ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Sencity: Τροποποίηση της εφαρμογής SenseCity του Πανεπιστημίου Πατρών με στόχο την αξιοποίηση και διαχείριση αισθητήρων και μετρήσεων στα πλαίσια ΙτΠ (κομμάτι διαχειριστή)

Βασίλειος Γ. Κωνσταντακόπουλος

Επιβλέποντες: **Αθανασία Αλωνιστιώτη,** Επίκουρος Καθηγήτρια
Σωκράτης Μπαρμπουνάκης, Μεταδιδακτορικός Ερευνητής
Δημήτριος Σουκάρας, Μεταδιδακτορικός Ερευνητής

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2019

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Sencity: Τροποποίηση της εφαρμογής sensecity του Πανεπιστημίου Πατρών με στόχο την αξιοποίηση και διαχείριση αισθητήρων και μετρήσεων στα πλαίσια ΙτΠ (κομμάτι διαχειριστή)

Βασίλειος Γ. Κωνσταντακόπουλος
Α.Μ.: 1115200900109

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Αθανασία Αλωνιστιώτη,** Επίκουρος Καθηγήτρια
Σωκράτης Μπαρμπουνάκης, Μεταδιδακτορικός Ερευνητής
Δημήτριος Σουκάρας, Μεταδιδακτορικός Ερευνητής

ΠΕΡΙΛΗΨΗ

Σκοπός της παρούσας πτυχιακής είναι μία μερική επέκταση του site από το Πανεπιστήμιο Πατρών, Sensecity, στην Αθήνα, με δημιουργία ενός server και επεξεργαστή δεδομένων από αισθητήρες. Για την υλοποίησή της χρησιμοποιήθηκε το πρότυπο MQTT για την επικοινωνία server-αισθητήρων σε σύνδεση τύπου IoT (Internet of Things), node js και express για τη δημιουργία του server, mongodb για τη βάση με τα δεδομένα των αισθητήρων και charts.js για την απεικόνισή τους.

Αρχικά στήθηκε, με τη βοήθεια ψευδο-αισθητήρων (dummy sensors) των οποίων το αρχείο συμπεριλαμβάνεται στον τελικό φάκελο, η δυνατότητα αποστολής δεδομένων από τους αισθητήρες, η δυνατότητα λήψης από τον server, η δυνατότητα αποστολής μηνυμάτων με τα οποία οι αισθητήρες ενεργοποιούνται/απενεργοποιούνται, ελέγχονται και καταμετρώνται. Στο δεύτερο βήμα στήθηκε η βάση δεδομένων με χρήση της πιο ελαφριάς τεχνολογίας mongodb. Η λειτουργίες της είναι η αποθήκευση, ανάκληση και εξαγωγή μέσων όρων για διαφορετικά χρονικά διαστήματα. Τρίτο βήμα ήταν η δημιουργία μίας βασικής σελίδας αναζήτησης, η οποία στηρίζεται στον τύπο αισθητήρα, σε γεωργαφικά και χρονικά στοιχεία για να επιστρέψει τις αντίστοιχες τιμές από τις διαφορετικές κατηγορίες αισθητήρων που έχουν στείλει δεδομένα στον server. Σαν επιπλέον χαρακτηριστικό, δίνεται η δυνατότητα επιστροφής στατιστικών αρχείων με τις ακραίες τιμές στο επιλεχθέν χρονικό διάστημα. Τέταρτο βήμα ήταν η προσθήκη χάρτη για την απεικόνιση των αισθητήρων και λοιπές βελτιστοποιήσεις.

Εν τέλει, υλοποιήθηκε μία σελίδα διαχειριστή (Admin Page) από όπου ο διαχειριστής, με βοήθεια χάρτη διαχειρίζεται τους αισθητήρες και τα δεδομένα τους, βλέπει αντίστοιχα γραφήματα και μπορεί να κατεβάσει αρχεία με στατιστικά.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Web Development/ Web Design/ Internet of Things

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: αισθητήρες, δίκτυο, sensecity, γράφημα, πόλη, βάση δεδομένων

ABSTRACT

The aim of this thesis is to make a web application similar to Sensecity, an application developed by the University of Patras, by creating a server and handler of data from sensors. For the purposes of such an application, MQTT was used to establish communication between server and sensors in the model of IoT (internet of Things), Node JS and Express for creating the server, MongoDB for the database holding the sensor data and Chart JS for their graphic display.

First, with the help of dummy sensors, their file included in the folder submitted, the presence and transfer of sensor data, the server's ability to receive messages, the ability to send messages to enable/disable, check and count sensors were established. In the second phase, the database was set up using lighter and more flexible database technology MongoDB. Its functions are storing, reading and exporting server data and averages of its entries. Third step as to create a search page, based on the type of sensor, geographical and time givens to return the corresponding values taken from the different categories of sensors sending data to the server. Additionally, the option to download statistical files with outlying values in each time interval. Fourth and final step was the addition of a map to display the sensors and some optimisations.

Finally, all that led to an Admin Page, where the administrator, with the help of a map, can manage sensors and their data, see charts of that data and download statistical files.

SUBJECT AREA: Web Development/ Web Design/ Internet of Things

KEYWORDS: sensors, network, sensecity, graph, chart, city, database

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

1: ΑΝΑΣΚΟΠΗΣΗ ΤΩΝ ΤΕΧΝΟΛΟΓΙΩΝ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

1.1 Διαδίκτυο των Πραγμάτων (Internet of Things):

1.1.1 Τι είναι το Internet of Things;

Το **Ίντερνετ των Πραγμάτων (Internet of Things – IoT)** απευθύνεται σε έναν τεράστιο αριθμό "πραγμάτων" που είναι συνδεδεμένα στο Ίντερνετ ώστε να μπορούν να μοιραστούν δεδομένα με άλλα "πράγματα"-εφαρμογές του IoT, συνδεδεμένες συσκευές, εργοστασιακά μηχανήματα και άλλα. Οι συνδεδεμένες στο Ίντερνετ συσκευές χρησιμοποιούν εσωτερικούς **αισθητήρες (sensors)** για να συλλέγουν δεδομένα και, σε μερικές περιπτώσεις, να λαμβάνουν δράση. Οι συνδεδεμένες στο Ίντερνετ των Πραγμάτων συσκευές και μηχανές μπορούν να βελτιώσουν τον τρόπο ζωής και εργασίας μας. Πραγματικά παραδείγματα του Ίντερνετ των Πραγμάτων συναντούμε από ένα **έξυπνο σπίτι (smart home)** που αυτόματα προσαρμόζει τη θέρμανση και το φωτισμό μέχρι ένα έξυπνο εργοστάσιο που παρακολουθεί εργοστασιακά μηχανήματα αναζητώντας προβλήματα και κάνει τις απαραίτητες προσαρμογές για την αποφυγή βλάβης.

1.1.2 Πώς λειτουργεί το Internet of Things και ποιά η σημασία του;

Στις συζητήσεις γύρω από το IoT, έχει αναγνωριστεί ότι οι τεχνολογίες **ανάλυσης δεδομένων (analytics)** είναι ζωτικής σημασίας για τη μετατροπή αυτής της «πλημμύρας» **ροής δεδομένων (streaming data)** σε κατατοπιστική και χρήσιμη γνώση. Αλλά πώς αναλύουμε τα δεδομένα καθώς «ρέουν» ασταμάτητα μέσα από τους αισθητήρες και τις συσκευές; Πώς αυτή η διαδικασία διαφέρει από τις άλλες κοινές μεθόδους ανάλυσης που υπάρχουν σήμερα;

Στην παραδοσιακή ανάλυση, τα δεδομένα αποθηκεύονται και μετά αναλύονται. Ωστόσο, στην περίπτωση των δεδομένων συνεχούς ροής όπως αυτά του IoT, τα μοντέλα και οι αλγόριθμοι είναι αυτοί που αποθηκεύονται και τα δεδομένα περνούν μέσα από αυτά για ανάλυση. Αυτό το είδος της ανάλυσης καθιστά δυνατό τον εντοπισμό και την εξέταση μοτίβων καθώς τα δεδομένα δημιουργούνται, σε πραγματικό χρόνο.

Έτσι, πριν αποθηκευτούν τα δεδομένα, σε οποιοδήποτε χώρο αποθήκευσης, υπόκεινται σε επεξεργασία . Έπειτα, χρησιμοποιούνται analytics ώστε να αποκρυπτογραφηθούν τα δεδομένα, ενώ όλοι οι συσκευές συνεχίζουν να εκπέμπουν και να λαμβάνουν δεδομένα.

Με τεχνικές advanced analytics, τα data stream analytics μπορούν να πάνε πέρα από την απλή παρακολούθηση των υπαρχουσών συνθηκών και την αξιολόγηση των κατώτατων ορίων στην πρόβλεψη μελλοντικών σεναρίων και στην εξέταση πολύπλοκων ερωτημάτων.

Για να εκτιμηθεί το μέλλον με τη χρήση αυτών των ροών δεδομένων, απαιτούνται τεχνολογίες υψηλής απόδοσης που μπορούν να προσδιορίζουν μοτίβα στα δεδομένα τη στιγμή που αυτά δημιουργούνται. Μόλις ένα μοτίβο αναγνωρίζεται, μετρήσεις ενσωματωμένες στη ροή δεδομένων, οδηγούν στην αυτόματη προσαρμογή των συνδεδεμένων συστημάτων ή δημιουργούν ειδοποιήσεις για άμεσες δράσεις και λήψη καλύτερων αποφάσεων.

Ουσιαστικά, αυτό σημαίνει ότι μπορούμε να προχωρήσουμε πέρα από την απλή παρακολούθηση συνθηκών και ορίων στην εκτίμηση πιθανών μελλοντικών γεγονότων και στον προγραμματισμό τους για αμέτρητα σενάρια.

1.1.3 Ιστορία του Ίντερνετ των Πραγμάτων

Η ιδέα ενός δικτύου από έξυπνες συσκευές πρωτοεμφανίστηκε το 1982, με έναν τροποποιημένο αυτόματο πωλητή Coca Cola στο Πανεπιστήμιο Κάρνεγκι Μέλον, στο Πίτσμπουργκ, που έγινε η πρώτη συνδεδεμένη στο Ίντερνετ συσκευή, ικανή να αναφέρει το περιεχόμενό της και αν τα ποτά που τελευταία είχαν φορτωθεί ήταν κρύα ή όχι. Το 1991, το Μαρκ Βάιζερ με την έρευνά του “Ο Υπολογιστής του 21ου Αιώνα”, όπως επίσης ακαδημαϊκοί χώροι όπως η UbiComp και PerComp γέννησαν το σύγχρονο όραμα του Ίντερνετ των Πραγμάτων. Το 1994, ο Ρέζα Ράτζι περιέγραψε την ιδέα στο IEEE Spectrum ως “κινούμενα μικρά πακέτα δεδομένων σε ένα μεγάλο σύνολο από κόμβους, ώστε να ενσωματώσει και να αυτοματοποιήσει τα πάντα από οικιακές συσκευές έως ολόκληρα εργοστάσια”. Ανάμεσα στο 1993 και το 1997, πολλές εταιρίες πρότειναν λύσεις όπως η Microsoft με το at Work ή η Novell με το NEST. Το πεδίο πήρε κι άλλη φόρα όταν ο Bill Joy οραματίστηκε επικοινωνία μηχανής με μηχανή ως μέρος του πλαισίου “Six Webs”, που παρουσίασε στο Οικονομικό Φόρουμ στο Νταβός το 1999.
Ο όρος “Ίντερνετ των Πραγμάτων”, πιθανότατα προέρχεται από τον Κέβιν Άστον της Procter & Gamble και αργότερα του Auto-Id Center του MIT, το 1999. Σε αυτό το σημείο, έβλεπε την **ταυτοποίηση μέσω ραδιοσυχνοτήτων (radio-frequency identification) (RFID)** ως μείζονος σημασίας για το Ίντερνετ των Πραγμάτων, που θα επέτρεπε στους υπολογιστές να διαχειρίζονται όλα τα ανεξάρτητα πράγματα.
Αν ορίσμουμε το Ίντερνετ των Πραγμάτων ως “απλώς το σημείο στο χρόνο όπου περισσότερα 'πράγματα ή αντικείμενα' ήταν συνδεδεμένα στο Ίντερνετ από ότι άνθρωποι”, η Cisco Systems εκτίμησε ότι το ΙτΠ “γεννήθηκε” ανάμεσα στο 2008 και το 2009, με το λόγο πράγματα/άνθρωποι να ανεβαίνει από 0.08 το 2003 σε 1.84 το 2010.

1.2 Ίντερνετ των Πραγμάτων με MQTT

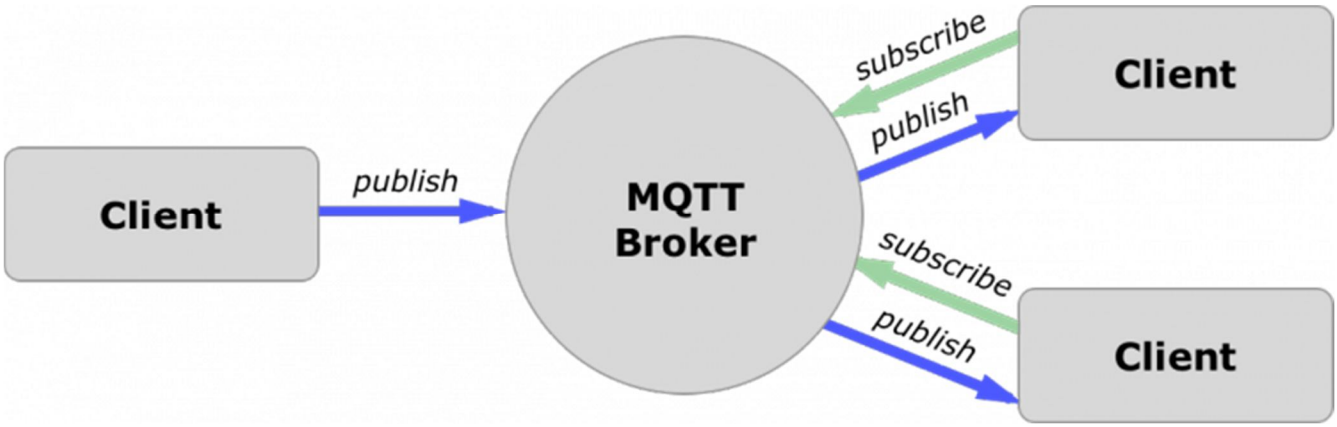
1.2.1 Τι είναι το MQTT;

MQTT ή Message Queueing Telemetry Transport είναι ένα ελαφρύ συμπαγές πρωτόκολλο ανταλλαγής μηνυμάτων για την μεταφορά δεδομένων σε απομακρυσμένες τοποθεσίες όπου απαιτείται "αποτύπωμα μικρού κώδικα" ή το εύρος ζώνης του δικτύου είναι περιορισμένο. Αυτά τα πλεονεκτήματα επιτρέπουν την εφαρμογή αυτού του πρωτοκόλλου στα συστήματα M2M (Machine to Machine)(μηχανή-με μηχανή) και IIoT (Industrial Internet of Things)(Βιομηχανικό ΙτΠ).
Εφευρέθηκε από τον Δρ. Άντυ Στάνφορντ-Κλαρκ της IBM και τον Άρλεν Νίππερ της Arcom (πλέον Eurotech), το 1999. Είναι ένα πρωτόκολλο αποστολής μηνυμάτων με **δημοσίευση/συνδρομή (publish/subscribe)**, εξαιρετικά απλό και ελαφρύ σχεδιασμένο για περιορισμένες συσκευές και χαμηλό **εύρος ζώνης (bandwidth)**, υψηλής συχνότητας ή αναξιόπιστα δίκτυα. Οι αρχές του σχεδιασμού είναι να ελαχιστοποιείται το απαιτούμενο

εύρος ζώνης του δικτύου και τις απαιτήσεις από τη συσκευή παράλληλα επιχειρείται να διασφαλιστεί η αξιοπιστία και σε κάποιο βαθμό η πιθανότητα παραλαβής. Αυτές οι αρχές εν τέλει σχηματίζουν τον αναδυόμενο **M2M** ή "Ίντερνετ των Πραγμάτων" κόσμο συνδεδεμένων μηχανών και για τις εφαρμογές κινητής τηλεφωνίας όπου το εύρος ζώνης και η ζωή μπαταρίας είναι πολύ ικανοποιητικά. Το MQTT υποστηρίζει SSL.
Υπάρχει επίσης μια παραλλαγή του πρωτοκόλλου MQTT-SN (MQTT για τα δίκτυα αισθητήρων), γνωστή ως MQTT-S, η οποία έχει σχεδιαστεί για τις ενσωματωμένες ασύρματες συσκευές χωρίς υποστήριξη δικτύων TCP / IP, για παράδειγμα ZigBee.

1.2.2 Πώς λειτουργεί το MQTT;

Όπως κάθε άλλο πρωτόκολο του Ίντερνετ, το MQTT βασίζεται σε **πελάτες (clients)** και έναν **διακομιστή (server)**. Ο διακομιστής είναι υπεύθυνος για να διαχειρίζεται τα αιτήματα του πελάτη για παραλαβή ή αποστολή δεδομένων με τους άλλους πελάτες. Ο διακομιστής του MQTT αποκαλείται **"μεσίτης" (broker)** και οι πελάτες είναι οι συνδεδεμένες συσκευές. Συνεπώς:



Σχήμα 1: Μοντέλο Επικοινωνίας MQTT

- Όταν μία συσκευή (πελάτης) θέλει να στείλει δεδομένα στο μεσίτη, η διαδικασία ονομάζεται "δημοσίευση".
- Όταν μία συσκευή (πελάτης) θέλει να λαμβάνει δεδομένα από το μεσίτη, η διαδικασία ονομάζεται "συνδρομή".

Επιπλέον, οι πελάτες δημοσιεύουν και εγγράφονται σε **θέματα (topics).** Συνεπώς, ο μεσίτης εδώ είναι αυτός που διαχειρίζεται τις δημοσιεύσεις/συνδρομές στα θέματα αυτά. Τα θέματα είναι χαρακτήρες με κωδικοποίηση UTF-8. Η ιεραρχία των θεμάτων έχει μορφή δέντρου, διευκολύνοντας έτσι την οργάνωση και την πρόσβαση στα δεδομένα. Τα θέματα μπορούν να αποτελούνται από ένα ή περισσότερα επίπεδα που χωρίζονται με το σύμβολο «/».

1.2.3 Χαρακτηριστικά του πρωτοκόλλου MQTT

Βασικά χαρακτηριστικά του πρωτοκόλλου MQTT:

- Ασύγχρονο πρωτόκολλο
- Συμπαγή μηνύματα
- Λειτουργία σε συνθήκες ασταθούς σύνδεσης της γραμμής μετάδοσης των δεδομένων
- Υποστήριξη διαφόρων επιπέδων ποιότητας υπηρεσιών (QoS)
- Εύκολη ενσωμάτωση νέων συσκευών

Στο επίπεδο της εφαρμογής, το πρωτόκολλο MQTT λειτουργεί πάνω από το πρωτόκολλο TCP / IP και χρησιμοποιεί την θύρα 1883 (8883 εάν συνδέεται μέσω SSL) ως προεπιλογή.
Στο πρωτόκολλο MQTT, η ανταλλαγή των μηνυμάτων πραγματοποιείται μεταξύ του Client, που μπορεί να είναι Publisher ή Subscriber των μηνυμάτων, και Broker των μηνυμάτων (π.χ. Mosquitto MQTT).
Ο Publisher στέλνει τα δεδομένα σε Broker MQTT ορίζοντας ένα συγκεκριμένο θέμα στο μήνυμα. Οι Subscribers μπορούν να λαμβάνουν διάφορα δεδομένα από πολλούς Publishers, ανάλογα με τη συνδρομή σε αντίστοιχα θέματα.
Οι συσκευές MQTT χρησιμοποιούν ορισμένους τύπους μηνυμάτων για την επικοινωνία τους με τον Broker. Εδώ είναι οι βασικοί τύποι:

- Connect - δημιουργία σύνδεσης με το Broker μηνυμάτων
- Disconnect - διακοπή σύνδεσης με το Broker μηνυμάτων
- Publish - δημοσίευση δεδομένων σχετικά με κάποιο θέμα μέσα στο Broker μηνυμάτων
- Subscribe - Εγγραφή σε ένα θέμα στο Broker μηνυμάτων
- Unsubscribe - διαγραφή του θέματος

1.3 Υλοποίηση του Ίντερνετ των Πραγμάτων στην εργασία

1.3.1 Λειτουργία πελάτη-διακομιστή (πελάτη)

Βοηθητικό ρόλο στην εργασία αυτή, γι' αυτό και συμπεριλαμβάνεται το αρχείο τους, είχαν οι dummy αισθητήρες που χρησιμοποιήθηκαν για να ελεγχθεί αφ’ ενός η επιτυχία της υλοποίησης του Ίντερνετ των Πραγμάτων, για να προσομοιώσει αφ’ ετέρου τη λειτουργία της εφαρμογής με πραγματικούς αισθητήρες. Ο αισθητήρας συνδέεται στον broker, εγγράφεται στα απαιτούμενα θέματα ώστε να μπορεί να ενημερώνεται όταν πρέπει να κλείσει/ανοίξει ή όταν γίνεται κάποια σημαντική αλλαγή στο δίκτυο και αρχίζει να δημοσιεύει δεδομένα στο αντίστοιχο θέμα με τον τύπο αισθητήρα που είναι.
Στην απέναντι πλευρά, ο διακομιστής συνδέεται στο ΙτΠ μέσω του broker, εγγράφεται στα θέματα δεδομένων για όλους τους τύπους αισθητήρων και ακούει. Όταν κάποιο μήνυμα, αναγνωρίζει το θέμα στο οποίο το άκουσε, σπάει το μήνυμα για να ξεχωρίσει τις πληροφορίες και το στέλνει στη βάση δεδομένων. Ο διακομιστής είναι υπεύθυνος να ξεχωρίσει

αν πρόκειται για νέο αισθητήρα, πληροφορία που παίρνει από το θέμα στο οποίο είχε φτάσει το μήνυμα. Σε περίπτωση που δε θέλουμε να λαμβάνουμε δεδομένα από έναν αισθητήρα, ο διακομιστής στέλνει μήνυμα στο θέμα “SensorControl”, ώστε ο αισθητήρας να αναγνωρίσει ότι πρέπει να σταματήσει να δημοσιεύει τιμές. Ομοίως σε περίπτωση που θέλουμε να ενεργοποιήσουμε ξανά έναν αισθητήρα.
Να σημειωθεί πως στα πλαίσια του MQTT, ο εδώ αποκαλούμενος διακομιστής είναι ένας ακόμη πελάτης. Για να τον ξεχωρίζουμε όμως από τους αισθητήρες και καθώς είναι ο κορμός της εν λόγω εφαρμογής, όπου γίνονται όλες οι διαδικασίες και θεωρητικά σε μία πραγματική κατάσταση θα “έτρεχε” διαρκώς, στο εξής θα τον αποκαλούμε διακομιστή.

1.3 MongoDB

1.3.1 Τι είναι το MongoDB και ιστορικά στοιχεία

Το MongoDB είναι ένα **διαπλατφορμικό (cross-platform)**, **αρχειοστραφές (document-oriented)** πρόγραμμα βάσεων δεδομένων. Κατηγοριοποιείται ως NoSQL πρόγραμμα, δηλαδή πρόγραμμα βάσεων δεδομένων που παρέχει ένα μηχανισμό αποθήκευσης και ανάκτησης δεδομένων σε διαφορετικό από το μοντέλο του σχεσιακού μοντέλου σε πίνακες της μορφής SQL. Χρησιμοποιεί αρχεία της μορφής JSON με **σχήμα(schema)**. Η MongoDB αναπτύχθηκε από τη MongoDB Inc. και έχει άδεια υπό την Server Side Public License (SSPL).
Η εταιρία λογισμικού 10gen software ξεκίνησε να υλοποιεί τη MongoDb το 2007 ως ένα συστατικό μίας “**πλατφόρμας ως υπηρεσία**” **(Platform as a Service - PaaS)**. Το 2009, η εταιρία άλλαξε στο μοντέλο ανοιχτού κώδικα, με την εταιρία να προσφέρει εμπορική υποστήριξη και σε άλλες υπηρεσίες. Το 2013, η 10gen άλλαξε το όνομά της σε MongoDB Inc.

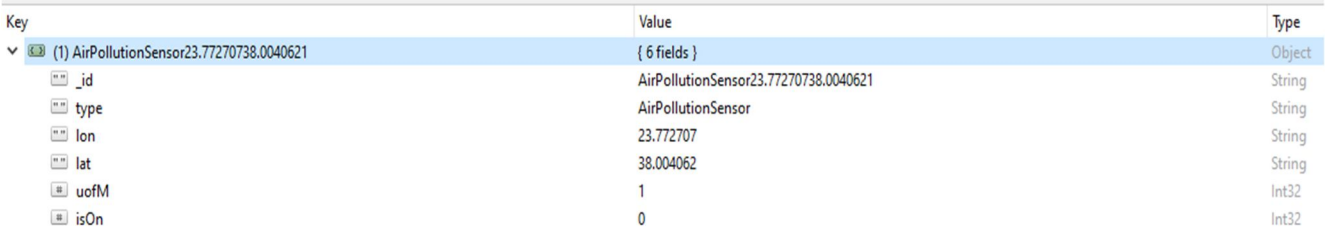
1.3.2 Η χρήση της MongoDB στο πρόγραμμά μας

Για την υλοποίηση της βάσης δεδομένων του προγράμματος προτιμήθηκε η MongoDB για τη δυνατότητα εφαρμογής της σε διαφορετικές πλατφόρμες και επειδή ως βάση δεδομένων είναι πολύ πιο ελαφριά από άλλα προγράμματα διαχείρησης βάσεων τύπου SQL. Επειδή αποφεύγονται οι δεσμεύσεις ενός σχήματος, αλλά τα δεδομένα αποθηκεύονται σε επιμέρους αρχεία, είναι πολύ πιο ευέλικτη και άμεση, χαρακτηριστικά απαραίτητα για τη λειτουργία μίας Internet of Things εφαρμογής, όπου μπορεί να εισέρχονται διαφορετικών ειδών δεδομένα τα οποία δε χρειάζόταν και δεν έχουν ληφθεί υπ'όψιν κατά την υλοποίηση, συνεπώς σε μία εφαρμογή όπως η παρούσα, στο μέλλον θα χρειαστεί πολύ λιγότερες έως καθόλου διορθώσεις η βάση. Επίσης, υποστηρίζει Javascript και τα αρχεία που επιστρέφει είναι της μορφής JSON, συνεπώς δουλεύει σε αρμονία με το Node JS, επίσης τεχνολογία που έχει χρησιμοποιηθεί για την εφαρμογή.
Πώς όμως χρησιμοποιείται η MongoDB στην εφαρμογή; Αρχικά οι αισθητήρες δηλώνουν την ένταξή τους στο διαδίκτυο, όπως έχει προαναφερθεί. Άμεσα, ο “διακομιστής” αποθηκεύει το id τους στη βάση δεδομένων, στη συλλογή “Sensors”. Η πληροφορία που αποθηκεύεται είναι της μορφής:

{

_id το “όνομα” του αισθητήρα
type ο τύπος δεδομένων που αποστέλλει ο συγκεκριμένος αισθητήρας
lon το γεωμετρικό μήκος της θέσης του αισθητήρα
lat το γεωμετρικό πλάτος της θέσης του αισθητήρα
uofM η μονάδα μέτρησης του δεδομένου που αποστέλλει ο αισθητήρας
isOn αν ο αισθητήρας είναι ενεργός ή ανενεργός

}



Εικόνα 1: Παράδειγμα καταχώρησης αισθητήρα στη βάση

Από εκεί, μπορεί να τους ανακαλέσει για να τους παρουσιάσει στο χάρτη στο **μονοπάτι (path) “**/sensors”, το οποίο μάλιστα χρησιμοποιείται και ως index, δηλαδή ως αρχική σελίδα της εφαρμογής. Από εκεί, αν ο χρήστης απενεργοποιήσει/ενεργοποιήσει έναν αισθητήρα, ενημερώνεται και η βάση για την αλλαγή αυτήν.
Παράλληλα με τη δημιουργία/ένταξη νέων αισθητήρων, οι ήδη υπάρχοντες στέλνουν δεδομένα στο ΙτΠ, τα οποία αφού “ακούσει” ο “διακομιστής”, τα περνάει στη βάση δεδομένων, στη συλλογή “Entries”. Από εκεί, μπορεί να τα ανακαλέσει είτε χωρίς καμία συνθήκη στο path “/entries”, είτε με παράμετρο το id του αισθητήρα που έστειλε την κάθε πληροφορία (/entries?search=<sensorId>), είτε, στο μονοπάτι “/entries/search”, όπου δίνεται η δυνατότητα στον διαχειριστή να αναζητήσει στοιχεία από τη βάση, ομαδοποιημένα σε μορφή διαγράμματος μέσων όρων ανά χρονικό διάστημα, ανάλογα με τον τύπο δεδομένων, την τοποθεσία του αισθητήρα που τα έστειλε, την ημερομηνία που τα έστειλε ο αισθητήρας και το επιθυμητό διάστημα ανάμεσα στις μετρήσεις.

1.4 Javascript

1.4.1 Τι είναι η JavaScript

Η **JavaScript** (**JS**) είναι διερμηνευμένη γλώσσα προγραμματισμού για ηλεκτρονικούς υπολογιστές. Αρχικά αποτέλεσε μέρος της υλοποίησης των φυλλομετρητών Ιστού, ώστε τα **σενάρια από την πλευρά του πελάτη (client-side scripts)** να μπορούν να επικοινωνούν με τον χρήστη, να ανταλλάσσουν δεδομένα ασύγχρονα και να αλλάζουν δυναμικά το περιεχόμενο του εγγράφου που εμφανίζεται.
Η JavaScript είναι μια γλώσσα **σεναρίων που βασίζεται στα πρωτότυπα (prototype-based)**, είναι δυναμική, με ασθενείς τύπους και έχει συναρτήσεις ως αντικείμενα πρώτης

τάξης. Η σύνταξή της είναι επηρεασμένη από τη C. Η JavaScript αντιγράφει πολλά ονόματα και συμβάσεις ονοματοδοσίας από τη Java, αλλά γενικά οι δύο αυτές γλώσσες δε σχετίζονται και έχουν πολύ διαφορετική σημασιολογία. Οι βασικές αρχές σχεδιασμού της JavaScript προέρχονται από τις γλώσσες προγραμματισμού Self και Scheme. Είναι γλώσσα βασισμένη σε **διαφορετικά προγραμματιστικά παραδείγματα (multi-paradigm)**, υποστηρίζοντας αντικειμενοστρεφές, προστακτικό και συναρτησιακό στυλ προγραμματισμού.
Η JavaScript χρησιμοποιείται και σε εφαρμογές εκτός ιστοσελίδων — τέτοια παραδείγματα είναι τα έγγραφα PDF, οι **εξειδικευμένοι φυλλομετρητές (site-specific browsers)** και οι μικρές **εφαρμογές της επιφάνειας εργασίας (desktop widgets)**. Οι νεότερες εικονικές μηχανές και πλαίσια ανάπτυξης για JavaScript (όπως το Node.js) έχουν επίσης κάνει τη JavaScript πιο δημοφιλή για την ανάπτυξη εφαρμογών Ιστού στην πλευρά του διακομιστή (server-side).
Το πρότυπο της γλώσσας κατά τον οργανισμό τυποποίησης ECMA ονομάζεται **ECMAscript**.

1.4.2 Ιστορία

Η γλώσσα προγραμματισμού JavaScript δημιουργήθηκε αρχικά από τον Brendan Eich της εταιρείας Netscape με την επωνυμία **Mocha**. Αργότερα, η Mocha μετονομάστηκε σε **LiveScript**, και τελικά σε **JavaScript**, κυρίως επειδή η ανάπτυξή της επηρεάστηκε περισσότερο από τη γλώσσα προγραμματισμού Java. LiveScript ήταν το επίσημο όνομα της γλώσσας όταν για πρώτη φορά κυκλοφόρησε στην αγορά σε **βήτα (beta)** εκδόσεις με το πρόγραμμα περιήγησης στο Web, Netscape Navigator εκδοχή 2.0 τον Σεπτέμβριο του 1995. Η LiveScript μετονομάστηκε σε JavaScript σε μια κοινή ανακοίνωση με την εταιρεία Sun Microsystems στις 4 Δεκεμβρίου, 1995, όταν επεκτάθηκε στην έκδοση του προγράμματος περιήγησης στο Web, Netscape εκδοχή 2.0B3.
Η **JavaScript** απέκτησε μεγάλη επιτυχία ως γλώσσα στην πλευρά του πελάτη (client-side) για εκτέλεση κώδικα σε ιστοσελίδες, και περιλήφθηκε σε διάφορα προγράμματα περιήγησης στο Web. Κατά συνέπεια, η εταιρεία Microsoft ονόμασε την εφάρμογή της σε JScript για να αποφύγει δύσκολα θέματα εμπορικών σημάτων. JScript πρόσθεσε νέους μεθόδους για να διορθώσει τα Y2K-προβλήματα στην JavaScript, οι οποίοι βασίστηκαν στην **"java.util.Date"** κλάση της Java. Η JScript περιλήφθηκε στο πρόγραμμα Internet Explorer 3.0, το οποίο κυκλοφόρησε τον Αύγουστο του 1996.
Τον Νοέμβριο του 1996, η Netscape ανακοίνωσε ότι είχε υποβάλει τη γλώσσα JavaScript στο **Ecma International** (*μια οργάνωση της τυποποίησης των γλωσσών προγραμματισμού*) για εξέταση ως βιομηχανικό πρότυπο, και στη συνέχεια το έργο είχε ως αποτέλεσμα την τυποποιημένη μορφή που ονομάζεται "ECMASript".
Η JavaScript έχει γίνει μία από τις πιο δημοφιλείς γλώσσες προγραμματισμού ηλεκτρονικών υπολογιστών στον **Παγκόσμιο Ιστό (Web)**. Αρχικά, όμως, πολλοί επαγγελματίες προγραμματιστές υποτίμησαν τη γλώσσα διότι το κοινό της ήταν ερασιτέχνες συγγραφείς ιστοσελίδων και όχι επαγγελματίες προγραμματιστές (μαζί με άλλους λόγους). Με τη χρήση της τεχνολογίας Ajax, η JavaScript γλώσσα επέστρεψε στο προσκήνιο. Το αποτέλεσμα ήταν ένα καινοτόμο αντίκτυπο στην εξάπλωση των πλαισίων και των βιβλιοθηκών,

τη βελτίωση προγραμματισμού με JavaScript, καθώς και αυξημένη χρήση της JavaScript έξω από τα προγράμματα περιήγησης στο Web.
Τον Ιανουάριο του 2009, το έργο CommonJS ιδρύθηκε με στόχο τον καθορισμό ενός κοινού προτύπου βιβλιοθήκης κυρίως για την ανάπτυξη της JavaScript έξω από το πρόγραμμα περιήγησης και μέσα σε άλλες τεχνολογίες (π.χ. Server-side).

1.4.3 Μοντέλο εκτέλεσης

Η αρχική έκδοση της JavaScript βασίστηκε στη σύνταξη στη γλώσσα προγραμματισμού C, αν και έχει εξελιχθεί, ενσωματώνοντας πια χαρακτηριστικά από νεότερες γλώσσες.
Αρχικά χρησιμοποιήθηκε για προγραμματισμό από την πλευρά του **πελάτη (client)**, που ήταν ο **φυλλομετρητής (browser)** του χρήστη, και χαρακτηρίστηκε σαν *client-side γλώσσα προγραμματισμού*. Αυτό σημαίνει ότι η επεξεργασία του κώδικα Javascript και η παραγωγή του τελικού περιεχομένου HTML δεν πραγματοποιείται στο διακομιστή, αλλά στο πρόγραμμα περιήγησης των επισκεπτών, ενώ μπορεί να ενσωματωθεί σε στατικές σελίδες HTML. Αντίθετα, άλλες γλώσσες όπως η PHP εκτελούνται στο διακομιστή (*server-side γλώσσες προγραμματισμού*).
Παρά την ευρεία χρήση της Javascript για συγγραφή προγραμμάτων σε περιβάλλον φυλλομετρητή, από την αρχή χρησιμοποιήθηκε και για τη συγγραφή κώδικα από την πλευρά του διακομιστή, από την ίδια τη Netscape στο προϊόν LiveWire, με μικρή επιτυχία. Η χρήση της Javascript στο διακομιστή εμφανίζεται πάλι σήμερα, με τη διάδοση του Node.js, ενός μοντέλου προγραμματισμού βασισμένο στα γεγονότα (events).

1.4.4 Γιατί χρησιμοποιείται η JavaScript στην εφαρμογή

Στη Javascript, ο κώδικας εκτελείται από τον επεξεργαστή του πελάτη, αντί του εξυπηρετητή, εξοικονομώντας έτσι εύρος ζώνης και περιορίζοντας την υπερφόρτωση του εξυπηρετητή. Όπως μάλιστα έεχι προαναφερθεί, ο “εξυπηρετητής” της εφαρμογής είναι στην πραγματικότητα πελάτης, διοτί ο βασικότερος σκοπός της εργασίας είναι η υλοποίηση μίας εφαρμογής στα πλαίσια του Ίντερνετ των Πραγμάτων, όπου όλοι είναι πελάτες.
Η Javascript έχει εκτεταμένη λειτουργικότητα για ιστοσελίδες, γιατί οι προσθήκες τρίτων βοηθούν τους προγραμματιστές να γράψουν αποσπάσματα κώδικα, τα οποία μπορεί να χρησιμοποιηθούν στις ιστοσελίδες, όπου χρειάζεται. Αυτό βοήθησε στην εφαρμογή επειδή σε συνδυασμό με το node js, χρησιμοποιήθηκαν jade **templates(πρότυπα)** και scripts τα οποία επαναχρησιμοποιήθηκαν από διαφορετικά paths.
Τέλος, η υλοποίησή της είναι απλή, παρ'όλο που μπορεί να χρησιμοποιηθεί και για πιο “βαριές” εφαρμογές, όπως εν προκειμένω μία εφαρμογή τύπου server, παρέχει συντομεύσεις για τον προγραμματιστή στο προγραμματιστικό στάδιο αλλά και στο testing και οι χρήστες του ιστοτόπου δε χρειάζονται ειδικό λογισμικό και λήψεις προγραμμάτων για να δουν τη JavaScript, οπότε κάθε χρήστης έχει την ίδια εμπειρία.

1.5 Node JS

1.5.1 Τι είναι το Node JS

Το **Node.js** είναι μια πλατφόρμα ανάπτυξης λογισμικού (κυρίως διακομιστών) χτισμένη σε περιβάλλον Javascript. Στόχος του Node είναι να παρέχει ένα εύκολο τρόπο δημιουργίας κλιμακωτών διαδικτυακών εφαρμογών. Σε αντίθεση από τα περισσότερα σύγχρονα περιβάλλοντα ανάπτυξης εφαρμογών δικτύων μία διεργασία *node* δεν στηρίζεται στην πολυνηματικότητα αλλά σε ένα μοντέλο ασύγχρονης επικοινωνίας εισόδου/εξόδου.

1.5.2 Ιστορία

Το Node.js δημιουργήθηκε από τον Ryan Dahl το 2009. Η δημιουργία και η συντήρηση του έργου χορηγήθηκε από την εταιρία Joyent. Η ιδέα για την ανάπτυξη του *node* προήλθε από την ανάγκη του Ryan Dahl να βρει τον πιο αποδοτικό τρόπο να ενημερώνει τον χρήστη σε πραγματικό χρόνο για την κατάσταση ενός αρχείου που ανέβαζε στο διαδίκτυο. Επίσης επηρεάστηκε από το Mongrel του Zed Shaw. Επιπροσθέτως μετά από αποτυχημένα έργα σε C, Lua, Haskell η κυκλοφορία της **μηχανής V8 (V8 JavaScript Engine)** της Google τον ώθησε να ασχοληθεί με την Javascript.

1.5.3 Χαρακτηριστικά

Το *Node* χαρακτηρίζεται από την έμφαση στην ασύγχρονη επικοινωνία μεταξύ των υπολογιστικών πόρων. Αυτό επιτυγχάνεται με την χρήση **συμβάντων (*events*)** που προσφέρει η Javascript και ονομάζονται callbacks. Για παράδειγμα όταν ένας περιηγητής ιστού φορτώσει πλήρως ένα αρχείο, ένας χρήστης πατάει κάποιο κουμπί, ολοκληρώνεται ένα αίτημα AJAX, τα συμβάντα αυτά πυροδοτούν ένα συγκεκριμένο *callback*. Αυτό με την σειρά του επιτρέπει την ροή του κώδικα χωρίς να αφήνει ανενεργό τον επεξεργαστή προκειμένου να εκτελεστεί μια λειτουργία, όπως μια επιτυχής ανάγνωση αρχείου από τον δίσκο.

1.6 Chart.js

Το Chart.js είναι μία βιβλιοθήκη της javaScript που επιτρέπει στον προγραμματιστή να εισάγει διαφόρων τύπων γραφήματα χρησιμοποιώντας το στοιχείο **canvas** της HTML5. Χρησιμοποιώντας τον καμβά, χρειάζεται μεσάζουσα βιβλιοθήκη polyfill για να χρησιμοποιηθεί με παλαιότερους περιηγητές.
Η βιβλιοθήκη δεν έχει εξαρτήσεις και το μέγεθός της είναι χαμηλό και μπορεί να μικρύνει ακόμη περισσότερο αν δε χρησιμοποιηθούν και οι έξι κύριοι τύποι γραφημάτων, αν συμπεριληφθούν μόνο οι απαιτούμενες ενότητες.
Τα γραφήματα είναι ανταποκρινόμενα, οπότε προσαρμόζονται στο διαθέσιμο χώρο. Τέλος, σε αντίθεση με πολλές βιβλιοθήκες στο διαδίκτυο, παρέχει εκτενή και ξεκάθαρη τεκμηρίωση που καθιστά τη χρήση των βασικών αλλά και πιο προχωρημένων χαρακτηριστικών του πολύ εύκολη.

1.7 Λοιπά χαρακτηριστικά

Για την εκπόνηση της εργασίας χρησιμοποιήθηκαν επίσης:

Google Maps Api
Node version 10.6.3
NPM version 6.9.0
MongoDB version 3.4.9

2: ΤΟ ΠΡΟΓΡΑΜΜΑ

2.1 Επίπεδο Ίντερνετ των Πραγμάτων

2.1.1 Αισθητήρας

Για τους σκοπούς της εργασίας, ήταν απαραίτητη η χρήση ενός τύπου **ψευδο-αισθητήρα (dummy sensor)**, ο κώδικας του οποίου είναι γραμμένος σε javascript και σκοπός του οποίου είναι να προσομοιώσει τη λειτουργία πραγματικών αισθητήρων που μπορούν να ενταχθούν στην εφαρμογή.
Αρχικά ο αισθητήρας "αποφασίζει" τυχαία τα στοιχεία του **(Type, Longitude, Latitude,Unit of Measurement)(Τύπος, Γεωμετρικό Μήκος, Γεωμετρικό Πλάτος, Μονάδα Μέτρησης)**. Η προεπιλεγμένη κατάσταση ενός αισθητήρα είναι "ενεργός". Καθώς "αποφασίζει" τα χαρακτηριστικά αυτά, συνθέτει το πρώτο του μήνυμα στο διακομιστή, το **regmess,** το οποίο εμπεριέχει και το **sensorId**.

```
// create a client instance
client = new Paho.MQTT.Client("broker.hivemq.com", 8000, sensorId);

// connect the client
connect();

function connect(){
    client.connect({onSuccess:onConnect});

    client.onConnectionLost = onConnectionLost;
    client.onMessageArrived = onMessageArrived;

}

function onConnect() {
    client.subscribe("SensorControl");
    client.subscribe("Check/"+type);
    client.subscribe("Data/"+type);
    console.log("Subscribed to Check/"+type+" and Data/"+type);

    sendRegistermessage();
    refreshIntervalId= setInterval(sendmessage,interval);
}
```

Εικόνα 2: Ο κώδικας για τη σύνδεση του αισθητήρα στο ΙτΠ

Όπως βλέπουμε στο σχήμα 2.1, για τη σύνδεση μέσω MQTT στο ΙτΠ, δημιουργείται αρχικά ένα **instance(στιγμιότυπο)** client, το οποίο έχει τις απαραίτητες μεθόδους ώστε να μπορεί να συνδέεται, να αποσυνδέεται, να εγγράφεται σε θέματα, να δημοσιεύει σε αυτά και να διαβάζει τα μηνύματα άλλων πραγμάτων. Στη συνάρτηση connect ανατίθενται **ακροατές(listeners)** για τις περιπτώσεις όπου χάνεται η σύνδεση και που έρχεται μήνυμα.

Μόλις επιτευχθεί ή σύνδεση, ο αισθητήρας εγγράφεται στα θέματα SensorControl, (από εκεί ενημερώνεται αν πρέπει να (απ)ενεργοποιηθεί), Check/* και Data/* (εδώ στέλνει τα δεδομένα του), όπου στη θέση του (*) μπαίνει ο τύπος του αισθητήρα, ώστε όλοι οι αισθητήρες ίδιου τύπου να ακούνε από το ίδιο θέμα και μόνον αυτοί. Έπειτα ο αισθητήρας στέλνει μήνυμα να ειδοποιήσει το server ότι δημιουργήθηκε, ώστε να συμπεριληφθεί στη βάση και να ληφθεί υπ'όψιν από το πρόγραμμα και από το διαχειριστή.
Τέλος με το setInterval(sendmessage, interval), αρχίζει να στέλνει τα δεδομένα ανά <interval> χρονικά διαστήματα. Η sendmessage δημιουργεί ένα τυχαίο μήνυμα και το στέλνει στο αντίστοιχο topic.

2.1.2 Εφαρμογή

```
1  const MongoClient = require( 'mongodb' ).MongoClient;
2  const url = "mongodb://localhost:27017";
3  const mqtt = require('mqtt');
```

Εικόνα 3: Οι απαραίτητες συμπεριλήψεις (/services/mqtt.js)

Στη μεριά της εφαρμογής, ο διακομιστής μας (πελάτης για τα “μάτια” του MQTT) ακούει τα νέα μηνύματα και λαμβάνει την επιθυμητή δράση.
Για να λειτουργήσει το σενάριο αυτό, χρειάζεται τη βιβλιοθήκη για το MongoDB για όποτε χρειάζεται να γίνει προσθήκη μέτρησης ή αισθητήρα στη βάση. Το url είναι η διεύθυνση στην οποία θα γίνει η σύνδεση με τη βάση. Τέλος, ζητά τη βιβλιοθήκη mqtt, η οποία είναι εγκατεστημένη εκ των προτέρων με την εντολή npm install mqtt.

```
5  let client;
6
7  // Initialize a new mqtt client
8  const mqttInit = () => {
9      client = mqtt.connect("https://broker.hivemq.com", 8000, "appServer");
10 }
11
12 //Connect to MQTT
13 const onConnect = () => {
14     client.on('connect', function(){
15         client.subscribe("Data/temp");
16         client.subscribe("Data/humi");
17         client.subscribe("Data/a_po");
18         client.subscribe("Data/a_pr");
19         client.subscribe("Data/wind");
20         client.subscribe("Entries");
21         client.subscribe("SensorControl");
22         console.log("Finished with subscriptions");
23
24     });
25
```

Εικόνα 4: Η σύνδεση του server στο MQTT (/services/mqtt.js)

Έπειτα, ο server, όμοια με τον αισθητήρα, συνδέεται στο MQTT. Αρχικά δημιουργείται ένα instance client συνδεδεμένου **στον ίδιο** broker με τον αισθητήρα. Μετά τη σύνδεση, ο server εγγράφεται στα topics όπου στέλνουν μηνύματα οι αισθητήρες.

```
    client.on('message', function(topic, message){
        mqttHandler(topic, message);
    })
}

// Get message from mqtt check the topic of the message
// and dispatch to specific function
const mqttHandler = (topic, message) => {

    if(topic.includes('Entries')) {
        sensorHandler(message);
    } else if(topic.includes('Data')) {
        dataHandler(message);
    }

}
```

Εικόνα 5: Ο κώδικας που διαχειρίζεται τα νέα μηνύματα

Όταν έρθει μήνυμα, ενεργοποιείται ο ακροατής client.on('message', function()), ο οποίος παραπέμπτει στη μέθοδο mqttHandler. Η μέθοδος αυτή, αναλόγως του topic, παραπέμπει είτε στη μέθοδο sensorHandler, αν το μήνυμα ήταν στο topic “Entries” (συνεπώς νέος αισθητήρας συνδέθηκε), είτε στην dataHandler αν το μήνυμα ήταν στο topic “Data/*”, οπότε έχει έρθει νέα μέτρηση από κάποιον αισθητήρα.
Εάν ο διαχειριστής επιθυμεί να απενεργοποιήσει ή ενεργοποιήσει έναν αισθητήρα από το χάρτη, όπως θα δούμε παρακάτω, στέλνει μήνυμα στο topic “SensorControl”, από το οποίο διαβάζουν οι αισθητήρες, το id του αισθητήρα και την επιθυμητή κατάσταση (0 για απενεργοποίηση, 1 για ενεργοποίηση) με την εντολή:
“mqttHandler.getClient().publish('SensorControl', req.params.id + ', 0');” ή
“mqttHandler.getClient().publish('SensorControl', req.params.id + ', 1');”
(αρχείο routes/sensors.js), όπου η getClient επιστρέφει το instance του client που έχει δημιουργηθεί στο mqtt.js και το req.params.id παίρνει το id του αισθητήρα από τις παραμέτρους του url.

2.2 Επίπεδο MongoDB

Για το εν λόγω εγχείρημα, όπως και γενικότερα για τις εφαρμογές ΙτΠ, το επίπεδο της βάσης δεδομένων είναι εξίσου σημαντικό με το επίπεδο επικοινωνίας των συσκευών. Χωρίς αυτό, η πληροφορία που ανταλλάσσεται χάνεται. Με εξαίρεση κάποιες πολύ συγκεκριμένες περιπτώσεις που δε μας ενδιαφέρει να διατηρήσουμε την πληροφορία που ανταλλάσσεται, αυτό είναι αποτρεπτικό για τη λειτουργία των εφαρμογών. Κάθε query

προς τη MongoDB, παίρνει σαν όρισμα και μία **συνάρτηση ανάκλησης (callBack function)** η οποία καθορίζει πώς θα χρησιμοποιηθούν τα αποτελέσματα της ερώτησης αυτής.

2.2.1 MongoDB στο MQTT κομμάτι

```
// Responsible to insert data to sensors document
const sensorHandler = (message) => {
    MongoClient.connect( url,  { useNewUrlParser: true }, function( err, client ) {
        if(err) {
            console.log(err);
            return false;
        }
        const db  = client.db('sencity');
        let data = message.toString().split(',');

        var obj = {
            _id: message.toString().replace(/,/g, ''),
            type: data[0],
            lon: parseFloat(data[1]).toFixed(6),
            lat: parseFloat(data[2]).toFixed(6),
            uofM: parseInt(data[3]),
            isOn: 1
        };
        console.log('sensor' + obj);
        db.collection('sensors').insertOne(obj);
        db.close();
    });
}
```

Εικόνα 6: Ο κώδικας της συνάρτησης sensorHandler(message) (/services/mqtt.js)

Συνεχίζοντας από εκεί που μας άφησε η Ενότητα 2.1, εάν το μήνυμα που έχει έρθει, είναι στο topic "Entries", τότε επιλέγεται η συνάρτηση sensorHandler που παίρνει το message ως παράμετρο.
Εδώ, η μέθοδος MongoClient.connect(url, options, function(err,client)), συνδέεται με τη βάση. Αν η σύνδεση είναι επιτυχής, σπάει το μήνυμα (που έχει έρθει σαν γραμματοσειρά με κάθε πεδίο ενωμένο με "," ώστε να δημιουργήσει ένα **object(αντικείμενο)** με τα εξής πεδία:

{

_id	το "όνομα" του αισθητήρα
type	ο τύπος δεδομένων που αποστέλλει ο συγκεκριμένος αισθητήρας
lon	το γεωμετρικό μήκος της θέσης του αισθητήρα
lat	το γεωμετρικό πλάτος της θέσης του αισθητήρα
uofM	η μονάδα μέτρησης του δεδομένου που αποστέλλει ο αισθητήρας
isOn	αν ο αισθητήρας είναι ενεργός ή ανενεργός

}

Αποθηκεύει το νέο αισθητήρα στη βάση (db.collection('sensors').insertOne(obj)) και κλείνει τη σύνδεση με τη βάση.
Ομοίως, στην περίπτωση που έχουμε νέα μέτρηση, το πρόγραμμα σπάει το μήνυμα σε ένα object με τα εξής πεδία:

{

data	τα δεδομένα που μετρήθηκαν
sensortype	ο τύπος του αισθητήρα (συνεπώς και των δεδομένων)
lon	γεωγραφικό μήκος αισθητήρα

lat	γεωγραφικό πλάτος αισθητήρα
sensoruom	μονάδα μέτρησης του αισθητήρα (συνεπώς και των δεδομένων)
sensorId	ταυτότητα του αισθητήρα
Datetime	χρονική στιγμή μέτρησης

}

```
// Responsible to insert data to entries document
const dataHandler = message => {
    MongoClient.connect( url,  { useNewUrlParser: true }, function( err, client ) {
        if(err) {
            console.log(err);
            return false;
        }
        const db  = client.db('sencity');
        let data = message.toString().split(',');
        //find the sender and break message into parts
        db.collection('sensors').findOne({
            _id: data[1] + parseFloat(data[2]).toFixed(6) + parseFloat(data[3]).toFixed(6) + data[4]
        }).then(result => {
            let obj = {
                data: parseInt(data[0]),
                sensortype: data[1].substring(0, data[1].indexOf('Sensor')).toLowerCase(),
                lon: parseFloat(data[2]).toFixed(6),
                lat: parseFloat(data[3]).toFixed(6),
                sensoruom: data[4],
                sensorId: result._id,
                Datetime: new Date(),
            };

            db.collection('entries').insertOne(obj);
            db.close();
        });
    });
}
```

Εικόνα 7: Ο κώδικας της συνάρτησης dataHandler(message) (/services/mqtt.js)

2.2.2 MongoDB στο “sensors.js”

```
// Responsible to handle / (sensors map)
router.get('/', (req, res) => {
    MongoClient.connect(url,  { useNewUrlParser: true }, function( err, client ) {
        if (err) {
            console.log('Unable to connect to the Server', err);
        } else {
            // Get the documents collection
            const db    = client.db('sencity');
            db.collection('sensors').find({}).toArray(function (err, result) {
                res.render('sensors/index',{
                    "sensors"       : result
                });
            });
        }
    });
});
```

Εικόνα 8: Ο κώδικας για την αρχική σελίδα (/routers/sensors.js)

Η αρχική σελίδα της εφαρμογής(index), παραπέμπει αυτόματα στη σελίδα ανασκόπησης των αισθητήρων. Για τη λειτουργία της σελίδας αυτής, τρέχει από πίσω το αρχείο "sensors.js". Σε αυτό το αρχείο, έχουν υλοποιηθεί οι μέθοδοι διαχείρισης των αισθητήρων.

Εδώ, γίνεται ένα βασικό **query(ερώτημα)** στον server για την εύρεση όλων των αισθητήρων που είναι συνδεδεμένοι στην εφαρμογή, ανεξαρτήτως του αν είναι ενεργοί ή ανενεργοί. Τα αποτελέσματα αυτού του query έρχονται σε JSON μορφή (δηλαδή σε μορφή **string(γραμματοσειράς)**). Η εμφάνιση της σελίδας καλείται μέσω της res.render(path, {}), όπου στο δεύτερο πεδίο στέλνονται τα δεδομένα που θέλουμε να έχει η σελίδα μας. Στην προκειμένη περίπτωση, θέλουμε να περάσουμε τους αισθητήρες.

```
db.collection('sensors').updateOne({'_id': req.params.id}, { $set: { 'isOn': 1 }}).then(result => {

    mqttHandler.getClient().publish('SensorControl', req.params.id + ', 1');
    res.redirect('/');
});
```

Εικόνα 9: Οι διαφορές στον κώδικα της sensor/:id/enable (/routes/sensors.js)

Σε περίπτωση που ο χρήστης θελήσει να απενεργοποιήσει ή να ενεργοποιήσει έναν αισθητήρα, καλείται το path "/sensors/:id/disable" ή "sensors/:id/enable αντίστοιχα. Σε αυτές τις περιπτώσεις, οι τρεις τελευταίες γραμμές αντικαθίστανται από:
όπου ενημερώνεται η βάση για την αλλαγή του πεδίου "isOn" από 0 σε 1 (αν ενεργοποιείται ο αισθητήρας) ή από 1 σε 0 (αν απενεργοποιείται ο αισθητήρας). Έπειτα, όπως έχουμε δει στην Ενότητα 2.1, στέλνεται ένα μήνυμα στο topic "SensorControl" ώστε να ενημερωθεί ο αισθητήρας για την αλλαγή αυτή. Ολοκληρώνεται η διαδικασία με επαναφορά στην αρχική σελίδα.

2.2.3 MongoDB στο "entries.js"

Το "entries.js" είναι το αρχείο που τρέχει για τις δύο **καρτέλες(tabs)** όπου ο διαχειριστής αλληλεπιδρά με τα δεδομένα των μετρήσεων των αισθητήρων ("Entries" ->"/entries" και "Query Entries"->"/entries/search").
Στην πρώτη περίπτωση, ο διαχειριστής μπορεί να δει όλες τις μετρήσεις από όλους τους αισθητήρες, χωρίς κάποια διάταξη και, εφ'όσον επιθυμεί, να **αναζητήσει(search)** τις μετρήσεις ενός συγκεκριμένου αισθητήρα με το sensor Id .

Εδώ, επειδή τα δεδομένα μας εμφανίζονται σε μεγάλο όγκο, μας ενδιαφέρει να υπάρχει **σελιδοποίηση(pagination)**. Το pagination στηρίζεται στο συνολικό πλήθος από entries που επιστρέφονται από τη βάση και στο μέγεθος σελίδας (εδώ προεπιλογή είναι το 30). Αρχικά γίνεται μία φορά το query (το οποίο παίρνουμε από το url με το req.query.search) ώστε να καθοριστεί το συνολικό πλήθος από entries και σε ποιά σελίδα θέλει ο διαχειριστής να μεταβεί. Έπειτα, το δεύτερο query, γίνεται με **skip (άλμα)** και **limit (όριο)**. Η σελίδα εμφανίζεται στο entries/index.jade με παραμέτρους τα αποτελέσματα του query

```
// Responsible for entries list
router.get('/', (req, res) => {
    MongoClient.connect(url,  { useNewUrlParser: true }, function( err, client ) {
        if (err) {
            console.log('Unable to connect to the Server', err);
        } else {
            // Get the documents collection
            const entries = client.db('sencity').collection('entries');

            // Total of entries used for pagination
            let total = 0;

            let query = {};

            if(req.query.search) {
                query.sensorId = req.query.search;
            }

            entries.find(query).toArray((err, result) => {
                total = result.length;

                const page = req.query.page == null ? 1 : req.query.page; //ternary operator
                const perPage = 30;
                const skip = perPage * (page - 1);

                entries.find(query).skip(skip).limit(perPage).toArray((err, result) => {
                    const pagination = paginate(total, page);

                    res.render('entries/index',{
                        "entries": result,
                        "pagination": pagination
                    });
                })
            });
        }
        db.close();
    });
});
```

Εικόνα 10: Ο κώδικας για τη σελίδα Entries (/routes/entries.js)

και το pagination, που περιέχει τον κώδικα html για το μενού των σελίδων.
Περνώντας στο πολυπλοκότερο κομμάτι της εργασίας, ο διαχειριστής έχει τη δυνατότητα να αναζητήσει, ώστε να δει σε γράφημα, τους μέσους όρους από τις αποθηκευμένες στη βάση μετρήσεις με κριτήριο τον τύπο μέτρησης, το γεωμετρικό μήκος και πλάτος, την

```
// Responsible to to show the entries query results
router.get('/search-results', (req, res) => {
    MongoClient.connect(url,  { useNewUrlParser: true }, function( err, client ) {
        if (err) {
            console.log('Unable to connect to the Server', err);
        } else {
            // Get the documents collection
            const entries = client.db('sencity').collection('entries');

            let intervalObj, resultsObj;

            if(req.query.freqtype == 'min') {   //if interval is in minutes
                intervalObj = {
                    type: "$sensortype",
                    year: { "$year": "$Datetime" },
                    dayOfYear: { "$dayOfYear": "$Datetime" },
                    hour: { "$hour": "$Datetime" },
                    interval: {
                        $subtract: [
                            { $minute: "$Datetime" },
                            { $mod: [{ $minute: "$Datetime"}, parseFloat(req.query.frequency) ] }
                        ]
                    }
                }
```

Εικόνα 11: Ο κώδικας για την εξειδικευμένη αναζήτηση (/routes/etries.js)

ημερομηνία λήψης της μέτρησης, τη συχνότητα και εν τέλει (θα ασχοληθούμε με αυτό σε επόμενη Ενότητα), να ζητήσει στατιστικά για κάποιες ακραίες τιμές.

Για να γίνει η ερώτηση στη βάση, χρειαζόμαστε δύο αντικείμενα, το intervalObj και το resultsObj. Το intervalObj χρησιμοποιείται ως id για την ομαδοποίηση των αποτελεσμάτων ως προς τη χρονική περίοδο. Το πεδίο interval είναι το κλειδί για τη σωστή ομαδοποίηση των απαντήσεων. Ενώ τα υπόλοιπα πεδία θα πάρουν τιμές που υπάρχουν ήδη στη βάση, το interval θα σχηματίσει την τιμή του με τον εξής τρόπο:

- Παραδοχή 1: Κάποια αποτελέσματα θα έχουν ίδιο year, dayOfYear, hour
- Παραδοχή 2: Ονομαστική Datetime θα οριστεί η κάθε Datetime σε λεπτά που απέχει n*req.query.frequency λεπτά από την αρχική Datetime σε λεπτά (αρχική Datetime είναι η πρώτη Datetime και την παίρνουμε από το req.query.since, n είναι ακέραιος αριθμός)
- Παραδοχη 3: Κάποια αποτελέσματα θα έχουν απόκλιση από την κοντινότερη “προς τα πίσω” χρονικά ονομαστική τους Datetime, μικρότερη του απαιτούμενου χρονικού διαστήματος.
- Παραδοχή 4: Αυτές οι Datetime αν διαιρέσουν την κοντινότερη ονομαστική τους θα έχουν υπόλοιπο μικρότερο του req.query.interval και ίσο με τη διαφορά του από την ονομαστική τους Datetime.
- Συμπέρασμα: Αν αυτό το υπόλοιπο αφαιρεθεί από την τρέχουσα Datetime σε λεπτά, θα καταλήξουν όλα αυτά τα results να έχουν την ίδια τιμή στο interval (που θα είναι η κοντινότερη “ονομαστική” τους Datetime σε λεπτά)

```
            resultsObj = {
                data: "$avg",
                year: "$_id.year",
                dayOfYear: "$_id.dayOfYear",
                hour: "$_id.hour",
                minutes: "$_id.interval"
            }
```

Εικόνα 12: Ο κώδικας για την εξειδικευμένη αναζήτηση (routes/entries.js)

Το intervalObj θα είναι το κλειδί της ομαδοποίησης στο pipeline που θα δούμε πιο κάτω και που θα επιστρέφει το μέσο όρο όλων των καταχωρήσεων με ίδιο intervalObj
Το resultsObj είναι το αντικείμενο με τη χρήσιμη για εμφάνιση πληροφορία. Εδώ όπως βλέπουμε αποθηκεύεται ο μέσος όρος για κάθε ημερομηνία και το παραπάνω interval (το "_id." στην αρχή προκύπτει από τον κώδικα του pipeline) καταλήγει να είναι το minute της κάθε καταχώρησης, όπως ήταν αναμενόμενο.

```
// Create the search query
let pipeline = [
    {
        // Match all the specific fields with
        // the values get from query string
        $match: {
            lon: {
                $gte: req.query.fromlong, // $gte = greater
                $lt: req.query.tolong // $lt = less than
            },
            lat: {
                $gte: req.query.fromlat,
                $lt: req.query.tolat
            },
            Datetime: {
                $gte: new Date(req.query.since),
                $lt: new Date(req.query.until)
            },
            sensortype: { $in: req.query.types} // $in needs an array of values
        }
    },
    {
        $group: {       //after getting all results in specified dates/ longitudes/ latitutes/etc, group them by interval
            _id: intervalObj,
            avg: {
                $avg: "$data"
            }
        }
    },

    {
        $sort: {        //after grouping by date, sort by date in following order
            "_id.year": 1,
            "_id.dayOfYear": 1,
            "_id.hour": 1,
            "_id.interval": 1
        }
    },
    {
        $group: {   //after sorting by date, group results by type
            _id: {
                type: "$_id.type",
            },
            results: { $push : resultsObj}  //push every group of results in each type group
        }
    }
]
```

Εικόνα 13: Ο κώδικας για την εξειδικευμένη αναζήτηση (routes/entries.js)

Το MongoDb **Aggregation Pipeline(Αγωγός Συνάθροισης)** ένα πλαίσιο για τη **συνάθροιση (aggregation)** δεδομένων μοντελοποιημένο σύμφωνα με την ιδέα των αγωγών

δεδομένων. Με το pipeline, κάθε ζητούμενο διαχωρισμένο από αγκύλες “{}” μπαίνει στην ερώτηση σειριακά.
Αρχικά θα αναζητηθούν όλες οι καταχωρήσεις που θα πλοιρούν τα κριτήρια Γεωγραφικού Μήκους/Πλάτους, τύπου δεδομένων και θα βρίσκονται εντός των επιθυμητών ημερομηνιών. Όλα αυτά θα ομαδοποιηθούν σύμφωνα με τα πεδία που ζητά το intervalObj και θα διαταχθούν με αύξουσα ημερομηνία. Η σειρά για την διάταξη ημερομηνία αύξουσα ή φθίνουσα σειρά, γίνεται σωστά μόνο εάν πρώτα διατάξουμε τα έτη, μετά τις ημέρες, μετά τις ώρες και τέλος τα λεπτά. Τέλος, τα αποτελέσματα θα ομαδοποιηθούν με βάση τον τύπο και θα σχηματίσουν πίνακες δεδομένων.
Αφού εκτελεστεί το query, καθορίζονται οι παράμετροι για τα στατιστικά αρχεία που θα εμφανιστούν στο διαχειριστή και αποστέλλονται στο view “entries/results.jade” όπου ο χρήστης μπορεί να δει γραφήματα με τα δεδομένα όσων τύπων ζήτησε και να κατεβάσει τα επιθυμητά αρχεία.

2.3 Chart JS και εμφάνιση αποτελεσμάτων

Τα αποτελέσματα της πιο πάνω συνάθροισης αξιοποιούνται για τη δημιουργία γραφημάτων από το view “/entries/results.jade” όπου με τη χρήση της βιβλιοθήκης Chart.js δημιουργείται και γεμίζεται ένα ή περισσότερα canvases.

```
entries.aggregate(pipeline).toArray((err, results) => {
    if(err) console.log(err);

    let stats_params = {
        bigger_than: req.query.frequency_bigger_than,
        smaller_than: req.query.frequency_smaller_than,
        deviation: req.query.frequency_deviation
    };

    res.render('entries/results',{
        results: results,
        stats_params: stats_params
    });
});
```

Εικόνα 14: Ο κώδικας για την εξειδικευμένη αναζήτηση (routes/entries.js)

Στο results.jade έχουν περαστεί τα αποτελέσματα (results) ως ένας πίνακας από αντικείμενα (όπου κάθε αντικείμενο με τη σειρά του περιέχει τον τύπο ως id και έναν πίνακα από τη χρήσιμη πληροφορία). Για κάθε αποτέλεσμα συνεπώς δημιουργείται ένα δισδιάστατο **context (συγκείμενο):**

var ctx = document.getElementById(result._id.type).getContext('2d');

```
block js
    <script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0"></script>

    script.
        results = !{JSON.stringify(results)}
        stats_params = !{JSON.stringify(stats_params)}
        console.log(results);
        results.forEach(result => {
            var ctx = document.getElementById(result._id.type).getContext('
            2d');

            let dates = result.results.map(result => {
                var date = new Date(result.year, 0);
                date.setDate(result.dayOfYear);
                date.setHours(result.hour);

                if(result.minutes != undefined) date.setMinutes(result.
                minutes)
                return date.toLocaleTimeString() + ' ' + date.getDate() + '
                -' + ( date.getMonth() + 1) + "-" + date.getFullYear();
            })

            let values = result.results.map(result => {
                return result.data;
            })

            var chart = new Chart(ctx, {
                type: 'line',
                data: {
                    labels: dates,
                    datasets: [{
                        label: result._id.type,
                        backgroundColor: 'lightblue',
                        borderColor: 'royalblue',
                        data: values
                    }]
                },

                options: {
                    responsive: true,
                    scales: {
                        yAxes: [{
                            ticks: {
                                beginAtZero:true
                            }
                        }]
                    }
                }
            });
```

Εικόνα 15: Ο κώδικας για την εμφάνιση των γραφημάτων (/views/entries/results.jade)

Το context είναι το “σπίτι” του chart. Οι ημερομηνίες αποθηκεύονται σε μορφή πίνακα στο dates. Οι τιμές μπαίνουν σε μορφή πίνακα στο values. Οι δύο αυτοί πίνακες θα χρησιμοποιηθούν για τους άξονες του γραφήματος. Η μέθοδος Array.map() δημιουργεί ένα νέο

πίνακα με τα αποτελέσματα από την κλήση μίας συνάρτησης για κάθε στοιχείο του αρχικού πίνακα, στη σειρά. Εδώ η βοηθητική συνάρτηση παίρνει τα επιμέρους στοιχεία της ημερομηνίας και συνθέτει μία ημερομηνία.
Δημιουργείται ένα instance για κάθε chart. Αυτό για να εμφανιστεί χρειάζεται σαν παραμέτρους τον τύπο chart, τον άξονα x, που στο chart.js είναι το πεδίο labels (εδώ οι ημερομηνίες), τον άξονα y (datasets.data)(εδώ είναι οι τιμές που μετρήθηκαν) και τα επιμέρους χαρακτηριστικά. Στο αντικείμενο-πεδίο options καθορίζονται η προσαρμοστικότητα του chart στην αλλαγή μεγέθους της σελίδας (responsive: true) και η κλίμακα που χρησιμοποιείται για τους άξονες:

```
scales: {
        yAxes: [{
                ticks: {
                        beginAtZero:true
                }
        }]
}
```

3: ΣΥΜΠΕΡΑΣΜΑΤΑ

3.1 Προβλήματα που εμφανίστηκαν κατά την εκπόνηση της εργασίας

Όπως σε κάθε εγχείρημα, έτσι και στην παρούσα εργασία προέκυψαν πολλά προβλήματα τα οποία ανέστειλαν την υλοποίησή της και βασάνισαν το μυαλό μου. Κύριο πρόβλημα, από την αρχή, ήταν το ηθικό με το οποίο αντιμετώπισα το εγχείρημα αυτό. Στο 7ο έτος, με μία αποτυχημένη προσπάθεια για άλλη πτυχιακή, έχοντας μόλις ξεμπερδέψει με τα μαθήματα και να αντιμετωπίσω κάτι που φαινόταν σαν βουνό από νέες τεχνολογίες και γνώσεις αλλά και ένα, αρχικά, μεγαλύτερο σχέδιο πτυχιακής, πολλές φορές ανέβαλα την προσπάθεια, ακόμη και για μήνες ανά φορά, με αποτέλεσμα να χάνω την πρόοδο και να ξεκινάω πάλι έχοντας κάνει βήματα πίσω.
Όλες οι τεχνολογίες που χρησιμοποιήθηκαν, σχεδόν, αποτελούσαν νέα γνώση για εμένα και η προσαρμογή μου σε αυτές συνοδεύτηκε από πολλές αναταραχές και πολλά σφάλματα και εμπόδια που έπαιρναν μέρες ολόκληρες για να υπερπηδηθούν
Αρχικό πλάνο ήταν η επέκταση ολόκληρου του Web Application και πιθανώς και του αντίστοιχου Smartphone Application. Επειδή όμως οι δημιουργοί της εφαρμογής Sensecity δεν έστειλαν ποτέ ούτε τα στοιχεία για τη βάση τους, ούτε τον ολοκληρωμένο τον κώδικα για την εφαρμογή, συμφωνήθηκε με τους βοηθούς της επιβλέποντος καθηγήτριας, Σωκράτη Μπαρμπουνάκη και Δημήτριο Σουκάρα, να περιοριστεί το εγχείρημα στην υλοποίηση όσων παρουσιάζονται σε αυτήν την εργασία, με καθόλου (αν και τελικώς έγινε κάποια προσπάθεια για λίγο καλύτερη εικόνα) βάρος στο front-end.
Αντιμετωπίστηκε μεγάλη δυσκολία στο query για την ειδική αναζήτηση που οδηγούσε στα charts, το οποίο με λίγο «στύψιμο» του μυαλού και αναζήτηση πιο εξειδικευμένων στοιχείων της MongoDB, επετεύχθη και αυτό.

3.2 Συμπεράσματα σε προσωπικό επίπεδο

Με βάση αυτά και όλα τα προβλήματα που αντιμετωπίστηκαν, το βασικότερο συμπέρασμα είναι πως, τελικώς, «μπορεί να γίνει οτιδήποτε, αρκεί να υπάρχει αποφασιστικότητα» και πως πρέπει να επιβαλλόμαστε στα συναισθήματα άγχους, χαμηλής αυτοπεποίθησης και πανικού και να προχωράμε διαρκώς, ακόμη και αν ο ρυθμός δε μας ικανοποιεί τη δεδομένη χρονική στιγμή, αλλά και να μην χάνουμε τη ροή από τον ενθουσιασμό, όταν όλα φαίνεται να πηγαίνουν καλά.

3.3 Συμπεράσματα επί της εφαρμογής

Το ΙτΠ είναι ήδη μεγάλο και στο μέλλον σίγουρα θα φέρει τον κόσμο στην παλάμη μας ή ακόμη και στην οδοντόβουρτσά μας και το αντίστροφο. Η συγκεκριμένη εφαρμογή μπορεί να εμπλουτιστεί με την αρχική ιδέα, ασφαλώς, αλλά ακόμη και με επικοινωνία αυτοκινήτων με το σύστημα για ασφαλέστερη οδήγηση, να χρησιμοποιηθεί από την πυροσβεστική για πρόληψη πυρκαγιών, από δημόσιες υπηρεσίες που ελέγχουν τη ρύπανση του αέρα, να χρησιμοποιηθεί για εφαρμογές που ελέγχουν την υγεία του ιδιοκτήτη του κινητού ώστε να βρεθούν λύσεις για προβλήματα υγείας έως και για καλύτερο ύπνο.

3.4 Σε περίπτωση που επεκταθεί η εφαρμογή

Σε περίπτωση που επεκταθεί η εφαρμογή, ένα καλό πρώτο βήμα θα ήταν η υλοποίηση της σελίδας στην οποία θα εισέρχεται ένας χρήστης ή μία συσκευή και θα μπορεί να αντλήσει δεδομένα, σε μορφή γραφήματος ή σε μορφή αρχείου. Μετά θα πρέπει να διαφοροποιηθεί από τη σελίδα του διαχειριστή και να βελτιστοποιηθεί η πολύ βασική εμφάνιση της εφαρμογής. Για να γίνει σωστά η επέκτασή της θεωρώ σχεδόν απαραίτητη τη συνεργασία δύο ατόμων, ώστε ο ένας να ασχολείται με το back-end ο άλλος με το front-end.

Πίνακας 1: ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	Ελληνικός Όρος
Internet of Things	Ίντερνετ των Πραγμάτων
Sensor	Αισθητήρας
Smart Home	Έξυπνο Σπίτι
Analytics	Ανάλυση Δεδομένων
Streaming Data	Ροή Δεδομένων
Radio-Frequency Identification	Ταυτοποίηση Μέσω Ραδιοσυχνοτήτων
Message Queuing Telemetry Transport	Τηλεμετρική Μεταφορά Ουράς Μηνυμάτων
Connect	Σύνδεση
Disconnect	Αποσύνδεση
Publish	Δημοσίευση
Subscribe	Εγγραφή/Συνδρομή
Unsubscribe	Απεγγραφή
Bandwidth	Εύρος Ζώνης
Machine-to-Machine	Μηχανή-με-Μηχανή
Client	Πελάτης
Server	Εξυπηρετητής/Διακομιστής
Broker	Μεσίτης
Topic	Θέμα
Cross-Platform	Διαπλατφορμικός
Document-Oriented	Αρχειοστραφής
Schema	Σχήμα
Platform	Πλατφόρμα
Service	Υπηρεσία
Script	Σενάριο
Client-Side Scripts	Σενάριο από την Πλευρά του Πελάτη
Prototype-Based	Βασισμένο σε Πρωτότυπα
Multi-Paradigm	Διαφορεικά (Προγραμματιστικά) Παραδείγ-ματα

Site	Ιστοσελίδα
Browser	Φυλλομετρητής/Περιηγητής
Site-Specific Browser	Εξειδικευμένος Φυλλομετρητής
Desktop Widget	Εφαρμογές Επιφάνειας Εργασίας
Beta	Βήτα (πρώιμη έκδοση)
Web	Ιστός
Template	Πρότυπο
Event	Συμβάν
Dummy Sensor	Ψευδο-Αισθητήρας
Event Listener	Ακροατής Συμβάντος
Type	Τύπος
Longitute	Γεωμετρικό Μήκος
Latitude	Γεωμετρικό Πλάτος
Unit of Measurement	Μονάδα Μέτρησης
Object	Αντικείμενο
Index	Αρχική Σελίδα/Πίνακας/Μετρητής
Query	Ερώτημα
String	Γραμματοσειρά
Tab	Καρτέλα
Search	Αναζήτηση
Pagination	Σελιδοποίηση
Entry	Καταχώρηση
Skip	Άλμα
Limit	Όριο
Pipeline	Αγωγός
Aggregation	Συνάθροιση
Callback Function	Συνάρτηση Ανάκλησης
Context	Συγκείμενο

ΑΝΑΦΟΡΕΣ

[1] Krill, Paul (23 Ιουνίου 2008). «JavaScript creator ponders past, future». InfoWorld. Ανακτήθηκε στις 19 Μαΐου 2009.

[2] Hamilton, Naomi (31 Ιουλίου 2008). «The A-Z of Programming Languages: JavaScript». computerworld.com.au.

[3] Press release announcing JavaScript, "Netscape and Sun announce Javascript", PR Newswire, December 4, 1995

[4] «TechVision: Innovators of the Net: Brendan Eich and JavaScript». Web.archive.org. Αρχειοθετήθηκε από το πρωτότυπο στις 8 Φεβρουαρίου 2008. Ανακτήθηκε στις 14 Ιουνίου 2010.

[5] Brendan Eich (3 Απριλίου 2008). «Popularity». Ανακτήθηκε στις 19 Ιανουαρίου 2012.

[6] «ECMAScript 3rd Edition specification» (PDF). Αρχειοθετήθηκε από το πρωτότυπο (PDF) στις 12 Απριλίου 2015. Ανακτήθηκε στις 15 Μαρτίου 2012.

[7] «JavaScript: The World's Most Misunderstood Programming Language». Crockford.com. Ανακτήθηκε στις 19 Μαΐου 2009.

[8] Kris Kowal (1 Δεκεμβρίου 2009). «CommonJS effort sets JavaScript on path for world domination». *Ars Technica*. Condé Nast Publications. Ανακτήθηκε στις 18 Απριλίου 2010.

[9] «ECMAScript Language Overview» (PDF). 23 Οκτωβρίου 2007. σελ. 4. Αρχειοθετήθηκε από το πρωτότυπο (PDF) στις 26 Μαρτίου 2009. Ανακτήθηκε στις 3 Μαΐου 2009.

[10] *«ECMAScript Language Specification»* (PDF). Αρχειοθετήθηκε από το πρωτότυπο (PDF) στις 12 Απριλίου 2015. Ανακτήθηκε στις 7 Φεβρουαρίου 2013.

[11] Douglas Crockford on Functional JavaScript **>**2:49**< blinkx** (flv) (αρχειοθετήθηκε από το πρωτότυπο 2009-09-23) (ανακτήθηκε 2013-02-07).

[12] The Little JavaScripter shows the relationship with Scheme in more detail.

[13] Πρότυπο ECMA-262

[14] https://www.sas.com/el_gr/insights/big-data/internet-of-things.html

[15] https://www.mobilenews.gr/internet-of-thingsti-einai-me-apla-logia-to-diadiktyo/

[16] http://mqtt.org/

[17] https://1sheeld.com/mqtt-protocol/

[18] https://en.wikipedia.org/wiki/MongoDB

[19] https://dzone.com/articles/why-mongodb

[20] https://github.com/joyent/node/tags?after=v0.0.4

[21] https://github.com/joyent/node/wiki

[22] Node.js: Using JavaScript to Build High-Performance Network Programs, Tilkov S. Vinoski S.

[23] https://cy.ipc2u.com/articles/articles-and-reviews/ti-einai-to-mqtt-kai-giati-to-chreiazomaste-sto-iiot/