198             System.out.println(errorMessage);
199             throw new ChaincodeException(errorMessage, FabHouseErrors.HOUSE_NOT_FOUND.toString());
200         }
201
202         House house = genson.deserialize(houseState, House.class);
203
204         House newHouse = new House(house.getBathrooms(), house.getBedrooms(), newImage, house.getFloor(), house.getOwner());
205         String newHouseState = genson.serialize(newHouse);
206         stub.putStringState(key, newHouseState);
207
208         return newHouse;
209     }
210 }
```

Κλάση **ApiController.java:** Η κλάση ApiController διαχειρίζεται τις εισερχόμενες αιτήσεις HTTP που σχετίζονται με τη διαχείριση ακινήτων. Παρέχει σημεία πρόσβασης για τη δημιουργία, ενημέρωση και αναζήτηση δεδομένων ακινήτων, αλληλεπιδρώντας με την υπηρεσία FabHouse, η οποία επικοινωνεί με το δίκτυο blockchain.

```
1  package com.hyperledger.demo;
2
3  import java.nio.file.Path;
26
27 @RestController
28 public class ApiController {
29
30     static {
31         System.setProperty("org.hyperledger.fabric.sdk.service_discovery.as_localhost", "true");
32     }
33
34     @GetMapping("/enrollAdmin")
35     public String enrollAdmin() throws Exception {
36         // Create a CA client for interacting with the CA.
37         Properties props = new Properties();
38         props.put("pemFile",
39             "../../test-network/organizations/peerOrganizations/org1.example.com/ca/ca.org1.example.com-cert.pem");
40         props.put("allowAllHostNames", "true");
41         HFCAClient caClient = HFCAClient.createNewInstance("https://localhost:7054", props);
42         CryptoSuite cryptoSuite = CryptoSuiteFactory.getDefault().getCryptoSuite();
43         caClient.setCryptoSuite(cryptoSuite);
44
45         // Create a wallet for managing identities
46         Wallet wallet = Wallets.newFileSystemWallet(Paths.get("wallet"));
```

```
47
48          // Check to see if we've already enrolled the admin user.
49          if (wallet.get("admin") != null) {
50              //System.out.println("An identity for the admin user \"admin\" already exists in the wallet");
51              return "An identity for the admin user \"admin\" already exists in the wallet";
52          }
53
54          // Enroll the admin user, and import the new identity into the wallet.
55          final EnrollmentRequest enrollmentRequestTLS = new EnrollmentRequest();
56          enrollmentRequestTLS.addHost("localhost");
57          enrollmentRequestTLS.setProfile("tls");
58          Enrollment enrollment = caClient.enroll("admin", "adminpw", enrollmentRequestTLS);
59          Identity user = Identities.newX509Identity("Org1MSP", enrollment);
60          wallet.put("admin", user);
61          //System.out.println("Successfully enrolled user \"admin\" and imported it into the wallet");
62
63          return "Successfully enrolled user \\\"admin\\\" and imported it into the wallet";
64      }
65
66      @GetMapping("/registerUser")
67      public String registerUser() throws Exception {
68          // Create a CA client for interacting with the CA.
69                  Properties props = new Properties();
70                  props.put("pemFile",
71                      "../../test-network/organizations/peerOrganizations/org1.example.com/ca/ca.org1.example.com-cert.pem");
72                  props.put("allowAllHostNames", "true");
73                  HFCAClient caClient = HFCAClient.createNewInstance("https://localhost:7054", props);
74                  CryptoSuite cryptoSuite = CryptoSuiteFactory.getDefault().getCryptoSuite();
75                  caClient.setCryptoSuite(cryptoSuite);
76
77                  // Create a wallet for managing identities
78                  Wallet wallet = Wallets.newFileSystemWallet(Paths.get("wallet"));
79
80                  // Check to see if we've already enrolled the user.
81                  if (wallet.get("appUser") != null) {
82                      //System.out.println("An identity for the user \"appUser\" already exists in the wallet");
83                      return "An identity for the user \"appUser\" already exists in the wallet";
84                  }
85
86                  X509Identity adminIdentity = (X509Identity)wallet.get("admin");
87                  if (adminIdentity == null) {
88                      //System.out.println("\"admin\" needs to be enrolled and added to the wallet first");
89                      return "\"admin\" needs to be enrolled and added to the wallet first";
90                  }
91                  User admin = new User() {
92
93                      @Override
94                      public String getName() {
95                          return "admin";
96                      }
97
98                      @Override
99                      public Set<String> getRoles() {
100                         return null;
101                     }
102
103                     @Override
104                     public String getAccount() {
105                         return null;
106                     }
107
108                     @Override
109                     public String getAffiliation() {
110                         return "org1.department1";
111                     }
```

```
                @Override
                public Enrollment getEnrollment() {
                    return new Enrollment() {

                        @Override
                        public PrivateKey getKey() {
                            return adminIdentity.getPrivateKey();
                        }

                        @Override
                        public String getCert() {
                            return Identities.toPemString(adminIdentity.getCertificate());
                        }
                    };
                }

                @Override
                public String getMspId() {
                    return "Org1MSP";
                }

            };
               // Register the user, enroll the user, and import the new identity into the wallet.
               RegistrationRequest registrationRequest = new RegistrationRequest("appUser");
               registrationRequest.setAffiliation("org1.department1");
               registrationRequest.setEnrollmentID("appUser");
               String enrollmentSecret = caClient.register(registrationRequest, admin);
               Enrollment enrollment = caClient.enroll("appUser", enrollmentSecret);
               Identity user = Identities.newX509Identity("Org1MSP", enrollment);
               wallet.put("appUser", user);
               //System.out.println("Successfully enrolled user \"appUser\" and imported it into the wallet");
               return "Successfully enrolled user \"appUser\" and imported it into the wallet";
    }

    @GetMapping("/clientAppTest")
    public String clientAppTest() throws Exception {
        // Load a file system based wallet for managing identities.
                Path walletPath = Paths.get("wallet");
                Wallet wallet = Wallets.newFileSystemWallet(walletPath);
                // load a CCP
                Path networkConfigPath = Paths.get("..", "..", "test-network", "organizations", "peerOrganizations",
                        "org1.example.com", "connection-org1.yaml");

                Gateway.Builder builder = Gateway.createBuilder();
                builder.identity(wallet, "appUser").networkConfig(networkConfigPath).discovery(true);

                // create a gateway connection
                try (Gateway gateway = builder.connect()) {

                    // get the network and contract
                    Network network = gateway.getNetwork("mychannel");
                    Contract contract = network.getContract("fabhouse");

                    byte[] result;
                    //byte[] result2;
                    String finalResult;



                    result = contract.evaluateTransaction("queryAllHouses");
                    //result2 = result;
                    //System.out.println(new String(result));
                    finalResult = new String(result);


                    return finalResult;
                }
        }

```

```
193         @GetMapping("/createNewHouse")
194         public String createNewHouse(String key, String bathrooms, String bedrooms, String image,
195                 String floor, String owner) throws Exception {
196             Path walletPath = Paths.get("wallet");
197             Wallet wallet = Wallets.newFileSystemWallet(walletPath);
198             // load a CCP
199             Path networkConfigPath = Paths.get("..", "..", "test-network", "organizations", "peerOrganizations",
200                     "org1.example.com", "connection-org1.yaml");
201
202             Gateway.Builder builder = Gateway.createBuilder();
203             builder.identity(wallet, "appUser").networkConfig(networkConfigPath).discovery(true);
204
205             // create a gateway connection
206             try (Gateway gateway = builder.connect()) {
207
208                 // get the network and contract
209                 Network network = gateway.getNetwork("mychannel");
210                 Contract contract = network.getContract("fabhouse");
211
212                 byte[] result;
213                 //byte[] result2;
214                 String finalResult = null;
215
214                  contract.submitTransaction("createHouse", key, bathrooms, bedrooms, image, floor, owner);
215
216                  result = contract.evaluateTransaction("queryHouse", key);
217                  //System.out.println(new String(result));
218                  finalResult = finalResult + new String(result);
219                  finalResult = finalResult + "\r\n";
220
221                  return finalResult + key;
222
223              }
224
225          }
227         @GetMapping("/updateHouseOwner")
228         public String updateHouseOwner(String key, String newOwner) throws Exception {
229
230             Path walletPath = Paths.get("wallet");
231             Wallet wallet = Wallets.newFileSystemWallet(walletPath);
232             // load a CCP
233             Path networkConfigPath = Paths.get("..", "..", "test-network", "organizations", "peerOrganizations",
234                     "org1.example.com", "connection-org1.yaml");
235
236             Gateway.Builder builder = Gateway.createBuilder();
237             builder.identity(wallet, "appUser").networkConfig(networkConfigPath).discovery(true);
238
239             // create a gateway connection
240             try (Gateway gateway = builder.connect()) {
241
242                 // get the network and contract
243                 Network network = gateway.getNetwork("mychannel");
244                 Contract contract = network.getContract("fabhouse");
245
246                 byte[] result;
247                 //byte[] result2;
248                 String finalResult = null;
249                 contract.submitTransaction("changeHouseOwner", key, newOwner);
250
251                 result = contract.evaluateTransaction("queryHouse", key);
252                 //System.out.println(new String(result));
253                 finalResult = finalResult + new String(result);
254
255                 return finalResult;
256
257             }
258
259         }
```

```
261        @GetMapping("/updateHouseImage")
262        public String updateHouseImage(String key, String newImage) throws Exception {
263
264            Path walletPath = Paths.get("wallet");
265            Wallet wallet = Wallets.newFileSystemWallet(walletPath);
266            // load a CCP
267            Path networkConfigPath = Paths.get("..", "..", "test-network", "organizations", "peerOrganizations",
268                    "org1.example.com", "connection-org1.yaml");
269
270            Gateway.Builder builder = Gateway.createBuilder();
271            builder.identity(wallet, "appUser").networkConfig(networkConfigPath).discovery(true);
272
273            // create a gateway connection
274            try (Gateway gateway = builder.connect()) {
275
276                // get the network and contract
277                Network network = gateway.getNetwork("mychannel");
278                Contract contract = network.getContract("fabhouse");
279
280                byte[] result;
281                //byte[] result2;
282                String finalResult = null;
283                contract.submitTransaction("changeHouseImage", key, newImage);
284
285                result = contract.evaluateTransaction("queryHouse", key);
286                //System.out.println(new String(result));
287                finalResult = finalResult + new String(result);
288
289                return finalResult;
290
291            }
292        }
```

Κλάση **HomeController.java:** Δέχεται και διαχειρίζεται δεδομένα για τα ακίνητα μέσω HTTP αιτήσεων με τις οποίες επικοινωνεί με το API. Έπειτα γεμίζει τα μοντέλα με τα δεδομένα που λαμβάνει και τα εμφανίζει σε μορφή προβολών (views) που επιστρέφει.

```
1  package com.example.demo;
2
3  import java.util.ArrayList;
16
17 @Controller
18 public class HomeController {
19
20     public HouseQueryResult FindHR(String houseId) throws Exception{
21         RestTemplate rT = new RestTemplate();
22         HttpHeaders header = new HttpHeaders();
23         header.add("Accept", "application/json");
24
25         ResponseEntity<String> response = rT.getForEntity("http://localhost:8080/clientAppTest", String.class);
26
27         ObjectMapper mapper = new ObjectMapper();
28
29         String json = response.getBody().toString();
30         HouseQueryResult[] hq = mapper.readValue(json, HouseQueryResult[].class);
42         for (HouseQueryResult x : hq) {
43
44             if(x.key.equals(houseId)) {
45
46                 return x;
47             }
48
49         }
50         return new HouseQueryResult();
51     }
```

```java
@GetMapping("/home")
public String Index(Model model) throws Exception {
    RestTemplate rT = new RestTemplate();
    HttpHeaders header = new HttpHeaders();
    header.add("Accept", "application/json");

    MultiValueMap<String,String> body = new LinkedMultiValueMap<String,String>();

    HttpEntity<MultiValueMap<String,String>> requestHttp = new HttpEntity<MultiValueMap<String,String>>(body,header);

    ResponseEntity<String> response = rT.getForEntity("http://localhost:8080/enrollAdmin", String.class);
    response = rT.getForEntity("http://localhost:8080/registerUser", String.class);
    response = rT.getForEntity("http://localhost:8080/clientAppTest", String.class);

    ObjectMapper mapper = new ObjectMapper();

    String json = response.getBody().toString();
    HouseQueryResult[] hq = mapper.readValue(json, HouseQueryResult[].class);
    List<HouseQueryResult> houses = Arrays.asList(hq);

    model.addAttribute("houses",houses);
    return "houses-list";
}
 @GetMapping("/houses/new")
 public String createHouseForm(Model model) throws Exception {
     House house = new House();
     model.addAttribute("house",house);
     return "houses-create";
 }
@PostMapping("/houses/new")
public String saveHouse(@ModelAttribute("house") House house) throws RestClientException, Exception {
    RestTemplate rT = new RestTemplate();
    HttpHeaders header = new HttpHeaders();
    header.add("Accept", "application/json");
    String url = "http://localhost:8080/createNewHouse?key=HOUSE" + (housesAmount() + 1) +
            "&bathrooms=" + house.getBathrooms() + "&bedrooms=" + house.getBedrooms() + "&floor=" + house.getFloor() +
            "&image=" + house.getImage() + "&owner=" + house.getOwner();
    ResponseEntity<String> response = rT.getForEntity(url, String.class);

    String json = response.getBody().toString();
    if(json.contains(house.getBathrooms())) {
        return "redirect:/home";
    }
    else {
        return "error";
    }

}

 @GetMapping("/houses/{houseId}/edit")
 public String editHouseForm(@PathVariable("houseId") String houseId, Model model) throws Exception {
     HouseQueryResult hqr = FindHR(houseId);
     model.addAttribute("house", hqr);
     return "houses-edit";
 }
@PostMapping("/houses/{houseId}/edit")
public String updateHouse(@PathVariable("houseId") String houseId, @ModelAttribute("house") HouseQueryResult hqr) {
    RestTemplate rT = new RestTemplate();
    HttpHeaders header = new HttpHeaders();
    header.add("Accept", "application/json");
    String url = "http://localhost:8080/updateHouseOwner?key=" + houseId + "&newOwner=" + hqr.record.owner;
    ResponseEntity<String> response = rT.getForEntity(url, String.class);

    String json = response.getBody().toString();
    /*-----------------------------------------------*/
    url = "http://localhost:8080/updateHouseImage?key=" + houseId + "&newImage=" + hqr.record.image;
    response = rT.getForEntity(url, String.class);

    json = response.getBody().toString();
    /*-----------------------------------------------*/
    if(json.contains(hqr.record.getBathrooms())) {
        return "redirect:/home";
    }
    else {
        return "error";
    }
}
```