ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

Π.Μ.Σ. «ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗ»

Κατεύθυνση 1: Τεχνολογίες και Εφαρμογές Ιστού

Adaptive and usable object oriented stores

Δημήτρης Παπαβασιλείου

Αθήνα, 2017

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ
Π.Μ.Σ. «ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗ»
Κατεύθυνση 1: Τεχνολογίες και Εφαρμογές Ιστού

Τριμελής Εξεταστική Επιτροπή

Κωνσταντίνος Τσερπές
Επίκουρος Καθηγητής, Τμήμα Πληροφορικής και Τηλεματικής,
Χαροκόπειο Πανεπιστήμιο

Δημοσθένης Αναγνωστόπουλος
Καθηγητής, Τμήμα Πληροφορικής και Τηλεματικής,
Χαροκόπειο Πανεπιστήμιο

Γεώργιος Δημητρακόπουλος
Επίκουρος Καθηγητής, Τμήμα Πληροφορικής και Τηλεματικής,
Χαροκόπειο Πανεπιστήμιο

Ο Δημήτρης Παπαβασιλείου

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.

Περίληψη

Η παρούσα διπλωματική εργασία εκπονήθηκε στο πλαίσιο του μεταπτυχιακού Προγράμματος Σπουδών «Πληροφορική και Τηλεματική», του τμήματος Πληροφορικής και Τηλεματικής, της Σχολής Ψηφιακής Τεχνολογίας του Χαροκοπείου Πανεπιστημίου.

Στις σύγχρονες εφαρμογές ιστού, για τη διαχείριση των πληροφοριών και τη δημιουργία υπηρεσιών υλοποιούνται μια σειρά από βήματα. Τα σημαντικότερα από αυτά είναι η αποθήκευση αντικειμένων και ο έλεγχος τους ενώ παράλληλα πρέπει να κατασκευαστεί ένας μηχανισμός έκθεσης αυτών των αντικειμένων (συνήθως μέσω REST API) ώστε να μπορούν να χρησιμοποιηθούν από τη γραφική διεπαφή της εφαρμογής ή από άλλα συστήματα. Αν και αυτή η διαδικασία είναι μοντελοποιημένη, εντούτοις επαναλαμβάνεται σε όλες τις εφαρμογές. Στόχος της διπλωματικής είναι η μελέτη και κατασκευή ενός εργαλείου, που θα μπορεί να απλοποιήσει τη διαδικασία υλοποίησης μιας εφαρμογής, από το επίπεδο της αποθήκευσης των αντικειμένων μέχρι εκείνο της διαχείρισής τους μέσω υπηρεσιών.

Αναλυτικότερα, μελετήθηκαν τα χαρακτηριστικά του υπολογιστικού νέφους, που αποτελεί το περιβάλλον στο οποίο καλείται να ανταποκριθεί η εφαρμογή, εμβαθύνοντας σε ζητήματα κλιμάκωσης (scalability) και προγραμματιστικών αρχιτεκτονικών που μπορούν να ακολουθηθούν. Στη συνέχεια, έγινε ανάλυση των αντίστοιχων λύσεων που υπάρχουν ήδη και περιεγράφηκαν τα χαρακτηριστικά μιας εφαρμογής προσαρμοστικής διαχείρισης αντικειμένων.

Επιπροσθέτως, περιγράφονται εκτενώς όλα τα στάδια της υλοποίησης της παρεχόμενης υπηρεσίας με ανάλυση του λογισμικού που χρησιμοποιήθηκε για τη κατασκευή των υπηρεσιών, την αποθήκευση, την γραφική διεπαφή και τη δυνατότητα επέκτασης σε υπολογιστικό νέφος. Εν κατακλείδι προσφέρεται μια λύση ανοιχτού κώδικα που αντιμετωπίζει προβλήματα επαναλαμβανόμενου κώδικα σε διαφορετικές εφαρμογές παρέχοντας μια μέθοδο καθολικής διαχείρισης των δεδομένων.

Λέξεις κλειδιά: Υπολογιστικό Νέφος, Επεκτασιμότητα, Διαχείριση αντικειμένων, Δυναμική διαχείριση

Abstract

This diploma thesis has been prepared within the framework of post graduate studies “Informatics and Telematics” of Harokopio University of Athens.

In the modern web applications, a series of steps are made in order to manage information and create services. The most important of these is the persistence of objects and their validation while at the same time a mechanism of display of these objects must be created (usually via REST API) so that they can be used by graphic interface of this application and by other systems. Although this procedure is modelized, it is repeated in all applications. The aim of this thesis is the study and creation of a tool which will be able to simplify the procedure of an application from the level of the persistence of the objects up to their management via services.

In addition, the characteristics of the cloud computing have been studied. They comprise the environment in which the application is required to respond, examining more closely issues of scalability and programming architectures which may be followed. Consecutively, an analysis has been made of existing corresponding solutions as well as a description of the characteristics of an adaptive object management.

Moreover, a detailed description has been made concerning all the steps which are required to implement the services provided by analyzing the software that has been used for the construction of the services, the persistence, the graphic interface and the ability to scale in the cloud infrastructure. In conclusion, an open source solution has been provided in order to face problems of repeated code in different applications, thus creating a universal method of data management.

Key words: Cloud infrastructure, Scalability, Adaptive object management, Data store, Adaptive data store

1. Εισαγωγή

1.1. Γενικά

Στο πλαίσιο της παρούσας εργασίας σχεδιάστηκε και υλοποιήθηκε η εφαρμογή Revertan που παρέχει έναν εύχρηστο τρόπο μοντελοποίησης αντικειμένων και δημιουργεί υπηρεσίες διαχείρισης των δεδομένων μιας εφαρμογής. Αναλυτικότερα παρέχεται μια σύγχρονη και εύχρηστη γραφική διεπαφή για την δημιουργία τύπων αντικειμένων (παρέχοντας τη δομή που έχουν τα δεδομένα), φορμών για την εισαγωγή των αντικειμένων που έχει περιγράψει ο χρήστης και υπηρεσιών για τη διενέργεια βασικών λειτουργειών όπως δημιουργίας, ανάκτησης, ενημέρωσης και διαγραφής δεδομένων.

1.2. Σκοπός

Η παρούσα εργασία έχει ως βασικό σκοπό να αποδείξει ότι με τη χρήση σύγχρονων και ανοικτών τεχνολογιών μπορεί να δημιουργηθεί λογισμικό που θα απλοποιεί την ανάπτυξη μελλοντικών εφαρμογών ιστού, μειώνοντας το χρόνο υλοποίησης, κυρίως, σε εφαρμογές νέφους που χρειάζονται ένα επεκτάσιμο (Scalable) περιβάλλον λειτουργίας.

1.3. Αντικείμενο

Στις σύγχρονες εφαρμογές ιστού, για τη διαχείριση των πληροφοριών και τη δημιουργία υπηρεσιών υλοποιούνται μια σειρά από βήματα. Τα σημαντικότερα από αυτά είναι η αποθήκευση αντικειμένων και ο έλεγχος τους, ενώ παράλληλα πρέπει να κατασκευαστεί ένας μηχανισμός έκθεσης αυτών των αντικειμένων (συνήθως μέσω REST API) ώστε να μπορούν να χρησιμοποιηθούν από τη γραφική διεπαφή της εφαρμογής ή από άλλα συστήματα. Αν και αυτή η διαδικασία είναι μοντελοποιημένη, εντούτοις επαναλαμβάνεται σε όλες τις εφαρμογές. Αντικείμενο της

διπλωματικής είναι η μελέτη και κατασκευή ενός εργαλείου, που θα μπορεί να απλοποιήσει τη διαδικασία υλοποίησης μιας εφαρμογής, από το επίπεδο της αποθήκευσης των αντικειμένων μέχρι εκείνο της διαχείρισής τους μέσω υπηρεσιών.

1.4. Δομή Εργασίας

Η παρούσα εργασία είναι οργανωμένη σε οχτώ(8) κεφάλαια:

Στο κεφάλαιο 1, περιλαμβάνονται τα γενικά στοιχεία για το αντικείμενο της παρούσας εργασίας και αναφέρονται περιληπτικά ο σκοπός, το αντικείμενο και η δομή της αυτής.

Στο 2ο κεφάλαιο, περιγράφονται βασικές έννοιες συστημάτων μεγάλης κλίμακας, όπως αυτό της επεκτασιμότητας μιας εφαρμογής, καθώς και τα χαρακτηριστικά σύγχρονων υπολογιστικών νεφών (Cloud computing). Επιπροσθέτως, παρουσιάζονται οι σύγχρονες αρχιτεκτονικές στο επίπεδο των διακομιστών και το πως επηρεάζουν την αποδοτικότητα μιας εφαρμογής. Το 2ο κεφάλαιο, εν κατακλείδι, ασχολείται με την καταγραφή σύγχρονων λύσεων του Cloud για τη διαχείριση δεδομένων.

Στο κεφάλαιο 3, περιγράφονται οι απαιτήσεις που πρέπει να ικανοποιεί ένα περιβάλλον ιστού καθώς και οι σχεδιαστικές προκλήσεις που έχει να αντιμετωπίσει. Προτείνεται, συγχρόνως, ο συνδυασμός ενός ολοκληρωμένου συνόλου λογισμικού προσαρμοσμένο για την κάλυψη όλων των αναγκών αλλά και των μελλοντικών επεκτάσεων της εφαρμογής.

Το κεφάλαιο 4 αναφέρεται στο μοντέλο δεδομένων που προτείνεται για ένα σύστημα προσαρμοστικής διαχείρισης πληροφορίας, ενώ το 5ο κεφάλαιο ασχολείται με το σύνολο των υπηρεσιών που τελικώς υλοποιήθηκε.

Το 6ο κεφάλαιο είναι αφιερωμένο στην υλοποίηση που έγινε με ανάλυση της αρχιτεκτονικής και της γραφικής διεπαφής.

Στο κεφάλαιο 7, καταγράφονται τα συμπεράσματα της παρούσας διπλωματικής καθώς και οι μελλοντικές επεκτάσεις που μπορούν να υλοποιηθούν

Τέλος, στο κεφάλαιο 8 παρατίθεται η βιβλιογραφία που χρησιμοποιήθηκε.

2. Χαρακτηριστικά υπολογιστικού νέφους και εφαρμογές

2.1. Γενικά

Το υπολογιστικό νέφος είναι η αποθήκευση, η επεξεργασία και η χρήση δεδομένων από απομακρυσμένους υπολογιστές στους οποίους εξασφαλίζεται πρόσβαση μέσω του διαδικτύου. Ειδικότερα, στα χαρακτηριστικά ενός υπολογιστικού νέφους υπάρχει η εξυπηρέτηση αιτημάτων κατ' απαίτηση, άμεσα και χωρίς καθυστερήσεις. Με τη πρόσβαση ενός υπολογιστικού νέφους στο διαδίκτυο, αυτό καλείται να παρέχει υπηρεσίες σε οποιαδήποτε συσκευή. Παράλληλα, τα νέφη αποτελούν ευέλικτα συστήματα που χειρίζονται αυξημένους υπολογιστικούς πόρους και καλούνται να αντιμετωπίσουν περιόδους ιδιαίτερα αυξημένου φόρτου. Για να πετύχουν το σκοπό τους, κύρια αρχή που καλούνται να υλοποιούν είναι εκείνη της επεκτασιμότητας, δηλαδή της λειτουργίας αυτής που τους επιτρέπει να αξιοποιούν συστάδες υπολογιστών, αθροίζοντας τους πόρους τους.

2.2. Υπολογιστικά συστήματα νέφους

Τα υπολογιστικά συστήματα νέφους (cloud computing) αποτελούν ένα χώρο που διαρκώς εξελίσσεται, προσελκύοντας όλο και μεγαλύτερη προσοχή τα τελευταία χρόνια, τόσο για τις τεχνολογίες που αναπτύσσονται, όσο και για τη δυναμική που εμφανίζει στην οικονομία. Αξιοσημείωτη είναι, επίσης, η θετική στάση που διατηρούν τα μέσα ενημέρωσης καθώς και οι επιστήμονες της πληροφορικής απέναντι στις ευκαιρίες που προσφέρει το cloud computing.

Είναι γεγονός ότι αρκετές μελέτες έχουν πραγματωθεί ώστε να δοθεί ένας σαφής ορισμός για το Cloud computing [1, 2], εντούτοις οι διαφορετικοί ορισμοί έχουν σημεία σύγκλησης. Το Cloud computing αναφέρεται στη δυνατότητα ενός υπολογιστικού συστήματος να παρέχει πόρους υλικού και λογισμικού ως υπηρεσία σε ένα κλιμακούμενο περιβάλλον. Αρχιτεκτονικά, το κύριο μέρος μιας Cloud υποδομής είναι το κέντρο δεδομένων (data center) που αποτελείται από υπολογιστικούς και αποθηκευτικούς πόρους, δηλαδή υλικό, αλλά και από λογισμικό που παρέχεται επί μίσθωση. Σύμφωνα με το Berkeley, αυτές οι υπηρεσίες που παρέχονται μέσω του διαδικτύου, αναφέρονται

ως «Λογισμικό ως υπηρεσία» -“Software as a Service (SaaS)”. Το υλικό και το λογισμικό αναφέρονται ως Cloud. Όταν οι υπηρεσίες Cloud είναι διαθέσιμες επί πληρωμή στο ευρύ κοινό τότε αυτές αποτελούν ένα δημόσιο Cloud (Public Cloud). Με τη χρήση του όρου ιδιωτικό Cloud (Private Cloud) αναφερόμαστε σε εσωτερικές υπολογιστικές υποδομές εταιριών ή άλλων οργανισμών που δεν είναι διαθέσιμα στο ευρύ κοινό. Βέβαια, στο γενικότερο όρο του Cloud περιλαμβάνονται οι υπηρεσίες SaaS μόνο των Public Cloud υποδομών και όχι εκείνων των Private Cloud.[3]

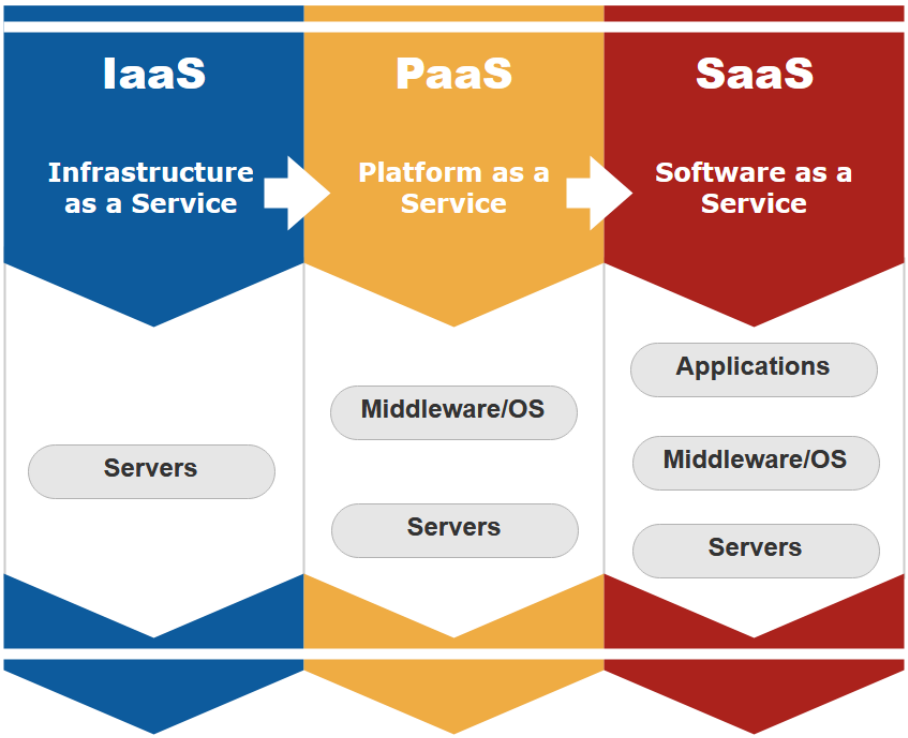
Από πλευράς υλικού, οι Cloud υποδομές διακρίνονται από τρία βασικά χαρακτηριστικά:

- Οι χρήστες των Cloud υποδομών δεν χρειάζεται να σχεδιάζουν μόνοι τους την ανάπτυξη της υπολογιστικής τους υποδομής. Το Cloud δίνει την ψευδαίσθηση των απεριόριστων υπολογιστικών πόρων στους χρήστες του.
- Οι χρήστες μπορούν να αυξομειώνουν το μέγεθος των πόρων που ζητούν από το Cloud, αυξάνοντας την υπολογιστική ισχύ, τη μνήμη ή τον αποθηκευτικό χώρο σύμφωνα με τις ανάγκες τους.
- Οι πάροχοι Cloud υπηρεσιών προσφέρουν μοντέλα πληρωμών τύπου pay-as-you-go, αποδεσμεύοντας επεξεργαστές και πόρους αποθήκευσης όταν το σύστημα δεν τους χρειάζεται ώστε να μειωθούν τα έξοδα.

Συμπερασματικά, στο πλαίσιο μιας πλατφόρμας διαχείρισης δεδομένων, όπου το μέγεθος και η πολυπλοκότητά τους μπορεί να τα ταξινομεί στην κατηγορία των Big Data, το Cloud μπορεί να αποτελέσει μια αξιόλογη λύση. Η παροχή απεριόριστων πόρων ανοίγει το δρόμο στην επεξεργασία Big Data για τους χρήστες που δεν έχουν τη δυνατότητα να δημιουργήσουν της δικές τους Scalable υποδομές (Συστάδες υπολογιστών). Με την ενοικίαση πόρων, οι χρήστες μπορούν να εκτελέσουν επεξεργασίες στα δεδομένα τους με οικονομικό τρόπο.

2.3. Αρχιτεκτονική Cloud υποδομής

Αρχιτεκτονικά, στο Cloud computing μπορούν να διακριθούν τρία επίπεδα όπως μπορούμε να δούμε στο σχήμα 2.1.



Σχήμα 2.1 Επίπεδα Cloud Computing

Αναλυτικότερα:

- **Infrastructure as a Service (IaaS).** Με τον όρο IaaS αναφερόμαστε σε παροχή υλικού υπολογιστή στο πλαίσιο υπηρεσίας. Πιο συγκεκριμένα, το IaaS προσφέρει υλικό όπως υπολογιστική ισχύ, μνήμη, δίκτυο και δίσκο κατά παραγγελία με τη μορφή εικονικών πόρων. Οι χρήστες IaaS Cloud, συνήθως, ενοικιάζουν εικονικές μηχανές, με χαρακτηριστικά αντίστοιχα των αναγκών τους, στις οποίες δίνεται η δυνατότητα επιλογής του επιθυμητού λειτουργικού συστήματος και εκτέλεσης λογισμικού της επιλογής τους. Τέλος, τα τέλη χρήσης της υπηρεσίας χρεώνονται με μοντέλα πληρωμών που αντικατοπτρίζουν την πραγματική ποσότητα των πόρων που καταναλώθηκαν, όπως για παράδειγμα κύκλοι της

CPU ανά ώρα. Παραδείγματα IaaS Cloud υποδομών είναι: Amazon Elastic Compute Cloud [4], Nimbus [5] και OpenNebula [6].

- **Platform as a Service (PaaS).** Το επίπεδο PaaS αποτελεί μια επέκταση του IaaS, για να παρέχει μια πιο υψηλού επιπέδου υπολογιστική πλατφόρμα, η οποία διαθέτει από μόνη της το λειτουργικό σύστημα, τη βάση δεδομένων και γενικότερα ένα περιβάλλον για την εκτέλεση εφαρμογών. Το PaaS στοχεύει στην κατηγορία των μηχανικών λογισμικού, επιτρέποντάς τους να σχεδιάζουν τις εφαρμογές τους χρησιμοποιώντας συγκεκριμένα πακέτα λογισμικού. Επιπροσθέτως, η πλατφόρμα παρέχει όλα τα εργαλεία που είναι απαραίτητα για την κατασκευή του λογισμικού, ενώ ο προγραμματιστής δεν ασχολείται με το υλικό, το οποίο συνήθως ελέγχεται από μια IaaS πλατφόρμα, στην οποία δεν έχει πρόσβαση ο χρήστης. Παραδείγματα PaaS είναι: Google App Engine [7], Microsoft Azure [8].
- **Software as a Service (SaaS).** Στο υψηλότερο επίπεδο της κλίμακας βρίσκεται το SaaS. Χαρακτηριστικά, αποτελεί το πιο ολοκληρωμένο επίπεδο για τους τελικούς χρήστες. Αυτό βρίσκει έρεισμα στο γεγονός ότι παρέχει ολοκληρωμένο λογισμικό ως υπηρεσία, επιτρέποντας στους χρήστες την αξιοποίηση εφαρμογών που έχουν αναπτυχθεί στο Cloud. Οι χρήστες δεν εγκαθιστούν λογισμικό στον υπολογιστή τους, ούτε είναι υπεύθυνοι για τη λήψη ενημερώσεων, αλλά απλώς, κάνουν χρήση των εφαρμογών συνήθως μέσω περιηγητών Ιστού (Web browser). Στα παραδείγματα SaaS συγκαταλέγονται: Google Docs [9] and Microsoft Office Live [10]

2.4. Scalability

Ως κλιμακοθετησιμότητα ή αλλιώς επεκτασιμότητα (Scalability) ορίζεται ως η ικανότητα ενός συστήματος, δικτύου ή διεργασίας να χειρίζεται με ικανό τρόπο τον διαρκώς αυξανόμενο όγκο εργασίας, ή τη δυνατότητά του να διευρύνεται για να εξυπηρετήσει τις περαιτέρω απαιτήσεις.[15]

Σύμφωνα με τον ανωτέρω ορισμό, η δυνατότητα επεκτασιμότητας ενός συστήματος αφορά ένα σύστημα συγκεντρωτικά και όχι μια μεμονωμένη ιδιότητα που μπορεί να έχει. Επί τη παραδείγματι, η κλιμακούμενη απόδοση δεδομένων ενός συστήματος αναφέρεται στην ικανότητά του να αυξάνει την απόδοσή του όταν προστίθενται νέοι πόροι για να χειριστεί ένα αυξημένο φόρτο εργασίας.

Ένα σύστημα που έχει κακή επεκτασιμότητα, μπορεί να οδηγήσει, επακολούθως, σε κακή απόδοση. Σε πολλές περιπτώσεις, μάλιστα, η προσθήκη περισσότερων πόρων σε ένα μη κλιμακούμενο σύστημα αποτελεί κακή επιλογή που δε δύναται, εν τέλει, να οδηγήσει σε ουσιαστικές βελτιώσεις.

Προφανώς, σε ένα σύστημα που καλείται να διαχειρίζεται δεδομένα μεγάλης κλίμακας απαιτείται η εξαρχής σωστή σχεδίαση ώστε να μπορούν να δημιουργηθούν πρόσφοροι όροι επεκτασιμότητάς του. Τέλος, υπάρχουν δύο είδη επεκτασιμότητας, η οριζόντια και η κάθετη κλιμάκωση οι οποίες θα αναλυθούν παρακάτω.

2.4.1. Οριζόντια κλιμάκωση

Σε αυτή τη προσέγγιση στόχος είναι να προστίθενται περισσότεροι κόμβοι σε ένα σύστημα, εν προκειμένω νέοι υπολογιστικοί κόμβοι, δημιουργώντας μια συστάδα υπολογιστών. Με τη πτώση των τιμών των τελευταίων, μια ισχυρή συστάδα υπολογιστών μπορεί να κατασκευαστεί με τη χρήση κοινών (commodity) υπολογιστών χαμηλού κόστους συνδεδεμένων σε ένα τοπικό δίκτυο. Ακολουθώντας μια «διαίρει και βασίλευε» λογική, όπου σε κάθε κόμβο εκχωρείται μόνο ένα υποσύνολο του συνολικού προβλήματος, το υπολογιστικό σύστημα μπορεί να κλιμακώνεται ως ένα βαθμό και ανάλογα με το πρόβλημα, ώστε να προσαρμόζεται στις ανάγκες.

2.4.2. Κάθετη κλιμάκωση

Στην περίπτωση της κάθετης κλιμάκωσης οι πόροι προστίθενται σε μερικούς κόμβους ενός συστήματος, δηλαδή πρακτικά μεγαλύτερη υπολογιστική ισχύ και περισσότερη μνήμη σε κάθε κόμβο. Η τακτική αυτή διαθέτει στις εφαρμογές που εκτελούνται περισσότερους πόρους για κατανάλωση. Με την προσθήκη περισσότερων επεξεργαστών μπορούν περισσότερες διεργασίες να εκτελούνται παράλληλα ενώ με την προσθήκη μνήμης ελαχιστοποιούμε την πρόσβαση στο δίσκο.

2.5. Εικονικοποίηση (Virtualization)

Η εικονικοποίηση (Virtualization) αποτελεί μια δομική τεχνολογία που χρησιμοποιείται κατά κόρων στο Cloud Computing. Με τον όρο εικονικοποίηση αναφερόμαστε στο λογισμικό εκείνο που κρύβει τους πόρους που βρίσκονται διασκορπισμένοι σε υπολογιστές και τους παρουσιάζει ως έναν ενιαίο υπολογιστή στις εφαρμογές και τους χρήστες του Cloud [11]. Εντοπίζονται διαφορετικά επίπεδα virtualization όπως:

- **Υλικού**. Εικονικοποίηση ολόκληρου του υλικού του κάθε υπολογιστή.
- **Δικτύου**. Συνδυασμός δικτυακών πόρων υλικού και λογισμικού σε ένα ενιαίο εικονικό δίκτυο.
- **Αποθήκευσης**. Ομαδοποίηση των φυσικών πόρων αποθήκευσης από πολλαπλές αποθηκευτικές συσκευές δημιουργώντας μια ενιαία εικονική συσκευή αποθήκευσης. Η συσκευή αυτή μπορεί να χωριστεί ξανά σε εικονικές συσκευές, καταργώντας τις εξαρτήσεις μεταξύ φυσικής αποθήκευσης των δεδομένων και της εικονικοποιημένης θέσης τους.
- **Μνήμης**. Εικονικοποίηση της μνήμης στα επίπεδα των λειτουργικών συστημάτων και εφαρμογών.

2.6. Αρχιτεκτονικές διακομιστών στις εφαρμογές

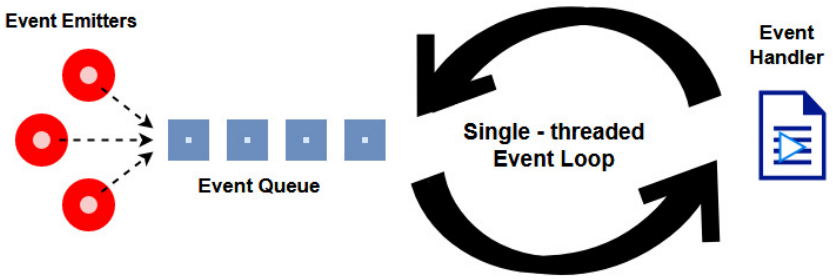
Παραδοσιακά, υπάρχουν δύο ανταγωνιστικές αρχιτεκτονικές που αφορούν την αρχιτεκτονική του λογισμικού που τρέχει σε μια Cloud υποδομή. Από τη μια πλευρά, είναι η αρχιτεκτονική Νημάτων (Thread Architecture) και η άλλη, αυτή που αφορά το προγραμματισμό χειρισμού γεγονότων (Event Driven Architecture). Οι προκλήσεις που συναντούν τα σύγχρονα υπολογιστικά συστήματα κρίνουν αναγκαία τη μελέτη των δύο αυτών αρχιτεκτονικών, για την επιλογή της κατάλληλης, καθώς αποτελεί δομικό στοιχείο μιας σύγχρονης εφαρμογής και μπορεί να κρίνει σε σημαντικό βαθμό τις επιδόσεις της στο Cloud.

2.6.1. Αρχιτεκτονική νημάτων

Στην αρχιτεκτονική νημάτων η σημαντικότερη εφαρμογή είναι εκείνη των διακομιστών πολλαπλών νημάτων (Multi-thread servers) οι οποίοι χρησιμοποιούν ένα thread ανά σύνδεση. Αυτό το χαρακτηριστικό, τους έχει κάνει ευρέως γνωστούς κυρίως λόγω της εύκολης υλοποίησης για την κατασκευή υπηρεσιών διαδικτύου. Αναλυτικότερα, το λειτουργικό σύστημα αναλαμβάνει την επικάλυψη πολλαπλών νημάτων μέσω χρονοπρογραμματισμού προεκχώρισης. Στις περισσότερες περιπτώσεις αρκεί μια χρονοβόρα διαδικασία για να μπλοκάρει το νήμα στο οποίο εκτελείται, και να ενεργοποιήσει τον επαναπρογραμματισμό, προκαλώντας τη διακοπή της προκειμένου να συνεχίσουν τα υπόλοιπα νήματα τη λειτουργεία τους. Γενικά, θεωρείται ένα σεβαστό μοντέλο για αξιοπρεπείς επιδόσεις και είναι ιδανικό όταν ο επεξεργαστής καλείται να εκτελέσει μια εύλογη ποσότητα λειτουργιών. Επιπλέον, πολλαπλοί πυρήνες του επεξεργαστή μπορούν να χρησιμοποιούνται απευθείας, καθώς τα νήματα και οι διεργασίες προγραμματίζονται αποδοτικά στους πυρήνες που είναι διαθέσιμοι.

2.6.2. Αρχιτεκτονική χειρισμού γεγονότων

Εναλλακτική επιλογή στην αρχιτεκτονική νημάτων αποτελεί αυτή του χειρισμού γεγονότων (event driven) η οποία είναι εξίσου διαδεδομένη. Λόγω της ασύγχρονης λογικής της, απαιτείται διαφορετικός χειρισμός των προς εκτέλεση λειτουργιών. Μια συνήθης τακτική είναι η ανάθεση σε ένα νήμα πολλαπλών συνδέσεων. Το νήμα με τη σειρά του χειρίζεται όλα τα συμβάντα που προκύπτουν από την επεξεργασία των αιτημάτων. Όπως φαίνεται και στο σχήμα 2.2, μια ουρά συμβάντων και τα νήματα, εκτελούν διεργασίες κάθε φορά που κάποιο αίτημα εισέρχεται σε αυτή την ουρά.



Σχήμα 2.2 Εννοιολογική παρουσίαση συμβάντων μιας Event-Driven αρχιτεκτονικής.

2.6.3. Συμπεράσματα αρχιτεκτονικής διακομιστών

Η μη συσχέτιση των συνδέσεων και των νημάτων μπορεί να μειώσει δραστικά τον αριθμό των νημάτων που απασχολούνται στον διακομιστή. Συνεπώς, αν απαλλαγούμε από το άσκοπο μπλοκάρισμα των νημάτων, δεν θα χρειαζόμαστε μια στοίβα νημάτων για κάθε σύνδεση. Το γεγονός αυτό επηρεάζει θετικά το μέγεθος της απαιτούμενης μνήμης, ενώ και ο επεξεργαστής σπαταλά λιγότερους πόρους. Η αρχιτεκτονική χειρισμού γεγονότων διαφαίνεται ως ιδανική λύση, που κρατά χαμηλά τα επίπεδα χρήσης του επεξεργαστή και της μνήμης, αφήνοντας σαν μοναδικό παράγοντα συμφόρησης, σε ένα σύγχρονο υπολογιστικό σύστημα, τη βάση δεδομένων.

2.7. Ανάλυση παροχών cloud διαχείρισης δεδομένων

Ο αριθμός των παρόχων Cloud υπηρεσιών παρουσιάζει σταθερή αύξηση τα τελευταία δέκα χρόνια [12]. Δεδομένου ότι δεν είναι στο πεδίο αυτής της εργασίας να συγκριθούν όλοι οι υπάρχοντες πάροχοι υπηρεσιών Cloud, θα αναλυθούν οι μεγαλύτεροι από αυτούς ως προς το κομμάτι των δυνατοτήτων αποθήκευσης, διαχείρισης και ανάλυσης δεδομένων.

2.7.1. Amazon Simple Storage Service

Το Amazon Simple Storage Service (Amazon S3) παρέχει μια υποδομή αποθήκευσης δεδομένων ως αντικείμενα μέσα σε πόρους που ονομάζονται «buckets» (δηλαδή κουβάδες) [13]. Υποστηρίζει τρία διαφορετικά είδη αποθήκευσης δεδομένων:

- **Standard.** Για αποθήκευση γενικού σκοπού
- **Standard-IA.** Για αποθήκευση δεδομένων που δεν θα ζητούνται συχνά από το σύστημα, οπότε θα υπάρχει μικρή προσπέλαση
- **Glacier.** Για την αποθήκευση δεδομένων συνήθως ιστορικού χαρακτήρα αφού παρέχει δυνατότητες αποθήκευσης και ανάλυσης χωρίς καμία άλλη επεξεργασία.

Το Amazon S3 αποτελείται από δύο μέρη. Αυτό που αφορά την παρεχόμενη διεπαφή τύπου Rest και εκείνο της αποθήκευσης των δεδομένων σε μια scalable βάση δεδομένων που η ίδια η Amazon έχει κατασκευάσει, την DynamoDB. Τα αντικείμενα που αποθηκεύονται στην DynamoDB είναι προσβάσιμα μέσω των hash keys τους, αλλά μπορούν επιπλέον να εκτελεστούν ερωτήματα σε αυτά με χρήση δευτερευόντων ευρετηρίων (secondary indexes). Η προβλεπόμενη απόδοση ενός τέτοιου συστήματος μπορεί να κλιμακωθεί αναλόγως με τη ζήτηση για υπολογιστική ισχύ κατά την επεξεργασία των αιτημάτων.

Βέβαια, ένας περιορισμός του Amazon S3 που προκύπτει από τη χρήση της DynamoDB στον πυρήνα του αφορά το μέγεθος των αντικειμένων. Το κάθε αντικείμενο δεν επιτρέπεται να υπερβαίνει τα 64KB σε μέγεθος, ώστε να εξασφαλιστεί χαμηλό latency και data replication των δεδομένων. Επιπροσθέτως, επειδή η DynamoDB είναι μια πιο πολύπλοκη βάση από ένα απλό key-value store, προϋποθέτει ότι οι χρήστες έχουν άριστη γνώση της τεχνολογίας καθώς η διαχείριση γίνεται εξολοκλήρου μέσω API χωρίς κάποια γραφική διεπαφή.

2.7.2. Amazon Relational Database Service

Το Amazon Relational Database Service (Amazon RDS) είναι μια λύση που παρέχει σχεσιακές βάσεις δεδομένων στο Cloud. Υποστηρίζει πλήθος σχεσιακών βάσεων όπως Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle, and Microsoft SQL Server [14]. Δε διατίθεται κάποια υπηρεσία που να παρέχει δυνατότητες επεξεργασίας και ανάκτησης των δεδομένων. Ουσιαστικά, φιλοξενεί βάσεις δεδομένων κατά παραγγελία, φροντίζοντας για την παραμετροποίηση, εγκατάσταση και ασφάλειά τους.

2.7.3. Azure Cosmos DB

Το Azure Cosmos DB αποτελεί την πρόταση της Microsoft στη παροχή υπηρεσίας αποθήκευσης δεδομένων στο Cloud. Επιτρέπει στους χρήστες του να δημιουργούν ελαστικά και ανεξάρτητος απόδοσης εικονικές μηχανές παρέχοντας εγγυήσεις στις χαμηλές καθυστερήσεις και την διαθεσιμότητά τους.

Παρέχει ευρεία γεωγραφική κάλυψη ενώ δίνει, παράλληλα, την επιλογή στο χρήστη να επιλέξει τι είδους βάση θέλει να χρησιμοποιήσει. Οι βάσεις δεδομένων που υποστηρίζονται είναι οι: DocumentDB, MongoDB, Azure Table και Azure Graph. Για τις βάσεις δεδομένων Azure Table και DocumentDB παρέχεται Rest API διαθέτοντας διαδικασίες αποθήκευσης και σύνθετων ερωτημάτων, ενώ για τις υπόλοιπες υποστηρίζεται ένα πλήθος APIs διαφορετικών γλωσσών προγραμματισμού.

2.8. Συμπεράσματα παρεχόμενων λύσεων

Οι λύσεις που παρέχονται σχετικά με την προσαρμοστική αποθήκευση αντικειμένων στο Cloud είναι ποικίλες και διαφορετικές μεταξύ τους. Η διαφορετικότητα δεν περιορίζεται μόνο στο γεγονός ότι ανήκουν σε διαφορετικές εταιρίες με διαφορετικά πακέτα ενοικίασης των υπηρεσιών τους,

αλλά στον τρόπο που διαθέτουν τα αντικείμενα μιας εφαρμογής. Η επικρατούσα τάση, πάντως, είναι η διάθεση των αντικειμένων αλλά και ο γενικότερος χειρισμός τους να γίνεται μέσω υπηρεσίες τύπου REST (Representational State Transfer), που σαν τεχνολογία δεν έχει αυστηρές προδιαγραφές. Το γεγονός αυτό από τη μια δίνει ευελιξία, από την άλλη όμως προκαλεί σύγχυση (για παράδειγμα τι γίνεται στις περιπτώσεις όπου για μια σειρά από λόγους, ένας οργανισμός αποφασίζει να αλλάξει πάροχο;) Τρόποι να επιτευχθεί το αναγκαίο data migration υπάρχουν πολλοί, όμως τα συστήματα που χρησιμοποιούσαν τις προηγούμενες υπηρεσίες χρειάζονται σημαντικές αλλαγές για να αποκτήσουν πρόσβαση στα δεδομένα τους, σύμφωνα με τις προδιαγραφές που ακολουθεί η καινούργια υπηρεσία.

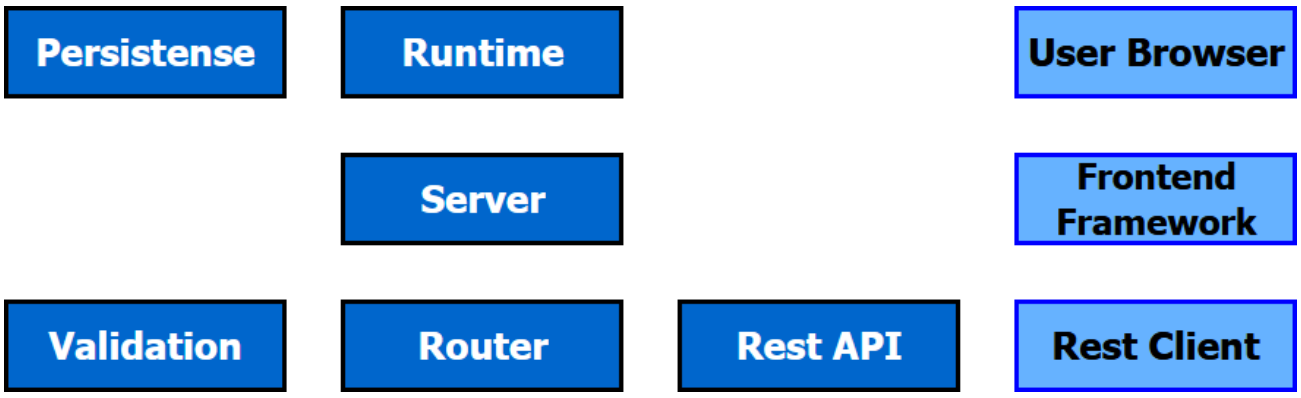
Ένα άλλο ζήτημα που προκύπτει με τις υπάρχουσες λύσεις είναι ότι βάζουν ένα όριο στην απόδοση των συστημάτων με βάση τα χρήματα που καταβάλλονται. Αν το σύστημα χρειάζεται να καλύψει υπερβολικά μεγάλη ζήτηση, το Cloud απαιτεί πολύ περισσότερα χρήματα, «τιμωρώντας» ουσιαστικά τους χρήστες είτε ανεβάζοντας το κόστος, είτε αυξάνοντας τις καθυστερήσεις στη διεκπεραίωση των αιτημάτων. Υπάρχουν, βέβαια, περιπτώσεις που η μετάβαση σε dedicated υλικό είναι πιο οικονομικές και αξιόπιστες για τη κλιμάκωση μιας εφαρμογής.

Συναθροίζοντας τα ανωτέρω, προκύπτει η ανάγκη για πιο ελαστικές σχέσεις με τους παρόχους και ιδανικά θα ήταν επιθυμητό να παρέχονται οι ίδιες υπηρεσίες στο επίπεδο προτυποποίησής τους από διαφορετικά συστήματα, ώστε να είναι εφικτή η αλλαγή παρόχου. Δεν υπάρχει, όμως, η δυνατότητα, το λογισμικό που τρέχει στο Cloud και παρέχει μέσω λογισμικού έτοιμες υπηρεσίες επεξεργασίας των αντικειμένων (SaaS), να εκτελεστεί εκτός της Cloud υποδομής του παρόχου. Είτε πρόκειται για λογισμικά κλειστού κώδικα, είτε είναι κατασκευασμένα ώστε να τρέχουν σε πολύ συγκεκριμένες αρχιτεκτονικές και υλικό μιας Cloud υποδομής. Χρειαζόμαστε πλατφόρμες ανοιχτού κώδικα, που να μπορούν να τρέξουν σε οποιαδήποτε IaaS, PaaS υποδομή, από το Cloud μέχρι ένα μέσο προσωπικό υπολογιστή παρέχοντας προσαρμοσμένη διαχείριση των αντικειμένων που χειρίζεται ένας ανεξαρτήτου μεγέθους οργανισμός.

3. Περιβάλλον ιστού για την προσαρμοστική διαχείριση πληροφορίας

3.1. Βασικός στόχος της εφαρμογής

Ο βασικός στόχος της εφαρμογής είναι η παροχή μιας ολοκληρωμένης λύσης που θα λειτουργεί σαν ένα ενδιάμεσο επίπεδο μεταξύ της αποθήκευσης, οργάνωσης και επεξεργασίας των αντικειμένων ενός οργανισμού και της εφαρμογής και της γραφικής διεπαφής ή κάποιου άλλου εξωτερικού συστήματος. Στις σύγχρονες εφαρμογές, ανεξαρτήτως μεγέθους, μπορούμεείναι εφικτή η διάκριση μιας σειράς από δομικά στοιχεία (σχήμα 3.1). Ως προς τη διαχείριση των αντικειμένων υπάρχουν δύο επίπεδα, εκείνο της αποθήκευσης και αυτό της επικύρωσης της δομής της πληροφορίας. Αμέσως μετά υπάρχει το επίπεδο της εσωτερικής διαχείρισης της εφαρμογής το οποίο αφορά τον διακομιστή, το λογισμικό που τρέχει σε αυτόν, καθώς και το κομμάτι της διαχείρισης των αιτημάτων. Αυτή η αρχιτεκτονική καταλήγει στην παροχή ενός συνόλου υπηρεσιών και διατίθεται μέσω του διαδικτύου για τη γραφική αναπαράσταση της πληροφορίας στους τελικούς χρήστες του διαδικτύου.



Δομικά στοιχεία μιας σύγχρονης εφαρμογής – Σχήμα 3.1

Από το σχήμα 3.1, προκύπτει ξεκάθαρα ότι ανεξαρτήτως από το είδος της εφαρμογής, κάθε φορά που δημιουργείται μια νέα, ένα μεγάλο σύνολο βημάτων επαναλαμβάνεται μεταξύ του επιπέδου αποθήκευσης της πληροφορίας και του εξωτερικού αιτήματος. Αυτό έχει ως αποτέλεσμα, έναν επαναλαμβανόμενο κώδικα ο οποίος είναι και παραδόξως δύσκολο να μεταφέρεται αυτούσιος σε διαφορετικές εφαρμογές. Κύριος στόχος της εφαρμογής που κλήθηκε η παρούσα εργασία να κατασκευάσει είναι να δημιουργήσει ένα νέο επίπεδο υπερκάλυψης των επαναλαμβανόμενων βημάτων και πιο συγκεκριμένα των:

- Server, ο οποίος αναλύει και χειρίζεται αιτήματα χρηστών του διαδικτύου ή άλλων εφαρμογών.
- Routing, δηλαδή του λογισμικού εκείνου που είναι υπεύθυνο να δρομολογεί τα αιτήματα εντός της εφαρμογής.
- Validation, προσφέροντας έλεγχο στη δομή των αντικειμένων που έρχονται ως είσοδο στο σύστημα.
- Persistence, που αφορά τη δυνατότητα του συστήματος να διατηρεί αποθηκευμένα τα αντικείμενα.

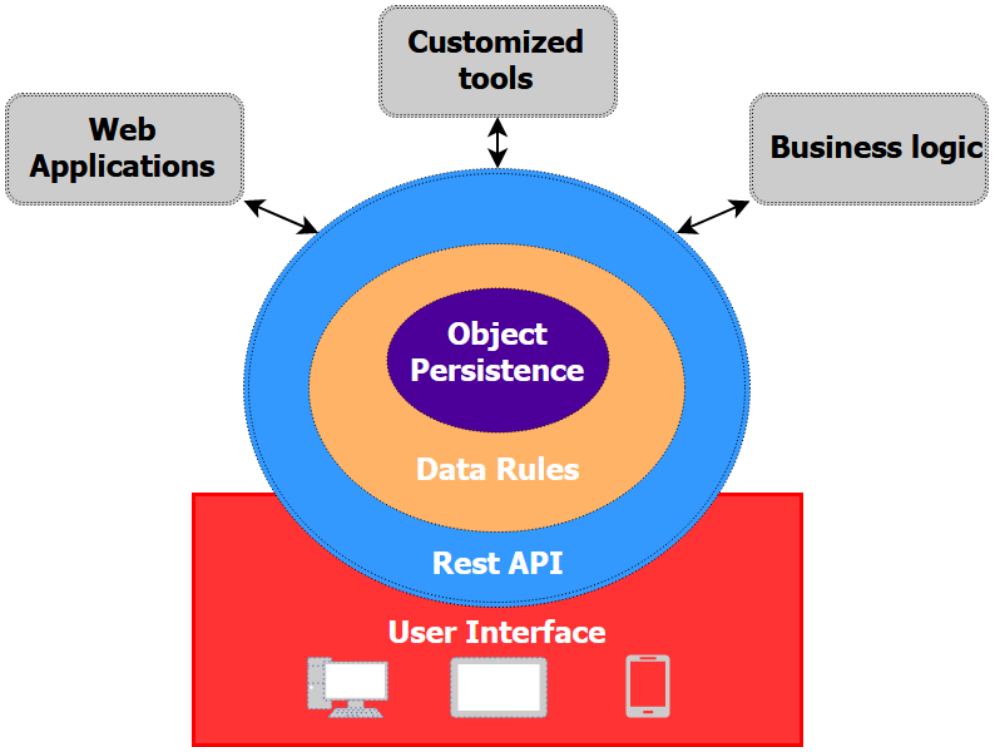
Το λογισμικό που υλοποιήθηκε παρέχει όλα τα επαναλαμβανόμενα αυτά βήματα παρέχοντας μια γραφική διεπαφή για τη μοντελοποίηση των αντικειμένων που θα κληθεί να χειριστεί. Επιπλέον, προσφέρει ένα middleware παρέχοντας τις βασικές λειτουργίες που χρειάζονται τα αντικείμενα (ανάκτηση, αποθήκευση, ενημέρωση, διαγραφή) και ένα τύπου REST API για τη διαχείριση των αντικειμένων εκτός του συστήματος. Κύριο μέλημα κατά την υλοποίηση αυτού του λογισμικού ήταν να μην έχει περιορισμούς που αφορούν το μέγιστο όγκο των αιτημάτων προς επεξεργασία αφού κατασκευάστηκε στα πρότυπα αντίστοιχων SaaS υπηρεσιών, δίνοντας τη δυνατότητα κλιμάκωσης από ένα προσωπικό υπολογιστή για μικρές εφαρμογές μέχρι το επίπεδο του Cloud.

3.2. Βασικές αρχές σχεδίασης

Η βάση ενός συστήματος προσαρμοστικής διαχείρισης δεδομένων είναι τα ίδια τα δεδομένα αλλά και οι βασικές υπηρεσίες που θα παρέχουν τη δυνατότητα αλληλεπίδρασης με αυτά. Όπως αποτυπώνεται και στο σχήμα 3.2, για την αποθήκευση και ανάκτηση των δεδομένων χρησιμοποιήθηκε μια μη σχεσιακή βάση δεδομένων η οποία απέκτησε τέτοια μορφή ώστε είναι σε θέση το σύστημα να παρέχει υψηλές επιδόσεις, δυνατότητες ανάλυσης των δεδομένων και απομόνωσης για το χειρισμό δεδομένων – αντικειμένων προερχόμενων από διαφορετικές εφαρμογές. Στη βάση δεδομένων, πρόσβαση έχουν μονάχα οι υπηρεσίες. Ένα σύνολο υπηρεσιών, απολύτως παραμετροποιήσιμων, είναι σε θέση να χειρίζεται τα δεδομένα του συστήματος μέσω εσωτερικής συνεργασίας υπηρεσιών αλλά και με παροχή λειτουργικότητας σε εξωτερικές εφαρμογές. Ενδεικτικά, με την εφαρμογή μπορούν να συνεργαστούν:

- Customized tools, όπου με συνδυασμό των υπηρεσιών που έχουν υλοποιηθεί στον πυρήνα της εφαρμογής μπορούν να προσθέσουν περαιτέρω λειτουργικότητα στο σύστημα.
- Web Applications, καθώς οι υπηρεσίες είναι προσβάσιμες από το διαδίκτυο, άλλες διαδικτυακές εφαρμογές μπορούν να συνεργαστούν με τις υπηρεσίες.
- BPM tools, σε περιπτώσεις υποδομών που υπάρχουν αυστηροί επιχειρησιακοί κανόνες, και πρέπει να εφαρμόζονται στα αντικείμενα που χειρίζεται το σύστημα.

Αξιοποιώντας όλες τις αρχές που πρέπει να έχει μια εφαρμογή που ουσιαστικά παίζει το ρόλο του ενδιάμεσου στα αντικείμενα μια εφαρμογής και στην αποθήκευσή τους και αποτυπώνοντάς τες στις υπηρεσίες, κατασκευάστηκε μια διεπαφή. Η διεπαφή αυτή είναι ικανή να χειριστεί ένα μεγάλο μέρος των διατεθούμενων υπηρεσιών. Επιπροσθέτως, επιτρέπει στους χρήστες να αλληλοεπιδρούν με τις υπηρεσίες της εφαρμογής με χρήση φυλλομετρητών σε υπολογιστές, ταμπλέτες και κινητά τηλέφωνα.



Περιβάλλον ιστού εφαρμογής – Σχήμα 3.2

3.3. Επιλογή τεχνολογιών

Έχοντας προχωρήσει σε μια αναλυτική εξέταση των χαρακτηριστικών που διαθέτει το Cloud επιλέχτηκαν τεχνολογίες σχεδιασμένες για υποστήριξη μεγάλων υπολογιστικών συστημάτων και επεκτασιμότητα. Η εξέταση τον τεχνολογιών διενεργήθηκε σε τρία επίπεδα:

- Τεχνολογίες στο επίπεδο του εξυπηρετητή που να προσφέρουν επιδώσεις, ασφάλεια και αξιοπιστία ενσωματώνοντας όλα τα χαρακτηριστικά εκείνα που περιεγράφηκαν αναλυτικά στο Κεφάλαιο 2.
- Βάση δεδομένων κατάλληλη για αποθήκευση αντικειμένων, που δεν είναι γνωστά τα χαρακτηριστικά τους, αλλά διαμορφώνονται σε πραγματικό χρόνο. Η έννοια της επεκτασιμότητας στο επίπεδο των πόρων, κατεύθυνε την μελέτη της επιλογής βάσης στη κατηγορία των NoSQL βάσεων δεδομένων.

- Σύγχρονες τεχνολογίες για την ανάπτυξη της γραφικής διεπαφής, στα πρότυπα μεγάλων πλατφόρμων. Στην επιλογή τεχνολογιών γραφικής διεπαφής, συνυπολογίστηκε η ανάγκη για μελλοντικές επεκτάσεις που κρίνονται σκόπιμες, καθώς και ο έλεγχος των υπηρεσιών που προσφέρει η πλατφόρμα αφού η διεπαφή θα μπορούσε να ενσωματώνει το σύνολο των REST υπηρεσιών που παράγονται.

3.4. Τεχνολογία εξυπηρετητή ιστού

Στα πλαίσια της μελέτης που έγινε στο Κεφάλαιο 2.6 (Αρχιτεκτονικές διακομιστών στις εφαρμογές), αποφασίστηκε το λογισμικό που θα χρησιμοποιηθεί, να υποστηρίζει το πρότυπο της αρχιτεκτονικής χειρισμού γεγονότων (Event-Driven Architecture). Ένα λογισμικό, δηλαδή, που στον πυρήνα να χειρίζεται ως γεγονότα τα αιτήματα που έρχονται από το διαδίκτυο προσφέροντας δυνατότητες επέκτασης της εφαρμογής από το επίπεδο του ενός υπολογιστή σε αυτό του Cloud. Οι λύσεις των παραδοσιακών εξυπηρετητών που δεσμεύουν νήματα του επεξεργαστή για το κάθε αίτημα μπορεί να δουλεύουν αξιοπρεπώς στο επίπεδο του ενός υπολογιστή, όταν όμως μιλάμε για το Cloud, αναφερόμαστε σε συστάδες υπολογιστών που αρχιτεκτονικές Virtualization (Κεφάλαιο 2.5)και δημιουργούν κακή αξιοποίηση των πόρων.

3.4.1. Vert.x

Το λογισμικό που διακρίθηκε στην έρευνα για την κατάλληλη τεχνολογία στο επίπεδο του εξυπηρετητή ιστού ονομάζεται Vertx.io [17]. Σύμφωνα με τον επίσημο ιστότοπο της πλατφόρμας το Vert.x είναι ένα πακέτο λογισμικού που βοηθά στην υλοποίηση συστημάτων που χρειάζεται να χειριστούν πολλαπλές ταυτόχρονες συνδέσεις με μεγάλη έμφαση στην μείωση του χρόνου εξυπηρέτησης των αιτημάτων και τη διαθεσιμότητα.

Στη φιλοσοφία του σαν λογισμικό, έχει επιρροές από το Node.js. Όπως το Node.js, έτσι και το Vert.x είναι ένα λογισμικό εξυπηρετητή που παρέχει ένα περιβάλλον Event-Driven προγραμματισμού.

Εντούτοις, η προγραμματιστική διεπαφή του Vert.x μοιάζει αρκετά με εκείνη του Node.js εξαιτίας της παροχής και από τις δύο τεχνολογίες μιας ασύγχρονης διεπαφής προγραμματισμού.

Τα πιο σημαντικά χαρακτηριστικά που βρίσκονται στο Vert.x είναι:

- Η υποστήριξη πολλών γλωσσών προγραμματισμού (Polyglot)
  Το Vert.x είναι κατασκευασμένο σε Java, ωστόσο δεν είναι υποχρεωτικό οι εφαρμογές που κατασκευάζονται σε αυτό να είναι επίσης γραμμένες σε Java. Εκτός από την υποστήριξη γλωσσών προγραμματισμού που τρέχουν στο JVM, το Vert.x μπορεί να υποστηρίξει προγραμματισμό σε Ruby, Python και JavaScript.

- Μοντέλο ταυτόχρονης υποστήριξης Event-Driven και Process-Driven Αρχιτεκτονικών
  Υλοποιώντας ένα διακομιστή ιστού με τη χρήση του Vert.x οι χρήστες μπορούν να γράφουν κώδικα όπως και στις Process-Driven αρχιτεκτονικές. Αυτό σημαίνει ότι τα προβλήματα συγχρονισμού κλειδώματος και μεταβλητότητας της απόδοσης που παρουσιάζονται στις multi-thread αρχιτεκτονικές εξαλείφονται. Το Vert.x επιτρέπει τη δημιουργία πολλαπλών νημάτων σύμφωνα με τον αριθμό των πυρήνων στον επεξεργαστή του εξυπηρετητή εκτελώντας ασύγχρονες διαδικασίες.

- Διαθέτει ένα δίαυλο μεταφοράς γεγονότων (Event Bus)
  Ο δίαυλος μεταφοράς γεγονότων εξυπηρετεί στη μεταφορά πληροφορίας μεταξύ των διάφορων τμημάτων λειτουργικότητας. Δίνεται η δυνατότητα δημιουργίας τέτοιων κομματιών που την ορολογία του Vert.x ονομάζονται Verticles. Τα Verticles μπορούν να είναι γραμμένα σε διαφορετικές γλώσσες προγραμματισμού και να επικοινωνούν μεταξύ τους χάρη στο Event Bus.

Το Vert.x έχει δημιουργήσει τη δικιά του τεχνολογία για την περιγραφή των διαφορετικών λειτουργιών του. Ο πιο σημαντικός όρος είναι αυτός του Verticle. Στη Java, Verticles θα θεωρούσαμε τις κλάσεις που περιέχουν μια main() μέθοδο. Ένα Verticle μπορεί να περιλαμβάνει κομμάτια κώδικα και βιβλιοθήκες. Μια σύγχρονη εφαρμογή συνήθως αποτελείται από περισσότερα Verticles, σπάζοντας την υλοποίηση σε μικρότερα κομμάτια βάση λειτουργικότητας. Τα Verticles επικοινωνούν μεταξύ τους μέσω του Event Bus.

Μια άλλη έννοια είναι αυτή του Vert.x Instance, όπου όπως ένα Verticle εκτελείται σε ένα Vert.x instance , έτσι και τα Vert.x Instances τρέχουν στο JVM. Σε αυτή τη λογική, πολλά Verticle εκτελούνται ταυτόχρονα σε ένα Vert.x Instance. Το Vert.x εγγυάται για το κάθε Vertice ενός Instance εκτέλεση σε ένα νήμα.

3.4.2. Hazelcast

Το Hazelcast αποτελεί ένα λογισμικό που στόχο έχει την διαχείριση δεδομένων και χρησιμοποιείται για την παράλληλη εκτέλεση λογισμικού σε πολλούς υπολογιστές [19]. Πρόκειται για μια κατανεμημένη πλατφόρμα που δίνει προγραμματιστικές διεπαφές χρήσης.

Το ενδιαφέρον σε αυτή τη τεχνολογία έγκειται στο γεγονός ότι το Vert.x μπορεί να χρησιμοποιήσει το Hazelcast επεκτείνοντας τον δίαυλο γεγονότων (Event bus) του. Με αυτό το τρόπο επεξεργαστικοί πυρήνες που βρίσκονται σε διαφορετικούς υπολογιστές είναι διαθέσιμοι για αξιοποίηση από το Vert.x παρέχοντας στην πλατφόρμα περισσότερους πόρους για την εξυπηρέτηση αυξημένου αριθμό αιτημάτων.

3.4.3. RethinkDB

Ως προς τη βάση δεδομένων που μπορεί να καλύψει τις ανάγκες της εφαρμογής επιλέχτηκε η RethinkDB [18]. Πρόκειται για μία μη σχεσιακή βάση δεδομένων (NoSQL), η οποία χειρίζεται JSON Documents. Ως προς τη κλιμάκωση που μπορούμε να συναντήσουμε σε αυτή, παρατηρούμε ότι παρέχει ένα μεγάλο εύρος σχετικών επιλογών με διαφορετικά σχήματα που περιλαμβάνουν πολλαπλούς Database Servers και Proxy nodes με δυνατότητες απεριόριστης επεκτασιμότητας σε Cloud υποδομές.

Επιπλέον περιλαμβάνει μηχανισμό αποστολής ειδοποιήσεων σε πραγματικό χρόνο. Ένας χρήστης της βάσης μπορεί να γίνεται συνδρομητής (subscribe) σε κάποιο ερώτημα προς βάση, η οποία με τη σειρά της τον ειδοποιεί σε πραγματικό χρόνο για οποιαδήποτε αλλαγή σημειωθεί στο

αποτέλεσμα του ερωτήματος. Βασικό πλεονέκτημα της RethinkDB έναντι άλλων συστημάτων που παρέχουν υπηρεσίες συγχρονισμού σε πραγματικό χρόνο είναι πως παρέχει μεγάλη ευελιξία στο είδος των ερωτημάτων προς συνδρομή. Πιο συγκεκριμένα, σχεδόν όλα τα ερωτήματα στη βάση μπορούν να μετατραπούν σε ροές ειδοποιήσεων (change feeds), επιτρέποντας κατ' αυτόν τον τρόπο σε πολύπλοκα ερωτήματα (πχ join ή map/reduce) να μετατραπούν πολύ εύκολα σε μια τέτοια ροή.

Η γλώσσα ερωτημάτων που υποστηρίζει η RethinkDB λέγεται ReQL. Πρόκειται για μια γλώσσα NoSQL, σχεδιασμένη να εκφράζει εύκολα ερωτήματα σε JSON έγγραφα. Ένα τέτοιο ερώτημα κατασκευάζεται μέσα από διαδοχικές κλήσεις συναρτήσεων που παρέχονται από τον οδηγό της γλώσσας που χρησιμοποιείται για την επικοινωνία με τη βάση. Η προσέγγιση αυτή καθιστά πιο προσιτή την εκμάθηση της ReQL καθώς δε διαφέρει από την εκμάθηση μιας προγραμματιστικής βιβλιοθήκης.

Η ReQL παρέχει όλη τη λειτουργικότητα της SQL (πχ joins, group_by, aggregation). Επιπλέον, υποστηρίζει τη δυνατότητα map/reduce τελεστών, τους οποίους μπορεί να ορίσει ο χρήστης- για την ακρίβεια, τελεστές όπως ο count δεν είναι τίποτα άλλο παρά map/reduce συναρτήσεις ορισμένες από την ίδια τη βάση. Παρέχονται οι κλασσικές δομές επανάληψης (forEach) και επιλογής (branch), επιστροφή προεπιλεγμένων τιμών (default), έλεγχος του δυναμικού τύπου του πεδίου ενός εγγράφου (typeOf) κλπ. Τέλος, η γλώσσα υποστηρίζει δομές ελέγχου της ροής εκτέλεσης ενός ερωτήματος.

3.4.4. Angular 4

Η επιλογή της πλατφόρμας εκείνης, που στηριζόταν η εμφάνιση της εφαρμογής, έπαιξε σπουδαίο ρόλο. Όταν αναφερόμαστε στην εμφάνιση, δεν μιλάμε για τα animation και τους κανόνες css αλλά για το πλαίσιο εκείνο που θα επιτρέψει τη δημιουργία μιας σύγχρονης εφαρμογής ιστού με δυνατότητες επέκτασης. Η τελευταία έκδοση της Angular κρίθηκε ως η ιδανικότερη, ακολουθώντας ένα component-based σχεδιασμό και στο επίπεδο της υλοποίησης τη γλώσσα προγραμματισμού

Typescript [16]. Η Typescript κατά βάση παρέχει ένα υπερσύνολο των εντολών της Javascript, οργανώνοντας τον κώδικα διευκολύνοντας μελλοντικές επεκτάσεις της εφαρμογής.

Η Angular αποτελεί μια λύση για γρήγορη ανάπτυξη client-side εφαρμογών και χρησιμοποιείται από τη Google, καθώς και ένα πλήθος προγραμματιστών που κατασκευάζουν εφαρμογές που προορίζονται εξολοκλήρου για εκτέλεση στα προγράμματα περιήγησης κάνοντας χρήση HTML, CSS και JavaScript στην πλευρά του πελάτη. Ο στόχος της Angular 4 είναι να αυξήσει τις εφαρμογές ιστού με δυνατότητα να υφίσταται έλεγχος στη χρήση προτύπων μοντέλου-ελεγκτή (MVC) για να διευκολύνει την ανάπτυξη και τον έλεγχο.

Είναι εύκολο να κατανοήσει ένας προγραμματιστής ότι η Angular προορίζεται για την ανάπτυξη ισχυρών εφαρμογών σε έργα οποιασδήποτε κλίμακας. Άλλωστε, μέρος της δημοτικότητάς που έχει γνωρίσει οφείλεται στην ικανότητά της να κάνει πιο δυναμικές τις στατικές ιστοσελίδες, επιτρέποντας έτσι στους σχεδιαστές ιστοσελίδων να προσθέσουν περισσότερα εργαλεία.

3.5. Περιγραφή παρεχόμενης λειτουργικότητας του συστήματος

Το σύστημα κατά βάση οφείλει με εύχρηστες και προσαρμοστικές λειτουργίες να αποθηκεύει και να επεξεργάζεται αντικείμενα δεδομένων άλλων εφαρμογών. Σε αυτό το πλαίσιο, κατασκευάστηκε ένα πλήθος υπηρεσιών κατάλληλο για χρήση είτε από άλλα συστήματα είτε απευθείας από προγράμματα περιήγησης στο διαδίκτυο. Οι υπηρεσίες αυτές μπορούν να διακριθούν σε τρία επίπεδα – σχήμα 3.3:

Παρεχόμενη λειτουργικότητα εφαρμογής – Σχήμα 3.3

- **Διαχείριση χρηστών του συστήματος** – Ολοκληρωμένη διαχείριση των χρηστών του συστήματος μέσω υπηρεσιών τύπου REST για υποστηρίζοντας τη δημιουργία, την επεξεργασία και την ανάκτηση χρηστών του συστήματος
- **Διαχείρισης βάσης δεδομένων του χρήστη** – Δυνατότητα δημιουργίας Βάσεων Δεδομένων, μόνο από τους χρήστες του συστήματος, στη λογική της δημιουργίας νέου «Έργου». Το κάθε «Έργο» καλείται να διαχειριστεί είδη αντικειμένων. Οπότε παρέχεται σε πρώτη φάση η δημιουργία της βάσης - έργου, ενώ στη πορεία η πλήρη διαχείριση του σχήματός της, στη λογική που χειριζόμαστε σχεσιακά σχήματα Βάσεων Δεδομένων.
- **Επίπεδο διαχείρισης των αντικειμένων της βάσης** – Ένα ολοκληρωμένο σύνολο υπηρεσιών ειδικά διαμορφωμένο για τα αντικείμενα που ο χρήστης έχει περιγράψει ότι έχει η βάση του στο προηγούμενο επίπεδο. Παρέχονται δυνατότητες ανάγνωσης (ενός ή συνόλου αντικειμένων), εισαγωγής, επεξεργασίας και διαγραφής αντικειμένων.

4. Μοντέλο δεδομένων

Για την αποθήκευση των δεδομένων, στο σύστημα χρησιμοποιήθηκε η RethinkDB, μια μη σχεσιακή βάση δεδομένων, βασισμένη στη λογική της αποθήκευσης εγγράφων. Η χρήση μιας τέτοιου είδους βάσης δεδομένων κρίνεται αναγκαία, αρκεί να αναλογιστεί κανείς ότι πρέπει να αποθηκεύονται αντικείμενα άγνωστου τύπου και μορφής, σε ένα περιβάλλον που έχει τη δυνατότητα κλιμάκωσης. Εντούτοις, υπάρχει ένα βασικό σχήμα που είναι γνωστό εξαρχής, και αυτό είναι η διαχείριση της ίδιας της πλατφόρμας. Η βάση που παρέχει στοιχεία διαχείρισης του ίδιου του συστήματος ονομάζεται ενδεικτικά «system». Οι οντότητες αυτού το σχήματος είναι οι εξής:

- Χρήστες της πλατφόρμας
- Βάσεις δεδομένων - Έργα
- Πίνακες – Αντικείμενα

4.1. Χρήστες της πλατφόρμας

Η πληροφορία που χρειάζεται να διατηρηθεί για τους χρήστες είναι τα στοιχεία τους και στοιχεία πρόσβασης στο σύστημα. Για την αποθήκευσή τους έχει δημιουργηθεί ένας πίνακας με όνομα «users».

Πίνακας users

Key	**Value Type**
id	String (UUID)
username	String
password	String
email	String

Περιγραφή ορισμάτων αντικειμένου

id: Μοναδικό και πρωτεύον κλειδί για τον κάθε χρήστη

username: Όνομα χρήστη, μοναδικό πεδίο που χρησιμοποιείται για τη σύνδεση του χρήστη και γενικότερα σαν ένα φιλικό στο μάτι κλειδί

password: Κωδικός χρήστη

email: Ηλεκτρονικό ταχυδρομείο, πληροφοριακό πεδίο αν και υποχρεωτικό, ίσως χρησιμοποιηθεί σε μελλοντικές επεκτάσεις της πλατφόρμας

Αποτύπωση σε JSON μορφή της βάσης δεδομένων

```
{
  "id" : "f19d2a32-4b70-43cb-9577-03400fa7952d",
  "username" : "testUser",
  "password" : "***passwordHash***",
  "email" : "test@test.com"
}
```

4.2. Βάσεις δεδομένων της πλατφόρμας

Η πληροφορία που χρειάζεται να διατηρηθεί για τις βάσεις δεδομένων της πλατφόρμας είναι το όνομα της και το αναγνωριστικό κλειδί τού χρήστη – διαχειριστή της. Σχεδιαστικά και για λόγους επιδόσεων του συστήματος, εντός του πίνακα – αντικείμενο “databases” υπάρχει και ένα πεδίο που περιέχει τα είδη των αντικειμένων που φιλοξενεί η βάση.

Πίνακας databases

Key	Value Type
id	String (UUID)
userId	String
name	String
tableList	List<Table>

Περιγραφή ορισμάτων αντικειμένου

id: Μοναδικό και πρωτεύον κλειδί για τη κάθε βάση δεδομένων

userId: Αναγνωριστικό κλειδί για το χρήστη του συστήματος, του οποίου ανήκει η διαχείριση της βάσης

name: Το όνομα της βάσης δεδομένων

tableList: Ένα πολυδιάστατο πεδίο που η χρήση της RethinkDB μας επιτρέπει να εισάγουμε στο σχήμα μας. Πρόκειται για μία λίστα αντικειμένων τύπου Table.

Αποτύπωση σε JSON μορφή της βάσης δεδομένων


```
{
  "id": "c2d126e9-0777-412b-99ac-be4e413f142b",
  "userId": "f19d2a32-4b70-43cb-9577-03400fa7952d",
  "name": "myDB",
  "tableList": [
    {…},
    {…},
    {…}
  ]
}
```

Εμφωλευμένος πίνακας table

Key	Value Type
id	String (UUID)
name	String
jsonSchema	JSON Object

Περιγραφή ορισμάτων αντικειμένου

id: Μοναδικό και πρωτεύον κλειδί για το κάθε είδος αντικειμένου

name: Όνομα του είδους - τύπου του αντικειμένου

jsonSchema: Το πεδίο αυτό είναι τύπου JSON. Δηλαδή ο τύπος του είναι ένα άλλο αντικείμενο. Δεν υπάρχει κάποιος περιορισμός σε επίπεδο Βάσης δεδομένων για το τι μπορεί να περιέχει αυτό το αντικείμενο, ωστόσο στο σύστημα γίνονται έλεγχοι με στόχο να είναι τύπου “JSON Schema”. Το “JSON Schema” που αποτελεί τη πιο διαδεδομένη προσπάθεια προτυποποίησης για την περιγραφή ενός JSON Document.

Αποτύπωση σε JSON μορφή της βάσης δεδομένων


```
{
  "jsonSchema": {
    "schema": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        },
        "age": {
          "type": "number"
        }
      }
    }
  },
  "name": "userObject",
  "id": "d64786d4-a845-45b5-b30f-f58b67e0f39b"
}
```


4.3. Αντικείμενα χρηστών

Απώτερος στόχος της πλατφόρμας είναι η αποθήκευση των αντικειμένων των χρηστών στην υποδομή. Για να εξασφαλιστεί ότι δεν θα επηρεάζει ο ένας χρήστης με κανένα τρόπο τα δεδομένα κάποιου άλλου χρήστη του συστήματος, η κάθε βάση δεδομένων που περιεγράφηκε παραπάνω ως μια καταχώριση σε έναν πίνακα, είναι μια πραγματική βάση δεδομένων για τη RethinkDB. Αντίστοιχα και οι πίνακες, αποτελούν φυσικούς πίνακες – έγγραφα για την βάση δεδομένων. Αυτή η πρακτική που ακολουθήθηκε προσφέρει ένα απαραίτητο επίπεδο φυσικής απομόνωσης στο επίπεδο του κάθε πίνακα, στο επίπεδο της κάθε βάσης και στο επίπεδο του κάθε χρήστη γενικότερα. Τέλος, αυτή η διαχείριση έχει τρομερά πλεονεκτήματα επιδόσεων (δεν χρειάζεται όταν αναζητηθεί ένα αντικείμενο να ψάχνεται όλη η πλατφόρμα) αλλά και ελέγχου, καθώς προσφέρει μια ασφαλή μέθοδο κατάργησης των αντικειμένων ενός χρήστη χωρίς να υπάρχει φόβος για την υπόλοιπη πλατφόρμα.

5. Υποστηριζόμενες υπηρεσίες

Το μεγαλύτερο μέρος της λειτουργικότητας της εφαρμογής που υλοποιήθηκε αποτυπώνεται με τη μορφή υπηρεσιών. Οποιαδήποτε συναλλαγή με τη βάση δεδομένων της εφαρμογής γίνεται μέσω των διαθέσιμων υπηρεσιών. Η γραφική διεπαφή που υλοποιήθηκε ή γενικότερα οποιαδήποτε άλλη εξωτερική εφαρμογή συνεργάζεται μόνο με τις διαθέσιμες υπηρεσίες, ώστε να εξασφαλιστεί η ακεραιότητα των δεδομένων και η οργάνωση της εφαρμογής στο σύνολό της. Με βάση την περιγραφή παρεχόμενης λειτουργικότητας (Κεφάλαιο 3.5) ακολουθεί αναλυτική περιγραφή της λειτουργικότητας της κάθε υπηρεσίας.

Κατηγορία	REST Endpoint
Διαχείρισης χρηστών	/system/user /system/connected_user
Εισόδου στο σύστημα	/system/login
Διαχείρισης βάσης δεδομένων	/system/database /system/connected_user_database
Διαχείρισης πινάκων / αντικειμένων	/system/table
Διαχείρισης αντικειμένων	/api/{databaseName}/{tableName}

5.1. Διαχείριση Χρηστών

Για τη διαχείριση των χρηστών υπάρχουν τρείς μέθοδοι που αφορούν τη δημιουργία (μέθοδος POST), διαγραφή(μέθοδος DELETE) και την ενημέρωση των στοιχείων (μέθοδος PUT) 5.2.ενός χρήστη.

Για την εγγραφή ενός χρήστη χρησιμοποιείται η μέθοδος POST. Σε περίπτωση που ο χρήστης υπάρχει (κοινό username με κάποιον άλλο χρήστη), επιστρέφεται κατάλληλο μήνυμα λάθους. Εφόσον δημιουργηθεί ο χρήστης, επιστρέφεται στην απάντηση, ενώ ταυτόχρονα εκτός από την καταχώρισή του στον πίνακα “users”, γίνεται και η δημιουργία νέας βάσης με όνομα ίδιο με το username του χρήστη. Επακόλουθα, είναι επιτακτική η ανάγκη το username να είναι υποχρεωτικό. Ο χρήστης αποκτά και ένα μοναδικό id το οποίο είναι τύπου UUID μήκους 36 χαρακτήρων. Τέλος, στην απάντηση οποιασδήποτε κλήσης, δεν επιστρέφει ποτέ ο κωδικός πρόσβασης του χρήστη.

Παράδειγμα κλήσης

POST	/system/user

Αίτημα

```
Content-Type: application/json
{
  "username": "testUser",
  "password": "testPassword",
  "email": "test@test.com"
}
```

Απάντηση

```
Access-Control-Allow-Credentials:true
Content-Type: application/json
Set-Cookie:vertx-web.session=95ebfb55-c595-47f0-95c8-3c774e8f148e; Path=/
{
  "id" : "f19d2a32-4b70-43cb-9577-03400fa7952d",
  "username" : "testUser",
  "email" : "test@test.com"
}
```

Περιγραφή μεταβλητών

createUser (User user, <AsyncResult<JsonObject>> resultHandler)	**Input:** αντικείμενο τύπου User, resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη).
	Output: Είτε τα στοιχεία του χρήστη σε JsonObject μορφή είτε κωδικός λάθους.

Για την ενημέρωση των στοιχείων ενός χρήστη χρησιμοποιείται η μέθοδος PUT. Δεν επιτρέπεται η αλλαγή του ονόματος χρήστη και του id, ενώ όλα τα υπόλοιπα πεδία μπορούν να αλλαχθούν κανονικά.

Παράδειγμα κλήσης

POST	/system/user/{userId}

Αίτημα

```
Content-Type: application/json
{
  "username": "testUser",
  "password": "testPassword",
  "email": "test@test.com"
}
```

Απάντηση

```
Access-Control-Allow-Credentials:true
Content-Type: application/json
Set-Cookie:vertx-web.session=95ebfb55-c595-47f0-95c8-3c774e8f148e; Path=/
{
  "id" : "f19d2a32-4b70-43cb-9577-03400fa7952d",
  "username" : "testUser",
  "email" : "test@test.com"
}
```

Περιγραφή μεταβλητών

createUser (User user, String userId, <AsyncResult<JsonObject>> resultHandler)	**Input:** αντικείμενο τύπου User, το id του χρήστη που αφορά το update, resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη).
	Output: Είτε τα στοιχεία του χρήστη σε JsonObject μορφή είτε κωδικός λάθους.

Για τη διαγραφή ενός χρήστη χρησιμοποιείται η μέθοδος DELETE. Κατά τη διαγραφή του χρήστη διαγράφονται, συγχρόνως, η βάση δεδομένων του και όλα τα αντικείμενα που έχει δημιουργήσει. Σε περίπτωση σφάλματος επιστρέφεται μήνυμα αποτυχίας.

Παράδειγμα κλήσης

DELETE	/system/user/{userId}

Αίτημα

```
Content-Type: application/json
```

Απάντηση

```
Status Code:200 OK
```

Περιγραφή μεταβλητών

deleteUser (String userId, <AsyncResult<JsonObject>> resultHandler)	**Input:** το id του χρήστη προς διαγραφή, resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη).
	Output: Κωδικός επιτυχίας ή αποτυχίας.

5.2. Είσοδος στο σύστημα

Η είσοδος στο σύστημα γίνεται χρήση της μεθόδου loginUser(). Η μέθοδος loginUser() χρησιμοποιείται ώστε να ελεγχτεί αν το username (όνομα χρήστη στο σύστημα) και το password (κωδικός χρήστη) αντιστοιχούν σε χρήστη στην βάση δεδομένων. Σε περίπτωση που υπάρχει ο συνδυασμός του ονόματος χρήστη με το κρυπτογράφημά του επιστρέφονται όλα τα στοιχεία του χρήστη εκτός από τον κωδικό πρόσβασης. Σε περίπτωση που ο συνδυασμός δεν υπάρχει επιστρέφεται σφάλμα. Κατά την επιτυχή σύνδεση δημιουργείται ένα session cookie ώστε να παρέχεται, ως ένα βαθμό, ασφάλεια για κλήσεις που αφορούν τη διαχείριση της βάσης και των πινάκων του χρήστη.

Παράδειγμα κλήσης

POST	/system/login

Αίτημα

```
Content-Type: application/json
{
  "username": "testUser",
  "password": "testPassword"
}
```

Απάντηση

```
Access-Control-Allow-Credentials:true
Content-Type: application/json
Set-Cookie:vertx-web.session=95ebfb55-c595-47f0-95c8-3c774e8f148e; Path=/
{
  "id" : "f19d2a32-4b70-43cb-9577-03400fa7952d",
  "username" : "testUser",
  "email" : "test@test.com"
}
```

Περιγραφή μεταβλητών

loginUser (String username, String password, <AsyncResult<JsonObject>> resultHandler)	**Input:** username (όνομα χρήστη), password (κωδικός χρήστη), resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη).
	Output: Είτε τα στοιχεία του χρήστη σε JsonObject μορφή είτε κωδικός λάθους.

Για τον έλεγχο του συνδεδεμένου χρήστη με βάση το session-cookie που έλαβε κατά την σύνδεσή του έχει δημιουργηθεί η κλήση /system/connected_user. Σε περίπτωση λάθους (δηλαδή δεν είναι συνδεδεμένος κάποιος χρήστης επιστρέφεται μήνυμα λάθους)

Παράδειγμα κλήσης

GET	/system/connected_user

Αίτημα

```
Content-Type: application/json
```

Απάντηση

```
Access-Control-Allow-Credentials:true
Content-Type: application/json
Set-Cookie:vertx-web.session=95ebfb55-c595-47f0-95c8-3c774e8f148e; Path=/
{
  "id" : "f19d2a32-4b70-43cb-9577-03400fa7952d",
  "username" : "testUser",
  "email" : "test@test.com"
}
```

Περιγραφή μεταβλητών

getConnectedUser (<AsyncResult<JsonObject>> resultHandler)	**Input:** resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη).
	Output: Είτε τα στοιχεία του χρήστη σε JsonObject μορφή είτε κωδικός λάθους.

5.3. Διαχείριση βάσης δεδομένων

Για τη διαχείριση της βάσης δεδομένων έχει δημιουργηθεί ένα σύνολο από υπηρεσίες τις οποίες χρησιμοποιεί η γραφική διεπαφή αλλά και εξωτερικές εφαρμογές. Όπως και στη διαχείριση χρηστών, υπάρχει session-cookie ταυτοποίηση για το σύνολο των υπηρεσιών που προσφέρονται για τη διαχείριση της βάσης δεδομένων. Κύριος σκοπός αυτής της ομάδας υπηρεσιών είναι η δημιουργία, ανάκτηση και διαγραφή μιας βάσης δεδομένων ενός χρήστη.

Για τη δημιουργία της βάσης δεδομένων υπάρχει μια μέθοδος τύπου POST, η οποία ελέγχει εάν υπάρχει βάση με το ίδιο όνομα στο σύστημα και επιστρέφεται σφάλμα. Επιπλέον, επειδή το αντικείμενο τύπου Database εμπεριέχει μια λίστα πινάκων (πεδίο tableList) ελέγχεται περαιτέρω, εάν η δομή των αντικειμένων που ζητείται η δημιουργία τους είναι σωστή. Εφόσον όλα είναι σωστά και υπάρχουν πίνακες στο αίτημα για δημιουργία της βάσης, καλείτεαι εσωτερικά και για όσους πίνακες ζητείται η δημιουργία τους, η μέθοδος createTable() που περιγράφεται στη συνέχεια.

Παράδειγμα κλήσης

POST	/system/database

Αίτημα

```
Content-Type: application/json
{
  "name":  "myDatabase",
  "tableList": []
}
```

Απάντηση

```
Access-Control-Allow-Credentials:true
Content-Type: application/json
Set-Cookie:vertx-web.session=95ebfb55-c595-47f0-95c8-3c774e8f148e; Path=/
{
  "id":  "c2d126e9-0777-412b-99ac-be4e413f142b" ,
  "name":  "myDatabase" ,
  "tableList": [] ,
  "userId":  "f19d2a32-4b70-43cb-9577-03400fa7952d"
}
```

Περιγραφή μεταβλητών

createDatabase (Database database, <AsyncResult<JsonObject>> resultHandler)	**Input:** αντικείμενο τύπου Database, resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη).
	Output: Είτε τα στοιχεία βάσης που δημιουργήθηκε σε JsonObject μορφή είτε κωδικός λάθους.

Για την προσπέλαση μιας βάσης γνωρίζοντας το id της μπορεί να χρησιμοποιηθεί η μέθοδος getDatabase(). Εξαιτίας του σχήματος της βάσης, μαζί με τη βάση επιστρέφει και το πεδίο tableList, που περιέχει τους πίνακες της βάσης.

Παράδειγμα κλήσης

GET	/system/ database/{databaseId}

Αίτημα

```
Content-Type: application/json
```

Απάντηση

```
Access-Control-Allow-Credentials:true
Content-Type: application/json
Set-Cookie:vertx-web.session=95ebfb55-c595-47f0-95c8-3c774e8f148e; Path=/
{
  "id":  "c2d126e9-0777-412b-99ac-be4e413f142b" ,
  "name":  "myDatabase" ,
  "tableList": [] ,
  "userId":  "f19d2a32-4b70-43cb-9577-03400fa7952d"
}
```

Περιγραφή μεταβλητών

getDatabase (String databaseId <AsyncResult<JsonObject>> resultHandler)	**Input:** id της βάσης δεδομένων, resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη)
	Output: Τα στοιχεία της βάσης μαζί με τη λίστα πινάκων σε JsonObject μορφή είτε κωδικός λάθους.

Για τη διαγραφή μιας βάσης δεδομένων χρησιμοποιείται η μέθοδος deleteDatabase(). Κατά τη διαγραφή της βάσης διαγράφονται τόσο η βάση δεδομένων του όσο και όλα τα αντικείμενα που έχει δημιουργήσει. Σε περίπτωση σφάλματος επιστρέφεται μήνυμα αποτυχίας.

Παράδειγμα κλήσης

DELETE	/system/database/{databaseId}

Αίτημα

```
Content-Type: application/json
```

Απάντηση

```
Status Code:200 OK
```

Περιγραφή μεταβλητών

deleteDatabase (String databaseId, <AsyncResult<JsonObject>> resultHandler)	**Input:** το id της βάσης δεδομένων προς διαγραφή, resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη).
	Output: Κωδικός επιτυχίας ή αποτυχίας.

Για την ανάκτηση της βάσης δεδομένων του συνδεδεμένου χρήστη, έχει δημιουργηθεί η μέθοδος getConnectedUserDatabase() που χρησιμοποιώντας το session-cookie ανιχνεύει τον συνδεμένο χρήστη και επιστρέφει τη βάση, εφόσον αυτή υπάρχει

Παράδειγμα κλήσης

GET	/system/connected_user_database

Αίτημα

```
Content-Type: application/json
```

Απάντηση

```
Access-Control-Allow-Credentials:true
Content-Type: application/json
Set-Cookie:vertx-web.session=95ebfb55-c595-47f0-95c8-3c774e8f148e; Path=/
{
  "id":  "c2d126e9-0777-412b-99ac-be4e413f142b" ,
  "name":  "myDatabase" ,
  "tableList": [] ,
  "userId":  "f19d2a32-4b70-43cb-9577-03400fa7952d"
}
```

Περιγραφή μεταβλητών

getConnectedUserDatabase (<AsyncResult<JsonObject>> resultHandler)	**Input:** resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη)
	Output: Τα στοιχεία της βάσης μαζί με τη λίστα πινάκων σε JsonObject μορφή είτε κωδικός λάθους.

5.4. Διαχείριση πινάκων / αντικειμένων

Για τη διαχείριση των τύπων αντικειμένων ενός χρήστη έχει δημιουργηθεί ένα σύνολο υπηρεσιών που επιτρέπουν την δημιουργία, ανάκτηση, ενημέρωση και διαγραφή τους. Στόχος είναι να μπορεί ο χρήστης να ορίζει τη δομή που επιθυμεί να έχουν τα αντικείμενά του με σκοπό τον αυτόματο έλεγχο (data validation) και όχι για να δημιουργήσει το σύστημα πίνακες με τη σχεσιακή έννοια (δηλαδή ορισμό των πεδίων στη βάση). Η RethinkDB είναι μια document-based βάση δεδομένων χωρίς τη δυνατότητα ελέγχου πάνω στα δεδομένα.

Για τη δημιουργία ενός αντικειμένου έχει υλοποιηθεί η μέθοδος createTable() με την πρόσβαση σε αυτή να γίνεται μέσω της μεθόδου POST. Για την δημιουργία, πάλι, του πίνακα ελέγχεται η ορθότητα του πεδίου jsonShema και το αν αυτό είναι της μορφής JSON Schema. Επιπλέον, ελέγχεται

αν υπάρχει ήδη αντικείμενο στη βάση του χρήστη με το ίδιο ακριβώς όνομα. Στην περίπτωση αυτή, επιστρέφεται μήνυμα λάθους, ενώ εάν είναι επιτυχής η καταχώριση και η δημιουργία του πίνακα, επιστρέφεται μήνυμα επιτυχίας.

Παράδειγμα κλήσης

POST	/system/table

Αίτημα

```
Content-Type: application/json
{
  "name": "testObject",
  "jsonSchema": {
    "schema": {
      "type": "object",
      "properties": {}
    }
  }
}
```

Απάντηση

```
Access-Control-Allow-Credentials:true
Content-Type: application/json
Set-Cookie:vertx-web.session=95ebfb55-c595-47f0-95c8-3c774e8f148e; Path=/
{
  "jsonSchema": {
    "schema": {
      "type": "object",
      "properties": {}
    }
  },
  "name": "testObject",
  "id": "b0c55593-d583-46c7-9443-a806cfa71a24"
}
```

Περιγραφή μεταβλητών

createTable (Table table, <AsyncResult<JsonObject>> resultHandler)	**Input:** αντικείμενο τύπου table, resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη).
	Output: τα στοιχεία του πίνακα που δημιουργήθηκε σε JsonObject μορφή είτε κωδικός λάθους.

Για τη προσπέλαση ενός πίνακα, ώστε να επιστραφεί το σχήμα ενός αντικειμένου γνωρίζοντας το το id του, μπορεί να χρησιμοποιηθεί η μέθοδος getTable(). Εξαιτίας του σχήματος της βάσης, μαζί με τη βάση επιστρέφει και το πεδίο tableList, που περιέχει τους πίνακες της βάσης.

Παράδειγμα κλήσης

GET	/system/table/{tableId}

Αίτημα

```
Content-Type: application/json
```

Απάντηση

```
Access-Control-Allow-Credentials:true
Content-Type: application/json
Set-Cookie:vertx-web.session=95ebfb55-c595-47f0-95c8-3c774e8f148e; Path=/
{
  "jsonSchema": {
    "schema": {
      "type": "object",
      "properties": {}
    }
  },
  "name": "testObject",
  "id": "b0c55593-d583-46c7-9443-a806cfa71a24"
}
```

Περιγραφή μεταβλητών

getTable (String tableId <AsyncResult<JsonObject>> resultHandler)	**Input:** id του πίνακα, resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη)
	Output: Τα στοιχεία του πίνακα μαζί με το πεδίο jsonSchema σε JsonObject μορφή είτε κωδικός λάθους.

Για τη διαγραφή ενός πίνακα χρησιμοποιείται η μέθοδος deleteTable(). Κατά τη διαγραφή του πίνακα καταστρέφεται το σχήμα και όλα τα αντικείμενα που έχουν δημιουργηθεί με βάση αυτό. Σε περίπτωση σφάλματος επιστρέφεται μήνυμα αποτυχίας.

Παράδειγμα κλήσης

DELETE	/system/table/{tableId}

Αίτημα

```
Content-Type: application/json
```

Απάντηση

```
Status Code:200 OK
```

Περιγραφή μεταβλητών

deleteTable (String databaseId, <AsyncResult<JsonObject>> resultHandler)	**Input:** το id του πίνακα προς διαγραφή, resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη).
	Output: Κωδικός επιτυχίας ή αποτυχίας.

5.5. Διαχείριση αντικειμένων

Μετά τη δημιουργία της δομής που έχουν τα αντικείμενα, επόμενο στάδιο είναι αυτό της δημιουργίας, ενημέρωσης και διαγραφής αντικειμένων όπως και αυτό της υποβολής ερωτημάτων στα αντικείμενα που έχουν δημιουργηθεί. Η πλατφόρμα παρέχει όλες αυτές τις λειτουργίες πετυχαίνοντας ένα βασικό στόχο της, την απόκρυψη των βημάτων της υλοποίησης μιας εφαρμογής από το επίπεδο της αποθήκευσης των αντικειμένων μέχρι εκείνο της παροχής πληροφορίας μέσω υπηρεσιών.

Το endpoint της εφαρμογής είναι σχετικά απλό. Αποτελείται από ένα συνδυασμό του ονόματος της βάσης δεδομένων ακολουθούμενο από τον ζητούμενο πόρο. Όπου πόρος είναι το όνομα του πίνακα, κατ' επέκταση το όνομα του αντικειμένου.

Για την εισαγωγή νέου αντικειμένου στην εφαρμογή γίνεται χρήση της rootingApiPostHandling () η οποία είναι προσβάσιμη μέσω της μεθόδου POST. Η πλατφόρμα καταλαβαίνει τη βάση δεδομένων που πρέπει να χρησιμοποιήσει, καθώς και στο σχήμα του αντικειμένου από το url. Το αντικείμενο

μπαίνει στο κύριο σώμα της μεθόδου. Αρχικά, η πλατφόρμα ελέγχει τη δομή του αντικειμένου και το κατά πόσο αυτή είναι σύμφωνη με τη δομή που έχει οριστεί για το αντικείμενο. Οι έλεγχοι που γίνονται αφορούν ένα μεγάλο σύνολο χαρακτηριστικών που η προτυποποίηση JSON-Schema παρέχει, και σε περίπτωση σφάλματος επιστρέφεται μήνυμα σχετικό με το σφάλμα και το πεδίο. Σε κάθε αντικείμενο που εισέρχεται στη πλατφόρμα, εκείνη του δίνει ένα μοναδικό id. Στις περιπτώσεις ο χρήστης δώσει ο ίδιος το πεδίο “id” στο αντικείμενό του, τότε θα χρησιμοποιηθεί αυτό που παρείχε ο ίδιος ο χρήστης. Σε αυτό το ενδεχόμενο γίνεται ένας επιπλέον έλεγχος και αν το “id” του χρήστη χρησιμοποιείται από άλλο αντικείμενο του ίδιου τύπου, επιστρέφει σφάλμα.

Παράδειγμα κλήσης

POST	/api/{{databaseName}}/{{tableName}}

Αίτημα

```
Content-Type: application/json
{
  "name": "revertan",
  "projectDescription": {
    "titles": [
      {
        "name": "Revertan"
      },
      {
        "name": "Adaptive and usable object oriented stores"
      }
    ]
  },
  "completed": true,
  "students": 1
}
```

Απάντηση

```
Content-Type: application/json
Set-Cookie:vertx-web.session=95ebfb55-c595-47f0-95c8-3c774e8f148e; Path=/
{
  "id": "b0c55593-d583-46c7-9443-a806cfa71a24",
  "name": "revertan",
  "projectDescription": {
    "titles": [
      {
        "name": "Revertan"
      },
      {
        "name": "Adaptive and usable object oriented stores"
      }
    ]
  },
  "completed": true,
  "students": 1
}
```

Περιγραφή μεταβλητών

rootingApiPostHandling (String database, String table, JsonObject objectToSave <AsyncResult<JsonObject>> resultHandler)	**Input:** το όνομα της βάσης δεδομένων, το όνομα του πίνακα, το αντικείμενο σε JsonObject μορφή, resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη).
	Output: το αντικείμενο που δημιουργήθηκε σε JsonObject μορφή είτε κωδικός λάθους.

Για την ενημέρωση ενός αντικειμένου χρησιμοποιείται η μέθοδος PUT. Συγχρόνως, γίνονται όλοι οι έλεγχοι σχετικά με τη δομή του αντικειμένου προς ενημέρωση, όπως πραγματοποιούνται και στην εισαγωγή.

Παράδειγμα κλήσης

POST	/api/{{databaseName}}/{{tableName}}/{{objectId}}

Αίτημα

```
Content-Type: application/json
{
  "name": "revertan",
  "projectDescription": {
    "titles": [
      {
        "name": "Revertan"
      },
      {
        "name": "Adaptive and usable object oriented stores"
      }
    ]
  },
  "completed": false,
  "students": 1
}
```

Απάντηση

```
Access-Control-Allow-Credentials:true
Content-Type: application/json
Content-Type: application/json
{
  "name": "revertan",
  "projectDescription": {
    "titles": [
      {
        "name": "Revertan"
      },
      {
        "name": "Adaptive and usable object oriented stores"
      }
    ]
  },
  "completed": false,
  "students": 1
}
```

Περιγραφή μεταβλητών

rootingApiPostHandling(String database, String table, String objectId, JsonObject objectToUpdate <AsyncResult<JsonObject>> resultHandler)	**Input:** το όνομα της βάσης δεδομένων, το όνομα του πίνακα, το id του αντικειμένου προς ενημέρωση, το αντικείμενο σε JsonObject μορφή, resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη).
	Output: Είτε το αντικείμενο που ενημερώθηκε σε JsonObject μορφή είτε κωδικός λάθους.

Για τη διαγραφή ενός αντικειμένου χρησιμοποιείται η μέθοδος DELETE.

Παράδειγμα κλήσης

DELETE	/api/{{databaseName}}/{{tableName}}/{{objectId}}

Αίτημα

```
Content-Type: application/json
```

Απάντηση

```
Status Code:200 OK
```

Περιγραφή μεταβλητών

rootingApiDeleteHandling(String database, String table, String objectId, <AsyncResult<JsonObject>> resultHandler)	**Input:** το όνομα της βάσης δεδομένων, το όνομα του πίνακατο id του αντικειμένου προς διαγραφή, resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη).
	Output: Είτε το αντικείμενο που ενημερώθηκε σε JsonObject μορφή είτε κωδικός λάθους.

Για την ανάγνωση ενός μόνο αντικειμένου γνωρίζοντας το id του χρησιμοποιείται η μέθοδος rootingApiGetHandling()

Παράδειγμα κλήσης

GET	/api/{{databaseName}}/{{tableName}}/{{objectId}}

Αίτημα

```
Content-Type: application/json
```

Απάντηση

```
Access-Control-Allow-Credentials:true
Content-Type: application/json
Content-Type: application/json
{
  "name": "revertan",
  "projectDescription": {
    "titles": [
      {
        "name": "Revertan"
      },
      {
        "name": "Adaptive and usable object oriented stores"
      }
    ]
  },
  "completed": false,
  "students": 1
}
```

Περιγραφή μεταβλητών

rootingApiGetHandling (String database, String table, String objectId, JsonObject objectToUpdate <AsyncResult<JsonObject>> resultHandler)	**Input:** το όνομα της βάσης δεδομένων, το όνομα του πίνακα, το αντικείμενο σε JsonObject μορφή, το id του αντικειμένου, resultHandler(ασύγχρονος χειριστής ώστε να επιστρέψει η απάντηση όταν είναι έτοιμη).
	Output: Είτε το αντικείμενο που ζητήθηκε σε JsonObject μορφή είτε κωδικός λάθους.

Στο πλαίσιο της δημιουργίας μιας τύπου REST υπηρεσίας η τύπου GET κλήση δέχεται επιπλέον παραμέτρους καλύπτοντας τις παρακάτω περιπτώσεις:

- Προβολή όλων των αντικειμένων ενός χρήστη

GET	/api/{{databaseName}}/{{tableName}}/{{objectId}}

- Σελιδοποίηση των αποτελεσμάτων

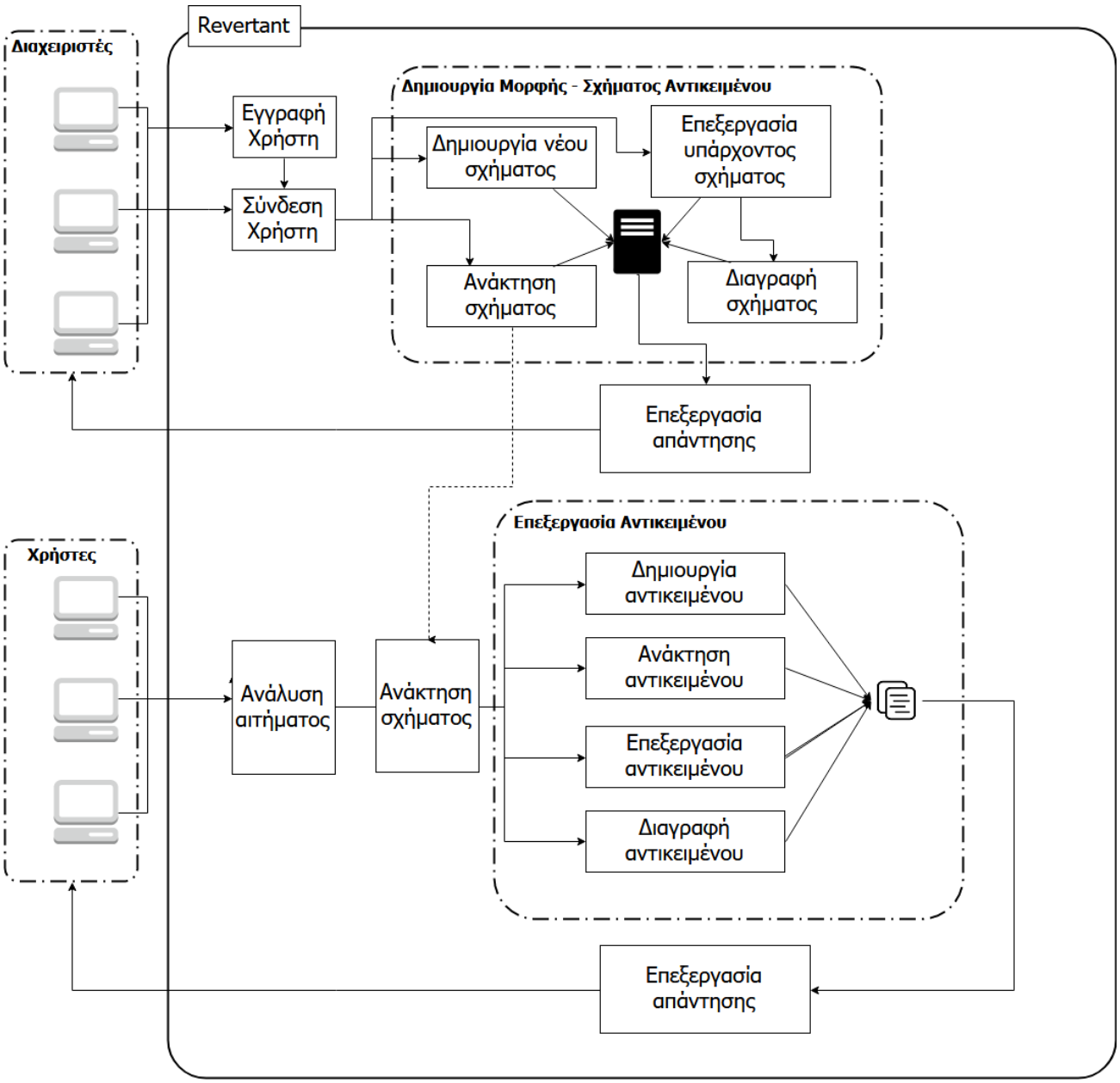
GET	/api/{{databaseName}}/{{tableName}}/{{objectId}}?offset={{off}}&limit={{lim}}

- Ταξινόμηση ως προς κάποιο πεδίο

GET	/api/{{databaseName}}/{{tableName}}/{{objectId}}?orderAsc={{field}}
GET	/api/{{databaseName}}/{{tableName}}/{{objectId}}? orderDesc ={{field }}

5.6. Σύνθεση υπηρεσιών

Η κάθε μια από τις υπηρεσίες που διατίθενται δεν μπορούν να καλύψουν το σύνολο των αναγκών. Ο συνδυασμός τους είναι αυτός που θα επιτρέψει την ομαλή διαχείριση των αντικειμένων όπως αυτός αποτυπώνεται στο σχήμα 5.1. Οι ρόλοι των χρηστών είναι δύο, οι διαχειριστές που μπορούν να επεξεργαστούν σχήματα αντικειμένων και οι απλοί χρήστες που αλληλοεπιδρούν με αυτά τα σχήματα δημιουργώντας, επεξεργάζοντας, ανακτώντας και διαγράφοντας αντικείμενα.



Σύνθεση υπηρεσιών – Σχήμα 5.1

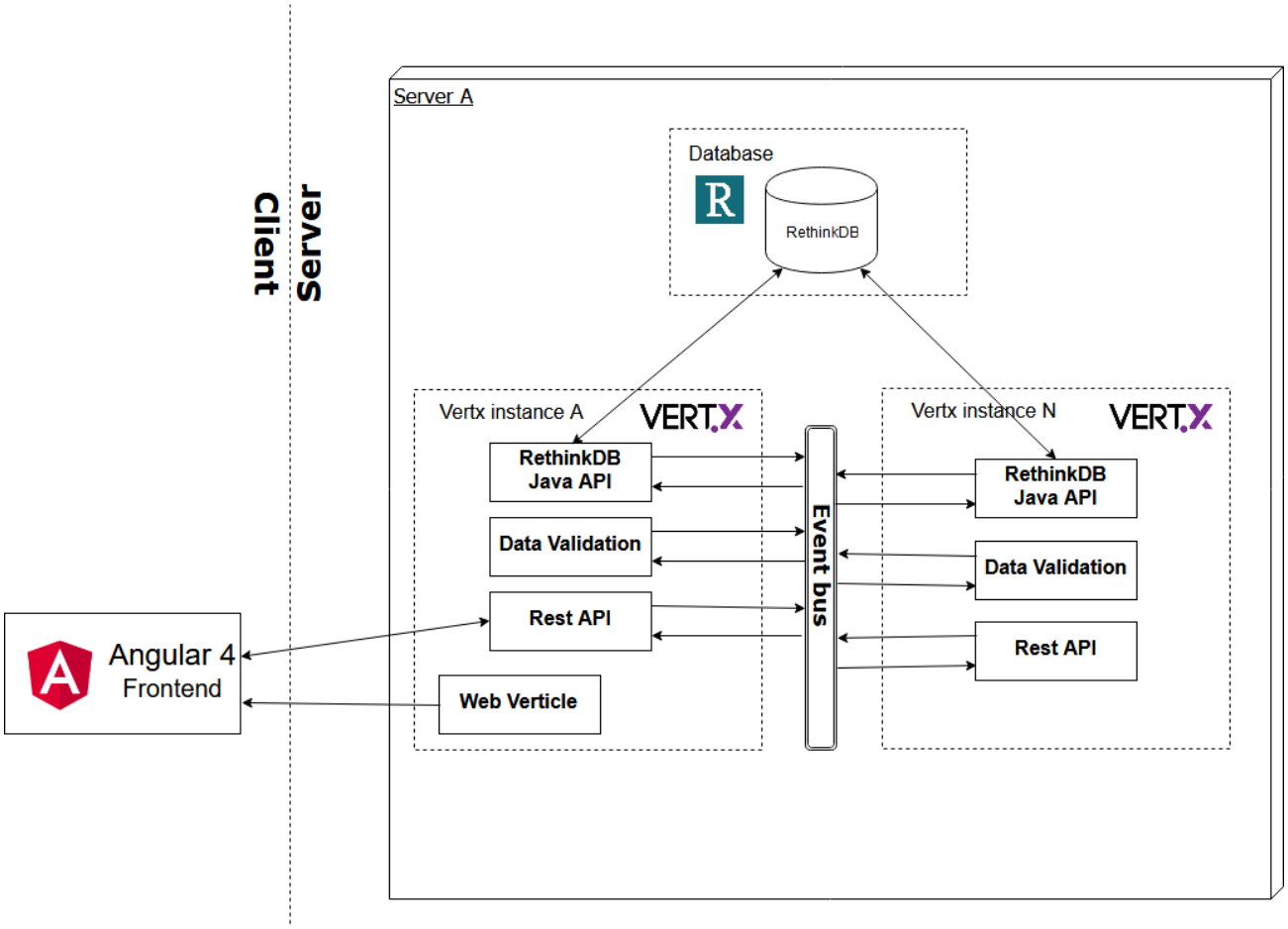
6. Πιλοτική υλοποίηση για την προσαρμοστική και εύχρηστη διαχείριση αντικειμένων

6.1. Αρχιτεκτονική της εφαρμογής

Όπως αναφέρεται στην παρούσα διπλωματική εργασία, στο επίπεδο της υλοποίησης κατασκευάστηκε μια πλατφόρμα προσαρμοστικής διαχείρισης αντικειμένων με προοπτική αξιοποίησης πόρων του Cloud. Για τη διαχείριση των αντικειμένων μιας εφαρμογής, χρειάζεται η υλοποίηση ενός μεγάλου πλήθους λειτουργιών που ξεκινούν από την αποθήκευση και εκτείνονται μέχρι την παροχή των αντικειμένων μέσω υπηρεσιών. Παράλληλα, έτοιμες λύσεις που υπάρχουν σε αυτό το χώρο, περιορίζουν τους οργανισμούς στη χρήση συγκεκριμένων Cloud παρόχων. Σε αυτά τα πλαίσια, και δίχως να υπάρχει στρατηγικός σχεδιασμός τέτοιος που να περιορίζει την πλατφόρμα σε μια συγκεκριμένη κατηγορία εφαρμογών που μπορεί να υποστηρίξει, κατασκευάστηκε το Revertan.

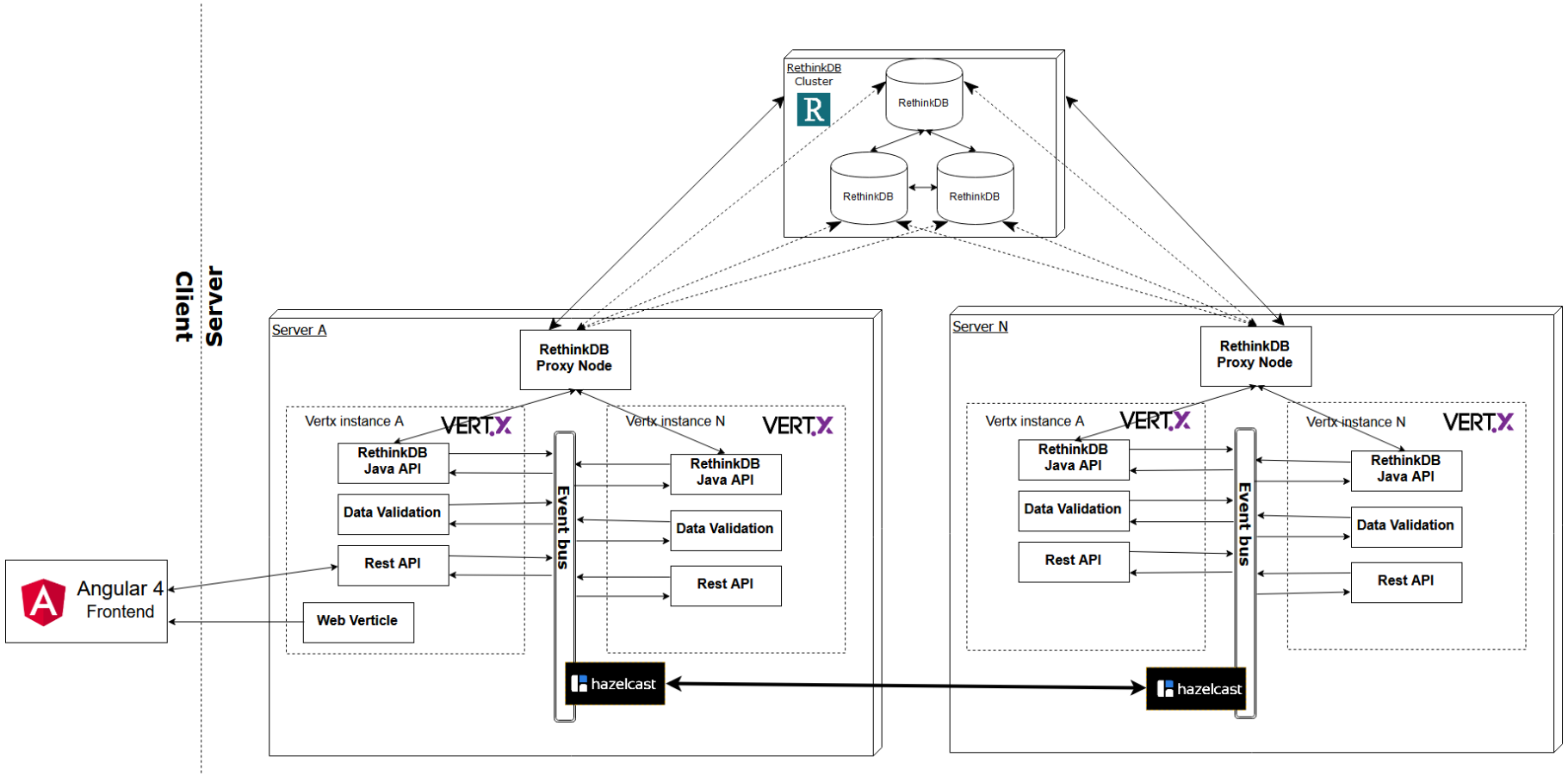
Υπάρχουν δύο διαφορετικές αρχιτεκτονικές που υποστηρίζονται, δίνοντας λύση στις ανάγκες εξυπηρέτησης εφαρμογών τόσο με μικρό όγκο αιτημάτων, όσο και εκείνων που απαιτούν εκατομμύρια ταυτόχρονες συνδέσεις χρηστών. Στην πρώτη αρχιτεκτονική που αποτυπώνεται στο σχήμα 6.1 υπάρχει μονάχα ένα μηχάνημα. Στο μηχάνημα αυτό έχει εγκατασταθεί το JDK 1.8 ως περιβάλλον εκτέλεσης του Vert.x 3. Επιπροσθέτως, η εγκατεστημένη βάση είναι η τελευταία έκδοση της RethinkDB, ενώ για τη γραφική διεπαφή έχει χρησιμοποιηθεί η Angular 4. Η λειτουργικότητα του Revertan έχει χωριστεί σε τέσσερα αυτόνομα επίπεδα με τη χρήση του Vert.x (Verticles), το RethinkDB Java Api στο οποίο βρίσκεται η επίσημη προγραμματιστική διεπαφή για Java της RethinkDB, το Data Validation Verticle το οποίο αναλαμβάνει τον έλεγχο της δομής των αντικειμένων και το Rest API Verticle που διαχειρίζεται τα αιτήματα που φτάνουν στο σύστημα και παρέχει το σύνολο των λειτουργιών εκτός του Server. ΥΠΆΡΧΕΙ, επίσης, το Web Verticle το οποίο παρέχει τα στατικά αρχεία διεπαφής της Angular. Τα τρία επίπεδα (όλα εκτός εκείνου του Web Verticle) επικοινωνούν μεταξύ τους μέσω ασύγχρονων μηνυμάτων.

Αυτή η υλοποίηση όπως έχει περιεγραφεί ως τώρα, χρησιμοποιεί μόνο από έναν μέχρι, σε ακραίες περιπτώσεις, τέσσερις πυρήνες του επεξεργαστή στον υπολογιστή που τρέχει. Πολύ εύκολα μπορεί να δηλωθεί στο Vert.x να χρησιμοποιήσει περισσότερους πυρήνες δημιουργώντας περισσότερα στιγμιότυπα εκτέλεσης (multiple instances).



Αρχιτεκτονική Revertan για εφαρμογές μικρής κλίμακας – Σχήμα 6.1

Σε Cloud περιβάλλον (γενικότερα στη πλειοψηφία των διαθέσιμων επιλογών) μπορεί να χρησιμοποιηθεί στο σχήμα 6.2 που προσφέρει αξιοποίηση περισσότερων του ενός μηχανημάτων. Στο σχήμα 6.2 περιγράφεται η πιο ακραία κλιμάκωση που μπορεί να υποστηρίξει το σύνολο των τεχνολογιών που χρησιμοποιήθηκαν. Οι βασικές αρχές είναι ίδιες με τη περίπτωση του ενός υπολογιστή, με τις διαφορές να εντοπίζονται στο γεγονός ότι υπάρχει ένα cluster της RethinkDb και επικοινωνία πολλών υπολογιστών μεταξύ τους με το Hazelcast να επεκτείνει το Vert.x Event Bus μέσω δικτύου σε περισσότερους υπολογιστές. Η τελευταία διαφοροποίηση αφορά τη χρήση ενός RethinkDB Proxy Node, συνδεσμολογία που προτείνεται στις οδηγίες χρήσης της RethinkD.



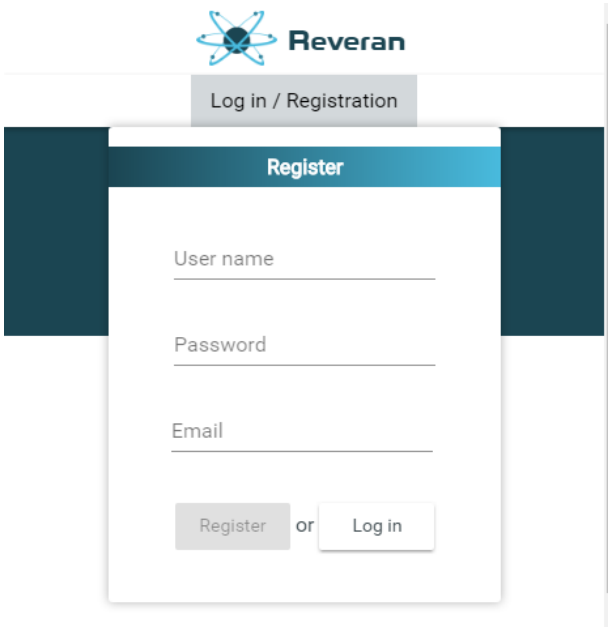
Αρχιτεκτονική Revertan για εφαρμογές μεγάλης κλίμακας – Σχήμα 6.2

6.2. Περιβάλλον ιστού

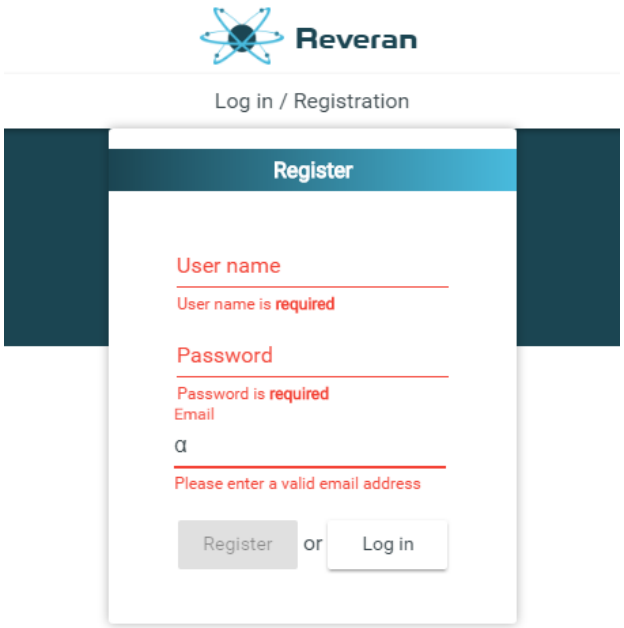
Στη διεπαφή του χρήστη έγινε προσπάθεια να δημιουργηθεί ένα ευχάριστο γραφικό περιβάλλον, λιτό και λειτουργικό που να ακολουθεί συγχρόνως τις πιο πρόσφατες τάσεις στη σχεδίαση ιστοτόπων. Έχει υλοποιηθεί με χρήση Angular 4 και Typescript που παρέχει βελτιστοποιημένο κώδικα Javascript (ES5). Υπάρχει απόλυτη προσαρμοστικότητα σε οθόνες υπολογιστών, ταμπλετών και κινητών τηλεφώνων, αλλάζοντας σε κάθε περίπτωση με βάση την ανάλυση της οθόνης ώστε να προβάλλεται όλο το περιεχόμενο (responsive css). Σύμφωνα με την κατηγοριοποίηση των λειτουργιών, παρατίθενται μερικά στιγμιότυπα χρήσης της εφαρμογής.

6.2.1. Εγγραφή νέου χρήστη

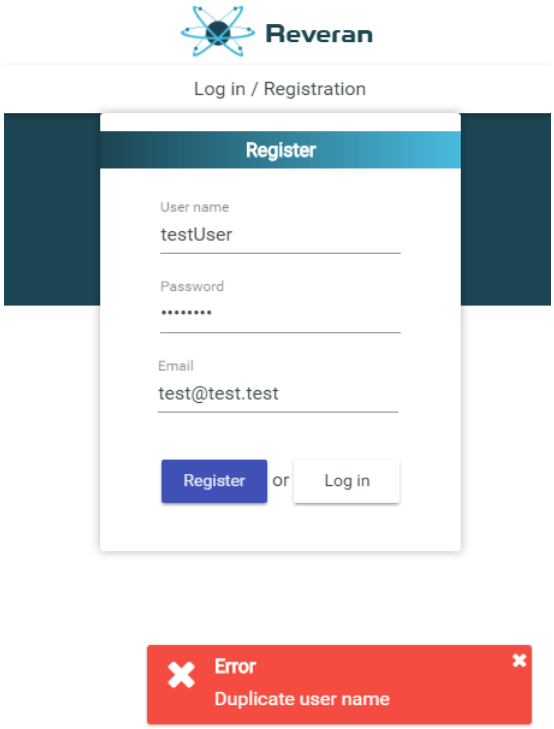
Κατά την πρώτη επίσκεψη στο σύστημα, το μόνο πράγμα που είναι διαθέσιμο στην εφαρμογή είναι η εγγραφή χρήστη. Στο σχήμα 6.3 βλέπουμε τη φόρμα εγγραφής της εφαρμογής που αποτελείται από τρία πεδία. Για την ορθότητα των στοιχείων, όπως συμπλήρωση όλων των πεδίων, μοναδικό όνομα χρήστη στην εφαρμογή και σωστού email, παράγονται κατάλληλα μηνύματα σφάλματος, όπως φαίνεται στο σχήμα 6.4. Την ίδια στιγμή, τα στοιχεία που παρέχει ο χρήστης της εφαρμογής ελέγχονται και από το API που έχει κατασκευαστεί για την εγγραφή των χρηστών (σχήμα 6.5).



Εγγραφή στην εφαρμογή – Σχήμα 6.3



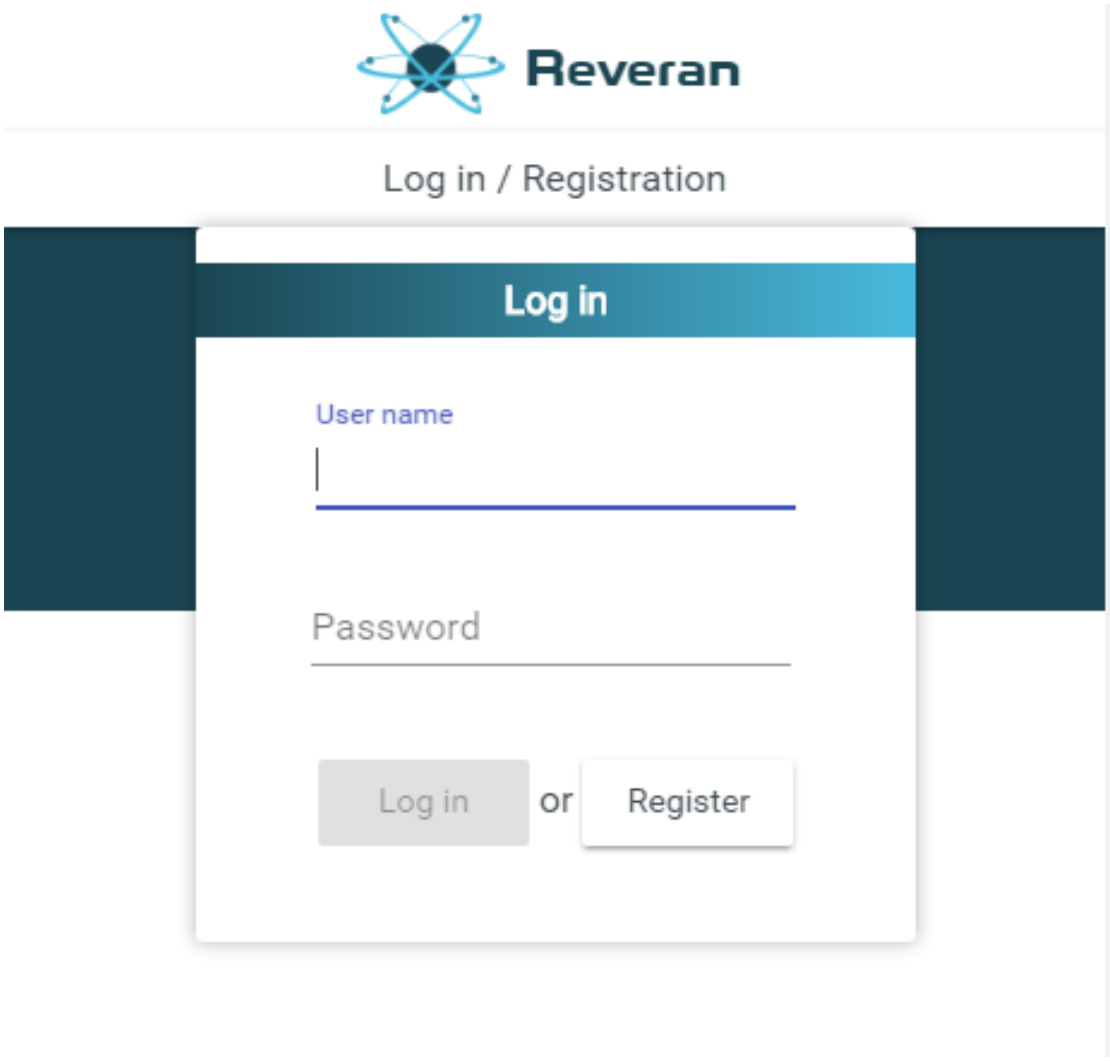
Έλεγχοι εγκυρότητας φόρμας εγγραφής – Σχήμα 6.4



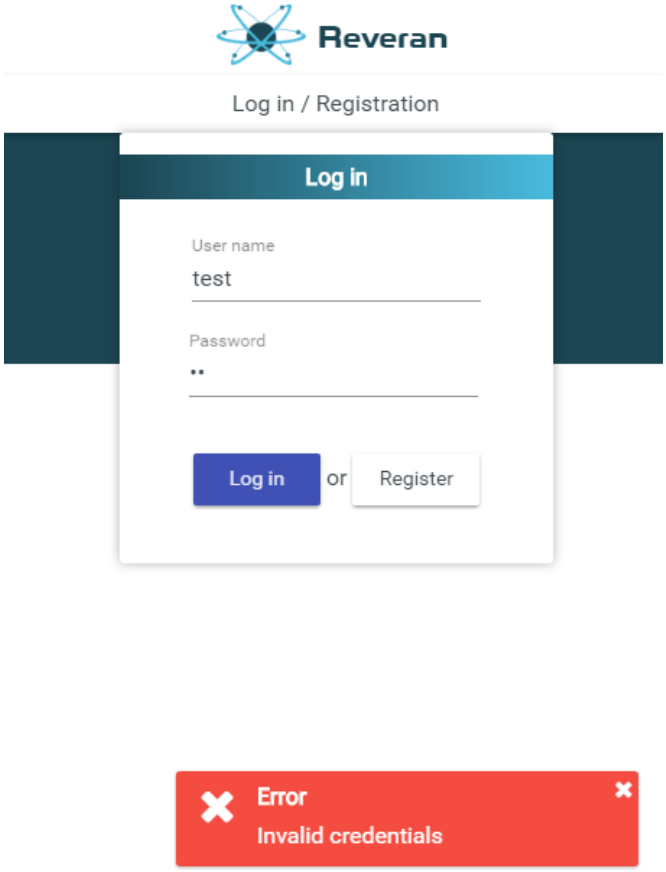
Έλεγχοι εγκυρότητας φόρμας εγγραφής μετά την λανθασμένη υποβολή – Σχήμα 6.5

6.2.2. Είσοδος στο σύστημα

Για την είσοδο του χρήστη έχει δημιουργηθεί μια φόρμα σύνδεσης – σχήμα 6.6 στο ίδιο εικαστικό πρότυπο με εκείνο της φόρμας εγγραφής. Και σε αυτή τη περίπτωση γίνονται έλεγχοι, ενώ μηνύματα σφάλματος που έρχονται από την κλήση της υπηρεσίας εμφανίζονται όταν είναι απαραίτητο – σχήμα 6.7.



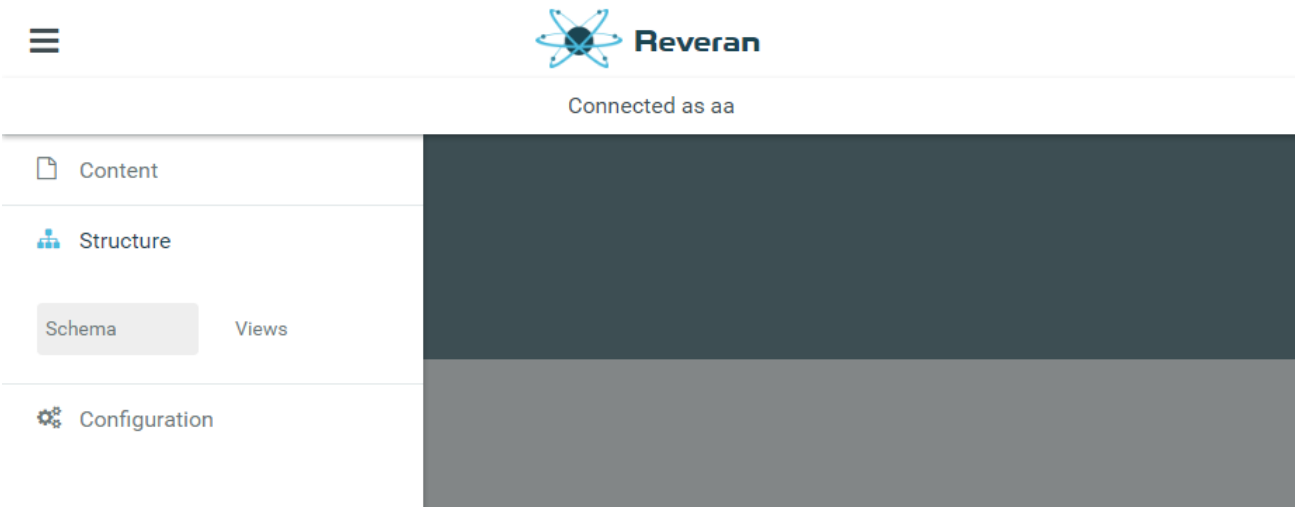
Έλεγχοι εγκυρότητας φόρμας εγγραφής μετά την λανθασμένη υποβολή – Σχήμα 6.6



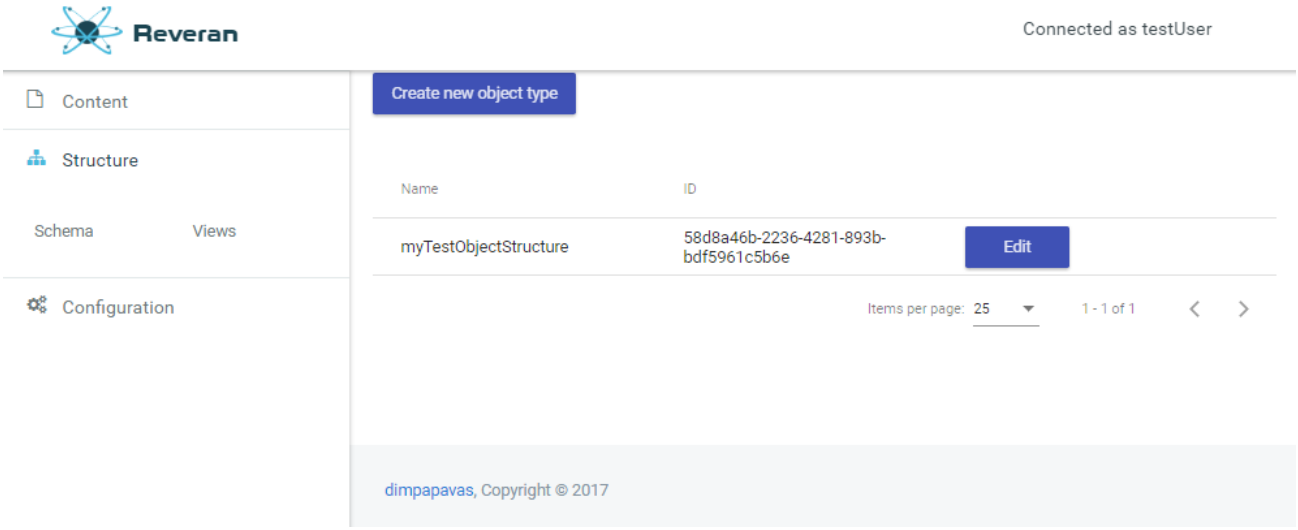
Έλεγχοι εγκυρότητας φόρμας εγγραφής μετά την λανθασμένη υποβολή – Σχήμα 6.7

6.2.3. Προβολή λίστας διαθέσιμων αντικειμένων

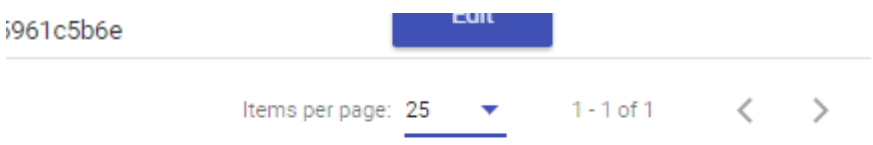
Με την επιτυχή σύνδεση στην εφαρμογή εμφανίζεται το μενού - σχήμα 6.8 από το οποίο μπορούμε να προηγηθούμε στις διαφορετικές λειτουργίες που παρέχονται. Πατώντας στην καρτέλα «Structure» και μετά το κουμπί «Schema» έχουμε πρόσβαση στη λειτουργικότητα που αφορά τα είδη αντικειμένων που έχουν περιγραφεί στην εφαρμογή – σχήμα 6.9. Από αυτή τη σελίδα, μπορούμε είτε να επεξεργαστούμε μια υπάρχουσα δομή αντικειμένου, είτε μπορούμε να προσθέσουμε μια νέα. Παρατηρούμε ότι στο κάτω μέρος του πίνακα υπάρχει παραμετροποίηση για τη σελιδοποίηση που επιθυμούμε να έχει – σχήμα 6.10. Δεν πρόκειται για μια εικαστική παρέμβαση, αλλά για ένα μέτρο προφύλαξης από περιττό φόρτο στην εφαρμογή.



Μενού χρήσης της εφαρμογής – Σχήμα 6.8



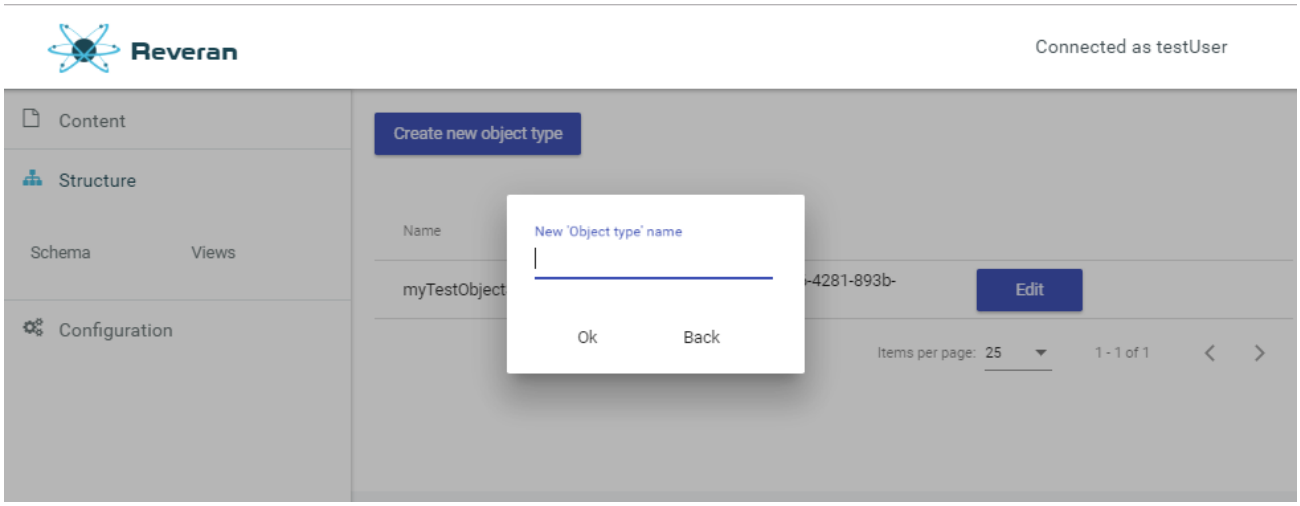
Φόρμα προβολής διαθέσιμων ειδών αντικειμένων – Σχήμα 6.9



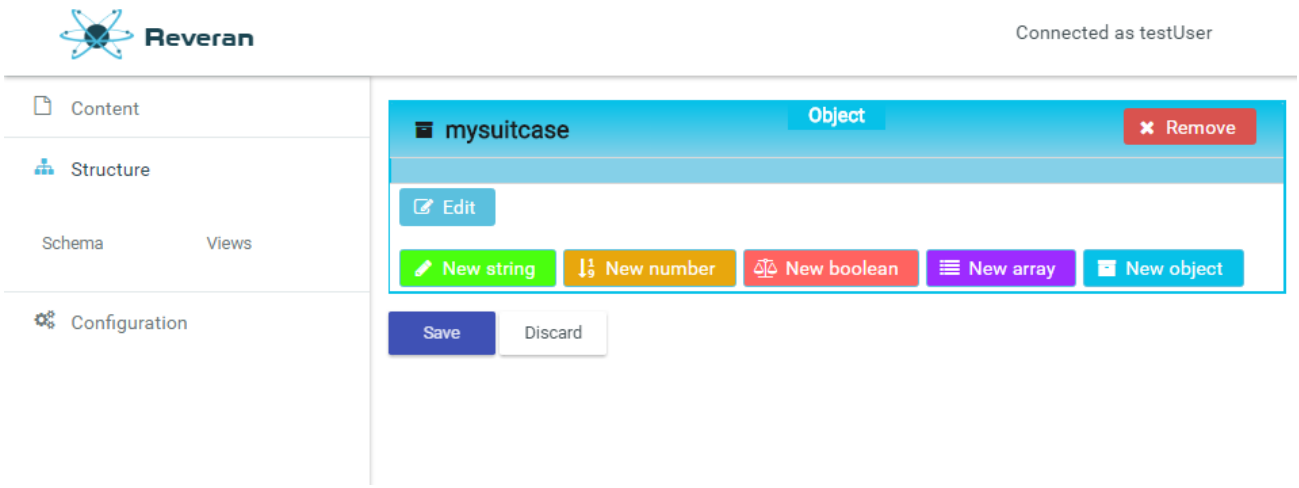
Σελιδοποίηση πινάκων – Σχήμα 6.10

6.2.4. Δημιουργία νέου είδους αντικειμένου

Πατώντας το κουμπί «Create new object type» προβάλλεται ένα αναδυόμενο παράθυρο που ζητά το όνομα του καινούργιου είδους αντικειμένου – σχήμα 6.11. Δίνοντας ένα έγκυρο όνομα κατευθυνόμαστε στη σελίδα δημιουργίας και επεξεργασίας σχήματος όπως βλέπουμε στο σχήμα 6.12. Πατώντας κάποιο από τα κουμπιά «New string», «New number», «New boolean», «New array» και «New object» παρέχεται η δυνατότητα προσθήκης νέων ορισμάτων τύπου κειμένου, αριθμού, λογικών, λίστας και άλλου υπο-αντικειμένου αντίστοιχα.

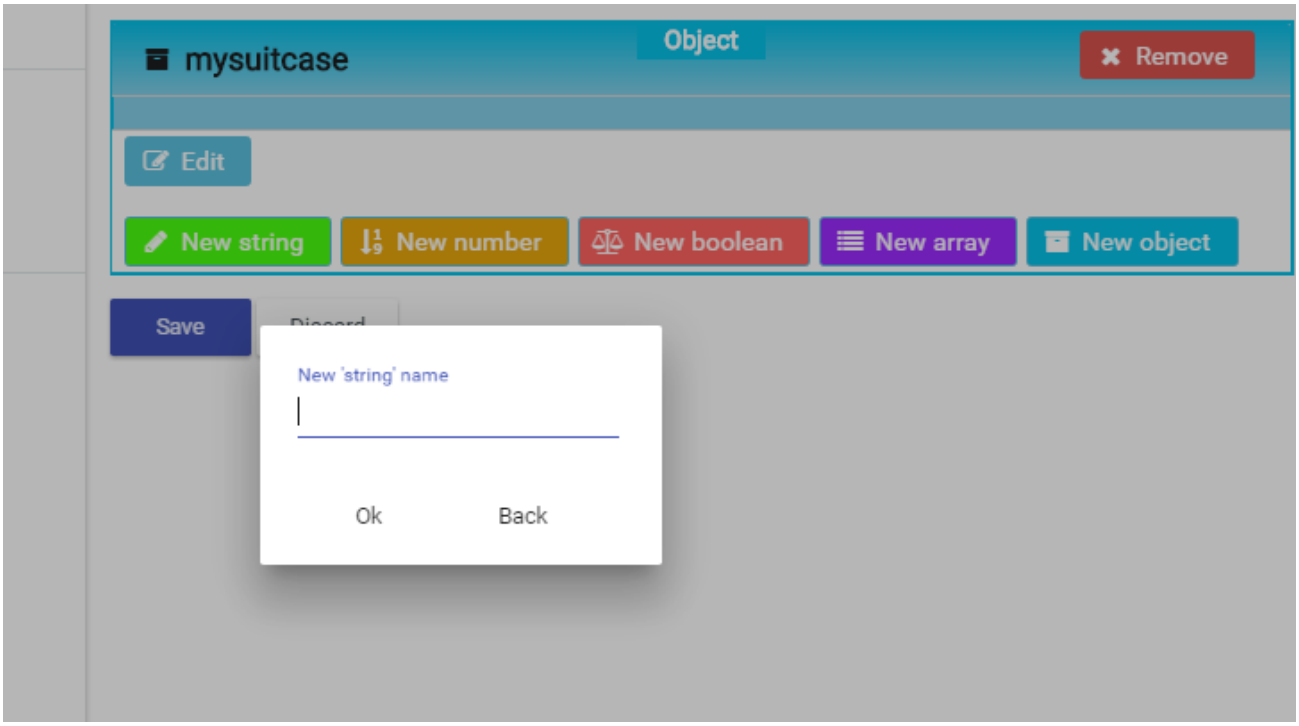


Δημιουργία τύπου αντικειμένου (Βήμα Α) – Σχήμα 6.11



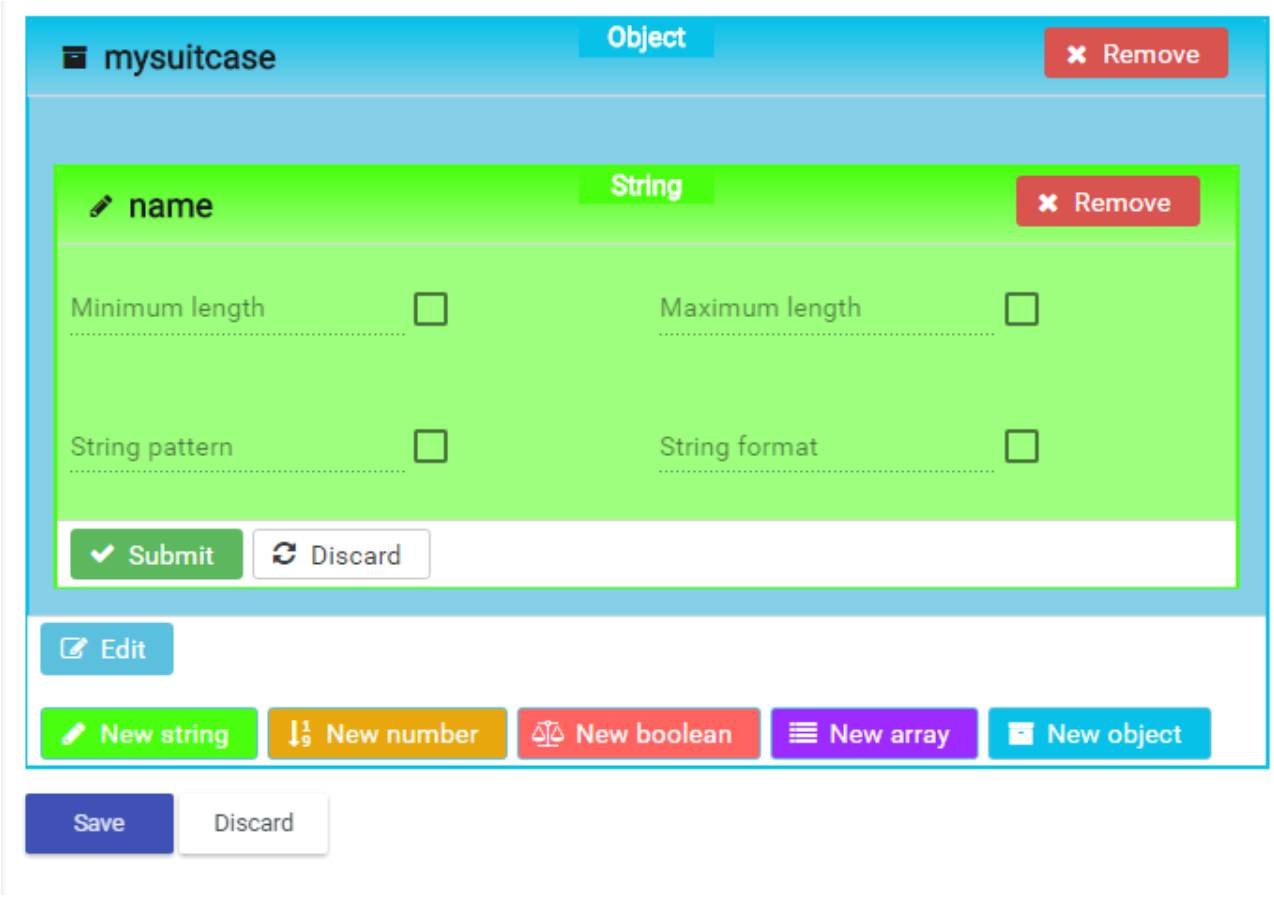
Δημιουργία και επεξεργασία τύπου αντικειμένου (Βήμα Β) – Σχήμα 6.12

Αναλυτικότερα, κατά τη προσθήκη οποιουδήποτε είδους ορίσματος στο πρώτο βήμα ζητείται η εισαγωγή του ονόματός του (σχήμα 6.13).



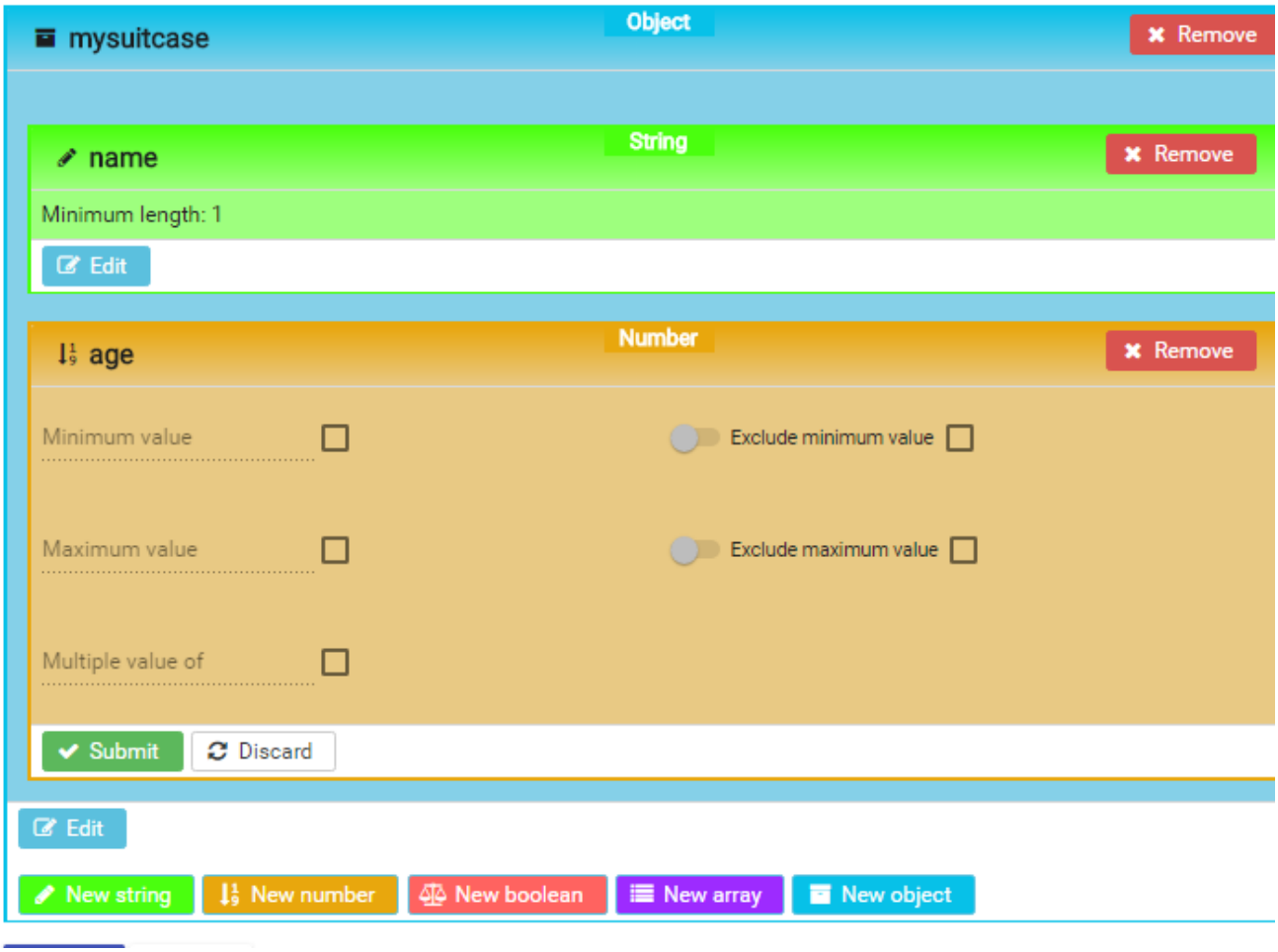
Δημιουργία ορίσματος, όνομα ορίσματος – Σχήμα 6.13

Ξεκινώντας με ένα όρισμα τύπου κειμένου παρατηρούνται κατά το edit οι επιλογές του σχήματος 6.14. Αυτές οι επιλογές αποτελούν προαιρετικά χαρακτηριστικά ενός ορίσματος. Στη περίπτωση του κειμένου, έχουμε τη δυνατότητα να χειριστούμε το ελάχιστο και το μέγιστο αριθμό χαρακτήρων, καθώς και να περιγράψουμε τη μορφή και πρότυπο του ορίσματος.



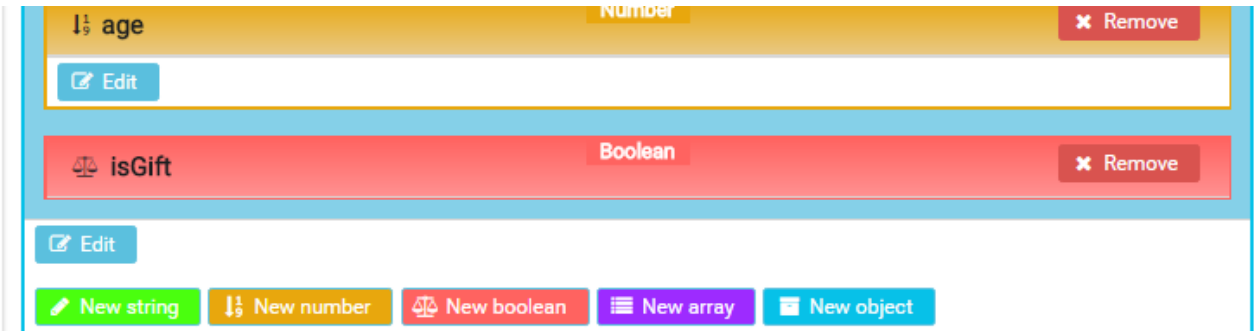
Δημιουργία ορίσματος κειμένου – Σχήμα 6.14

Επιλέγοντας την προσθήκη αριθμού μπορούμε να θέσουμε περιορισμούς μέγιστης/ελάχιστης τιμής καθώς και το αν αποτελεί πολλαπλάσιο κάποιου αριθμού – σχήμα 6.15



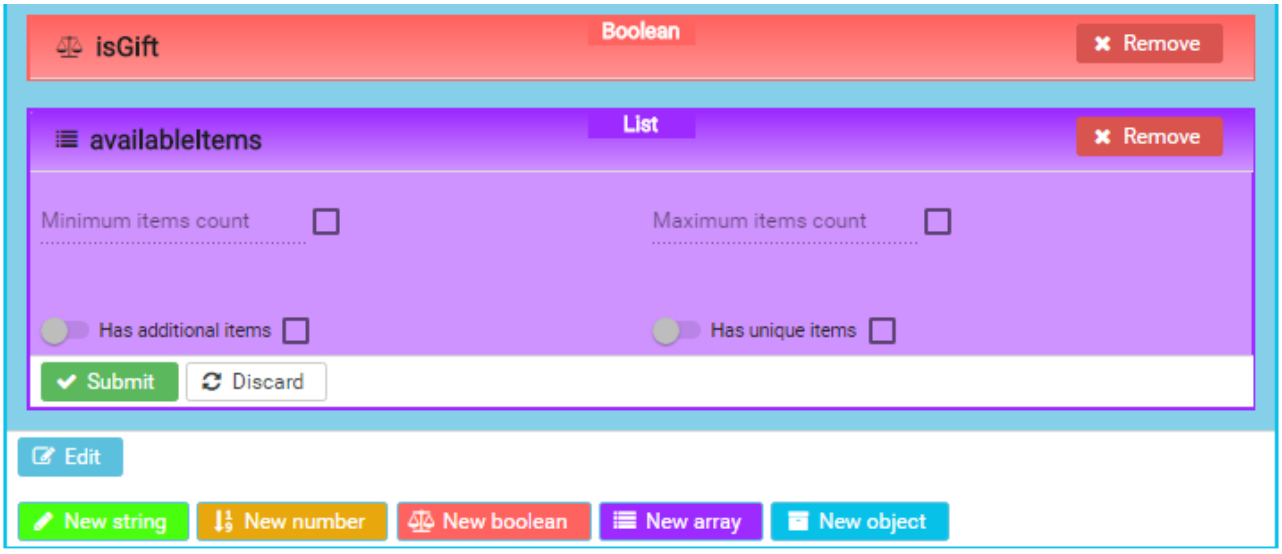
Δημιουργία ορίσματος κειμένου – Σχήμα 6.15

Κατά τη δημιουργία ενός λογικού ορίσματος δεν υπάρχει λόγος ορισμού κάποιου περιορισμού, αφού οι τιμές που μπορεί να πάρει το πεδίο είναι “true - false” – σχήμα 6.16



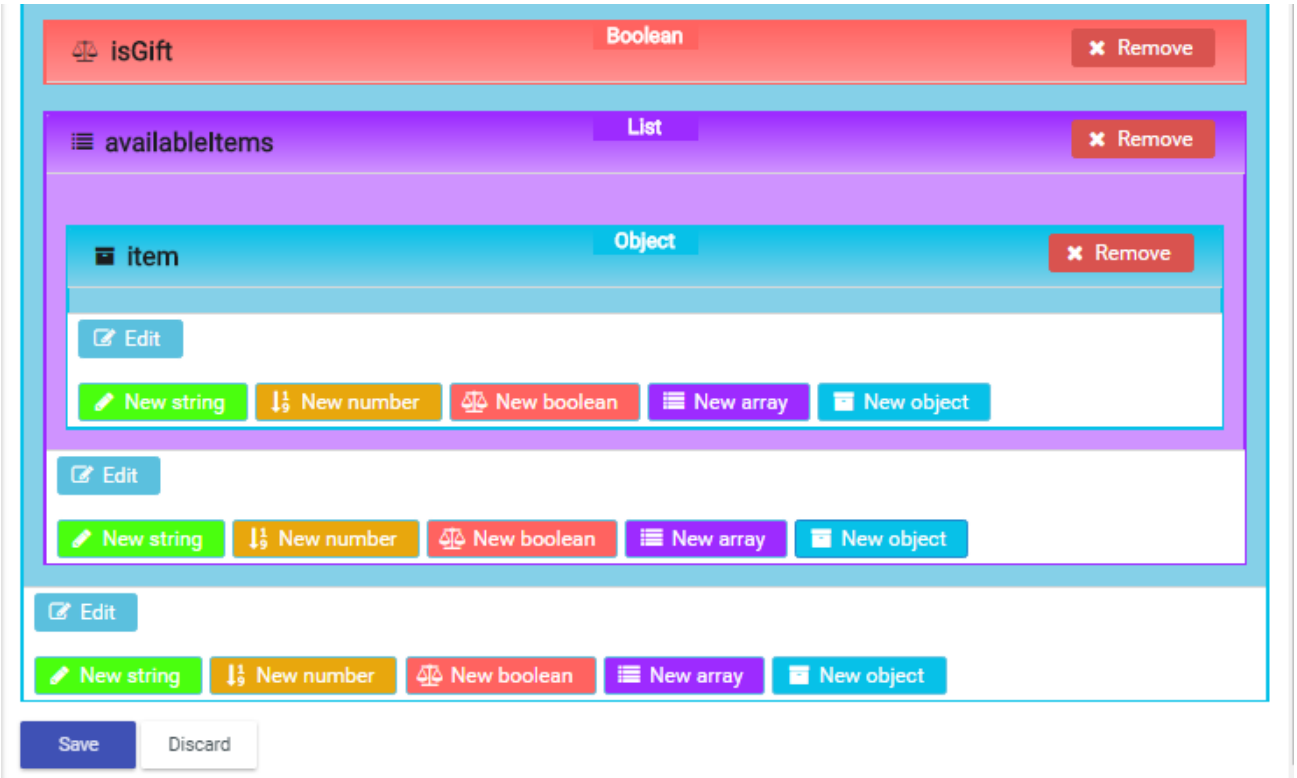
Δημιουργία λογικού ορίσματος – Σχήμα 6.16

Στην επιλογή προσθήκης λίστας παρέχεται η δυνατότητα ορισμού ελάχιστου / μέγιστου αριθμού στοιχείων καθώς και ελέγχου μοναδικότητας των στοιχείων που φιλοξενεί – Σχήμα 6.17.



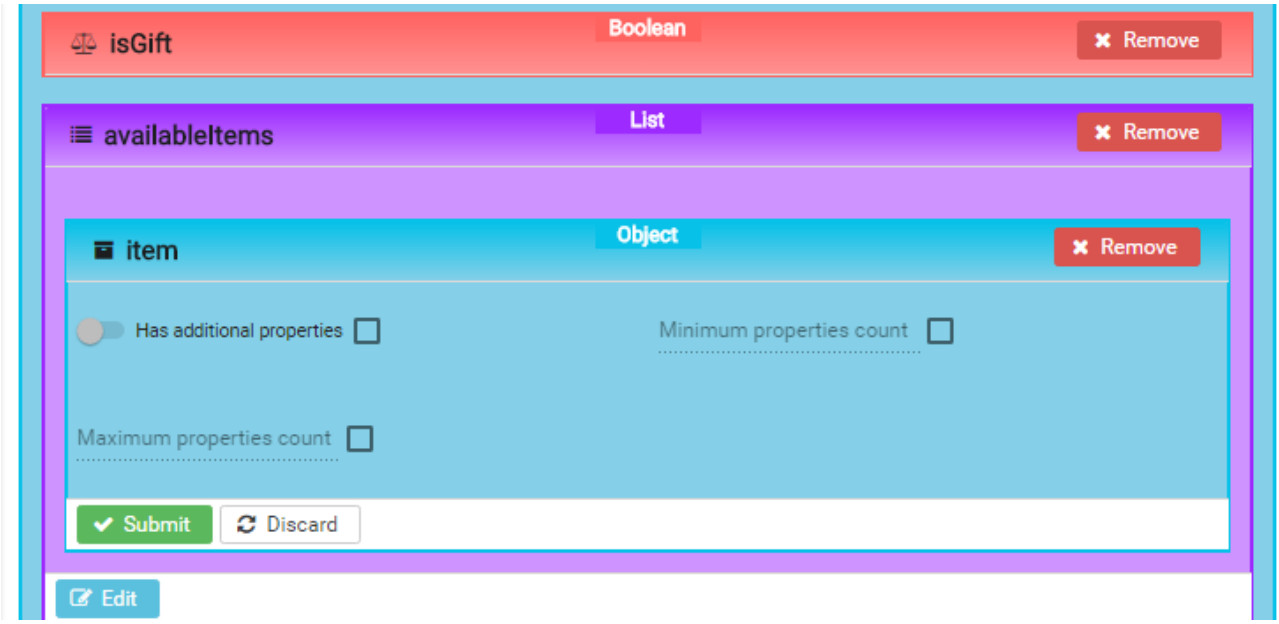
Δημιουργία λίστας – Σχήμα 6.17

Μια λίστα, όπως είναι το λογικό, πρέπει να έχει τη δυνατότητα φιλοξενίας άλλων αντικειμένων. Στο σχήμα 6.18 μπορούμε να παρατηρήσουμε ότι μια λίστα δίνει επιλογές προσθήκης ορισμάτων ίδιων με αυτά που παρέχει ένα όρισμα τύπου αντικειμένου. Στο σχήμα παρατηρούμε, επίσης, την προσθήκη ενός υπο-αντικειμένου εντός της λίστας



Περιγραφή της δομής μιας λίστας – Σχήμα 6.18

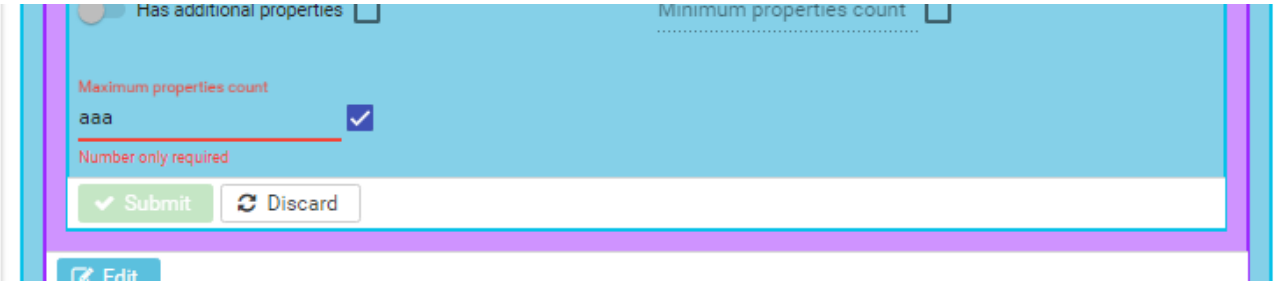
Μπορούμε να ορίσουμε τα χαρακτηριστικά του κάθε ορίσματος ανεξάρτητα του πόσο βαθιά βρίσκονται στο δέντρο ορισμάτων ενός αντικειμένου.



Χαρακτηριστικά υπο-αντικειμένου λίστας – Σχήμα 6.19

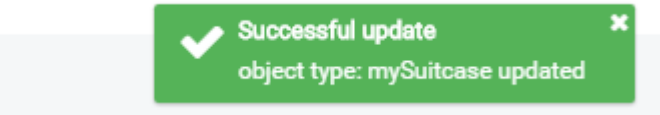
Στο σχήμα 6.19 μπορούμε να δούμε ένα όρισμα τύπου αντικειμένου (στην προκειμένη περίπτωση λειτουργεί ως υπο-αντικείμενο μιας λίστας). Υπάρχει η υποστήριξη της μη περιγραφής των ορισμάτων ενός αντικειμένου και ο ορισμός του μέγιστου / ελάχιστου αριθμού ορισμάτων.

Σε όλες τις παραπάνω φόρμες ελέγχεται η ορθότητα των επιλογών που ζητά ο χρήστης, όπως ενδεικτικά μπορούμε να παρατηρήσουμε στο σχήμα 6.20.



Έλεγχος επιλογών χρήστη – Σχήμα 6.20

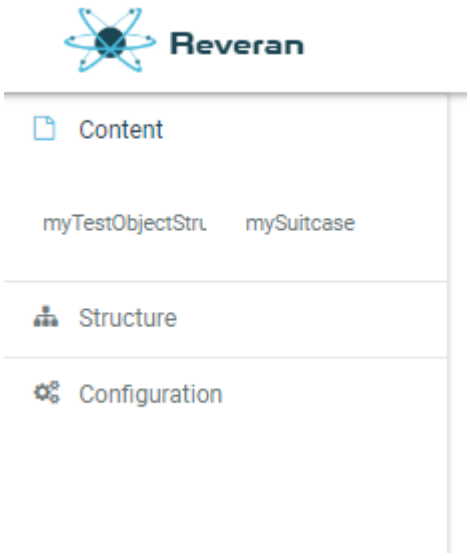
Κατά την αποθήκευση του νέου αντικειμένου υπάρχει ένδειξη επιτυχίας που επιβεβαιώνει την ορθότητα του σχήματος από τη μεριά του server και την καταχώρισή του – σχήμα 6.21.



Επιτυχής καταχώριση νέου σχήματος – Σχήμα 6.21

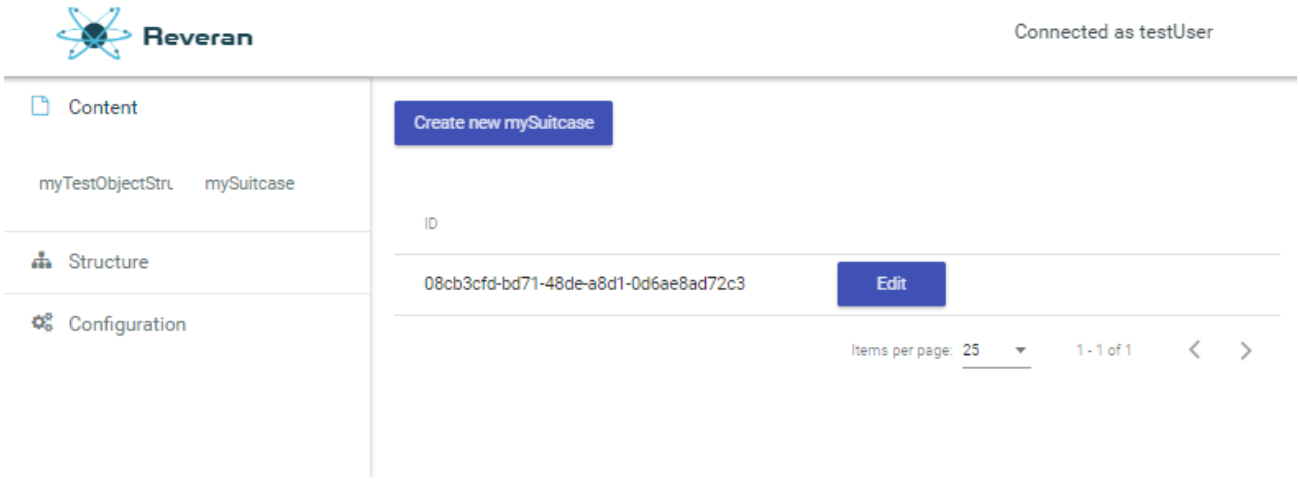
6.2.1. Προβολή λίστας αντικειμένων βάση τύπου

Στο σύστημα υπάρχει δυνατότητα προβολής των αντικειμένων που έχουν δημιουργηθεί. Για τη προβολή της λίστας των διαθέσιμων αντικειμένων ο χρήστης από την καρτέλα του μενού «Content», επιλέγει τον τύπο αντικειμένων που τον ενδιαφέρει – σχήμα 6.22.



Μενού διαχείρισης αντικειμένων βάση τύπου – Σχήμα 6.22

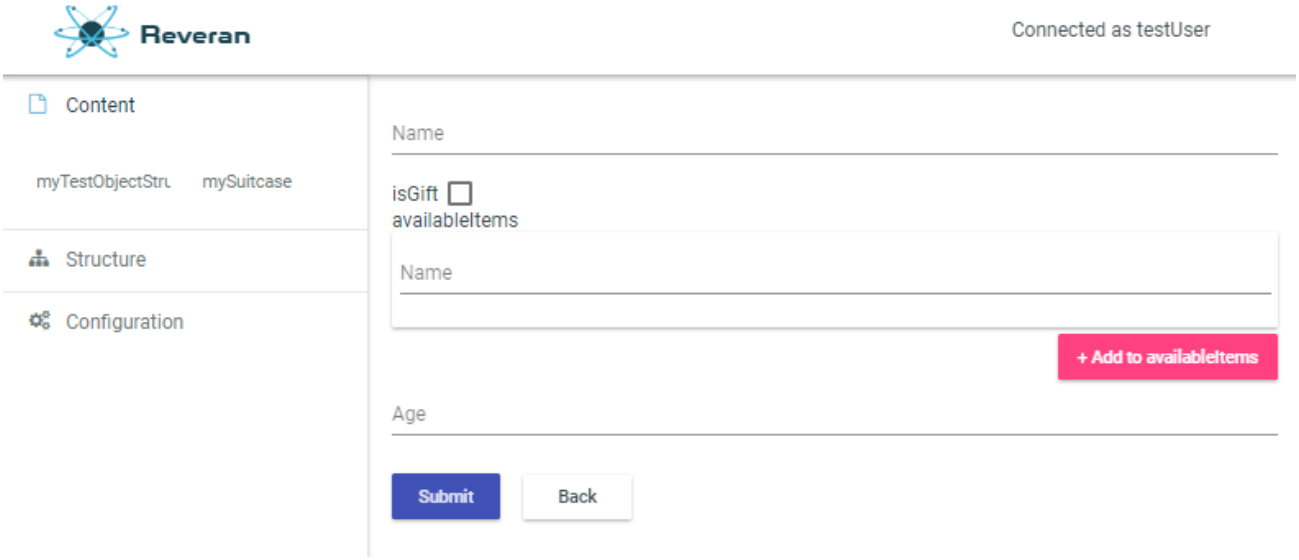
Από τη σελίδα διαχείρισης υπάρχουν επιλογές δημιουργίας νέου αντικειμένου του επιλεγμένου τύπου, καθώς και προβολή σε σελιδοποιημένη λίστα των αντικειμένων που υπάρχουν στο σύστημα με δυνατότητα επεξεργασίας σχήμα 6.23.



Πίνακας αντικειμένων συγκεκριμένου τύπου – Σχήμα 6.23

6.2.1. Εισαγωγή και επεξεργασία αντικειμένων

Για την εισαγωγή και την επεξεργασία αντικειμένων έχει ενσωματωθεί στην εφαρμογή ένα έτοιμο Angular 4 component, παραμετροποιημένο στην υπόλοιπη πλατφόρμα. Το component που χρησιμοποιήθηκε δέχεται ως είσοδο μια περιγραφή ενός αντικειμένου της μορφής JSON-schema και παράγει τη φόρμες σαν και αυτή του σχήματος 6.24. Η χρήση προτυποποίησης στο επίπεδο δημιουργίας της δομής των αντικειμένων βοήθησε στη χρήση αυτού του εργαλείου.



Επεξεργασία αντικειμένου – Σχήμα 6.24

7. Συμπεράσματα

Στο πλαίσιο της παρούσης διπλωματικής εργασίας, έγινε μελέτη των σύγχρονων, κατανεμημένων αρχιτεκτονικών που χρησιμοποιούνται στα υπολογιστικά νέφη σήμερα. Μελετήθηκαν τα απαραίτητα χαρακτηριστικά μιας εφαρμογής ώστε να μπορεί με αποδοτικό τρόπο να αξιοποιήσει πόρους που προέρχονται από συστάδες υπολογιστών. Εν συνεχεία, αναλύθηκαν τα χαρακτηριστικά που πρέπει να υποστηρίζει ένα σύστημα που στόχο έχει την προσαρμοστική διαχείριση και αποθήκευση αντικειμένων μιας τρίτης εφαρμογής σε ένα κατανεμημένο περιβάλλον. Το σύστημα παρέχει υπηρεσίες δημιουργίας της δομής των αντικειμένων, αποθήκευσης, ενημέρωσης και ανάκτησης μέσω REST API προσαρμοσμένο στις ανάγκες της εφαρμογής-χρήστη. Έπειτα, κατασκευάστηκε μια φιλική διεπαφή χρήστη, ώστε ένα μέρος των υπηρεσιών να είναι διαθέσιμες χωρίς να απαιτείται η παραγωγή κώδικα από τη μεριά του προγραμματιστή. Τοιουτοτρόπως, ικανοποιείται ένας από τους βασικούς στόχους της παρεχόμενης εφαρμογής, εκείνος, δηλαδή, της παροχής έτοιμης λύσης για τα στάδια που καλείται να υλοποιήσει ένας προγραμματιστής ξεκινώντας από το επίπεδο της φυσικής αποθήκευσης αντικειμένων και καταλήγοντας σε αυτό της παροχής υπηρεσιών που προκύπτουν από αυτά.

Αντίστοιχες πλατφόρμες που είναι διαθέσιμες εμπορικά βασίζονται στο υπολογιστικό νέφος των παρόχων τους, με αποτέλεσμα αν και παρέχουν μια προσαρμοστική διαχείριση αντικειμένων, δεσμεύουν τους χρήστες τους στη δικιά τους υποδομή. Η αλλαγή παρόχου μπορεί να αποδεχθεί μια εξαιρετικά πολύπλοκη διαδικασία στο επίπεδο του χειρισμού των αλλαγών που πρέπει να γίνουν σε τρίτες εφαρμογές που χρησιμοποιούν την υπηρεσία. Στα πλαίσια εκπόνησης της παρούσας εργασίας, υλοποιήθηκε εκ του μηδενός ένα σύστημα διαχείρισης αντικειμένων, ενώ από το επίπεδο της σχεδίασης μέχρι εκείνο της τελικής υλοποίησης πραγματώθηκε με τη χρήση τεχνολογιών που να επιτρέπουν την αποδοτική εκτέλεση στο Cloud.

Το σύστημα επιδέχεται αρκετές επεκτάσεις πολυεπίπεδα. Εξάλλου, οι τεχνολογίες που χρησιμοποιήθηκαν στη μεριά του server (πακετάρισμα του κώδικα υπό μορφή Verticles) και στη μεριά του client (Angular 4 με κατασκευή Components) αποδεικνύουν τη διάθεση της παρούσης διπλωματικής για παροχή επεκτάσεων από τις κοινότητες ανοιχτού λογισμικού. Το λογισμικό που υλοποιήθηκε αποτελεί μια βάση για τη δημιουργία μιας πιο ολοκληρωμένης πλατφόρμας. Στις

απαραίτητες προεκτάσεις θα ήταν χρήσιμη η υποστήριξη και άλλων βάσεων δεδομένων, με διαφορετικά χαρακτηριστικά που θα μπορούν να αξιοποιηθούν από το σύστημα, όπως η υποστήριξη συναλλαγών (transactions). Το REST API μπορεί να επεκταθεί περεταίρω, υποστηρίζοντας πιο πολύπλοκα ερωτήματα. Καταληκτικά, μια σημαντική επέκταση μπορεί να είναι εκείνη του χειρισμού ροών δεδομένων (data streams) καθώς στο χώρο δεν υπάρχουν πολλές επιλογές διαχείρισης, και από τεχνολογικής πλευράς μπορεί να υποστηριχθεί από το σύστημα.

8. Βιβλιογραφία

[1] K. Stanoevska-Slabeva and T. Wozniak, “Cloud basics – an introduction to Cloud computing,” in Grid and Cloud Computing, pp. 47–61, Springer-Verlag, 2010.

[2] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, “A break in the Clouds: Towards a Cloud definition,” ACM SIGCOMM: Computer Communication Review, vol. 39, pp. 50–55, January 2009.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “Above the Clouds: A Berkeley view of Cloud computing,” Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, February 2009.

[4] D. Robinson, Amazon Web Services Made Simple: Learn how Amazon EC2, S3, SimpleDB and SQS Web Services enables you to reach business goals faster. Emereo Publishing, 2008.

[5] K. Keahey and T. Freeman, “Science Clouds: Early experiences in Cloud computing for scientific applications,” in CCA ’08: Cloud Computing and Its Applications, 2008.

[6] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, “Elastic management of cluster-based services in the Cloud,” in ACDC’09: Proceedings of the 1st workshop on automated control for datacenters and Clouds, pp. 19–24, ACM, 2009.

[7] D. Sanderson, Programming Google App Engine: Build and Run Scalable Web Apps on Google’s Infrastructure. O’Reilly, 2009.

[8] T. Redkar, Windows Azure Platform. Apress, 2010.

[9] “Google Docs.” https://docs.google.com.

[10] “Microsoft Office Live.” https://office.live.com.

[11] Yuping Xing and Yongzhao Zhan. Virtualization and cloud computing. In Future Wireless Networks and Information Systems, pages 305{312. Springer, 2012.

[12] Yulin Yao. Cloud computing: A practical overview between year 2009 and year 2015. International Journal of Organizational and Collective Intelligence (IJOCI), 2015.

[13] Amazon Simple Storage Service API Reference API Version 2006-03-01, 2017.

[14] Amazon Relational Database Service (RDS) https://aws.amazon.com/rds/

[15] M. D. Hill, “What is scalability?” ACM SIGARCH Computer Architecture News, vol. 18, 1990.

[16] https://angular.io/

[17] http://vertx.io/

[18] https://rethinkdb.com/

[19] https://hazelcast.com/