ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ Ψηφιακής Τεχνολογίας

ΤΜΗΜΑ Πληροφορικής και Τηλεματικής

Τίτλος Εργασίας

Σύστημα Διαχείρησης Απομακρυσμένων Κλήσεων Διαδικασιών

Πτυχιακή εργασία

Τσαγανός Απόστολος

Αθήνα, 2017

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ Ψηφιακής Τεχνολογίας

ΤΜΗΜΑ Πληροφορικής και Τηλεματικής

Τριμελής Εξεταστική Επιτροπή

Τσερπές Κωνσταντίνος (Επίκουρος Καθηγητής - Επιβλέπων)
Νικολαΐδη Μαρία (Καθηγήτρια - 2^{ο} Μέλος)
Μιχαήλ Δημήτριος (Επίκουρος Καθηγητής - 3^{ο} Μέλος)

Ο Τσαγανός Απόστολος

δηλώνω υπεύθυνα ότι:

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

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

Γνωμικό

“If you learn only methods, you’ll be tied to your methods. But if you learn principles, you can devise your own methods.” —Ralph Waldo Emerson

“Αν μάθετε μόνο μεθόδους, θα δεσμευτείτε από αυτές. Αλλά αν μάθετε αρχές, μπορείτε να επινοήσετε τις δικές σας μεθόδους.” —Ralph Waldo Emerson

Ευχαριστίες

Θέλω να ευχαριστήσω θερμά τους καθηγητές μου, κ. Τσαδήμα και κ. Τσερπέ, για τον χρόνο που διέθεσαν και τις εποικοδομητικές συζητήσεις που έκαναν μαζί μου και με βοήθησαν να σκεφτώ πράγματα τα οποία δεν είχα σκεφτεί μέχρι στιγμής, αλλά και για την καθοδήγηση που μου έδωσαν. Επίσης, θέλω να ευχαριστήσω και τα υπόλοιπα μέλη της εξεταστικής επιτροπής που ανέλαβαν την αξιολόγηση της πτυχιακής μου εργασίας, την Πρύτανη του πανεπιστημίου μας κ. Νικολαΐδη και τον κ. Μιχαήλ.

Σας ευχαριστώ πολύ!

Τέλος, θέλω να ευχαριστήσω τον συμφοιτητή μου, Γκουλή Δημήτρη, για όλες τις συζητήσεις σχετικά με την σύλληψη του concept του Project αλλά και κάποιες ιδέες που μου έδωσε οι οποίες έκαναν τελικά το Project ακόμα πιο ενδιαφέρον από ότι νόμιζα.

Σε ευχαριστώ!

1 Περιεχόμενο

Περίληψη

Στις μέρες μας, οι περισσότεροι οργανισμοί και επιχειρήσεις απαρτίζονται από πολλούς τομείς (τμήματα) από τους οποίους, ο κάθε ένας ασχολείται με τις δικιές του αρμοδιότητες. Όταν όμως όλα τα τμήματα έχουν υλοποιήσει όλες τις λειτουργίες που τους χρειάζονται και θέλουν να αφήσουν τρίτους να έχουν περιορισμένη πρόσβαση σε αυτά παρουσιάζεται το πρόβλημα, αφού για να πάρει κάποιος τρίτος δεδομένα και να χρησιμοποιήσει υπηρεσίες από αύτον τον οργανισμό-επιχείρηση πρέπει να εμπλέκεται ένας προγραμματιστής και να δίνει συγκεκριμένα περιορισμένη πρόσβαση κάθε φορά σε όποιον το έχει ανάγκη. Το πρόβλημα αυτό το λύνει το RPCMS καθώς μπορούμε να φτιάξουμε μέσω αυτού έναν «κατάλογο» από τις υπηρεσίες που παρέχουμε ακόμα κι αν προέρχονται από διαφορετικούς κομβους. Επίσης, μπορούμε πάλι μέσω του RPCMS να ορίσουμε σε ποιές υπηρεσίες μπορεί να έχει πρόσβαση ο κάθε χρήστης και όλα τα παραπάνω μέσω ενός UI και όχι μέσω κώδικα. Τέλος, το RPCMS δίνει τη δυνατότητα σε κάποιον να κάνει log τις κλήσεις που δέχονται τα συστήματα που περιέχονται σε αυτό, αλλά και να απαγορέψει την πρόσβαση σε συγκεκριμένες IP διευθύνσεις.

Abstract

Nowadays, most organizations and enterprises are composed by many domains (departments) from which, each of those is dealing with its own responsibilities. But when every department has implemented all of their own operations that they need and they want to let other third-party to gain restricted access to them, the problem arises since in order for a third-party to retrieve data and use those operations from this organization-enterprise a programmer must be involved and permit restricted access to anyone who needs it every single time. In these situations RPCMS comes to rescue as we can create a “list” of the services that we provide even if they come from multiple different nodes-domains. Also, we can again using the RPCMS define which services each user can access and all of the above through a UI and not through coding. Finally, using the RPCMS someone has the ability to log the API calls which are included in the RPCMS but also deny access to specific IP addresses.

2 Κατάλογος Εικόνων

1 Εισαγωγή

1.1 Επιστημονικό Υπόβαθρο

RPCMS (Remote Procedure Call Management System) ή αλλιώς Σύστημα Διαχείρισης Απομακρυσμένων Κλήσεων Διαδικασιών, είναι ένα σύστημα το οποίο καλεί απομακρυσμένες λειτουργίες από APIs εφαρμογών άσχετα με το που αυτές βρίσκονται (ίδιο domain ή διαφορετικό). Έτσι λοιπόν, είναι ένας ενδιάμεσος μεταξύ χρήστη και εφαρμογής ο οποίος μπορεί να παρέχει μια πληθώρα εφαρμογών σαν ένα πακέτο υπηρεσιών. Επίσης, μπορεί να δώσει την δυνατότητα ανάπτυξης μιας ενιαίας εφαρμογής άσχετα με το πως κάθε κομμάτι της υλοποιεί την λογική του.

Επιπλέον, θα μπορούσε να χαρακτηριστεί ως ένα υποσύστημα, που έχει τη δυνατότητα να προσαρμόζεται σε οποιοδήποτε άλλο προϋπάρχον σύστημα. Ένας οποιοσδήποτε π.χ. μη καταρτισμένος τεχνικά υπάλληλος μπορεί να το χειρίζεται, επιλέγοντας ο ίδιος για κάθε άλλο χρήστη ποιες λειτουργίες θα του παρέχει και ποιές όχι. Λέγοντας "χρήστη" εννοούμε είτε ένα φυσικό πρόσωπο, είτε μια εταιρία ή ακόμα και μια άλλη εφαρμογή.

Χρησιμοποιώντας τεχνικούς όρους θα μπορούσε κάποιος να πει ότι συνδυάζει τις λειτουργίες ενός Reverse Proxy Server αλλά και ενός Service Aggregator. Προς το παρόν, μόνο το κομμάτι της ασφάλειας από τον όρο του Reverse Proxy Server παρέχεται από το RPCMS αλλά μελλοντικά έχει τεθεί ο στόχος να επεκταθεί ώστε να παρέχει όλα τα χαρακτηριστικά γνωρίσματα του.

Ας ξεκινήσουμε παρουσιάζοντας την πιο γενική αρχιτεκτονική Service-Oriented Architecture (SOA) ή αλλιώς Υπηρεσιοστραφής Αρχιτεκτονική.

1.1.1 Service-Oriented Architecture

Η Υπηρεσιοστραφής Αρχιτεκτονική είναι μια προσέγγιση η οποία χρησιμοποιείται για την δημιουργία συστημάτων τα οποιά αποτελούνται από χαλαρά συνδεδεμένα υποσυστήματα. Η λειτουργικότητα των τελευταίων παρέχεται ως υπηρεσία μέσα από πλήρως καθορισμένες διεπαφές. Οι υπηρεσίες αυτές (π.χ. RESTful Web Services) πραγματοποιούν κατά κανόνα

κάποιες μικρές λειτουργίες, π.χ. παραγωγή δεδομένων, επικύρωση εισόδου χρήστη ή την παροχή απλών αναλυτικών υπηρεσιών. Ως αποτέλεσμα, η Υπηρεσιοστραφής Αρχιτεκτονική επιτρέπει την επαναχρησιμοποίηση υπηρεσιών, κάνοντας το περιττό να ξεκινήσεις από την αρχή όταν χρειάζονται αναβαθμίσεις ή αλλαγές. Αυτό είναι ένα όφελος για τις επιχειρήσεις που αναζητούν τρόπους για να εξοικονομήσουν χρόνο και χρήμα όπως θα δούμε παρακάτω για το time-to-market.

Εκτός από την παραγωγή και έκθεση υπηρεσιών, η Υπηρεσιοστραφής Αρχιτεκτονική έχει τη δυνατότητα να αξιοποιήσει τις υπηρεσίες αυτές ξανά και ξανά μέσα από εφαρμογές (γνωστές και ως σύνθετες εφαρμογές - composite applications). Η Υπηρεσιοστραφής Αρχιτεκτονική δεσμεύει αυτές τις υπηρεσίες ώστε να τις οργανώσει, ή τις αξιοποιεί μεμονωμένα. Έτσι, η Υπηρεσιοστραφής Αρχιτεκτονική έχει ως στόχο να διορθώσει υπάρχουσες αρχιτεκτονικές αντιμετωπίζοντας τα περισσότερα από τα μεγάλα συστήματα ως ένα σύνολο υπηρεσιών, αλλά και αποσπώντας αυτές τις υπηρεσίες σε ένα ενιαίο τομέα, όπου εκεί συνδυάζονται.

Η Υπηρεσιοστραφής Αρχιτεκτονική είναι γνωστό ότι παρέχει τόσο time-to-market πλεονεκτήματα, όσο και ευελιξία σε μια επιχείρηση. Η χρήση των μηχανών ενορχήστρωσης, ή η αξιοποίηση του περιβάλλοντος ανάπτυξης το οποίο αξιοποιεί τις υπηρεσίες και την αρχιτεκτονική αυτή, επιτρέπουν την πολύ γρήγορη δημιουργία εφαρμογών, δεδομένου ότι οι υπηρεσίες παρέχουν πολλά από αυτά που απαιτεί η εφαρμογή. Αυτό παρέχει το πλεονέκτημα του time-to-market.

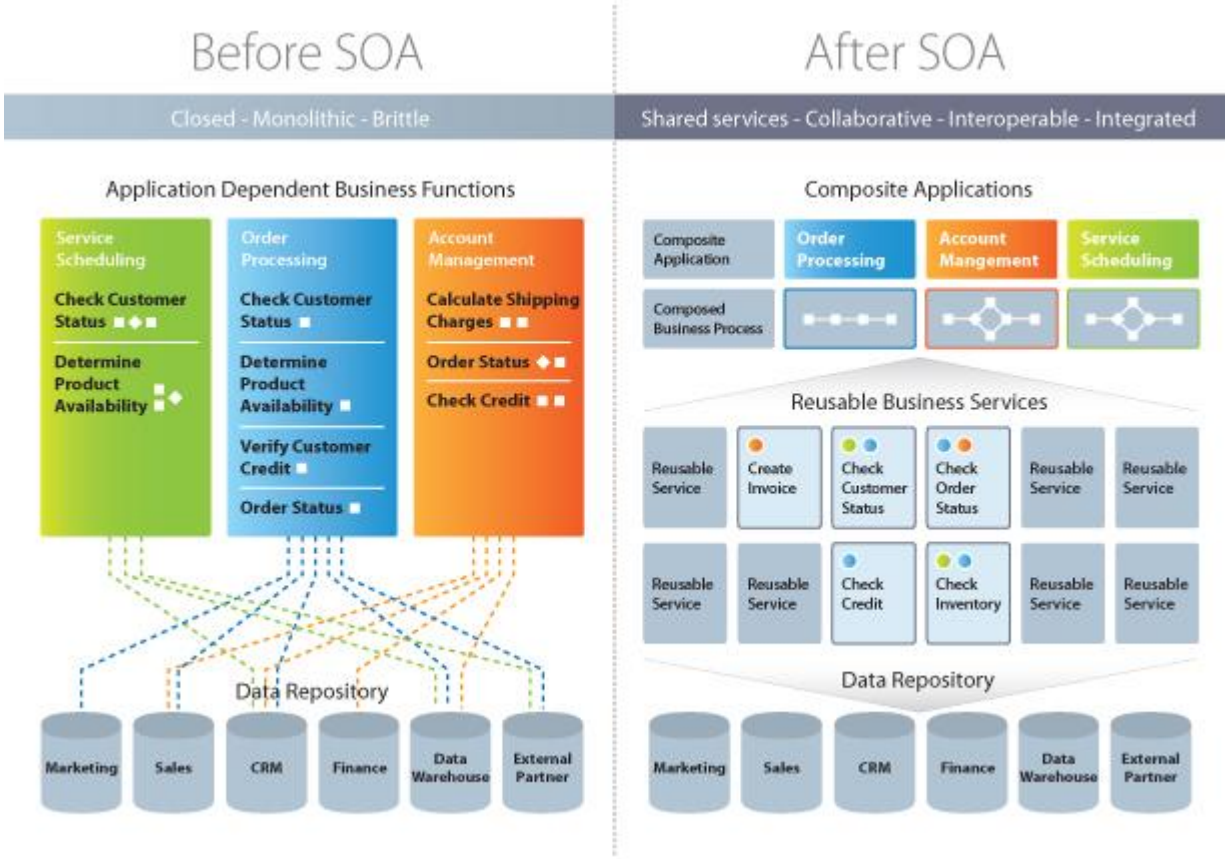
Η διάθεση μεταβλητότητας σε έναν τομέα (όπως η χρήση μιας μηχανής ενορχήστρωσης) επιτρέπει εφαρμογές κτισμένες πάνω στην Υπηρεσιοστραφή Αρχιτεκτονική να προσαρμόζονται γρήγορα γύρω από τις μεταβαλλόμενες απαιτήσεις των επιχειρήσεων. Σε πολλές περιπτώσεις, αυτό είναι απλώς ένα θέμα του εκ νέου προσδιορισμού της αλληλουχίας των υπηρεσιών που καλούνται, ή της αναδιαμόρφωσης της ενορχήστρωσης ώστε να αλλάξει η εφαρμογή.

Η Υπηρεσιοστραφής Αρχιτεκτονική είναι επίσης μια βέλτιστη πρακτική για να διορθώσουμε "σπασμένες" αρχιτεκτονικές. Με την ευρεία χρήση των προτύπων όπως τα Web Services, η Υπηρεσιοστραφής Αρχιτεκτονική προωθείται ως ο καλύτερος τρόπος για να φέρει την αρχιτεκτονική ευελιξία, αυτό βέβαια γίνεται αν εφαρμόσουμε σωστά την Υπηρεσιοστραφή Αρχιτεκτονική. Το πρόβλημα ήταν ότι οι τρόποι με τους οποίους οι επιχειρήσεις αξιοποιούν την

Υπηρεσιοστραφή Αρχιτεκτονική ως ένα αρχιτεκτονικό μοτίβο ποικίλλει σημαντικά από επιχείρηση σε επιχείρηση. Έτσι, η απόδοση της επένδυσης (Return of Investment) από τη μετάβαση σε Υπηρεσιοστραφείς Αρχιτεκτονικές κυμαίνονται από μεγάλες επιτυχίες σε οριστικές αποτυχίες. Η Υπηρεσιοστραφής Αρχιτεκτονική είναι μια έγκυρη προσέγγιση για να λύσει πολλά από τα αρχιτεκτονικά προβλήματα που αντιμετωπίζουν σήμερα οι επιχειρήσεις. Ωστόσο, εκείνοι που εφαρμόζουν Υπηρεσιοστραφείς Αρχιτεκτονικές συνήθως το βλέπουν ως κάτι που αγοράζεται και όχι κάτι που κάνεις.

Ενώ η Υπηρεσιοστραφής Αρχιτεκτονική απολάμβανε ποικίλες επιτυχίες στο παρελθόν, η κίνηση που υπάρχει προς το Υπολογιστικό Νέφος (cloud computing) δίνει μια πιο ανανεωμένη αξία σε αυτή την αρχιτεκτονική. Τα Clouds είναι συνήθως API ή εστιάζουν στις υπηρεσίες και έτσι λειτουργούν με γνώμονα την παροχή υπηρεσιών. Επειδή το cloud computing γίνεται όλο και πιο δημοφιλές, πολλές επιχειρήσεις θα επανεξετάσουν τη χρήση της Υπηρεσιοστραφούς Αρχιτεκτονικής, η οποία περιλαμβάνει τη χρήση των service directories, service governance, orchestration και άλλες τεχνολογίες που σχετίζονται με την Υπηρεσιοστραφή Αρχιτεκτονική.

Παρακάτω βλέπουμε πως είναι μια μονολιθική εφαρμογή σε σχέση με μια εφαρμογή που έχει κτιστεί πάνω στην Υπηρεσιοτραφή Αρχιτεκτονική.



Εικόνα 1-1 Service Oriented Architecture

(Πηγη: https://s-media-cache-ak0.pinimg.com/736x/0a/29/93/0a299365932e075c75b220910774e012.jpg)

Έχοντας αναλύσει λοιπόν την παραπάνω αρχιτεκτονική, βλέπουμε πόσο σημαντικό είναι - στις περισσότερες περιπτώσεις - να χρησιμοποιούμε την Υπηρεσιοστραφή Αρχιτεκτονική αφού μπορούμε εύκολα να δούμε ότι μας διευκολύνει σε ότι αφορά την ανάπτυξη αλλά και την επιδιόρθωση μιας εφαρμογής - συστήματος. Στις μέρες μας όμως μπορούμε να καταλάβουμε ότι το να δομήσουμε το σύστημα μας ώστε να περιστρέφετε γύρω από τις υπηρεσίες δεν είναι αρκετό σε κάποιες περιπτώσεις. Σαφώς, κάνει οτιδήποτε θέλουμε να πετύχουμε επαναχρησιμοποιήσιμο, διακριτό και πλήρως κατανοητό, αλλά τι συμβαίνει όταν θέλουμε να το επεκτείνουμε και να το κάνουμε ακόμα πιο διακριτό όπως π.χ. κάθε υπηρεσία να λειτουργεί ως μια πλήρως ανεξάρτητη και αποκεντρωμένη ενότητα;

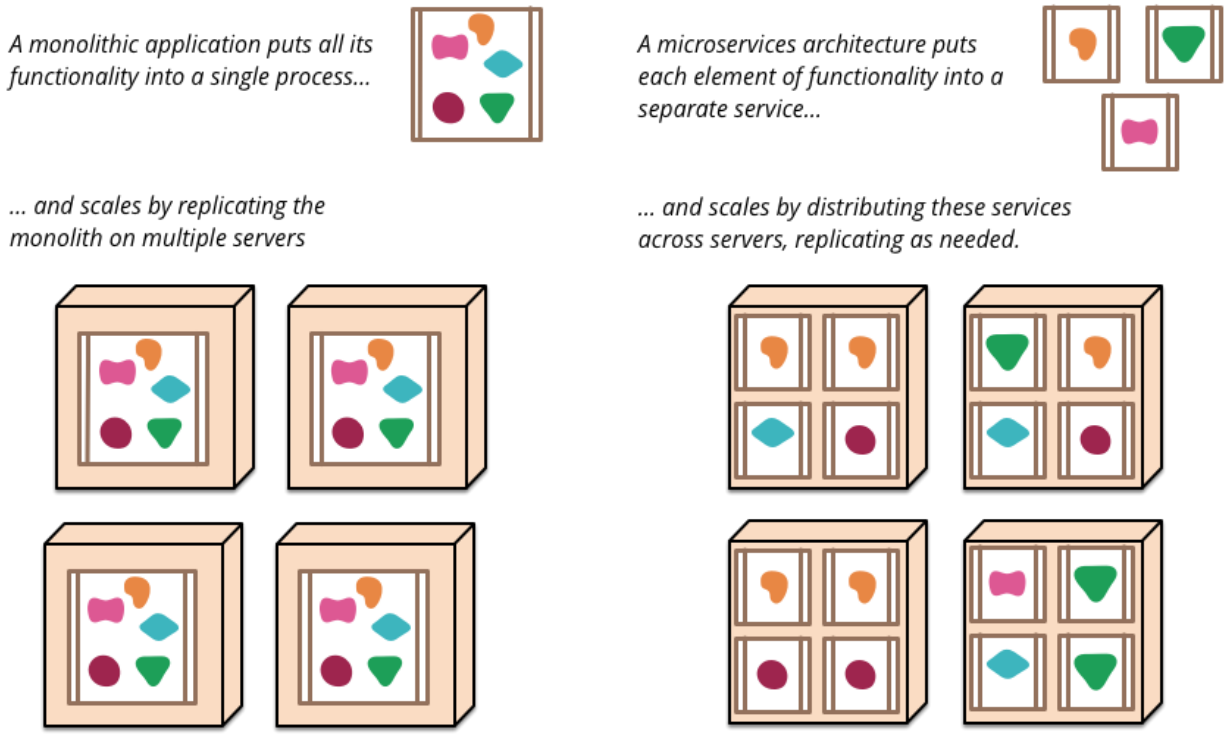
Σε αυτήν την περίπτωση, τον πρόβλημα αυτό τον λύνει η προσέγγιση των Μικρο-Υπηρεσιών (Microservices), οι οποίες είναι ένα "υποσύνολο" της Υπηρεσιοστραφούς Αρχιτεκτονικής. Ας δούμε λοιπόν εν συντομία, τι είναι αυτές οι Μικρο-Υπηρεσίες.

1.1.2 Microservices

Οι Μικρο-Υπηρεσίες είναι μια προσέγγιση για την ανάπτυξη εφαρμογών με την οποία μια μεγάλη εφαρμογή είναι χτισμένη ως ένα σύνολο σπονδυλωτών (modular) υπηρεσιών. Κάθε ενότητα υποστηρίζει ένα συγκεκριμένο επιχειρηματικό στόχο και χρησιμοποιεί μια απλή, καλά καθορισμένη διεπαφή για την επικοινωνία της με άλλες ενότητες.

Οι Μικρο-Υπηρεσιες ήρθαν για να βοηθήσουν στις μεγάλες εφαρμογές οι οποίες απαιτούν οι κύκλοι των αλλαγών να συνδέονται. Στο deployment μιας μονολιθικής εφαρμογής, κάθε μικρή αλλαγή σήμαινε ότι ολόκληρη η μονόλιθος έπρεπε να ανακατασκευαστεί και αυτό, με τη σειρά του, σημαίνει ότι η ανακατασκευή δεν συμβαίνει τόσο γρήγορα όσο θα έπρεπε να συμβαίνει. Σε μια αρχιτεκτονική Μικρο-Υπηρεσιών, κάθε Μικρο-Υπηρεσία τρέχει μια μοναδική διαδικασία και συνήθως διαχειρίζεται τη δική της βάση δεδομένων. Αυτό, όχι μόνο παρέχει στις ομάδες ανάπτυξης μια πιο αποκεντρωμένη προσέγγιση για την δημιουργία λογισμικού, αλλά και επιτρέπει επίσης σε κάθε υπηρεσία να αναπτυχθεί, να επανα-αναπτυχθεί, να γίνει redeployed και να διαχειριστεί ανεξάρτητα.

Παρακάτω φένεται η απεικόνιση μίας μονολιθικής εφαρμογής σε σχέση με μια εφαρμογή που έχει κτιστεί με χρήση των Μικρο-Υπηρεσιών.



Εικόνα 1-2

Microservices vs Monolithic applications

(Πηγή: https://martinfowler.com/articles/microservices.html)

Ένας άλλος όρος που πιστεύω αξίζει να αναφερθεί είναι ο "Aggregator" ή αλλιώς, Φορέας Συλλογής.

1.1.3 Aggregator

Όπως και το συνώνυμο του "συγκεντρωτής", ένας Φορέας Συλλογής είναι οποιαδήποτε συσκευή εξυπηρετεί πολλαπλές άλλες συσκευές ή χρήστες είτε με τις δικές του δυνατότητες είτε προωθεί διαβιβάσεις με έναν συγκεντρωμένο και οικονομικό τρόπο. Ένα απομακρυσμένο κομβικό σημείο πρόσβασης (HUB) μερικές φορές αναφέρεται και ως Aggregator.

Στο RPCMS θα δούμε μια παραλλαγή ενός aggregator και αυτή δεν είναι άλλη από τον Service Aggregator. Ένας Service Aggregator, δηλαδή ένας φορέας συλλογής υπηρεσιών, συλλέγει υπηρεσίες και όταν του ζητηθεί, τις χρησιμοποιεί ώστε να πάρει ένα αποτέλεσμα από τον πάροχο της.

Παρακάτω, θα δούμε πως χρησιμοποιούμε τον όρο του φορέα συλλογής στο RPCMS αλλά προτού γίνει αυτό, θα πρέπει να δούμε και το τι είναι και πως λειτουργεί ένας διακομιστής μεσολάβησης, κανονικός και αντίστροφος.

1.1.4 Forward/Reverse Proxy Server

Για να καταλάβουμε την έννοια του Reverse Proxy Server ας ξεκινήσουμε αναλύοντας έναν κανονικό (Forward) Proxy Server ή αλλιώς Διακομιστής Μεσολάβησης.

Ένας Διακομιστής Μεσολάβησης είναι ένα ειδικός υπολογιστής ή ένα σύστημα λογισμικού που τρέχει σε έναν υπολογιστή που λειτουργεί ως μεσάζων μεταξύ ενός endpoint, όπως ένας υπολογιστής, και έναν άλλο διακομιστή από τον οποίο ο χρήστης ή ο πελάτης ζητά μια υπηρεσία. Ο Διακομιστής Μεσολάβησης μπορεί να υπάρχει στο ίδιο μηχάνημα με ένα τείχος προστασίας ή μπορεί να είναι σε ένα ξεχωριστό διακομιστή, ο οποίος προωθεί τα αιτήματα μέσα από το τείχος προστασίας.

Ένα πλεονέκτημα ενός Διακομιστή Μεσολάβησης είναι ότι προσωρινή μνήμη του μπορεί να εξυπηρετήσει όλους τους χρήστες. Εάν συχνά ζητούνται μία ή περισσότερες ιστοσελίδες στο Διαδίκτυο, αυτά είναι πιθανό να είναι στη προσωρινή μνήμη του Διακομιστή Μεσολάβησης, η οποία θα βελτιώσει το χρόνο ανταπόκρισης των χρηστών. Ο Διακομιστής Μεσολάβησης μπορεί επίσης να καταγράψει τις αλληλεπιδράσεις του, κάτι το οποίο μπορεί να φανεί χρήσιμο για την αντιμετώπιση προβλημάτων.

Ένας Διακομιστής Μεσολάβησης λαμβάνει μια αίτηση για ένα πόρο του Διαδικτύου (όπως μια σελίδα Web), ψάχνει στην τοπική προσωρινή μνήμη του όπου βρίσκονται καταχωρημένες σελίδες που έχουν προβληθεί προηγουμένως. Αν βρει τη σελίδα, την επιστρέφει στον χρήστη, χωρίς να χρειάζεται να διαβιβάσει την αίτηση στο διαδίκτυο. Εάν η σελίδα δεν είναι στη μνήμη cache, ο διακομιστής μεσολάβησης, που ενεργεί ως πελάτης για λογαριασμό του χρήστη, χρησιμοποιεί μία από τις δικές του IP διευθύνσεις για να ζητήσει τη σελίδα από το διακομιστή στο διαδίκτυο. Όταν η σελίδα επιστραφεί, ο Διακομιστής Μεσολάβησης τη συσχετίζει με την αρχική αίτηση και τη διαβιβάζει στο χρήστη.

Οι Διακομιστές Μεσολάβησης χρησιμοποιούνται τόσο για νόμιμους όσο και παράνομους σκοπούς. Στις επιχείρησης, ένας διακομιστής μεσολάβησης χρησιμοποιείται για να διευκολύνει την ασφάλεια, τις υπηρεσίες διαχείρισης ή προσωρινής αποθήκευσης, κ.α. Στο πλαίσιο των προσωπικών υπολογιστών, οι Διακομιστές Μεσολάβησης χρησιμοποιούνται για να καταστεί δυνατή προστασία της ιδιωτικής ζωής των χρηστών και το ανώνυμο "σερφάρισμα". Οι Διακομιστές Μεσολάβησης μπορεί επίσης να χρησιμοποιηθούν για τον αντίθετο σκοπό: Για την παρακολούθηση της κυκλοφορίας και να υπονομεύσει την προστασία της ιδιωτικής ζωής των χρηστών.

Προς τον χρήστη, ο Διακομιστής Μεσολάβησης είναι αόρατος, όλες οι αιτήσεις Διαδικτύου και οι επιστρεφόμενες απαντήσεις φαίνονται να γίνονται άμεσα με τον διακομιστή Internet που εκείνοι απευθύνονται. (Ο Διακομιστής Μεσολάβησης δεν είναι πραγματικά αόρατος, η Διεύθυνση IP του πρέπει να προσδιοριστεί στις ρυθμίσεις του προγράμματος περιήγησης ή σε άλλο πρόγραμμα πρωτοκόλλου.)

Οι χρήστες μπορούν να έχουν πρόσβαση σε Διακομιστές Μεσολάβησης στο διαδίκτυο ή να ρυθμίσουν τα προγράμματα περιήγησης τους για να χρησιμοποιούν συνεχώς ένα διακομιστή μεσολάβησης. Οι ρυθμίσεις του προγράμματος περιήγησης περιλαμβάνουν είτε αυτόματη ανίχνευση είτε χειροκίνητη για HTTP, SSL, FTP, και SOCKS proxies. Οι Διακομιστές Μεσολάβησης μπορούν να εξυπηρετήσει πολλούς χρήστες ή μόνο έναν ανά διακομιστή. Οι επιλογές αυτές ονομάζονται κοινοί και ειδικοί Διακομιστές Μεσολάβησης αντίστοιχα. Υπάρχουν διάφοροι λόγοι για να χρησιμοποιήσει κάποιος έναν Διακομιστή Μεσολάβησης και έτσι υπάρχουν και διάφοροι τύποι από αυτούς, συχνά σε επικαλυπτόμενες κατηγορίες.

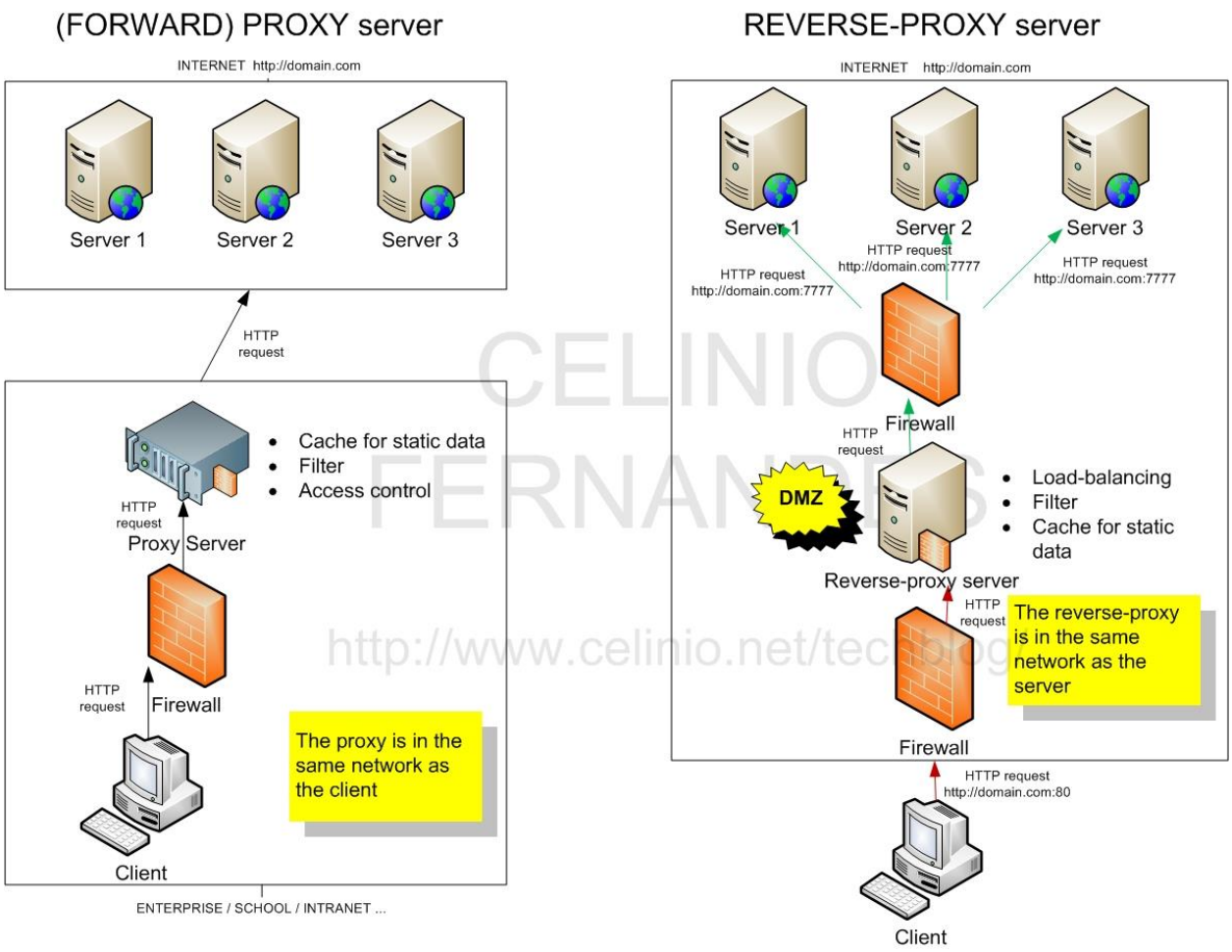
Οι (Προωθητικοί) Διακομιστές Μεσολάβησης αποστέλλουν τις αιτήσεις του πελάτη σε έναν web server. Οι χρήστες έχουν πρόσβαση σε αυτού του είδους τους Διακομιστές Μεσολάβησης είτε με απευθείας surfing σε μια διεύθυνση web proxy ή με τη διαμόρφωση των ρυθμίσεων του Internet. Αυτοί οι Διακομιστές Μεσολάβησης επιτρέπουν την παράκαμψη των firewalls και να αύξηση της προστασίας της ιδιωτικής ζωής και της ασφάλειας του χρήστη, αλλά μερικές φορές μπορεί κάποιος να τους χρησιμοποιήσει για να κατεβάσει παράνομο υλικό, όπως υλικό προστατευμένο με πνευματικά δικαιώματα ή παιδική πορνογραφία.

Οι Αντίστροφοι Διακομιστές Μεσολάβησης μπορούν να χειριστούν με διαφάνεια όλα τα αιτήματα για πόρους σε διακομιστές που προορίζονται να πάνε, χωρίς να απαιτείται καμία ενέργεια εκ μέρους του αιτούντος.

Οι Αντίστροφοι Διακομιστές Μεσολάβησης χρησιμοποιούνται:

- Για να ενεργοποιήσετε την έμμεση πρόσβαση, όταν ένας δικτυακός τόπος δεν επιτρέπει απευθείας συνδέσεις ως μέτρο ασφαλείας.
- Για να καταστεί δυνατή η εξισορρόπηση φορτίου (Load Balancing) μεταξύ διακομιστών.
- Για να κάνετε streaming εσωτερικό περιεχόμενο στους χρήστες του Διαδικτύου.
- Για να απενεργοποιήσετε την πρόσβαση σε μια τοποθεσία, για παράδειγμα, όταν ένας ISP ή κυβέρνηση επιθυμεί να αποκλείσει μια ιστοσελίδα.
- Για να συμπιέσει τα εισερχόμενα και εξερχόμενα δεδομένα, καθώς και την προσωρινή μνήμη από την οποία συνήθως ζητείται περιεχόμενο, έχοντας ως αποτέλεσμα την επιτάχυνση της ροής της κυκλοφορίας μεταξύ των πελατών και των διακομιστών. Μπορούν επίσης να εκτελέσουν πρόσθετες εργασίες όπως η κρυπτογράφηση SSL για να αναλάβουν φορτίο από τους διακομιστές ιστού, ενισχύοντας έτσι την απόδοσή τους.

Κάποιες τοποθεσίες θα μπορούσαν να μπλοκαριστιστούν για περισσότερο ή λιγότερο νόμιμους λόγους. Οι Αντίστροφοι Διακομιστές Μεσολάβησης μπορούν να χρησιμοποιηθούν για να εμποδίσουν την πρόσβαση σε ανήθικο, παράνομο ή περιεχομένο προστατευμένο με πνευματικά δικαιώματα. Μερικές φορές αυτοί οι λόγοι είναι δικαιολογημένοι, αλλά κάποιες άλλες η αιτιολόγηση είναι αμφίβολη. Οι Αντίστροφοι Διακομιστές μερικές φορές εμποδίζουν την πρόσβαση σε ειδησεογραφικές ιστοσελίδες όπου οι χρήστες θα μπορούσαν προβάλουν πληροφορίες που έχουν διαρρεύσει. Μπορούν επίσης να αποτρέψει τους χρήστες από την πρόσβαση σε δικτυακούς τόπους όπου μπορούν να αποκαλύπτουν πληροφορίες σχετικά με την κυβέρνηση ή τις βιομηχανικές ενέργειες. Παρεμπόδιση της πρόσβασης σε αυτές τις ιστοσελίδες μπορεί να παραβιάζει τα δικαιώματα της ελευθερίας του λόγου.



Εικόνα 1-3 Forward/Reverse Proxy Server
(Πηγή: http://www.celinio.net/techblog/wp-content/uploads/2011/09/ReverseProxy.jpg)

Από ότι φαίνεται, ένας reverse proxy server μπορεί να μας παρέχει διαφάνεια (transparency), ασφάλεια, απόδοση και ανωνυμία αν τον προσθέσουμε σε ένα σύστημα το οποίο λειτουργεί ως τώρα χωρίς κάτι τέτοιο. Ακόμα, ένας reverse proxy server διατίθεται να κάνει και load balancing εάν το σύστημα το οποίο εφαρμόζεται είναι ένα κατανεμημένο σύστημα και χρειάζεται μια τέτοια δυνατότητα.

Έχοντας λοιπόν όλα αυτά κατά νου θα μπορούμε να πούμε ότι το RPCMS είναι κατά κάποιο τρόπο ένας φορέας συλλογής λειτουργιών και υπηρεσιών και υιοθετεί κάποια από τα χαρακτηριστικά ενός Αντίστροφου Διακομιστή Μεσολάβησης (προς το παρών: ασφάλεια και ανωνυμία). Στη συνέχεια, θα μιλήσουμε για έναν κλασσικό τρόπο αυθεντικοποίησης.

1.1.5 Token-based Authentication

Η πιστοποίηση με την χρήση τεκμηρίων (token-based authentication) είναι κυρίαρχο χαρακτηριστικό του web στις μέρες μας. Με τις περισσότερες εταιρίες να χρησιμοποιούν APIs, τα τεκμήρια (tokens) είναι ο καλύτερος τρόπος να χειριστούμε την αυθεντικοποίηση για πολλαπλούς χρήστες. Υπάρχουν μερικοί πολύ σημαντικοί παράγοντες κατά την επιλογή του token-based authentication για μια εφαρμογή. Οι κυριότεροι είναι:

- Stateless και επεκτάσιμοι διακομιστές
- Μπορούν να χρησιμοποιηθούν από εφαρμογές για κινητά τηλέφωνα
- Μπορούν να περάσουν την αυθεντικοποίηση σε άλλες εφαρμογές
- Επιπλέον ασφάλεια

Το token-based authentication είναι stateless. Δεν αποθηκεύουμε καμία πληροφορία σχετικά με τον χρήστη σε κάποιον διακομιστή ή στο session. Αυτή η προσέγγιση φροντίζει για την αντιμετώπιση αρκετών προβλημάτων που δημιουργούνται όταν αποθηκεύουμε δεδομένα σε έναν διακομιστή.

Δίχως πληροφορίες στις συνεδρίες (sessions) μια εφαρμογή μπορεί να κλιμακωθεί και να προστεθούν περισσότερα μηχανήματα χωρίς να χρειάζεται να ανησυχούμε για το πού ένας χρήστης είναι συνδεδεμένος.

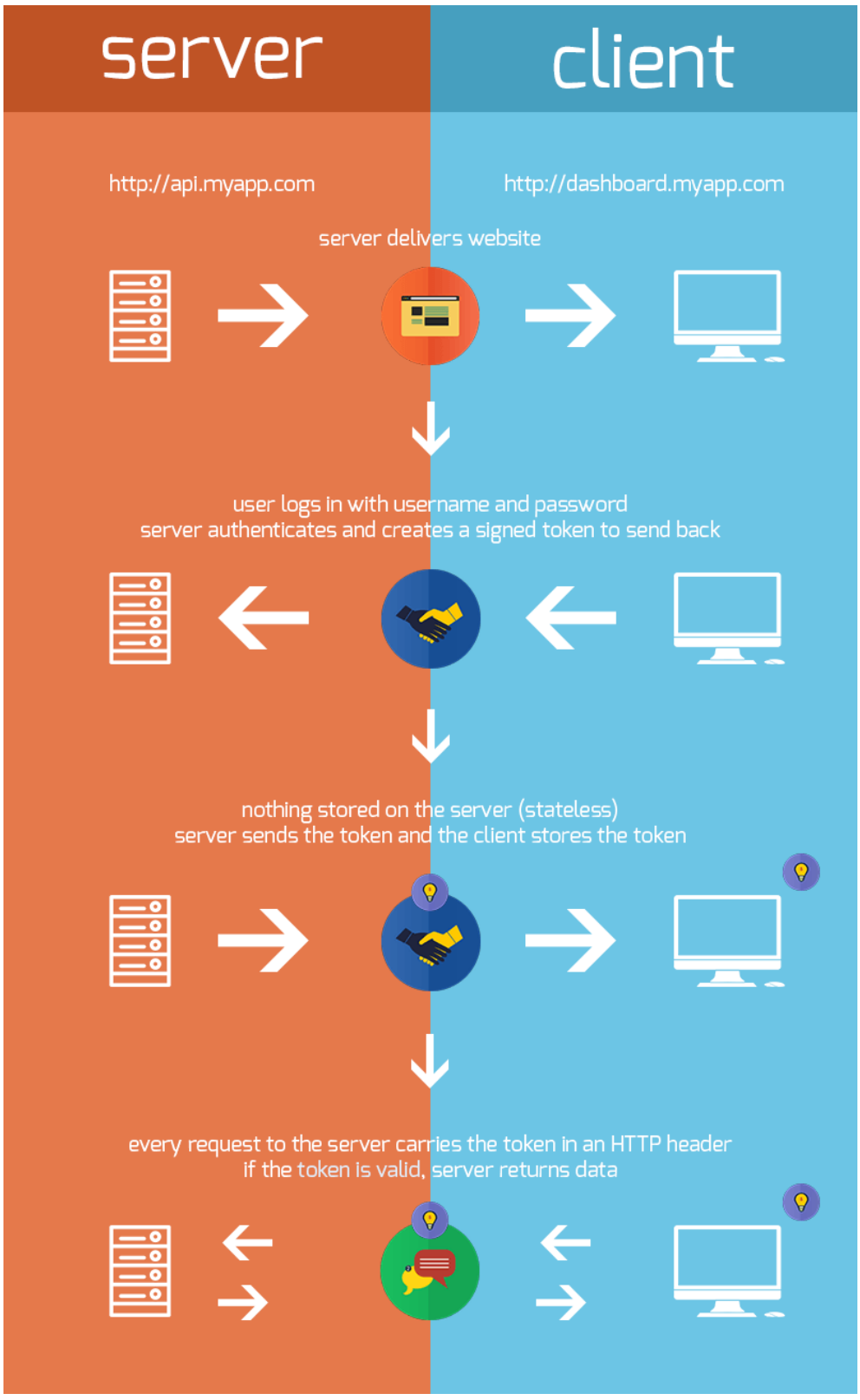
Αν και αυτή η εφαρμογή μπορεί να ποικίλει, η ουσία είναι ως εξής:

1. Ο χρήστης ζητάει πρόσβαση μέσω ενός ονόματος και ενός συνθηματικού
2. Η εφαρμογή επιβεβαιώνει τα διαπιστευτήρια
3. Η εφαρμογή δίνει στον πελάτη ένα υπογεγραμμένο τεκμήριο (signed token)
4. Ο πελάτης αποθηκεύει αυτό το τεκμήριο και το στέλνει μαζί με κάθε επόμενο αίτημα που στέλνει
5. Ο διακομιστής επιβεβαιώνει το διαπιστευτήριο και απαντά με δεδομένα/πληροφορίες

Κάθε αίτημα απαιτεί ένα τεκμήριο. Αυτό το τεκμήριο μπορεί να σταλεί και μέσω των HTTP headers ώστε να παραμείνει η ιδέα των stateless HTTP requests. Θα χρειαστεί επίσης να

αφήσουμε τον διακομιστή που δέχεται τις αιτήσεις, να τις δέχεται από διάφορα domains χρησιμοποιώντας το Access-Control-Allow-Origin: *. Το ενδιαφέρον κομμάτι όταν το θέτουμε σε * είναι ότι δεν αφήνουμε τις αιτήσεις να μας παρέχουν διαπιστευτήρια π.χ. HTTP authentication, client-side SSL certificates, cookies.

Σχηματικά μπορούμε να δούμε τα εξής:



Εικόνα 1-4 Token Based Authentication
(Πηγή: https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication)

Αφού έχουμε αυθεντικοποιηθεί με τα διαπιστευτήρια μας και πάρουμε το τεκμήριο, μπορούμε να κάνουμε πολλά πράγματα με αυτό.

Θα μπορούσαμε ακόμα και να δημιουργήσουμε ένα τεκμήριο βασισμένο σε συγκεκριμένες άδειες και να το δώσουμε σε μια εφαρμογή τρίτου (ας πούμε, μια καινούργια εφαρμογή για κινητά), αυτή η εφαρμογή θα μπορεί να έχει πρόσβαση σε δεδομένα μας - αλλά μόνο σε πληροφορία μου εμείς αφήσαμε να ανακτηθεί με το συγκεκριμένο τεκμήριο.

Ένα πιο συγκεκριμένο είδος τεκμηρίων είναι τα JSON Web Tokens τα οποία είναι βασισμένα στην JSON σύνταξη.

1.1.6 JSON Web Tokens

Το JSON Web Token (JWT) είναι ένα ανοιχτό πρότυπο (RFC 7519) που ορίζει ένα συμπαγή και αυτόνομο τρόπο για την ασφαλή διαβίβαση πληροφοριών μεταξύ των ομάδων σαν ένα JSON αντικείμενο. Αυτές οι πληροφορίες μπορούν να επαληθευτούν και είναι αξιόπιστα επειδή είναι ψηφιακά υπογεγραμμένα. Τα JWTs μπορούν να υπογραφούν χρησιμοποιώντας ένα μυστικό κλειδί (με τον αλγόριθμο HMAC) ή ένα ζεύγος δημόσιου / ιδιωτικού κλειδιού χρησιμοποιώντας RSA.

Ας εξηγήσουμε κάποιες έννοιες του ορισμού αυτού περαιτέρω.

- Συμπαγές: Λόγω του μικρού τους μεγέθους, τα JWTs μπορούν να αποσταλούν μέσω ενός URL, η σαν μια POST παράμετρος, ή μέσα σε μια κεφαλίδα HTTP. Επιπλέον, το μικρότερο μέγεθος έχει ως αποτέλεσμα μια γρήγορη διαβίβαση.
- Αυτοτελές: Το φορτίο (payload) περιέχει όλες τις απαιτούμενες πληροφορίες για τον χρήστη, αποφεύγοντας την ανάγκη για την αναζήτηση της βάσης δεδομένων πάνω από μία φορά.

Να μερικά σενάρια που θα ήταν χρήσιμα τα JWTs:

- Έλεγχος ταυτότητας (Authentication): Αυτό είναι το πιο σύνηθες σενάριο για τη χρήση JWTs. Μόλις ο χρήστης έχει συνδεθεί, κάθε μεταγενέστερη αίτηση θα περιλαμβάνει το JWT, επιτρέποντας στο χρήστη να έχει πρόσβαση σε routes, υπηρεσίες, και πόρους που επιτρέπονται με αυτό το κουπόνι. Το Single Sign On είναι ένα χαρακτηριστικό που χρησιμοποιεί ευρέως JWTs στις μέρες μας, λόγω του μικρού overhead του και την ικανότητά του να χρησιμοποιηθεί εύκολα σε διαφορετικούς τομείς..

- Ανταλλαγή πληροφοριών: Τα JWTs είναι ένας καλός τρόπος για την ασφαλή διαβίβαση πληροφοριών μεταξύ των ομάδων, διότι, δεδομένου ότι μπορούν να υπογραφούν, για παράδειγμα με τη χρήση δημόσιων / ιδιωτικών ζευγών κλειδιού, μπορείτε να είστε βέβαιοι ότι οι αποστολείς είναι αυτοί που λένε ότι είναι. Επιπλέον, αφού η υπογραφή υπολογίζεται χρησιμοποιώντας την κεφαλίδα και το φορτίο, μπορείτε επίσης να βεβαιωθείτε ότι το περιεχόμενο δεν έχει αλλοιωθεί

Τα JSON Web Tokens αποτελούνται από τρία μέρη που χωρίζονται από τελείες (.), τα οποία είναι:

- Header (Κεφαλίδα)
- Payload (Φορτίο)
- Signature (Υπογραφή)

Ως εκ τούτου, ένα JWT μοιάζει τυπικά όπως το ακόλουθο. xxxxx.yyyyy.zzzzz

1.1.6.1 Κεφαλίδα

Η κεφαλίδα συνήθως αποτελείται από δύο μέρη: το είδος του τεκμηρίου, το οποίο είναι JWT, και τον αλγόριθμο κατακερματισμού που χρησιμοποιείται, όπως HMAC SHA256 ή RSA.

Για παράδειγμα:

```
{
 "alg": "HS256",
 "typ": "JWT"
}
```

Στη συνέχεια, αυτό το JSON κωδικοποιείται με Base64Url για να σχηματίσει το πρώτο μέρος του JWT.

1.1.6.2 Φορτίο

Το δεύτερο μέρος του token είναι το φορτίο, το οποίο περιέχει τους ισχυρισμούς (claims). Οι ισχυρισμοί είναι δηλώσεις σχετικά με μια οντότητα (συνήθως, ο χρήστης) και επιπλέον μεταδεδομένα. Υπάρχουν τρεις τύποι των ισχυρισμών: reserved, public, and private ισχυρισμοί.

Reserved Claims: Αυτά είναι ένα σύνολο προκαθορισμένων ισχυρισμών που δεν είναι υποχρεωτική, αλλά συνιστάται, για να παρέχει μια σειρά από χρήσιμους, διαλειτουργικούς ισχυρισμούς. Μερικά από αυτά είναι: ISS (εκδότης), exp (ώρα λήξης), sub(θέμα), aud(κοινό), και άλλα.

Σημειώστε ότι τα ονόματα των ισχυρισμών είναι μόνο τρεις χαρακτήρες για να είναι το JWT συμπαγές.

Public Claims: Αυτά μπορούν να οριστούν κατά βούληση από εκείνους που χρησιμοποιούν τα JWTs. Αλλά για να αποφευχθούν οι συγκρούσεις θα πρέπει να ορίζονται στο Μητρώο IANA JSON Web Token ή να οριστεί ως ένα URI που περιέχει ένα ανθεκτικό στη σύγκρουση πεδίο ονομάτων (namespace).

Private Claims: Αυτές είναι οι προσαρμοσμένοι ισχυρισμοί που δημιουργούνται για την ανταλλαγή πληροφοριών μεταξύ των ομάδων που συμφωνούν για τη χρήση τους.

Ένα παράδειγμα θα μπορούσε να είναι ωφέλιμο φορτίο:

```
{
 "sub": "1234567890",
 "name": "John Doe",
 "admin": true
}
```

Το ωφέλιμο φορτίο είναι τότε Base64Url κωδικοποιημένα για να σχηματίσουν το δεύτερο μέρος του JSON Web Token.

1.1.6.3 Υπογραφή

Για να δημιουργήσετε το τμήμα της υπογραφής θα πρέπει να πάρετε την κωδικοποιημένη κεφαλίδα, το κωδικοποιημένο φορτίο, ένα μυστικό κλειδί, τον αλγόριθμο που καθορίζεται στην κεφαλίδα, και να το υπογράψετε.

Για παράδειγμα, εάν θέλετε να χρησιμοποιήσετε το HMAC SHA256 αλγόριθμο, η υπογραφή θα δημιουργηθεί με τον ακόλουθο τρόπο:

HMACSHA256 (

base64UrlEncode (header) + "." +

base64UrlEncode (ωφέλιμο φορτίο),

μυστικό κλειδί

)

Η υπογραφή χρησιμοποιείται για να επαληθεύσουμε ότι ο αποστολέας του JWT είναι αυτός που λέει ότι είναι και για να εξασφαλιστεί ότι το μήνυμα δεν άλλαξε.

1.1.6.4 Συνοψίζοντας

Η έξοδος είναι τρεις Base64 γραμματοσειρές που χωρίζονται από τελείες κάτι το οποίο μπορεί να περάσει εύκολα σε HTML και HTTP περιβάλλοντα, ενώ είναι πιο συμπαγής σε σύγκριση με τα πρότυπα που βασίζονται σε XML, όπως SAML.

Παρακάτω φαίνεται ένα JWT που έχει την προηγούμενη κεφαλίδα και φορτίο κωδικοποιημένα, και έχει υπογραφεί με ένα μυστικό.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Εικόνα 1-5 JWT hash

Μέχρι τώρα μιλήσαμε για αρχιτεκτονικές και όρους που σχετίζονται με το RPCMS, αλλά και τον τρόπο που κάνει την αυθεντικοποίηση. Δεν έχουμε πει τίποτα όμως, σχετικά με την αρχιτεκτονική που δημιουργήθηκε το RPCMS η οποία είναι το MVC, η περισσότερο χρησιμοποιημένη αρχιτεκτονική τα τελευταία χρόνο σε ότι αφορά το Back-End και της εφαρμογές ιστού.

1.1.7 MVC

Στην ανάπτυξη αντικειμενοστραφούς προγραμματισμού, το model-view-controller (MVC) είναι το όνομα μιας μεθοδολογίας ή πρότυπο σχεδίου (design pattern) για την επιτυχή και αποτελεσματική συσχέτιση - όσον αφορά τη διεπαφή χρήστη - στα υποκείμενα μοντέλα δεδομένων. Το πρότυπο MVC χρησιμοποιείται ευρέως στην ανάπτυξη προγραμμάτων με γλώσσες προγραμματισμού όπως Java, Smalltalk, C, και C ++.

Το πρότυπο MVC έχει ανακυρηχθεί από πολλούς προγραμματιστές ως ένα χρήσιμο πρότυπο για την επαναχρησιμοποίηση του κώδικα και ένα μοτίβο που τους επιτρέπει να μειώσουν σημαντικά το χρόνο που χρειάζεται για να αναπτύξουν εφαρμογές με διεπαφές χρήστη.

Το πρότυπο model-view-controller προτείνει τρία κύρια συστατικά στοιχεία ή αντικείμενα που θα χρησιμοποιηθούν στην ανάπτυξη λογισμικού:

- Ένα Μοντέλο (Model), το οποίο αντιπροσωπεύει την υποκείμενη, λογική δομή των δεδομένων σε μια εφαρμογή λογισμικού και την υψηλού επιπέδου κλάση που συνδέεται με αυτό. Αυτό το μοντέλο αντικειμένου δεν περιέχει καμία πληροφορία σχετικά με το περιβάλλον εργασίας χρήστη.
- Μια Όψη (View), η οποία είναι μια συλλογή από κλάσεις που αντιπροσωπεύουν τα στοιχεία της διεπαφής χρήστη (όλα τα πράγματα που ο χρήστης μπορεί να δει και να αλληλεπιδράσει στην οθόνη, όπως κουμπιά, παράθυρα, και ούτω καθεξής)
- Ένας ελεγκτής (Controller), που αντιπροσωπεύει τις κατηγορίες που συνδέουν το μοντέλο και τη όψη και χρησιμοποιείται για την επικοινωνία μεταξύ των κλάσεων στο μοντέλο και όψη.

1.1.8 Data Access Object (DAO)

Στο λογισμικό υπολογιστών, ένα αντικείμενο πρόσβασης δεδομένων (DAO) είναι ένα αντικείμενο που παρέχει μία διεπαφή σε κάποιο είδος βάσης δεδομένων ή άλλο μηχανισμό persistance. Κάνοντας χαρτογράφηση (mapping) των κλήσεων που κάνει η εφαρμογή στο persistence layer, το DAO παρέχει κάποιες συγκεκριμένες λειτουργίες χωρίς να εκθέτει τα στοιχεία της βάσης. Η απομόνωση αυτή υποστηρίζει την αρχή της ενιαίας ευθύνης (Single responsibility principle). Χωρίζει το τι πρόσβαση στα δεδομένα χρειάζεται η εφαρμογή, από την άποψη των αντικειμένων σε συγκεκριμένους τομείς (domains) και τύπους δεδομένων (η δημόσια διεπαφή του DAO), από το πώς αυτές οι ανάγκες μπορούν να ικανοποιηθούν με ένα συγκεκριμένο DBMS, σχήμα βάσης, κλπ (με την εφαρμογή του DAO).

Αν και αυτό το πρότυπο σχεδιασμού ισχύει εξίσου και για τα ακόλουθα:

1. Τις περισσότερες γλώσσες προγραμματισμού
2. Τα περισσότερα είδη λογισμικού με τις ανάγκες persistence
3. Τους περισσότερους τύπους βάσεων δεδομένων

συνδέονται συνήθως με τις εφαρμογές Java EE και με σχεσιακές βάσεις δεδομένων.

Αλλάζοντας θέμα, έρχεται η βάση δεδομένων. Για βάση δεδομένων χρησιμοποίησα το MongoDB η οποία είναι document-oriented και δεν ακολουθεί συγκεκριμένο schema. Κυρίως, χρησιμοποίησα αυτή την βάση δεδομένων διότι έχει πολύ καλό integration με το Node.js.

1.1.9 MongoDB

Το MongoDB είναι μια ανοικτού κώδικα βάση δεδομένων που χρησιμοποιεί document-oriented μοντέλο δεδομένων. Το MongoDB είναι ένα από τα πολλά είδη βάσης δεδομένων που προέκυψαν στα μέσα της δεκαετίας του 2000 κάτω από την NoSQL “σημαία”. Αντί να χρησιμοποιεί πίνακες και σειρές όπως οι σχεσιακές βάσεις δεδομένων, το MongoDB είναι χτισμένο σε μια αρχιτεκτονική των συλλογών και των εγγράφων. Τα έγγραφα περιλαμβάνουν σύνολα ζευγών κλειδιού-τιμής και είναι η βασική μονάδα δεδομένων στο MongoDB. Οι Συλλογές περιέχουν σύνολα των εγγράφων και να λειτουργούν ως το ισοδύναμο των πινάκων των σχεσιακών βάσεων δεδομένων.

Όπως και άλλες NoSQL βάσεις δεδομένων, το MongoDB υποστηρίζει δυναμική σχεδίαση του σχήματος, επιτρέποντας στα έγγραφα σε μια συλλογή για να έχουν διαφορετικά πεδία και δομές. Η βάση δεδομένων χρησιμοποιεί μια μορφή αποθήκευσης εγγράφου και ανταλλαγής δεδομένων που ονομάζεται BSON, η οποία παρέχει μια δυαδική αναπαράσταση από τύπου JSON έγγραφα.Το αυτόματο sharding επιτρέπει τα δεδομένα σε μια συλλογή που να διανεμηθούν σε πολλαπλά συστήματα ώστε να είναι δυνατή η οριζόντια επεκτασιμότητα και την αύξηση του όγκου των δεδομένων.

Το MongoDB δημιουργήθηκε από Dwight Merriman και Eliot Horowitz, οι οποίοι είχαν αντιμετωπίσει τα ζητήματα ανάπτυξης και επεκτασιμότητας με την παραδοσιακή προσεγγίση της σχεσιακής βάσης δεδομένων ενώ δημιουργούσαν εφαρμογές Web στην εταιρία DoubleClick (μια διαφημιστική εταιρεία στο Διαδίκτυο που τώρα ανήκει στην Google Inc). Σύμφωνα με τον Merriman, το όνομα της βάσης δεδομένων προήλθε από τη λέξη γιγαντιαία(humongous) να εκπροσωπεί την ιδέα της υποστήριξης μεγάλων ποσοτήτων δεδομένων. Οι Merriman και Horowitz βοήθησαν να διαμορφώσουν την 10Gen Inc. το 2007 για να εμπορευματοποιήσουν το MongoDB και το σχετικό λογισμικό. Η εταιρεία μετονομάστηκε MongoDB Inc. το 2013.

Ένα λογισμικό που χρησιμοποίησα κατά κόρον στο Project αυτό είναι το ενδιάμεσο λογισμικό (middleware) ώστε να μπορώ να εκτελώ κομμάτια κώδικα “ενδίαμεσα” από τα request και response π.χ. Token verification, admin check, rate limiter, ip blacklist και ip limiter

1.1.10 Ενδιάμεσο Λογισμικό

Η επικοινωνία μεταξύ του πελάτη και του διακομιστή δεν είναι άμεση. Συχνά, όταν ένας πελάτης απαιτεί μία υπηρεσία από έναν διακομιστή το αίτημα περνά μέσω πολλών ενδιάμεσων στρωμάτων λογισμικού που παρεμβάλλονται μεταξύ των πελατών και των διακομιστών. Αυτό το μεσαίο λογισμικό ονομάζεται **ενδιάμεσο λογισμικό (middleware).** Αποτελεί ζωτικό μέρος κάθε συστήματος πελάτη-διακομιστή. Ευτυχώς, τις περισσότερες φορές, ο προγραμματιστής και ο σχεδιαστής δεν χρειάζεται να ανησυχούν για το πώς λειτουργεί. Χρειάζεται μόνο να το χρησιμοποιούν μέσω κλήσεων μεθόδων. Η έννοια του ενδιάμεσου λογισμικού είναι κάπως ομιχλώδης. Ο καλύτερος ορισμός είναι των Orfali, Harkey και Edwards:

> *Το ενδιάμεσο λογισμικό είναι ένας ευρύς όρος που καλύπτει όλα τα κατανεμημένα λογισμικά που χρειάζονται για την υποστήριξη της*

συνεργασίας πελατών και διακομιστών....Πού αρχίζει το ενδιάμεσο λογισμικό και πού τελειώνει; Αρχίζει με την ομάδα API στην πλευρά του πελάτη που χρησιμοποιείται για την κλήση μιας υπηρεσίας, και καλύπτει την μεταβίβαση του αιτήματος στο δίκτυο και την αντίστοιχη απάντηση.

Το καλύτερο παράδειγμα ενδιάμεσου λογισμικού είναι το λογισμικό διεπαφής ενός περιηγητή και του Παγκόσμιου Ιστού. Όταν γίνεται αίτηση για μια ιστοσελίδα, ας πούμε όταν ένας χρήστης κάνει κλικ σε έναν υπερσύνδεσμο μιας σελίδας, ένα κείμενο, που αποτελεί μέρος ενός πρωτοκόλλου, αποστέλλεται στο ενδιάμεσο λογισμικό ζητώντας για την σελίδα. Στην συνέχεια, το ενδιάμεσο λογισμικό θα εντοπίσει την σελίδα, θα την ανακτήσει και θα την στείλει πίσω στον περιηγητή. Ως τμήμα του ενδιάμεσου λογισμικού υπάρχει κώδικας που ασχολείται με την ανεύρεση σελίδων, την παρακολούθηση λαθών, την αναφορά λαθών, την μεταβίβαση δεδομένων και την επικοινωνία με τα κατώτερα στρώματα λογισμικού, που είναι μέρος του TCP/IP που συζητήθηκε στο προηγούμενο κεφάλαιο.

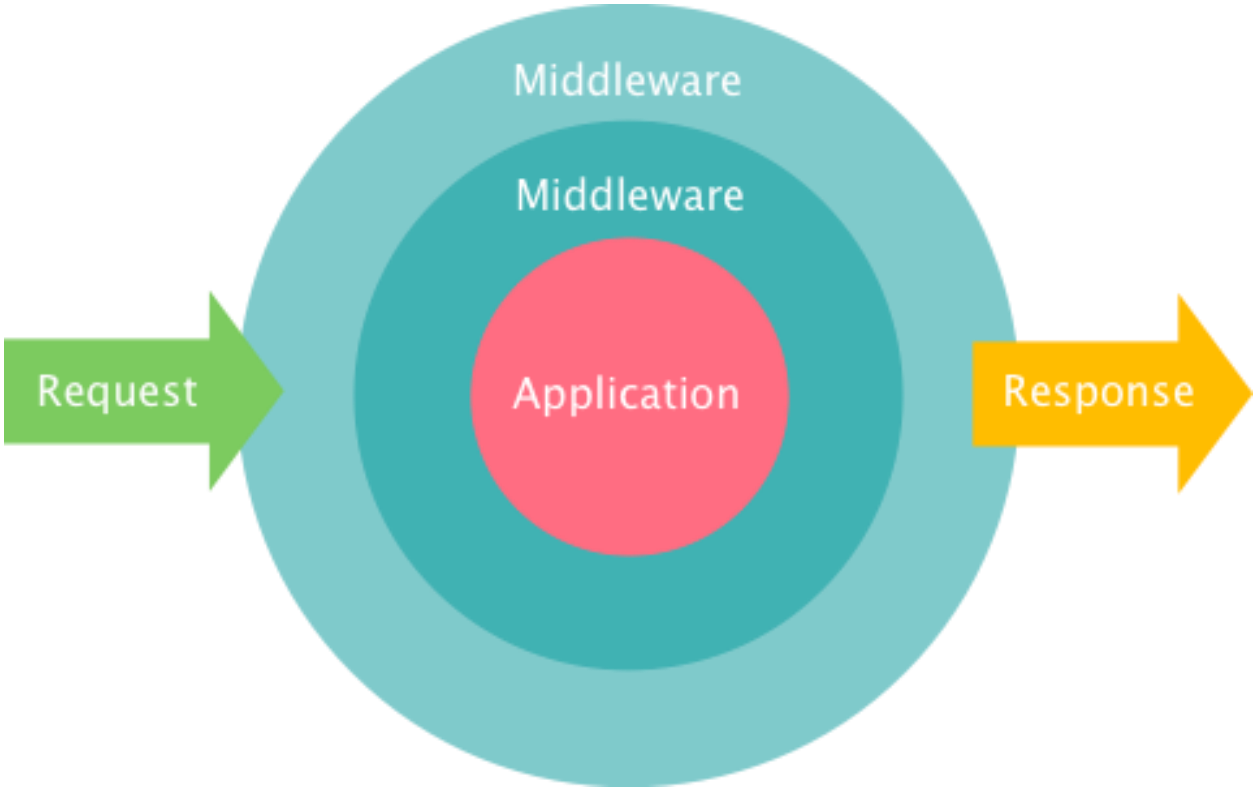
Υπάρχουν δύο κατηγορίες ενδιάμεσου λογισμικού: το γενικό ενδιάμεσο λογισμικό και το ειδικό ενδιάμεσο λογισμικό. Το γενικό ενδιάμεσο λογισμικό είναι λογισμικό που σχετίζεται με υπηρεσίες που ζητούνται από *όλους* τους πελάτες και τους διακομιστές. Κανονικά περισσότερο από αυτό το ενδιάμεσο λογισμικό αφορά τα λειτουργικά συστήματα. Το τυπικό λογισμικό που υπάγεται σ' αυτό το πλαίσιο περιλαμβάνει:

- Λογισμικό για την εκτέλεση διεργασιών όπως η μεταφορά ακατέργαστων δεδομένων χαρακτήρων στο Διαδίκτυο.
- Λογισμικό που συγχρονίζει πανομοιότυπα αντίγραφα αρχείων.
- Λογισμικό που διαχειρίζεται μία κατανεμημένη συλλογή αρχείων.

Το ειδικό ενδιάμεσο λογισμικό συσχετίζεται με μία συγκεκριμένη υπηρεσία όπως η εκτύπωση αρχείων σε έναν απομακρυσμένο υπολογιστή. Τα τυπικά παραδείγματα περιλαμβάνουν:

- Λογισμικό που επιτρέπει σ' έναν πελάτη να θέσει ερώτημα σε μία βάση δεδομένων, π.χ. λογισμικό που ερμηνεύει και εκτελεί ένα ερώτημα που τίθεται από τον χρήστη ενός πελάτη υπολογιστή και επεξεργάζεται τα δεδομένα στέλνοντας τα αποτελέσματα στον πελάτη.

- Λογισμικό που αφορά κατανεμημένα αντικείμενα όπως το RMI. Το λογισμικό αυτό επιτρέπει στον χρήστη να δημιουργήσει ένα αντικείμενο και στους πελάτες ν' αποστείλουν μηνύματα σ' αυτό το αντικείμενο. Υλοποιεί λειτουργίες που αφορούν την ασφάλεια, τον εντοπισμό ενός αντικειμένου, την διαβίβαση δεδομένων που σχετίζονται με την μέθοδο ορισμάτων σ' ένα κατανεμημένο αντικείμενο και την μεταφορά των αποτελεσμάτων ενός μηνύματος που έχει αποσταλεί σ' ένα κατανεμημένο αντικείμενο.
- Το ενδιάμεσο λογισμικό που αφορά τις ομάδες συζητήσεων που δίνει την δυνατότητα στον χρήστη να διαβάζει και να στέλνει μηνύματα στην ομάδα.



Εικόνα 1-6 Middleware
(Πηγή: https://dracony.org/replacing-controllers-with-middleware/)

Έχοντας αναλύσει όλα τα του software καλό θα ήταν να κάνουμε και μια αναφορά στο DMZ ή αλλιώς "ζώνη αποστρατικοποίησης", η οποία χρησιμοποιείται συνήθως για να χωρίσει το intranet από το internet σε μια επιχείρηση.

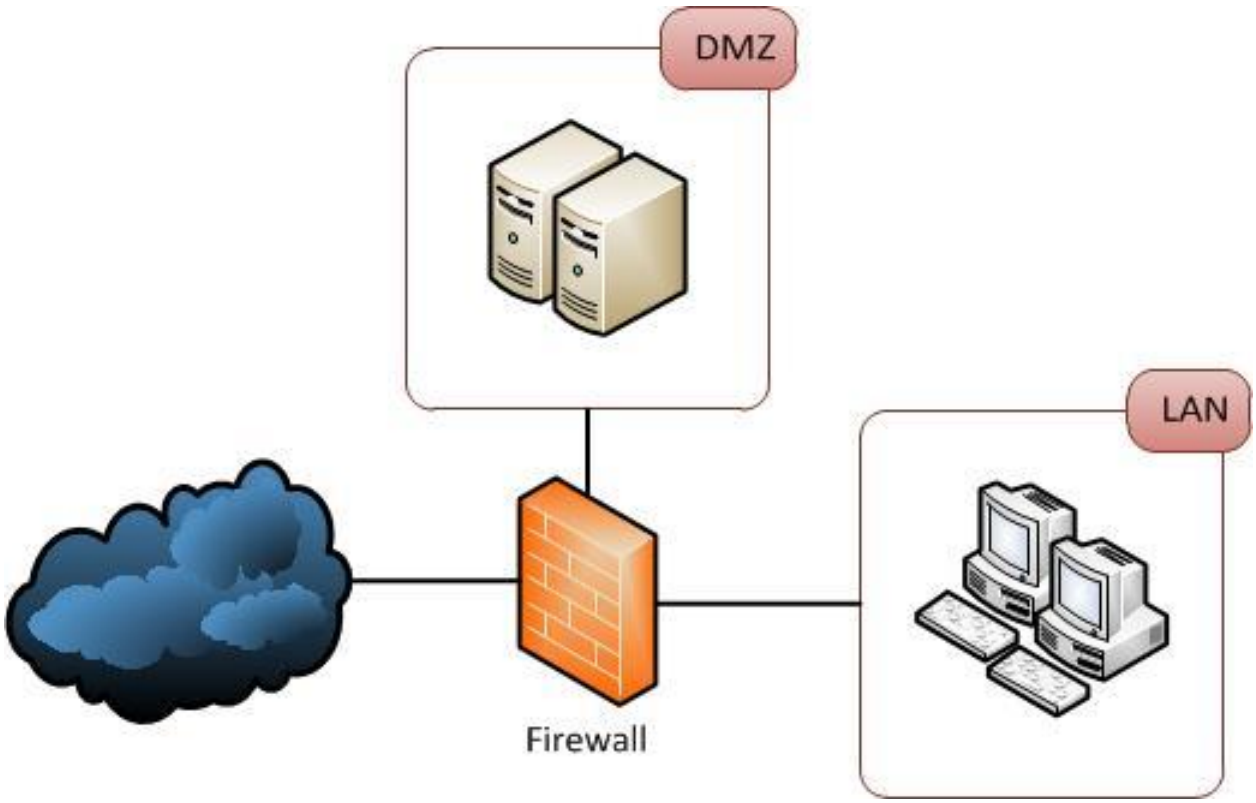
1.1.11 DMZ (Demilitarized Zone)

Στα δίκτυα υπολογιστών, το DMZ (αποστρατικοποιημένη ζώνη) είναι μια φυσικό ή λογικό υποδίκτυο που χωρίζει ένα εσωτερικό τοπικό δίκτυο (LAN) από άλλα μη αξιόπιστα δίκτυα, συνήθως στο Διαδίκτυο. Οι διακομιστές, πόροι και υπηρεσίες που βγαίνουν προς τα έξω

βρίσκονται στο DMZ έτσι ώστε να είναι προσβάσιμες από το Διαδίκτυο, αλλά το υπόλοιπο του εσωτερικού δικτύου LAN παραμένει απρόσιτο. Αυτό παρέχει ένα επιπλέον επίπεδο ασφάλειας στο δίκτυο LAN, δεδομένου ότι περιορίζει τη δυνατότητα των χάκερ να έχουν άμεση πρόσβαση εσωτερικούς διακομιστές και δεδομένα μέσω του Διαδικτύου.

Κάθε υπηρεσία που παρέχεται στους χρήστες του Διαδικτύου θα πρέπει να τοποθετηθεί στο DMZ. Οι πιο κοινές από αυτές τις υπηρεσίες είναι: Web, Mail, DNS, FTP, και VoIP. Τα συστήματα που εκτελούν αυτές τις υπηρεσίες στον DMZ είναι προσβάσιμα από τους χάκερ και εγκληματίες του κυβερνοχώρου σε όλο τον κόσμο και πρέπει να να αντέχουν συνεχή επίθεση. Ο όρος DMZ προέρχεται από τη γεωγραφική ζώνη απομόνωσης που είχε συσταθεί μεταξύ της Βόρειας Κορέας και της Νότιας Κορέας κατά το τέλος του Πολέμου της Κορέας. Το DMZ αναφέρεται συχνά ως ένα περιμετρικό δίκτυο.

Μια πιο ασφαλής προσέγγιση είναι να χρησιμοποιήσετε δύο τείχη προστασίας για να δημιουργήσετε ένα DMZ. Το πρώτο τείχος προστασίας που ονομάζεται επίσης και περίμετρος του τείχους προστασίας έχει ρυθμιστεί ώστε να επιτρέπει την κίνηση που προορίζεται μόνο στο DMZ. Το δεύτερο ή εσωτερικό τείχος προστασίας επιτρέπει μόνο την κίνηση από το DMZ προς το εσωτερικό δίκτυο. Αυτό θεωρείται πιο ασφαλές δεδομένου ότι και οι δύο συσκευές θα πρέπει να τεθούν σε κίνδυνο πριν ένας εισβολέας μπορεί να έχει πρόσβαση στο εσωτερικό LAN. Αφού το DMZ χωρίζει ένα δίκτυο, οι έλεγχοι ασφαλείας μπορούν να ρυθμιστούν ξεχωριστά σε κάθε τμήμα. Για παράδειγμα, ένα σύστημα ανίχνευσης και πρόληψης της διείσδυσης του δικτύου που βρίσκεται σε ένα DMZ που περιέχει μόνο έναν Web διακομιστή μπορεί να μπλοκάρει όλη την κίνηση, εκτός από HTTP και HTTPS αιτήσεις στις θύρες 80 και 443.



Εικόνα 1-7 DMZ
(Πηγή: http://i.stack.imgur.com/aFNLH.jpg)

1.2 Πρόταση

1.2.1 Γιατί JSON-RPC

Ξεκινώντας από το γεγονός ότι οι κυρίαρχες αρχιτεκτονικές για την δημιουργία APIs είναι τα RPC (Remote Procedure Call), REST (Representational State Transfer) και SOAP βλέπουμε ότι κάθε ένα έχει τα θετικά, τα αρνητικά και το καθένα μπορεί να είναι καταλληλότερο από το άλλο σε διάφορες περιπτώσεις. Θα προσπεράσω το SOAP αφού το REST έχει επικρατήσει σε ότι αφορά τα web services όπως αναφέρετε και εδώ https://en.wikipedia.org/wiki/SOAP αν κοιτάξουμε στα μειονεκτήματα.

Η REST αρχιτεκτονική δημιουργήθηκε με την νοοτροπία της πρόσβασης και διαχείρισης πόρων. Μπορεί κανείς, με μεγάλη ευκολία να διαχειριστεί τους πόρους αυτούς μέσω HTTP ρημάτων (GET, POST, PUT, PATCH, DELETE) και μέσω ουσιαστικών που αναπαριστούν αυτούς τους πόρους στο URI. Από την άλλη το RPC ασχολείται με λειτουργίες (operations) αντί για πόρους. Κατά την γνώμη μου τα δύο αλληλοσυμπληρώνονται και δεν θα πρέπει να υπάρχει τύπου "ποιο είναι το καλύτερο" αλλά "ποιο ταιριγιάζει καλύτερα σε αυτό το σενάριο".

Εικόνα 1-8 JSON vs XML over time

Το JSON-RPC, αφού το JSON είναι από τα πιο ραγδαία αναπτυσσόμενα format για σειριοποίηση δεδομένων (αυτό θα επιλεγόταν και σε περίπτωση που χρησιμοποιούσαμε την REST αρχιτεκτονική) είναι αντικειμενικά μικρότερο σε μέγεθος από το XML και πιο ευανάγνωστο (και τα δυο είναι human readable formats). Επίσης, λέγεται ότι υπάρχουν κάποια πράγματα στα οποία δεν θα μπορεί να ανταποκριθεί το REST και μπορούμε να ανατρέξουμε σε αυτά τα δυο άρθρα για ότι αφορά αυτό το θέμα:

1. https://joost.vunderink.net/blog/2016/01/03/why-we-chose-json-rpc-over-rest/
2. http://stackoverflow.com/questions/15056878/rest-vs-json-rpc.

Ας πάρουμε για παράδειγμα μια εφαρμογή που περιέχει χρήστες και θέλουμε να βρούμε τα διπλότυπα επίθετα αυτών και να τα ενώσουμε. Για να υλοποιηθεί κάτι τέτοιο σε REST θα πρέπει να φτιάξουμε ένα καινούργιο endpoint π.χ. "/users/merge-duplicate-surnames" κάτι το οποίο δεν ταιριάζει και τόσο στο REST γιατί δεν μπορούμε ένα HTTP ρήμα που να το αναπαριστά βγάζοντας νόημα. Από την άλλη κάνοντας μια αίτηση σε ένα JSON-RPC endpoint με τα εξής:

```
{
  “jsonrpc”: “2.0”,
  “method”: “mergeDuplicateSurnames”,
  “params”: [
    “Tsaganos”
  ]
}
```

είναι απολύτως σαφές το τι κάνουμε και τι θέλουμε να κάνουμε χωρίς δεύτερη σκέψη.

Ακόμα, σε ένα REST GET request δεν μπορούμε να βάλουμε υπο-δομές (μόνο του στυλ: “?a=1&b=2”), καθώς στο JSON-RPC μπορούμε να βάλουμε αντικείμενα μέσα σε πίνακες που ανήκουν σε άλλα αντικείμενα και ούτω κάθε εξής αν και αυτό ερχετε σε αντίθεση με το HTTP πρωτόκολλο.

1.2.2 RPCMS Αναλυτικά

Έχοντάς αναλύσει όλους τους παραπάνω όρους, μπορούμε να πούμε ότι ένας αντίστροφος διακομιστής μεσολάβησης είναι μια συσκευή ασφαλείας που συνήθως εγκαθίσταται στο DMZ ενός δικτύου, ώστε να προστατεύσει τους HTTP διακομιστές, σε ένα εταιρικό intranet, εκτελώντας λειτουργίες ασφαλείας που προστατεύουν τους εσωτερικούς διακομιστές από το δεχθούν επιθέσεις από χρήστες του διαδικτύου.

Κάτι που δεν έχει τονιστεί αρκετά μέχρι στιγμής, είναι πως το RPCMS είναι - όπως υποδεικνύουν και τα αρχικά του (RPC-”MS”) - ένα σύστημα διαχείρισης. Αυτό σημαίνει πως οποιοσδήποτε εκπαιδευμένος χρήστης μπορεί να το χρησιμοποιήσει. Αυτό δίνει την δυνατότητα στην επιχείρηση που το χρησιμοποιεί να είναι ανεξάρτητη από έναν προγραμματιστή ώστε να δώσει ή να πάρει την λειτουργικότητα που χρειάζεται εύκολα και γρήγορα, αλλά και να την παρακινήσει να χρησιμοποιήσει ένα σύστημα με το οποίο από την μια πλευρά θα προσθέτει μια επιπλέον ασφάλεια σε ότι θέλει να κρατήσει ιδιωτικό και από την άλλη θα κάνει την έκθεση των υπηρεσιών της πολύ εύκολη και παραμετροποιήσιμη. Βέβαια, η εγκατάσταση και η

παραμετροποίηση ενός τέτοιου συστήματος θα πρέπει να γίνει από έναν έμπειρο προγραμματιστή.

Θα αναρωτιέστε μέχρι στιγμής "και γιατί εγώ να χρησιμοποιήσω ένα τέτοιο σύστημα και να μην κάνω απλά μια κλήση σε ένα Application Programming Interface;".

Ας ξεκινήσουμε με το γεγονός ότι δεν μπορούμε να συγκρίνουμε κάτι τέτοιο γιατί το πρώτο είναι ένα κατανεμημένο σύστημα, αφού το RPCMS θα χρησιμοποιηθεί σε περίπτωση που χρειάζεται να συνδυαστούν πολλά συστήματα και να εκπροσωπηθούν από ένα, ενώ το δεύτερο είναι απλά μια κλήση από ένα σύστημα σε ένα άλλο.

Επιπλέον, το σύστημα μας δίνει την δυνατότητα να επιλέξουμε το endpoint στο οποίο θα προωθηθεί η κλήση την οποία θα λάβουμε από τον πελάτη (client) είτε αυτό το endpoint βρίσκεται στον ίδιο διακομιστή είτε σε κάποιον άλλο εξωτερικό. Τι μας παρέχει αυτό; Αυτό μας παρέχει ότι, αν μια υπηρεσία από έναν εξωτερικό πάροχο σταματήσει να λειτουργεί για κάποιον λόγο, τότε εμείς θα μπορούμε να αντικαταστήσουμε αυτήν την υπηρεσία τόσο εύκολα όσο αλλάζοντας το URL στο οποίο γίνεται η κλήση. Αυτό φυσικά προϋποθέτει να δέχονται αυτά τα API τις ίδιες παραμέτρους και να επιστρέφουν σε ίδια μορφή το αποτέλεσμα ώστε να μην "σπάσει" η συμβατότητα της υπηρεσίας αυτής στους πελάτες που ήδη την χρησιμοποιούν. Αυτό μας εξασφαλίζει μια εναλλακτική αν και όταν κάτι δεν πάει καλά με μια υπηρεσία ενός τρίτου και θέλουμε να την αντικαταστήσουμε είτε προσωρινά είτε μόνιμα.

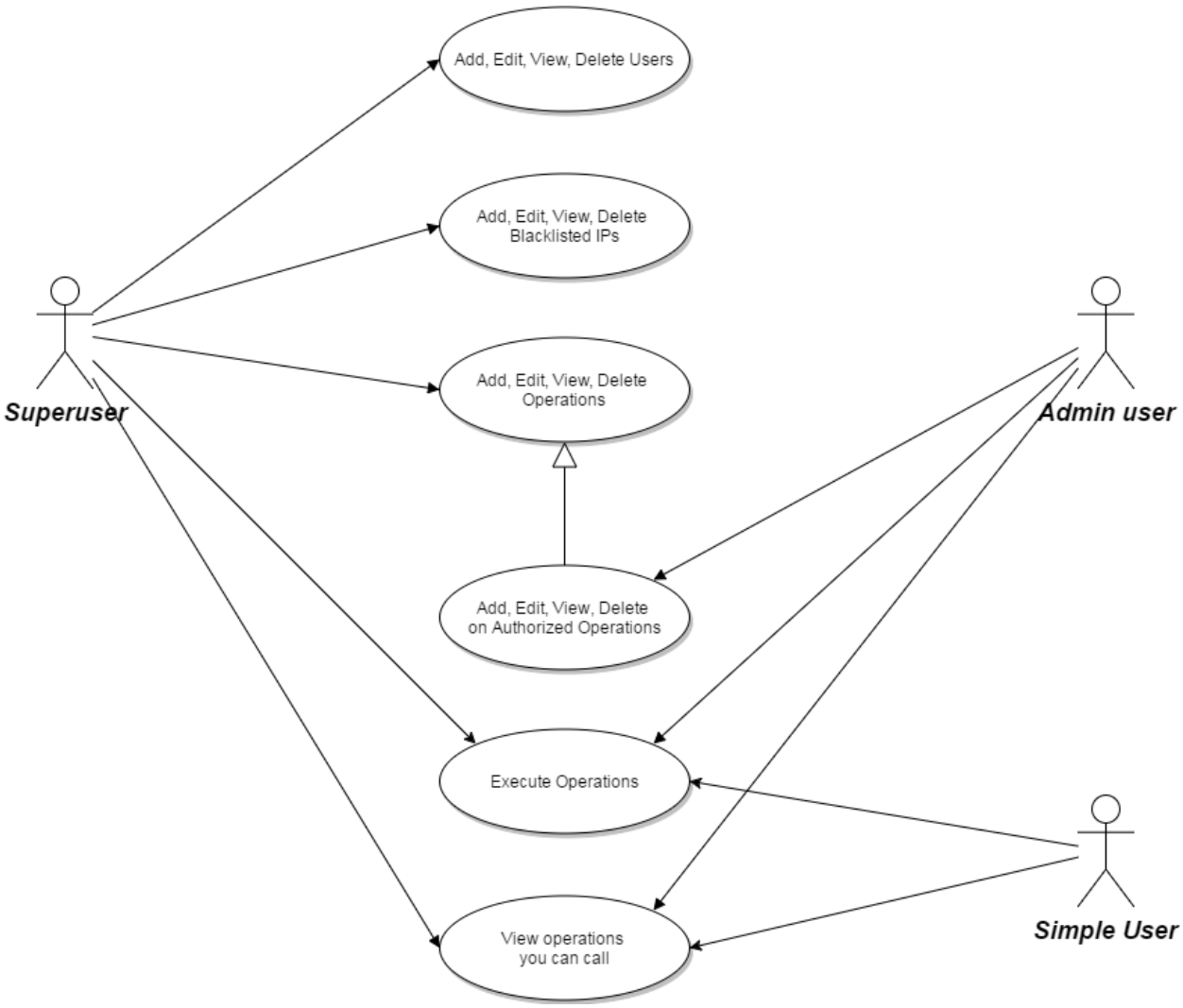
Ακόμα, ένα σύστημα σαν κι αυτό μπορεί να χρησιμοποιηθεί ώστε να εγκαθιδρύσει την επικοινωνία μεταξύ πολλών κόμβων μεταξύ τους. Με λίγα λόγια, όλοι απευθύνονται στο RPCMS και αυτό αναλαμβάνει το πού και το πώς θα στείλει την αίτηση. Ένα σενάριο χρήσης που θα δούμε παρακάτω στις μελλοντικές επεκτάσεις, είναι ότι κάθε επιχείρηση μπορεί να αποφασίσει να συμμετάσχει σε ένα πρόγραμμα και να μην χρειάζεται καμία άλλη ενέργεια, εκτός από το να παρέχει τα URL και τις παραμέτρους για τις δικές της υπηρεσίες. Βέβαια αν δεν μοιράζονται την ίδια δομή στις αιτήσεις που δέχονται και τις απαντήσεις που δίνουν θα χρειαστεί να φτιάξουν οι ίδιοι ένα API που να ακολουθεί αυτούς τους κανόνες και να χρησιμοποιήσει αυτό το API αντί του αυθεντικού (στυλ repository).

Ένας εναλλακτικός τρόπος αντιμετώπισης στο “σπάσιμο” της συμβατότητας, είναι να ορίσουμε εμείς ένα interface και να μετατρέπουμε τα δεδομένα που έρχονται από τους διακομιστές προσαρμόζοντας τα σε αυτό το interface. Επίσης, ένας ακόμα καλύτερος τρόπος για την διαχείριση αυτού του προβλήματος είναι να αφήνουμε τις ίδιες τις επιχειρήσεις να προσαρμόζουν τα δεδομένα που επιστρέφουν, αφού μπορούν να φτιάξουν απλά ένα παραπάνω endpoint μαζί με ένα repository που να κάνει τις απαραίτητες αλλαγές.

Τέλος, είναι οι ανάγκες που θέλουμε να καλύψουμε με αυτήν την εργασία. Το RPCMS έρχεται για να τυποποιήσει αυτά που χρειάζονται σχεδόν σε κάθε API π.χ. έλεγχος πρόσβασης, rate limiting, analytics κλπ , να παρουσιάσει διαφορετικά APIs ως ένα συνεκτικό στους comsumers του αλλά και να προσφέρει ένα κοινό σημείο ελέγχου όπου θα είναι αποθηκευμένα όλα τα keys άλλων API και οι comsumers θα χρειάζεται να δίνουν μόνο το token του RPCMS για να έχουν πρόσβαση σε αυτές τις υπηρεσίες.

2 Ανάλυση και Σχεδίαση

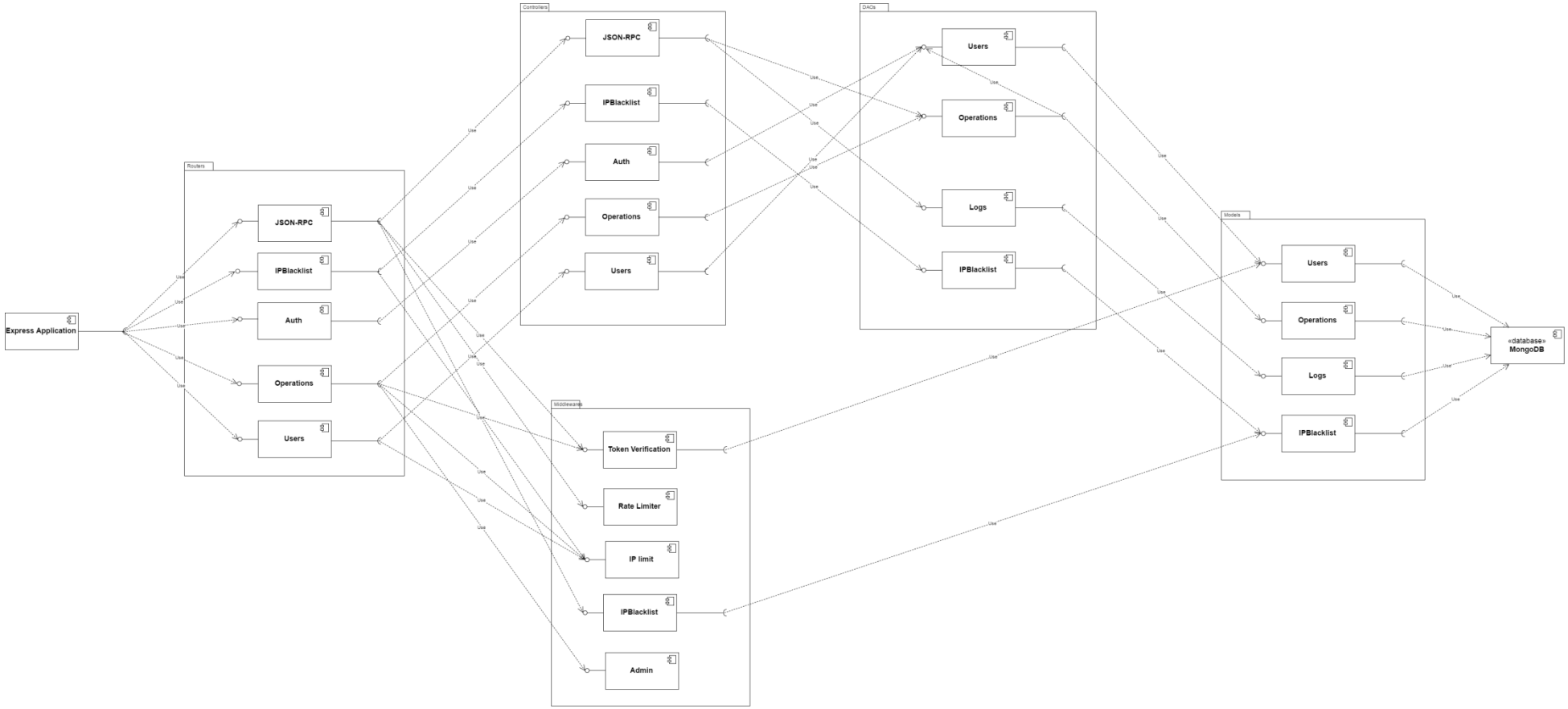
2.1 Use Case Diagram



Εικόνα 2-1 Use Case Diagram

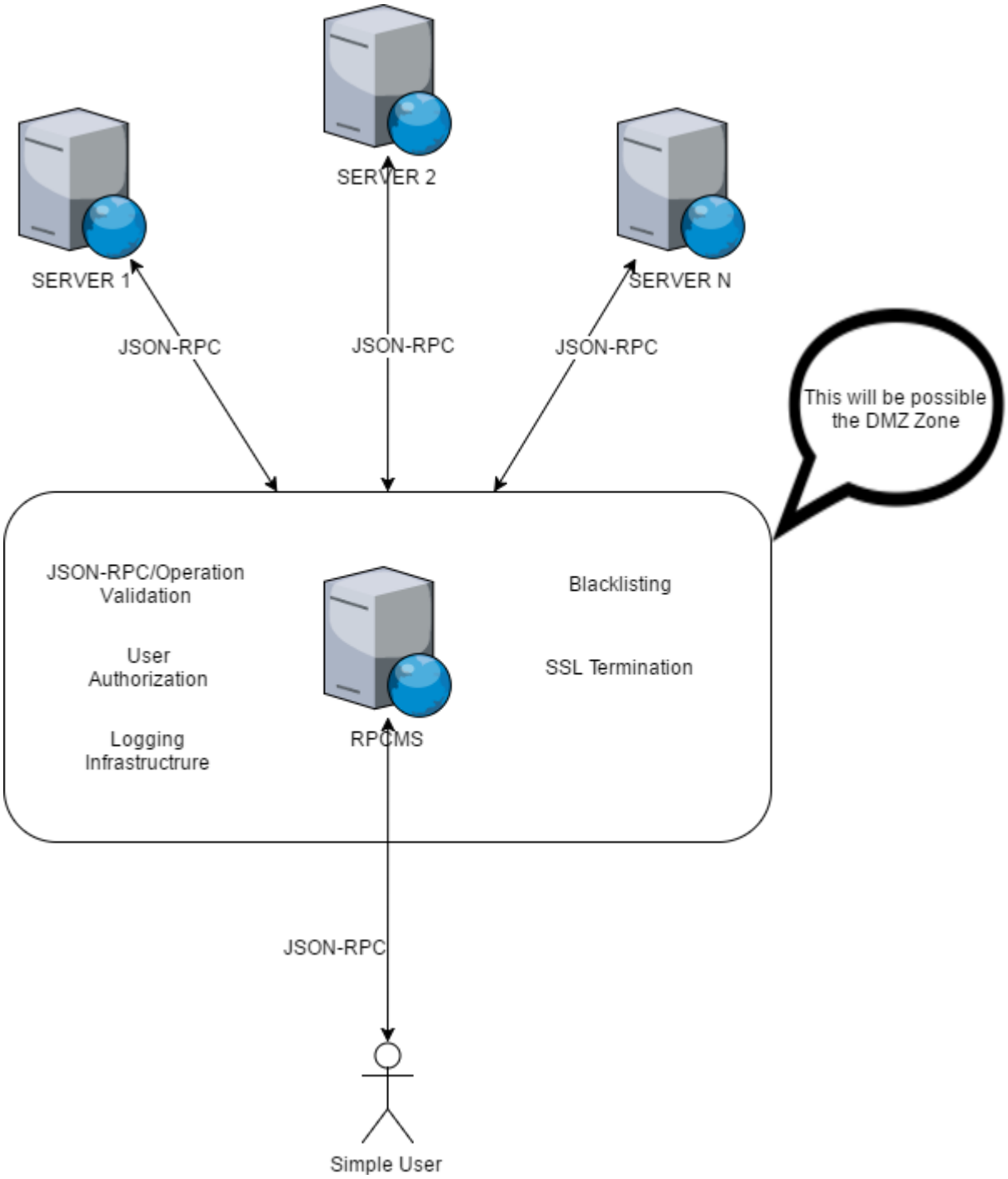
Παραπάνω, βλέπουμε ένα use case διάγραμμα το οποίο μας δείχνει τι μπορεί κάθε χρήστης να κάνει. Ένας Admin user μπορεί να κάνει Add, Edit, View, Delete Operations μόνο σε όσες υπηρεσίες του ανήκουν (τις έχει δημιουργήσει ο ίδιος).

2.2 Component-based Diagram



Εικόνα 2-2 Component Based Diagram
(Πηγή: http://83.212.122.31/Component.png)

2.3 Διάγραμμα Απεικόνισης Επικοινωνίας Συστημάτων



Εικόνα 2-3 System Communication Diagram

Στο παραπάνω διάγραμμα βλέπουμε πως επικοινωνούν οι υπόλοιποι διακομιστές με το RPCMS. Το RPCMS θα βρίσκεται πιθανότητα στο DMZ και χρησιμοποιεί το JSON-RPC για να επικοινωνήσει με όλα τα συστήματα με τα οποία το έχουμε συνδέσει.

2.4 Παρεχόμενες Υπηρεσίες του RPCMS

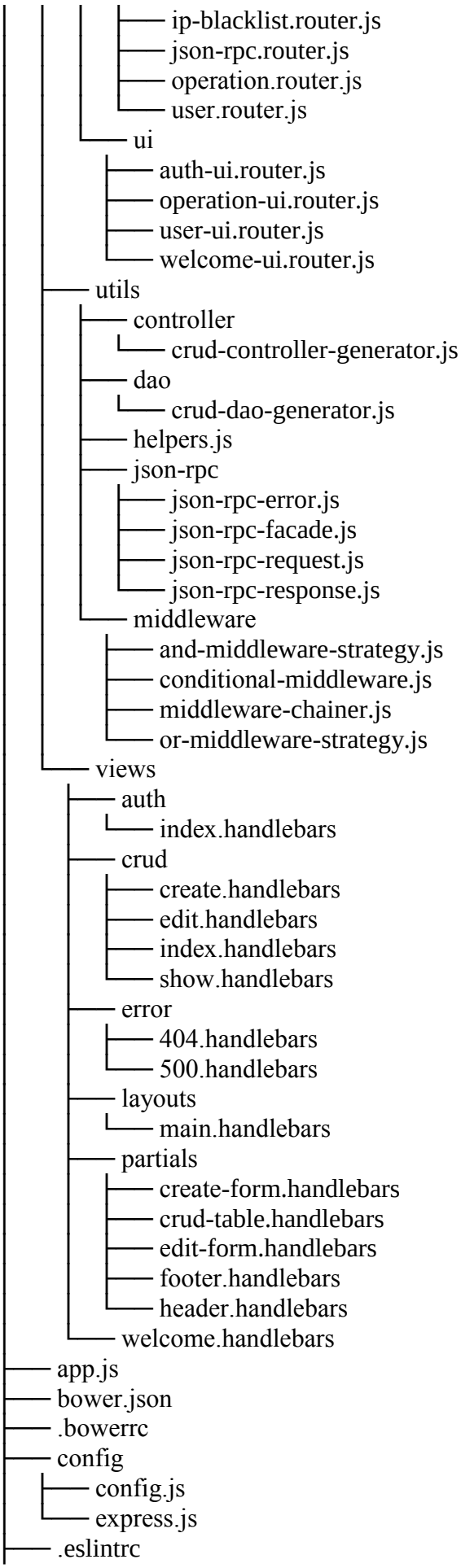
1. **Απλοποιεί τις εργασίες ελέγχου πρόσβασης**: Επειδή έχουμε μόνο ένα σημείο πρόσβασης, μπορούμε να επικεντρωθούμε στον έλεγχο πρόσβασης σε αυτό το σημείο. Για παράδειγμα, αντί να προσδιορίζουμε σε κάθε μεμονωμένο διακομιστή ποιες διευθύνσεις IP θα επιτρέπεται να συνδεθούν, μπορούμε απλά να δημιουργήσουμε ένα σύνολο από κανόνες πρόσβασης βασισμένους σε IP για το RPCMS. Αν ένας χρήστης προσπαθεί να συνδεθεί από μια μη εξουσιοδοτημένη IP, η προσπάθεια θα τερματιστεί άμεσα από το RPCMS. Προς το παρών, μπορούμε να ορίσουμε IPs που δεν επιτρέπεται να κάνουν αιτήσεις (IP Blacklist) όσων αφορά όλους τους διακομιστές σαν σύνολο και όχι ξεχωριστά για τον κάθε έναν.
2. **Μετακινεί τα διαπιστευτήρια χρηστών σε ασφαλέστερο μέρος**: Τα περισσότερα διαπιστευτήρια χρήστη απλά αποθηκεύονται στους ίδιους τους διακομιστές. Έτσι, εάν οι διακομιστές μας τοποθετηθούν στο DMZ μας, ένας επιτιθέμενος με κίνητρο μπορεί σχετικά εύκολα να έχει πρόσβαση σε αυτά. Με το να μετακινήσουμε τους διακομιστές μας στο εσωτερικό μας δίκτυο και θέσουμε σε λειτουργία το RPCMS για τον έλεγχο της πρόσβασης, μπορούμε να παρέχουμε καλύτερη ασφάλεια σε αυτά τα διαπιστευτήρια και κατά συνέπεια, με τα δεδομένα που προστατεύουν.
3. **Ασφάλεια**: Το RPCMS μπορεί να κρύψει την τοπολογία και τα χαρακτηριστικά των διακομιστών μας αφαιρώντας την ανάγκη για άμεση πρόσβαση στο Internet σε αυτά. Μπορούμε να τοποθετήσουμε το RPCMS σε DMZ και να κρύψουμε τους υπόλοιπους διακομιστές μας μέσα σε ένα μη δημόσιο υποδίκτυο.
4. **Αυθεντικοποίηση (Authentication)**: Το RPCMS μπορεί να παρέχει ένα μοναδικό σημείο που θα γίνεται το authentication για όλα τα HTTP αιτήματα.
5. **SSL Termination**: Εδώ το RPCMS χειρίζεται τις εισερχόμενες συνδέσεις HTTPS, να αποκρυπτογραφώντας τα αιτήματα και περνώντας τα, χωρίς κρυπτογράφηση, στους άλλους διακομιστές. Τα πλεονεκτήματα αυτού είναι τα εξής:
    a. Καταργεί την ανάγκη να εγκαταστήσετε πιστοποιητικά σε πολλαπλούς διακομιστές.
    b. Παρέχει ένα μοναδικό σημείο διαμόρφωσης και διαχείρισης για SSL / TLS
    c. Παίρνει το φορτίο επεξεργασίας για κρυπτογράφηση / αποκρυπτογράφηση της HTTPS κυκλοφορίας μακριά από διακομιστές.
    d. Διευκολύνει τις δοκιμές και την παρακολούθηση των HTTP αιτημάτων σε μεμονωμένους διακομιστές.

6. **Κεντρική Καταγραφή και Έλεγχος**: Επειδή όλες οι αιτήσεις HTTP δρομολογούνται μέσω του RPCMS, το κάνει τέλειο σημείο για την καταγραφή και τον έλεγχο. Μέχρι στιγμής γίνεται καταγραφή όλων των επιτυχημένων κλήσεων που γίνονται στο RPCMS.
7. **Επικύρωση JSON-RPC**: η κλήση που λαμβάνει το RPCMS επικυρώνεται πρώτα και μετά προωθείται στον κατάλληλο διακομιστή με βάση το schema που έχει ορίσει ο χρήστης.

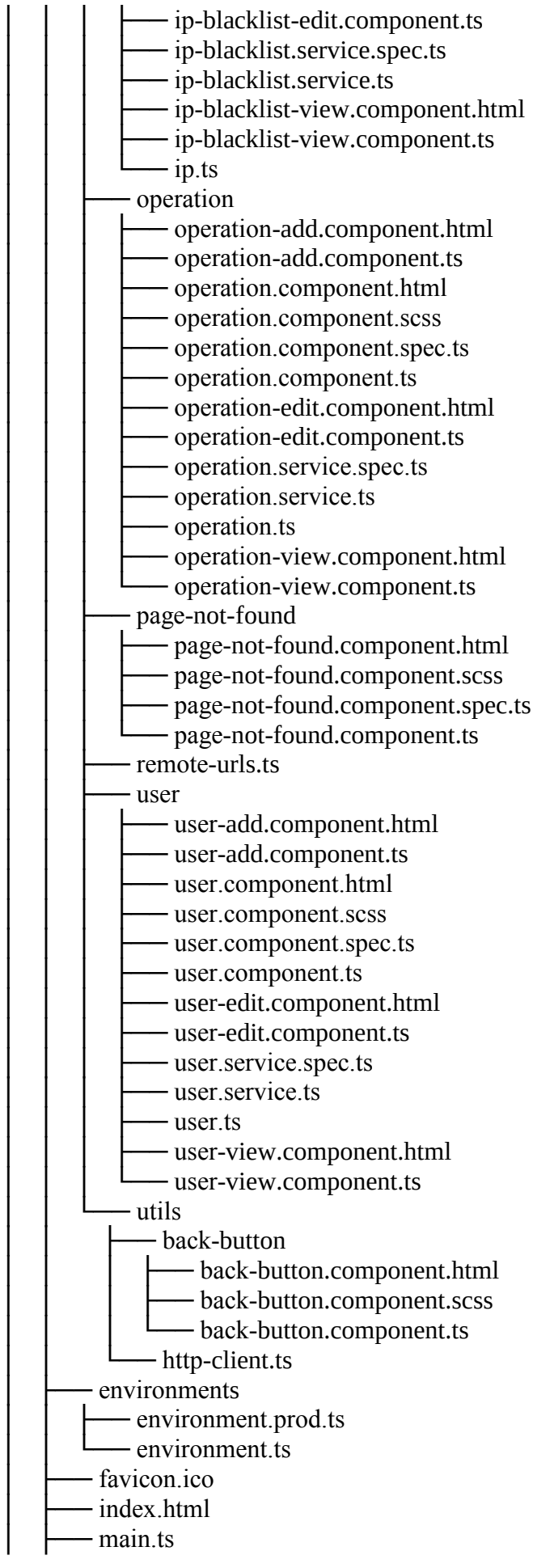
3 Υλοποίηση

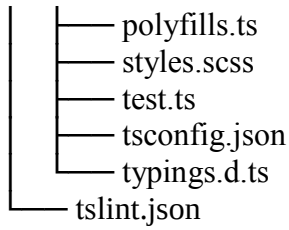
3.1 Directory Structure

```
├── app
│   ├── controllers
│   │   ├── api
│   │   │   ├── auth.controller.js
│   │   │   ├── ip-blacklist.controller.js
│   │   │   ├── json-rpc.controller.js
│   │   │   ├── operation.controller.js
│   │   │   └── user.controller.js
│   │   └── ui
│   │       ├── auth-ui.controller.js
│   │       ├── operation-ui.controller.js
│   │       ├── user-ui.controller.js
│   │       └── welcome-ui.controller.js
│   ├── dao
│   │   ├── ip-blacklist.dao.js
│   │   ├── logs.dao.js
│   │   ├── operation.dao.js
│   │   └── user.dao.js
│   ├── express-application.js
│   ├── middleware
│   │   ├── admin.middleware.js
│   │   ├── ip-blacklist.middleware.js
│   │   ├── ip-limit.middleware.js
│   │   ├── rate-limiter.middleware.js
│   │   └── token-verification.middleware.js
│   ├── models
│   │   ├── ip-blacklist.model.js
│   │   ├── logs.model.js
│   │   ├── operation.model.js
│   │   └── user.model.js
│   ├── routers
│   │   ├── api
│   │   │   ├── auth.router.js
```


ip-blacklist.router.js
json-rpc.router.js
operation.router.js
user.router.js
ui
auth-ui.router.js
operation-ui.router.js
user-ui.router.js
welcome-ui.router.js
utils
controller
crud-controller-generator.js
dao
crud-dao-generator.js
helpers.js
json-rpc
json-rpc-error.js
json-rpc-facade.js
json-rpc-request.js
json-rpc-response.js
middleware
and-middleware-strategy.js
conditional-middleware.js
middleware-chainer.js
or-middleware-strategy.js
views
auth
index.handlebars
crud
create.handlebars
edit.handlebars
index.handlebars
show.handlebars
error
404.handlebars
500.handlebars
layouts
main.handlebars
partials
create-form.handlebars
crud-table.handlebars
edit-form.handlebars
footer.handlebars
header.handlebars
welcome.handlebars
app.js
bower.json
.bowerrc
config
config.js
express.js
.eslintrc

```
├── .gitignore
├── gulpfile.babel.js
├── jsconfig.json
├── package.json
└── public
    ├── angular-cli.json
    ├── e2e
    │   ├── app.e2e-spec.ts
    │   ├── app.po.ts
    │   └── tsconfig.json
    ├── .editorconfig
    ├── .gitignore
    ├── karma.conf.js
    ├── package.json
    ├── protractor.conf.js
    ├── README.md
    ├── src
    │   ├── app
    │   │   ├── app.component.html
    │   │   ├── app.component.scss
    │   │   ├── app.component.spec.ts
    │   │   ├── app.component.ts
    │   │   ├── app.module.ts
    │   │   ├── auth
    │   │   │   ├── already-logged-in.guard.ts
    │   │   │   ├── auth.component.html
    │   │   │   ├── auth.component.scss
    │   │   │   ├── auth.component.spec.ts
    │   │   │   ├── auth.component.ts
    │   │   │   ├── auth.guard.ts
    │   │   │   ├── auth.service.spec.ts
    │   │   │   ├── auth.service.ts
    │   │   │   └── credentials.ts
    │   │   ├── client
    │   │   │   ├── client.component.html
    │   │   │   ├── client.component.scss
    │   │   │   ├── client.component.spec.ts
    │   │   │   └── client.component.ts
    │   │   ├── home
    │   │   │   ├── home.component.html
    │   │   │   ├── home.component.scss
    │   │   │   ├── home.component.spec.ts
    │   │   │   └── home.component.ts
    │   │   ├── index.ts
    │   │   ├── ip-blacklist
    │   │   │   ├── ip-blacklist-add.component.html
    │   │   │   ├── ip-blacklist-add.component.ts
    │   │   │   ├── ip-blacklist.component.html
    │   │   │   ├── ip-blacklist.component.scss
    │   │   │   ├── ip-blacklist.component.spec.ts
    │   │   │   ├── ip-blacklist.component.ts
    │   │   │   ├── ip-blacklist-edit.component.html
```


ip-blacklist-edit.component.ts
ip-blacklist.service.spec.ts
ip-blacklist.service.ts
ip-blacklist-view.component.html
ip-blacklist-view.component.ts
ip.ts
operation
operation-add.component.html
operation-add.component.ts
operation.component.html
operation.component.scss
operation.component.spec.ts
operation.component.ts
operation-edit.component.html
operation-edit.component.ts
operation.service.spec.ts
operation.service.ts
operation.ts
operation-view.component.html
operation-view.component.ts
page-not-found
page-not-found.component.html
page-not-found.component.scss
page-not-found.component.spec.ts
page-not-found.component.ts
remote-urls.ts
user
user-add.component.html
user-add.component.ts
user.component.html
user.component.scss
user.component.spec.ts
user.component.ts
user-edit.component.html
user-edit.component.ts
user.service.spec.ts
user.service.ts
user.ts
user-view.component.html
user-view.component.ts
utils
back-button
back-button.component.html
back-button.component.scss
back-button.component.ts
http-client.ts
environments
environment.prod.ts
environment.ts
favicon.ico
index.html
main.ts



Παραπάνω βλέπουμε την δομή των φακέλων του RPCMS. Να επισημάνω ότι ο φάκελος app/views δεν χρησιμοποιείται πια αφού για την διαχείριση του Front-End χρησιμοποίησα το Angular2 Framework το οποίο μόλις πριν λίγους μήνες βγήκε σε stable έκδοση.

Αρχικά θα ξεκινήσω από το γεγονός ότι το Front-End βρίσκεται στον φάκελο "public" και στον υποκατάλογο "public/src/app" βρίσκεται ο πηγαίος κώδικας σε ότι αφορά την εμφάνιση και την ανάκτηση δεδομένων από το API της εφαρμογής, βρίσκεται σε αυτόν τον φάκελο. Έχω χωρίσει όλες τις ξεχωριστές οντότητες σε φακέλους όπου ο κάθε ένας αντιπροσωπεί την δικιά του. Ο φάκελος "utils" περιέχει μέσα οτιδήποτε μπορεί να χρειαστεί από περισσότερες από μία οντότητες. Για παράδειγμα, το "back-button" χρησιμοποιείται από όλες τις άλλες οντότητες εκτός του "auth" και του "home".

Εν συνεχεία, ο φάκελος "app" περιέχει όλο τον πηγαίο κώδικα του Back-End της εφαρμογής. Όπως και στο Front-End έχω χωρίσει τον κώδικα σε ξεχωριστές έννοιες και οντότητες ώστε να μπορεί να περιηγηθεί εύκολα κάποιος μέσα στο Project και να βρει οτιδήποτε χρειάζεται χωρίς προσπάθεια. Η δομή αυτή δεν θα κρύψω ότι με βοήθησε πολύ και στο debugging της εφαρμογής όταν μου εμφανίζονταν κάποια προβλήματα και έπρεπε να τα διορθώσω.

Για να δούμε λοιπόν από τι απαρτίζεται αλλά πριν από αυτό να σας πω ότι οι φάκελοι "ui" δεν χρησιμοποιούνται πια λόγο της μεταφοράς του Front-End σε Angular2.

Πάνω πάνω βρίσκονται όλοι οι "controllers" οι οποίοι αναλαμβάνουν να βρουν και να επιστρέψουν τα δεδομένα που ζητούνται από τον πελάτη.

Έπειτα, είναι τα "dao" τα οποία χρησιμοποιούνται από οποιαδήποτε οντότητα (κυρίως τους controllers) και έχουν την χαμηλού επιπέδου λειτουργικότητα ως προς το ποιό μοντέλο θα χρησιμοποιήσουν και τι πράξεις κάνουν με αυτό (create , read, update, delete, mass delete, etc.).

Από κάτω βρίσκεται το “middleware” το οποίο περιέχει όλο το λογισμικό το οποίο εκτελείτε πριν από κύρια λειτουργικότητα της εφαρμογής μας. Το “middleware” χρησιμοποιείται από τους Routers ώστε να εκτελέσουν κάποιες λειτουργίες πρωτού εκτελεστεί η ενέργεια που ζήτησε ο πελάτης. Συνήθως, το “middleware” χρησιμοποιείται για να περιορίσει τη χρήση της εφαρμογής με βάση κάποια κριτήρια παραδείγματος χάρην, το TokenVerificationMiddleware ελέγχει αν υπάρχει κάποιο token στο αίτημα που έχει στείλει ο πελάτης και το επικυρώνει. Αν κάτι πάει στραβά σε αυτές τις δύο φάσεις, δεν αφήνει να εκτελεστεί η ενέργεια που ζήτησε ο πελάτης.

Στην συνέχεια, βλέπουμε τα “models”. Τα “models” είναι ο τρόπος που βάζουμε τα δεδομένα μας στην βάση δεδομένων. Τα “models” περιέχουν το Schema που χρησιμοποιείται από την βάση δεδομένων ώστε να αποθηκευτούν, κανόνες για επαλήθευση των πεδίων που παρέχονται από τον πελάτη και μεθόδους που μερικές φορές είναι απαραίτητες να εκτελεστούν πριν και μετά από μια συναλλαγή με την βάση δεδομένων (π.χ. Κρυπτογράφηση του κωδικού ενός χρήστη).

Αμέσως μετά, βρίσκονται οι “routers”. Οι “routers” είναι υπεύθυνοι να ορίζουν ποιος Controller πρέπει να εκτελείται για κάθε συγκεκριμένη ενέργεια που κάνει ο πελάτης. Εκεί μπορούμε να ορίσουμε το URL που θα βρίσκεται ένα endpoint και τα Middleware που θα χρησιμοποιηθούν πριν εκτελεστεί η ενέργεια που μας έχει ζητηθεί.

Τέλος, είναι τα “utils” τα οποία περιέχουν διάφορες λειτουργίες που μπορούν να χρησιμοποιηθούν από οποιαδήποτε άλλα αρχεία. Για παράδειγμα, οι φάκελοι “controller” και “dao” μέσα στον φάκελο “utils” περιέχουν κλάσεις που έχουν την κύρια λειτουργικότητα ενός CRUD(create, read, update, delete) controller και ενός CRUD dao που είναι υπεύθυνα για την διαχείριση των περισσοτέρων κλήσεων στο σύστημα. Ο “json-rpc” φάκελος χρησιμοποιείται για την ανάλυση των json-rpc κλήσεων, την δημιουργία των json-rpc απαντήσεων και σφαλμάτων. Ο φάκελος “middleware” περιέχει ένα npm (Node Package Manager) module που έφτιαξα στα πλαίσια αυτής της εργασίας και θα το αναλύσω πατακάτω.

3.2 Τεχνολογίες

Για την ανάπτυξη της εργασίας χρησιμοποιήθηκαν αρκετά εργαλεία και βιβλιοθήκες για διάφορους σκοπούς. Παρακάτω είναι η λίστα με τα εργαλεία και τις βιβλιοθήκες που χρησιμοποιήθηκαν μαζί με μια περιγραφή για το καθένα και μια επεξήγηση για το που φάνηκαν χρήσιμα.

- Node.js (https://nodejs.org/en/)
  Το Node.js είναι ένα περιβάλλον για ανάπτυξη Back-End εφαρμογών σε Javascript και στρέφεται γύρω από γεγονότα (event-driven).
- NPM (https://www.npmjs.com/)
  Το npm είναι ένα εργαλείο κονσόλας που χρησιμοποιείται για την αλληλεπίδραση με ένα τεράστιο αποθετήριο από NodeJS Projects.
- Bower (https://bower.io/)
  Το Bower είναι παρόμοιο με το npm αλλά χρησιμοποιείται κυρίως στο Front-End
- Yeoman (http://yeoman.io/)
  Το Yeoman είναι ένα γενικό σύστημα "σταδιοποίησης" (scaffolding) και επιτρέπει τη δημιουργία κάθε είδους εφαρμογής. Επιτρέπει την γρήγορη εκκίνηση project και απλοποιεί τη συντήρηση των ήδη υπαρχόντων. Χρησιμοποίησα τον https://github.com/petecoop/generator-express generator.
- Express.js (http://expressjs.com/)
  Το Express.js είναι ένα web application framework για το Node.js το οποίο έχει ανακοινωθεί ως δωρεάν και ανοιχτού κώδικα λογισμικό. Σχεδιάστηκε για να χτίζει εφαρμογές ιστού και APIs. Έως τώρα είναι εκ των πραγμάτων το πρότυπο για server-side development σε Node.js. Χρησιμοποιήθηκε ως βασικό Back-End framework για την ανάπτυξη του RPCMS.
- Gulp (http://gulpjs.com/)
  Το Gulp είναι μια εργαλειοθήκη που μας βοηθά να αυτοματοποιήσουμε επώδυνες ή χρονοβόρες εργασίες στο development workflow μας. Χρησιμοποιήθηκε στο Back-end για να αυτοματοποιήσει την διαδικασία του compilation και transpilation σε όλα τα κομμάτια του Project
- Babel 6 (https://babeljs.io/)
  Το Babel 6 είναι ένας transpiler (source-to-source) που μας δίνει την δυνατότητα να γράφουμε κώδικα σε καινούργιες εκδόσεις της Javascript οι οποίες δεν είναι συμβατές με

πολλές σύγχρονες τεχνολογίες (όπως για παράδειγμα: Chrome browser), μετατρέποντας τον κώδικα αυτόν σε παλαιότερες εκδόσεις της Javascript οι οποίες εκδόσεις είναι συμβατές με τις τεχνολογίες αυτές. Χρησιμοποιήθηκε για να κάνει τον κώδικα να μοιάζει με αντικειμενοστραφή και να αποφύγει την "άσχημη" δομή που έχει κατά την γνώμη μου η Javascript καθώς προσθέτει νέα features.

- Needle (https://github.com/tomas/needle)
  Βιβλιοθήκη για το Node.js η οποία κάνει την αποστολή και λήψη των HTTP request εύκολη και απλή. Χρησιμοποιήθηκε για να προωθεί τις αιτήσεις στους κατάλληλους διακομιστές.
- Json Web Tokens (https://jwt.io/)
  (Έχει αναλυθεί ήδη στην Εισαγωγή) Ο κύριος τρόπος αυθεντικοποίησης.
- MongoDB (https://www.mongodb.com/)
  (Έχει αναλυθεί ήδη στην Εισαγωγή) Η κύρια βάση δεδομένων του RPCMS.
- Mongoose (http://mongoosejs.com/)
  Object Document Mapper βιβλιοθήκη για το Node.js. Το Mongoose παρέχει μια απλή, schema-based λύση για τη μοντελοποίηση των δεδομένων της εφαρμογής μας. Περιλαμβάνει ενσωματωμένο type-casting, επικύρωση, την κατασκευή ερωτημάτων, bussiness logic hooks και άλλα. Χρησιμοποιήθηκε για την δημιουργία schemas, μοντέλων και επικύρωσης των μοντέλων αυτών στο Back-End.
- Bcrypt (https://www.npmjs.com/package/bcrypt-nodejs)
  Το Bcrypt είναι ένα password hashing function και βασίζεται στο Blowfish cipher. Πέρα από την ενσωμάτωση ενός salt για την προστασία από rainbow table τύπου επιθέσεις, το bcrypt είναι μια προσαρμοστική λειτουργία: με την πάροδο του χρόνου, ο αριθμός επανάληψης μπορεί να αυξηθεί για να γίνει πιο αργά, έτσι ώστε να παραμένει ανθεκτικό σε brute-force επιθέσεις αναζήτησης, ακόμη και με την αύξηση της υπολογιστικής ισχύος. Χρησιμοποιήθηκε για την one-way κρυπτογράφηση των password των χρηστών για την προστασία τους από τυχόν εισβολείς.
- Sass (http://sass-lang.com/)
  Το Sass είναι μια επέκταση του CSS που προσθέτει δύναμη και κομψότητα στη βασική γλώσσα. Μας επιτρέπει να χρησιμοποιήσουμε μεταβλητές, εμφωλευμένους κανόνες, μείγματα (mixins), inline εισαγωγή, και πολλά άλλα, όλα με πλήρως CSS-συμβατή σύνταξη. Όλα τα stylesheets στο Front-end που δημιουργήθηκαν απο την Angular2 ήταν σε αυτήν την μορφή μετά από δική μου επιλογή.
- ESLint (http://eslint.org/)
  Ένα άμεσα συνδέσιμο και παραμετροποιήσιμο εργαλείο linter για τον εντοπισμό και την

δημιουργία αναφορών σε ότι αφορά τα πρότυπα της JavaScript. Χρησιμοποιήθηκε για να αναφέρει τυχόν λάθη κατά την ανάπτυξη κώδικα σε Ecmascript 2015.

- Angular2 (https://angular.io/)
  Η Angular2 είναι ένα πλήρες βασισμένο σε JavaScript, ανοιχτού κώδικα Front-end web application framework. Χρησιμοποιήθηκε για την ανάπτυξη όλου του Front-end του RPCMS.
- Angular-cli (https://github.com/angular/angular-cli)
  Εργαλείο της Angular2 που έχει προέλθει από το Ember.js και επιτρέπει την δημιουργία Project και οντοτήτων που είναι συμβατά με την Angular2. Χρησιμοποιήθηκε για την δημιουργία των components, services, stylesheets, models, views καθώς και το Front-End του project που δημιουργήθηκε το RPCMS, σε Angular2.
- Semantic-UI (http://semantic-ui.com/)
  Όμορφο και εύχρηστο UI Framework. Χρησιμοποιήθηκε για τον σχεδιασμό και την εμφάνιση του RPCMS.

3.3 Διαμόρφωση (configuration)

Στα πλαίσια της διαμόρφωσης, για να είναι πολύ εύκολο να προσαρμόσουμε το RPCMS, έφτιαξα ένα αρχείο “config.js” στον κατάλογο “config” στο οποίο βρίσκεται οποιαδήποτε πληροφορία σχετικά με τις μεταβλητές που χρησιμοποιούνται στο RPCMS. Οπότε αν θέλει κάποιος για παράδειγμα να αλλάξει τις IP που έχουν πρόσβαση στην εφαρμογή (superusers) μπορεί απλά να βάλει ή να βγάλει ένα κελί σε έναν πίνακα της javascript. Τα παρακάτω είναι οι παραμετροποιήσιμες μεταβλητές:

- Το root path του project.
- Το όνομα της εφαρμογής.
- Η θύρα που θα τρέχει η εφαρμογή.
- Οι IP διευθύνσεις που θα έχουν πρόσβαση superuser.
- Το μυστικό κλειδί της εφαρμογής το οποίο χρησιμοποιείται για όλα τα κομμάτια της κρυπτογραφήσης.
- Το JSON-RPC endpoint του κεντρικού συστήματος (αν υπάρχει).

Σε ότι αφορά την διαμόρφωση του περιβάλλοντος της εφαρμογής, χρειάζεται μόνο ένα instance του MongoDB να τρέχει ώστε να μπορεί να γίνει η σύνδεση από το RPCMS, και ένα rule στο

firewall ώστε να μπορούμε να έχουμε εισερχόμενο και εξερχόμενο traffic. Οτιδήποτε άλλο αφορά το configuration του server μπορούμε να ανατρέξουμε στον ίδιο κατάλογο, στο αρχείο “express.js” το οποίο περιέχει όλα όσα σχετίζονται με τις ρυθμίσεις που μας δίνει πρόσβαση το Express.js framework.

3.4 Παραμετροποίηση

Η παραμετροποίηση του Project είναι πολύ εύκολη καθώς η δομή του είναι πολύ ευέλικτη και χρησιμοποιεί έτοιμο κώδικα και βιβλιοθήκες, που έχω φτιάξει είτε μόνος μου ή τις έχω βρει online από τρίτους, ώστε να μπορώ να προσθέτω και να βρίσκω γρήγορα και εύκολα κομμάτια κώδικα που κάνουν αυτό που χρειάζομαι. Αφού παραπάνω φαίνεται η δομή των καταλόγων του Project, μπορώ να περάσω στον οδηγό επέκτασης.

Οτιδήποτε δεν έχει αναλυθεί μέχρι στιγμής στο Project είναι self-explanatory. Στον φάκελο “utils” βρίσκεται όλο το generic υλικό που βοηθάει στην ανάπτυξη της εφαρμογής. Ότι δούμε παρακάτω είναι στον κάτω από τον κατάλογο “app/utils”.

Το “crud-controller.js” στον κατάλογο “controller” περιέχει μια συνάρτηση η οποία δημιουργεί κλάσεις με δυναμικό περιεχόμενο ανάλογα με της παραμέτρους που θα πάρει η παραπάνω συνάρτηση. Οπότε μπορούμε να δημιουργήσουμε μια κλάση που να έχει το πολύ τις μεθόδους “store”, “index”, “show”, “update”, “destroy”, destroyMass” και την “catchFunction” που μπορούμε εύκολα να καταλάβουμε τι κάνει η κάθε μια εκτός της catchFunction η οποία διαχειρίζεται τα σφάλματα που παράγονται από τις υπόλοιπες μεθόδους.

Το “crud-dao-generator.js” στον κατάλογο “dao” περιέχει επίσης μια συνάρτηση η οποία δημιουργεί, όπως και παραπάνω, κλάσεις με δυναμικό περιεχόμενο ανάλογα με τις παραμέτρους που θα πάρει. Οι μέθοδοι αυτής της κλάσεις μπορούν να είναι το πολύ οι getAll, getById, save, updateById, deleteById, deleteMultiple και catchFunction.

Τα δύο αρχεία που είδαμε, χρησιμοποιούνται για την δημιουργία CRUD (Create, Read, Update, Delete) Controllers και DAOs αφού χρησιμοποιούνται συνέχεια όταν θέλουμε να φτιάξουμε το Back-End ενός πόρου. Οπότε, όταν χρειαστεί να δημιουργήσουμε έναν καινούργιο πόρο,

μπορούμε να συνδυάσουμε αυτά τα δύο και να έχουμε μόνο με δύο “extend” την βασική λειτουργικότητα σε ότι αφορά το Back-End.

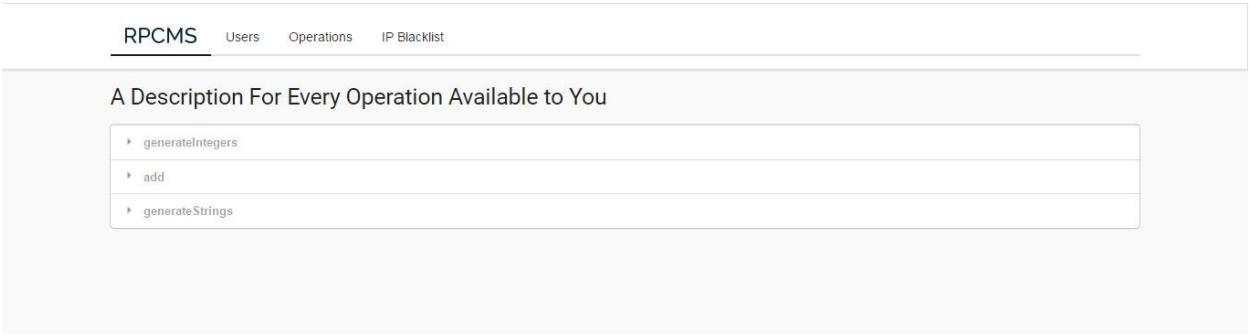
Στον υποκατάλογο “json-rpc” ο οποίος βρίσκεται στον φάκελο “utils” υπάρχουν οι κλάσεις και η λειτουργικότητα που σχετίζεται με την λήψη, την διαχείριση και την επαλήθευση αιτήσεων JSON-RPC αλλά και την δημιουργία σφαλμάτων και απαντήσεων σε JSON-RPC. Σε αυτό λοιπόν το σημείο, έχω φτιάξει μια κλάση στο αρχείο “json-rpc-facade.js” η οποία αναλαμβάνει να συνδυάσει όλα τα του JSON-RPC και να χρησιμοποιείται σαν “πρόσοψη” (facade) ώστε να κάνει ότι χρειάζεται ώστε να πάρουμε το σωστό αποτέλεσμα. Στο “json-rpc-request.js” αρχείο αποθηκεύεται η JSON-RPC αίτηση, στην συνέχεια επικυρώνεται (αν όλα τα στοιχεία που μας παρέχει ο πελάτης συμβαδίζουν με το specification του JSON-RPC 2.0) και αν είναι όλα καθώς πρέπει το “facade” προωθεί την αίτηση στον κατάλληλο διακομιστή μέσω της κλάσης που βρίσκεται στο αρχείο “json-rpc-response.js”. Σε περίπτωση λάθους, δημιουργεί ένα instance της κλάσης που βρίσκεται στο αρχείο “json-rpc-error.js” και το στέλνει στον πελάτη με το κατάλληλο error code το οποίο επίσης συμβαδίζει με το specification του JSON-RPC.

Εκτός από όλα τα παραπάνω που χρησιμοποιούνται σε ότι αφορά το JSON-RPC, δημιουργήθηκε και ένα npm module στα πλαίσια αυτής της εργασίας, το οποίο μας δίνει την δυνατότητα να μπορούμε να συνδυάσουμε ασύγχρονα Middlewares φτιάχνοντας ένα δέντρο από αυτά. Τα φύλλα του δέντρου είναι τα ίδια τα middleware ενώ τα υπόλοιπα είναι “στρατηγικές” συνδυασμού αυτών των middleware. Έχοντας χρησιμοποιήσει το Composition design pattern για να δημιουργούμε την δομή του δέντρου φτιάξαμε ξεχωριστά σε ένα κατάλογο όλες τις στρατηγικές (ονομάζονται έτσι αφού είναι κτισμένες βάσει του Strategy design pattern) και τις χρησιμοποιούμε για να προσδιορίσουμε τον τρόπο που θα συνδυάζονται τα φύλλα κάτω από ένα κόμβο. Μέχρι στιγμής έχουνφημιουργηθέι οι στρατηγικές “or” και “and” αλλά μπορούν να υλοποιηθούν πολύ εύκολα και άλλες στρατηγικές αφού το μόνο που χρειάζεται είναι η υλοποίηση της ίδιας της κλάσης που θα έχει μέσα την λογική της στρατηγικής. Τα υπόλοιπα γίνονται από τους προγραμματιστές που θα χρησιμοποιήσουν αυτήν την βιβλιοθήκη (ένα import είναι μόνο δηλαδή). Το παραπάνω module δημιουργήθηκε από την ανάγκη της εφαρμογής να μπορεί να κάνει authorize έναν χρήστη είτε με βάση την διεύθυνση IP (ένα middleware) Ή με τον συνδυασμό του αν ένας χρήστης κάνει login μέσω token ΚΑΙ είναι administrator (άλλα δυο middleware). Όλα αυτά έχουν εφαρμογή στα middleware του Express.js Framework.

Ο σύνδεσμος που παραπέμπει στο npm module είναι ο εξής: https://www.npmjs.com/package/express-conditional-tree-middleware

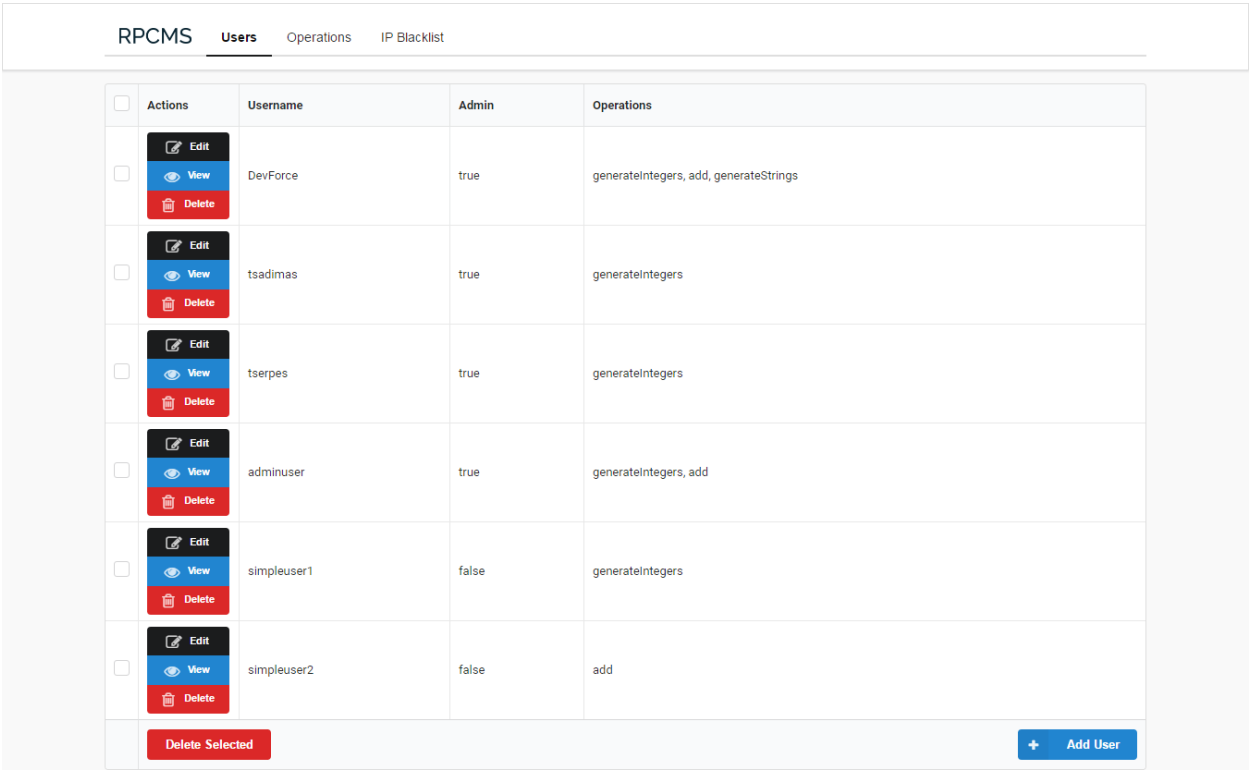
3.5 Documentation

Αρχικά, αν πάμε να συνδεθούμε στο RPCMS από μια IP διεύθυνση η οποία είναι καταχωρημένη στο configuration αρχείο του RPCMS, θα δούμε ένα UI σαν αυτό:



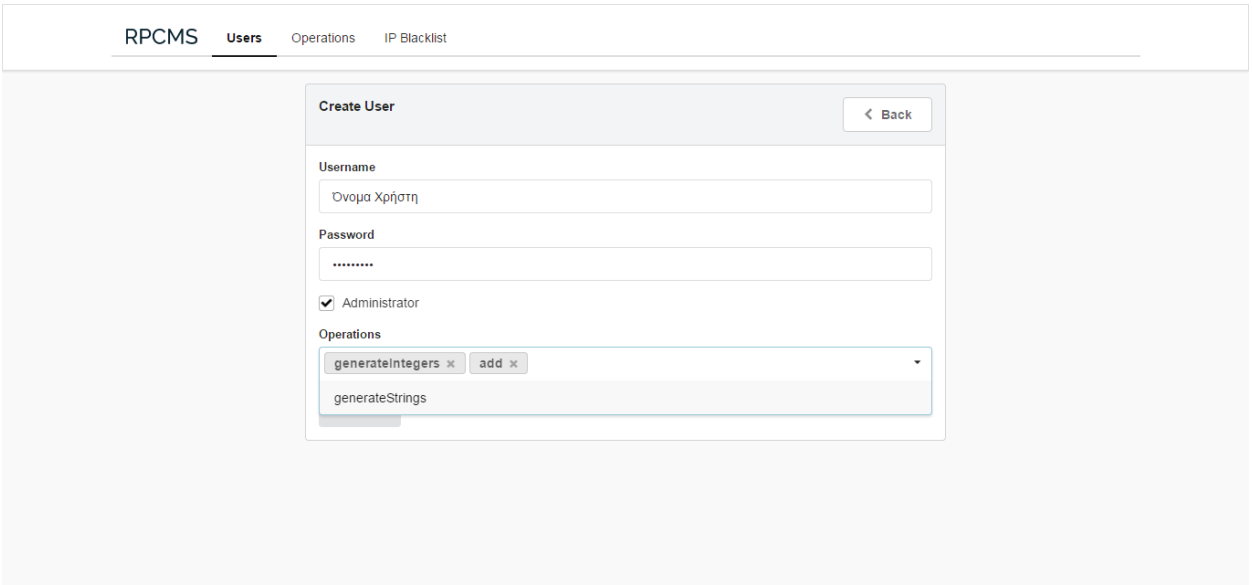
Εικόνα 3-1 RPCMS Home Screen

Παραπάνω βλέπουμε όλες τις λειτουργίες που έχουν καταχωρηθεί από όλους τους χρήστες του συστήματος καθώς και την περιγραφή της κάθε μιας. Αν πατήσουμε στο tab "Users" γίνεται η μετάβαση στο panel όπου μπορούμε να διαχειριστούμε τους χρήστες του συστήματος. Μπορούμε να κάνουμε Edit, View, Delete και Add για Επεξεργασία, Προβολή και διαγραφή αντίστοιχα. Μπορούμε ακόμα, με πολλαπλή επιλογή να διαγράψουμε πολλούς χρήστες ταυτόχρονα με το κουμπί Delete Selected.



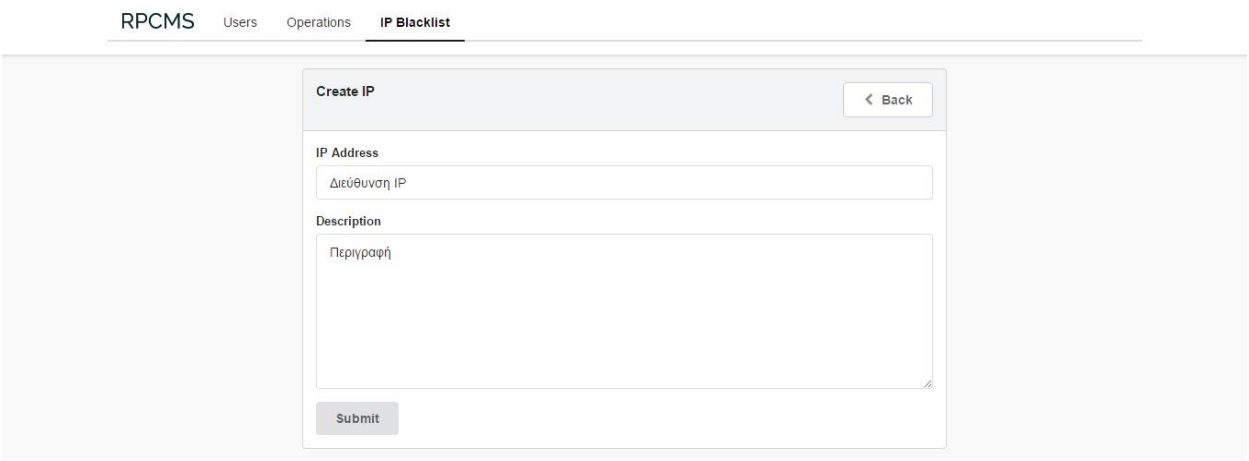
Εικόνα 3-2 User Management UI

Κατά την προσθήκη ενός χρήστη αφού έχουμε πατήσει το κουμπί Add User βάζουμε στα πεδία username και password του χρήστη που θέλουμε να προσθέσουμε. Επιλέγουμε στο checkbox αν αυτός ο χρήστης θα είναι administrator, δηλαδή αν θα μπορεί να προσθέτει και να διαχειρίζεται δικιές του λειτουργίες, και τέλος στην dropdown λίστα μπορούμε να επιλέξουμε σε ποιές άλλες εκτός των δικών του λειτουργιών θα έχει πρόσβαση μέσω του RPCMS. Πατώντας το κουμπί Submit καταχωρείται ο χρήστης στο σύστημα. Όλα αυτά φένονται παρακάτω.



Εικόνα 3-3 Inserting a User

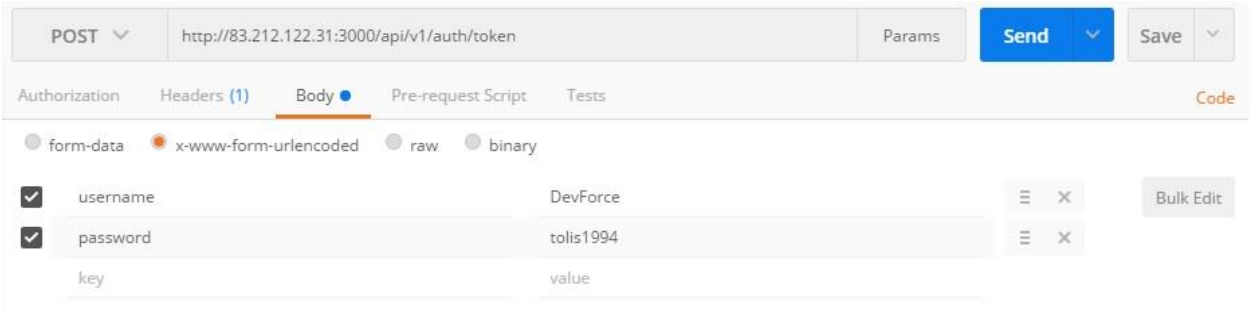
Θα προσπεράσουμε την διαχείρηση λειτουργιών καθώς υπάρχει ένα παράδειγμα παρακάτω στα σενάρια χρήσης. Θα προσπεράσουμε επίσης και το panel διαχείρησης του IP Blacklist αφού είναι ακριβώς ίδιο με των χρηστών και των λειτουργιών. Επείτα λοιπόν, είναι το IP Blacklist όπου υπάρχουν μόνο δύο πεδία να συμπληρώσουμε και αυτά είναι η IP διεύθυνση κάποιου που δεν θέλουμε να έχει πρόσβαση στο σύστημα μας καθόλου αλλά και η περιγραφή της. Κάθε απόπειρα αίτησης από τις IP που βρίσκονται στην blacklist απορρίπτονται αμέσως.



Εικόνα 3-4 Inserting an IP address to Blacklist

3.6 Παραδείγματα εκτέλεσης

Στο παράδειγμα παρακάτω βλέπουμε πως κάποιος μπορεί μέσω του API του RPCMS να κάνει μια κλήση και να πάρει δεδομένα από μια τρίτη εφαρμογή.



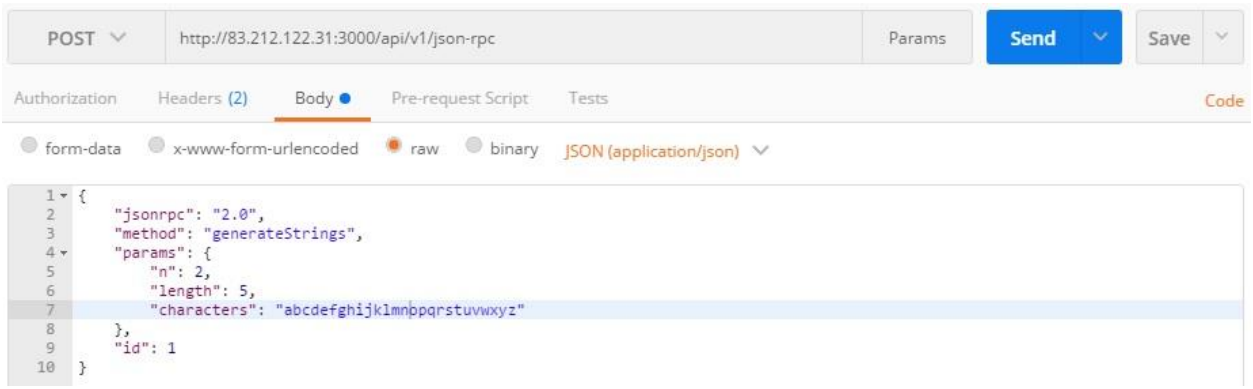
Εικόνα 3-5 Authentication API Call

Εικόνα 3-6 Authentication API Call Headers


```
{
  "success": true,
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
      .eyJ1c2VyX2lkIjoiNTgxYzYzZWE1ZGEzZWEzM2NkY2Y1YjliIiwidXNlcm5hbWUiOiJEZXZGb3JjZSIsIm9wZXJhdGlvbnMiOlsiNTgzMzZiOWI4MzA0MmI3MzI
      4N2U4NGNiIiwiNTg4MTAxNTg1MWNjNGYyZjQxZjJjYmIwIiwiNTg5Mzc0NGE1MWNjNGYyZjQxZjJjYmIyIl0sImFkbWluIjp0cnV1LCJpYXQiOjE0ODYzMjYwMjQ
      sImV4cCI6MTQ4NjQxMjQyNH0.kQECNT0uQX68XE4a88YDryJfCaWSee7CRK7eLZnn-ho"
}
```


Εικόνα 3-7 Authentication API Response

Με το παραπάνω κάνουμε το authentication και παίρνουμε το τεκμήριο, το οποίο θα χρησιμοποιήσουμε για να έχουμε πρόσβαση στις λειτουργίες που μπορούμε να καλέσουμε. Στην συνέχεια κάνουμε την κλήση στο RPCMS και αυτή την ανακατευθύνει στην κατάλληλη εφαρμογή, και όταν η τρίτη εφαρμογή επιστρέψει το αποτέλεσμα μας το εμφανίζει.



Εικόνα 3-8 JSON-RPC Call

Εικόνα 3-9 JSON-RPC Call Headers

Εικόνα 3-10 JSON-RPC Response

Για αυτές τις κλήσεις χρησιμοποιήθηκε το εργαλείο Postman του Google Chrome.

3.7 Οδηγός εγκατάστασης

Για το deployment του RPCMS είναι τα ακόλουθα:

- Node.js και NPM
- Angular-cli
- MongoDB

Θα αναφερθούμε μόνο στην έκδοση Debian των Linux για την εγκατάσταση του RPCMS. Αρχικά κατεβάζουμε και κάνουμε εγκατάσταση το Node.js μαζί με το NPM με τις οδηγίες από τον ακόλουθο σύνδεσμο: https://nodejs.org/en/download/package-manager/#debian-and-ubuntu-based-linux-distributions. Έπειτα, ακολουθούμε αυτόν τον οδηγό για την εγκατάσταση του MongoDB https://docs.mongodb.com/v3.0/tutorial/install-mongodb-on-debian/. Ακόμα, κάνουμε εγκατάσταση το Angular-cli χρησιμοποιώντας το NPM που μόλις εγκαταστήσαμε, τρέχοντας την εντολή: "npm install -g @angular/cli". Έχοντας τελειώσει αυτή την διαδικασία, κάνουμε clone το repository του RPCMS σε έναν κατάλογο της αρεσκείας μας και μέσα σε αυτόν τον φάκελο τρέχουμε την εντολή "npm install". Πρέπει να τρέξουμε την ίδια εντολή μέσα και στον υποκατάλογο "public". Η πρώτη φορά είναι για το Back-End και η δεύτερη για το Front-end. Όταν όλα εγκατασταθούν επιτυχώς, πηγαίνουμε στον φάκελο "public/src/app" και στο αρχείο "remote-urls.js" αλλάζουμε την μεταβλητή "prefixDev" και την αναθέτουμε στην IP διεύθυνση του server που κάνουμε deploy το σύστημα. Έχοντας διαμορφώσει το σύστημα μας, προχωράμε εκτελώντας την εντολή "ng build" στον φάκελο "public". Μόλις ολοκληρωθεί η διαδικασία, τρέχουμε την εντολή "export SERVER_IP=εδώ_την_ip_του_server" ώστε να μην γίνει deployed το project μόνο τοπικά και ύστερα πηγαίνουμε στον γονικό φάκελο και τρέχουμε την εντολή "npm start" και το project μας είναι έτοιμο και τρέχει.

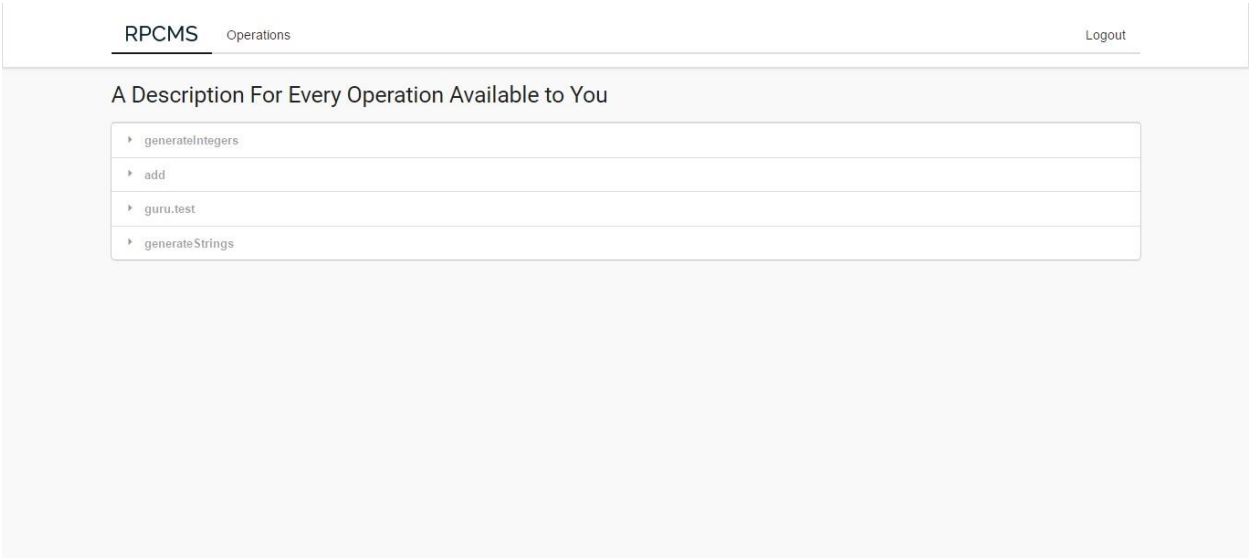
4 Σενάρια Χρήσης

Παρακάτω βλέπουμε ένα σενάριο χρήσης στο οποίο ένας χρήστης προσθέτει μία δική του λειτουργία στο σύστημα.

Αρχικά πρέπει να αυθεντικοποιηθεί:

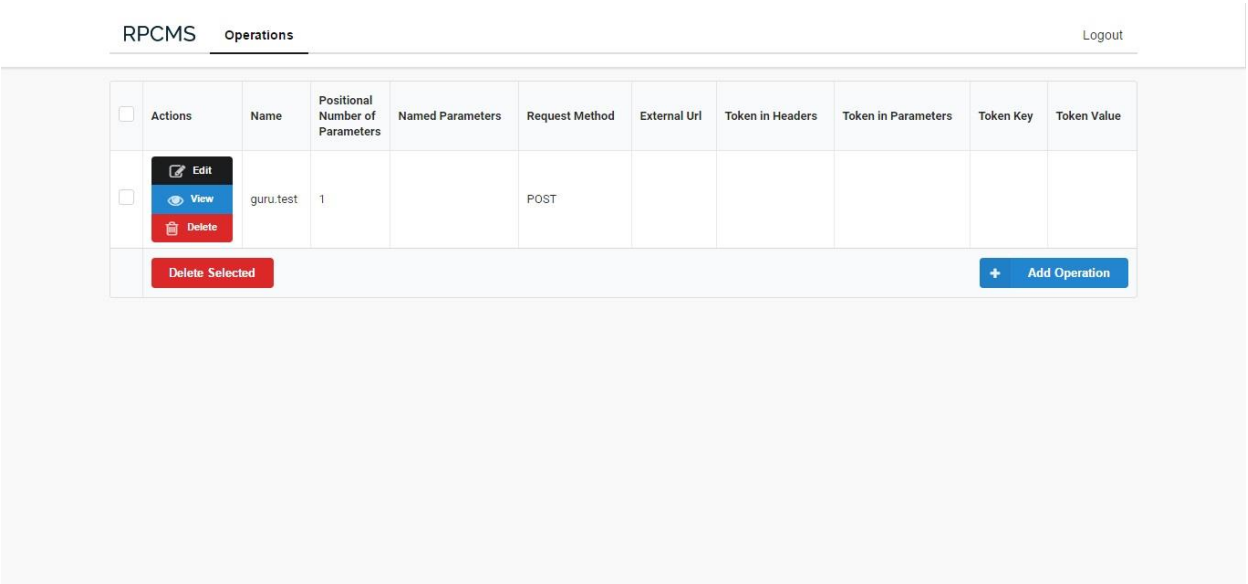
Εικόνα 4-1 Authentication UI

Μετά την επιτυχή συνδεσή του στο σύστημα, βλέπει τα ονόματα των λειτουργιών που μπορεί να εκτελέσει και άν πατήσει σε κάποια από αυτές του εμφανίζει την περιγραφή της.



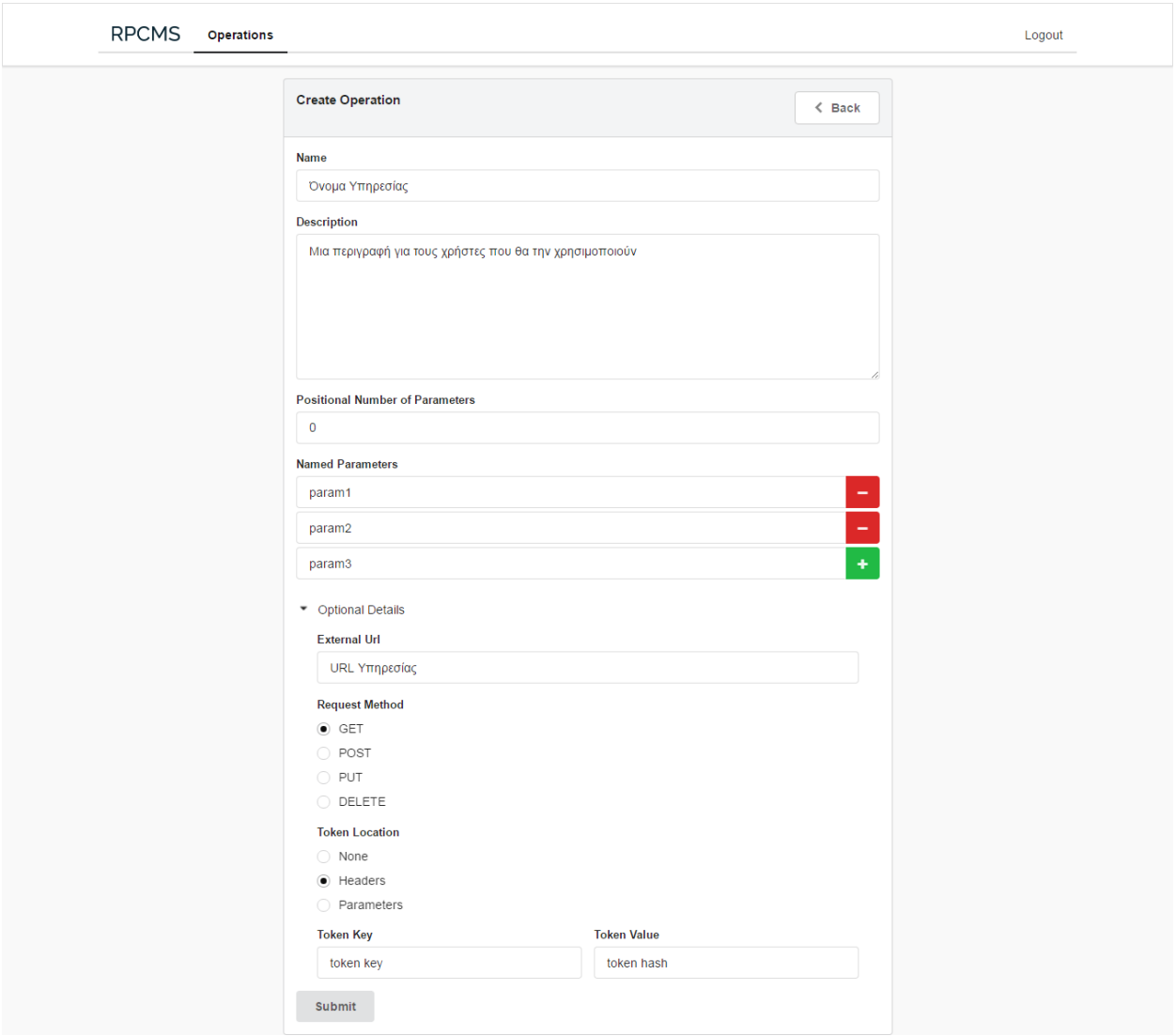
Εικόνα 4-2 Admin User Home Screen

Πατώντας στο μενού “Operations” του εμφανίζεται ο πίνακας με τις λειτουργίες που έχει προσθέσει ο ίδιος μόνο.



Εικόνα 4-3 Operation Management UI

Έπειτα, κάνωντας click στο κουμπί που βρίσκεται στο κάτω δεξιά μέρος του πίνακα "Add Operation" εμφανίζεται η φόρμα για την δημιουργία μιας νέας λειτουργίας.



Εικόνα 4-4 Inserting an Operation

Κάνοντας “Submit” η λειτουργία αποθηκέυεται στο σύστημα και οποιοσδήποτε του έχει δωθεί πρόσβαση σε αυτήν, μπορεί να στείλει ένα JSON-RPC request και να την εκτελέσει.

5 Συμπεράσματα

5.1 Πλεονεκτήματα

- Διαφάνεια (πρόσβασης και τοποθεσίας), δηλαδή απόκρυψη των πιθανών διαφορών στην αναπαράσταση των δεδομένων και του πώς προσπελαύνονται μεταξύ των κόμβων του συστήματος (διαφάνεια πρόσβασης) και απόκρυψη της πληροφορίας σχετικά με το σε ποιον κόμβο τοποθετείται ένας πόρος (διαφάνεια τοποθεσίας).
- Ασφάλεια, κάθε χρήστης έχει περιορισμένη πρόσβαση στις υπηρεσίες που προσφέρει το σύστημα όσων αφορά την διαχείριση και την εκτέλεσή τους. Επίσης αν κάποιος

κακόβουλος χρήστης πάρει πρόσβαση στο RPCMS οι υπόλοιποι διακομιστές παραμένουν ασφαλείς.

- Έχει μεγάλη ευελιξία στην χρήση, δηλαδή μπορεί να χρησιμοποιηθεί από επιχειρήσεις, ανοιχτού κώδικα εφαρμογές και όπου αλλού χρειάζεται να ελέγχονται πολλές υπηρεσίες από πολλούς κόμβους σε ένα ενιαίο σύστημα.

5.2 Προβληματισμοί - Μελλοντικές επεκτάσεις

- Δεν έχουμε κανέναν έλεγχο στο hardware υπηρεσιών τρίτων. Ας υποθέσουμε ότι ένας τρίτος προσθέτει μία λειτουργία στο συστημά μας. Αν αυτή η υπηρεσία αργεί να αποκριθεί δεν μπορούμε εμείς με κάποιο τρόπο εκτός του caching να επιταχύνουμε την διαδικασία όπως για παράδειγμα να αυξήσουμε την επεξεργαστική ισχύ ώστε να γίνει πιο γρήγορα η διαδικασία.
- Προς το παρόν, μόνο το JSON-RPC πρωτόκολλο υποστηρίζεται από το RPCMS.
- Προστίθεται ένα παραπάνω επίπεδο στην αρχιτεκτονική του συστήματος που εφαρμόζεται, δηλαδή ένας παραπάνω κόμβος στο υπάρχων σύστημα.
- Χρειάζεται μια μικρή διαμόρφωση πριν να ξεκινήσει κάποιος να το χρησιμοποιεί.

6 Μελλοντικές Επεκτάσεις

Αφού έχουν υλοποιηθεί όλα τα παραπάνω, καλό θα ήταν να γίνει και μια αναφορά στο πως θα μπορούσε να επεκταθεί αυτό το Project. Μετά από όλη την ανάπτυξη λοιπόν γεννήθηκαν οι εξής προτάσεις.

Υλοποίηση:

- Alias για τα operations που βάζουν οι χρήστες (να μπορούν να αλλάξουν το όνομα μιας υπηρεσίας τρίτου)
- Από Domain name σε IP διεύθυνση για το IP Blacklist
- Προαιρετικών named parameters (αυτήν την στιγμή πρέπει να υπάρχουν όλα τα named parameters στην αίτηση ώστε να εγκριθεί για προώθηση από το RPCMS)
- Περισσότερων αρχιτεκτονικών και μορφών δεδομένων (REST, RSS, XML κλπ.)

- Μηχανισμού ώστε μία μέθοδος να μπορεί να εκτελέσει πολλές άλλες (ασύγχρονα/σύγχρονα)
- Clustering, υλοποίηση:
    - πεδίου ονομάτων (ομάδες διακομιστών με ίδια λειτουργικότητα για δημιουργία clusters)
    - Load Balancing (μπορεί κάποιος να διαλέγει τον scheduler που θα χρησιποιήσει)
    - Fail-over
- Αλυσιδωτής εκτέλεσης μεθόδων (το output της μιας, στο input μιας άλλης)
- Caching Περιεχομένου
- Token invalidation με χρήση in-memory database (Redis πιθανότητα)
- Docker container ώστε να μπορεί εύκολα να γίνεται εγκατάσταση με όλα τα dependencies που χρειάζεται.

Όταν όλα τα παραπάνω υλοποιηθούν, το RPCMS θα είναι ένα εργαλείο που θα δίνει την δυνατότητα διαχείρισης και τις λειτουργίες που παρέχει ένας reverse proxy server μέσω ενός web interface αλλά σε επίπεδο υπηρεσιών και όχι σε HTTP requests με ότι αυτό συνεπάγεται.

7 Βιβλιογραφία

- TechTarget - service-oriented architecture (SOA) [online] Available from: http://searchsoa.techtarget.com/definition/service-oriented-architecture
- TechTarget – microservices [online] Available from: http://searchsoa.techtarget.com/definition/microservices
- TechTarget – aggregator [online] Available from: http://searchnetworking.techtarget.com/definition/aggregator
- TechTarget – proxy server [online] Available from: http://whatis.techtarget.com/definition/proxy-server
- Scotch.io - The Ins and Outs of Token Based Authentication [online] Available from: https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication
- JWT - Introduction to JSON Web Tokens [online] Available from: https://jwt.io/introduction/
- TechTarget – MongoDB [online] Available from: http://searchdatamanagement.techtarget.com/definition/MongoDB
- Βικιπαίδεια - Data access object, [online] Available from: https://en.wikipedia.org/wiki/Data_access_object/
- TechTarget – middleware [online] Available from: http://searchmicroservices.techtarget.com/definition/middleware
- Πανεπιστήμιο Μακεδονίας - Ενδιάμεσο Λογισμικό [online] Available from: http://pdplab.it.uom.gr/teaching/ince_2e_gr/Text/C3/Middleware_3.htm
- TechTarget – DMZ (demilitarized zone) [online] Available from: http://searchsecurity.techtarget.com/definition/DMZ