ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ & ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ

Διπλωματική Εργασία

Επικοινωνία Πάνω Από το Διαδίκτυο των Πραγμάτων με Χρήση Διεπαφών Τύπου REST

Περίπτωση χρήσης:
Παρακολούθηση Δεδομένων Αισθητήρων σε Περιβάλλον Νεφοϋπολογισμού

Φοιτήτρια: Μαρία Πολυχρονάκη
ΑΜ: 50344164

Επιβλέπων Καθηγητής

Πατρικάκης Ζ. Χαράλαμπος
Καθηγητής

ΑΘΗΝΑ-ΑΙΓΑΛΕΩ, ΟΚΤΩΒΡΙΟΣ, 2020

UNIVERSITY OF WEST ATTICA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

Diploma Thesis

Communication Over the Internet of Things Using RESTful APIs
Use Case: Sensor Data Monitoring in Cloud Computing Environment

Student: Maria Polychronaki
Registration Number: 50344164

Supervisor

Charalampos Z. Patrikakis, Dr.Ing
Professor

ATHENS-EGALEO, OCTOBER, 2020

ΔΗΛΩΣΗ ΠΕΡΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ ΚΑΙ ΛΟΓΟΚΛΟΠΗΣ

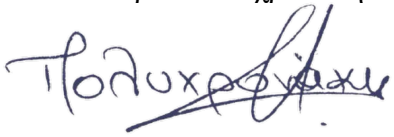
Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπόγραφα ότι η παρούσα εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα αποκλειστικά και ότι είμαι ο αποκλειστικός συγγραφέας του κειμένου της.

Η εργασία μου δεν προσβάλλει οποιασδήποτε μορφής δικαιώματα πνευματικής ιδιοκτησίας, προσωπικότητας ή προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής ή λογοκλοπής.

Κάθε βοήθεια που έλαβα για την ολοκλήρωση της εργασίας είναι αναγνωρισμένη και αναφέρεται λεπτομερώς στο κείμενό της. Ειδικότερα, έχω αναφέρει ευδιάκριτα μέσα στο κείμενο και με την κατάλληλη παραπομπή όλες τις πηγές δεδομένων, κώδικα προγραμματισμού Η/Υ, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών που χρησιμοποιήθηκαν, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης, και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Επιπλέον, όλες οι πηγές που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης κατά τα διεθνή πρότυπα.

Τέλος δηλώνω ενυπόγραφα ότι αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της είναι προϊόν λογοκλοπής.

Ημερομηνία 05/10/2020
Μαρία Πολυχρονάκη



(Υπογραφή)

Ευχαριστίες

Θα ήθελα να ευχαριστήσω προσωπικά όλα τα άτομα που συνέβαλαν είτε έμμεσα είτε άμεσα στην πραγματοποίηση της διπλωματικής αυτής εργασίας, αφού ακόμα και η πιο μικρή καθοδήγηση συνέβαλε στο αποτέλεσμα αυτό. Ιδιαίτερες ευχαριστίες θα ήθελα να εκφράσω στον επιβλέποντα καθηγητή κ. Χ. Πατρικάκη, ο οποίος μου έδωσε την ευκαιρία να ασχοληθώ με ένα τόσο ενδιαφέρον και καίριο θέμα στον τομέα της τεχνολογίας, καθώς επίσης και την επιστημονική του καθοδήγηση με όλα τα απαραίτητα εφόδια. Επίσης, ευχαριστίες οφείλω και στους ερευνητές Δρ. Δημήτριο Κόγια και Δρ. Παναγιώτη Κασνέση, για την συνεργασία τους και το χρόνο που διέθεσαν, προκειμένου να μου δώσουν επεξηγήσεις και μερικές από τις εξειδικευμένες γνώσεις τους όχι μόνο πάνω στα θέματα της διπλωματικής, αλλά και σε άλλους παραπλήσιους τομείς που αφορούν αυτό.

Σας ευχαριστώ θερμά.

Περίληψη

Ο τομέας του Διαδικτύου των Πραγμάτων (ΔτΠ), αποτελεί αναμφίβολα έναν από τους πιο ανερχόμενους τομείς της σύγχρονης εποχής. Αυτό συμβαίνει διότι αποτελεί τη συμβολή πολλών συστατικών, όπως το λογισμικό (π.χ. εφαρμογές), το υλικό (π.χ. αισθητήρες, μικροελεγκτές) και το Διαδίκτυο, με τέτοιο τρόπο ώστε να επηρεάζει και να διευκολύνει την καθημερινότητά μας. Στο κέντρο όλων αυτών βρίσκονται έννοιες όπως τα Δεδομένα, η Επικοινωνία, η Αλληλεπίδραση, αλλά και η Ασφάλεια των Δεδομένων που αποκτάει ιδιαίτερο νόημα στον «ευαίσθητο» κόσμο του ΔτΠ.

Καθώς όμως το ΔτΠ αποτελεί συνδυασμό αυτών των συστατικών είναι πολύ σημαντικό να δίνουμε προσοχή σε κάθε ένα από αυτά ξεχωριστά, προκειμένου να τα μελετήσουμε αλλά και να τα εξελίξουμε όσο το δυνατόν περισσότερο. Έτσι, μπορούμε να πούμε πως αν όλα τα επιμέρους «κομμάτια» που απαρτίζουν ένα σύστημα ΔτΠ αποδίδουν το βέλτιστο δυνατό αποτέλεσμα, τότε και το ίδιο το σύστημα έχει τις προοπτικές για εξαιρετική απόδοση και λειτουργία.

Η Επικοινωνία είναι ένα από αυτά τα επιμέρους κομμάτια και πιο συγκεκριμένα είναι αυτό θα μας απασχολήσει στα πλαίσια της διπλωματικής αυτής. Στο ΔτΠ επικοινωνία υπάρχει τόσο μεταξύ μηχανών/πραγμάτων, όσο και μεταξύ Ανθρώπου – Μηχανής.

Το θέμα στο οποίο εστιάζει η διπλωματική αυτή είναι η Επικοινωνία σε ένα σύστημα ΔτΠ με τη βοήθεια της Αρχιτεκτονικής REST. Βασιζόμενοι σε αυτήν μπορούμε να κατασκευάσουμε εύκολα Διεπαφές Προγραμματισμού Εφαρμογών (Application Programmable Interfaces, APIs), μέσω των οποίων πραγματοποιείται η απομακρυσμένη, και κυρίως πάνω από το Διαδίκτυο, μεταφορά δεδομένων μεταξύ των διαφόρων επί μέρους τμημάτων του συστήματος.

Μάλιστα, στα πλαίσια των προαναφερθέντων, πραγματοποιείται η σχεδίαση και η ανάπτυξη διεπαφών τύπου REST, προκειμένου να φανούν τα πλεονεκτήματα της χρήσης αυτής μέσα σε ένα περιβάλλον νεφοϋπολογισμού, βασισμένο να υποστηρίζει την πλατφόρμα Openstack.

Λέξεις – κλειδιά

Διαδίκτυο των Πραγμάτων, Διαδίκτυο, Επικοινωνία, Αρχιτεκτονική REST, Java, Node – RED, Python, RESTful API, Προγραμματισμός, Περιβάλλον Νεφοϋπολογισμού, Νεφοϋπολογιστική Υποδομή, Openstack

Abstract

Internet of Things (IoT) is continuously drawing attention in the field of modern Information and Communication Technologies (ICT). Its basic characteristic is its ability to combine various different technologies of software (e.g. applications), hardware (e.g. microcontrollers, sensors) and the Internet, in such a way that the result can substantially improve people's everyday lives. In the center of all these fields that IoT brings together, lay concepts like Data, Communication, Interaction and Data Security, while the latter becomes a major concept in the field of IoT, since the nature of the data coming through most of these systems are sensitive and sometimes personal.

However, the union of all these technologies increases the need of being able to exploit and develop as much as possible each and every single one of them separately. Consequently, if every part of an IoT system is developed properly and to the full extent of its capacity, then the system itself can produce its maximum possible performance.

Communication is one of the most vital parts of an IoT system, but more importantly, it is the property on which the ability of "talking" and "listening" to Things is depended on. Some of the components will have to interact with each other (e.g. M2M Communication), whereas some other components will have to interact with Humans (e.g. Human – Machine Communication)

The topic focused in this thesis is the Communication in an IoT system with the help of REST Architecture, due to which the easy design Application Programmable Interfaces (APIs) is possible. Using these, we can achieve remote Communication and above all, transfer data over the Internet between the different components of not only an IoT, but any system. Moreover, RESTful APIs will be developed to present the advantages of using REST architecture in a cloud computing environment used to support the Openstack platform.

Keywords

Internet of Things, Internet, Communication, REST Architecture, RESTful API, Java, Node – RED, Python, Programming, Cloud Computing Environment, Datacenter, Openstack

Περιεχόμενα

Κατάλογος Πινάκων

Κατάλογος Εικόνων

1 ΕΙΣΑΓΩΓΗ

Η παρούσα διπλωματική εργασία εκπονείται στα πλαίσια του προπτυχιακού προγράμματος σπουδών του τμήματος Ηλεκτρολόγων και Ηλεκτρονικών Μηχανικών του Πανεπιστημίου Δυτικής Αττικής. Το μεγαλύτερο ερέθισμα για το θέμα της εργασίας αποτέλεσε το μάθημα Διαδίκτυο των Πραγμάτων (ΔτΠ), με σημαντική τη συμβολή του περιεχομένου των μαθημάτων Δίκτυα Η/Υ, Αντικειμενοστραφής Προγραμματισμός, Δομές Δεδομένων.

1.1 Αντικείμενο της διπλωματικής εργασίας

Το θέμα της διπλωματικής αυτής εργασίας περιστρέφεται γύρω από το Διαδίκτυο των Πραγμάτων, το οποίο είναι μία από τις τεχνολογίες που προβλέπεται πως θα στηρίξει τις μελλοντικές εξελίξεις στον τομέα της Τεχνολογίας. Ήδη, το ΔτΠ έχει συνεισφέρει τόσο στον ίδιο τον τεχνολογικό τομέα όσο και σε άλλους τομείς βοηθώντας σημαντικά πολλές σημαντικές διεργασίες των ανθρώπων.

1.2 Σκοπός και στόχοι

Το ΔτΠ θα ενισχύσει σημαντικά την καθημερινότητά των ανθρώπων, έχοντας τη δυνατότητα να βελτιώσει σημαντικά τον τρόπο με τον οποίο λειτουργεί η κοινωνία σε ένα ευρύ φάσμα εφαρμογών. Σκοπός της παρούσας διπλωματικής εργασίας είναι να αναδείξει τα προτερήματα της τεχνολογίας αυτής εστιάζοντας στο κομμάτι της επικοινωνίας, η οποία αποτελεί έναν από τους βασικότερους πυλώνες στον κόσμο της τεχνολογίας. Επιπλέον, εφαρμόζοντας συγκεκριμένες τεχνικές της αρχιτεκτονικής REST και κάνοντας χρήση προγραμματιστικών εργαλείων, θα επιτευχθεί επικοινωνία μεταξύ συσκευών πάνω από το internet, για τον απομακρυσμένο έλεγχο περιβαλλοντικών συνθηκών σε ένα περιβάλλον νεφοϋπολογισμού (datacenter) και αντίδραση της ίδιας της υποδομής και των ΔτΠ συσκευών σε περίπτωση που κριθεί αναγκαίο.

1.3 Μεθοδολογία

Προκειμένου να επιτευχθούν οι παραπάνω στόχοι, αρχικά θα γίνει μία θεωρητική σε βάθος μελέτη για τις αρχές λειτουργίας και σχεδίασης του ΔτΠ, καθώς επίσης θα γίνει έρευνα σχετικά με τις ήδη υπάρχουσες και εφαρμοζόμενες λύσεις συστημάτων ΔτΠ σε όλο τον κόσμο. Ομοίως, μέσω της θεωρητικής κατανόησης και ανάλυσης της αρχιτεκτονικής REST θα γίνουν κατανοητοί οι κανόνες αυτής και τα πλεονεκτήματα που μπορεί να προσφέρει οι εφαρμογή τους σε ένα σύστημα ΔτΠ. Τέλος, θα γίνει η σχεδίαση και η υλοποίησης ενός συστήματος ΔτΠ, το οποίο θα ακολουθεί τις προδιαγραφές της αρχιτεκτονικής REST, προκειμένου να αναδειχθούν τα πλεονεκτήματα χρήσης της στο ΔτΠ.

1.4 Δομή

Ξεκινώντας, θα αναφερθούν μερικές έννοιες τόσο για το ΔτΠ όσο και για την αρχιτεκτονική REST, ενώ στη συνέχεια θα παρουσιαστεί αναλυτικά η σχεδίαση και η ανάπτυξη μίας Διεπαφής Προγραμματισμού Εφαρμογών (Application Programmable Interface, API) ακολουθώντας τις προδιαγραφές που ορίζει η αρχιτεκτονική REST, χρησιμοποιώντας τη γλώσσα Java, ενώ στη συνέχεια θα παρουσιαστεί και μία περίπτωση χρήσης αυτής σε ένα περιβάλλον νεφοϋπολογισμού.

2 ΚΕΦΑΛΑΙΟ 1: Διαδίκτυο των Πραγμάτων

Στην πρώτη ενότητα της εργασίας αυτής αναλύεται διεξοδικά το θέμα του Διαδικτύου των Πραγμάτων (ΔτΠ), ξεκινώντας από έναν ορισμό και την προέλευση αυτού, ενώ συνεχίζοντας με κάποιες πιο ειδικές και τεχνικές έννοιες που αφορούν το ΔτΠ. Τέλος θα αναπτυχθούν γενικότερα θέματα, όπως οι επιδράσεις του ΔτΠ στον τεχνολογικό τομέα και η συνεργασία αυτού με άλλες τεχνολογίες αιχμής.

2.1 Τι Είναι το Διαδίκτυο των Πραγμάτων

Ένας από τους ορισμούς που δίνονται για το ΔτΠ είναι ότι πρόκειται για την ιδέα της σύνδεσης οποιαδήποτε συσκευής στο Διαδίκτυο, αλλά ταυτόχρονα και με άλλες συσκευές. Σκοπός της ιδέας αυτής είναι η δημιουργία ενός δικτύου διασυνδεδεμένων πραγμάτων και ανθρώπων που συλλέγουν και μοιράζονται μεταξύ τους δεδομένα, σχετικά με το περιβάλλον τους, [1].

Στο ΔτΠ, Αντικείμενο θεωρείται οποιαδήποτε πηγή δεδομένων που μπορεί να μεταδώσει ή και να αναμεταδώσει πληροφορίες. Ένα παράδειγμα είναι ένας κάρδιο-μετατροπέας με απινιδωτή, ο οποίος περιέχει έναν αισθητήρα παρακολούθησης της καρδιάς και μεταδίδει συνεχώς τις μετρήσεις που παίρνει. Παρόμοιο παράδειγμα Αντικειμένου μπορεί να αποτελούν και τα ζώα μιας φάρμας στα οποία έχει τοποθετηθεί βιομετρικό τσιπ για την παρακολούθηση της υγείας τους, ή ακόμα και ένα αμάξι το οποίο μπορεί να στείλει ειδοποίηση όταν μετρηθεί μέσω αισθητήρων μικρότερη πίεση στα λάστιχά του.

Ωστόσο, δεν είναι τα Πράγματα, τα οποία ορίζουν το ΔτΠ, αλλά η διασύνδεση αυτών, ιδιαίτερα με χρήση του Διαδικτύου. Αναφορικά με τα παραπάνω παραδείγματα, ο άνθρωπος με τον κάρδιο-μετατροπέα θα αποτελούσε μέρος συστήματος ΔτΠ αν ήταν δυνατόν να σταλεί ειδοποίηση σε κάποια υπηρεσία παροχής πρώτων βοηθειών σε περίπτωση που υπήρχαν ανωμαλίες στις μετρήσεις του αισθητήρα στην καρδιά, ή ακόμα και αν υπήρχε διαθέσιμος γιατρός να δει αυτά τα δεδομένα και να επιληφθεί της κατάστασης επιτόπου.

Όπως μπορεί κανείς να παρατηρήσει, βάσει των παραδειγμάτων, ένα από τα κλειδιά στην λειτουργία ενός συστήματος ΔτΠ είναι η Επικοινωνία. Μάλιστα, εξίσου σημαντικό ρόλο έχει τόσο η Επικοινωνία μεταξύ μηχανών ή πραγμάτων (γνωστή και ως Machine-To-Machine ή M2M επικοινωνία), όσο και η Επικοινωνία Ανθρώπου-Μηχανής. Η αμφίδρομη αυτή επικοινωνία είναι που ξεχωρίζει το ΔτΠ από ένα οποιοδήποτε αυτοματοποιημένο σύστημα, ιδιαίτερα όταν συνδυάζεται με το Διαδίκτυο.

Εικόνα 1: Βασικές Έννοιες ΔτΠ

Το ΔτΠ έχει ως σκοπό τον διαμοιρασμό των δεδομένων μεταξύ των αντικειμένων του εκάστοτε συστήματος. Επομένως είναι απόλυτα λογικό, τα δεδομένα να αναφέρονται ως εξίσου βασική έννοια. Ωστόσο, αυτή η βαρύτητα στα δεδομένα είναι που φέρνει την ασφάλεια ενός τέτοιου συστήματος στο προσκήνιο. Αν σκεφτεί κανείς ελάχιστα παραδείγματα συστημάτων ΔτΠ, θα καταλάβει ότι τα δεδομένα που καταγράφονται και μεταδίδονται είναι, στις περισσότερες τουλάχιστον περιπτώσεις ευαίσθητα.

Πολλά από τα δεδομένα που μπορεί να μεταφέρονται σε ένα σύστημα, ακόμη και οι διατροφικές συνήθειες ενός ανθρώπου, είναι δυνατόν να δημ ιουργήσουν συμπεράσματα για τον ίδιο αλλά και να φανερώσουν στοιχεία και συνήθειες, τις οποίες μπορεί να χρησιμοποιήσει κάποιος κακόβουλα.

Έτσι, συμπεραίνουμε ότι η ασφάλεια των δεδομένων είναι άκρως απαραίτητη στο ΔτΠ, του οποίου όμως η ίδια η φύση αυξάνει τη δυσκολία της ασφάλειας. Τις περισσότερες φορές, ένα σύστημα ΔτΠ, αποτελείται από πολλά και μικρά επιμέρους στοιχεία τα οποία μεταδίδουν πληροφορίες ασύρματα, αυξάνοντας έτσι τα σημεία στα οποία μπορεί κάποιος να επιτεθεί στο σύστημα, είτε για να επηρεάσει τη συμπεριφορά του συστήματος, είτε για να υποκλέψει δεδομένα.

Συνοψίζοντας, το ΔτΠ είναι μία έννοια σύμφωνα με την οποία η ασφαλής διασύνδεση αντικειμένων μπορεί να παράγει αλλά και να προσφέρει στοχευμένα πληροφορίες και δεδομένα χτίζοντας έτσι την επικοινωνία μεταξύ αντικειμένων, πράγμα το οποίο μπορεί να έχει πολλά οφέλη σε πολλούς τομείς, από τον οικιακό μέχρι τον βιομηχανικό και τον τομέα υγείας, [2],[4].

2.2 Ιστορική Αναδρομή

Το ΔτΠ είναι μία έννοια, η οποία δεν επισημοποιήθηκε μέχρι τις αρχές του 1999. Ωστόσο, αν ψάξει κανείς, θα βρει πως η ιδέα του ΔτΠ ξεκίνησε να σχηματίζεται πολύ νωρίτερα, όταν ακόμα σχηματιζόταν η ιδέα της απομακρυσμένης επικοινωνίας. Επιστήμονες που έζησαν στα τέλη του 18ου και αρχές του 19ου αιώνα και άφησαν τα σημάδια τους στην ιστορία του τεχνολογικού τομέα, όπως ο Νίκολα Τέσλα και ο Άλαν Τούρινγκ, μπορούσαν ακόμα και τότε να προβλέψουν ότι κάποια στιγμή, η επικοινωνία (ή ακόμα και ο ηλεκτρισμός) θα μπορεί να είναι ασύρματη και μάλιστα να επεκτείνεται σε όλο τον πλανήτη, όπως και να είναι προσβάσιμη από όλους τους ανθρώπους.

Το 1926, ο Νίκολα Τέσλα, δίνοντας συνέντευξη είπε : «Όταν η ασύρματη σύνδεση εφαρμοστεί με τέλειο τρόπο ολόκληρη η γη θα μετατραπεί σε έναν πελώριο εγκέφαλο, κάτι το οποίο ισχύει ήδη, όπου όλα τα πράγματα θα είναι επιμέρους στοιχεία ενός πραγματικού και αρμονικού συνόλου... και τα όργανα με τα οποία θα έχουμε τη δυνατότητα να το κάνουμε αυτό θα είναι εκπληκτικά απλά συγκριτικά με το τρέχον τηλέφωνο που γνωρίζουμε. Ένας άνθρωπος θα μπορεί να κουβαλήσει ένα στην τσέπη του.», [3].

Προφανώς ο Τέσλα δεν μιλούσε συγκεκριμένα για το ΔτΠ ή την έκδοση IEEE 802.11 του WiFi, όμως οι γνώσεις του και η ικανότητα της κατανόησής του για τον ηλεκτρισμό και την επικοινωνία ήταν αρκετά ώστε να μπορεί να προβλέψει την εξέλιξη της τεχνολογίας που έχουμε σήμερα. Συμπεριλαμβανομένων του Διαδικτύου, της ασύρματης σύνδεσης, καθώς και της εξέλιξης της μίκρο-νάνο τεχνολογίας, με την οποία έχουν συμπυκνωθεί όλες οι συσκευές και εξαρτήματα που υπάρχουν.

Αρκετά χρόνια αργότερα, ο «Πατέρας της Επιστήμης Υπολογιστών» και γνωστός για το «σπάσιμο» της κρυπτογράφησης της μηχανής «Enigma», Άλαν Τούρινγκ, το 1950 διατύπωσε το ερώτημα αν οι μηχανές είναι ικανές να σκεφτούν, σχεδόν 20 χρόνια προτού το Διαδίκτυο δημιουργηθεί. Μάλιστα, είπε το εξής: «...Μπορεί να υποστηριχθεί ότι το καλύτερο είναι να παρέχουμε στο μηχάνημα τα καλύτερα αισθητήρια όργανα που μπορούν να αγοραστούν, και μετά να το μάθουμε να καταλαβαίνει και να μιλάει Αγγλικά. Αυτή η διαδικασία θα μπορούσε να ακολουθήσει την κανονική διδασκαλία ενός παιδιού.» , [3].

Υπάρχει σίγουρα μία πληθώρα επιστημόνων αλλά και σύγχρονων φιλόσοφων, οι οποίοι κατείχαν την ικανότητα της κατανόησης για την εξέλιξη της τεχνολογίας. Όντως, στην σημερινή εποχή, όπου κι αν κοιτάξει κανείς θα δει ανθρώπους να έχουν στις τσέπες τους συσκευές με τις οποίες μπορούν να επικοινωνήσουν με όλο τον κόσμο ανεξαρτήτου απόστασης. Παράλληλα, σε ένα σύστημα ΔτΠ, όσο καλύτερα είναι τα «αισθητήρια όργανα» που χρησιμοποιεί τόσο πιο αποδοτικό είναι, ενώ αν το συνδυάσει κανείς με την Μηχανική Μάθηση ή Βαθιά Μάθηση και με την Τεχνητή Νοημοσύνη, δίνεται μία πολύ σαφής απάντηση στο ερώτημα του Τούρινγκ.

Προτού φτάσουμε στο 1999, υπάρχει ακόμα ένα σημείο της ιστορίας το οποίο αξίζει να αναφερθεί. Κατά τη δεκαετία 1980 και αρχές του 1990, υπάρχουν αρκετές αναφορές για περιπτώσεις τροποποίησης συσκευών με στόχο την διευκόλυνση των ανθρώπων. Για παράδειγμα, μία από τις πρώτες αναφορές που υπάρχουν είναι στις αρχές του 1980, όπου στο Πανεπιστήμιο Carnegie Mellon είχε τοποθετηθεί ένα μηχάνημα αναψυκτικών.

Μία ομάδα προγραμματιστών συνδεόντουσαν μέσω Διαδικτύου προκειμένου να διαπιστώσουν αν υπάρχουν τα αναψυκτικά που επιθυμούν και αν είναι παγωμένα, ώστε να μην «ταξιδεύουν» άδικα μέχρι το μηχάνημα. Αργότερα, το 1991 κατασκευάστηκε η πρώτη τοστιέρα, η οποία μπορούσε να ενεργοποιηθεί και να απενεργοποιηθεί μέσω διαδικτύου κάνοντας χρήση του TCP/IP πρωτοκόλλου. Έτσι, ξεκίνησαν τα πρώτα απτά βήματα του Διαδικτύου των Πραγμάτων, [5].

Φτάνοντας στο 1999, ο Κέβιν Άστον, συνιδρυτής της εταιρίας που κατασκεύασε τα RFID (Radio Frequency Identification), έφτιαξε τον όρο «Διαδίκτυο των Πραγμάτων», σε μία ομιλία του. Στόχο είχε την εξήγηση ενός συστήματος που κάνει χρήση των ετικετών RFID σε συνδυασμό με το Διαδίκτυο προκειμένου να επιτευχθεί επικοινωνία μεταξύ μηχανών για την διευκόλυνση αποθήκευσης και μεταφοράς προϊόντων, ενώ ταυτόχρονα οι άνθρωποι να μην αποτελούν την κύρια και αποκλειστική πηγή δεδομένων για το Διαδίκτυο, [3].

Από εκείνο το σημείο της ιστορίας και έπειτα, είχε δημιουργηθεί ένας τομέας ενδιαφέροντος στον κόσμο της τεχνολογίας, και τα πράγματα εξελίχθηκαν αρκετά γρήγορα από τότε. Από τη δημιουργία του πρώτου πρωτοκόλλου επικοινωνίας μεταξύ μηχανών MQTT από την IBM το 1999, μέχρι τη δημιουργία γυαλιών οράσεως με αναγνώριση φωνής και κουμπιά αφής από τη Google το 2013, μπορεί κανείς να συμπεράνει ότι από τη στιγμή που μερικές από τις σκέψεις και τα οράματα του Τέσλα και του Τούρινγκ απέκτησαν το όνομα του Διαδικτύου των Πραγμάτων, τα εξελικτικά άλματα ήταν και θα συνεχίσουν να γίνονται ολοένα και μεγαλύτερα.

Πλέον, το ΔτΠ αποτελεί κάτι παραπάνω από ένα έξυπνο αυτοματοποιημένο σύστημα και μπορεί να μην είναι τόσο δημοφιλές στους ανθρώπους όσο ένα smartphone, αλλά γεγονός είναι πως αποτελεί ένα από τα πιο καίρια και προοδευτικά θέματα της εποχής. Μαζί με άλλες εξελισσόμενες τεχνολογίες το ΔτΠ θα παίζει βασικό ρόλο στο επόμενο επίπεδο της τεχνολογικής καθημερινότητας των ανθρώπων.

2.3 Δομή και Οργάνωση Συστημάτων ΔτΠ

Όπως προαναφέρθηκε το ΔτΠ είναι η ιδέα της σύνδεσης πραγμάτων προκειμένου να επικοινωνούν μεταξύ τους, μεταδίδοντας και κατ' επέκταση χρησιμοποιώντας δεδομένα σε εφαρμογές. Ωστόσο, για να πραγματοποιηθεί αυτή η ιδέα θα πρέπει να υπάρχει μία γενική

οργάνωση και δομή, σύμφωνα με την οποία θα κατατάσσονται όλα τα επιμέρους εξαρτήματα που απαρτίζουν ένα τέτοιο σύστημα.

Για παράδειγμα, ήδη τονίστηκε η σημασία των αισθητήρων και των δεδομένων που παράγουν. Ομοίως, υπάρχουν και άλλες συσκευές εκτός των αισθητήρων, οι οποίες διασφαλίζουν την εύρυθμη λειτουργία ενός συστήματος ΔτΠ.

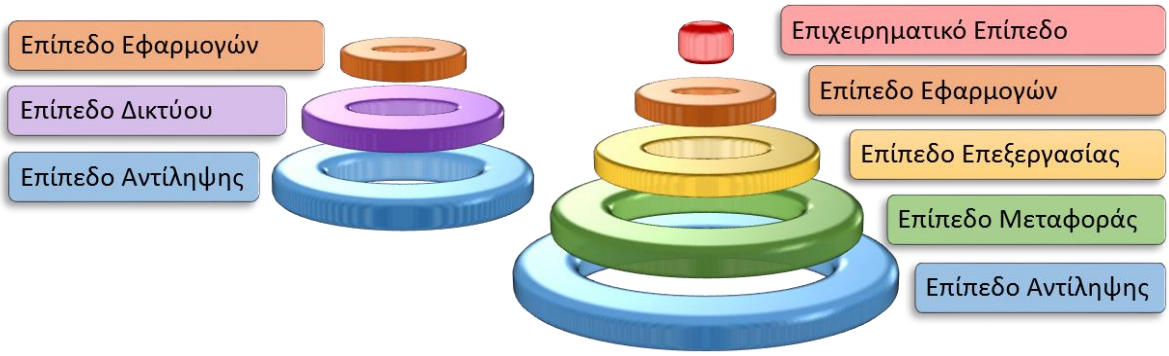
Για αυτόν τον λόγο, πολλοί ερευνητές στον τομέα του ΔτΠ έχουν προτείνει διάφορες δομές ή πιο ειδικά αρχιτεκτονικές, οι οποίες αποτελούν ένα είδος οδηγού για το πώς μπορεί να σχεδιαστεί σε τεχνικό επίπεδο ένα τέτοιο ολοκληρωμένο σύστημα. Επίσης σημαντικό κομμάτι με την κατάταξη των συσκευών είναι ο τρόπος συνδεσιμότητας αυτών και για αυτό χρησιμοποιούνται τα Πρωτόκολλα Επικοινωνίας. Αυτά ουσιαστικά είναι που ενώνουν τα επίπεδα των αρχιτεκτονικών, τα οποία θα αναφερθούν αμέσως μετά.

2.3.1 Αρχιτεκτονικές ΔτΠ

Επίσημα, δεν έχει συμφωνηθεί ομόφωνα κάποια αρχιτεκτονική για το ΔτΠ σε παγκόσμια βάση. Τα τελευταία χρόνια, όμως, έχουν προταθεί αρκετές διαφορετικές αρχιτεκτονικές αναφοράς με αυτές που χρησιμοποιούν τρία και πέντε επίπεδα να είναι οι επικρατέστερες. Επίσης, καθώς το ΔτΠ έχει εφαρμογές σε μία πληθώρα τομέων, αντίστοιχα υπάρχουν και κάποιες εξειδικευμένες αρχιτεκτονικές για κάποιους τομείς. Ιδιαίτερα στο βιομηχανικό και επιχειρησιακό τομέα όπου η επεξεργασία και η ανάλυση των δεδομένων παίρνει «μεγάλες διαστάσεις», [6].

Στη συνέχεια θα αναλύσουμε τις δύο επικρατέστερες αρχιτεκτονικές που αναφέραμε και τα επίπεδα από τα οποία απαρτίζονται και φαίνονται στην Εικόνα 2.

Ξεκινώντας με το πιο απλό μοντέλο αρχιτεκτονικής, αυτό των τριών επιπέδων, παρουσιάστηκε στα πρώτα στάδια έρευνας του ΔτΠ και αποτελείται από τα επίπεδα Αντίληψης, Δικτύου και Εφαρμογών.



Εικόνα 2: Αρχιτεκτονικές 3 και 5 Επιπέδων στο ΔτΠ

- Επίπεδο Αντίληψης: Το επίπεδο αυτό είναι ουσιαστικά το φυσικό επίπεδο, στο οποίο περιλαμβάνονται οι κάθε είδους και αριθμού αισθητήρες. Αυτοί είναι που «αντιλαμβάνονται» το περιβάλλον και πραγματικά σε εμάς μεγέθη και συνθήκες και τα μεταφράζουν σε ψηφιακά δεδομένα με τρόπο που μπορούν οι υπολογιστές να τα χειριστούν και να τα καταλάβουν.

- Επίπεδο Δικτύου: Αυτό το επίπεδο είναι υπεύθυνο για την συνδεσιμότητα τόσο μεταξύ όλων των συσκευών του συστήματος. Εδώ θα μπορούσαμε να πούμε ότι περιλαμβάνονται οι

διακομιστές, δρομολογητές, όπως και κάποια από τα πρωτόκολλα επικοινωνίας που θα αναλυθούν στην επόμενη υποενότητα.

- Επίπεδο Εφαρμογών: Το επίπεδο εφαρμογών είναι υπεύθυνο για την ακεραιότητα των υπηρεσιών που μπορεί να προσφέρει ένα σύστημα ΔτΠ στον τελικό χρήστη. Ουσιαστικά περιλαμβάνει το λογισμικό κυρίως, με το οποίο μπορεί ο χρήστης να αλληλοεπιδράσει με το σύστημα.

Ωστόσο, τα παραπάνω τρία επίπεδα είναι η πιο απλή μορφή ενός συστήματος ΔτΠ, ενώ με τις τρέχουσες τεχνολογίες και πρόοδο που έχει επιτευχθεί, πλέον δεν είναι αρκετά για να εκπροσωπήσουν ένα πιο εξελιγμένο σύστημα. Για αυτόν το λόγο δημιουργήθηκε η αρχιτεκτονική πέντε επιπέδων, ή αλλιώς εκτεταμένη αρχιτεκτονική, προκειμένου να καλυφτούν και να περιγραφούν μέσω αυτής μεγαλύτερες ανάγκες.

Η διαφορά αυτής με την αρχιτεκτονική τριών επιπέδων είναι ουσιαστικά ότι το επίπεδο δικτύου αναλύεται σε δύο νέα επίπεδα, αυτά της Μεταφοράς και Επεξεργασίας, ενώ έχει προστεθεί ένα επιπλέον, το Επιχειρηματικό Επίπεδο. Στη συνέχεια αναλύουμε μόνο τα τρία επιπλέον επίπεδα της αρχιτεκτονικής αυτής, καθώς τα επίπεδα Αντίληψης και Εφαρμογών δεν διαφέρουν καθόλου από προηγουμένως.

- Επίπεδο Μεταφοράς: Το επίπεδο αυτό είναι υπεύθυνο για την μεταφορά δεδομένων μεταξύ των αισθητήρων και του επιπέδου επεξεργασίας. Το επίπεδο Μεταφοράς φροντίζει αυτήν την ασφαλή και ακέραιη επικοινωνία κάνοντας χρήση των πρωτοκόλλων επικοινωνίας και δεδομένων, όπως το Bluetooth, 3G.

- Επίπεδο Επεξεργασίας: Σε αυτό το επίπεδο ανήκουν τα ενδιάμεσα λογισμικά, από τα οποία θα περάσουν όλα τα δεδομένα από το επίπεδο Αντίληψης και του επιπέδου Μεταφοράς προκειμένου να αποθηκευτούν, αναλυθούν, ή ακόμα και να υποστούν κάποια επεξεργασία πριν καταλήξουν σε κάποια μόνιμη κατάσταση και χρησιμοποιηθούν αναλόγως. Σε αυτό το επίπεδο μπορεί να ανήκουν οι βάσεις δεδομένων, οι υποδομές νεφοϋπολογιστικής, όπως και πολλές ακόμα τεχνολογίες για την αξιοποίηση των δεδομένων.

- Επιχειρηματικό Επίπεδο: Αυτό το επίπεδο είναι το ανώτερο που υπάρχει, αν υπάρχει, συνήθως σε ένα σύστημα ΔτΠ. Περιέχει τις διαδικασίες εκείνες που ελέγχουν και διαχειρίζονται ολόκληρο το σύστημα, ανάλογα τις πληροφορίες που δέχεται από όλα τα υπόλοιπα επίπεδα, συμπεριλαμβανομένων εφαρμογών, υπηρεσιών, επιχειρηματικών μοντέλων, γραφημάτων, ιδιωτικότητας δεδομένων και πολλά άλλα.

Στο πρακτικό μέρος της διπλωματικής αυτής, ακολουθείται η δεύτερη αρχιτεκτονική που παρουσιάστηκε παραπάνω.

2.3.2 Πρωτόκολλα Επικοινωνίας

Στην προηγούμενη υποενότητα αναλύθηκαν τα επίπεδα των δύο πιο γνωστών αρχιτεκτονικών ΔτΠ, ποιες είναι οι αρμοδιότητες του καθενός όπως και τι είδους συσκευές μπορεί να περικλείουν. Ωστόσο, η επικοινωνία μεταξύ αυτών δεν είναι μία αυτόματη διαδικασία. Αντίθετα, είναι κάτι το απτό και πραγματικό, το οποίο περιλαμβάνει κανόνες και αρκετά τεχνικά ζητήματα. Για αυτόν το λόγο υπάρχουν τα Πρωτόκολλα. Ουσιαστικά πρόκειται για ομάδες κανόνων και περιορισμών, οι

οποίοι τίθενται στα δύο άκρα επικοινωνίας, δηλαδή τις συσκευές, που θέλουν να ανταλλάξουν δεδομένα.

Έτσι, δύο συσκευές που έχουν τις ίδιες ρυθμίσεις μπορούν να μεταφέρουν πληροφορίες συντονισμένα χωρίς αυτές να χάνονται. Στη συνέχεια, θα αναφερθούν τα πιο γνωστά πρωτόκολλα Επικοινωνίας που χρησιμοποιούνται σε υλοποιήσεις συστημάτων του ΔτΠ.

Έτσι, στον επόμενο πίνακα παρουσιάζονται κάποια από τα πιο γνωστά Πρωτόκολλα Επικοινωνίας, όπως επίσης και οι δυνατότητες αυτών σε σχέση με κάποια χαρακτηριστικά, όπως την εμβέλεια, ή τη συχνότητα λειτουργίας τους αν είναι ασύρματα και πολλά άλλα. Το καθένα έχει τα προτερήματα και τα μειονεκτήματά του και η επιλογή αυτού γίνεται καθαρά και μόνο βάσει των αναγκών του συστήματος ΔτΠ προς σχεδίαση.

Στην υλοποίηση που θα ακολουθήσει στην ενότητα 3 της εργασίας αυτής, έχει θεωρηθεί η χρήση μόνο του πρωτοκόλλου IEEE 802.11 (Wi-Fi), που θα μπορούσε να αντικατασταθεί σε ορισμένες περιστάσεις από το 3G/4G. Περεταίρω ανάλυση θα ακολουθήσει στην αντίστοιχη ενότητα.

Πρωτόκολλο	**Συχνότητα**	**Κατανάλωση Ισχύος**	**Ρυθμός Δεδομένων**	**Εμβέλεια**	**Τοπολογία**	**Κόστος**
BluetoothLE	2.4 GHz	Χαμηλή	1 - 3 Mbps	~ 90 m	P2P	Χαμηλό
Zigbee	2.4 GHz	Χαμηλή	250 Kbps	~ 90 m	Κατανεμημένη	Μεσαίο
Sigfox	Sub – GHz	Χαμηλή	<1 Kbps	1-10 Km/κελί	Αστέρα	Μεσαίο
IEEE 802.11 (WiFi)	Sub, 2.4, 5 GHz	Μεσαία	1 - 135 Mbps	~ 90 m	---	Χαμηλό
LoRaWan	Sub – GHz	Χαμηλή	<50 Kbps	1,5 – 5 Km	Αστέρα – Αστέρα	Μεσαίο
Z-Wave	Sub – GHz	Χαμηλή	40 Kbps	~ 30 m	Κατανεμημένη	Μεσαίο
3G/4G/5G	εξαρτάται	Υψηλή	10 Mbps	~ 35 Km/κελί	P2P	Υψηλό

Πίνακας 1: Πρωτόκολλα Επικοινωνίας στο ΔτΠ

Τα πρωτόκολλα που φαίνονται στον Πίνακα 1 μπορούν να τοποθετηθούν σε διαφορετικά επίπεδα στις αρχιτεκτονικές που αναφέρθηκαν, ή μπορούν ακόμα και να ενώνουν συσκευές που ανήκουν σε διαφορετικά επίπεδα.

Αντίστοιχα, υπάρχουν και τα Πρωτόκολλα Δεδομένων, που παρουσιάζονται στον Πίνακα 2 μαζί με κάποια χαρακτηριστικά τους. Γενικά, για την αποστολή δεδομένων για το ΔτΠ, τα μοντέλα ανταλλαγής μηνυμάτων που προτείνονται είναι δύο: το μοντέλο Αιτήματος – Απόκρισης (Request - Response) και το μοντέλο Εκδότη – Συνδρομητή (Publish - Subscribe). Το πρώτο μοντέλο, μάλιστα, είναι πολύ γνωστό για την αξιοποίησή του στο πρωτόκολλο επιπέδου εφαρμογής του διαδικτύου, το πρωτόκολλο HTTP.

Πρόκειται για το είδος επικοινωνίας που βασίζεται στην ζήτηση πληροφοριών και δεδομένων, ενώ οποιαδήποτε επικοινωνία τύπου Αιτήματος – Απόκρισης, ξεκινάει πάντα με ένα αίτημα από την εφαρμογή που ζητάει δεδομένα, ενώ ο μόνος τρόπος για να αποκτήσει τα δεδομένα αυτά είναι μέσω της απόκρισης της αντίστοιχης εφαρμογής στην οποία φτάνει το αίτημα.

Αντίθετα, το μοντέλο ¬Εκδότη – Συνδρομητή, είναι ένας τρόπος διαμοιρασμού δημόσιων δεδομένων, ο οποίος βασίζεται στην λειτουργία ενός Ενδιάμεσου (Broker). Πρόκειται για ένα είδος λογισμικού που δρα ως «ταχυδρόμος» σε ένα τέτοιο σύστημα, διαμοιράζοντας τα δεδομένα που δέχεται από τις εφαρμογές «Εκδότες», στους «Συνδρομητές», του θέματος που χαρακτηρίζει το εκάστοτε μήνυμα.

Πρωτόκολλο	**Πρωτόκολλο Μεταφοράς**	**Μοντέλο**	**Ασφάλεια**	**Αρχιτεκτονική Δικτύου**
MQTT	TCP	Εκδότη – Συνδρομητή	Μεσαία - Προαιρετική	Client - Server
CoAP	UDP	Αίτημα – Απόκριση	Μεσαία - Προαιρετική	Client – Server
RESTful APIs	TCP	Αίτημα – Απόκριση	Χαμηλή - Προαιρετική	Client – Server
AMQP	TCP	Εκδότη – Συνδρομητή	Μεσαία - Προαιρετική	Client – Server
XMPP	TCP	Εκδότη – Συνδρομητή & Αίτημα – Απόκριση	Υψηλή - Προαπαιτούμενη	Client – Server

Πίνακας 2: Πρωτόκολλα Δεδομένων στο ΔτΠ

Οι διεπαφές τύπου REST (RESTful APIs) είναι αυτές στις οποίες επικεντρώνεται η παρούσα διπλωματική εργασία και αναλύεται διεξοδικά στην επόμενη ενότητα. Συνοπτικά πρόκειται για έναν τρόπο επικοινωνίας που στηρίζεται σημαντικά στο πρωτόκολλο HTTP και χαρακτηρίζεται από την έλλειψη κατάστασης τόσο του διακομιστή όσο και του χρήστη, κάτι το οποίο είναι παραπάνω από θεμιτό σε ένα σύστημα ΔτΠ.

2.3.3 Ασφάλεια

Όπως αναφέρθηκε και νωρίτερα, η ασφάλεια είναι αποτελεί μία από τις μεγαλύτερες προκλήσεις ενός συστήματος ΔτΠ. Λόγω του ότι τα δεδομένα που μεταφέρονται είναι ευαίσθητα, αλλά και μεταφέρουν σημαντική πληροφορία που δεν πρέπει σε καμία περίπτωση να χάνεται ή να αλλοιώνεται, είναι υψίστης σημασίας να δίνεται προσοχή σε όλα επίπεδα ενός συστήματος προκειμένου να διασφαλιστούν τόσο η ακεραιότητα όσο και η ιδιωτικότητα των δεδομένων τους, [10].

Θα μπορούσε κάποιος να παρομοιάσει την ασφάλεια σαν ένα επιπλέον επίπεδο σε κάθε αρχιτεκτονική, το οποίο όμως περιβάλλει όλα τα υπόλοιπα. Πρέπει να σημειωθεί σε αυτό το σημείο ότι η ασφάλεια γενικότερα είναι ένα θέμα το οποίο έχει συγκεκριμένη αντιμετώπιση με συγκεκριμένες τεχνικές. Όσες περισσότερες από αυτές τις τεχνικές μπορούμε να εφαρμόσουμε σε

ένα σύστημα, τόσο καλύτερο και ασφαλές θα είναι το αποτέλεσμα, προσέχοντας βέβαια να μην επηρεάζεται η απόδοσή του.

Στην ασφάλεια συσκευών εντός ενός συστήματος ΔτΠ θα πρέπει να διασφαλίσουμε την προστασία της συσκευής τόσο από την πλευρά του υλικού και των εξαρτημάτων της, όσο και από την πλευρά του λογισμικού που έχει. Αρχικά θα πρέπει να υπάρχει προστασία από την επιτηδευμένη μερική ή και ολική παραβίαση της συσκευής. Αυτό επιτυγχάνεται με φυσικά κυκλώματα προστασίας, τα οποία περιέχονται μέσα στις συσκευές και σε περίπτωση που υπάρξει κάποιου είδους παραβίαση αντιδρούν με διάφορους τρόπους, όπως με κάποιο συναγερμό, ή με την πλήρη απενεργοποίηση της συσκευής ώστε να μην είναι δυνατόν κάποιος να διαβάλει τη συσκευή, [9].

Από την άλλη, η ασφάλεια από την πλευρά του λογισμικού μιας συσκευής περιλαμβάνει προστασία συνδεσιμότητας (τυχόν ανοιχτές TCP/ UDP θύρες, σημεία εισαγωγής κωδικών και πότε εμφανίζονται), προστασία της ταυτότητας της συσκευής, για την οποία θα πρέπει απαραιτήτως να χρησιμοποιούνται κρυπτογραφικές μέθοδοι κατά την μεταφορά αυτής της πληροφορίας (PKI, ψηφιακά πιστοποιητικά). Τέλος, προκειμένου να μην αλλαχθεί ή τροποποιηθεί με κανέναν τρόπο το λογισμικό της συσκευής με απώτερο σκοπό την κακόβουλη προσαρμογή ή αλλαγή ρυθμίσεων, θα πρέπει να γίνονται τακτικές ενημερώσεις λογισμικού.

Η ασφάλεια δικτύου και επικοινωνίας είναι εξίσου σημαντική με την ασφάλεια συσκευών, αν όχι και περισσότερο, καθώς αν δεν προστατεύονται τα δεδομένα κατά την μεταφορά τους μπορούν πολύ εύκολα να υποκλαπούν ή να τροποποιηθούν προτού φτάσουν στον προορισμό τους. Επιπλέον, αυτού του τύπου η ασφάλεια προστατεύει από κάθε είδους ασύρματες επιθέσεις με στόχο τη εύρυθμη λειτουργία του συστήματος.

Τρόποι με τους οποίους θα μπορούσαμε να επιτύχουμε την προστασία από τέτοιες επιθέσεις είναι η χρήση ισχυρής αυθεντικοποίησης χρήστη, η χρήση κρυπτογραφήσεων για την μεταφορά δεδομένων μεταξύ δύο οποιωνδήποτε συσκευών. Επίσης, σημαντικά θα μπορούσε να βοηθήσει η χρήση κάποιου Τείχους Προστασίας, φιλτράροντας με ευέλικτο αλλά και αυστηρό συνάμα τρόπο την κίνηση στο δίκτυο, [11].

Τέλος, δε θα πρέπει να παραμελήσουμε και την ασφάλεια στο υπολογιστικό νέφος, όποτε αυτό χρησιμοποιείται, καθώς είναι εξίσου ευάλωτο σε επιθέσεις, τόσο για την λειτουργία του όσο και για τα δεδομένα που επεξεργάζεται ή αποθηκεύονται σε αυτό. Οι περισσότερες αν όχι όλες οι μέθοδοι που προαναφέρθηκαν για την ασφάλεια σε κάθε επίπεδο του ΔτΠ θα πρέπει να εφαρμοστούν και στο υπολογιστικό νέφος αν αποτελεί και αυτό μέρος του συστήματος, [8].

2.4 Διαδικασία Σχεδίασης Ενός Συστήματος ΔτΠ

Η σχεδίαση ενός συστήματος ΔτΠ εξαρτάται από κάποιους παράγοντες, οι οποίοι μπορεί να έχουν και άμεση επίδραση στην λειτουργία αυτού, [12][13]. Επομένως, για να σχεδιάσει κανείς ένα τέτοιο σύστημα, θα πρέπει να υπολογίσει καθέναν από αυτούς αλλά και να τους προσαρμόσει στην εφαρμογή για την οποία προορίζεται, [15].

Κάποιοι από τους πιο βασικούς παράγοντες που βασίζεται η σχεδίαση είναι:

- Ο Σκοπός Χρήσης: Κάθε σύστημα που σχεδιάζεται έχει κάποιο σκοπό χρήσης και αυτός είναι που θα καθορίσει όλα τα υπόλοιπα χαρακτηριστικά του. Ο σκοπός χρήσης μπορεί εύκολα

να επηρεάσει τις αποφάσεις του σχεδιαστή για πολύ συγκεκριμένα πράγματα όπως τα πρωτόκολλα επικοινωνίας, τα οποία απαιτούν διάφορες τοπολογίες, έχουν συγκεκριμένο ρυθμό δεδομένων και πολλά ακόμα.

- Περιβάλλον & Συνθήκες Λειτουργίας: Κάθε περιβάλλον έχει και διαφορετικές συνθήκες, κάτι το οποίο σημαίνει ότι θα πρέπει να προσαρμόσουμε αλλά και να προφυλάξουμε το εκάστοτε σύστημα σε και από αυτές. Για παράδειγμα, αν οι αισθητήρες του συστήματος θα λειτουργούν σε υπαίθριο χώρο, τότε θα πρέπει είτε να φροντίσουμε εμείς για την προστασία του από τη βροχή, είτε να βρούμε ειδικούς αδιάβροχους αισθητήρες, κάτι το οποίο όμως επηρεάζει το κόστος.

- Ασφάλεια: Όπως εξηγήθηκε στην προηγούμενη ενότητα, η ασφάλεια του συστήματος είναι άκρως απαραίτητη, για τη διασφάλιση τόσο των δεδομένων, όσο και της ακεραιότητας των ίδιων των συσκευών. Καθώς στο ΔτΠ τα κενά ασφαλείας που μπορεί να προκύψουν είναι πολλά, είναι υψίστης σημασίας να μην παραλείψουμε κανένα βήμα στην προστασία του συστήματος, έτσι ώστε να γίνει όσο το δυνατόν πιο ανθεκτικό απέναντι σε πιθανές απειλές.

- Κόστος: Το κόστος μπορεί πολύ εύκολα να επηρεάσει την σχεδίαση ενός συστήματος, αφού συνήθως απαιτούνται μία πληθώρα από συσκευές που είναι απαραίτητες. Για αυτόν τον λόγο θα πρέπει να διασφαλίσουμε πως ακόμα και αν κατά τη σχεδίαση ο προϋπολογισμός που διατίθεται είναι μικρός, υπάρχει η δυνατότητα αναβάθμισης συσκευών και λογισμικού ανά πάσα ώρα και στιγμή.

- Διαθέσιμα Υλικά & Λογισμικό: Προτού ξεκινήσει η συγκεκριμενοποίηση του συστήματος, θα πρέπει να είμαστε πολύ καλά ενημερωμένοι για τα διαθέσιμα υλικά (π.χ. μικροελεγκτές, αισθητήρες, δρομολογητές), αλλά και λογισμικά με τα οποία θέλουμε να πραγματοποιήσουμε το σύστημά μας.

- Συντήρηση: Φυσικό επακόλουθο κάθε συστήματος που δημιουργείται και αποτελείται τόσο από υλικά όσο και από λογισμικό, είναι η ανάγκη για κάποιου είδους συντήρηση. Για παράδειγμα κάποιες από τις συσκευές μπορεί να λειτουργούν με μπαταρίες, κάποιες ίσως χρειάζονται τακτικό καθαρισμό αν βρίσκονται σε αντίστοιχο περιβάλλον. Οι περιορισμοί που μπορεί να προκύψουν λόγω συντήρησης συστήματος μπορεί εύκολα να μεταβάλλουν τις αποφάσεις μας κατά την σχεδίαση.

Το πρώτο βήμα στη σχεδίαση ενός συστήματος ΔτΠ είναι πάντα να τίθενται τα όρια και οι απαιτήσεις του συστήματος σχετικά με τους παραπάνω παράγοντες. Μόλις αυτά γίνουν συγκεκριμένα μπορεί ο σχεδιαστής να προχωρήσει στο επόμενο βήμα, το οποίο είναι η επιλογή υλικών κυρίως αλλά και λογισμικού αναλόγως των ορίων και των απαιτήσεων αυτών. Αντίστοιχα, μπορούμε να πούμε ότι υπάρχουν κάποια βασικά «συστατικά» στοιχεία σε ένα σύστημα ΔτΠ, τα οποία θα πρέπει να επιλέξουμε ανεξαρτήτως συστήματος. Μόλις προσδιοριστούν αυτά, η σχεδίαση του συστήματος έχει τελειώσει και πλέον περνάμε στη φάση της υλοποίησης.

Η επιλογή όλων αυτών των βασικών στοιχείων μπορεί να βασιστεί σε κάποια από τις αρχιτεκτονικές που αναφέρθηκαν παραπάνω και οι οποίες είναι έτσι σχεδιασμένες ώστε να προσφέρουν δομή και οργάνωση σε ένα σύστημα, αυξάνοντας τις δυνατότητες επεκτασιμότητας του. Έτσι, οι επιλογές που πρέπει να κάνουμε αφορούν:

1. Το Υλικό & Πλατφόρμες: Πριν από όλα, θα πρέπει να επιλέξουμε σε ποια ή ποιες συγκεκριμένες πλατφόρμες και υλικά θέλουμε να πραγματοποιήσουμε το σύστημά μας. Για παράδειγμα μπορεί να θέλουμε μικροελεγκτές, οι οποίοι δρουν ως κεντρική πύλη για την μεταφορά των δεδομένων κατευθείαν από τους αισθητήρες, κάποιο cloud για αποθήκευση και επεξεργασία δεδομένων, ή και κάποιο διακομιστή για τη διαχείριση αυτών.

2. Το Λογισμικό για το Υλικό & Πλατφόρμες: Η επιλογή του λογισμικού τόσο για τους μικροελεγκτές όσο και για τυχόν πλατφόρμες που επιλέξουμε είναι σχεδόν αυτόματη, καθώς για κάθε πλατφόρμα συνήθως υπάρχει ένα λογισμικό στο οποίο αντιστοιχεί, ή σε διαφορετικές περιπτώσεις υπάρχουν εκδόσεις λογισμικών που εξειδικεύονται στο ΔτΠ και προσφέρουν τις ανάλογες δυνατότητες.

3. Τους Αισθητήρες & Ενεργοποιητές: Οι αισθητήρες και οι ενεργοποιητές είναι τα εξαρτήματα εκείνα τα οποία θα αλληλοεπιδρούν με το φυσικό περιβάλλον του συστήματος και ουσιαστικά θα το ενώνουν με το ψηφιακό. Επομένως σε αυτό το σημείο θα πρέπει να επιλέξουμε τι ακριβώς περιλαμβάνει αυτή η αλληλεπίδραση. Για παράδειγμα, μπορεί να χρειαζόμαστε αισθητήρες για την μέτρηση θερμοκρασίας και αντίστοιχα ενεργοποιητές οι οποίοι θα ρυθμίζουν τον κλιματισμό μίας αίθουσας.

4. Τις Εισόδους – Εξόδους: Αυτά τα εξαρτήματα μπορεί να είναι και προαιρετικά ανάλογα με την πολυπλοκότητα του συστήματος. Οι είσοδοι και οι έξοδοι περιλαμβάνουν τρόπους με τους οποίους μπορούμε να επικοινωνήσουμε εμείς με το σύστημα απευθείας, όπως για παράδειγμα ενδείξεις LED στους μικροελεγκτές, ή οθόνες πάνω στις πλατφόρμες για προβολή τυχόν σφαλμάτων του αντίστοιχου λογισμικού.

5. Την Συνδεσιμότητα και την Τοπολογία: Αφού πλέον γνωρίζουμε το περιβάλλον και τον χώρο στον οποίο θέλουμε να δρα το σύστημά μας, καθώς και τι είδους συσκευές το απαρτίζουν, μπορούμε να επιλέξουμε τα Πρωτόκολλα Δικτύου με τα οποία θα συνδέονται αυτές. Αυτό εξαρτάται σε έναν πολύ μεγάλο βαθμό από τις ικανότητες των αισθητήρων και των μικροελεγκτών που έχουμε επιλέξει προηγουμένως, όπως επίσης και τις απαιτήσεις που έχουμε θέσει για τον ρυθμό δεδομένων και την εμβέλεια που απαιτείται ανάλογα το περιβάλλον.

6. Την Τοπολογία: Η θέση των συσκευών μας εξαρτάται σε έναν πολύ μεγάλο βαθμό από το περιβάλλον και τις συνθήκες λειτουργίας του συστήματος, ενώ υπάρχει μεγάλη πιθανότητα να έχουμε ήδη προσδιορίσει την τοπολογία μας από την επιλογή του είδους των αισθητήρων και των μικροελεγκτών.

7. Τα Πρωτόκολλα Δεδομένων: Φυσικά, μετά την επιλογή του τρόπου σύνδεσης, ακολουθεί ο τρόπος με τον οποίο θα μεταφέρονται αυτά τα δεδομένα ανάλογα της σύνδεσης. Σε αυτή τη φάση επιλέγουμε τα πρωτόκολλα με τα οποία οι εφαρμογές των συσκευών θα μεταφέρουν δεδομένα μεταξύ τους με σκοπό την επικοινωνία των Πραγμάτων.

8. Την Ασφάλεια: Για μία ακόμη φορά επισημαίνεται η σημασία της ασφάλειας σε ένα σύστημα ΔτΠ. Σε αυτό το σημείο πρέπει να δούμε τι είδους προοπτικές ασφάλειας υπάρχουν και αν χρειάζεται να τις αυξήσουμε αναλόγως. Για παράδειγμα, αν είναι προσβάσιμες και ευάλωτες οι συσκευές από ανεπιθύμητους παράγοντες, ή ακόμα και αν είναι να σκεφτούμε να προσθέσουμε για παράδειγμα αυθεντικοποίηση μέσω βιομετρικών στοιχείων.

9. Την Συντήρηση: Τέλος, θα πρέπει να προβλέψουμε πιθανή συντήρηση που χρειάζεται το σύστημά μας. Αυτό μπορεί να περιλαμβάνει συχνούς ελέγχους για τις πηγές ενέργειας των συσκευών (μπαταρίες κ.α.), ενημερώσεις λογισμικών για καλύτερη απόδοση εφαρμογών αλλά και συσκευών, καθώς και διάφορες ακόμη ενέργειες. Είναι σημαντικό να υπάρχει η απαραίτητη συντήρηση έτσι ώστε η απόδοση του συστήματος να μην φθίνει στον χρόνο και να αποφευχθούν τυχόν δυσλειτουργίες στο μέλλον.

Πλέον, μετά από τις παραπάνω επιλογές που θα έχουμε κάνει, το σύστημα που θέλουμε να σχεδιάσουμε θα έχει πάρει μία πολύ συγκεκριμένη μορφή. Έτσι μπορούμε να προχωρήσουμε στην υλοποίηση αυτού χωρίς περεταίρω καθυστερήσεις.

Εδώ θα πρέπει να σημειωθεί πως είναι πολύ πιθανόν κατά τη διάρκεια της υλοποίησης να χρειαστεί να αλλάξουν κάποια από τα παραπάνω δεδομένα. Πολλές φορές, σε τεχνικό επίπεδο προκύπτουν δυσκολίες και εμπόδια τα οποία μας αναγκάζουν να αλλάξουμε και να προσαρμοστούμε σε διαφορετικά υλικά ή ίσως και σε λογισμικά. Σε κάθε περίπτωση όμως, αν έχουμε κάνει μία προσεχτική επιλογή τόσο των ορίων και των απαιτήσεων όσο και των εξαρτημάτων κατά τη σχεδίαση, τότε οι αλλαγές που μπορεί να προκύψουν είναι μικρές και επιλύονται εύκολα, [14].

2.5 Τομείς Εφαρμογών του ΔτΠ

Σύμφωνα με έρευνες που έχουν γίνει, το ΔτΠ έχει πολύ καλές προοπτικές εξέλιξης σε μία πληθώρα από τομείς, [17]. Η γενικότητα που το χαρακτηρίζει, τόσο σε επίπεδο αρχιτεκτονικής όσο και σε επίπεδο υλοποίησης, είναι αυτό που κάνει τόσο προσαρμόσιμα τα συστήματα ΔτΠ, και εν τέλει δίνει μία τεράστια επιλογή από εφαρμογές σε πάρα πολλούς τομείς. Συνοπτικά αναφέρονται στη συνέχεια κάποιοι από τους τομείς μέσα από τους οποίους εμφανίζεται μεγάλο ενδιαφέρον για το ΔτΠ.

- Βιομηχανία: Είναι ίσως ο τομέας με το μεγαλύτερο ενδιαφέρον για την ενσωμάτωση του ΔτΠ σε παγκόσμιο επίπεδο. Οι επιλογές που μπορεί να προσφέρει το ΔτΠ στην Βιομηχανία είναι τρομερά πολλές και μπορεί να εξοικονομήσει χρόνο, κόστος και πολλά άλλα κατά την παραγωγή αναλόγως την περίπτωση. Για παράδειγμα, στον τομέα της εξόρυξης πετρελαίου και φυσικού αερίου, τόσο η παρακολούθηση όσο και ο απομακρυσμένος έλεγχος του βαρύ εξοπλισμού που χρησιμοποιείται αποτελεί μεγάλο πλεονέκτημα.

- Έξυπνη Πόλη: Μαζί με τον τομέα της βιομηχανίας, η Έξυπνη Πόλη περιέχει την πλειονότητα των εφαρμογών του ΔτΠ σε όλη την Ευρώπη. Ουσιαστικά πρόκειται για την εγκατάσταση συστημάτων σε έκταση ολόκληρων πόλεων, τα οποία σκοπό έχουν την βελτίωση της βελτίωσης της ζωής των κατοίκων τους. Ήδη σε εφαρμογή βρίσκονται συστήματα σε διάφορες πόλεις της Ευρώπης, όπως για παράδειγμα στην πόλη Αϊντχόφεν της Ολλανδίας όπου συστήματα ζωντανής παρακολούθησης αναλύουν τα επίπεδα θορύβου σε συνδυασμό με δεδομένα μέσων κοινωνικής δικτύωσης προκειμένου να γίνεται άμεση ανταπόκριση ευαίσθητων περιστατικών, όπως επίσης και ρύθμιση των φώτων της πόλης αναλόγως, [18].

- Έξυπνη Ενέργεια: Σε αυτόν τον τομέα περιλαμβάνεται το έξυπνο δίκτυο παροχής ρεύματος στην Κεντρική Αμερική, μέρος στο οποίο υπάρχει τεράστια προώθηση της Έξυπνης Ενέργειας, [18]. Η ενέργεια είναι ένα αγαθό, το οποίο το αποκτούμε από μη ανανεώσιμες πηγές. Έτσι είναι υψίστης σημασίας να το διατηρήσουμε όσο το δυνατόν περισσότερο ή έως

ότου βρεθούν εναλλακτικές λύσεις. Το ΔτΠ μπορεί να συμβάλλει σημαντικά στον τομέα αυτόν με τη συλλογή δεδομένων και συμπεριφοράς χρηστών και την ανάλυσή τους σχετικά με την κατανάλωση. Σύμφωνα με αυτή οι πάροχοι ηλεκτρισμού μπορούν να αυξήσουν την απόδοση, την αξιοπιστία αλλά και το κόστος του ρεύματος, [17].

- Έξυπνα Αμάξια: Τα τελευταία χρόνια οι βιομηχανίες αυτοκινήτων έχουν εστιάσει πολύ την προσοχή τους στην βελτίωση των εσωτερικών λειτουργιών των αμαξιών. Ωστόσο, το ΔτΠ μπορεί να εξελίξει την εμπειρία χρήστη σε άλλο επίπεδο καθώς μέσω των δεδομένων που μπορεί να προσφέρει ένα αμάξι, ο οδηγός μπορεί τόσο να παρακολουθεί όσο και να ελέγχει τις λειτουργίες του. Για παράδειγμα, σημαντικό πλεονέκτημα για έναν επαγγελματία οδηγό είναι η παρακολούθηση όλων των συστημάτων του αυτοκινήτου, αλλά και η ανάλυση ζωντανών δεδομένων κατά τη διάρκεια της οδήγησης. Έτσι, οι αποφάσεις που θα πρέπει να πάρει ο οδηγός κατά την οδήγηση, θα είναι πολύ πιο εύκολες και άμεσες αφού θα έχει περισσότερες πληροφορίες, [17].

- Έξυπνο Σπίτι: Πρόκειται ίσως για τον πιο δημοφιλή τομέα για οικιακή χρήση και αποτελεί το επόμενο βήμα στους αυτοματισμούς σπιτιών. Ένα σύστημα ΔτΠ μέσα σε ένα σπίτι μπορεί να προσφέρει πολλά περισσότερα από απλές αυτόματες ενέργειες, όπως την ενεργοποίηση των φώτων την ώρα που κάποιος εισέρχεται σε κάποιο μέρος του σπιτιού. Με το ΔτΠ, το ίδιο το σπίτι μπορεί να γνωρίζει ποιος είναι αυτός που εισέρχεται αλλά και αν αυτός που εισέρχεται έχει εξουσιοδότηση να μπει. Σε κάθε περίπτωση, όμως, η απομακρυσμένη επικοινωνία του ιδιοκτήτη με την οικία του πραγματοποιείται μέσω διαδικτύου, προσφέροντας έτσι μία πληθώρα επιλογών στην αλληλεπίδραση μεταξύ τους και όχι μόνο, [17][19].

- Φορετές Συσκευές: Αυτός ο τομέας θεωρείται πως είναι ο τομέας με τις περισσότερες επενδύσεις στο ΔτΠ, [17]. Αυτό είναι απόλυτα λογικό αν σκεφτεί κανείς πως οι φορετές συσκευές είναι κατά κύριο λόγο οι αισθητήρες από τους οποίους μπορούμε να συλλέξουμε βιομετρικά δεδομένα. Αυτά αργότερα περνάνε από ανάλυση δίνοντας έτσι περισσότερες λύσεις για την βελτίωση εμπειριών και απόδοσης. Όπως μπορεί κανείς να φανταστεί αυτό μπορεί να έχει σπουδαίες εφαρμογές στα συστήματα τα οποία ασχολούνται με τον τομέα υγείας ή αθλητισμού για παράδειγμα.

- Υγεία: Μετά από έρευνες που έχουν γίνει τα τελευταία χρόνια, ο τομέας της υγείας θεωρείται πως θα είναι ο πιο εξελισσόμενος τομέας για το ΔτΠ τα προσεχή χρόνια, [17]. Όντως, το ΔτΠ μπορεί να προσφέρει λύσεις σε πολλά από τα προβλήματα υγείας που υπάρχουν κάνοντας την καθημερινότητα αλλά και την εργασία πολλών ατόμων πιο εύκολη. Παραδείγματα εξελισσόμενων συστημάτων είναι η καλύτερη και πιο άμεση παρακολούθηση υγείας νοσηλευόμενων ατόμων είτε από συγγενείς είτε από τους ίδιους τους γιατρούς, η βοήθεια ηλικιωμένων ατόμων και ατόμων με ειδικές ανάγκες προκειμένου να έχουν την δυνατότητα ανεξαρτητοποίησης.

Το γεγονός ότι τα δεδομένα είναι το επίκεντρο των συστημάτων, σε αντίθεση με την αυτόματη αντίδραση στα αυτοματοποιημένα συστήματα, είναι που κάνει το ΔτΠ να διαφέρει αλλά και να μπορεί να εφαρμοστεί σε κάθε πτυχή της καθημερινότητας όλων των ανθρώπων. Επιπλέον, δίνεται η δυνατότητα επικοινωνίας με το ίδιο το σύστημα για προσαρμογή του συστήματος στις

ανάγκες μας ανά πάσα ώρα και στιγμή, χαρακτηριστικό που δίνει στο ΔτΠ την ευελιξία και την προσαρμοστικότητα που λείπει από τους αυτοματισμούς.

2.6 Άλλες Τεχνολογίες που Ενισχύουν τη Λειτουργία του ΔτΠ

Στην προηγούμενη υποενότητα αναφέρθηκαν μόνο κάποιοι από τους τομείς τους οποίους το ΔτΠ μπορεί να επηρεάσει με τα θετικότερα των αποτελεσμάτων, αλλά και να επεκτείνει σε μεγάλο βαθμό τόσο την αποδοτικότητα των ήδη υπαρχόντων συστημάτων, όσο την ίδια τη λειτουργία τους.

Μάλιστα αυτές οι εφαρμογές στους τομείς αυτούς μπορούν να βελτιώσουν τον τρόπο ζωής των ανθρώπων σε επίπεδο καθημερινότητας επηρεάζοντας τον τρόπο που χρησιμοποιούν την τεχνολογία στην εργασία τους, στην οικία τους ή ακόμα και στον τρόπο επικοινωνίας με άλλους ανθρώπους, όπως ακριβώς και τα μέσα κοινωνικής δικτύωσης. Το ΔτΠ είναι ένα εργαλείο, το οποίο αφενός μπορεί να δώσει ευελιξία και ευκολία σε μαζικό επίπεδο, αφετέρου όμως για να προσφέρει σε τόσο μεγάλη κλίμακα συνήθως συνδυάζεται με άλλες τεχνολογίες, οι οποίες βρίσκονται επίσης στο επίκεντρο της των τεχνολογικών ερευνών, [20].

Έτσι, η Τεχνητή Νοημοσύνη και η Βαθιά Μάθηση είναι τεχνολογίες που μπορούν να επεκτείνουν σημαντικά τις λειτουργίες του ΔτΠ. Χρησιμοποιώντας τέτοια συστήματα σε επίπεδο επεξεργασίας δεδομένων, μπορεί να εξαχθούν πληροφορίες που επιτρέπουν σε ένα σύστημα ΔτΠ να γίνει πιο ακριβές, πιο έξυπνο και πιο ευέλικτο προκειμένου να προσφέρει περισσότερα στον τελικό χρήστη, [21][22].

Επίσης, ένας από τους πιο εξελισσόμενους τομείς είναι η Νεφοϋπολογιστική (Cloud Computing) και η Υπολογιστική στα Άκρα του Δικτύου (Edge Computing). Αυτές οι τεχνολογίες έχουν εξελιχθεί ραγδαία τα τελευταία χρόνια και πρόκειται για τις πιο επαναστατικές στον χώρο. Σε συνδυασμό με το ΔτΠ σε επίπεδο δικτύου, μπορούν να κάνουν ένα σύστημα ΔτΠ πιο γρήγορο, πιο αποδοτικό καθώς και να αυξήσουν την απομακρυσμένη συνδεσιμότητα, αφού μπορούν να αναλαμβάνουν βαριές διεργασίες οι οποίες μπορούν να μεταφερθούν από και προς το κυρίως σύστημα μόνο μέσω διαδικτύου, [21].

Η Επαυξημένη Πραγματικότητα και η Εικονική Πραγματικότητα βρίσκονται στο κέντρο του ενδιαφέροντος στον τομέα της τεχνολογίας καθώς τα προτερήματα και τα πλεονεκτήματα που έχουν και μπορούν να προσφέρουν είναι αδιαμφισβήτητα πολλά. Είναι δυνατόν να συνδυάσουμε αυτές τις δύο εξελισσόμενες τεχνολογίες με το ΔτΠ με τέτοιο τρόπο ώστε η εμπειρία του τελικού χρήστη να βελτιωθεί σε πολύ μεγάλο βαθμό, [22].

Ένα χαρακτηριστικό παράδειγμα είναι ένα τραπέζι που έχει σχεδιαστεί το οποίο χρησιμοποιεί επαυξημένη πραγματικότητα ώστε να αναγνωρίσει τα διάφορα τρόφιμα που βρίσκονται πάνω σε αυτό και μετά από αναζήτηση να εμφανίσει συνταγές που μπορούν να πραγματοποιηθούν με αυτά τα υλικά. Με παρόμοιο τρόπο, θα μπορούσαν τα δεδομένα της επαυξημένης πραγματικότητας να περάσουν σε κάποιο σύστημα εικονικής πραγματικότητας και φυσικά και το αντίστροφο. Το ΔτΠ μπορεί να γίνει ο τρόπος με τον οποίο συνδυάζονται προκειμένου να παρέχουν βελτιωμένη εμπειρία χρήστη.

Τέλος, δε θα μπορούσε να μην αναφερθεί το θέμα της κυβερνοασφάλειας στο ΔτΠ, το οποίο έχει μεγάλη ανάγκη από ασφάλεια κάθε είδους. Ο λόγος για τον οποίο το ΔτΠ δεν έχει εισβάλλει ήδη στην καθημερινότητα των ανθρώπων είναι κυρίως η έλλειψη ασφάλειας των συστημάτων του.

Είναι εξαιρετικά σημαντικό να μπορούμε να εφαρμόσουμε τεχνικές και εργαλεία κυβερνοασφάλειας σε ένα σύστημα ΔτΠ, ειδικότερα εάν αυτό διαχειρίζεται ευαίσθητα δεδομένα. Συνεπώς είναι σαφής η σημασία του τομέα της κυβερνο-ασφάλειας για το ΔτΠ και πώς η εξέλιξη του ενός προκαλεί την εξέλιξη του δεύτερου και αντίστροφα,[22].

Συνοπτικά, μπορεί εύκολα κανείς να συμπεράνει πως το ΔτΠ είναι ένα υπερσύνολο, το οποίο μπορεί να συνδυάσει πολλές τεχνολογίες μαζί και να αλλάξει ριζικά τον τρόπο με τον οποίο αλληλοεπιδρούμε όχι μόνο μεταξύ μας, αλλά και με τα Πράγματα. Αν σκεφτούμε ότι το μόνο που χρειάζεται είναι απλώς να δώσουμε σε αντικείμενα που χρησιμοποιούμε σε καθημερινή βάση, την ικανότητα να μετράνε φυσικές ποσότητες και να συνδέονται με το διαδίκτυο, τότε μπορούμε και να συμπεράνουμε τις πραγματικές προοπτικές που μπορεί να προσφέρει.

3 ΚΕΦΑΛΑΙΟ 2: Αρχιτεκτονική REST και RESTful APIs

Στα τέλη της δεκαετίας του 1990 με αρχές του 2000, ο Roy Fielding ανέπτυξε, στα πλαίσια της διδακτορικής του διατριβής, το πρωτόκολλο HTTP 1.1, το οποίο βασίστηκε στην ήδη υπάρχουσα έκδοση HTTP 1.0. Κατά το διάστημα αυτό έλαβε έναν μεγάλο αριθμό από σχόλια και μηνύματα σχετικά με την έκδοση που ανέπτυσσε.

Έτσι, κλήθηκε να εξηγήσει και να δικαιολογήσει όλες αυτές τις επιλογές τις οποίες έκανε. Ταυτόχρονα, λοιπόν, με την HTTP 1.1, ανέπτυξε και αυτό που αποκαλούμε Representational State Transfer, ή εν συντομία Αρχιτεκτονική REST, την οποία θα μελετήσουμε διεξοδικώς στην ενότητα αυτή.

3.1 Η Αρχιτεκτονική REST

Η αρχιτεκτονική REST είναι κατ' ουσία ένα μοντέλο σχεδίασης και περικλείει ένα σύνολο από ιδιότητες και περιορισμούς για την ανάπτυξη διαδικτυακών εφαρμογών. Ο R.Fielding ισχυρίστηκε πως οποιαδήποτε πράξη χρειαζόταν να εκτελέσει μία εφαρμογή μέσα στο Διαδίκτυο μπορούσε να πραγματοποιηθεί με τη χρήση των τεσσάρων βασικών μεθόδων διαχείρισης πληροφοριών που χρησιμοποιεί και η HTTP σε θεμελιώδες επίπεδο, γνωστές και ως CRUD (Create, Read, Update, Delete).

Σήμερα, η αρχιτεκτονική REST έχει εδραιώσει τη θέση του στην αγορά λογισμικού, έναντι του παρόμοια δημοφιλούς μοντέλου SOAP ([29]), καθώς ικανοποιεί την πλειονότητα των απαιτήσεων των αναπτυσσόμενων εφαρμογών. Εν συντομία, συγκριτικά με το SOAP, η αρχιτεκτονική REST έχει πολύ καλύτερη απόδοση δικτυακά, αφού έχει μεγαλύτερο throughput και αξιοποιεί σε καλύτερο βαθμό το διαθέσιμο εύρος ζώνης, καθιστώντας το δεύτερο καλύτερη επιλογή σε κάποιες περιπτώσεις, [23].

3.1.1 Περιορισμοί και Ιδιότητες Αρχιτεκτονικής REST

Όπως ειπώθηκε και αρχικά σε αυτήν την ενότητα, η αρχιτεκτονική REST είναι ένα μοντέλο σχεδίασης που αποτελείται από κάποιους κανόνες, οι οποίοι ονομάζονται περιορισμοί και είναι αυτοί ακριβώς που την έχουν κάνει να είναι τόσο προσιτή και δημοφιλή στην αγορά, [24]. Ας δούμε αναλυτικά ποιοι είναι αυτοί:

- Αρχιτεκτονική Πελάτη-Διακομιστή: Με αυτόν τον πρώτο περιορισμό επιτυγχάνεται η φορητότητα της διεπαφής του χρήστη σε διάφορες πλατφόρμες, η επεκτασιμότητα του διακομιστή λόγω απλοποίησης των εξαρτημάτων που το αποτελούν, ενώ ταυτόχρονα είναι εφικτή η βελτίωση/εξέλιξη και των δύο πλευρών ανεξάρτητα.

- Έλλειψη Κατάστασης (Statelessness): Κάθε αίτημα που αποστέλλεται προς τον διακομιστή πρέπει να έχει όλες τις απαραίτητες πληροφορίες για να το καταλάβει ο τελευταίος χωρίς να χρειάζεται να ανατρέχει σε προηγούμενα μηνύματα με τον συγκεκριμένο χρήστη. Με αυτόν τον τρόπο η επικοινωνία περιορίζεται σε απλά αιτήματα και αποκρίσεις, χωρίς να απαιτεί από τον διακομιστή να βρίσκεται σε διάφορες καταστάσεις καθ' όλη την επικοινωνία.

Εικόνα 3: Περιορισμοί Αρχιτεκτονικής REST

- Δυνατότητα χρήσης cache (cacheability): Στην RESTful επικοινωνία μεταξύ διακομιστή-χρήστη, ο χρήστης είναι αυτός που κρατάει τον έλεγχο της επικοινωνίας με το να στέλνει όποια αιτήματα είναι απαραίτητα κάθε φορά. Στην προσπάθεια εξοικονόμησης πόρων και από τις δύο πλευρές ορίστηκε πως όλα τα δεδομένα πρέπει ανεξαιρέτως να διευκρινίζονται αν επιτρέπεται να αποθηκευτούν σε προσωρινή μνήμη (cacheable), ή όχι (non-cacheable). Η πρώτη περίπτωση σημαίνει ότι τα συγκεκριμένα δεδομένα δεν αλλάζουν με την πάροδο του χρόνου και άρα μπορούν να επαναχρησιμοποιηθούν από το χρήστη αν το θεωρήσει απαραίτητο. Αντίθετα, η δεύτερη περίπτωση σημαίνει ότι είναι πιθανό αυτά τα δεδομένα να είναι δυναμικού περιεχομένου και διακυβεύεται η εγκυρότητα των δεδομένων.

- Πολυεπίπεδο Σύστημα (Layered System): Αυτός ο περιορισμός αναφέρεται κυρίως στην περίπτωση ολόκληρων συστημάτων υπερμέσων, τα οποία ακολουθούν την αρχιτεκτονική REST. Η οργάνωση ενός τέτοιου συστήματος σε επίπεδα μπορεί να έχει πολλά οφέλη, συμπεριλαμβανομένων της απλότητας και του κατανεμημένου φόρτου εργασίας σε όλα τα εξαρτήματα, γεγονός που βελτιώνει σημαντικά την απόδοση του συνολικού συστήματος.

- Ομοιόμορφη διεπαφή (Uniform Interface): Ο περιορισμός της χρήσης των ομοιόμορφων διεπαφών είναι επίσης ένα από τα πιο σημαντικά χαρακτηριστικά της αρχιτεκτονικής REST, μαζί με την έλλειψη κατάστασης, και είναι αυτό που τη διαχωρίζει από άλλα μοντέλα διαδικτυακών εφαρμογών. Προκειμένου να επιτευχθεί αυτό το χαρακτηριστικό, εφαρμόζουμε τέσσερις επιπλέον περιορισμούς διεπαφών:

- Ταυτοποίηση πόρων (Resource Identification in Requests): Για να βρούμε τους πόρους χρησιμοποιούμε URIs (Uniform Resource Identifiers). Πρόκειται για διευθύνσεις του διαδικτύου, οι οποίες βοηθούν την εύρεση ενός πόρου μέσα στον ιστό του διαδικτύου.

- Επεξεργασία πόρων μέσω καταστάσεων (Resource Manipulation through representation): Η επικοινωνία επιτυγχάνεται μέσω των πρότυπων του πρωτοκόλλου HTTP. Προκειμένου να περιγραφεί μία λειτουργία (μέθοδος), αρκούν μία μέθοδος HTTP και ένα URI.

- «Αυτό-περιγραφόμενα» μηνύματα (Self-descriptive messages): Κάθε μήνυμα εμπεριέχει όλες τις απαραίτητες πληροφορίες για το πώς μπορεί ο χρήστης να επεξεργαστεί το μήνυμα, για παράδειγμα ποιος parser απαιτείται για την ανάγνωση δεδομένων.

- Τα Υπερμέσα ως η Μηχανή Κατάστασης Εφαρμογής (Hypermedia As The Engine Of Application State ή HATEOAS): Ουσιαστικά πρόκειται για μία μέθοδο χαρτογράφησης όλων των διαθέσιμων πόρων, η οποία παρέχεται από έναν πόρο και λειτουργεί σαν μηχανή αναζήτησης για χρήστες. Αντί να απαιτείται οι χρήστες να γνωρίζουν όλους τους πόρους και τα URI που τους αντιστοιχούν, χρειάζεται να γνωρίζουν μονάχα το URI αυτού του

πόρου, ο οποίος μπορεί να απαντάει σε αιτήματα με κείμενο, το οποίο περιέχει υπερσυνδέσμους για τους οποιουσδήποτε πόρους που ζητάει ο χρήστης.

- Κώδικας κατ' απαίτηση (Code on Demand): Ο τελευταίος αυτός περιορισμός είναι προαιρετικός για το λόγο ότι μόνο εάν ολόκληρο το σύστημα είναι σχεδιασμένο κατάλληλα, τότε οι χρήστες μπορούν να επεκτείνουν τις ικανότητές τους και τη λειτουργικότητά τους κατεβάζοντας και εκτελώντας κώδικα από κάποια πηγή.

Βάσει των παραπάνω περιορισμών προκύπτει ότι η αρχιτεκτονική REST χαρακτηρίζεται από κάποιες ιδιότητες, τι οποίες προσδίδει και στα συστήματα τα οποία ενσωματώνεται, όπως:

- Απόδοση: Υπερέχει ως προς την απόδοσή της δικτυακά και αυτό γίνεται χάρη στην βέλτιστη αλληλεπίδραση μεταξύ των δύο άκρων που επικοινωνούν μέσω διαδικτύου

- Επεκτασιμότητα: Λόγω του γεγονότος ότι η αρχιτεκτονική REST απαιτεί πολύ-επίπεδο σύστημα, μπορούμε να προσθέσουμε ενδιάμεσα εξαρτήματα (π.χ. διαμεσολαβητές, τείχη προστασίας κ.α.) ώστε να υποβοηθούν την επικοινωνία και να μοιράζονται τον φόρτο εργασίας του συστήματος.

- Απλότητα λόγω ομοιόμορφης διεπαφής: Είναι αυτονόητο πως όταν είναι δυνατόν να χρησιμοποιηθούν διάφορες μορφές διεπαφών, τότε επέρχεται μία σχετική ελευθερία κινήσεων και μειώνεται η πολυπλοκότητα αφήνοντας έτσι χώρο για εξέλιξη σε κάθε εξάρτημα ξεχωριστά.

- Δυνατότητα τροποποίησης: Ένα ακόμα όφελος της έλλειψης κατάστασης της διεπαφής μας είναι ότι υπάρχει η δυνατότητα τροποποίησης κάποιου εξαρτήματος μέσα στο σύστημα, ενώ αυτό παραμένει σε λειτουργία.

- Φορητότητα εξαρτημάτων: Η δυνατότητα μεταφοράς κώδικα (σε μορφή π.χ. script) μέσα στο περιεχόμενο των δεδομένων ενός μηνύματος από τον διακομιστή στο χρήστη ώστε να επεκτείνει προσωρινά τις δυνατότητες του τελευταίου.

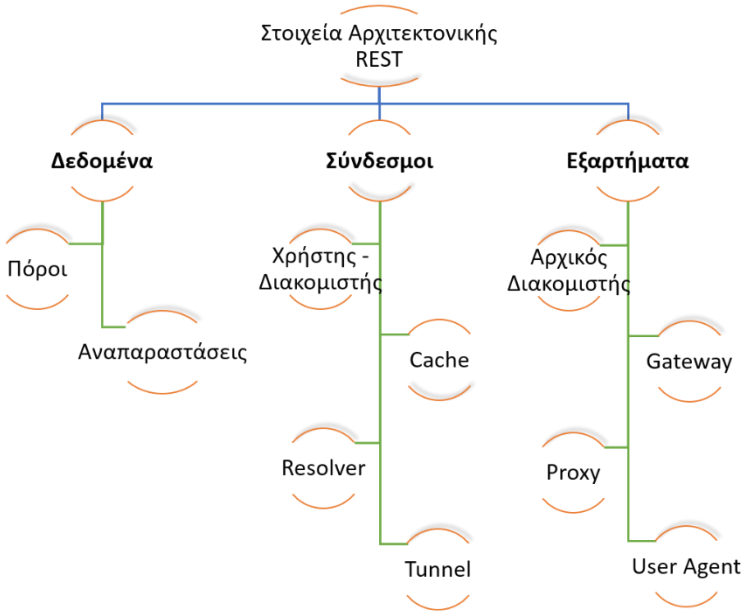
- Ορατότητα επικοινωνίας: Δίνεται η δυνατότητα για ανάγνωση και αναγνώριση της επικοινωνίας μεταξύ δύο εξαρτημάτων.

- Αξιοπιστία: Ως αξιοπιστία λαμβάνουμε την αντοχή του συστήματος απέναντι στα σφάλματα. Πιο συγκεκριμένα, την αξιόπιστη λειτουργία ολόκληρου του αντίστοιχου επιπέδου, όταν σε αυτό υπάρχει εξάρτημα που εμφανίζει σφάλμα στη μεμονωμένη λειτουργία του.

3.1.2 Στοιχεία Αρχιτεκτονικής REST σε Ένα Σύστημα Υπερμέσων

Όπως μπορεί κανείς να καταλάβει και από τα παραπάνω, η αρχιτεκτονική REST είναι μία αφηρημένη έννοια που καθοδηγεί τον τρόπο με τον οποίο λειτουργούν τα στοιχεία ενός κατανεμημένου συστήματος υπερμέσων. Πρόκειται για ένα μοντέλο σχεδίασης, το οποίο θέτει μεν περιορισμούς, αλλά αφήνει πλήρη ελευθερία στα τεχνικά ζητήματα όσον αφορά την εφαρμογή της, [24].

Ωστόσο, σε πιο πρακτικό επίπεδο, εστιάζει στους θεμελιώδεις περιορισμούς όσον αφορά τα εξαρτήματα (Components), τους συνδέσμους (Connectors) και τα δεδομένα (Data), τα οποία καθορίζουν τη συμπεριφορά μέσα στο Διαδίκτυο. Αυτοί οι περιορισμοί σαφώς, αφορούν την οργάνωση ενός ολόκληρου συστήματος, σε αντίθεση με τους παραπάνω βασικούς περιορισμούς που αναφέρθηκαν τους οποίους θα ακολουθήσουμε κατά την ανάπτυξη του RESTful API στην τρίτη ενότητα.



Εικόνα 4: Στοιχεία ενός Συστήματος REST

Τα στοιχεία δεδομένων που χρησιμοποιούνται στις διεπαφές τύπου REST είναι κυρίως δύο: οι πόροι (Resources) και οι αναπαραστάσεις (Representations). Οποιαδήποτε πληροφορία μπορεί να κατονομαστεί, μπορεί να είναι και ένας πόρος. Οι πόροι είναι μία αφηρημένη έννοια που αντιπροσωπεύουν κάποια συγκεκριμένη πληροφορία, ή ακόμα και πολλές ταυτόχρονα, είτε αυτή είναι στατική είτε δυναμική, συναρτήσει του χρόνου. Με άλλα λόγια, πόρος δεν είναι τα ίδια τα δεδομένα αλλά η τοποθεσία και το «δοχείο» της πληροφορίας.

Για παράδειγμα έχουμε δύο πόρους, i) «Η Φωτογραφία της Ημέρας» και ii) «Πανοραμική Βραδινή Λήψη του Παρθενώνα», από την ιστοσελίδα της Wikipedia. Παρατηρούμε, ότι σε επίπεδο περιεχομένου, ο πρώτος πόρος είναι δυναμικός καθώς τα δεδομένα που περιέχει αλλάζουν με τον χρόνο, ενώ ο δεύτερος πόρος περιέχει πολύ συγκεκριμένη πληροφορία η οποία δεν διαφοροποιείται και άρα είναι στατικός. Ακόμα και αν σε κάποια χρονική στιγμή το περιεχόμενο των δύο πόρων ήταν ταυτόσημο, οι ίδιοι οι πόροι θα εξακολουθούσαν να είναι δύο τελείως διαφορετικά πράγματα.

Επιπλέον, η αρχιτεκτονική REST χρησιμοποιεί ταυτότητες πόρων (Resource Identifiers), οι οποίες αντιπροσωπεύουν τον μοναδικό εκείνο τρόπο μέσω του οποίου μπορεί να βρεθεί ένας συγκεκριμένος πόρος. Αυτό επιτυγχάνεται με τα URIs, των οποίων το μονοπάτι υποδηλώνει την τοποθεσία του πόρου μέσα στο Διαδίκτυο.

Συνεχίζοντας, τα στιγμιότυπα ή αλλιώς αναπαραστάσεις αποτελούν τα ίδια τα δεδομένα που μπορεί να περιέχονται σε κάποιον πόρο κάποια δεδομένη στιγμή. Η Εικόνα 5 δείχνει το παράδειγμα που χρησιμοποιήθηκε για την περιγραφή των πόρων και των αναπαραστάσεων. Όπως φαίνεται, οι πόροι είναι μία γενική έννοια, όπου στην συγκεκριμένη περίπτωση μπορούμε να πούμε ότι είναι το κομμάτι εκείνο του διακομιστή της Wikipedia που περιέχει την εκάστοτε εικόνα. Για παράδειγμα,

όταν κάποιος κάνει ένα αίτημα HTTP GET στο URI που δείχνει ο κάθε πόρος θα του επιστραφεί μία εικόνα και ίσως κάποιες ακόμη πληροφορίες.

Εικόνα 5: Παράδειγμα Χρήσης Πόρων και Αναπαραστάσεων

Ο συνδυασμός της εικόνας και των πληροφοριών αυτών αποτελούν την αναπαράσταση του πόρου, τη δεδομένη στιγμή. Στο συγκεκριμένο παράδειγμα, το περιεχόμενο του πόρου «Φωτογραφία της Ημέρας» αλλάζει κάθε μέρα (άρα και η αναπαράστασή του), ενώ το περιεχόμενο του πόρου «Πανοραμική Βραδινή Λήψη Παρθενώνα» μένει στατικό με το χρόνο. Τυχαίνει μία συγκεκριμένη μέρα το περιεχόμενο των δύο πόρων να είναι ίδιο, αυτό όμως δεν σημαίνει πως οι δύο πόροι είναι ίδιοι.

Επίσης ένα πολύ σημαντικό χαρακτηριστικό των δεδομένων στην αρχιτεκτονική REST είναι η μορφή με την οποία λαμβάνονται και αποστέλλονται τα δεδομένα. Για λόγους ομοιομορφίας και ευελιξίας έχει συμφωνηθεί ότι τα δεδομένα που κινούνται μέσω Διαδικτύου πρέπει να είναι μορφής XML ή JSON. Αυτό καθιστά την σχεδίαση περεταίρω εξαρτημάτων ή χρηστών πολύ πιο εύκολη, αφού γνωρίζουν εκ των προτέρων τη δομή των δεδομένων που θα λάβουν και άρα μπορούν να τα διαχειριστούν πολύ πιο εύκολα.

Το δεύτερο σημαντικό στοιχείο της αρχιτεκτονικής REST είναι οι σύνδεσμοι μεταξύ των εξαρτημάτων ενός συστήματος. Ουσιαστικά αποτελούν τη διεπαφή μεταξύ διάφορων εξαρτημάτων και είναι υπεύθυνοι για την επικοινωνία. Υπάρχουν πέντε τύποι συνδέσμων σε ένα σύστημα:

- Χρήστης και Διακομιστής: Πρόκειται για τους δύο κύριους τύπους συνδέσμων, οι οποίοι είναι δυνατόν να περιέχονται και στο ίδιο εξάρτημα. Η μόνη διαφορά τους είναι ότι ο χρήστης είναι υπεύθυνος για τη δημιουργία αιτημάτων, ενώ ο διακομιστής «ακούει» για διαθέσιμες συνδέσεις και απαντά στα αιτήματα.

- Προσωρινή Μνήμη Cache: Όπως αναφέρθηκε και στον αντίστοιχο περιορισμό για τη χρήση μνήμης cache, τα διάφορα εξαρτήματα που φέρονται ως χρήστες και ζητούν δεδομένα, θα πρέπει να διαθέτουν προσωρινή μνήμη ώστε να μπορούν να τα αποθηκεύσουν αν αυτά

χαρακτηρίζονται ως "cacheable". Έτσι αποκλείονται πιθανές επαναλήψεις πανομοιότυπων αιτημάτων.

- Resolver: Αυτό είναι το στοιχείο μεταφράζει μερικώς τις ταυτότητες πόρων (URIs) σε πληροφορία σχετικά με την αντίστοιχη (δια)δικτυακή διεύθυνση, ώστε να επιτευχθεί η επικοινωνία μεταξύ εξαρτημάτων

- Tunnel: Είναι ο τύπος συνδέσμου, ο οποίος είναι υπεύθυνος για την αναμετάδοση της πληροφορίας πέρα από κάποιο όριο σε μία σύνδεση (π.χ. firewall ή gateway δικτύου).

Τέλος, σε ένα σύστημα υπάρχουν και τα εξαρτήματα, τα οποία είναι τα πραγματικά δομικά κομμάτια που το αποτελούν. Υπάρχουν τέσσερις διαφορετικοί τύποι εξαρτημάτων:

- Αρχικός Διακομιστής: Είναι ο διακομιστής που χρησιμοποιεί έναν διακομιστή ως σύνδεσμο ώστε να κυβερνά τα ονόματα όλων των πόρων. Είναι πάντα ο τελικός δέκτης των αιτημάτων και η κύρια πηγή των αναπαραστάσεων των δεδομένων.

- Πύλη: Ένα ενδιάμεσο εξάρτημα από το δίκτυο, το οποίο παρεμβάλλεται στην ενδιάμεση επικοινωνία δύο άκρων ώστε να προσφέρει διαφορετικές υπηρεσίες (π.χ. ασφάλεια, μετάφραση δεδομένων, ενίσχυση απόδοσης κ.α.)

- Διακομιστής Μεσολάβησης: Ένα ενδιάμεσο εξάρτημα με τις ίδιες λειτουργίες όπως και ένα gateway, με τη διαφορά ότι ο χρήστης αποφασίζει αν θα χρησιμοποιήσει proxy ή όχι.

- Χρήστης: Χρησιμοποιεί συνδέσμους τύπου χρήστη, ώστε να δημιουργήσει και να προωθήσει αιτήματα. Είναι πάντα ο τελικός αποδέκτης της εκάστοτε απάντησης στα αιτήματα (π.χ. web browsers).

3.2 Η Σημασία του Πρωτοκόλλου HTTP για Ένα RESTful API

Έχει ήδη αναφερθεί αρκετές φορές ότι το πρωτόκολλο HTTP είναι ένα από τα κύρια συστατικά στοιχεία της αρχιτεκτονικής REST. Και ο λόγος πίσω από αυτό είναι το γεγονός ότι στο Διαδίκτυο χρησιμοποιούμε κυρίως αυτό το πρωτόκολλο για την επικοινωνία μεταξύ εφαρμογών, προκειμένου να μοιράζονται δεδομένα.

Παράλληλα, το Διαδίκτυο έχει συμβάλλει σε μεγάλο βαθμό, ώστε το ΔτΠ να πραγματοποιήσει ένα τεράστιο άλμα τα τελευταία χρόνια. Οι έννοιες του Διαδικτύου, του πρωτοκόλλου HTTP και του ΔτΠ, είναι πλέον συνυφασμένες μεταξύ τους και στο επίκεντρο βρίσκονται τα δεδομένα και η ασφαλής και γρήγορη μεταφορά αυτών.

Σε αυτό το σημείο έρχεται η αρχιτεκτονική REST, ώστε να ενώσει όλες αυτές τις έννοιες με τέτοιο τρόπο ώστε να προσφέρει τη μέγιστη ευελιξία και λειτουργικότητα. Συνεπώς, είναι λογικό συμπέρασμα ότι πολλοί από τους περιορισμούς που θέτει έχουν άμεση επιρροή στον τρόπο χρήσης του πρωτοκόλλου HTTP.

Αυτό δεν απαιτεί συνεχή συνδεσιμότητα μεταξύ των δύο υπολογιστών που επικοινωνούν, ωστόσο αποτελείται από έναν κύκλο αιτήματος – απάντησης, κατά τη διάρκεια του οποίου δημιουργείται μία σύνδεση μέσω ανταλλαγής μηνυμάτων HTTP. Άμεση συνέπεια της ιδιότητας αυτής του πρωτοκόλλου HTTP είναι η έλλειψη κατάστασης, περιορισμός που επιβάλλει η αρχιτεκτονική REST.

Για παράδειγμα, σε μία σύνδεση μεταξύ ενός χρήστη που ζητά ένα αρχείο από έναν διακομιστή δικτύου, τόσο ο χρήστης όσο και ο διακομιστής γνωρίζουν ο ένας την ύπαρξη του άλλου μόνο κατά τη διάρκεια της συναλλαγής μηνυμάτων τους, ενώ αμέσως μετά η σύνδεσή τους τερματίζεται.

Εικόνα 6: Βασική Διαδικασία Ανταλλαγής Μηνυμάτων Πρωτοκόλλου HTTP

Τα μηνύματα HTTP που ανταλλάσσονται έχουν μία βασική δομή, η οποία αποτελείται από τρία μέρη: τη γραμμή αιτήματος, τις κεφαλίδες και τέλος το σώμα. Το περιεχόμενο αυτών διαφοροποιείται ελαφρώς ανάμεσα στα αιτήματα και στις απαντήσεις. Παράλληλα, ο μόνος τρόπος σύνδεσης για ένα RESTful API, βάσει των περιορισμών, είναι τύπου χρήστη – διακομιστή, ενώ το API που παρουσιάζεται στην παρούσα διπλωματική αποτελεί το κομμάτι του διακομιστή. Επομένως, στην υλοποίηση του RESTful API θα μας απασχολήσει έντονα η δομή μιας απόκρισης HTTP.

Στην Εικόνα 7 φαίνεται το περιεχόμενο ενός αιτήματος HTTP και μίας απόκρισης HTTP. Οι Κεφαλίδες είναι ένα από τα πιο βασικά εργαλεία όχι μόνο του πρωτοκόλλου HTTP, αλλά και ολόκληρου του Διαδικτύου και είναι αυτό που θα μας απασχολήσει περισσότερο όσον αφορά τα μηνύματα αυτά. Στην ουσία πρόκειται για παραμέτρους, οι οποίες επηρεάζουν σημαντικά τη συμπεριφορά των συσκευών σε επίπεδο εφαρμογής, σχετικά με το περιεχόμενο που μεταφέρει το οποιοδήποτε HTTP μήνυμα.

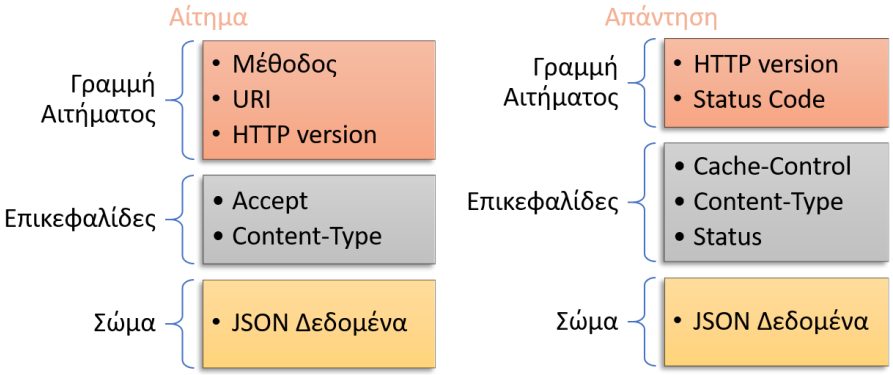
Όπως φαίνεται και στην Εικόνα 7, οι κεφαλίδες που θα μας απασχολήσουν για το RESTful API της συνέχειας είναι τέσσερις, οι εξής:

- Cache-Control: Πρόκειται για μία κεφαλίδα εξαιρετικά σημαντική για ένα RESTful API ή και σύστημα, αφού αναφέρεται ρητά στους περιορισμούς της αρχιτεκτονικής REST ότι όλα τα μηνύματα και τα δεδομένα που μεταφέρονται πρέπει να χαρακτηρίζονται για τη δυνατότητα αποθήκευσής τους σε μνήμη cache.

- Content-Type: Η συγκεκριμένη κεφαλίδα χαρακτηρίζει τον τύπο δεδομένων είτε που ζητά ένας χρήστης, είτε που παρέχει ένας διακομιστής. Αυτή είναι μία πολύ σημαντική πληροφορία τόσο για έναν χρήστη, προκειμένου να ξέρει για παράδειγμα ποιόν parser χρειάζεται για τα αντίστοιχα δεδομένα, όσο και για έναν διακομιστή, ο οποίος σύμφωνα με τον περιορισμό της αρχιτεκτονικής REST πρέπει να ανταλλάσσει αυτο-περιγραφόμενα μηνύματα. Σε κάθε περίπτωση ένα RESTful API οφείλει να ενημερώνει τον αποστολέα για τον τύπο δεδομένων που του αποστέλλει.

- Content-Length: Με αυτήν την κεφαλίδα δίνεται η πληροφορία του μεγέθους σε bytes του σώματος, των δεδομένων δηλαδή, ενός μηνύματος HTTP. Συνήθως αυτή η πληροφορία καταχωρείται αυτόματα κατά τη διαδρομή του μηνύματος σε όλα τα επίπεδα του διαδικτύου και χρησιμεύει στην επαλήθευση της ακεραιότητας του μηνύματος. Στο API της εργασίας αυτής θα καταχωρείται επίσης με αυτόματο τρόπο από το ίδιο το πλαίσιο Jersey.

- Date: Πρόκειται για απλή αποτύπωση ημερομηνίας και ώρας αποστολής του αντίστοιχου μηνύματος HTTP σε ώρα Γκρίνουιτς. Και αυτή η κεφαλίδα είναι προ απαιτούμενη και καταχωρείται αυτόματα όπως και η προηγούμενη του Content-Length.



Εικόνα 7: Γενική Δομή Μηνυμάτων HTTP

Τέλος, αμέσως μετά τις Κεφαλίδες, σε ένα μήνυμα HTTP ακολουθεί το κυρίως Σώμα. Αυτό περιέχει τα δεδομένα που πρόκειται να μεταφερθούν μεταξύ εφαρμογών, αν υπάρχουν. Για παράδειγμα τα αιτήματα GET που θα εξετάσουμε στη συνέχεια για το RESTful API δε θα υπάρχει κανένα περιεχόμενο στο τμήμα του Σώματος. Αντίθετα, στα αιτήματα POST όπου φυσικά θα μεταφέρεται η πληροφορία τύπου JSON από τους μικροελεγκτές, όλα τα δεδομένα για μεταφορά θα ανήκουν στο κομμάτι του Σώματος.

3.3 Η Χρησιμότητα των RESTful APIs

Προφανώς, ο πιο απλοϊκός ορισμός που μπορεί να αποδοθεί για ένα RESTful API, είναι ότι πρόκειται για μία Διεπαφή Προγραμματισμού Εφαρμογών (API) η οποία όμως ακολουθεί τις βασικές αρχές και περιορισμούς της αρχιτεκτονικής REST. Επομένως, η αρχιτεκτονική REST δίνει κάποιες κατευθυντήριες γραμμές, σύμφωνα με τις οποίες μπορούμε να φτιάξουμε διεπαφές (RESTful διεπαφές, RESTful APIs), με σκοπό την ανταλλαγή δεδομένων από και προς μία διαδικτυακή εφαρμογή ή και ολόκληρο σύστημα.

Όταν ο R.Fielding ανέπτυσσε την αρχιτεκτονική αυτή δεν το έκανε τυχαία, αλλά έχοντας στο μυαλό του το πρωτόκολλο HTTP, κάτι το οποίο είναι απόλυτα λογικό αφού αυτό είναι το κυρίαρχο πρωτόκολλο εφαρμογών για ολόκληρο το διαδίκτυο. Παρόλο που η αρχιτεκτονική REST συνδέεται άμεσα λοιπόν με το HTTP, δε θα έπρεπε σε καμία περίπτωση να υποθέσουμε ότι είναι το ίδιο πράγμα.

Αντιθέτως, είναι δύο πολύ διαφορετικά εργαλεία, τα οποία όταν κάποιος τα συνδυάσει κατάλληλα, προκύπτει ένα RESTful API, σκοπός του οποίου είναι η γρήγορη, εύκολη και επιτυχής επικοινωνία και ανταλλαγή δεδομένων πάνω από το Διαδίκτυο. Ένα ακόμη πλεονέκτημα της χρήσης αρχιτεκτονικής REST είναι ότι δεν δεσμεύει τον προγραμματιστή ως προς τη γλώσσα

προγραμματισμού, την οποία θα πρέπει να χρησιμοποιήσει κατά την ανάπτυξη διαδικτυακής εφαρμογής. Αρκεί αυτή η γλώσσα προγραμματισμού να ενσωματώνει κάποιο πλαίσιο για την ανάπτυξή τους και να υποστηρίζει το πρωτόκολλο HTTP.

Οι περισσότερες εταιρίες κοινωνικής δικτύωσης (Facebook, Twitter, Instagram, κ.α.) διαθέτουν RESTful APIs, στα οποία μπορούμε να στείλουμε από κάποιον χρήστη αιτήματα GET και να μας επιστραφούν τα αντίστοιχα δεδομένα σε μορφή JSON (ή και XML). Μάλιστα, πολλές ιστοσελίδες συγκέντρωσης δεδομένων (Wikipedia, Spotify, Flickr, Magento) προσφέρουν πλατφόρμες ώστε να αναπτύξουν οι ίδιοι οι χρήστες RESTful APIs χρησιμοποιώντας τους πόρους της αντίστοιχης σελίδας. Με αυτόν τον τρόπο είναι δυνατή η ανάκτηση δεδομένων (και όχι πόρων), ώστε να χρησιμοποιηθούν με οποιοδήποτε τρόπο από τον χρήστη.

Σε αυτό το σημείο, εννοείται πως αν υπάρχουν ευαίσθητα δεδομένα, όπως για παράδειγμα αυτά των μέσων κοινωνικής δικτύωσης, τότε αυτά αποστέλλονται μέσω των REST APIs μόνο αν είναι δημόσια ή αν η εκάστοτε εφαρμογή έχει τα απαραίτητα στοιχεία και διακριτικά ώστε να επιτρέπεται η εμφάνισή τους σε αυτήν και μόνο αυτήν την εφαρμογή. Σε αυτό το κομμάτι χρησιμοποιείται υποχρεωτικά η μέθοδος της εξουσιοδότησης (authorization), συνήθως με χρήση token.

Συμπερασματικά, λοιπόν, σκοπός των RESTful APIs είναι η διαχείριση δεδομένων πάνω από το διαδίκτυο, ενώ το πρωτόκολλο HTTP είναι το εργαλείο, το οποίο χρησιμοποιείται για αυτόν τον σκοπό. Έτσι, για να κάνουμε προσθήκη δεδομένων σε κάποιον πόρο ενός RESTful συστήματος χρησιμοποιούμε τη μέθοδο POST του HTTP. Αντίστοιχα, στην ανάγνωση δεδομένων αντιστοιχεί η μέθοδος GET, στην ενημέρωση δεδομένων η μέθοδος PUT και τέλος στην διαγραφή δεδομένων η μέθοδος DELETE.

3.4 RESTful APIs στο Διαδίκτυο των Πραγμάτων

Η αρχιτεκτονική REST συνδέεται άρρηκτα με το πρωτόκολλο HTTP, το οποίο είναι πρωτόκολλο επιπέδου εφαρμογής στη στοίβα πρωτοκόλλων του μοντέλου OSI και TCP/IP. Επομένως, την χρησιμοποιούμε για να αναπτύξουμε εφαρμογές με σκοπό την ανταλλαγή πληροφοριών πάνω από το Διαδίκτυο. Αυτές οι εφαρμογές λειτουργούν ως διεπαφές στην επικοινωνία δύο άκρων. Με τον ίδιο τρόπο, είναι εφικτό να εφαρμοστεί η ίδια λογική στο ΔτΠ, [26].

Όπως αναφέρθηκε και στην πρώτη ενότητα, το ΔτΠ είναι ένας ταχύτατα εξελισσόμενος τομέας, περισσότερο τώρα στην τρέχουσα εποχή παρά ποτέ άλλοτε. Δεν είναι τυχαίο το γεγονός ότι τόσο η αρχιτεκτονική REST όσο και το ΔτΠ εξελίσσονται την ίδια χρονική περίοδο. Η ανάγκη του ΔτΠ να υπάρξει αυτή η σύνδεση μεταξύ του φυσικού κόσμου και των υπολογιστικών εφαρμογών, κυρίως με τη χρήση του διαδικτύου, είναι επίσης ένας παράγοντας που επέτρεψε στην αρχιτεκτονική REST να εξελιχθεί τόσο ραγδαία.

Το κυρίως συστατικό του ΔτΠ, και ίσως το πιο σημαντικό, είναι τα ίδια τα δεδομένα. Οι διαδικτυακές εφαρμογές και πιο συγκεκριμένα οι RESTful εφαρμογές, επιτρέπουν στα δεδομένα να μεταφέρονται και να περνάνε από τον φυσικό κόσμο στον υπολογιστικό, χωρίς υπερβολικά περίπλοκες διαδικασίες.

Ενώ το ΔτΠ μπορεί να έχει πολλούς ορισμούς, ένας από τους πυλώνες σε κάθε περίπτωση είναι η Επικοινωνία. Πλέον οι μηχανές οποιασδήποτε τάξης και χρήσης θέλουμε να γίνονται πιο έξυπνες και να καταλαβαίνουν τον φυσικό κόσμο μέσα από τις καθημερινές ανάγκες των ανθρώπων, όπως και να δρουν άμεσα εάν κρίνεται απαραίτητο, [28]. Επομένως, είναι πολύ σημαντικό να υπάρχει σωστή και αξιόπιστη επικοινωνία μεταξύ μηχανών.

Σε αυτό το σημείο είναι που εδραιώνει τη θέση της η αρχιτεκτονική REST, καθώς προσφέρει έναν τρόπο σύνταξης οδηγιών ώστε να αλληλοεπιδρούν με κατάλληλο τρόπο οι κόμβοι ενός συστήματος ΔτΠ. Έτσι, κατά έναν οξύμωρο τρόπο, αυτοί οι ελάχιστοι αλλά κρίσιμα σημαντικοί περιορισμοί της υποενότητας 2.1.1. δίνουν τεράστια ευελιξία στο ΔτΠ.

Χρησιμοποιώντας την αρχιτεκτονική αυτή, οι κόμβοι μπορούν να επικοινωνούν μέσω διαδικτύου βάσει του πρωτοκόλλου HTTP, κάτι το οποίο προσδίδει τεράστια ελευθερία κινήσεων σε ένα σύστημα τόσο στην ανάκτηση δεδομένων, όσο και στην αποστολή. Υπάρχει μία θεμελιώδης και αρκετά καλή ασφάλεια δεδομένων λόγω κρυπτογράφησης SSL/TLS[1], η οποία είναι ενσωματωμένη στο HTTPS, παρόλα ταύτα είναι απόλυτα εφικτό αυτή να ενισχυθεί με όποιο τρόπο κρίνει ο προγραμματιστής καλύτερα. Τέλος η κυρίαρχη μορφή δεδομένων πλέον είναι η JSON, ίσως και η XML, την οποία καταλαβαίνουν οι περισσότεροι αν όχι όλοι οι web διακομιστές που αναλαμβάνουν τη διαχείριση των δεδομένων, [25].
Συνοψίζοντας, η αρχιτεκτονική REST μπορεί να προσφέρει πολλές ιδιότητες καθώς, αν και οξύμωρο, ακολουθώντας μερικούς απλούς περιορισμούς κερδίζουμε σε απόδοση, ευελιξία, επεκτασιμότητα και πολλά ακόμα. Αυτές είναι σημαντικές ιδιότητες που μπορούν να μεταφερθούν στο ΔτΠ, αν χρησιμοποιήσουμε διεπαφές τύπου REST.

[1] SSL/TLS: Secure Socket Layer/ Transport Layer Security. Πρόκειται για πρωτόκολλα κρυπτογράφησης, τα οποία προσδίδουν ασφάλεια στην επικοινωνία δικτύου.

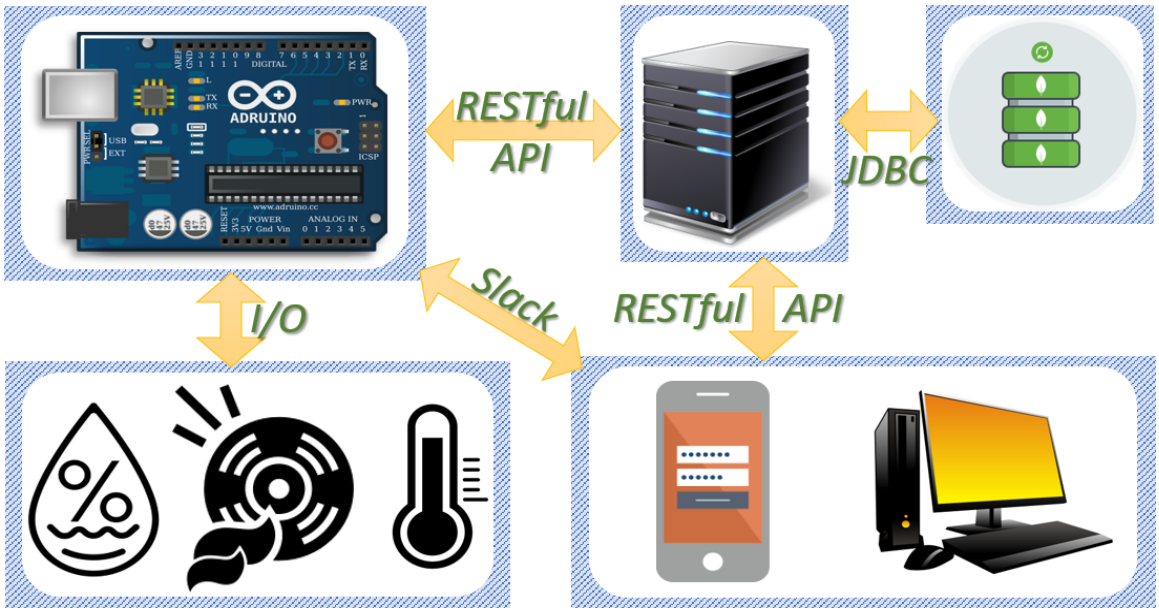
4 ΚΕΦΑΛΑΙΟ 3: Σχεδιασμός και Υλοποίηση RESTful API με Χρήση Java

Σε αυτήν την ενότητα παρουσιάζεται η σχεδίαση και η ανάπτυξη μιας Διεπαφής Προγραμματισμού Εφαρμογών που ακολουθεί τους περιορισμούς της αρχιτεκτονικής REST, με χρήση της γλώσσας προγραμματισμού Java. Συμπληρωματικά, χρησιμοποιούνται και κάποια επιπλέον προγραμματιστικά εργαλεία ώστε να τεθεί σε λειτουργία η διεπαφή και να δοκιμαστεί.

Εδώ πρέπει να σημειωθεί ότι ο κώδικας ο οποίος θα επεξηγηθεί σε αυτήν την ενότητα θα είναι ένα ενδεικτικό κομμάτι του συνολικού κώδικα, ενώ ο κώδικας ολόκληρος και αριθμημένος θα βρίσκεται στο τέλος της εργασίας αυτής, στην ενότητα του παραρτήματος.

4.1 Σκοπός και Λειτουργία Υλοποίησης

Στόχος της εφαρμογής αυτής είναι να αναδείξει στην πράξη τα πλεονεκτήματα της αρχιτεκτονικής REST μέσω των APIs και να αναδείξει τα οφέλη, τα οποία προσφέρει τόσο γενικά όσο και στον τομέα του ΔτΠ.



Εικόνα 8: Σύστημα ΔτΠ ως Υλοποίηση της Εργασίας

Σε αυτό το πλαίσιο οφείλεται και η παρούσα εφαρμογή, καθώς πρόκειται για μία εφαρμογή μεταφοράς δεδομένων από αισθητήρες, οι οποίοι μετράνε διάφορα μεγέθη. Όπως φαίνεται και στην Εικόνα 8, αυτά τα δεδομένα αποστέλλονται προς το RESTful API ανεπτυγμένο σε γλώσσα Java, ώστε να αποθηκευτούν σε μία μη-σχεσιακή βάση δεδομένων. Ταυτόχρονα, το ίδιο API δίνει τη δυνατότητα αναζήτησης αυτών των δεδομένων μέσω αιτημάτων GET σε συγκεκριμένα URIs από τη βάση δεδομένων.

Προκειμένου να φανεί στην πράξη η λειτουργία της εφαρμογής, χρησιμοποιείται και το εργαλείο Node-RED, της πλατφόρμας Node.js, αφενός για να γίνει μία προσομοίωση εισαγωγής των δεδομένων και αφετέρου για να πραγματοποιηθεί ένας τρόπος επικοινωνίας από κινητό τηλέφωνο ή υπολογιστή με το API, πάνω από το Διαδίκτυο χρησιμοποιώντας την εφαρμογή ανταλλαγής μηνυμάτων Slack.

4.1.1 Σενάριο Υλοποίησης και Δεδομένα

Στα πλαίσια της παρούσας διπλωματικής εργασίας θεωρούμε ότι έχουμε αποθηκευτικούς χώρους, στους οποίους θέλουμε να παρακολουθούμε τις συνθήκες περιβάλλοντος. Πρωταρχικό ρόλο για αυτές παίζουν η θερμοκρασία, η υγρασία, το διοξείδιο του άνθρακα (CO_2) και σε κάποιες περιπτώσεις και τα επίπεδα ηλιακής ακτινοβολίας, καθώς αυτά τα μεγέθη είναι ενδείξεις για την περιβαλλοντική κατάσταση ενός χώρου.

Οι συνθήκες του χώρου μπορεί να επηρεάσουν είτε θετικά είτε αρνητικά διάφορα αγαθά και μάλιστα σε μεγάλο βαθμό. Για παράδειγμα, τα ηλεκτρονικά/ηλεκτρολογικά αγαθά είναι ευαίσθητα στην υγρασία, καθώς μεγάλα ποσοστά αυτής μπορούν να προκαλέσουν ανεπανόρθωτη βλάβη. Τα φαρμακευτικά είδη είναι εξαιρετικά ευαίσθητα στην θερμοκρασία αφού πρέπει να φυλάσσονται σε πολύ συγκεκριμένες θερμοκρασίες, ειδάλλως είναι εφικτό να επηρεαστούν οι χημικές ενώσεις από τις οποίες μπορεί να αποτελούνται. Σαφώς, το ίδιο ευαίσθητα είναι πολλά βρώσιμα αγαθά όπως τα διάφορα συσκευασμένα τρόφιμα, τα κρασιά και πολλά άλλα.

Πιο συγκεκριμένα, για το API που υλοποιήσαμε και παρουσιάζεται στη συνέχεια, υποθέτουμε πως παρακολουθούνται ταυτόχρονα πέντε διαφορετικοί αποθηκευτικοί χώροι, μετρώντας την θερμοκρασία, την υγρασία και εντοπίζοντας την ύπαρξη καπνού σε αυτούς. Στη συνέχεια αυτά τα δεδομένα πηγαίνουν σε κάποιον μικροελεγκτή και αυτός είναι που φτιάχνει ένα σετ από δεδομένα σε μορφή JSON μαζί με δεδομένα σχετικά με την ημερομηνία και την ώρα της μέτρησης και την ταυτότητα του μικροελεγκτή. Εν τέλει, το API που υλοποιήσαμε αναμένει να λάβει μέσω πρωτοκόλλου HTTP δεδομένα τα εξής δεδομένα απλής μορφής:

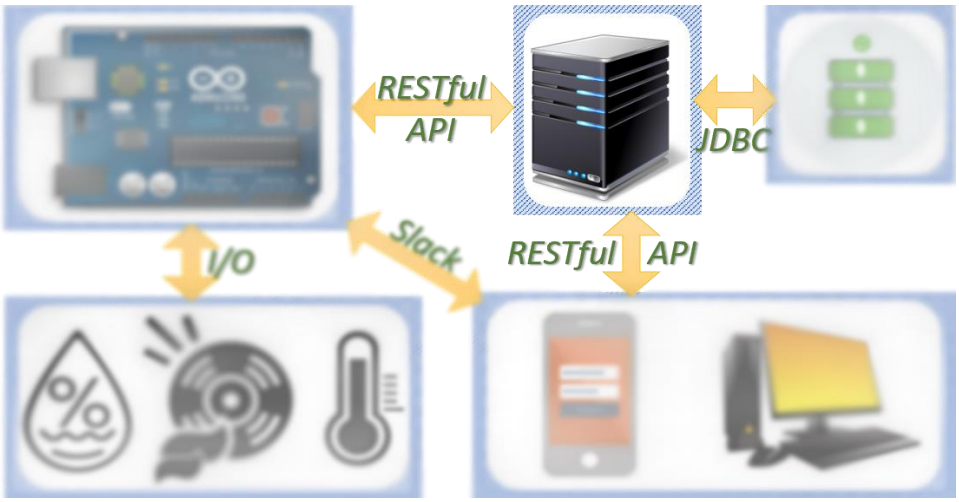
```
{
  “id”: “abcd1”
  “location”: “warehouse-A”
  “temperature”: “27”
  “humidity”: “12”
  “smoke”: “false”
  “DateTime”:“2018-07-13T13:28:25+03:00[Europe/Athens]”
}
```

Εικόνα 9: Βασική Μορφή Δεδομένων JSON που θα Λαμβάνει το RESTful API

Οι παράμετροι *id, location* και *DateTime* είναι αλφαριθμητικές με την τελευταία, να περιγράφει την ημερομηνία και την τοπική ώρα Ελλάδος. Τα *temperature* και *humidity* είναι ακέραιοι αριθμοί με το πρώτο να εκφράζει βαθμούς Κελσίου, ενώ το δεύτερο να εκφράζει το ποσοστό θερμοκρασίας στον χώρο. Τέλος, η παράμετρος *smoke* φτάνει στο API ως τιμή τύπου boolean, κάτι το οποίο σημαίνει πως ο μικροελεγκτής που είναι συνδεδεμένος ο αισθητήρας καπνού είναι υπεύθυνος για να κρίνει αν η τιμή του αισθητήρα προμηνύει την ύπαρξη φωτιάς ή όχι.

Φυσικά το πόρισμα αυτό του μικροελεγκτή θα είναι περισσότερο αξιόπιστο αν προκύψει από τον συνδυασμό των δεδομένων όλων των αισθητήρων (υψηλή θερμοκρασία, εξαιρετικά πολύ χαμηλή υγρασία), αλλά αυτό αφενός αφορά καθαρά τον προγραμματισμό του μικροελεγκτή και αφετέρου είναι θέμα ανάλυσης δεδομένων, κάτι το οποίο δεν αφορά ένα API επικοινωνίας.

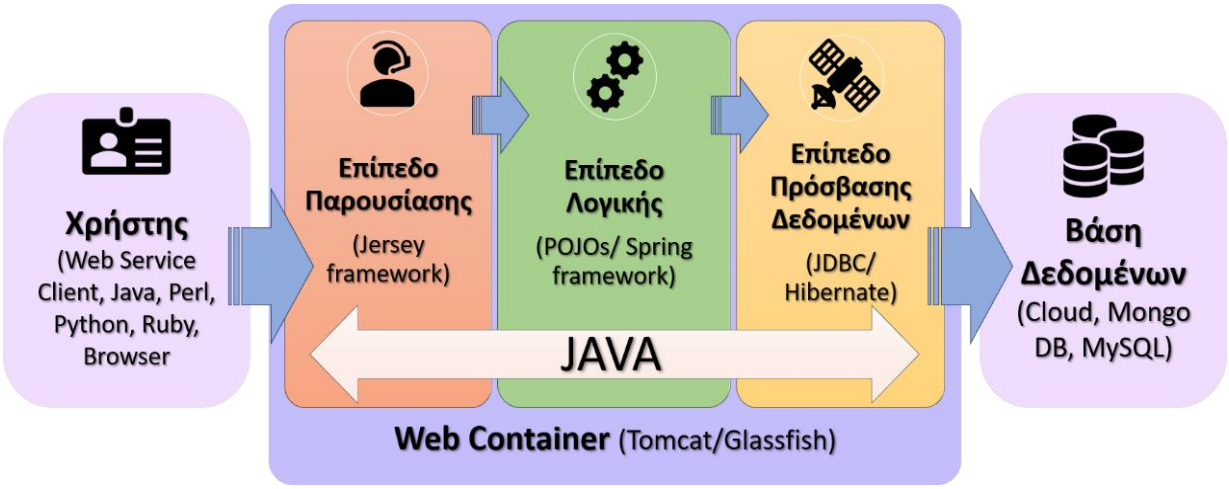
4.2 Αρχιτεκτονική Τριών Επιπέδων Ανάπτυξης RESTful API σε Java



Εικόνα 10: Σημείο του Συστήματος ΔτΠ, στο Οποίο Αντιστοιχεί το RESTful API

Σε αυτήν την υποενότητα, παρουσιάζεται ο τρόπος η δομή του RESTful API, την οργάνωση του κώδικα δηλαδή, σε συνδυασμό με τα χαρακτηριστικά της αρχιτεκτονικής REST. Ας επισημανθεί σε αυτό το σημείο, πως το περιεχόμενο της υποενότητας αυτής αναφέρεται στο σημείο του τοπικού διακομιστή του συστήματος ΔτΠ που εξηγήθηκε προηγουμένως.

Στην Εικόνα 11 φαίνεται ο τρόπος με τον οποίο ένα RESTful API χρησιμοποιεί άλλα προγραμματιστικά εργαλεία προκειμένου να μεταδοθεί η πληροφορία. Οι χρήστες μπορούν να αλληλοεπιδράσουν με το API μέσω του επίπεδου παρουσίασης, το οποίο στην συγκεκριμένη περίπτωση περιλαμβάνει όλα τα URIs, ή αλλιώς τους πόρους, τα οποία απαντούν σε αιτήματα HTTP.



Εικόνα 11: Αρχιτεκτονική Σχεδίασης RESTful API της Παρούσας Εργασίας

Στην υλοποίηση της εργασίας αυτής, το επίπεδο παρουσίασης φτιάχτηκε με τη βοήθεια του πλαισίου Jersey, το οποίο είναι ένα ειδικό πλαίσιο βιβλιοθηκών γλώσσας Java για την ανάπτυξη διαδικτυακών υπηρεσιών. Το πλαίσιο αυτό θα αναφερθεί και σε επόμενη υποενότητα αναλυτικότερα, μαζί με τις λεπτομέρειες των υπόλοιπων εργαλείων που χρησιμοποιούνται στην παρούσα υλοποίηση.

Στη συνέχεια, το επίπεδο λογικής περιλαμβάνει οποιεσδήποτε διεργασίες και λειτουργίες μπορεί να υπάρχουν και πρέπει να εκτελεσθούν μόλις υπάρξει κάποιο αίτημα HTTP. Αυτό μπορεί να περιλαμβάνει την οργάνωση και την επεξεργασία των εισερχόμενων δεδομένων, αν υπάρχουν, μέσω αιτημάτων POST, έως και απλές μεθόδους παραγωγής μηνυμάτων απάντησης στα αιτήματα HTTP, για παράδειγμα τη σύνθεση της απόκρισης HTTP σε κάποιο αίτημα GET. Το επίπεδο αυτό μπορεί να υλοποιηθεί με απλό κώδικα σε γλώσσα Java, ή και με κάποιο ειδικό πλαίσιο της τελευταίας, όπως το Spring framework.

Φυσικά, στη συνέχεια ακολουθεί το επίπεδο πρόσβασης δεδομένων, το οποίο δίνει στην εφαρμογή τη δυνατότητα να διαχειρίζεται οποιαδήποτε δεδομένα λαμβάνει ή αποστέλλει. Σαφώς, για να υπάρχει επίπεδο πρόσβασης δεδομένων θα πρέπει να υπάρχει στο άλλο άκρο της αρχιτεκτονικής μας και κάποια βάση δεδομένων για αποθήκευση αυτών και, ενδεχομένως, κάποιο σύστημα διαχείρισης αυτής. Κάθε βάση δεδομένων έχει το αντίστοιχο δικό της API για τη γλώσσα Java, το οποίο ονομάζεται JDBC (Java DataBase Connectivity) και προσφέρει σύνδεση και επεξεργασία δεδομένων μέσω κώδικα.

Τέλος, ταυτόχρονα με όλα τα επίπεδα, υπάρχει κι ένα ακόμη επίπεδο, το οποίο αποτελεί ουσιαστικά την πλατφόρμα πάνω στην οποία θα τρέχει το Java RESTful API. Αυτό το επίπεδο αποτελείται από έναν web container και συγκεκριμένα για αυτήν την υλοποίηση από έναν Apache Tomcat 9, ο οποίος είναι συγκεκριμένα ένας server για να τρέχει διαδικτυακές υπηρεσίες ανεπτυγμένες με τη γλώσσα Java.

4.3 Προγραμματιστικά Εργαλεία και Λογισμικό

Σε αυτήν την υποενότητα θα δοθεί ένα περίγραμμα των προγραμματιστικών εργαλείων που χρησιμοποιούνται για την υλοποίηση της εφαρμογής, αλλά και του λογισμικού που βοηθά σημαντικά τη διαδικασία της ανάπτυξης και της λειτουργίας αυτής.

4.3.1 Eclipse IDE

Το eclipse IDE (Integrated Development Environment) είναι ένα ολοκληρωμένο περιβάλλον, στο οποίο μπορεί κανείς να γράψει και να εκτελέσει κώδικα. Είναι σχεδιασμένο κυρίως για την ανάπτυξη Java εφαρμογών, αλλά πλέον είναι δυνατόν να υποστηρίξει και διάφορες άλλες γλώσσες προγραμματισμού μετά την εγκατάσταση των αντίστοιχων πρόσθετων.

4.3.2 Apache Maven

Η Maven είναι ένα προγραμματιστικό εργαλείο, το οποίο μπορεί να βοηθήσει σημαντικά στην εργονομία της ανάπτυξης μίας Java εφαρμογής. Τις περισσότερες φορές, κατά τη διάρκεια της ανάπτυξης, τα βασικά πακέτα κώδικα που προσφέρει αποκλειστικά η Java δεν είναι αρκετά και χρειάζεται η προσθήκη εξωτερικού κώδικα υπό μορφή αρχείων JAR (Java ARchive). Χρησιμοποιώντας το εργαλείο Maven μπορούμε να ενσωματώσουμε τέτοιου είδους αρχεία στον κώδικά μας αυτόματα, απλώς επιλέγοντας το όνομα του πακέτου που επιθυμούμε.

4.3.3 Jersey RESTful Web Services Πλαίσιο

Το Jersey πλαίσιο είναι μία ομάδα εργαλείων, η οποία κάνει την ανάπτυξη RESTful εφαρμογών πολύ πιο εύκολη και απλή. Η ανάπτυξη RESTful εφαρμογών στηρίζεται σημαντικά σε μία λειτουργία που προσφέρει η Java και ονομάζεται Annotation. Τα Annotations δεν είναι τόσο κώδικας όσο πληροφορία και μάλιστα τύπου μεταδεδομένων.

Το πλαίσιο Jersey, λοιπόν, προσφέρει μεταξύ άλλων, τα κατάλληλα Annotations ώστε να μπορεί να αναπτυχθεί μία RESTful εφαρμογή χωρίς ο προγραμματιστής να πρέπει να γράψει επιπλέον κώδικα για να τα δημιουργεί κάθε φορά.

4.3.4 Apache Tomcat

Όπως προαναφέρθηκε σύντομα και προηγουμένως, θα πρέπει να υπάρχει ένα είδος πλατφόρμας πάνω στην οποία μπορεί να τρέξει η εφαρμογή και μπορεί να δίνει όλα τα προαπαιτούμενα για μία διαδικτυακή εφαρμογή. Στην προκειμένη περίπτωση χρησιμοποιείται ένας Tomcat Διακομιστής Δικτύου και είναι το εργαλείο το οποίο μπορεί να τρέξει το RESTful API και να το κάνει διαθέσιμο μέσω διαδικτύου ώστε να είναι δυνατή η αλληλοεπίδραση με αυτό.

4.3.5 MongoDB Atlas

Η MongoDB Atlas είναι μία υποδομή υπολογιστικού νέφους και προσφέρει την υπηρεσία της βάσης δεδομένων. Στην συγκεκριμένη εφαρμογή χρησιμοποιείται για να αλληλοεπιδρά με αυτήν το API και να μπορεί να αποθηκεύει και να ανακτά δεδομένα. Σημαντικό πλεονέκτημα είναι ότι πρόκειται για μία υπηρεσία μέσω υπολογιστικού νέφους και αυτό σημαίνει ότι δεν απαιτούνται ρυθμίσεις ή εγκατάσταση σε υπολογιστή.

Αντίστοιχα, μέσω της Maven προστίθεται το κατάλληλο JAR αρχείο για τη συνδεσιμότητα με τη βάση δεδομένων, το οποίο είναι ουσιαστικά το JDBC πακέτο της βάσης Δεδομένων MongoDB. Η συγκεκριμένη βάση δεδομένων είναι μη σχεσιακή, το οποίο σημαίνει ότι μπορεί να αποθηκεύσει δεδομένα μορφής JSON, κάτι πολύ σημαντικό για την αρχιτεκτονική REST όπως εξηγήθηκε στην προηγούμενη ενότητα σχετικά με τα στοιχεία αυτής.

4.3.6 Node-RED (Node.js)

Το Node-RED είναι ένα ακόμη προγραμματιστικό εργαλείο και χρησιμοποιείται για να συνδέσει κανείς συσκευές, APIs και διαδικτυακές υπηρεσίες με διάφορους τρόπους. Ουσιαστικά, μέσα από την διεπαφή χρήστη που προσφέρει, συνδέουμε έτοιμους κόμβους, δημιουργώντας ροές, προκειμένου να συνθέσουμε την επιθυμητή λειτουργία, συνδέοντας λογισμικό και ίσως και συσκευές.

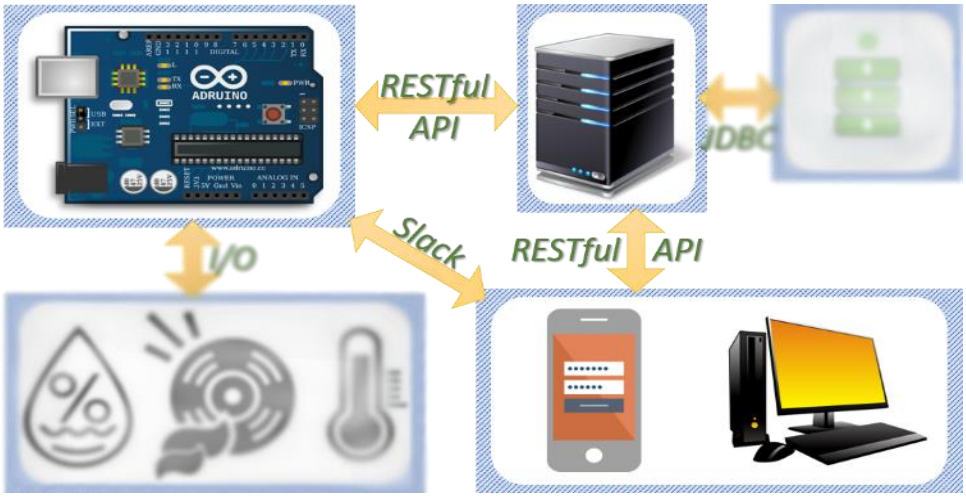
4.3.7 Slack

Το slack είναι ένα εργαλείο επικοινωνίας ανάμεσα σε ομάδες εργασίας προκειμένου να υπάρχει βέλτιστη οργάνωση μεταξύ τους. Λειτουργεί έχοντας ως κεντρικό άξονα τον εργασιακό χώρο μέσα στον οποίο δημιουργούνται κανάλια επικοινωνίας με συγκεκριμένα μέλη βάσει των λογαριασμών τους ηλεκτρονικού ταχυδρομείου.

4.4 Επίπεδο Παρουσίασης

Το επίπεδο παρουσίασης είναι αυτό το οποίο χρησιμοποιείται για την αλληλεπίδραση του χρήστη με το RESTful API. Πιο συγκεκριμένα στο σύστημά μας, αυτό αντιστοιχεί στην αλληλεπίδραση του τοπικού διακομιστή με τους μικροελεγκτές και τις συσκευές, όπως τονίζεται στην Εικόνα 12.

Σε προηγούμενη ενότητα αναφέρθηκε ότι ένα από τα βασικότερα πλεονεκτήματα της αρχιτεκτονικής REST είναι η απλότητα που συνεπάγεται από τη χρήση του πρωτοκόλλου HTTP, καθώς το μόνο που απαιτείται από τον χρήστη προκειμένου να αλληλοεπιδράσει με το API είναι μία μέθοδος HTTP και ένα URI. Το επίπεδο παρουσίασης στην συγκεκριμένη περίπτωση, είναι



Εικόνα 12: Αλληλεπίδραση του RESTful API με ολόκληρο το Σύστημα ΔτΠ

στην ουσία η κλάση SensorResource, της οποίας ο κώδικας της κλάσης αυτής φαίνεται στην πρώτη ενότητα του παραρτήματος. Περιέχει όλες τις μεθόδους που εκτελούνται για κάθε πόρο με περιεχόμενο αντικείμενα τύπου Sensor.

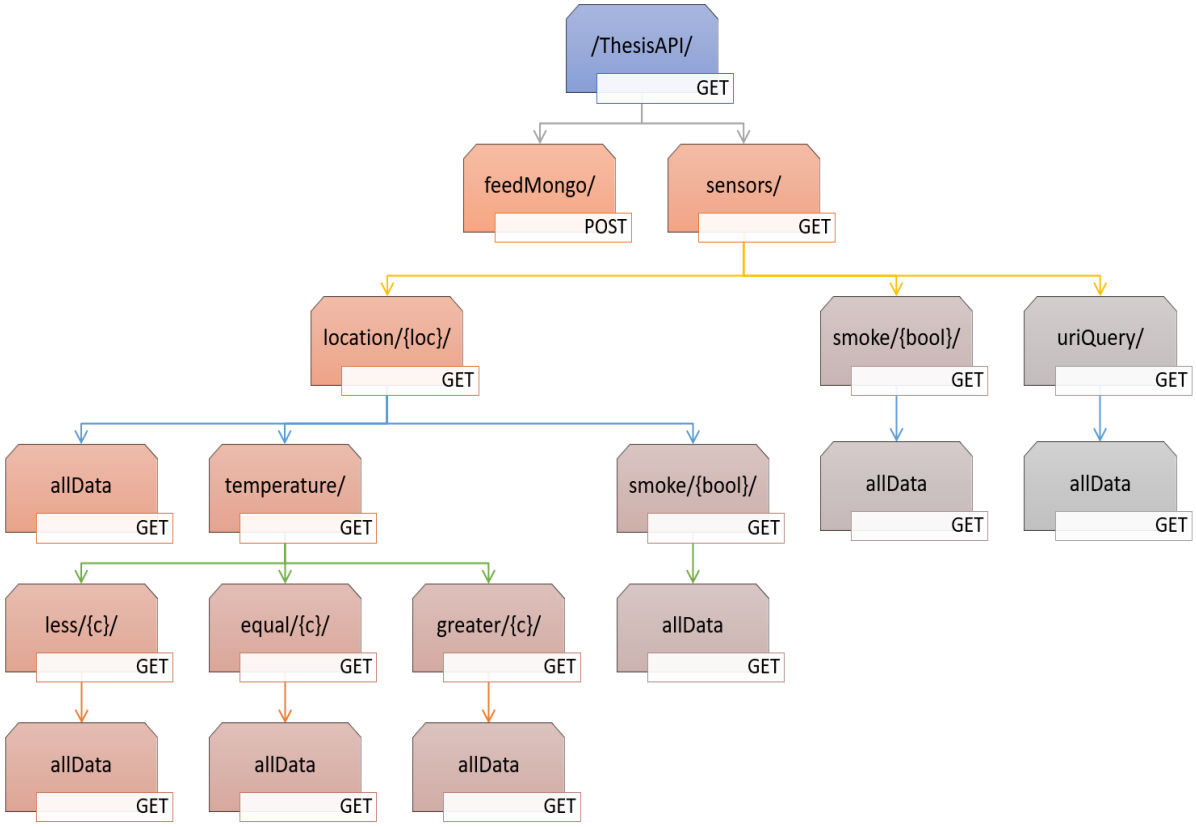
Το πλαίσιο Jersey βοηθά σε ακριβώς αυτό το σημείο της σχεδίασης, προσθέτοντας τη δυνατότητα χρήσης ειδικών Java Annotations, τα οποία επιτρέπουν την εύκολη «χαρτογράφηση» μεταξύ μεθόδων HTTP και Java με τη χρήση «μονοπατιών» από URIs. Ουσιαστικά αυτά τα μονοπάτια δείχνουν στους πόρους στους οποίους θα κάνουν αιτήματα οι χρήστες και θα απαντά ο διακομιστής. Προκειμένου να επικοινωνήσει ένας οποιοσδήποτε χρήστης με τον διακομιστή, απαιτώντας μία οποιαδήποτε ενέργεια, αρκεί να γνωρίζει αυτό το μονοπάτι.

Στην Εικόνα 13 απεικονίζεται ένα διάγραμμα ιεράρχησης για τα «μονοπάτια» από URIs των πόρων που σχηματίζονται και μπορεί να εξυπηρετήσει το API της εργασίας αυτής, ενώ σε επόμενη υποενότητα εξηγείται αναλυτικά τι κάνει ο κάθε πόρος.

Ας σημειωθεί εδώ, ότι στο ολοκληρωμένο URI υπάρχει και άλλο ένα κομμάτι που προηγείται της ρίζας αυτής και εξαρτάται κατά ένα μεγάλο βαθμό από τον διακομιστή δικτύου που χρησιμοποιούμε ώστε να τρέξει το API, και τις ρυθμίσεις αυτού. Για παράδειγμα, προκειμένου να στείλει κανείς το αίτημα GET στη ρίζα του API, το ολοκληρωμένο URI θα είχε ίσως την εξής μορφή: *"http://hostname:portNo/ThesisAPI"*. Για την υπόλοιπη ενότητα όμως θα παραλείπουμε αυτό το κομμάτι καθώς δεν εξαρτάται και ούτε αφορά το API.

Επιπροσθέτως, ένας χρήστης μπορεί να είναι οποιαδήποτε συσκευή που έχει πρόσβαση στο διαδίκτυο και μπορεί να κάνει αιτήματα HTTP. Για παράδειγμα, στο πλαίσιο της εργασίας αυτής, είναι ένας μικροελεγκτής, ο οποίος συγκεντρώνει δεδομένα από αισθητήρες ανά τακτά χρονικά διαστήματα και τα στέλνει σε μορφή JSON, στον πόρο *"ThesisAPI/post2Mongo"* χρησιμοποιώντας ένα αίτημα HTTP POST.

Από την άλλη, ως χρήστης μπορεί να λειτουργήσει και ένας απλός φυλλομετρητής, ο οποίος μπορεί να μας βοηθήσει να εκτελέσουμε αιτήματα HTTP με σχετικά χειροκίνητο τρόπο. Με αυτόν τον τρόπο θα επαληθεύσουμε εύκολα το API μας, εκτελώντας απλά αιτήματα GET για ανάκτηση δεδομένων από τη βάση δεδομένων.



Εικόνα 13: Δέντρο Πόρων URI

Για παράδειγμα, αν κάποιος «χτυπήσει» τη διεύθυνση *"ThesisAPI/SensorLocation/Warehouse-A/allData"*, τότε θα φτάσει στο RESTful API ένα αίτημα HTTP GET και θα του επιστραφούν όλες οι καταχωρήσεις που υπάρχουν στη βάση δεδομένων, οι οποίες έχουν στην παράμετρο *Location* την τιμή "Warehouse-A". Με παρόμοιο τρόπο λειτουργούν και όλοι οι υπόλοιποι πόροι, μέσω των «μονοπατιών» που φαίνονται στην Εικόνα 13.

Στη συνέχεια παρουσιάζεται αναλυτικά ο κώδικας πρόσβασης του πόρου του παραδείγματος που μόλις αναφέρθηκε, ενώ ύστερα θα γίνει μία ονομαστική περιγραφή όλων των πόρων που δίνει το συγκεκριμένο RESTful API. Ολόκληρος ο κώδικας του API παρουσιάζεται στο παράρτημα, ενώ παράλληλα ίσως χρειαστεί να γίνουν αρκετές αναφορές σε αυτό χωρίς να αναλυθούν όλα τα κομμάτια του κώδικα εκτενέστερα.

4.4.1 Προγραμματιστική Ανάλυση Διεπαφής - Πλαίσιο Βιβλιοθηκών Jersey(JAX-RS)

Όπως αναφέρθηκε σύντομα και στην περιγραφή των προγραμματιστικών εργαλείων που θα χρησιμοποιηθούν για το API, ένα πολύ σημαντικό κομμάτι της Java όσον αφορά τη σχεδίαση RESTful διεπαφών είναι τα Annotations, τα οποία είναι κομμάτια κώδικα και ουσιαστικά δίνουν οδηγίες στον μεταφραστή σχετικά με την εκτέλεση ενός μπλοκ κώδικα. Το πλαίσιο Jersey παρέχει όλα τα απαραίτητα Annotations για τη δημιουργία των RESTful APIs.

Ένα Annotation ξεκινά πάντα με το σύμβολο «@» και έτσι γνωρίζει ο μεταφραστής ότι για την κλάση/ μέθοδο/ που ακολουθεί θα πρέπει να ακολουθήσει συγκεκριμένες οδηγίες προκειμένου να το εκτελέσει. Το πιο γνωστό Annotation που υπάρχει και

μάλιστα ενσωματωμένο στο βασικό πακέτο της Java είναι το "@Override", το οποίο συνήθως εφαρμόζεται σε ήδη υπάρχουσες μεθόδους και υπαγορεύει στον compiler ότι η μέθοδος που ακολουθεί δεν θα εκτελέσει τον κώδικα που ήδη υπάρχει, αλλά ένα διαφορετικό μπλοκ από κώδικα που ακολουθεί.

Στο παρακάτω πλαίσιο παρουσιάζεται ο κώδικας του πόρου του παραδείγματος που προαναφέρθηκε και μπορεί κανείς να παρατηρήσει αρκετά annotations, τα οποία τα δίνει έτοιμα το πλαίσιο Jersey για Java προγραμματισμό.

```
@GET
@Path("sensors/location/{loc}/allData")
@Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
public Response sensorLocationAllData( @PathParam ("loc") String loc, @Context
                                    HttpHeaders headers) {

        List<Sensor> list = repo.giveAllDataWith("location", loc, "REST-MongoDB",
        "SensorData");

        CacheControl cc = new CacheControl();
        cc.setNoCache(true);
        cc.setMaxAge(3600);

        return buildResp(list, cc, headers);
}
```

Εικόνα 14: Κώδικας Java ενός Πόρου του RESTful API για Αιτήματα GET

Αναλυτικότερα:

- ***@GET***: Το annotation αυτό προσδιορίζει ότι προκειμένου να κληθεί η μέθοδος την οποία χαρακτηρίζει, ένας χρήστης να πρέπει να στείλει ένα αίτημα HTTP GET σε κάποιο συγκεκριμένο URI. Αντίστοιχα υπάρχουν και για τις υπόλοιπες μεθόδους HTTP τα annotations *@POST*, *@UPDATE*, *@DELETE*.
- ***@Path***: Με αυτό το annotation προσδιορίζεται το μονοπάτι στο οποίο θα πρέπει να δώσει ένας χρήστης ένα αίτημα HTTP. Το *@Path* είναι το σημαντικότερο ίσως Annotation για ένα RESTful API καθώς αυτό είναι το οποίο μας βοηθά να χαρτογραφήσουμε όλους τους πόρους, σχηματίζοντας τα μονοπάτια της Εικόνας 13.
- ***@Produces***: Αναφέρθηκε σε προηγούμενη υποενότητα ότι είναι πολύ σημαντικό να αναφέρεται στις κεφαλίδες της απάντησης HTTP η μορφή των δεδομένων που αποστέλλονται. Με το Annotation *@Produces* ουσιαστικά δηλώνονται οι μορφές στις οποίες μπορεί να μετατρέψει μία μέθοδος Java τα δεδομένα για να τα αποστείλει. Στην συγκεκριμένη περίπτωση δίνεται η δυνατότητα μορφής JSON και μορφής XML.
- ***@PathParam***: Πολλές φορές, προκειμένου να πραγματοποιηθεί ένα αίτημα GET, το οποίο απαιτεί κάποιου είδους φιλτράρισμα που πρέπει να γίνει σε μία ποσότητα δεδομένων, είναι απαραίτητο να περαστούν κάποιες παράμετροι. Αυτό, στην περίπτωση χρήσης URI, μπορεί να γίνει με δύο τρόπους: Είτε η παράμετρος να είναι κομμάτι του ίδιου του URI, είτε να

συνταχτεί ένα *query* αμέσως μετά το URI. Το Annotation *@PathParam* καλύπτει την πρώτη περίπτωση όπου αναμένεται να υπάρχει η τιμή μιας συγκεκριμένης παραμέτρου σε συγκεκριμένο σημείο μέσα στο URI.

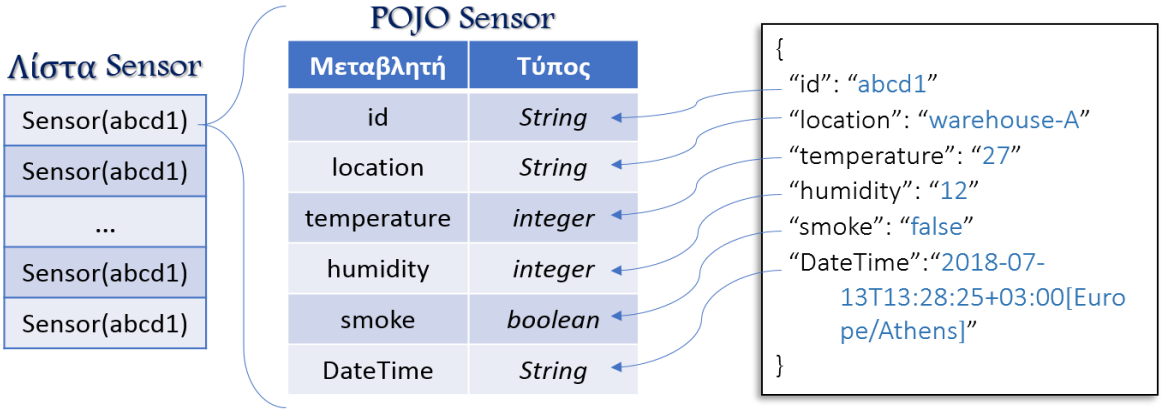
- ***@Context***: Το Annotation αυτό χρησιμοποιείται για να τεθούν μέσα σε μεταβλητές διάφορες πληροφορίες σχετικά με το αίτημα που πραγματοποιείται, προκειμένου να χρησιμοποιηθούν για την καλύτερη εξυπηρέτηση ενός χρήστη από το RESTful API. Έτσι, είναι δυνατόν να γνωρίζει η κάθε μέθοδος Java τις απαραίτητες πληροφορίες για τις κεφαλίδες ενός αιτήματος, τις λεπτομέρειες του URI από το οποίο κλήθηκε, το περιεχόμενο ασφάλειας και αρκετά ακόμα χαρακτηριστικά, και έτσι να αποστείλει την αντίστοιχη απάντηση HTTP. Στην συγκεκριμένη περίπτωση, μπορεί να χρησιμοποιηθεί ώστε να επιστραφούν τα εκάστοτε δεδομένα στον χρήστη σε μορφή (JSON ή XML) που έχει εκφράσει μέσω της αντίστοιχης κεφαλίδας στο αίτημα που έχει αποστείλει.
- ***@XmlRootElement***: Το Annotation αυτό είναι υπεύθυνο για την ακριβή μετατροπή που κάνει η Java μεταξύ των δικών της δομών δεδομένων (μεταβλητές, διανύσματα κ.α.) και στοιχείων XML.

Το μεγαλύτερο κομμάτι της λειτουργίας του επιπέδου παρουσίασης του RESTful API οφείλεται στα έτοιμα Annotations που παρέχει το πλαίσιο Jersey. Τα παραπάνω Annotations που εξηγήθηκαν αναλυτικά, χρησιμοποιούνται από τον κώδικα όλων των πόρων που έχουν δημιουργηθεί σε αυτό το επίπεδο. Ωστόσο, προκειμένου να είναι ολοκληρωμένη η επεξήγηση του κώδικα που χρησιμοποιείται τόσο στο παραπάνω παράδειγμα όσο και για τους υπόλοιπους πόρους, θα ακολουθήσει η πλήρης εξήγηση και της Java μεθόδου, όπως και μίας ακόμη επιπλέον μεθόδου που θα χρειαστεί.

Όπως μπορεί κανείς να δει στην Εικόνα 14, ο πόρος με το URI "ThesisAPI/sensorLocation/{location}/allData", εξυπηρετείται από μία μέθοδο Java τύπου *Response* κάτι το οποίο σημαίνει ότι θα πρέπει να επιστρέφει Java αντικείμενα, του ίδιου τύπου. Η Response είναι μία οντότητα Java που φτιάχνει μηνύματα HTTP. Συνεπώς περιέχει μεταβλητές ώστε να προσφέρει ένα πλήρες μήνυμα HTTP, συμπεριλαμβανομένων των παραμέτρων γραμμής εκκίνησης, επικεφαλίδων και πλήρους σώματος.

Στη συνέχεια, η πρώτη εντολή της μεθόδου είναι να φτιάξει μία λίστα από αντικείμενα τύπου Sensor (Plain Java Object, POJO) και εκεί να βάλει όλες τις καταχωρήσεις της βάσης δεδομένων, σύμφωνα με τις παραμέτρους που δίνουμε, οι οποίες είναι η μεταβλητή την οποία ελέγχουμε για μία συγκεκριμένη τιμή, η τιμή αυτή, το όνομα και η συλλογή της βάσης δεδομένων MongoDB Atlas.

Ύστερα, καθορίζεται η κεφαλίδα *Cache-Control* της απάντησης HTTP, ορίζοντας ένα αντικείμενο τύπου CacheControl και θέτοντας τις μεταβλητές *setNoCache* και *setMaxAge* ανάλογα με τις απαιτήσεις των δεδομένων κάθε φορά. Αν τα δεδομένα που αποστέλλονται δεν αλλάζουν με το χρόνο τότε η πρώτη μεταβλητή τίθεται *false,* καθώς τα δεδομένα μπορούν να αποθηκευτούν σε μνήμη cache, ενώ στην αντίθετη περίπτωση τίθεται *true* και επιπλέον τίθεται και ο χρόνος σε δευτερόλεπτα στον οποίο σταματάνε αυτά τα δεδομένα να είναι έγκυρα. Στην συγκεκριμένη περίπτωση αυτό ο χρόνος ορίζεται στην μία ώρα καθώς αυτός είναι ο χρόνος για τον οποίο έχει οριστεί να προστίθενται καινούργια δεδομένα από τους αισθητήρες.



Εικόνα 15: Ενσωμάτωση Δεδομένων JSON σε ένα POJO και Προσθήκη Αυτού σε μία Λίστα Τύπου Sensor

Τέλος, η τελευταία εντολή ουσιαστικά γυρνά ένα αντικείμενο τύπου Response καλώντας μία δεύτερη μέθοδο, η οποία είναι φτιαγμένη με τέτοιο τρόπο, ώστε να περνάνε τα δεδομένα της βάσης δεδομένων, οι παράμετροι της Cache και οι κεφαλίδες του αιτήματος και να φτιάχνει αυτόματα την HTTP απάντηση που πρέπει να αποσταλεί.

4.4.2 Τα Υπερμέσα ως η Μηχανή Κατάστασης Εφαρμογής (HATEOAS)

Ένα από τα πιο σημαντικά χαρακτηριστικά της αρχιτεκτονικής REST είναι οι ομοιόμορφες διεπαφές, όπως αναφέρθηκε στην ενότητα δύο, και για αυτόν τον λόγο εφαρμόζονται τέσσερις επιπλέον βασικοί περιορισμοί. Ένας από αυτούς είναι η χρήση των Υπερμέσων ως Μηχανή Αναζήτησης Κατάστασης Εφαρμογής ή όπως προκύπτει από την αγγλική μετάφραση και είναι γνωστό ως *HATEOAS* (Hypermedia As The Engine Of Application State).

Στην ουσία πρόκειται για ένα εργαλείο ώστε να μπορεί ένας χρήστης να πλοηγηθεί στο δέντρο των URI της Εικόνας 13 και να βρει τον πόρο που απαιτεί η όποια λειτουργία. Με αυτόν τον τρόπο δεν χρειάζεται να γνωρίζει κάποιος εκ των προτέρων όλα τα URI για τους πόρους που μπορεί να προσφέρει μία RESTful διεπαφή, παρά μόνο το ένα αρχικό URI, το οποίο χρησιμοποιείται ως *Μηχανή Κατάστασης Εφαρμογής*.

Πρακτικά, ένας πόρος, ο οποίος λειτουργεί ως HATEOAS, στέλνει ένα μήνυμα HTTP που περιέχει μία λίστα από URIs μαζί με μία σύντομη, αν όχι μονολεκτική, περιγραφή τους. Μάλιστα αυτές οι πληροφορίες θα πρέπει να είναι διαθέσιμες τόσο σε XML, όσο και σε JSON, πρότυπα που χρησιμοποιούνται ευρέως σε όλο το διαδίκτυο. Ο κώδικας για ένα τέτοιο μήνυμα HTTP δεν διαφέρει από τον παραπάνω που μόλις αναλύθηκε, παρά μόνο στο σημείο όπου αντί στο σώμα του μηνύματος να αποστέλλεται μία λίστα από αντικείμενα τύπου Sensor, αποστέλλεται ένα αντικείμενο τύπου Message, το οποίο είναι έτσι φτιαγμένο ώστε να ενσωματώνει δεδομένα HATEOAS.

Στο παράρτημα Α μπορεί κανείς να δει στις γραμμές 50-77 του κώδικα, μία μέθοδο ενός μηνύματος HTTP με περιεχόμενο HATEOAS. Αξίζει κανείς να παρατηρήσει ότι σε αντίθεση με το παράδειγμα της προηγούμενης υποενότητας, το συγκεκριμένο μήνυμα μπορεί να αποθηκευτεί σε μνήμη cache ώστε τα δεδομένα του να είναι επαναχρησιμοποιούμενα. Αυτό φαίνεται στην γραμμή όπου τίθεται στην μεταβλητή *NoCache* η τιμή *false*.

Στο συγκεκριμένο API υπάρχουν δύο πόροι με δεδομένα HATEOAS με τον πρώτο να είναι η ρίζα του (“ThesisAPI/”), ο οποίος παρέχει URIs για όλους τους πόρους που προκύπτουν άμεσα από

αυτόν, όπως φαίνεται στην Εικόνα 13, ενώ ο δεύτερος, είναι σχετικά με την αναζήτηση δεδομένων από τους αισθητήρες βάσει θερμοκρασίας ("ThesisAPI/temperature/allData").

Στην Εικόνα 16 φαίνονται τα δεδομένα σε μορφή XML που αποστέλλονται από ένα αίτημα HTTP GET στο "ThesisAPI/temperature/allData". Όπως φαίνεται υπάρχουν τρία πεδία τύπου *link* μέσα στα οποία υπάρχουν οι μεταβλητές *link* και *rel* αντίστοιχα για κάθε έναν υπερσύνδεσμο που υπάρχει ως συνέχεια του παρόν πόρου. Η μεν *link* μεταβλητή περιέχει τον αντίστοιχο σύνδεσμο, η δε *rel* μεταβλητή περιέχει πολύ σύντομα μία περιγραφή του τελευταίου.

Έτσι, μέσω των δεδομένων HATEOAS, δεν είναι απαραίτητο ούτε ένας διακομιστής να διαφημίζει όλους τους πόρους που προσφέρει, ούτε ένας χρήστης να ψάχνει πολύ ώστε να βρει αυτό που απαιτείται. Αρκεί να είναι γνωστό το URI του πόρου που οδηγεί στις κεντρικές πληροφορίες όπως φαίνεται στην παραπάνω εικόνα.

```
<message>
        <link>
            <link>http://localhost:8081/rest/webapi/ThesisAPI/temperatu
            re/allData/greater/25</link>
            <rel>Greater25</rel>
        </link>
        <link>
            <link>http://localhost:8081/rest/webapi/ThesisAPI/temperatu
            re/allData/equal/25</link>
            <rel>Equal25</rel>
        </link>
        <link>
            <link>http://localhost:8081/rest/webapi/ThesisAPI/temperatu
            re/allData/less/25</link>
            <rel>LessThan25</rel>
        </link>
        <message>Select criteria for temperature</message>
</message>
```

Εικόνα 16: Δεδομένα HATEOAS από Απόκριση HTTP σε μορφή XML

4.4.3 Πόροι URI

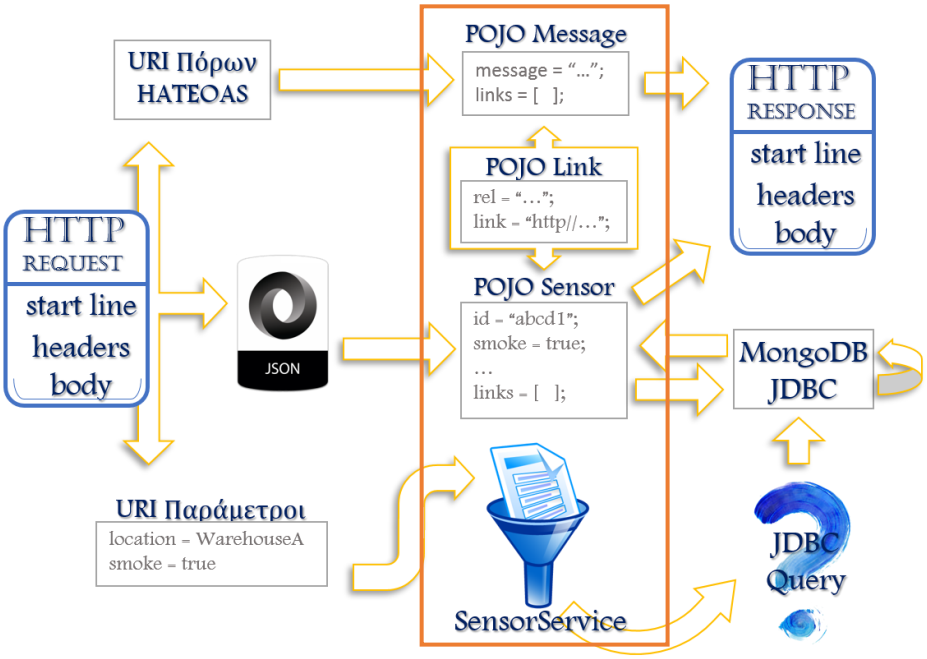
Σε αυτήν την υποενότητα γίνεται αναλυτική επεξήγηση των λειτουργιών όλων των URI που οδηγούν στους πόρους της Εικόνας 13. Σε επόμενη ενότητα θα παρατεθούν και εικόνες σχετικά με τη λειτουργία του API, δείχνοντας αιτήματα HTTP και απαντήσεις προς και από τους πόρους αυτούς.

- *"ThesisAPI/"*: Πόρος μεθόδου GET. Χρησιμοποιείται ως πόρος HATEOAS, παραπέμποντας στα αμέσως επόμενα URI βάσει της Εικόνας 13.
- *"ThesisAPI/feedMongo/"*: Πόρος μεθόδου POST. Χρησιμοποιείται από τον εκάστοτε μικροελεγκτή προκειμένου να αποστείλει τα δεδομένα των αισθητήρων μέσω διαδικτύου στο RESTful API.
- *"ThesisAPI/sensors/":* Πόρος μεθόδου GET. Πρόκειται για την επιστροφή ενός απλού JSON το οποίο περιέχει σε κείμενο μία περιγραφή των δεδομένων και της λειτουργίας ολόκληρου του API.

- *"ThesisAPI/sensors/smoke/{boolean}/"*: Πόρος μεθόδου GET. Χρησιμοποιείται από έναν χρήστη προκειμένου να πραγματοποιήσει αναζήτηση μίας παραμέτρου στη βάση δεδομένων, με βάση την τιμή της μεταβλητής *smoke*.
- *"ThesisAPI/sensors/location/{location}/"*: Πόρος μεθόδου GET. Χρησιμοποιείται από τον εκάστοτε χρήστη προκειμένου να πραγματοποιήσει αναζήτηση δεδομένων βάσει μίας παραμέτρου, η οποία εμπεριέχεται μέσα στο URI. Ο συγκεκριμένος πόρος αναζητά την τελευταία καταχώρηση χρονικά, της οποίας η τοποθεσία είναι ίδια με την τιμή *{location}*.
- *"ThesisAPI/sensors/location/{location}/temperature/"*: Πόρος μεθόδου GET. Χρησιμοποιείται ως σελίδα HATEOAS, παραπέμποντας στα κατώτερα URI του δέντρου, όπως φαίνονται στην Εικόνα 13, στα οποία πλέον η αναζήτηση αφορά δύο μεταβλητές και όχι μία.
- *"ThesisAPI/sensors/location/{location}/temperature/less/{c}"*: Πόρος μεθόδου GET. Χρησιμοποιείται για αναζήτηση δεδομένων δεύτερης παραμέτρου, η επιθυμητή τιμή της οποίας εμπεριέχεται μέσα στο URI, στην μεταβλητή *c, ως συνέχεια της προηγούμενης μεταβλητής*. Πιο συγκεκριμένα, αυτό το URI αναζητά την χρονικά τελευταία καταχώρηση στην οποία η θερμοκρασία είναι μεγαλύτερη της τιμής c.
- *"ThesisAPI/ sensors/location/{location}/temperature/equal/{c}"*: Πόρος μεθόδου GET. Χρησιμοποιείται όπως και τα προηγούμενα URI με την διαφορά ότι αναζητά την χρονικά τελευταία καταχώρηση στην οποία η θερμοκρασία είναι ίση με την τιμή *c*.
- *"ThesisAPI/sensors/location/{location}/temperature/greater/{c}"*: Πόρος μεθόδου GET. Αντίστοιχα με τους προηγούμενους πόρους, έτσι και αυτός αναζητά τις καταχωρήσεις ανάλογα με την θερμοκρασία. Ο συγκεκριμένος αναζητά την χρονικά τελευταία καταχώρηση όπου η θερμοκρασία είναι μικρότερη της δοθείσας τιμής *c*.
- *"ThesisAPI/ sensors/location/{location}/smoke/{boolean}"*: Πόρος μεθόδου GET. Χρησιμοποιείται, ομοίως με τους πόρους αναζήτησης δεύτερης μεταβλητής βάσει τοποθεσίας και θερμοκρασίας εμφανίζοντας την τελευταία καταχώρηση ανάλογα της τιμής *{boolean}*. Ωστόσο η συγκεκριμένη μεταβλητή είναι τύπου *boolean*, κάτι το οποίο σημαίνει πως μπορεί να πάρει μόνο δύο τιμές, *true* ή *false*, ανάλογα της ύπαρξης καπνού ή όχι.
- *"ThesisAPI/sensors/uriQuery/"*: Πόρος μεθόδου GET. Χρησιμοποιείται από τον εκάστοτε χρήστη του RESTful API προκειμένου να πραγματοποιήσει URI αναζήτηση στα μέχρι εκείνη τη στιγμή αποθηκευμένα δεδομένα, εμφανίζοντας την πιο πρόσφατη καταχώρηση. Α9παιτεί οι παράμετροι της αναζήτησης να παρατίθενται ακριβώς μετά το URI. Παράδειγμα μιας τέτοια περίπτωσης είναι: *"ThesisAPI/dataByQuery/?smoke=true&location=Warehouse-B"*, όπου απαιτούμε όλες τις καταγραφές στις οποίες εντοπίστηκε καπνός στην τοποθεσία Warehouse-B.
- *"ThesisAPI/sensors/{ ... }/allData"*: Πόροι μεθόδου GET. Για τους περισσότερους από τους παραπάνω πόρους, υπάρχει η δυνατότητα να εμφάνισης όλων των καταχωρήσεων της εκάστοτε αναζήτησης. Αυτό επιτυγχάνεται απλώς με την προσθήκη του "/allData" στα αντίστοιχα URI.

4.5 Επίπεδο Λογικής

Το επίπεδο λογικής είναι το κομμάτι εκείνο του κώδικα, το οποίο οργανώνει την πληροφορία, είτε εισερχόμενη είτε εξερχόμενη, βάσει λογικής και την αντιστοιχεί σε οντότητες που καταλαβαίνει η Java. Πιο συγκεκριμένα, πρόκειται για ένα συγκεκριμένο είδος κλάσεων που παράγουν Απλά Αντικείμενα Java ή POJO (*Plain Old Java Object*), προκειμένου να περιγράψουμε πολύ συγκεκριμένες έννοιες. Σύμφωνα με τον ορισμό τα POJOs είναι αντικείμενα, τα οποία δεν περιορίζονται από τίποτα πέρα από την ίδια τη γλώσσα Java. Έτσι ένα POJO ιδανικά, δεν θα πρέπει να προεκτείνει ή να ενσωματώνει μεθόδους ή λειτουργίες από άλλες κλάσεις.



Εικόνα 17: Η Διαδρομή που Ακολουθούν τα Δεδομένα Μέσα στο RESTful API και η Θέση του Επιπέδου Λογικής με Κόκκινο Χρώμα

Επίσης στο επίπεδο λογικής ανήκουν και τα κομμάτια κώδικα που βοηθούν και στην αναζήτηση δεδομένων. Πιο συγκεκριμένα υπάρχει η κλάση SensorService, όπως φαίνεται και στο παράρτημα Ε, το οποίο περιέχει τέσσερις μεθόδους που βοηθούν το επίπεδο παρουσίασης στην διαχείριση δεδομένων.

Στην Εικόνα 17 φαίνεται συνοπτικά η διαδρομή που ακολουθούν τα δεδομένα μέσα στον κώδικα του API, καθώς και τη θέση κάθε κλάσης του επιπέδου αυτού στην λειτουργία.

4.5.1 Φορέας Δεδομένων JSON (POJO Sensor)

Τα αντικείμενα τύπου *Sensor* είναι οι αμιγώς φορείς της πληροφορίας των αισθητήρων που λαμβάνονται είτε από τους μικροελεγκτές, είτε από τη βάση δεδομένων. Περιέχουν τις μεταβλητές *id, location, temperature, humidity, smoke, dateTime*, οι οποίες εκφράζουν το JSON της Εικόνας 8. Επιπλέον, υπάρχει και μία μεταβλητή τύπου *List*, η οποία αποθηκεύει αντικείμενα τύπου *Link*.

Προκειμένου να μπορέσει μια οποιαδήποτε εφαρμογή Java να διαχειριστεί πληροφορίες και δεδομένα θα πρέπει με κάποιο τρόπο να τα συγκεκριμενοποιήσει, να τα βάλει δηλαδή σε μεταβλητές, τις οποίες μπορούμε εμείς να διαχειριστούμε αναλόγως. Τα αντικείμενα τύπου Sensor χρησιμοποιούνται για ακριβώς αυτόν τον σκοπό μέσα στο API. Κάθε φορά που λαμβάνει το API δεδομένα από τον συγκεκριμένο μικροελεγκτή, αυτόματα δημιουργεί ένα αντικείμενο τύπου

Sensor, και χρησιμοποιώντας τα δεδομένα JSON, εκχωρεί αυτόματα όλα τα μεγέθη σε μεταβλητές. Στη συνέχεια, αυτό το αντικείμενο το παραλαμβάνει το επίπεδο πρόσβασης δεδομένων και το διαχειρίζεται με τις ανάλογες εντολές προκειμένου να αποσταλεί στη βάση δεδομένων.

Ωστόσο, θα μπορούσε κανείς να σκεφτεί πως αφού η Java μπορεί να διαχειριστεί δεδομένα JSON δεν υπάρχει λόγος για όλη τη «φασαρία» των POJOs, καθώς θα μπορούσαμε να στείλουμε τα ίδια τα δεδομένα στην κλάση που διαχειρίζεται τη σύνδεση με τη βάση δεδομένων και από εκεί να σταλούν κατευθείαν.

Το πρόβλημα με αυτήν την τακτική όμως, είναι πως ο κώδικας κορμού της Java δεν έχει φτιαχτεί για να διαχειρίζεται δεδομένα συγκεκριμένης δομής ή μορφής, με εξαίρεση τις δικές της δομές δεδομένων (*integer, float, String, arrays* κ.α.).

Έτσι, η οποιαδήποτε διαχείριση δεδομένων JSON γίνεται με τη χρήση τρίτου κώδικα, δηλαδή με τη χρήση επιπλέον πακέτων κώδικα. Παράλληλα, οι υπεύθυνοι σχεδίασης του κώδικα συνδεσιμότητας (Java DataBace Connectivity, JDBC) των βάσεων δεδομένων, είναι αδύνατον να γνωρίζουν εκ των προτέρων τις δομές δεδομένων που θα χρησιμοποιήσει ο εκάστοτε προγραμματιστής που κάνει χρήση του JDBC. Επομένως, είναι πλέον σίγουρο ότι σε κάποιο σημείο κατά τη διάρκεια της ανάπτυξης μιας Java εφαρμογής θα δημιουργηθούν προβλήματα συμβατότητας στους τύπους δεδομένων. Για αυτόν το λόγο χρησιμοποιούμε τα POJOs για ομαλή μετάβαση των δεδομένων, προκειμένου να απεικονίσουμε τις οποιεσδήποτε πληροφορίες σε αντικείμενα.

Επιπροσθέτως, στο συγκεκριμένο POJO υπάρχει και η μεταβλητή *links*, όπως αναφέρθηκε και νωρίτερα. Η μεταβλητή αυτή χρησιμοποιείται προκειμένου να μπορέσουμε να προσθέσουμε το χαρακτηριστικό του HATEOAS που αναλύθηκε εκτενώς στην προηγούμενη υποενότητα. Έτσι, σε κάθε αίτημα GET που μπορεί κάποιος να κάνει μέσω του επιπέδου παρουσίασης, κάθε καταχώρηση τοποθετείται σε ένα αντικείμενο τύπου Sensor και προσθέτει διαθέσιμα URI που μπορεί να υπάρχουν, ως αντικείμενα στη λίστα *links*.


```
{
 "dateTime":"2018-07-13T21:30:37+03:00[Europe/Athens]",
 "humidity":5,
 "id":"abcd1",
 "links":[{   "link":"ThesisAPI/sensors/location/Warehouse-A/allData",
              "rel":"all data"
               }],
 "location":"Warehouse-A",
 "smoke":false,
 "temperature":38
}
```


Εικόνα 18: JSON Αναπαράσταση ενός Sensor Αντικειμένου

Για παράδειγμα, στην Εικόνα 18 φαίνεται το σώμα της απόκρισης HTTP σε JSON μορφή, ως απάντηση ενός αιτήματος GET στο URI *"ThesisAPI/SensorID/abcd1/"*, το οποίο επιστρέφει την τελευταία καταχώρηση του μικροελεγκτή με αναγνωριστικό "abcd1".

Το παραπάνω JSON αντιστοιχεί σε ένα αντικείμενο τύπου *Sensor* με τις μεταβλητές που αναφέρθηκαν να έχουν τις αντίστοιχες τιμές, ενώ ταυτόχρονα μέσα στη λίστα Links υπάρχει ένα

αντικείμενο τύπου *Link* με τις μεταβλητές *link* και *rel* να έχουν επίσης τις αντίστοιχες μεταβλητές. Ο κώδικας της κλάσης Sensor φαίνεται υπάρχει στην ενότητα Β του παραρτήματος.

Συνοψίζοντας, η κλάση Sensor.java αποτελεί το καλούπι για την δημιουργία *Sensor* POJOs, τα οποία χρησιμοποιούνται για την τοποθέτηση των δεδομένων, είτε αυτά προέρχονται από τον μικροελεγκτή και τους αισθητήρες, είτε από τη βάση δεδομένων μετά από αναζήτηση. Περιέχουν όλες τις απαραίτητες μεταβλητές για να μπορούν να ενσωματώνουν τη βασική πληροφορία των αισθητήρων, για την οποία λειτουργεί το API αυτό, ενώ ταυτόχρονα έχουν και τις απαραίτητες μεθόδους για την αλλαγή αυτών, αν χρειαστεί.

4.5.2 Φορέας Δεδομένων HATEOAS (POJO Message)

Η κλάση *Message* χρησιμοποιείται επίσης για να παράγει POJOs ώστε να εισαχθούν στο Σώμα των απαντήσεων HTTP, όπως και η κλάση *Sensor* που μόλις αναλύθηκε, αλλά με βασική διαφορά το περιεχόμενο της πληροφορίας που φέρει. Στις περιπτώσεις όπου το περιεχόμενο των HTTP μηνυμάτων δεν αφορά δεδομένα αισθητήρων ή καταχωρήσεις από τη βάση δεδομένων, αλλά αντ' αυτού αφορά δεδομένα HATEOAS, τότε τα αντικείμενα τύπου *Sensor* δεν έχουν καμία χρησιμότητα.

Αντίστοιχα με την κλάση *Sensor*, η κλάση *Message*, αποτελεί το καλούπι των POJOs, τα οποία μπορούν να μεταφέρουν δεδομένα HATEOAS. Έτσι, αυτά τα αντικείμενα περιέχουν μόνο δύο μεταβλητές, την *message*, η οποία είναι τύπου String, και την *links*, η οποία είναι τύπου List και αποθηκεύει αντικείμενα τύπου *Link*, τα οποία θα αναλυθούν αμέσως μετά.

Στην Εικόνα 19 βλέπουμε το περιεχόμενο των δεδομένων σε μορφή JSON, τα δεδομένα HATEOAS που περιέχονται στον έναν από τους δύο πόρους με τέτοιο περιεχόμενο. Καθώς δεν υπάρχουν δεδομένα αισθητήρων, αλλά μόνο δεδομένα HATEOAS, βλέπουμε ότι πρόκειται απλά για μία μεταβλητή με το όνομα *message*, η οποία περιγράφει τι ακριβώς είναι αυτά τα δεδομένα που βλέπουμε, και επιπλέον μία μεταβλητή με το όνομα links μέσα στην οποία βρίσκονται όλα τα αντικείμενα τύπου Link και περιγράφουν το καθένα από αυτά και έναν πόρο.

Επίσης, πρέπει να αναφερθεί ότι επειδή θα πρέπει και αυτά τα αντικείμενα να μπορούν να προστεθούν σε HTTP απαντήσεις υπό μορφή XML ή JSON, η κλάση *Message* συνοδεύεται από το annotation "@XmlRootElement".

4.5.3 Χρήση του POJO Link για την Διαχείριση Δεδομένων HATEOAS

Η κλάση *Link* είναι η πιο απλή και η πιο μικρή σε γραμμές κώδικα μέσα στο RESTful API. Η χρησιμότητά της είναι η δημιουργία ξεχωριστών POJOs, τα οποία περιέχουν την πιο απλή πληροφορία HATEOAS. Τα δεδομένα HATEOAS, όπως φάνηκε και στην Εικόνα 19, αποτελούνται από πολλά JSON αντικείμενα, εκ των οποίων το καθένα περιέχει δύο μεταβλητές, την *rel* και την *link*. Η πρώτη υπάρχει προκειμένου να καταχωρείται μία πολύ σύντομη έως και μονολεκτική περιγραφή του εκάστοτε πόρου που θέλουμε να προσθέσουμε είτε σε ένα αντικείμενο *Sensor*, είτε σε αντικείμενο *Message*, ενώ η δεύτερη υπάρχει για να καταχωρείται το URI που του αντιστοιχεί.

```
{ "message":"HATEOAS Home",
  "link":[
        { "link":"/ThesisAPI/feedMongo",
          "rel":"POST Data"},
        { "link":"/ThesisAPI/sensors",
          "rel":"Sensors Data Information"},
        { "link":"/ThesisAPI/sensors/location/Warehouse-A",
          "rel":"Last Data from Warehouse-A"},
        { "link":"/ThesisAPI/sensors/location/Warehouse-A/allData",
          "rel":"All Data from Warehouse-A"},
        ...
        { "link":"/ThesisAPI/sensors/location/Warehouse-E/allData",
          "rel":"All Data from Warehouse-E"},
        { "link":"/ThesisAPI/sensors/smoke/true",
          "rel":"Last seen smoke in any Warehouse"},
        { "link":"/ThesisAPI/sensors/smoke/true",
          "rel":"All Data where there has been smoke in any Warehouse"},
        { "link":"/ThesisAPI/sensors/uriQuery",
          "rel":"Last Data with a Custom URI Query"},
        { "link":"/ThesisAPI/sensors/uriQuery/allData",
          "rel":"All Data Data with a Custom URI Query"
        } ] }
```


Εικόνα 19: Δεδομένα HATEOAS από το URI ρίζα του RESTful API

Όπως προαναφέρθηκε, τα δύο POJOs του API αυτού, τα οποία φέρουν πληροφορία που μπαίνει σε HTTP απαντήσεις, περιέχουν και τα δύο μία μεταβλητή τύπου List, η οποία καταχωρεί αντικείμενα τύπου *Link*. Αξίζει να σημειωθεί σε αυτό το σημείο, πως η κλάση *Link* δεν συνοδεύεται από το annotation "@XmlRootElement", καθώς δεν χρειάζεται τα δεδομένα που φέρουν αυτά τα αντικείμενα να μεταφραστούν άμεσα σε μορφή XML. Αυτό πραγματοποιείται αργότερα, αφού τα *Link* αντικείμενα έχουν εισαχθεί στην μεταβλητή *links* των POJOs *Sensor* και *Message*, με την μετατροπή να πραγματοποιείται πάνω σε εκείνα τα αντικείμενα και κατ' επέκταση και σε αυτά.

4.5.4 Φιλτράρισμα Δεδομένων Μέσω της Κλάσης SensorService

Η κλάση SensorService διαφέρει σημαντικά από τα POJOs που αποτελούν επίσης το επίπεδο λογικής. Καταρχάς, δεν πρόκειται για κλάση που παράγει απλά αντικείμενα όπως τα POJOs, αντίθετα δημιουργείται μόνο ένα αντικείμενο της κλάσης αυτής και με αυτό αλληλοεπιδρά το επίπεδο παρουσίασης και το επίπεδο πρόσβασης δεδομένων. Αυτό επιτυγχάνεται επειδή με την κλάση αυτή δεν επικεντρωνόμαστε στα δεδομένα, στις μεταβλητές δηλαδή, αλλά στη διαχείριση αυτών μέσω των μεθόδων.

Πιο συγκεκριμένα, αυτή η κλάση περιέχει δέκα μεθόδους, οι οποίες αρχικοποιούν ένα αντικείμενο τύπου *BasicDBObject* και προέρχεται από τον JDBC κώδικα της MongoDB βάσης δεδομένων που χρησιμοποιούμε με αυτό το API. Το αντικείμενο αυτού του τύπου δίνει τη δυνατότητα σύνθεσης ενός ερωτήματος (*query*), το οποίο περιέχει συνθήκες και παραμέτρους που «φιλτράρουν» τα δεδομένα μία συλλογής δεδομένων της MongoDB βάσης.

Για παράδειγμα, από τον συνολικό όγκο δεδομένων, θέλουμε μόνο αυτά όπου η παράμετρος *location* έχει την τιμή *"Warehouse – B"*. Στο Πλαίσιο 20 φαίνεται ο κώδικας μίας εκ των τεσσάρων μεθόδων της κλάσης αυτής που σχηματίζουν ερωτήματα σαν αυτό που μόλις αναφέρθηκε. Όπως φαίνεται, η μέθοδος αυτή παίρνει τέσσερις παραμέτρους εισόδου τύπου String, μία για το όνομα της μεταβλητής που φιλτράρουμε, μία για την τιμή της παραμέτρου που επιθυμούμε, μία για το όνομα της *Βάσης Δεδομένων* και μία για το όνομα της *Συλλογής* της βάσης MongoDB.

```
public List<Sensor> giveAllDataWith( String key1, String value1, String dbName,
String collectionName){

        BasicDBObject query = new BasicDBObject();
        query.append(paramKey, paramValue);

        return db.allData(query, dbName, collectionName);
    }
```

Εικόνα 20: Κώδικας Μεθόδου "giveAllData" της Κλάσης SensorService

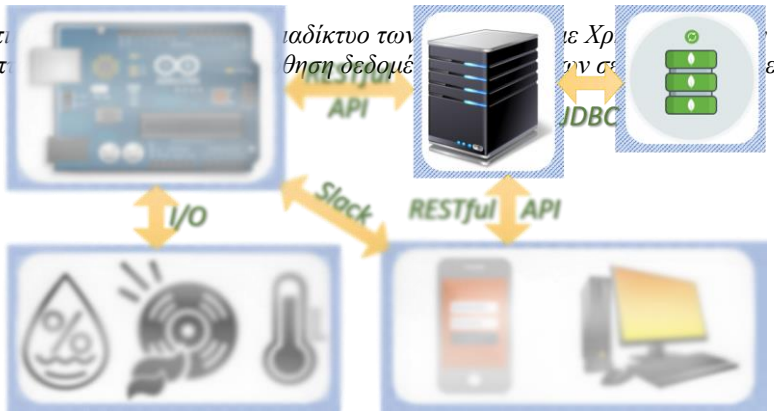
Η πρώτη εντολή της μεθόδου *"giveAllDataWith"* είναι να φτιάξει ένα καινούργιο αντικείμενο τύπου *BasicDBObject* με το όνομα *query*, το οποίο είναι οντότητα που αναγνωρίζει το JDBC της MongoDB, ενώ στη συνέχεια τίθενται σε αυτό το όνομα και η τιμή της παραμέτρου που περνάνε . Τέλος, επιστρέφει στο Επίπεδο Παρουσίασης ένα αντικείμενο τύπου *List* αντικείμενα τύπου Sensor γεμάτα με δεδομένα που πληρούν τη συνθήκη της παραμέτρου αυτής, καλώντας μία μέθοδο από την κλάση του Επιπέδου Πρόσβασης Δεδομένων.

Με τον ίδιο ακριβώς τρόπο λειτουργούν κατά βάση και οι υπόλοιπες μέθοδοι της κλάσης αυτής. Συνοπτικά, οι μέθοδοι αυτές δίνουν τη δυνατότητα δημιουργίας αντικειμένων query με μία ή δύο παραμέτρους κάθε φορά και να επιστρέφουν είτε ένα αντικείμενο τύπου *Sensor* με τα χρονικά τελευταία δεδομένα είτε *Λίστα* τέτοιων αντικειμένων με όλα δεδομένα που πληρούν τις συνθήκες, όπως και στο παραπάνω παράδειγμα. Εξαίρεση αποτελούν δύο από αυτές τις μεθόδους, οι οποίες είναι φτιαγμένες έτσι ώστε να παίρνουν ως παράμετρο το κομμάτι εκείνο του URI που ακολουθεί το ερωτηματικό (?) και σχηματίζει ερώτημα μέσω αυτού. Αναλυτικά, όλες οι μέθοδοι υπάρχουν με σχόλια στην ενότητα Ε του παραρτήματος.

4.6 Επίπεδο Πρόσβασης Δεδομένων

Το επίπεδο πρόσβασης δεδομένων είναι το πιο απλό και μικρό σε γραμμές κώδικα και αποτελείται από μία μόνο κλάση, τη MongoDBconnection, η οποία είναι υπεύθυνη για την άμεση επικοινωνία με τη MongoDB υπηρεσία ως βάση δεδομένων. Περιέχει μόνο τέσσερις μεθόδους, εκ των οποίων η μία είναι βοηθητική και *private*, δηλαδή χρησιμοποιείται μόνο από τις υπόλοιπες μεθόδους της ίδιας κλάσης και δεν καλείται εξωτερικά από το αντικείμενό της, ενώ από τις υπόλοιπες τρεις, οι δύο χρησιμοποιούνται για αποστολή ερωτημάτων και η τελευταία για αποστολή δεδομένων στη βάση δεδομένων έχοντας ένα αντικείμενο τύπου *Sensor*.

Επι… …αδίκτυο των… …με Χρ… …Τύπου REST
Περίπ… …θηση δεδομέ… …ων σε… …εφοϋπολογισμού



Η πρώτη μέθοδος, της οποίας ο κώδικας φαίνεται στις γραμμές 40 – 73 του Παραρτήματος ΣΤ, ονομάζεται “fetchByQuery”. Ουσιαστικά είναι η μέθοδος, η οποία στέλνει τα ερωτήματα (queries) στην βάση δεδομένων, ανοίγοντας μία ενεργή σύνδεση και, αφού λάβει τα οποιαδήποτε αποτελέσματα στείλει η MongoDB, κλείνει τη σύνδεση και παραμένει σε ετοιμότητα για το επόμενο αίτημα.

Η δεύτερη μέθοδος, η οποία ονομάζεται *“postJson”*, όπως υποδηλώνει και το όνομά της, χρησιμοποιείται στις περιπτώσεις στις οποίες υπάρχουν δεδομένα που πρέπει να ανέβουν στη βάση δεδομένων. Αρχικά προσπαθεί να καταχωρήσει τη μορφή JSON του αντικειμένου τύπου *Sensor* σε ένα αντικείμενο τύπου *Document*, το οποίο είναι συμβατό με το JDBC και στη συνέχεια επιχειρεί να ανοίξει μία σύνδεση ώστε να ανεβάσει αυτό το αντικείμενο και αφού ολοκληρωθεί η μεταφορά κλείνει τη σύνδεση.

Λόγω της συμβατότητας και καλής «συνεργασίας» του Jersey πλαισίου με κλάσεις που παράγουν POJOs, τα δεδομένα που περιέχονται στο σώμα ενός αιτήματος HTTP POST, μπορούν να καταχωρηθούν σε ένα *Sensor POJO* χωρίς την κλήση πολλών μεθόδων. Αρκεί απλώς να τοποθετήσουμε ένα αντικείμενο τύπου *Sensor* ως παράμετρο εισόδου στην αντίστοιχη μέθοδο που διαχειρίζεται τον πόρο, στο επίπεδο παρουσίασης και οι καταχωρήσεις γίνονται αυτόματα.

Τέλος, οι δύο τελευταίες μέθοδοι είναι σε μεγάλο βαθμό ίδιες, καθώς είναι εκείνες οι μέθοδοι που καλεί το αντικείμενο της κλάσης *SensorService*, όταν πρέπει να εκτελέσει μία αναζήτηση. Και οι δύο αυτές μέθοδοι έχουν ως είσοδο το αντικείμενο τύπου *BasicDBObject* που αναφέρθηκε νωρίτερα και περιέχει τις παραμέτρους φιλτραρίσματος των δεδομένων, το όνομα της *Βάσης Δεδομένων* και της *Συλλογής* της MongoDB, στα οποία γίνεται η αναζήτηση.

Η μόνη και κύρια διαφορά τους είναι ότι η μία, που ονομάζεται *“allData”* καταχωρεί όλα τα δεδομένα που λαμβάνει σε μία λίστα από αντικείμενα τύπου *Sensor* και επιστρέφει μία λίστα με όλα τα δεδομένα. Από την άλλη, η δεύτερη που ονομάζεται *“lastData”*, ταξινομεί με χρονολογική σειρά όλα τα αποτελέσματα που θα πάρει, και βασιζόμενη στην ημερομηνία και την ώρα που αποθηκεύτηκαν διαλέγει τα πιο πρόσφατα και τα επιστρέφει σε ένα αντικείμενο Sensor.

Συνοψίζοντας, αυτές οι τέσσερις μέθοδοι που φτιάχτηκαν με τη βοήθεια του JDBC της MongoDB, οι οποίες είναι τόσο για ανέβασμα δεδομένων όσο και για λήψη αυτών, είναι υπεραρκετές ώστε να υπάρχει μία πολύ καλή συνδεσιμότητα με τη βάση δεδομένων.

4.7 Προσομοίωση Χρηστών με Χρήση Node – RED

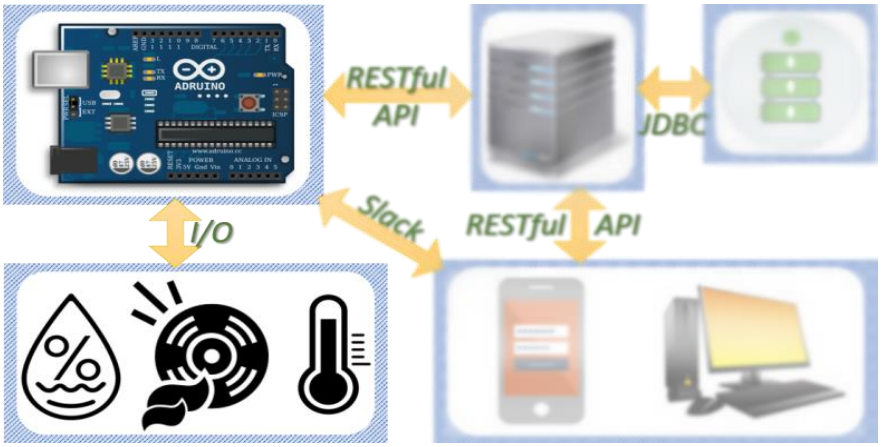
Το Node – RED είναι ένα από τα εργαλεία που περιέχονται μέσα στο πακέτο node.js, το οποίο είναι μία πλατφόρμα ανάπτυξης λογισμικού σε Javascript. Το συγκεκριμένο εργαλείο ενδείκνυται για σχεδιασμό συστημάτων ΔτΠ και οι εφαρμογές που φτιάχνονται μέσω αυτού

μπορούν να φορτωθούν σε μικροεπεξεργαστές, όπως οι Arduino, έχοντας άμεσο έλεγχο στα δεδομένα των αισθητήρων.

Στην συγκεκριμένη εργασία, θα χρησιμοποιήσουμε το Node – RED προκειμένου να προσομοιώσουμε έναν μικροελεγκτή τύπου Arduino εξοπλισμένο με WiFi, συνδεδεμένο με τρεις αισθητήρες, ο οποίος στέλνει τα δεδομένα με χρήση πρωτοκόλλου HTTP στον πόρο τύπου POST του RESTful API.

Επιπλέον, χρησιμοποιώντας την πλατφόρμα ως χρήστη προς το RESTful API, δημιουργούμε διάφορες ροές, οι οποίες αλληλοεπιδρούν με το API και αντιδρούν αναλόγως τα δεδομένα που λαμβάνουν.

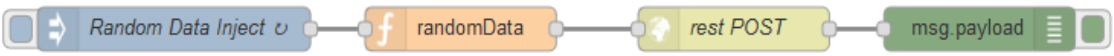
4.7.1 Προσομοίωση Αισθητήρων και Μικροελεγκτή



Εικόνα 21: Τα Εξαρτήματα τα οποία Προσομοιώνουμε μέσω του Node-RED

Η προσομοίωση των αισθητήρων πάνω σε έναν μικροελεγκτή είναι σχετικά εύκολη, καθώς το μόνο που χρειάζεται είναι η κατάλληλη σύνθεση JSON δεδομένων και η αποστολή τους με αιτήματα HTTP. Στην περίπτωση του Node – RED αρκεί μία μόνο ροή, όπως αυτή που φαίνεται στην Εικόνα 22.

Στην συγκεκριμένη ροή υπάρχουν τέσσερις κόμβοι. Ο πρώτος είναι κόμβος εισόδου και υπάρχει απλώς για να μπορεί να δίνει ένα σήμα εκκίνησης στη ροή ώστε να εκτελείται μεταξύ συγκεκριμένων διαστημάτων, για παράδειγμα ανά μία ώρα, όπως ακριβώς θα ήταν στην περίπτωση του μικροελεγκτή, ο οποίος ανά μία ώρα θα έπαιρνε μετρήσεις μέσω αισθητήρων και θα τις έστελνε.



Εικόνα 22: Node - RED Ροή που προσομοιώνει ένα αίτημα HTTP POST από έναν μικροελεγκτή

Ο δεύτερος κόμβος είναι κόμβος τύπου *function*, και χρησιμοποιείται για την προσθήκη κώδικα Javascript. Στην συγκεκριμένη περίπτωση ο κώδικας που χρησιμοποιήθηκε είναι ο παρακάτω και είναι αυτός που προσομοιώνει την παραγωγή δεδομένων από τους αισθητήρες και υπάρχει στην ενότητα Z του παραρτήματος.

Ουσιαστικά, αυτό που κάνει ο κώδικας του κόμβου “randomData” είναι να ορίζει τις τιμές των αναγνωριστικών και των τοποθεσιών μέσα σε διανύσματα κατ’ αντιστοιχία και ύστερα επιλέγονται τυχαία όλες οι ποσότητες, τόσο μέσα από τα διανύσματα, όσο και για τις αριθμητικές

τιμές της θερμοκρασίας και της υγρασίας. Η παραγωγή της εκάστοτε ημερομηνίας και ώρας, υπόκειται σε μία μικρή μετατροπή τύπου String, προκειμένου να είναι συμβατή με την Java, καθώς ειδικά στο θέμα των χρονολογιών υπάρχουν πολλές φορές δυσκολίες λόγω των διαφορετικών μορφών που μπορεί μία ημερομηνία να πάρει.

Τέλος αξίζει να αναφερθεί ότι για την μεταβλητή ύπαρξης καπνού, προκειμένου να προσομοιωθεί μία σχεδόν ρεαλιστική κατάσταση, έχει τοποθετηθεί βάρος στην τυχαία επιλογή, το οποίο προσομοιώνει την πιθανότητα ύπαρξης καπνού ανά μέτρηση στο δέκα τοις εκατό (10%).

Όλες οι παραπάνω διαδικασίες έχουν ως αποτέλεσμα τη σύνθεση ενός Javascript αντικειμένου με το όνομα *msg* και τις μεταβλητές του προσομοιωμένου αισθητήρα, μέσα στο στοιχείο *payload*. Όλες αυτές οι μεταβλητές θα αποτελέσουν ένα JSON αντικείμενο, το οποίο θα αποσταλεί με ένα αίτημα HTTP POST. Επιπλέον, με τον συγκεκριμένο κώδικα, έχουμε προσθέσει και ένα ακόμη στοιχείο στο αντικείμενο *msg*, με το όνομα *headers* και εκεί έχουμε θέση σε μία μεταβλητή με όνομα *Content-type* την τιμή *"application/json"*. Όπως είχε προαναφερθεί και στην ενότητα 2.2 (Η Σημασία του HTTP πρωτοκόλλου), αυτό είναι το όνομα μίας κεφαλίδας για ένα μήνυμα HTTP, ενώ στον επόμενο κόμβο θα την περάσουμε προκειμένου να τεθεί η αντίστοιχη κεφαλίδα στον HTTP κόμβο.

Ο HTTP κόμβος που ακολουθεί ακριβώς μετά προσφέρει τη δυνατότητα επιλογής ανάμεσα στις τέσσερις HTTP μεθόδους. Οι ρυθμίσεις που προσφέρει για όλες τις μεθόδους είναι ίδιες και πιο συγκεκριμένα είναι το URL, στο οποίο θέλουμε να στείλουμε το αίτημα και τι είδους απάντηση περιμένουμε από αυτό (JSON, UTF-8 String, ή binary buffer). Όταν είναι ρυθμισμένος ο κόμβος σε αίτημα τύπου POST, τότε τα δεδομένα που στέλνει και αποτελούν το σώμα του αιτήματος, είναι το περιεχόμενο του *msg.payload* όπως αυτό έρχεται από τον προηγούμενο κόμβο.

Επιπλέον, ανεξαρτήτως της μεθόδου HTTP, αν στο αντικείμενο msg υπάρχει στοιχείο με όνομα *headers*, επίσης εισάγονται οι αντίστοιχες κεφαλίδες αυτόματα. Επομένως, το περιεχόμενο του HTTP μηνύματος φτιάχνεται ουσιαστικά από τον προηγούμενο κόμβο με τον HTTP να είναι υπεύθυνος μόνο για τη σύνδεση και την αποστολή του μηνύματος.

Ο τελευταίος κόμβος της ροής είναι ένας κόμβος εξόδου και ονομάζεται *debug*. Ο κόμβος αυτός έχει σκοπό να δείξει τυχόν αποτελέσματα ή δεδομένα στην κονσόλα εξόδου της πλατφόρμας Node – RED. Στην συγκεκριμένη περίπτωση λαμβάνει τις αποκρίσεις HTTP από το RESTful API και τις εμφανίζει στην κονσόλα.

Με την ροή αυτήν καταφέρνουμε να προσομοιώσουμε πλήρως έναν μικροελεγκτή, συνδεδεμένο με αισθητήρες.

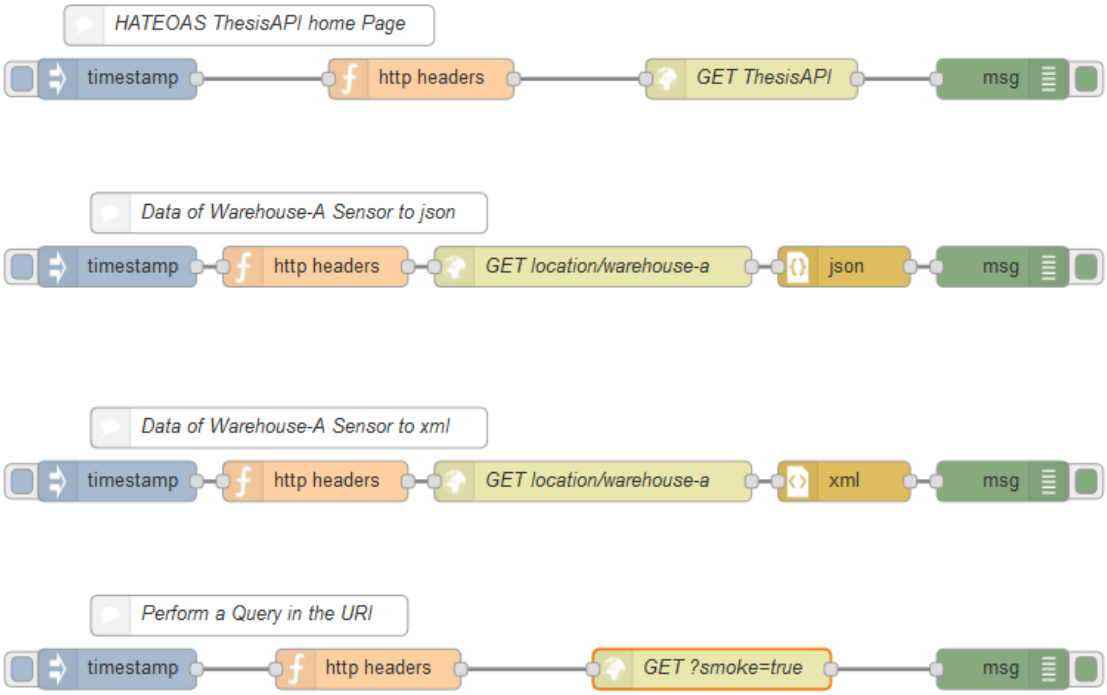
4.7.2 Προσομοίωση Χρήστη

Σε αυτήν την υποενότητα, θα δούμε μέσω παραδειγμάτων, τα περιεχόμενα κάποιων πόρων, από την πλευρά των χρηστών μέσω του RESTful API. Όπως και προηγουμένως θα κάνουμε χρήση της πλατφόρμας του Node-RED, ενώ θα το συνδυάσουμε και με το *Slack* ώστε να είναι δυνατή η αλληλοεπίδραση με το API με την απλή αποστολή συγκεκριμένων μηνυμάτων σε ένα κανάλι του.

Αρχικά, έχουν φτιαχτεί μερικές ροές προκειμένου να φανεί η ανταλλαγή των HTTP μηνυμάτων μεταξύ του Node-RED και του API, όπου η καθεμία ζητά κάποια δεδομένα αποστέλλοντας αιτήματα σε κάποιους πόρους του τελευταίου.

Επίσης με αυτές τις ροές θα δείξουμε πόσο εύκολο είναι να απαιτήσουμε από το API, τα δεδομένα να είτε σε JSON είτε σε XML μορφή. Στην Εικόνα 23 φαίνονται τέσσερις τέτοιες ροές.

Οι ροές της Εικόνας 23 εκτελούν απλά αιτήματα HTTP στο RESTful API και λαμβάνουν τα αντίστοιχα δεδομένα. Ξεκινώντας από πάνω, η πρώτη ροή που φαίνεται στέλνει ένα αίτημα προκειμένου να λάβει τα δεδομένα HATEOAS του αρχικού πόρου – ρίζας του API, τα οποία περιέχουν πληροφορίες για όλους τους υπόλοιπους πόρους που υπάρχουν ως συνέχεια του URI αυτού.



Εικόνα 23: Απλές Node-RED Ροές για Επικοινωνία μέσω των Πόρων

Πιο συγκεκριμένα, η ροή ξεκινά με έναν κόμβο εισόδου, ο οποίος δεν προσφέρει τίποτα πέρα από το σήμα εκκίνησης της ροής. Ακολουθεί ένας κόμβος εισαγωγής κώδικα Javascript, ο οποίος προσθέτει την κεφαλίδα *"Content-Type : application/json"* στο αίτημα HTTP. Ακολουθεί ο κόμβος HTTP και έχει ρυθμιστεί στην μέθοδο GET και σε διεύθυνση *"ThesisAPI/"*, ενώ τέλος υπάρχει ένας κόμβος εξόδου που παράγει το περιεχόμενο της απόκρισης του προηγούμενου στην κονσόλα.

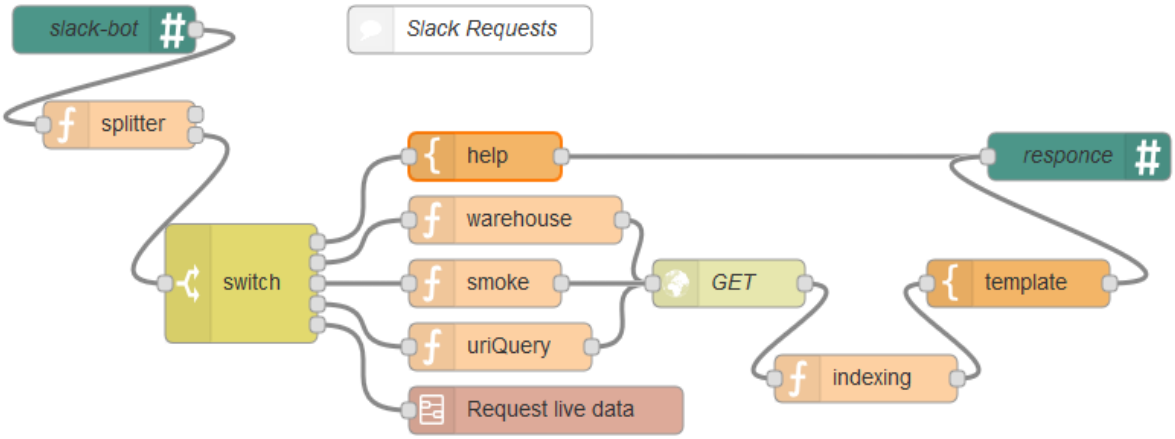
Συνεχίζοντας, η δεύτερη ροή στέλνει ένα αίτημα HTTP, ζητώντας τα τελευταία δεδομένα του μικροελεγκτή υπευθύνου για την τοποθεσία *Warehouse-A*. Οι κόμβοι που χρησιμοποιούνται είναι ακριβώς ίδιοι με της προηγούμενης ροής, με τη μόνη διαφορά ότι έχουμε ρυθμίσει τον HTTP κόμβο να δεχτεί απόκριση τύπου *UTF-8*, και ακριβώς μετά έχουμε προσθέσει έναν κόμβο *"json"*. Αυτός ο τρόπος διασφαλίζει το «ξεπακετάρισμα» των δεδομένων της απόκρισης HTTP σε ένα αντικείμενο JSON, σε αντίθεση με τον προηγούμενο που μπορεί να φέρει τυχόν σφάλματα, λόγω διαφορετικής απόκρισης από το API.

Με αντίστοιχο τρόπο λειτουργεί και η τρίτη ροή της ίδιας Εικόνας, η οποία ζητά τα ίδια ακριβώς δεδομένα, όμως σε αυτήν την περίπτωση, στον κόμβο εισαγωγής κώδικα Javascript, η τιμή της κεφαλίδας *"Content-Type"* τίθεται ως *"application/xml"* και όχι όπως στις μέχρι τώρα ροές. Επιπλέον ο κόμβος *json* έχει αντικατασταθεί από έναν κόμβο *xml*, έχοντας ως αποτέλεσμα το

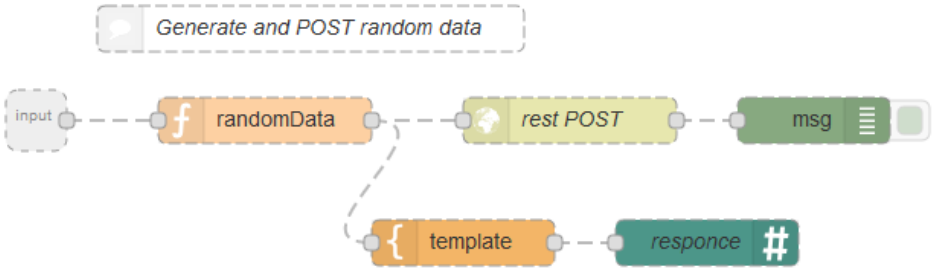
«ξεπακετάρισμα» της απόκρισης HTTP σε μορφή XML, όπως θα δούμε στην ενότητα της ανάλυσης των αποτελεσμάτων που ακολουθεί.

Τέλος, η τελευταία ροή στέλνει ένα αίτημα HTTP με URI *"ThesisAPI/sensors/uriQuery/?smoke=true"*, το οποίο όπως φαίνεται , περιέχει και μία παράμετρο φιλτραρίσματος όλων των δεδομένων. Με αυτό το αίτημα, γίνεται χρήση του πόρου με URI "*ThesisAPI/sensors/uriQuery/"*, ενώ ταυτόχρονα η παράμετρος φιλτραρίσματος είναι *"smoke=true"*, η οποία προφανώς σημαίνει πως θέλουμε να μας επιστραφούν τα τελευταία δεδομένα, οποιασδήποτε αποθήκης όπου εντοπίστηκε καπνός.

Όλες οι παραπάνω όμως, δεν είναι παρά πολύ απλές ροές αποστολής αιτημάτων HTTP, ενώ το μόνο που προσφέρουν είναι δεδομένα μέσω του RESTful API. Προκειμένου όμως, να φανεί η χρησιμότητα του API αυτού στο διαδίκτυο των πραγμάτων, φτιάχτηκε και μία ακόμη ροή ελαφρώς πιο πολύπλοκη, η οποία συνδέεται με ένα κανάλι ενός εργασιακού χώρου του Slack και φαίνεται στις Εικόνες 24 - 25. Από εκεί μπορεί οποιοδήποτε μέλος του καναλιού να αλληλοεπιδράσει με το API.



Εικόνα 25: Ροή Node-RED για την Αλληλεπίδραση με το RESTful API Μέσω Slack



Εικόνα 24: Υπορροή Node-RED με όνομα Request live data

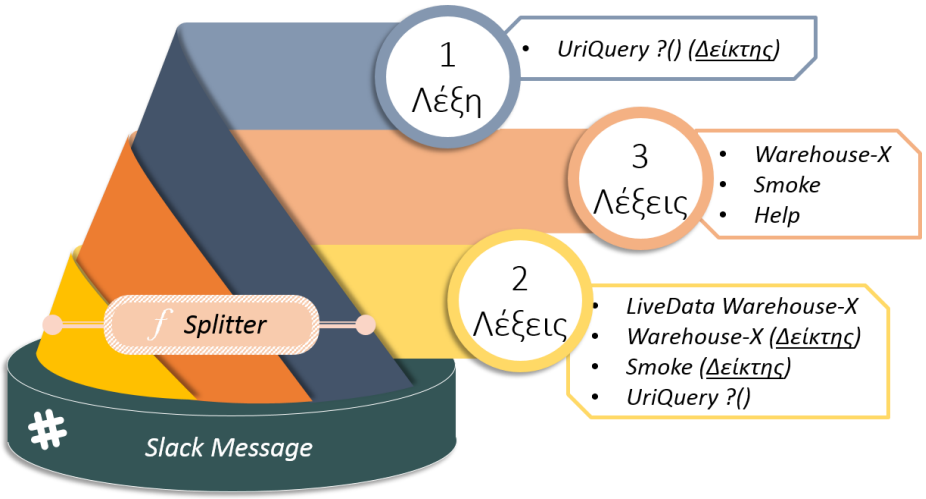
Μέσω αυτού του καναλιού, μπορεί κανείς να ζητήσει οποιαδήποτε δεδομένα, όπως για παράδειγμα μία συγκεκριμένη καταχώρηση από συγκεκριμένη αποθήκη ή μπορεί ακόμα και να ζητήσει να γίνει μία επί τόπου μέτρηση από τους αισθητήρες μίας αποθήκης χρησιμοποιώντας συγκεκριμένες λέξεις και φράσεις. Αυτά μπορούν να επιτευχθούν με την ροή και την υπορροή των παρακάτω εικόνων.

Όπως είναι προφανές, αυτή η ροή είναι αρκετά πιο περίπλοκη από τις προηγούμενες που αναφέρθηκαν, ωστόσο αυτή αποτελεί ολόκληρη την επικοινωνία μέσω του Slack. Η επικοινωνία αυτή βασίζεται στην παρακολούθηση ενός Slack καναλιού για συγκεκριμένες λέξεις – φράσεις κλειδιά από ένα Slack-bot και την εκκίνηση συγκεκριμένων διεργασιών αντιστοίχως. Στην Εικόνα 25 φαίνονται οι λέξεις – κλειδιά και ποιο είναι το αποτέλεσμα αποστολής τους στο κανάλι του Slack.

Τα *bots* είναι διαδικτυακά ρομπότ και πρόκειται για ένα είδος προγραμμάτων ικανά να εκτελέσουν αυτοματοποιημένες διεργασίες μέσω διαδικτύου. Για την λειτουργία ολόκληρης της ροής, έχουν φτιαχτεί δύο *bots*, εκ των οποίων το ένα χρησιμοποιείται για να «ακούει» τη συνομιλία και να λαμβάνει τα μηνύματα για ανάλυση στη ροή, και το δεύτερο χρησιμοποιείται για να στέλνει μηνύματα και απαντήσεις στο κανάλι αναλόγως.

Στη συνέχεια, αν ανιχνευτεί κάποια από αυτές τις λέξεις – φράσεις, τότε αυτή αναλύεται αναλόγως και από εκεί και έπειτα υπάρχουν τρία ενδεχόμενα: είτε η άμεση απάντηση στο κανάλι του Slack με κάποιες πληροφορίες σχετικά με το πώς λειτουργεί αυτό το σύστημα, είτε η αποστολή HTTP αιτημάτων στο RESTful API και η απάντηση των αντίστοιχων δεδομένων στο κανάλι, είτε η κλήση μίας υποροής και μέσω αυτής, προσομοίωση μέτρησης κατ' απαίτηση από τους αισθητήρες συγκεκριμένης αποθήκης.

Παρατηρώντας τα ενδεχόμενα που προκύπτουν από τον αριθμό των λέξεων κλειδιών που φαίνονται στην Εικόνα 26, συμπεραίνουμε τρεις πιθανότητες. Ανάλογα με αυτές, διαμορφώνεται το περιεχόμενο τριών μεταβλητών (*msg.term0, msg.term1, msg.term2*), οι οποίες έχουν καθοριστικό ρόλο στην εξέλιξη της ροής, αφού οι επόμενοι κόμβοι βασίζονται σε αυτές για την λειτουργία τους.



Εικόνα 26: Πιθανά Ενδεχόμενα Λέξεων Κλειδιών βάσει του Πλήθους τους

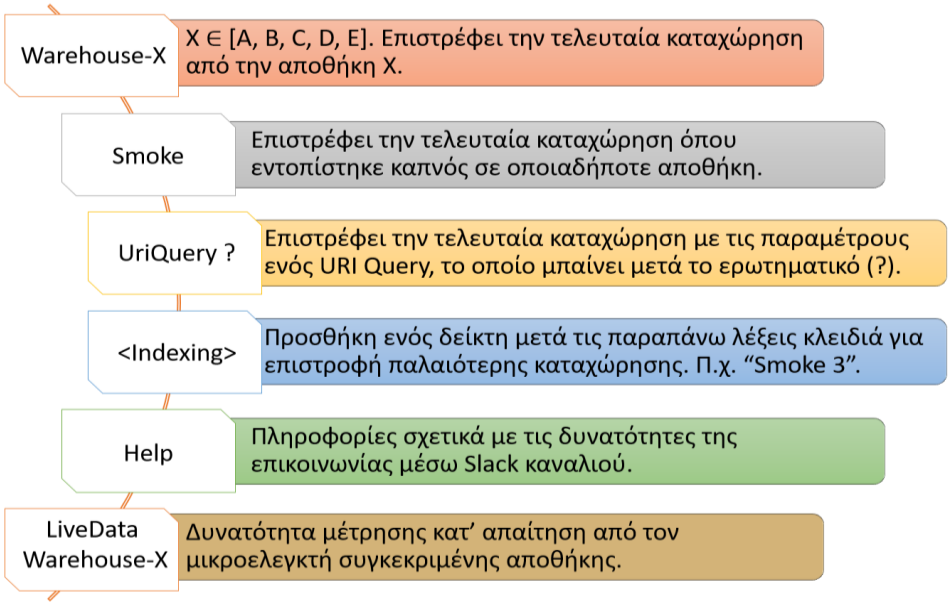
Η πρώτη μεταβλητή (*msg.term0*) κρατάει πάντα την πρώτη λέξη – κλειδί, η οποία είναι χαρακτηριστική και σχεδόν πάντα οδηγεί στην κλήση ενός HTTP αιτήματος. Η δεύτερη μεταβλητή (*msg.term1*) περιέχει τον δείκτη της καταχώρησης του MongoDB που θέλουμε, ξεκινώντας από τη νεότερη, εκτός των περιπτώσεων όπου το περιεχόμενο της πρώτης μεταβλητής είναι είτε *"uriquery"* είτε *"livedata"*. Σε αυτές τις περιπτώσεις παίρνει την τιμή του στοιχείου που ακολουθεί της πρώτης λέξης – κλειδί. Τέλος, η τρίτη μεταβλητή (*msg.term2*) χρησιμοποιείται μόνο στην

περίπτωση του URI ερωτήματος, όπου παίρνει την τιμή του δείκτη. Πιο συγκεκριμένα, όπως υποδεικνύει και η Εικόνα 26, αν το διάνυσμα περιέχει:

- ***Μία μόνο λέξη***: Είτε πρόκειται για τη λέξη *"Help"*, είτε για τις λέξεις *"Warehouse-X"* ή *"Smoke"*, χωρίς όμως να ακολουθούνται από αριθμό δείκτη. Τότε οι τρεις μεταβλητές που αναφέρθηκαν θα πάρουν τις εξής τιμές *msg.term0 = warehouse-X/smoke, msg.term1 = 0, msg.term2 = null*.
- ***Δύο λέξεις:*** Είτε πρόκειται για τις λέξεις *"Warehouse-X"* ή *"Smoke"*, οι οποίες όμως ακολουθούνται από δείκτη, είτε για URI ερώτημα *"UriQuery ?"* χωρίς δείκτη, είτε για μέτρηση κατ' απαίτηση *"LiveData Warehouse-X"*. Στην κάθε περίπτωση οι τρεις μεταβλητές παίρνουν περιεχόμενο αντίστοιχα ως εξής: *msg.term0 = warehouse-X/smoke, msg.term1 = (δείκτης), msg.term2 = null, ή msg.term0 = uriquery, msg.term1 = ?(...), msg.term2 = 0, ή msg.term0 = livedata, msg.term1 = warehouse-X, msg.term2 = null*.
- ***Τρεις λέξεις***: Προκύπτει μόνο ένα ενδεχόμενο, αυτό του URI ερωτήματος με δείκτη, στην οποία περίπτωση οι μεταβλητές διαμορφώνονται με τον εξής τρόπο: *msg.term0 = uriquery, msg.term1 = ?(...), msg.term2 = (δείκτης)*.

Προφανώς, αν το μήνυμα περιέχει περισσότερες από τρεις λέξεις δεν είναι κάτι που αφορά την ροή του Node-RED. Έτσι, Ξεκινώντας από πάνω αριστερά στην Εικόνα 24, ο πρώτος κόμβος που βλέπουμε έχει το όνομα *slack-bot* και είναι ο κόμβος που «ακούει» για μηνύματα στο κανάλι του Slack.

Μόλις σταλεί ένα μήνυμα, το *slack-bot* το τοποθετεί μέσα στην μεταβλητή *msg.payload* και από εκεί περνάει στον επόμενο κόμβο, ο οποίος είναι κόμβος εισαγωγής κώδικα Javascript, με το όνομα *splitter*. Όπως υποδεικνύει και το όνομά του, ο κόμβος αυτός αποσκοπεί στο να χωρίσει το περιεχόμενο της μεταβλητής αυτής βάσει του κώδικα που υπάρχει στην ενότητα Z του παραρτήματος, σε λέξεις μεμονωμένες, προκειμένου να μπορέσει το μήνυμα τα εξεταστεί καλύτερα.



Εικόνα 27: Λέξεις - Φράσεις Κλειδιά για Επικοινωνία με το RESTful API μέσω Slack

Αυτό επιτυγχάνεται με το να χωρίζει το περιεχόμενο της μεταβλητής msg.payload σε κομμάτια μετά από κάθε κενό χαρακτήρα, αφού αφαιρεθεί τυχόν κενό από το τέλος του μηνύματος, και εισάγονται σε ένα διάνυσμα. Υποθέτοντας ότι τα κομμάτια που έχουν χωριστεί είναι λέξεις στη

σωστή τους μορφή, τότε ανάλογα το μέγεθος του διανύσματος μπορούμε να καταλάβουμε σε πρώτο επίπεδο αν ο αποστολέας του μηνύματος ζητά επικοινωνία με το API.

Συνεχίζοντας, οι υπόλοιπες γραμμές του κώδικα διαμορφώνουν το περιεχόμενο των μεταβλητών *msg.term0, msg.term1, msg.term2* και υλοποιούν τη λογική της Εικόνας 27.

Ο επόμενος κόμβος που ακολουθεί, είναι ένας "*Switch"* κόμβος. Αφού πλέον έχουν τεθεί οι τρεις αυτές μεταβλητές, μπορούμε ανάλογα το περιεχόμενο της πρώτης να καθορίσουμε το URI στο οποίο θα πραγματοποιηθεί το αίτημα HTTP. Έτσι:

- Αν **msg.term0 = "help"**: Η έξοδος αυτή δίνει σήμα σε έναν κόμβο τύπου *template*, το οποίο περιέχει σε κείμενο όλες τις απαραίτητες πληροφορίες για την επικοινωνία με το API μέσω του Slack καναλιού. Στη συνέχεια το κείμενο περνάει στον κόμβο εξόδου με το όνομα *response,* ο οποίος είναι το δεύτερο slack-bot που χρησιμοποιείται για να στέλνει μηνύματα στο κανάλι.
- Αν ***msg.term0 = "livedata"***: Το περιεχόμενο του μηνύματος msg περνάει στην υπορροή της Εικόνας 25. Η υπορροή αυτή προσομοιώνει την μέτρηση κατ' απαίτηση από τον μικροελεγκτή συγκεκριμένης βιβλιοθήκης, συντάσσοντας το κατάλληλο URI βάσει της μεταβλητής *msg.term1* και παράγει δεδομένα όπως η ροή προσομοίωσης μικροελεγκτή.
- Αν ***msg.term0 = "warehouse"***: Το μήνυμα μεταβαίνει σε έναν κόμβο κώδικα Javascript, όπου εντοπίζει για ποια αποθήκη ζητούνται πληροφορίες. Αναλόγως φτιάχνει ένα ζεύγος κλειδιού – τιμής, το οποίο έχει όνομα "*path*" και παίρνει τιμή τον URI πόρο που αντιστοιχεί στον πόρο της αποθήκης αυτής. Επιπλέον, προσθέτει ένα αντικείμενο JSON με όνομα *"headers",* μέσα στο οποίο τοποθετεί την κεφαλίδα για το αίτημα HTTP που θα ακολουθήσει. Στη συνέχεια, τα αποστέλλει στο RESTful API για αποθήκευση, ενώ ταυτόχρονα τα δημοσιεύει και στο slack κανάλι.
- Αν ***msg.term0 = "smoke"***: Το μήνυμα μεταβαίνει σε κόμβο κώδικα Javascript όπου θέτει τον αντίστοιχο URI πόρο στο ίδιο ζεύγος κλειδιού – τιμής *"path"* και προσθέτει το αντικείμενο JSON με όνομα *"headers",* που αναφέρθηκε και προηγουμένως.
- Αν **msg.term0 = "uriquery"**: Το μήνυμα μεταβαίνει ξανά σε κόμβο κώδικα Javascript όπου θέτει το URI του πόρου στο ίδιο ζεύγος κλειδιού – τιμής *"path"* και προσθέτει το αντικείμενο JSON με όνομα *"headers",* που αναφέρθηκε και προηγουμένως. Η μόνη διαφορά με την προηγούμενη περίπτωση είναι ότι το URI που χρησιμοποιείται είναι ο πόρος που αντιστοιχεί συν το περιεχόμενο της μεταβλητής *msg.term1*, η οποία περιέχει τις παραμέτρους αναζήτησης.

Οι τρεις τελευταίες περιπτώσεις του *Switch*, μετά τους αντίστοιχους κόμβους εισαγωγής κώδικα Javascript, συνεχίζουν με τρεις ακόμα κόμβους πριν καταλήξουν στο bot δημοσίευσης στο Slack. Αμέσως μετά τους κόμβους εισαγωγής κώδικα, ακολουθεί ο κόμβος HTTP αιτημάτων, ο οποίος στέλνει τα αιτήματα στα URI που ορίζει η μεταβλητή *"msg.payload.path"* σε κάθε περίπτωση και λαμβάνει.

Ας σημειωθεί εδώ ότι η ροή ζητά όλα τα δεδομένα που πληρούν τα αντίστοιχα κριτήρια, αυτά επιστρέφουν μέσα σε ένα διάνυσμα από JSON αντικείμενα και στη συνέχεια με τη βοήθεια ενός κόμβου κώδικα Javascript παίρνουμε το αντικείμενο που υποδεικνύει η μεταβλητή *msg.term1* ή η *msg.term2* αναλόγως την περίπτωση.

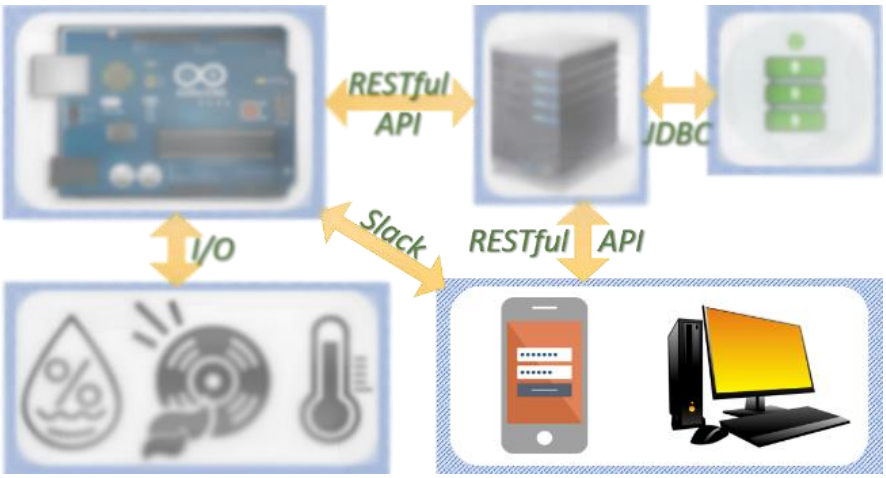
Ο κόμβος με το όνομα *"indexing"* που ακολουθεί τον HTTP κόμβο, θα πρέπει να διαλέξει την μέτρηση που αντιστοιχεί στον εκάστοτε δείκτη που επέλεξε ο αποστολέας του οποιοδήποτε αιτήματος. Επιπλέον, έχουμε ως δεδομένο ότι η μέτρηση των καταχωρήσεων μέσα σε ένα

διάνυσμα ξεκινά από το μηδέν (0), το μέγεθος του διανύσματος είναι όσο ο δείκτης της τελευταίας καταχώρησης συν ένα και ότι ο αποστολέας του αιτήματος στο slack θεωρεί την πιο πρόσφατη καταχώρηση να έχει δείκτη ένα (1).

Βάσει αυτών των δεδομένων, επιτυγχάνουμε να επιλέξουμε την επιθυμητή καταχώρηση, με την επιλογή του στοιχείου εκείνου μέσα στο διάνυσμα με δείκτη ίσο με το αποτέλεσμα της αφαίρεσης του περιεχομένου της μεταβλητής *msg.term1* (ή *msg.term2* αντίστοιχα) από το μέγεθος του ίδιου του διανύσματος. Ο κώδικας που εκτελεί τη διαδικασία αυτή βρίσκεται στην ενότητα Z του παραρτήματος.

Τέλος, ακολουθεί ένας κόμβος τύπου *Template*, ο οποίος μας βοηθά στην σύνθεση ενός κειμένου εύκολο σε ανάγνωση που περιέχει τα δεδομένα που ζητήθηκαν και προέκυψαν, όπως ακριβώς στην περίπτωση της λέξης *"Help"*, ενώ αμέσως μετά αυτό προωθείται στο δεύτερο slack-bot (κόμβος *"response"*) για να δημοσιευτεί στο κανάλι.

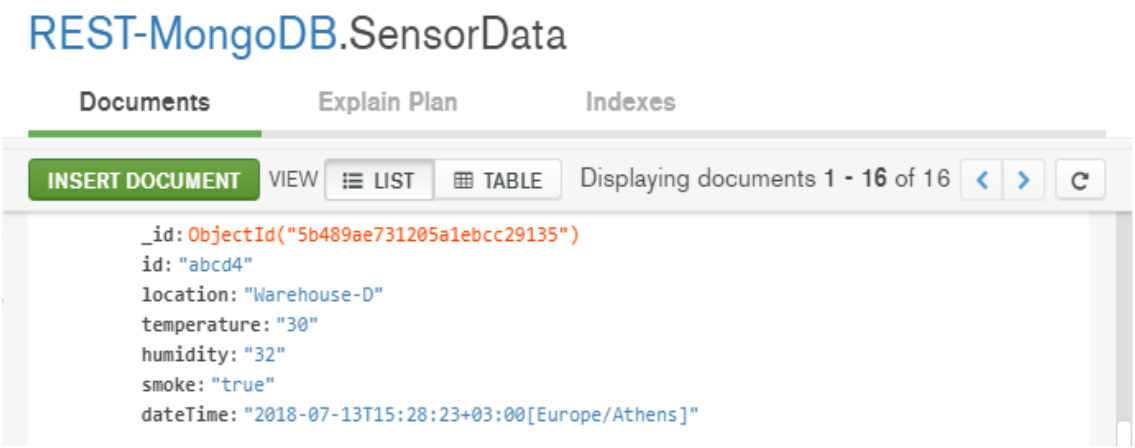
5 ΚΕΦΑΛΑΙΟ 4: Πλεονεκτήματα Υλοποίησης



Στο κεφάλαιο αυτό θα παρατεθούν και θα σχολιαστούν εικόνες και στιγμιότυπα της λειτουργίας του RESTful API καθώς και της αλληλεπίδρασής του με τις ροές του Node – RED και κατ' επέκταση το Slack, καθώς και κάποιες δοκιμές που έγιναν με τον περιηγητή διαδικτύου *Firefox*.

Ωστόσο, καθώς οι πόροι του RESTful API λειτουργούν με σχεδόν πανομοιότυπο τρόπο και φτάνουν σε αριθμό τις δεκαοχτώ (18) συνολικά, τα στιγμιότυπα τα οποία θα παρουσιαστούν στη συνέχεια είναι ενδεικτικά παραδείγματα και αναφέρονται σε κάποιες από αυτές. Πάρα ταύτα, η κλήση και στους υπόλοιπους πόρους δεν διαφέρει καθόλου σε σχέση με τα παραδείγματα που θα ακολουθήσουν.

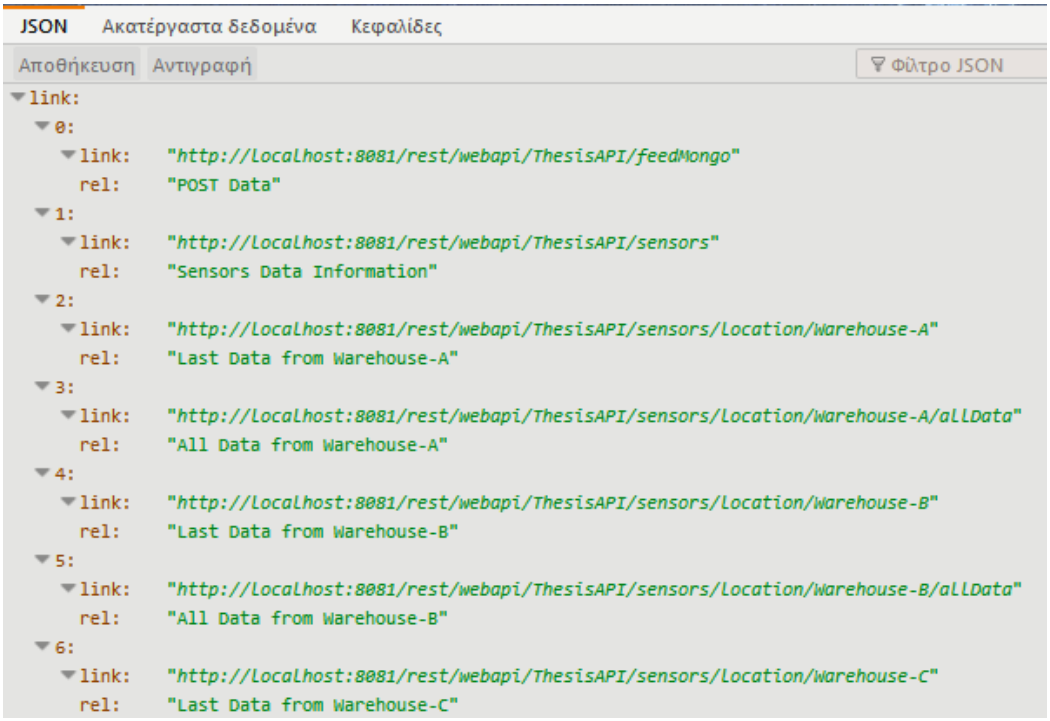
Στην Εικόνα 28 φαίνεται ένα δείγμα του περιεχόμενου της βάσης με όνομα *"REST-MongoDB"* και συλλογής με όνομα *"SensorData"* της MongoDB.



Εικόνα 28: Δείγμα Δεδομένων από τη MongoDB

5.1 Αιτήματα HTTP με τον Περιηγητή Firefox

Κατά την ανάπτυξη του RESTful API, ένας πολύ απλός τρόπος για να δοκιμάσουμε την λειτουργικότητά του ήταν η αποστολή αιτημάτων HTTP σε αυτό μέσω ενός περιηγητή διαδικτύου. Στην συγκεκριμένη περίπτωση χρησιμοποιήθηκε ένας από τους ευρεία διαδεδομένους, ο *Firefox* (έκδοση *Developer Edition/ 61.0*).



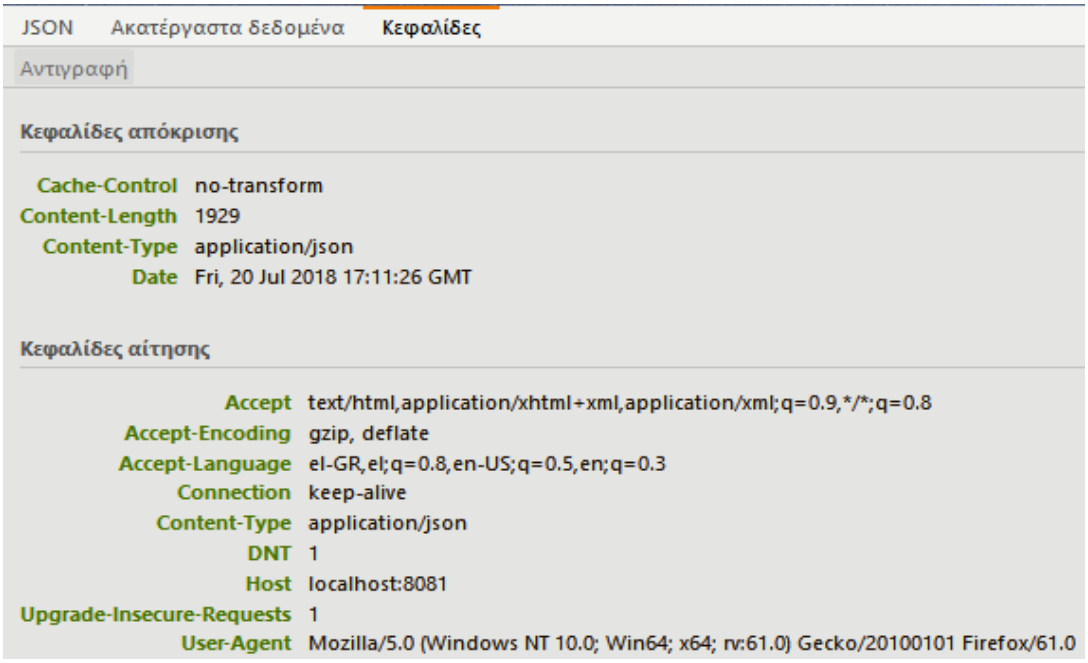
Εικόνα 29: Δείγμα από την Απόκριση σε Firefox από τον Πόρο "ThesisAPI"

Στην Εικόνα 29 βλέπουμε την απόκριση που δίνει ο περιηγητής *Firefox* όταν πραγματοποιούμε ένα αίτημα HTTP GET στον πόρο με URI *"ThesisAPI/"*. Πρόκειται για τον πόρο εκείνο που δίνει τα περισσότερα δεδομένα τύπου HATEOAS, ενώ ταυτόχρονα παρατηρούμε ότι τα

δεδομένα είναι σε μορφή JSON. Αυτό μπορούμε να το δούμε και από τις κεφαλίδες των μηνυμάτων αιτήματος και απόκρισης, τις οποίες μας δίνει τη δυνατότητα ο *Firefox* να δούμε, και φαίνονται στην Εικόνα 31.

Στην Εικόνα 30, φαίνονται όλες οι κεφαλίδες και του HTTP αιτήματος που έστειλε ο *Firefox,* καθώς επίσης και της απόκρισης από το RESTful API. Προφανώς, ο περιηγητής θέτει αυτόματα κάποιες από τις κεφαλίδες με τιμές όταν στέλνει HTTP μηνύματα, ενώ δεν μας δίνει τη δυνατότητα να αλλάξουμε ή να προσθαφαιρέσουμε κάποια από αυτές. Ωστόσο, υπάρχουν πρόσθετα για τον συγκεκριμένο περιηγητή, τα οποία μας επιτρέπουν τέτοιες αλλαγές. Έτσι, μπορούμε να προσθέσουμε την κεφαλίδα *"Content-Type"* σε όποια τιμή επιθυμούμε και όταν είναι ενεργό το πρόσθετο αυτή θα προτίθεται σε όλα τα HTTP μηνύματα που στέλνονται.

Όπως φαίνεται, στην Εικόνα 30, τόσο στο αίτημα όσο και στην απόκριση η κεφαλίδα αυτή έχει οριστεί για ανταλλαγή περιεχομένου JSON. Επίσης, αξίζει να παρατηρήσουμε και την κεφαλίδα *"Cache-Control"*, η οποία έχει τεθεί με την τιμή *"no-transform"*. Αυτό σημαίνει ότι το περιεχόμενο αυτής της σελίδας δεν πρόκειται να αλλάξει με την πάροδο του χρόνου και άρα μπορεί να αποθηκευτεί στην προσωρινή μνήμη ενός χρήστη αν σκοπεύει να χρησιμοποιήσει τα δεδομένα αυτά μεταγενέστερα.



Εικόνα 30: Κεφαλίδες Αιτήματος και Απόκρισης Μέσω του Πόρου "ThesisAPI"

Στις Εικόνες 31 και 32 φαίνονται οι αποκρίσεις του περιηγητή *Firefox* στο αίτημα για τον πόρο με URI *"ThesisAPI/sensors/location/Warehouse-A"*, ο οποίος επιστρέφει την τελευταία καταχώρηση από τις μετρήσεις στην αποθήκη Α. Η διαφορά μεταξύ των δύο εικόνων είναι ότι η μία παρουσιάζει την απόκριση όταν η τιμή της κεφαλίδας *"Content-Type"* είναι ρυθμισμένη για JSON μορφή, ενώ η δεύτερη την απόκριση όταν το περιεχόμενο της κεφαλίδας αυτής είναι για XML μορφή.

Προφανώς πρόκειται για δύο τελείως διαφορετικές μορφές παρουσίασης δεδομένων, ωστόσο αποτελούν το περιεχόμενο του ίδιου ακριβώς πόρου. Με άλλα λόγια αποτελούν δύο αναπαραστάσεις του ίδιου πόρου.


```
−<sensor>
    <dateTime>2018-07-13T21:30:37+03:00[Europe/Athens]</dateTime>
    <humidity>5</humidity>
    <id>abcd1</id>
   −<links>
    −<link>
        http://localhost:8081/rest/webapi/ThesisAPI/sensors/location/Warehouse-A/allData
      </link>
      <rel>all data</rel>
    </links>
   −<links>
    −<link>
        http://localhost:8081/rest/webapi/ThesisAPI/sensors/location/Warehouse-A/temperature
      </link>
    −<rel>
        Get data of this warehouse for various temperature values
      </rel>
    </links>
   −<links>
    −<link>
        http://localhost:8081/rest/webapi/ThesisAPI/sensors/location/Warehouse-A/smoke/true
      </link>
    −<rel>
        Get the last data when smoke was detected in this warehouse
      </rel>
    </links>
    <location>Warehouse-A</location>
    <smoke>false</smoke>
    <temperature>38</temperature>
  </sensor>
```

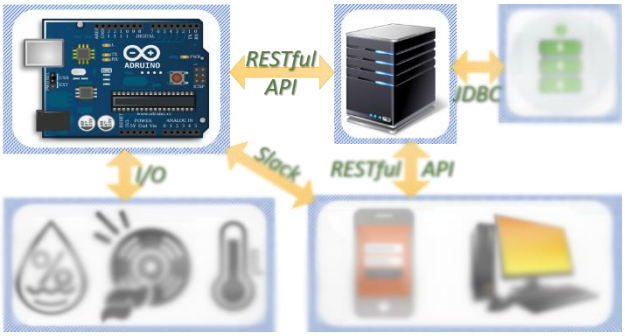

Εικόνα 31: Απόκριση Πόρου "ThesisAPi/sensors/location/Warehouse-A" με Κεφαλίδα "Content-Type : application/xml"


```
JSON   Ακατέργαστα δεδομένα   Κεφαλίδες
Αποθήκευση   Αντιγραφή
dateTime:      "2018-07-13T21:30:37+03:00[Europe/Athens]"
humidity:      5
id:            "abcd1"
▼links:
  ▼0:
    ▼link:     "http://localhost:8081/rest/webapi/ThesisAPI/sensors/location/Warehouse-A/allData"
      rel:     "all data"
  ▼1:
    ▼link:     "http://localhost:8081/rest/webapi/ThesisAPI/sensors/location/Warehouse-A/temperature"
    ▼rel:      "Get data of this warehouse for various temperature values"
  ▼2:
    ▼link:     "http://localhost:8081/rest/webapi/ThesisAPI/sensors/location/Warehouse-A/smoke/true"
    ▼rel:      "Get the last data when smoke was detected in this warehouse"
location:      "Warehouse-A"
smoke:         false
temperature:   38
```


Εικόνα 32: Απόκριση Πόρου "ThesisAPi/sensors/location/Warehouse-A" με α "Content-Type : application/json"

5.2 Αλληλεπίδραση με το Node – RED

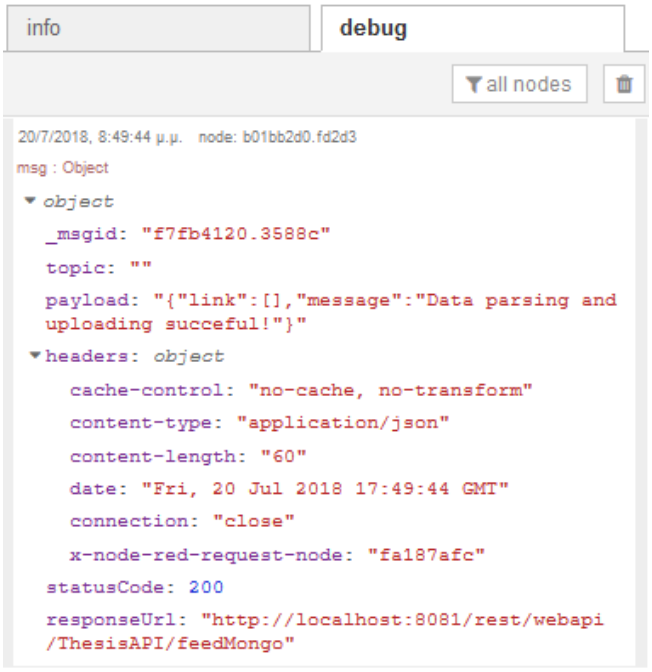
Η λειτουργία των ροών του Node – RED δεν διαφέρει και πολύ από αυτή του περιηγητή. Ξανά, πρόκειται για αιτήματα HTTP και ανταλλαγή δεδομένων μέσω αυτών, με τη μόνη διαφορά ότι σε αυτήν την περίπτωση μπορούμε να εκμεταλλευτούμε αυτά τα δεδομένα και να αλληλοεπιδράσουμε ανάλογα με το περιεχόμενό τους. Σε αυτήν την υποενότητα θα παρουσιαστούν και θα αναλυθούν εικόνες και στιγμιότυπα μετά την εκτέλεση των ροών που αναλύθηκαν στην ενότητα 3.7.



Εικόνα 33: Η Αλληλεπίδραση μεταξύ του Node-RED και του RESTful API σε όλο το Σύστημα

Έτσι, στην Εικόνα 34 που ακολουθεί, απεικονίζεται στην κονσόλα του Node – RED η απόκριση της ροής η οποία προσομοιώνει τον μικροελεγκτή με τους αισθητήρες και στέλνει τα δεδομένα στο RESTful API, μέσω HTTP POST.

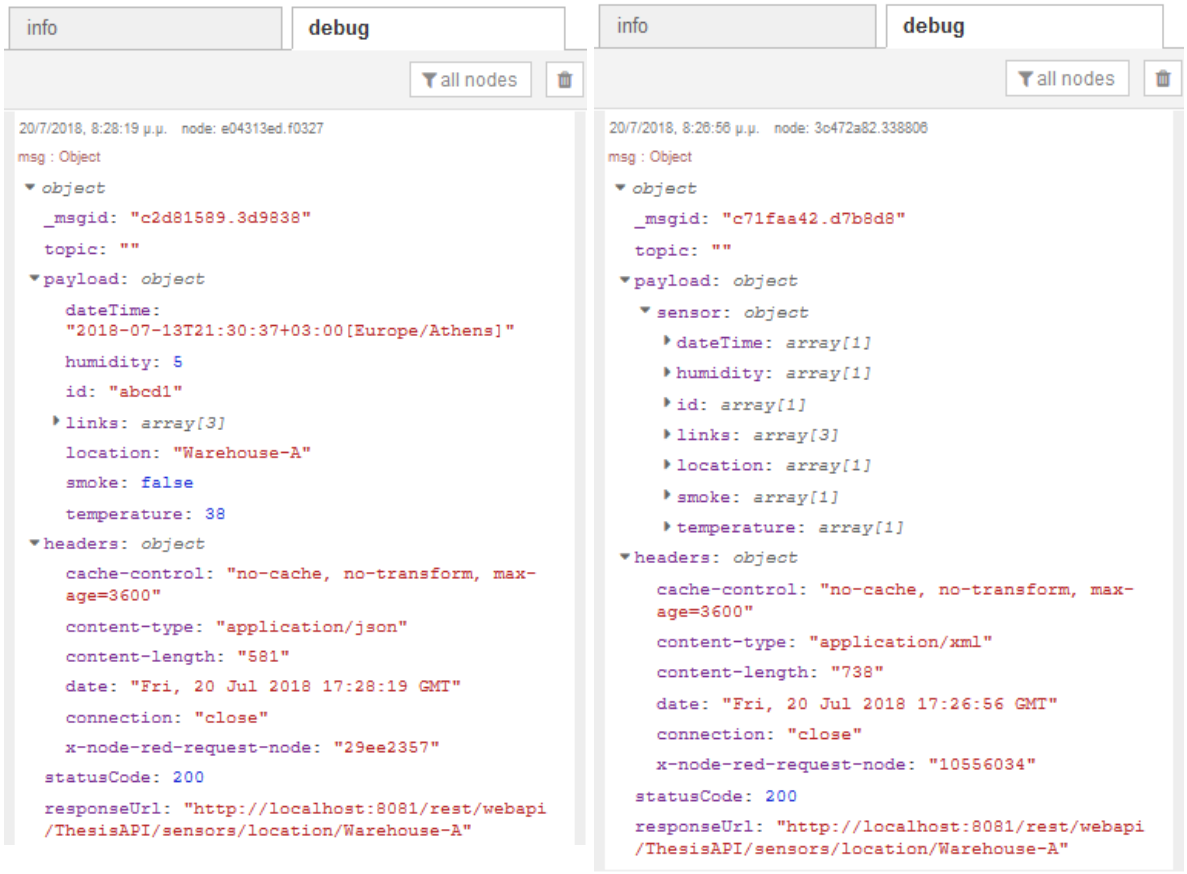
Όπως φαίνεται, πέρα από τις κεφαλίδες της απόκρισης, υπάρχει ένα μήνυμα από το API το οποίο δηλώνει πως η διαχείριση και το ανέβασμα στη βάση των δεδομένων ήταν επιτυχής. Αυτό, ωστόσο είναι μία πρόταση αναγνώσιμη από άνθρωπο και όχι από μηχανή.



Εικόνα 34: Απόκριση Node - RED Ροής για την Αποστολή Δεδομένων με HTTP POST

Για να καταλάβει το Node – RED πως το API επεξεργάστηκε και ολοκλήρωσε το αίτημά του επιτυχώς, θα αναγνωρίσει τον κωδικό κατάστασης του αιτήματος. Ωστόσο, το *"statusCode"* δεν περιέχεται στις κεφαλίδες του αιτήματος, αλλά μεταφέρεται σαν απλή μεταβλητή. Αυτό οφείλεται στο γεγονός ότι ο κωδικός κατάστασης δεν περιέχεται στις κεφαλίδες, αλλά περιέχεται στην *Γραμμή Αιτήματος*, όπως εξηγήθηκε στην ενότητα 2.2.

Συνεχίζοντας, για να είναι εφικτή και μία σύγκριση με τα αποτελέσματα χρήσης ενός περιηγητή, παρατίθενται τα στιγμιότυπα από τις αποκρίσεις στα αιτήματα HTTP GET στον πόρο με URI *"ThesisAPI/sensors/location/Warehouse-A"*, τόσο σε μορφή JSON όσο και σε μορφή XML.



Εικόνα 35: (Αριστερά) Απόκριση για "Content-Type" = application/json, (Δεξιά) Απόκριση για "Content-Type" = application/xml στο Node - RED

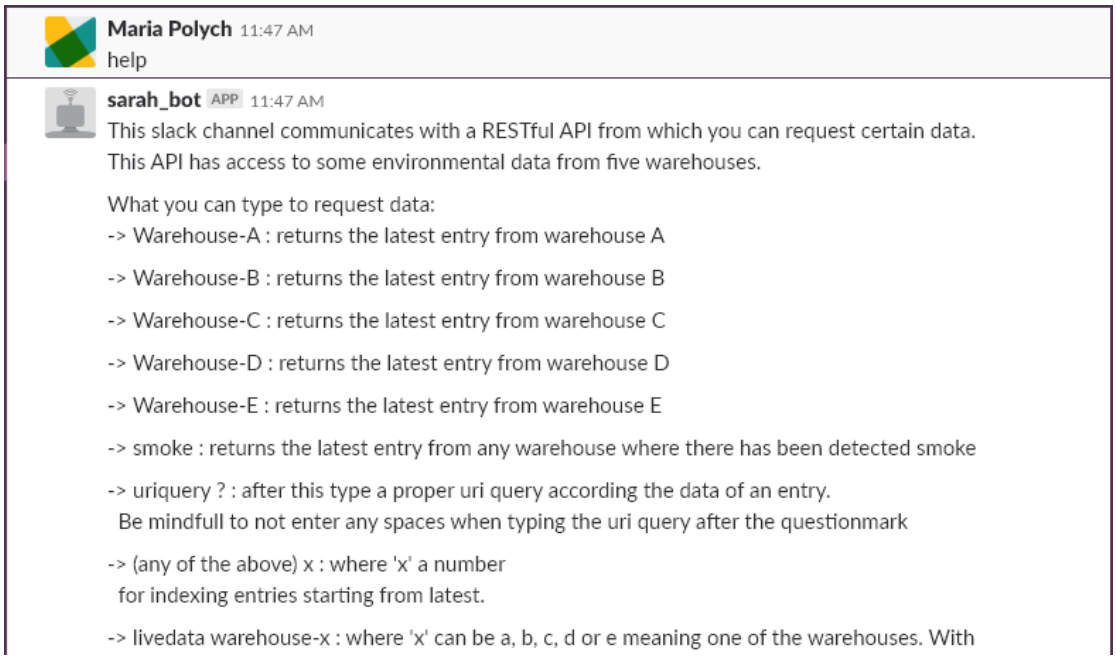
Η κεφαλίδα *"cache-control"* όπως έχει ειπωθεί έχει σημαντικό ρόλο στην επικοινωνία με χρήση REST και αυτή τη φορά έχει διαφορετική τιμή από προηγουμένως που είδαμε τα δεδομένα HATEOAS από *Firefox*. Αυτή τη φορά, η τιμή της αποτελείται από την τιμή *"no-cache, no-transform, max-age=3600"* και η σημαίνει ότι δεν μπορεί κανένας να αποθηκεύσει τα δεδομένα αυτά, ή να τα μεταβάλει για μεταγενέστερη χρήση. Επιπλέον, το κομμάτι *"max-age=3600"* δίνει την πληροφορία ότι αυτά τα δεδομένα που περιέχονται σε αυτό το μήνυμα θα είναι έγκυρα για την επόμενη ώρα (3600 δευτερόλεπτα).

Μετά το πέρας αυτού του διαστήματος, το περιεχόμενο της απόκρισης του ίδιου πόρου θα είναι διαφορετικό. Αυτό φυσικά το έχουμε ορίσει για κάθε πόρο ξεχωριστά με χρήση κώδικα Java, ανάλογα το περιεχόμενο των δεδομένων που διαχειρίζεται. Είναι πολύ λογικό το περιεχόμενο του πόρου που επιστρέφει την τελευταία καταχώρηση για μία δεδομένη αποθήκη να αλλάζει μετά από μία ώρα, καθώς ο αντίστοιχος μικροελεγκτής έχει ρυθμιστεί να παίρνει μετρήσεις ανά μία ώρα και να τις στέλνει στο API.

5.3 Επικοινωνία Μέσω Slack

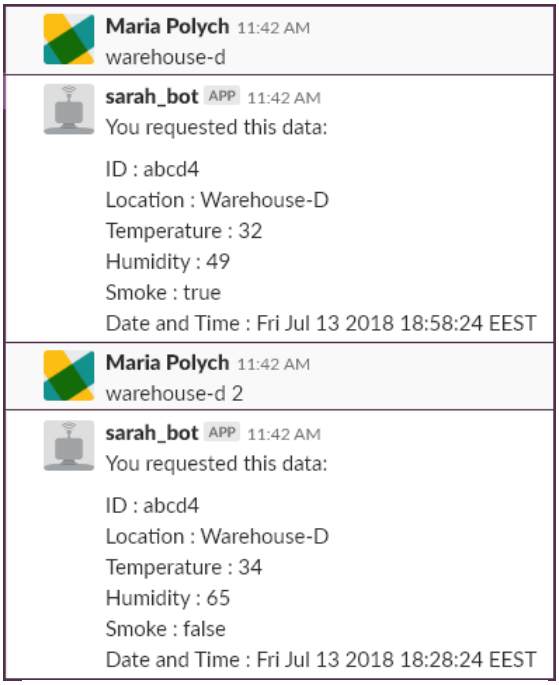
Η αλληλεπίδραση ενός χρήστη με το API μέσω ενός Slack καναλιού είναι ίσως ένα από τα σημαντικά χαρακτηριστικά της εργασίας αυτής, καθώς δείχνει πόσο πολύ μπορεί η αρχιτεκτονική REST να συμβάλλει σημαντικά στο διαδίκτυο των πραγμάτων. Σε αυτήν την υποενότητα θα παρατεθούν εικόνες και στιγμιότυπα όπου πραγματοποιούνται αιτήματα για πληροφορίες μέσω ενός Slack καναλιού.

Ξεκινώντας, στην Εικόνα 36 απεικονίζεται ένα δείγμα του μηνύματος που εμφανίζεται με την αποστολή της λέξης *"help"*. Όπως φαίνεται, περιέχει σύντομες πληροφορίες σχετικά με τα ίδια τα δεδομένα, ενώ αναλύει διεξοδικά και με παραδείγματα τι είδους αναζητήσεις μπορεί να κάνει κάποιος για να επικοινωνήσει με το API. Επίσης επισημαίνεται ότι καμία από τις λέξεις κλειδιά δεν απαιτείται να έχει συγκεκριμένη μορφή ως προς τα κεφαλαία ή πεζά γράμματα.



Εικόνα 36: Απόκριση Node - RED στη λέξη κλειδί "help"

Έτσι, μέσω Slack, μπορεί οποιοσδήποτε είναι μέλος του καναλιού αυτού, να ζητήσει τις τελευταίες μετρήσεις από οποιαδήποτε αποθήκη, την τελευταία καταχώρηση στην οποία εντοπίστηκε καπνός, ακόμη και να στείλει ένα URI ερώτημα με τις παραμέτρους που θέλει ο ίδιος. Επιπλέον, τοποθετώντας έναν αύξον αριθμό - δείκτη δίπλα από οποιαδήποτε αναζήτησή του, δίνεται η δυνατότητα να ζητήσει πιο παλιές μετρήσεις και όχι αποκλειστικά και μόνο την πιο πρόσφατη.



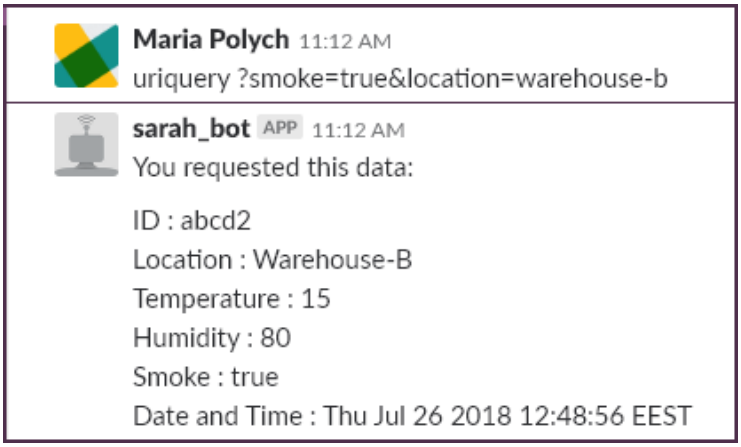
Εικόνα 37: Ζήτηση Δεδομένων Αποθήκης D με Δείκτη και Χωρίς

Τέλος, χρησιμοποιώντας τη λέξη κλειδί *"livedata"* και προσθέτοντας δίπλα της με κενό το όνομα μίας αποθήκης, εκτελείται επιτόπου μέτρηση από τον μικροελεγκτή της αποθήκης αυτής και επιστρέφονται τα δεδομένα στο κανάλι του Slack, ενώ ταυτόχρονα στέλνονται και στην βάση δεδομένων μέσω του RESTful API.

Στην Εικόνα 37 φαίνεται η αναζήτηση της τελευταίας καταχώρησης της αποθήκης D, με και χωρίς δείκτη. Στην πρώτη περίπτωση εμφανίστηκαν οι τελευταίες και πιο πρόσφατες μετρήσεις που υπάρχουν για την αποθήκη D, ενώ στην δεύτερη περίπτωση, με την προσθήκη του δείκτη δύο (2), εμφανίστηκαν οι αμέσως πιο παλιές, που όπως μπορούμε να παρατηρήσουμε πάρθηκαν ακριβώς 30 λεπτά

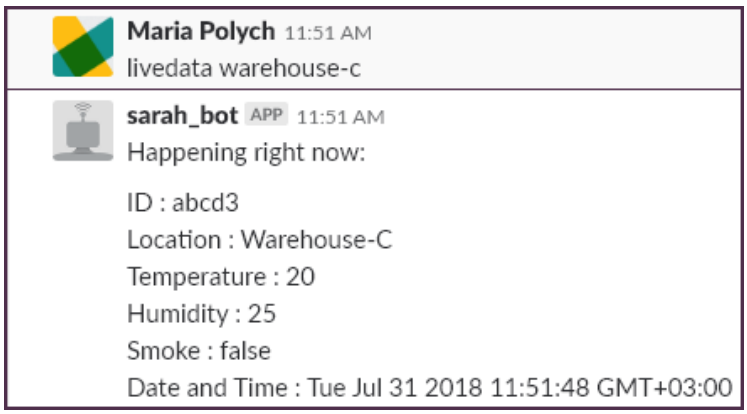
νωρίτερα.

Συνεχίζοντας, στην Εικόνα 38, φαίνεται μία σχετικά πιο σύνθετη αναζήτηση με χρήση της λέξης κλειδί *"uriquery"*, με την οποία μπορούμε να εκτελέσουμε μία αναζήτηση μέσω URI ερωτημάτων. Εδώ, τοποθετήθηκαν δύο παράμετροι οι οποίες ζητούν την τελευταία καταχώρηση της αποθήκης B, όπου εντοπίστηκε καπνός από τον αντίστοιχο αισθητήρα.



Εικόνα 38: Ζήτηση Τελευταίων Μετρήσεων όπου Εντοπίστηκε Καπνός στην Αποθήκη B

Τέλος, ένα στιγμιότυπο με την λειτουργία με την οποία ζητάμε επιτόπου μέτρηση σε μία αποθήκη. Έτσι, στην Εικόνα 39 φαίνεται η αίτηση μέτρησης από τον μικροελεγκτή που βρίσκεται στην αποθήκη C, ενώ αμέσως μετά από μερικά δευτερόλεπτα έρχεται ως απάντηση από το αντίστοιχο *Slack - bot* του Node – RED με τα δεδομένα εκείνης ακριβώς της στιγμής.



Εικόνα 39: Απαίτηση Μετρήσεων από τους Αισθητήρες του Μικροελεγκτή στην Αποθήκη C

Μπορεί κανείς να παρατηρήσει ότι η ώρα του αιτήματος, μέτρησης αλλά και της απάντησης από το Node – RED αναφέρεται ακριβώς στο ίδιο λεπτό της ώρας. Συνεπώς, όλα αυτά τα αιτήματα δεν παίρνουν παρά μερικά δευτερόλεπτα, το αργότερο, για να πραγματοποιηθούν.

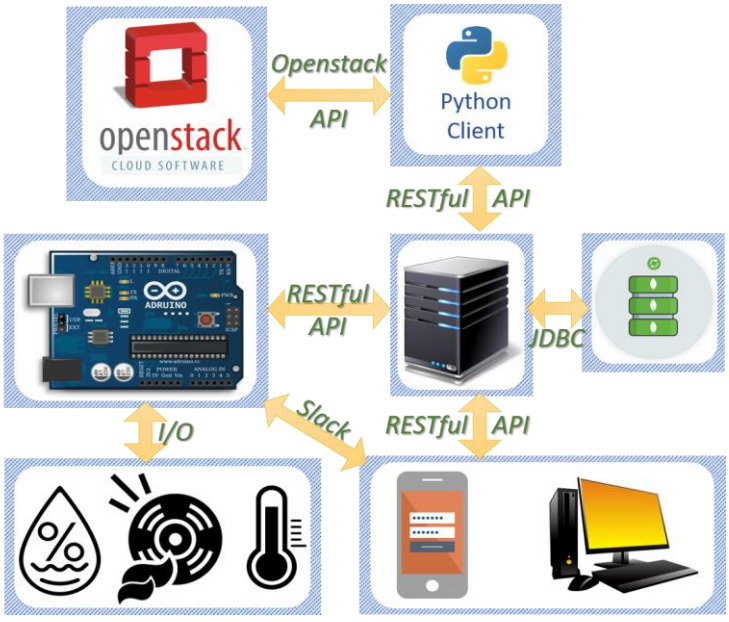
6 ΚΕΦΑΛΑΙΟ 5 : Περίπτωση Χρήσης σε Περιβάλλον Νεφοϋπολογισμού

Στο τελευταίο κεφάλαιο αυτό, αξιοποιούνται όλες οι διεπαφές που δημιουργήθηκαν στο προηγούμενο κεφάλαιο, προκειμένου να παρουσιαστεί η λειτουργικότητα της εφαρμογής της αρχιτεκτονικής REST. Η περίπτωση χρήσης αφορά ένα σενάριο σε ένα χώρο νεφοϋπολογιστικής (datacenter) στον οποίο στεγάζεται η υποδομή για την υποστήριξη και τη λειτουργία της πλατφόρμας Openstack. Είναι εξαιρετικά σημαντικό, ένας χώρος νεφοϋπολογιστικής να διατηρείται περιβαλλοντικά σταθερός, καθώς μεγέθη όπως η θερμοκρασία ή τυχόν πτώσεις τάσεις μπορούν να προκαλέσουν σημαντική ζημιά στην ίδια την υποδομή, τόσο σε επίπεδο υλικών, όσο και σε επίπεδο λογισμικού, ενώ παράλληλα σε τυχόν προβλήματα μπορεί να προκύψει απώλεια δεδομένων.

Η λογική πίσω από την συγκεκριμένη περίπτωση χρήσης είναι η δυνατότητα υποστήριξης λειτουργιών που προσφέρει το ΔτΠ σε ένα περιβάλλον συγκεκριμένων αναγκών, όπως μία νεφοϋπολογιστική υποδομή. Ενώ αυτό το σενάριο θα μπορούσε να πραγματοποιηθεί με χρήση συστημάτων αυτοματισμού, το πλεονέκτημα που προσφέρει η εφαρμογή ενός συστήματος ΔτΠ είναι η ανεξαρτησία αυτού, από το περιβάλλον στο οποίο πρόκειται να εφαρμοστεί, ενώ σε συνδυασμό με την αρχιτεκτονική REST προσφέρει ευελιξία προγραμματιστικής πολυγλωσσικότητας, καθώς επίσης αυξάνει την επεκτασιμότητα και την απόδοση όλου του συστήματος.

Στη συνέχεια της ενότητας, θα ακολουθήσει η περιγραφή της νεφοϋπολογιστκής υποδομής καθώς και ο σκοπός της περίπτωσης χρήσης στα πλαίσια της διπλωματικής εργασίας, ενώ αμέσως μετά θα γίνει μία σύντομη περιγραφή της πλατφόρμας openstack. Στη συνέχεια, θα ακολουθήσει ο τρόπος ενσωμάτωσης της υλοποίησης σε αυτό το περιβάλλον χρήσης, ενώ τέλος θα γίνει ανάλυση συμπερασμάτων για το παραπάνω πείραμα.

6.1 Σκοπός και Περιβάλλον Χρήσης



Εικόνα 40: Η αλληλεπίδραση του Client με το Openstack και το RESTful API

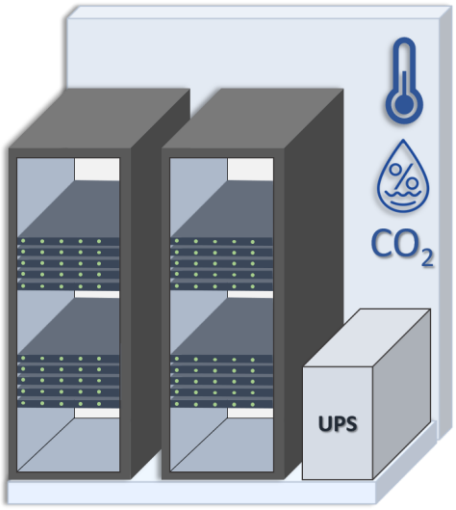
Ο σκοπός της περίπτωσης χρήσης αυτής είναι να αναδειχθούν τα πλεονεκτήματα της ενσωμάτωσης τόσο του ΔτΠ, όσο και της αρχιτεκτονικής REST, μέσα από την εφαρμογή τους σε ένα πραγματικό περιβάλλον, το οποίο ανάλογα με τις συνθήκες που επικρατούν (π.χ. θερμοκρασία, υγρασία) θα πρέπει να λάβει τις αντίστοιχες δράσεις.

Όπως είναι γνωστό στον «τεχνολογικό κόσμο», μία νεφοϋπολογιστική υποδομή έχει πολλές απαιτήσεις τόσο κατά το σχεδιασμό, όσο και για την συντήρησή της κατά τη διάρκεια λειτουργίας της. Για παράδειγμα, η συνεχής και απαιτητική

λειτουργία των διακομιστών μπορεί να αυξήσει όχι μόνο την εσωτερική τους θερμοκρασία αλλά ακόμα και του ίδιου του χώρου στον οποίο βρίσκονται. Η θερμοκρασία μπορεί να αυξηθεί σε τέτοιο βαθμό όπου μπορεί να προκληθεί μόνιμη βλάβη στον εξοπλισμό έχοντας ως συνέπεια την αύξηση του κόστους συντήρησης της υποδομής.

Για τον λόγο αυτό, καθώς και για κάποιους ακόμη (π.χ. προσβασιμότητα, διαθέσιμο επίπεδο τάσης και ρεύματος), πολλές φορές, οι χώροι για τη λειτουργία «datacenters» επιλέγονται να είναι υπόγειοι. Έτσι, είναι εύκολο να γίνει κατανοητό πόσο σημαντική είναι όχι μόνο η δυνατότητα για συνεχή παρακολούθηση του χώρου μιας τέτοιας υποδομής, αλλά και η δυνατότητα να μπορούν να ληφθούν κατάλληλα μέτρα για την πρόληψη βλαβών του υλικό-λογισμικού και απώλειας δεδομένων.

Η υποδομή στην οποία θα εφαρμοστεί το σύστημα που σχεδιάστηκε και αναπτύχθηκε στην προηγούμενη ενότητα, αποτελείται από δύο ερμάρια ύψους περίπου δύο μέτρων, τα οποία υποστηρίζουν τη στήριξη και την καλωδίωση συνολικά 20 διακομιστών. Στην τροφοδοσία τους από την κεντρική παροχή ρεύματος του ορόφου παρεμβάλλεται μία μονάδα αδιάλειπτης παροχής ρεύματος (UPS) για συγκεκριμένο χρονικό διάστημα, έτσι ώστε σε περιπτώσεις πτώσεων τάσης να υπάρχει ένα μικρό περιθώριο χρόνου για την ομαλή απενεργοποίηση του συστήματος από κάποιον υπεύθυνο.



Εικόνα 41: Η Νεφοϋπολογιστική Υποδομή

Ένας αριθμός από τους διακομιστές χρησιμοποιούνται για την λειτουργία της πλατφόρμας Openstack, η οποία είναι μία πλατφόρμα διαχείρισης πόρων (μνήμη RAM, αποθηκευτικός χώρος, επεξεργαστική ισχύς) για την παροχή Εικονικών Μηχανών (VMs). Συνήθως, η διαχείριση πόρων πραγματοποιείται είτε μέσω της γραμμής εντολών ή με τη χρήση του έτοιμου γραφικού περιβάλλοντος που παρέχει. Ωστόσο, η πλατφόρμα αυτή προσφέρει προγραμματιστικές βιβλιοθήκες, οι οποίες μπορούν να χρησιμοποιηθούν στον προγραμματισμό εκτελέσιμων αρχείων και να επιδρούν στην διαχείριση των πόρων ανάλογα με τις επιθυμίες του προγραμματιστή.

Στόχος της παρούσας περίπτωσης χρήσης είναι η χρήση αυτών των βιβλιοθηκών για την συγγραφή προσαρμοσμένων εκτελέσιμων αρχείων, τα οποία θα περιλαμβάνουν και την κλήση των RESTful διεπαφών της προηγούμενης ενότητας, προκειμένου να ληφθούν αποφάσεις βάσει των δεδομένων για τις περιβαλλοντικές συνθήκες της υποδομής. Τα δεδομένα, τα οποία μπορούν να προσομοιωθούν από την ενότητα της υλοποίησης είναι αυτά της θερμοκρασίας, της υγρασίας και της ύπαρξης καπνού για πέντε διαφορετικούς χώρους με το όνομα «Warehouse - X», όπου X={A,B,C,D,E}. Για λόγους ομαλής σύνδεσης με την περίπτωση χρήσης, υποθέτουμε πως η περιγραφείσα υποδομή βρίσκεται σε μία από αυτές τις αποθήκες, με όνομα «Warehouse - A».

Λόγω της πολυπλοκότητας που διακατέχει την πλατφόρμα Openstack, η περίπτωση χρήσης θα περιοριστεί σε δύο κανόνες που θα πρέπει να εφαρμοστούν σε συγκεκριμένες περιπτώσεις ανάλογα των δεδομένων που έρχονται από το API. Οι δύο κανόνες αυτοί είναι :

1. **«Αν η θερμοκρασία παραμένει πάνω από τους 25^{o} C για πάνω από 5 ώρες, τότε να ξεκινήσει η διαδικασία επείγουσας απενεργοποίησης των εικονικών μηχανών μετά»**

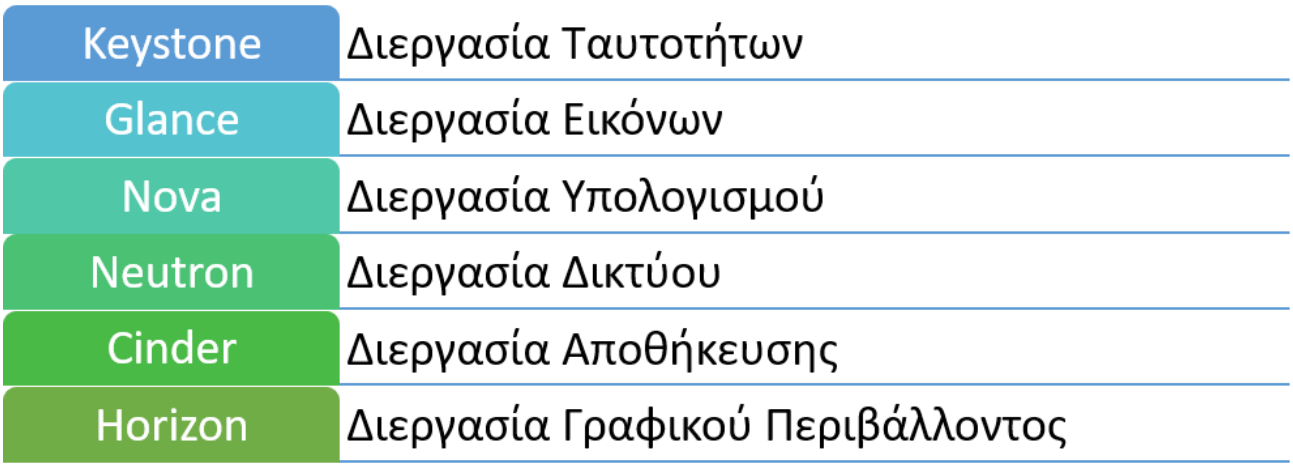
2. **«Αν εντοπιστεί καπνός, τότε να ξεκινήσει η διαδικασία δημιουργίας και αποστολής αντιγράφων ασφαλείας των εικονικών μηχανών»**

Προφανώς, πρόκειται για δύο πολύ απλούς κανόνες, οι οποίοι όμως μπορούν να διαμορφωθούν και να επεκταθούν με οποιοδήποτε τρόπο κρίνει ο διαχειριστής της υποδομής απαραίτητο.

6.2 Η Πλατφόρμα OpenStack

Η πλατφόρμα Openstack είναι ένα εργαλείο νεφοϋπολογιστικής ανοιχτού κώδικα το οποίο χρησιμοποιείται για τη δημιουργία ιδιωτικού υπολογιστικού νέφους τύπου «Υποδομής ως Υπηρεσία». Το συγκεκριμένο μοντέλο νέφους προσφέρει τη δυνατότητα αξιοποίησης του υλικού της υποδομής κατ' απαίτηση του χρήστη (ή πελάτη αν είναι επί πληρωμή), δίνοντάς του την ελευθερία επιλογής σε επεξεργαστική ισχύ, μνήμη, αποθηκευτικό χώρο και δικτυακής συνδεσιμότητας. Η δημιουργία VMs αξιοποιεί το υλικό αυτό, ενώ παράλληλα είναι μία υπηρεσία που μπορεί να προσφέρει πολλά πλεονεκτήματα σε χρήστες, οι οποίοι δεν επιθυμούν ή δεν προτίθενται να επενδύσουν οικονομικά σε υλικολογισμικό για την οποιαδήποτε διεργασία χρειάζεται να ολοκληρώσουν.

Το Openstack προσφέρει αυτήν την υπηρεσία, βασίζοντας τη λειτουργία του σε έξι βασικές εσωτερικές διεργασίες (services) που «τρέχουν» και επικοινωνούν χρησιμοποιώντας ειδικές προγραμματιζόμενες διεπαφές που έχουν φτιαχτεί για αυτόν το λόγο. Οι διεργασίες αυτές είναι:



Εικόνα 42: Οι 6 Βασικές Διεργασίες του Openstack

- Διεργασία Ταυτοτήτων (Keystone): Είναι υπεύθυνη για την ταυτοποίηση και την ασφάλεια των χρηστών και των λειτουργιών, ολόκληρης της υποδομής. Δημιουργεί ρόλους χρηστών και επαληθεύει προσωπικά κλειδιά δίνοντας τις κατάλληλες άδειες για τη διαχείριση των πόρων σε κάθε επίπεδο.

- Διεργασία Εικόνων (Glance): Είναι υπεύθυνη για την διαχείριση και τη δημιουργία εικόνων λειτουργικών συστημάτων, καθώς και τη δυνατότητα τα τραβάει «στιγμιότυπα» (snapshots) αυτών, τα οποία μπορούν να χρησιμοποιηθούν είτε ως αντίγραφα ασφάλειας, είτε για τη δημιουργία ενός δεύτερου VM έχοντας ως πρότυπο ένα συγκεκριμένο και προσαρμοσμένο σε ρυθμίσεις και λειτουργίες λειτουργικό σύστημα.

- Διεργασία «Υπολογισμού» (Nova): Είναι υπεύθυνη για τη δημιουργία των VMs, καθώς επικοινωνεί με τον «ενορχηστρωτή» ώστε να διανείμει κατάλληλα τους πόρους για κάποιο εικόνα λειτουργικού συστήματος, το οποίο παρέχεται από την διεργασία Glance.

- Διεργασία Δικτύου (Neutron): Είναι υπεύθυνη για την παροχή δικτυακής και διαδικτυακής συνδεσιμότητας στις εικονικές μηχανές, τη δημιουργία εικονικών δρομολογητών, δικτύων και υποδικτύων.

- Διεργασία Αποθήκευσης (Cinder): Είναι υπεύθυνη για τη δημιουργία τόμων αποθήκευσης, οι οποίοι χρησιμοποιούνται ως οι «σκληροί δίσκοι» των VMs, ενώ το μέγεθος τους είναι μεταβλητό και μπορεί να ρυθμιστεί κατ' απαίτηση των αναγκών του χρήστη.

- Διεργασία Γραφικού Περιβάλλοντος (Horizon): Είναι υπεύθυνη για τη λειτουργία ενός φιλικού προς τον χρήστη γραφικού περιβάλλοντος βασισμένο σε έναν Apache Server, μέσω του οποίου μπορεί να γίνει η διαχείριση της πλατφόρμας από τον Διαχειριστή, αλλά και των VMs του κάθε χρήστη, κάνοντας χρήση όλων των προηγούμενων διεργασιών.

Επιπλέον, το openstack μπορεί και συνεργάζεται με το CEPH, το οποίο είναι ένα ανοιχτού κώδικα, αποκεντρικοποιημένο αποθηκευτικό σύστημα (software defined storage – SDS). Λόγω του μεγάλου φορτίου δεδομένων και του τρόπου με το οποίο διαμοιράζονται στις νεφοϋπολογιστικές υποδομές, ένα σύστημα όπως το CEPH προσφέρει επεκτασιμότητα και άμυνα έναντι στο πρόβλημα σφάλματος ενός σημείου (single point of failure – SPOF).

Όλα τα παραπάνω αποτελούν ξεχωριστές μονάδες εντολών και κάνουν χρήση διάφορων εργαλείων και εξαρτημάτων χαμηλού επιπέδου προκειμένου να επιτευχθούν όλες αυτές οι διεργασίες και λειτουργίες. Η επικοινωνία μεταξύ αυτών των εξαρτημάτων πραγματοποιείται με τη χρήση προγραμματιζόμενων διεπαφών (APIs), τα οποία έχουν σχεδιαστεί ώστε να προσφέρουν στον διαχειριστή της υποδομής ευελιξία και ευκολία στην διατήρησή της.

Σε αυτό το σημείο ας σημειωθεί πως η πλατφόρμα openstack είναι μία περίπλοκη πλατφόρμα και οι παραπάνω λειτουργίες αποτελούν μόνο μία πολύ συνοπτική και απομακρυσμένη περιγραφή της. Η εμβάθυνση στις λειτουργίες της είναι εκτός πεδίου για την διπλωματική εργασία, ενώ στις επόμενες υπο-ενότητες θα χρησιμοποιηθεί μόνο η διεργασία εικόνων Glance, προκειμένου να ολοκληρωθεί η περίπτωση χρήσης.

6.3 Ενσωμάτωση της RESTful Υλοποίησης

Στην υποενότητα αυτή θα παρουσιαστεί ο τρόπος με τον οποίο μπορεί να ενσωματωθεί η λειτουργία της υλοποίησης της προηγούμενης ενότητας, αναδεικνύοντας το γεγονός ότι η χρήση της αρχιτεκτονικής REST για την επικοινωνία και την μεταφορά δεδομένων μπορεί να προσφέρει πλήρη ανεξαρτησία κατά την ανάπτυξη και τον προγραμματισμό των επιμέρους εξαρτημάτων σε ένα τεχνολογικό περιβάλλον, όπως μία νεφοϋπολογιστική υποδομή. Παράλληλα θα αναδειχθεί και η δυνατότητα να προσφέρει επεκτασιμότητα και ευελιξία στην συντήρηση του περιβάλλοντος αυτού, καθώς χωρίς κανένα εμπόδιο ή τεχνική δυσκολία, θα μπορούσαν για παράδειγμα να προστεθούν επιπλέον αισθητήρες, προσφέροντας ενισχυμένη πληροφορία στο σύστημα ΔτΠ, προκειμένου να παρθούν ανάλογες αποφάσεις για την διατήρηση των συνθηκών του χώρου.

6.3.1 Λογισμικό και Προγραμματιστικά Πλαίσια

Όπως ακριβώς και στην ενότητα της υλοποίησης του RESTful API, έτσι και στην περίπτωση χρήσης αυτού θα γίνει χρήση κάποιων προγραμματιστικών εργαλείων, προκειμένου να ολοκληρωθεί το σύστημα παρακολούθησης περιβαλλοντικών συνθηκών μιας νεφοϋπολογιστικής υποδομής.

- Visual Studio Code: Πρόκειται για ένα ολοκληρωμένο περιβάλλον ανάπτυξης κώδικα, ομοίως με το Eclipse IDE που χρησιμοποιήθηκε για την ανάπτυξη Java. Είναι ιδιαίτερα δημοφιλές, ειδικά μεταξύ της κοινότητας ανάπτυξης ιστοσελίδων, λόγω της ευελιξίας που μπορεί και παρέχει στους προγραμματιστές για την ανάπτυξη προγραμμάτων πολλών διαφορετικών γλωσσών παράλληλα. Προσφέρει σημαντικές και εργονομικές λειτουργίες όπως την ενσωμάτωση επεκτάσεων που βοηθούν στην επέκταση της λειτουργικότητάς του, και τη δυνατότητα επιλογής ερμηνευτή για έλεγχο σφαλμάτων ανάλογα τη γλώσσα προγραμματισμού επιλογής.

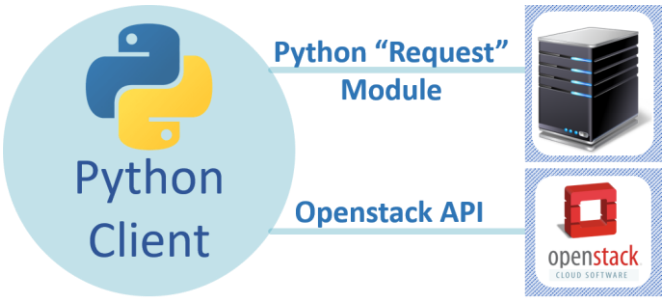
- Python Request Module: Αποτελεί μέρος του κορμού των βιβλιοθηκών της γλώσσας Python, και είναι το πακέτο εκείνο, το οποίο μπορεί να προσφέρει τις απαραίτητες εντολές για την κλήση αιτημάτων HTTP. Δεν αποτελεί την πιο δημοφιλή επιλογή, καθώς πρόκειται για την πιο απλή εφαρμογή βασικών κανόνων για την επικοινωνία με χρήση αυτού του πρωτοκόλλου, ωστόσο, στα πλαίσια του προγραμματισμού αυτής της απλής περίπτωσης εφαρμογής θεωρείται επαρκές προκειμένου να υπάρξει επικοινωνία με το RESTful API της προηγούμενης ενότητας.

- Openstack Instance: Για λόγους ασφαλείας και διασφάλισης της ίδιας της υποδομής, προφανώς, η δοκιμή και η σχεδίαση του πελάτη δεν θα πραγματοποιηθεί στην ίδια την υποδομή του openstack, αλλά σε μία δεύτερη που έχει δημιουργηθεί με τις πιο βασικές υπολογιστικές ανάγκες. Σε έναν οικιακό υπολογιστή, έχει δημιουργηθεί μία εικονική μηχανή, στην οποία έχει γίνει εγκατάσταση της πλατφόρμας openstack και

- Openstack Glance Python API: Πρόκειται για την διεργασία εκείνη που διαχειρίζεται τις εικόνες του λειτουργικού συστήματος κάθε εικονικής μηχανή που δημιουργείται από το openstack. Θα χρησιμοποιηθεί το Python API αυτού, προκειμένου να σταλούν οι κατάλληλες εντολές απομακρυσμένα από τον πελάτη προκειμένου να ολοκληρωθεί η διαδικασία αντιγράφων ασφαλείας. Στην παρακάτω περίπτωση χρήσης, θα γίνει χρήση της εντολής που δημιουργεί στιγμιότυπα των εικονικών μηχανών και τα στέλνει προς αποθήκευση σε απομακρυσμένο σημείο.

- Openstack Nova Python API: Είναι η διεργασία με την οποία γίνεται η διαχείριση των εικονικών μηχανών, συμπεριλαμβανομένου και της ενεργοποίησης και απενεργοποίησής τους. Θα χρησιμοποιηθεί προκειμένου να πραγματοποιηθεί η άμεση απενεργοποίηση των εικονικών μηχανών στον δεύτερο κανόνα, όπου εντοπίζεται υψηλή θερμοκρασία συνεχόμενα για πάνω από πέντε ώρες, έτσι ώστε να διασφαλιστεί η ασφάλεια των δεδομένων τους σε περίπτωση αύξησης της θερμοκρασίας με αποτέλεσμα την απότομη απενεργοποίηση των διακομιστών.

6.3.2 Αρχιτεκτονική και Λειτουργικότητα



Εικόνα 43: Ο «πελάτης» Openstack

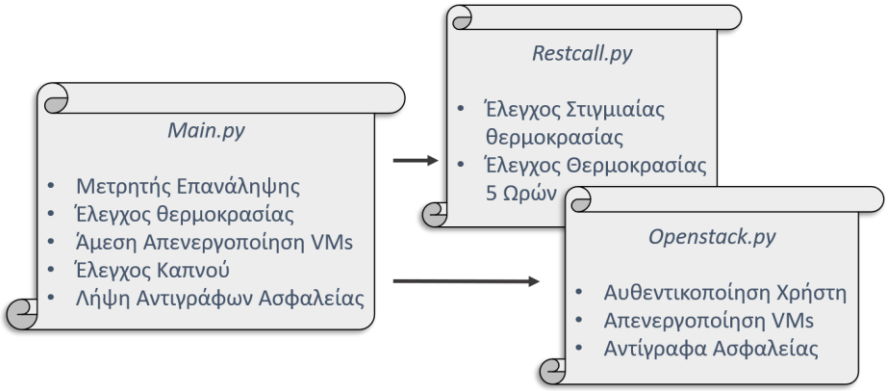
Όπως ειπώθηκε στην αντίστοιχη ενότητα, η αρχιτεκτονική REST βασίζεται στο δικτυακό μοντέλο πελάτη – εξυπηρετητή (client - server). Οι RESTful διεπαφές της προηγούμενης ενότητας αποτελούν τη λειτουργία του εξυπηρετητή, για αυτό γίνεται και η χρήση ενός εργαλείου όπως ο Apache Tomcat που προσφέρει λειτουργία διακομιστή.

Για την αξιοποίησή των διεπαφών από την πλατφόρμα του Openstack θα χρειαστεί να σχεδιαστεί και να προγραμματιστεί ένας «πελάτης» (client), ο οποίος θα είναι υπεύθυνος για την κλήση των HTTP αιτημάτων σε συγκεκριμένα διαστήματα. Παράλληλα θα έχει την άδεια να εκτελεί εντολές και να επιδρά στο Openstack κάνοντας χρήση της διεργασίας Glance, που αναφέρθηκε στην αρχή της ενότητας, προκειμένου να εκκινεί τη διαδικασία αντιγράφων ασφαλείας. Επίσης, καθώς το Openstack είναι γραμμένο κυρίως με χρήση της γλώσσας Python, ο πελάτης θα προγραμματιστεί στην ίδια γλώσσα, έτσι ώστε να είναι πιο ομαλή η ενσωμάτωσή του σε αυτό.

Σε αντίθεση με την Java, η Python είναι μία υψηλού επιπέδου γλώσσα, η οποία υποστηρίζει τόσο την αντικειμενοστραφή λογική όσο και την δομημένη (ή διαδικαστική). Κατά το σχεδιασμό του openstack πελάτη, η επιλογή του δομημένου προγραμματισμού είναι καλύτερη καθώς διαχωρίζει τη διαδικασία που πρέπει να ακολουθηθεί σε μικρότερες υπορουτίνες, ανάλογα με την πληροφορία που δίνουν τα δεδομένα.

Στην περίπτωση της υλοποίησης στην προηγούμενη ενότητα, ήταν σημαντικό να δοθεί προσοχή κατά τον προγραμματισμό στην αντικειμενοστραφή λογική, καθώς το RESTful API, προκειμένου να ταξινομήσει την πληροφορία στις διαφορετικές διεπαφές, έπρεπε τα δεδομένα να είναι σημασιολογικά κατηγοριοποιημένα στις διάφορες μεταβλητές. Ωστόσο, στην περίπτωση της πλευράς του πελάτη, η προσοχή δίνεται όχι τόσο στη σημασιολογία που μεταφέρει η πληροφορία, αλλά στις διεργασίες που πρέπει να πραγματοποιηθούν βάσει αυτών. Έτσι, επιλέγοντας δομημένο προγραμματισμό, οι διεργασίες που πρέπει να ακολουθηθούν μπορούν να χωριστούν σε διαφορετικές υπορουτίνες, οι οποίες μπορούν να κληθούν από την βασική ρουτίνα μόλις κριθεί απαραίτητο.

Στη δομή του πελάτη προγραμματισμένο σε Python, υπάρχει μία βασική ρουτίνα (main.py), η οποία είναι υπεύθυνη ώστε να οργανώνει τις διεργασίες που πρέπει να γίνονται και για τα χρονικά διαστήματα κατά τα οποία θα πρέπει αυτές να εκτελούνται. Οι διεργασίες αυτές είναι οργανωμένες σε δύο υπορουτίνες (restcall.py και openstack.py), οι οποίες περιλαμβάνουν όλες τις μεθόδους για την επικοινωνία με τον RESTful server της προηγούμενης ενότητας και με την Openstack υποδομή.



Εικόνα 44: Οργάνωση Python Κώδικα σε Ρουτίνες

6.3.3 Ο RESTful Πελάτης

Ξεκινώντας από την κορυφή των διεργασιών, παρακάτω φαίνεται ο κώδικας της κεντρικής ρουτίνας *main*. Μόλις τεθεί σε λειτουργία, ξεκινάει ένας χρονομετρητής (γραμμή 7), βάσει του οποίου ανά πέντε λεπτά δίνεται εντολή, ώστε χρησιμοποιώντας την υπορουτίνα *restcall* (γραμμή 8) να πραγματοποιηθεί επικοινωνία με το RESTful API και να πάρει τα τελευταία δεδομένα από τον χώρο της υποδομής.

Τα δεδομένα που λαμβάνει με αυτήν την εντολή είναι μορφής JSON, όπως ακριβώς και στα παραδείγματα της υλοποίησης της προηγούμενης ενότητας. Έτσι, έχοντας πλέον τα δεδομένα αυτά, μπορεί να γίνει ένα απλός έλεγχος (γραμμή 9) αν η στιγμιαία θερμοκρασία την δεδομένη στιγμή είναι υψηλή.

```
1  import threading
2  import restcall
3  import openstack
4  import json
5
6  def thesis():
7     threading.Timer(300.0, thesis).start()
8     dataroom = json.loads(restcall.getLocationLastData)
9     if dataroom["temperature"] >= 25:
10       if restcall.getFiveHoursPastTimeFormat()>= 25:
11          threading.Timer(600.0, thesis).start()
12          openstack.emergencyPowerDown
13       else:
14          return "False Alarm"
15    else:
16       return "Dataroom Safe from Overheating"
17
18    if dataroom["smoke"] == True:
19       openstack.beginBackUp
20    else:
21       return "Dataroom Safe from Fire"
22
23 thesis()
```

Εικόνα 45: Βασική Ρουτίνα

Στην περίπτωση αυτήν, αμέσως μετά τίθεται σε λειτουργία μία ειδική συνάρτηση της υπορουτίνας *restcall*, η οποία παράγει τον μέσο όρο της θερμοκρασίας του χώρου για τις τελευταίες πέντε ώρες (γραμμή 10). Στην περίπτωση όπου και πάλι η θερμοκρασία υπερβαίνει ή είναι ίση με 25 βαθμούς Κελσίου, τότε τίθεται σε ισχύ, η υπορουτίνα *openstack* και ξεκινάει η συνάρτηση άμεσης απενεργοποίησης των εικονικών μηχανών για διαφύλαξη της ομαλής λειτουργίας τους (γραμμή 12).

Στη συνέχεια, γίνεται έλεγχος για την ύπαρξη καπνού στον χώρο (γραμμή 18). Σε αυτό το σημείο του ελέγχου των δεδομένων, η μεταβλητή που σχετίζεται με τον καπνό είναι δυαδική, ενώ η ακριβής μέτρηση και το κατώφλι των PM (Particle Materials) για τίθεται στο επίπεδο του μικροελεγκτή στον οποίο βρίσκεται ο αισθητήρας. Στην περίπτωση που εντοπιστεί καπνός, αμέσως μετά εκτελείται η εντολή, η οποία θέτει σε λειτουργία την μέθοδο της υπορουτίνας για την εκκίνηση της λήψης αντιγράφων ασφαλείας από τις εικονικές μηχανές.

Επίσης, στη γραμμή 11 φαίνεται η εντολή, η οποία δίνει καθυστέρηση δέκα λεπτών πριν την εντολή για τη λήψη αντιγράφων ασφαλείας. Αυτό γίνεται καθώς, η διαδικασία απενεργοποίησης των εικονικών μηχανών χωρίς να σταλεί προειδοποίηση στους χειριστές αυτών, μπορεί να επιφέρει απώλειες δεδομένων ή ακόμα και βλάβες στην ομαλή λειτουργία των διεργασιών που υποστηρίζουν. Για αυτόν το λόγο, δίνεται ένα χρονικό περιθώριο δέκα λεπτών, προκειμένου να ειδοποιηθούν τόσο ο διαχειριστής της υποδομής, όσο και οι χειριστές των εικονικών μηχανών, έτσι ώστε να ολοκληρώσουν τις διεργασίες τους. Η ειδοποίηση αυτή μπορεί να σταλεί με τη χρήση της αντίστοιχης Node RED ροής.

Προχωρώντας στην υπορουτίνα *restcall*, της οποίας ο κώδικας φαίνεται παρακάτω, είναι υπεύθυνη για την επικοινωνία με το RESTful API. Συνολικά αποτελείται από τρεις μεθόδους, εκ των οποίων οι δύο χρησιμοποιούν τις αντίστοιχες διεπαφές, προκειμένου να τραβήξουν τα JSON δεδομένα του χώρου (γραμμές 8 και 17).

```
1  import requests
2  from datetime import date, datetime, timedelta
3  import re
4  import json
5
6  def getLocationLastData():
7     headers = {'content-type': 'application/json'}
8     response = requests.get('http://localhost:8081/rest/webapi/ThesisA
                      PI/SensorLocation/Warehouse-A/', headers=headers)
9     if response.status_code == 200:
10        warehouseEntry = json.loads(response.text)
11        return warehouseEntry
12    else:
13        return "Cannot Retrieve Data"
14
15 def checkFiveHourTemp():
16     headers = {'content-type': 'application/json'}
17     response = requests.get('http://localhost:8081/rest/webapi/Thesis
              API/SensorLocation/Warehouse-A/allData', headers=headers)
```

Εικόνα 46: Υπορουτίνα *restcall (a)*

```
18      if response.status_code == 200:
19         temperature = 0
20         counter = 1
21         warehouse = json.loads(response.text)
22         dateTime = getFiveHoursPastTimeFormat()
23         for entry in warehouse:
24             if entry["dateTime"]> dateTime:
25                 temperature = (temperature + entry["temperature"])/counter
26                 counter = counter + 1
27         return temperature
28      else:
29          return 30
30
31  def getFiveHoursPastTimeFormat():
32      today = date.today().strftime("%Y-%m-%dT")
33      now = (datetime.now() - timedelta(hours=5)).strftime("%H:%M:%S")
34      currentDateTimeFormat = today + now + "+03:00[Europe/Athens]"
35      return (currentDateTimeFormat)
```

Εικόνα 47: Υπορουτίνα *restcall* (β)

Η τρίτη και τελευταία συνάρτηση χρησιμοποιείται για συντακτικούς λόγους, φτιάχνοντας ένα αλφαριθμητικό με την ημερομηνία και ώρα που απαιτείται, κατά την διαδικασία ελέγχου της θερμοκρασίας για τις τελευταίες πέντε ώρες. Το αλφαριθμητικό αυτό αποτελεί το κατώφλι των πέντε ωρών, βάσει του οποίου πραγματοποιείται ο έλεγχος ολόκληρων των δεδομένων του χώρου αυτού και συγκρίνεται με την ώρα των δεδομένων από το RESTful API. Αν η ώρα των δεδομένων είναι μικρότερη, δηλαδή πιο πρόσφατη από το κατώφλι, τότε η αντίστοιχη θερμοκρασία συνυπολογίζεται στο τελικό αποτέλεσμα.

Τέλος, στην υπορουτίνα *openstack*, ορίζονται οι δύο συναρτήσεις, οι οποίες χρησιμοποιούνται για την λήψη αντιγράφων ασφαλείας των εικονικών μηχανών και την άμεση απενεργοποίησή τους, στις αντίστοιχες περιπτώσεις. Το πρώτο βήμα και στις δύο συναρτήσεις είναι να γίνει αυθεντικοποίηση χρήστη, προκειμένου να είναι δυνατή η αποστολή εντολών στην πλατφόρμα. Για να επιτευχθεί η αυθεντικοποίηση και να δημιουργηθεί μία σύνδεση, θα πρέπει να δοθούν τα κατάλληλα στοιχεία σύνδεσης, τα οποία εχουν παραχθεί κατά τη δημιουργία ενός χρήστη στο Openstack. Στις γραμμές 9 και 23 της εικόνας 48 φαίνονται αυτά τα στοιχεία σύνδεσης, καθώς περνάνε ως παράμετροι στην εντολή "*session*" (γραμμές 10 και 24), η οποία δημιουργεί μία σύνδεση με την πλατφόρμα του openstack.

Στην πρώτη συνάρτηση, αφού πραγματοποιηθεί σύνδεση με το openstack, πρώτα αποκτάται μία λίστα με όλες τις διαθέσιμες εικονικές μηχανές και στη συνέχεια γίνεται έλεγχος της τρέχουσας κατάστασής τους. Σε αυτό το σημείο γίνεται η υπόθεση πώς οποιαδήποτε εικονική μηχανή είναι εκείνη τη στιγμή ενεργή (ως "ACTIVE"), εκτελεί κάποια σημαντική διεργασία και επομένως θα πρέπει να κρατηθεί αντίγραφο αυτής. Έτσι, οι εικονικές μηχανές, των οποίων η κατάσταση είναι οτιδήποτε άλλο, λαμβάνονται ως δευτερεύοντος σημασίας και δεν συμμετέχουν στην διαδικασία λήψης αντιγράφων ασφαλείας.

```
1   from keystoneauth1 import loading
2   from keystoneauth1 import session
3   from novaclient import client as novaclient
4   from glanceclient import client as glanceclient
5
6   def beginBackUp():
7      print("Back Up Images sequence")
8      loader = loading.get_plugin_loader("password")
9      auth = loader.load_from_options(auth_url=AUTH_URL, username=USERNAM
       E, password=PASSWORD, project_id=PROJECT_ID, user_domain_name=USER_DOM
       AIN_NAME)
10     sess = session.Session(auth=auth)
11     nova = novaclient.Client(sess)
12     server_list = nova.servers.list()
13     for server in server_list:
14         if server.status == "ACTIVE":
15             server.stop()
16             glanceclient.getSnapshot(server.id)
17         else:
18              server_list.remove(server.name)
19
20  def emergencyPowerDown():
21      print("Start Power down images and hardware")
22      loader = loading.get_plugin_loader("password")
23      auth = loader.load_from_options(auth_url=AUTH_URL, username=USERNA
       ME, password=PASSWORD, project_id=PROJECT_ID, user_domain_name=USER_DO
       MAIN_NAME)
24      sess = session.Session(auth=auth)
25      nova = novaclient.Client(sess)
26      server_list = nova.servers.list()
27      for server in server_list:
28          server.stop()
```

Εικόνα 48: Υπορουτίνα *Openstack*

Οποιαδήποτε μηχανή χαρακτηρίζεται ως ενεργή, πρώτα στέλνεται η εντολή να παύσει η λειτουργία της και αμέσως μετά στέλνεται η εντολή της λήψης ενός στιγμιότυπου αυτής. Τα στιγμιότυπα αποτελούν μία στιγμιαία εικόνα ολόκληρου του λειτουργικού συστήματος καθώς και της κατάστασης όλων των διεργασιών που τρέχουν τη δεδομένη στιγμή. Στο στιγμιότυπο συμπεριλαμβάνεται ακόμα και κάποια βάση δεδομένων που τυχόν έχει αποθηκευτεί σε κάποια μηχανή. Έτσι, η λήψη ενός στιγμιότυπου αποτελεί ένα είδος αντιγράφου ασφαλείας, καθώς μπορεί να χρησιμοποιηθεί στην εκ νέου δημιουργία μιας δεύτερης πανομοιότυπης εικονικής μηχανής, σε περίπτωση που προκύψει κάποια βλάβη στην πρώτη.

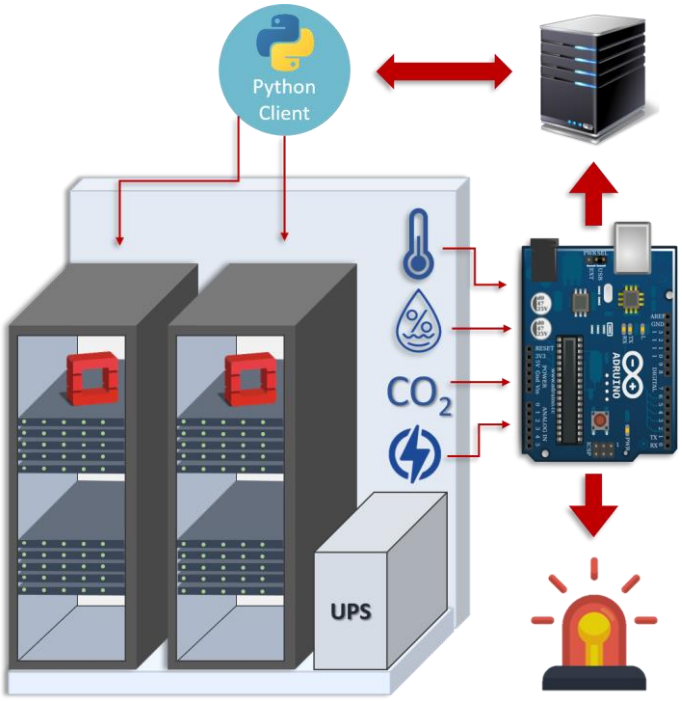
Τέλος, η δεύτερη και τελευταία συνάρτηση της υπορουτίνας *openstack*, είναι εκείνη κατά την οποία εκτελείται άμεση απενεργοποίηση των εικονικών μηχανών στην περίπτωση όπου εντοπιστεί υψηλότερη από το επιτρεπτό θερμοκρασία για πάνω από πέντε ώρες. Ο κώδικας αποτελεί κομμάτι της πρώτης συνάρτησης, καθώς εκτελεί την ίδια αλληλουχία εντολών με την πρώτη, αρχικά προκειμένου να επιτευχθεί η αυθεντικοποίηση χρήστη και στη συνέχεια προκειμένου να γίνει έλεγχος των μηχανών με ενεργή κατάσταση ώστε να τεθούν εκτός λειτουργίας.

6.4 Αποτελέσματα – Συμπεράσματα Περίπτωσης Χρήσης

Η παραπάνω περίπτωση χρήσης δεν αποτελεί παρά ένα μόνο παράδειγμα ενσωμάτωσης ενός συστήματος ΔτΠ σε ένα πραγματικό περιβάλλον, ενώ η κλίμακα στην οποία πραγματοποιήθηκε ήταν σχετικά μικρή. Οι βιομηχανικές νεφοϋπολογιστικές υποδομές, είναι συνήθως αρκετά πιο μεγάλης κλίμακας και αφιερώνονται σε αυτές ειδικά διαμορφωμένες αίθουσες. Σε αυτές υπάρχει επαρκής κλιματισμός και σύστημα εξαερισμού του θερμού αέρα, προκειμένου να δημιουργείται η κατάλληλη ροή αέρα για την ψύξη των μηχανημάτων.

Ωστόσο, η συντήρηση τέτοιων υποδομών δεν είναι σε καμία περίπτωση εύκολη σε οποιαδήποτε κλίμακα. Πολλές φορές υποστηρίζουν μερικά από τα βασικότερα και πιο κρίσιμα εργαλεία στην ανάπτυξη μεγαλύτερων συστημάτων, παρέχοντας απομακρυσμένη υπολογιστική ισχύ, απομακρυσμένη αποθήκευση και πολλές ακόμα πιο εξειδικευμένες λειτουργίες, τις οποίες περιλαμβάνει ο τομέας της νεφοϋπολογιστικής. Έτσι, είναι εξαιρετικά σημαντικό να διαφυλάσσεται η ομαλή λειτουργία τους, προκειμένου να μην υπάρξουν βλάβες ή και απώλειες στις διεργασίες που υποστηρίζουν κάθε στιγμή.

Το ΔτΠ μπορεί να βοηθήσει σημαντικά στη συντήρηση και τη διαχείριση υποδομών, καθώς μπορεί ανεξάρτητα από τη λειτουργικότητά αυτής να δώσει πληροφορίες και δεδομένα σχετικά με το περιβάλλον, δίνοντας βαρύτητα σε παραμέτρους που επιδρούν στην λειτουργία των διακομιστών. Οι πιο σημαντικές από αυτές τις παραμέτρους αποτελούν η θερμοκρασία του χώρου και το επίπεδο τάσης και ρεύματος που παρέχονται. Αν κάποιο από αυτά τα μεγέθη παρεκκλίνουν από τα σωστά όρια, η υποδομή ξεκινά να υπολειτουργεί προκαλώντας προβλήματα επικοινωνίας και ταχύτητας λειτουργίας των διακομιστών.



Εικόνα 49: Έξυπνο Περιβάλλον Νεφοϋπολογιστικής Υποδομής

Παράλληλα, η χρήση διεπαφών REST επιτρέπει την άμεση και στιγμιαία επικοινωνία μεταξύ ΔτΠ και υποδομής, καθώς επιτρέπει τη μεταφορά δεδομένων με τη χρήση του πρωτοκόλλου HTTP πάνω από το διαδίκτυο. Επιπλέον, ο συνδυασμός ΔτΠ και REST προσδίδει επεκτασιμότητα στο σύστημα, επιτρέποντάς του να αποκτήσει επιπλέον δυνατότητες. Για παράδειγμα, στην περίπτωση χρήσης παραπάνω, έχουν φτιαχτεί δύο κανόνες δράσης της υποδομής βάσει δύο παραμέτρων, της θερμοκρασίας και της ύπαρξης καπνού στο χώρο. Με την απλή προσθήκη ειδικών αισθητήρων μέτρησης τάσης, ρεύματος και ταχύτητας δικτύου, μπορούν να δημιουργηθούν κανόνες, οι οποίοι θα στηρίξουν τη διαχείριση και συντήρηση της υποδομής μετατρέποντας το περιβάλλον της σε «έξυπνο», ενώ παράλληλα θα είναι και περισσότερο αυτόνομο, διασφαλίζοντας την ασφάλεια και την ακεραιότητα συνεχούς και αδιάκοπης λειτουργίας του.

7 Επίλογος

Στις προηγούμενες ενότητες, προσπαθήσαμε να δείξουμε τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο, τα πλεονεκτήματα που μπορεί να προσφέρει η αρχιτεκτονική REST, όταν την ενσωματώνουμε σε συστήματα ΔτΠ. Η αρχιτεκτονική REST μπορεί να λειτουργήσει ως μοντέλο σχεδίασης Διεπαφών Προγραμματισμού Εφαρμογών, τις οποίες μπορούμε να χρησιμοποιήσουμε ως πρωτόκολλο μεταφοράς δεδομένων, πάντα μέσα από το πρίσμα του ΔτΠ.

Χρησιμοποιώντας ένα ήδη υπάρχον και καθιερωμένο πρωτόκολλο εφαρμογών, το πρωτόκολλο HTTP, σε συνδυασμό με μία προγραμματιστική γλώσσα όπως η Java, εφαρμόζουμε τους περιορισμούς που ορίζει η αρχιτεκτονική REST. Αν και οξύμωρο, ακολουθώντας την διαδικασία αυτή, κερδίζουμε σε απλότητα, καθώς διαχωρίζονται τα καθήκοντα του διακομιστή και του πελάτη και μπορούμε να αναπτύξουμε τον καθέναν από αυτούς ξεχωριστά, όπως και να επεκτείνουμε τις λειτουργίες τους.

Σαφώς, το σύστημα ΔτΠ που έχει υλοποιηθεί στην διπλωματική εργασία αυτή είναι ένα εξαιρετικά απλό σύστημα, με σχετικά πολύ χαμηλό κόστος. Ωστόσο οι προοπτικές επεκτασιμότητάς του είναι εξαιρετικά μεγάλες και αυτό οφείλεται κατά ένα μεγάλο βαθμό στον τρόπο με τον οποίο λειτουργεί η αρχιτεκτονική REST. Μπορούμε εύκολα να διακρίνουμε για παράδειγμα, ότι η ροή στην πλατφόρμα του Node-RED για την επικοινωνία με το API μέσω Slack, θα μπορούσε να αντικατασταθεί από μία εφαρμογή για Android ειδικά σχεδιασμένη για αυτήν τη χρήση, χωρίς να αλλάξει απολύτως τίποτα στην πλευρά του RESTful API που θα το εξυπηρετεί. Μάλιστα, δεν υπάρχει τίποτα που να μας εμποδίζει από το να θέσουμε σε λειτουργία και τις δύο αυτές ιδέες ταυτόχρονα.

Η ίδια λογική εφαρμόζεται ακόμη και σε πολύ πιο μεγάλα και πολύπλοκα συστήματα, όπου σχεδιάζονται πολλά RESTful APIs και μάλιστα τρέχουν σε υπολογιστικό νέφος. Προφανώς μπορούμε να δούμε ότι η ευελιξία που προσφέρει η ανεξαρτησία των εξαρτημάτων μπορεί να βελτιώσει τόσο την απόδοση του συστήματος, όσο και την αξιοπιστία του. Όλα τα εξαρτήματα μοιράζονται τον φόρτο εργασίας μέσα στο σύστημα, ενώ ταυτόχρονα αν κάποιο από αυτά αποτύχει να λειτουργήσει, δεν επηρεάζει σε μοιραίο ή ακόμα και σημαντικό βαθμό την λειτουργία ολόκληρου του συστήματος.

Είναι φανερό πλέον λοιπόν, ότι η αρχιτεκτονική REST μπορεί στην πλειονότητα των περιπτώσεων να βελτιώσει την επικοινωνία αλλά και τη λειτουργία ενός συστήματος, ενώ ταυτόχρονα το ΔτΠ έχει την ικανότητα να ενώσει τεχνολογίες όπως τεχνητή νοημοσύνη και υπολογιστικό νέφος σε μία αρμονική συνεργασία. Ο συνδυασμός όλων αυτών σε συστήματα με άμεση επίδραση στις καθημερινότητες των ανθρώπων είναι εύκολο να ανεβάσει τον πήχη της ποιότητας ζωής μας κατά πολύ και μάλιστα σε παγκόσμιο επίπεδο, καθώς δεν πρέπει να ξεχνάμε πως χρησιμοποιούμε το Διαδίκτυο προκειμένου να επιτύχουμε Επικοινωνία με τη βοήθεια της αρχιτεκτονικής REST.

8 Βιβλιογραφία – Αναφορές - Διαδικτυακές Πηγές

1. Jen Clarke, (2016), IBM Blog, "*What is the Internet of Things, and how does it work?*" URL: https://www.ibm.com/blogs/internet-of-things/what-is-the-iot/ , [Προσπελάστηκε 30 Αυγούστου 2020]
2. Mobile News Greece, (2018), «*Internet of Things: Τι είναι με απλά λόγια το Διαδίκτυο των Πραγμάτων*», URL: http://www.mobilenews.gr/internet-of-thingsti-einai-me-apla-logia-to-diadiktyo/ , [Προσπελάστηκε 30 Αυγούστου 2020]
3. Postscapes, "*Internet of Things (IoT): History*", URL: https://www.postscapes.com/internet-of-things-history/ , [Προσπελάστηκε 30 Αυγούστου 2020]
4. SAS, "*Internet of Things (IoT): Τι είναι | Διαδίκτυο των πραγμάτων",* URL: https://www.sas.com/el_gr/insights/big-data/internet-of-things.html , [Προσπελάστηκε 30 Αυγούστου 2020]
5. Keith D. Foote, (2016), "*A Brief History of the Internet of Things*", DATAVERSITY, URL: http://www.dataversity.net/brief-history-internet-things/, [Προσπελάστηκε 30 Αυγούστου 2020]
6. Pallavi Sethi and Smruti R. Sarangi, (2017), "*Internet of Things: Architectures, Protocols, and Applications",* Hindawi Journal of Electrical and Computer Engineering, URL: https://doi.org/10.1155/2017/9324035 , [Προσπελάστηκε 30 Αυγούστου 2020]
7. Postscapes, (2018), "*IoT Standards & protocols | 2018 Comparisons on Network, Wireless Comms, Security, Industrial",* URL: https://www.postscapes.com/internet-of-things-protocols/ , [Προσπελάστηκε 30 Αυγούστου 2020]
8. Tudor Bugnar, (2018), "*Blockchain and IoT. The future of M2M Communication - The Why and How",* Medium, URL: *https://medium.com/the-why-and-how/blockchain-iot-the-future-of-m2m-communication-1970d50acd82* , [Προσπελάστηκε 30 Αυγούστου 2020]
9. Padraig Scully, (2017), "*Understanding IoT Security (Part 1 of 3): IoT Security Architecture on the Device and Communication Layers",* DZone IoT, URL: *https://dzone.com/articles/iot-security-part-1-of-3-architecture-on-the-device-and-communication-layers* , [Προσπελάστηκε 30 Αυγούστου 2020]
10. Margaret Rouse, "*IoT security (Internet of Things security)?",* WhatIs.com , URL: *https://internetofthingsagenda.techtarget.com/definition/IoT-security-Internet-of-Things-security* , [Προσπελάστηκε 30 Αυγούστου 2020]
11. Dean Hamilton, (2018), "*Best practices for Iot security",* Network World, URL: *https://www.networkworld.com/article/3266375/internet-of-things/best-practices-for-iot-security.html* , [Προσπελάστηκε 30 Αυγούστου 2020]
12. Paromik Chakraborty, (2018), "*How to Design IoT Systems",* Electronics for You, URL: *https://electronicsforu.com/technology-trends/tech-focus/design-how-to-design-iot-systems* , [Προσπελάστηκε 30 Αυγούστου 2020]
13. TechBeacon, "*How to Design an IoT-ready infrastructure: The 4-stage architecture",* TechBeacon, URL: *https://techbeacon.com/4-stages-iot-architecture* , [Προσπελάστηκε 30 Αυγούστου 2020]
14. Eran Shlomo, (2018), "*Designing an IoT solution in 2018",* Towards Data Science, Available at: *https://towardsdatascience.com/designing-an-iot-solution-in-2018-7fe1356e63d6* , [Προσπελάστηκε 30 Αυγούστου 2020]
15. Futurice, "*7 Design principles for IoT",* URL: *https://www.futurice.com/blog/7-design-principles-for-iot/* , [Προσπελάστηκε 30 Αυγούστου 2020]

16. Maciej Kranz, (2018), "*6 Ways the Internet of Things is improving our lives",* World Economic Forum, URL: *https://www.weforum.org/agenda/2018/01/6-ways-the-internet-of-things-is-improving-our-lives/* , [Προσπελάστηκε 30 Αυγούστου 2020]
17. Saavy Relations Software, (2018), "*Application Areas of IoT!"*, Medium, URL: *https://medium.com/@saavyrelations/application-areas-of-iot-24c784223b00* , [Προσπελάστηκε 30 Αυγούστου 2020]
18. Janina Bartje, (2016), "*The top 10 IoT application areas - based on real IoT projects",* IoT Analytics, URL: *https://iot-analytics.com/top-10-iot-project-application-areas-q3-2016/* , [Προσπελάστηκε 30 Αυγούστου 2020]
19. Data Flair, (2018), "*IoT Application | The top 10 Uses of Internet of Things",* Data Flair: IoT Analytics, URL: *https://data-flair.training/blogs/iot-applications/* , [Προσπελάστηκε 30 Αυγούστου 2020]
20. Jim Poole, (2018), "*The future of AI, IoT, VR/AR is coming faster than ever. Why?",* Interconnections – The Equinix Blog, URL: *https://blog.equinix.com/blog/2018/02/14/the-future-of-ai-iot-vrar-is-coming-faster-than-ever-why/* , [Προσπελάστηκε 30 Αυγούστου 2020]
21. IoT.Business.News, (2018), "*Blockchain, IoT and 5 Other Technologies Changing Mission Critical Processes",* IoT Business News, URL: *https://iotbusinessnews.com/2018/05/13/30989-blockchain-iot-and-5-other-technologies-changing-mission-critical-processes/* , [Προσπελάστηκε 30 Αυγούστου 2018]
22. Tony Rizzo, (2017), "*2018 Technology Predictions - Mobile, Blockchain, VR, AI and Much More!",* Pulse | Linkedin, URL: *https://www.linkedin.com/pulse/2018-technology-predictions-mobile-blockchain-vr-ai-tony-rizzo* , [Προσπελάστηκε 30 Αυγούστου 2020]
23. Bruno Pedro, (2015), "*Is REST better than SOAP? Yes, in some use-cases",* Nordic APIs Blog, URL: *https://nordicapis.com/rest-better-than-soap-yes-use-cases/* , [Προσπελάστηκε 30 Αυγούστου 2020]
24. Fielding T. Roy, (2000), "Architectural Styles and the Design of Network-based Software Architectures", Dissertation, University of California, URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf, [Προσπελάστηκε 30 Αυγούστου 2020]
25. Alexandra Bowen, (2016), "*IoT is eating the world: APIs and REST",* Medium, URL: *https://medium.com/@AlexandraBowen/iot-is-eating-the-world-apis-and-rest-9e0321bc6cbf* , [Προσπελάστηκε 30 Αυγούστου 2020]
26. Laine Markku, "RESTful Web Services for the Internet of Things", Semantic Scholar, URL: https://pdfs.semanticscholar.org/560b/ed0dfc17a829d45a6a199a23eacfacf4b6cb.pdf , Department of Media Technology, Aalto University School of Science , [Προσπελάστηκε 30 Αυγούστου 2020]
27. Jim Webber, Savas Parastatidis, Ian Robinson (2010) *REST in Practice: Hypermedia and Systems Architecture*, 1st edn., Newgen North America: O'Reilly Media.
28. Bill Doerrfeld, (2015), "*The State of IoT Information Design: Why Every IoT Device Needs an API",* Nordic APIs Blog, URL: *https://nordicapis.com/the-state-of-iot-information-design-why-every-iot-device-needs-an-api/* , [Προσπελάστηκε 30 Αυγούστου 2020]
29. Wikipedia, (2018), "*SOAP",* URL: *https://en.wikipedia.org/wiki/SOAP* , [Προσπελάστηκε 30 Αυγούστου 2020]

9 Παράρτημα Α: Κλάση SensorResource.java

```java
package com.maria.rest;

import java.util.List;

import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.CacheControl;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.HttpHeaders;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;
import javax.ws.rs.core.UriInfo;

//-----------------------------------------------------------------------------
//****   This is the Sensor Resource class which belongs in the presentation layer
//****   of this rest API. This is where all the URIs and HTTP posts are defined.
//-----------------------------------------------------------------------------
@Path("ThesisAPI")
public class SensorResource {
    //---------------------------------------------------------------------
    //**** One Object for SensorRepository, DatabaseLayer and Message Classes
    static SensorService repo = new SensorService();
    static MongoDBconnection mongo = new MongoDBconnection();
    private Message mes = new Message();

    //---------------------------------------------------------------
    //**** The private 'buildResp' method is used in all other methods
    //**** in order to build a proper HTTP response including the
    //**** cache control, the status and the content-type header
    private Response buildResp(Object s, CacheControl cc, HttpHeaders head) {
        ResponseBuilder builder = Response.ok(s);
        try {
            builder.header(HttpHeaders.CONTENT_TYPE, "application/json".equals(head.getMediaType().
            toString()) ? MediaType.APPLICATION_JSON : MediaType.APPLICATION_XML);
        }catch(Exception e) {
            System.out.println("No content type, setting the default responce");
        }
        builder.cacheControl(cc);
        return builder.build();

    }

    //-----------------------------------------------------------------
    //**** Each and every one of the following methods have one unique URI
    //**** and POSTs or GETs data according the request of the client
    @GET
    @Produces({MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON})
    public Response hateoasHome(@Context UriInfo info, @Context HttpHeaders headers) {

        String uri = info.getAbsolutePathBuilder().build().toString();

        mes.setMessage("HATEOAS Home");
        mes.addLink(uri + "feedMongo", "POST Data");
        mes.addLink(uri + "sensors", "Sensors Data Information");
        mes.addLink(uri + "sensors/location/Warehouse-A", "Last Data from Warehouse-A");
        mes.addLink(uri + "sensors/location/Warehouse-A/allData", "All Data from Warehouse-A");
        mes.addLink(uri + "sensors/location/Warehouse-B", "Last Data from Warehouse-B");
        mes.addLink(uri + "sensors/location/Warehouse-B/allData", "All Data from Warehouse-B");
        mes.addLink(uri + "sensors/location/Warehouse-C", "Last Data from Warehouse-C");
        mes.addLink(uri + "sensors/location/Warehouse-C/allData", "All Data from Warehouse-C");
        mes.addLink(uri + "sensors/location/Warehouse-D", "Last Data from Warehouse-D");
        mes.addLink(uri + "sensors/location/Warehouse-D/allData", "All Data from Warehouse-D");
        mes.addLink(uri + "sensors/location/Warehouse-E", "Last Data from Warehouse-E");
```

```
68          mes.addLink(uri + "sensors/location/Warehouse-E/allData", "All Data from Warehouse-E");
69          mes.addLink(uri + "sensors/smoke/true", "Last seen smoke in any Warehouse");
70          mes.addLink(uri + "sensors/smoke/true", "All Data where there has been smoke in any
            Warehouse");
71          mes.addLink(uri + "sensors/uriQuery", "Last Data with a Custom URI Query");
72          mes.addLink(uri + "sensors/uriQuery/allData", "All Data Data with a Custom URI Query");
73
74          CacheControl cc = new CacheControl();
75          cc.setNoCache(false);
76
77          return buildResp(mes, cc, headers);
78      }
79
80      @POST
81      @Path("feedMongo")
82      @Consumes({MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON})
83      public Response postToMongo(Sensor s1, @Context UriInfo info, @Context HttpHeaders headers) {
84
85          mes.setMessage(mongo.postJson("REST-MongoDB", "SensorData", s1));
86
87          CacheControl cc = new CacheControl();
88          cc.setNoCache(true);
89          cc.setMaxAge(-1);
90
91          return buildResp(mes, cc, headers);
92
93      }
94
95      @GET
96      @Path("sensors")
97      @Consumes({MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON})
98      public Response getResponse(@Context UriInfo info, @Context HttpHeaders headers) {
99
100         mes.setMessage("General Information Regarding the Data that this RESTful API handles \n\n"
101                 + "This Data comes from various microcontroller units that \n"
102                 + "measure and detect throught a set of sensors: \n"
103                 + "temperature in Celcious \n"
104                 + "humidity percentage in the air \n"
105                 + "and smoke, if there is any.\n"
106                 + "Along with these data it also sends additional information \n"
107                 + "regarding the location of the mcu, the date and time of each \n"
108                 + "measurment and an id that has been asigned to it. \n"
109                 + "A client can perform various HTTP GET requests to this API \n"
110                 + "and receive some of these data or all of them, in JSON or/ \n"
111                 + "and XML format. \n"
112                 + "When a uri is mentioned by the HATEOAS property and it contains \n"
113                 + "in its path a value for any of the properties, it is higly \n"
114                 + "possible that this value can be changed according to the \n"
115                 + "clients needs.");
116
117         CacheControl cc = new CacheControl();
118         cc.setNoCache(true);
119
120         return buildResp(mes, cc, headers);
121
122     }
123
124     @GET
125     @Path("sensors/location/{location}")
126     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
127     public Response getLocation(@PathParam("location") String loc, @Context UriInfo info, @Context
        HttpHeaders headers) {
128         Sensor s1 = repo.giveLastDataWith("location", loc, "REST-MongoDB", "SensorData");
129         String uri = info.getAbsolutePathBuilder().build().toString();
130         s1.addLink(uri + "/allData" , "all data");
131         s1.addLink(uri + "/temperature", "Get data of this warehouse for various temperature
            values");
132         s1.addLink(uri + "/smoke/true", "Get the last data when smoke was detected in this
            warehouse");
133
134         CacheControl cc = new CacheControl();
135         cc.setNoCache(true);
```

```
136             cc.setMaxAge(3600);
137
138             return buildResp(s1, cc, headers);
139         }
140
141         @GET
142         @Path("sensors/location/{location}/allData")
143         @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
144         public Response getLocationAllData(@PathParam("location") String loc, @Context HttpHeaders
            headers) {
145             List<Sensor> list = repo.giveAllDataWith("location", loc, "REST-MongoDB", "SensorData");
146
147             CacheControl cc = new CacheControl();
148             cc.setNoCache(true);
149             cc.setMaxAge(3600);
150
151             return buildResp(list, cc, headers);
152         }
153
154         @GET
155         @Path("sensors/location/{location}/temperature")
156         @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
157         public Response getLocationTempHateoas(@Context UriInfo info, @Context HttpHeaders headers) {
158
159             mes.setMessage("Select criteria for temperature");
160
161             String uri = info.getAbsolutePathBuilder().build().toString();
162
163             mes.addLink(uri + "/less/25", "Get the last data of this warehouse with temperature up to
                25 C degrees");
164             mes.addLink(uri + "/equal/25", "Get the last data of this warehouse with temperature of 25
                C degrees");
165             mes.addLink(uri + "/greater/25", "Get the last data of this warehouse with temperature
                higher than 25 C degrees");
166
167             CacheControl cc = new CacheControl();
168             cc.setNoCache(false);
169
170             return buildResp(mes, cc, headers);
171         }
172
173         @GET
174         @Path("sensors/location/{location}/temperature/less/{c}")
175         @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
176         public Response getLocAndTempLess(@PathParam("location") String loc, @PathParam("c") String
            temp, @Context UriInfo info, @Context HttpHeaders headers) {
177
178             Sensor s1 = repo.giveLastDataWithGtEqLt(0, "temperature", temp, "location", loc,
                "REST-MongoDB", "SensorData");
179             String uri = info.getAbsolutePathBuilder().build().toString();
180             s1.addLink(uri + "/allData" , "all data");
181
182             CacheControl cc = new CacheControl();
183             cc.setNoCache(true);
184             cc.setMaxAge(3600);
185
186             return buildResp(s1, cc, headers);
187         }
188
189         @GET
190         @Path("sensors/location/{location}/temperature/less/{c}/allData")
191         @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
192         public Response getLocAndTempLessAllData(@PathParam("location") String loc, @PathParam("c")
            String temp, @Context UriInfo info, @Context HttpHeaders headers) {
193
194             List<Sensor> list = repo.giveAllDataWithGtEqLt(0, "temperature", temp, "location", loc,
                "REST-MongoDB", "SensorData");
195
196             CacheControl cc = new CacheControl();
197             cc.setNoCache(true);
198             cc.setMaxAge(3600);
```

```
199
200             return buildResp(list, cc, headers);
201         }
202
203         @GET
204         @Path("sensors/location/{location}/temperature/equal/{c}")
205         @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
206         public Response getLocAndTempEqual(@PathParam("location") String loc, @PathParam("c") String
            temp, @Context UriInfo info, @Context HttpHeaders headers) {
207
208             Sensor s1 = repo.giveLastDataWithGtEqLt(1, "temperature", temp, "location", loc,
                "REST-MongoDB", "SensorData");
209             String uri = info.getAbsolutePathBuilder().build().toString();
210             s1.addLink(uri + "/allData" , "all data");
211
212             CacheControl cc = new CacheControl();
213             cc.setNoCache(true);
214             cc.setMaxAge(3600);
215
216             return buildResp(s1, cc, headers);
217         }
218
219         @GET
220         @Path("sensors/location/{location}/temperature/equal/{c}/allData")
221         @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
222         public Response getLocAndTempEqualAllData(@PathParam("location") String loc, @PathParam("c")
            String temp, @Context UriInfo info, @Context HttpHeaders headers) {
223
224             List<Sensor> list = repo.giveAllDataWithGtEqLt(1, "temperature", temp, "location", loc,
                "REST-MongoDB", "SensorData");
225
226             CacheControl cc = new CacheControl();
227             cc.setNoCache(true);
228             cc.setMaxAge(3600);
229
230             return buildResp(list, cc, headers);
231         }
232
233         @GET
234         @Path("sensors/location/{location}/temperature/greater/{c}")
235         @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
236         public Response getLocAndTempGreater(@PathParam("location") String loc, @PathParam("c") String
            temp, @Context UriInfo info, @Context HttpHeaders headers) {
237
238             Sensor s1 = repo.giveLastDataWithGtEqLt(2, "temperature", temp, "location", loc,
                "REST-MongoDB", "SensorData");
239             String uri = info.getAbsolutePathBuilder().build().toString();
240             s1.addLink(uri + "/allData" , "all data");
241
242             CacheControl cc = new CacheControl();
243             cc.setNoCache(true);
244             cc.setMaxAge(3600);
245
246             return buildResp(s1, cc, headers);
247         }
248
249         @GET
250         @Path("sensors/location/{location}/temperature/greater/{c}/allData")
251         @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
252         public Response getLocAndTempGreaterAllData(@PathParam("location") String loc, @PathParam("c")
            String temp, @Context UriInfo info, @Context HttpHeaders headers) {
253
254             List<Sensor> list = repo.giveAllDataWithGtEqLt(2, "temperature", temp, "location", loc,
                "REST-MongoDB", "SensorData");
255
256             CacheControl cc = new CacheControl();
257             cc.setNoCache(true);
258             cc.setMaxAge(3600);
259
260             return buildResp(list, cc, headers);
261         }
262
```

```
263     @GET
264     @Path("sensors/location/{location}/smoke/{bool}")
265     @Produces({MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON})
266     public Response getLocAndSmoke(@PathParam("location") String loc, @PathParam("bool") String
        bool, @Context UriInfo info, @Context HttpHeaders headers) {
267         Sensor s1 = repo.giveLastDataWith("location", loc, "smoke", bool, "REST-MongoDB",
            "SensorData");
268         String uri = info.getAbsolutePathBuilder().build().toString();
269         s1.addLink(uri + "/allData" , "all data");
270
271         CacheControl cc = new CacheControl();
272         cc.setNoCache(true);
273         cc.setMaxAge(3600);
274
275         return buildResp(s1, cc, headers);
276     }
277
278     @GET
279     @Path("sensors/location/{location}/smoke/{bool}/allData")
280     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
281     public Response getLocAndSmokeAllData(@PathParam("location") String loc, @PathParam("bool")
        String bol, @Context HttpHeaders headers) {
282         List<Sensor> list = repo.giveAllDataWith("location", loc, "smoke", bol, "REST-MongoDB",
            "SensorData");
283
284         CacheControl cc = new CacheControl();
285         cc.setNoCache(true);
286         cc.setMaxAge(3600);
287
288         return buildResp(list, cc, headers);
289     }
290
291     @GET
292     @Path("sensors/smoke/{bool}")
293     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
294     public Response getSmoke(@PathParam("bool") String bool, @Context UriInfo info, @Context
        HttpHeaders headers) {
295         Sensor s1 = repo.giveLastDataWith("smoke", bool, "REST-MongoDB", "SensorData");
296         String uri = info.getAbsolutePathBuilder().build().toString();
297         s1.addLink(uri + "allData" , "Get all data where there has detected smoke");
298         s1.addLink( uri + "ThesisAPI/sensors/location/" + s1.getLocation() + "/allData", "Get all
            data of this Warehouse");
299
300         CacheControl cc = new CacheControl();
301         cc.setNoCache(true);
302         cc.setMaxAge(3600);
303
304         return buildResp(s1, cc, headers);
305     }
306
307     @GET
308     @Path("sensors/smoke/{bool}/allData")
309     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
310     public Response getSmokeAllData(@PathParam("bool") String bool, @Context UriInfo info, @Context
         HttpHeaders headers) {
311         List<Sensor> list = repo.giveAllDataWith("smoke", bool, "REST-MongoDB", "SensorData");
312         String uri = info.getBaseUri().toString();
313
314         for(Sensor s:list) {
315             s.addLink( uri + "ThesisAPI/sensors/location/" + s.getLocation() + "/allData", "Get
                all data of this Warehouse");
316         }
317
318         CacheControl cc = new CacheControl();
319         cc.setNoCache(true);
320         cc.setMaxAge(3600);
321
322         return buildResp(list, cc, headers);
323     }
324
325     @GET
326     @Path("sensors/uriQuery")
```

```
327     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
328     public Response getSensorsByQueryAllData(@Context UriInfo info, @Context HttpHeaders headers){
329         Sensor s1 = repo.giveLastDataByQuery(info, "REST-MongoDB", "SensorData");
330         String uri = info.getAbsolutePathBuilder().build().toString();
331         s1.addLink(uri + "/allData", "All Data");
332         uri = info.getBaseUri().toString();
333         s1.addLink( uri + "ThesisAPI/sensors/location/" + s1.getLocation() + "/allData", "Get all
            data of this Warehouse");
334
335         CacheControl cc = new CacheControl();
336         cc.setNoCache(true);
337         cc.setMaxAge(3600);
338
339         return buildResp(s1, cc, headers);
340     }
341
342     @GET
343     @Path("sensors/uriQuery/allData")
344     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
345     public Response getSensorsByQuery (@Context UriInfo info, @Context HttpHeaders headers){
346         List<Sensor> list = repo.giveAllDataByQuery(info, "REST-MongoDB", "SensorData");
347         String uri = info.getBaseUri().toString();
348
349         for(Sensor s:list) {
350             s.addLink( uri + "ThesisAPI/sensors/location/" + s.getLocation() + "/allData", "Get
                all data of this Warehouse");
351         }
352
353         CacheControl cc = new CacheControl();
354         cc.setNoCache(true);
355         cc.setMaxAge(3600);
356
357         return buildResp(list, cc, headers);
358     }
359
360 }
```

10 Παράρτημα Β: Κλάση Sensor.java

```java
package com.maria.rest;

import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;

import javax.xml.bind.annotation.XmlRootElement;

//---------------------------------------------------------
//****   This is a POJO which is the representation
//****   form for the Sensor Resource. All data that
//****   flows in either direction (Client <=> MongoDB)
//****   takes the form of this Sensor POJO with these
//****   specific variables and characteristics. The Sensor
//****   POJO belongs in the Logic Layer of this rest API.
//---------------------------------------------------------

@XmlRootElement
public class Sensor {


    //-------------------------------------------------------------------------------------------------
    //**** Date Formatters are needed in order to match the date and time types of MongoDB and Java
    private DateTimeFormatter parseform = DateTimeFormatter.ofPattern("EEE MMM dd yyyy HH:mm:ss z");
    private DateTimeFormatter form2 = DateTimeFormatter.ISO_DATE_TIME.withZone(ZoneId.of(
    "Europe/Athens"));

    //-----------------------------------
    //**** Variables of 'Sensor' Objects
    private String id;
    private String location;
    private int temperature;
    private int humidity;
    private boolean smoke;
    private String dateTime;
    private List<Link> links = new ArrayList<Link>();

    //-----------------
    //**** Constructor
    public Sensor() {}

    //-----------------------------------------------------
    //**** Getters and Setters for this Classe's Variables
    public boolean isSmoke() {
        return smoke;
    }

    public String getDateTime() {
        return dateTime;
    }

    public String getId() {
        return id;
    }

    public String getLocation() {
        return location;
    }

    public int getTemperature() {
        return temperature;
    }

    public int getHumidity() {
        return humidity;
    }

```

```
68      public List<Link> getLinks() {
69          return links;
70      }
71
72      public void setSmoke(boolean smoke) {
73          this.smoke = smoke;
74      }
75
76      public void setDateTime(String dt) {
77          this.dateTime = dt;
78      }
79
80      public void setId(String id) {
81          this.id = id;
82      }
83
84      public void setLocation(String location) {
85          this.location = location;
86      }
87
88      public void setTemperature(int temperature) {
89          this.temperature = temperature;
90      }
91
92      public void setHumidity(int humidity) {
93          this.humidity = humidity;
94      }
95
96      public void setLinks(List<Link> links) {
97          this.links = links;
98      }
99
100     //-------------------------------------------------------
101     //**** Method addLink is used in order to add as many
102     //**** as necessary HATEOAS links in a Sensor Object
103     public void addLink(String uri, String rel) {
104         Link link = new Link();
105         link.setRel(rel);
106         link.setLink(uri);
107         links.add(link);
108     }
109
110     //-------------------------------------------------------
111     //**** Override of the toString Method to set JSON format
112     @Override
113     public String toString() {
114         return "{id:\"" + id + "\", location:\"" + location + "\", temperature:\"" + temperature
115                 + "\", humidity:\"" + humidity + "\", smoke:\"" + smoke + "\", dateTime:\""
116                 + ZonedDateTime.parse(dateTime, parseform).format(form2).toString() + "\", links:"
                    + links.toString() + "}";
117     }
118
119 }
```

11 Παράρτημα Γ: Κλάση Message.java

```java
package com.maria.rest;

import java.util.ArrayList;
import java.util.List;

import javax.xml.bind.annotation.XmlRootElement;

//---------------------------------------------------
//****   This a POJO which is sometimes used instead
//****   of the main resource's POJO. It is used only
//****   to give information to the client about a
//****   specific state of the resource and maybe give
//****   some HATEOAS links with it. The Message class
//****   belongs in the Logic Layer of this rest API.
//---------------------------------------------------

@XmlRootElement
public class Message {

    //-----------------------------------
    //**** Variables of 'Message' Objects
    private String message;
    private List<Link> links = new ArrayList<Link>();

    //-----------------
    //**** Constructor
    public Message() {}

    //-----------------------------------------------------
    //**** Getters and Setters for this Classe's Variables
    public List<Link> getLink() {
        return links;
    }

    public String getMessage() {
        return message;
    }

    public void setLink(List<Link> link) {
        this.links = link;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public void addLink(String uri, String rel) {
        Link link = new Link();
        link.setRel(rel);
        link.setLink(uri);
        links.add(link);
    }

    //-----------------------------------------------------
    //**** Override of the toString Method to set JSON format
    @Override
    public String toString() {
        return "{ \"message\":" + "\"" + message + "\", links:" + links.toString() + "}";
    }

}
```

12 Παράρτημα Δ: Κλάση Link.java

```
1  package com.maria.rest;
2
3  //-------------------------------------------------------------
4  //****   'Link' Objects are used in order to add the HATEOAS
5  //****   property in some states of the Sensor Resource.
6  //****   Link class belongs in the Logic Layer of the rest API.
7  //-------------------------------------------------------------
8
9  public class Link {
10
11     //-----------------------------------------
12     //**** 'Link' Objects have two Variables
13     private String link;
14     private String rel;
15
16     //-----------------------------------------
17     //**** Constructor
18     public Link() {}
19
20     //-------------------------------------------------------
21     //**** Getters and Setters for this Classe's Variables
22     public String getLink() {
23         return link;
24     }
25
26     public String getRel() {
27         return rel;
28     }
29
30     public void setLink(String link) {
31         this.link = link;
32     }
33
34     public void setRel(String rel) {
35         this.rel = rel;
36     }
37
38     //-------------------------------------------------------
39     //**** Override of the toString Method to set JSON format
40     @Override
41     public String toString() {
42
43         return "{rel:\""+ rel + "\", link:\"" + link + "\"}";
44     }
45
46 }
```

13 Παράρτημα Ε: Κλάση SensorService.java

```
1  package com.maria.rest;
2
3  import java.time.ZoneId;
4  import java.time.format.DateTimeFormatter;
5  import java.util.ArrayList;
6  import java.util.List;
7  import java.util.Map.Entry;
8
9  import javax.ws.rs.core.UriInfo;
10
11 import com.mongodb.BasicDBObject;
12
13 //-------------------------------------------------------
14 //****   SensorService class belongs in the Logic Layer
15 //****   of this rest API and it contains some methods
16 //****   for building queries given the nature of the
17 //****   query parameters and requested data.
18 //-------------------------------------------------------
19
20 public class SensorService {
21
22     //-------------------------------------------------
23     //**** Date Formatters are needed in order to match
24     //**** the date and time types of MongoDB and Java
25     DateTimeFormatter form = DateTimeFormatter.ofPattern("EEE MMM dd yyyy HH:mm:ss z");
26     DateTimeFormatter form2 = DateTimeFormatter.ISO_DATE_TIME.withZone(ZoneId.of("Europe/Athens"));;
27     List<Sensor> results = new ArrayList<Sensor>();
28     MongoDBconnection db = new MongoDBconnection();
29
30     //-------------------------------------------------------
31     //**** Four methods for four different types of queries
32     //**** that is possible to be requested given the query
33     //**** parameters and the name of the database and
34     //**** collection in MongoDB
35
36
37     public Sensor giveLastDataWith(String key1, String value1, String dbName, String collectionName
       ) {
38
39         BasicDBObject query = new BasicDBObject();
40         query.append(key1, value1);
41
42         return db.lastData(query, dbName, collectionName);
43     }
44
45     public Sensor giveLastDataWith(String key1, String value1, String key2, String value2, String
       dbName, String collectionName) {
46
47         BasicDBObject query = new BasicDBObject();
48         query.append(key1, value1);
49         query.append(key2, value2);
50
51         return db.lastData(query, dbName, collectionName);
52     }
53
54
55     public List<Sensor> giveAllDataWith(String key1, String value1, String dbName, String
       collectionName){
56
57         BasicDBObject query = new BasicDBObject();
58         query.append(key1, value1);
59
60         return db.allData(query, dbName, collectionName);
61     }
62
63     public List<Sensor> giveAllDataWith(String key1, String value1, String key2, String value2,
       String dbName, String collectionName){
64
65         BasicDBObject query = new BasicDBObject();
66         query.append(key1, value1);
```

```
67          query.append(key2, value2);
68
69          return db.allData(query, dbName, collectionName);
70      }
71
72
73      public List<Sensor> giveAllDataWithGtEqLt(int mode, String numberKey, String numberValue,
        String dbName, String collectionName){
74
75          BasicDBObject query = new BasicDBObject();
76
77          // **** mode: 0->less or equal, 1->equal, 2->greater or equal ****
78
79          if (mode == 0) {
80              query.append(numberKey, new BasicDBObject("$lte", numberValue));
81          }
82          else if (mode == 1) {
83              query.append(numberKey, numberValue);
84          }
85          else if (mode == 2) {
86              query.append(numberKey, new BasicDBObject("$gte", numberValue));
87          }
88
89          return db.allData(query, dbName, collectionName);
90      }
91
92      public List<Sensor> giveAllDataWithGtEqLt(int mode, String numberKey, String numberValue,
        String key2, String value2, String dbName, String collectionName){
93
94          BasicDBObject query = new BasicDBObject();
95          query.append(key2, value2);
96          // **** mode: 0->less or equal, 1->equal, 2->greater or equal ****
97
98          if (mode == 0) {
99              query.append(numberKey, new BasicDBObject("$lte", numberValue));
100         }
101         else if (mode == 1) {
102             query.append(numberKey, numberValue);
103         }
104         else if (mode == 2) {
105             query.append(numberKey, new BasicDBObject("$gte", numberValue));
106         }
107
108         return db.allData(query, dbName, collectionName);
109     }
110
111
112     public Sensor giveLastDataWithGtEqLt(int mode, String numberKey, String numberValue, String
        dbName, String collectionName){
113
114         BasicDBObject query = new BasicDBObject();
115
116         // **** mode: 0->less or equal, 1->equal, 2->greater or equal ****
117
118         if (mode == 0) {
119             query.append(numberKey, new BasicDBObject("$lte", numberValue));
120         }
121         else if (mode == 1) {
122             query.append(numberKey, numberValue);
123         }
124         else if (mode == 2) {
125             query.append(numberKey, new BasicDBObject("$gte", numberValue));
126         }
127
128         return db.lastData(query, dbName, collectionName);
129     }
130
131     public Sensor giveLastDataWithGtEqLt(int mode, String numberKey, String numberValue, String
        key2, String value2, String dbName, String collectionName){
132
133         BasicDBObject query = new BasicDBObject();
134         query.append(key2, value2);
135         // **** mode: 0->less or equal, 1->equal, 2->greater or equal ****
```

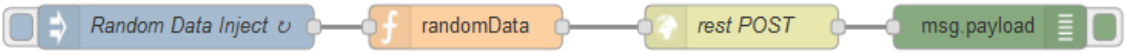
```
136
137         if (mode == 0) {
138             query.append(numberKey, new BasicDBObject("$lte", numberValue));
139         }
140         else if (mode == 1) {
141             query.append(numberKey, numberValue);
142         }
143         else if (mode == 2) {
144             query.append(numberKey, new BasicDBObject("$gte", numberValue));
145         }
146
147         return db.lastData(query, dbName, collectionName);
148     }
149
150
151
152     public List<Sensor> giveAllDataByQuery(UriInfo info, String dbName, String collectionName){
153
154         BasicDBObject query = new BasicDBObject();
155
156         for(Entry<String, List<String>> entry : info.getQueryParameters().entrySet()) {
157
158             if (entry.getValue().size()>1) {
159                 query.append(entry.getKey(), new BasicDBObject("$in", entry.getValue()));
160             } else {
161                 query.append(entry.getKey().toString(), entry.getValue().get(0).toString());
162
163             }
164
165         }
166
167         return db.allData(query, dbName, collectionName);
168     }
169
170     public Sensor giveLastDataByQuery(UriInfo info, String dbName, String collectionName){
171
172         BasicDBObject query = new BasicDBObject();
173
174         for(Entry<String, List<String>> entry : info.getQueryParameters().entrySet()) {
175
176             if (entry.getValue().size()>1) {
177                 query.append(entry.getKey(), new BasicDBObject("$in", entry.getValue()));
178             } else {
179                 query.append(entry.getKey().toString(), entry.getValue().get(0).toString());
180
181             }
182
183         }
184
185         return db.lastData(query, dbName, collectionName);
186     }
187
188 }
```

14 Παράρτημα ΣΤ: Κλάση MongoDBconnection.java

```java
package com.maria.rest;

import org.bson.Document;

import com.google.gson.Gson;
import com.mongodb.BasicDBObject;
import com.mongodb.MongoClient;
import com.mongodb.MongoClientURI;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;

import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;

//------------------------------------------------------------
//****   MongoDBconnection is the main and only class of the
//****   Data Access Layer of this rest API. It manages the
//****   JDBC with MongoDB and performs the the given queries
//****   after opening a session with MongoDB.
//------------------------------------------------------------

public class MongoDBconnection {

    //-------------------------------------------------
    //**** Date Formatters are needed in order to match
    //**** the date and time types of MongoDB and Java
    DateTimeFormatter form = DateTimeFormatter.ofPattern("EEE MMM dd yyyy HH:mm:ss z");
    DateTimeFormatter form2 = DateTimeFormatter.ISO_DATE_TIME.withZone(ZoneId.of("Europe/Athens"));;

    //-----------------------------------------
    //**** Necessary variables for establishing
    //**** a connection with MongoDB
    private static String uri = "mongodb+srv://admin:password@mongodb-jyfye.mongodb.net/test";
    private static MongoClientURI clientUri = new MongoClientURI(uri);
    private static MongoClient mongoClient = new MongoClient(clientUri);
    private MongoCollection<Document> collection;
    private static Gson gson = new Gson();
    List<Sensor> results = new ArrayList<Sensor>();

    //-----------------------------------------
    //**** Method fetchByQuery performs the given
    //**** query in a given collection of MongoDb
    private List<Sensor> fetchByQuery(BasicDBObject query, MongoCollection<Document> collection){
        if (!results.isEmpty()) {

            results.clear();
        } else {
            System.out.println("");
        }

        try {

            FindIterable<Document> docs = collection.find(query);

            for(Document doc : docs) {

                doc.append("dateTime", ZonedDateTime.parse(doc.get("dateTime").toString(), form2).
                format(form).toString());
                results.add(gson.fromJson(doc.toJson(), Sensor.class));
            }

            return results;


        } catch (Exception e) {

            results.add(gson.fromJson("{\"id\":\"Oops, something went wrong. Please try again.",
            Sensor.class));
```

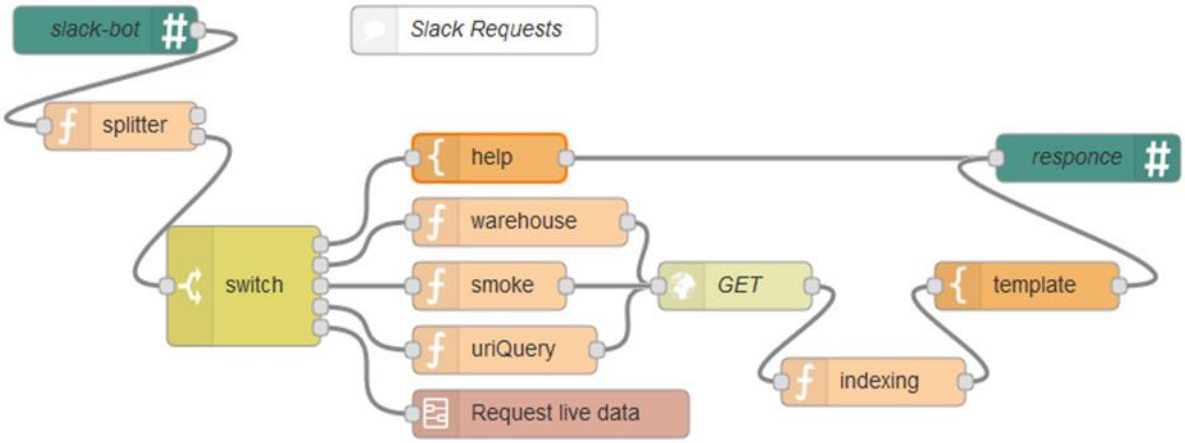
```
70              System.out.println(e.getMessage());
71
72              return results;
73          }
74
75      }
76
77      //-------------------------------------------------
78      //**** Method postJson POSTs data of type Sensor in a
79      //**** given MongoDB collection of a given database
80      public String postJson(String dbName, String collectionName, Sensor s1) {
81
82          collection = mongoClient.getDatabase(dbName).getCollection(collectionName);
83
84          try{
85              Document doc = Document.parse(s1.toString());
86              doc.remove("links");
87              collection.insertOne(doc);
88              return "Data parsing and uploading succeful!";
89          }catch(Exception e) {
90              System.out.println(e.toString());
91              System.out.println(e.getMessage());
92              return "Parsing or uploading of s1 data failed";
93          }
94
95      }
96
97      //-------------------------------------------------
98      //**** Method lastData returns the most recent entry from
99      //**** MongoDB in a Sensor Object, given a specific query
100     public Sensor lastData (BasicDBObject query, String dbName, String collectionName) {
101
102         collection = mongoClient.getDatabase(dbName).getCollection(collectionName);
103         Sensor s1 = gson.fromJson(collection.find(query).sort(new BasicDBObject("_id",-1)).first().
            toJson(), Sensor.class);
104
105         return s1;
106
107     }
108
109     //-------------------------------------------------
110     //**** Method allData returns all entries from MongoDB in
111     //**** a list of Sensor Objects , given a specific query
112     public List<Sensor> allData(BasicDBObject query, String dbName, String collectionName){
113
114         collection = mongoClient.getDatabase(dbName).getCollection(collectionName);
115
116         return fetchByQuery(query, collection);
117     }
118
119 }
```

15 Παράρτημα Z: Κώδικας Javascript από τη λειτουργία του Node-RED




```
var IDs = ["abcd1", "abcd2", "abcd3", "abcd4", "abcd5"];
var loc = ["Warehouse-A", "Warehouse-B", "Warehouse-C",
        "Warehouse-D", "Warehouse-E"];
var random = Math.floor(Math.random()*5);
msg.payload = {
   "dateTime" : new Date().toString().slice(0,24) +
            "GMT+03:00",
   "id" : IDs[random],
   "location" : loc[random],
   "temperature" : Math.floor(Math.random()*28+15),
   "humidity" : Math.floor(Math.random()*100+1),
   "smoke" : false } ;
If (Math.floor(Math.random()*100)<10){
   msg.payload.smoke = true;
}
msg.headers = {
   "Content-type" : "application/json" }
return msg;
```

Εικόνα 50: Κώδικας Javascript Προσομοίωσης Παραγωγής Μετρήσεων στο Node - RED



```
var terms = msg.payload.replace(/\s+$/, '').split(" ");
if (terms.length == 1) {
   msg.term0 = terms[0].toLowerCase().replace(/\s+/g, '');
   msg.term1 = 1;
   msg.term2 = null;
   return [null, msg];
}
else if (terms.length == 2) {
   msg.term0 = terms[0].toLowerCase().replace(/\s+/g, '');
   msg.term1 = terms[1].toLowerCase().replace(/\s+/g, '');
   msg.term2 = 1;
   return [null, msg];
}
else if (terms.length == 3) {
   msg.term0 = terms[0].toLowerCase().replace(/\s+/g, '');
   msg.term1 = terms[1].toLowerCase().replace(/\s+/g, '');
   msg.term2 = terms[2].toLowerCase().replace(/\s+/g, '');
   return [null, msg];
}
else {
   return [msg, null]; }
```

Εικόνα 51: Κώδικας Κόμβου Splitter

```
if (msg.term0 == 'uriquery'){
   msg.payload = {'path' : 'ThesisAPI/sensors/uriQuery/allData/' + msg.term1.replace(/\s+/g,
'')};
}
msg.headers = {
   'Content-type' : 'application/json'
}
return msg;
```

Εικόνα 52: Κώδικας Κόμβου Javascript για την Περίπτωση Όπου το msg.term0 Περιέχει τον Όρο "uriquery"

```
var length = msg.payload.length;
if (msg.term0.includes("warehouse")|| msg.term0.includes("smoke")) {
   msg.payload = msg.payload[length - msg.term1]
}
else if (msg.term0.includes("uriquery")){
   msg.payload = msg.payload[length - msg.term2];
}
return msg;
```

Εικόνα 53: Κώδικας Κόμβου indexing