Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών &
Μηχανικών Υπολογιστών
Τομέας Ηλεκτρονικής και Υπολογιστών

Διπλωματική Εργασία

Σύστημα για την Αυτόματη Παρακολούθηση του Χρόνου Λειτουργίας και Απόκρισης ενός Ιστότοπου

Εκπόνηση:
Σεντονάς Σταύρος
ΑΕΜ: 9386

Επίβλεψη:
Υπ. Δρ. Καρανικιώτης Θωμάς
Δρ. Παπαμιχαήλ Μιχαήλ
Καθ. Συμεωνίδης Ανδρέας

Θεσσαλονίκη, Ιούνιος 2023

Περίληψη

Η εξέλιξη της τεχνολογίας και της πληθώρας εφαρμογών που αναπτύσσονται στα πλαίσιο αυτής, καθιστούν επιτακτική την ανάγκη ύπαρξης συστημάτων που θα ελέγχουν την εύρυθμη λειτουργία τους. Πιο συγκεκριμένα μιλάμε για την ελέξιλη στο χώρο του διαδικτύου και των δομών που έχουν υλοποιηθεί πάνω σε αυτό.

Πλέον αναφερόμαστε σε ένα συνεχώς αυξανόμενο και ευρύ δίκτυο web εφαρμογών - λογισμικών ως υπηρεσίας (SaaS - Software as a Service) που ζουν στον Διαδίκτυο (World Wide Web). Η λειτουργία αυτών μπορεί να ελεχθεί με διάφορους τρόπους. Από Unit Testing, στο πλαίσιο του κύκλου ανάντυξης του λογισμικού (continuous integration, continuous deployment cycle) προκειμένου να ελεχθεί λειτουργικά το σύστημα για την αποφυγή bugs, μέχρι και Παρακολούθηση Δικτύου (Network Monitoring), για να επιβεβαιωθεί η σωστή λειτουργία των συστημάτων καθόλη της διάρκεια του κύκλου ζωής τους.

Η παρούσα διπλωματική εστιάζει στην ανάπτυξη ενός συστήματος Παρακολούθησης Δικτύου και κατεπέκταση εφαρμογής που θα δίνει της δυνατότητα στους χρήστες της να παρακολουθούν, εύκολα, την ομαλή λειτουργία των διαδικτυακών σελιδών τους, είτε αυτά είναι εφαρμογές, είτε απλά στατικές σελίδες. Το σύστημα στηρίζεται στη βασική μέθοδο εντοπισμού διαθεσιμότητας μίας ιστοσελίδας, γνωστή και ως ping. Κάνοντας ping μπορούμε να πάρουμε χρήσιμη πληροφορία σχετικά με το αν το υπό μελέτη σύστημα μπορεί να ανταποκριθεί και εφόσον ανταποκριθεί σχετικά με το χρόνο που μεσολάβησε μέχρι να απαντήσει. Συνεχίζοντας την λογική πορεία ενός τέτοιου συστήματος μπορούμε ακόμα στο μήνυμα που στέλνουμε να έχουμε πληροφορία που θα επηρεάζει την απάντηση που θα περιμέναμε να δούμε, έχοντας έτσι έναν ακόμα μηχανισμό για την αναγνώριση και αποφυγή πιθανών bugs, ή λαθών κατά τη διαδικασία ανάπτυξης λογισμικών ως υπηρεσία.

Title

Development of a System for Uptime Status Monitoring

Abstract

The evolution of technology and the abundance of applications developed within this framework make it imperative to have systems that will control their smooth operation. More specifically, we are talking about monitoring in the realm of the internet and the structures implemented on it.

Today we refer to a constantly growing and extensive network of web applications - software as a service (SaaS) - that reside on the Internet (World Wide Web), whose operations can be monitored in various ways. From unit testing, within the software development cycle (continuous integration, continuous deployment) to ensure that the system functions are running properly and avoid bugs, to network monitoring, to verify the correct functioning of the systems throughout their life cycle.

This thesis focuses on the development of a Network Monitoring system and a web application that will enable its users to easily monitor how their systems operate, whether they are applications or simply static pages. The system is based on the basic method of checking the availability of a website, known as ping. By pinging, we can obtain useful information regarding whether the system under study can respond and, if so, the time it takes to respond. Continuing the logical progression of such a system, we can further enhance the message we send with predetermined data to check the response of the system and verify the returned data, thereby identifying and avoiding potential bugs or error during the software's development process

Sentonas Stavros
Electrical & Computer Engineering Department,
Aristotle University of Thessaloniki, Greece

June 2023

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Ακρωνύμια Εγγράφου

Παρακάτω παρατίθενται ορισμένα από τα πιο συχνά χρησιμοποιούμενα ακρωνύμια της παρούσας διπλωματικής εργασίας:

RUM ⟶ Real User Monitoring
API ⟶ Application Programming Interface
SaaS ⟶ Software as a Service
HTTP ⟶ Hypertext Transfer Protocol
GCP ⟶ Google Cloud Storage

1
Εισαγωγή

Τα τελευταία χρόνια, ο κλάδος του Διαδικτύου προσεγγίζει ένα μεγαλύτερο κομμάτι ανθρώπων, τόσο από τη μεριά του καταναλωτή όσο και από τη μεριά του παραγωγού. Όσο αφορά τον καταναλωτή οι δυνατότητες που του προσφέρονται μπορούν να διακριθούν στους εξής τομείς:

- Επικοινωνία: το διαδίκτυο παρέχει τη δυνατότητα άμεσης επικοινωνίας μεταξύ μεγάλων αποστάσεων, που δεν περιορίζεται μόνο στο ακουστικό ερέθισμα, αλλά επιτρέπει και την μετάδοση οπτικο-ακουστικής πληροφορίας

- Πρόσβαση Πληροφορίας: ίσως το σημεντικότερο αγαθό που προσφέρει το διαδίκτυο είναι η πληθώρα πληροφορίας που στεγάζει. Μηχανές Αναζήτηση (search engines), Online Βάσεις Δεδομένων (online databases), και άλλου είδους εφαρμογών εκπαιδευτικού χαρακτήρα που δίνουν πρόσβαση σε άτομα που το επιθυμούν, να κάνουν έρευνα

- Ποιότητα ζωής: σε αυτή την κατηγορία περιλαμβάνονται όλες εκείνες οι υπηρεσίες που διευκολύνουν την καθημερινότητα των χρηστών. Online αγορές (eshops) που γλιτώνουν την αναμονή σε ουρές ή ακόμα επιτρέπουν την εύκολη αγορά προϊόντων από απομακρυσμένες περιοχές του πλανήτη, ψυχαγωγία και πρόσβαση σε υπηρεσίες που επιταχύνουν ενέργειες που υπό άλλες περιπτώσεις θα ήταν χρονοβόρες (online banking, πληρωμή λογαριασμών, κρατήσεις ξενοδοχείων/εισητηρίων)

Από τη μεριά του παραγωγού, τα μέσα που υπάρχουν για την ανάπτυξη τέτοιων εφαρμογών/υπηρεσιών/συστημάτων μέρα με τη μέρα αυξάνονται. Η ραγδαία εξέλιξη στον χώρο των cloud υποδομών, καθιστά ευκολότερη και επισπεύδει τόσο την δημιουργία διαδικτυακών εφαρμογών, και σελιδών σε ένα γενικότερο πλαίσιο, όσο και την μεγέθυνση και αύξηση αυτών (scale up). Μάλιστα η επιλογή κατάλληλου παρόχου τέτοιων υπηρεσιών αποτελεί ένα αρκετά σημεντικό αντικείμενο μελέτης

[1]. Πέρα από τον οικομικό παράγοντα θα πρέπει να προσμετρηθούν οι παροχές, τα πλεονεκτήματα αλλά και η αποδοτικότητα που κάθε ένας προσφέρει.

Βλέποντας λοιπόν το πόσο συνυφασμένη είναι η ζωή του σύγχρονου ανθρώπου με το δίκτυο αλλά και τις δυνατότητες και τα μέσα που έχει ο καθένας για να αναπτύξει εφαρμογές σε αυτό, καθίσταται επιτακτική η ανάγκη ύπαρξης μηχανισμών που θα αναγνωρίζουν σφάλματα (bugs) και θα επιβλέπουν την ορθή λειτουργία των υπό μελέτη συστημάτων καθόλη τη διάρκεια ζωής τους.

1.1 Περιγραφη του Προβληματος

Η Παρακολούθηση (Monitoring) ενός συστήματος που "ζει" στο χώρο του διαδικτύου μπορεί να γίνει κυρίως με δύο τρόπους:

- **Ενεργή Παρακολούθηση (Active Monitoring)**: έχει περισσότερο προγνωστικό και προληπτικό χαρακτήρα. Συχνά αναφέρεται και ως **Συνθετική παρακολούθηση (Synthetic Monitoring)**, λόγω της φύσης των ενεργειών της. Ουσιαστικά δημιουργεί πλασματικά api calls και όχι πραγματικά δεδομένα χρηστών προκειμένου να ελεγχθεί η απόκριση του υπό μελέτη συστήματος. Η συχνότητα αποστολής των συνθετικών αιτημάτων συνήθως ρυθμίζεται από το χρήστη.
- **Παθητική Παρακολούθηση (Passive Monitoring)**: παρέχει μία πιο πλήρη εικόνα σχετικά με πως χρησιμοποιούνται οι πόροι του δικτύου καταγράφοντας, αποθηκεύοντας και αναλύοντας τα δεδομένα του χρήστη. Για αυτό πολλές φορές αναφέρεται στη βιβλιογραφία ως **Παρακολούθηση Πραγματικών Χρηστών (Real User Monitoring - RUM)**. Έτσι μπορεί κανείς να εντοπίσει τις τάσεις χρήσης του δικτύου για τη βελτίωση και βελτιστοποίησή του συστήματος.

Πίνακας 1.1: Χαρακτηριστικά Ενεργής και Παθητικής Παρακολούθησης

Ενεργή Παρακολούθση (Active Monitoring)	**Παθητική Παρακολούθηση (Passive Monitoring)**
• Στηρίζεται σε συνθετικά API calls	• Αναλύει δεδομένα πραγματικών χρηστών
• Παράγει δεδομένα για συγκεκριμένες πτυχές του δικτύου	• Πλήρης εικόνα της απόδοσης του δικτύου
• Μπορεί να μετρήσει την κίνηση εντός και εκτός του δικτύου	• Μετράει κίνηση μόνο εντός του δικτύου
• Μπορεί να εντοπίσει προβλήματα πριν ακόμα μπορέσουν να τα εντοπίσουν οι χρήστες	• Εντοπίζει προβλήματα που εμφανίζονται εκείνη τη στιγμή

Και οι δύο μέθοδοι έχουν πλεονεκτήματα και μειονεκτήματα, τα οποία φαίνονται και στον παραπάνω πίνακα 1.1. Όπως είναι εμφανές η παθητική παρακολούθηση γίνεται πάνω στο σύστημα που θέλουμε να μελετήσουμε, πράγμα το οποίο σήμαινει ότι σαν εξωτερικοί παράγοντες στο σύστημα δεν θα μπορέσουμε να προσφέρουμε ανάλογες υπηρεσίες. Για το λόγο αυτό συνεχίζουμε την ανάλυση στο πλαίσιο της Ενεργής Παρακολούθησης Δικτύων.

Οι βασικοί λόγοι που χρειάζονται τέτοιου είδους υπηρεσίες όπως αναφέρεται και στα [2], [3] είναι οι εξής:

- Βελτίωση προβλημάτων που σχετίζονται με την απόδοση του συστήματος πρωτού τα βιώσουν οι πραγματικοί χρήστες του συστήματος
- Ύπαρξη κάποιας μονάδας αξιολόγησης της απόδοσης του
- Αξιολόγηση του συστήματος υπό μεγαλύτερο φορτίο
- Διασφάλιση της Συμφωνίας Επιπέδου Υπηρεσιών (Service Level Agreement - SLA), μεταξύ του παρόχου υπηρεσιών και των χρηστών
- Παρέχει χρήσιμα δεδομένα ακόμα και σε καινούργια συστήματα που ακόμα μπορεί να μην έχουν χρήστες

1.2 Σκοπος - Συνεισφορα της Διπλωματικης Εργασιας

Η παρούσα διπλωματική εργασία μελετά τη χρήση σύγχρονων τεχνολογιών για τη δημιουργία ενός συστήματος Ενεργής Παρακολούθησης (Active Monitoring) σε συνδυασμό με μία SaaS εφαρμογή που θα παρουσιάζει μέσα από διαγράμματα τα αποτελέσματα της ανάλυσης της πληροφορίας που εξάγεται.

Εξετάζονται διάφοροι τρόποι και υλοποιήσεις που δοκιμάστηκαν κατά τη διάρκεια εκπόνησεις της διπλωματικής αυτής εργασία, και τέλος θα αναλύσουμε τα αποτελέσματα που παράξαμε καθόλη της διάρκεια των πειραμάτων που διενεργήθηκαν.

1.3 Διαρθρωση της Αναφορας

Η διάρθρωση της παρούσας διπλωματικής εργασίας είναι η εξής:

- **Κεφάλαιο 2**: Περιγράφονται τα βασικά εργαλεία και θεωρητικά στοιχεία στα οποία βασίστηκαν οι υλοποιήσεις
- **Κεφάλαιο 3** Αναφορά συστημάτων που ήδη χρησιμοποιούνται και παράθεση διαφορών με την υλοποίησή μας
- **Κεφάλαιο 4** Περιγραφή των υλοποιήσεων και πλήρης περιγραφή του τελικού συστήματος
- **Κεφάλαιο 5** Παρουσιάζονται τα τελικά συμπεράσματα.

- **Κεφάλαιο 6** Προτείνονται θέματα για μελλοντική μελέτη, αλλαγές και επεκτάσεις.

2
Θεωρητικό Υπόβαθρο

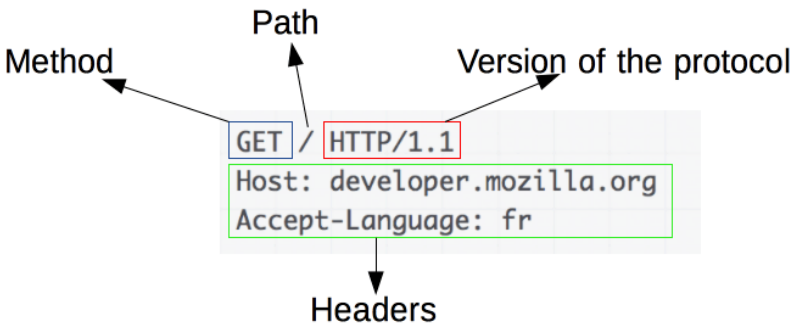
Στο κεφάλαιο αυτό θα παρουσιαστούν εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση του Συστήματος Ενεργής Παρακολούθησης, καθώς και έννοιες και τεχνολογίες που αξιοποιήθηκαν για το σκοπό αυτό.

2.1 Hypertext Transfer Protocol

Το πρωτόκολλο επικοινωνίας HTTP (Hypertext Transfer Protocol) αποτελεί το πιο διαδεδομένο και ευρέως γνωστό πρωτόκολλο στο χώρο του διαδικτύου. Αναπτύχθηκε από τους Tim Berners-Lee και την ομάδα του το 1990 και από τότε έχει περάσει πολλές αλλαγές προκειμένου να μπορεί να ανταπεξέλθει στις ολοένα και συνεχώς αυξανόμενες ανάγκες του σήμερα.

Αποτελεί τη βάση κάθε μετάδοσης πληροφορίας στο διαδίκτυο. Στηρίζεται στην επικοινωνία δύο υπολογιστών, ενός που κάνει τα αιτήματα (client) και ενός που απαντά σε αυτά (server). Στο τέλος της επικοινωνίας στην μεριά του παραλήπτη θα υπάρχει ανακατασκευασμένο ένα ολοκληρωμένο αρχείο, από τα διάφορα υποαρχεία που μαζεύτηκαν, που μπορεί να είναι αρχεία ήχου, εικόνας, video. Τα αιτήματα αυτού που ξεκινάει την επικοινωνία ονομάζονται requests, ενώ οι απαντήσεις του αποστολέα responses.

Η βασική δομή ενός http αιτήματος σχήμα 2.1 περιληπτικά περιλαμβάνει τη μέθοδο (method) του αιτήματος, που περιγράφει τη βασική λειτουργία του, το μονοπάτι (path) στο οποίο θα επικοινωνήσει με τον server, την έκδοση του πρωτοκόλλου που θα χρησιμοποιηθεί και τέλος headers προκειμένου να κρίνει ο server αν πρέπει να απαντήσει ή όχι πίσω στον client



Σχήμα 2.1: Βασική Δομή ενός αιτήματος http

2.1.1 Μέθοδοι

Πιο συγκεκριμένα οι βασικές μέθοδοι που παρέχει το http και οι συνήθεις λειτουργίες τους είναι οι εξής:

- **GET**: παίρνει πληροφορία από τον server
- **POST**: υποβάλλει πληροφορία, προκαλώντας αλλαγές στον τρόπο λειτουργίας του server. Σχετίζεται συχνά με τη δημιουργία πληροφορίας που προηγουμένως δεν υφίστατο
- **PUT**: όπως και πριν στέλνει πληροφορία στον παραλήπτη υπολογιστή, αλλά αυτή τη φορά επηρεάζει πόρους που ήδη υπήρχαν στο σύστημα. Σχετίζεται συχνά με την τροποποίηση ήδη υπάρχουσας πληροφορίας
- **DELETE**: διαγράφει από το σύστημα του server το συγκεκριμένο πόρο.

Αξίζει να σημειωθεί ότι πέρα από τις τέσσερις αυτές βασικές μεθόδους υπάρχουν και άλλες όπως είναι η **PATCH** που αποτελεί ειδική περίπτωση της PUT, η **HEAD** που αποτελεί ειδική περίπτωση της GET, καθώς και άλλες που σχετίζονται με τη σύνδεση μεταξύ server και client. Αυτές είναι οι **CONNECT**, **OPTIONS** και **TRACE**

2.1.2 Εκδόσεις HTTP

Η πρώτη έκδοση του HTTP παρόλλο που δεν είχε κάποια συγκεκριμένο τίτλο, εκ των υστέρων το αποκάλεσαν HTTP/0.9. Αποτελεί την πιο απλή έκδοση του πρωτοκόλλου. Δεν υποστηρίζονταν headers, κωδικοί κατάστασης (status codes). Εξυπηρετούσε μόνο GET αιτήματα και η μοναδική απάντηση που μπορούσε να επιστρέψει ήταν hypertext αρχεία. Κάθε φορά που ο server ανταποκρινόταν και έστελνε απάντηση η επικοινωνία με τον client έκλεινε κατευθείαν.

Στη συνέχεια και με την ανάπτυξη του διαδικτύου προστέθηκαν και άλλες λειτουργίες. Πέριξ του 1996, με τη επόμενη έκδοση του πρωτοκόλλου (HTTP/1.0) τα αιτήματα πλέον συνοδεύονταν από headers, μεταπληροφορία σχετικά με τη

κατάσταση του αιτήματος, τον τύπο της πληροφορίας που περιμένουμε να έρθει (stylesheets, media, hypertext) καθώς και την έκδοση του HTTP που χρησιμοποιήθηκε στη συγκεκριμένη επικοινωνία. Επιπλέον πέρα από τη GET μέθοδο υπάρχει η δυνατότητα για POST και PUT, δημιουργία και τροποποίηση πληροφορίας δηλαδή.

Στη συνέχεια το HTTP/1.1 προσπαθεί να βελτιώσει τις ήδη υπαρχουσες δυνατότητες κάνοντας την επικοινωνία μεταξύ server και client πιο αποδοτική. Αντί να κλείνει η επικοινωνία μετά από κάθε μήνυμα, η σύνδεση παραμένει ανοιχτή γλιτώνοντας έτσι μία σταθερή καθυστέρηση που υπήρχε σε κάθε αίτημα

Φτάνοντας στο σήμερα, μιλάμε για το HTTP/2.0 [4]. Αξιοποιώντας το πρωτόκολλο SPDY (Speedy) που αναπτύχθηκε κάποια χρόνια πριν την κυκλοφορία του, και κτίζοντας πάνω σε αυτό, κατάφερε να μειώσει τους χρόνους επικοινωνίας server-client. Μερικοί από τους τρόπους που επιτυγχάνεται αυτό είναι η μετατροπή του http από text πρωτόκολλο, σε δυαδικό (binary protocoll), επιτρέποντας έτσι χρήση καλύτερων και αποδοτικότερων τεχνικών επικοινωνίας. Επιπλέον συμπιέζει τους headers (header compression) καθώς αποτελούν πληροφορία που επαναλαμβάνεται όταν τα αιτήματα στον server είναι συνεχή. Ο server ακόμα, αποκτά έναν μηχανισμό (server-push) που του επιτρέπει να προωθεί πληροφορία στον client (στην cache του client συγκεκριμένα), που δεν έχει ζητήσει ακόμα, αλλά βάση αυτού που αιτήται, μάλλον θα ζητήσει εντός του ιδίας συνεδρίας.

Τέλος, πρέπει να αναφερθούμε στην τελευταία, αν και όχι ακόμα ευρέως διαδεδομένη, έκδοση HTTP/3.0. Η βασική διαφορά με τους πρωκατόχους του είναι ότι αλλάζει το πρωτόκολλο επικοινωνίας που χρησιμοποιεί όλα αυτά τα χρόνια, από TCP (Transfer Communication Protocoll) σε έναν συνδυασμό UDP (User Datagram Protocoll) και QUIC, μίας νέας τεχνολογίας που λύνει το πρόβλημα και βελτιστοποιεί τόσο το πρόβλημα της ασφάλειας των επικοινωνιών (TLS handshakes), όσο και της απώλειας πληροφορίας που μπορεί να υπήρχε λόγω UDP, πρωτοκόλλου που είναι γνωστό για την ταχύτερη απόδοσή του σε σχέση με το TCP, αλλά και το γεγονός ότι είναι πιο επιρρεπές σε σφάλματα. Η νέα αυτή έκδοση από τα αποτελέσματα του [5] φαίνεται να έχει ήδη καλύτερους χρόνους σε σχέση με τις παλαιότερες εκδόσης και ήδη το 28% του διαδικτύου αξιοποεί τις δυνατότητές του.

2.1.3 Κωδικοί Κατάστασης

Οι κωδικοί κατάστασεις (status codes) αποτελούν μέρος της απάντησης του server. Σου επιτρέπουν με μία ματιά να καταλάβεις αν το αίτημα που έχεις κάνει έχει επιστρέψει σωστά, ή έχει γίνει κάποιο λάθος στη μεριά του server. Υπάρχουν πέντε μεγαλύτερες κατηγορίες που στεγάζουν όλες τις υποπεριπτώσεις αυτών. Πιο συγκεκριμένα:

- **Εύρος 100-199**: Υποδηλώνουν ενημερωτική απάντηση σχετικά με τη λειτουργία του server
- **Εύρος 200-299**: Επιτυχή αιτήματα.
- **Εύρος 300-399**: Υποδηλώνουν την ανακατεύθυνση του μηνύματος του client. Συνήθως συνοδεύονται από το νέο url στο οποίο πρέπει να αποστείλουν το αίτημα

- **Εύρος 400-499**: Ανεπιτυχές αίτημα, που οφείλεται στον client. Ένα σύνηθες παράδειγμα είναι αίτησης πρόσβασης σε προστατευόμενους πόρους χωρίς κάποιου είδους αυθεντικοποίησης, ή χωρίς τα σωστά στοιχεία για αυθεντικοποίηση

- **Εύρος 500-599**: Ανεπιτυχές αίτημα, που οφείλεται στον server.

2.2 API

Ένα ακόμα πολύ σημαντικό συστατικό του διαδικτύου αποτελούν οι Διεπαφές Εφαρμογών Προγραμμάτων (Application Programming Interfaces - APIs). Πρόκειται για το σύνολο των ορισμών, κανόνων και πρωτοκόλλων για τη δημιουργία και ενσωμάτωση μίας εφαρμογής. Ουσιαστικά λειτουργεί ως το συμβόλαιο μεταξύ ενός συστήματος παροχής υπηρεσίας και του χρήστη του συστήματος αυτού, καθορίζοντας την απαραίτητη πληροφορία που απαιτεί για να λειουργήσει σωστά ο server αλλά και αντίστροφά, καθορίζοντας την απαραίτητη πληροφορία που απαιτεί ο client στην απάντηση που θα του επιστραφεί.

Είναι εμφανές λοιπόν ότι σε κάθε περίπτωση, αν θέλει κανείς να αλληλεπιδράσει με κάποιο σύστημα είτε για να αντλήσει πληροφορία, είτε για να στείλει πληροφορία (αποθήκευση ή τροποποίηση ήδη υπάρχουσας) θα πρέπει να υπάρχουν κανόνες που ορίζουν και καθιστούν πιο εύκολη και απλή την επικοινωνία αυτή.

Η έννοια του api δεν περιορίζεται φυσικά μόνο στο πλαίσιο του διαδικτύου, αλλά σε όλων των ειδών εφαρμογές που υπάρχει επικοινωνία ενός κεντρικού σημείου (server) με κάποιον χρήστη (client) που θέλει να κάνει χρήση των υπηρεσιών που αυτό προσφέρει.

Όσων αφορά τη διαθεσιμότητα και ασφάλεια των API, μπορούμε να διακρίνουμε τις εξής κατηγορίες:

- **Ανοιχτά (Open)**: Σε αυτού του τύπου διεπαφών λογισμικού, έχει ελεύθερη πρόσβαση καθένας, χωρίς να απαιτείται κάποια παραπάνω πληροφορία που αφορά την αυθεντικοποίηση ή ταυροποίηση του χρήστη. Επειδή ακριβώς η πληροφορία που παρέχεται (ανοιχτά) είναι τεράστια, χρησιμοποιούνται πολύ συχνά σε έρευνες, όπως αυτή [6] που αξιοποιεί ελεύθερα προσβάσιμα πόρους για να αναλύσει τα πιο διαδεδομένα APIs.

- **Δημόσια (Public)**: Μοιάζουν πολύ με τα **Ανοιχτού Τύπου API**, με τη μόνη διαφορά, τον περιορισμό πρόσβασης σε ορισμένα σημεία που απαιτούν κλειδιά προκειμένουν να γίνει σε αντίθεση με πριν αυθεντικοποίηση και ταυτοποίηση.

- **Ιδιωτικά (Private)**: Αφορούν διεπαφές λογισμικού που αναπτύσσονται και χρησιμοποιούνται μόνο εντός ενός κλειστού πλαισίου, όπως θα μπορούσε να είναι μία επιχείρηση ή κάποιο πανεπιστήμιο που παρέχει ορισμένες υπηρεσίες μόνο εντός του χώρου του. Πολλές φορές στη βιβλιογραφία αναφέρονται και ως **Εσωτερικά (Internal APIs)**

- **Εταιρικά (Partner)**: Είναι περιορισμένα στο πλήθος χρηστών που έχουν πρόσβαση σε αυτό. Χρησιμοποιούνται για επικοινωνία μεταξύ συστημάτων εταιριών/επιχειρήσεων για την ανάπτυξη και αξιοποίηση εφαρμογών και υπηρεσιών. Συνήθως η ασφάλεια σε όλες αυτές τις αλληλεπιδράσεις είναι αρκετά πιο αυστηρή.

- **Σύνθετα (Composite)**: Συνδυάζουν περισσότερα από ένα API αιτημάτων σε ένα, κάνοντας έτσι την επικοινωνία πιο αποδοτική (κερδίζοντας χρόνο από πολλά διαδοχικά API).

Αξίζει να σημειωθεί σε αυτό το σημείο ότι ο τρόπος δημιουργίας, η δομή καθώς και τα πρωτόκολλα που χρησιμοποιούν τα διαφόρα APIs που υπάρχουν στο διαδίκτυο δεν είναι πάντα κοινά. Υπάρχουν κάποιες γνωστές αρχιτεκτονικές και πρωτόκολλα όπως θα δούμε και στη συνέχεια, που πέρα από τη δομή των συστημάτων παρέχουν και κανόνες "Καλύτερων Πρακτικών" (Best Practices). Οι πιο γνωστές αριχτεκτόνικές παρατίθενται στη συνέχεια:

- **SOAP (Simple Object Access Protocol)**: Χρησιμοποιούνταν ευρέως στο παρελθόν πριν την εμφάνιση του REST. Η επικοινωνία μεταξύ server και client επιτυγχάνεται με την αποστολή αρχείων XML.

- **RPC (Remote Procedure Call)**: Αποτελεί πρωτόλλο επικοινωνίας που επιτρέπει σε ένα σύστημα (client) να καλεί διαδικασίες (procedures) ή συναρτήσεις σε ένα άλλο σύστημα (server) είτε αυτά είναι στο ίδιο δίκτυο, είτε απομακρυσμένα. Σκοπός είναι να κάνει τα δύο συστήματα να επικοινωνούν σαν να βρίσκονται στο ίδιο υπολογιστικό σύστημα. Επιγραμματικά η διαδικασία επικοινωνίας δύο συστημάτων ξεκινάει με ένα αίτημα στον server να εκτελέσει κάποια συγκεκριμένη λειτουργία, παρέχοντας ορίσματα που ίσως χρειαστούν για αυτό. Ο server τέλος επιστρέφει την απάντηση της παραπάνω διαδικασίας στον client. Σήμερα το πρωτόκολλο αυτό χρησιμοποιείται σε διάφορες εφαρμογές και αναπτύσσονται μάλιστα συνεχώς, σύχρονες τεχνολογίες που πατώντας πάνω στη βασική λειτουργία του πρωτοκόλλου κάνουν περαιτέρω βελτιστοποιήσεις (gRPC).

- **REST (Representational State Transfer)**: Η πιο διαδεδομένη μορφή API στο διαδίκτυο. Η λειτουργία του είναι απλή. Η χρήστης στέλνει αίτημα σε έναν απομακρυσμένο σύστημα (server). Το σύστημα αντιδράει και βάσει των δεδομένων που του στέλνει ο χρήστης, τρέχει διαδικασίες ώστε να παράξει το επιθυμητό αποτέλεσμα, το οποίο τέλος το αποστέλει στον client.

- **Websocket**: Αποτελεί πρωτόκολλο που επιτρέπει την αμφίδρομη επικοινωνία μεταξύ client και server. Σε αντίθεση με το REST API, που κάθε φορά που γίνεται ένα αίτημα στον server πρέπει να αποστέλλονται οι κατάλληλοι headers και να γίνεται η κατάλληλη σύνδεση μεταξύ των δύο συστημάτων, με τη χρήση των websockets η σύνδεση αυτή εκτελείται μόνο μία φορά στο αρχικό αίτημα που αποστέλλεται. Αξίζει να σημειωθεί ακόμα ότι θεωρείται ιδανικό για εφαρμογές που απαιτούν άμεση ενημέρωση [7] (real-time-applications) τόσο για τον προαναφερθέντα λόγο, βάσει του οποίου μειώνεται δραστικά ο

χρόνος επικοινωνίας μεταξύ των εμπλεκόμενων, όσο και για το γεγονός ότι μηνύματα μπορούν να σταλούν και από τη μεριά του client προς τον server αλλά και το ανάποδο (server σε client) χωρίς να χρειάζεται να γίνει κάποιο αίτημα πρώτα.

2.2.1 RESTful API

Πρόκειται για ένα τύπο αρχιτεκτονικής λογισμικού για APIs, που αποτελείται από κατευθυντήριες γραμμές και βέλτιστες πρακτικές για τη δημιουργία επεκτάσιμων εφαρμογών στο διαδίκτυο. Πρωτάθηκε το 2000, από τον Thomas Fielding [8] σαν ένας τρόπος για να κατευθύνει την ανάπτυξη εφαρμογών προκειμένου να υπάρχει ένα κοινός τρόπος "γραφής" ώστε να υπάρξη εξέλιξη στον τομέα αυτό ακόμα πιο γρήγορα και οργανωμένα. Το δομικό συστατικό του είναι το πρωτόκολλο HTTP που χρησιμοποιεί για να πραγματοποιεί κλήσεις μεταξύ συστημάτων.

Όπως προαναφέρθηκε ο τρόπος λειτουργίας στην απλούστερη έκδοσή του ξεκινάει με ένα αίτημα κάποιου χρήστη σε ένα σύστημα της επιλογής του, παρέχοντας την απαραίτητη πληροφορία προκειμένου ο server να μπορέσει να αναταποκριθεί επιτυχώς. Έπειτα, και αφού το σύστημα διεκπεραιώσει όλες τις εσωτερικές λειτουργίες του, απαντά μεταφέροντας πίσω στον χρήστη την επιθυμητή πληροφορία και ενημερώνοντας τον σχετικά με την κατάσταση του αιτήματος του (σε περίπτωση που κάτι πήγε λάθος ο χρήστης θα πρέπει να ενημερώνεται αναλόγως).

Ο βασικός τρόπος αλληλεπίδρασης με τους πόρους του συστήματος θα πρέπει να γίνεται μέσα από τις τέσσερις βασικές μεθόδους του HTTP (**GET**, **POST**, **PUT**, **DELETE**), χωρίς αυτό να απαγορεύει τη χρήση των υπόλοιπων. Κάθε σύστημα θα πρέπει τυπικά να υποστηρίζει CRUD λειτουργίες (Create, Read, Update, Delete)

Για να χαρακτηριστεί ένα API ώς RESTful θα πρέπει να ικανοποιεί τα παρακάτω, ενώ στο σχήμα 2.2 μπορούμε να τα δούμε σε ένα γενικότερο πλαίσιο:

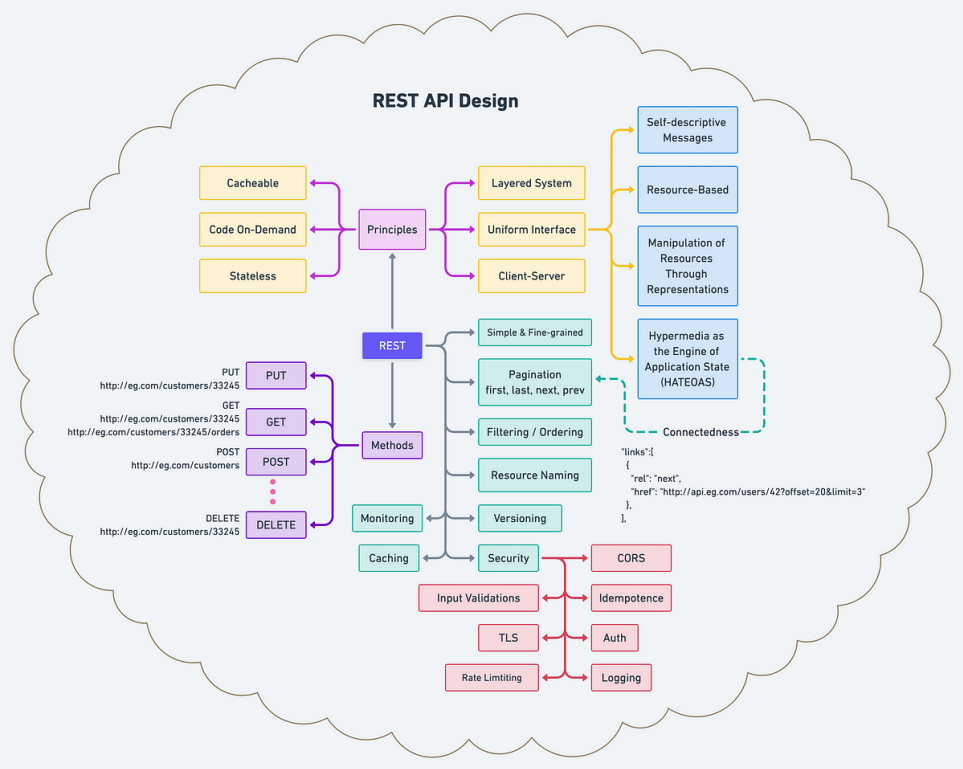
- **Client-Server**: Τα δύο αυτά υπολογιστικά συστήματα θα πρέπει να είναι ανεξάρτητα. Πρακτικά αυτό σημαίνει ότι ο client θα ασχοληθεί αποκλειστικά και μόνο με το κομμάτι παρουσιάσης της πληροφορίας, που του παρέχει ο server, στον χρήστη μέσα από διεπαφές γραφικού χαρακτήρα, ενώ ο server θα εκτελεί μόνο λειτουργίες που αφορούν την δημιουργία, τροποποίηση και διαγραφή των πόρων του συστήματος. Με αυτό τον διαχωρισμό πλέον ο client είναι πιο ελαφρής, καθώς διαχειρίζεται μόνο πληροφορία που του έρχεται έτοιμη από το σύστημα, και ο server μπορεί να κάνει πιο εύκολα scaling.

- **Cacheable**: ο client θα πρέπει να αποθηκεύει προσωρινά (cache) τις απαντήσεις που λαμβάνει από τον server για την αποφυγή συνεχών επαναλαμβανόμενων αιτημάτων, βελτιώνοντας έτσι την απόδοση του συστήματος.

- **Stateless**: Η πληροφορία της κατάστασης του συστήματος δεν θα πρέπει να αποθηκεύεται αλλά κάθε αίτημα θα πρέπει να περιέχει την απαραίτητη πληροφορία για να εκτελεστεί από τη μεριά του server. Η πληροφορία αυτή μπορεί να αποτελεί μέρος του ίδου του url, του body, των headers ή του query.

- **Πολυεπίπεδο Σύστημα (Layered System)**: Τα υπολογιστικά συστήματα που συμμετέχουν στη διαδικασία αυτή δεν θα πρέπει να γνωρίζουν αν είναι άμεσα

συνδεδεμένα μεταξύ τους ή υπάρχουν ενδιάμεσοι κόμβοι που παρεμβάλλονται. Το σύστημα θα πρέπει να λειτουργεί δίχως να έχει επίγνωση των γειτόνων του, περιμένοντας απλά την κατάλληλη πληροφορία για να λειτουργήσει, τόσο από τη μεριά του server όσο και από τη μεριά του client.



Σχήμα 2.2: Αρχές και Καλές Πρακτικές Σχεδίασης REST API

2.3 Avro

Αποτελεί μία μορφή αποθήκευσης πληροφορίας που λόγω της υλοποίησης του παράγει αρχεία πολύ μικρότερου μεγέθους από ότι θα αναμέναμε σε άλλου είδους μορφές αρχείων. Αναπτύχθηκε αρχικά για τη για την αποθήκευση και τη μετάδοση πληροφορίας εντός του framework Apache Hadoop, αλλά έκτοτε έχει χρήση και σε πλήθος άλλων εφαρμογών. Διαθέτει διεπαφές (API) που επιτρέπουν την ενσωμάτωση της τεχνολογίας αυτής σε προγμάμματα γραμμένα σε πλήθος γλωσσών, μερικές εκ των οποίων είναι οι Java, C/C++/C#, Python, PHP, Ruby, Rust και JavaScript.

Η βασικότερη διαφορά μεταξύ αρχείων avro και άλλων αρχείων αποθήκευσης πληροφορίας, όπως είναι τα JSON αρχεία που αποτελούν τον πιο διαδεδομένο τρόπο αποθήκευσης στο χώρο του διαδικτύου, είναι το γεγονός ότι διαθέτουν το schema των δεδομένων (data schema definition) που περιέχουν. Ξέροντας τη μορφή της προς αποθήκευσης πληροφορίας καταφέρνουν να γλιτώσουν χώρο στο τελικό προϊόν. Αυτό σε συνδύασμό με το γεγονός ότι η πληροφορία αποθηκεύεται συμπιεσμένη

σε δυαδική μορφή (binary format) την καθιστά αρκετά σημαντική σε περιπτώσεις που έχουμε μεγάλο πλήθος πληροφορίας που θέλουμε να αποθηκεύσουμε.

Τα βασικότερα πλεονεκτήματα του παρατίθενται στη συνέχεια ενώ η δομή ενός τέτοιου αρχείου φαίνεται στο σχήμα 2.3:

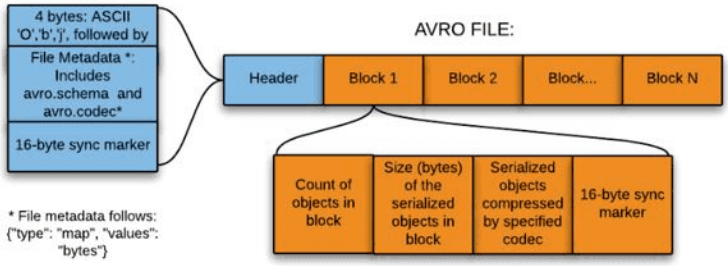
- Είναι χρήσιμος για την μετάδοση πληροφορίας, καθώς η αποθηκευμένη πληροφορία είναι αρκετά συμπυκνωμένη.

- Επιτρέπει την τροποποίηση και εξέλιξη του σχήματος των δεδομένων (schema evolution), δίνοντας τη δυνατότητα για αλλαγές και data migration που πολλές φορές καθίστανται επιτακτικά στη διάρκεια ζωής ενός προγράμματος που συνεχώς εξελίσσεται.

- Διαθέτει πλήθος τύπων δεδομένων απλών (primitive) και σύνθετων (complex) που μπορούν να χρησιμοποιηθούν

  - Primitive: boolean, int, long, float, double, bytes, string
  - Complex: record, enum, array, map, fixed, union

- Το schema εμπεριέχεται σε κάθε αρχείο. Κάνοντας χρήση αυτού κάθε χρήστης που έχει πρόσβαση σε ένα τέτοιο αρχείο μπορεί να διαβάσει τα περιεχόμενα του (εξαρχής binary, μη αναγνώσιμών από τον άνθρωπο) χωρίς να ξέρει από πριν τη μορφή της πληροφορίας.



Σχήμα 2.3: Δομή ενός avro αρχείου

2.4 Στατιστικες μετρικες

Στην υποενότητα αυτή θα δούμε τον τρόπο υπολογισμού μερικών κλασικών μετρικών της στατιστικής που χρησιμοποιούνται για την εξαγωγή συμπερασμάτων στο σύνολο των δεδομένων που αποθηκεύουμε. Σε όλους τους παρακάτω τύπους έχουμε δεδομένα x_i όπου, $i = 0, ..., k$

1. **Αριθμητική Μέση Τιμή**:

$$mean(x) = \frac{\sum_{n=1}^{k} x_n}{k} \tag{2.1}$$

2. **Διάμεσος**:

$$median(x) = \begin{cases} x_{\frac{n+1}{2}} & \text{για n περιττό} \\ \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2} & \text{για n ζυγό} \end{cases} \tag{2.2}$$

3. **Τυπική Απόκλιση Πληθυσμού**:

$$std(x) = \sqrt{\frac{\sum_{i=1}^{n} (x_i - mean(x))^2}{n-1}} \tag{2.3}$$

4. **Τεταρτημόρια**:

$$\begin{cases} q1(x) = mean(x_i) & \text{όπου i = 0, ..., k/2} \\ q3(x) = mean(x_j) & \text{όπου j = k/2 + 1, ..., k} \end{cases} \tag{2.4}$$

2.5 Εργαλεια

Στην τελευταία υποενότητα αυτού του κεφαλαίου θα μιλήσουμε για τα εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση του συστήματος Αυτόματης Παρακολούθησης που υλοποιήσαμε στο πλαίσιο της διπλωματικής αυτής εργασίας.

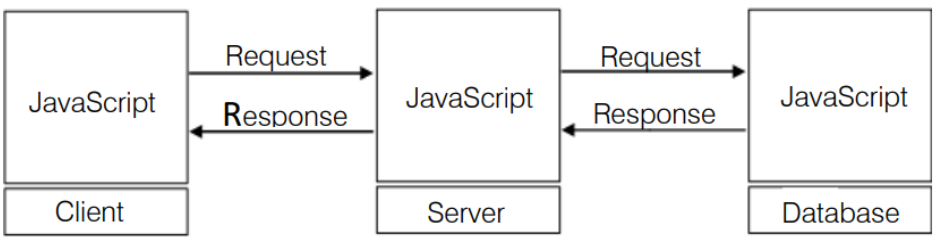
2.5.1 Node.js

Είναι ένα περιβάλλον εκτέλεσης (runtime) της προγραμματιστικής γλώσσας JavaScript. Πληθώρα μηχανικών λογισμικού το χρησιμοποιούνε σήμερα, για την γρήγορη και, σχετικά με άλλα εργαλεία, εύκολη ανάπτυξη διαδικτυακών εφαρμογών. Η Node.js μάλιστα διαθέτει το δικό της package manager (NPM - Node Package Manager), στον οποίο διαρκώς προστίθενται καινούργιες βιβλιοθήκες από χρήστες για χρήστες.

Είναι σχεδιασμένο για να μπορεί να χτίζει κλιμακούμενες (scalable) εφαρμογές, καθώς η αρχιτεκτονική του, επιτρέπει τη σύνδεση πολλών εξωτερικών συστημάτων/χρηστών και την εξυπηρέτηση αυτών ταυτόχρονα. Κάθε φορά που γίνεται κάποια σύνδεση στην εφαρμογή και κατεπακόλουθω κάποιο αίτημα στο σύστημα εκτελείται μία callback συνάρτηση προκειμένου να μπορέσει να απαντήσει πίσω. Όσο το σύστημα δεν δέχεται αιτήματα "κοιμάται" και περιμένει το επόμενο που θα έρθει.

Όλα όσα προαναφέρθηκαν την καθιστούν κατάλληλη για τη δημιουργία συστημάτων server. Αξίζει να σημειωθεί μάλιστα ότι για το σκοπό αυτό υπάρχουν πλήθος βιβλιοθηκών που παρέχουν έτοιμες συναρτήσεις και διεπαφές που κάνουν την διαδικασία ανάπτυξης ακόμα πιο εύκολη και γρήγορη. Πολλές φορές μάλιστα, κάποιες βιβλιοθήκες πέρα από βοηθητικές συναρτήσεις και υπορουτίνες, επηρεάζουν τον τρόπο σύγγραφής κώδικα, μέσα από APIs που παρέχουν. Αυτές χαρακτηρίζονται ως frameworks, και επιταχύνουν τόσο τη διαδικασία ελέγχου (testing), όσο και τη διαδικασία ανάπτυξης (development) του λογισμικού. Μερικές από τα πιο γνωστά backend framerworks είναι τα: Express, Jest, Koa, Socket.io, Meteor, Loopback

Τα παραπάνω σε συνδυασμό με το ότι η JavaScript είναι μία ευρέως διαδεδομένη και πολυχρησιμοποιούμενη γλώσσα προγραμματισμού στο διαδίκτυο δίνει την ευκαιρία σε developers να αναπτύξουν μία εφαρμογή πλήρως (frontend και backend) με τη χρήση ενός κοινού εργαλείου. Ένα τυπικό παράδειγμα εφαρμογής που μπορεί να αναπτυχθεί με τον τρόπο αυτό φαίνεται στο σχήμα 2.4.



Σχήμα 2.4: Σχεδίαση συστήματος με χρήση Node.js [9]

2.5.2 PM2

Η διαχείρηση διεργασιών (processes) που τρέχουν σε ένα υπολογιστικό σύστημα μπορεί να είναι δύσκολη και αρκετά περίπλοκη για κάποιον που δεν διαθέτει μεγάλη πείρα σε αυτό τον τομέα. Το PM2 ή αλλιώς Process Manager 2, αποτελεί ένα open source πρότζεκτ που κάνει την παραπάνω διαδικασία πιο απλή και κατανοητή, κερδίζοντας χρόνο για τον developer που μπορεί να τον αξιοποιήσει για την ανάπτυξη εφαρμογών.

Με την έννοια διαχείρηση διεργασιών, αναφερόμαστε σε όλες εκείνες τις ενέργειες που σχετίζονται με τον (πρόωρο ή όχι) τερματισμό και την παρακολούθηση ήδη υπάρχοντων διεργασιών (σχήμα 2.5), αλλά και τη δημιουργία καινούργιων. Προγράμματα όπως το pm2 προσφέρουν ακόμα δυνατότητες όπως είναι η αυτόματη επανεκκίνηση διεργασιών σε περίπτωση σφάλματος αποτρέποντας έτσι την ύπαρξη downtime των εφαρμογών που τρέχουμε.

Ένα από τα πιο χρήσιμα εργαλεία που παρέχει το PM2 είναι η λειτουργία cluster mode. Αν και δεν αναφέρθηκε πιο πάνω ο συγκεκριμένος διαχειριστής διεργασιών εξειδικεύεται σε Node.js εφαρμογές και στη διαχείρηση αυτών. Εξ ορισμού εφαρμογές γραμμένες σε JavaScript τρέχουν σε ένα thread στο υπολογιστικό σύστημα που εκτελούνται. Χρησιμοποιώντας όμως το cluster mode, μπορούμε να ξεκινήσουμε πολλαπλές διεργασίες που θα λειτουργούν ταυτόχρονα και θα μοιράζουν το φόρτο κατάλληλα (load-balancing) ώστε το τελικό σύστημα να έχει καλύτερη απόδοση και να μπορεί να εξυπηρετεί περισσότερο κόσμο.

id	name	namespace	version	mode	pid	uptime	↺	status	cpu	mem	user	watching
0	lychte-server	default	N/A	fork	10432	66s	0	online	0%	66.4mb	stavs	disabled
1	phiphisus-worker	default	N/A	fork	33936	66s	0	online	0%	68.9mb	stavs	disabled
2	sisiphus-worker	default	N/A	fork	34312	66s	0	online	0%	68.7mb	stavs	disabled

Σχήμα 2.5: Παράδειγμα Παρακολούθησης διεργασιών με τη χρήση του PM2

2.5.3 MongoDB

Η MongoDB αποτελεί μία Μη Σχεσιακή Βάση Δεδομένων (Non Relational Database), που χρησιμοποιεί documents, για να αποθηκεύσει πληροφορία, αντί στήλες και γραμμές όπως θα είχαμε σε κλασσικές SQL βάσεις δεδομένων. Είναι σχεδιασμένη για αποθήκευση δεδομένων μεγάλης κλίμακας και παράλληλη επεξεργασία δεδομένων μοιρασμένα σε έναν μεγάλο αριθμό από servers.

Τα documents που αναφέρθηκαν αποτελούν τον ακρογωνιαίο λίθο της Mongo, καθώς αποτελούν τη βασική μονάδα της αποθηκευμένης πληροφορίας. Μορφοποιούνται ως BJSON (Binary JSON) και παρέχουν πληθώρα τύπων δεδομένων που μπορείς να αποθηκεύσεις (string, integer, double, boolean, array, object, date, timestamp, null, binary). Κάθε βάση mongo μπορεί να περιέχει μία ή περισσότερες συλλογές (collections), τα δεδομένα των οποίων πρέπει να υπακούουν στο ίδιο σχήμα (schema). Επειδή όμως η βάση αυτή παρέχει δυνατότητες δυναμικού σχήματος (dynamic schema) μπορούμε να αποθηκεύουμε πληροφορία και να προσθέτουμε πεδία (fields) που δεν είχαμε ορίσει από την αρχή δημιουργίας του συστήματος.

Μερικοί από τους λόγους που την επιλέξαμε είναι οι εξής:

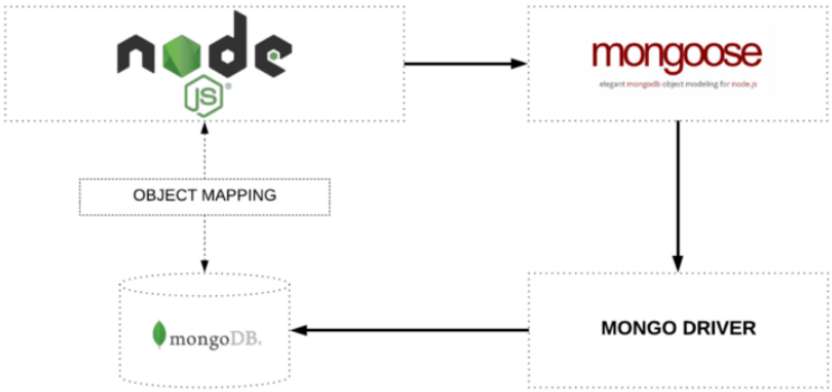
- **Document Oriented Storage**: η αποθήκευση και διαχείρηση των δεδομένων (στο πλαίσιο της εφαρμογής) είναι εύκολη καθώς στηρίζεται σε δεδομένα σε μορφή JSON.

- **Ευρετήρια (Indexes)**: Μπορείς κατά τη διαδικασία του στησίματος της βάσης (αλλά και μετέπειτα) να ορίσεις ευρετήρια σε πεδία που χρησιμοποιούνται συχνά σε queries προκειμένου να βελτιώσεις την απόδοση του συστήματος στο σύνολο. Όσο πιο γρήγορη είναι βάση, τόσο καλύτερη θα είναι η ανταπόκριση του συστήματος.

- **Ομοιοτυπία (Replication) και μεγάλη Διαθεσιμότητα**: Δημιουργώντας πολλαπλά αντίτυπα των αποθηκευμένων δεδομένων σε περισσότερο από έναν servers, μπορούμε να είμαστε σίγουροι ότι θα έχουμε πρόσβαση στην πληροφορία ακόμα και αν η κίνηση (data traffic) της εφαρμογής αυξηθεί.

- **Αυτόματο sharding**: Διασπώντας την αποθηκευμένη πληροφορία και έχοντας τμήματα αυτής σε διαφορετικούς server μας δίνεται η δυνατότητα να κάνουμε οριζόντιο scaling πολύ πιο εύκολα.

- **Εύκολο ενσωμάτωση**: Η mongo διαθέτει βιβλιοθήκες στο περιβάλλον του node.js που καθιστούν τη χρήση και το στήσιμό της πολύα απλό. Πέρα από τη mongo, την επίσημη βιβλιοθήκη που υπάρχει, διατίθεται και η mongoose που αποτελεί μία Object Data Modelling (ODM) βιβλιοθήκη για τη Mongo και τη nodejs (σχήμα 2.6)



Σχήμα 2.6: Η mongoose λειτουργεί ώς ένα abstract layer μεταξύ της Node και των driver της mongo προκειμένου η επικοινωνία μεταξύ των δύο να γίνεται πιο εύκολα

2.5.4 Google Cloud Storage

Πέρα από την κλασική βάση δεδομένων που προαναφέραμε θέλουμε να αποθηκεύουμε ιστορικά δεδομένα, δεδομένα δηλαδή που δεν θα δείχνουμε άμεσα στον χρήστη, αλλά θα κρατάμε για να υπολογίζουμε μετρικές και στατιστικά σημαντικά αποτελέσματα στο σύνολο όλης της μέχρι τώρα αποθηκευμένης πληροφορίας.

Σε αυτό λοιπόν το σημείο έρχεται το Google Cloud Storage (GCP), το οποίο παρέχει μεταξύ άλλων δυνατότητες Αποθήκευσης Αρχείων (File Storage). Τα δεδομένα αυτά δεν χρειάζεται να υπακούουν σε κάποιο σχήμα ενώ παράλληλα ο τρόπος αρχειοθέτησης των δεδομένων ταυτίζεται με ένα κλασικό σύστημα αρχείων ενός υπολοσιστικού συστήματος. Διαθέτει paths και τύπους αρχείων όπως ακριβώς και οι υπολογιστές και όλες οι συσκευές που χρησιμοποιούμε. Για να διαβάσεις κανείς πληροφορία, χρειάζεται να ξέρει μόνο το μονοπάτι που οδηγεί στο επιθυμητό αρχείο καθώς και τη μορφή αυτού. Οι υποστηριζόμενοι τύποι αρχείων μέχρι τώρα είναι οι εξής:

- Binary
- Flat
- JSON
- Avro
- Parquet

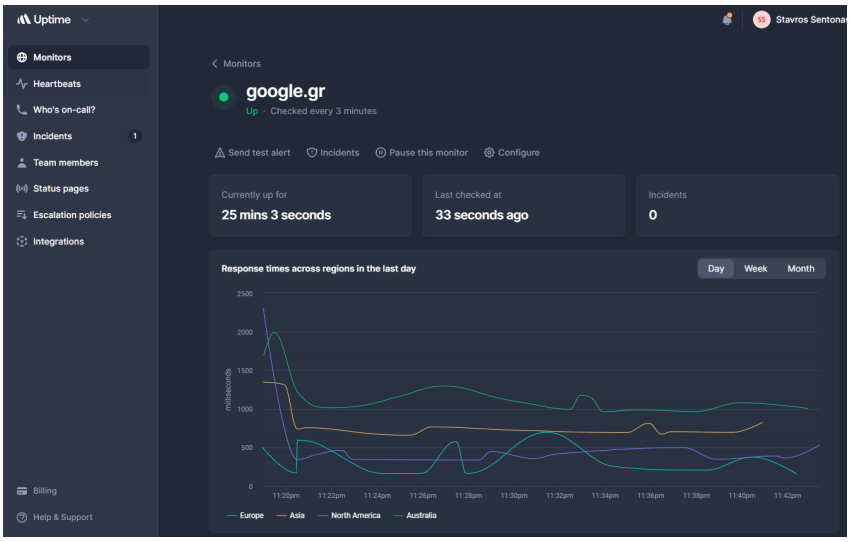
3
Βιβλιογραφική Αναζήτηση Τεχνολογιών Αιχμής

Πριν αναλυθεί η υλοποίηση του συστήματος που δημιουργήσαμε για την Ενεργή Παρακολούθηση Εφαρμογών ως Υπηρεσίες (SaaS) και ιστοσελιδών που εδρεύουν στο διαδίκτυο, θα πασουσιάσουμε τεχνολογίες που έχουν χτιστεί ήδη για το σκοπό αυτό και θα δείξουμε τους τομείς στους οποίους διαφέρει το δικό μας σύστημα.

Εφαρμογές τέτοιου τύπου έχουν αναπτυχθεί κυρίως από εταιρίες αλλά υπάρχουν πολλά open source project μικρότερου βεληνεκούς που επιτυγχάνουν τον ίδιο στόχο. Στη συνέχεια θα αναφερθούμε κυρίως στα πιο γνωστά και διαδεδομένα εργαλεία παρουσιάζοντας τα δυνατά τους σημεία και περιγράφοντας τις λειτουργίες που παρέχουν.

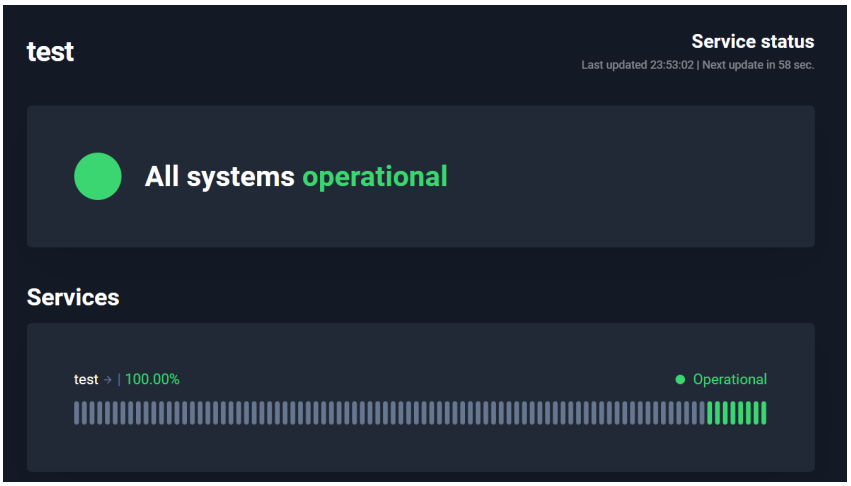
- **Better Uptime**: Προσφέρει εύκολη ενσωμάτωση των ιστοσελιδών που θέλει κανείς να παρακολουθήσει. Λειτουργεί κάνοντας ping κάθε τριάντα δευτερόλεπτα στο url που ορίζει ο χρήστης και παρουσιάζει τα παραγόμενα δεδομένα σε ευπαρουσίαστα διαγράμματα (σχήμα 3.1). Ένα από τα μεγάλα πλεονεκτήματα που έχει αφορά τη δυνατότητα για πολλαπλά ping από διαφορετικές περιοχές του κόσμου (Ευρώπη, Ασία, Βόρεια Αμερική, Αυστραλία) ώστε οι χρήστες να διαθέτουν μία πιο πλήρη εποπτεία του υπό μελέτη συστήματος/ιστοσελίδας. Αξίζει να σημειωθεί ότι παρέχει και μηχανισμούς ενημέρωσης για να ειδοποιεί το χρήστη σε περίπτωση μη απόκρισης τους συστήματος, μέσα από mail, εφαρμογές chatting και τηλεφωνικών κλήσεων.

- **Uptime Robot**: Επιτρέπει πέρα από την επιλογή του url και την επιλογή παραπάνων παραμέτρων που επηρεάζουν την απάντηση που θα επιστρέψει το υπό μελέτη σύστημα. Οι παράμετροι αυτοί σχετίζονται με τους headers του μηνύματος που αποστέλλεται και πιο συγκεκριμένα με αυτούς που αφορούν την αυθεντικοποίηση του χρήστη (στην προκειμένου περίπτωση του συστήματος παρακολούθησης). Πέρα από αυτά μπορεί να καθορίσει την επιθυμητή



Σχήμα 3.1: Παράδειγμα χρήσης του εργαλείου Better Uptime

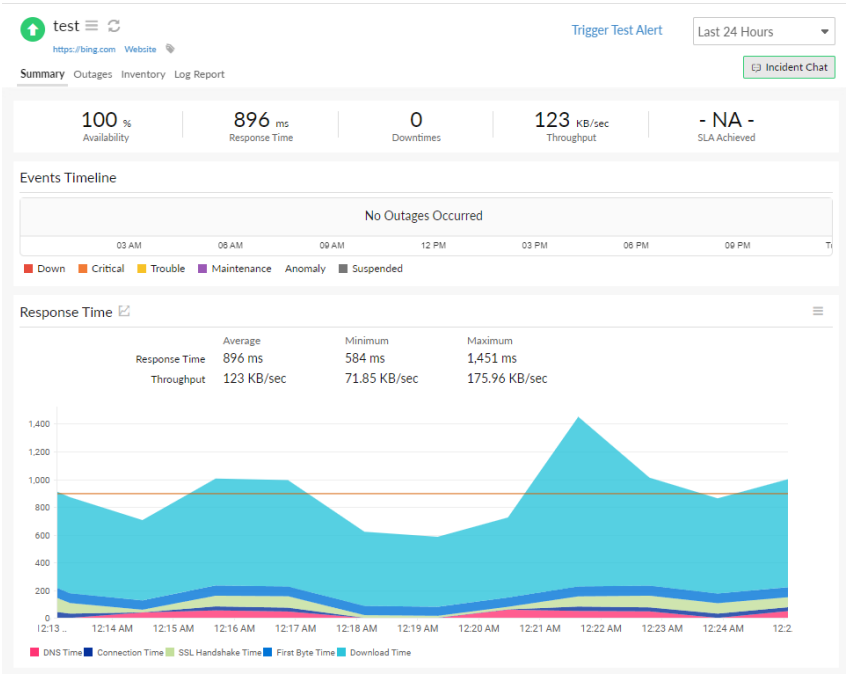
http κατάσταση της απόκρισης της ιστόσελίδας και το χρόνο που θα παρεμβάλλεται μεταξύ διαδοχικών αιτημάτων (pings). Τέλος διαθέτει κάποια βασικά διαγράμματα που σχετίζονται με το αν η απόκριση του συστήματος είναι ορθή ή όχι (σχήμα 3.2)



Σχήμα 3.2: Παράδειγμα χρήσης του εργαλείου Uptime Robot

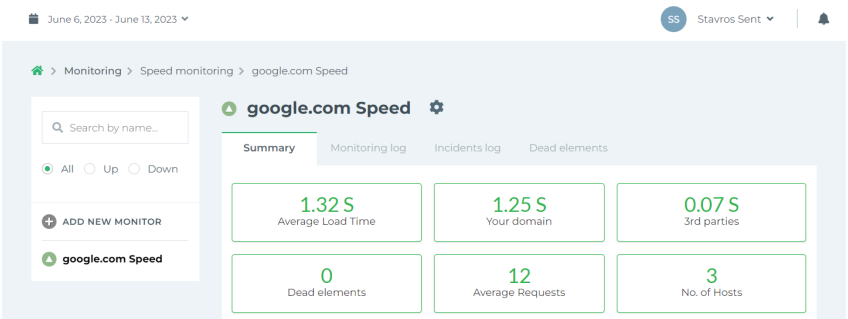
- **Site24x7**: Διαθέτει μετρικές, που αφορούν τη μέγιστη, ελάχιστη τιμή του χρόνου απόκρισης του συστήματος, καθώς και τη μέση τιμή του, ενώ παράλληλα δίνει μία εικόνα του throughput του συστήματος. Δεν λείπει φυσικά και ένα διάγραμμα απόκρισης χρόνου (σχήμα 3.3) που καθιστα τα δεδομένα που συλλέγονται πιο εύκολα στην κατανόηση και οπτικοποίηση.

- **Uptimia**: Πέρα από τα κλασικού τύπου http αιτήματα μπορεί να κάνει ελέγχους παρακολούθησης (uptime monitoring) σε DNS, UDP, TCP και email με απόσταση έως και τριάντα δευτερολέπτων (μεταξύ αιτημάτων). Αξίζει να



Σχήμα 3.3: Παράδειγμα χρήσης του εργαλείου Site24x7

σημειωθεί ότι η συγκεκριμένη εφαρμογή έχει δυνατότες και παθητικής παρακολούθησης (RUM). Αρχικά επιλέγουμε το site το οποίο θέλουμε να παρακολουθήσουμε καθώς και τα δεδομένα για τα οποία θέλουμε να ενημερωνόμαστε/παρακολουθούμε. Αυτά σχετίζονται κυρίως με σφάλματα ή καταστάσεις στις οποίες βρίσκεται το σύστημα και μπορεί να δηλώνουν κάποιο πρόβλημα. Καταστάσεις όπως η μειωμένη απόδοση του συστήματος εξαιτίας του χρόνου που παίρνει για να φορτώσει δεδομένα ή η απότομη πτώση του πλήθους των χρηστών μίας σελίδας. Για να επιτύχει τέτοιας μορφής ελέγχους παράγει (ανάλογα με τον τύπο των ελέγχων που επιλέγουμε) ένα script γραμμένο σε JavaScript που τοποθείται στην αρχή της ιστοσελίδας την οποία θέλουμε να παρακολουθήσουμε. Με αυτό τον τρόπο έχουν τη δυνατότητα συλλογής δεδομένων πραγματικών χρηστών. Στο πλαίσιο της Ενεργής βέβαια παρακολούθησης η έλλειψη διαγραμμάτων καθιστά το εργαλείο λίγο πιο δύσκολο στην κατανόηση της μεγαλύτερης εικόνας (σχήμα 3.4).

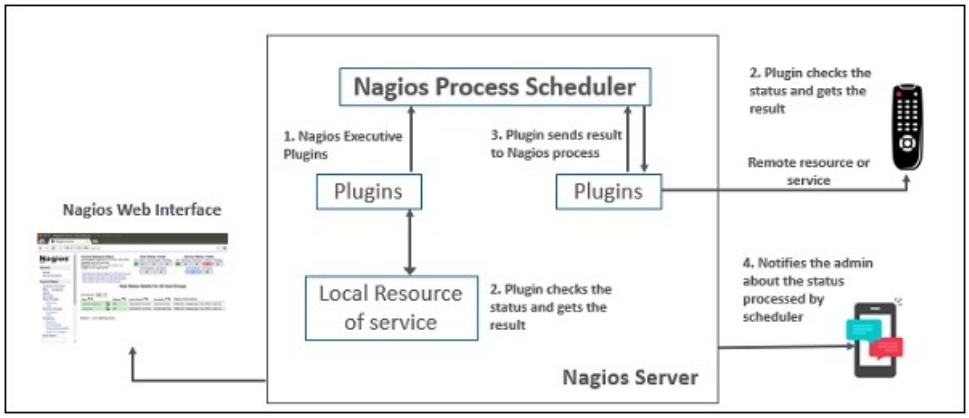


Σχήμα 3.4: Παράδειγμα χρήσης του εργαλείου Uptimia

Παραπάνω αναφέρθηκαν μερικά μόνο κάποια από τα εργαλεία που υπάρχουν σήμερα για την Ενεργη Παρακολούθηση του Χρόνου και Απόκρισης Ιστοτόπων και Διαδικτυακών Εφαρμογών. Σε αυτά θα πρέπει να προστεθούν πληθώρα εφαρμογών όπως τα: **StatusCake**, **SemaText**, **Uptrends**, **Dotcom-monitor**, **Updown**, **Datadog Synthetics**. Οι δυνατότες που προσφέρουν ως επί το πλείστον μπορούν περιγραφούν πλήρως από όσα αναπτύξαμε προηγουμένως, για αυτό το λόγω δεν θα αναφερθούμε περαιτέρω.

Πρέπει σε αυτό το σημείο όμως να τονίσουμε ότι όσα προαναφέρθηκαν αποτελούν προϊόντα εταιριών. Αυτό όμως δεν σταματάει την ανάπτυξη open source projects που υλοποιήθηκαν από χρήστες είτε ως προσωπικά project, είτε ως project μίας μεγαλύτερης ομάδας από developers. Έτσι και στο πλαίσιο της Ενεργής Παρακολούθησης μερικά από τα πιο γνωστά και δημοφιλή μεταξύ developers αποθετήρια αποτελούν τα:

- Upptime: Αποτελεί μία ενδιαφέρουσα προσέγγιση στο πρόβλημα της διαχείρησης των schedulers που θα πρέπει να έχει το σύστημα για να κάνει αιτήματα ανά ένα συγκεκριμένο και σταθερό χρονικό διάστημα. Προκειμένου να επιτύχει κάτι τέτοιο αξιοποιεί τις δυνατότητες του GitHub (cloud-based υπηρεσία αποθήκευσης git αποθετηρίων) και των actions που αυτό σου επιτρέπει να εκτελείς κάθε πέντε λεπτά. Έτσι λοιπόν ανά πέντε λεπτά (ελάχιστος χρόνος ελέγχου απόκρισης) τρέχει αυτοματοποιημένα και μέσω του GitHub (ανεξάρτητα από το σύστημα αυτό) μία διαδικασία που κάνει αιτήματα στην σελίδα που ο χρήστης ορίζει. Φυσικά τα αποτελέσματα αυτά αποθηκεύονται και ανά έξι ώρες παράγονται διαγράμματα που μπορείς να τα δεις μέσα από μία σελίδα που παράγεται αυτόματα.

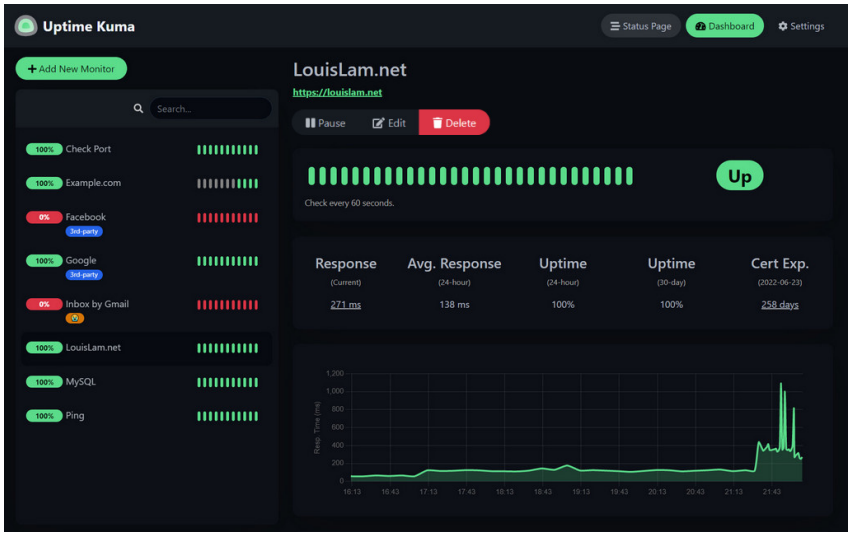


Σχήμα 3.5: Αρχιτεκτόνική Συστήματος Nagios

- Nagios: Είναι ένα πρόγραμμα γραμμένο στη γλώσσα προγραμματισμού C. Πέρα από το backend, που θα δούμε στη συνέχεια διαθέτει και Γραφικό Περιβάλλον Χρήστη (Graphical User Interface - GUI). Αποτελείται από ένα σύστημα ενός server (nagios server), οποίος λειτουργεί σαν ένας scheduler που στέλνει σήματα για να ξεκινήσει την εκτέλεση plugins στα απομακρυσμένα συστήματα που θέλουμε να παρακολουθήσουμε. Μόλις τα plugins δεχτούν απάντηση επιστρέφουν στον server ο οποίος προωθεί την πληροφορία στο

GUI για να τη δούνε και οι χρήστες. Δεν χρησιμοποιείται κάποια βάση δεδομένων, καθώς η πληροφορία που μαζεύεται αποθηκεύεται μόνο σε logs στο σύστημα που τρέχει την υπηρεσία αυτή. Την αρχιτεκτονική σου συστήματος μπορούμε δούμε στην σχήμα 3.5

- Kuma Uptime: Αποτελεί μία εύκολή να χρησιμοποιήσεις και να στήσεις self-hosted εφαρμογή γραμμένη σε JavaScript για Ενεργή Παρακολούθηση. Για την αποστολή των αιτημάτων σε απομακρυσμένες (στο διαδίκτυο) σελίδες χρησιμοποιεί **child_proccesses**, ένα api δήλαδή που περιλαμβάνει η Node.js για τη δημιουργία διεργασιών (processes) εντός άλλων διεργασιών. Η εφαρμογή περιλαμβάνει backend και frontend, στο οποίο μπορείς να δεις τα αποτελέσματα του monitoring. Αξίζει να σημειωθεί ότι τα δεδομένα αποθηκεύονται σε βάση SQLite, ένα προσωρινό σύνολο δεδομένων που υφίσταται μόνο στο πλαίσιο εκτέλεσης μίας εφαρμογής. Αποτελεί μία server-less βάση δεδομένων και είναι άρρηκτα συνδεδεμένη με την εφαρμογή στην οποία υπάρχει.



Σχήμα 3.6: Παράδειγμα χρήσης του εργαλείου Kuma Uptime

Κάθε σύστημα που αναφέρθηκε έχει πλεονεκτήματα αλλά και μειονεκτήματα σε σχέση με άλλα. Αυτό που θέλαμε να διαθέτει ιδανικά ένα σύστημα Παρακολούθησης, βάση όλων αυτών που είδαμε μέχρι τώρα, είναι τα εξής:

1. Σταθερό και Αξιόπιστο σύστημα διαχείρησης προγραμματισμού (scheduling system) που θα καθορίζει το πότε πρέπει να ξεκινάνε τα αιτήματα προς τα εξωτερικά υπό παρακολούθηση συστήματα.

2. Κάποια μορφή βάσης δεδομένων στην οποία θα αποθηκεύουμε τα δεδομένα που συλλέγουμε.

3. Χρήση ιστορικών δεδομένων για τον υπολογισμό μετρικών στατιστικής φύσης σε βάθος χρόνου.

4. Προβολή των δεδομένων σε ευπαρουσίαστα και εύπεπτα διαγράμματα που θα βοηθούν το χρήστη να αντιλαμβάνεται γρήγορα την κατάσταση των συστημάτων του

5. Διάθεση τρόπων ειδοποίησης του χρήστη σε περίπτωση που κάποιο από τα συστήματα δεν ανταποκρίνεται

6. Δυνατότητα για οριζόντια κλιμάκωση (**horizontal scaling**)

7. Ελαχιστοποίηση downtime του συστήματος

Πολλές απο τις εφαρμογές που είδαμε καλύπτουν σε κάποιο βαθμό μερικά από τα παραπάνω, αλλά καμία δεν τα καλύπτει όλα ταυτόχρονα. Τα εργαλεία που αναπτύχθηκαν στο πλαίσιο εταιριών έχουν δυνατότητες κλιμάκωσης αλλά στα περισσότερα αν όχι σε όλα μπορείς να δεις πληροφορία μέχρι ένα συγκεκριμένο χρονικό διάστημα πίσω στο χρόνο, κρύβοντας δεδομένα που είτε δεν αποθηκεύουν πλέον είτε δεν μπορούν να αντλήσουν αρκετά γρήγορα. Απο την άλλη οι περισσότερες open source εφαρμογές που αναφέραμε δεν μπορούν να κάνουν τόσο εύκολα scaling καθώς είναι φτιαγμένες να δουλεύουν για περιορισμένο πλήθος χρηστών.

Στη διπλωματική αυτή εργασία λοιπόν καλούμαστε να δημιουργήσουμε ένα τέτοιο υπολογιστικό σύστημα που θα καλύπτει τα προαναφερθέντα.

Βιβλιογραφία

[1] Santhosh S and Narayana Swamy Ramaiah. "*Cloud-Based Software Development Lifecycle: A Simplified Algorithm for Cloud Service Provider Evaluation with Metric Analysis*". Big Data Mining and Analytics, 6(2):127–138, june 2023.

[2] Sorin POPA. "*WEB Server monitoring*". Annals of University of Craiova - Economic Sciences Series, 2(36):710–715, may 2008.

[3] Siddhesh Vaidya and Prabhat Padhy. "*View towards Synthetic Monitoring using HTTP Archive*". International Journal of Engineering Research and Technology, august 2022.

[4] Daniel Stenberg. "*HTTP2 explained*". ACM SIGCOMM Computer Communication Review, 44:120–128, 07 2014.

[5] Martino Trevisan, Danilo Giordano, Idilio Drago, and Ali Safari Khatouni. "*Measuring HTTP/3: Adoption and Performance*", 02 2021.

[6] Andy Neumann, Nuno Laranjeiro, and Jorge Bernardino. "*An Analysis of Public REST Web Service APIs*". IEEE Transactions on Services Computing, 14:957–970, 07 2021.

[7] Liu Qigang and Xiangyang Sun. "*Research of Web Real-Time Communication Based on Web Socket*". International Journal of Communications, Network and System Sciences, 05:797–801, 01 2012.

[8] Roy Thomas Fielding and Richard N. Taylor. "*Architectural Styles and the Design of Network-Based Software Architectures*". PhD thesis, University of California, Irvine, 2000. AAI9980887.

[9] Hezbullah Shah and Tariq Soomro. "*Node.js Challenges in Implementation*". Global Journal of Computer Science and Technology, 17:72–83, 05 2017.