ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

# Ενοποίηση συστημάτων blockchain με υπάρχοντα υπολογιστικά οικοσυστήματα με χρήση Service oriented architecture Modeling Language

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παναγιώτης Π. Ζευγολατάκος

**Επιβλέπων :** Βασίλειος Βεσκούκης

Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2022

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

# Ενοποίηση συστημάτων blockchain με υπάρχοντα υπολογιστικά οικοσυστήματα με χρήση Service oriented architecture Modeling Language

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παναγιώτης Π. Ζευγολατάκος

**Επιβλέπων :** Βασίλειος Βεσκούκης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 13η Ιουλίου 2022.

| | | |
|---|---|---|
| ……. | ……. | ……. |
| Βασίλειος Βεσκούκης | Άρης Παγουρτζής | Παναγιώτης Τσανάκας |
| Καθηγητής Ε.Μ.Π. | Καθηγητής Ε.Μ.Π. | Καθηγητής Ε.Μ.Π. |

Αθήνα, Ιούλιος 2022

…….
Παναγιώτης Π. Ζευγολατάκος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

# Περίληψη

Η επικοινωνία μεταξύ επαγγελματιών που συμμετέχουν στην ανάπτυξη σύνθετων υπολογιστικών οικοσυστημάτων είναι ζωτικής σημασίας. Σε περίπτωση συνεργασίας οργανισμών, είναι σύνηθες να υπάρχουν προβλήματα στην ανάπτυξη, τον εμπλουτισμό ή την ενοποίηση υπαρχουσών (legacy) εφαρμογών με νέες, εξαιτίας της δυσκολίας ανταλλαγής δεδομένων και υπηρεσιών μεταξύ διαφορετικών εφαρμογών, ιδιαίτερα όταν αυτές προέρχονται από διαφορετικούς κατασκευαστές, χρησιμοποιούν διαφορετικές τεχνολογίες, κ.ά..

Με την εξάπλωση της τεχνολογίας blockchain και του Web 3.0, ολοένα και περισσότεροι οργανισμοί στρέφονται στην ενοποίηση των συστημάτων τους με τις καινούριες αυτές τεχνολογίες προκειμένου να ωφεληθούν από τη χρήση τους. Ωστόσο, η επιτυχής επικοινωνία των υπαρχόντων (legacy) πιθανώς πολύπλοκων πληροφοριακών συστημάτων με συστήματα που βασίζονται σε blockchain, με τρόπο που να είναι παραγωγική η ενοποίηση και η αξιοποίηση των δυνατοτήτων τους παραμένει προβληματική, δυσκολεύοντας τη συνεργασία μεταξύ οργανισμών και επιβραδύνοντας, ή καθιστώντας ακριβή και επιρρεπή σε σφάλματα την υιοθέτηση νέων τεχνολογιών.

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η διερεύνηση της αξιοποίησης πρότυπης τεκμηρίωσης υπαρχόντων πληροφοριακών συστημάτων και της χρήσης της στην ανάπτυξη ενός αξιόπιστου τρόπου περιγραφής και τεκμηρίωσης της ενοποίησης (integration) συστημάτων, έτσι ώστε να γίνεται ευκολότερη η επικοινωνία τους με νέα πληροφοριακά συστήματα που εισάγονται στο εκάστοτε οικοσύστημα, και συγκεκριμένα με συστήματα που χρησιμοποιούν blockchain.

Συγκεκριμένα, θα σχεδιαστεί και υλοποιηθεί ένα σενάριο επικοινωνίας των χαρακτηριστικών (υπηρεσιών) ενός συστήματος με τη χρήση Service oriented architecture Modeling Language (SoaML). Αρχικά, θα πραγματοποιηθεί η ανάπτυξη μιας εφαρμογής ιστού με αρχιτεκτονική SOA για την οποία θα κατασκευαστεί και η αντίστοιχη τεκμηρίωση με SoaML. Στη συνέχεια, θα εκτελεστεί ένα υποθετικό σενάριο στο οποίο θα αναπτυχθεί μια επιπλέον εφαρμογή με τεχνολογία blockchain, και θα διερευνηθεί η χρησιμότητα της τεκμηρίωσης SOA για να πραγματοποιηθεί η ενοποίηση (integration) με την αρχική εφαρμογή.

**Λέξεις κλειδιά**

SoaML, SOA, υπηρεσία, UML, εφαρμογή ιστού, frontend, backend, blockchain, web3, έξυπνα συμβόλαια, blockchain oracle

# Abstract

The communication between professionals involved in the development of complex computer ecosystems is vital. In the case of cooperation amongst organizations, it is usual for problems to exist in the development, enrichment or integration of existing (legacy) applications with new ones, due to the difficulty of information and service exchange between different applications, especially when they derive from different manufacturers, use different technologies, etc.

With the spreading of blockchain technology and Web 3.0, more and more organizations turn to the unification of their systems with these new technologies in order to benefit from their use. However, the successful communication between existing (legacy) possibly complex information systems and systems based on blockchain, in a way where the integration and utilization of their capabilities remain problematic, making the cooperation amongst organizations difficult and slowing down, or rendering the adoption of new technologies expensive and problematic.

The purpose of the present diploma thesis is the investigaion of a reliable model documentation of existing information systems and its use in the development of a reliable way to describe and document system integration, so that their communication with new information systems that are integrated in each ecosystem becomes easier, specifically with systems that use blockchain.

Specifically, a communication of a system's characteristics (services) scenario will be designed and implemented with the use of Service oriented architecture Modeling Language (SoaML). Initially, a web application with SOA architecture will be developed, along with the respective SoaML documentation. Subsequently, a hypothetical scenario will be executed in which another application with blockchain technology will be developed, and the use of the SOA documentation will be investigated so that the integration with the initial application is realized.

**Keywords**

SoaML, SOA, service, UML, web application, frontend, backend, blockchain, web3, smart contracts, blockchain oracle

# Ευχαριστίες

Με την ολοκλήρωση της εργασίας αυτής, σηματοδοτείται το τέλος ενός πολύχρονου και κοπιαστικού, αλλά εποικοδομητικού ταξιδιού. Πέντε χρόνια γεμάτα εμπειρίες, που με άλλαξαν και με έκαναν έναν καλύτερο άνθρωπο.

Αρχικά θέλω να ευχαριστήσω την οικογένεια μου, που ήταν δίπλα μου σε κάθε βήμα. Δεν είναι υπερβολή να πω πως επειδή ξέρω πως ο πατέρας μου, η μητέρα μου και η αδερφή μου είναι εκεί για μένα, μπορώ να συγκεντρωθώ και να πετύχω τους στόχους μου.

Επίσης θέλω να ευχαριστήσω τους φίλους μου, τόσο εντός όσο και εκτός σχολής, εφόσον δε θα τα είχα καταφέρει χωρίς την υποστήριξη τους.

Τέλος, θέλω να ευχαριστήσω τον καθηγητή και επιβλέπων μου κ. Βασίλειο Βεσκούκη και τον υποψήφιο διδάκτορα και ερευνητή Ιωάννη Τζαννέτο, οι οποίοι με βοήθησαν να φέρω εις πέρας την παρούσα εργασία και μου παρείχαν βοήθεια όταν τη χρειαζόμουν.

# Πίνακας Περιεχομένων

# Κατάλογος Σχημάτων

*Κεφάλαιο 1*

# Εισαγωγή

## 1.1 Κίνητρο

Η εξάπλωση της τεχνολογίας blockchain και του Web 3.0 έχει παρακινήσει τον κόσμο, ιδιαίτερα αυτούς που αναπτύσσουν εφαρμογές και πληροφοριακά συστήματα, να τα εντάξουν στις επιχειρήσεις, αλλά και στις ζωές τους. Ωστόσο, όπως με κάθε καινούρια τεχνολογία, χρειάζεται χρόνος και κόπος για την επιτυχή υιοθέτηση και αποτελεσματική χρήση τους.

Επιπλέον, η ενοποίησή τους με τα υπάρχοντα συστήματα και εφαρμογές δεν είναι απλή, ειδικά όσον αφορά περιπτώσεις ανταλλαγής μεγάλου όγκου πληροφορίων, όπως στην περίπτωση της συνεργασίας οργανισμών, αλλά και εσωτερικά σε μεγάλους οργανισμούς. Είναι σημαντικό να υπάρχει ένας αξιόπιστος τρόπος τεκμηρίωσης των υπαρχόντων συστημάτων, έτσι ώστε να γίνεται ξεκάθαρο το ποιες είναι οι λειτουργίες τους, ώστε να είναι πιο ομαλή η ενοποίηση με νέα πληροφοριακά συστήματα και συγκεκριμένα με υιοθέτηση της τεχνολογίας blockchain.

Στην παρούσα εργασία θα διερευνηθεί μέσα από ένα παράδειγμα το πώς μπορεί να χρησιμοποιηθεί το πρότυπο SoaML για την τεκμηρίωση ενός υπάρχοντος συστήματος που βασίζεται σε αρχιτεκτονική SOA, προκειμένου να μπορεί να γίνει όσο το δυνατόν πιο αποτελεσματικά, δηλαδή παραγωγικά και, ει δυνατόν, χωρίς σφάλματα, η ενοποίηση μιας εφαρμογής blockchain σε αυτό. Με την έννοια "ενοποίηση" αναφερόμαστε στη χρήση υπηρεσιών που προσφέρονται πάνω από τον κεντρικό δίαυλο της αρχιτεκτονικής SOA (Enterprise Service Bus) από μια νέα εφαρμογή με τεχνολογία Blockchain.

## 1.2 Δομή της εργασίας

Στο **Κεφάλαιο 2** γίνεται ο ορισμός της Service oriented architecture Modeling Language (SoaML) βάσει του specification της από το Object Management Group (OMG) [1]. Επίσης παρουσιάζονται τα διαγράμματα που προσφέρει το Visual Paradigm [2].

Στο **Κεφάλαιο 3** γίνεται συνοπτικός ορισμός του blockchain, των βασικών χαρακτηριστικών και τρόπου λειτουργίας του, καθώς και των έξυπνων συμβολαίων, του Web 3.0 και των oracles.

Στο **Κεφάλαιο 4** αναλύεται η περίπτωση χρήσης, η αρχιτεκτονική και οι τεχνολογίες που χρησιμοποιήθηκαν προκειμένου να δημιουργηθούν οι εφαρμογές, καθώς και η μεθοδολογία που ακολουθήθηκε.

Στο **Κεφάλαιο 5** παρουσιάζεται η πρώτη φάση της περίπτωσης χρήσης: η δημιουργία του αρχικού συστήματος και των διαγραμμάτων SoaML που θα περιγράψουν τις υπηρεσίες που προσφέρονται.

Στο **Κεφάλαιο 6** αναλύεται η δεύτερη φάση της περίπτωσης χρήσης: η δημιουργία μιας νέας υπηρεσίας blockchain αποκλειστικά βάσει των διαγραμμάτων και η ενοποίησή της με το αρχικό σύστημα.

Στο **Κεφάλαιο 7** αναφέρονται τα συμπεράσματα της εργασίας καθώς και πιθανές μελλοντικές χρήσεις και επεκτάσεις.

*Κεφάλαιο 2*

# Service oriented architecture Modeling Language

## 2.1 Τι είναι SoaML

Μια υπηρεσία είναι αξία που προσφέρεται μέσω μιας καλά ορισμένης διεπαφής, είναι διαθέσιμη σε μια κοινότητα (που μπορεί να είναι το ευρύ κοινό) και έχει ως αποτέλεσμα την παροχή έργου στον καταναλωτή.

Η Service Oriented Architecture (SOA) είναι ένας τρόπος περιγραφής και κατανόησης οργανισμών, κοινωνιών, συστημάτων και ανθρώπων οι οποίοι προσφέρουν υπηρεσίες ο ένας στον άλλον. Οι υπηρεσίες αυτές επιτρέπουν σε κάποιον να φέρει ένα στόχο εις πέρας χωρίς να τον πραγματοποιήσει ο ίδιος, πολλές φορές μη γνωρίζοντας καν τον τρόπο να επιτευχθεί, αυξάνοντας έτσι την αποτελεσματικότητα. Με τις υπηρεσίες είναι επίσης εφικτό να προσφερθούν και οι ικανότητες ανθρώπων με κάποιο αντάλλαγμα, επιτρέποντας έτσι την εγκαθίδρυση κοινωνιών, διαδικασιών και οργανισμών.

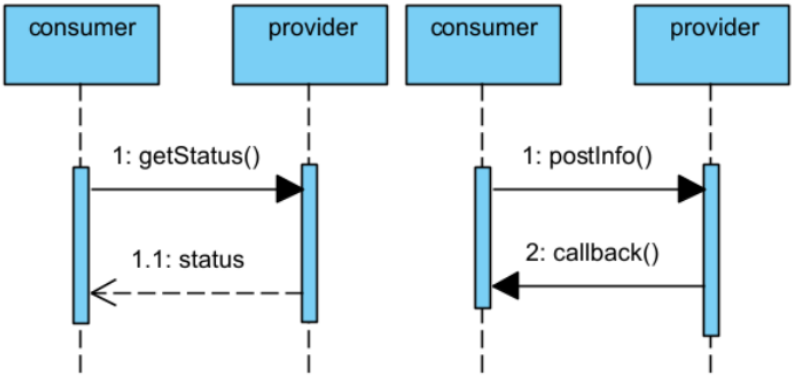
Η SoaML είναι ένας τρόπος να σχεδιάζονται και να τεκμηριώνονται αρχιτεκτονικές που ακολουθούν τη SOA χρησιμοποιώντας τη Unified Modeling Language (UML). Επιτρέπει την υποστήριξη αρχιτεκτονικών επιχειρήσεων, κοινωνιών ή και πληροφοριακών συστημάτων, εφόσον όλα μπορούν να είναι εξίσου service oriented. Με αυτόν τον τρόπο διαχωρίζεται το αποτέλεσμα μιας υπηρεσίας από τον τρόπο και τον τόπο της εφαρμογής της, καθώς και το ποιος ή τι θα την πραγματοποιήσει.

Υπάρχουν δύο προσεγγίσεις για την τεκμηρίωση υπηρεσιών σε SoaML: με χρήση *διεπαφής* ή με χρήση *συμβολαίου*. Συγκεκριμένα, για την πρώτη περίπτωση προσφέρονται δύο είδη διεπαφών:

- Απλά Interfaces: χρησιμοποιούνται για τη μονόδρομη αλληλεπίδραση που προσφέρεται από έναν συμμετέχοντα σε μια θύρα και εκπροσωπείται ως ένα UML *Interface* στοιχείο. Προαιρετικά, μπορεί να είναι και ένα *ServiceInterface* στοιχείο. Ο πάροχος λαμβάνει λειτουργίες στη θύρα και μπορεί να παραδώσει αποτελέσματα στον καταναλωτή.
- ServiceInterface: χρησιμοποιούνται για τις αμφίδρομες υπηρεσίες μεταξύ του καταναλωτή ο οποίος συμφωνεί να χρησιμοποιήσει μια υπηρεσία όπως αυτή είναι ορισμένη από τη διεπαφή που ο πάροχος έχει συμφωνήσει να προμηθεύσει. Η συμβατότητα των διεπαφών υπηρεσιών καθορίζει αν οι συμφωνίες αυτές είναι συνεπείς και επομένως μπορεί να πραγματοποιηθεί η σύνδεση προκειμένου να επιτευχθεί η υπηρεσία και ανταλλαγή σε αξία. Για την τεκμηρίωση χρησιμοποιείται το στοιχείο *ServiceInterface*, το οποίο είναι ένα

είδος *θύρας "Υπηρεσίας"* από την πλευρά του παρόχου και ένα είδος *θύρας "Αιτήματος"* από την πλευρά του καταναλωτή.

Η διαφορά μεταξύ των δύο είναι ότι ένα απλό interface είναι μια εκφυλισμένη περίπτωση του ServiceInterface (και του ServiceContract), όπου η υπηρεσία είναι μονόδρομη – ο καταναλωτής καλεί λειτουργίες του παρόχου – ο πάροχος δεν κάνει callback στον καταναλωτή και μπορεί να μη γνωρίζει καν ποιος είναι ο καταναλωτής.




**Σχήμα 2.1** Simple Interface vs ServiceInterface

Όσον αφορά τη χρήση συμβολαίου:

- ServiceContract: ένα *συμβόλαιο υπηρεσίας* ορίζει τον τρόπο με τον οποίο οι πάροχοι, οι καταναλωτές και (ενδεχομένως) τρίτοι συνεργάζονται προκειμένου να ανταλλάξουν αξία, που είναι εν τέλει το αποτέλεσμα εκτέλεσης μιας υπηρεσίας. Εκπροσωπεί τη συμφωνία μεταξύ των συμμετεχόντων και τον τρόπο με τον οποίο η υπηρεσία παρέχεται και καταναλώνεται, τις διεπαφές, το choreography, καθώς και υπάρχοντες όρους & προϋποθέσεις. Η συμφωνία αυτή μπορεί να οριστεί εκ των προτέρων ή να επέλθει δυναμικά, εφόσον υπάρχει τη στιγμή που το συμβόλαιο εκτελεστεί. Αν ο πάροχος και ο καταναλωτής υποστηρίζουν διαφορετικά συμβόλαια τότε πρέπει να υπάρξει συμφωνία πως τα συμβόλαια αυτά είναι συμβατά και συνεπή με τις δεσμεύσεις των συμμετεχόντων. Εν ολίγοις, η προσέγγιση του συμβολαίου υπηρεσίας βασίζεται στο ότι το συμβόλαιο είναι "στη μέση" και τα άκρα, δηλαδή οι συμμετέχοντες, συμφωνούν στο ρόλο τους στο συμβόλαιο ή προσαρμόζονται σε αυτό.

## 2.2 Στερεότυπα και κατηγοριοποίηση

Στο SoaML specification του Object Management Group [1] ορίζονται τα παρακάτω στερεότυπα για χρήση στα διαγράμματα:

- *Agent*: είναι μια κατηγοριοποίηση αυτόνομων οντοτήτων που μπορούν να προσαρμοστούν και να αλληλεπιδράσουν με το περιβάλλον τους. Περιγράφει ένα σετ από στιγμιότυπα Agent που έχουν κοινά χαρακτηριστικά, περιορισμούς και σημασιολογία. Στη SoaML οι Agents είναι και Participants, παρέχοντας και χρησιμοποιώντας υπηρεσίες.

- *Attachment*: μέρος ενός *MessageType* που επισυνάπτεται αντί να περιέχεται στο μήνυμα.
- *Capability*: η ικανότητα για ενέργεια και δημιουργία έκβασης που φέρει εις πέρας ένα αποτέλεσμα. Μπορεί να ορίζει τόσο τη γενική ικανότητα ενός συμμετέχοντα όσο και την ικανότητα να παρέχεται μια υπηρεσία.
- *Consumer*: μοντελοποιεί τον τύπο του καταναλωτή υπηρεσίας. Στη συνέχεια χρησιμοποιείται ως τύπος ενός ρόλου στο συμβόλαιο και ως τύπος μιας θύρας ενός συμμετέχοντα.
- *Collaboration*: δείχνει αν οι δεσμεύσεις των ρόλων στα άκρα των *CollaborationUses* που ορίζονται από ένα Collaboration επιβάλλονται αυστηρά ή όχι.
- *CollaborationUse*: δηλώνει αν οι δεσμεύσεις των ρόλων στα άκρα του επιβάλλονται αυστηρά ή όχι.
- *Expose*: χρησιμοποιείται για να δείξει ένα Capability το οποίο εκτίθεται μέσω ενός ServiceInterface. Η πηγή (source) του είναι το ServiceInterface και ο στόχος (target) είναι το εκτεθειμένο Capability.
- *MessageType*: ο ορισμός πληροφορίας που ανταλλάσσεται μεταξύ παρόχων και καταναλωτών υπηρεσιών.
- *Milestone*: μια μέθοδος απεικόνισης προόδου συμπεριφορών προκειμένου να αναλυθεί το liveness. Χρησιμεύουν ιδιαίτερα σε συμπεριφορές που είναι μακράς ή και άπειρης διαρκείας.
- *Participant*: είναι ο τύπος του παρόχου ή/και του καταναλωτή υπηρεσιών. Στον επιχειρηματικό τομέα ένας Participant μπορεί να είναι άνθρωπος, οργανισμός ή σύστημα. Στον τομέα συστημάτων ένας Participant μπορεί να είναι σύστημα, εφαρμογή ή component.
- *Port*: επεκτείνει το UML *Port* με τρόπο που δείχνει αν ένα Connection είναι απαραίτητο σε αυτό το Port ή όχι.
- *Property*: Ενισχύει το UML *Property* με την ικανότητα να μπορούν να διακριθούν στιγμιότυπα του περιβάλλοντος *Classifier*, το οποίο είναι γνωστό και ως «πρωτεύον κλειδί».
- *Provider*: μοντελοποιεί τον τύπο του παρόχου υπηρεσίας σε μια σχέση παρόχου/καταναλωτή. Στη συνέχεια χρησιμοποιείται ως τύπος ενός ρόλου στο συμβόλαιο και ως τύπος μιας θύρας ενός συμμετέχοντα.
- *Request*: αντιπροσωπεύει ένα χαρακτηριστικό ενός *Participant*, που είναι η κατανάλωση μιας υπηρεσίας από ένα συμμετέχοντα που παρέχεται από άλλους, χρησιμοποιώντας καλά ορισμένες διεπαφές, όρους και προϋποθέσεις. Προσδιορίζει θύρες που είναι το συνδετικό σημείο μέσω του οποίου ένας *Participant* ικανοποιεί τις ανάγκες του μέσω της κατανάλωσης υπηρεσιών που παρέχονται από άλλους. Η θύρα αιτήματος είναι "συζυγής" θύρα, με την έννοια ότι οι διαθέσιμες και απαιτούμενες διεπαφές του τύπου της θύρας είναι ανεστραμμένες· αυτό δημιουργεί μια θύρα που χρησιμοποιεί έναν τύπο θύρας αντί να τον υλοποιεί.
- *ServiceChannel*: παρέχει το μονοπάτι επικοινωνίας μεταξύ των αιτημάτων καταναλωτών και τις υπηρεσίες παρόχων.

- *ServiceContract*: η επισημοποίηση της δεσμευτικής ανταλλαγής πληροφοριών, αγαθών, ή υποχρεώσεων μεταξύ συμμετεχόντων που καθορίζουν μια υπηρεσία.
- *ServiceInterface*: ορίζει τη διεπαφή και τον τρόπο με τον οποίο ένας συμμετέχοντας μπορεί να παρέχει ή να καταναλώσει μια υπηρεσία. Χρησιμοποιείται ως τύπος ενός Service ή Request Port. Μέρη του καθορίζονται από τα Interfaces που παρέχονται (Realization) και χρησιμοποιούνται (Usage) από το ServiceInterface και αντιπροσωπεύουν πιθανούς καταναλωτές και παρόχους των λειτουργιών που ορίζονται στις διεπαφές αυτές.
- *Service*: αντιπροσωπεύει ένα χαρακτηριστικό ενός *Participant*, που είναι η προσφορά μιας υπηρεσίας από ένα συμμετέχοντα σε άλλους χρησιμοποιώντας καλά ορισμένες διεπαφές, όρους και προϋποθέσεις. Προσδιορίζει μια θύρα που καθορίζει το συνδετικό σημείο μέσω του οποίου ένας *Participant* προσφέρει τις ικανότητές του και παρέχει μια υπηρεσία σε πελάτες.
- *ServicesArchitecture*: όψη υψηλού επιπέδου του SOA που περιγράφει τον τρόπο με τον οποίο οι συμμετέχοντες λειτουργούν μαζί, συγκροτώντας μια κοινότητα, για κάποιο σκοπό με την παροχή και χρήση υπηρεσιών.

Επίσης προσφέρεται και ένας γενικευμένος μηχανισμός για την κατηγοριοποίηση στοιχείων στη UML με κατηγορίες και τιμές κατηγοριών που περιγράφουν κάποια πληροφορία για τα στοιχεία. Αυτό προκύπτει από την ανάγκη οργάνωσης του μοντέλου έτσι ώστε να μπορεί να χρησιμοποιηθεί για πολλούς σκοπούς και να μπορεί να προβληθεί από τις προοπτικές διαφόρων ενδιαφερόμενων. Για αυτό ορίζονται οι παρακάτω κλάσεις:

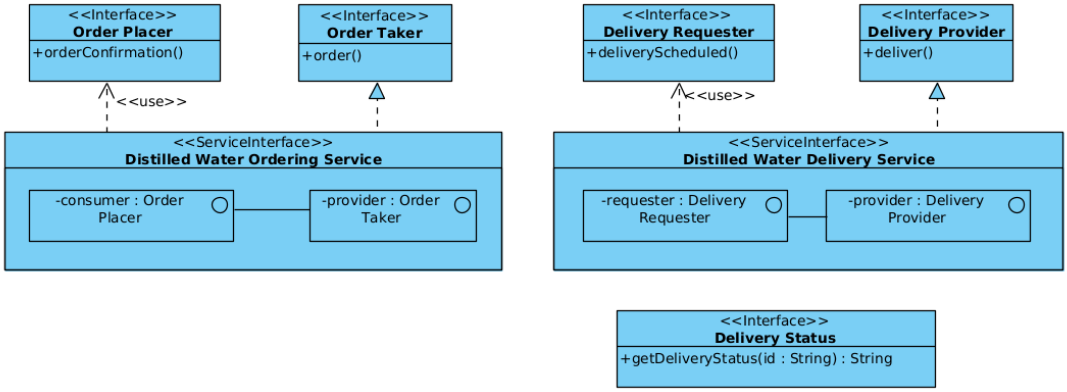
- *Catalog*: προσφέρει έναν τρόπο κατηγοριοποίησης και οργάνωσης των στοιχείων ανά κατηγορίες για οποιονδήποτε σκοπό.
- *Categorization*: χρησιμοποιείται για να κατηγοριοποιήσει ένα στοιχείο ανά *Category* ή *CategoryValue*.
- *Category*: μια κατηγοριοποίηση ή διαίρεση που χρησιμοποιείται για να χαρακτηρίσει τα στοιχεία ενός καταλόγου και να κατηγοριοποιήσει στοιχεία μοντέλων.
- *CategoryValue*: προσφέρει συγκεκριμένες τιμές ώστε ένα *Category* να κατηγοριοποιήσει περαιτέρω στοιχεία μοντέλου.

Το Visual Paradigm [2] είναι ένα UML CASE εργαλείο από το OMG, το οποίο προσφέρει τα SoaML διαγράμματα που θα επεξηγηθούν στα παρακάτω υποκεφάλαια. Τα διαγράμματα αυτά είναι τα εξής:

- Service Interface Diagram
- Service Participant Diagram
- Service Contract Diagrams
- Services Architecture Diagram
- Service Categorization Diagram

## 2.3 Service Interface Diagram

Το Service Interface Diagram είναι ένα διάγραμμα UML το οποίο επιτρέπει την τεκμηρίωση υπηρεσιών και των διεπαφών που χρειάζονται.



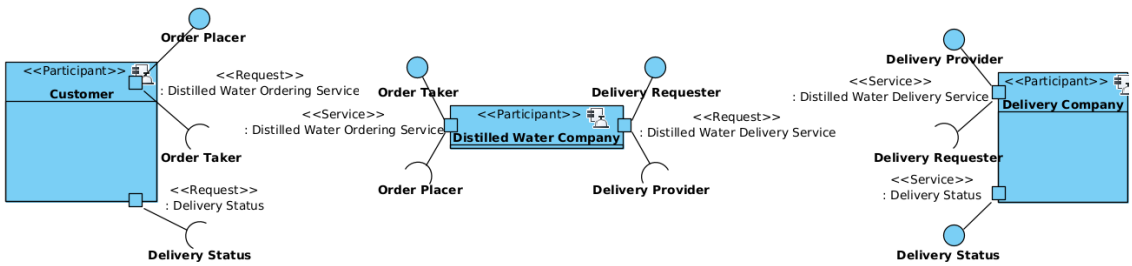

**Σχήμα 2.2** Service Interface Diagram [3]

Τα στοιχεία UML που προσφέρονται σε αυτό το διάγραμμα βάσει των SoaML στερεοτύπων είναι τα εξής: *ServiceInterface*, *Capability*, *Expose*, *MessageType*, *Milestone*, καθώς και ένα στοιχείο ακόμα:

- *Role*: αντιπροσωπεύει το ρόλο που θα έχουν οι συμμετέχοντες στην υπηρεσία.

## 2.4 Service Participant Diagram

Το Service Participant Diagram είναι ένα διάγραμμα UML το οποίο επιτρέπει την τεκμηρίωση των συμμετεχόντων και των ρόλων που έχουν στην αρχιτεκτονική υπηρεσιών, καθώς και τις υπηρεσίες που παρέχονται και καταναλώνονται από τους συμμετέχοντες.

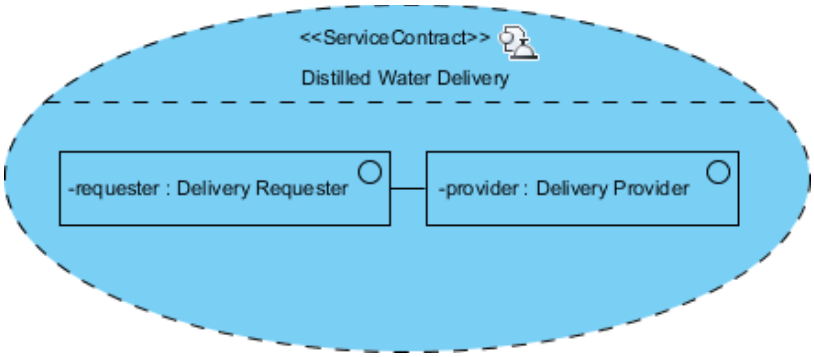



**Σχήμα 2.3** Service Participant Diagram [3]

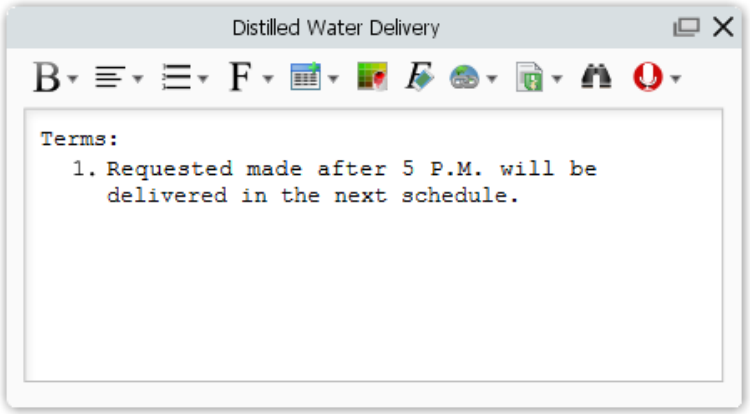
Τα στοιχεία UML που προσφέρονται σε αυτό το διάγραμμα βάσει των SoaML στερεοτύπων είναι τα εξής: *Participant*, *Agent*, *Property*, *Port*, *ServiceChannel*, *Capability*, *Service Port* (*Service*) και *Request Port* (*Request*).

## 2.5 Service Contract Diagram

Το Service Contract Diagram είναι ένα διάγραμμα UML το οποίο επιτρέπει την καταγραφή της συμφωνίας μεταξύ συμμετεχόντων και τον τρόπο με τον οποίο μια υπηρεσία πρέπει να παρέχεται και να καταναλώνεται. Ο όρος «συμφωνία» αναφέρεται σε διεπαφές, choreography και τυχόν όρους και προϋποθέσεις. Οι συμμετέχοντες πρέπει να συμφωνήσουν προκειμένου η υπηρεσία να εκτελεστεί.




**2.4** Service Contract Diagram [3]



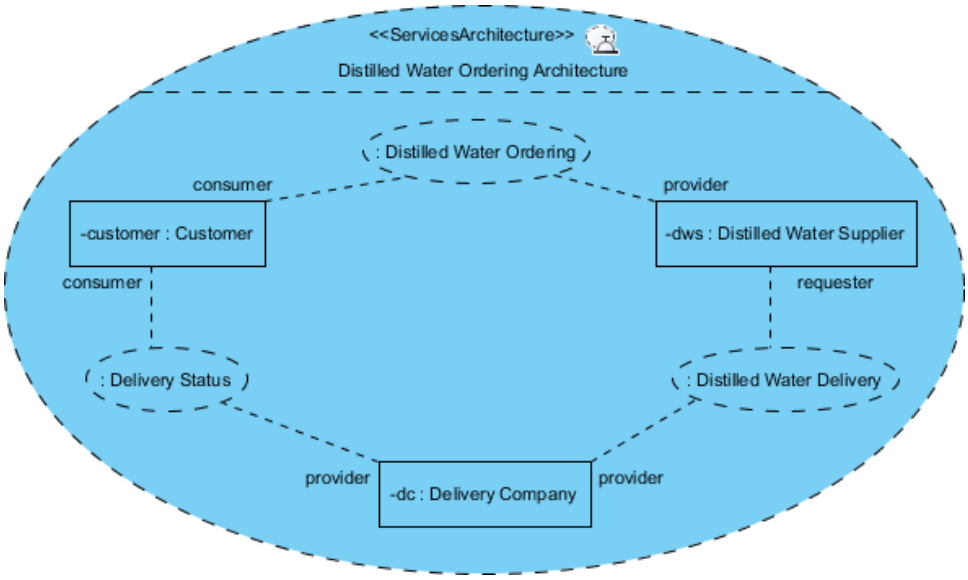

**Σχήμα 2.5** Service Contract Description [3]

Τα στοιχεία UML που προσφέρονται σε αυτό το διάγραμμα βάσει των SoaML στερεοτύπων είναι τα εξής: *ServiceContract*, *Provider*, *Consumer*, καθώς και ένα στοιχείο ακόμα:

- *Role*: αντιπροσωπεύει το ρόλο που θα έχουν οι συμμετέχοντες στην υπηρεσία.

## 2.6 Services Architecture Diagram

Το Services Architecture Diagram είναι ένα διάγραμμα UML το οποίο επιτρέπει την τεκμηρίωση των ρόλων των συμμετεχόντων που παρέχουν και καταναλώνουν υπηρεσίες για να πετύχουν κάποιον στόχο.



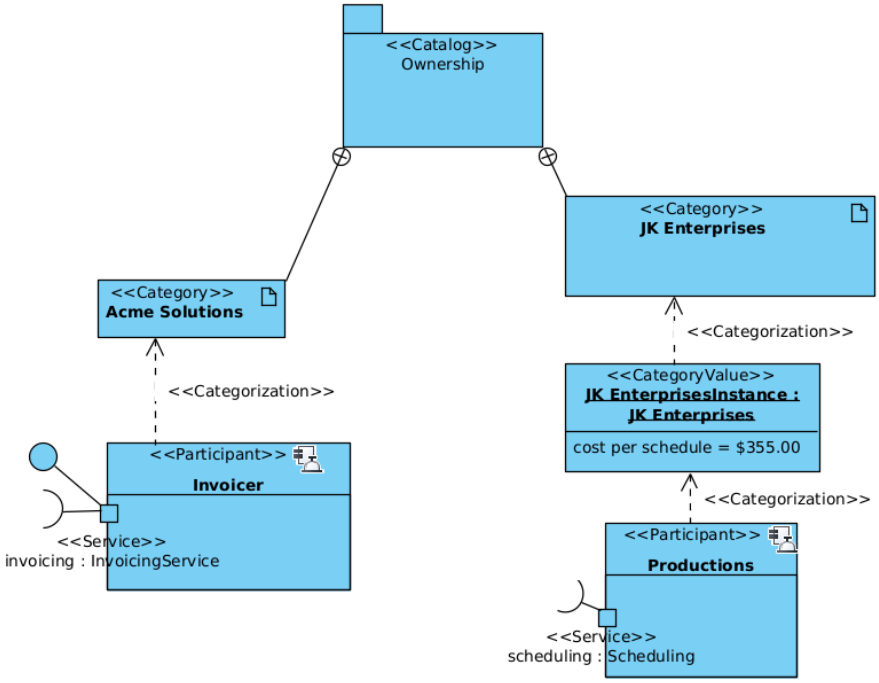

**Σχήμα 2.6** Services Architecture Diagram [3]

Τα στοιχεία UML που προσφέρονται σε αυτό το διάγραμμα βάσει των SoaML στερεοτύπων είναι τα εξής: *ServicesArchitecture, Service Contract Use* (*CollaborationUse*), *Role Binding* (*Collaboration*) καθώς και δύο στοιχεία ακόμα:

- *Internal Participant* (*Participant*): εσωτερικός ρόλος που παρέχει και καταναλώνει υπηρεσίες για να δημιουργήσει κάποια αξία στην αρχιτεκτονική υπηρεσιών.
- *External Participant* (*Participant*): εξωτερικός ρόλος που παρέχει και καταναλώνει υπηρεσίες για να δημιουργήσει κάποια αξία στην αρχιτεκτονική υπηρεσιών.

## 2.7 Service Categorization Diagram

Το Service Categorization Diagram είναι ένα διάγραμμα UML το οποίο επιτρέπει την οργάνωση των περιεχομένων του μοντέλου SOA έτσι ώστε τα στοιχεία του να μπορούν να χρησιμοποιηθούν για πολλαπλούς σκοπούς και να προβληθούν από διαφορετικές προοπτικές.




**Σχήμα 2.7** Service Categorization Diagram

Τα στοιχεία UML που προσφέρονται σε αυτό το διάγραμμα είναι τα εξής: *Catalog*, *Category*, *CategoryValue*, *Categorization*.

*Κεφάλαιο 3*

# Blockchain

Στο κεφάλαιο αυτό δε θα γίνει εκτενής παρουσίαση του blockchain, εφόσον δεν είναι ο σκοπός της εργασίας αυτής. Ακολουθούν σύντομες επεξηγήσεις για τις βασικές έννοιες, οι οποίες είναι απαραίτητες για την κατανόηση του.

## 3.1 Τι είναι blockchain

Το blockchain είναι μια κατανεμημένη βάση δεδομένων η οποία είναι διαμοιρασμένη ανάμεσα στους κόμβους ενός δικτύου υπολογιστών [4]. Η τεχνολογία αυτή είναι γνωστή κυρίως για τη χρήση της στα κρυπτονομίσματα για τη διατήρηση ενός ασφαλούς και κατανεμημένου καταλόγου συναλλαγών. Ο λόγος για την εξάπλωσή του είναι η εγγύηση της πιστότητας και της ασφάλειας δεδομένων χωρίς την ανάγκη εξάρτησης σε έναν έμπιστο τρίτο.

Η ασφάλεια που προσφέρει βασίζεται στην αμεταβλητότητα των υπαρχόντων πληροφοριών. Όπως φαίνεται και από το όνομά του, η μορφή ενός blockchain είναι μια αλυσίδα από blocks [5]. Ένα block είναι μια δομή δεδομένων η οποία περιέχει μεταξύ άλλων μια λίστα από συναλλαγές, το hash του προηγούμενου block και το hash του εαυτού του. Για να προστεθεί ένα νέο block στην αλυσίδα πρέπει να χρησιμοποιήσει το hash του τελευταίου block αυτής, το οποίο έχει βρεθεί μέσω μιας κρυπτογραφικής συνάρτησης. Η συνάρτηση αυτή χρησιμοποιεί ό,τι πληροφορία υπάρχει μέσα στο block, συμπεριλαμβανομένου και του hash του προηγούμενου block που είναι αποθηκευμένο σε αυτό. Επομένως, η προσπάθεια αλλαγής ενός block που υπάρχει ήδη στην αλυσίδα (δηλαδή η αλλαγή αποθηκευμένων πληροφοριών), σημαίνει πως θα αλλάξει το hash εκείνου του block, το οποίο είναι αποθηκευμένο και στο επόμενο block, το οποίο θα οδηγήσει στην αλλαγή του hash του επόμενου block, κ.ο.κ.

## 3.2 Αλγόριθμοι συναίνεσης

Οι αλγόριθμοι (ή μηχανισμοί) συναίνεσης είναι μεθοδολογίες που χρησιμοποιούνται για να επιτευχθεί συμφωνία, εμπιστοσύνη και ασφάλεια σε ένα κατανεμημένο δίκτυο υπολογιστών [6]. Όσον αφορά τεχνολογίες blockchain, οι πιο διαδεδομένοι αλγόριθμοι συναίνεσης είναι ο Proof of Work (PoW) [7] και ο Proof of Stake (PoS) [8].

Ο αλγόριθμος Proof of Work χρησιμοποιείται από κρυπτονομίσματα όπως το Bitcoin [9]. Με τη χρήση αυτού του αλγορίθμου, οι miners προσπαθούν να λύσουν ένα μαθηματικό γρίφο προκειμένου να προσθέσουν ένα νέο block στην αλυσίδα. Αναλυτικότερα, δημιουργείται το καινούριο block χρησιμοποιώντας το hash του προηγούμενου block και αποθηκεύει ένα συγκεκριμένο αριθμό συναλλαγών. Στη συνέχεια, οι miners προσπαθούν να λύσουν μια κρυπτογραφική συνάρτηση 'μαντεύοντας' έναν 32-bit αριθμό (nonce) που θα χρησιμοποιηθεί για το hash του τρέχοντος block. Μόλις βρεθεί ο κατάλληλος αριθμός, δημιουργείται το επόμενο block και αυτός που τον βρήκε επιβραβεύεται με ένα μικρό ποσό κρυπτονομίσματος. Εξαιτίας αυτών των υπολογισμών, ωστόσο, καταναλώνεται πολλή ενέργεια.

Ο αλγόριθμος Proof of Stake δημιουργήθηκε ως εναλλακτικός αλγόριθμος για τον Proof of Work, λόγω της χαμηλής κατανάλωσής του σε ενέργεια. Το μοντέλο συναίνεσης βασίζεται στην ποσότητα κρυπτονομισμάτων που διαθέτει το άτομο που επικυρώνει τις συναλλαγές. Εφόσον έχει ένα συγκεκριμένο αριθμό κρυπτονομισμάτων στο σύστημα (τα κάνει "stake") είναι δυνατόν να επιβεβαιώσει ένα block και να λάβει επιβράβευση.

## 3.3 Έξυπνα συμβόλαια

Ένα έξυπνο συμβόλαιο είναι ένα πρόγραμμα που εκτελείται αυτόματα για να τηρήσει τους όρους ενός συμβολαίου ή συμφωνίας [10]. Ο κώδικας και οι συμφωνίες που εμπεριέχονται σε αυτόν υπάρχουν σε ένα κατανεμημένο δίκτυο blockchain, επομένως είναι αμετάβλητα και μπορούν να εκτελεστούν χωρίς κάποια κεντρική αρχή, νομικό σύστημα ή εξωτερικό μηχανισμό επιβολής τους.

Το Ethereum [11] είναι το πρώτο blockchain που υλοποίησε έξυπνα συμβόλαια με τη μορφή που γνωρίζουμε σήμερα. Προσφέρει μια Turing-complete γλώσσα προγραμματισμού, τη Solidity [12], για τη δημιουργία συμβολαίων με μερικές γραμμές κώδικα.

## 3.4 Web 3.0

Τη δεκαετία του 1990 δημιουργήθηκε η πρώτη μορφή του παγκόσμιου ιστού όπως τον γνωρίζουμε σήμερα, το Web 1.0 και υποστήριζε μόνο την ανάγνωση σελίδων, κυρίως στατικών [13].

Στη συνέχεια, αναπτύχθηκε μια νέα μορφή διαδικτύου, το Web 2.0 και είναι η μορφή του διαδικτύου με την οποία είμαστε και πιο γνώριμοι. Σε αυτήν τη μορφή του, οι χρήστες μπορούν όχι μόνο να διαβάζουν περιεχόμενο, αλλά και να το παράγουν, να το χρησιμοποιούν και να αλληλεπιδρούν με άλλους χρήστες.

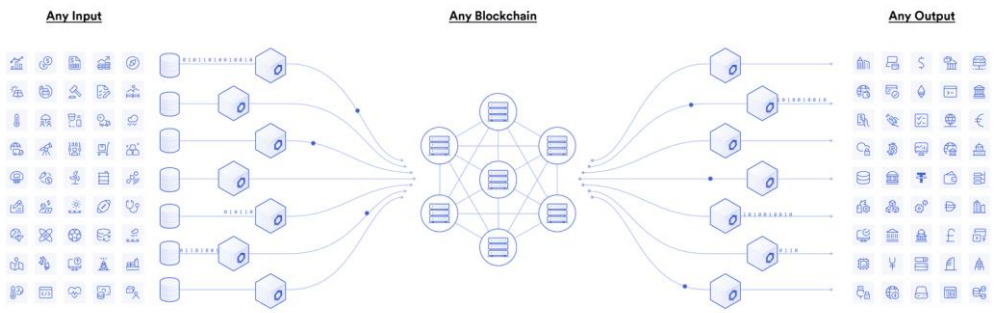
Χαρακτηρίζεται κυρίως από δυναμικό περιεχόμενο με το οποίο αποκρίνεται στο χρήστη και την πληροφορία που εισάγει στο σύστημα.

Πρόσφατα, έχει αναπτυχθεί ως ιδέα το Web 3.0, το οποίο βασίζεται στην ανάπτυξη της τεχνολογίας του blockchain και των έξυπνων συμβολαίων. Πέρα από το διάβασμα και την εγγραφή που προσφέρει ήδη το Web 2.0, το Web 3.0 δίνει επίσης την ικανότητα στους χρήστες να μπορούν να εκτελούν κώδικα μέσω αυτού.

Η ανάπτυξη του Web 3.0 έχει οδηγήσει στην ανάπτυξη πολλών αποκεντρωμένων ιδεών, όπως: Decentralized Autonomous Organizations (DAOs), Decentralized Finance (DeFi), ακόμα και Decentralized Applications (DApps), οι οποίες βασίζονται στην χρήση των έξυπνων συμβολαίων.

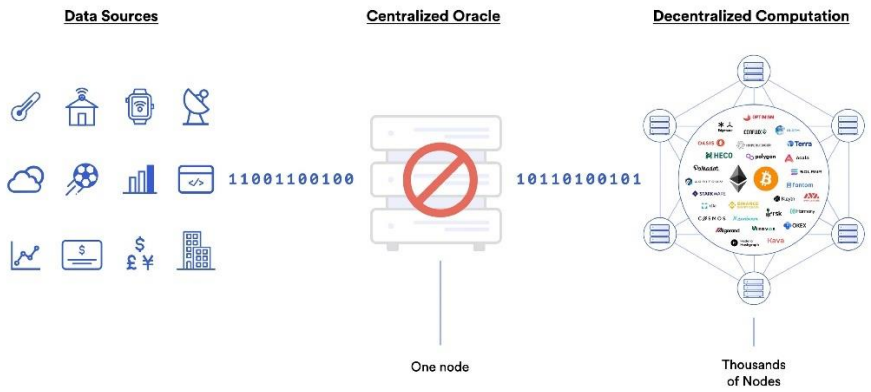
## 3.5 Oracles

Ένας oracle είναι μια γέφυρα μεταξύ ενός blockchain και του πραγματικού κόσμου [14]. Λειτουργεί ως API επάνω στην αλυσίδα, προκειμένου να μπορούν να γίνουν ερωτήματα (queries) στα έξυπνα συμβόλαια. Για παράδειγμα, DApps πρόβλεψης της αγοράς χρησιμοποιούν oracles για να διευθετήσουν πληρωμές βάσει γεγονότων. Μπορούν επίσης να είναι αμφίδρομα και να χρησιμοποιηθούν για την αποστολή δεδομένων στον πραγματικό κόσμο.




**Σχήμα 3.1** Blockchain oracles connect blockchains to inputs and outputs in the real world [15]

Ένας oracle συνήθως αποτελείται από ένα έξυπνο συμβόλαιο και μερικά component εκτός αλυσίδας που μπορούν να χρησιμοποιήσουν API, και στέλνουν περιοδικά συναλλαγές για να ενημερώσουν τα δεδομένα του έξυπνου συμβολαίου. Εφόσον χρησιμοποιούν πληροφορία που προσφέρονται στον πραγματικό κόσμο, είναι τόσο ασφαλή όσο οι πηγές πληροφοριών τους, για αυτό και είναι σημαντικό να χρησιμοποιούνται πολλαπλές πηγές, έτσι ώστε να μην υπάρξει πρόβλημα αν μια αποτύχει.




**Σχήμα 3.2** Centralized oracles are a single point of failure [15]

Υπάρχουν διάφοροι τύποι oracles [15]:

- Software Oracles: διαχειρίζονται δεδομένα που προέρχονται από online πηγές, όπως θερμοκρασία, τιμές αγαθών, καθυστερήσεις τρένων ή πτήσεων, κλπ. Ο software oracle εισάγει την πληροφορία στο έξυπνο συμβόλαιο.
- Hardware Oracles: μερικά έξυπνα συμβόλαια χρειάζονται πληροφορίες απευθείας από τον πραγματικό κόσμο, για παράδειγμα RFID sensors οι οποίοι στέλνουν πληροφορία στο έξυπνο συμβόλαιο.
- Inbound Oracles: παρέχει δεδομένα από τον έξω κόσμο.
- Outbound Oracles: παρέχει στα έξυπνα συμβόλαια την ικανότητα να στείλουν δεδομένα στον έξω κόσμο, όπως για παράδειγμα μια έξυπνη κλειδαριά που λαμβάνει πληρωμή σε μια διεύθυνση blockchain και ξεκλειδώνει αυτόματα.
- Consensus-based Oracles: λαμβάνουν πληροφορία από συναίνεση ανθρώπων και αγορά προβλέψεων (prediction market). Μια πηγή πληροφοριών μπορεί να είναι ριψοκίνδυνη, επομένως για την αποφυγή χειραγώγησης της αγοράς οι αγορές προβλέψεων έχουν συστήματα αξιολόγησης για τα oracles. Για παραπάνω ασφάλεια, χρησιμοποιούνται πολλαπλοί oracles.

*Κεφάλαιο 4*

# Περίπτωση χρήσης, αρχιτεκτονική και τεχνολογίες

## 4.1 Ανάλυση περίπτωσης χρήσης

Η περίπτωση χρήσης βασίζεται στην εργασία του μαθήματος "Τεχνολογίες Υπηρεσιών Λογισμικού" της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου το ακαδημαϊκό έτος 2020-2021. Η εκφώνησή της είναι η εξής:

«Η εργασία αυτή περιλαμβάνει την πλήρη ανάπτυξη μιας μικρής διαδικτυακής εφαρμογής λογισμικού [...]. Η εστίαση βρίσκεται στην αρχιτεκτονική, τις τεχνολογίες και τα εργαλεία ανάπτυξης εφαρμογών που μπορούν να παρέχονται ως SaaS, καθώς και στην εφαρμογή ευέλικτων (agile) μεθοδολογιών ανάπτυξης με τα αντίστοιχα σύγχρονα εργαλεία.

Το "AskMeAnything" είναι μια εφαρμογή ερωτήσεων – απαντήσεων η οποία θα προσφέρεται ως υπηρεσία λογισμικού σε υποθετικούς πελάτες όπως εταιρείες που προσφέρουν υπηρεσίες τύπου help desk, εταιρείες που περιγράφουν τις εσωτερικές τους διαδικασίες με μορφή ερωτήσεων – απαντήσεων, εκπαίδευση κ.ά. Στο πλαίσιο της εργασίας θα αναπτυχθεί μια απλουστευμένη εκδοχή της εφαρμογής η οποία εκτελεί τις ακόλουθες λειτουργίες:

- Δημιουργία ερώτησης
- Χαρακτηρισμός με λέξεις-κλειδιά
- Απάντηση ερώτησης
- Περιήγηση σε ερωταπαντήσεις
- Στατιστικά ερωτήσεων με λέξεις-κλειδιά
- Στατιστικά ερωτήσεων με ημερομηνίες
- Sign Up, Sign In

Οι διδάσκοντες, σε λόγο υποθετικού "πελάτη", διαθέτουν την αρχική τεκμηρίωση των λειτουργιών αυτών, καθώς και μια πρώτη περιγραφή του τρόπου που ο "πελάτης" φαντάζεται την αλληλεπίδραση των χρηστών με την εφαρμογή [...]»

Από τις λειτουργίες της εφαρμογής που περιγράφονται στην παραπάνω εκφώνηση, υλοποιήθηκαν όλες πλην της τελευταίας (Sign Up, Sign In). Αρχικά, αναπτύσσεται μια εφαρμογή SOA από έναν stakeholder, η οποία να υποστηρίζει τις παρακάτω λειτουργίες:

- Δημιουργία ερώτησης
- Χαρακτηρισμός με λέξεις-κλειδιά

- Απάντηση ερώτησης
- Περιήγηση σε ερωταπαντήσεις

Η εφαρμογή αυτή τεκμηριώνεται με χρήση SoaML. Στη συνέχεια, ένας δεύτερος stakeholder χρησιμοποιεί την τεκμηρίωση που έχει προηγηθεί για να δημιουργήσει μια εφαρμογή που χρησιμοποιεί blockchain και να επεκτείνει την υπάρχουσα εφαρμογή υλοποιώντας τις παρακάτω λειτουργίες:

- Στατιστικά ερωτήσεων με λέξεις-κλειδιά
- Στατιστικά ερωτήσεων με ημερομηνίες
- Στατιστικά απαντήσεων με ημερομηνίες

Εδώ πρέπει να αναφερθεί πως η δεύτερη εφαρμογή που θα χρησιμοποιεί blockchain θα μπορούσε να έχει υλοποιηθεί ως μια απλή εφαρμογή ιστού, όπως τις υπόλοιπες. Ο λόγος που χρησιμοποιείται blockchain είναι για να αναδειχθεί η τεκμηρίωση με SoaML και η ενοποίηση της με το υπάρχον σύστημα που δε χρησιμοποιεί blockchain.

## 4.2 Αρχιτεκτονική και τεχνολογίες

Οι εφαρμογές δε θα τρέχουν τοπικά, αλλά θα είναι deployed. Για το deployment των εφαρμογών θα χρησιμοποιηθεί το Heroku [16]. Εφόσον ο στόχος είναι η ανάπτυξη μιας εφαρμογής SOA, θα γίνει χρήση microservices. Συγκεκριμένα, για την εφαρμογή ιστού θα δημιουργηθούν τα παρακάτω:

- Ask Questions service
- Answers service
- Browse Questions service
- Frontend

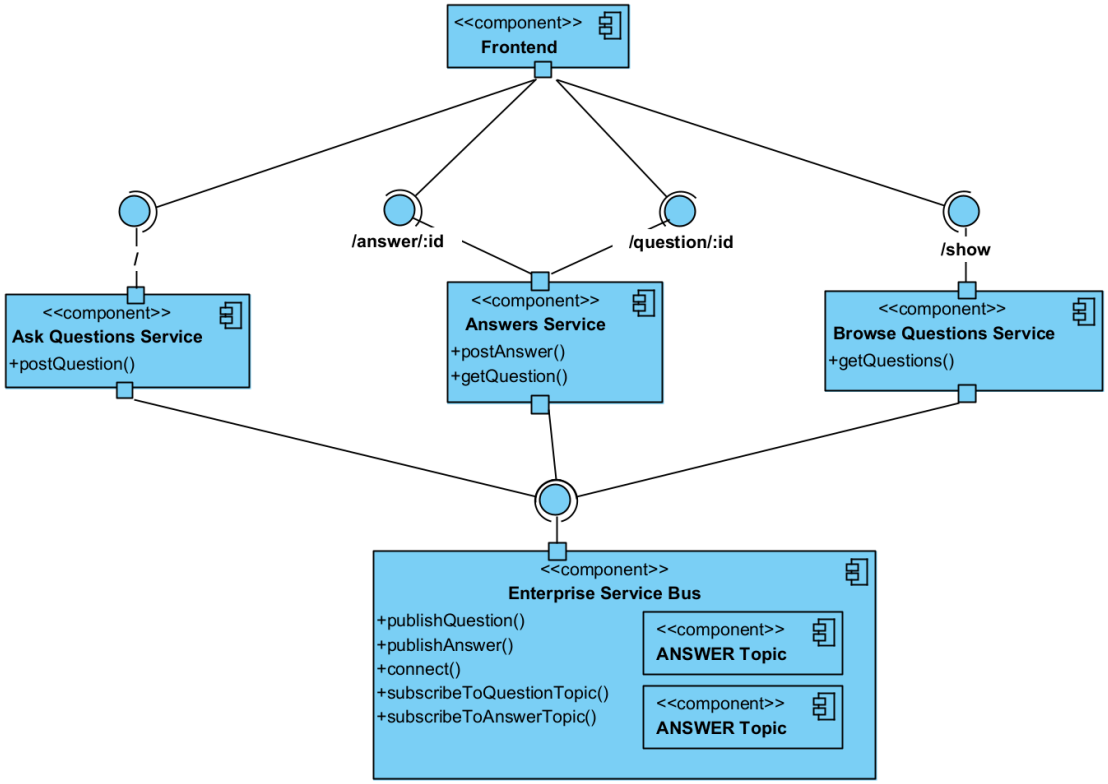
Για την επικοινωνία των microservices θα δημιουργηθεί ένα enterprise service bus που θα ακολουθεί το πρότυπο publish/subscribe. Αυτό θα τρέχει στον okeanos [17], εφόσον την ώρα της συγγραφής υπάρχουν διαθέσιμοι πόροι στη διεύθυνση *83.212.78.171*.

Η υπηρεσία **Ask Questions** θα κάνει publish μηνύματα τύπου “Question” στο service bus, εφόσον αυτά τα μηνύματα έχουν τίτλο και κείμενο.

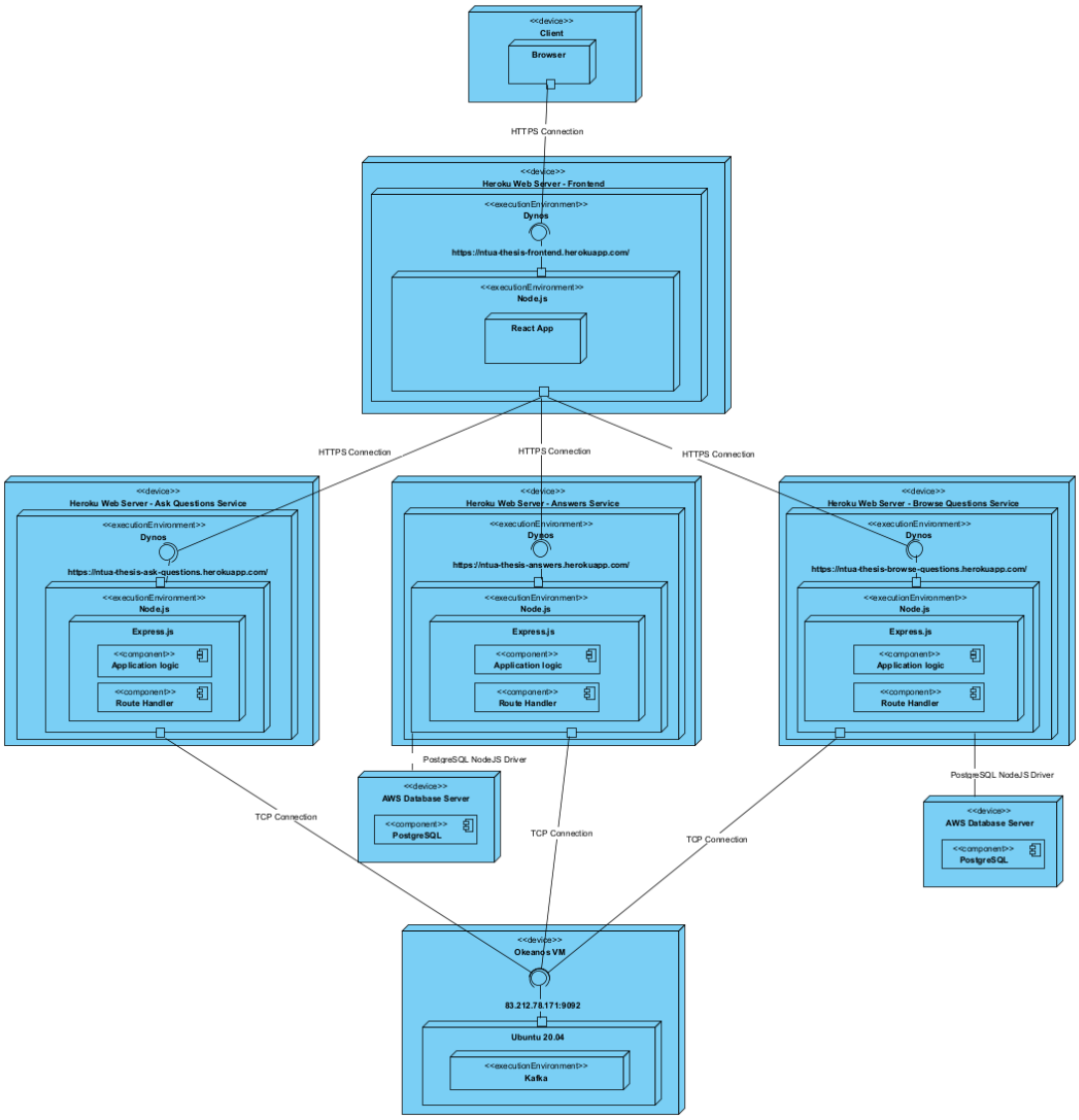
Η υπηρεσία **Answers** θα κάνει subscribe για να μπορεί να δέχεται μηνύματα τύπου “Question” από το service bus και να τα αποθηκεύει. Επίσης, θα αποθηκεύει και θα κάνει publish μηνύματα τύπου “Answer” στο service bus. Τέλος, θα μπορεί να επιστρέφει τα περιεχόμενα μιας συγκεκριμένης ερώτησης συμπεριλαμβανομένων και των απαντήσεων, αν υπάρχουν.

Η υπηρεσία **Browse Questions** θα κάνει subscribe για να μπορεί να δέχεται μηνύματα τύπου “Question” και “Answer” από το service bus και να τα αποθηκεύει. Στη συνέχεια θα μπορεί να επιστρέφει πληροφορίες για πολλές ερωτήσεις, καθώς και τον αριθμό των απαντήσεων που έχει η κάθε μια από αυτές, έτσι ώστε να επιτρέπεται η περιήγηση σε αυτές.

Το **Frontend** θα προσφέρει ένα UI στο χρήστη, προκειμένου να μπορεί να αλληλεπιδράσει με όλες αυτές τις υπηρεσίες και να ωφεληθεί από τις λειτουργίες τους.




**Σχήμα 4.1** Microservices Component Diagram




**Σχήμα 4.2** Microservices Deployment Diagram

Όσον αφορά την εφαρμογή που χρησιμοποιεί blockchain, θα χρησιμοποιηθεί το Truffle Suite [18]. Συγκεκριμένα, θα χρησιμοποιηθεί το Truffle [19] για την ανάπτυξη και το testing του έξυπνου συμβολαίου και το Ganache [20] για το τοπικό deployment. Μόλις πετύχει το τοπικό deployment, μέσω του Infura [21] θα γίνει deploy στο Ropsten [22], ένα Ethereum test network. Με το deployment στο Ropsten, δε χρειάζονται πραγματικά χρήματα και το έξυπνο συμβόλαιο θα χρησιμοποιείται σε ένα πραγματικό blockchain.

Για την αλληλεπίδραση με το έξυπνο συμβόλαιο στο Ropsten και την ανάπτυξη της εφαρμογής θα δημιουργηθούν τα παρακάτω:

- Blockchain service
- Graphs service

Η υπηρεσία **Blockchain** θα αποτελεί ένα είδος oracle, εφόσον ο σκοπός του είναι η προσθήκη πληροφορίας στο blockchain μέσω του έξυπνου συμβολαίου. Συγκεκριμένα, θα κάνει subscribe στο service bus για να μπορεί να δέχεται

μηνύματα τύπου “Question” και “Answer” και μέσω κλήσης μιας συνάρτησης στο έξυπνο συμβόλαιο θα τα αποθηκεύει στο blockchain.

Η υπηρεσία **Graphs** θα προσφέρει ένα UI στο χρήστη, προκειμένου να μπορεί να δει γραφήματα που περιέχουν στατιστικά για ερωτήσεις και απαντήσεις. Τα γραφήματα που θα παρέχονται θα είναι τρία:

- Οι 5 περισσότερο χρησιμοποιημένες λέξεις-κλειδιά των τελευταίων N ημερών, εφόσον υπάρχουν
- Αριθμός ερωτήσεων ανά ημέρα για τις τελευταίες N ημέρες
- Αριθμός απαντήσεων ανά ημέρα για τις τελευταίες N ημέρες

Όπου $N = \{5, 7, 9\}$.

Τα παραπάνω θα είναι όλα Node [23] projects, επομένως θα χρησιμοποιηθεί ο Node Package Manager [24]. Τα frontend θα αναπτυχθούν με ReactJS [25] και οι εφαρμογές ιστού θα χρησιμοποιούν PostgreSQL [26] για βάση δεδομένων. Για το enterprise service bus θα χρησιμοποιηθεί το Apache Kafka [27]. Τα πακέτα που θα χρησιμοποιηθούν είναι:

- cors [28]
- express [29]
- kafkajs [30]
- nodemon [31]
- pg [32]
- sequelize [33]

Ο κώδικας και τα διαγράμματα που χρησιμοποιήθηκαν είναι διαθέσιμα στο repository του οργανισμού [34].

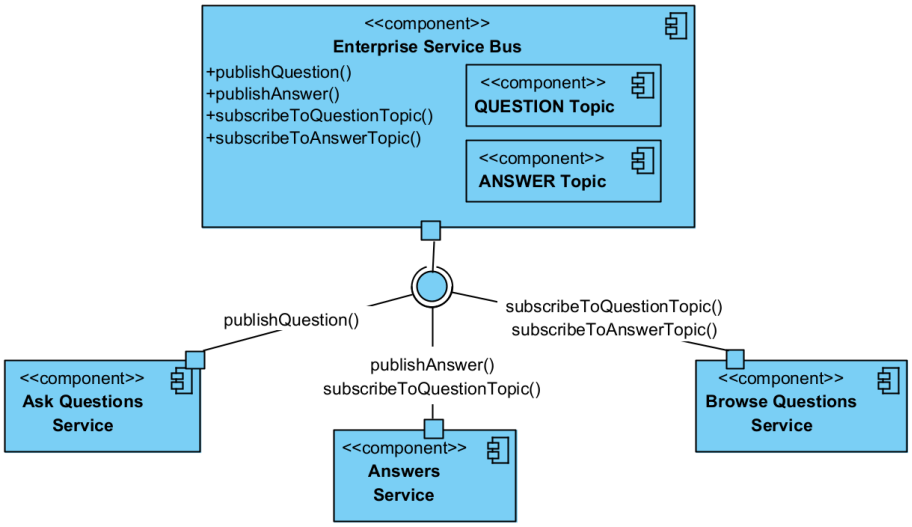
*Κεφάλαιο 5*

# Ανάπτυξη εφαρμογής SOA και τεκμηρίωση με SoaML

## 5.1 Ανάπτυξη εφαρμογής SOA

### 5.1.1 Enterprise Service Bus

Όπως αναφέρθηκε παραπάνω, για την υλοποίηση του enterprise service bus θα χρησιμοποιηθεί το Apache Kafka. Εφόσον θα γίνει χρήση του προτύπου publish/subscribe, αυτό σημαίνει πως το Kafka χρησιμοποιείται ως message broker.




**Σχήμα 5.1** Enterprise Service Bus Component Diagram

Για να επιτευχθεί η ανάγνωση, εγγραφή, αποθήκευση και επεξεργασία μηνυμάτων, αυτά οργανώνονται και αποθηκεύονται σε *topics*. Ένα topic είναι συγκρίσιμο με ένα φάκελο και τα μηνύματα με αρχεία που βρίσκονται μέσα στον φάκελο αυτό. Για την παρούσα περίπτωση χρήσης χρειάζονται δύο topics, ένα για τα μηνύματα τύπου "Question" και ένα για τα μηνύματα τύπου "Answer".

Συνεπώς, θα δημιουργηθούν δύο topics: το "*QUESTION*" και το "*ANSWER*". Για την εγγραφή μηνυμάτων, στο *QUESTION* θα κάνει publish η υπηρεσία Ask Questions και στο *ANSWER* θα κάνει publish η υπηρεσία Answers. Για την ανάγνωση μηνυμάτων, στο *QUESTION* θα κάνουν subscribe οι υπηρεσίες Answers και Browse Questions και στο *ANSWER* θα κάνει subscribe η υπηρεσία Browse Questions.

Για τη δημιουργία των topics χρησιμοποιούνται οι παρακάτω εντολές:

```
kafka-topics.sh --create --topic QUESTION --bootstrap-server
localhost:9092 --replication-factor 1

kafka-topics.sh --create --topic ANSWER --bootstrap-server
localhost:9092 --replication-factor 1
```

Εφόσον το Kafka είναι μια κατανεμημένη πλατφόρμα, επιτρέπεται η ανάγνωση ενός topic από πολλαπλούς subscribers. Αυτό επιτυγχάνεται με *consumer groups*, τα οποία είναι ουσιαστικά ένα σετ από καταναλωτές που συνεργάζονται για την ανάγνωση μηνυμάτων. Για την παρούσα περίπτωση χρήσης δεν είναι επιθυμητή η ανάγνωση ενός topic από πολλούς καταναλωτές κατανεμημένα, αλλά από κάθε υπηρεσία ξεχωριστά.

Επομένως, για κάθε topic θα δημιουργηθούν τόσα consumer groups όσες είναι και οι υπηρεσίες που θα κάνουν subscribe σε αυτό. Συγκεκριμένα, κάθε υπηρεσία θα κάνει subscribe στα topics που χρειάζεται χρησιμοποιώντας το όνομά της· η υπηρεσία Answers θα κάνει subscribe στο *QUESTION* ως μέρος του consumer group "*answers-group*" και η υπηρεσία Browse Questions θα κάνει subscribe στο *QUESTION* και στο *ANSWER* ως μέρος του consumer group "*browse-questions-group*". Η δημιουργία των consumer groups δε χρειάζεται να γίνει εκ των προτέρων, εφόσον κατά τη σύνδεση εάν δεν υπάρχει ένα consumer group, το Kafka το δημιουργεί.

Τέλος, για να είναι δυνατή η ανάγνωση όλων των μηνυμάτων που έχουν προηγηθεί, στο φάκελο *server.properties* θα αλλάξουμε την τιμή της παρακάτω τιμής ως εξής:

```
log.retention.hours=-1
```

Αυτό εγγυάται πως τα μηνύματα θα διατηρηθούν για πάντα και επομένως θα είναι διαθέσιμα για ανάγνωση από οποιαδήποτε άλλη υπηρεσία εισαχθεί στο σύστημα μετέπειτα.

### 5.1.2 Ask Questions Service

Η υπηρεσία Ask Questions έχει ως κύρια λειτουργία την αποστολή μηνυμάτων τύπου "Question" στο enterprise service bus. Συγκεκριμένα, θα δημιουργηθεί ένα endpoint που να δέχεται ένα μήνυμα σε μορφή json με χρήση του HTTPS, το οποίο θα περιέχει την πληροφορία που χρειάζεται για τον ορισμό μιας ερώτησης και θα γίνει publish στο service bus.

Επομένως γίνεται αρχικοποίηση ενός Node project με όνομα *ask-questions*, εγκαθίστανται τα απαραίτητα πακέτα και δημιουργούνται τα αρχεία που αποτελούν την υπηρεσία. Στη συνέχεια, ορίζεται η λογική του endpoint που θα δέχεται τα μηνύματα json και θα τα κάνει publish στο service bus. Παρακάτω βρίσκεται το snippet κώδικα που κάνει το publish:

```
let messageKey = 0;

const { Kafka } = require('kafkajs');

const kafka = new Kafka({
  clientId: "ask-questions",
```

```
  brokers: ['83.212.78.171:9092'],
})

const producer = kafka.producer();

const run = async() => {
  await producer.connect();
  await producer.send({
    topic: "QUESTION",
    messages: [{
      key: messageKey.toString(),
      value: JSON.stringify(req.body)
    }]
  })
  messageKey++;
}

run().catch(e => console.error(`[kafka-producer] ${e.message}`, e));
```

Ορίζονται το topic στο οποίο θα γίνει publish το μήνυμα, δηλαδή το topic *QUESTION*, και το μήνυμα που θα γίνει publish στο service bus, το οποίο περιέχει δύο πεδία, το πεδίο *key* και το πεδίο *value*. Το πεδίο *key* περιέχει έναν ακέραιο που έχει μετατραπεί σε string και περιγράφει τον αύξοντα αριθμό του μηνύματος που θα σταλεί. Το πεδίο value περιέχει ένα json string που περιγράφει το μήνυμα τύπου "Question" και έχει τα εξής περιεχόμενα:

```
{ qname: string,
  qtext: string,
  qkeywords: string
}
```

Όπου *qname* ένα string που περιέχει τον τίτλο της ερώτησης, *qtext* ένα string που περιέχει το κείμενο που επεξηγεί την ερώτηση και *qkeywords* ένα comma separated string που περιέχει τις λέξεις-κλειδιά, αν υπάρχουν.

Ο server δέχεται μηνύματα με POST request στο endpoint "/" με χρήση HTTPS και εφόσον το body του μηνύματος περιέχει τα πεδία *qname* και *qtext* το μήνυμα γίνεται publish στο service bus.

### 5.1.3 Answers Service

Η υπηρεσία Answers έχει ως κύρια λειτουργία την αποστολή μηνυμάτων τύπου "Answer" στο enterprise service bus. Συγκεκριμένα, θα δημιουργηθεί ένα endpoint που να δέχεται ένα μήνυμα σε μορφή json με χρήση του HTTPS, το οποίο θα περιέχει την πληροφορία που χρειάζεται για τον ορισμό μιας απάντησης και θα γίνει publish στο service bus, αφού την αποθηκεύσει στη βάση δεδομένων της υπηρεσίας.

Προκειμένου να είναι γνωστή η ερώτηση στην οποία δημιουργείται η απάντηση είναι απαραίτητο να υπάρχουν και οι διαθέσιμες ερωτήσεις, επομένως η υπηρεσία κάνει subscribe στο topic *QUESTION* και αποθηκεύει τις ερωτήσεις

στη βάση δεδομένων της υπηρεσίας. Ο λόγος που αποθηκεύονται οι ερωτήσεις, αλλά και οι απαντήσεις, είναι επειδή η υπηρεσία μπορεί να επιστρέφει τα περιεχόμενα μιας συγκεκριμένης ερώτησης, συμπεριλαμβανομένων και των απαντήσεών της.

Γίνεται αρχικοποίηση ενός Node project με όνομα *answers*, εγκαθίστανται τα απαραίτητα πακέτα και δημιουργούνται τα αρχεία που αποτελούν την υπηρεσία. Στη συνέχεια, ορίζεται η λογική του endpoint που θα δέχεται τα μηνύματα json και θα τα κάνει publish στο service bus αφού τα αποθηκεύσει. Ο κώδικας που κάνει publish στο service bus είναι αντίστοιχος με τον κώδικα της υπηρεσίας Ask Questions, ωστόσο έχει της εξής διαφορές: το topic είναι το *ANSWER* και το πεδίο *value* έχει τα εξής περιεχόμενα:

```
{ answer: string,
  questionId: int,
  answerId: int
}
```

Όπου *answer* ένα string το οποίο περιέχει το κείμενο που αποτελεί την απάντηση, *questionId* ένας ακέραιος που αντιπροσωπεύει τον αύξων αριθμό της ερώτησης, όπως αυτή έχει αποθηκευτεί στη βάση δεδομένων και *answerId* ένας ακέραιος που αντιπροσωπεύει τον αύξων αριθμό της απάντησης, όπως αυτή έχει αποθηκευτεί στη βάση δεδομένων.

Παρατίθεται ένα snippet κώδικα που χρησιμοποιείται για την εγγραφή στο topic *QUESTION* ως μέρος του "*answers-group*" και την αποθήκευση των ερωτήσεων στη βάση δεδομένων:

```
const subscribe = () => {
  const { Kafka } = require("kafkajs")

  const kafka = new Kafka({
      clientId: "consumer",
      brokers: ['83.212.78.171:9092'],
  })

  const consumer = kafka.consumer({ groupId: "answers-group" })

  const run = async() => {
    await consumer.connect()
    await consumer.subscribe({ topic: "QUESTION", fromBeginning: true })
    await consumer.run({
      eachMessage: async({ topic, partition, message }) => {
        let jsonMessage = JSON.parse(message.value);
        const now = new Date(parseInt(message.timestamp));
        now.setTime(now.getTime());
        if (topic == "QUESTION") {
          models.Questions.create({
          title: jsonMessage.qname,
          text: jsonMessage.qtext,
          keywords: [jsonMessage.qkeywords.replace(/\s/g, "").split(",")],
          dateCreated: now,
          answers: null
          })
        }
      }
```

```
    })
  }

  run().catch(e => console.error(`[kafka-consumer] ${e.message}`, e))
}

subscribe();
```

Υλοποιείται η αρχικοποίηση της βάσης δεδομένων με το κατάλληλο schema και η σύνδεση σε αυτή και ο server παρέχει τις παρακάτω λειτουργίες, με χρήση του HTTPS:

- Στο endpoint "`/question/:id`", όπου *:id* ένας ακέραιος αριθμός, με χρήση GET request επιστρέφεται ένα json μήνυμα που περιέχει τις πληροφορίες μια ερώτησης και όλες τις απαντήσεις που αντιστοιχούν στην ερώτηση αυτή.
- Στο endpoint "`/answer/:id`", όπου *:id* ένας ακέραιος αριθμός, με χρήση POST request δέχεται μηνύματα που στο body τους περιέχουν μια απάντηση, τα αποθηκεύει στη βάση δεδομένων και τα κάνει publish στο service bus.

## 5.1.4 Browse Questions Service

Η υπηρεσία Browse Questions έχει ως κύρια λειτουργία την επιστροφή πληροφοριών για πολλαπλές απαντήσεις, καθώς και τον αριθμό των απαντήσεων που έχει η κάθε μια από αυτές, έτσι ώστε να επιτρέπεται η περιήγηση σε αυτές.

Προκειμένου να αποθηκευτούν οι ερωτήσεις και οι απαντήσεις, η υπηρεσία κάνει subscribe στα topics *QUESTION* και *ANSWER* και αποθηκεύει την απαραίτητη πληροφορία στη βάση δεδομένων της.

Γίνεται αρχικοποίηση ενός Node project με όνομα *answers*, εγκαθίστανται τα απαραίτητα πακέτα και δημιουργούνται τα αρχεία που αποτελούν την υπηρεσία.

Όπως πριν, γίνεται χρήση του κώδικα που εκτελείται από έναν Kafka consumer, αλλά τώρα θα προστεθεί και η παρακάτω γραμμή:

```
    await consumer.subscribe({ topic: "ANSWER", fromBeginning: true })
```

Με αυτόν τον τρόπο, η υπηρεσία καταναλώνει και τα δύο είδη μηνυμάτων από το Kafka ως μέλος του "*browse-questions-group*" και τα αποθηκεύει στη βάση δεδομένων του.

Υλοποιείται η αρχικοποίηση της βάσης δεδομένων με το κατάλληλο schema και η σύνδεση σε αυτή και ο server παρέχει τις παρακάτω λειτουργίες, με χρήση του HTTPS:

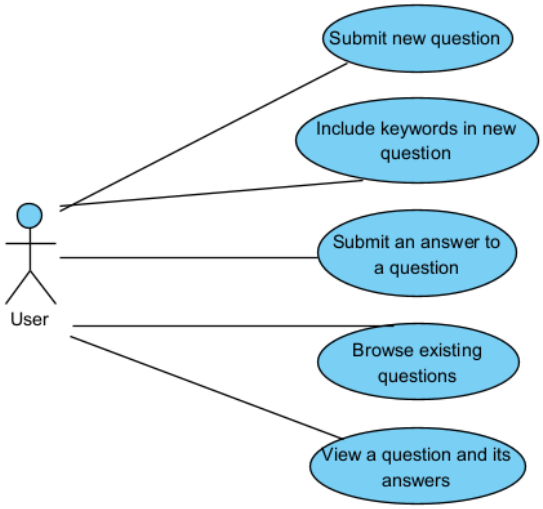
- Στο endpoint "`/show`", με χρήση POST request δέχεται μηνύματα που στο body τους περιέχουν έναν ακέραιο αριθμό που αντιπροσωπεύει τον αριθμό της σελίδας περιήγησης και με query στη βάση δεδομένων επιστρέφει έως και 10 ερωτήσεις που βρίσκονται στη σελίδα αυτή, εφόσον υπάρχουν

### 5.1.5 Frontend

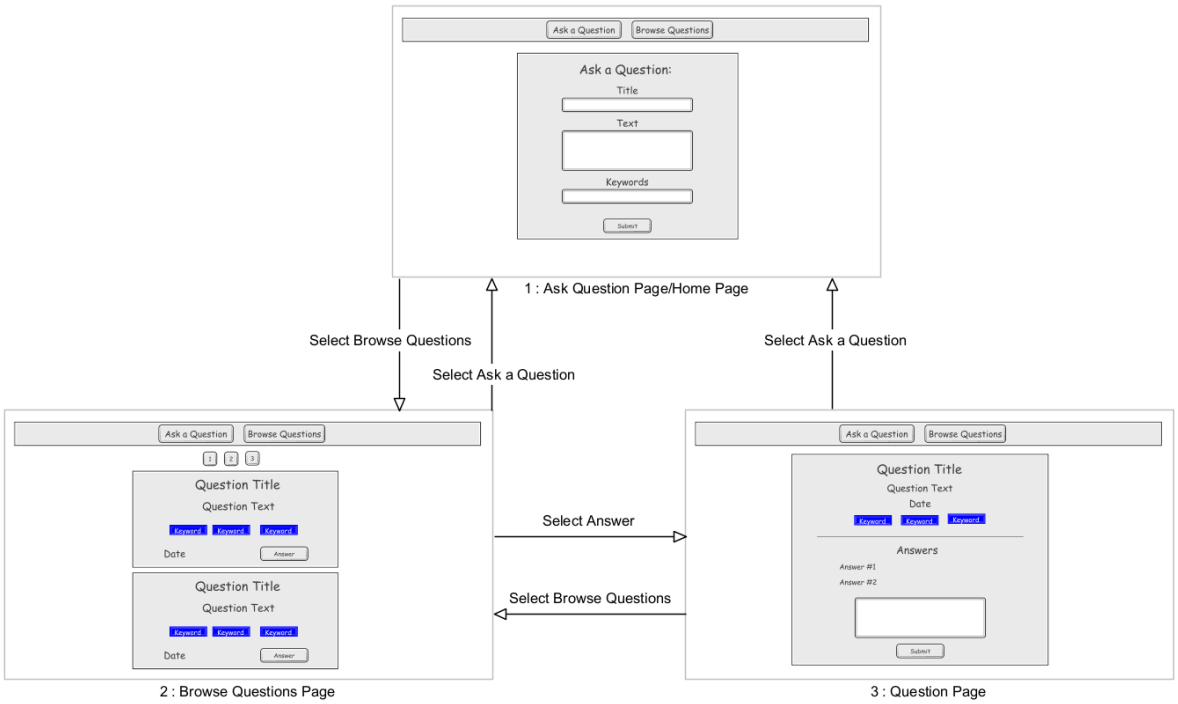
Όπως εξηγήθηκε προηγουμένως, το **Frontend** θα προσφέρει ένα UI στο χρήστη έτσι ώστε να αλληλεπιδρά με τις υπηρεσίες που έχουν οριστεί και να προσφέρει τις παρακάτω λειτουργίες στο χρήστη:

- Δημιουργία ερώτησης
- Χαρακτηρισμός με λέξεις-κλειδιά
- Απάντηση ερώτησης
- Περιήγηση σε ερωταπαντήσεις

Προκειμένου να υλοποιηθεί η λειτουργία 'Απάντηση ερώτησης', ο χρήστης πρέπει να μπορεί να δει μια ερώτηση και τις απαντήσεις που έχει, οπότε θα υλοποιηθεί η αντίστοιχη λειτουργία.




**Σχήμα 5.2** Frontend Use Case Diagram



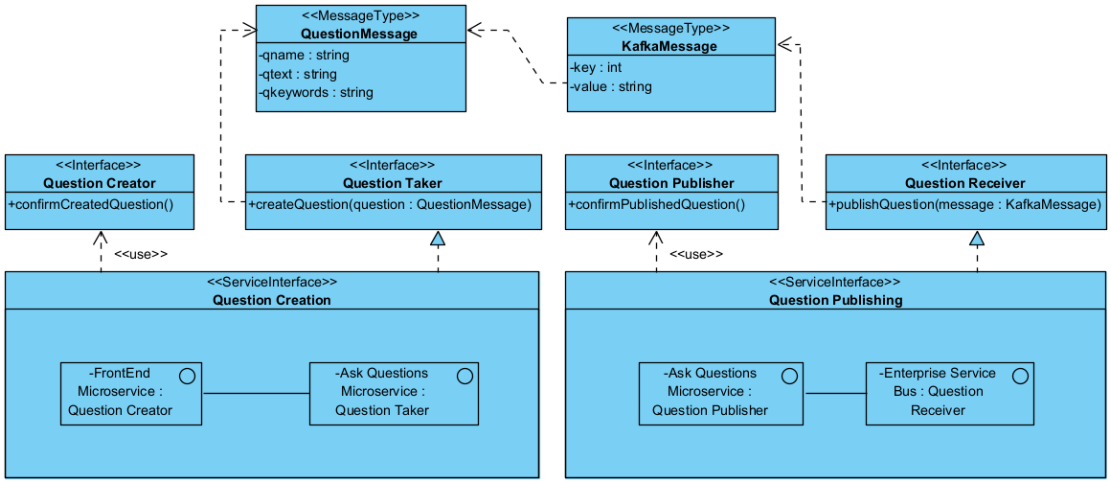

**Σχήμα 5.3** Frontend Wireflow Diagram

# 5.2 Τεκμηρίωση με SoaML

## 5.2.1 Service Interface Diagrams

Αρχικά, θα δημιουργηθούν τα Service Interface Diagrams, προκειμένου να τεκμηριωθούν οι υπηρεσίες που προσφέρονται και καταναλώνονται από κάθε συμμετέχοντα στο σύστημα:

Για την υπηρεσία **Ask Questions** δημιουργείται το παρακάτω διάγραμμα:




**Σχήμα 5.4** Ask Questions Service Interface Diagram

Έχουν δημιουργηθεί τα εξής ServiceInterfaces:

- Question Creation: υπηρεσία δημιουργίας ερώτησης.
- Question Publishing: υπηρεσία δημοσίευσης ερώτησης.

Για τα παραπάνω ServiceInterfaces, ορίζονται τέσσερα απλά Interfaces:

- Question Taker: διεπαφή που υλοποιεί τη λειτουργία *confirmQuestion* και χρησιμοποιείται από τον παραλήπτη της νεοσύστατης ερώτησης. Εφόσον ορίζει τη συμπεριφορά και το ServiceInterface την πραγματοποιεί, έχουν τη σχέση *Realization*. Αποτελεί τον πάροχο της υπηρεσίας.
- Question Creator: διεπαφή που υλοποιεί τη λειτουργία *confirmCreatedQuestion* και χρησιμοποιείται από το δημιουργό της ερώτησης. Εφόσον το ServiceInterface πρέπει να υποστηρίζει τη λειτουργία αυτή, έχουν τη σχέση *Usage*. Αποτελεί τον καταναλωτή της υπηρεσίας.
- Question Receiver: διεπαφή που υλοποιεί τη λειτουργία *publishQuestion* και χρησιμοποιείται από τον παραλήπτη της δημοσιευμένης ερώτησης. Εφόσον ορίζει τη συμπεριφορά και το ServiceInterface την πραγματοποιεί, έχουν τη σχέση *Realization*. Αποτελεί τον πάροχο της υπηρεσίας.
- Question Publisher: διεπαφή που υλοποιεί τη λειτουργία *confirmPublishedQuestion*, και χρησιμοποιείται από το δημοσιευτή της ερώτησης. Εφόσον το ServiceInterface πρέπει να υποστηρίζει τη λειτουργία αυτή, έχουν τη σχέση *Usage*. Αποτελεί τον καταναλωτή της υπηρεσίας.

Στη συνέχεια ορίζονται οι παρακάτω τέσσερις Roles στα ServiceInterfaces:

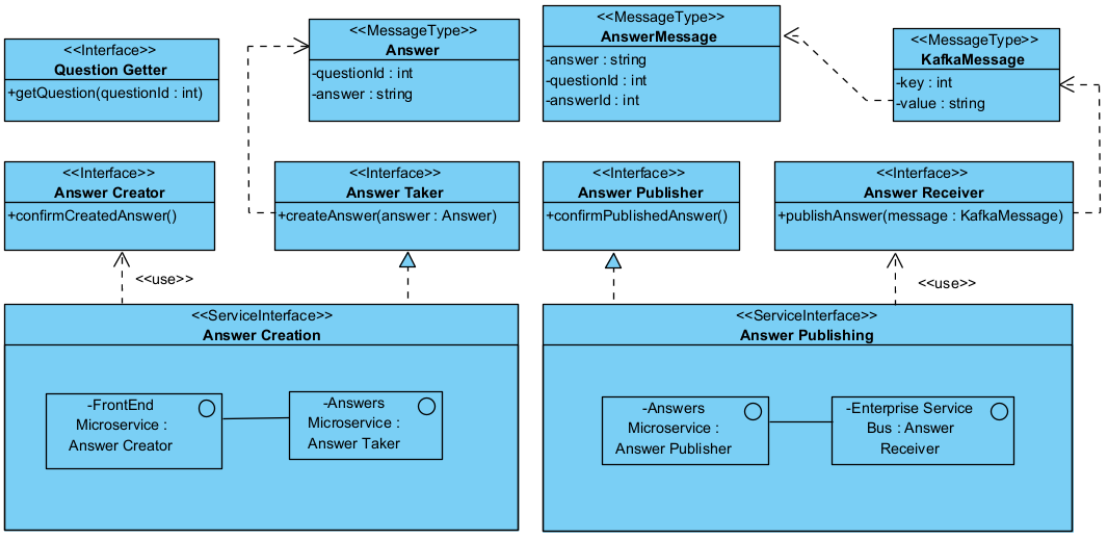
- FrontEnd Microservice: το microservice που δημιουργεί μια ερώτηση, για αυτό του προσδίδεται και ο τύπος "Question Creator" από τις διεπαφές που έχουν οριστεί.
- Ask Questions Microservice: το microservice που λαμβάνει μια ερώτηση, για αυτό του προσδίδεται και ο τύπος "Question Taker" από τις διεπαφές που έχουν οριστεί.
- Ask Questions Microservice: πέραν της παραλαβής μιας ερώτησης, το microservice αυτό τη δημοσιεύει, για αυτό του προσδίδεται και ο τύπος "Question Publisher" από τις διεπαφές που έχουν οριστεί.
- Enterprise Service Bus: το service bus που θα λαμβάνει μια ερώτηση, για αυτό του προσδίδεται και ο τύπος "Question Receiver" από τις διεπαφές που έχουν οριστεί:

Έπειτα ορίζονται δύο MessageTypes, οι οποίοι είναι κρίσιμοι για την επικοινωνία μεταξύ των υπηρεσιών, για αυτό και χρησιμοποιείται η σχέση *Dependency* για τη σύνδεσή τους με τις λειτουργίες *createQuestion* και *publishQuestion* των διεπαφών:

- QuestionMessage: ο τύπος μηνύματος αυτός περιέχει τρία πεδία: το *qname*, το *qtext* και το *qkeywords*, τα οποία είναι όλα strings.
- KafkaMessage: αντιπροσωπεύει τα μηνύματα που δημοσιεύονται στο Kafka, τα οποία πρέπει να είναι της μορφής *(key, value)*. Συνεπώς, ο τύπος μηνύματος αυτός περιέχει δύο πεδία: το *key*, το οποίο είναι τύπου int και το *value*, το οποίο είναι τύπου string. Επειδή το πεδίο *value* πρέπει να περιέχει ένα

*QuestionMessage*, χρησιμοποιείται η σχέση *Dependency* μεταξύ τους. Ο λόγος που το πεδίο *value* δεν είναι τύπου *QuestionMessage* είναι επειδή το Kafka αναμένει τύπο string.

Για την υπηρεσία **Answers** δημιουργείται το παρακάτω διάγραμμα:




**Σχήμα 5.5** Answers Service Interface Diagram

Έχουν δημιουργηθεί τα εξής ServiceInterfaces:

- Answer Creation: υπηρεσία δημιουργίας απάντησης.
- Answer Publishing: υπηρεσία δημοσίευσης απάντησης.

Στη συνέχεια, ορίζονται πέντε απλά Interfaces:

- Answer Taker: διεπαφή που υλοποιεί τη λειτουργία *createAnswer* και χρησιμοποιείται από τον παραλήπτη της νεοσύστατης απάντησης. Εφόσον ορίζει τη συμπεριφορά και το ServiceInterface την πραγματοποιεί, έχουν τη σχέση *Realization*. Αποτελεί τον πάροχο της υπηρεσίας.
- Answer Creator: διεπαφή που υλοποιεί τη λειτουργία *confirmCreatedAnswer* και χρησιμοποιείται από το δημιουργό της απάντησης. Εφόσον το ServiceInterface πρέπει να υποστηρίζει τη λειτουργία αυτή, έχουν τη σχέση *Usage*. Αποτελεί τον καταναλωτή της υπηρεσίας.
- Answer Receiver: διεπαφή που υλοποιεί τη λειτουργία *publishAnswer* και χρησιμοποιείται από τον παραλήπτη της δημοσιευμένης απάντησης. Εφόσον ορίζει τη συμπεριφορά και το ServiceInterface την πραγματοποιεί, έχουν τη σχέση *Realization*. Αποτελεί τον πάροχο της υπηρεσίας.
- Answer Publisher: διεπαφή που υλοποιεί τη λειτουργία *confirmPublishedAnswer*, και χρησιμοποιείται από το δημοσιευτή της απάντησης. Εφόσον το ServiceInterface πρέπει να υποστηρίζει τη λειτουργία αυτή, έχουν τη σχέση *Usage*. Αποτελεί τον καταναλωτή της υπηρεσίας.
- Question Getter: διεπαφή που υλοποιεί τη λειτουργία *getQuestion*. Εφόσον ορίζει υπηρεσία μια μονόδρομη επικοινωνία δε χρειάζεται να χρησιμοποιηθεί κάποιο ServiceInterface.

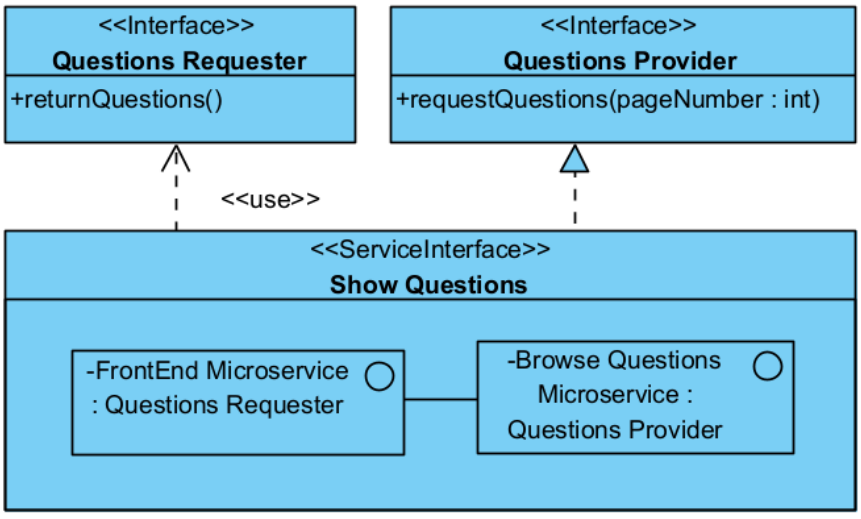
Στη συνέχεια ορίζονται οι παρακάτω τέσσερις Roles στα ServiceInterfaces:

- FrontEnd Microservice: το microservice που δημιουργεί μια απάντηση, για αυτό του προσδίδεται και ο τύπος "Answer Creator" από τις διεπαφές που έχουν οριστεί.
- Answers Microservice: το microservice που λαμβάνει μια απάντηση, για αυτό του προσδίδεται και ο τύπος "Answer Taker" από τις διεπαφές που έχουν οριστεί.
- Answers Microservice: πέραν της παραλαβής μιας απάντησης, το microservice αυτό τη δημοσιεύει, για αυτό του προσδίδεται και ο τύπος "Answer Publisher" από τις διεπαφές που έχουν οριστεί.
- Enterprise Service Bus: το service bus που θα λαμβάνει μια απάντηση, για αυτό του προσδίδεται και ο τύπος "Answer Receiver" από τις διεπαφές που έχουν οριστεί.

Έπειτα ορίζονται τρεις MessageTypes, οι οποίοι είναι κρίσιμοι για την επικοινωνία μεταξύ των υπηρεσιών, για αυτό και χρησιμοποιείται η σχέση *Dependency* για τη σύνδεσή τους με τις λειτουργίες *createAnswer* και *publishAnswer* των διεπαφών.

- Answer: ο τύπος μηνύματος αυτός περιέχει δύο πεδία: το *questionID*, το οποίο είναι τύπου int και το answer, το οποίο είναι τύπου string.
- KafkaMessage: ομοίως με πριν, αντιπροσωπεύει τα μηνύματα που δημοσιεύονται στο Kafka, τα οποία πρέπει να είναι της μορφής *(key, value)*. Συνεπώς, ο τύπος μηνύματος αυτός περιέχει δύο πεδία: το *key*, το οποίο είναι τύπου int και το *value*, το οποίο είναι τύπου string. Επειδή το πεδίο *value* πρέπει να περιέχει ένα *AnswerMessage*, χρησιμοποιείται η σχέση *Dependency* μεταξύ τους. Ο λόγος που το πεδίο *value* δεν είναι τύπου *AnswerMessage* είναι επειδή το Kafka αναμένει τύπο string.
- AnswerMessage: ο τύπος μηνύματος αυτός χρησιμοποιείται ως το πεδίο *value* για τον τύπο μηνύματος *KafkaMessage*. Περιέχει τρία πεδία: το *answer*, το οποίο είναι τύπου string, το *questionId*, το οποίο είναι τύπου int και το *answerId*, το οποίο είναι επίσης τύπου int.

Για την υπηρεσία **Browse Questions** δημιουργείται το παρακάτω διάγραμμα:




**Σχήμα 5.6** Browse Questions Service Interface Diagram

Έχει δημιουργηθεί το εξής ServiceInterface:

- Show Questions: υπηρεσία προβολής ερωτήσεων.

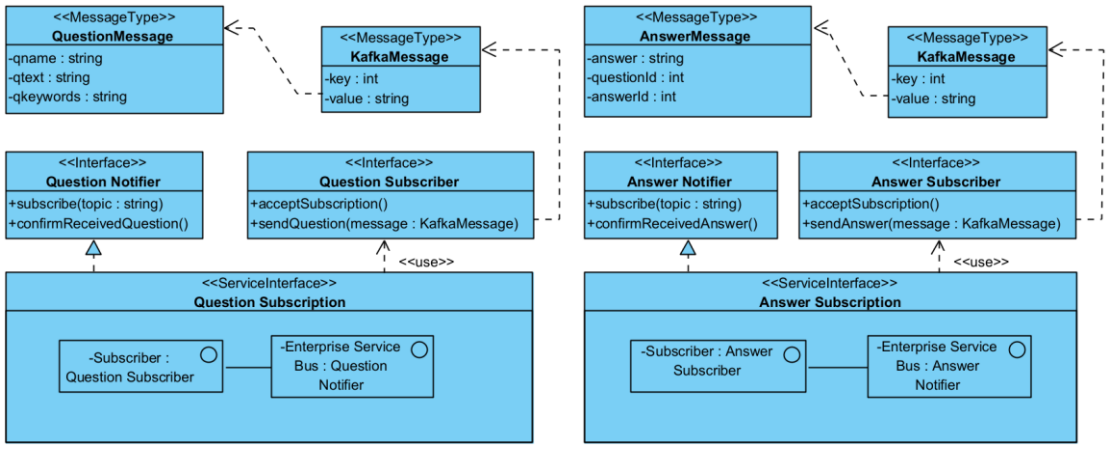
Στη συνέχεια, ορίζονται δύο απλά Interfaces:

- Questions Provider: διεπαφή που υλοποιεί τη λειτουργία *returnQuestions* και αποτελεί τον πάροχο της υπηρεσίας. Εφόσον ορίζει τη συμπεριφορά και το ServiceInterface την πραγματοποιεί, έχουν τη σχέση *Realization*.
- Questions Requester: διεπαφή που υλοποιεί τη λειτουργία *requestQuestions* και αποτελεί τον καταναλωτή της υπηρεσίας. Εφόσον το ServiceInterface πρέπει να υποστηρίζει τη λειτουργία αυτή, έχουν τη σχέση *Usage*.

Στη συνέχεια ορίζονται οι παρακάτω δύο Roles στο ServiceInterface:

- FrontEnd Microservice: το microservice που κάνει αίτημα για να λάβει τις πληροφορίες μιας ερώτησης, για αυτό του προσδίδεται και ο τύπος "Questions Requester" από τις διεπαφές που έχουν οριστεί.
- Browse Questions Microservice: το microservice που επιστρέφει τις πληροφορίες μιας ερώτησης, για αυτό του προσδίδεται και ο τύπος "Questions Provider" από τις διεπαφές που έχουν οριστεί.

Για το **Enterprise Service Bus** δημιουργείται το παρακάτω διάγραμμα:




**Σχήμα 5.7** Enterprise Service Bus Service Interface Diagram

Έχουν δημιουργηθεί τα εξής ServiceInterfaces:

- Question Subscription: υπηρεσία συνδρομής στο topic *QUESTION*.
- Answer Subscription: υπηρεσία συνδρομής στο topic *ANSWER*.

Για τα παραπάνω ServiceInterfaces, ορίζονται τέσσερα απλά Interfaces:

- Question Notifier: διεπαφή που υλοποιεί τις λειτουργίες *subscribe* και *confirmReceivedQuestion* και χρησιμοποιούνται από το συνδρομητή της ερώτησης. Εφόσον ορίζει τη συμπεριφορά και το ServiceInterface την

πραγματοποιεί, έχουν τη σχέση *Realization*. Αποτελεί τον πάροχο της υπηρεσίας.

- Question Subscriber: διεπαφή που υλοποιεί τις λειτουργίες *acceptSubscription* και *sendQuestion* και χρησιμοποιούνται από το δημοσιευτή της ερώτησης. Εφόσον το ServiceInterface πρέπει να υποστηρίζει τη λειτουργία αυτή, έχουν τη σχέση *Usage*. Αποτελεί τον καταναλωτή της υπηρεσίας.
- Answer Notifier: διεπαφή που υλοποιεί τις λειτουργίες *subscribe* και *confirmReceivedAnswer* και χρησιμοποιείται από το συνδρομητή της απάντησης. Εφόσον ορίζει τη συμπεριφορά και το ServiceInterface την πραγματοποιεί, έχουν τη σχέση *Realization*. Αποτελεί τον πάροχο της υπηρεσίας.
- Answer Subscriber: διεπαφή που υλοποιεί τις λειτουργίες *acceptSubscription* και *sendAnswer* και χρησιμοποιείται από το δημοσιευτή της απάντησης. Εφόσον το ServiceInterface πρέπει να υποστηρίζει τη λειτουργία αυτή, έχουν τη σχέση *Usage*. Αποτελεί τον καταναλωτή της υπηρεσίας.

Στη συνέχεια ορίζονται οι παρακάτω τέσσερις Roles στα ServiceInterfaces:

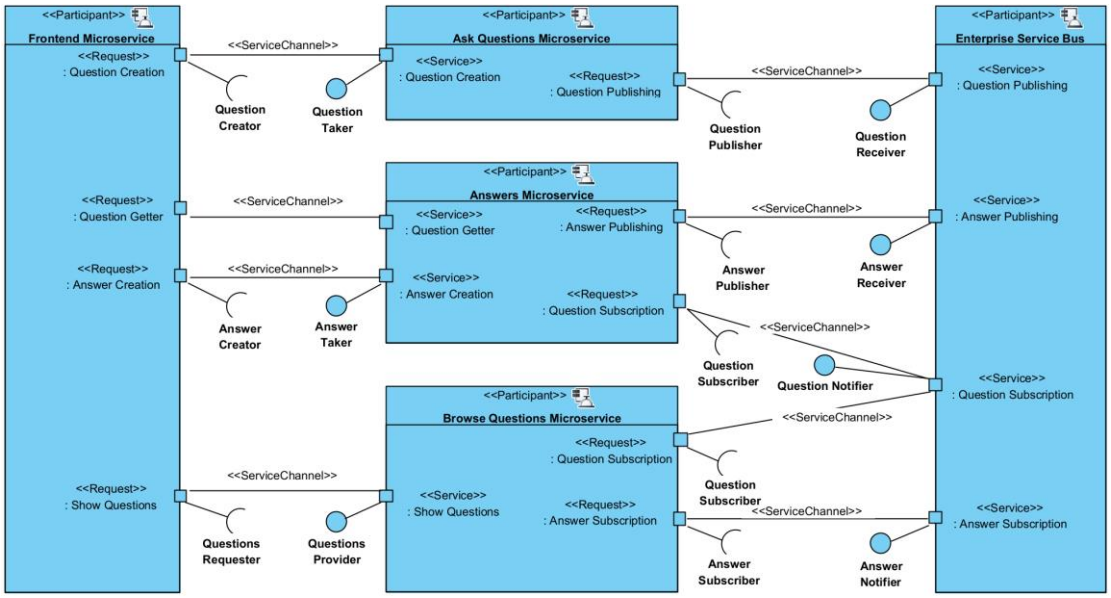
- Enterprise Service Bus: το microservice που ενημερώνει τους συνδρομητές για νέο μήνυμα τύπου ερώτησης, για αυτό του προσδίδεται και ο τύπος "Question Notifier" από τις διεπαφές που έχουν οριστεί.
- Subscriber: το microservice που κάνει subscribe στο topic *QUESTION*, για αυτό του προσδίδεται και ο τύπος "Question Subscriber" από τις διεπαφές που έχουν οριστεί.
- Enterprise Service Bus: το microservice που ενημερώνει τους συνδρομητές για νέο μήνυμα τύπου απάντησης, για αυτό του προσδίδεται και ο τύπος "Answer Notifier" από τις διεπαφές που έχουν οριστεί.
- Subscriber: το microservice που κάνει subscribe στο topic *ANSWER*, για αυτό του προσδίδεται και ο τύπος "Answer Subscriber" από τις διεπαφές που έχουν οριστεί.

Έπειτα ορίζονται τρεις MessageTypes, οι οποίοι είναι κρίσιμοι για την επικοινωνία μεταξύ των υπηρεσιών, για αυτό και χρησιμοποιείται η σχέση *Dependency* για τη σύνδεσή τους με τις λειτουργίες *sendQuestion* και *sendAnswer* των διεπαφών.

- QuestionMessage: ο τύπος μηνύματος αυτός περιέχει τρία πεδία: το *qname*, το *qtext* και το *qkeywords*, τα οποία είναι όλα strings.
- AnswerMessage: ο τύπος μηνύματος αυτός χρησιμοποιείται ως το πεδίο *value* για τον τύπο μηνύματος *KafkaMessage*. Περιέχει τρία πεδία: το *answer*, το οποίο είναι τύπου string, το *questionId*, το οποίο είναι τύπου int και το *answerId*, το οποίο είναι επίσης τύπου int.
- KafkaMessage: αντιπροσωπεύει τα μηνύματα που δημοσιεύονται στο Kafka, τα οποία πρέπει να είναι της μορφής *(key, value)*. Συνεπώς, ο τύπος μηνύματος αυτός περιέχει δύο πεδία: το *key*, το οποίο είναι τύπου int και το *value*, το οποίο είναι τύπου string. Επειδή το πεδίο *value* πρέπει να περιέχει είτε ένα μήνυμα τύπου *QuestionMessage* ή ένα τύπου *AnswerMessage*, χρησιμοποιείται η σχέση *Dependency* μεταξύ τους. Ο λόγος που το πεδίο *value* δεν είναι τύπου *QuestionMessage* ή *AnswerMessage* είναι επειδή το Kafka αναμένει τύπο string.

## 5.2.2 Service Participant Diagram

Στη συνέχεια, θα οριστεί το Service Participant Diagram για να τεκμηριωθούν οι συμμετέχοντες στο σύστημα, καθώς και οι διεπαφές που χρησιμοποιούν για επικοινωνία:




**Σχήμα 5.8** Microservices Service Participant Diagram

Αρχικά ορίζονται όλοι οι Participants του συστήματος:

- Frontend Microservice
- Ask Questions Microservice
- Answers Microservice
- Browse Questions Microservice
- Enterprise Service Bus

Για κάθε Participant ορίζονται οι θύρες οι οποίες υλοποιούν τα ServiceInterfaces που ορίστηκαν στα Service Interface Diagrams και χρησιμοποιούνται προκειμένου να παρέχονται και να καταναλώνονται υπηρεσίες, καθώς και τα απαραίτητα Interface Realizations και Usages που υλοποιούν τις αντίστοιχες διεπαφές:

Για τον Frontend Microservice Participant ορίζονται τα παρακάτω:

- Request Port τύπου “Question Creation”: υλοποιεί την κατανάλωση της υπηρεσίας που περιγράφει το ServiceInterface “Question Creation”. Μέσω της θύρας αυτής χρησιμοποιείται ένα *Usage* το οποίο υλοποιεί τη διεπαφή “Question Creator”.
- Request Port τύπου “Question Getter”: υλοποιεί την κατανάλωση της υπηρεσίας που περιγράφει η διεπαφή ‘Question Getter’.
- Request Port τύπου “Answer Creation”: υλοποιεί την κατανάλωση της υπηρεσίας που περιγράφει το ServiceInterface “Answer Creation”. Μέσω της

θύρας αυτής χρησιμοποιείται ένα *Usage* το οποίο υλοποιεί τη διεπαφή "Answer Creator".

- Request Port τύπου "Questions Requester": υλοποιεί την κατανάλωση της υπηρεσίας που περιγράφει το ServiceInterface "Show Questions". Μέσω της θύρας αυτής χρησιμοποιείται ένα *Usage* το οποίο υλοποιεί τη διεπαφή "Questions Requester".

Για τον Ask Questions Microservice Participant ορίζονται τα παρακάτω:

- Service Port τύπου "Question Creation": υλοποιεί την παροχή της υπηρεσίας που περιγράφει το ServiceInterface "Question Creation". Μέσω της θύρας αυτής χρησιμοποιείται ένα *Realization* το οποίο υλοποιεί τη διεπαφή "Question Taker".
- Request Port τύπου "Question Publishing": υλοποιεί την κατανάλωση της υπηρεσίας που περιγράφει το ServiceInterface "Question Publishing". Μέσω της θύρας αυτής χρησιμοποιείται ένα *Usage* το οποίο υλοποιεί τη διεπαφή "Question Publisher".

Για τον Answers Microservice Participant ορίζονται τα παρακάτω:

- Service Port τύπου "Question Getter": υλοποιεί την παροχή της υπηρεσίας που περιγράφει η διεπαφή 'Question Getter'.
- Service Port τύπου "Answer Creation": υλοποιεί την παροχή της υπηρεσίας που περιγράφει το ServiceInterface "Answer Creation". Μέσω της θύρας αυτής χρησιμοποιείται ένα *Realization* το οποίο υλοποιεί τη διεπαφή "Answer Taker".
- Request Port τύπου "Answer Publishing": υλοποιεί την κατανάλωση της υπηρεσίας που περιγράφει το ServiceInterface "Answer Publishing". Μέσω της θύρας αυτής χρησιμοποιείται ένα *Usage* το οποίο υλοποιεί τη διεπαφή "Answer Publisher".
- Request Port τύπου "Question Subscription": υλοποιεί την κατανάλωση της υπηρεσίας που περιγράφει το ServiceInterface "Question Subscription". Μέσω της θύρας αυτής χρησιμοποιείται ένα *Usage* το οποίο υλοποιεί τη διεπαφή "Question Subscriber".

Για τον Browse Questions Microservice Participant ορίζονται τα παρακάτω:

- Service Port τύπου "Show Questions": υλοποιεί την παροχή της υπηρεσίας που περιγράφει το ServiceInterface "Show Questions". Μέσω της θύρας αυτής χρησιμοποιείται ένα *Realization* το οποίο υλοποιεί τη διεπαφή "Questions Provider".
- Request Port τύπου "Question Subscription": υλοποιεί την κατανάλωση της υπηρεσίας που περιγράφει το ServiceInterface "Question Subscription". Μέσω της θύρας αυτής χρησιμοποιείται ένα *Usage* το οποίο υλοποιεί τη διεπαφή "Question Subscriber".
- Request Port τύπου "Answer Subscription": υλοποιεί την κατανάλωση της υπηρεσίας που περιγράφει το ServiceInterface "Answer Subscription". Μέσω της θύρας αυτής χρησιμοποιείται ένα *Usage* το οποίο υλοποιεί τη διεπαφή "Answer Subscriber".

Για τον Enterprise Service Bus Participant ορίζονται τα παρακάτω:

- Service Port τύπου “Question Publishing”: υλοποιεί την παροχή της υπηρεσίας που περιγράφει το ServiceInterface “Question Publishing”. Μέσω της θύρας αυτής χρησιμοποιείται ένα *Realization* το οποίο υλοποιεί τη διεπαφή “Question Receiver”.
- Service Port τύπου “Answer Publishing”: υλοποιεί την παροχή της υπηρεσίας που περιγράφει το ServiceInterface “Answer Publishing”. Μέσω της θύρας αυτής χρησιμοποιείται ένα *Realization* το οποίο υλοποιεί τη διεπαφή “Answer Receiver”.
- Service Port τύπου “Question Subscription”: υλοποιεί την παροχή της υπηρεσίας που περιγράφει το ServiceInterface “Question Subscription”. Μέσω της θύρας αυτής χρησιμοποιείται ένα *Realization* το οποίο υλοποιεί τη διεπαφή “Question Notifier”.
- Service Port τύπου “Answer Subscription”: υλοποιεί την παροχή της υπηρεσίας που περιγράφει το ServiceInterface “Answer Subscription”. Μέσω της θύρας αυτής χρησιμοποιείται ένα *Realization* το οποίο υλοποιεί τη διεπαφή “Answer Notifier”.

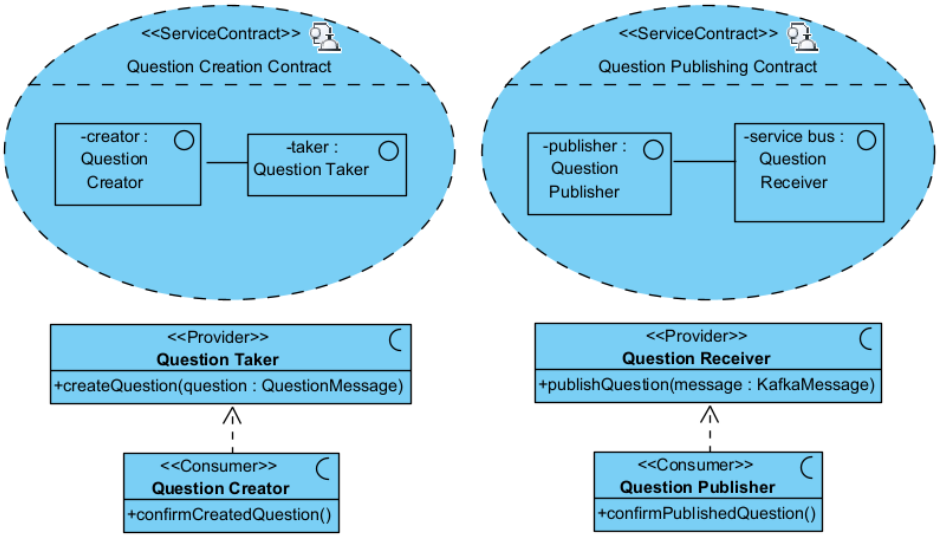
Γίνεται παντού η χρήση του *ServiceChannel*, εφόσον τα Request Ports είναι συμβατά και μπορούν να ενωθούν με τα Service Ports σύμφωνα με τις συνθήκες που υπάρχουν στη σελίδα 64 του specification [1]. Συγκεκριμένα, την 3η συνθήκη:

«Το Request και το Service έχουν συμβατές ανάγκες και ικανότητες αντίστοιχα. Αυτό σημαίνει πως το Service πρέπει να παρέχει ένα Operation για κάθε Operation που χρησιμοποιείται μέσω του Request, το Request πρέπει να παρέχει ένα Operation για κάθε Operation που χρησιμοποιείται μέσω του Service, και τα πρωτόκολλα για το πως οι ικανότητες είναι ικανές μεταξύ του Request και του Service.»

### 5.2.3 Service Contract Diagrams

Στη συνέχεια, θα οριστούν τα Service Contract Diagrams για να τεκμηριωθούν οι συμφωνίες μεταξύ των συμμετεχόντων στο σύστημα, για κάθε ServiceContract δημιουργείται και ένα Sequence Sub Diagram για να τεκμηριωθεί περαιτέρω η αλληλεπίδραση μεταξύ των συμμετεχόντων:

Για την υπηρεσία **Ask Questions** δημιουργείται το παρακάτω διάγραμμα:




**Σχήμα 5.9** Ask Questions Service Contract Diagram

Το παραπάνω διάγραμμα ορίζει δύο ServiceContracts, ένα για την υπηρεσία δημιουργίας ερώτησης (Question Creation Contract) και ένα για τη δημοσίευση ερώτησης (Question Publishing Contract).

Στο Question Creation Contract, ορίζονται δύο ρόλοι με τύπους που ορίζονται ως διεπαφές στο Ask Questions Service Interface Diagram:

- creator, με τύπο Question Creator
- taker, με τύπο Question Taker

Στο Question Publishing Contract, ορίζονται δύο ρόλοι με τύπους που επίσης ορίζονται ως διεπαφές στο Ask Questions Service Interface Diagram:

- publisher, με τύπο Question Publisher
- service bus, με τύπο Question Receiver

Επίσης, για κάθε ServiceContract δημιουργείται ένας Provider και ένας Consumer που αντιστοιχούν στους ρόλους που έχουν οριστεί.

Για το Question Creation Contract έχουν δημιουργηθεί:

- Provider: Question Taker
- Consumer: Question Creator

Για το Question Publishing Contract έχουν δημιουργηθεί:

- Provider: Question Receiver
- Consumer: Question Publisher

Εφόσον και στις δύο περιπτώσεις εκείνος που εκκινεί την επικοινωνία είναι ο Consumer, δημιουργείται ένα Dependency από τον κάθε Consumer στον αντίστοιχο Provider.

Ακολουθούν οι περιγραφές και τα Sequence Sub Diagrams των παραπάνω ServiceContracts. Οι περιγραφές ορίζουν τον τρόπο επικοινωνίας μεταξύ των συμμετεχόντων στο συμβόλαιο, κάνοντας τις απαραίτητες αναφορές στους τύπους μηνυμάτων που ορίστηκαν στο Ask Questions Service Interface Diagram και τα περιεχόμενά τους, έτσι ώστε να υλοποιηθούν οι ζητούμενες λειτουργίες. Τα Sequence Diagrams δημιουργούνται ως Sub Diagrams των αντίστοιχων συμβολαίων και επιδεικνύουν την αλληλεπίδραση μεταξύ των ρόλων που έχουν οριστεί.

Description:

This contract is created for and used in conjunction with the Question Creation service.
The taker : Question Taker is the Provider and the creator : Question Creator is the Consumer.

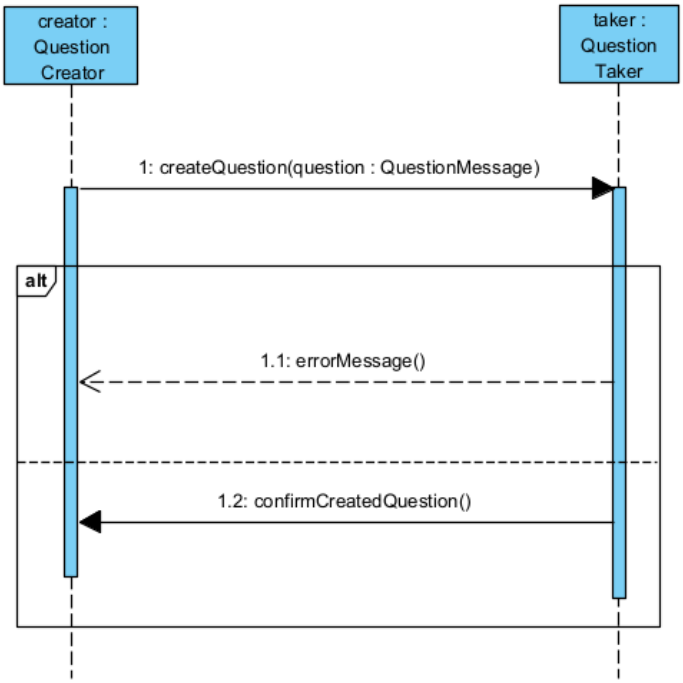
Terms:

1. The creator : Question Creator will implement createQuestion(question : QuestionMessage) by sending an HTTPS POST Request to the taker : Question Taker.
2. The HTTPS Request must have a body that contains a QuestionMessage in JSON format and the members qname : string, qtext : string must exist and not be null, otherwise the creator : Question Creator will receive a validation error.
3. The taker : Question Taker will implement confirmCreatedQuestion() by replying to the HTTPS Request with one of the following status codes:

- 200: if the message is valid.
- 400: if there was a validation error (at least one of the members qname : string, qtext : string was undefined or null).
- 500: if there was an internal server error.

This communication is also documented in the Question Creation Contract Sequence Diagram.

**Σχήμα 5.10** Question Creation Contract Description




**Σχήμα 5.11** Question Creation Contract Sequence Diagram

Description:

This contract is created for and used in conjunction with the Question Publishing service.
The service bus : Question Receiver is the Provider and the publisher : Question Publisher is the Consumer.

Terms:

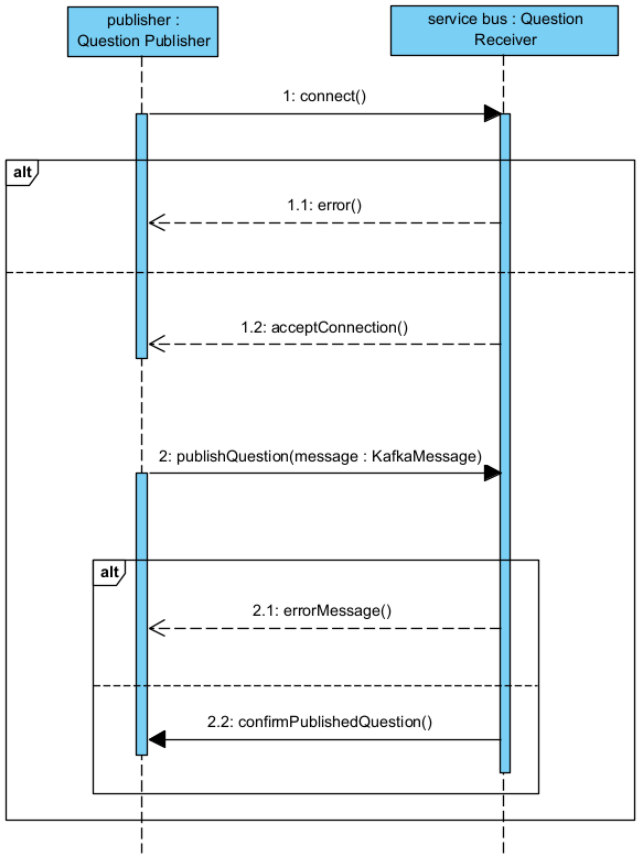
1. The publisher : Question Publisher will send a message to the service bus : Question Receiver. The message that will be sent must be a KafkaMessage that contains a unique key : int and a value : string that contains a QuestionMessage the publisher : Question Publisher has received.
2. In case the service bus : Question Receiver doesn't receive the message, an error is logged.

Conditions:

1. The publisher : Question Publisher must be connected to the service bus : Question Receiver which is a Kafka broker and send a message to the topic "**QUESTION**".

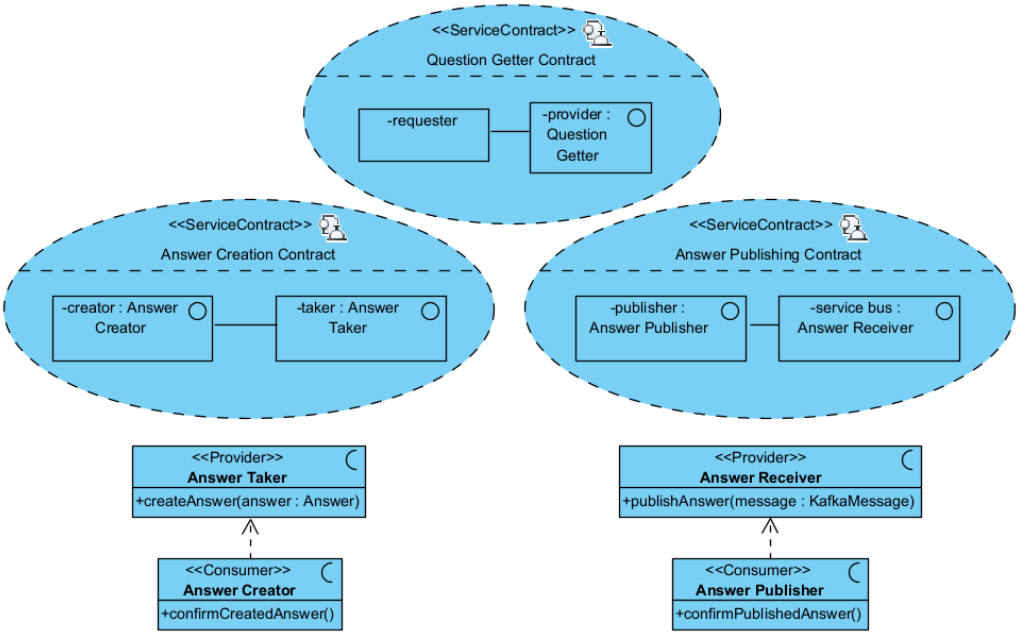
This communication is also documented in the Question Publishing Contract Sequence Diagram.

**Σχήμα 5.12** Question Publishing Contract Description




**Σχήμα 5.13** Question Publishing Contract Sequence Diagram

Για την υπηρεσία **Answers** δημιουργείται το παρακάτω διάγραμμα:




**Σχήμα 5.14** Answers Service Contract Diagram

Το παραπάνω διάγραμμα ορίζει τρία ServiceContracts, ένα για την υπηρεσία δημιουργίας απάντησης (Answer Creation Contract), ένα για τη δημοσίευση απάντησης (Answer Publishing Contract) και ένα για τη λήψη μιας ερώτησης (Question Getter Contract).

Στο Answer Creation Contract, ορίζονται δύο ρόλοι με τύπους που ορίζονται ως διεπαφές στο Answers Service Interface Diagram:

- creator, με τύπο Answer Creator
- taker, με τύπο Answer Taker

Στο Answer Publishing Contract, ορίζονται δύο ρόλοι με τύπους που επίσης ορίζονται ως διεπαφές στο Answers Service Interface Diagram:

- publisher, με τύπο Answer Publisher
- service bus, με τύπο Answer Receiver

Στο Question Getter Contract, ορίζονται δύο ρόλοι, εκ των οποίων μόνο ένας έχει τύπο που ορίζεται ως διεπαφή στο Answers Service Interface Diagram, εφόσον περιγράφει μονόδρομη επικοινωνία:

- requester
- provider, με τύπο Question Getter

Επίσης, για κάθε ServiceContract που δημιουργήθηκε για ServiceInterface, δημιουργείται ένας Provider και ένας Consumer που αντιστοιχούν στους ρόλους που έχουν οριστεί.

Για το Answer Creation Contract έχουν δημιουργηθεί:

- Provider: Answer Taker

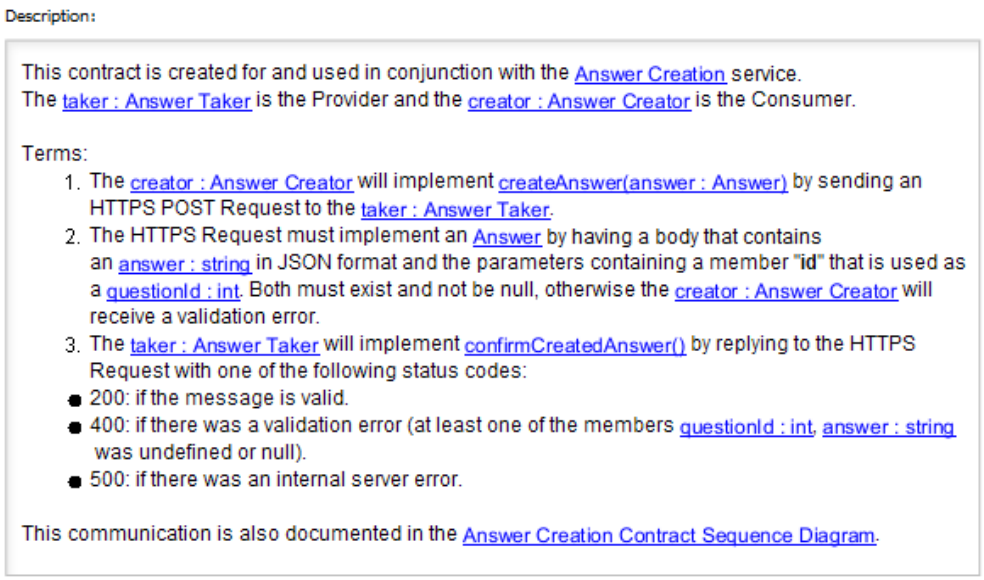
- Consumer: Answer Creator

Για το Answer Publishing Contract έχουν δημιουργηθεί:

- Provider: Answer Receiver
- Consumer: Answer Publisher

Εφόσον και στις δύο περιπτώσεις εκείνος που εκκινεί την επικοινωνία είναι ο Consumer, δημιουργείται ένα Dependency από τον κάθε Consumer στον αντίστοιχο Provider.

Ακολουθούν οι περιγραφές και τα Sequence Sub Diagrams των παραπάνω ServiceContracts. Οι περιγραφές ορίζουν τον τρόπο επικοινωνίας μεταξύ των συμμετεχόντων στο συμβόλαιο, κάνοντας τις απαραίτητες αναφορές στους τύπους μηνυμάτων που ορίστηκαν στο Answers Service Interface Diagram και τα περιεχόμενά τους, έτσι ώστε να υλοποιηθούν οι ζητούμενες λειτουργίες. Τα Sequence Diagrams δημιουργούνται ως Sub Diagrams των αντίστοιχων συμβολαίων και επιδεικνύουν την αλληλεπίδραση μεταξύ των ρόλων που έχουν οριστεί.


Description:

This contract is created for and used in conjunction with the Answer Creation service.
The taker : Answer Taker is the Provider and the creator : Answer Creator is the Consumer.

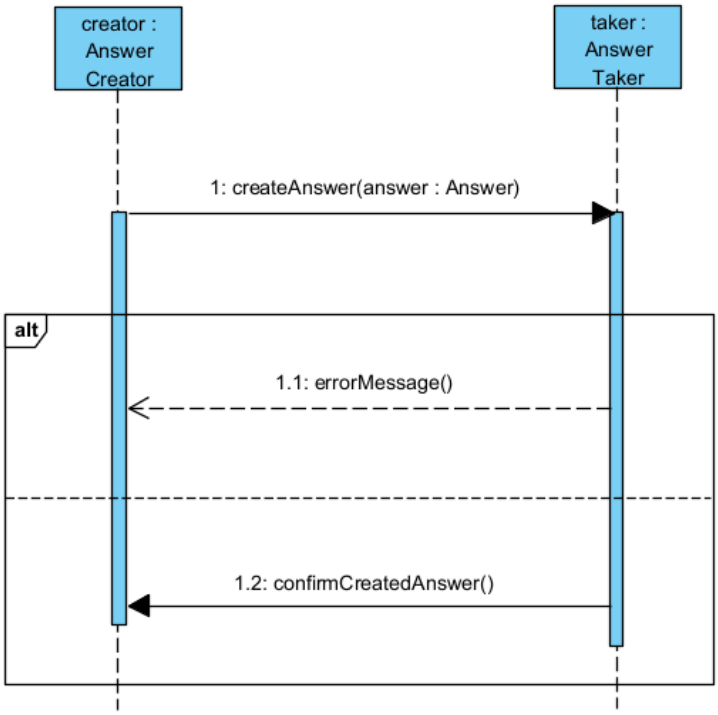
Terms:

1. The creator : Answer Creator will implement createAnswer(answer : Answer) by sending an HTTPS POST Request to the taker : Answer Taker.
2. The HTTPS Request must implement an Answer by having a body that contains an answer : string in JSON format and the parameters containing a member "**id**" that is used as a questionId : int. Both must exist and not be null, otherwise the creator : Answer Creator will receive a validation error.
3. The taker : Answer Taker will implement confirmCreatedAnswer() by replying to the HTTPS Request with one of the following status codes:

- 200: if the message is valid.
- 400: if there was a validation error (at least one of the members questionId : int, answer : string was undefined or null).
- 500: if there was an internal server error.

This communication is also documented in the Answer Creation Contract Sequence Diagram.

**Σχήμα 5.15** Answer Creation Contract Description




**Σχήμα 5.16** Answer Creation Contract Sequence Diagram

Description:

This contract is created for and used in conjunction with the Answer Publishing service.
The Answer Receiver is the Provider and the Answer Publisher is the Consumer.

Terms:

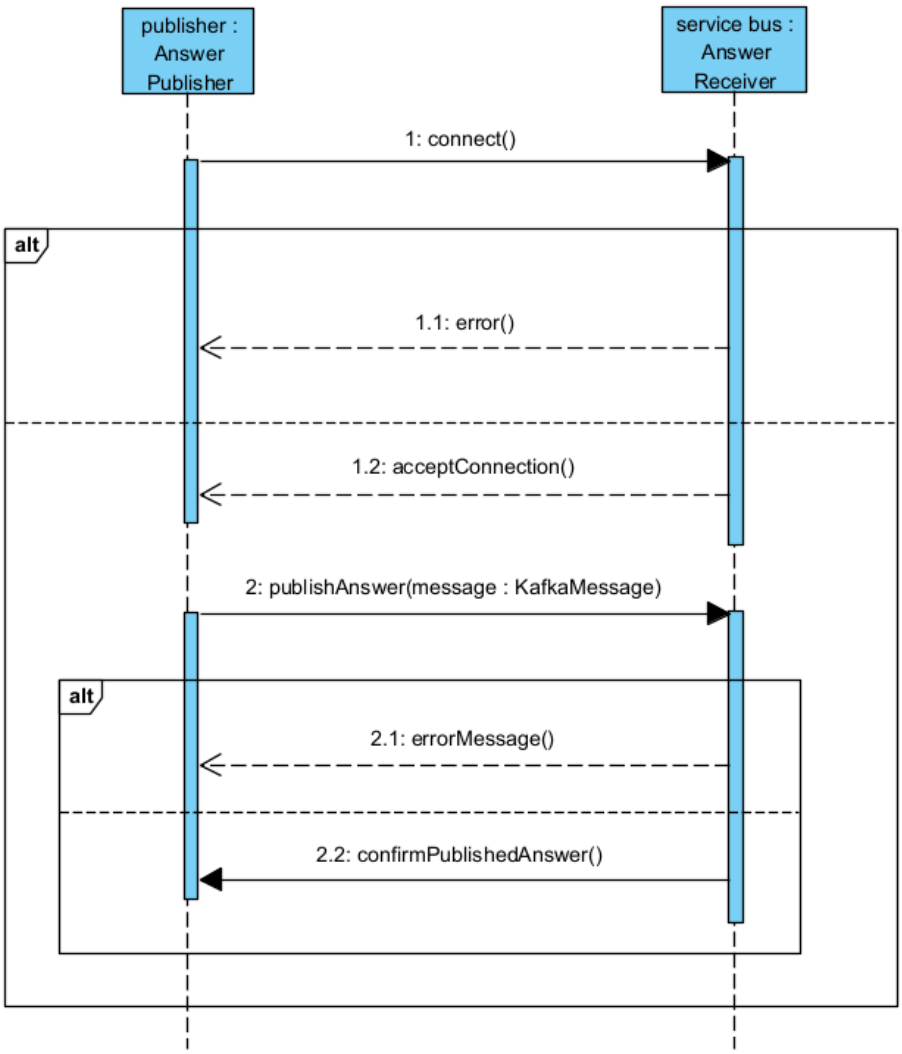
1. The publisher : Answer Publisher will send a message to the service bus : Answer Receiver. The message that will be sent must be a AnswerMessage that contains a key : int and a value : string that contains an Answer the publisher : Answer Publisher has received.
2. In case the service bus : Answer Receiver doesn't receive the message, an error is logged.

Conditions:

1. The publisher : Answer Publisher must be connect to the service bus : Answer Receiver which is a Kafka broker and send a message to the topic "**ANSWER**".

This communication is also documented in the Answer Publishing Contract Sequence Diagram.

**Σχήμα 5.17** Answer Publishing Contract Sequence Diagram




**Σχήμα 5.18** Answer Publishing Contract Sequence Diagram

Description:

This contract is created for and used in conjunction with the Question Getter interface.

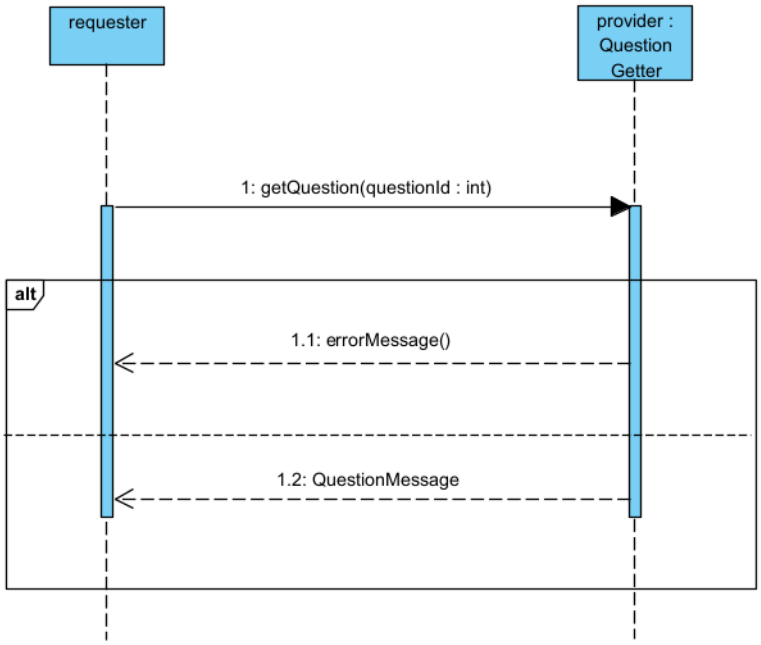
Terms:

1. The requester will implement a getQuestion(questionId : int) by sending an HTTPS GET Request to the provider : Question Getter.
2. The HTTPS Request must contain a member "**id**" in its parameters that is of type **int** and is used as a questionId.
3. The provider : Question Getter will reply to the HTTPS Request with one of the following status codes:

- 200: if the message is valid, and will also return the question (QuestionMessage) that was requested.
- 404: if the question doesn't exist.
- 500: if there was an internal server error.

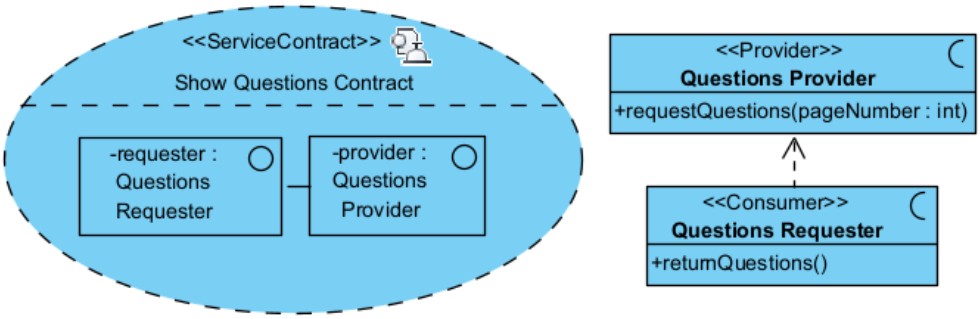
This communication is also documented in the Question Getter Contract Sequence Diagram.

**Σχήμα 5.19** Question Getter Contract Description




**Σχήμα 5.20** Question Getter Contract Sequence Diagram

Για την υπηρεσία **Browse Questions** δημιουργείται το παρακάτω διάγραμμα:




**Σχήμα 5.21** Browse Questions Service Contract Diagram

Το παραπάνω διάγραμμα ορίζει ένα ServiceContract, για την υπηρεσία προβολής ερωτήσεων (Show Questions Contract).

Στο Show Questions Contract, ορίζονται δύο ρόλοι με τύπους που ορίζονται ως διεπαφές στο Browse Questions Service Interface Diagram:

- requester, με τύπο Questions Requester
- provider, με τύπο Questions Provider

Επίσης, δημιουργείται ένας Provider και ένας Consumer που αντιστοιχούν στους ρόλους που έχουν οριστεί.

- Provider: Questions Provider
- Consumer: Questions Requester

Εφόσον εκείνος που εκκινεί την επικοινωνία είναι ο Consumer, δημιουργείται ένα Dependency από τον Consumer στον Provider.

Ακολουθούν οι περιγραφές και τα Sequence Sub Diagrams των παραπάνω ServiceContracts. Οι περιγραφές ορίζουν τον τρόπο επικοινωνίας μεταξύ των συμμετεχόντων στο συμβόλαιο, κάνοντας τις απαραίτητες αναφορές στους τύπους μηνυμάτων που ορίστηκαν στο Browse Questions Service Interface Diagram και τα περιεχόμενά τους, έτσι ώστε να υλοποιηθούν οι ζητούμενες λειτουργίες. Τα Sequence Diagrams δημιουργούνται ως Sub Diagrams των αντίστοιχων συμβολαίων και επιδεικνύουν την αλληλεπίδραση μεταξύ των ρόλων που έχουν οριστεί.

Description:

This contract is created for and used in conjunction with the Show Questions service.
The provider : Questions Provider is the Provider and the requester : Questions Requester is the Consumer.

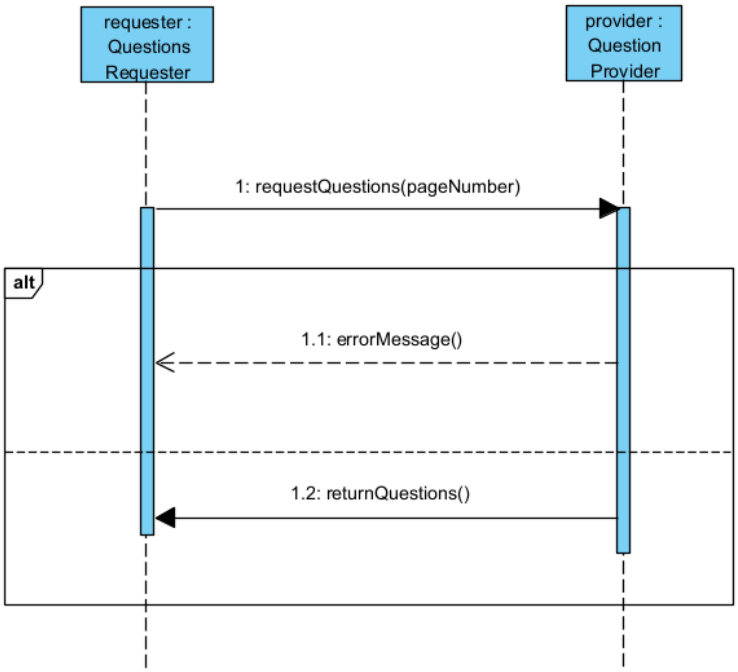
Terms:

1. The requester : Questions Requester will implement requestQuestions(pageNumber : int) by sending an HTTPS POST Request to the provider : Questions Provider.
2. The HTTPS Request must have a body that contains an integer "pageNumber" in JSON format.
3. The provider : Questions Provider will implement returnQuestions() by replying to the HTTPS Request with one of the following status codes:

- 200: if the message is valid and will also return all the questions (list of QuestionMessage) that exist in the page that was requested.
- 404: if the page doesn't exist (invalid pageNumber).
- 500: if there was an internal server error.

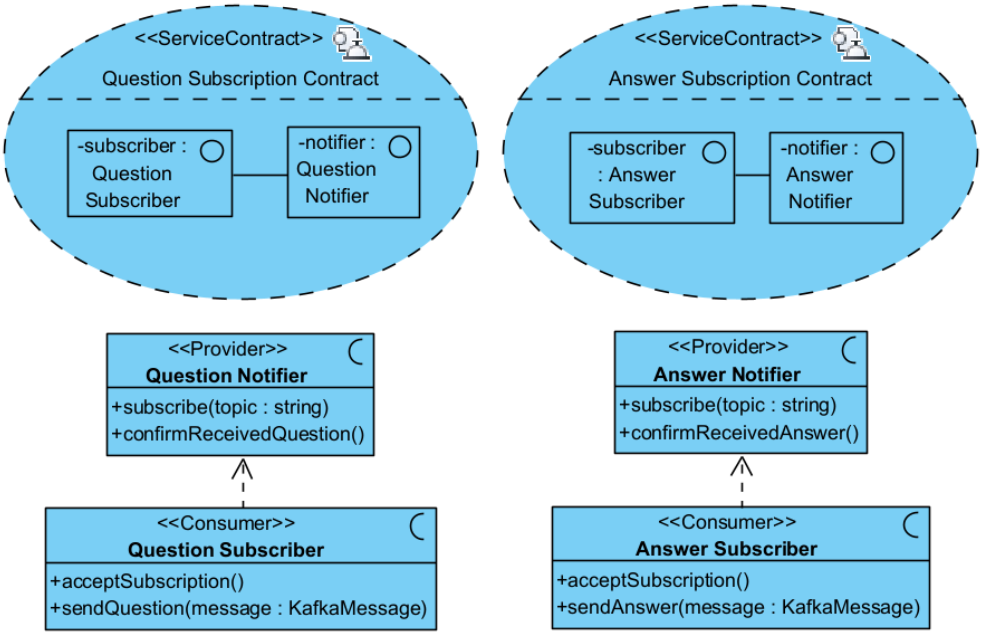
This communication is also documented in Show Questions Contract Sequence Diagram.

**Σχήμα 5.22** Show Questions Contract Description




**Σχήμα 5.23** Show Questions Contract Sequence Diagram

Για το **Enterprise Service Bus** δημιουργείται το παρακάτω διάγραμμα:




**Σχήμα 5.24** Subscription Service Contract Diagram

Το παραπάνω διάγραμμα ορίζει δύο ServiceContracts, ένα για την υπηρεσία συνδρομής στο topic *QUESTION* (Question Subscription Contract) και ένα για την υπηρεσία συνδρομής στο topic *ANSWER* (Answer Subscription Contract).

Στο Question Subscription Contract, ορίζονται δύο ρόλοι με τύπους που ορίζονται ως διεπαφές στο Enterprise Service Bus Service Interface Diagram:

- subscriber, με τύπο Question Subscriber
- notifier, με τύπο Question Notifier

Στο Answer Subscription Contract, ορίζονται δύο ρόλοι με τύπους που επίσης ορίζονται ως διεπαφές στο Enterprise Service Bus Service Interface Diagram:

- subscriber, με τύπο Answer Subscriber
- notifier, με τύπο Answer Notifier

Επίσης, για κάθε ServiceContract δημιουργείται ένας Provider και ένας Consumer που αντιστοιχούν στους ρόλους που έχουν οριστεί.

Για το Question Subscription Contract έχουν δημιουργηθεί:

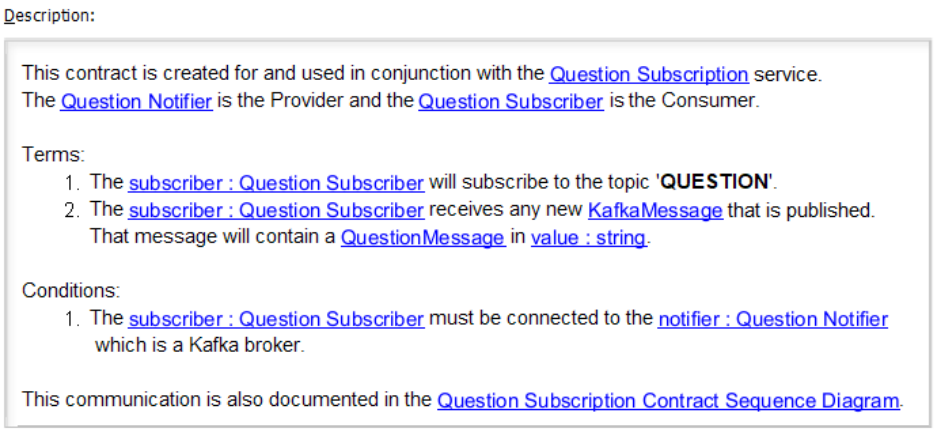
- Provider: Question Subscriber
- Consumer: Question Notifier

Για το Answer Subscription Contract έχουν δημιουργηθεί:

- Provider: Answer Subscriber
- Consumer: Answer Notifier

Εφόσον και στις δύο περιπτώσεις εκείνος που εκκινεί την επικοινωνία είναι ο Consumer, δημιουργείται ένα Dependency από τον κάθε Consumer στον αντίστοιχο Provider.

Ακολουθούν οι περιγραφές και τα Sequence Sub Diagrams των παραπάνω ServiceContracts. Οι περιγραφές ορίζουν τον τρόπο επικοινωνίας μεταξύ των συμμετεχόντων στο συμβόλαιο, κάνοντας τις απαραίτητες αναφορές στους τύπους μηνυμάτων που ορίστηκαν στο Enterprise Service Bus Service Interface Diagram και τα περιεχόμενά τους, έτσι ώστε να υλοποιηθούν οι ζητούμενες λειτουργίες. Τα Sequence Diagrams δημιουργούνται ως Sub Diagrams των αντίστοιχων συμβολαίων και επιδεικνύουν την αλληλεπίδραση μεταξύ των ρόλων που έχουν οριστεί.


Description:

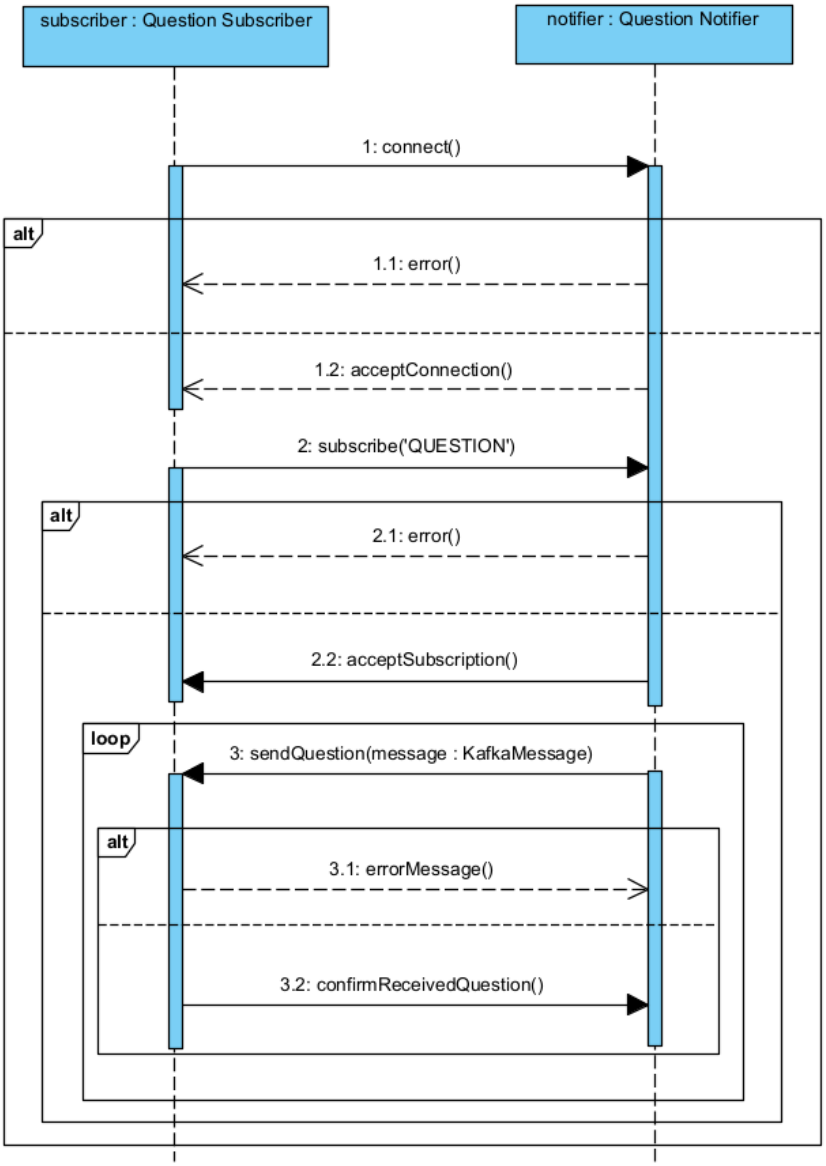
This contract is created for and used in conjunction with the Question Subscription service.
The Question Notifier is the Provider and the Question Subscriber is the Consumer.

Terms:
1. The subscriber : Question Subscriber will subscribe to the topic '**QUESTION**'.
2. The subscriber : Question Subscriber receives any new KafkaMessage that is published. That message will contain a QuestionMessage in value : string.

Conditions:
1. The subscriber : Question Subscriber must be connected to the notifier : Question Notifier which is a Kafka broker.

This communication is also documented in the Question Subscription Contract Sequence Diagram.

**Σχήμα 5.25** Question Subscription Contract Description




**Σχήμα 5.26** Question Subscription Contract Sequence Diagram

Description:

This contract is created for and used in conjunction with the Answer Subscription service.
The Answer Notifier is the Provider and the Answer Subscriber is the Consumer.

Terms:

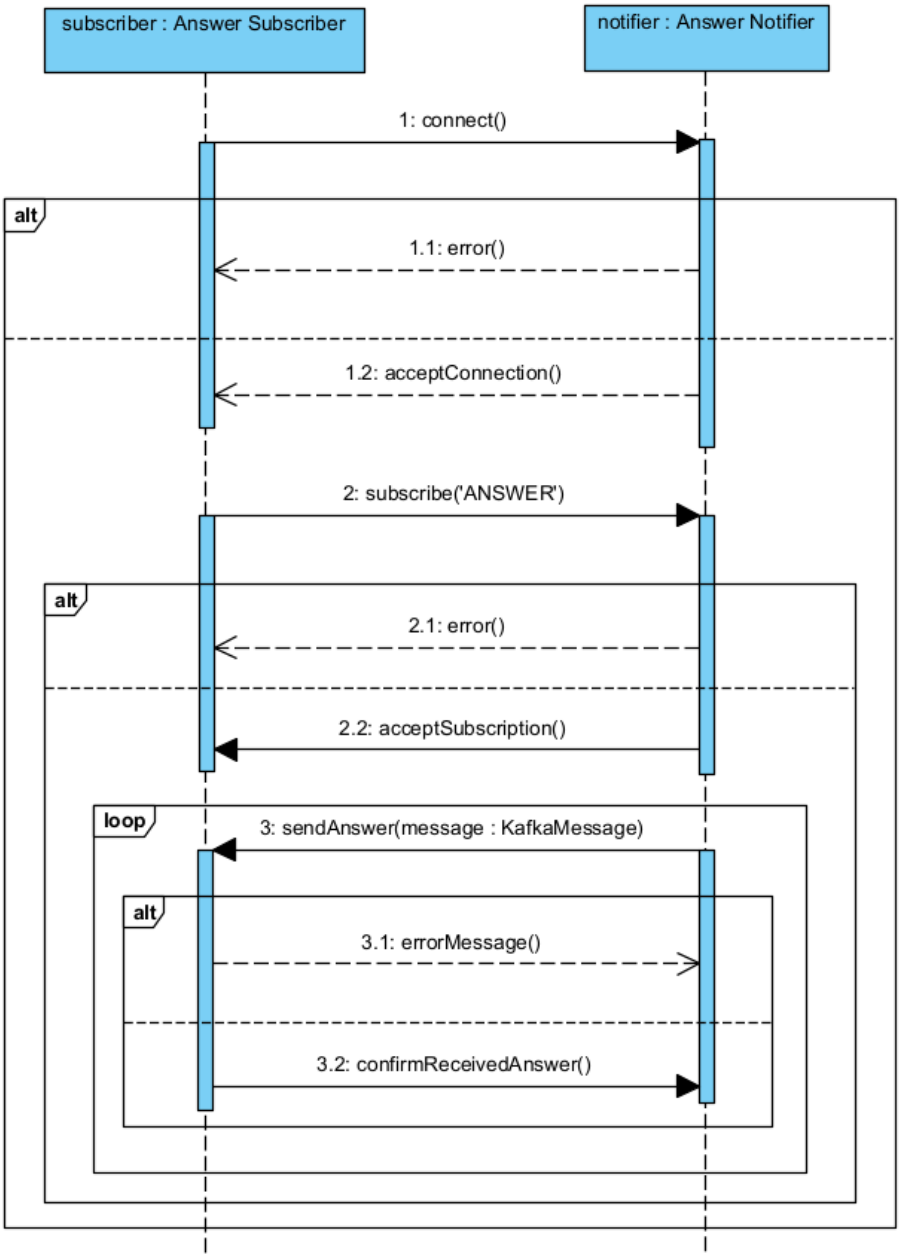
1. The subscriber : Answer Subscriber will subscribe to the topic '**ANSWER**'.
2. The subscriber : Answer Subscriber receives any new KafkaMessage that is published. That message will contain a AnswerMessage in value : string.

Conditions:

1. The subscriber : Answer Subscriber must be connected to the notifier : Answer Notifier which is a Kafka broker.

This communication is also documented in the Answer Subscription Contract Sequence Diagram.

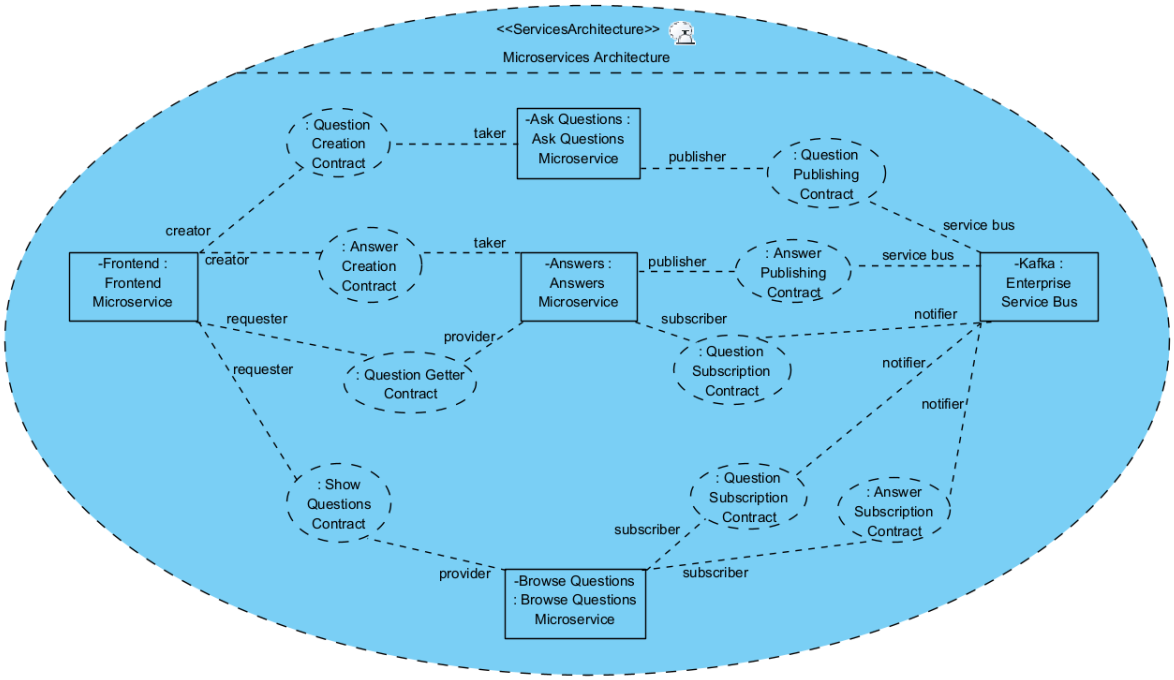
**Σχήμα 5.27** Answer Subscription Contract Description




**Σχήμα 5.28** Answer Subscription Contract Sequence Diagram

## 5.2.4 Services Architecture Diagram

Τέλος, θα οριστεί το Services Architecture Diagram, το οποίο θα δείξει την αρχιτεκτονική του συστήματος, δηλαδή την αλληλεπίδραση μεταξύ των συμμετεχόντων, με χρήση των ServiceContracts που έχουν οριστεί:




**Σχήμα 5.29** Microservices Services Architecture Diagram

Δημιουργείται το ServicesArchitecture με όνομα "Microservices Architecture". Με βάση τους Participants που έχουν οριστεί στο Microservices Service Participant Diagram, έχουν οριστεί 5 Internal Participants:

- Frontend, με τύπο Frontend Microservice
- Ask Questions, με τύπο Ask Questions Microservice
- Answers, με τύπο Answers Microservice
- Browse Questions, με τύπο Browse Questions Microservice
- Kafka, με τύπο Enterprise Service Bus

Παρακάτω αναλύονται τα collaboration μεταξύ τους:

Μεταξύ του Frontend και του Ask Questions

- Γίνεται χρήση του *Question Creation Contract*, με το Frontend να παίρνει το ρόλο του creator και το Ask Questions να παίρνει το ρόλο του taker.

Μεταξύ του Frontend και του Answers

- Γίνεται χρήση του *Answers Creation Contract*, με το Frontend να παίρνει το ρόλο του creator και το Answers να παίρνει το ρόλο του taker.
- Γίνεται χρήση του *Question Getter Contract*, με το Frontend να παίρνει το ρόλο του requester και το Answers να παίρνει το ρόλο του provider.

Μεταξύ του Frontend και του Browse Questions

- Γίνεται χρήση του *Show Questions Contract*, με το Frontend να παίρνει το ρόλο του requester και το Answers να παίρνει το ρόλο του provider.

Μεταξύ του Ask Questions και του Kafka

- Γίνεται χρήση του *Question Publishing Contract*, με το Ask Questions να παίρνει το ρόλο του publisher και το Kafka να παίρνει το ρόλο του service bus.

Μεταξύ του Answers και του Kafka

- Γίνεται χρήση του *Answers Publishing Contract*, με το Answers να παίρνει το ρόλο του publisher και το Kafka να παίρνει το ρόλο του service bus.
- Γίνεται χρήση του *Question Subscription Contract*, με το Answers να παίρνει το ρόλο του subscriber και το Kafka να παίρνει το ρόλο του notifier.

Μεταξύ του Browse Questions και του Kafka

- Γίνεται χρήση του *Question Subscription Contract*, με το Answers να παίρνει το ρόλο του subscriber και το Kafka να παίρνει το ρόλο του notifier.
- Γίνεται χρήση του *Answer Subscription Contract*, με το Answers να παίρνει το ρόλο του subscriber και το Kafka να παίρνει το ρόλο του notifier.

*Κεφάλαιο 6*

# Ενοποίηση με blockchain με χρήση της τεκμηρίωσης SoaML

## 6.1 Συγγραφή Έξυπνου Συμβολαίου

Το έξυπνο συμβόλαιο που θα χρησιμοποιηθεί χρειάζεται να μπορεί να αποθηκεύσει τις πληροφορίες που είναι απαραίτητες για τις λειτουργίες της υπηρεσίας Graphs, δηλαδή τις λέξεις-κλειδιά, τον αριθμό των ερωτήσεων και τον αριθμό των απαντήσεων ανά μέρα. Επίσης πρέπει να έχει συναρτήσεις που να επιτρέπουν την προσθήκη και την ανάκτηση πληροφοριών από το συμβόλαιο. Συνεπώς δημιουργείται το παρακάτω συμβόλαιο:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;
import "./Strings.sol";

contract Graphs {
  using Strings for string;

  // state variables
  struct Day {
    string[] keywords;
    uint questions;
    uint answers;
    uint noKeywords; // number of questions that don't have keywords
  }

  // map a uint that represents date to a custom struct
  mapping(uint => Day) daysStorage;

  // add keywords, question, answers, noKeywords
  function addToDay(uint date, string memory newKeywordsString, uint
newQuestions, uint newAnswers, uint newNoKeywords) public {
    require(date % 86400 == 0 , "Date given must have a time of
00:00:00.");

    // keywords are given
    if (bytes(newKeywordsString).length != 0) {
      require(newQuestions >= (newNoKeywords + 1), "The number of questions
doesn't match the sum of questions with and without keywords.");
      string[] memory newKeywords = newKeywordsString.split(",");
      // add new keywords to day
      for (uint i = 0; i < newKeywords.length; i++) {
        daysStorage[date].keywords.push(newKeywords[i]);
      }
    }
    else {
      require(newQuestions >= newNoKeywords, "The number of questions
doesn't match the sum of questions with and without keywords.");
    }
    // update questions/answers/noKeywords number
    daysStorage[date].questions += newQuestions;
    daysStorage[date].answers += newAnswers;
    daysStorage[date].noKeywords += newNoKeywords;
  }
```

```
  // get (keywords, questionsPerDay, answersPerDay) for a period of time
(dateFrom, DateTo)
  function getDays(uint dateFrom, uint dateTo) public view
  returns (string[] memory keywords, uint[] memory questions, uint[] memory
answers, uint[] memory noKeywords) {
    require(dateFrom % 86400 == 0 , "First date given must have a time of
00:00:00");
    require(dateTo % 86400 == 0 , "Second date given must have a time of
00:00:00");
    require(dateFrom <= dateTo , "Second date given must be the same or
later than the first date");

    uint totalDays = ((dateTo - dateFrom) / 86400) + 1;

    // get number of all keywords that will be returned to create fixed
size array
    uint keywordsSize = 0;
    for (uint tempDate = dateFrom; tempDate <= dateTo; tempDate += 86400) {
      keywordsSize += daysStorage[tempDate].keywords.length;
    }

    // define accumulators to return
    keywords = new string[](keywordsSize);
    questions = new uint[](totalDays);
    answers = new uint[](totalDays);
    noKeywords = new uint[](totalDays);

    // add (all keywords)/(questions per day)/(answers per day)/(noKeywords
per day) to return arrays
    uint keywordsIndex = 0;
    uint dayIndex = 0;
    for (uint tempDate = dateFrom; tempDate <= dateTo; tempDate += 86400) {
      for (uint k = 0; k < daysStorage[tempDate].keywords.length; k++) {
        keywords[keywordsIndex++] = daysStorage[tempDate].keywords[k];
      }
      questions[dayIndex] = daysStorage[tempDate].questions;
      answers[dayIndex] = daysStorage[tempDate].answers;
      noKeywords[dayIndex++] = daysStorage[tempDate].noKeywords;
    }
    return (keywords, questions, answers, noKeywords);
  }
}
```

Αρχικά, το παραπάνω συμβόλαιο χρησιμοποιεί μια βιβλιοθήκη [35] που απλοποιεί την επεξεργασία string, εφόσον η Solidity δεν προσφέρει αντίστοιχη λειτουργία. Ένα μειονέκτημα της χρήσης μιας τέτοιας βιβλιοθήκης είναι τα αυξημένα έξοδα σε gas [36].

Για την αποθήκευση των δεδομένων ανά ημέρα, δημιουργείται μια δομή δεδομένων που περιέχει τις απαραίτητες πληροφορίες και ορίζεται ένα mapping από uint σε αυτή τη δομή. Ο λόγος που χρησιμοποιείται uint είναι επειδή η Solidity δεν υποστηρίζει ημερομηνίες, επομένως όλες οι ημερομηνίες μετατρέπονται σε χρόνο Unix (αριθμός δευτερολέπτων από την 1η Ιανουαρίου 1970). Για να λειτουργεί σωστά το mapping αυτό, οι δύο συναρτήσεις του συμβολαίου επιτρέπουν μόνο ημερομηνίες που έχουν χρόνο *00:00:00*, δηλαδή είναι πολλαπλάσια του 86400, αλλιώς το mapping δε θα γινόταν σωστά και για να γίνει ανάκτηση των πληροφοριών μιας μέρας θα έπρεπε να ελεγχθούν όλα τα δευτερόλεπτα μιας μέρας, το οποίο είναι υπολογιστικά βαρύ, αχρείαστο και οδηγεί σε αυξημένα κόστη σε gas.

Στη συνέχεια, ορίζονται οι δύο συναρτήσεις *addToDay* και *getDays*, που εκτελούν την αποθήκευση πληροφοριών σε μια μέρα και την ανάκτησή τους για μια περίοδο, αντίστοιχα. Καμία από αυτές δεν ορίζεται ως *payable*, ωστόσο η συνάρτηση *addToDay* επεξεργάζεται string και αποθηκεύει πληροφορία, επομένως είναι απαραίτητη η χρήση ETH για την πληρωμή των gas fees. Η συνάρτηση *getDays* μπορεί να χρησιμοποιηθεί χωρίς ETH.

Τέλος, χρησιμοποιείται το Infura για να γίνει deploy στη διεύθυνση *0x292BC609f99b1Fe0bEBf89ed5C6bCB3B27c796dB* [37] και για κλήσεις θα χρησιμοποιηθεί το endpoint που προσφέρεται: https://mainnet.infura.io/v3/515fa7c9a7bc4f2a90bcbca9a0e94ea0.

## 6.2 Blockchain Service

Θα δημιουργηθεί η υπηρεσία Blockchain, που ο σκοπός της θα είναι να προσθέτει πληροφορίες στο blockchain χρησιμοποιώντας το έξυπνο συμβόλαιο. Θα είναι ένα Node project το οποίο θα είναι καταναλωτής Kafka και θα χρησιμοποιεί την πληροφορία που θα καταναλώνει από το service bus για να καλεί τη συνάρτηση *addToDay* του συμβολαίου, επομένως είναι ένα είδος oracle (software, inbound).

Προκειμένου να πραγματοποιηθεί η σωστή σύνδεση και κατανάλωση πληροφορίας από το service bus, θα χρησιμοποιηθεί η τεκμηρίωση SoaML που προηγήθηκε. Συγκεκριμένα, θα γίνει χρήση του *Subscription Service Contract Diagram* (Σχήμα 5.24) και όλων των περιεχομένων του (Σχήματα 5.25, 5.26, 5.27, 5.28), καθώς και η πληροφορία για την υπηρεσία που θα χρησιμοποιηθεί και τον τύπο μηνυμάτων που υποστηρίζονται σε αυτήν, η οποία υπάρχει στο *Enterprise Service Bus Service Interface Diagram* (Σχήμα 5.7). Επίσης, θα χρησιμοποιηθούν τα *Microservices Service Participant Diagram* (Σχήμα 5.8) και *Microservices Services Architecture Diagram* (Σχήμα 5.29) για περαιτέρω κατανόηση της αρχιτεκτονικής του συστήματος και τη θέση που πρέπει να έχει η καινούρια υπηρεσία.

Σύμφωνα με το *Enterprise Service Bus Service Interface Diagram* (Σχήμα 5.7), οι υπηρεσίες που προσφέρονται είναι η *Question Subscription* και η *Answer Subscription*, οι οποίες και θα χρησιμοποιηθούν. Όσον αφορά την αρχιτεκτονική της χρήσης των υπηρεσιών του νέου συμμετέχοντα, θα μοιάζει με το συμμετέχοντα *Browse Questions Microservice* στο *Microservices Service Participant Diagram* (Σχήμα 5.29), δηλαδή θα χρησιμοποιεί τις 2 αυτές υπηρεσίες με τον ίδιο τρόπο. Τα μηνύματα που θα λαμβάνονται είναι τα *KafkaMessage* που περιγράφονται και στο πεδίο *value* περιέχουν τους τύπους μηνυμάτων:

- QuestionMessage, σε περίπτωση χρήσης της Question Subscription
- AnswerMessage, σε περίπτωση χρήσης της Answer Subscription

Όσον αφορά την ένταξη στην αρχιτεκτονική του συστήματος, η νέα υπηρεσία θα μοιάζει με το συμμετέχοντα *Browse Questions* του *Microservices Services Architecture Diagram* (Σχήμα 5.29), δηλαδή θα κάνει χρήση των συμβολαίων *Question Subscription Contract* και *Answer Subscription Contract* για να αλληλεπιδρά με το συμμετέχοντα *Kafka*. Συγκεκριμένα, θα αποκτήσει το ρόλο του *subscriber* που υπάρχει στα συμβόλαια αυτά (Σχήματα 5.24, 5.25, 5.26, 5.27, 5.28).

Γνωρίζοντας τις υπηρεσίες που θα χρησιμοποιηθούν και τα είδη μηνυμάτων που αναμένονται να ληφθούν μέσω αυτών, καθώς και τη συμφωνία που χρειάζεται μεταξύ του νέου συμμετέχοντα και του συμμετέχοντα που αντιπροσωπεύει το service bus, δηλαδή τον τρόπο επικοινωνίας, είναι εφικτή η ενοποίηση της νέας υπηρεσίας στο σύστημα.

Εν τέλει χρησιμοποιείται ο παρακάτω κώδικας για τη σύνδεση στο πορτοφόλι και την κλήση της συνάρτησης *addToDay*:

```
const fs = require('fs');
const contract =
JSON.parse(fs.readFileSync('./truffle/build/contracts/Graphs.json',
'utf8'));

const abi = contract.abi;
const contractAddress = process.env.CONTRACT_ADDRESS

const Web3 = require('web3');
const HDWalletProvider = require('@truffle/hdwallet-provider');
const provider = new HDWalletProvider(process.env.MNEMONIC,
`https://ropsten.infura.io/v3/515fa7c9a7bc4f2a90bcbca9a0e94ea0`);
const web3 = new Web3(provider);
const graphsContract = new web3.eth.Contract(abi, contractAddress);
graphsContract.setProvider(provider);

const subscribe = () => {
  const { Kafka } = require("kafkajs")

  const kafka = new Kafka({
    clientId: "consumer",
    brokers: ['83.212.78.171:9092'],
  })

  const consumer = kafka.consumer({ groupId: "blockchain-service" })

  const run = async() => {
    await consumer.connect()
    await consumer.subscribe({ topic: "QUESTION", fromBeginning: true })
    await consumer.subscribe({ topic: "ANSWER", fromBeginning: true })
    await consumer.run({
      eachMessage: async({ topic, partition, message }) => {
        let jsonMessage = JSON.parse(message.value);
        let date = new Date(parseInt(message.timestamp));
        date.setHours(0, 0, 0, 0);
        date = (date.getTime() / 1000);
        // contract.methods.addToDay(key, elem.keywords, elem.questions,
elem.answers, elem.noKeywords).send({ from: currentAccount })
        if (topic === 'QUESTION') {
          if (jsonMessage.qkeywords === "")
            await graphsContract.methods.addToDay(date, '', 1, 0,
1).send({ from: process.env.ACCOUNT_ADDRESS });
          else
```

```
            await graphsContract.methods.addToDay(date,
jsonMessage.qkeywords, 1, 0, 0).send({ from: process.env.ACCOUNT_ADDRESS
});
          } else {
            await graphsContract.methods.addToDay(date, '', 0, 1, 0).send({
from: process.env.ACCOUNT_ADDRESS });
          }
        }
      })

  run().catch(e => console.error(`[kafka-consumer] ${e.message}`, e))
}

subscribe();
```

Σύμφωνα με τον παραπάνω κώδικα, πραγματοποιείται σύνδεση στο enterprise service bus και συνδρομή στα topics *QUESTION* και *ANSWER*. Στη συνέχεια, ανάλογα με το topic γίνεται κλήση του έξυπνου συμβολαίου με τις σωστές παραμέτρους. Για την κλήση της συνάρτησης έχει γίνει χρήση ενός wallet, το οποίο έχει δημιουργηθεί μέσω Metamask [38] και έχει λάβει ETH από ένα faucet [39] προκειμένου να πληρωθούν τα gas fees. Εφόσον έχει χρησιμοποιηθεί η μυστική φράση του wallet για την υπογραφή των συναλλαγών, δεν είναι απαραίτητη η επιβεβαίωση τους. Το παραπάνω είναι εφικτό με τη χρήση των βιβλιοθηκών:

- web3.js [40]
- @truffle/hdwallet-provider [41]

Εδώ να αξίζει να σημειωθεί πως για παραπάνω ασφάλεια, θα μπορούσε η παραπάνω υπηρεσία να γίνει deploy παραπάνω από μια φορές, έτσι ώστε να μην αποτύχει το σύστημα αν μια από αυτές πέσει και εφόσον ανήκουν στο ίδιο consumer group δεν υπάρχει κίνδυνος διπλών μηνυμάτων.

## 6.3 Graphs Service

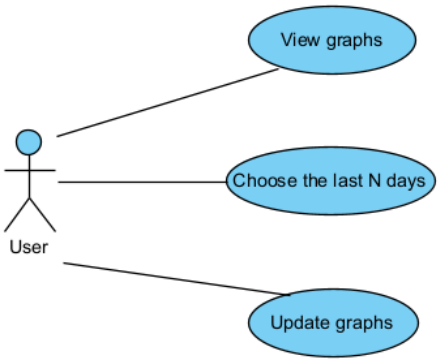
Προκειμένου να είναι πλήρως λειτουργική η υπηρεσία Graphs, θα δημιουργηθεί ένα frontend το οποίο χρησιμοποιεί τις πληροφορίες που υπάρχουν στο blockchain, μέσω κλήσης της συνάρτησης *getDays* του έξυπνου συμβολαίου. Συγκεκριμένα, πρέπει να υλοποιηθούν οι παρακάτω λειτουργίες:

- Οι 5 περισσότερο χρησιμοποιημένες λέξεις-κλειδιά των τελευταίων N ημερών, εφόσον υπάρχουν
- Αριθμός ερωτήσεων ανά ημέρα για τις τελευταίες N ημέρες
- Αριθμός απαντήσεων ανά ημέρα για τις τελευταίες N ημέρες

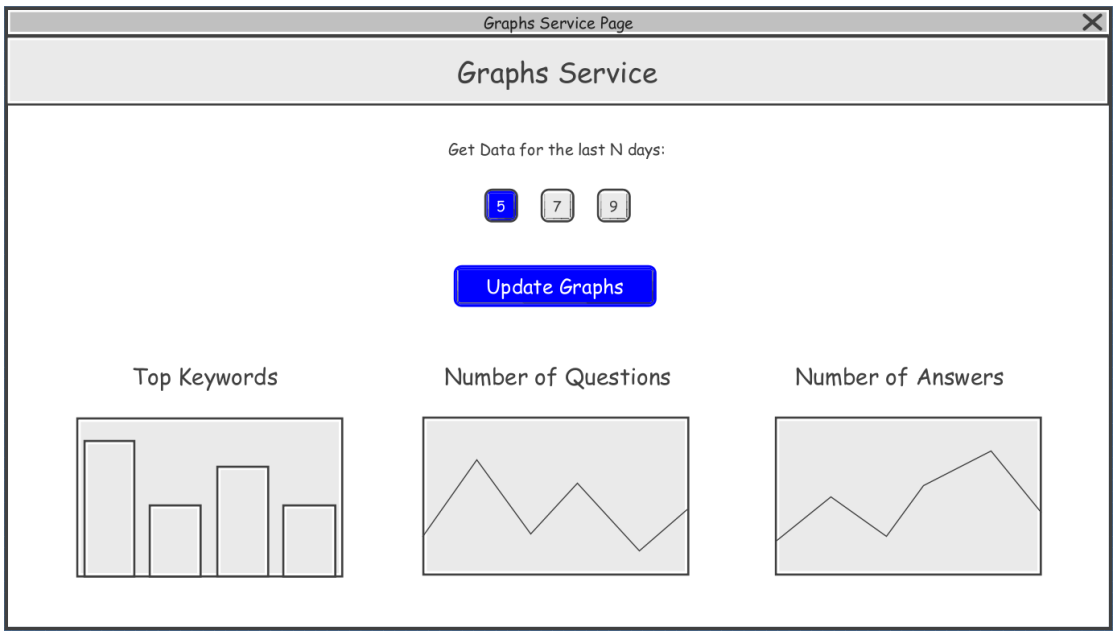
Όπου N = {5, 7, 9}.

Επομένως ο χρήστης πρέπει να έχει τη δυνατότητα:

- Να δει τα γραφήματα
- Να επιλέξει τον αριθμό N
- Να ενημερώσει τα γραφήματα




**Σχήμα 6.1** Graphs Service Use Case Diagram




**Σχήμα 6.2** Graphs Service Wireframe

Το frontend υλοποιείται με React, επομένως παρατίθεται ο κώδικας που ορίζει τα παρακάτω states και οι συναρτήσεις που τα τροποποιούν:

```
const [keywordsData, setKeywordsData] = useState([]);
const [questionsData, setQuestionsData] = useState([]);
const [answersData, setAnswersData] = useState([]);
const [noKeywordsData, setNoKeywordsData] = useState([]);
const [days, setDays] = useState(5);
```

Τα περιεχόμενα των παραπάνω states περιέχουν τις απαραίτητες πληροφορίες για τη δημιουργία των σωστών γραφημάτων και θα τροποποιούνται κάθε φορά που γίνεται κλήση της συνάρτησης *getDays* του έξυπνου συμβολαίου, με εξαίρεση το state *days*, που ορίζει τον αριθμό των τελευταίων ημερών για τις οποίες πρέπει να ληφθούν πληροφορίες και επιλέγεται από το χρήστη.

Παρατίθεται ο κώδικας που εκτελεί τη σύνδεση στο wallet, την κλήση στη συνάρτηση *getDays* του έξυπνου συμβολαίου και την ανάθεση των πληροφοριών που λήφθηκαν στα υπάρχοντα states:

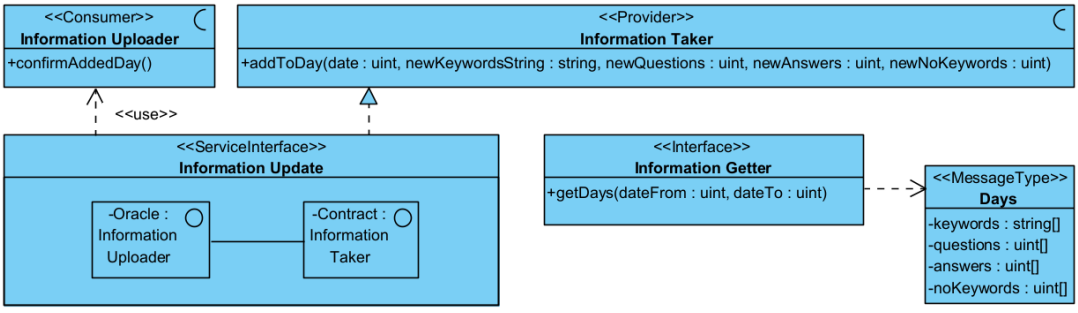
```
const abi = contract.abi;
const contractAddress =
"0x292BC609f99b1Fe0bEBf89ed5C6bCB3B27c796dB"

const provider = new HDWalletProvider(process.env.MNEMONIC,
`https://ropsten.infura.io/v3/515fa7c9a7bc4f2a90bcbca9a0e94ea0`)
const web3 = new Web3(provider);
const graphsContract = new web3.eth.Contract(abi, contractAddress);
graphsContract.setProvider(provider);

const today = new Date();
today.setHours(0, 0, 0, 0);
const todayInSeconds = Math.floor(today / 1000) + (86400 / 8);
const previousDate = todayInSeconds - ((days - 1) * 86400);

let result = await graphsContract.methods.getDays(previousDate,
todayInSeconds).call();

setKeywordsData(result.keywords);
setQuestionsData(result.questions);
setAnswersData(result.answers);
setNoKeywordsData(result.noKeywords);
```

Ομοίως με πριν, γίνεται σύνδεση σε ένα wallet με χρήση της μυστικής φράσης του, προκειμένου να μη χρειάζεται επιβεβαίωση κάθε συναλλαγή, αλλά ούτε και σύνδεση μέσω Metamask. Επιπλέον, επειδή γίνεται κλήση συνάρτησης για λήψη δεδομένων, αυτήν τη φορά δε χρειάζονται ETH για να χρησιμοποιηθούν ως gas fees.

## 6.4 Τεκμηρίωση με SoaML

Τέλος, θα γίνει τεκμηρίωση των νέων υπηρεσιών με SoaML. Η υπηρεσία Graphs Service δε θα επεκταθεί, ούτε θα δημιουργηθούν νέες υπηρεσίες οι οποίες να χρησιμοποιούν το έξυπνο συμβόλαιο. Ωστόσο, για λόγους πληρότητας, θα γίνει τεκμηρίωση με SoaML με τη δημιουργία καινούριων διαγραμμάτων και την επέκταση των υπαρχόντων.

Αρχικά δημιουργείται το παρακάτω διάγραμμα:




**Σχήμα 6.3** Graphs Service Interface Diagram

Έχουν δημιουργηθεί το εξής ServiceInterface:

- Information Update: υπηρεσία για προσθήκη πληροφορίας στο blockchain.

Στη συνέχεια, ορίζονται τρία απλά Interfaces:

- Information Taker: διεπαφή που υλοποιεί τη λειτουργία *addToDay* και χρησιμοποιείται από τον παραλήπτη των νέων πληροφοριών. Εφόσον ορίζει τη συμπεριφορά και το ServiceInterface την πραγματοποιεί, έχουν τη σχέση *Realization*. Αποτελεί τον πάροχο της υπηρεσίας.
- Information Uploader: διεπαφή που υλοποιεί τη λειτουργία *confirmAddedDay* και χρησιμοποιείται από την υπηρεσία που μεταφορτώνει νέες πληροφορίες. Εφόσον το ServiceInterface πρέπει να υποστηρίζει τη λειτουργία αυτή, έχουν τη σχέση *Usage*. Αποτελεί τον καταναλωτή της υπηρεσίας.
- Information Getter: διεπαφή που υλοποιεί τη λειτουργία *getDays*. Εφόσον ορίζει υπηρεσία μια μονόδρομη επικοινωνία δε χρειάζεται να χρησιμοποιηθεί κάποιο ServiceInterface.

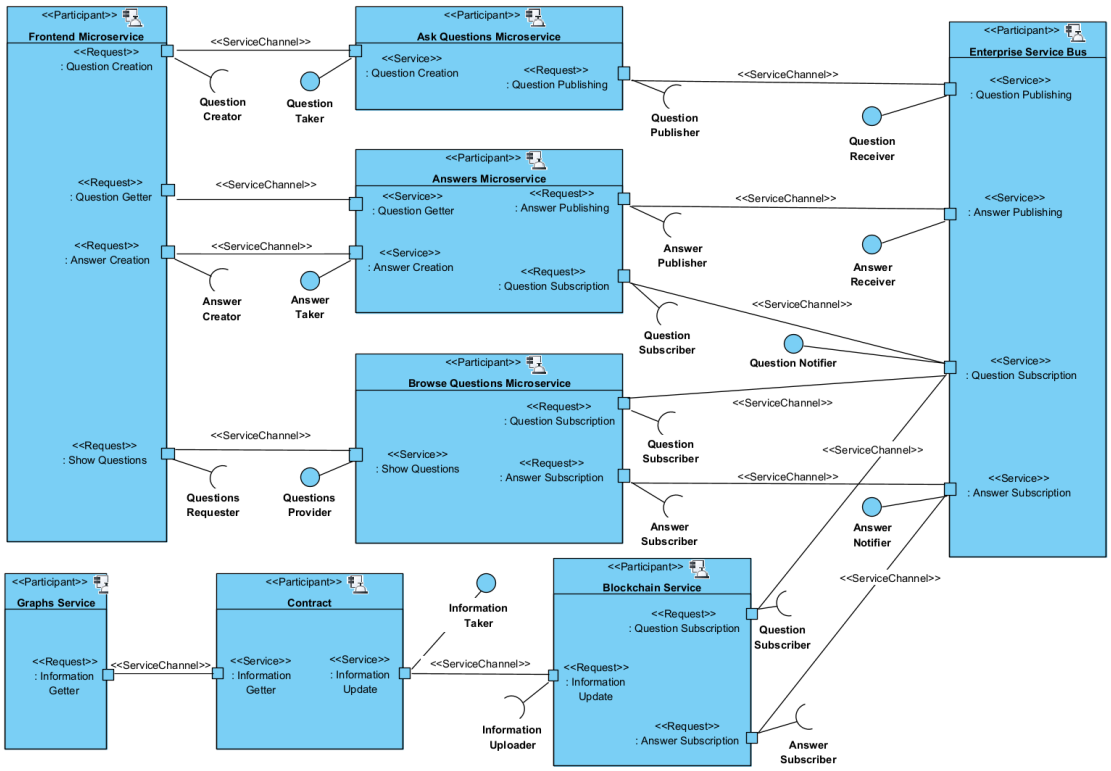
Στη συνέχεια ορίζονται οι παρακάτω δύο Roles στο ServiceInterface:

- Oracle: το service που μεταφορτώνει νέες πληροφορίες, για αυτό του προσδίδεται και ο τύπος “Information Uploader” από τις διεπαφές που έχουν οριστεί.
- Contract: το service που λαμβάνει νέες πληροφορίες, για αυτό του προσδίδεται και ο τύπος “Information Taker” από τις διεπαφές που έχουν οριστεί.

Έπειτα ορίζονται ένας MessageType, ο οποίος είναι κρίσιμος για την επικοινωνία μεταξύ των υπηρεσιών, για αυτό και χρησιμοποιείται η σχέση *Dependency* για τη σύνδεσή του με τη λειτουργία *getDays* της διεπαφής.

- Days: ο τύπος μηνύματος αυτός περιέχει τέσσερα πεδία: το *keywords*, το οποίο είναι τύπου string array και τα *questions*, *answers* και *noKeywords*, τα οποία είναι τύπου uint array.

Στη συνέχεια επεκτείνεται το προηγούμενο Service Participant Diagram:




**Σχήμα 6.4** Graphs Service Participant Diagram

Ορίζονται τρεις καινούριοι Participants στο σύστημα:

- Blockchain Service
- Contract
- Graphs Service

Για κάθε Participant ορίζονται οι θύρες οι οποίες υλοποιούν τα ServiceInterfaces που ορίστηκαν στα Service Interface Diagrams και χρησιμοποιούνται προκειμένου να παρέχονται και να καταναλώνονται υπηρεσίες, καθώς και τα απαραίτητα Interface Realizations και Usages που υλοποιούν τις αντίστοιχες διεπαφές:

Για τον Graphs Service Participant ορίζεται το παρακάτω:

- Request Port τύπου “Information Getter”: υλοποιεί την κατανάλωση της υπηρεσίας που περιγράφει η διεπαφή ‘Information Getter’.

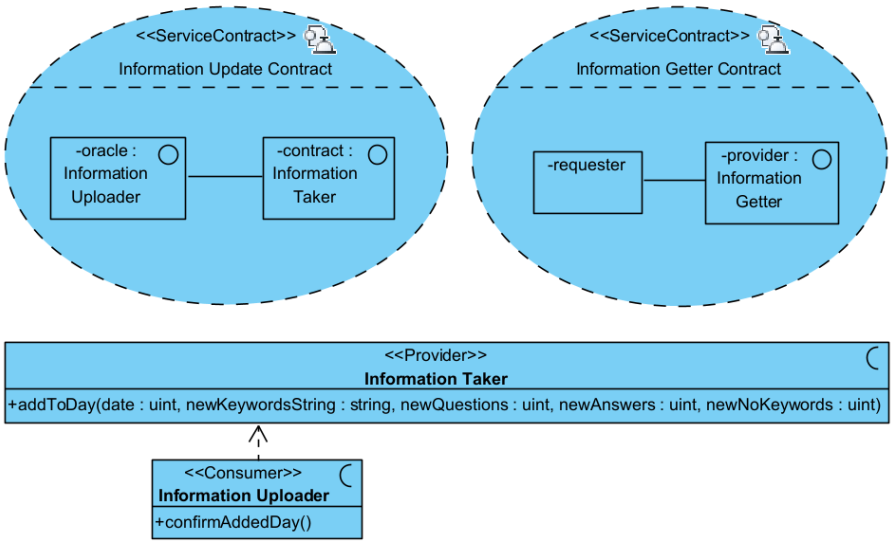
Για τον Contract Participant ορίζονται τα παρακάτω:

- Service Port τύπου “Information Getter”: υλοποιεί την παροχή της υπηρεσίας που περιγράφει η διεπαφή ‘Information Getter’.
- Service Port τύπου “Information Update”: υλοποιεί την παροχή της υπηρεσίας που περιγράφει το ServiceInterface “Information Update”. Μέσω της θύρας αυτής χρησιμοποιείται ένα *Realization* το οποίο υλοποιεί τη διεπαφή “Information Taker”.

Για τον Blockchain Service Participant ορίζονται τα παρακάτω:

- Request Port τύπου “Information Update”: υλοποιεί την κατανάλωση της υπηρεσίας που περιγράφει το ServiceInterface “Information Update”. Μέσω της θύρας αυτής χρησιμοποιείται ένα *Usage* το οποίο υλοποιεί τη διεπαφή “Information Uploader”.
- Request Port τύπου “Question Subscription”: υλοποιεί την κατανάλωση της υπηρεσίας που περιγράφει το ServiceInterface “Question Subscription”. Μέσω της θύρας αυτής χρησιμοποιείται ένα *Usage* το οποίο υλοποιεί τη διεπαφή “Question Subscriber”.
- Request Port τύπου “Answer Subscription”: υλοποιεί την κατανάλωση της υπηρεσίας που περιγράφει το ServiceInterface “Answer Subscription”. Μέσω της θύρας αυτής χρησιμοποιείται ένα *Usage* το οποίο υλοποιεί τη διεπαφή “Answer Subscriber”.

Στη συνέχεια, δημιουργείται το παρακάτω Service Contract Diagram:




**Σχήμα 6.5** Graphs Service Contract Diagram

Το παραπάνω διάγραμμα ορίζει δύο ServiceContracts, ένα για την υπηρεσία προσθήκης πληροφορίας (Information Update Contract) και ένα για τη λήψη πληροφοριών (Information Getter Contract).

Στο Information Update Contract, ορίζονται δύο ρόλοι με τύπους που ορίζονται ως διεπαφές στο Graphs Service Interface Diagram:

- oracle, με τύπο Information Uploader
- contract, με τύπο Information Taker

Στο Information Getter Contract, ορίζονται δύο ρόλοι, εκ των οποίων μόνο ένας έχει τύπο που ορίζεται ως διεπαφή στο Graphs Service Interface Diagram, εφόσον περιγράφει μονόδρομη επικοινωνία:

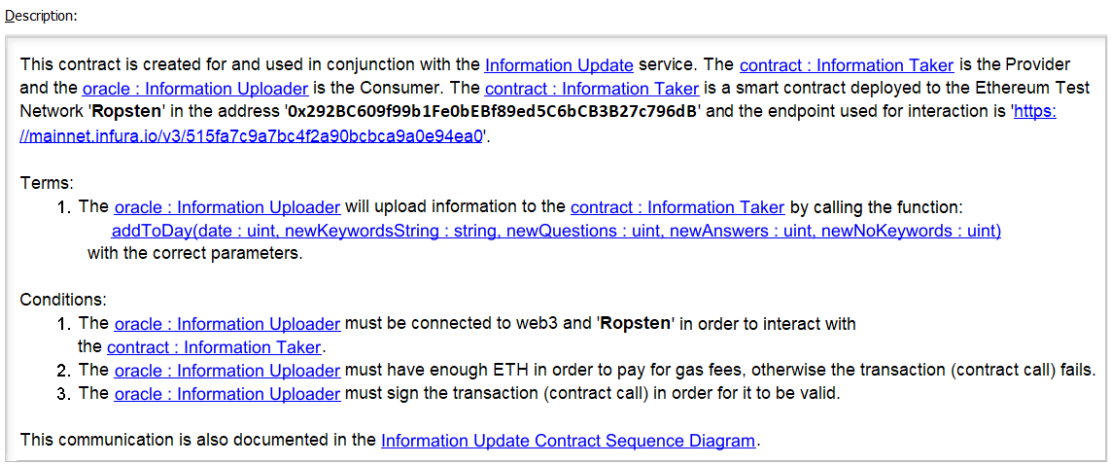
- requester
- provider, με τύπο Information Getter

Επίσης, για το Graphs Information Update Contract που δημιουργήθηκε για ServiceInterface, δημιουργείται ένας Provider και ένας Consumer που αντιστοιχούν στους ρόλους που έχουν οριστεί:

- Provider: Information Taker
- Consumer: Information Uploader

Εφόσον εκείνος που εκκινεί την επικοινωνία είναι ο Consumer, δημιουργείται ένα Dependency από τον Consumer στον Provider.

Ακολουθούν οι περιγραφές και τα Sequence Sub Diagrams των παραπάνω ServiceContracts. Οι περιγραφές ορίζουν τον τρόπο επικοινωνίας μεταξύ των συμμετεχόντων στο συμβόλαιο, κάνοντας τις απαραίτητες αναφορές στον τύπο μηνύματος που ορίστηκε στο Graphs Service Interface Diagram και τα περιεχόμενά του, έτσι ώστε να υλοποιηθούν οι ζητούμενες λειτουργίες. Τα Sequence Diagrams δημιουργούνται ως Sub Diagrams των αντίστοιχων συμβολαίων και επιδεικνύουν την αλληλεπίδραση μεταξύ των ρόλων που έχουν οριστεί.


Description:

This contract is created for and used in conjunction with the Information Update service. The contract : Information Taker is the Provider and the oracle : Information Uploader is the Consumer. The contract : Information Taker is a smart contract deployed to the Ethereum Test Network '**Ropsten**' in the address '**0x292BC609f99b1Fe0bEBf89ed5C6bCB3B27c796dB**' and the endpoint used for interaction is 'https://mainnet.infura.io/v3/515fa7c9a7bc4f2a90bcbca9a0e94ea0'.

Terms:

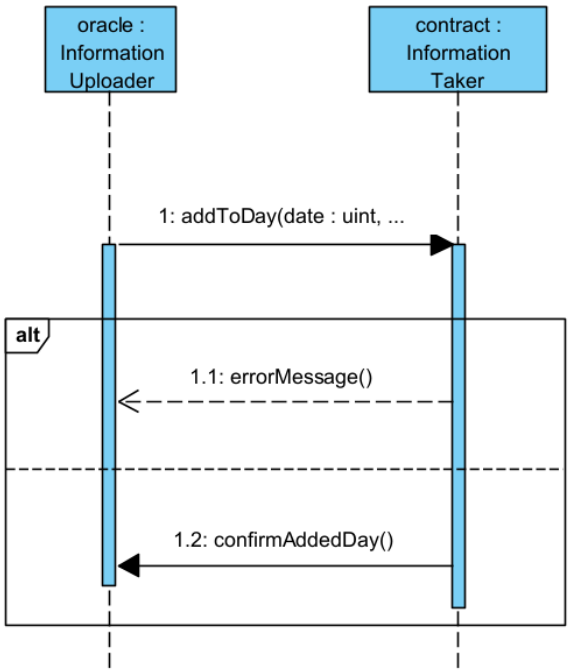
1. The oracle : Information Uploader will upload information to the contract : Information Taker by calling the function:
   addToDay(date : uint, newKeywordsString : string, newQuestions : uint, newAnswers : uint, newNoKeywords : uint)
   with the correct parameters.

Conditions:

1. The oracle : Information Uploader must be connected to web3 and '**Ropsten**' in order to interact with the contract : Information Taker.
2. The oracle : Information Uploader must have enough ETH in order to pay for gas fees, otherwise the transaction (contract call) fails.
3. The oracle : Information Uploader must sign the transaction (contract call) in order for it to be valid.

This communication is also documented in the Information Update Contract Sequence Diagram.

**Σχήμα 6.6** Information Update Contract Description




**Σχήμα 6.7** Information Update Contract Sequence Diagram

Description:

This contract is created for and used in conjunction with the Information Getter service. The provider : Information Getter is a smart contract deployed to the Ethereum Test Network '**Ropsten**' in the address '**0x292BC609f99b1Fe0bEBf89ed5C6bCB3B27c796dB**' and the endpoint used for interaction is 'https://mainnet.infura.io/v3/515fa7c9a7bc4f2a90bcbca9a0e94ea0'.

Terms:

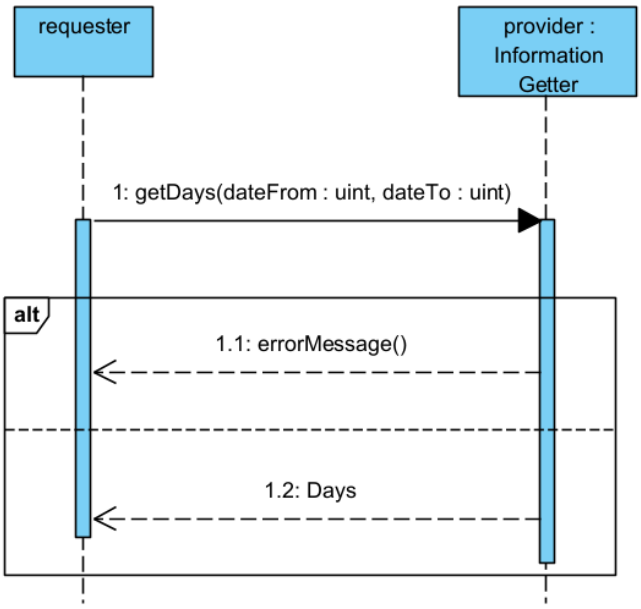
1. The requester will get information from the provider : Information Getter by calling the function: getDays(dateFrom : uint, dateTo : uint) with the correct parameters.

Conditions:

1. The requester must be connected to web3 and '**Ropsten**' in order to interact with the provider : Information Getter.
2. The requester must sign the transaction (contract call) in order for it to be valid.

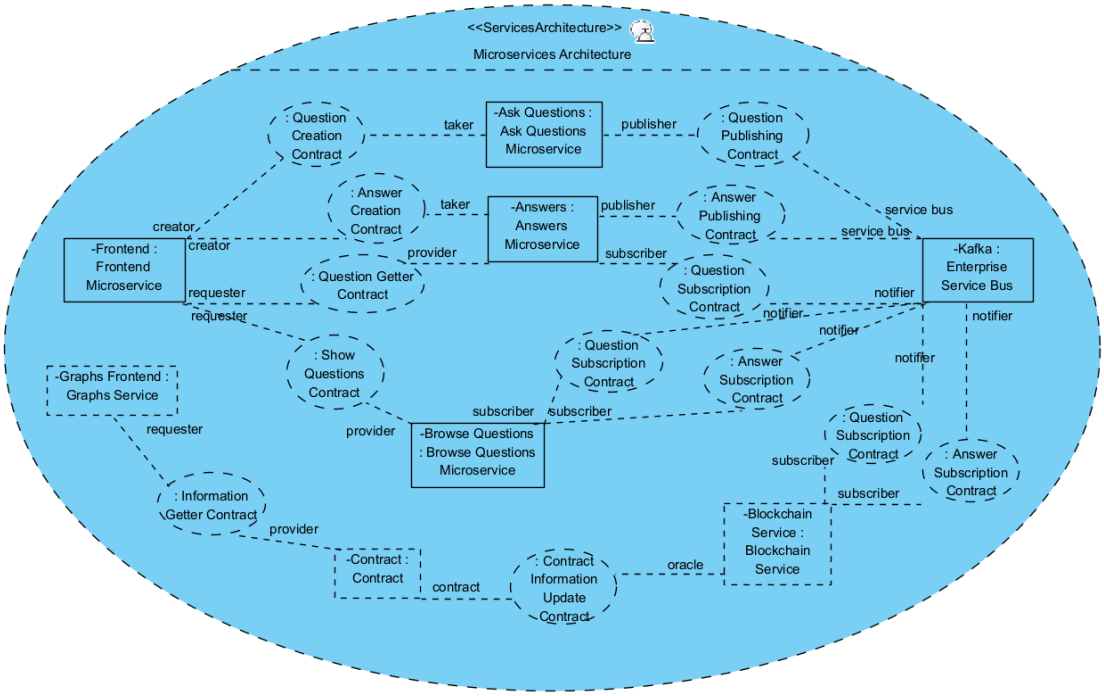
This communication is also documented in the Information Getter Contract Sequence Diagram.

**Σχήμα 6.8** Information Getter Contract Description




**Σχήμα 6.9** Information Getter Contract Sequence Diagram

Στη συνέχεια, επεκτείνεται το Services Architecture Diagram, έτσι ώστε να περιλαμβάνει και τους νέους Participants:




**Σχήμα 6.10** Graphs Service Services Architecture Diagram

Με βάση τους Participants που έχουν οριστεί στο Graphs Service Participant Diagram, δημιουργούνται 3 νέοι External Participants:

- Graphs Frontend, με τύπο Graphs Service

- Contract, με τύπο Contract
- Blockchain Service, με τύπο Blockchain Service

Παρακάτω αναλύονται τα collaboration μεταξύ τους:

Μεταξύ του Graphs Frontend και του Contract

- Γίνεται χρήση του *Information Getter Contract*, με το Graphs Frontend να παίρνει το ρόλο του requester και το Contract να παίρνει το ρόλο του provider.

Μεταξύ του Contract και του Blockchain Service

- Γίνεται χρήση του *Contract Information Update Contract*, με το Contract να παίρνει το ρόλο του contract και το Blockchain Service να παίρνει το ρόλο του oracle.

Μεταξύ του Blockchain Service και του Kafka

- Γίνεται χρήση του *Question Subscription Contract*, με το Blockchain Service να παίρνει το ρόλο του subscriber και το Kafka να παίρνει το ρόλο του notifier.
- Γίνεται χρήση του *Answer Subscription Contract*, με το Blockchain Service να παίρνει το ρόλο του subscriber και το Kafka να παίρνει το ρόλο του notifier.

*Κεφάλαιο 7*

# Συμπεράσματα και μελλοντική εργασία

## 7.1 Συμπεράσματα

Η SoaML χρησιμοποιείται για την περιγραφή υπηρεσιών και του τρόπου επικοινωνίας μεταξύ των συμμετεχόντων αυτών. Δεν προορίζεται για την αναλυτική περιγραφή συστημάτων ή των μερών από τα οποία απαρτίζονται αυτά. Συνεπώς, για μια καλύτερη περιγραφή ενός συστήματος, είναι απαραίτητη και η χρήση άλλων διαγραμμάτων (Component, Sequence, Deployment, κλπ.).

Μέσα από αυτήν την εργασία έγινε εμφανές πως είναι δυνατή η χρήση SoaML προκειμένου να γίνει η τεκμηρίωση ενός συστήματος με τέτοιον τρόπο έτσι ώστε να απλοποιείται η ενοποίηση με ένα άλλο σύστημα. Εφόσον γίνει σωστή περιγραφή των υπηρεσιών που παρέχονται για την επικοινωνία εντός του συστήματος (στην προκειμένη περίπτωση το Enterprise Service Bus), καθίσταται εφικτή η ανάπτυξη συστήματος που να καταναλώνει τις υπηρεσίες αυτές.

## 7.2 Μελλοντική εργασία

Στην εργασία αυτή δε μελετήθηκε πολύπλοκο σύστημα αρχιτεκτονικής SOA, εφόσον δεν ήταν αυτός ο σκοπός. Θα ήταν, ωστόσο, χρήσιμη η μελέτη ενός πιο σύνθετου συστήματος, προκειμένου να διερευνηθεί πόσο απλοποιείται η επικοινωνία των χαρακτηριστικών του και επομένως η κατανόησή του, προκειμένου να γίνει πιο ομαλή η εισαγωγή ενός νέου συστήματος σε αυτό.

Ιδιαίτερα ενδιαφέρουσα επίσης φαίνεται η τεκμηρίωση ενός έξυπνου συμβολαίου με SoaML, δεδομένου του ότι οι συναρτήσεις που περιέχει ουσιαστικά αποτελούν υπηρεσίες. Εφόσον εντός ενός ServiceContract μπορεί να γίνει περιγραφή της επικοινωνίας μεταξύ των συμμετεχόντων σε μια υπηρεσία, θα μπορούσε να γίνει τεκμηρίωση ενός έξυπνου συμβολαίου με περισσότερες δυνατότητες και εφόσον θεωρηθεί υπάρχον (legacy) σύστημα, να αναπτυχθεί μια εφαρμογή που να καταναλώνει τις υπηρεσίες αυτού.

# Βιβλιογραφία


[1] SoaML Specification by OMG. https://www.omg.org/spec/SoaML/1.0.1/PDF

[2] Visual Paradigm. https://www.visual-paradigm.com/

[3] How to draw SoaML diagrams. https://www.visual-paradigm.com/tutorials/how-to-draw-soaml-diagrams.jsp

[4] What is a blockchain? https://www.investopedia.com/terms/b/blockchain.asp

[5] What is a block? https://www.investopedia.com/terms/b/block-bitcoin-block.asp

[6] What is a Consensus Mechanism? https://www.investopedia.com/terms/c/consensus-mechanism-cryptocurrency.asp

[7] Proof of Work. https://www.investopedia.com/terms/p/proof-work.asp

[8] Proof of Stake. https://www.investopedia.com/terms/p/proof-stake-pos.asp

[9] Bitcoin Whitepaper. https://bitcoin.org/bitcoin.pdf

[10] Smart Contracts. https://www.investopedia.com/terms/s/smart-contracts.asp

[11] Ethereum Whitepaper. https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf

[12] Solidity. https://docs.soliditylang.org/en/v0.8.15/

[13] Web 1.0/2.0/3.0. https://www.geeksforgeeks.org/web-1-0-web-2-0-and-web-3-0-with-their-difference/

[14] Oracles. https://ethereum.org/en/developers/docs/oracles/

[15] Oracle Types. https://blockchainhub.net/blockchain-oracles/

[16] Heroku. https://www.heroku.com/

[17] Okeanos. https://okeanos.grnet.gr/home/

[18] Truffle Suite. https://trufflesuite.com/

[19] Truffle. https://trufflesuite.com/truffle/

[20] Ganache. https://trufflesuite.com/ganache/

[21] Infura. https://infura.io/

[22] Ropsten. https://www.anyblockanalytics.com/networks/ethereum/ropsten/

[23] NodeJS. https://nodejs.org/

[24] Node Package Manager. https://www.npmjs.com/

[25] ReactJS. https://reactjs.org/

[26] PostgreSQL. https://www.postgresql.org/

[27] Apache Kafka. https://kafka.apache.org/

[28] CORS. https://github.com/expressjs/cors

[29] ExpressJS. https://github.com/expressjs/express

[30] KafkaJS. https://github.com/tulios/kafkajs

[31] Nodemon. https://github.com/remy/nodemon

[32] Node-Postgres. https://github.com/brianc/node-postgres

[33] Sequelize. https://github.com/sequelize/sequelize

[34] Github Repository. https://github.com/ntua/SOA2Blockchain

[35] Solidity String Library. https://github.com/willitscale/solidity-util/blob/master/lib/Strings.sol

[36] Ethereum Gas. https://ethereum.org/en/developers/docs/gas/

[37] Graphs Contract. https://ropsten.etherscan.io/address/0x292BC609f99b1Fe0bEBf89ed5C6bCB3B27c796dB

[38] Metamask. https://metamask.io/

[39] Ethereum Faucet. https://faucet.metamask.io/

[40] Web3.js. https://github.com/ChainSafe/web3.js

[41] @truffle/hdwallet-provider. https://github.com/trufflesuite/truffle/tree/develop/packages/hdwallet-provider