ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη συστήματος παρακολούθησης και διαχείρισης εξυπηρετητών

Δημοσθένης Καστρινάκης
Α.Μ. 71344119

Εισηγητής: Δρ. Γεώργιος Πρεζεράκος

ΑΘΗΝΑ
Μάρτιος 2023

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΥΞΗ ΣΥΣΤΗΜΑΤΟΣ ΠΑΡΑΚΟΛΟΥΘΗΣΗΣ ΚΑΙ ΔΙΑΧΕΙΡΙΣΗΣ ΕΞΥΠΗΡΕΤΗΤΩΝ

Δημοσθένης Καστρινάκης
Α.Μ. 71344119

Εισηγητής:

Δρ. Γεώργιος Πρεζεράκος, Καθηγητής

Εξεταστική Επιτροπή:

Δρ. Γεώργιος Πρεζεράκος, Καθηγητής Georgios Prezerakos Digitally signed by Georgios Prezerakos Date: 2023.03.21 12:34:23 +02'00'

Δρ. Ιωάννης Βογιατζής, Καθηγητής Ioannis Voyiatzis Digitally signed by Ioannis Voyiatzis Date: 2023.03.21 14:55:31 +02'00'

Δρ. Καρκαζής Παναγιώτης, Καθηγητής Panagiotis Karkazis Digitally signed by Panagiotis Karkazis Date: 2023.03.28 10:18:09 +03'00'

Ημερομηνία εξέτασης 10/03/2023

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Δημοσθένης Καστρινάκης του Ιωάννη, με αριθμό μητρώου 71344119 φοιτητής του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής, δηλώνω ότι:

«Είμαι συγγραφέας αυτής της διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών

Δημοσθένης Καστρινάκης

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα διπλωματική εργασία ολοκληρώθηκε μετά από επίμονες προσπάθειες, σε ένα ενδιαφέρον γνωστικό αντικείμενο, όπως αυτό της διαχείρισης εξυπηρετητών. Την προσπάθειά μου αυτή υποστήριξε ο επιβλέπων καθηγητής μου, κ. Γεώργιο Πρεζεράκο, τον οποίο θα ήθελα να ευχαριστήσω. Ακόμα θα ήθελα να ευχαριστήσω την οικογένειά μου και την Λεμονιά Ραφαήλοβιτς για τη συμπαράσταση τους κατά τη διάρκεια των σπουδών μου, καθώς και τον Κλέων Χηράκη για τις πολύτιμες συμβουλές του.

ΠΕΡΙΛΗΨΗ

Σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάλυση και ανάπτυξη συστήματος διαχείρισης εξυπηρετητών, αποτελούμενο από ένα κεντρικό εξυπηρετητή και υλοποίηση πρότυπης βιβλιοθήκης. Ο κεντρικός εξυπηρετητής διαχειρίζεται, συλλέγει, παρουσιάζει και αποστέλλει πληροφορίες σχετικές με το υλικο/λογισμικό των υπολοίπων εξυπηρετητών, ενώ λαμβάνει και εντολές ελέγχου, μέσα από ένα αυθεντικοποιημένο περιβάλλον χρήστη κάνοντας χρήση της βιβλιοθήκης.

ABSTRACT

The purpose of this thesis is the analysis and development of a server management system, consisting of a central server and implementation of a standard library. The central server manages, collects, presents and sends information related to the hardware/software of the other servers, while also receiving control commands, through an authenticated user interface using the library.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: Σχεδίαση και ανάπτυξη διαδικτυακών συστημάτων πραγματικού χρόνου
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: servers, graphs, logging, monitoring, real-time data, REST, TCP, communication protocol, WebSocket, socket programming

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ

SNMP Simple Network Management Protocol

HTML HyperText Markup Language

CSS Cascading Style Sheets

SVG Scalable Vector Graphics

DOM Document Object Model

JSON JavaScript Object Notation

BSON Binary JSON

REST Representational state transfer

AJAX Asynchronous JavaScript and XML

CSRF Cross Site Request Forgery

AES-256 Advanced Encryption Standard 256 bit

SHA-256 Secure Hash Algorithm 256 bit

WS Websocket

TCP Transmission Control Protocol

ΔΚ Δέντρο Κατηγοριοποίησης

VLN/VLQ Variable-Length Number/Variable-Length Quantity

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Η παρούσα διπλωματική εργασία ασχολείται με την ανάπτυξη συστήματος διαχείρισης εξυπηρετητών (servers). Ο στόχος αυτού του συστήματος είναι η εύκολη και γρήγορη συλλογή συμβάντων/αρχείων καταγραφής και μετρικών. Τα συλλεγμένα αυτά δεδομένα κατηγοριοποιούνται και παρουσιάζονται με σκοπό την διεξαγωγή συμπερασμάτων για τα συστήματα που διαχειριζόμαστε. Παράλληλα με την παρουσίαση δεδομένων το σύστημα μας επιτρέπει την αποστολή εντολών στους εξυπηρετητές. Υλοποίηση από το μηδέν.

Παρακάτω αναφέρεται στο SNMP και η πιο πετυχημένη εμπορική του υλοποίηση το HP OpenView Node Manager. Με βάση το SNMP, στην παρούσα διπλωματική εργασία έγινε προσπάθεια δημιουργίας πρότυπου συστήματος διαχείρισης από το μηδέν. Το σύστημα που υλοποιήθηκε στην τρέχουσα διπλωματική εργασία, αποτελεί πρότυπη υλοποίηση συστήματος διαχείρισης εξυπηρετητών, παρόμοιο με το HP OpenView Node Manager.

1.1 SNMP

Το SNMP είναι ένα πρωτόκολλο που χρησιμοποιείται για τη διαχείριση και την παρακολούθηση συσκευών δικτύου όπως δρομολογητές, μεταγωγείς (switches), εξυπηρετητές και άλλες συσκευές δικτύου. Το SNMP έκανε για πρώτη φορά την εμφάνισή του το 1988 και παρέχει ένα σύνολο λειτουργιών με σκοπό να καλύψει την αυξανόμενη ανάγκη για ένα τυποποιημένο απομακρυσμένο τρόπο συλλογής και οργάνωσης πληροφοριών σχετικών με τις συσκευές δικτύου όπως η κατάστασή και ή απόδοσή τους.

Το SNMP δίνει στους διαχειριστές τη δυνατότητα να αλλάξουν την κατάσταση κάποιας συσκευής η οποία λειτουργεί με SNMP. Για παράδειγμα, ο διαχειριστής μπορεί να απενεργοποιήσει τη λειτουργία μιας διεπαφής δικτύου ή μπορεί να ελέγξει την ταχύτητα λειτουργίας της. Με το SNMP μπορεί ακόμη και να παρακολουθείται ή θερμοκρασία μιας συσκευής και να ειδοποιεί τον διαχειριστή όταν η θερμοκρασία της συσκευής υπερβεί κάποιο όριο.

Συνήθως το SNMP σχετίζεται με τη διαχείριση δρομολογητών, αλλά μπορεί να χρησιμοποιηθεί και για την διαχείριση πολλών άλλων τύπων συσκευών. Καθίσταται δυνατή η διαχείριση οποιασδήποτε συσκευής η οποία λειτουργεί με λογισμικό το οποίο επιτρέπει την ανάκτηση πληροφοριών SNMP. Αυτό σημαίνει ότι με το SNMP μπορούν να διαχειριστούν όχι μόνο φυσικές συσκευές, αλλά μπορεί να διαχειριστεί ακόμα και λογισμικό όπως για παράδειγμα λογισμικό εξυπηρετητών ή βάσεις δεδομένων. (Douglas Mauro, 2001)

1.2 HP OpenView Network Node Manager

Το HP OpenView Network Node Manager είναι μια εφαρμογή λογισμικού διαχείρισης δικτύου η οποία παρέχει στους διαχειριστές πληροφοριακών συστημάτων μια σειρά εργαλείων και λειτουργιών που τους βοηθούν στον έλεγχο και την παρακολούθησή τους. Ο έλεγχος και η παρακολούθηση των συσκευών πραγματοποιείται σε πραγματικό χρόνο. Οι συσκευές μπορεί να είναι δρομολογητές, μεταγωγείς, εξυπηρετητές, εκτυπωτές ή άλλες συσκευές οι οποίες είναι συνδεδεμένες στο δίκτυο. Το λογισμικό έχει τη δυνατότητα να χαρτογραφήσει αυτόματα το δίκτυο παρέχοντας μια γραφική αναπαράσταση των συσκευών δικτύου και των διασυνδέσεών τους. Επίσης, παρέχει τη δυνατότητα να φιλτραριστούν ή και να συσχετιστούν συμβάντα επιτρέποντας στους διαχειριστές να δίνουν προτεραιότητα στα σημαντικότερα συμβάντα πιο αποτελεσματικά.

Το SNMP είναι ένα κρίσιμο στοιχείο του HP OpenView Network Node Manager, παρέχοντας το μηχανισμό για τη συλλογή των πληροφοριών που απαιτούνται για την παρακολούθηση και διαχείριση των συσκευών του δικτύου. Οι λειτουργίες του HP OpenView Network Node Manager επιτρέπουν στους διαχειριστές να αξιοποιήσουν το SNMP για την αποτελεσματική παρακολούθηση και τη διαχείριση του πληροφοριακού συστήματός τους.

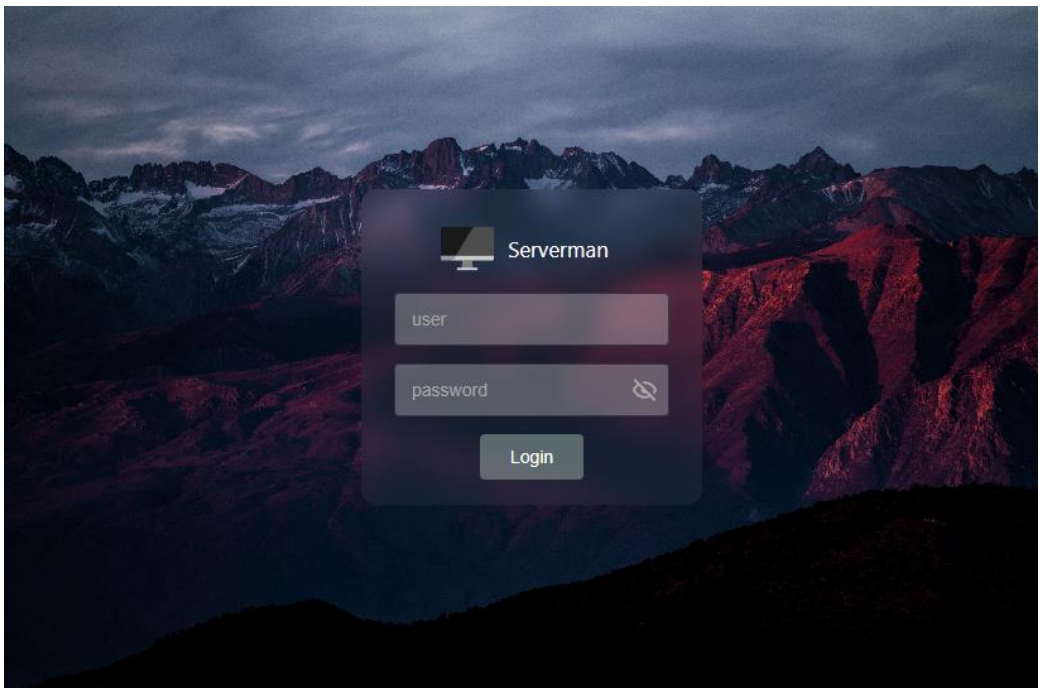
(Lucendo, 1998) (Green, 1998) (HPE OneView, χ.χ.)

ΚΕΦΑΛΑΙΟ 2

ΕΦΑΡΜΟΓΗ

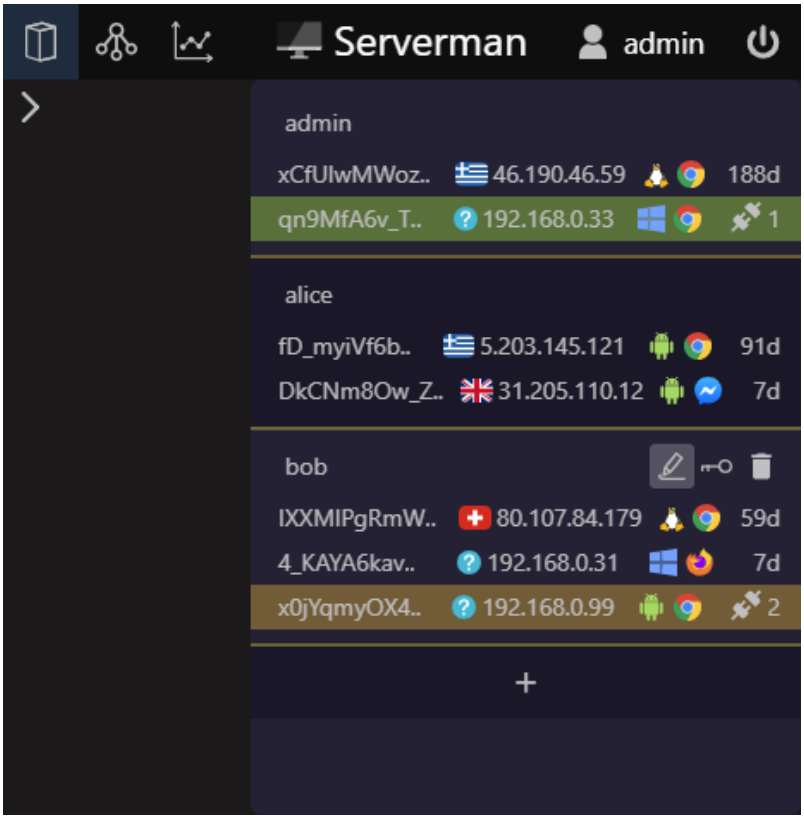
2.1 Χρήστες

Για να μπορέσει κάποιος να χρησιμοποιήσει το σύστημα, είναι αναγκαίο να συνδεθεί από κάποιο λογαριασμό του συστήματος. Το σύστημα υποστηρίζει λογαριασμούς διαχείρισης για τους χρήστες του, ενώ ο κάθε χρήστης του συστήματος μπορεί να συνδεθεί από διαφορετικό ή από τον ίδιο λογαριασμό με κάποιον άλλο χρήστη.



Εικόνα 2.1: Σύνδεση χρήστη (login).

Οι χρήστες του συστήματος μπορούν ταυτόχρονα να χειρίζονται το σύστημα και έχουν όλοι τα ίδια δικαιώματα μεταξύ τους. Ο κάθε λογαριασμός χαρακτηρίζεται από username, password και sessions. Το κάθε session είναι ένα αρχείο στη βάση δεδομένων το οποίο δημιουργείται κατά τη νέα σύνδεση ενός χρήστη και χαρακτηρίζεται από session id, χώρα/περιοχή, IP, browser info, χρόνους τελευταίας σύνδεσης/αποσύνδεσης και αριθμό ενεργών συνδέσεων.



Εικόνα 2.2: Παράθυρο διαχείρισης χρηστών.

Ο χρήστης έχει τη δυνατότητα να επεξεργαστεί τα usernames και passwords οποιουδήποτε χρήστη. Επίσης, μπορεί να δημιουργήσει νέους χρήστες ή να διαγράψει κάποιο χρήστη με τα sessions του ή να διαγράψει μεμονωμένα sessions.

2.2 Οντότητες συστήματος

Το σύστημα αποτελείται από σχεσιακές οντότητες οι οποίες έχουν ιδιότητες και συνεργάζονται μεταξύ τους για να μπορέσει να επιτευχθεί το αποτέλεσμα της εύκολης

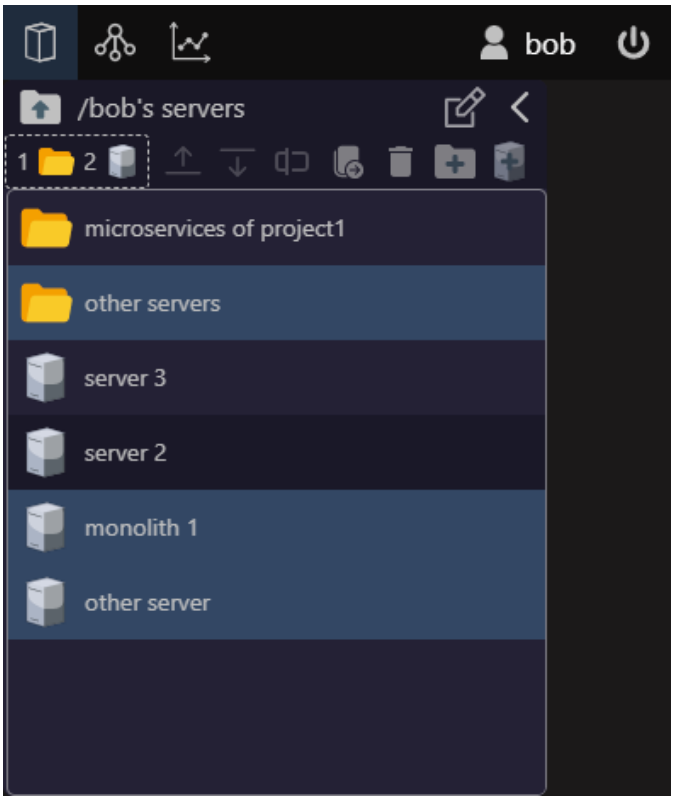
διαχείρισης των εξυπηρετητών. Οι οντότητες χωρίζονται σε είδη ενώ για τα περισσότερα είδη οντοτήτων υπάρχει ένα παράθυρο επεξεργασίας. Ως οντότητες ορίζονται οι εξυπηρετητές (servers), τα δέντρα κατηγοριοποίησης, τα φίλτρα, και οι μετρικές.

Εικόνα 2.3: Εικόνες οντοτήτων συστήματος με σειρά: εξυπηρετητής, δέντρο κατηγοριοποίησης, φίλτρο, μετρική.

Για την αποτελεσματικότερη οργάνωση και πρόσβαση των οντοτήτων, για το κάθε είδος οντότητας, έχει δημιουργηθεί ένα σύστημα αρχείων.

2.3 Σύστημα αρχείων (File System)

Στόχος του συστήματος αρχείων είναι η πιο αποτελεσματική οργάνωση και προσβασιμότητα των οντοτήτων. Το σύστημα αρχείων αποτελείται από ένα παράθυρο εκ του οποίου μπορούν να δημιουργηθούν, να προβληθούν και να επεξεργαστούν οι οντότητες.



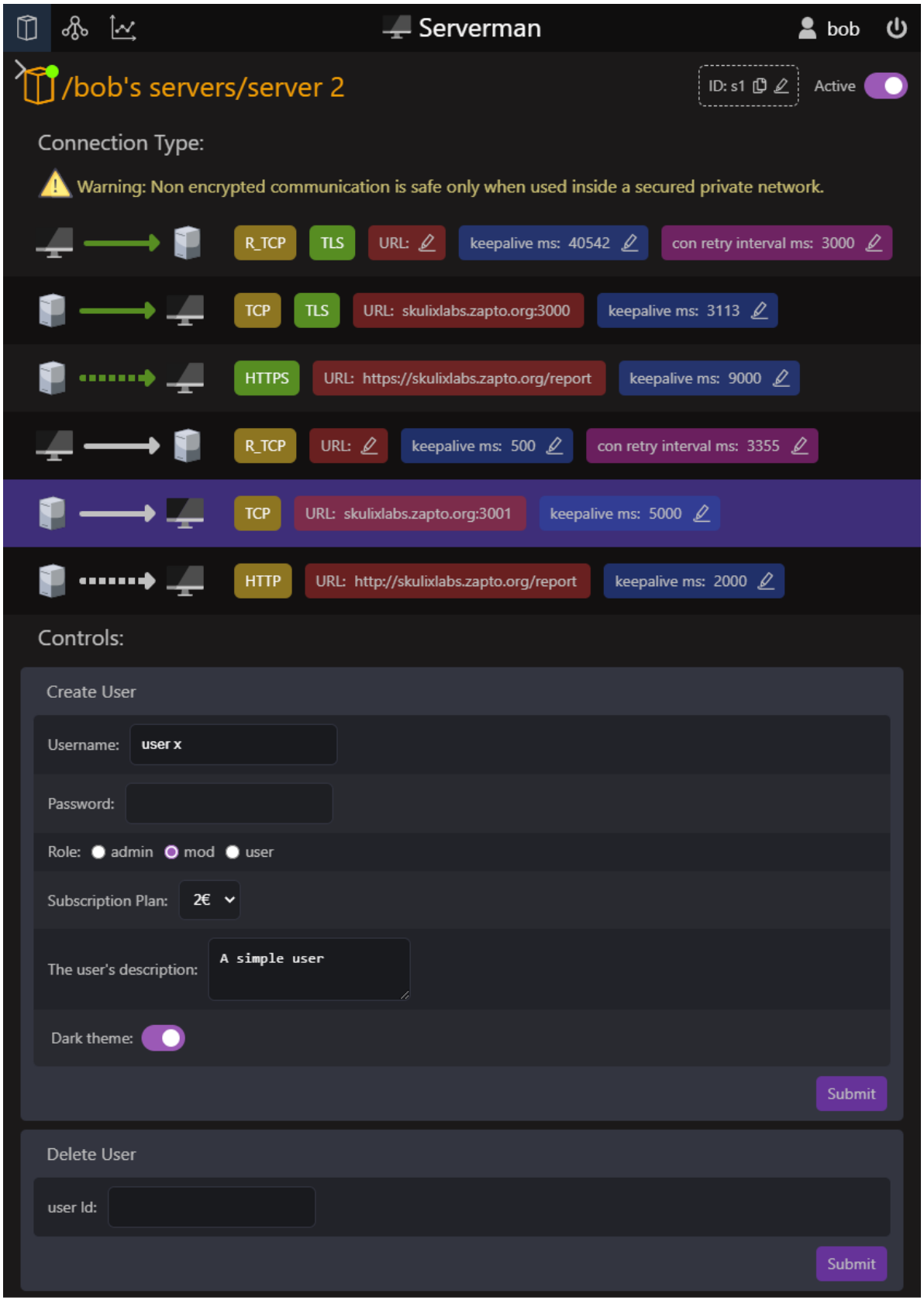
Εικόνα 2.4: Σύστημα αρχείων, επιλεγμένος φάκελος και δύο αρχεία.

Πιο συγκεκριμένα, στο πάνω μέρος του παραθύρου εμφανίζονται τα κουμπιά επεξεργασίας των οντοτήτων μαζί με το κουμπί διάσχισης προς τα επάνω (γονικό φάκελο) και το τρέχον μονοπάτι. Οι λειτουργίες των κουμπιών επεξεργασίας οντοτήτων είναι οι εξής: μετακίνηση προς τα επάνω, μετακίνηση προς τα κάτω, μετονομασία, μετακίνηση, διαγραφή, δημιουργία φακέλου και δημιουργία οντότητας. Το παράθυρο κατά την επεξεργασία δίνει στο χρήστη τη δυνατότητα να επιλέξει την οντότητα που θέλει να επεξεργαστεί. Για την επεξεργασία μιας οντότητάς θα πρέπει αυτή να «ανοίξει» πατώντας πάνω της. Επίσης, το παράθυρο έχει την δυνατότητα να κρυφτεί.

2.4 Εξυπηρετητές (Servers) 

Οι εξυπηρετητές αποτελούν οντότητες από τις οποίες αντλούνται αρχεία καταγραφής (logs), μετρικές και αποστέλλονται εντολές. Αφού δημιουργηθεί και «ανοίξει» η οντότητα «εξυπηρετητής», εμφανίζεται δεξιά του παραθύρου του συστήματος αρχείων η αναλυτική κατάσταση και οι πληροφορίες σύνδεσης του εξυπηρετητή. Επιπλέον, εμφανίζονται μαζί με το μοναδικό ID του και οι εντολές, εφόσον ο τρόπος σύνδεσης είναι τύπου TCP και υπάρχει ενεργή σύνδεση με το απομακρυσμένο άκρο.

Η σύνδεση της εφαρμογής με τον κάθε εξυπηρετητή, γίνεται με δύο τρόπους: α) σύνδεση της εφαρμογής στον εξυπηρετητή, β) σύνδεση του εξυπηρετητή στην εφαρμογή. Στην πρώτη περίπτωση, εισάγεται η IP ή το URL μαζί με την πόρτα σύνδεσης του εξυπηρετητή ενώ στην δεύτερη περίπτωση, παρέχεται URL σύνδεσης που θα χρησιμοποιήσει ο εξυπηρετητής για να αρχικοποιήσει ο ίδιος την σύνδεση με την εφαρμογή.



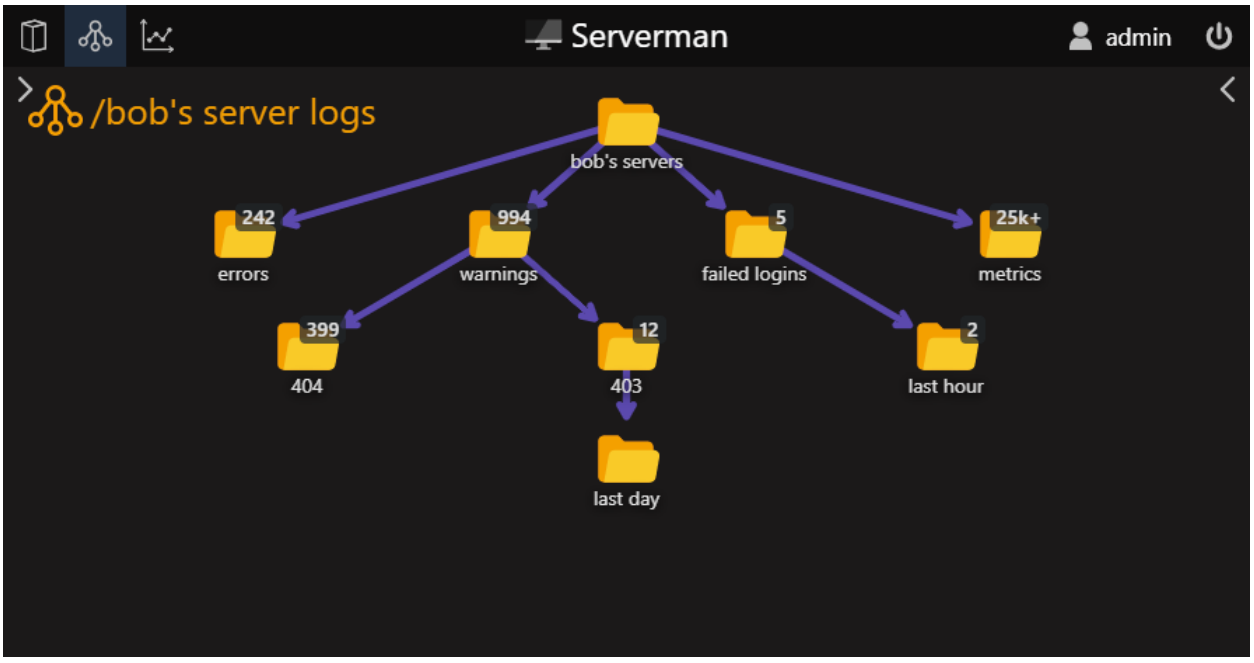
Εικόνα 2.5: Εξυπηρετητής και οι ρυθμίσεις του.

2.5 Δέντρα κατηγοριοποίησης (Categorization Trees)

Τα δέντρα κατηγοριοποίησης χαρακτηρίζονται από ένα γράφο που οι κόμβοι του γράφου αποτελούν φακέλους, οι οποίοι είναι συνδεδεμένοι μεταξύ τους με ιεραρχικό τρόπο και χρησιμοποιούνται στην κατηγοριοποίηση των αρχείων καταγραφής. Ο κάθε φάκελος του ΔΚ, διαχωρίζει τα αρχεία καταγραφής ανάλογα με τα φίλτρα (βλ. 1.6) που έχουν τεθεί στον κάθε φάκελο και τα δεδομένα που περιέχονται σε κάθε αρχείο καταγραφής.

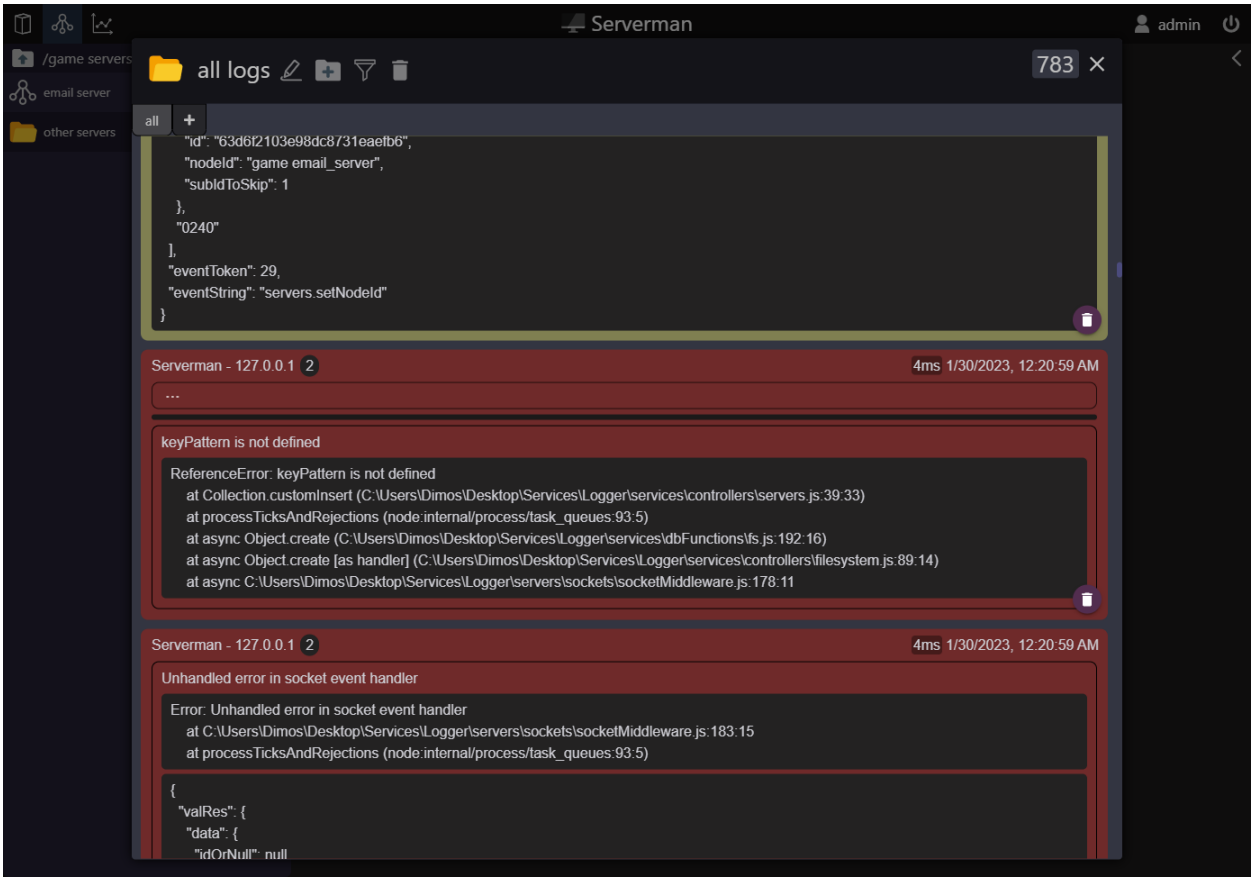
Όλα τα ΔΚ έχουν ένα αρχικό φάκελο στον οποίο μπορεί να τεθεί μια πηγή από την οποία θα αντληθούν τα αρχεία καταγραφής. Η πηγή μπορεί να είναι κενή ή φάκελος από άλλο ΔΚ. Στην περίπτωση που η πηγή είναι κενή, ο αρχικός φάκελος θα περιέχει τα αρχεία καταγραφής όλου του συστήματος, ενώ εάν είναι φάκελος άλλου ΔΚ θα περιέχει τα αρχεία καταγραφής εκείνου του φακέλου. Και στις δύο περιπτώσεις υπάρχει η δυνατότητα να τεθούν φίλτρα ώστε ο μητρικός φάκελος να περιέχει ένα υποσύνολο των αρχείων καταγραφής της επιλεγμένης πηγής.

Κάθε αρχείο καταγραφής δεν μπορεί να περιέχεται σε παραπάνω από ένα φάκελο του ΔΚ, για αυτό το λόγο κάθε φορά που ένα αρχείο καταγραφής πληροί τις προϋποθέσεις των φίλτρων ενός υποφακέλου, “μεταφέρεται” στον ίδιον τον υποφάκελο.



Εικόνα 2.6: ΔΚ με εννέα φακέλους.

Οι λειτουργίες που παρέχονται στην επεξεργασία των ΔΚ είναι οι εξής: α) προσθήκη φακέλου, β) μετονομασία φακέλου, γ) επεξεργασία επιλεγμένων φίλτρων δ) διαγραφή φακέλου, ε) προβολή αρχείων καταγραφής φακέλου και στ) δημιουργία/διαγραφή/προβολή κατηγοριοποίησης αρχείων καταγραφής φακέλου.



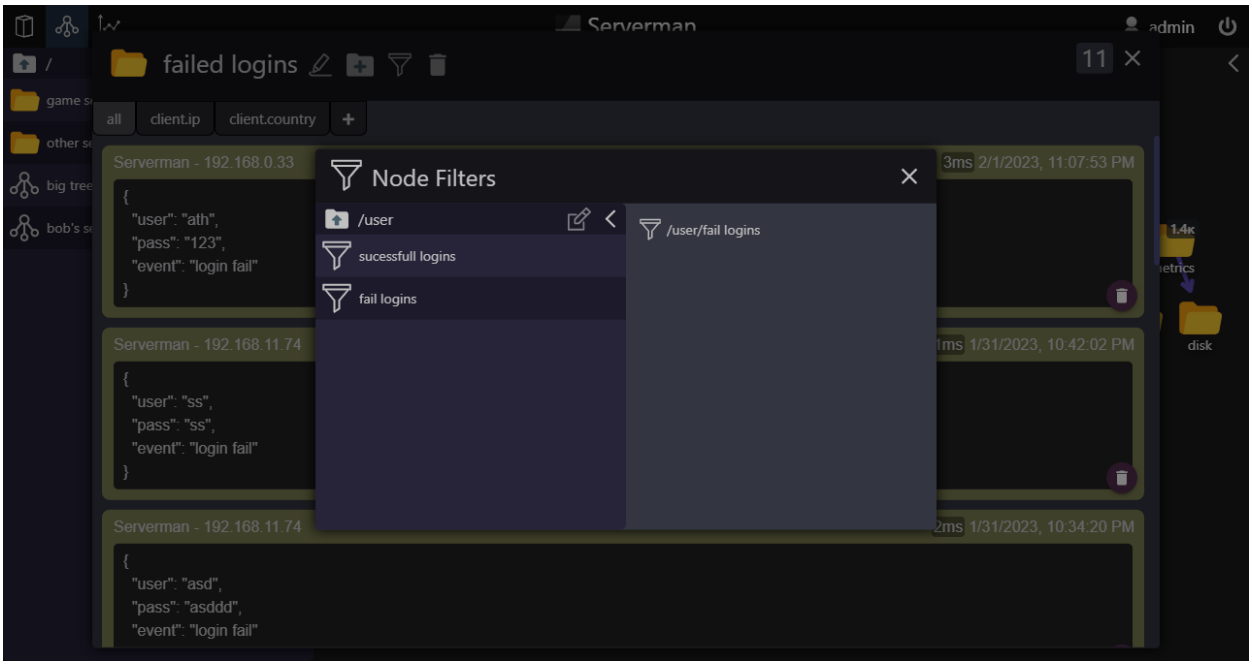
Εικόνα 2.7: Ανοιχτός φάκελος ΔΚ χωρίς φίλτρα.

2.6 Φίλτρα (Filters)

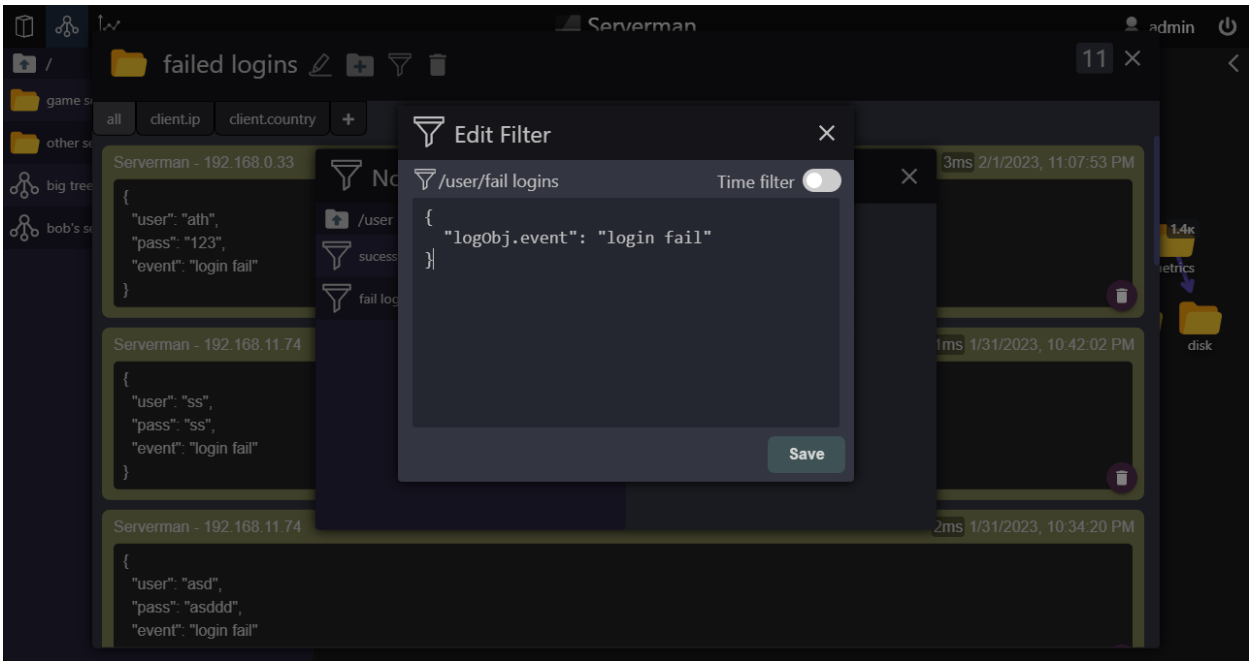
Τα φίλτρα είναι οντότητες οι οποίες είναι οργανωμένες σε σύστημα αρχείων, αλλά δεν υπάρχει ειδικό παράθυρο για αυτές, ενώ υπάρχουν στο παράθυρο των δέντρων κατηγοριοποίησης τα οποία αναλύθηκαν παραπάνω.

Τα φίλτρα κατηγοριοποίησης αποτελούν οντότητες οι οποίες περιέχουν πληροφορία η οποία χρησιμοποιείται για τον διαχωρισμό των αρχείων καταγραφής. Τα φίλτρα χωρίζονται σε δύο κατηγορίες: JSON και χρονικά. Τα JSON φίλτρα εμφανίζουν την πληροφορία σε μορφή JSON και η σύνταξή της είναι πανομοιότυπη με τους τελεστές

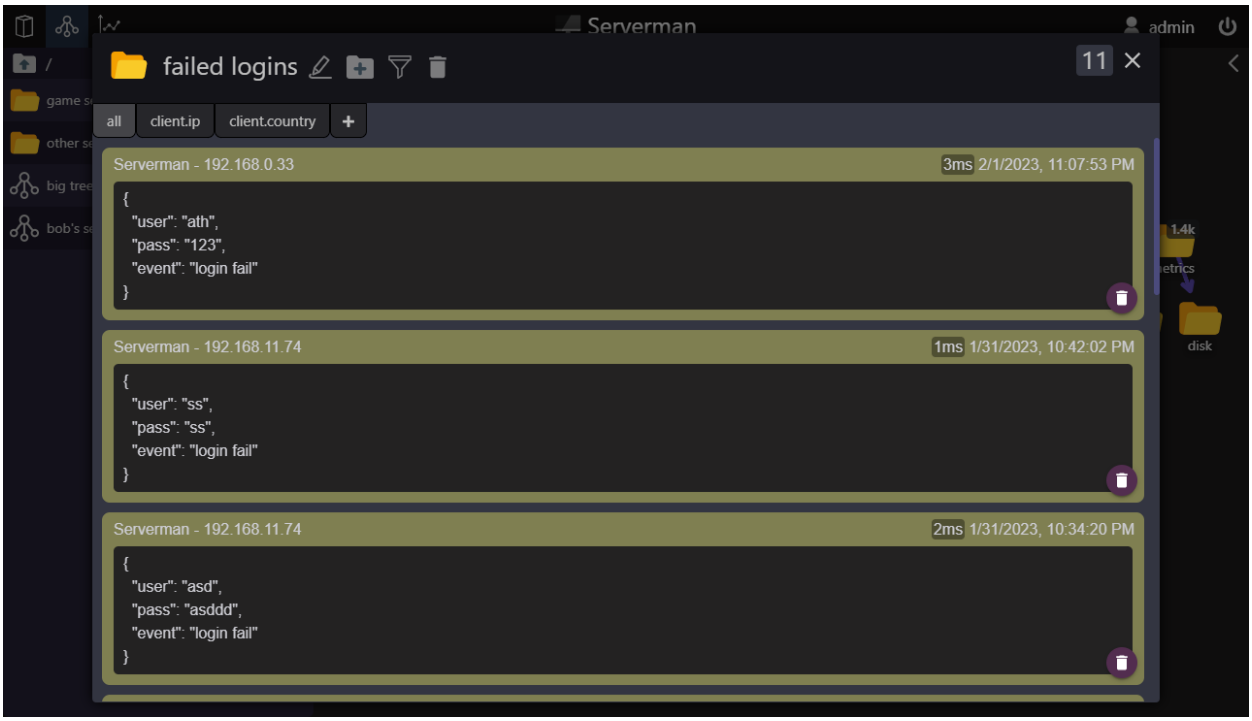
ερωτήματος της MongoDB. Στην περίπτωση των χρονικών φίλτρων, τα φίλτρα εμφανίζουν αριθμητικό ποσό ημερών, ωρών, λεπτών και δευτερολέπτων.



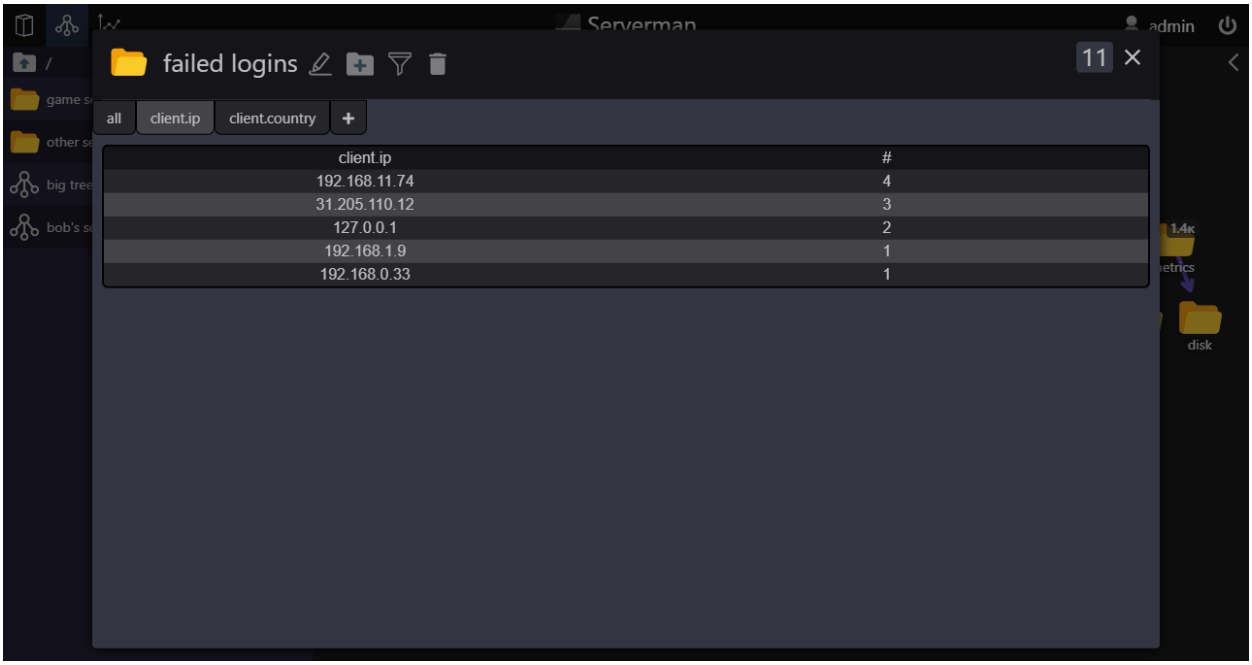
Εικόνα 2.8: Προσθήκη φίλτρου σε φάκελο.



Εικόνα 2.9: Επεξεργασία μη χρονικού φίλτρου.



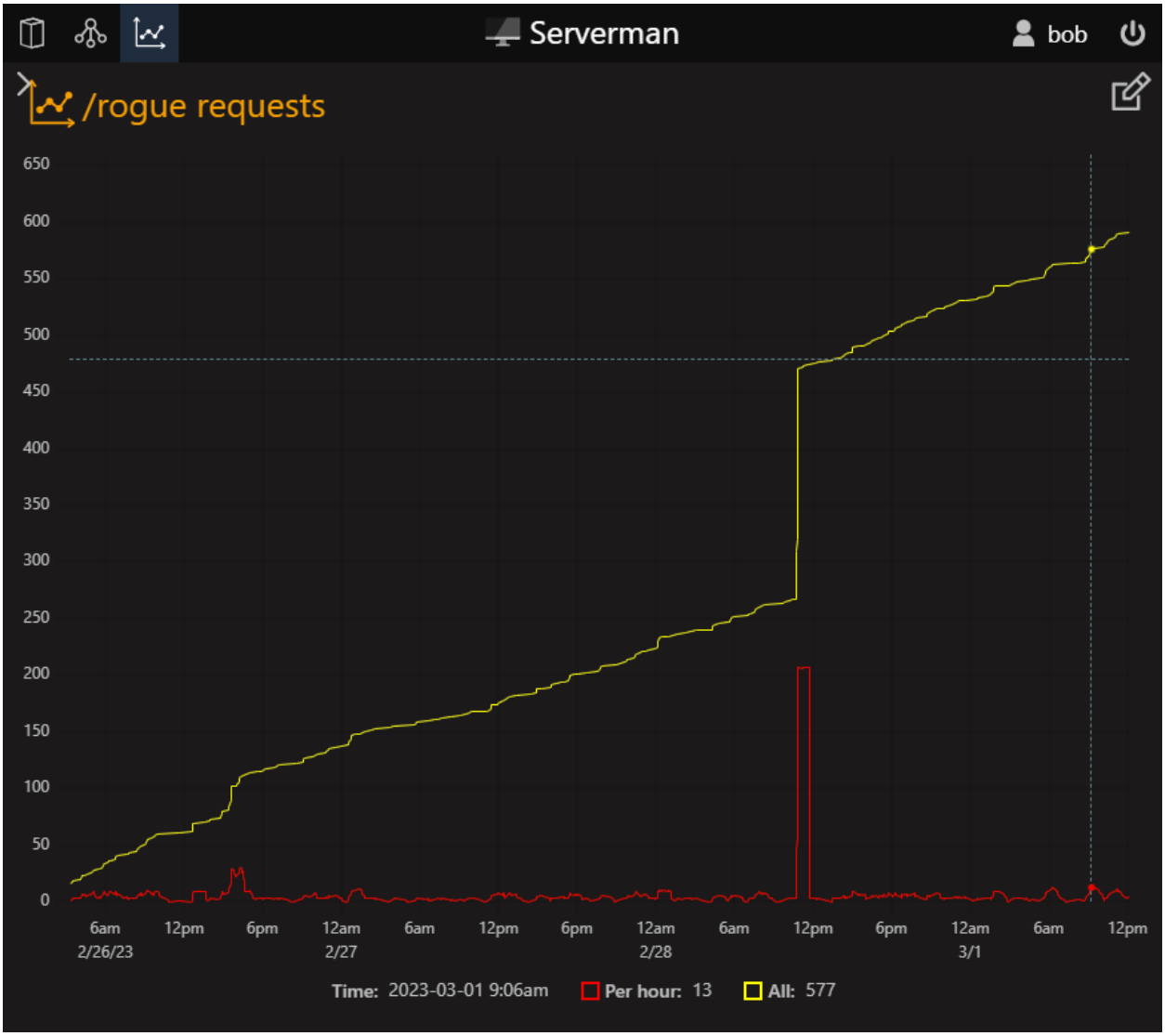
Εικόνα 2.10: Φάκελος που εμφανίζει όλες τις αποτυχημένες προσπάθειες σύνδεσης.



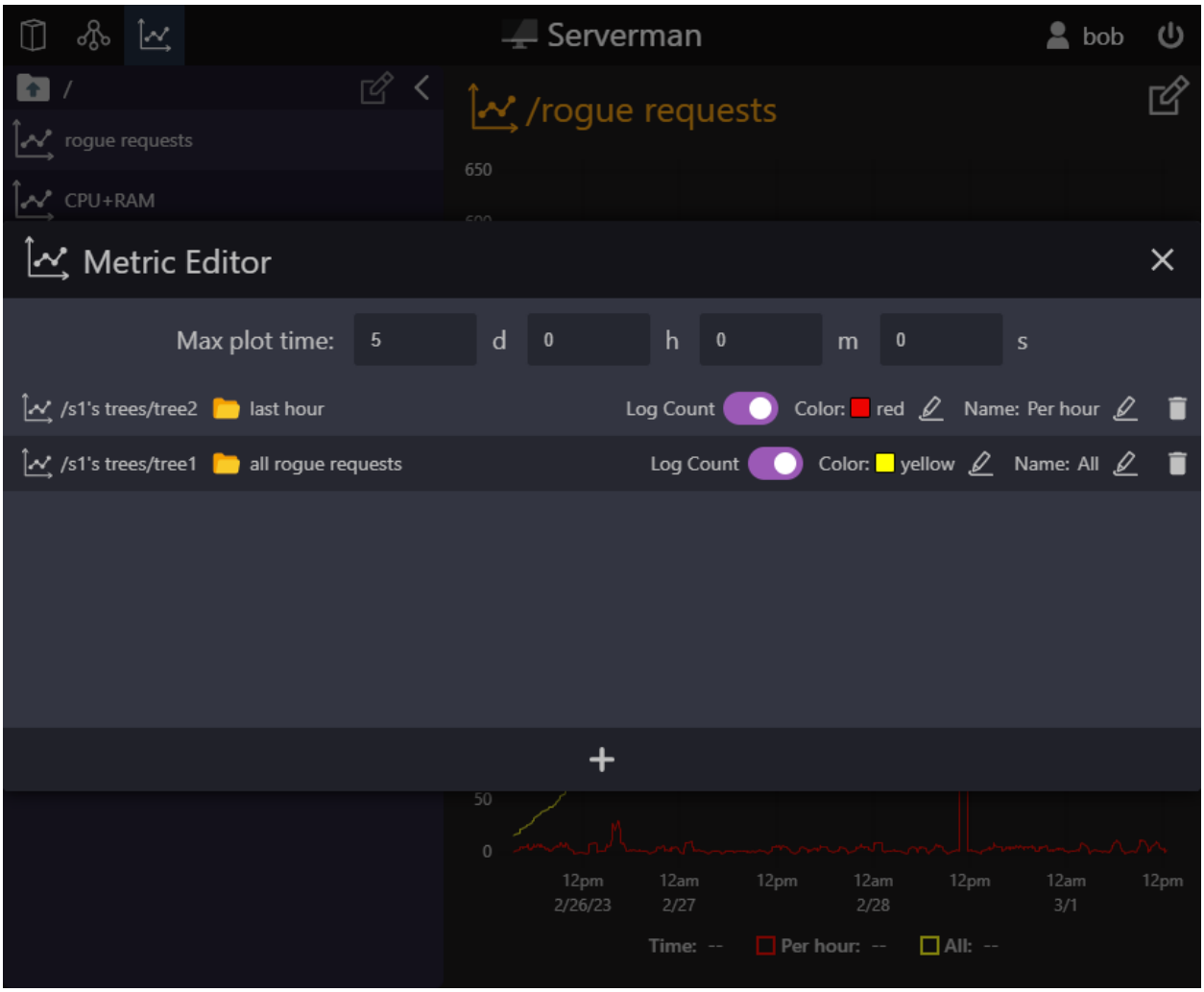
Εικόνα 2.11: Φάκελος που εμφανίζει τον αριθμό των αποτυχημένων προσπαθειών σύνδεσης, κατηγοριοποιημένες ανά διεύθυνση IP.

2.7 Μετρικές (Metrics)

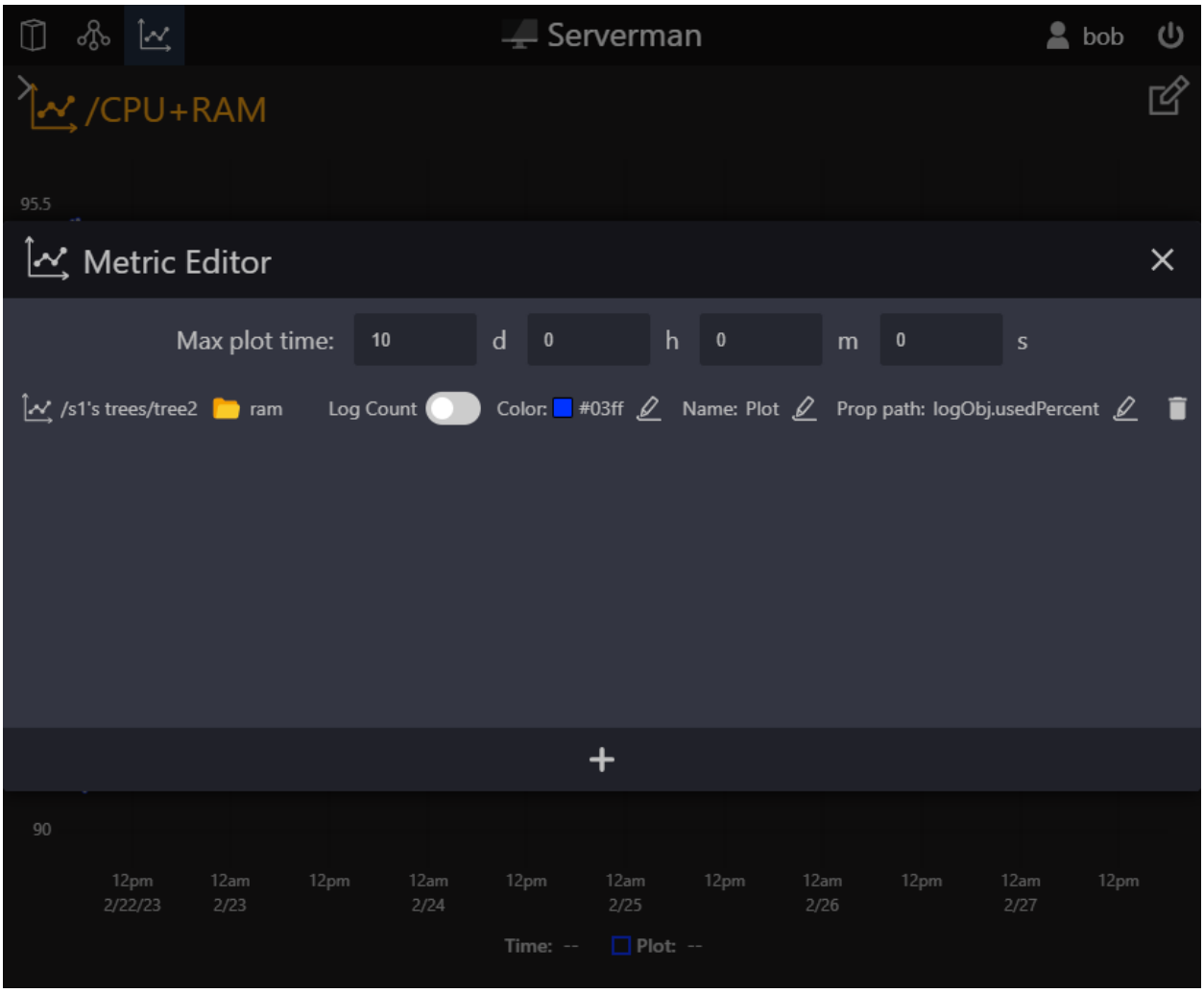
Οι μετρικές αποτελούν οντότητες οι οποίες παρουσιάζουν σε γράφημα πληροφορίες όπως επιλεγμένες τιμές δεδομένων στο χρόνο ή αριθμό αρχείων καταγραφής φακέλου στο χρόνο.



Εικόνα 2.12: Μετρική η οποία εμφανίζει τον αριθμό όλων, και ανά ώρα, ύποπτων αιτημάτων που δέχτηκε ο εξυπηρετητής τις τελευταίες 5 ημέρες.



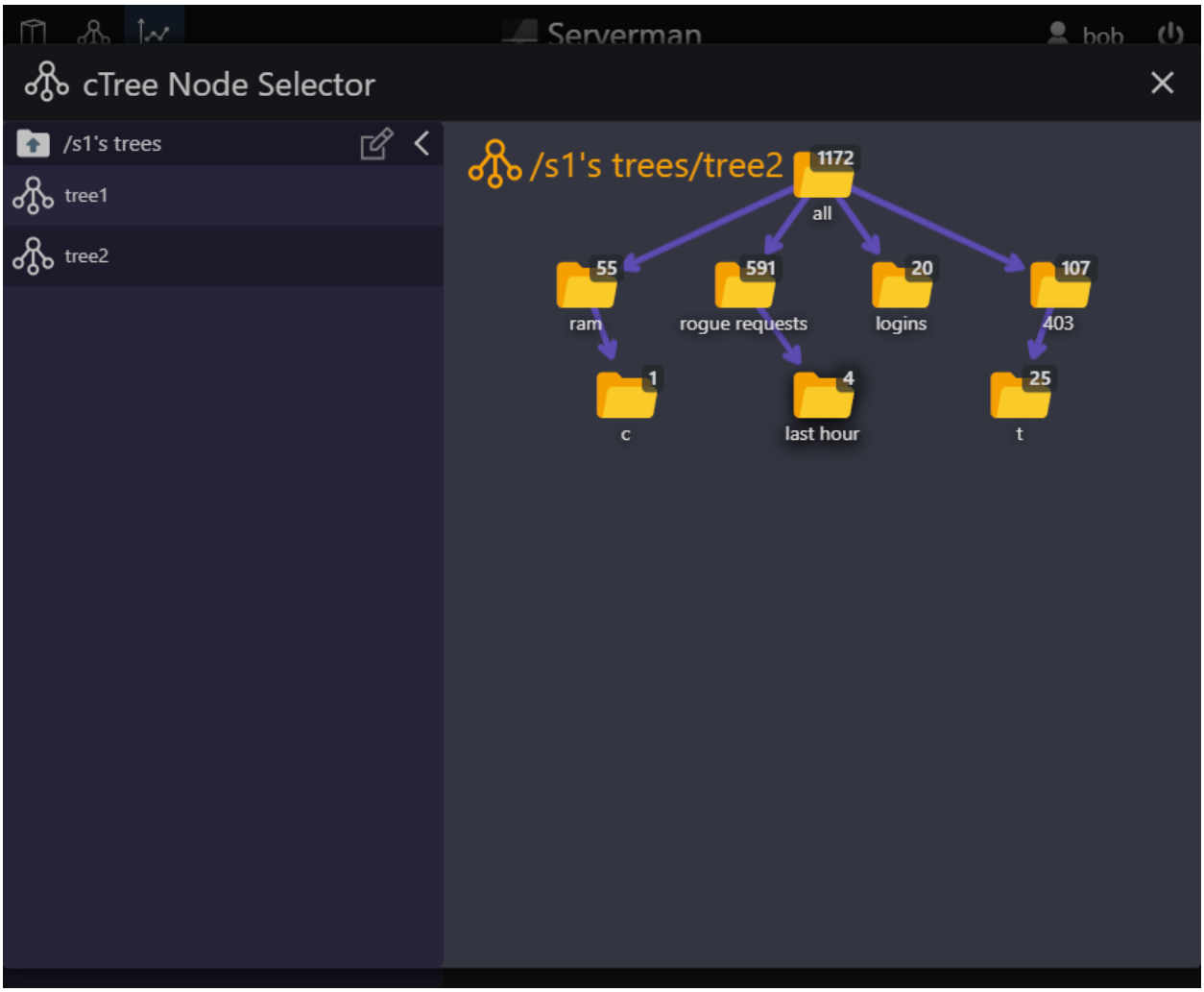
Εικόνα 2.13: Επεξεργασία μετρικών εικόνας.



Εικόνα 2.14: Επεξεργασία μετρικών, παράδειγμα μετρικής με σχεδίαση τιμών.

Στις παραπάνω εικόνες παρατηρούμε το παράθυρο επεξεργασίας μετρικών, στο οποίο μπορεί να επεξεργαστεί ο χρόνος σχεδίασης, να προστεθεί μια νέα μετρική, να επιλεχθεί αν θα σχεδιαστεί το πλήθος των αρχείων καταγραφής ή η τιμή αρχείων καταγραφής, κ.α.

Πατώντας το κουμπί «+» στο κάτω μέρος του παραθύρου, εμφανίζεται ο επιλογέας φακέλου από ΔΚ. Αμέσως μετά την επιλογή φακέλου, προσθέτετε νέα μετρική.



Εικόνα 2.15: Επιλογέας φακέλου από ΔΚ.

ΚΕΦΑΛΑΙΟ 3

BACKEND

3.1 Τεχνολογίες Ανάπτυξης

Το λογισμικό του εξυπηρετητή έχει αναπτυχθεί στην γλώσσα προγραμματισμού JavaScript και εκτελείται μέσω του περιβάλλοντος εκτέλεσης Node.js. Η γλώσσα προγραμματισμού JavaScript αρχικά έκανε την εμφάνισή της το 1995 οπού αποτελεί ακόμα και τώρα μια από τις βασικότερες τεχνολογίες ανάπτυξης webapps και δια δραστικών ιστοσελίδων. Η JavaScript μέχρι το 2009 μπορούσε να εκτελεστεί μόνο σε περιβάλλον browser (Severance C., 2012). Αυτό ήρθε να αλλάξει το Node.js επιτρέποντας στους προγραμματιστές να αναπτύξουν λογισμικό το οποίο έχει πρόσβαση στις διεπαφές του λειτουργικού συστήματος (Heller, 2022).

Οι λόγοι για τους οποίους επιλέχθηκε η JavaScript σε συνδυασμό με το περιβάλλον Node.js, είναι οι παρακάτω:

- Δημοτικότητα: Η JavaScript είναι μια από τις πιο δημοφιλείς γλώσσες προγραμματισμού στον κόσμο και το Node.js είναι ένα από τα πιο δημοφιλή περιβάλλοντα ανάπτυξης εξυπηρετητών. Αυτό σημαίνει ότι υπάρχει μια μεγάλη κοινότητα προγραμματιστών που εργάζονται με JavaScript και Node.js, γεγονός που διευκολύνει την εύρεση βοήθειας σε προβλήματα που προκύπτουν κατά την ανάπτυξη συστημάτων.
- Ευελιξία: Η JavaScript μπορεί να χρησιμοποιηθεί τόσο για ανάπτυξη frontend όσο και για backend. Αυτό σημαίνει ότι μπορεί να χρησιμοποιηθεί η ίδια γλώσσα για προγραμματισμό τόσο από την πλευρά του πελάτη όσο και από την πλευρά του εξυπηρετητή, γεγονός που κάνει την ανάπτυξη ταχύτερη και πιο αποτελεσματική.
- Γρήγορη ανάπτυξη: Για του παραπάνω λόγους, η JavaScript και το Node.js διευκολύνουν τη γρήγορη δημιουργία πρωτοτύπων.
- Επεκτασιμότητα: Το Node.js έχει σχεδιαστεί για να είναι επεκτάσιμο, πράγμα που σημαίνει ότι με την κατάλληλή αρχιτεκτονική, μπορεί να χειριστεί μεγάλο πλήθος αιτημάτων.

Ο εξυπηρετητής χρησιμοποιεί το framework Express.js το οποίο παρέχει εργαλεία και βιβλιοθήκες για την πιο αποτελεσματική ανάπτυξη ενός εξυπηρετητή και της αρχιτεκτονικής του. Η ευκολία στη χρήση του, η μινιμαλιστική φύση του και η μεγάλη κοινότητα στην οποία μπορεί να απευθυνθεί κάποιος κατά την ανάπτυξη ενός συστήματος εξυπηρετητή καθιστούν το Express.js ιδανική επιλογή για πολλούς προγραμματιστές.

Σημαντική τεχνολογία ανάπτυξης για το real-time κομμάτι της εφαρμογής, αποτελούν τα WebSockets. Επιλέχθηκε η χρήση των τους για την επικοινωνία μεταξύ του backend και του frontend επειδή επιτρέπουν αμφίδρομη επικοινωνία σε σύγκριση με το HTTP το οποίο είναι μονής κατεύθυνσης και απαιτεί νέα σύνδεση για κάθε αίτημα. Οι browser υποστηρίζουν WebSockets χωρίς κάποια επιπρόσθετη βιβλιοθήκη σε αντίθεση με το backend. Η βιβλιοθήκη που χρησιμοποιήθηκε στο backend είναι το ws το οποίο αποτελεί υλοποίηση του WebSocket server (RFC-6455).

Η βάση δεδομένων που έχει χρησιμοποιηθεί είναι η MongoDB η οποία είναι μια βάση δεδομένων τύπου NoSQL και κάνει χρήση εγγράφων τύπου JSON. Η βάση επιτρέπει μεγαλύτερη ευελιξία όσον αφορά τη μοντελοποίηση δεδομένων σε αντίθεση με σχεσιακές βάσεις δεδομένων και παρέχει λειτουργίες όπως η graphLookup, η οποία διευκόλυνε την ανάπτυξη του συστήματος αρχείων και τα change streams τα οποία χρησιμοποιήθηκαν για την παρακολούθηση των δεδομένων του συστήματος σε πραγματικό χρόνο. Επίσης επιτρέπει εύκολη οριζόντια κλιμάκωση κρατώντας τις ACID εγγυήσεις κάτι το οποίο καθίσταται αναγκαίο λόγο τις πληθώρας των αρχείων που θα χρειαστεί να διαχειριστεί.

3.2 Μοτίβα σχεδίασης λογισμικού και αρχιτεκτονική

Τα μοτίβα σχεδίασης λογισμικού χαρακτηρίζουν τον τρόπο με τον οποίο σχεδιάζουμε και αναπτύσσουμε ένα σύστημα. Είναι ένα πρότυπο για τον τρόπο επίλυσης ενός προβλήματος που μπορεί να χρησιμοποιηθεί σε πολλές διαφορετικές καταστάσεις. Η χρήση μοτίβων σχεδίασης έχει δείξει ότι βελτιώνει την παραγωγικότητα του

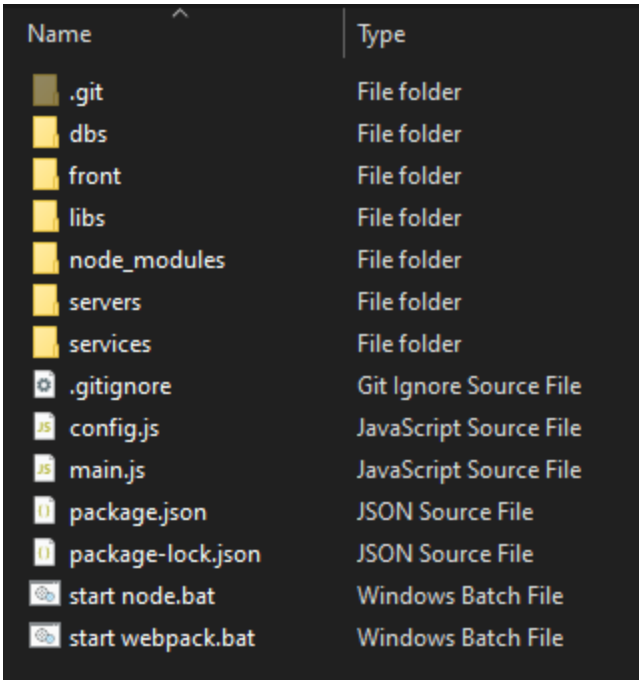
προγραμματιστή, αλλά και την ποιότητα του κώδικα, κάτι το οποίο καταλήγει σε μικρότερο χρόνο ανάπτυξης και συντήρησης του συστήματος (IEEE, 2002).

Στη μηχανική λογισμικού, μια μονολιθική εφαρμογή περιγράφει μια εφαρμογή λογισμικού η οποία έχει σχεδιαστεί και αναπτυχθεί ως μία ενιαία υπηρεσία, ανεξάρτητη από άλλες υπηρεσίες. Αυτού του είδους οι εφαρμογές είναι βολικές στην αρχή της ζωής ενός έργου εξαιτίας της ευκολίας διαχείρισης κώδικα.

Η αρχιτεκτονική μικροϋπηρεσιών βασίζεται σε μια σειρά από ανεξάρτητα αναπτυσσόμενες υπηρεσίες, οι οποίες έχουν δικό τους business logic και βάση δεδομένων. Η κάθε μικροϋπηρεσία έχει ένα συγκεκριμένο στόχο. Έτσι η ενημέρωση, η δοκιμή και η ανάπτυξη πραγματοποιείται ξεχωριστά σε κάθε υπηρεσία. Οι μικροϋπηρεσίες δεν μειώνουν την πολυπλοκότητα, αλλά καθιστούν οποιαδήποτε πολυπλοκότητα ορατή και πιο διαχειρίσιμη, διαχωρίζοντας τις εργασίες σε μικρότερες διαδικασίες που λειτουργούν ανεξάρτητα η μία από την άλλη και συμβάλλουν στη συνολική λειτουργία του συστήματος. (CHANDLER, n.d.)

Η αρχιτεκτονική του συστήματος που επιλέχθηκε είναι η μονολιθική και το σύστημα δεν χρειάζεται επιπρόσθετες υπηρεσίες για να δουλέψει. Η επιλογή έγινε για την πιο γρήγορη ανάπτυξη του συστήματος και της μικρής κλίμακάς του. Για την ευκολία ανάπτυξης, αναγνωσιμότητας και επαναχρησιμοποίησης, έχει γίνει ο διαχωρισμός αρμοδιοτήτων και ο κώδικας είναι modular.

Παρακάτω παρατηρούμε τα αρχεία του συστήματος και των διαχωρισμό των αρμοδιοτήτων:



Εικόνα 3.1: Αρχεία συστήματος

- dbs – connector, αρχικοποιητής βάσης δεδομένων (collection/tables, indexes)
- front – views, loaders, assets, κώδικας frontend.
- libs – βιβλιοθήκες
- servers – HTTP, WebSocket, TCP servers, connection handlers, REST routes
- services – system logic, controllers, database functions

Το σύστημα χρησιμοποιεί μοντέλο αποθετηρίου (repository pattern). Το μοντέλο αποθετηρίου χρησιμοποιείται για τη δημιουργία ενός επιπέδου αφαίρεσης μεταξύ του επιπέδου πρόσβασης δεδομένων και του business logic. Χρησιμοποιώντας αυτά τα μοτίβα, γίνεται ευκολότερη η διαχείριση συστημάτων τα οποία περιέχουν αποθήκευση δεδομένων και μπορούμε να παρέχουμε αυτοματοποιημένο unit testing και test-driven development (TDD) (Mukesh, 2019).

Για την επικοινωνία των διαφόρων modules και συνδεμένων WebSocket μεταξύ τους, έχει χρησιμοποιηθεί το μοτίβο publish-subscribe. Στην αρχιτεκτονική λογισμικού, το μοτίβο publish-subscribe είναι ένα μοτίβο ανταλλαγής μηνυμάτων όπου οι subscribers

κάνουν εγγραφή σε συμβάντα (events) που τους ενδιαφέρουν. Με αυτό τον τρόπο ειδοποιούνται ασύγχρονα από τους publishers όταν αυτοί εκδώσουν κάποιο συμβάν στο οποίο έχουν δηλώσει ενδιαφέρον (Patrick T Eugster, 2003).

Παρακάτω παρατηρούμε την συνάρτηση η οποία πραγματοποιεί την εγγραφή ενός WebSocket σε ένα συμβάν που το ενδιαφέρει.

```
const subscriptions = {};
const onEventSubscriptions = {};

services.sub = (event, ws, dataForWatcher) => {
  if (!isString(event)) throw new Error('Event must be a string.');

  let newEvent = false;
  if (!subscriptions[event]) {
    newEvent = true;
    subscriptions[event] = new Set();
  }

  if (!ws.subs.has(event)) {
    subscriptions[event].add(ws);
    ws.subs.add(event);
  }

  let eventSubId = undefined;
  if (event.indexOf(':') > -1) {  // if event is classful
    const tmp = event.split(':');
    event = tmp[0];
    eventSubId = tmp[1];
    if (event == '' || eventSubId == '') throw new Error('Invalid event');
  }

  if (onEventSubscriptions[event]) {
    for (let f of onEventSubscriptions[event])
      f(newEvent, eventSubId, dataForWatcher);
  }
};
```

Απόσπασμα Κώδικα 3.1: Υλοποίηση εγγραφών συστήματος

Η συνάρτηση έχει τις επιπρόσθετες λειτουργίες χειρισμού κλάσεων συμβάντων και ενημέρωση των subscribers οι οποίοι έχουν εγγραφτεί στο συμβάν που ενημερώνει τους

subscribers όταν πραγματοποιηθεί καινούργια εγγραφή σε συγκεκριμένη κλάση συμβάντος. Οι κλάσεις συμβάντων αποτελούν επιπρόσθετη λειτουργία στο μοντέλο publisher-subscriber, η οποία διαχωρίζει τα συμβάντα σε classful (με κλάση) και μη. Εν συνεχεία στην περίπτωση που το συμβάν είναι classful, διαχωρίζετε η κλάση από το συμβάν και καλούνται όλοι οι subscribers οι οποίοι έχουν εγγραφτεί για να ενημερώνονται όταν δημιουργείται μία καινούργια εγγραφή στη συγκεκριμένη κλάση συμβάντων.

Παρακάτω παρατηρούμε την συνάρτηση η οποία πραγματοποιεί την ενημέρωση (publish) στα WebSocket/εσωτερικές συναρτήσεις τα/οι οποία/ες έχουν εγγραφτεί στα κατάλληλα events.

```
  const WS_OPEN = require('ws').OPEN;
  const subscriptions = services.getSubscriptions();

  services.broadcast = function(event, level, { eventSubId, data, clientState,
userId, ws, wsToSkip, subIdToSkip }) {

    if (!wsToSkip && subIdToSkip !== undefined)
      throw new Error('subIdToSkip is set but wsToSkip is missing');
    let tokenizedEvent = global.metadata.tokens[event];
    if (tokenizedEvent === undefined)
      throw new Error('Event \'' + event + '\' not tokenized.');
    if (level == 'subs' && eventSubId !== undefined)
      tokenizedEvent += ':' + eventSubId;
    let body = [tokenizedEvent];
    if (data !== undefined) body.push(data);
    const serializedBody = JSON.stringify(body);

    switch (level) {
      case 'socket':
        ws.readyState == WS_OPEN && ws.send(serializedBody);
        break;
      case 'session':
        for (let i=0; i<clientState.wsocks.length; i++) {
          clientState.wsocks[i].readyState == WS_OPEN && wsToSkip !==
clientState.wsocks[i] && clientState.wsocks[i].send(serializedBody);
        }
        break;
      case 'user':
        const user = services.users.get(userId);
        if (!user) return;
```

```
        for (let sessId of user.sessIds) {
          let cS = services.clientStates.get(sessId);
          for (let i=0; i<cS.wsocks.length; i++) {
            cS.wsocks[i].readyState == WS_OPEN && wsToSkip !== cS.wsocks[i] &&
cS.wsocks[i].send(serializedBody);
          }
        }
        break;
      case 'subs':
        if (eventSubId !== undefined) event += ':' + eventSubId;
        services.localPub(event, data);
        if (!subscriptions[event]) return;
        subscriptions[event].forEach(ws => {
          if (ws.readyState == WS_OPEN) {
            if (wsToSkip !== ws) ws.send(serializedBody);
            else if (subIdToSkip !== undefined) { // if subIdToSkip is undefined,
ws will be skipped
              body.push(subIdToSkip);
              ws.send(JSON.stringify(body));
            }
          }
        });
        break;
      case 'global':
        services.wss.clients.forEach(ws => ws.readyState == WS_OPEN && wsToSkip
!== ws && ws.send(serializedBody));
        break;
      default:
        throw new Error('Unknown broadcasting level.');
    }
  };
```

Απόσπασμα Κώδικα 3.2: Μέθοδος broadcast/publish συστήματος

Η συνάρτηση έχει επιπρόσθετη λειτουργία χειρισμού αποστολής μνημάτων σε διαφορετικά επίπεδα ενημέρωσης WebSocket. Τα επίπεδα ενημέρωσης WebSocket, είναι τα ακόλουθα:

- socket - αποστολή μηνύματος σε συγκεκριμένο WebSocket
- session – αποστολή μηνύματος σε όλα τα WebSocket συγκεκριμένου session. Συνήθως ο browser έχει ένα session ανά domain, οπότε στο συγκεκριμένο επίπεδο το μήνυμα θα σταλεί σε όλα τα WebSocket που έχουν αρχικοποιηθεί

από τον ίδιο browser ή αλλιώς, σε κάθε παράθυρο (tab) στο οποίο υπάρχει ενεργή σύνδεση WebSocket με το σύστημα.

- user – αποστολή μηνύματος σε όλα τα ενεργά sessions συγκεκριμένου χρήστη.
- subs – αποστολή μηνύματος σε όλα τα/όλες τις WebSocket/συναρτήσεις συστήματος τα οποία/οι οποίες έχουν εγγραφτεί σε συγκεκριμένο event.
- global – αποστολή μηνύματος σε όλα τα ενεργά WebSocket.

Επιπρόσθετα παραδείγματα σχετικά με την κλήση του publisher (συνάρτηση broadcast) υπάρχει στην ενότητα 3.5.

3.3 Βάση δεδομένων

3.3.1 Δομή βάσης

Μια διαφορά της MongoDB σε σχέση με σχεσιακές βάσεις δεδομένων είναι ότι στην MongoDB δεν υπάρχουν πίνακες με εγγραφές, αλλά υπάρχουν συλλογές εγγράφων. Τα έγγραφα εσωτερικά αποτελούνται από αρχεία BSON αλλά κατά την χρήση της βάσης, τα αρχεία χειρίζονται σαν απλά αντικείμενα JSON.

Οι συλλογές της βάσης και η δομή των αρχείων τους:

- users – Λογαριασμοί χρηστών εφαρμογής

```
_id: ObjectId('638f03cfd551a66959b6131b')
user: "bob"
pass: "$2b$10$CkHhZ47.qkZHWxKhgn5Jgu7J0zHZKgCGyccPWWJutvfpcy5d22Fcu"
```

- blacklist

```
_id: ObjectId('626fef60aa119e7abed3b990')
ip: "192.168.68.153"
ts: 2022-09-02T14:49:04.024+00:00
reason: "Too many failed logins"
data: "["username...","password..."]"
```

- sessions

```
_id: "fD_myiVf6bqAJ_C9iVHvaQCSQsCUo4j-"
expires: 2023-11-03T22:25:17.755+00:00
session: Object
  cookie: Object
  userId: "62d72fbbf809dce1634527a7"
  ip: "5.203.145.121"
  country: "GR"
  city: "Athens"
  browserName: "Chrome"
  browserVersion: "107.0.0.0"
  osName: "Android"
  osVersion: "10"
  startConTs: 1667514692674
  endConTs: 1667514714899
```

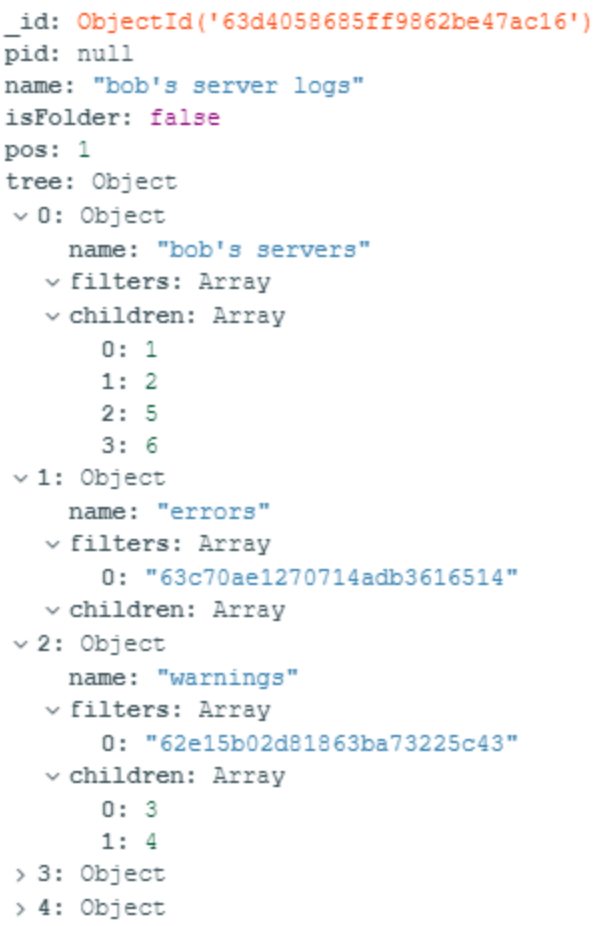

- logs


```
_id: ObjectId('63df2060ba73b3dea3f4c91f')
nodeId: "Serverman"
mts: 2023-02-05T03:20:00.664+00:00
cts: 2023-02-05T03:20:00.663+00:00
logObj: Array
  0: Object
    stack: "Error: Not Found
                at C:\Users\Dimos\Desktop\Services\Dependencies\m…"
    message: "Not Found"
    level: "warn"
  1: Object
    status: 404
logIndex: 0
extraLogObjects: Array
client: Object
  ip: "5.203.131.35"
  sid: "X10rE0E2s8_U0_8IVEq-71I6Eet4-jPt"
  uid: "62c83d8e4d1452e10fb994ab"
  method: "GET"
  url: "/randomURL"
  headers: Object
  country: "GR"
  city: "Athens"
  browser: Object
  os: Object
```

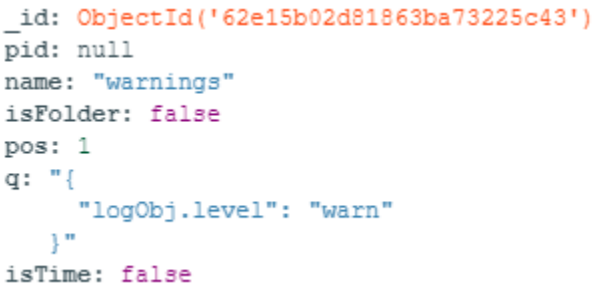

- servers

```
_id: ObjectId('63e069637b3487cf65376bb2')
pid: ObjectId('63e069597b3487cf65376baf')
name: "server 1"
isFolder: false
pos: 1
active: false
ct: "RTCP"
connectionInfo: Object
  RTCP_TLS: Object
      keepaliveMS: 3000
      retryMS: 3000
  TCP_TLS: Object
      keepaliveMS: 3001
  HTTPS: Object
      keepaliveMS: 9000
  RTCP: Object
      keepaliveMS: 3002
      retryMS: 3002
      url: "myserverurl.com:50505"
  TCP: Object
      keepaliveMS: 3003
  HTTP: Object
      keepaliveMS: 9000
nodeId: "5UCOhq2JlosPTcZ"
```

- cTrees



- filters



- metrics

Οι συλλογές οντοτήτων (servers, cTrees, filters, metrics) μπορούν επίσης να περιέχουν και φακέλους:

```
_id: ObjectId('63dd5f0014ad9fd262c6d7b9')
pid: null
name: "bob's servers"
isFolder: true
pos: 9
```

```
_id: ObjectId('63dd5f9b14ad9fd262c6d7bd')
pid: ObjectId('63dd5f0014ad9fd262c6d7b9')
name: "other servers"
isFolder: true
pos: 11
```

Όλα τα αρχεία των συλλογών οντοτήτων, περιέχουν το πεδίο «pid» και είναι δείκτης στον γονικό φάκελο. Στην περίπτωση που η τιμή του είναι «null», το αρχείο βρίσκετε στον αρχικό φάκελο (/) του συστήματος αρχείων.

Παρατηρούμε πως οι συλλογές οι οποίες είναι υπεύθυνες για τις οντότητες του συστήματος (servers, cTrees, filters, metrics), αντιπροσωπεύουν στο σύνολό τους το σύστημα αρχείων. Κάθε έγγραφο αυτών των συλλογών, είναι ένα αρχείο (οντότητα ή φάκελος) του.

3.3.2 Ευρετήριο (Indexes)

Όλες οι συλλογές εγγράφων έχουν από προεπιλογή την παράμετρο «_id» η οποία βρίσκεται πάντα στο ευρετήριο. Οι επιπρόσθετες παράμετροι για κάθε συλλογή οι οποίες έχουν προστεθεί στο ευρετήριο είναι:

Συλλογή	Παράμετροι
blacklist	• ip • date
users	• user (unique)
servers	• name + pid + isFolder (unique)

	• nodeId (unique, sparse)
cat_trees	• name + pid + isFolder (unique)
metrics	• name + pid + isFolder (unique)

3.3.3 Αρχικοποίηση

Η αρχικοποίηση της βάσης πραγματοποιείται με την βοήθεια κώδικα αλλά πριν την χρήση του, η βάση είναι απαραίτητο να λειτουργεί ως «replica set». Η λειτουργία replica set επιτρέπει την χρησιμοποίηση change streams τα οποία θα αναλυθούν στο επόμενο κεφάλαιο. Η μετατροπή της βάσης πραγματοποιείται με τις παρακάτω εντολές:

1. Σταματάμε την MongoDB στην περίπτωση που λειτουργεί.

2. Προσθέτουμε στο αρχείο mongod.cfg το παρακάτω:

```
replication:
  replSetName: rs0
```

3. Κάνουμε εκκίνηση της MongoDB με την παράμετρο replSet όπως παρακάτω:

```
mongod --replSet rs0
```

4. Εκκινούμε το MongoDB Shell και εκτελούμε την παρακάτω εντολή:

```
> rs.initiate()
```

Ακολουθούν αποσπάσματα του κώδικα και η επεξήγησή τους.

```
  const collections = {
    blacklist: [
      [ {ip: 1}, {name: 'ip_index'} ],
      [ {date: 1}, {name: 'date_index'} ]
    ],
    users: [
      [ {user: 1}, {name: 'unique_user_index', unique: true} ]
    ],
    servers: [
      [ {name: 1, pid: 1, isFolder:1}, {name: 'unique_name_in_folder_index',
unique: true} ],
      [ {nodeId: 1}, {name: 'unique_node_id_index', unique: true, sparse: true} ]
    ],
```

```
    cat_trees: [
      [ {name: 1, pid: 1, isFolder:1}, {name: 'unique_name_in_folder_index',
unique: true} ]
    ],
    metrics: [
      [ {name: 1, pid: 1, isFolder:1}, {name: 'unique_name_in_folder_index',
unique: true} ]
    ],
    logs: []
  };
```

Απόσπασμα Κώδικα 3.3: Δήλωση συλλογών και ευρετηρίων τους για δημιουργία

```
  for (let c in collections) {
    if (!await db.listCollections({name: c}).next()) {
      await db.createCollection(c);
      print('- Creating collection ' + c, 'magenta', 'bg-green');
    } else {
      print('- Found collection ' + c, 'black', 'bg-green');
    }
    const collection = db.collection(c);
    const existingIndexes = await collection.indexes();
    if (existingIndexes.length > 1) {
      print('Dropping existing indexes:', 'bright', 'magenta');
      console.log(existingIndexes);
      await collection.dropIndexes();
    }
    const collectionIndexes = collections[c];
    for (let i=0; i<collectionIndexes.length; i++) {
      const index = collectionIndexes[i];
      const dbResIndexName = await collection.createIndex(...index);
      print('Created: ' + dbResIndexName, 'green');
    }
  }
```

Απόσπασμα Κώδικα 3.4: Δημιουργία συλλογών και ευρετηρίων

```
  await initUser(db);
  print('DB initialization finished!', 'bright', 'green');
  await dbs.mg.client.close();
  print('MongoDB client disconnected.', 'green');

//...

async function initUser(db) {
  const colUsers = db.collection(config.collectionNames.users);
  if (await colUsers.countDocuments() != 0) return;
  const bcrypt = require('bcrypt');
```

```
  const newPassword = randStr(5);
  await colUsers.insertOne({
    user: config.defaultUser,
    pass: await bcrypt.hash(newPassword, config.saltRounds)
  });
  print('Creted new user with credentials:', 'bright', 'green', 'bg-magenta');
  print('username: "' + config.defaultUser + '" password: "' + newPassword + '"',
'bright', 'yellow');
}
```

Απόσπασμα Κώδικα 3.5: Δημιουργία χρήστη «admin»

Για την εκτέλεση του κώδικα έχει προστεθεί στο αρχείο package.json το παρακάτω script:

```
  "scripts": {
    "init-db": "node ./dbs/initialize.js",
```

Απόσπασμα Κώδικα 3.6: Script αρχικοποίησης βάσης δεδομένων

3.3.4 Ροές αλλαγών (Change Streams)

Οι ροές αλλαγών είναι μια λειτουργία της MongoDB με την οποία μπορεί να εγγραφτεί μια εφαρμογή σε αλλαγές δεδομένων ενός υποσυνόλου ή και ολόκληρης της συλλογής (MongoDB collection). Οι αλλαγές των δεδομένων μπορεί να είναι εισαγωγή, ενημέρωση ή και διαγραφή των εγγράφων της συλλογής. Η ενημερώσεις πραγματοποιούνται σε πραγματικό χρόνο, πράγμα που καθιστούν τις ροές αλλαγών ιδανικές για εφαρμογές παρακολούθησης ή άλλες εφαρμογές που πραγματοποιούν αναλύσεις και υπολογισμούς σε δεδομένα καθώς αυτά αλλάξουν.

Ο τρόπος δήλωσης του υποσυνόλου των εγγράφων που θα παρακολουθούνται για αλλαγές, πραγματοποιείται κατά την αρχικοποίηση της ροής αλλαγών. Η μέθοδος αρχικοποίησης της ροής αλλαγών δέχεται ως πρώτη παράμετρο ένα «aggregation pipeline» στο οποίο αναφέρεται ποια αρχεία ανήκουν στο υποσύνολο παρακολούθησης. Το aggregation pipeline είναι ένας μονοδιάστατος πίνακας ο οποίος περιέχει οδηγίες προς την MongoDB για το πως να ξεχωρίσει και να επεξεργαστεί έγγραφα.

Η μέθοδος αρχικοποίησης της ροής αλλαγών δέχεται και δεύτερη παράμετρο στην οποία αναφέρεται τί δεδομένα θα στέλνονται κατά την ενημέρωση. Η προεπιλογή είναι να στέλνονται οι αλλαγές που προκλήθηκαν στο έγγραφο σε ένα έγγραφο. Στην περίπτωση

που χρειαστεί ολόκληρο το έγγραφο, αυτό μπορεί να επιτευχθεί χρησιμοποιώντας την παράμετρο: «{ fullDocument: 'updateLookup' }».

Στην ενότητα 3.9 υπάρχει παράδειγμα χρήσης ροής αλλαγών.

3.4 Επικοινωνία με το Frontend

Η επικοινωνία με το Frontend χαρακτηρίζεται πάντα αυθεντικοποιημένη ή μη αυθεντικοποιημένη. Στην πρώτη περίπτωση η επικοινωνία πραγματοποιείται με REST, όπου ο χρήστης έχει πρόσβαση στο δημόσιο μη αυθεντικοποιημένο φάκελο “front/public” και στην αρχική σελίδα “/” στην οποία το backend θα στείλει την κατάλληλη σελίδα για να πραγματοποιηθεί η αυθεντικοποίηση. Η αυθεντικοίηση πραγματοποιείται αποστέλλοντας POST request με τα στοιχεία αυθεντικοίησης στο Rest endpoint “/”.

Έπειτα από την αυθεντικοποίηση, ο χρήστης έχει πρόσβαση στον αυθεντικοποιημένο φάκελο “secured” και στο REST endpoint “/t”, το οποίο παρέχει CSRF tokens για την ασφαλή αρχικοποίηση αυθεντικοποιημένων WebSocket. Επίσης έχει πρόσβαση στην αρχική σελίδα “/” που αυτή τη φορά αν ζητηθεί το backend θα στείλει την αυθεντικοποιημένη σελίδα της εφαρμογής.

Μετά την αυθεντικοποίηση και φόρτωση της αυθεντικοποιημένης σελίδας της εφαρμογής, το frontend θα ζητήσει με AJAX request, ένα CSRF token και στη συνέχεια θα αρχικοποιήσει επικοινωνία με το backend μέσω WebSocket. Το CSRF token είναι χρονικά εξαρτημένο (time sensitive), ενώ περιέχει κρυπτογραφημένη (AES-256) και κατακερματισμένη (hashed, SHA-256) πληροφορία σχετική με το session του αυθεντικοποιημένου χρήστη.

3.5 WebSocket controllers

Όλες οι λειτουργίες του συστήματος που μπορεί κάποιος χρήστης να εκτελέσει καλούνται μέσω της σύνδεσης WebSocket. Για την ευκολότερη διαχείριση και ανάπτυξη των λειτουργιών που παρέχει το backend προς το frontend, έχουν δημιουργηθεί WebSocket controllers τα οποία είναι κομμάτια κώδικα στα οποία θέτεται η λειτουργία για

το κάθε πιθανό μήνυμα που μπορεί να σταλθεί από τα frontend. Παρακάτω βλέπουμε ένα παράδειγμα controller:

```
controllers['users.create'] = {
  data: true,
  validators: [
    libs.validators.username,
    libs.validators.password
  ],
  cb: true,
  handler: createUser
};

async function createUser({ sessId, ws, valRes, userId, cb }) {
  const {username, password} = valRes.data;
  let dbRes;
  try {
    dbRes = await services.dbFunctions.createUser(username, password);
  } catch(e) {
    if (e.code === 11000) return cb('Duplicate username');
    throw e;
  }

  broadcast(
    'userList',
    'subs',
    {
      data: {
        action: 'createdUser',
        id: dbRes.insertedId.toString(),
        username
      }
    }
  );

  cb(null);
}
```

Απόσπασμα Κώδικα 3.7: WebSocket controller

Ο παραπάνω κώδικας περιγράφει την δημιουργία ενός νέου λογαριασμού χρήστη.

Ένας controller πρέπει να έχει αναγκαστικά μήνυμα στο οποίο «ακούει» το backend (event, στην συγκεκριμένη περίπτωση το string “users.create”) και αναφορά σε ειδική

συνάρτηση η οποία θα χειριστεί το μήνυμα (handler, στην συγκεκριμένη περίπτωση createUser). Προαιρετικά, ο controller μπορεί να αναφέρει ότι περιμένει συνάρτηση επανάκτησης (callback) και δεδομένα. Στην περίπτωση που ο controller δέχεται δεδομένα από το χρήστη, είναι αναγκαίο να τεθούν επικυρωτές δεδομένων (validators) οι οποίοι θα αναλυθούν στο επόμενο κεφάλαιο.

Ο handler πρέπει να είναι ασύγχρονη συνάρτηση (async function) και όταν εκτελείτε δέχεται σαν παράμετρο αντικείμενο από το οποίο μπορούν να αντληθούν σχετικές πληροφορίες του λογαριασμού που καλεί όπως το session, το id, το WebSocket, τα επικυρωμένα δεδομένα και το callback.

Για λόγους οργάνωσης, τα controllers έχουν χωριστεί ανάλογα με την αρμοδιότητα της κάθε κατηγορίας: users, filesystem, servers, categorizationTrees, filters και metrics.

3.6 Σταθερές κώδικα και βελτιστοποιήσεις επικοινωνίας (backend)

Στην προηγούμενη ενότητα παρατηρήσαμε ένα παράδειγμα δομής ενός controller. Σε αυτή την ενότητα θα αναλυθεί η λειτουργία «metadata» και ο τρόπος μετατροπής των event σε σταθερές/σύμβολα (tokenization).

Η λειτουργία metadata αντιστοιχεί strings με άλλα strings ή αριθμούς και δημιουργεί αρχείο «metadata.json» με τις αντιστοιχήσεις. Οι αντιστοιχήσεις αποσκοπούν να παρέχουν στον frontend κώδικα αναφορά σε σταθερές του backend συστήματος.

```
  global.metadata.addToken('appName', libs.stringify(config.title));
  global.metadata.addToken('newLog');
  global.metadata.addToken('wsUrl', libs.stringify(config.app.wsUrl));
  global.metadata.addToken('userList');
```

Απόσπασμα Κώδικα 3.8: Παραδείγματα δημιουργίας token

Όλοι οι controllers περνάνε από την διαδικασία δημιουργίας token:

```
let controllers = {};

require('./users.js')(libs, services, controllers);
require('./servers.js')(libs, services, controllers);
require('./categorizationTrees.js')(libs, services, controllers);
```

```
require('./filters.js')(libs, services, controllers);
require('./logs.js')(libs, services, controllers);
require('./metrics.js')(libs, services, controllers);

let tokenizedControllers = {};

for (let prop in controllers) {
  let controller = controllers[prop];
  // ... checks
  tokenizedControllers[ metadata.addToken(prop) ] = controller;
}
return tokenizedControllers;
```

Απόσπασμα Κώδικα 3.9: Δημιουργία tokens των controller

Με τον παραπάνω κώδικα επιτυγχάνετε η συμπίεση της παραμέτρου event των μηνυμάτων. Αυτό συμβαίνει επειδή το event θα μετατραπεί σε ακέραιο. Έτσι μπορούν τα events να έχουν αυθαίρετα πολλούς χαρακτήρες. Παρακάτω παρουσιάζονται όλα τα WebSocket events του συστήματος και οι υπόλοιπες σταθερές στις οποίες μπορεί να έχει πρόσβαση ο frontend κώδικας.

```
{
  "servers.fs": 0,
  "servers.item": 1,
  "cTrees.fs": 2,
  "cTrees.item": 3,
  "filters.fs": 4,
  "filters.item": 5,
  "logs": 6,
  "metrics.fs": 7,
  "metrics.item": 8,
  "users.list": 9,
  "users.unlist": 10,
  "users.create": 11,
  "users.delete": 12,
  "users.seessions.delete": 13,
  "users.changeUsername": 14,
  "users.changePassword": 15,
  "logout": 16,
  "servers.item.get": 17,
  "servers.item.unsub": 18,
  "servers.fs.list": 19,
  "servers.fs.unlist": 20,
  "servers.fs.create": 21,
```

```
  "servers.fs.count": 22,
  "servers.fs.delete": 23,
  "servers.fs.move": 24,
  "servers.fs.rename": 25,
  "servers.fs.setPositions": 26,
  "servers.setActivationState": 27,
  "servers.setConnectionType": 28,
  "servers.setNodeId": 29,
  "servers.setConnectionUrl": 30,
  "servers.setKeepaliveMS": 31,
  "servers.setRetryMS": 32,
  "servers.call": 33,
  "cTrees.item.get": 34,
  "cTrees.item.unsub": 35,
  "cTrees.fs.list": 36,
  "cTrees.fs.unlist": 37,
  "cTrees.fs.create": 38,
  "cTrees.fs.count": 39,
  "cTrees.fs.delete": 40,
  "cTrees.fs.move": 41,
  "cTrees.fs.rename": 42,
  "cTrees.fs.setPositions": 43,
  "cTrees.nodes.create": 44,
  "cTrees.nodes.delete": 45,
  "cTrees.nodes.rename": 46,
  "cTrees.nodes.addFilter": 47,
  "cTrees.nodes.removeFilter": 48,
  "cTrees.nodes.addGrouping": 49,
  "cTrees.nodes.removeGrouping": 50,
  "cTrees.nodes.subData.getLogs": 51,
  "cTrees.nodes.subData.getGroups": 52,
  "cTrees.nodes.unsubData": 53,
  "filters.item.get": 54,
  "filters.item.unsub": 55,
  "filters.fs.list": 56,
  "filters.fs.unlist": 57,
  "filters.fs.create": 58,
  "filters.fs.count": 59,
  "filters.fs.delete": 60,
  "filters.fs.move": 61,
  "filters.fs.rename": 62,
  "filters.fs.setPositions": 63,
  "filters.set": 64,
  "logs.getG": 65,
  "logs.get": 66,
```

```
  "logs.delete": 67,
  "metrics.item.get": 68,
  "metrics.item.unsub": 69,
  "metrics.fs.list": 70,
  "metrics.fs.unlist": 71,
  "metrics.fs.create": 72,
  "metrics.fs.count": 73,
  "metrics.fs.delete": 74,
  "metrics.fs.move": 75,
  "metrics.fs.rename": 76,
  "metrics.fs.setPositions": 77,
  "metrics.createPlot": 78,
  "metrics.setPlotTime": 79,
  "metrics.setLogCount": 80,
  "metrics.setColor": 81,
  "metrics.setName": 82,
  "metrics.setPropPath": 83,
  "metrics.setPointCount": 84,
  "metrics.deletePlot": 85,
  "appName": "\"Serverman\"",
  "TCP_TLS_CONNECT_URL": "\"skulixlabs.zapto.org:3000\"",
  "TCP_CONNECT_URL": "\"skulixlabs.zapto.org:3001\"",
  "HTTPS_REPORT_URL": "\"https://skulixlabs.zapto.org/report\"",
  "HTTP_REPORT_URL": "\"http://skulixlabs.zapto.org/report\"",
  "newLog": 86,
  "wsUrl": "\"wss://skulixlabs.zapto.org/s\"",
  "userList": 87,
  "defaultMaxPlots": "6",
  "defaultPlotTime": "864000",
  "minPlotTimeSeconds": "10",
  "maxPlotTimeSeconds": "946080000"
}
```

Απόσπασμα Κώδικα 3.10: Όλα τα WebSocket events του συστήματος και οι υπόλοιπες σταθερές

Όλες οι σταθερές του αποσπάσματος, προέρχονται από το παραγόμενο αρχείο «metadata.json». Στην ενότητα 3.6 παρουσιάζεται ο τρόπος αξιοποίησης των δεδομένων του αρχείου.

```
const metadata = {
    tokenCount: 0,
    tokens: {},
    addToken: (token, explicitData) => {
      if (explicitData !== undefined) {
```

```
        metadata.tokens[token] = explicitData;
        return explicitData;
      }
      let tokenMatch = metadata.tokens[token];
      if (tokenMatch !== undefined) return tokenMatch;
      tokenMatch = metadata.tokenCount++;
      metadata.tokens[token] = tokenMatch;
      return tokenMatch;
    },
    getTokenString: token => {
      for (str in global.metadata.tokens)
        if (global.metadata.tokens[str] == token) return str;
      return undefined;
    }
  };
```

Απόσπασμα Κώδικα 3.61: Υλοποίηση του κώδικα δημιουργίας token

3.7 Validation

Στο προηγούμενο κεφάλαιο παρατηρήσαμε ένα παράδειγμα δομής ενός controller ο οποίος δέχεται δεδομένα. Σε κάθε περίπτωση που ο controller δέχεται δεδομένα, είναι απαραίτητο να αναφέρονται οι κατάλληλοι επικυρωτές δεδομένων οι οποίοι θα χρησιμοποιηθούν για να ελέγξουν ή/και να μορφοποιήσουν κατάλληλα (sanitization) τα δεδομένα. Οι επικυρωτές δεδομένων είναι αρχεία στα οποία περιγράφονται οι έλεγχοι οι οποίοι πρέπει να γίνουν ώστε να θεωρηθεί το εξεταζόμενο δεδομένο έγκυρο. Παρακάτω παρατηρούμε αρχεία επικυρωτών:

```
module.exports = check => check('username')
  .isString()
  .not().isEmpty()
  .trim()
  .isLength({max: 20}).withMessage('Username too long')
```

Απόσπασμα Κώδικα 3.72: Username validator (string)

```
const connectionTypes = ['RTCP_TLS', 'TCP_TLS', 'HTTPS', 'RTCP', 'TCP', 'HTTP'];

module.exports = check => check('connectionType')
  .isString()
  .custom(val => connectionTypes.indexOf(val) > -1);
```

Απόσπασμα Κώδικα 3.83: Connection type validator (enum)

```
module.exports = check => check('ids')
  .custom(val => isArray(val) && val.length > 0 && val.every(id => isString(id)
&& id.length == 24))
```

Απόσπασμα Κώδικα 3.94: Multiple ids validator (array)

```
module.exports = check => check('posChanges')
  .isObject()
  .custom(posChanges => {
    const ids = Object.keys(posChanges);
    if (ids.length == 0) return false;
    for (let i=0; i<ids.length; i++) {
      let id = ids[i];
      if (id.length != 24) return false;
      if (!Number.isInteger(posChanges[id])) return false;
    }
    return true;
  })
```

Απόσπασμα Κώδικα 3.105: Position change validator (object)

3.8 Σύστημα αρχείων

Το σύστημα αρχείων αποτελεί λειτουργία οργάνωσης οντοτήτων. Για κάθε είδος οντότητας του συστήματος υπάρχει και ένα σύστημα αρχείων. Το σύστημα αρχείων αποτελείται από controllers γενικής χρήσης, οι οποίοι αρχικοποιούνται για την κάθε οντότητα του συστήματος. Η αρχικοποίηση πραγματοποιείται όπως φαίνεται παρακάτω:

```
require('./filesystem')(libs, services, controllers,
config.collectionNames.categorizationTrees);
```

Απόσπασμα Κώδικα 3.116: Αρχικοποίηση συστήματος αρχείων των controller

Η τελευταία παράμετρος χρησιμοποιείται μόνο για το όνομά της και είναι το διαχωριστικό μεταξύ των διαφόρων συστημάτων αρχείων. Μετά την αρχικοποίηση του συστήματος αρχείων, η κάθε οντότητα θα έχει αποκτήσει όλους τους controllers που επιτρέπουν τις λειτουργίες του συστήματος αρχείων. Οι λειτουργίες αυτές αναγνωρίζονται από: [οντότητα].item.[ενέργεια], [οντότητα].fs.[ενέργεια] και είναι οι παρακάτω:

- servers.item.get: Αίτημα δεδομένων οντότητας και εγγραφή σε αυτήν.
- servers.item.unsub: Απεγγραφή από οντότητα.
- servers.fs.list: Αίτημα εμφάνισης αρχείων φακέλου και εγγραφή σε αυτόν.
- servers.fs.unlist: Απεγγραφή από φάκελο.
- servers.fs.create: Δημιουργία φακέλου.
- servers.fs.count: Εμφάνιση πλήθους υποφακέλων και αρχείων φακέλου.
- servers.fs.delete: Διαγραφή οντότητας/φακέλου/πλήθους τους
- servers.fs.move: Μετακίνηση οντοτήτων/φακέλου/πλήθους τους.
- servers.fs.rename: Μετονομασία οντότητας/φακέλου.
- servers.fs.setPositions: Αλλαγή θέσης οντότητας/φακέλου.

Το κάθε σύστημα αρχείων περιέχει εσωτερικά έναν γράφο ο οποίος είναι υπεύθυνος για την απεικόνιση του συγκεκριμένου συστήματος αρχείων στη μνήμη. Ο γράφος δεν περιέχει όλα τα στοιχεία του συστήματος αρχείων, πάρα μόνο αυτά στα οποία υπάρχει ενεργή εγγραφή. Οι μέθοδοι που υποστηρίζει η υλοποίηση του γράφου, είναι οι κατάλληλες έτσι ώστε να υπάρχει πάντα στην μνήμη η ενημερωμένη δομή του συστήματος αρχείων.

```
async function move ({ sessId, ws, valRes, userId, cb }) {
  const { ids, idOrNull } = valRes.data;
  let dbRes;
  try {
    dbRes = await dbfs.move(ids, idOrNull);
    subscriptionNodesGraph.subMove(ids, dbRes.pathIds);
  } catch(e) {
    if (e.code === 11000) return cb('Duplicate name');
    throw e;
  }
  // For each folder that had something moved
  for (let rootFolder of dbRes.rootFolders) {
    broadcast(  // Notify folder subbers that files got moved (deleted)
      subClass,
      'subs',
      {
        eventSubId: rootFolder._id,
        data: {
          action: 'delete',
          ids: rootFolder.ids
```

```
          }
        }
      );
      // Notify possible subscriptions under moved folders
      for (let id of rootFolder.ids) {  // For each moved folder in this
rootFolders
        let relevantFolderIds =
subscriptionNodesGraph.getSubscribedChildNodes(id.toString()); // Get
subscriptions
        for (let subscribedId of relevantFolderIds) { // And for each
subscription
          broadcast(  // Notify path change
            subscriptionNodesGraph.isFile(subscribedId) ? itemSubClass :
subClass,
            'subs',
            {
              eventSubId: subscribedId,
              data: {
                action: 'pathMove',
                path: subscriptionNodesGraph.getPath(subscribedId),
                names: dbRes.names,
                split: id
              }
            }
          );
        }
      }
    }
    broadcast(
      subClass,
      'subs',
      {
        eventSubId: idOrNull,
        data: {
          action: 'moved here',
          items: dbRes.items
        }
      }
    );
    cb(null, 'matched: ' + dbRes.dbRes2.matchedCount + ' moved: ' +
dbRes.dbRes2.modifiedCount);
  }
```

Απόσπασμα Κώδικα 3.127: Κώδικας χειριστή controller που υλοποιεί μετακίνηση οντότητας/φακέλου/πλήθους τους

Παρατηρούμε πως με τη χρήση του γράφου (subscriptionNodesGraph) μπορούμε να ανακτήσουμε όλα τα στοιχεία (φακέλους ή και οντότητας) στα οποία υπάρχει ενεργή εγγραφή και να τον ενημερώσουμε με την κατάλληλη ενέργεια.

Ο κώδικας του γράφου είναι ο ακόλουθος:

```
Const SubscriptionNodeGraph = function() {
    this.subscriptionNodes = new Set();
    this.subscriptionNodesGraph = new Map();
    this.fileIds = new Set();

    this.sub = (path, isFile = false) => {
      if (!path.length) return;
      for (let i=0; i<path.length; i++) {
        // if node already exist and is not last node
        if (this.subscriptionNodesGraph.has(path[i])) {
          if (i < path.length-1)
this.subscriptionNodesGraph.get(path[i]).children.add(path[i+1]);
        } else this.subscriptionNodesGraph.set(
          path[i],
          {
            pid: (i==0) ? null : path[i-1],
            children: (i == path.length-1) ? new Set() : new Set([path[i+1]])
          }
        );
      }
      this.subscriptionNodes.add(path[path.length-1]);
      if (isFile) this.fileIds.add(path[path.length-1]);
    };

    this.has = id => this.subscriptionNodes.has;
    this.isFile = id => this.fileIds.has(id);

    this.getSubscribedChildNodes = (id, nodes = []) => {
      if (!this.subscriptionNodesGraph.has(id)) return nodes;
      if (this.subscriptionNodes.has(id)) nodes.push(id);
      for (let node of this.subscriptionNodesGraph.get(id).children) {
        this.getSubscribedChildNodes(node, nodes);
      }
      return nodes;
    };

    this.unsub = id => {
      this.subscriptionNodes.delete(id);
```

```
      while (
        this.subscriptionNodesGraph.has(id) &&
        this.subscriptionNodesGraph.get(id).children.size == 0 &&
        !this.subscriptionNodes.has(id)
      ) {
        let deleteId = id;
        id = this.subscriptionNodesGraph.get(id).pid;
        if (id !== null)
this.subscriptionNodesGraph.get(id).children.delete(deleteId);
        this.subscriptionNodesGraph.delete(deleteId);
      }
    };

    this.unsubR = ids => {
      for (let id of ids) {
        if (!this.subscriptionNodesGraph.has(id)) continue;
        this.unsubR(this.subscriptionNodesGraph.get(id).children);
        this.unsub(id);
      }
    };

    this.subMove = (ids, path) => {
      const targetFolderId = path.length ? path[path.length - 1] : null;
      let targetPathSet = this.subscriptionNodesGraph.has(targetFolderId);
      for (let id of ids) {
        // Get moving node
        if (!this.subscriptionNodesGraph.has(id)) continue;
        const node = this.subscriptionNodesGraph.get(id);
        let parentId = node.pid;
        let childId = id;
        // If node already has target parent, continue
        if (parentId == targetFolderId) continue;
        // Remove old path
        while (
          parentId != null
        ) {
          const parentNode = this.subscriptionNodesGraph.get(parentId);
          if (
            parentNode.children.size > 1 ||
            parentId == targetFolderId ||
            this.subscriptionNodes.has(parentId) ||
            ids.indexOf(parentId) > -1 ||
            path.indexOf(parentId) > -1
          ) {
            parentNode.children.delete(childId);
```

```
            break;
          }
          this.subscriptionNodesGraph.delete(parentId);
          // Also remove parent from grandfather node
          if (parentNode.pid != null)
this.subscriptionNodesGraph.get(parentNode.pid).children.delete(parentId);
          parentId = parentNode.pid;
        }
        // If target path does not exist, create it
        if (!targetPathSet) {
          for (let i=0; i<path.length; i++) {
            if (!this.subscriptionNodesGraph.has(path[i])) {
              this.subscriptionNodesGraph.set(
                path[i],
                {
                  pid: (i == 0) ? null : path[i-1],
                  children: (i < path.length-1) ? new Set([ path[i + 1] ]) : new
Set()
                }
              );
            }
          }
          targetPathSet = true;
        }
        // Link moving node with target node
        if (targetFolderId != null)
this.subscriptionNodesGraph.get(targetFolderId).children.add(id);
        node.pid = targetFolderId;
      }
    };

    this.getPath = id => {
      const path = [];
      if (!this.subscriptionNodesGraph.has(id)) return path;
      let pid = this.subscriptionNodesGraph.get(id).pid;
      for (; pid != null; pid = this.subscriptionNodesGraph.get(pid).pid)// pid
of undefined
        path.unshift(pid);
      return path;
    };

    return this;
  };
```

Απόσπασμα Κώδικα 3.18: Γράφος subscriptionNodesGraph

3.9 Δέντρα κατηγοριοποίησης και φίλτρα.

Τα ΔΚ αποτελούν δομή στην πτητική μνήμη η οποία υπάρχει μόνο όταν αυτή χρειάζεται. Πιο συγκεκριμένα, όταν ένα ΔΚ προβάλλεται ή αποτελεί εξάρτηση άλλου ΔΚ ή μετρικής η οποία προβάλλεται.

Ένα κομμάτι της δομής ενός ΔΚ στην πτητική μνήμη είναι η παρακάτω:

```
{
  "subscription": {
    "event": "cTrees.item:63d4058685ff9862be47ac16",
    "f": [Function: f]
  },
  "directSubscription": true,
  "childReferences": {},
  "getDataPromises": [],
  "onFullInitPromises": [],
  "initialized": true,
  "initializedFilters": true,
  "watcherNeedUpdate": false,
  "runningWatcherInitialization": false,
  "tree": { "0":  {/* ... */}, "1":  {/* ... */}, "2":  {/* ... */},... },
  //...
}
```

Απόσπασμα Κώδικα 3.149: Δομή ΔΚ στην πτητική μνήμη.

Η παράμετρος «subscription» αποτελεί αναφορά εγγραφής του συστήματος για αλλαγές στο συγκεκριμένο ΔΚ, ενώ η παράμετρος «directSubscription» αναφέρει αν η τρέχουσα δομή του ΔΚ δημιουργήθηκε από άμεση ή έμμεση εγγραφή. Η παράμετρος χρειάζεται επειδή το σύστημα πρέπει να γνωρίζει αν θα χρειαστεί να διαγράψει την δομή από την μνήμη τη στιγμή που πραγματοποιείται απεγγραφή του παρόντος ΔΚ ή ΔΚ το οποίο έχει εξάρτηση σε αυτό. Η παράμετρος «childReferences» κρατάει αναφορά στις δομές ΔΚ οι οποίες έχουν εξάρτηση στην παρούσα δομή. Η παράμετροι «getDataPromises» και «onFullInitPromises» κρατάνε αναφορές σε υποσχέσεις (promises) οι οποίες θα εκπληρωθούν την στιγμή που θα φορτωθούν τα δεδομένα του ΔΚ στη δομή και την στιγμή που θα φορτωθούν και τα φίλτρα στη δομή αντίστοιχα. Η παράμετρος «initialized» αναφέρει αν τα δεδομένα του ΔΚ έχουν φορτωθεί στη δομή, ενώ η παράμετρος «initializedFilters» αναφέρει αν έχουν φορτωθεί και τα φίλτρα των φακέλων του ΔΚ. Οι παράμετροι «watcherNeedUpdate» και «runningWatcherInitialization»

αναφέρουν αν οι ροές αλλαγών του ΔΚ χρειάζονται ενημέρωση ή αν πραγματοποιείται ενημέρωση αντίστοιχα. Η παράμετρος «tree» περιέχει τους φακέλους του ΔΚ και ο κάθε φάκελος αποτελεί και ο ίδιος JavaScript αντικείμενο. Παράδειγμα δομής φακέλου:

```
"2": {
      "name": "warnings",
      "filters": [ "62e15b02d81863ba73225c43" ],
      "children": [ 3 ],
      "groupings": [
        { "n": "client.ip", "q": "client.ip" },
        { "n": "client.url", "q": "client.url" },
        { "n": "Country", "q": "client.country" },
        { "n": "City", "q": "client.city" }
      ],
      "aggregatedFilters": [
        [
          {
            "negative": true,
            "data": { "logObj.level": "error" },
            "isTime": false
          }
        ],
        [
          {
            "data": { "logObj.level": "warn" },
            "isTime": false
          }
        ]
      ],
      "aggregatedChildFilters": [
        [
          {
            "negative": true,
            "data": { "logObj.status": 404 },
            "isTime": false
          }
        ]
      ],
      "logCount": 0
      "changeStream": ...
    },
```

Απόσπασμα Κώδικα 3.150: Δομή φακέλου ΔΚ στην πτητική μνήμη

Ο κάθε φάκελος αναφέρετε με το id του και περιέχει:

- name: Όνομα φακέλου
- filters: Id φίλτρων φακέλου
- children: Id απογόνων φακέλου.
- groupings: Κατηγοριοποιήσεις αρχείων καταγραφής.
- aggregatedFilters: Συλλογή δεδομένων φορτωμένων φίλτρων φακέλου.
- aggregatedChildFilters: Συλλογή δεδομένων φορτωμένων φίλτρων άμεσων απογόνων φακέλου.
- logCount: Αριθμός αρχείων καταγραφής φακέλου.
- changeStream: Αναφορά ροής αλλαγών

Η παράμετρος «aggregatedFilters» περιέχει όλα τα δεδομένα των φορτωμένων φίλτρων των προγόνων του φακέλου και των φακέλων αριστερά από αυτόν (αδερφικών). Στην περίπτωση που υπάρχουν φίλτρα φακέλων αριστερά από αυτόν, αυτά φέρουν την πληροφορία «negative». Όλα τα φίλτρα των άμεσων απογόνων φέρουν και αυτά την πληροφορία «negative».

Η παράμετρος changeStream είναι αναφορά σε ροή αλλαγών και χρησιμοποιείται για να τη σταματήσει όταν καταργηθεί η δομή του ΔΚ από τη μνήμη. Η αρχικοποίηση της ροής πραγματοποιείται με τη μέθοδο «watch» η οποία δέχεται ένα «pipeline». Το pipeline περιέχει την κατάλληλη λειτουργία για την επεξεργασία των αλλαγών και το ταίριασμά με τα φίλτρα ενός φακέλου. Το πρώτο βήμα για την δημιουργία του pipeline είναι να περαστούν σε αυτήν τα δεδομένα των φίλτρων και για αυτή την διαδικασία έχει δημιουργηθεί η μέθοδος «pipelineFilters».

```
const pipelineFilters = (pipeline, filterBundles) => {
  for (const filterBundle of filterBundles) {
    const negativeFilters = [];
    for (const filter of filterBundle) {
      if (filter.isTime) continue;
      if (filter.negative) negativeFilters.push(filter.data);
      else pipeline.push({ $match: filter.data });
    }
    if (negativeFilters.length) pipeline.push({ $match: {$nor: [ {$and:
negativeFilters} ]} });
```

```
  }
};
```

Απόσπασμα Κώδικα 3.161: Μέθοδος δημιουργίας pipeline

```
  const filterPipeline = [];
  pipelineFilters(filterPipeline, node.aggregatedFilters);
  pipelineFilters(filterPipeline, node.aggregatedChildFilters);
```

Απόσπασμα Κώδικα 3.172: Κλήση μεθόδου δημιουργίας pipeline

Όπου «node», φάκελος ενός ΔΚ.

Η δημιουργία του τελικού pipeline και χρησιμοποίησή του για αρχικοποίηση της ροής αλλαγών φαίνεται παρακάτω:

```
  const pipeline = [
    {
      $set: {
        'fullDocument._id': '$_id',
        'fullDocument._changeStream': {
          fullDocumentId: '$fullDocument._id',
          operationType: '$operationType',
          updateDescription: '$updateDescription'
        }
      }
    },
    {
      $replaceRoot: { newRoot: '$fullDocument' }
    },
    ...filterPipeline
  ];
  node.changeStream = col.watch(pipeline, { fullDocument: 'updateLookup' });
  node.changeStream.on('change', document => { /* ... */ };
```

Απόσπασμα Κώδικα 3.183: Δημιουργία pipeline και αρχικοποίηση ροής αλλαγών

Οι δύο πρώτα στάδια ($set και $replaceRoot) χρησιμοποιούνται για τη μορφοποίηση του αρχείο αλλαγής. Το αρχείο αλλαγών «document» αποτελεί αντικείμενο που επιστρέφεται ως πρώτη παράμετρος από τη ροή αλλαγών όταν πραγματοποιείται κάποια αλλαγή. Σε αυτό το αντικείμενο περιγράφετε ο τύπος της ενέργειας που έγινε στο έγγραφο ή και οι αλλαγές του αρχείου αν έγιναν. Σε αυτή την περίπτωση χρειάζεται να μορφοποιηθεί το αρχείο αλλαγών για να μπορέσει να ταιριάξει με τα φίλτρα του pipeline.

Τα φίλτρα έχουν και αυτά τη δική τους δομή στη μνήμη και παρουσιάζεται παρακάτω:

```
{
  treeIds: ['63d4058685ff9862be47ac16'],
  data: { 'logObj.level': 'warn' },
  isTime: false,
  subscription: {
    event: 'filters.item:62e15b02d81863ba73225c43',
    f: [Function: wrapperFunction]
  },
  initialized: true
}
```

Απόσπασμα Κώδικα 3.194: Δομή φίλτρου στην πτητική μνήμη

Οι παράμετροι της δομής των φίλτρων είναι:

- treeIds: Id ΔΚ από τα οποία υπάρχει αναφορά στο συγκεκριμένο φίλτρο.
- data: Δεδομένα φίλτρου (query).
- isTime: Αναφέρει αν πρόκειται για χρονικό φίλτρο.
- subscription: Αναφορά εγγραφής του συστήματος για αλλαγές φίλτρου.
- initialized: Αναφέρει αν έχουν φορτωθεί τα δεδομένα του φίλτρου.

ΚΕΦΑΛΑΙΟ 4

FRONT END

4.1 Τεχνολογίες Ανάπτυξης

Το λογισμικό του πελάτη έχει αναπτυχθεί με Pug, Stylus και JavaScript. Το Pug αποτελεί HTML template engine χρησιμοποιείται για την δημιουργία πιο ευανάγνωστου κώδικα HTML. Το Stylus είναι μια δυναμική γλώσσα προ-επεξεργαστή CSS (CSS preprocessor) και παρέχει επεκτάσεις στην γλώσσα περιγραφής CSS. Πρέπει να σημειωθεί ότι ο τελικός κώδικας που δημιουργεί το Pug και το Stylus είναι HTML και CSS αντίστοιχα. Το τελικό λογισμικό του πελάτη εκτός από αρχεία PUG και Stylus, αποτελείται και από αρχεία JavaScript και SVG (μορφή διανυσματικής εικόνας). Όλα αυτά τα αρχεία τυλίσσονται (bundling) σε ένα τελικό αρχείο JavaScript με χρήση του Webpack, το οποίο αποτελεί εργαλείο που επιτρέπει την modular προσέγγιση ανάπτυξης με χρήση γραφήματος εξάρτησης.

4.2 Webpack

Ο κώδικας είναι χωρισμένος σε λογικά τμήματα από τα οποία το κάθε τμήμα είναι υπεύθυνο για μία ξεχωριστή λειτουργία. Για την επίτευξη την διαίρεσης, αλλά και για περισσότερες λειτουργίες και μεθόδους μετασχηματισμού του τελικού κώδικα κατά το χτίσιμο του, έχει χρησιμοποιηθεί το Webpack με ειδικό configuration. Μερικές από αυτές τις λειτουργίες αποτελούν:

- Η διάσπαση του κώδικα σε αρχεία/λογικά κομμάτια.
- Η φόρτωση και ο μετασχηματισμός αρχείων/στοιχείων (js, pug, stylus, svg).
- Ελαχιστοποίηση κώδικα (minification).
- Ο επιλεκτικός τρόπος μεταγλώττισης ανάλογα με το mode μεταγλώττισης (development, production).
- Ο μετασχηματισμός κώδικα.
- Δημιουργία χαρτών πηγής (source maps)

Στο αρχείο package.json έχουν προστεθεί δύο νέα scripts τα οποία εκκινούν το Webpack με τις κατάλληλες παραμέτρους.

```
  "scripts": {
    "prod-front": "webpack --config ./front/webpack.config.js --mode production",
    "dev-front": "webpack --config ./front/webpack.config.js --mode development --watch",
```

Απόσπασμα Κώδικα 20.1: Webpack scripts

Το script «dev-front» είναι υπεύθυνο για την εκκίνηση του Webpack σε κατάσταση ανάπτυξης εφαρμογής και ενεργοποιεί την επιλογή watch (αν πραγματοποιηθεί αλλαγή στα επεξεργαζόμενα αρχεία, ξαναπραγματοποιείται μετασχηματισμός των αρχείων) ενώ το «prod-front» εκκινεί το Webpack σε κατάσταση παραγωγής. Και τα δύο script αναφέρουν τη θέση του αρχείο config που θα χρησιμοποιηθεί. Ανάλογα με το script που θα χρησιμοποιηθεί, κάποιες επιπλέον διαφορές εκτός από την προφανή διαφορά που έχουν με την επιλογή watch, είναι ότι στην επιλογή «development», πραγματοποιείτε δημιουργία κανονικών χαρτών πηγής (source maps) ενώ στην επιλογή «production» οι χάρτες πηγής είναι κρυφοί και τα τελικά αρχεία που θα παραχθούν έχουν δεχθεί ελαχιστοποίηση (minification). Παρακάτω παρατηρούμε την διαφορά στο τελικό μέγεθος των αρχείων (αριστερά development, δεξιά production):

Name	Type	Size
app.css	CSS Source File	138 KB
app.css.map	MAP File	131 KB
app.js	JavaScript Source ...	541 KB
app.js.map	MAP File	593 KB
login.js	JavaScript Source ...	3 KB
login.js.map	MAP File	4 KB
loginStyl.css	CSS Source File	6 KB
loginStyl.css.map	MAP File	7 KB

Name	Type	Size
app.css	CSS Source File	123 KB
app.css.map	MAP File	123 KB
app.js	JavaScript Source ...	217 KB
app.js.map	MAP File	738 KB
login.js	JavaScript Source ...	2 KB
login.js.map	MAP File	5 KB
loginStyl.css	CSS Source File	5 KB
loginStyl.css.map	MAP File	7 KB

Εικόνα 4.2: Διαφορά μεγέθους αρχείων development/production mode

Το configuration αρχείο του Webpack που χρησιμοποιήθηκε, είναι το ακόλουθο:

```
var path = require('path');
const { CleanWebpackPlugin } = require('clean-webpack-plugin'); // For removing all files of build dir
```

```
const TerserPlugin = require('terser-webpack-plugin'); // For removing header
comments
const MiniCssExtractPlugin = require('mini-css-extract-plugin'); // For
extracting css files from bundle
// For removing empty js files (created when requiring styl files directly)
const RemoveEmptyScriptsPlugin = require('webpack-remove-empty-scripts');
module.exports = function(env, argv) {
    const isProductionMode = argv.mode !== 'development';
    return {
        devtool: isProductionMode ? 'hidden-source-map' : 'source-map',
        mode: argv.mode,
        entry: {
            login: path.resolve(__dirname, './src/login/index.js'),
            loginStyl: path.resolve(__dirname, './src/login/index.styl'),
            app: path.resolve(__dirname, './src/app/index.js')
        },
        output: {
            path: path.resolve(__dirname, './dist'),
            filename: '[name].js',
            publicPath: ''
        },
        module: {
            rules: [
                { test: /\.styl$/,
                  use: [
                        MiniCssExtractPlugin.loader,
                        { loader: 'css-loader', options: { url: false } },
                        {   // better minification
                            loader: 'postcss-loader',
                            options: {
                                postcssOptions: {
                                    plugins: isProductionMode ? [
require('cssnano')({ preset: 'default' }) ] : []
                                }
                            }
                        },
                        {
                           loader: 'stylus-loader',
                           options: { stylusOptions: { resolveURL: false } }
                        }
                    ]
                },
                { test: /\.pug$/, use: ['apply-loader', 'pug-loader'] },
                { test: /\.svg$/, loader: 'svg-inline-loader' },
                { test: /\.js$/,
```

```
                loader: path.resolve(__dirname,
'./loaders/SocketEventsLoader.js'),
                options: {
                  metatadaFilepath: path.resolve('./services/metadata.json')
                }
              }
            ]
        },
        optimization: {
            minimize: isProductionMode,
            minimizer: [
                new TerserPlugin({
                    terserOptions: {
                        format: { comments: false },
                    },
                    extractComments: false,
                }),
            ]
        },
        plugins: [
            new RemoveEmptyScriptsPlugin(),
            new CleanWebpackPlugin(),
            new MiniCssExtractPlugin({
                filename: '[name].css',
            })
        ]
    };
};
```

Απόσπασμα Κώδικα 4.21: Webpack configuration

Το config αρχείο πραγματοποιεί εξαγωγή μεθόδου η οποία όταν κληθεί επιστρέφει το τελικό αντικείμενο config με τα πεδία:

- devtool: Ελέγχει εάν και πώς δημιουργούνται οι χάρτες πηγής.
- mode: Ανάλογα το mode, το Webpack πραγματοποιεί της κατάλληλες βελτιστοποιήσεις.
- entry: Δήλωση σημείων για αρχή δημιουργίας πακέτου.
- output: Δήλωση τρόπου και σημείου εξαγωγής αρχείων.
- module: Καθορισμός τρόπου αντιμετώπισης διαφόρων τύπων αρχείων.
- optimization: Δήλωση για το αν θα πραγματοποιηθεί ελαχιστοποίηση και αναφορά της βιβλιοθήκης που θα χρησιμοποιηθεί.

- plugins: Δήλωση πρόσθετων επιλογών για προσαρμογή της διαδικασίας δημιουργίας.

4.3 Αρχιτεκτονική

Όπως παρατηρήσαμε στην προηγούμενη ενότητα κατά τη δήλωση σημείων για αρχή δημιουργίας πακέτου (entry), υπάρχει αναφορά σε δύο JavaScript αρχεία. Το ένα αρχείο περιέχει τον κώδικα της εφαρμογής που είναι υπεύθυνος για την αυθεντικοποίηση του χρήστη και άλλος για όταν ο χρήστης έχει ήδη αυθεντικοποιηθεί.

Ο κώδικας αυθεντικοποίησης είναι απλός και το πιο ενδιαφέρον τμήμα του αποτελεί η ίδια η αυθεντικοποίηση:

```
    let res = await fetch('', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify([ username.value.trim(), password.value ])
    });

    let data = await res.text();

    if (res.status != 200) throw new Error(res.status + ' ' + res.statusText +
(res.statusText !== data ? '\n' + data : ''));

    const s = document.createElement('script');
    s.innerHTML = data;
    document.body.appendChild(s);
```

Απόσπασμα Κώδικα 4.3: Αποστολή στοιχείων αυθεντικοποίησης

Η αυθεντικοποίηση πραγματοποιείται αποστέλλοντας τα κατάλληλα στοιχεία στο backend με χρήση αίτησης AJAX. Αν η αυθεντικοποίηση ήταν επιτυχής, το backend αποστέλλει τον κώδικα της κύριας εφαρμογής, ο οποίος εφαρμόζεται κατάλληλα έτσι ώστε να είναι σαν να είχε φορτωθεί εξαρχής. Ακολουθεί υπόδειξη και επεξήγηση αποσπάσματος του κύριου κώδικα από το αρχικό αρχείο που αναφέρεται στο entry του Webpack:

```
import './index.styl'
```

```
const actions = {};
var state = {
  r: async function(err, category, event) {
    console.error(err);
  },
  actions,
  sub: function (event, f) {
    if (!actions[event]) actions[event] = new Set();
    let wrapperFunction;
    if (f.constructor.name == 'AsyncFunction')
      wrapperFunction = args => {
        f.call(...args).catch(e => {
          state.r(e, 'subscribed_function', event);
        });
      };
    else
      wrapperFunction = args => {
        try {
          f.call(...args);
        } catch(e) {
          state.r(e, 'subscribed_function', event);
        }
      };
    actions[event].add(wrapperFunction);
    return {event, f: wrapperFunction};
  },
  unsub: function (subscription) {
    if (!subscription || !actions[subscription.event]) return;
    actions[subscription.event].delete(subscription.f);
    if (!actions[subscription.event].size) delete actions[subscription.event];
  },
  pub: function (event) {
    if (!actions[event]) return;
    let args = Array.from(arguments);
    args[0] = this;
    for (let f of actions[event]) f(args);
  },
  workplace: undefined,
  openModals: []
};

require('./lib')(state);
state.body = $('body');
require('./assetLoader');
require('./toasts')(state);
```

```
require('./ws.js')(state);
require('./classes')(state);
require('./nav')(state);
require('./workplaces')(state);
require('./modals').default(state);
```

Απόσπασμα Κώδικα 4.4: Webpack entry / αρχικό αρχείο

Μεταξύ των τμημάτων κώδικα υπάρχει αναφορά σε κοινό αντικείμενο κατάστασης (state object) και η επικοινωνία μεταξύ των τμημάτων κώδικα πραγματοποιείται με το μοτίβο publish-subscribe. Οι μέθοδοι publish-subscribe βρίσκονται στο state object του οποίου η αναφορά περνάει σαν παράμετρος στα τμήματα κώδικα. Τα τμήματα κώδικα είναι τα ακόλουθα:

- **lib**: Βιβλιοθήκες
- **assetLoader**: Φορτωτής στοιχείων (περιέχει εικόνες και παρέχει κατάλληλη λειτουργικότητα για την φόρτωσή τους).
- **toasts**: Ενημερωτικά μηνύματα
- **ws**: Λειτουργίες WebSocket
- **classes**: Κλάσεις αντικειμένων
- **nav**: Μπάρα πλοήγησης
- **workplaces**: Οντότητες συστήματος (εξυπηρετητές, ΔΚ, φίλτρα, μετρικές).
- **modals**: Αναδυόμενα παράθυρα

4.4 Επικοινωνία με το backend

Για την επικοινωνία της εφαρμογής με το backend επιλέχθηκε η χρήση WebSocket, λόγω της αμεσότητας στην αποστολή δεδομένων και για την δυνατότητα αρχικοποίησης αποστολής δεδομένων και από το backend. Τα μόνα σημεία στα οποία δεν υπάρχει επικοινωνία μέσω WebSocket είναι:

- Κατά την αυθεντικοποίηση του χρήστη (POST request).
- Κατά τη λήψη CSRF token για την αυθεντικοποίηση του ίδιου του WebSocket (POST request).

- Κατά την λήψη σπάνια χρησιμοποιούμενων εικόνων όπως οι σημαίες χωρών (GET request)

Την αρχικοποίηση της σύνδεσης μέσω WebSocket την πραγματοποιεί το frontend. Αρχικά, αποστέλλεται POST request στο backend για την απόκτηση CSRF token. Ο τρόπος δημιουργίας του CSRF token και η χρησιμότητά του αναλύθηκαν στην ενότητα 3.4. Παρακάτω παρατηρούμε τον τρόπο ανάκτησης CSRF token:

```
let t;
try {
  const r = await fetch('t', { method: 'POST' });
  if (r.status != 200) throw r;
  t = await r.text();
} catch(e) {
  if (e.status == 403) return location.reload();
  state.r(e, 'ws_token_fetch');
  onConClose(e);
  return;
}
```

Απόσπασμα Κώδικα 4.5: Ανάκτηση CSRF token

Στην περίπτωση που ο χρήστης δεν είναι αυθεντικοποιημένος ή έχει μπει σε λίστα απόρριψης (blacklist), το backend θα αρνηθεί την αποστολή token, θα επιστραφεί μήνυμα σφάλματος (403) και θα πραγματοποιηθεί επανάληψη φόρτωσης της σελίδας. Για οποιοδήποτε άλλο σφάλμα θα κληθεί η μέθοδος «onConClose», η οποία θέτει timeout για την επανάληψη της διαδικασίας και πραγματοποιεί τις κατάλληλες δημοσιεύσεις (disconnection event) αν υπήρξε πρωτίστως WebSocket σύνδεση.

Μετά την ανάκτηση token, πραγματοποιείται η σύνδεση WebSocket με το backend:

```
ws = new WebSocket(_backendToken(wsUrl));
```

Απόσπασμα Κώδικα 4.6: Αρχικοποίηση σύνδεσης WebSocket

Και θέτονται οι παρακάτω χειριστές συμβάντων:

- ws.onerror

```
ws.onerror = err => {
  state.r(err, 'ws_onerror');
};
```

Απόσπασμα Κώδικα 4.7: Χειριστής σφαλμάτων WebSocket

- ws.onclose

```
ws.onclose = onConClose;
```

Απόσπασμα Κώδικα 4.8: Ανάθεση χειριστή κλεισίματος WebSocket

- ws.onmessage

```
ws.onmessage = msg => {
  ws.onmessage = handleMessage;
  state.wsConnected = true;
  state.ss = send;
  if (wasConnected) state.pub('reconnection', JSON.parse(msg.data));
  else state.pub('connection', JSON.parse(msg.data));
  wasConnected = true;
  setPingTimeout();
};
```

Απόσπασμα Κώδικα 4.9: Αρχικός χειριστής μηνυμάτων WebSocket

- ws.onopen

```
ws.onopen = e => { ws.send(t) };
```

Απόσπασμα Κώδικα 4.10: Χειριστής αρχικοποίησης σύνδεσης WebSocket

Οποιοδήποτε σφάλμα εμφανιστεί σχετικό με το WebSocket, θα κληθεί ο χειριστής σφάλματος «onerror» και στη συνέχεια θα τερματιστεί η σύνδεση. Όταν τερματίζεται η σύνδεση, τότε καλείται ο χειριστής «onclose» ο οποίος θέτει timeout για την προσπάθεια επανασύνδεσης με το backend. Όταν αρχικοποιηθεί η σύνδεση, καλείται ο χειριστής «onopen» ο οποίος στέλνει στο backend το token το οποίο έλαβε πρώτιστος. Τέλος, ο χειριστής «onmessage» καλείται όταν υπάρχει μήνυμα από το backend.

Το πρώτο μήνυμα από το backend περιέχει πληροφορίες σχετικές με το χρήστη. Αμέσως μετά την αποδοχή αυτού το μηνύματος η σύνδεση θεωρείται αρχικοποιημένη. Για τον χειρισμό των νέων μηνυμάτων από το backend θα τεθεί ο νέος χειρίστης μηνυμάτων «handleMessage» και για την αποστολή μηνυμάτων προς το backend θα τεθεί η μέθοδος «ss» (socket send).

Η μέθοδος «ss»/«send» είναι υπεύθυνη για την αποστολή μηνυμάτων προς το backend. Στην περίπτωση που υπάρχει επανάκληση, αυτή αποθηκεύεται και δημιουργείται ένα timeout, εφόσον δεν υπάρξει κλήση της σε επιτρεπτό χρονικό διάστημα. Ο κώδικας την μεθόδου «ss»/«send» είναι ο ακόλουθος:

```
function send({ event, data, cb }) {
  let sendObj = [ event ];
  if (typeof data !== 'undefined') sendObj.push(data);
  if (typeof cb === 'function') {
    let callbackEventId = getEventId();
    const timeout = setTimeout(() => {
      delete callbackEvents[callbackEventId];
      cb(new Error('Timed out'));
    }, timeOutMs);
    callbackEvents[callbackEventId] = {
      cb,
      event,
      timeout
    };
    sendObj.push(callbackEventId);
  }
  setPingTimeout();
  ws.send(JSON.stringify(sendObj));
}
```

Απόσπασμα Κώδικα 4.11: Μέθοδος αποστολής μηνημάτων Websocket

Ο χειριστής μηνυμάτων «handleMessage» αποσειριοποιεί (deserialize) τα δεδομένα από το backend και τα δημοσιεύει στην εφαρμογή με χρήση της μεθόδου “pub”. Στην περίπτωση που το μήνυμα αποτελεί αποτέλεσμα επανάκλησης (data[0]=-1), τότε καλείται η αποθηκευμένη επανάκληση, εφόσον ο χρόνος αναμονής της δεν έχει λήξει. Παρακάτω παρουσιάζεται ο κώδικας της μεθόδου «handleMessage»:

```
function handleMessage({data}) {
  data = JSON.parse(data);
  if (data[0] != -1) {
    state.pub.call(data[2], data[0], data[1]);  // data[2] is the "this" arg
    return;
  }

  let cbEvent = callbackEvents[data[2]];
  if (!cbEvent) return;   // callback timed out

  clearTimeout(cbEvent.timeout);
  let f = cbEvent.cb;
  let event = cbEvent.event;
  delete callbackEvents[data[2]];

  if (f.constructor.name == 'AsyncFunction') {
```

```
      f(...data[1]).catch(e => {
        state.r(e, 'ws_callback', event);
      });
    } else {
      try {
        f(...data[1]);
      } catch(e) {
        state.r(e, 'ws_callback', event);
      }
    }
  }
```

Απόσπασμα Κώδικα 4.12: Νέος χειριστής μηνυμάτων WebSocket

4.5 Σημαντικότερες βιβλιοθήκες και κλάσεις

4.5.1 Query lib

Για τον ευκολότερο χειρισμό των HTML στοιχείων έχει δημιουργηθεί ειδική βιβλιοθήκη, η οποία επιτρέπει την εύκολη αναζήτηση στοιχείων από το DOM και ανάθεση σε αυτά εύχρηστες μεθόδους. Η μέθοδος η οποία κάνει εύρεση των στοιχείων και ανάθεση αυτών των μεθόδων είναι η ακόλουθη:

```
window.$ = function $(t) {
  if (arguments.length > 1) {
    const funcObj = {};
    const elements = Array.from(arguments);
    Object.keys(functions).forEach(f => {
      funcObj[f] = function() {
        for(var i = 0; i < elements.length; ++i)
$(elements[i])[f](...Array.from(arguments) );
      }
    });
    return funcObj;
  }
  var s;
  if (typeof t !== 'string') s = t;
  else s = document.querySelector(t);
  if (!s) return false;
  if (s.$initialized) return s;
  s.$initialized = true;
  s.realRemove = s.remove;
  Object.keys(functions).forEach(f => {
```

```
    s[f] = functions[f].bind(s);
  });
  return s;
};
```

Απόσπασμα Κώδικα 4.13: Μέθοδος «$»

Το αντικείμενο «functions» περιέχει όλες τις μεθόδους οι οποίες θα προστεθούν στα στοιχεία και μπορούν να κληθούν όπως στο παρακάτω παράδειγμα:

```
$('.user[data-id=5]').find('.close').hasDescendant(e.target);
$('#backdrop').toggleClass('open');
```

Απόσπασμα Κώδικα 4.14: Χρήση μεθόδου «$»

Οι μέθοδοι που θα ανατεθούν στα στοιχεία είναι οι παρακάτω:

- find: Εύρεση στοιχείου με χρήση querySelector μέσα από το τρέχον στοιχείο. Η διαφορά από το querySelector του browser είναι ότι το επιστρεφόμενο στοιχείο είναι αρχικοποιημένο με την βιβλιοθήκη ($).
- findAll: Εύρεση όλων των στοιχείων μέσα από το τρέχον στοιχείο και αρχικοποίησή τους ($).
- pre: Πρόσθεση στοιχείου (αρχικοποιημένου, ή σε μορφή HTML string, ή τύπου Nodelist/Element array) μπροστά από το τρέχον στοιχείο.
- add: Πρόσθεση στοιχείου (αρχικοποιημένου, ή σε μορφή HTML string, ή τύπου Nodelist/Element array) μέσα στο τρέχον στοιχείο.
- after: Πρόσθεση στοιχείου (αρχικοποιημένου, ή σε μορφή HTML string, ή τύπου Nodelist/Element array) μετά από το τρέχον στοιχείο.
- set: Άδειασμα τρέχοντος στοιχείου και πρόσθεση στοιχείου (αρχικοποιημένου, ή σε μορφή HTML string, ή τύπου Nodelist/Element array).
- empty: Άδειασμα τρέχοντος στοιχείου.
- txt: Ανάθεση τιμής τύπου string στο τρέχον στοιχείο. Αντικατάσταση όλων των περιεχομένων του τρέχοντος στοιχείου ή απλή με την τιμή ή απλή προσάρτηση. Η μέθοδος είναι XSS ασφαλής.
- hasClass: Επιστροφή true αν το τρέχον στοιχείο έχει συγκεκριμένη κλάση.
- addClass: Προσθήκη κλάσης στο τρέχον στοιχείο.
- removeClass: Αφαίρεση κλάσης από το τρέχον στοιχείο.

- toggleClass: Προσθήκη ή αφαίρεση κλάσης από το τρέχον στοιχείο.
- click: Προσθήκη χειριστή συμβάντος click.
- offClick: Αφαίρεση χειρίστη συμβάντος click.
- click1: Προσθήκη χειριστή συμβάντος click και αυτόματη αφαίρεσή του μετά την εκτέλεσή του.
- on: Προσθήκη χειριστή συμβάντος.
- off: Αφαίρεση χειριστή συμβάντος ή όλων των χειριστών συμβάντων.
- one: Προσθήκη χειριστή συμβάντος και αυτόματη αφαίρεσή του μετά την εκτέλεσή του.
- trigger: Σκανδάλη για εκτέλεση χειριστών συμβάντων του τρέχοντος στοιχείου με συγκεκριμένο συμβάν.
- tEnd: Προσθήκη χειριστή συμβάντος για λήξη μεταβάσεων τρέχοντος στοιχείου.
- forveTEnd: Εξαναγκασμός τελειωμού μετάβασης στοιχείου.
- remove: Αφαίρεση τρέχοντος στοιχείου.
- getStryle: Ανάκτηση τελικών ανατεθειμένων στυλ τρέχοντος στοιχείου.
- getHeight: Ανάκτηση τιμής ύψους στοιχείου.
- getWidth: Ανάκτηση πλάτους στοιχείου.
- hasDescendant: Επιστροφή true αν το εξεταζόμενο στοιχείο είναι απόγονος του τρέχοντος στοιχείου.
- isDescendant: Επιστροφή true αν το τρέχον στοιχείο είναι απόγονος του εξεταζόμενου.

Εκτός από την μέθοδο «$», η οποία προστίθεται στην καθολική μεταβλητή «window», προστίθενται και άλλες μέθοδοι από τις οποίες οι σημαντικότερες είναι:

- domify: Δημιουργία ενός ή πολλών στοιχείων από HTML string και αρχικοποίησή τους με την μέθοδο «$».
- $new: Δημιουργία νέου στοιχείου και ανάθεση κλάσεων/id από query selector string.
- rAF: requestAnimationFrame το οποίο έχει τη δυνατότητα να ακυρωθεί.

- cAFCb: Ακύρωση requestAnimationFrame και εκτέλεση συγκεκριμένης επανάκλησης.
- cancelAF: Ακύρωση συγκεκριμένης επανάκλησης requestAnimationFrame.
- clone: Βαθιά κλωνοποίηση αντικειμένου.

4.5.2 Asset loader

Όλες οι εικόνες που χρησιμοποιούνται στην εφαρμογή είναι τύπου SVG, με εξαίρεση του φόντου και των εικόνων που αναπαριστούν σημαίες χωρών. Το σημείο από το οποίο πραγματοποιείται η φόρτωση τους είναι μέσα από τον κώδικα (js) της εφαρμογής.

Για να φορτωθεί μια εικόνα στον κώδικα HTML ενός στοιχείου, είναι απαραίτητο να:

- Υπάρχει η εικόνα στο αποθετήριο εικόνων.
- Υπάρχει αναφορά στην εικόνα μέσα από τον HTML κώδικα.
- Γίνει χρήση του asset loader για την κατάλληλη μορφοποίηση του HTML κώδικα.

Παρακάτω παρουσιάζεται ένα παράδειγμα χρήσης του asset loader:

```
#nav.hidden.disconnected
  .navIcon#servers(title='Servers')
    asset navServer
  .navIcon#logCategorizationTrees(title='Categorization trees')
    asset navCTree
  .navIcon#metrics(title='Metrics')
    asset navMetrics
  #logo @logo
  .navIcon#logout(title='Logout')
    asset navLogout
  .navIcon#btnUser(title='Users')
    asset navUser
    #userName
```

Απόσπασμα Κώδικα 4.15: Παράδειγμα asset loader, το αρχείο index.pug τύπου HTML (pug)

Το JavaScript αρχείο εσωτερικά καλεί:

```
loadAssets(require('./index.pug'))
```

Απόσπασμα Κώδικα 4.16: Φόρτωση εικόνων κώδικα

Και το αποτέλεσμα είναι ο μορφοποιημένος HTML κώδικας μαζί με τις εικόνες.

```
<div class="hidden disconnected" id="nav">
  <div class="navIcon" id="servers" title="Servers">
    <svg viewBox="0 0 4.3294 93.912">···
    </svg></div>
  <div class="navIcon" id="logCategorizationTrees" title="Categorization trees">
    <svg viewBox="0 0 178.08 165.28">···
    </svg></div>
  <div class="navIcon" id="metrics" title="Metrics">
    <svg viewBox="0 0 512 512">···
    </svg></div>
  <div id="logo"
    style="background-image:url(
  </div>
  <div class="navIcon" id="logout" title="Logout">
    <svg viewBox="0 0 1000 1000">···
    </svg></div>
  <div class="navIcon" id="btnUser" title="Users">
    <svg viewBox="0 0 72.434 90.424">···
    </svg>
    <div id="userName"></div>
  </div>
</div>
```

Απόσπασμα Κώδικα 4.17: Μορφοποιημένος κώδικας HTML με εικόνες

Παρατηρούμε πως υπάρχουν δύο τρόποι φόρτωσης εικόνων:

- γράφοντας το στοιχείο «asset» και αναφέροντας το όνομα της εικόνας
- γράφοντας τον χαρακτήρα «@» και αμέσως μετά το όνομα της εικόνας.

Στην πρώτη περίπτωση το στοιχείο «asset» αντικαθίσταται με την ίδια την εικόνα SVG ενώ στη δεύτερη περίπτωση η εικόνα μορφοποιείται κατάλληλα και μπαίνει σαν «background-image». Και οι δύο επιλογές έχουν πλεονεκτήματα και μειονεκτήματα. Τα πλεονεκτήματα της πρώτης επιλογής είναι ότι η εικόνα αποτελεί στοιχείο HTML και υπάρχει δυνατότητα περεταίρω επεξεργασίας μέσω JavaScript ή CSS. Ως προς τα μειονεκτήματα, πρέπει να υπολογιστεί ότι περιέχεται η εικόνα κατά την επεξεργασία του πατρικού στοιχείου (αν αντικατασταθεί ο εσωτερικός HTML κώδικας, η εικόνα θα χαθεί). Τα πλεονεκτήματα και τα μειονεκτήματα της δεύτερης επιλογής φόρτωσης εικόνας είναι ακριβώς τα αντίθετα, δηλαδή δεν υπάρχει πρόβλημα κατά την αντικατάσταση του εσωτερικού κώδικα HTML του στοιχείου, αλλά δεν υπάρχει και η επιλογή της δυναμικής επεξεργασίας της εικόνας.

Παρακάτω παρουσιάζεται ο κώδικας του asset loader:

```
let x = 'xmlns="http://www.w3.org/2000/svg"';
window.loadAssets = html => {
  for (const tag in assetCollection) {
    let index, j, svg = assetCollection[tag];
    if (svg.indexOf(x) == -1) svg = '<svg ' + x + svg.substring(4);
    while ((index = html.indexOf('@' + tag)) != -1) {
      j = index - 1;
      while (html.charAt(j) != '>') j--;
      html = html.substring(0, j) + ' style="background-
image:url(' + btoa(svg) + ');">' + html.substring(j +
1, index) + html.substring(index + tag.length + 1);
    }
  }
  let searchStart = '<asset';
  let searchEnd = '</asset>';
  let index = 0, index2, data;
  while ( (index = html.indexOf(searchStart, index)) != -1 ) {
    let nextChar = html.charAt(index + searchStart.length);
    if (nextChar != ' ' && nextChar != '>') {
      index += searchStart.length + 2;
      continue;
    }
    index2 = html.indexOf(searchEnd);
    data = html.substring(index + searchStart.length, index2).split('>');
    if (assetCollection[data[1]] === undefined) throw new Error('Asset "' +
data[1] + '" does not exist');
    let svg = '<svg ' + data[0] + assetCollection[data[1]].substring(4);
    html = html.substring(0, index) + svg + html.substring(index2 +
searchEnd.length);
    index += svg.length;
  }
  return html;
};
```

Απόσπασμα Κώδικα 4.18: Υλοποίηση assetLoader

Το αντικείμενο «assetCollection» αποτελεί απλό JavaScript αντικείμενο το οποίο περιέχει τον κώδικα των εικόνων:

```
const assetCollection = {
  logo: require('./logo.svg'),
  navServer: require('./navServer.svg'),
  navCTree: require('./navCTree.svg'),
  navMetrics: require('./navMetrics.svg'),
  navLogout: require('./navLogout.svg'),
  navUser: require('./navUser.svg'),
```

```
folder: require('./folder.svg'),
edit: require('./edit.svg'),
arrow: require('./arrow.svg'),
//...
```

Απόσπασμα Κώδικα 4.19: Συλλογή εικόνων «assetCollection»

4.5.3 Component

Είναι γνωστό ότι όσο περισσότερα HTML στοιχεία βρίσκονται στο DOM κάθε δεδομένη στιγμή, τόσο πιο αργή θα είναι η απόκριση της εφαρμογής και μεγαλύτερη η χρήση της μνήμης της. Για αυτό το λόγο δημιουργήθηκε η κλάση «Component», η οποία παρέχει ένα επίπεδο αφαίρεσης (abstraction layer), δημιουργεί δυναμικά στοιχεία HTML και τα χειρίζεται ανάλογα με την κατάστασή στην οποία βρίσκονται. Επίσης, με τη βοήθεια αυτού του επίπεδου αφαίρεσης γίνεται πολύ ευκολότερος ο χειρισμός των στοιχείων στην περίπτωση που εμφανίζονται ή εξαφανίζονται από το DOM με χρήση μεταβάσεων (transitions). Αυτό οφείλεται στο γεγονός ότι η κλάση χειρίζεται όλες τις περιπτώσεις καταστάσεων και δεν επιτρέπει την εμφάνιση συνθηκών ανταγωνισμού (race condition). Η κλάση υποστηρίζει προαιρετικούς χειριστές συμβάντων (event handlers) για όλες της αλλαγές καταστάσεων του χειριζόμενου στοιχείου. Οι χειριστές συμβάντων είναι οι ακόλουθοι:

- beforeOpen: Καλείται ακριβώς πριν την δημιουργία του στοιχείου
- onCreate: Καλείται κατά την δημιουργία του αντικειμένου.
- onOpening: Καλείται κατά το άνοιγμα του αντικειμένου.
- onOpen: Καλείται κατά το τέλος ανοίγματος του αντικειμένου.
- beforeClose: Καλείται πριν την αρχή κλεισίματος του αντικειμένου.
- onClosing: Καλείται κατά το κλείσιμο του αντικειμένου
- beforeRemove: Καλείται πριν την αφαίρεση του αντικειμένου.
- onClose: Καλείται μετά την αφαίρεση του αντικειμένου.
- onFreeze: Καλείται κατά το «πάγωμα» του αντικειμένου.
- onUnFreeze: Καλείται κατά το «ξεπάγωμα» του αντικειμένου.

Η «παγωμένη» κατάσταση του αντικειμένου είναι μια επιπρόσθετη κατάσταση στην οποία μπορεί να βρίσκεται το αντικείμενο και χρησιμοποιείται για όσο το frontend βρίσκεται αποσυνδεμένο από το backend.

Οι μέθοδοι οι οποίες παρέχονται από την κλάση, είναι οι ακόλουθες:

- open: Άνοιγμα στοιχείου.
- close: Κλείσιμο στοιχείου.
- toggle: Μεταβολή κατάστασης στοιχείου στην αντίθετη.
- endTransitions: Τερματισμός τυχών μεταβάσεων στοιχείου.
- freeze: Πάγωμα στοιχείου.
- unfreeze: Ξεπάγωμα στοιχείου.
- destroy: Διαγραφή του στοιχείου (δεν μπορεί να ξανανοίξει).

Για την αρχικοποίηση της κλάσης, οι μόνες αναγκαστικές παράμετροι που χρειάζονται είναι το «html» και το «id».

Όλα τα στοιχεία του frontend, τα οποία δημιουργούνται δυναμικά έχουν αρχικοποιηθεί και χειρίζονται από την κλάση Component. Οι τύποι των στοιχείων οι οποίοι χειρίζονται με την κλάση είναι οι ακόλουθοι:

- nav bar item wrapper
- workplaces
- filesystems
- users drop-down
- modals
- toasts

Τα workplaces αποτελούν στοιχεία στα οποία θα προσαρτιούνται οι προς επεξεργασία οντότητες του συστήματος οι οποίες έχουν δικό τους παράθυρο (servers, ΔΚ, metrics). Παρακάτω παρατηρούμε πώς δημιουργείται ένα workplace με τη χρήση της κλάσης Component:

```
const workplace = new Component({
  html: loadAssets(require('./index.pug')),
```

```
    id: config.id + '.workplace',
    parent: config.parent,

    beforeOpen: config.beforeOpen,
    onCreate: function({ el }) {
      el.addClass(config.id);
      if (config.onCreate) config.onCreate.apply(null, arguments);
    },
    onOpening: config.onOpening,
    onOpen: config.onOpen,
    beforeClose: config.beforeClose,
    onClosing: function ({ frozen }) {
      if (!frozen) fs.close();
      if (config.onClosing) config.onClosing.apply(null, arguments);
    },
    beforeRemove: config.beforeRemove,
    onClose: config.onClose,
    onFreeze: config.onFreeze,
    onUnFreeze: config.onUnFreeze,
    autoFreezeState: state
  });
```

Απόσπασμα Κώδικα 4.20: Δημιουργία workplace

Στο αντικείμενο config αναφέρονται όλες οι εξειδικευμένες ιδιότητες και χειριστές συμβάντων οι οποίοι θα χρησιμοποιηθούν για την δημιουργία του συγκεκριμένου workplace. Το «html» και το «id» είναι οι μόνες αναγκαστικές παράμετροι οι οποίες απαιτούνται για την δημιουργία ενός Component. Η παράμετρος html μπορεί να είναι κώδικας HTML τύπου string ή μπορεί να είναι αναφορά σε αρχικοποιημένο στοιχείο HTML. Το id είναι πάντα τύπου string και πρέπει να είναι διαφορετικό για κάθε Component.

4.5.4 BackendSubscriber

Το BackendSubscriber είναι κλάση η οποία διευκολύνει στο frontend την εγγραφή και τη διαχείρισή της, σε οντότητες του backend συστήματος. Όταν πραγματοποιείται μια εγγραφή σε μια οντότητα, τότε το frontend ενημερώνεται για κάθε αλλαγή που γίνεται και την αφορά.

Για να πραγματοποιήσουμε μια εγγραφή χρησιμοποιώντας την κλάση BackendSubscriber, θα χρειαστεί πρώτα να έχουμε αρχικοποιήσει μια μεταβλητή ή σταθερά με την κλάση και στη συνέχεια να καλέσουμε τη μέθοδο «sub» με παράμετρο το αντικείμενο επιλογών. Το αντικείμενο επιλογών είναι απαραίτητο να περιέχει τις εξής παραμέτρους:

- wsSubEvent: Συμβάν εγγραφής οντότητας.
- wsSubData: Δεδομένα αποστολής για εγγραφή οντότητας.
- wsSubCb: Επανάκληση αποτελέσματος εγγραφής.
- subEvent: Συμβάν τοπικής εγγραφής οντότητας (state.pub/sub)
- onMessage: Επανάκληση ενημέρωσης οντότητας.
- wsUnsubEvent: Συμβάν απεγγραφής οντότητας.
- wsUnsubData: Δεδομένα αποστολής για απεγγραφή οντότητας.

Όταν πραγματοποιηθεί η εγγραφή, θα κληθεί η συνάρτηση «wsSubCb» με δεύτερη παράμετρο τα δεδομένα της οντότητας. Όλες οι αλλαγές ή ενημερώσεις οι οποίες αφορούν την οντότητα θα σταλούν στη συνάρτηση onMessage για κατάλληλο χειρισμό. Η απεγγραφή από την οντότητα πραγματοποιείται με τη μέθοδο «unsub» της κλάσης. Παρακάτω παρατηρούμε τον κώδικα που είναι υπεύθυνος για την εγγραφή της οντότητας server.

```
const subber = new BackendSubscriber();
const onServerOpen = id => {
  subber.sub({
    wsSubEvent: _backendToken(servers.item.get),
    wsSubData: { id },
    wsSubCb: (err, server) => {
      if (err) return state.toast(err, 'error');
      path.set(server.path, server.pathIds, id);
      placeServer(id, server);
    },
    subEvent: _backendToken(servers.item) + ':' + id,
    onMessage: event => {
      switch (event.action) {
        case 'activation state':
          setActivationState(event.active);
          break;
```

```
          case 'connection status':
            setControls(event.controls);
            setConnectionStatus(event.connected);
            break;
          case 'connection type':
            setConnectionType(event.connectionType);
            break;
          case 'nodeId update':
            setNodeId(event.nodeId);
            break;
          case 'connection info':
            setConnectionInfo(event.connectionInfo, true);
            break;
          case 'delete':
            wp.remove();
            break;
          case 'pathRename':
            path.rename(event.id, event.name);
            break;
          case 'pathMove':
            path.move(event.path, event.names);
            break;
          default:
            throw new Error('Unknown action');
        };
      },
      wsUnsubEvent: _backendToken(servers.item.unsub),
      wsUnsubData: { id }
    });
  };
```

Απόσπασμα Κώδικα 4.21: Παράδειγμα χρήσης BackendSubscriber

Οι σημαντικότερες μέθοδοι της κλάσης BackendSubscriber είναι οι ακόλουθες:

```
const subscriptions = new Map();
let maxId = -1;
const freeIds = new Set();

function getId() {
  const obj = freeIds.values().next();
  let id;
  if (obj.done) id = ++maxId;
  else {
    id = obj.value;
    freeIds.delete(id);
  }
```

```
  return id;
}
function freeId(id) {
  if (id == maxId) maxId--;
  else freeIds.add(id);
}

module.exports = state => function BackendSubscriber() {
const me = this;
  me.sub = ({ wsSubEvent, wsSubData, wsSubCb, subEvent, onMessage, wsUnsubEvent,
wsUnsubData }) => {
    if (me.s) me.unsub();
    me.s = {
      id: getId(),
      subEvent,
      cancelWsCallback: state.ss({
        event: wsSubEvent,
        data: wsSubData,
        cb: function (err, res) {
          if (!me.s) return;  // Subscription already removed
          delete me.s.cancelWsCallback;
          wsSubCb.apply(null, arguments);
        }
      }),
      localSubscription: state.sub(subEvent, function() {
        if (this == me.s.id) return;
        onMessage.apply(null, arguments);
      }),
      wsUnsubEvent,
      wsUnsubData
    };
    if (!subscriptions.has(subEvent)) subscriptions.set(subEvent, new Set());
    subscriptions.get(subEvent).add(me.s);
    return me;
  };

  me.unsub = (returnIfNotSubscribed) => {
    if (!me.s) return;
    if (me.s.cancelWsCallback) me.s.cancelWsCallback();
    const subscriptionCollection = subscriptions.get(me.s.subEvent);
    if (!subscriptionCollection) throw new Error('Subscription collection does
not exist');
    if (!subscriptionCollection.delete(me.s)) throw new Error('Subscription does
not exist');
    if (!subscriptionCollection.size) {
```

```
    state.ss({
      event: me.s.wsUnsubEvent,
      data: me.s.wsUnsubData,
      returnIfWsDisconnected: true
    });
    subscriptions.delete(me.s.subEvent);
  }
  state.unsub(me.s.localSubscription);
  freeId(me.s.id);
  delete me.s;
  return me;
};
```

Απόσπασμα Κώδικα 4.22: Υλοποίηση σημαντικότερων μεθόδων του BackendSubscriber

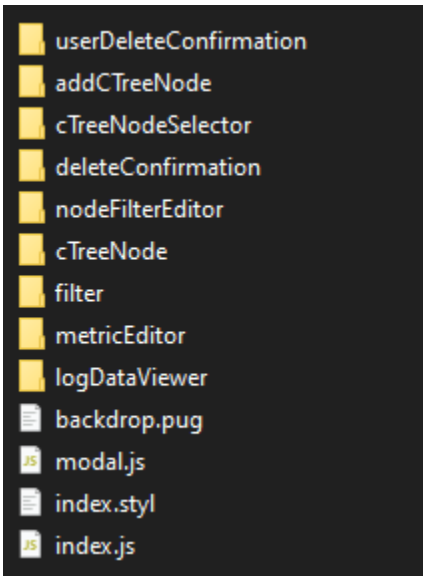
4.5.5 Modals

Τα modals είναι αναδυόμενα παράθυρα τα οποία εμφανίζονται στην κορυφή της εφαρμογής και απαιτούν από τον χρήστη να αλληλεπιδράσει μαζί τους προτού μπορέσει να συνεχίσει να χρησιμοποιεί την υπόλοιπη εφαρμογή. Τα modals αποτελούν επέκταση της κλάσης «Component» και ο τρόπος δημιουργίας τους είναι ο ακόλουθος:

```
Modal({
  html: loadAssets(require('./index.pug')),
  id: '#deleteConfirmationModal',
  onOpening: ({el, close}) => {},
  onClosing: ({el}) => {},
  //... supports all Component event handlers
});
```

Απόσπασμα Κώδικα 4.23: Δημιουργία Modal

Όλος ο κώδικας των modals της εφαρμογής βρίσκεται στον φάκελο «modals».



Εικόνα 4.1: Αρχεία των Modals

Το κάθε modal περιέχει τον κώδικα υλοποίησης του, μαζί με τα pug και styl αρχεία. Από το αρχείο index.js πραγματοποιείται η αναφορά των modals στο state object της εφαρμογής όπως φαίνεται παρακάτω:

```
export default function(state) {
  const Modal = require('./modal')(state);
  state.modals = {
    deleteConfirmation: require('./deleteConfirmation')(state, Modal),
    userDeleteConfirmation: require('./userDeleteConfirmation')(state, Modal),
    cTreeNode: require('./cTreeNode')(state, Modal),
    logDataViewer: require('./logDataViewer')(state, Modal),
    addCTreeNode: require('./addCTreeNode')(state, Modal),
    cTreeNodeSelector: require('./cTreeNodeSelector')(state, Modal),
    filter: require('./filter')(state, Modal),
    nodeFilterEditor: require('./nodeFilterEditor')(state, Modal),
    metricEditor: require('./metricEditor')(state, Modal),
  };
};
```

Απόσπασμα Κώδικα 4.24: Αναφορά Modals στο state object

Ο κώδικας υλοποίησης της γενικής κλάσης «Modal» βρίσκεται στο αρχείο «modal.js» και τα βασικά σημεία του κώδικα είναι τα παρακάτω:

```
const backdrop = require('./backdrop.pug');
```

Απόσπασμα Κώδικα 4.25: Δημιουργία αναφοράς κώδικα παρασκηνίου Modal

Ανάθεση χειριστή συμβάντος ανοίγματος γενικού Component, ανάθεση click ειδικού κουμπιού κλεισίματος.

```
options.onOpening = (me, e) => {
    me.el.before($('#backdrop').addClass('open'));
    me.el.find('.close').click(e => {
      if (me.el.find('.close').hasDescendant(e.target))
state.openModals[state.openModals.length - 1].close();
    });
    if (onOpening) onOpening(me, e);
  };
```

Απόσπασμα Κώδικα 4.26: Χειριστή συμβάντος ανοίγματος του Modal

Ανάθεση χειριστή συμβάντος αρχής κλεισίματος modal, αφαίρεση modal από το state.

```
  options.onClosing = (me, e) => {
    for (let i=0; i<state.openModals.length; i++) {
      if (state.openModals[i].id == options.id) {
        state.openModals.splice(i, 1);
        break;
      }
    }
    me.el.find('.close').offClick();
    const backdrop = $('#backdrop');
    if (state.openModals.length == 0) backdrop.offClick().removeClass('open');
    else state.openModals[state.openModals.length - 1].el.before(backdrop);
    if (onClosing) onClosing(me, e);
  };
```

Απόσπασμα Κώδικα 4.227: Χειριστής συμβάντος αρχής κλεισίματος Modal

Ανάθεση χειριστή συμβάντος ολοκλήρωσης κλεισίματος modal, αφαίρεση παρασκηνίου στην περίπτωση που δεν υπάρχει άλλο ανοιχτό modal.

```
  options.onClose = (me, e) => {
    if (state.openModals.length == 0) {
      const backgdrop = $('#backdrop');
      if (backgdrop) backgdrop.remove();
    }
    if (onClose) onClose(me, e);
  };
```

Απόσπασμα Κώδικα 4.28: Χειριστής συμβάντος κλεισίματος Modal

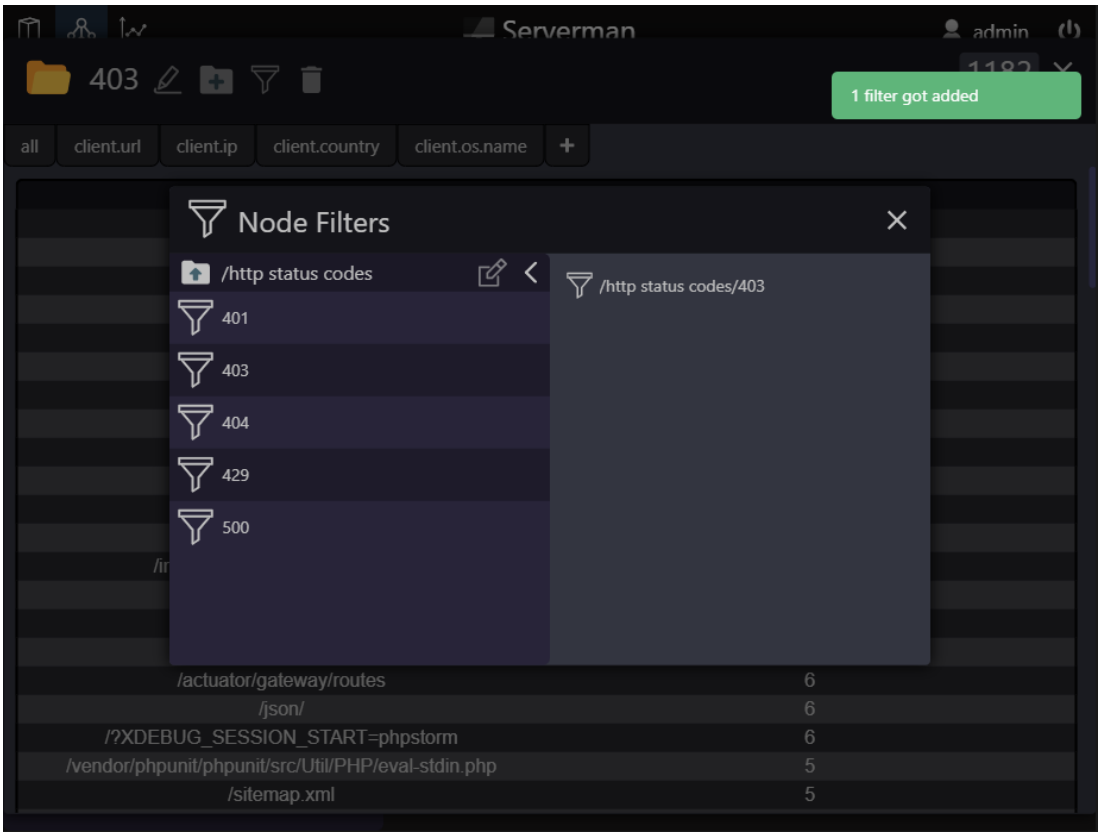
Αρχικοποίηση Component, προσθήκη backdrop στο DOM, ανάθεση χειριστή συμβάντος click του backdrop, άνοιγμα modal και προσθήκη αναφοράς του στο state object.

```
  let modal = new Component(options);
  if (state.openModals.length == 0) {
    if (!$('#backdrop')) state.body.add(backdrop);
    $('#backdrop').click(() => state.openModals[state.openModals.length -
1].close());
  }
  modal.open();
  state.openModals.push(modal);
  return modal;
```

Απόσπασμα Κώδικα 4.29: Αρχικοποίηση Modal

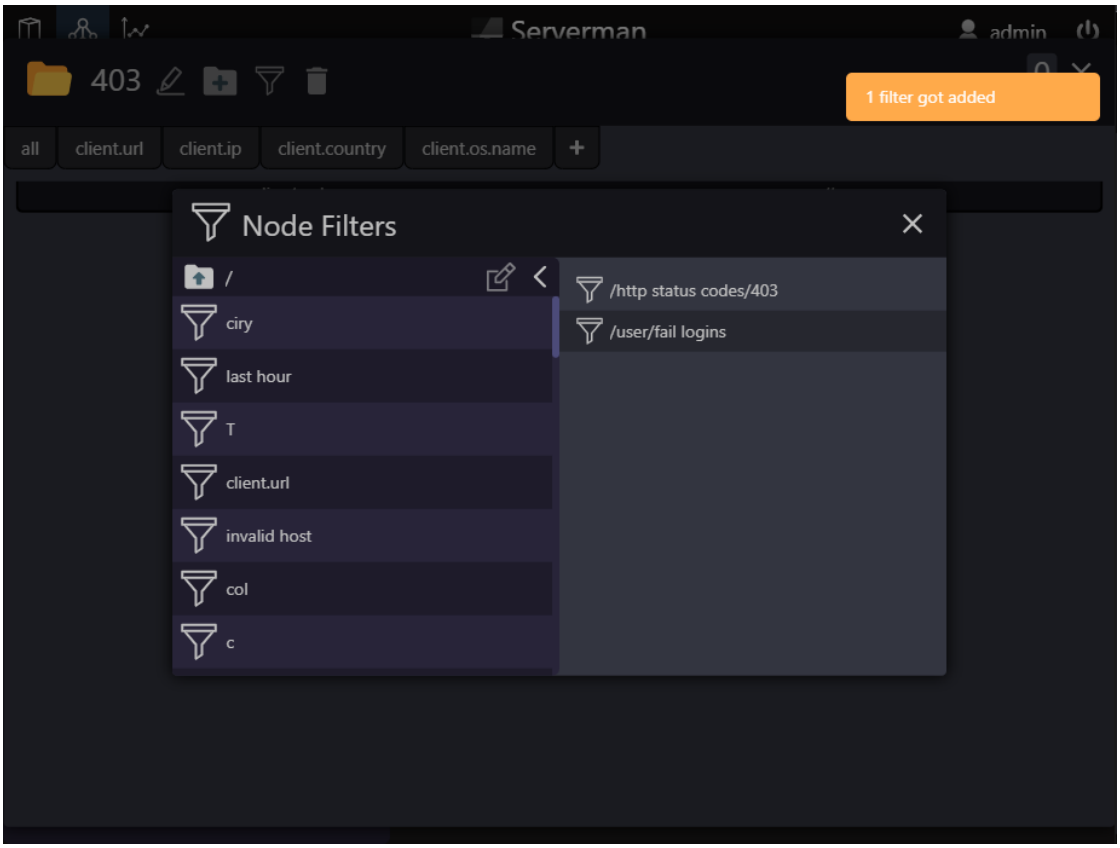
4.5.6 Toasts

Τα toasts είναι μικρά μηνύματα που εμφανίζονται για μικρό χρονικό διάστημα στην οθόνη, συνήθως σε κάποια γωνία. Τα toasts χρησιμοποιούνται για την παροχή σχολίων στον χρήστη μετά την ολοκλήρωση μίας ενέργειας ή για την ενημέρωση του χρήστη στην περίπτωση αλλαγής κατάστασης της οντότητας που είναι εγγεγραμμένος. Παρακάτω παρατηρούμε δυο παραδείγματα χρησιμότητας toasts:



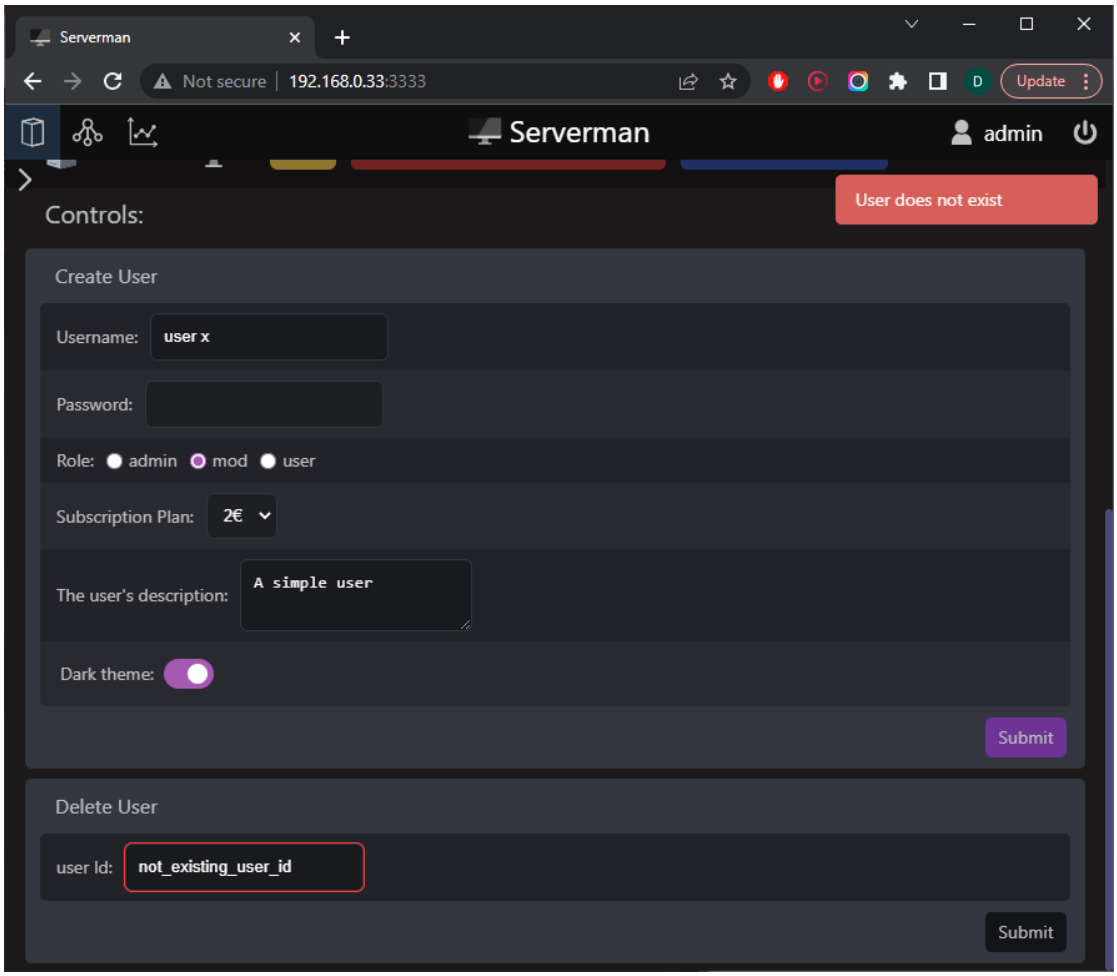
Εικόνα 4.2: Toast επιτυχίας (success)

Στην παραπάνω εικόνα ο ίδιος ο χρήστης πρόσθεσε το φίλτρο και εμφανίστηκε μήνυμα επιτυχίας (πράσινο).



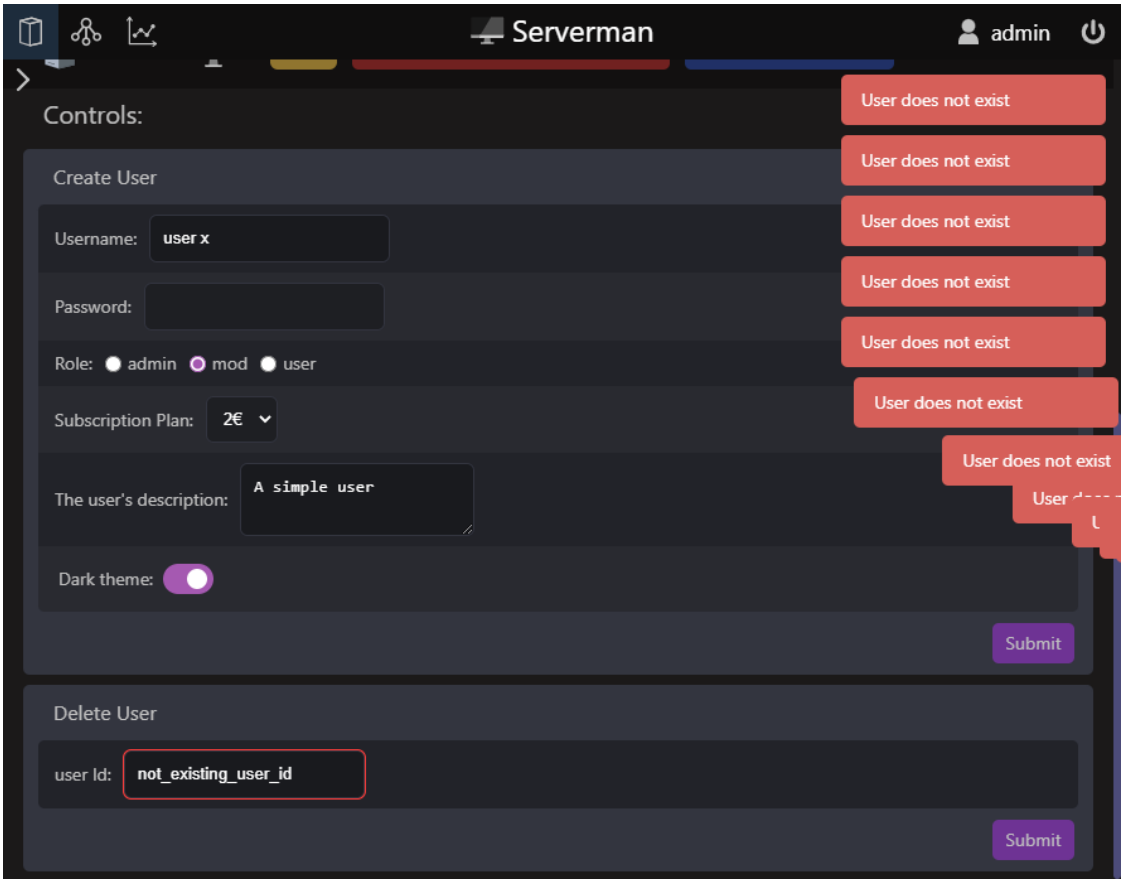
Εικόνα 4.3: Toast προειδοποίησης (warn)

Στο παραπάνω παράδειγμα ένας άλλος χρήστης πρόσθεσε νέο φίλτρο στον κόμβο και για αυτό τον λόγο εμφανίστηκε μήνυμα προσοχής (κίτρινο).



Εικόνα 4.4: Toast σφάλματος (error)

Στο παραπάνω παράδειγμα ο χρήστης προσπάθησε να διαγράψει έναν μη υπαρκτό χρήστη ενός απομακρυσμένου εξυπηρετητή. Το αποτέλεσμα ήταν να εμφανιστεί κόκκινο toast με το μήνυμα λάθους. Τα toasts μπορούν να εμφανίζουν παραπάνω από ένα μηνύματα τη φορά. Κάθε μήνυμα που εμφανίζεται «σπρώχνει» τα υπόλοιπα μηνύματα προς τα κάτω. Στην παρακάτω περίπτωση, ο χρήστης πάτησε πολλές φορές το κουμπί διαγραφής χρήστη.



Εικόνα 4.5: Πολλαπλά Toasts

Η κώδικας για την υλοποίηση της λειτουργίας toasts είναι ο παρακάτω:

```
  state.toast = function(text, level = 'info') {
    const toast = domify(require('./toast.pug'), {'.toast':
text}).addClass(level);
    toastContainer.el.prepend(toast);
    requestAnimationFrame(() => {
      requestAnimationFrame(() => {
        toast.removeClass('closed');
        setTimeout(() => {
          toast.addClass('closed').tEnd(toast.remove);
        }, 2000);
      });
    });
  };
```

Απόσπασμα Κώδικα 4.30: Υλοποίηση λειτουργίας toast

Αρχικά τα toasts προστίθενται στο DOM. Οι μέθοδοι «requestAnimationFrame» χρησιμοποιούνται για την αφαίρεση της κλάσης «closed» μετά την πρόσθεση των toasts στο DOM. Τέλος, δημιουργείται ένα timeout το οποίο θα ξαναπροσθέσει την κλάση closed και αφού τελειώσει η μετάβαση, θα αφαιρέσει το toast από το DOM.

Ο κώδικας stylus που χρησιμοποιήθηκε είναι ο ακόλουθος:

```
#toastContainer
  width 210px
  margin 12px
  right 0
  position fixed
  z-index 999
  .toast
    border-radius 5px
    color white
    padding 10px 16px
    transition .8s ease
    position relative
    right 0
    margin-bottom 8px
    overflow-wrap break-word
  .info
    background #73b573
  .warn
    background #f9a937
  .error
    background #ca5e58
  .closed
    right -222px
    margin-bottom -40px
```

Απόσπασμα Κώδικα 4.31: Κώδικας stylus toast

4.6 Σταθερές κώδικα και βελτιστοποιήσεις επικοινωνίας (frontend)

Ο frontend κώδικας που μετασχηματίζει το Webpack, είναι πολύ χρήσιμο να έχει αναφορά σε σταθερές του backend συστήματος. Για παράδειγμα όταν θέλουμε να μην γράφουμε τις ίδιες τιμές παραπάνω από μια φορά, το URL του backend ή ακόμα και μερικές από τις προκαθορισμένες τιμές τις οποίες θέτουμε στο καθολικό αρχείο config.js χρειάζεται να υπάρχουν στον frontend κώδικα. Στην ενότητα 3.6 αναφέρθηκε ότι η

λειτουργία «metadata» στην οποία δημιουργείται ένα JSON αρχείο από το backend κατά την εκκίνηση του συστήματος. Σε αυτή την ενότητα θα αναλύσουμε τον τρόπο με τον οποίο χρησιμοποιείται αυτό το αρχείο από το Webpack για να θέσει τις τιμές του αρχείου στα κατάλληλα σημεία του κώδικα.

Για την εκμετάλλευση των τιμών που βρίσκονται στο αρχείο metadata.json χρειάζεται να υπάρχει μέσα στον κώδικα κατάλληλη αναφορά. Για την επίτευξη αυτής την αναφοράς, χρησιμοποιείται η ένδειξη «_backendToken()». Είναι σημαντικό να κατανοήσουμε ότι η ένδειξη αυτή δεν αποτελεί κλήση μεθόδου και είναι απλά μια οδηγία για το Webpack. Η ένδειξη δεν χρησιμοποιείται στο χρόνο εκτέλεσης (runtime) της εφαρμογής, παρά μόνο κατά την διαδικασία μετασχηματισμού του κώδικα.

Ακολουθούν παραδείγματα χρήσης της ένδειξης και ο τελικός μετασχηματισμένος κώδικας.

Πραγματοποίηση σύνδεσης WebSocket, χρήση ένδειξης για ανάκτηση backend URL.

```
ws = new WebSocket(_backendToken(wsUrl));
```

Απόσπασμα Κώδικα 4.32: Παράδειγμα χρήσης _backendToken για ανάκτηση backend URL

Τελικός μετασχηματισμένος κώδικας:

```
ws = new WebSocket("wss://skulixlabs.zapto.org/s");
```

Απόσπασμα Κώδικα 4.33: Παράδειγμα χρήσης _backendToken, αποτέλεσμα ανάκτησης backend URL

Αρχικοποίηση navigation bar, χρήση ένδειξης για ανάθεση ονόματος εφαρμογής.

```
state.body.prepend(
  domify(
    loadAssets(require('./index.pug')),
    {'#logo': _backendToken(appName)}
  )
;
```

Απόσπασμα Κώδικα 4.34: Παράδειγμα χρήσης _backendToken για ανάκτηση ονόματος εφαρμογής

Αποτέλεσμα μετασχηματισμού κώδικα:

```
state.body.prepend(
  domify(
```

```
    loadAssets(__webpack_require__(/*! ./index.pug */
"./front/src/app/nav/index.pug")),
    {'#logo': "Serverman"}
  )
);
```

Απόσπασμα Κώδικα 4.35: Παράδειγμα χρήσης _backendToken, τελικός μετασχηματισμένος κώδικας ανάκτησης ονόματος εφαρμογής

Διαγραφή φακέλου ΔΚ. Χρήση ένδειξης για ανάθεση τιμής συμβάντος διαγραφής φακέλου ΔΚ.

```
      function deleteNode(deleteChildren = false) {
        state.ss({
          event: _backendToken(cTrees.nodes.delete),
          data: {
            id: cTreeId,
            cTreeNodeId,
            deleteChildren,
            subIdToSkip: subber.getId()
          },
          cb: (err, cTree) => {
            if (err) state.toast(err, 'error');
            close();
          }
        });
      }
```

Απόσπασμα Κώδικα 4.36: Παράδειγμα χρήσης _backendToken για ανάκτηση τιμής συμβάντος διαγραφής φακέλου ΔΚ

Αποτέλεσμα μετασχηματισμού κώδικα:

```
      function deleteNode(deleteChildren = false) {
        state.ss({
          event: 45,
          data: {
            id: cTreeId,
            cTreeNodeId,
            deleteChildren,
            subIdToSkip: subber.getId()
          },
          cb: (err, cTree) => {
            if (err) state.toast(err, 'error');
            close();
          }
        });
      }
```

Απόσπασμα Κώδικα 4.37: Παράδειγμα χρήσης _backendToken, τελικός μετασχηματισμένος κώδικας ανάκτησης τιμής συμβάντος διαγραφής φακέλου ΔΚ

Αποσύνδεση χρήστη συστήματος. Χρήση ένδειξης για ανάθεση τιμής συμβάντος αποσύνδεσης.

```
function logout() {
  state.ss({
    event: _backendToken(logout),
    cb: err => {
      if (err) {
        $('#logout').one('click', logout);
        return state.toast(err, 'error');
      }
      location.reload();
    }
  });
}
```

Απόσπασμα Κώδικα 4.38: Παράδειγμα χρήσης _backendToken για ανάκτηση τιμής συμβάντος logout

Αποτέλεσμα μετασχηματισμού κώδικα:

```
function logout() {
  state.ss({
    event: 16,
    cb: err => {
      if (err) {
        $('#logout').one('click', logout);
        return state.toast(err, 'error');
      }
      location.reload();
    }
  });
}
```

Απόσπασμα Κώδικα 4.39: Παράδειγμα χρήσης _backendToken, τελικός μετασχηματισμένος κώδικας ανάκτησης τιμής συμβάντος logout

Όπως παρατηρήσαμε στα παραπάνω παραδείγματα, με τη χρήση της ένδειξης, μπορεί να επιτευχθεί όχι μόνο μείωση μεγέθους του τελικού κώδικα, αλλά και μείωση μεγέθους των μηνυμάτων που στέλνονται κατά την επικοινωνία του WebSocket. Τα συμβάντα «cTrees.nodes.delete» και «logout» μετασχηματίζονται σε απλές αριθμητικές τιμές, οπότε μπορούν να χρησιμοποιηθούν περιφραστικά ονόματα συμβάντων αλλά και

άλλων σταθερών που κάνουν χρήση την ένδειξης, χωρίς να υπάρχει κάποιο μειονέκτημα.

Για την επίτευξη του μετασχηματισμού του κώδικα, έχει χρησιμοποιηθεί Webpack φορτωτής (loader) JavaScript αρχείων. Η ανάθεση του φορτωτή πραγματοποιείται στο configuration αρχείο του Webpack και έχει την παρακάτω μορφή:

```
                { test: /\.js$/,
                  loader: path.resolve(__dirname,
'./loaders/SocketEventsLoader.js'),
                  options: {
                    metatadaFilepath: path.resolve('./services/metadata.json')
                  }
                }
```

Απόσπασμα Κώδικα 4.4023: Ανάθεση Webpack φορτωτή JavaScript αρχείων

Για κάθε JavaScript αρχείο το οποίο περιλαμβάνεται στον κώδικα (με «require» ή «include»), θα περάσει από την παραπάνω μέθοδο φόρτωσης και θα μετασχηματιστεί κατάλληλα. Το αντικείμενο «options» θα περαστεί ως αναφορά στον κώδικα μετασχηματισμού και έχει την παράμετρο «metatadaFilepath», η οποία αναφέρει την απόλυτη (absolute) ή σχετική (relative) θέση του αρχείου metadata.json. Το αρχείο metadata.json έχει δημιουργηθεί από το backend και περιέχει όλες τις συσχετίσεις σταθερών που θα χρησιμοποιηθούν.

Ο κώδικας ο οποίος πραγματοποιεί αυτό το μετασχηματισμό στον τελικό κώδικα, είναι ο ακόλουθος:

```
const fs = require('fs');
const loaderUtils = require('loader-utils');

let keyword = '_backendToken(';
let endDelimiter = ')';
module.exports = function SocketEventsLoader(source) {
  const options = loaderUtils.getOptions(this);
  const metatadaFilepath = options.metatadaFilepath;
  this.addDependency(metatadaFilepath);
  let metadata = fs.readFileSync(metatadaFilepath, 'utf8');
  let tokens = JSON.parse(metadata);
  let errorString = '';
  let indx = source.indexOf(keyword);
  while (indx > -1) {
```

```
    let left = source.substring(0, indx);
    let right = source.substring(indx + keyword.length);
    let eventNameEndIndex = right.indexOf(endDelimiter);
    let eventName = right.substring(0, eventNameEndIndex);
    right = right.substring(eventNameEndIndex + endDelimiter.length);
    if ((typeof tokens[eventName]) === 'undefined')
      errorString += '\nSocket token "' + eventName + '" is not recognized. -> '
+ keyword + eventName + endDelimiter;
    source = left + tokens[eventName] + right;
    indx = source.indexOf(keyword);
  }
  if (errorString.length) throw new Error(errorString);
  return source;
};
```

Απόσπασμα Κώδικα 4.41: Υλοποίηση Webpack φορτωτή (loader) JavaScript αρχείων

Ο κώδικας δέχεται τον frontend κώδικα προς επεξεργασία (ανά αρχείο) με τη μορφή string και πραγματοποιεί απλή αντικατάσταση της εντολής «_backendToken([χχχ])» με την αντίστοιχη τιμή. Επιπλέων, θέτει εξάρτηση στο αρχείο έτσι ώστε να γνωρίζει το Webpack (αν βρίσκετε σε λειτουργία «watch») πότε το πραγματικό αρχείο κώδικα που επεξεργάζεται έχει τροποποιηθεί στο δίσκο.

ΚΕΦΑΛΑΙΟ 5

LIBRARY

5.1 Τεχνολογίες Ανάπτυξης

Η βιβλιοθήκη η οποία επιτρέπει την επικοινωνία του εξυπηρετητή του συστήματος με τους εξυπηρετητές που χειρίζεται, έχει αναπτυχθεί με την γλώσσα προγραμματισμού JavaScript. Η συγκεκριμένη γλώσσα προγραμματισμού επιλέχθηκε κυρίως για την ταχύτητα ανάπτυξης της γλώσσας αλλά και επειδή όλη η υπόλοιπη εφαρμογή έχει γραφτεί στη συγκεκριμένη γλώσσα προγραμματισμού. Τα πλεονεκτήματα της JavaScript και του Node.js, αναφέρονται στην ενότητα 3.1.

Για την αποθήκευση των μη απεσταλμένων δεδομένων χρησιμοποιείται μη πτητική ουρά FIFO (First-In-First-Out) και η υλοποίησή της κάνει χρήση του συστήματος αποθήκευσης κλειδιών-τιμών, LevelDB. Επιλέχθηκε η LevelDB επειδή παρέχει καλή απόδοση στις αναγνώσεις και στις εγγραφές των δεδομένων χρησιμοποιώντας ελάχιστους πόρους του συστήματος. Επίσης παρέχει ευελιξία στη μοντελοποίηση των δεδομένων και δεν χρειάζεται επιπρόσθετο εκτελέσιμο αρχείο όπως η SQLite.

5.2 Αρχικοποίηση βιβλιοθήκης

Ιδανικά, η βιβλιοθήκη αρχικοποιείται μαζί με την αρχικοποίηση του προγράμματος που τη χρησιμοποιεί. Είναι αναγκαίο να δοθεί αναφορά στο αντικείμενο «config» το οποίο περιέχει πληροφορίες σχετικές με τον τρόπο λειτουργίας της βιβλιοθήκης. Τα πεδία του αντικειμένου είναι τα ακόλουθα:

- **retryTimeMS:** (number) Χρόνος (ms) προσπάθειας επαναποστολής αποθηκευμένων αρχείων της ουράς FIFO.
- **maxNumberOfPendingDocuments:** (number) Μέγιστος αριθμός αποθηκευμένων αρχείων στην ουρά FIFO.
- **dbPath:** (string) Μονοπάτι αποθήκευσης αρχείων συστήματος LevelDB.
- **emptyQ:** (boolean) Μεταβλητή εκκαθάρισης ουράς.

- **connectionType:** (string) Τρόπος σύνδεσης στο backend.
- **domain:** (string) Όνομα/IP backend server.
- **port:** (number) Θύρα TCP backend server.
- **urlPath:** (string) Μονοπάτι URL αποστολής αρχείων καταγραφής (HTTP/HTTPS)
- **keepAliveMS:** (number) Χρόνος (ms) αποστολής πακέτου ping.
- **connectionRetryMS:** (number) Χρόνος (ms) προσπάθειας επανασύνδεσης στο backend.
- **myNodeId:** (string) Ταυτότητα τοπικού κόμβου.
- **maxCallbacksOut:** (number) Μέγιστος επιτρεπτό πλήθος επανακλήσεων (callbacks) προς το backend.
- **maxContentLengthOut:** (number) Μέγιστο επιτρεπτό πλήθος bytes δεδομένων πακέτου προς το backend.
- **callbackTimeoutMS:** (number) Μέγιστος χρόνος (ms) αναμονής επανάκλησης.
- **authTimeoutMS:** (number) Μέγιστος χρόνος διαδικασίας αυθεντικοποίησης.
- **maxConObjSize:** (number) Μέγιστο επιτρεπτό πλήθος bytes αντικειμένου αυθεντικοποίησης.
- **otherNodeId:** (string) Ταυτότητα απομακρυσμένου κόμβου.
- **maxOtherEventCount:** (number) Μέγιστος επιτρεπτό πλήθος μεθόδων απομακρυσμένου κόμβου.
- **maxCallbacksIn:** (number) Μέγιστο επιτρεπτό πλήθος επανακλήσεων απομακρυσμένου κόμβου.
- **maxContentLengthIn:** (number) Μέγιστο επιτρεπτό πλήθος bytes δεδομένων εισερχόμενου πακέτου.
- **controls:** (object) Περιγραφή μεθόδων προς το απομακρυσμένο άκρο.

Παρακάτω παρουσιάζεται παράδειγμα αντικειμένου config και σχολιασμός αντικειμένου controls.

```
const config = {
  retryTimeMS: 10000,
  maxNumberOfPendingDocuments: 9999,
  dbPath: 'pqdata',
  emptyQ: false,
```

```
connectionType:  'TCP',
domain:  'serverman.lan',
port:  3001,
// urlPath:  'report, // Valid only for HTTP/HTTPS
keepAliveMS: 3500,
connectionRetryMS: 3000,  // Valid only for TCP coms
myNodeId: 's1',
maxCallbacksOut: 100,
maxContentLengthOut: 1000000,
callbackTimeoutMS: 10000,
authTimeoutMS: 3000,
maxConObjSize: 500000,
otherNodeId: 'Serverman',
maxOtherEventCount: 300,
maxCallbacksIn: 100,
maxContentLengthIn: 1000000,
controls: {
  createUser: {
    title: 'Create User', // optional
    params: [
      {
        title: 'Username:', // optional
        element: 'textbox',  // required if remote displays form
        validation: 'user', // optional
        default: 'user x' // optional
      },
      {
        title: 'Password:',
        element: 'textbox',
        validation: 'pass'
      },
      {
        title: 'Role:',
        element: 'radio',
        values: ['admin', 'mod', 'user'],
        default: 'mod'
      },
      {
        title: 'Subscription Plan:',
        element: 'select',
        values: ['1€', '2€', '5€'],
        default: '2€'
      },
      {
        title: 'The user\'s description:',
```

```
          element: 'textarea',
          validation: 'description',
          default: 'A simple user'
        },
        {
          title: 'Dark theme:',
          element: 'checkbox',
          default: true
        }
      ],
      cb: true,
      handler: function (user, pass, role, subscriptionPlan, description,
darkThemed, cb) {
        const userId = users.create(...Array.from(arguments));
        return userId;
      }
    },
    deleteUser: {
      // title: 'Delete User',  // Example without title
      cb: true,
      params: [
        { // Example without title
          element: 'textbox'
        }
      ],
      handler: (userId, cb) => {
        if (!users.delete(userId)) {
          const e = new Error('User does not exist');
          e.forRemote = true;
          throw e;
        }
        cb(null);
      }
    }
  }
};
```

Απόσπασμα Κώδικα 5.1: Παράδειγμα αντικειμένου παραμετροποίησης (config) της βιβλιοθήκης

Το πεδίο «controls» περιέχει φόρμες όπου η κάθε φόρμα περιγράφει τα πεδία της και κρατάει αναφορά στον χειριστή (handler) των απεσταλμένων δεδομένων. Σημειώνεται ότι καμία ιδιότητα των παραμέτρων δεν είναι απαραίτητη στην περίπτωση που δεν θα υπάρξει προβολή φόρμας, και στην περίπτωση που υπάρξει προβολή φόρμας, απαραίτητη είναι μόνο η ιδιότητα «element».

Η βιβλιοθήκη αρχικοποιείται με τον τρόπο που παρατηρούμε στο παρακάτω απόσπασμα κώδικα.

```
    const { setRouteErrorHandling, report } =
        await require('./libs/servermanLib')(config);
```

Απόσπασμα Κώδικα 5.2: Αρχικοποίηση βιβλιοθήκης

Η βιβλιοθήκη επιστρέφει αντικείμενο JavaScript το οποίο περιέχει αναφορά στις μεθόδους «setRouteErrorHandling» και «report».

Η πρώτη μέθοδος αποτελεί χειριστή σφαλμάτων (error handling middleware) ο οποίος μπορεί να χρησιμοποιηθεί στην περίπτωση που το πρόγραμμα/εξυπηρετητής έχει σχεδιαστεί με βάση την αρχιτεκτονική «middleware» και χειρίζεται όλα τα σφάλματα τα οποία μπορεί να προκληθούν στους χειριστές (controllers) των «routes».

Η δεύτερη μέθοδος καλείται για να στείλει η βιβλιοθήκη ένα αρχείο καταγραφής στην εφαρμογή (backend). Στην επόμενη ενότητα θα δούμε παραδείγματα κλήσης αυτής της μεθόδου και εμφάνισης των αρχείων καταγραφής στην εφαρμογή.

5.3 Παραδείγματα χρήσης μεθόδου «report»

```
report({msg: 'Hello World!'});
```

Απόσπασμα Κώδικα 5.3: Απλό παράδειγμα report

Εικόνα 5.1: Απλό παράδειγμα report

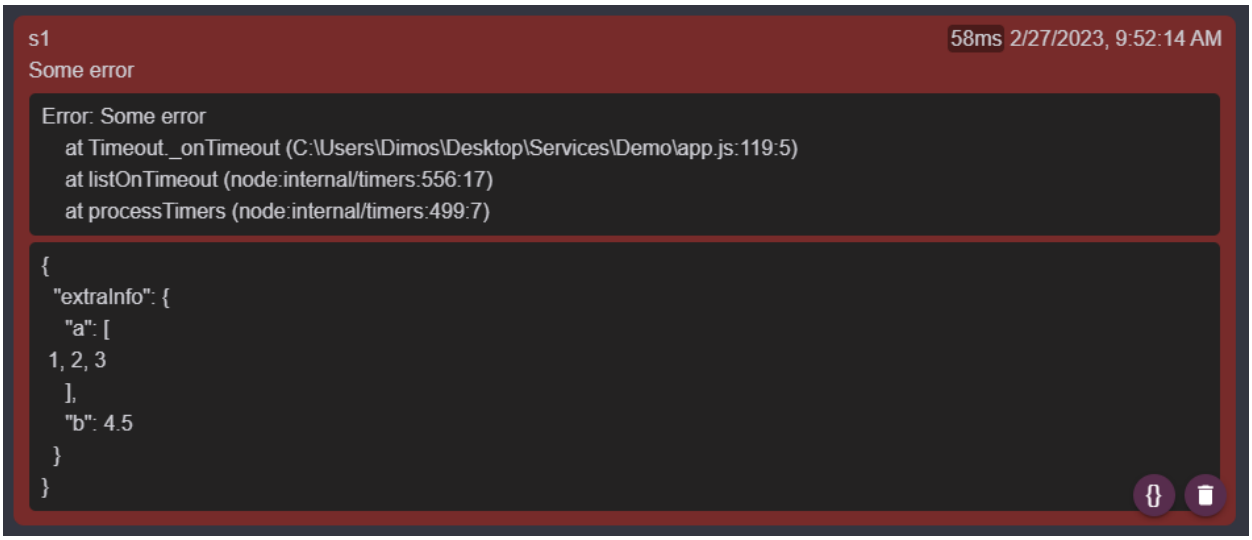
```
report({ message: 'Hello World!', level: 'info' });
```

Απόσπασμα Κώδικα 5.244: Παράδειγμα report με παράμετρο «level»

Εικόνα 5.2: Παράδειγμα report με παράμετρο «level»

```
report(new Error('Some error'), { level: 'error' },
  {
    extraInfo: {
      a: [ 1, 2, 3 ], b: 4.5
    }
  }
);
```

Απόσπασμα Κώδικα 525.5: Παράδειγμα report με παραπάνω από ένα αντικείμενα

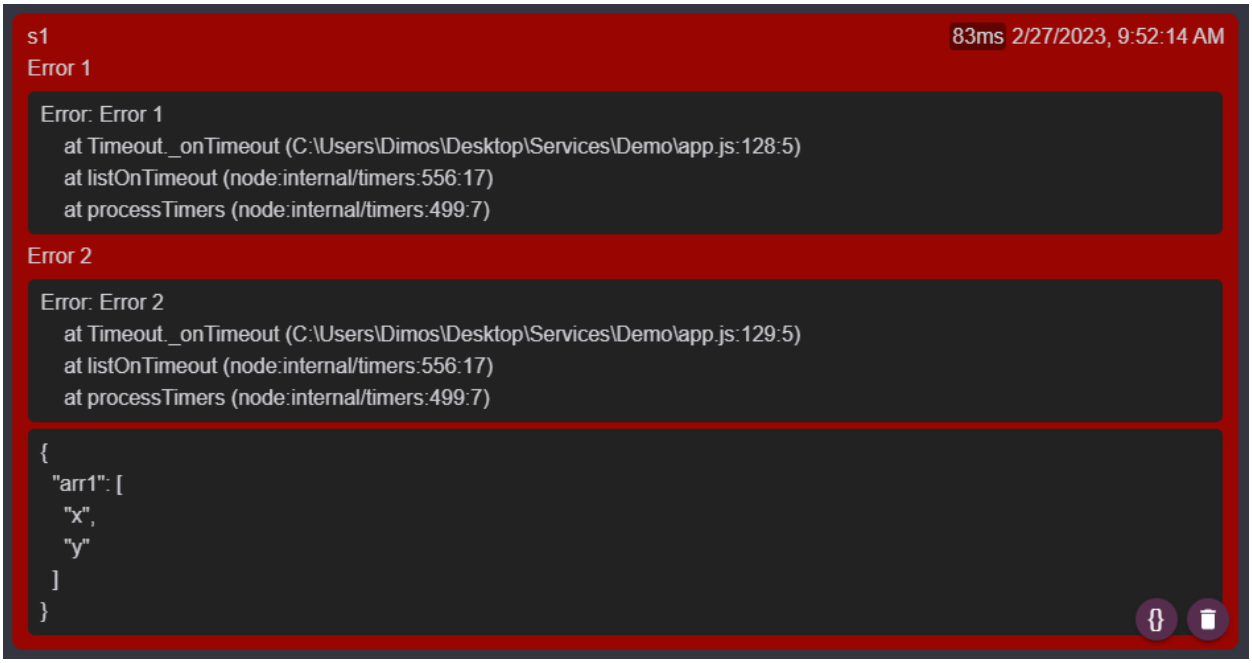


Εικόνα 5.3: Παράδειγμα report με παραπάνω από ένα αντικείμενα πληροφορίας

πληροφορίας

```
report([ new Error('Error 1'), new Error('Error 2'),
  { level: 'fatal' }, { arr1: ['x', 'y'] }
]);
```

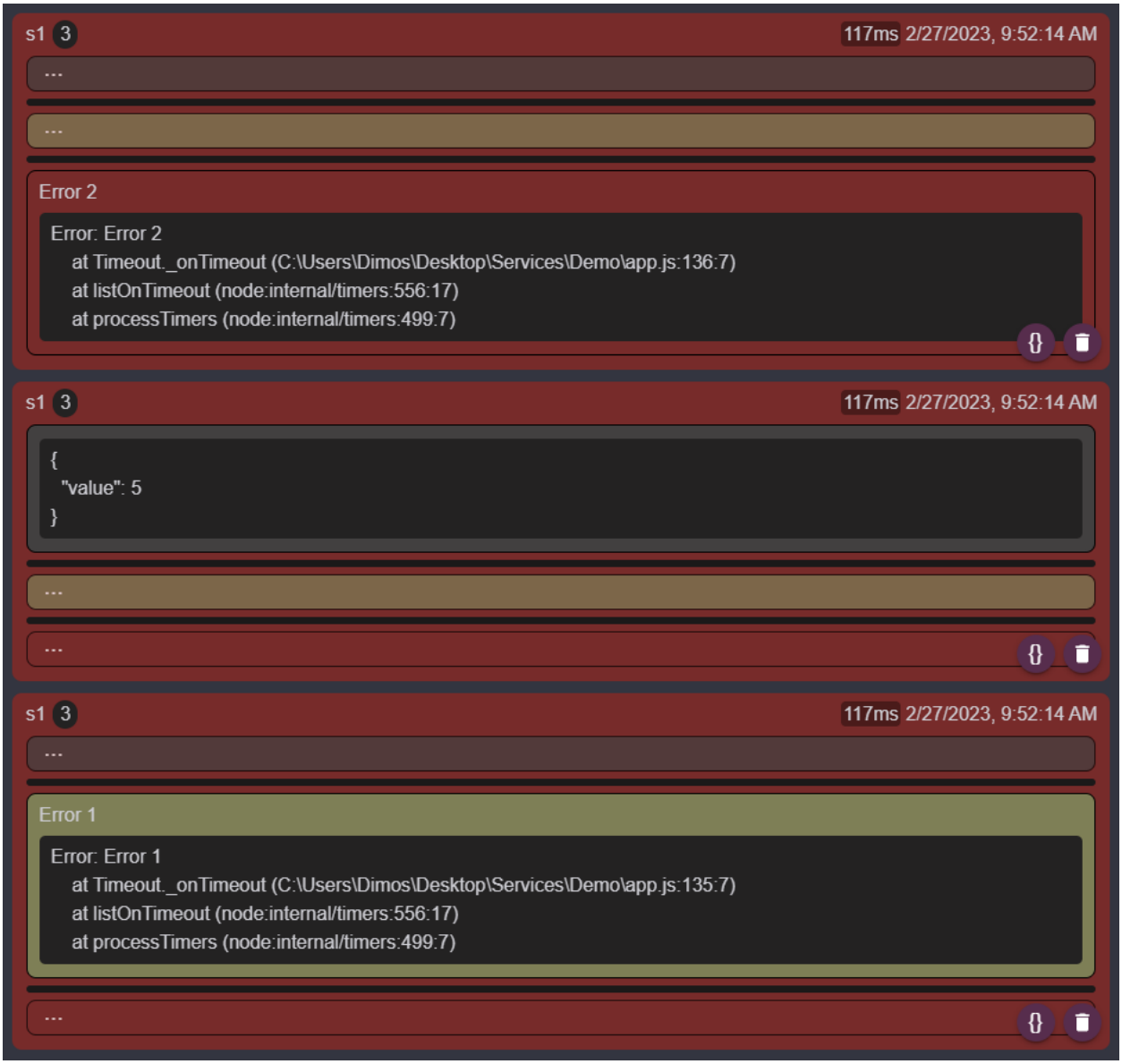
Απόσπασμα Κώδικα 5.266: Παράδειγμα report με παραπάνω από ένα αντικείμενα τύπου Error



Εικόνα 5.4: Παράδειγμα report με παραπάνω από ένα αντικείμενα τύπου Error

```
report([
  [ { value: 5 } ],
  [ new Error('Error 1'), {level: 'warn' } ],
  [ new Error('Error 2'), { level: 'error' }]
]);
```

Απόσπασμα Κώδικα 5.277: Παράδειγμα report με παραπάνω από ένα αρχεία καταγραφής



Εικόνα 5.5: Παράδειγμα report με παραπάνω από ένα αρχεία καταγραφής

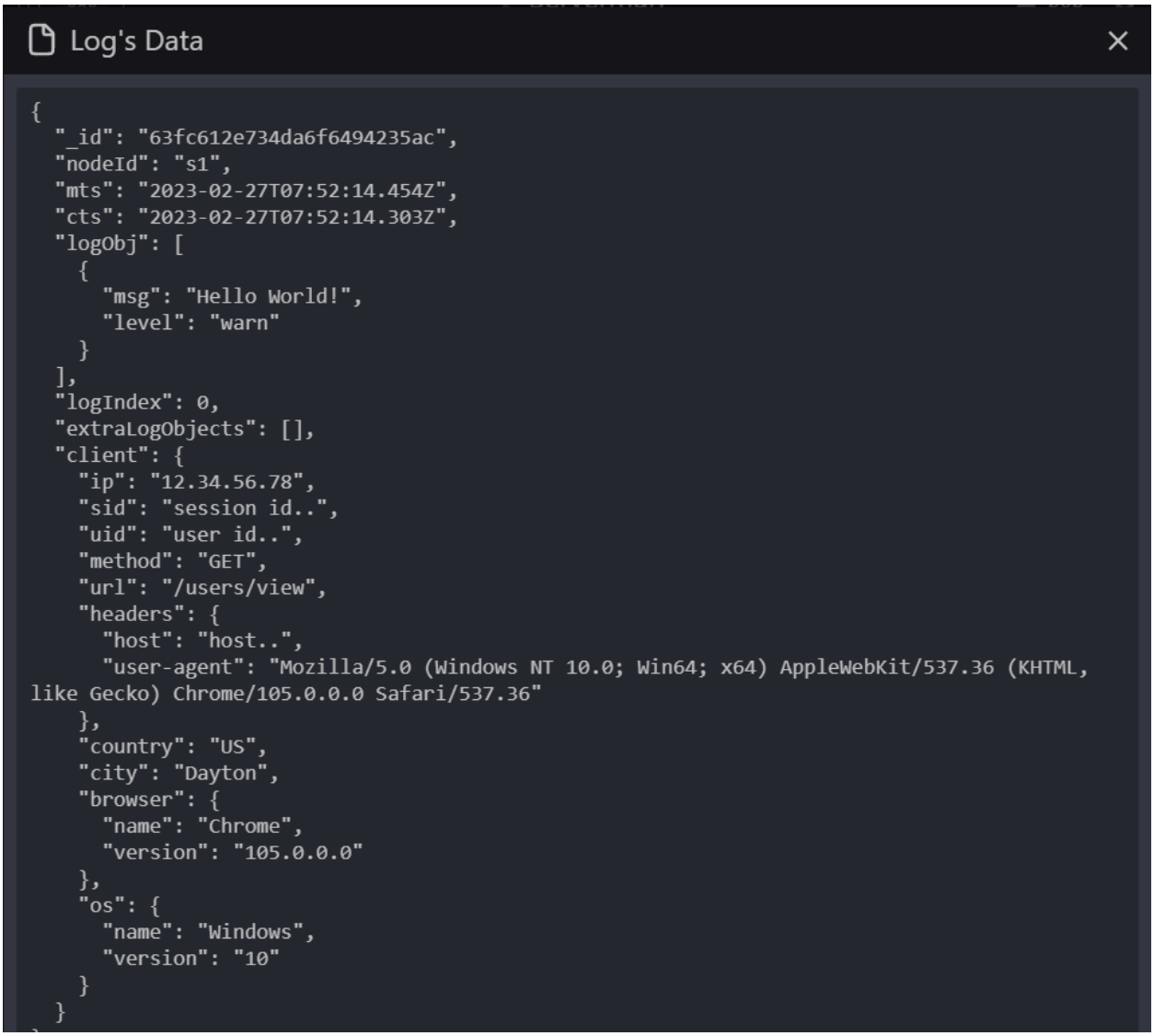
```
const context = {
  ip: '12.34.56.78',
  sid: 'session id..',
  uid: 'user id..',
  method: 'GET',
  url: '/users/view',
  headers: {
    host: 'host..',
```

```
      'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36'
      // ....
    }
  };

  report.call(context, {msg: 'Hello World!', level: 'warn'});
```

Απόσπασμα Κώδικα 5.828: Παράδειγμα report με αναφορά context

Εικόνα 5.6: Παράδειγμα report με αναφορά context



Εικόνα 5.7: Παράδειγμα report με αναφορά context, τα δεδομένα του αρχείου καταγραφής

```
report({
  item: 'RAM',
  freeMB: 697,
  usedPercent: 91.4,
  level: 'metric'
});
```

Απόσπασμα Κώδικα 5.929: Παράδειγμα report με αποστολή μετρικής

Εικόνα 5.8: Παράδειγμα report με αποστολή μετρικής

5.4 Μη πτητική ουρά FIFO

Η μη πτητική ουρά FIFO (First-In-First-Out), χρησιμοποιείτε για την αποθήκευση των μη απεσταλμένων δεδομένων προς το backend τηρώντας σειρά προτεραιότητας αποστολής. Η αρχικοποίηση της ουράς, φαίνεται παρακάτω.

```
const pq = new PersistentJobQueue({ retryTimeMS, maxNumberOfPendingDocuments,
dbPath, process: () => { throw new Error('Not connected.') } });
  if (emptyQ) await pq.clear();
```

Απόσπασμα Κώδικα 5.1030: Αρχικοποίηση μη πτητικής ουράς FIFO

Οι παράμετροι που δέχεται ο αρχικοποιητής της ουράς, αναφέρονται στην ενότητα 5.2 Αρχικοποίηση βιβλιοθήκης. Παρατηρούμε πως αμέσως μετά την αρχικοποίηση, πραγματοποιείται έλεγχος για το αν θα χρειαστεί να διαγραφτούνε όλα τα δεδομένα της ουράς.

Εσωτερικά, κατά την αρχικοποίηση της ουράς, πραγματοποιείται ανάκτηση πληροφοριών, όπως είναι το πρώτο και το τελευταίο Id των αρχείων που βρίσκονται αποθηκευμένα στη βάση. Εν συνεχεία, με τις παρακάτω εντολές, τίθεται η μέθοδος η

οποία θα επεξεργαστεί τα δεδομένα της ουράς και ξεκινάει η επεξεργασία των δεδομένων της.

```
pq.setProcess(send);
await pq.startRunner();
```

Απόσπασμα Κώδικα 5.1131: Ανάθεση μεθόδου επεξεργασίας δεδομένων ουράς και εκκίνηση λειτουργίας της

Στην περίπτωση που η μέθοδος επεξεργασίας αποτύχει, η ουρά θα αποθηκεύσει το δεδομένο στο δίσκο αν αυτό δεν είναι ήδη αποθηκευμένο και θα ξαναπροσπαθήσει να το επεξεργαστεί όταν περάσει ο χρόνος που έχει τεθεί κατά την αρχικοποίησή της.

```
 pq.process({ nodeId: myNodeId, cts: new Date(), client}, Array.from(arguments));
```

Απόσπασμα Κώδικα 5.12: Αποστολή δεδομένων προς την ουρά

Για την αποστολή δεδομένων προς την ουρά, χρησιμοποιείται η μέθοδος «process» η οποία προσθέτει τα δεδομένα στην πτητική μνήμη και καλεί τη μέθοδο «memRunner» και επιστρέφει ένα promise. Η memRunner με την σειρά της ελέγχει αν η ουρά περιέχει παλαιότερα δεδομένα ή είναι κενή. Στην περίπτωση που η ουρά είναι κενή, τα δεδομένα θα περάσουν κατευθείαν στην μέθοδο επεξεργασίας, ενώ στην περίπτωση που η ουρά περιέχει δεδομένα, το καινούργιο δεδομένο θα προστεθεί και αυτό στη βάση. Για να μη σταλεί κάποιο δεδομένο εκτός σειράς, τα δεδομένα τα οποία είναι να σταλούν, αποθηκεύονται προσωρινά στην πτητική μνήμη μέχρι να έρθει και σε αυτά η σειρά τους για να σταλούν. Παρακάτω παρουσιάζεται απόσπασμα κώδικα που περιγράφει τον τρόπο υλοποίησης αυτής της λειτουργίας:

```
  let oldestId = null;
  let lastId = null;
  let running = false;
  let runnerTimedOut = false;
  const getNextId = prevId => (prevId === null || prevId ===
maxNumberOfPendingDocuments - 1) ? 0 : prevId + 1;
  // ...
  const toProcessQueue = [];
  let memRunnerRunning = false;

  this.process = async function() {
    return new Promise((resolve, reject) => {
      toProcessQueue.push([Array.from(arguments), resolve, reject]);
      if (!memRunnerRunning) memRunner();
```

```
    });
  };

  async function memRunner() {
    memRunnerRunning = true;

    while (toProcessQueue.length) {
      const [args, resolve, reject] = toProcessQueue.shift();
      let nextId = null;
      if (oldestId === null) {
        try {
          await config.process(...args);
          resolve();
          continue;
        } catch(e) {}
        nextId = 0;
      } else {
        nextId = getNextId(lastId);
      }

      // Check if nextId exists in db, the id value has make a full circle and
      // thus the maximum allowed document count has been reached;
      if (nextId === oldestId) reject(new Error('Reached max allowed number of
saved documents.'));

      let batch = [
        { type: 'put', key: nextId, value: stringify(args) },
        { type: 'put', key: 'lastId', value: nextId }
      ];
      // If it is the first item, set also the oldestId
      if (oldestId === null) {
        batch.unshift({ type: 'put', key: 'oldestId', value: nextId });
        oldestId = nextId;
      }
      lastId = nextId;
      await db.batch(batch);
      if (!running) {
        running = true;
        runnerTimedOut = true;
        runnerTimeout = setTimeout(runner, retryTimeMS);
      }
    }
    memRunnerRunning = false;
  }
```

Απόσπασμα Κώδικα 5.13: Μέθοδος memRunner ουράς FIFO

Αν η επεξεργασία των δεδομένων αποτύχει, πριν από την αποθήκευση των δεδομένων στη βάση, πραγματοποιείται σειριοποίηση (serialization) των δεδομένων και αυτά προστίθεντο μαζί με επιπρόσθετη πληροφορία σχετική με τους δείκτες (id) των δεδομένων, σε μια συλλογή ενεργειών. Η συλλογή ενεργειών στέλνεται στη βάση για εκτέλεση και έπειτα θέτεται ένα timeout το οποίο με τη σειρά του θα καλέσει τη μέθοδο «runner». Η σειριοποίηση των δεδομένων είναι απαραίτητη επειδή η προεπιλεγμένη μέθοδος σειριοποίησης των δεδομένων της LevelDB παραλείπει τα μη απαριθμήσιμα (non-enumerable) πεδία των αντικειμένων.

Η μέθοδος «runner» προωθεί στη μέθοδο «runJob» τον δείκτη id του κάθε αποθηκευμένου δεδομένου στη βάση με σειρά προτεραιότητας. Η μέθοδος runJob ανακτά τα κατάλληλα δεδομένα και τα προωθεί για επεξεργασία. Στην περίπτωση που αποτύχει η επεξεργασία, θέτεται ένα timeout για την μέθοδο runner και σταματάει η εκτέλεση. Αν η επεξεργασία επιτύχει, δημιουργείται μια συλλογή ενεργειών στην οποία περιγράφεται η διαγραφή των επεξεργασμένων δεδομένων και η κατάλληλη επεξεργασία των δεικτών id. Τέλος, η συλλογή ενεργειών στέλνεται στη βάση για εκτέλεση. Παρακάτω παρουσιάζεται απόσπασμα κώδικα των μεθόδων «runner» και «runJob»:

```
const runner = async () => {
  if (running && !runnerTimedOut) return;
  running = true;
  runnerTimedOut = false;

  if (oldestId === null) throw new Error('Not initialized or empty database.');

  while (oldestId != lastId) {
    if (!await runJob(oldestId, false)) return;
    oldestId = getNextId(oldestId);
  }
  if (!await runJob(oldestId, true)) return;

  oldestId = lastId = null;
  running = false;
};

async function runJob(oldestId, isLast) {
  const id = oldestId.toString();
```

```
  const args = JSON.parse(await db.get(id));

  try {
    await config.process(...args);
  } catch(e) {
    runnerTimedOut = true;
    runnerTimeout = setTimeout(runner, retryTimeMS);
    return false;
  }

  let batch = [{ type: 'del', key: id }];

  if (isLast) {
    batch.push(
      { type: 'del', key: 'oldestId'},
      { type: 'del', key: 'lastId'}
    );
  } else {
    batch.push(
      { type: 'put', key: 'oldestId', value: getNextId(oldestId) }
    );
  }

  await db.batch(batch);
  return true;
}
```

Απόσπασμα Κώδικα 5.14: Μέθοδοι runner και runJob ουράς FIFO

5.5 Επικοινωνία μέσω HTTP

Η επικοινωνία μέσω HTTP είναι η πιο απλή στην υλοποίηση, αλλά έχει το μειονέκτημα ότι είναι πιο αργή επειδή πραγματοποιείτε σύνδεση με το backend κάθε φορά που στέλνεται κάποιο αρχείο καταγραφής και δεν επιτρέπετε η αποστολή εντολών από το backend. Ο κώδικας που επιτρέπει την επικοινωνία μέσω HTTP είναι ο ακόλουθος:

```
    const fetch = (await import('node-fetch')).default;
    const pingData = { nodeId: myNodeId };
    let pingTimeout = null;
```

```
      function updatePingTimeout() {
        if (pingTimeout) clearTimeout(pingTimeout);
        pingTimeout = setTimeout(() => send(pingData), keepAliveMS);
      }

      async function send() {
        updatePingTimeout();
        const data = stringify(Array.from(arguments));
        try {
          const response = await fetch(
          connectionType.toLowerCase() + '://' + domain + ':' + port + '/' +
urlPath,
            {method: 'POST', body: data, headers: {'Content-Type':
'application/json'}}
          );
          await response.text();
        } catch(e) {
          if (data === stringify([pingData])) return;
          e.data = data;
          throw e;
        }
      };

      pq.setProcess(send);
      await pq.startRunner();
      if (!pingTimeout) updatePingTimeout();
```

Απόσπασμα Κώδικα 5.1532: Υλοποίηση HTTP επικοινωνίας βιβλιοθήκης

Επιπρόσθετα, ο κώδικας υποστηρίζει αποστολή πακέτων «ping» τα οποία ενημερώνουν ότι το πρόγραμμα βρίσκετε σε λειτουργία και μπορεί να υπάρξει επικοινωνία σε περίπτωση που χρειαστεί να σταλεί αρχείο καταγραφής.

5.6 Επικοινωνία μέσω TCP

Η επικοινωνία μέσω TCP απαιτεί επιπρόσθετη υλοποίηση πρωτοκόλλου επικοινωνίας μεταξύ του χειριζόμενου προγράμματος/εξυπηρετητή και του backend. Για τον λόγο αυτό η υλοποίησή του είναι πιο περίπλοκη. Τα πλεονεκτήματα είναι ότι η επικοινωνία μπορεί να είναι λιγότερο βερμπαλιστική και πιο άμεση, αφού το πρωτόκολλο μπορεί να σχεδιαστεί ειδικά για την περίπτωση και υπάρχει ήδη σύνδεση μεταξύ των δύο άκρων. Παρακάτω παρουσιάζεται ο κώδικας που αρχικοποιεί την επικοινωνία μέσω TCP:

```
      function connect() {
        setupTCPClient(
          config,
          function onControlError(err) {
            report(err);
          },
          async function onConnection (remoteControls) {
            pq.setProcess(remoteControls.report || (() => { throw new
Error('Remote does not support this method.') }))
            await pq.startRunner();
          },
          function onDisconnection() {
            setTimeout(connect, connectionRetryMS);
          }
        );
      }
      connect();
```

Απόσπασμα Κώδικα 5.16: Αρχικοποίηση TCP επικοινωνίας βιβλιοθήκης

Η συνάρτηση «setupTCPClient» αρχικοποιεί μια καινούργια TCP σύνδεση και δέχεται ως παραμέτρους το αντικείμενο «config» της βιβλιοθήκης και τρείς χειριστές συμβάντων (event handlers) οι οποίοι είναι:

- onControlError: Καλείται στην περίπτωση σφαλμάτων σε τοπικό control handler.
- onConnection: Καλείται κατά την αρχκοποίηση σύνδεσης με το απομακρυσμένο άκρο.
- onDisconnection: Καλείται κατά την αποσύνδεση με το απομακρυσμένο άκρο.

Ο χειριστής του συμβάντος σύνδεσης (onConnection) δέχεται σαν παράμετρο ένα αντικείμενο το οποίο περιέχει τις μεθόδους τις οποίες υποστηρίζει το άλλο άκρο. Οι μέθοδοι αυτοί, μπορούν να κληθούν σαν να ήταν τοπικές μέθοδοι με την διαφορά ότι σε κάθε περίπτωση επιστρέφουν υπόσχεση (promise) ή δέχονται επανάκληση (callback).

Και τα δύο άκρα της σύνδεσης θεωρούνται ισάξια μεταξύ τους και η μόνη διαφορά είναι ότι το ένα άκρο λειτουργεί ως TCP Server, και το άλλο άκρο ως TCP Client.

5.7 Αριθμοί μεταβλητού μήκους

Οι αριθμοί μεταβλητού μήκους, Variable-Length Number (VLN) ή αλλιώς Variable-Length Quantity (VLQ) αποτελούν αριθμητικό τύπο δεδομένων ο οποίος επιτρέπει την αναπαράσταση του σε ποικίλα μήκη byte σε αντίθεση με τους πιο διαδεδομένους αριθμητικούς τύπους όπως int32, double, float κ.α. στους οποίους είναι προκαθορισμένο το πλήθος byte που χρησιμοποιούν.

Η υλοποίηση VLN που χρησιμοποιήθηκε είναι η εξής:

Σε κάθε αριθμό υπάρχει ένα αρχικό «0» το οποίο δηλώνει ότι από το bit που βρίσκεται δεξιά του ξεκινάει η αριθμητική τιμή. Αν το αρχικό «0» βρίσκεται στην πρώτη θέση από αριστερά (μέσα στο byte), σημαίνει ότι δεν χρειάζεται να διαβαστούν περεταίρω bytes από το εξεταζόμενο. Για όσα bit που έχουν την τιμή «1» και βρίσκονται αριστερά από το αρχικό bit «0», τόσα έξτρα bytes θα χρειαστεί να διαβαστούν.

Παραδείγματα αριθμών VLN:

Διάδικό → Δεκαδικό

00000000 → 0

00100001 → 33

01111111 → 127

10000000 10000000 →128

10000001 00000000 →256

11011111 11111111 11111111 → 2097151

11111111 10000100 10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 → 212506491729134034616 32

Παρατηρούμε ότι ο κάθε αριθμός αναφέρει πόσα παραπάνω bytes θα χρειαστούν από το εξεταζόμενο. Για κάθε bit «1», το πλήθος των bytes που θα εξαχθεί αθροίζεται κατά ένα.

Στο πρώτο bit «0» που θα διαβαστεί, σταματάει το άθροισμα των επιπρόσθετων byte και ξεκινάει η ανάγνωση των byte του αριθμού.

Παρακάτω παρατηρούμε κώδικα που δέχεται JavaScript number και επιστρέφει JavaScript Buffer με τον JavaScript number κωδικοποιημένο ως αριθμό μεταβλητού μήκους:

```
function getVLN(number) {
  let dataLenBits = number.toString(2);
  let bitsLen = dataLenBits.length;
  let extraBytes = Math.floor(bitsLen / 8);
  let extraBits = bitsLen % 8;
  let headerBits1 = extraBytes + 1 + extraBits;
  let paddingBits = (8 - headerBits1 % 8) % 8;
  let finalHeaderBits = headerBits1 + paddingBits;
  let finalHeaderBytes = (finalHeaderBits / 8) + extraBytes;
  finalHeaderBits += extraBytes * 8;
  let tmpArr = new Array(finalHeaderBits);
  for (let i=0; i<extraBytes; i++) tmpArr[i] = 1;
  tmpArr[extraBytes] = 0;
  for (let i=extraBytes + 1; i < extraBytes + 1 + paddingBits; i++) {
    tmpArr[i] = 0;
  }
  for (let i = paddingBits + extraBytes + 1; i < extraBytes + 1 + paddingBits +
bitsLen; i++) {
    tmpArr[ i ] = parseInt(dataLenBits[ i - (paddingBits + extraBytes + 1) ]);
  }

  const buf = Buffer.alloc(finalHeaderBytes);
  let k = 0;
  for (let i=0; i<finalHeaderBytes; i++) {
    for (let j=0; j<8; j++, k++) {
      setBit(buf, i, j, tmpArr[k]);
    }
  }
  return buf;
}

function setBit(buffer, i, bit, value) {
  bit = 7-bit;
  if (value == 0) buffer[i] &= ~(1 << bit);
  else buffer[i] |= (1 << bit);
}
```

Απόσπασμα Κώδικα 5.1733: Μέθοδος δημιουργίας αριθμών τύπου VLN

Για την ανάγνωση των αριθμών μεταβλητού μήκους μέσα από ροές δεδομένων, είναι απαραίτητο να χρησιμοποιηθεί μηχανή πεπερασμένων καταστάσεων (Finite-State Machine) επειδή τα δεδομένα μπορεί να εισαχθούν κατακερματισμένα στον αναγνώστη αυτών των αριθμών. Παρακάτω παρουσιάζεται ο σχολιασμένος κώδικας υλοποίησης της μηχανής πεπερασμένων καταστάσεων η οποία πραγματοποιεί ανάγνωση αριθμών μεταβλητού μήκους μέσα από κατακερματισμένα δεδομένα:

```
  function createLimitViolationError(msg) {
    var err = new Error(msg);
    err.isLimitViolation = true;
    Error.captureStackTrace(err, createLimitViolationError);
    return err;
  }
  function readBit(buffer, i, bit){
    bit = 7-bit;
    return (buffer[i] >> bit) % 2;
  }
  function setBit(buffer, i, bit, value) {
    bit = 7-bit;
    if (value == 0) buffer[i] &= ~(1 << bit);
    else buffer[i] |= (1 << bit);
  }

  let extraHeaderBytes = 0;
  let sizeBuffer;
  let sizeBufferOffset = 0;
  let numReaderAction = 0;

  const numReaderActions = [
    // Extract extra header bytes
    function readHeader(max_header_bytes, max_accepted_bytes, data, i) {
      for (; i<data.length; i++) {  // For each byte
        for (let j=0; j<8; j++) { // For each bit
          if (readBit(data, i, j) == 1) {
            extraHeaderBytes++;
            // check size
            if (extraHeaderBytes > max_header_bytes) throw
createLimitViolationError(state + ' header bytes greater than allowed maximum
(max: ' + max_accepted_bytes + ').');
          } else {
```

```
            return numReaderActions[++numReaderAction](max_header_bytes,
max_accepted_bytes, data, i, j + 1);
          }
        }
      }
    },

    // Exctract padding
    function (max_header_bytes, max_accepted_bytes, data, i, j = 0) {
      // The padding will always be < 8 bits (1 byte)
      // Read until the end of this byte
      for (; j<8; j++) { // For each bit
        if (readBit(data, i, j) == 1) {
          // check also here if max size bytes was exceeded
          for (let k=j-1; k>=0; k--) setBit(data, i, k, 0);
          // Padding was needed, so together with the padding, are size header
bits.
          // extraHeaderBytes in this byte where cleared,
          // so this byte has now become a normal header byte
          extraHeaderBytes++;
          if (extraHeaderBytes > max_header_bytes) throw
createLimitViolationError(state + ' header bytes greater than allowed maximum
(max: ' + max_accepted_bytes + ').');
          return numReaderActions[++numReaderAction](max_header_bytes,
max_accepted_bytes, data, i);
        }
      }

      // If j == 8, we have reached at the end of the byte that could have
padding.
      // The next action will be called, starting at the next byte.
      return numReaderActions[++numReaderAction](max_header_bytes,
max_accepted_bytes, data, i + 1);
    },

    // Extract number
    function (max_header_bytes, max_accepted_bytes, data, i) {
      if (!sizeBuffer) sizeBuffer = Buffer.allocUnsafe(extraHeaderBytes);
      let bytesWritten = data.copy(sizeBuffer, sizeBufferOffset, i, i +
extraHeaderBytes);

      extraHeaderBytes -= bytesWritten;
      sizeBufferOffset += bytesWritten;

      if (extraHeaderBytes == 0) {
```

```
        // Set size of packet
        let vnum = (sizeBuffer.length == 0) ? 0 : sizeBuffer.readUIntBE(0,
sizeBuffer.length);
        if (vnum > max_accepted_bytes) {
          console.log(vnum);
          console.log(max_accepted_bytes);
          throw createLimitViolationError(state + ' size greater than allowed
maximum (max: ' + max_accepted_bytes + ').');
        }
        // Reset offset and sizeBuffer for next packet
        sizeBufferOffset = 0;
        sizeBuffer = null;

        // send data to next step
        numReaderAction = 0;
        return states[state](data, i + bytesWritten, vnum); // vnum: final number
      }
    }
  ];
```

Απόσπασμα Κώδικα 5.1834: Μηχανή πεπερασμένων καταστάσεων ανάγνωσης VLN

Το παραπάνω απόσπασμα κώδικα αποτελεί κομμάτι της ανάγνωσης και επεξεργασίας των εισερχόμενων δεδομένων μέσα από τη ροή δεδομένων του απομακρυσμένου άκρου. Το τελικό «return» καλεί γονική μηχανή πεπερασμένων καταστάσεων με την τελική αριθμητική τιμή η οποία είναι το «vnum». Η γονική μηχανή πεπερασμένων καταστάσεων είναι υπεύθυνη για την ανάγνωση περισσοτέρων δεδομένων και ο κώδικάς της θα παρουσιαστεί στην επόμενη ενότητα.

5.8 Πρωτόκολλο επιπέδου εφαρμογής

Και τα δύο άκρα της σύνδεσης θεωρούνται ισάξια μεταξύ τους και η μόνη διαφορά είναι ότι το ένα άκρο λειτουργεί ως TCP Server, και το άλλο άκρο ως TCP Client. Κατά την αρχική σύνδεση, τα δύο άκρα είναι απαραίτητο να ανταλλάξουν αντικείμενα σύνδεσης μέσα σε ένα προκαθορισμένο χρονικό διάστημα. Τα αντικείμενα αυτά αποτελούνται από ένα JSON πίνακα τεσσάρων θέσεων ο οποίος περιέχει:

- **myNodeId:** (string) Ταυτότητα τοπικού κόμβου.
- **controls:** (object) Περιγραφή μεθόδων προς το απομακρυσμένο άκρο.

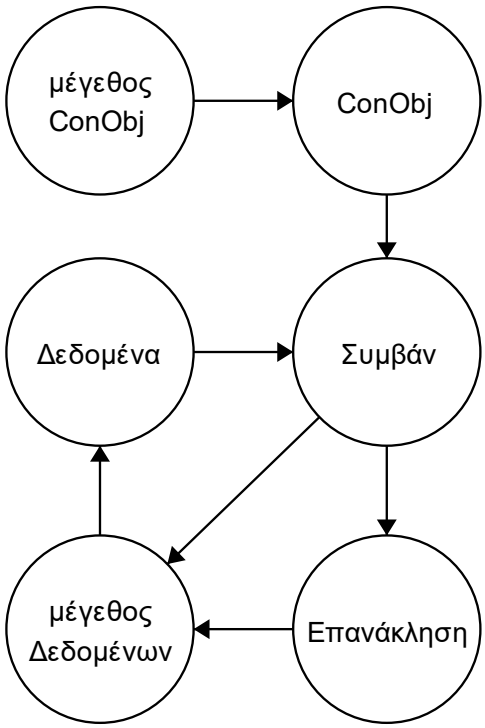
- **maxCallbacksIn:** (number) Μέγιστο επιτρεπτό πλήθος επανακλήσεων απομακρυσμένου κόμβου.
- **maxContentLengthIn:** (number) Μέγιστο επιτρεπτό πλήθος bytes δεδομένων εισερχόμενου πακέτου.

Για να μπορέσουν τα δύο άκρα να καταλάβουν που τελειώνει το κάθε αντικείμενο μέσα από τη ροή (stream), οι αποστολείς πριν γράψουν το αντικείμενο στη ροή, γράφουν το πλήθος byte του αντικειμένου σε μορφή VLN.

Το κάθε άκρο, μετά την λήψη του αντικειμένου σύνδεσης, θεωρεί τη σύνδεση αρχικοποιημένη και επιτρέπει την αποστολή και λήψη πακέτων. Τα πακέτα αποτελούνται από:

- Συμβάν (event)
- Επανάκληση (callback)
- Πλήθος byte δεδομένων
- Δεδομένα

Η επανάκληση είναι προαιρετική. Παρακάτω παρατηρούμε ένα σχεδιάγραμμα αναπαράστασης της μηχανής πεπερασμένων καταστάσεων η οποία «διαβάζει» μέσα από τη ροή δεδομένων τα στοιχεία:



Εικόνα 5.9: Σχεδιάγραμμα αναπαράστασης της μηχανής πεπερασμένων καταστάσεων του πρωτοκόλλου

Το συμβάν, η επανάκληση και το πλήθος δεδομένων αποτελούν αριθμούς μεταβλητού μήκους οι οποίοι διαβάζονται με τη βοήθεια της μηχανής πεπερασμένων καταστάσεων της ενότητας 5.7. Ο κώδικας που υλοποιεί την παραπάνω μηχανή πεπερασμένων καταστάσεων είναι ο ακόλουθος:

```
const getByteCount = num => Math.ceil(num.toString(2).length / 8);
const maxConObjSizeHB = getByteCount(maxConObjSize);
const myMaxTokenizedControlCount = myTokenizedControls.length;
const myMaxTokenizedControlCountHB = getByteCount(myMaxTokenizedControlCount);
const maxCallbacksInHB = getByteCount(maxCallbacksIn - 1);
const maxContentLengthHB = getByteCount(maxContentLength);
const maxCallbacksOutHB = getByteCount(maxCallbacksOut - 1);
let state = 'readConObjContentLength';
```

```
  let eventNum;  // Event
  let callbackNum;  // Callback
  let contentLength;  // Content
  let dataBuffer;  // Data extraction
  let dataBufferOffset = 0;

  const states = {
    readConObjContentLength: function (data, i=0, vnum=false) {
      if (vnum === false) return numReaderActions[numReaderAction](
maxConObjSizeHB, maxConObjSize, data, i);
      contentLength = vnum;
      state = 'extractConObjData';
      return states[state](data, i);
    },
    extractConObjData: function (data, i) {
      if (!dataBuffer) dataBuffer = Buffer.allocUnsafe(contentLength);
      let bytesWritten = data.copy(dataBuffer, dataBufferOffset, i, i +
contentLength);

      contentLength -= bytesWritten;
      dataBufferOffset += bytesWritten;
      if (contentLength == 0) {
        onAccept(dataBuffer);
        callbackNum = null;

        // Reset offset for next packet
        dataBufferOffset = 0;
        dataBuffer = null;
        // Forward remaining data to readHeader action
        states[state = 'readEvent'](data, i + bytesWritten);
      }
    },

    readEvent: function (data, i, vnum = false) {
      if (vnum === false) return numReaderActions[numReaderAction](
myMaxTokenizedControlCountHB, myMaxTokenizedControlCount, data, i);
      eventNum = vnum;
      if (vnum == 0 || myTokenizedControls[eventNum-1].cb) state =
'readCallback';
      else state = 'readContentLength';
      return states[state](data, i);
    },

    readCallback: function (data, i, vnum = false) {  // only when event = 0
      if (vnum === false) {
```

```
        if (eventNum == 0)
          return numReaderActions[numReaderAction](maxCallbacksOutHB,
maxCallbacksOut - 1, data, i);
        else
          return numReaderActions[numReaderAction](maxCallbacksInHB,
maxCallbacksIn - 1, data, i);
      }
      callbackNum = vnum;
      state = 'readContentLength';
      return states[state](data, i);
    },

    readContentLength: function (data, i, vnum = false) {
      if (vnum === false) return numReaderActions[numReaderAction](
maxContentLengthHB, maxContentLength, data, i);
      contentLength = vnum;
      state = 'extractData';
      return states[state](data, i);
    },

    extractData: function (data, i) {
      if (!dataBuffer) dataBuffer = Buffer.allocUnsafe(contentLength);
      let bytesWritten = data.copy(dataBuffer, dataBufferOffset, i, i +
contentLength);

      contentLength -= bytesWritten;
      dataBufferOffset += bytesWritten;
      if (contentLength == 0) {
        packetHandler(eventNum, callbackNum, dataBuffer);
        callbackNum = null;
        // Reset offset for next packet
        dataBufferOffset = 0;
        dataBuffer = null;

        // Forward remaining data to readHeader action
        states[state = 'readEvent'](data, i + bytesWritten);
      }
    }
  };
```

Απόσπασμα Κώδικα 5.19: Μηχανή πεπερασμένων καταστάσεων ανάγνωσης πακέτων TCP πρωτοκόλλου

ΚΕΦΑΛΑΙΟ 6

ΒΕΛΤΙΩΣΕΙΣ - ΣΥΜΠΕΡΑΣΜΑΤΑ

6.1 Δυναμικά ευρετήρια

Τα δυναμικά ευρετήρια αποτελούν ευρετήρια (indexes) της βάσης δεδομένων τα οποία προτείνονται στο χρήστη να δημιουργηθούν ανάλογα με τα δεδομένα, τα φίλτρα και τις λειτουργίες που χρησιμοποιεί. Επιπλέον, ο χρήστης μπορεί να τα προσθέσει και να τα αφαιρέσει μέσα από την εφαρμογή. Για μεγάλο όγκο δεδομένων είναι απαραίτητη η ύπαρξή τους επειδή χωρίς αυτά η βάση δεδομένων θα χεριάζετε σε κάθε αναζήτηση να αναλύει όλα τα δεδομένα μέχρι να φτάσει σε αυτό που αναζητά (COLSCAN).

6.2 Εναύσματα (Triggers) ⚡

Τα εναύσματα αποτελούν οντότητες οι οποίες πράττουν μια ενέργεια στην περίπτωση εκπλήρωσης μιας συνθήκης. Παραδείγματα συνθηκών:

- Αν ο αριθμός των logs που περιέχονται σε ένα φάκελο ενός ΔΚ υπερβεί το **x.**
- Αν ο αριθμός των logs που περιέχονται σε ένα φάκελο ενός ΔΚ αυξηθεί.
- Αν η τιμή μιας παραμέτρου ενός log υπερβεί το **y.**

Παραδείγματα ενεργειών εναυσμάτων:

- Δημιουργία νέου log, προαιρετική προσθήκη δεδομένων ή μέρος των δεδομένων του τελευταίου log που εκπλήρωσε την συνθήκη.
- Κλήση μεθόδου (control) απομακρυσμένου εξυπηρετητή, προαιρετικά με τα δεδομένα ή μέρος των δεδομένων του log που εκπλήρωσε την συνθήκη.

Με την χρήση εναυσμάτων θα μπορούν έμμεσα οι απομακρυσμένοι εξυπηρετητές να επικοινωνούν μεταξύ τους.

Παράδειγμα 1:

- Έχουν προστεθεί δύο απομακρυσμένοι εξυπηρετητές, Ο *s1* και ο *s2*.

- Ο *s1* αποτελεί firewall για τον *s2* και έχει εκθέσει control «μπλοκάρισμα IP».

- Ο *s2* είναι εφαρμογή η οποία πραγματοποιεί αυθεντικοποίηση χρηστών και κάνει report αρχείου καταγραφής σε περίπτωση αποτυχίας αυθεντικοποίησης χρήστη.

- Έχει προστεθεί ΔΚ με φάκελο *φ1* του οποίου τα φίλτρα επιτρέπουν μόνο αρχεία καταγραφής από αποτυχίες αυθεντικοποίησης του εξυπηρετητή *s2*.

- Έχει προστεθεί έναυσμα με συνθήκη → (Αν ο αριθμός των logs που περιέχονται στον φάκελο *φ1* αυξηθεί) και εντολή → κάλεσε control «μπλοκάρισμα IP» από τον εξυπηρετητή *s1* με παράμετρο την IP του χρήστη που βρίσκεται στα δεδομένα του log.

Με το παραπάνω παράδειγμα επιτυγχάνεται ο αποκλεισμός IP των χρηστών που πραγματοποίησαν αποτυχημένη προσπάθεια αυθεντικοποίησης.

Παράδειγμα 2:

- Έχουν προστεθεί δύο απομακρυσμένοι εξυπηρετητές, Ο *s1* και ο *s2*.

- Ο *s1* αποτελεί εξυπηρετητή που στέλνει SMS εκθέτοντας το control «αποστολή SMS».

- Ο *s2* αποτελεί βάση δεδομένων και κάνει report μετρικής χωρητικότητας δίσκου κάθε ώρα.

- Έχει προστεθεί ΔΚ με φάκελο *φ1* του οποίου τα φίλτρα επιτρέπουν μόνο μετρικές δίσκου του εξυπηρετητή *s2*.

- Έχει προστεθεί έναυσμα με συνθήκη → (Αν η τιμή usedPercent από τον φάκελο *φ1* ξεπεράσει το 85) και εντολή → κάλεσε control «αποστολή SMS» από τον εξυπηρετητή *s1* με παράμετρο το usedPercent του τελευταίου log.

Με το παραπάνω παράδειγμα επιτυγχάνεται η ενημέρωση μέσω SMS στην περίπτωση που η χωρητικότητα του δίσκου του εξυπηρετητή *s2* μειωθεί κάτω από 15%.

6.3 Πίνακες ελέγχου (Controls)

Οι πίνακες ελέγχου αποτελούν οντότητες οι οποίες περιέχουν αποθηκευμένα controls (μεθόδους) απομακρυσμένων εξυπηρετητών. Ένα control είναι στην ουσία μια συλλογή φορμών η οποία θα μπορεί να περιέχει οποιοδήποτε controls οποιουδήποτε ανακρουσμένου εξυπηρετητή.

6.4 Διαγράμματα (Diagrams)

Τα διαγράμματα αποτελούν οντότητες οι οποίες παρουσιάζουν:

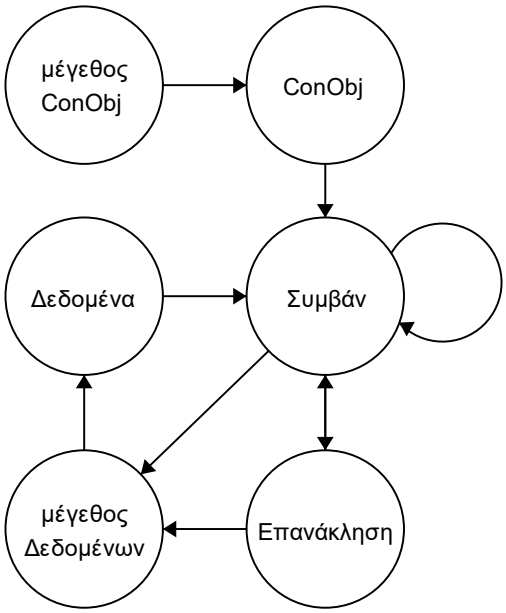
- Απεικονίσεις εξυπηρετητών
- Συνδέσεις μεταξύ των εξυπηρετητών
- Καταστάσεις εξυπηρετητών (online/offline/deactivated)
- Πλήθος αρχείων καταγραφής του κάθε εξυπηρετητή (απλή αναφορά σε φάκελο ΔΚ)

6.5 Οθόνες (Displays/Dashboards)

Οι οθόνες αποτελούν οντότητες οι οποίες περιέχουν αναφορές σε άλλες οντότητες και έχουν τη δυνατότητα να τις παρουσιάζουν πλήρως ή εν μέρη (π.χ. πλήθος αρχείων καταγραφής φακέλου ΔΚ, αρχεία καταγραφής φακέλου ΔΚ, πίνακας κατηγοριοποίησης αρχείων καταγραφής φακέλου ΔΚ)

6.6 Επέκταση πρωτοκόλλου επιπέδου εφαρμογής

Το πρωτόκολλο επιπέδου εφαρμογής (ενότητα 5.8) υποστηρίζει πακέτα τα οποία μπορεί να μην έχουν επανάκληση αλλά δεν υποστηρίζει πακέτα τα οποία δεν έχουν δεδομένα. Η επέκταση που θα μπορούσε να γίνει, είναι να μπορεί να υποστηρίζει όχι μόνο πακέτα τα οποία δεν έχουν επανάκληση, αλλά και πακέτα που τα δεδομένα είναι προαιρετικά.



Εικόνα 6.1: Σχεδιάγραμμα αναπαράστασης της μηχανής πεπερασμένων καταστάσεων του βελτιωμένου πρωτοκόλλου

6.7 Συμπεράσματα

Συνοψίζοντας, η ανάπτυξη ενός πληροφοριακού συστήματος είναι μια εργασία η οποία χαρακτηρίζεται ως δύσκολη, χρονοβόρα, κοστοβόρα σε πόρους αλλά και σε σχεδιαστική προσπάθεια. Οι περίπλοκες διαδικασίες που συνδέονται με την ανάπτυξη ενός πληροφοριακού συστήματος περιλαμβάνουν τη συλλογή απαιτήσεων, τον σχεδιασμό, την υλοποίηση, τον έλεγχο ποιότητας και τελικά την ολοκλήρωση και συντήρηση του συστήματος.

Κύρια πρόκληση της παρούσας διπλωματικής εργασίας, ήταν η αντιμετώπιση των συνθηκών ανταγωνισμού (racing conditions) με την υλοποίηση κατάλληλων δομών δεδομένων και μηχανισμών συγχρονισμού σε συνδυασμό με εκτενείς δοκιμές (testing), ώστε να εξασφαλιστεί η απρόσκοπτη λειτουργία στο ασύγχρονο περιβάλλον του συστήματος. Η ύπαρξή τους δεν ήταν προφανής, ενώ μπορεί να προκύψουν σποραδικά και με απρόβλεπτους τρόπους. Αυτό δυσκολεύει τον εντοπισμό σφαλμάτων και την αντιμετώπιση προβλημάτων, ενώ αρκετές φορές εξανάγκαζε σε επανασχεδιασμό των δομών και των λειτουργιών του συστήματος.

ΒΙΒΛΙΟΓΡΑΦΙΑ

CHANDLER, H. (χ.χ.). *Microservices vs. monolithic architecture*. Ανάκτηση από www.atlassian.com: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

Douglas Mauro, K. S. (2001, 7). *Essential SNMP.* Ανάκτηση από doc.lagout.org: https://doc.lagout.org/network/Essential%20SNMP%202001.pdf

Green, J. (1998, 10 31). *HP OpenView Network Node Manager 5.02 for Windows NT*. Ανάκτηση από www.itprotoday.com: https://www.itprotoday.com/compute-engines/hp-openview-network-node-manager-502-windows-nt

Heller, M. (2022, 10 1). *www.infoworld.com*. (InfoWorld) Ανάκτηση από www.infoworld.com: https://www.infoworld.com/article/3210589/what-is-nodejs-javascript-runtime-explained.html

HPE OneView. (χ.χ.). Ανάκτηση από www.hpe.com: https://www.hpe.com/us/en/integrated-systems/software.html

IEEE. (2002, 6). *ieeexplore.ieee.org*. doi:10.1109/TSE.2002.1010061

Jeffrey D. Case, M. F. (χ.χ.). *RFC 1098: A Simple Network Management Protocol (SNMP).* Ανάκτηση από https://www.rfc-editor.org/rfc/rfc1098

Lucendo, J. (1998, 4). Ανάκτηση από https://citeseerx.ist.psu.edu/: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=977eb32d1d035a3fe23fa68cdf62425690edfaa6

Mukesh, P. (2019, 8). *www.ijarw.com.* doi:IJARW | ISSN (O) - 2582-1008

Patrick T Eugster, P. F.-M. (2003, 6). *The many faces of publish/subscribe.* doi:https://doi.org/10.1145/857076.857078

Severance C., S. o. (2012, 2). *IEEE*. doi:10.1109/MC.2012.57