ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Ανάπτυξη εφαρμογής IOT για την συλλογή και αποθήκευση δεδομένων με την χρήση βιβλιοθηκών και λογισμικού ανοικτού κώδικα

ΧΕΛΙΩΤΗΣ ΚΙΜΩΝ-ΚΩΝΣΤΑΝΤΙΝΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΤΖΙΑΛΛΑΣ ΓΡΗΓΟΡΙΟΣ
Καθηγητής

Λαμία ..... έτος .....

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Ανάπτυξη εφαρμογής IOT για την συλλογή και αποθήκευση δεδομένων με την χρήση βιβλιοθηκών και λογισμικού ανοικτού κώδικα

ΧΕΛΙΩΤΗΣ ΚΙΜΩΝ-ΚΩΝΣΤΑΝΤΙΝΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΤΖΙΑΛΛΑΣ ΓΡΗΓΟΡΙΟΣ
Καθηγητής

Λαμία ..... έτος .....

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

Development of an IOT application for data collection and storage using libraries and open software code

CHELIOTIS KIMON-CONSTANTINOS

FINAL THESIS

ADVISOR

TZIALLAS GRIGORIOS
Professor

Lamia ..... year .....

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

*1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

*2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: ……/…../20……

Ο – Η Δηλ.

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Στα πλαίσια αυτής της πτυχιακής εργασίας θα μελετηθεί και θα παρουσιασθεί το «Διαδίκτυο των Πραγμάτων» (Internet of Things). Θα αναφερθούν ορισμένα πρωτόκολλα με τα οποία γίνεται η επικοινωνία μεταξύ των IOT συσκευών και του χειριστή, καθώς και Frameworks τα οποία εφαρμόζουν αυτά τα πρωτόκολλα μέσω Node.js. Ύστερα, θα αναπτυχθεί μία εφαρμογή σε Node.js η οποία θα μπορεί για συγκεκριμένα πρωτόκολλα να συλλέγει δεδομένα και να τα αποθηκεύει σε μία βάση δεδομένων μέσω του PostgreSQL.

ABSTRACT

In this thesis the concept of the “Internet of Things” will be studied and presented. Some communication protocols that create communication between IoT devices and the operators and some crucial Frameworks will be explained. I will present an IoT application that I wrote in Node.js, which collects data from IoT sensors and devices, and stores them in a PostgreSQL database for later use.

Contents

1. Εισαγωγή

1.1. Σκοπός της πτυχιακής εργασίας

Ο σκοπός της πτυχιακής εργασίας είναι η μελέτη και ο τρόπος λειτουργίας διάφορων πρωτοκόλλων επικοινωνίας. Θα γίνει έρευνα και ανάλυση ορισμένων πρωτοκόλλων επικοινωνίας, συσκευών IoT, IoT Frameworks και ύστερα θα παρουσιαστεί ένα Framework το οποίο υλοποίησα με τη γλώσσα JavaScript στο περιβάλλον node.js για τη συλλογή δεδομένων IoT και την αποθήκευσή τους σε μία βάση δεδομένων PostgreSQL. Τέλος θα παρουσιαστεί μία αρχιτεκτονική βάσει μίας νέας υπουργικής απόφασης, για τη σταθεροποίηση των θερμοκρασιών σε δημόσια κτήρια στους 27^{o}C κατά την περίοδο του καλοκαιριού και στους 19^{o}C κατά την περίοδο του χειμώνα. Για τη μελέτη θα χρησιμοποιηθούν ενδεικτικά τα κτήρια της σχολής και θα δημιουργηθεί ένα σύστημα παρακολούθησης της θερμοκρασίας σε κάθε εσωτερικό χώρο.

2. Internet of Things Γενικά

2.1. Τι είναι το Internet of Things

Ο όρος Internet of Things ξεκίνησε το 1999 από τον Kevin Ashton, ο οποίος πρότεινε τη χρήση Radio – Frequency Identification (RFID) chips σε προϊόντα ώστε να παρακολουθούνται μέσω εφοδιαστικής αλυσίδας. Το Internet of Things αναφέρεται στη διαδικτυακή διασύνδεση καθημερινών αντικειμένων, τα οποία είναι εξοπλισμένα με ενσωματωμένους αισθητήρες. Χρησιμοποιείται για τη συλλογή δεδομένων και την ανάληψη κάποιας δράσης από αυτά τα αντικείμενα.

Το Internet of Things θα αυξήσει τη χρησιμότητα του Internet, εφαρμόζοντάς του καθημερινά αντικείμενα μέσω ενσωματωμένων συστημάτων. Αυτό οδηγεί στην ευκολία επικοινωνίας των συσκευών με τον άνθρωπο καθώς και με άλλες συσκευές.[1]

Παρά το γεγονός ότι ο όρος Internet of Things είναι σχετικά καινούριος, η ιδέα της σύνδεσης υπολογιστών και δικτύων για τον έλεγχο και την παρακολούθηση συσκευών επικρατεί πολλές δεκαετίες. Τη δεκαετία του ’90 υπήρχαν ήδη εξελίξεις στην ασύρματη τεχνολογία, οι οποίες επέτρεπαν στην «Μηχανή-με-Μηχανή» (M2M) επικοινωνία. Η χρήση IP για τη σύνδεση συσκευών εκτός υπολογιστών έγινε πρώτη φορά σε μία διάσκεψη για το Internet το 1990, όπου μία τοστιέρα με IP μπορούσε να ανάψει και να σβήσει μέσω του Internet.[2]

2.2. Κεντρικές ιδέες για το Internet Of Things

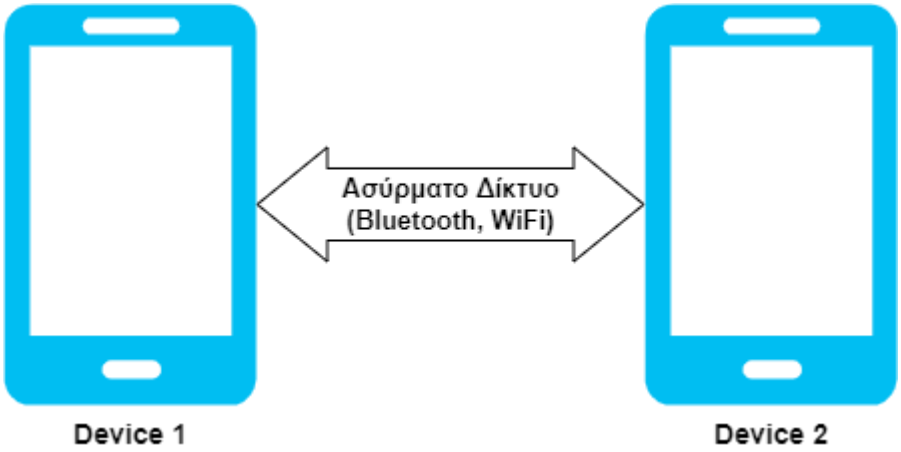
2.2.1. Μοντέλα Επικοινωνίας

2.2.1.1. Device-To-Device Επικοινωνίες

Οι Device-To-Device (D2D) επικοινωνίες ξεκίνησαν στα κυψελοειδή δίκτυα ως ένα νέος τρόπος για την ενίσχυση της απόδοσης των δικτύων. Το μοντέλο αυτό λειτουργεί δίνοντας τη δυνατότητα σε ασύρματες συσκευές που παραβρίσκονται εγγύς μεταξύ τους να επικοινωνούν με τη μέθοδο peer-to-peer, (δίκτυο που επιτρέπει τις συσκευές να μοιράζονται τους πόρους ισοδύναμα.[3]) χωρίς τη συμμετοχή του επιπέδου χρήστη μιας κυψελοειδούς υποδομής.

Μέσω IoT μπορούν να χρησιμοποιηθούν σε E-Medicine για ιατρική περίθαλψη, σε υπηρεσίες εγγύτητας για διαμοιρασμό δεδομένων, σε Gaming, στη διαφήμιση, ακόμα και σε ευφυή συστήματα Vehicle-to-Vehicle επικοινωνίας για την αποφυγή τρακαρισμάτων.[4]

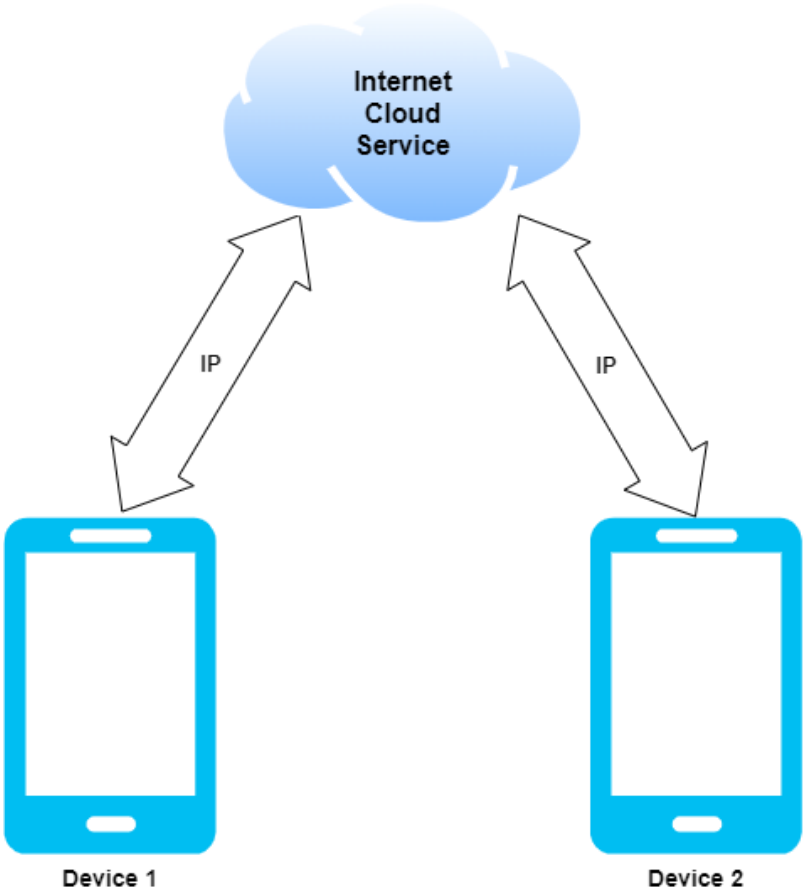
Μπορεί να γίνει μεταφορά αρχείων σε αυξημένους ρυθμούς δεδομένων και με μικρότερη χρήση ενέργειας απ' ότι στα συμβατικά κυψελωτά κανάλια. Υπηρεσίες ροής όπως το Google Chromecast, IPTV και γενικότερα τα TV Boxes χρησιμοποιούν τις D2D επικοινωνίες δημιουργώντας συστάδες και μεταδίδοντας ομαδικά μέσα σε αυτές τις συστάδες.



Διάγραμμα 1: Παράδειγμα επικοινωνίας Device-to-Device

2.2.1.2. Device-To-Cloud Επικοινωνίες

Το Cloud Computing είναι η επόμενη φάση της ανάπτυξής των διαδικτυακών εφαρμογών και επιτρέπει τις ICT (Information and Communications Technology) υπηρεσίες να παραδίδονται ως μεταφορείς. Δυνατότητες υπολογιστών, δομές server και δομές αποθήκευσης, συστήματα, διαδικασίες επιχειρήσεων και άλλοι σημαντικοί πόροι μπορούν να συνδεθούν μέσω Cloud Computing.[5]



Διάγραμμα 2: Παράδειγμα Device-to-Cloud Επικοινωνίας.

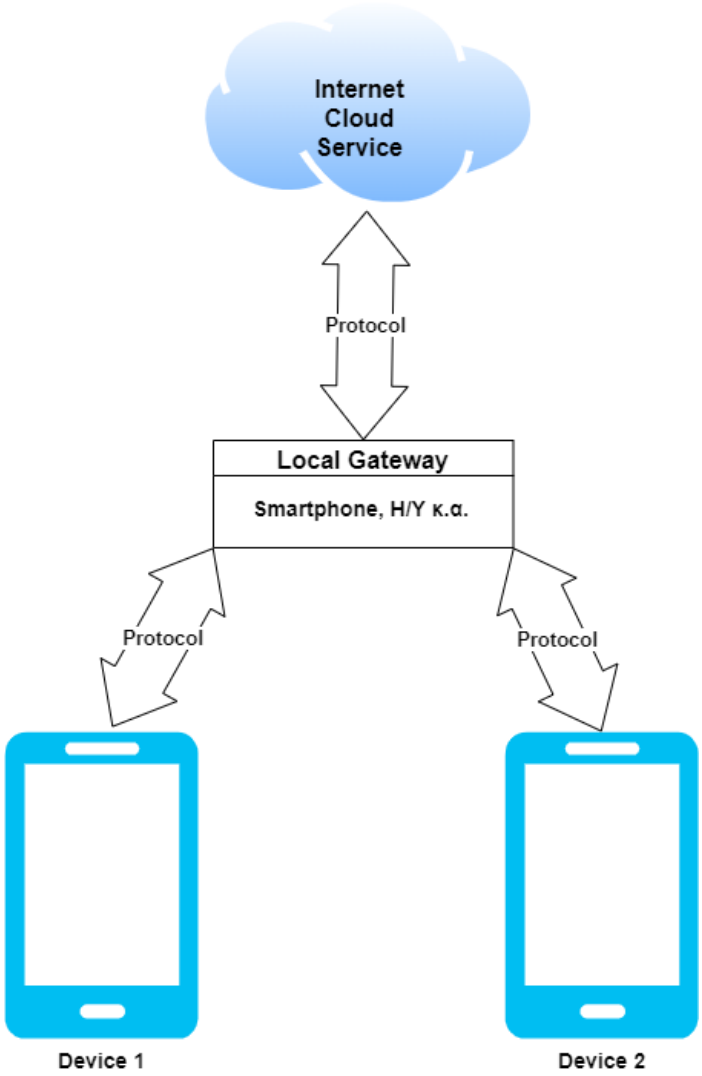
2.2.1.3. Device-To-Gateway Μοντέλο

Στο μοντέλο Device-To-Gateway ή αλλιώς Device-To-Application-Layer Gateway (ALG) μοντέλο, η IoT συσκευή συνδέεται μέσω μιας ALG υπηρεσίας ως αγωγός για να φτάσει σε μία υπηρεσία Cloud. Το Gateway, μπορεί να είναι μέχρι και ένα Smart Phone, το οποίο λειτουργεί ως μεσολαβητής μεταξύ της συσκευής και του Cloud Service.

Αυτό το Gateway μπορεί να προσφέρει ασφάλεια και άλλες λειτουργίες, όπως μετάφραση δεδομένων ή πρωτοκόλλου. Εάν το gateway στο επίπεδο της εφαρμογής (application-layer) είναι ένα Smart Phone, το λογισμικό αυτής της εφαρμογής μπορεί να πάρει τη μορφή ενός App, το οποίο συνδέεται με τη συσκευή IoT και επικοινωνεί με μια υπηρεσία Cloud.

Για παράδειγμα, θα μπορούσε να είναι μία συσκευή που συνδέεται στο Cloud μέσω κάποιας εφαρμογής, όπως το Google Fit, ή κάποια εφαρμογή αυτοματοποίησης που συμπεριλαμβάνει συσκευές που συνδέονται σε ένα hub (κεντρική εφαρμογή), όπως το SmarThings ecosystem της Samsung.

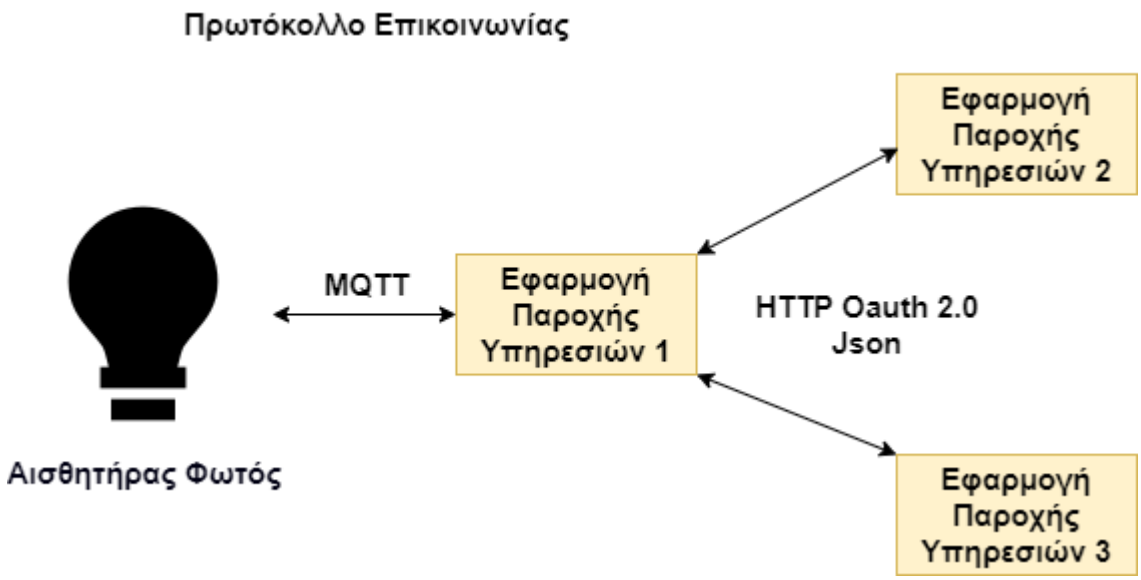
Συσκευές που λειτουργούν με gateway, μπορούν πιθανώς να γεφυρώσουν το κενό της διαλειτουργικότητας μεταξύ συσκευών που επικοινωνούν με διαφορετικά πρότυπα. Για παράδειγμα, οι Z-Wave και το Zigbee πομποδέκτες του SmartThings μπορούν να επικοινωνήσουν και με τους δύο τύπους συσκευών.



2.2.1.4. Back-End Data-Sharing Μοντέλο

Το μοντέλο Back-End Data-Sharing αναφέρεται σε μία αρχιτεκτονική επικοινωνίας που δίνει τη δυνατότητα στον χρήστη να εξάγει και να αναλύει δεδομένα έξυπνων συσκευών από μία Cloud υπηρεσία, σε συνδυασμό με δεδομένα από άλλες πηγές. Επομένως, είναι μία επέκταση του μοντέλου Device-to-Cloud. Με αυτό το μοντέλο, οι χρήστες μπορούν να εξάγουν και να αναλύσουν δεδομένα έξυπνων αντικειμένων από κάποια Cloud υπηρεσία σε συνδυασμό με δεδομένα από άλλες πηγές, και να τα στείλουν σε άλλες υπηρεσίες για συσσωμάτωση και ανάλυση.

Λέγεται ότι η εφαρμογή Map My Fitness είναι ένα καλό παράδειγμα αυτού του μοντέλου, αφού συλλέγει δεδομένα άθλησης από πολλές συσκευές, που κυμαίνονται από το Fitbit (Smart Band), μέχρι και το Wahoo Bike Cadence Sensor (για τη μέτρηση δεδομένων ρυθμού ποδηλασίας).



Διάγραμμα 3: Παράδειγμα λειτουργίας Back-End Data-Sharing μοντέλου.

3. IoT Πρωτόκολλα και Συσκευές

3.1. Πρωτόκολλα IoT

Στο στρώμα μεταφοράς (Transport Layer) υπάρχει ένα πρωτόκολλο επικοινωνίας το οποίο είναι υπεύθυνο για τη δημιουργία επικοινωνίας και την εξασφάλιση ότι τα δεδομένα μεταφέρονται με ασφάλεια. Βρίσκεται πάνω από το πρωτόκολλο δικτύου και ελέγχει τις επικοινωνίες συνήθως μεταξύ μηχανών. Πάνω στο IoT τα πρωτόκολλα μεταφοράς είναι σχεδιασμένα για να λειτουργούν με μικρή κατανάλωση ενέργειας, αφού οι έξυπνες συσκευές συνήθως λειτουργούν με περιορισμένη μπαταρία. Τα πρωτόκολλα με τα οποία θα ασχοληθούμε σε αυτή την πτυχιακή είναι: MQTT (Message Query Telemetry Transport), HTTP (Hyper Text Transfer Protocol), το AMQP (Advance Message Queuing Protocol) το CoAP (Constrained Application Protocol).

3.1.1. Πρωτόκολλο MQTT

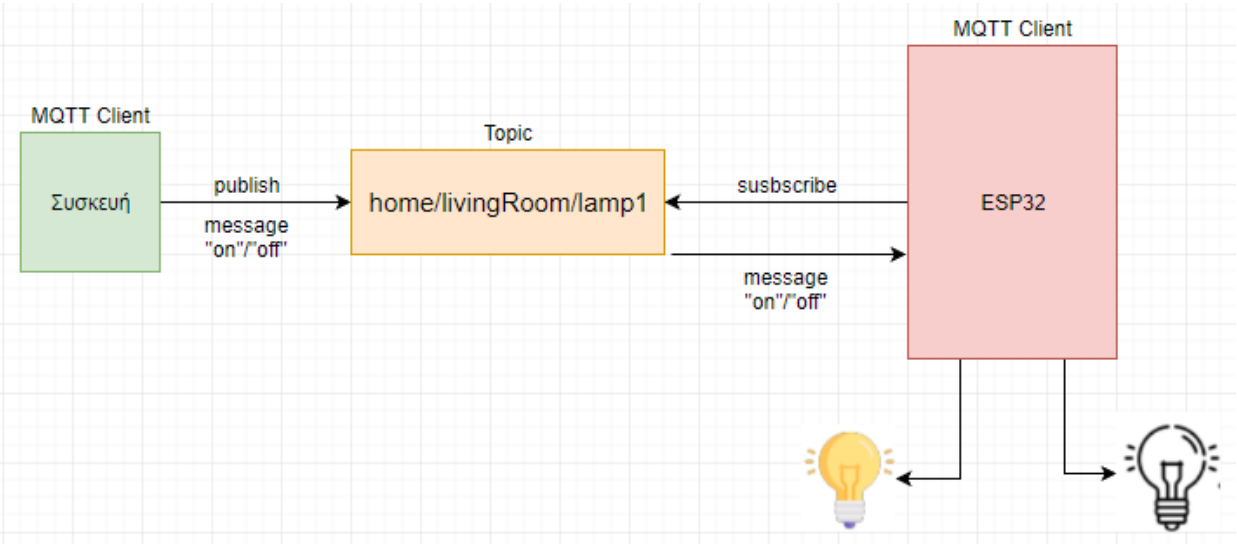
Το MQTT είναι ένα τυποποιημένο πρωτόκολλο με τη χρήση της publish/subscribe μεθόδου, που δημιουργήθηκε το 1999 από την IBM. Το MQTT σχεδιάστηκε για την εύστοχη αποστολή δεδομένων υπό συνθήκες υψηλών καθυστερήσεων δικτύου και καταστάσεις δικτύων χαμηλού εύρους ζώνης.[6]

Οι χρήστες συνδέονται σε έναν broker, ο οποίος δημιουργεί συνδεσιμότητα μεταξύ δύο συσκευών. Κάθε συσκευή μπορεί να κάνει Subscribe σε συγκεκριμένα topics. Όταν ο Client κάνει publish ένα μήνυμα σε ένα subscribed topic, ο broker το προωθεί σε οποιονδήποτε client έχει κάνει Subscribe.

3.1.1.1. Βασικές έννοιες MQTT

a) **Publish/Subscribe:** Στο πρωτόκολλο MQTT οι publishers στέλνουν μηνύματα και οι χρήστες κάνουν subscribe σε συγκεκριμένα topics. Οι εγγεγραμμένοι χρήστες σε αυτά τα topics λαμβάνουν τα μηνύματα τα οποία είναι published σε αυτά τα topics. Από την άλλη και οι χρήστες μπορούν να κάνουν publish μηνύματα στα topics, ώστε όλοι οι subscribers να μπορούν να έχουν πρόσβαση σε αυτά τα μηνύματα.[7]

b) **Topics:** Τα topics θεωρούνται τα θέματα των μηνυμάτων. Λειτουργούν ως strings που χωρίζονται με «/», όπου κάθε «/» δείχνει ένα επίπεδο των topics.



Διάγραμμα 4: Παράδειγμα λειτουργίας MQTT

c) **Quality of Service levels (QoS):** Ανάμεσα σε 2 συσκευές, υπάρχουν 3 επίπεδα για την εξακρίβωση της διανομής δεδομένων. Επειδή το MQTT είναι υπεύθυνο για την αναμετάδοση μηνυμάτων, το QoS δημιουργεί επικοινωνία σε αναξιόπιστα δίκτυα πολύ πιο εύκολα.

i) **QoS0 (το πολύ μία φορά):** Είναι το χαμηλότερο επίπεδο QoS. Θυμίζει το πρωτόκολλο TCP, αφού δεν υπάρχει εγγύηση διανομής του μηνύματος. Ο παραλήπτης δε λαμβάνει απόδειξη για το μήνυμα και το μήνυμα δεν αποθηκεύεται, ούτε αναμεταδίδεται από τον αποστολέα.

ii) **QoS1 (το λιγότερο μία φορά):** Το QoS1 εγγυάται ότι το μήνυμα διανέμεται τουλάχιστον μία φορά στον παραλήπτη. Ο αποστολέας αποθηκεύει το μήνυμα μέχρι να λάβει ένα πακέτο από τον παραλήπτη για την απόδειξη της αποστολής του μηνύματος. Ένα μήνυμα έχει τη δυνατότητα να σταλθεί και να παραληφθεί πολλές φορές.

iii) **QoS2 (ακριβώς μία φορά):** Είναι το υψηλότερο επίπεδο υπηρεσίας στο MQTT. Εγγυάται ότι κάθε μήνυμα θα ληφθεί μόνο μία φορά από τους προγραμματισμένους παραλήπτες. Είναι το πιο ασφαλές και αργό QoS επίπεδο. Η εγγύηση γίνεται διότι υπάρχουν 2 μεταφορές Request/Response του μηνύματος μεταξύ του αποστολέα και του παραλήπτη.[8]

Η δομή των μηνυμάτων MQTT αποτελείται από 2-byte fixed header, έναν variable header και ένα payload. Ο 2-byte fixed header, θα υπάρχει πάντα σε όλα τα πακέτα, ενώ τα άλλα 2 δεν είναι πάντα υπαρκτά.

3.1.2. Πρωτόκολλο HTTP

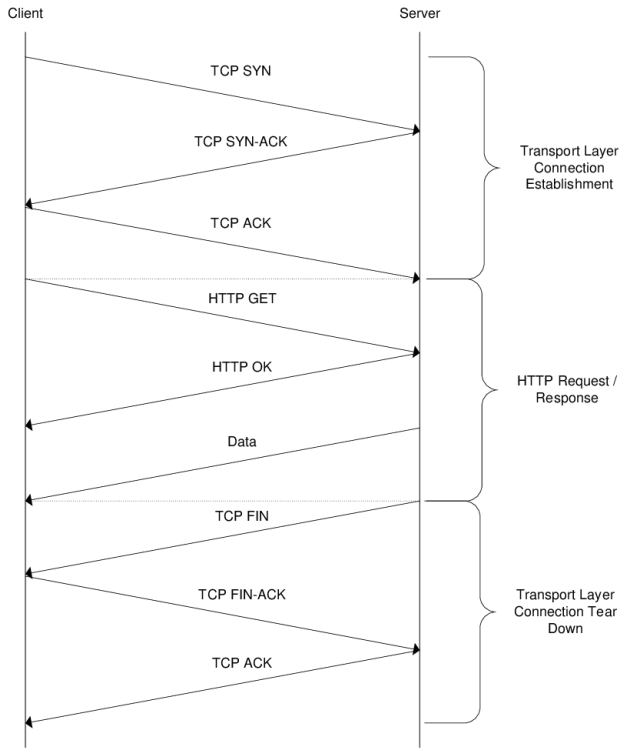
Το HTTP (HyperText Transfer Protocol) είναι το καλύτερο παράδειγμα σε IoT για ένα πρωτόκολλο δικτύου. Το HTTP είναι κυρίως ένα πρωτόκολλο διαδικτυακών μηνυμάτων, το οποίο αναπτύχθηκε από τον Tim Berners-Lee και αργότερα συνεχίστηκε η ανάπτυξη από τις εταιρίες IETF και W3C και δημοσιεύθηκε ως ένα πρότυπο πρωτοκόλλου το 1997[9]. Το HTTP διαθέτει την αρχιτεκτονική Request/Response RESTful Web, όπως και το CoAP για το οποίο θα αναφερθούμε παρακάτω. Χρησιμοποιεί URI (Universal Resource Identifier), σε αντίθεση με το MQTT το οποίο χρησιμοποιεί τα topics. Ο server στέλνει δεδομένα μέσω του URI και ο client λαμβάνει τα δεδομένα. Είναι ένα πρωτόκολλο βασισμένο σε text, και αντί να καθορίζει το μέγεθος του μηνύματος, εξαρτάται από τον web server ή την προγραμματιστική τεχνολογία [10].

3.1.2.1. Το HTTP σε IoT επικοινωνίες

Υποτίθεται πως το πρωτόκολλο HTTP εφαρμόζεται σε επικοινωνίες για IoT. Πρέπει να μεταφέρει έναν μεγάλο αριθμό μικρών πακέτων. Είναι πιθανό να προκαλέσει μερικά σοβαρά προβλήματα, όπως η κατανάλωση πόρων δικτύου και μεγάλες καθυστερήσεις.

Από τη στιγμή που το HTTP λειτουργεί πάνω στο TCP/IP, παρέχονται αξιόπιστες επικοινωνίες. Όμως, επειδή λειτουργεί μέσω TCP, σε κάθε πρόσβαση οι επικοινωνίες απελευθερώνονται, αφού τα δεδομένα που μεταφέρονται είναι βασισμένα σε IP διεύθυνση και το URL και οι σχέσεις τους αλλάζουν δυναμικά. Επομένως, η επικοινωνία ολοκληρώνεται μετά από πολλές απελευθερώσεις σύνδεσης, άρα για IoT υπάρχει μεγάλη κατανάλωση πόρων δικτύου κατά τη διάρκεια της επικοινωνίας.

Η επικοινωνία μέσω HTTP φαίνεται στο διάγραμμα 5.



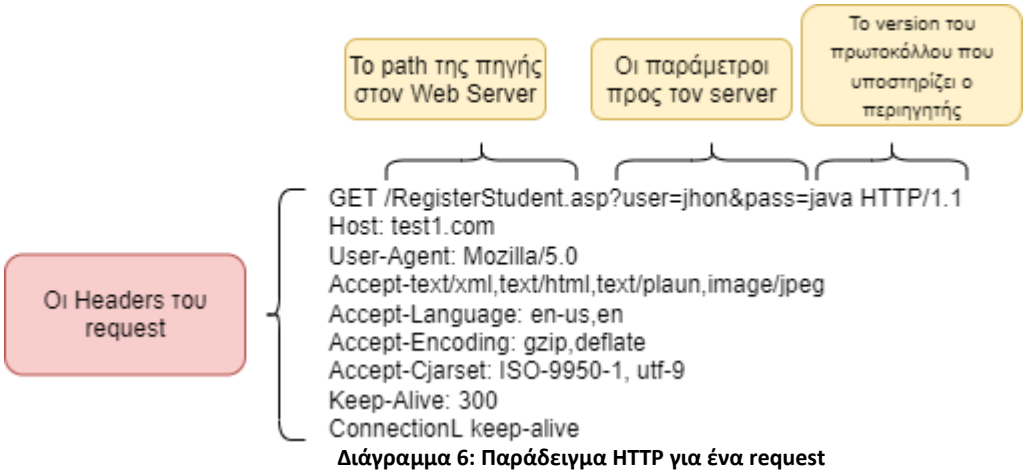
Διάγραμμα 5: Επικοινωνία HTTP [11]

3.1.2.2. Πως λειτουργεί το HTTP

Ως ένα πρωτόκολλο με request/response λειτουργία, το HTTP δίνει τη δυνατότητα στους χρήστες να αλληλοεπιδρούν με στοιχεία όπως HTML αρχεία, μεταδίδοντας hypertext μηνύματα μεταξύ των clients και των servers. Χρησιμοποιεί συγκεκριμένες μεθόδους request για να εκτελέσει διάφορα καθήκοντα. Αυτά είναι:

GET: Είναι μία μέθοδος η οποία ανακτά οποιαδήποτε πληροφορία αναγνωρίζεται από το Request-URI.
HEAD: Είναι πανομοιότυπη με την GET με τη διαφορά ότι ο server δεν πρέπει να επιστρέψει ένα message-body στο response.[15]
POST: Χρησιμοποιείται για να στείλει δεδομένα στον server. Συνήθως στέλνεται σε μορφή HTML και φέρει αλλαγές στον server.
PUT: Αλλάζει όλα τα τρέχοντα δεδομένα του επιλεγμένου πόρου με το payload του request.
DELETE: Διαγράφει τον επιλεγμένο πόρο.

Ο περιηγητής ξεκινάει την επικοινωνία με έναν HTTP server ανοίγοντας μία σύνδεση TCP με αυτόν. Οι συνεδρίες περιήγησης στο Internet χρησιμοποιούν το port 80 από προεπιλογή, όμως υπάρχουν και άλλα port που μπορούν να χρησιμοποιηθούν. Αφού καθιερωθεί μία συνεδρία, δίνεται το έναυσμα της αποστολής και της παραλαβής HTTP μηνυμάτων καθώς γίνεται η επίσκεψη σε μία σελίδα. Είναι ένα stateless σύστημα, αυτό σημαίνει πως η σύνδεση πέφτει αφού ολοκληρωθεί το request. Επομένως, ύστερα από την αποστολή του request από τον browser και ο server απαντήσει με την σελίδα, η σύνδεση κλείνει.



Διάγραμμα 6: Παράδειγμα HTTP για ένα request

3.1.3. Πρωτόκολλο CoAP

Το CoAP πρωτόκολλο είναι ένα πρωτόκολλο μεταφοράς το οποίο χρησιμοποιείται σε constrained κόμβους ή δίκτυα, όπως το WSN, IoT, M2M κλπ. Αυτό το πρωτόκολλο χρησιμοποιείται κυρίως για συσκευές IoT οι οποίες έχουν λιγότερη μνήμη και μικρότερη

κατανάλωση ενέργειας. Το γεγονός ότι χρησιμοποιείται σε διαδικτυακές εφαρμογές αναφέρεται κιόλας ως “The Web of Things Protocol”.[12]

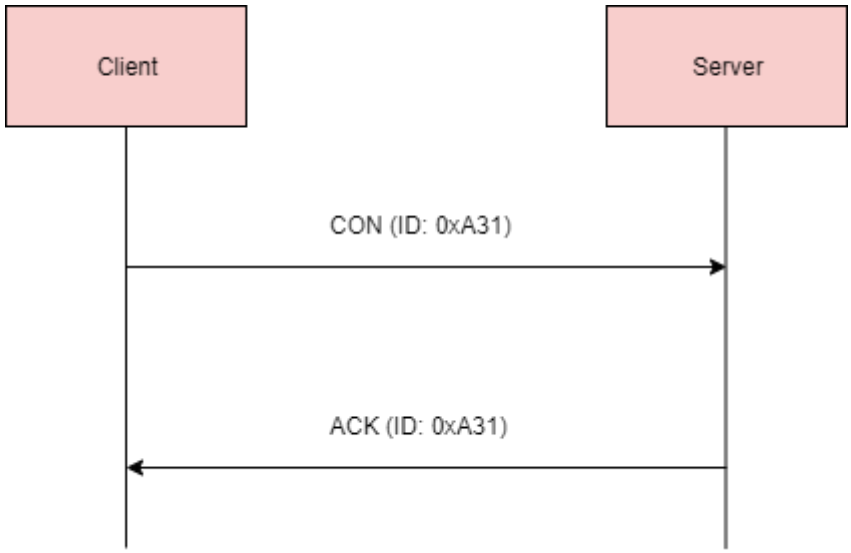
3.1.3.1. Λειτουργία CoAP

Το CoAP μπορεί να θεωρηθεί ως ένα πανομοιότυπο πρωτόκολλο του HTTP αφού είναι φτιαγμένο πάνω σε UDP αντί για TCP, το οποίο είναι κοινό για το HTTP. Χρησιμοποιεί την Efficient XML Interchanges (EXI) μορφή, η οποία είναι πιο αποτελεσματική στο χώρο από το XML/HTML. [13]

Το CoAP αποτελείται από 2 επίπεδα, το messaging και το request/response. Το messaging επίπεδο είναι υπεύθυνο για την εφεδρεία και την συνοχή των μηνυμάτων, καθώς το request/response. [14]

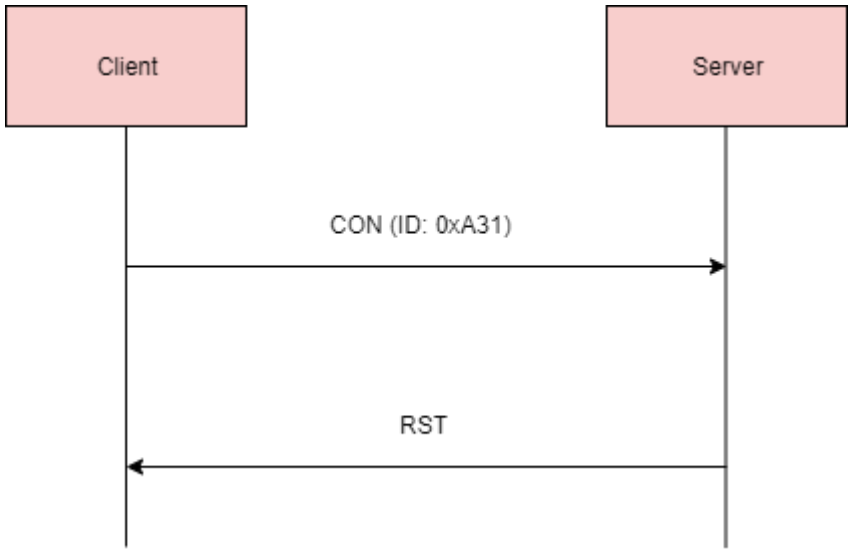
- **Message επίπεδο:** Υπάρχουν τέσσερις τύποι μηνυμάτων που υποστηρίζονται από το CoAP:
  - **Confirmable (CON)**
  - **Non-Confirmable (NON)**
  - **Acknowledgment (ACK)**
  - **Reset (RST)**

Ένα confirmable (CON) μήνυμα είναι ένα αξιόπιστο μήνυμα. Όταν ανταλλάσσονται μηνύματα μεταξύ δύο endpoints, αυτά τα μηνύματα είναι αξιόπιστα. Χρησιμοποιώντας αυτού του είδους μηνύματα, ο client μπορεί να σιγουρευτεί ότι το μήνυμα θα παραδοθεί στον server. Τα confirmable μηνύματα στέλνονται ξανά και ξανά, έως ότου να υπάρξει απάντηση με ένα acknowledge μήνυμα (ACK). Ένα acknowledge μήνυμα περιέχει το ίδιο ID με το confirmable μήνυμα. Στο διάγραμμα 7, παρατίθεται το παράδειγμα αυτής της λειτουργίας.



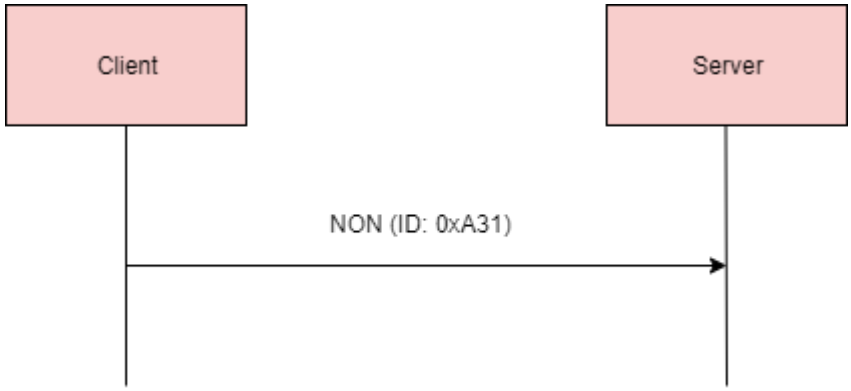
Διάγραμμα 7: Παράδειγμα λειτουργίας CON, ACK

Αν ο server προβληματιστεί με τη διαχείριση του request που λαμβάνει, στέλνει ένα reset (RST) μήνυμα αντί για το acknowledge, όπως φαίνεται στο διάγραμμα 8.



Διάγραμμα 8: Παράδειγμα CON, RST

Ο άλλος τύπος μηνύματος είναι ο Non-Confirmable (NON). Αυτά είναι μηνύματα, τα οποία δε χρειάζονται αναγνώριση από τον server. Είναι μη αξιόπιστα μηνύματα, ή αλλιώς μηνύματα τα οποία δεν περιέχουν σημαντικές πληροφορίες που πρέπει να φτάσουν στον server. Αυτή η κατηγορία μηνυμάτων περιέχει συνήθως δεδομένα που διαβάζονται από sensors.[25]



Διάγραμμα 9: Παράδειγμα NON

3.1.4. Πρωτόκολλο AMQP

Το AMQP είναι ένα ελαφρύ M2M πρωτόκολλο, δημιουργήθηκε από τον John O'Hara το 2003. Είναι ένα συλλογικό πρωτόκολλο, σχεδιασμένο για αξιοπιστία, ασφάλεια, παροχή και διαλειτουργικότητα.[19] Υποστηρίζει τις αρχιτεκτονικές request/response και publish/subscribe. Προσφέρει ένα ευρύ φάσμα χαρακτηριστικών σχετικά με το μοντέλο των μηνυμάτων, όπως αξιόπιστο queuing, publish/subscribe messaging βασισμένο στα topics (όπως στο MQTT), ευέλικτη δρομολόγηση, συναλλαγές κ.α. [20]

Είναι σχεδιασμένο να διευκολύνει τον διάλογο μεταξύ των στοιχείων ενός συστήματος, κάνοντας πιο εύκολη την εναλλαγή μηνυμάτων. Επιπλέον, το AMQP είναι φτιαγμένο για να επιλύσει θέματα ασφάλειας και εμπιστευτικότητας χωρίς να έχει επιπτώσεις στην επίδοση της επικοινωνίας.

3.1.4.1. Λειτουργία AMQP

Τυπικά, ένας client, ως παραγωγός στέλνει ένα μήνυμα για ανταλλαγή, εκεί το μήνυμα διανέμεται με αντίγραφά του σε ουρές, αναλόγως τους κανόνες που έχουν δοθεί για τον τρόπο ανταλλαγής και το routing key που διατίθεται στο μήνυμα. Ύστερα, το μήνυμα καταναλώνεται από τον subscriber. Το μοντέλο του AMQP φαίνεται στο διάγραμμα 10.

Δομικά στοιχεία του AMQP:

- **Ουρά Μηνυμάτων (Message Queue):** Η ουρά λειτουργεί ως Buffer, ο οποίος αποθηκεύει μηνύματα, τα οποία καταναλώνονται αργότερα. Μία ουρά μπορεί επίσης να δηλωθεί με έναν αριθμό χαρακτηριστικών κατά τη διάρκεια της δημιουργίας της. Για παράδειγμα, μπορεί να σημειωθεί ως durable, auto-delete και exclusive, όπου exclusive σημαίνει ότι μπορεί να χρησιμοποιηθεί από μία μόνο σύνδεση και μετά διαγράφεται όταν κλείσει η σύνδεση.
- **Ανταλλαγές και Τύποι Ανταλλαγών:** Ένα κανάλι δρομολογεί τα μηνύματα σε μία ουρά, ανάλογα τον τύπο ανταλλαγής και τη δέσμευση μεταξύ της ανταλλαγής και της ουράς. Για να λάβει μηνύματα μία ουρά, πρέπει πρώτα να δεσμευτεί σε μία τουλάχιστον ανταλλαγή.

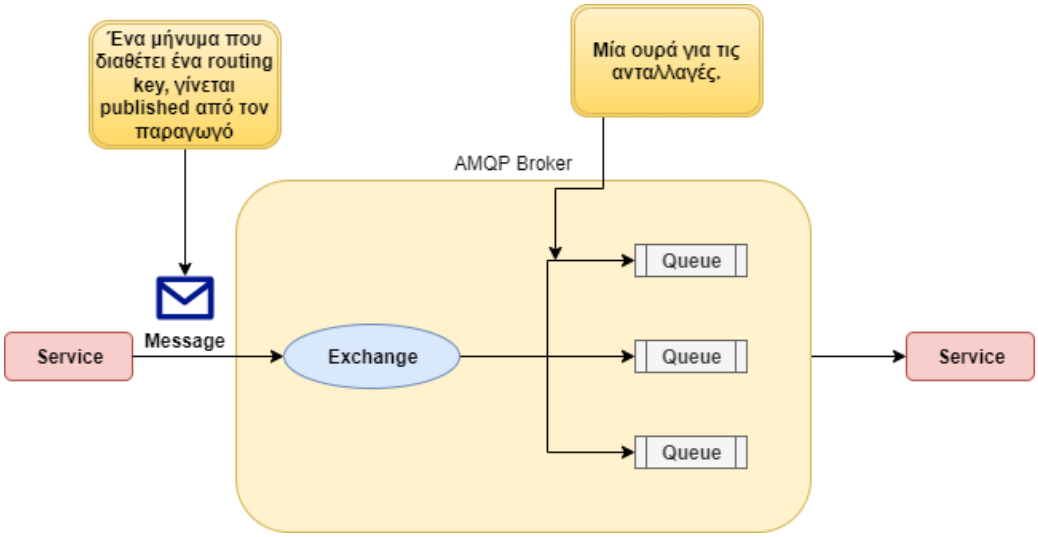
- **Δέσμευση (Binding):** Μία δέσμευση είναι μία συσχέτιση μεταξύ μίας ουράς και μίας ανταλλαγής, που αποτελείται από ένα σετ κανόνων που χρησιμοποιεί η ανταλλαγή για τη δρομολόγηση μηνυμάτων στις ουρές.

- **Μήνυμα και περιεχόμενο:** Ένα μήνυμα είναι μία οντότητα σταλμένη από τον publisher στην ουρά και ύστερα subscribed από τον καταναλωτή (consumer). Κάθε μήνυμα περιέχει ένα σετ από τίτλους που καθορίζουν τις ιδιότητες, όπως τη διάρκεια ζωής, την αντοχή και την προτεραιότητα.

- **Σύνδεση (Connection):** Μία σύνδεση στο AMQP είναι ένα δίκτυο διασυνδέσεων μεταξύ της εφαρμογής και του AMQP Broker, π.χ. μία TCP/IP σύνδεση.

- **Κανάλι (Channel):** Ένα κανάλι είναι μία εικονική σύνδεση μεταξύ 2 AMQP εφαρμογών. Το publish και το consuming ενός μηνύματος από ή προς μία ουρά, εκτελείται μέσω ενός καναλιού. Ο αριθμός των καναλιών αυξάνεται, μία σύνδεση, μπορεί να έχει πολλαπλά κανάλια.

- **Virtual Hosts:** Οι Virtual Hosts παρέχουν τρόπους διαχωρισμού των εφαρμογών στον Broker. Διαφορετικοί χρήστες, μπορούν να έχουν διαφορετικά προνόμια σύνδεσης σε διαφορετικούς Virtual Hosts. Ουρές και ανταλλαγές φτιάχνονται ώστε να μπορούν να υπάρχουν σε έναν μόνο Virtual Host.[21]



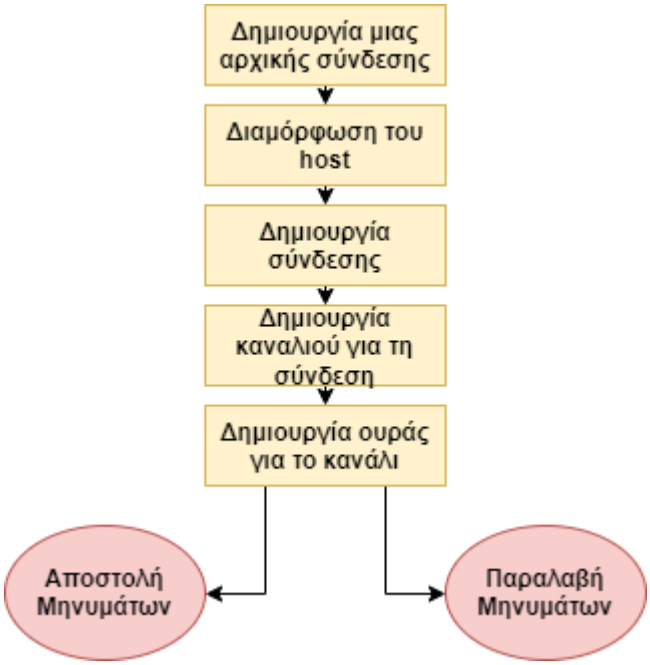
Διάγραμμα 10: Παράδειγμα λειτουργίας AMQP

3.1.4.2. RabbitMQ

Το RabbitMQ είναι ο πιο ευρέως ανεπτυγμένος broker ανοιχτού κώδικα, χρησιμοποιείται παγκοσμίως από μικρές, νεοσύστατες μέχρι και μεγάλες επιχειρήσεις. Είναι ένα εύκολο σύστημα για να αναπτυχθεί σε κάποιο κτίριο ή στο νέφος. Το RabbitMQ έχει μερικά βασικά χαρακτηριστικά που συμβάλλουν στην διαδεδομένη χρήση του.

Για αρχή, οι servers του RabbitMQ σε ένα τοπικό δίκτυο μπορούν να ομαδοποιηθούν, δημιουργώντας έναν μεγάλο broker. Έτσι παρουσιάζονται χαρακτηριστικά όπως εξισορρόπηση φόρτου και ανεκτικότητα σφάλματος.

Ένα ακόμα βασικό χαρακτηριστικό του RabbitMQ, είναι η χρήση του πρωτοκόλλου AMQP, όπου δέχεται συνδέσεις μεταξύ διαφορετικών platforms. Για παράδειγμα μπορεί να δεχτεί message queuing πρωτόκολλα, όπως το MSMQ, με τη χρήση C#, και το STOMP, με τη χρήση Ruby. Η εκτέλεση μίας RabbitMQ εφαρμογής, φαίνεται στο Διάγραμμα 11.



Διάγραμμα 11 Παράδειγμα λειτουργίας μιας εφαρμογής με τη χρήση RabbitMQ

3.2. Συσκευές IoT

Οι συσκευές IoT είναι συσκευές υλικού, που ποικίλλουν από αισθητήρες, μικροσυσκευές, οικιακές συσκευές και άλλες μηχανές, οι οποίες συλλέγουν και ανταλλάζουν δεδομένα μέσω του Internet. Είναι προγραμματισμένες για συγκεκριμένες χρήσεις και μπορούν να ενσωματωθούν και με άλλες συσκευές IoT. Για παράδειγμα, μία συσκευή IoT μέσα στο αμάξι μπορεί να αναγνωρίσει την κίνηση που υπάρχει στο δρόμο, και να στείλει ένα αυτοματοποιημένο μήνυμα στο άτομο που πρόκειται να συναντηθεί με τον οδηγό για την καθυστέρησή του.

Από την πλευρά διαχείρισης δικτύου, μία συσκευή μπορεί να θεωρηθεί ως ένας κόμβος που έχει διεύθυνση δικτύου και θα μπορούσε να είναι μέρος μιας συλλογής διασυνδεδεμένων συσκευών. Θα μπορούσαν να υπάρχουν πολλαπλά endpoints διευθύνσεων δικτύου ανά φυσική συσκευή. Επιπλέον, αν έχουμε πολλά network interfaces, η ίδια συσκευή μπορεί να εμφανιστεί ως πολλοί κόμβοι σε άλλες συσκευές.

3.2.1. Παραδείγματα IoT συσκευών

- **Βιομηχανική ασφάλεια και ασφάλεια:** Συστήματα ασφαλείας IoT, αισθητήρες και κάμερες μπορούν να εγκατασταθούν σε απαγορευμένες περιοχές για να ανιχνεύουν τους παραβάτες. Επίσης, μπορούν να αναγνωρίσουν την πίεση που δημιουργείται και μικρές διαρροές επικίνδυνων χημικών και να επιδιορθώσουν τις βλάβες πριν γίνουν σημαντικά προβλήματα.[16]

- **Παρακολούθηση δραστηριότητας:** Είναι συσκευές οι οποίες παρακολουθούν και ανιχνεύουν μετρήσεις που έχουν να κάνουν με την υγεία και τον αθλητισμό, όπως η απόσταση που έτρεξε ή περπάτησε κάποια/ος, την κατανάλωση θερμίδων, τους

παλμούς της καρδιάς κ.α. Είναι ένας τύπος φορετού υπολογιστή και σαν όρος, χρησιμοποιείται κυρίως για τα Smartwatches/Smart bands που είναι συγχρονισμένα ασύρματα με τον υπολογιστή ή το Smartphone για μακροχρόνια ανίχνευση δεδομένων.[17]

- **Οικιακή ασφάλεια:** Ο κύριος τρόπος ασφάλειας σε ένα σπίτι είναι το IoT. Με αισθητήρες, φωτισμό, συναγερμούς και κάμερες, τα οποία μπορούν να ελεγχθούν μέσω του Smartphone. Συνδέονται μέσω IoT για 24ωρη ασφάλεια.

- **Ανίχνευση Κίνησης:** Αισθητήρες κίνησης μπορούν να ανιχνεύσουν δονήσεις σε κτήρια, γέφυρες, φράγματα και άλλες μεγάλες κατασκευές. Αυτές οι συσκευές μπορούν να αναγνωρίσουν ανωμαλίες και ενοχλήσεις σε δομές που θα μπορούσαν να οδηγήσουν σε καταστροφικές συνέπειες.

- **Γυαλιά επαυξημένης πραγματικότητας (AR):** Είναι γυαλιά τα οποία μπορούν να βοηθήσουν το χρήστη να λάβει πληροφορίες, μέσω 3D animations και βίντεο, προσαρμόζοντάς τα σε πραγματικές σκηνές. Η πληροφορία παρουσιάζεται μέσα στους φακούς των γυαλιών και δίνουν πρόσβαση σε διαδικτυακές εφαρμογές.

Μερικές από τις πιο διάσημες συσκευές IoT, είναι το Google Home Voice Controller, το οποίο δίνει τη δυνατότητα στο χρήστη να ελέγξει υπηρεσίες όπως συναγερμούς, φώτα, θερμοστάτες, τον ήχο κ.α.

Ένα καλό παράδειγμα για τη Χρήση IoT συσκευών, είναι ένα Smart Home. Δηλαδή, ένα τέτοιο σπίτι, θα μπορούσε να διαθέτει από αυτοματοποιημένες κλειδαριές, μέχρι και αυτόματες οικιακές συσκευές.

4. IOT Frameworks

Ένα IoT Framework είναι ένα ενδιάμεσο λογισμικό πάνω στο οποίο λειτουργούν οι IoT εφαρμογές. Είναι ένα σημαντικό κομμάτι του IoT, το οποίο προωθεί και συνδέει όλα τα στοιχεία στον σχεδιασμό του IoT. Επιτρέπει τη διαχείριση συσκευών, χειρίζεται τα πρωτόκολλα επικοινωνιών σε επίπεδα λογισμικού και υλικού, συλλέγει και αναλύει πληροφορίες και βελτιώνει τη ροή πληροφοριών και τις λειτουργίες των εφαρμογών. Παρακάτω θα αναφερθούμε σε μερικά από τα πιο σημαντικά IoT Frameworks.

Τα IoT Frameworks έχουν 4 κύριους στόχους σχεδιασμού:

i. Τη μείωση του χρόνου ανάπτυξης και να φέρουν IoT λύσεις συντομότερα.
ii. Τη μείωση της πολυπλοκότητας της ανάπτυξης και της διαχείρισης ενός IoT δικτύου
iii. Τη βελτίωση της φορητότητας και τηςς διαλειτουργικότητας των εφαρμογών

iv. Τη βελτίωση της λειτουργικότητας, της αξιοπιστίας και της δυνατότητας συντήρησης. [22]

4.1. AWS IoT

AWS (Amazon Web Services) IoT είναι μία πλατφόρμα νέφους για το IoT από την Amazon. Αυτό το framework έχει ως στόχο την εύκολη σύνδεση και ασφαλή αλληλεπίδραση των έξυπνων συσκευών με το AWS cloud και άλλες συνδεδεμένες συσκευές. Δίνει τη δυνατότητα να χρησιμοποιηθεί μαζί με άλλες υπηρεσίες της Amazon, όπως το DynamoDB, Amazon S#, Amazon Machine Learning κ.α. Άλλη μία δυνατότητα που δίνει το AWS IoT είναι ότι επιτρέπει στις εφαρμογές να επικοινωνούν με τις συσκευές ακόμα και όταν είναι offline.[17]

4.2. ARM mbed IoT

Η ARM mbed IoT είναι μία πλατφόρμα για την ανάπτυξη εφαρμογών για ARM microcontrollers βασισμένα σε IoT. [23] Παρέχει όλες τις απαιτήσεις μέσω του οικοσυστήματός του για να δημιουργήσει είτε αυτόνομες IoT εφαρμογές, είτε δικτυωμένες. [24] Στοχεύει στην παροχή κλιμακούμενου, συνδεδεμένου και ασφαλούς περιβάλλοντος για τις IoT συσκευές, ενσωματώνοντας mbed εργαλεία και υπηρεσίες, ARM Microcontrollers, mbed OS, mbed Device Connector, και mbed Cloud.

Το ARM mbed IoT Framework, έχει το πλεονέκτημα σε σχέση με πολλά άλλα Frameworks, αφού παρέχει ένα κοινό θεμέλιο λειτουργικού συστήματος για την ανάπτυξη IoT. Υποστηρίζει τα πιο σημαντικά πρωτόκολλα επικοινωνίας για τη σύνδεση των συσκευών μεταξύ τους και με το cloud. Επιπλέον, υποστηρίζει αυτοματοποιημένη διαχείριση ενέργειας, έτσι ώστε να λύσει το πρόβλημα κατανάλωσης ενέργειας.

4.3. Kaa IoT

Το Kaa είναι ένα εξαιρετικά ευέλικτο ενδιάμεσο λογισμικό πολλαπλών χρήσεων. Είναι μία πλατφόρμα για την εφαρμογή ολοκληρωμένων IoT συστημάτων, συνδεδεμένων εφαρμογών και έξυπνων προϊόντων. Τα χαρακτηριστικά που προσφέρει, περιλαμβάνουν τη διαχείριση συσκευών, συλλογή δεδομένων, διαχείριση παραμέτρων, επικοινωνία με μηνύματα κ.α.[28]

4.4. Node-RED

Το Node-RED είναι ένα προγραμματιστικό εργαλείο για την καλωδίωση hardware συσκευών, APIs και διαδικτυακών υπηρεσιών με νέους, ενδιαφέροντες τρόπους. Παρέχει έναν editor, ο οποίος είναι προσβάσιμος από browser. Χρησιμοποιεί τον γραφικό προγραμματισμό ως μοντέλο λειτουργίας, καθιστώντας εύκολη τη χρήση του σε νέους και άπειρους προγραμματιστές. Διαθέτει μία μεγάλη ποικιλία από βιβλιοθήκες και χρησιμοποιείται από ιδιωτικής χρήσης, μέχρι και σε μεγάλες εταιρίες.

4.5. ZettaJS

Το Zetta είναι ένα framework ανοικτού κώδικα, το οποίο είναι φτιαγμένο πάνω στη Node.js, για τη δημιουργία server IoT, οι οποίοι τρέχουν εγκάρσια σε γεω-διανεμημένους υπολογιστές και στο cloud. Το Zetta συνδυάζει τα REST APIs, τα Web Sockets και διαδραστικό προγραμματισμό, το οποίο δίνει τη δυνατότητα της δημιουργίας πολλών συσκευών σε πραγματικό χρόνο.[30]

5. Υλοποίηση Εφαρμογής

5.1. Σκοπός της εφαρμογής

Ο σκοπός της εφαρμογής είναι η χρήση διαφόρων πρωτοκόλλων για τη συλλογή δεδομένων IoT από αισθητήρες και η αποθήκευσή τους σε μία βάση δεδομένων. Η χρήση αυτής της εφαρμογής είναι πολύ σημαντική για τον έλεγχο δεδομένων και για την παρακολούθηση της λειτουργίας διαφόρων συστημάτων. Για παράδειγμα, μπορεί να εφαρμοστεί σε ένα κτίριο για την παρακολούθηση της θερμοκρασίας σε κάθε χώρο ή σε ένα θερμοκήπιο για την παρακολούθηση της υγρασίας σε κάθε φυτό. Αυτό μπορεί να επεκταθεί και να λύσει σημαντικά προβλήματα στην κατανάλωση ενέργειας και κατ' επέκταση στην οικονομία, αφού μπορεί να γίνει αυτοματοποίηση των εργαλείων όπως θερμοστάτες, ποτιστικά, συστήματα ασφαλείας κ.α.. Σε αυτήν την πτυχιακή θα δοθεί ένα παράδειγμα, το οποίο λαμβάνει υπόψιν μία καινούρια Υπουργική Απόφαση που αφορά τη μείωση της κατανάλωσης ηλεκτρικής ενέργειας. Για την παρουσίαση της εφαρμογής θα χρησιμοποιηθούν ενδεικτικά τα κτίρια της σχολής Θετικών Επιστημών του Πανεπιστημίου Θεσσαλίας στην Λαμία.

5.2. Περιγραφή της εφαρμογής

Στα πλαίσια της πτυχιακής εργασίας, αναπτύχθηκε μία εφαρμογή για τη συλλογή και αποθήκευση δεδομένων IoT, όπως θερμοκρασία, υγρασία κ.α., με τη χρήση των τριών πρωτοκόλλων επικοινωνίας: MQTT, HTTP, AMQP. Η υλοποίηση της εφαρμογής έγινε σε γλώσσα JavaScript στο περιβάλλον Node.js και η αποθήκευση δεδομένων, γίνεται στη βάση PostgreSQL. Η υλοποίηση έχει γίνει στο backend κομμάτι και είναι δυνατή η επέκτασή της, ώστε να γίνει μία ολοκληρωμένη εφαρμογή πάνω στην παρακολούθηση και τη διαχείριση δεδομένων, από ένα smart σπίτι, μέχρι και σε μεγάλα και γραφεία και κτίρια.

5.3. Ανάλυση της εφαρμογής

Θα αναλυθεί παρακάτω ο τρόπος της ανάπτυξης της εφαρμογής, τα εργαλεία που χρησιμοποιήθηκαν και το που θα μπορούσε να εφαρμοστεί. Περιέχονται τρία κομμάτια back-end τα οποία χωρίζονται ανάλογα με τα πρωτόκολλα. Θα παρουσιαστούν ορισμένες εικόνες και κομμάτια κώδικα για να κατανοηθεί η λειτουργία της εφαρμογής.

5.3.1. Βάση δεδομένων PostgreSQL

Το σύστημα βάσης δεδομένων που χρησιμοποιείται για την αποθήκευση των δεδομένων, είναι η PostgreSQL. Η PostgreSQL είναι ένα ισχυρό object-relational σύστημα βάσης δεδομένων, το οποίο χρησιμοποιεί τη και επεκτείνει τη γλώσσα SQL σε συνδυασμό με πολλά χαρακτηριστικά τα οποία με ασφάλεια αποθηκεύουν δεδομένα τα οποία είναι

περίπλοκα και σε μεγάλο μέγεθος. Είναι αρκετά εύκολο να χρησιμοποιηθεί και ιδιαίτερα στον τομέα της λήψης δεδομένων, είναι πολύ χρήσιμη, αφού έχει ένα μεγάλο εύρος χαρακτηριστικών για τη βοήθεια της σωστής και ασφαλούς αποθήκευσής τους σε αυτή.[26]

Η δομή που χρησιμοποιήθηκε για την υλοποίησή της, αποτελείται από ένα σχήμα τριών πινάκων, οι οποίοι μπορούν να παραμετροποιηθούν για να δημιουργηθεί κάποιο σύστημα συλλογής δεδομένων οπουδήποτε.

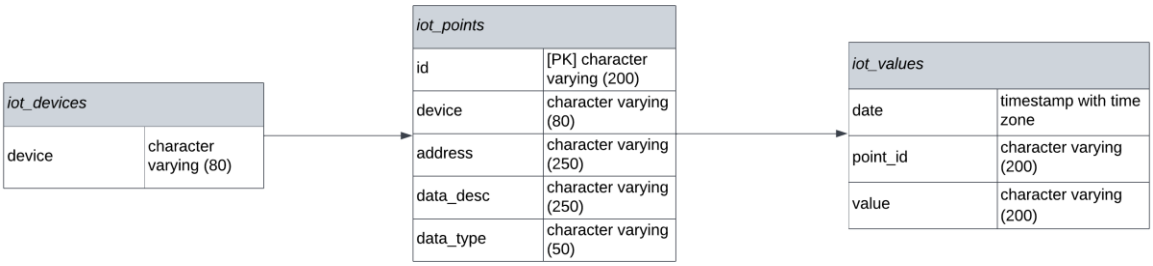
Οι πίνακες που δημιουργήθηκαν είναι οι εξής:

- **iot_devices:** Σε αυτόν τον πίνακα, μπορούν να προστεθούν οι συσκευές οι οποίες χρησιμοποιούνται στο κάθε σύστημα.

- **iot_points:** Σε αυτόν τον πίνακα υπάρχουν ορισμένες στήλες για την περιγραφή των χαρακτηριστικών των συσκευών από τις οποίες λαμβάνονται τα δεδομένα. Το id είναι το χαρακτηριστικό με το οποίο ξεχωρίζονται οι συσκευές μεταξύ τους. Στο device προστίθεται ο τύπος της συσκευής. Στο address, αναφέρεται η διεύθυνση IP, το πρωτόκολλο και το port μέσω των οποίων γίνεται η επικοινωνία του client με τους αισθητήρες. Στο data_desc περιλαμβάνονται ο τύπος του δεδομένου που λαμβάνεται (θερμοκρασία, υγρασία, πίεση κ.α.) και αν χρειάζεται, η τοποθεσία στην οποία βρίσκονται οι συσκευές (Θερμοκρασία σε εξωτερικό χώρο). Τέλος, το data_type είναι ο τύπος της μεταβλητής που χρησιμοποιείται (int, float, double κλπ.).

- **iot_values:** Σε αυτόν τον πίνακα προστίθενται τα δεδομένα τα οποία λαμβάνονται από τις συσκευές. Στη στήλη date, δημιουργείται ένα timestamp για την παρακολούθηση του χρόνου στον οποίο λήφθηκε το δεδομένο, το point_id το οποίο περιέχει το id της συσκευής του πίνακα iot_points έτσι ώστε ξεχωρίζεται η συσκευή που έστειλε τα δεδομένα και τέλος το value, η οποία είναι η στήλη στην οποία αποθηκεύονται τα δεδομένα των συσκευών.

Παρακάτω δίνεται η δομή της βάσης δεδομένων.



Διάγραμμα 12: Δομή της βάσης δεδομένων

5.3.2. MQTT Publisher-Subscriber-Broker

Το πρώτο κομμάτι του framework έχει αναπτυχθεί με τη χρήση του πρωτοκόλλου MQTT, χωρίζεται σε τρία αρχεία JavaScript, το broker.js, publisher.js, subscriber.js και με τη χρήση βιβλιοθηκών, όπως mqtt.js, mosca.js, pg.js. Λειτουργεί ως εξής :

Τρέχοντας το broker.js ξεκινάει η επικοινωνία του προγράμματος με τη βάση δεδομένων και ανοίγει ένας broker, μέσω του οποίου θα στέλνονται και θα λαμβάνονται τα δεδομένα, και θα αποθηκεύονται μέσα στη βάση.

```
C:\Users\kimon\Desktop\πτυχιακη\Project\MQTT\SUB-PUB-BROKER-TESTS\MQTT_MOSCA
BROKER_SUB_PUB>node broker
Broker is online
Connected to Postgresql
```

Εικόνα 1: Τρέχοντας το broker.js

Ύστερα, ο broker περιμένει το publisher.js και το subscriber.js, για να ξεκινήσει η αποστολή και η λήψη δεδομένων. Τρέχοντας λοιπόν και τα άλλα δύο προγράμματα, γίνεται η σύνδεση τους με τον broker. Ο publisher ξεκινάει να κάνει publish τα μηνύματα στον broker και ο subscriber ξεκινάει να κάνει subscribe στα topics που βρίσκονται μέσα τα μηνύματα.

```
C:\Users\kimon\Desktop\πτυχιακη\Project\MQTT\SUB-PUB-BROKER-TESTS\MQTT_MOSCABR
OKER_SUB_PUB>node pub
sending temperature: {"value":"19.47"}
sending temperature: {"value":"20.10"}
sending temperature: {"value":"24.41"}
sending temperature: {"value":"23.16"}
^C
```

Εικόνα 2: Τρέχοντας το pub.js

Όταν γίνει η σύνδεση μεταξύ του broker και του publisher, ο broker λαμβάνει τα δεδομένα και τα αποθηκεύει απευθείας στη βάση δεδομένων.

```
{ value: '24.00' }
Data inserted[object Object]
{ value: '25.62' }
Data inserted[object Object]
{ value: '20.73' }
Data inserted[object Object]
```

Εικόνα 3: Το terminal στην λήψη των δεδομένων.

5.3.3. HTTP service

Στο δεύτερο κομμάτι του framework χρησιμοποιείται το πρωτόκολλο HTTP. Έχει δημιουργηθεί ένα api και μία ενδεικτική σελίδα, τρέχοντας την εφαρμογή λοιπόν, είναι δυνατή η πρόσβαση σε ένα τοπικό δίκτυο μέσα στο οποίο μπορούν να σταλθούν δεδομένα και να αποθηκευτούν στη βάση με τη μέθοδο POST. Τα αρχεία που δημιούργησα είναι τα εξής :

- **app.js**: στο οποίο ξεκινάει ο server και κάνει listen σε ένα συγκεκριμένο port για να αρχίσει η επικοινωνία.
- **querries.js**: στο οποίο γίνεται η σύνδεση με τη βάση PostgreSQL με τη χρήση της βιβλιοθήκης pg.js και η εισαγωγή των δεδομένων στον πίνακα.

- **index.html**: το οποίο δημιουργεί μία σελίδα στην οποία έχω βάλει ένα button, όπου αν πατηθεί δημιουργεί και στέλνει τα δεδομένα στον server ώστε να τα εισχωρήσει στη βάση.
- **client.js**: όπου είναι το backend κομμάτι της επικοινωνίας και αναγνωρίζει πότε έγινε το click στο button έτσι ώστε να ακούσει ο server ότι πρέπει να ξεκινήσει την αποστολή των δεδομένων.

Το api τρέχει με τη χρήση της Express, ένα εύκολο και εύχρηστο framework για τη node.js. Είναι σχεδιασμένο για τη δημιουργία api, web και mobile applications στο κομμάτι του server.[27]

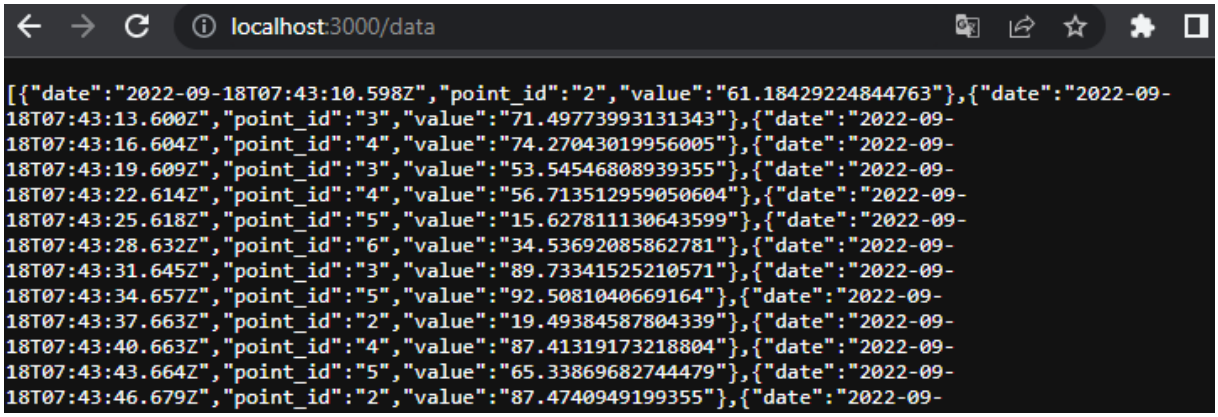
Ανοίγοντας τον server, μπορούμε να μπούμε μέσω του browser στην τοπική σελίδα.

```
C:\Users\kimon\Desktop\πτυχιακη\Project\HTTP\HTTP-SEQUELIZE-EXPRESS\api>nodemon start
[nodemon] 2.0.16
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node start app.js`
Server side code running
App running on port 3000
```

Εικόνα 4: Τρέχοντας τον κώδικα, γίνεται η επικοινωνία με το port

Εικόνα 5: Η ενδεικτική σελίδα

Μπορούμε επίσης, να κάνουμε ένα GET request μεταβαίνοντας στη διεύθυνση «http://localhost:3000/data» όπου εμφανίζει τον πίνακα iot_values με όλα τα δεδομένα και τις τιμές που έχουν καταχωριστεί στη βάση.



Εικόνα 6: Το αποτέλεσμα της μεθόδου GET

5.3.4. AMQP RabbitMQ Broker, Client-Server

Στο τελευταίο κομμάτι του framework έχει υλοποιηθεί η επικοινωνία με το πρωτόκολλο AMQP. Μιας και το AMQP είναι το πρωτόκολλο που παρέχει τη μεγαλύτερη ασφάλεια, έχει δοθεί μεγαλύτερη βάση σε αυτό. Αποτελείται από δύο κομμάτια, τον Client και τον Server, και είναι αρκετά παραμετροποιήσιμο, ώστε να μπορέσει να λειτουργήσει σε πολλά συστήματα.

Έχει δημιουργηθεί ένας Client και ένας Server και η επικοινωνία μεταξύ τους γίνεται μέσω του broker RabbitMQ. Ο Client αποτελείται από τα εξής αρχεία :

- client.js: για τη δημιουργία της επικοινωνίας με το port και για τη λήψη των δεδομένων και την εισχώρησή τους στη βάση δεδομένων.
- amqp.js: το οποίο χρησιμοποιεί τη βιβλιοθήκη amqplib,js, η οποία δίνει τη δυνατότητα της σύνδεσης στον broker καθώς και τη δυνατότητα λήψης και αποστολής των μηνυμάτων, χρησιμοποιείται και στο κομμάτι του server και στο κομμάτι του client, καθώς υπάρχει για πολλές χρήσεις.
- amqpOpt.js: στο οποίο γίνεται η σύνδεση με τον broker με τη μέθοδο connect() και σε περίπτωση που δεν επιτευχθεί σύνδεση, ξαναγίνονται προσπάθειες μέχρι να γίνει.
- pg.js: στο οποίο γίνεται η σύνδεση με τη βάση δεδομένων, όπου πάλι, σε περίπτωση που δεν γίνει η σύνδεση, ξαναγίνονται προσπάθειες μέχρι να επιτευχθεί.
- utils.js: το οποίο χρησιμοποιείται για να θέσει ο χρήστης τη συχνότητα λήψης δεδομένων από τις συσκευές.

Στο κομμάτι του Server, υπάρχουν τα εξής αρχεία:

- server.js: το οποίο μέσω του amqp.js δημιουργεί την επικοινωνία με τον broker καθώς και στέλνει τα δεδομένα στον client.
- amqp.js: βλέπε παραπάνω

Τρέχοντας τον client λοιπόν, ξεκινάει η διαδικασία της επικοινωνίας και περιμένει τη λήψη των δεδομένων από τον server.

```
C:\Users\kimon\Desktop\πτυχιακη\Project\AMQP\amqp-pg-node\src\client>node client
Connecting to RabbitMQ
Attempting to connect to Postgres
Example app listening on port 3000!
Successfully connected to RabbitMQ
Connected to Postgres
Connected to RabbitMQ
```

Εικόνα 7: Δημιουργία επικοινωνίας με τον broker και τη βάση δεδομένων

Τρέχοντας τον server ξεκινάει η επικοινωνία με τον broker και η αποστολή των δεδομένων. Όπως φαίνεται στην εικόνα 7.

```
C:\Users\kimon\Desktop\πτυχιακη\Project\AMQP\amqp-pg-node\src\server>node server
Connecting to RabbitMQ
connected
sending temperature: {"value":"21.31","timestamp":"2022-09-21T09:17:30.759Z","point_id":"14"}
(node:10548) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usabili
ty issues. Please use the Buffer.alloc(), Buffer.allocUnsafe(), or Buffer.from() methods inst
ead.
(Use `node --trace-deprecation ...` to show where the warning was created)
sending temperature: {"value":"23.47","timestamp":"2022-09-21T09:17:33.766Z","point_id":"5"}
sending temperature: {"value":"25.90","timestamp":"2022-09-21T09:17:36.767Z","point_id":"13"}
sending temperature: {"value":"24.15","timestamp":"2022-09-21T09:17:39.778Z","point_id":"7"}
```

Εικόνα 8: Δημιουργία επικοινωνίας με τον broker και αποστολή δεδομένων.

Συνεχίζοντας, όταν ο client λάβει το πρώτο μήνυμα, ξεκινάει τη λήψη των δεδομένων (consume) και ξεκινάει την αποθήκευσή τους στη βάση.

```
C:\Users\kimon\Desktop\πτυχιακη\Project\AMQP\amqp-pg-node\src\client>node client
Connecting to RabbitMQ
Attempting to connect to Postgres
Example app listening on port 3000!
Successfully connected to RabbitMQ
Connected to Postgres
Connected to RabbitMQ
Recieved: {"value":"20.48","timestamp":"2022-09-21T09:39:25.415Z","point_id":"5"}
Inserted
Recieved: {"value":"17.34","timestamp":"2022-09-21T09:39:28.418Z","point_id":"6"}
Inserted
Recieved: {"value":"13.28","timestamp":"2022-09-21T09:39:31.427Z","point_id":"12"}
Inserted
Recieved: {"value":"21.97","timestamp":"2022-09-21T09:39:34.429Z","point_id":"7"}
Inserted
Recieved: {"value":"18.44","timestamp":"2022-09-21T09:39:37.433Z","point_id":"16"}
Inserted
```

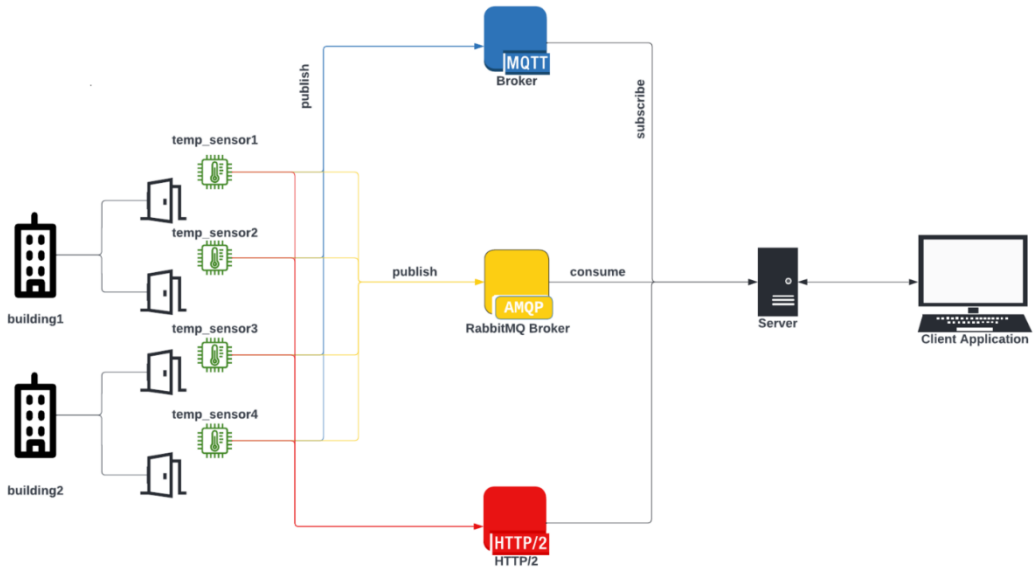
5.4. Παρουσίαση της πιλοτικής εφαρμογής

Θα παρουσιαστεί το παράδειγμα της χρήσης του Framework σε ένα σύστημα παρακολούθησης θερμοκρασίας στα κτίρια του Πανεπιστημίου Θεσσαλίας στην περιοχή της Λαμίας. Σύμφωνα με την υπουργική απόφαση που τίθεται σε ισχύ από την τωρινή χειμερινή περίοδο, κάθε δημόσιο κτήριο θα πρέπει να σταθεροποιεί τις θερμοκρασίες του στους 19°C, κατά τη χειμερινή περίοδο, και στους 27°C, κατά τη διάρκεια της καλοκαιρινής περιόδου. Αυτό θα έχει ως αποτέλεσμα την μείωση της κατανάλωσης ρεύματος και πετρελαίου κατά 10%, επομένως μόνο από αυτόν τον έλεγχο, μπορεί να γίνει μία σημαντική μείωση στα έξοδα λόγω κατανάλωσης ενέργειας.

Η διαμόρφωση του συστήματος έχει ως εξής, στο κομμάτι του front-end, θα δημιουργηθεί μία ιστοσελίδα/εφαρμογή, στην οποία ο χρήστης θα έχει τη δυνατότητα να επιλέξει τη διαμόρφωση που θέλει για να λειτουργήσει το σύστημα, στον χώρο που θέλει να το εγκαταστήσει. Για παράδειγμα, θα δίνεται η δυνατότητα στο χρήστη, να καθορίσει τους χώρους που θέλει να γίνεται η παρακολούθηση, τα πρωτόκολλα που θέλει να χρησιμοποιήσει, να καθορίσει τη συχνότητα λήψης των δεδομένων κ.α. Επιπροσθέτως, είναι εφικτή η δημιουργία ενός αυτοματοποιημένου σύστημα θερμοστάτη, ώστε να διατηρούνται οι θερμοκρασίες στα σημεία που έχει ορίσει.

Η λειτουργία του συστήματος γίνεται μέσω της επικοινωνίας του server με τον κάθε αισθητήρα που έχει εγκατασταθεί σε συγκεκριμένους χώρους ενός κτιρίου. Ξεκινάει η επικοινωνία μεταξύ αισθητήρα και server, και ανάλογα με την επιλογή του πρωτοκόλλου, στέλνονται τα δεδομένα και δρομολογούνται από τον server στη βάση δεδομένων.

Στο διάγραμμα 13 φαίνεται η χρήση ενός συστήματος παρακολούθησης θερμοκρασίας.



Διάγραμμα 13: Λειτουργία συστήματος με αισθητήρες θερμοκρασίας

Για τα δεδομένα του κτιρίου της σχολής θετικών επιστημών του Πανεπιστημίου Θεσσαλίας, έχει δημιουργηθεί ο ανάλογος πίνακας, ο οποίος δείχνει τους αισθητήρες θερμοκρασίας που υπάρχουν σε κάθε χώρο του κτιρίου. Στον πίνακα iot_points της εφαρμογής έχουν δημιουργηθεί δεδομένα, τα οποία δείχνουν τον κάθε αισθητήρα, τον χώρο στον οποίο βρίσκεται, καθώς και το id για να είναι δυνατή η αναγνώριση του αισθητήρα, από τον οποίο γίνεται η λήψη των δεδομένων, όπως φαίνεται στην εικόνα 9.

	id [PK] character varying (200)	device character varying (80)	address character varying (250)	data_desc character varying (250)	data_type character varying (50)
1	1	temp_sensor	mqtt:192.168.1.31:8080	temperature_hall1	float
2	10	temp_sensor	http:192.168.1.32:8080	temperature_control_r...	float
3	11	temp_sensor	mqtt:192.168.1.33:8080	temperature_auditoriu...	float
4	12	temp_sensor	mqtt:192.168.1.34:8080	temperature_auditoriu...	float
5	13	temp_sensor	mqtt:192.168.1.35:8080	temperature_computer...	float
6	14	temp_sensor	mqtt:192.168.1.36:8080	temperature_computer...	float
7	15	temp_sensor	mqtt:192.168.1.37:8080	temperature_computer...	float
8	16	temp_sensor	mqtt:192.168.1.38:8080	temperature_computer...	float
9	17	temp_sensor	amqp:192.168.1.55:50...	temperature_biology_lab	float
10	18	humidity_sensor	http:192.168.1.11:3000	humidity_building1	float
11	19	humidity_sensor	http:192.168.1.13:3000	humidity_building2	float
12	2	temp_sensor	amqp:192.168.1.70:50...	temperature_office1	float
13	20	humidity_sensor	http:192.168.1.14:3000	humidity_auditorium1	float
14	21	humidity_sensor	http:192.168.1.15:3000	humidity_auditorium2	float
15	3	temp_sensor	amqp:192.168.1.72:50...	temperature_office2	float
16	4	temp_sensor	amqp:192.168.1.71:50...	temperature_secretary...	float
17	6	temp_sensor	mqtt:192.168.1.80:8080	temperature_classroo...	float
18	7	temp_sensor	mqtt:192.168.1.81:8080	temperature_classroo...	float
19	8	temp_sensor	mqtt:192.168.1.82:8080	temperature_classroo...	float
20	9	temp_sensor	mqtt:192.168.1.83:8080	temperature_classroo...	float

Εικόνα 9: Πίνακας iot_points

Στη συνέχεια, αφού γίνει η λήψη των δεδομένων από τους αισθητήρες, αυτά αποθηκεύονται στη βάση δεδομένων, στον πίνακα iot_values, όπως φαίνεται στην εικόνα 9.

	date timestamp with time zone	point_id character varying (200)	value character varying (200)
1	2022-09-25 02:38:38.389...	14	23.87
2	2022-09-25 02:38:38.607...	6	15.91
3	2022-09-25 02:38:38.872...	21	19.46
4	2022-09-25 02:38:39.171...	9	26.84
5	2022-09-25 02:38:39.448...	6	27.22
6	2022-09-25 02:38:39.676...	12	21.80
7	2022-09-25 02:38:39.971...	9	21.74
8	2022-09-25 02:38:40.212...	14	18.68
9	2022-09-25 02:38:40.467...	6	23.62
10	2022-09-25 02:38:40.734...	5	24.25

Εικόνα 10: Πίνακας iot_values με ενδεικτικά δεδομένα.

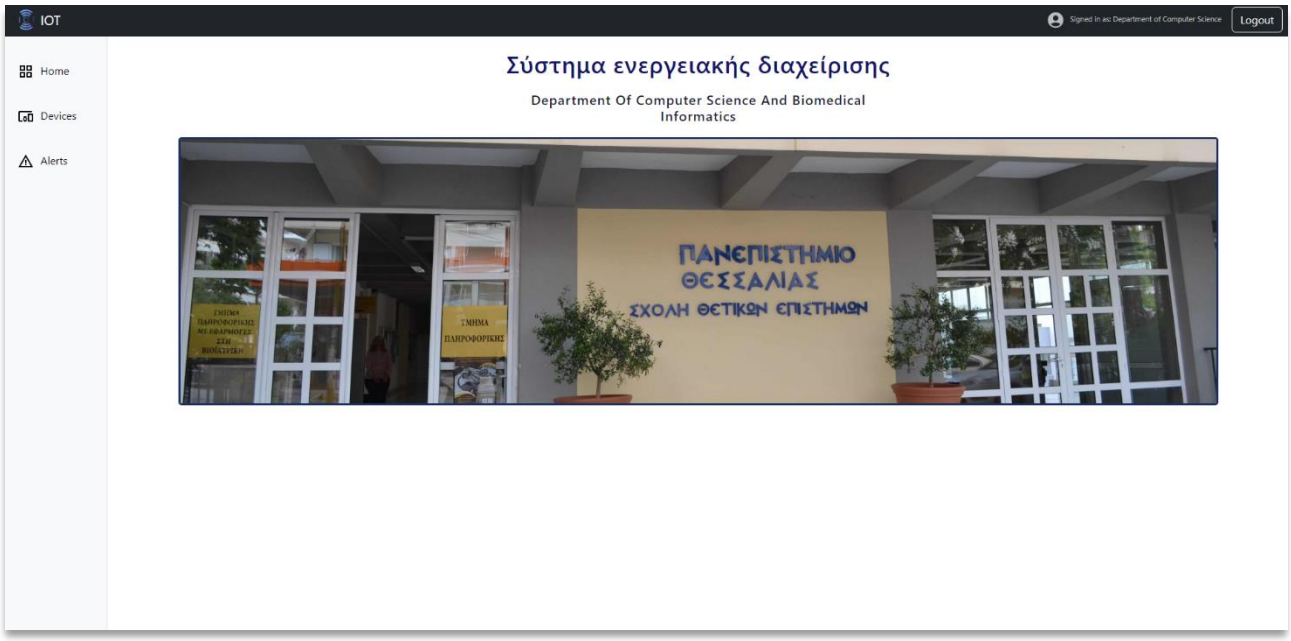
Στο κομμάτι του front-end, έχει δημιουργηθεί μία ενδεικτική σελίδα, από έναν συνάδελφο, για την παρουσίαση της πιλοτικής εφαρμογής στο κτίριο της σχολής. Μέσω του back-end που δημιούργησα, και του front-end κομματιού του συναδέλφου, μπορεί να υλοποιηθεί μία ολοκληρωμένη demo εφαρμογή, η οποία λειτουργεί ως εξής:

- Δίνεται η δυνατότητα να δημιουργηθούν και να αφαιρεθούν αίθουσες στις οποίες πραγματοποιείται ο έλεγχος της θερμοκρασίας.

- Ανάλογα με τα δεδομένα που λαμβάνονται και εντάσσονται στην βάση δεδομένων, μπορούν να δημιουργηθούν χρονικά γραφήματα για τον περιοδικό έλεγχο των δεδομένων.

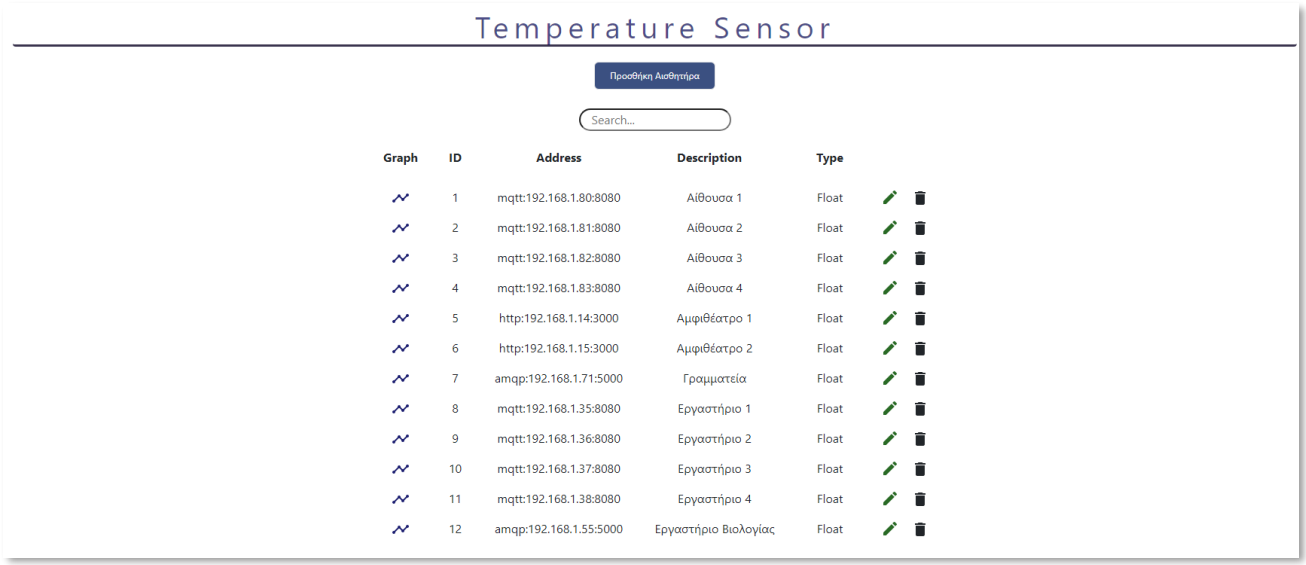
- Σύμφωνα με την υπουργική απόφαση πάνω στην οποία δημιουργήθηκε η πιλοτική εφαρμογή, οι θερμοκρασίες την περίοδο του καλοκαιριού πρέπει να διατηρούνται στους 27^{o}C και τον χειμώνα στους 19^{o}C, επομένως, σε περίπτωση που ξεφεύγουν οι θερμοκρασίες από τα όρια, ο χρήστης ενημερώνεται, έτσι ώστε να ληφθούν τα αντίστοιχα μέτρα.

Πιο συγκεκριμένα, η αρχική οθόνη της πιλοτικής εφαρμογής, για το κτίριο του Τμήματος Πληροφορικής με Εφαρμογές στη Βιοϊατρική του Πανεπιστημίου Θεσσαλίας, φαίνεται στην εικόνα 11.

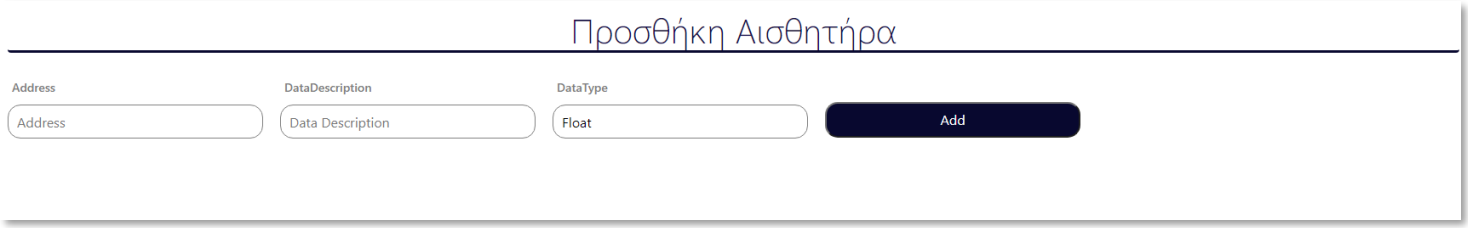


Εικόνα 11: Αρχική σελίδα

Από εκεί ο χρήστης μπορεί να μεταφερθεί στη σελίδα με τις συσκευές και να επιλέξει αν θέλει να χρησιμοποιήσει αισθητήρες θερμοκρασίας ή υγρασίας. Επιλέγοντας λοιπόν μία συσκευή θερμοκρασίας, μεταβαίνει στη σελίδα που εμφανίζεται η κάθε μία από τις συσκευές θερμοκρασίας από τον πίνακα iot_points. Επίσης, μπορεί να προσθέσει, να αφαιρέσει ή να ενημερώσει κάποιον από τους αισθητήρες.



Εικόνα 13: Αισθητήρες θερμοκρασίας (Πίνακας iot_points)



Εικόνα 12: Σελίδα προσθήκης αισθητήρα

Επιλέγοντας έναν συγκεκριμένο αισθητήρα, γίνεται μετάβαση σε μία σελίδα που δίνονται οι πληροφορίες για τον συγκεκριμένο αισθητήρα, καθώς και ένα γράφημα, το οποίο δείχνει τις θερμοκρασίες που έχει μετρήσει ο αισθητήρας στο διάστημα των τελευταίων 24 ωρών. Επιπροσθέτως, φαίνεται μία γραμμή που δείχνει το όριο της θερμοκρασίας για κάθε χρονική περίοδο. Πιο συγκεκριμένα, αν η θερμοκρασία κατά τη διάρκεια της θερινής περιόδου είναι χαμηλότερη από 24^{o}C, τότε βρίσκεται στα πλαίσια εκτός του ορίου, ενώ αν είναι παραπάνω, βρίσκεται σε σωστά επίπεδα.

Εικόνα 14: Η σελίδα κάποιας αίθουσας και το γράφημα των θερμοκρασιών της.

Σε κάθε αίθουσα, έχει προστεθεί ένα πλαίσιο με alerts, το οποίο ενημερώνει τον χρήστη όταν οι θερμοκρασίες βρίσκονται εκτός ορίων. Δίνεται επίσης η δυνατότητα να ελέγξει τα alerts που έχουν γίνει από μία καρτέλα που έχει όλα τα alerts.

Επομένως, με τη χρήση του front-end κομματιού, είναι δυνατή η εγγραφή ενός χρήστη στην εφαρμογή, η προσθήκη αισθητήρων και ο συνεχής έλεγχος για την κατάσταση των θερμοκρασιών των χώρων της σχολής. Το κομμάτι του back-end υλοποιήθηκε για τη δημιουργία επικοινωνίας των αισθητήρων με τον server, τη συλλογή των δεδομένων τους και την αποθήκευση των θερμοκρασιών σε έναν πίνακα στη βάση δεδομένων.

Συνδυάζοντας αυτά τα δύο μοντέλα, δημιουργείται μία ολοκληρωμένη εφαρμογή, η οποία θα έχει πάρα πολλές δυνατότητες. Είναι δυνατή η επέκτασή της ώστε να γίνει εφικτή η αυτοματοποίηση του κλιματισμού για τη διατήρηση των θερμοκρασιών των χώρων σε επιτρεπτά επίπεδα, σύμφωνα με τα δεδομένα τα οποία στέλνονται από την εφαρμογή. Επίσης, θα μπορούσε να χρησιμοποιηθεί από έναν γεωργό στο θερμοκήπιό του για την παρακολούθηση της υγρασίας των φυτών και ταυτόχρονα να δίνεται η δυνατότητα σύνδεσης της εφαρμογής με άλλες εφαρμογές αυτόματου ποτίσματος.

Συμπερασματικά, το framework που δημιουργήθηκε, έχει πολλές δυνατότητες επέκτασης, καθώς και μεγάλη χρησιμότητα, αφού μπορεί να βοηθήσει στην μείωση της κατανάλωσης ενέργειας σε μεγάλο σκέλος. Επίσης, το γεγονός ότι είναι παραμετροποιήσιμη, σημαίνει ότι μπορεί να χρησιμοποιηθεί για ιδιωτική χρήση, μέχρι και σε μεγάλες επιχειρήσεις του δημοσίου, ή και του ιδιωτικού τομέα.

6. Συμπεράσματα

6.1. Σύνοψη

Συμπερασματικά, η χρήση τεχνολογιών IoT είναι πολύ σημαντική για την ανάπτυξη της τεχνολογίας. Το παράδειγμα που έφερα της χρήσης ενός Framework για τον έλεγχο και την αυτοματοποίηση λειτουργιών, όπως ένας θερμοστάτης, ή ο έλεγχος της υγρασίας, με την πιο μαζική χρήση, θα μπορούσε να φέρει σημαντικές αλλαγές στην αύξηση της οικονομίας.

6.2. Προτεινόμενες Επεκτάσεις

Μετά τη δημιουργία του κομματιού back-end του Framework, χρειάζεται και η front-end πλατφόρμα, πάνω στην οποία θα μπορέσει να ολοκληρωθεί το πρόγραμμα. Το front-end κομμάτι του framework έχει ήδη υλοποιηθεί από έναν συνάδελφο, όμως υπάρχουν και άλλες αλλαγές ή προσθήκες, που θα μπορούσαν να αυξήσουν τη λειτουργικότητα του προγράμματος. Πιο συγκεκριμένα, θα ήταν σημαντικό να αναπτυχθούν τρόποι, έτσι ώστε να αντιμετωπιστούν κάποια προβλήματα, όπως:

- *Έλλειψη ασφάλειας*: Θα μπορούσε να δημιουργηθεί ένα περεταίρω κομμάτι στο back-end, με τη χρήση κλειδιών και certificates, ή και πλατφόρμας εγγραφής. Αυτό είναι πολύ σημαντικό να προστεθεί, αφού αυτή η πλατφόρμα θα μπορεί να συλλέγει μέχρι και σημαντικά δεδομένα από χώρους, ή ακόμα και για την προστασία των προσωπικών ευαίσθητων δεδομένων των χρηστών.

- *Προσθήκη πρωτοκόλλων*: Θα ήταν πολύ χρήσιμη η προσθήκη και άλλων ΄πρωτοκόλλων επικοινωνίας για τη λειτουργία της πλατφόρμας. Υπάρχουν πολλές συσκευές, οι οποίες χρησιμοποιούν πρωτόκολλα, τα οποία είναι ειδικά για αυτές , για παράδειγμα το Zigbee, το οποίο χρειάζεται για ασύρματες συσκευές χαμηλού κόστους και ενέργειας για δίκτυα Machine-to-Machine (M2M).

7. Παράρτημα-Πηγαίος Κώδικας

Παράρτημα Α

Για την εκτέλεση και λειτουργία του framework, χρειάζεται η εγκατάσταση και η προσθήκη ορισμένων εφαρμογών και βιβλιοθηκών.

Για την εγκατάσταση της βάσης δεδομένων PostgreSQL, χρειάζεται η λήψη της εφαρμογής pgAdmin και η δημιουργία ενός χρήστη, όπου θα έχει τη δυνατότητα της διαχείρισης της βάσης.

Για τη λήψη PostgreSQL: https://www.postgresql.org/download/

Επίσης, χρειάζεται και η τελευταία έκδοση του node.js, διότι το πρόγραμμα είναι γραμμένο σε αυτό το framework.

Για τη λήψη του node.js: https://nodejs.org/en/download/

Οι βιβλιοθήκες που χρησιμοποιήθηκαν για την υλοποίηση του προγράμματος, είναι:

Για το κομμάτι του MQTT:

- *mosca*: για τη δημιουργία του broker.
- *mqtt*: για τη σύνδεση του publisher και του subscriber με τον broker.
- *mqtt*-packet: για τη σωστή λειτουργία της αποστολής και της λήψης των πακέτων με τα μηνύματα.
- *pg*: Για την δημιουργία σύνδεσης και την επικοινωνία μεταξύ του προγράμματος και της βάσης δεδομένων.

Για την εγκατάσταση των βιβλιοθηκών, χρειάζεται η εντολή:

```
$ npm install mosca mqtt mqtt-packet pg –save
```

Για το κομμάτι του HTTP:

- *express*: για τη διευκόλυνση της δρομολόγησης, τη χρήση του μοντέλου HTTP και την επικοινωνία με το port
- *nodemon*: ένα εργαλείο το οποίο βοηθάει στην ανάπτυξη εφαρμογών βασισμένες στη node.js επανεκκινώντας αυτόματα την εφαρμογή, όταν υπάρχουν αλλαγές στα αρχεία του κώδικα.
- *pg*: αναφέρθηκε παραπάνω.

```
$ npm install express nodemon pg –save
```

Για το κομμάτι του AMQP:

- *amqp*: είναι ένας client για το RabbitMQ (και πιθανώς άλλους server). Υλοποιεί μερικώς την εκδοχή 0.9.1 του πρωτοκόλλου AMQP.

- *amqplib*: μία βιβλιοθήκη για τη δημιουργία client σε node.js με το πρωτόκολλο AMQP
- *express*: αναφέρθηκε παραπάνω
- *pg*: αναφέρθηκε παραπάνω
- pg-format: για την ασφαλή δημιουργία δυναμικών SQL querries. Τα αναγνωριστικά SQL και τα κυριολεκτικά διαγράφονται για να αποτραπούν τα SQL injections.

```
$ npm install amqp amqplib express pg pg-format –save
```

Παράρτημα Β: MQTT Framework

Ο πηγαίος κώδικας για το κομμάτι του MQTT:

broker.js:

```
//MQTT broker
var mosca = require('mosca');
var port = { port: 7574 }
var broker = new mosca.Server(port);

console.log(Date.now())

function get_date() {
    var date = new Date()
    return date
}

function random_number_from_2_to_7() {
    var num = 0;
    num = Math.floor(Math.random() * (7 - 2 + 1)) + 2;
    return num.toString();
}

// Postgresql connection
var Pool = require('pg').Pool;
const pool = new Pool({
    user: process.env.USER,
    host: process.env.HOST_ADDRESS,
    database: process.env.DATABASE,
    password: process.env.PASSWORD,
    port: process.env.DB_PORT
})

pool.connect(() => {
    console.log('Connected to Postgresql');
})
```

```
broker.on('ready', () => {
    console.log("Broker is online")
})

broker.on('published', (packet, client) => {
    let { payload } = packet

    if (payload) {
        let payloadString = payload.toString()
        try {
            let payloadJson = JSON.parse(payloadString)
            console.log(payloadJson)
            var sql = "INSERT INTO iot_values (date, point_id, value) VALUES
($1, $2, $3)";
            var data = [get_date(), random_number_from_2_to_7(),
payloadJson.data_value];
            pool.query(sql, data, (err, res) => {
                if (err) {
                    console.log('ERROR: ', err)
                } else {
                    console.log('Data inserted' + res)
                }
            })



        } catch (err) {
            console.log("invalid json" + err)
        }
    }
}
)
```

publisher.js

```
// MQTT publisher
var mqtt = require('mqtt')
var client = mqtt.connect('mqtt://localhost:7574')
var topic = 'iotpoints'
var num = 0;

client.on('connect', () => {
    setInterval(() => {
        client.publish(topic, JSON.stringify({
            "data_value": random_temperature_value(),
        }))
    }, 5000)
})
```

```
function random_temperature_value() {
    var temp = 0;
    temp = Math.floor(Math.random() * (100 - 0 + 1)) + 0;
    return temp.toString();
}
```

subscriber.js

```
// MQTT subscriber
var mqtt = require('mqtt')
var client = mqtt.connect('mqtt://localhost:7574')
var topic = 'iotpoints'

client.on('message', (topic, message) => {
    let payloadString = message.toString()
    let payloadJson = JSON.parse(payloadString)
    console.log(payloadJson)
})

client.on('connect', () => {
    client.subscribe(topic)
})
```

Παράρτημα Γ: HTTP Framework

app.js

```
console.log('Server side code running');

const express = require('express');
const app = express();
const port = 3000
const db = require('./queries.js')

app.use(express.static('public'));


app.post('/clicked', db.insertClickedData)
app.get('/data', db.getAllData)
app.post('/data', db.insertData)


app.listen(port, () => console.log(`App running on port ${port}`));

app.get('/', (req, res) => {
    res.sendFile(__dirname + '/index.html')
})
```

client.js

```
console.log('Client side code running');

const button = document.getElementById('myButton');
button.addEventListener('click', (e) => {

    console.log('Button clicked');
    fetch('/clicked', { method: 'POST' })
        .then(function (response) {
            if (response.ok) {
                console.log('click was recorded')
                return;
            }
            throw new Error('Request failed.');
        })
        .catch(function (error) {
            console.log(error);
        })
})
```

queries.js

```
const Pool = require('pg').Pool
const pool = new Pool({
    user: process.env.USER,
    host: process.env.HOST_ADDRESS,
    database: process.env.DATABASE,
    password: process.env.PASSWORD,
    port: process.env.DB_PORT
})

const insertData = (req, res) => {
    const { id, data_desc, data_type } = req.body;

    pool.query("INSERT INTO iot_values (date, point_id, value) VALUES ($1, $2,
$3)", [id, data_desc, data_type], (error, results) => {
        if (error) {
            throw error
        }
        res.status(201).send(`Data added with ID: ${id}`)
    })
}

const getAllData = (req, res) => {
    pool.query('SELECT * FROM iotpoints', (error, results) => {
        if (error) {
            throw error
        }
        res.status(200).json(results.rows)
```

```
    }
    )
}

const insertClickedData = (req, res) => {
    const click = {
        date: get_date(),
        point_id: random_number_from_2_to_7(),
        value: random_from_1_to_100()
    }

    function get_date() {
        var date = new Date()
        return date
    }

    function random_number_from_2_to_7() {
        return Math.floor(Math.random() * 6) + 2
    }

    function random_from_1_to_100() {
        var random_val = Math.floor(Math.random() * 100)
        return random_val
    }

    pool.query("INSERT INTO iot_values (date, point_id, value) VALUES ($1, $2,
$3)", [click.date, click.point_id, click.value], (error, results) => {
        if (error) {
            throw error
        }
        res.status(201).send(`Data added with ID: ${click.date}`)
    }
    )


}

module.exports = {
    insertData,
    getAllData,
    insertClickedData
}
```

index.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
```

```
        <title>NODE + EXPRESS + POSTGRESQL EXAMPLE</title>
    </head>
    <body>
        <h1>NODE + EXPRESS + POSTGRESQL EXAMPLE</h1>
        <p id="counter">Loading button click data</p>
        <button id="myButton">Click here</button>
    </body>
    <script src="client.js"></script>
</html>
```

Παράρτημα Δ: AMQP Framework

index.js

```
const Rabbit = require('./amqp');
const RabbitOpt = require('./amqpOpt');
const pg = require('./pg');
const express = require('express');

let connectionString = 'amqp://localhost'
const rabbit = new Rabbit()
const app = express()




function onConnected() {
    console.log("Connected to RabbitMQ")
    rabbit.receiveMessages(onMessageRecieved)
}

function onMessageRecieved(message) {
    console.log(`Recieved: ${message}`)
    var json = JSON.parse(message)
    var query = "INSERT INTO iot_values (date, point_id, value) VALUES ('" +
json.timestamp + "', " + json.point_id + ", "+ json.value +");"
    var data = [json.timestamp, json.point_id, json.value]
    pg.query(query).then(()=> console.log('Inserted'))
}

function getData() {
    return new Promise((resolve, reject) => {
        pg.query('SELECT * FROM iot_values').then((result) => {
            resolve(result.rows);
        })
    });
}
```

```
app.get('/', (req, res) => {
    getData().then(data => {
        res.send(JSON.stringify(data))
    })
})

app.listen(3001, () =>
    console.log('Example app listening on port 3000!'))

var rabbitConnection = RabbitOpt.connectRabbit(rabbit, connectionString)
var pgConnect = pg.connect( process.env.HOST_ADDRESS, process.env.DATABASE,
process.env.USER, process.env.PASSWORD)

Promise.all([rabbitConnection, pgConnect]).then(() => {
    onConnected()
})
```

amqp.js

```
var amqplib = require('amqplib');

class Rabbit {
    constructor() {
        this.connection = null;
        this.channel = null;
        this.queueName = 'message';
    }

    connect(hostName) {
        return new Promise((resolve, reject) => {
            amqplib.connect(hostName).then(conn => {
                this.connection = conn
                conn.createChannel().then(channel => {
                    this.channel = channel;

                    channel.assertQueue(this.queueName, {
                        durable: false,
                        autoDelete: true
                    }).then(() => {
                        resolve()
                    }).catch(err => reject('Error asserting queue'))
                }).catch(err => reject('Error asserting queue'))
            }).catch(err => reject('Error asserting queue'))
        })
    }

    disconnect() {
        return new Promise((resolve, reject) => {
```

```
            if (null == this.connection) {
                this.channel.deleteQueue(queueName);
                this.channel.close()
                this.connection.close()
                this.connection = null
                this.channel = null
                resolve();
            }
            reject('Not Connected')
        })
    }

    sendMessage(message) {
        return new Promise((resolve, reject) => {
            if (null != this.channel) {
                this.channel.sendToQueue(this.queueName, Buffer(message));
                resolve()
            } else {
                reject('Not Connected')
            }
        })
    }

    receiveMessages(callback) {
        this.channel.consume(this.queueName, message => {

            if (null != message) {
                this.channel.ack(message);
                callback(message.content.toString());

            }
        })
    }
}

module.exports = Rabbit;
```

amqpOpt.js

```
const utils = require('./utils');

connectionString = 'amqp://localhost'

function connect(rabbit, connectionString) {
    return new Promise((resolve, reject) => {
        console.log('Connecting to RabbitMQ');
        rabbit.connect(connectionString).then(() => {
            console.log('Successfully connected to RabbitMQ');
            resolve()
```

```
        }).catch(err => {
            console.log('Error connecting to RabbitMQ');
            utils.delay(3000, rabbit, connectionString).then(() => {
                connect(rabbit, connectionString).then(() => {
                    resolve();
                })
            })
        })
    })
}

module.exports.connectRabbit = connect
```

pg.js

```
const pg = require('pg');
const format = require('pg-format');

var MyClient;

module.exports.connect = (host, database, user, password) => {
    return new Promise((resolve, reject) => {
        const pool = new pg.Pool({
            host: host,
            database: database,
            user: user,
            password: password
        })
        connectToPostgres(pool).then(client => {
            console.log('Connected to Postgres')
            MyClient = client
            resolve()
        })
    });
}

function connectToPostgres(pool) {
    return new Promise((resolve, reject) => {

        const timerPromise = new Promise((resolve, reject) => {
            setTimeout(() => null, 5000)
        })

        const connectPromise = new Promise((resolve, reject) => {
            attemptConnection(pool).then(client => {
                resolve(client)
            }).catch(err => {
                console.log('Error connecting to Postgres')
                resolve(null)
```

```
            })
        })

        Promise.race([timerPromise, connectPromise]).then(client => {
            if (null != client) {
                resolve(client)
                return
            }

            console.log('Postgres connection failure, retrying...')

            Utils.delay(3000, pool).then((pool) => {
                connectToPostgres(pool).then(client => resolve(client)
                )
            })
        })
    })
}

function attemptConnection(pool) {
    return new Promise((resolve, reject) => {
        console.log('Attempting to connect to Postgres')
        pool.connect().then(client => {
            resolve(client)
        }).catch(err => {
            reject(err)
        })
    })
}

module.exports.query = (query => {
    return new Promise((resolve, reject) => {
        if (null == MyClient) {
            reject('Not connected to Postgres')
        }
        MyClient.query(query, (err, res) => {
            if (err) {
                reject(err)
            }
            resolve(res)
        })
    })
})
```

utils.js

```
module.exports.delay = (timeout, arg1, arg2) => {
    return new Promise(function(resolve){
```

```
        setTimeout(resolve.bind(null, arg1, arg2), timeout)
    });
}
```

server.js

```
var Rabbit = require('./amqp');
const rabbit = new Rabbit()

function get_date() {
    var date = new Date()
    return date
}

function random_number_from_2_to_7() {
    var num = 0;
    num = Math.floor(Math.random() * (7 - 2 + 1)) + 2;
    return num.toString();
}

function send_temperature() {
    var temperature = Math.random() * 100
    var message = {
        "value": temperature.toString(),
        "timestamp": get_date(),
        "point_id": random_number_from_2_to_7()
    }

    const messageString = JSON.stringify(message)
    console.log('sending temperature: ' + messageString)
    rabbit.sendMessage(messageString)
}


function connect(connectionString) {
    console.log('Connecting to RabbitMQ');
    rabbit.connect('amqp://localhost').then(() => {
        console.log('connected')
        setInterval(send_temperature, 3000)
    }).catch(err => {
        console.log(err)
        setTimeout(connect, 3000, connectionString)
    })
}

connect('amqp://localhost')
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1]. Internet of Things, Feng Xia, Laurence T. Yang, Lizhe Wang,,Alexey Vinel, 2012
[2]. Internet of Things: An Overview, Understanding the Issues and Challenges of a More Connected World, Karen Rose, Scott Eldridge, Layman Chapin, 2015
[3]. Wikipedia d2d
[4]. Device-to-Device Communication based IoT System: Benefits and Challenges, Abdirahman Ahmed Khalif, Dr. Muzahidul Islam, 2022
[5]. Cloud-Based IoT Applications and Their Roles in Smart Cities, Tanweer Alam, 2021
[6]. A survey on MQTT: A Protocol of Internet of Things, Dipa Soni, Ashwin Makwana.
[7]. MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. In Communication systems software and middleware and workshops, Hunkeler, U., Truong, H. L., & Stanford-Clark, A., (2008, January).
[8]. Quality of Service (QoS) 0,1, & 2 MQTT Essentials: Part 6, The HiveMQ Team, 2015
[9]. Semantic service provisioning for 6LoWPAN: powering internet of things applications on web, N.S. Han, Institut National des Tel´ ecommunications, 2015
[10]. Making the web faster with HTTP 2.0, I. Gregorik, 2013
[11]. Predicting short-transfer latency from TCP arcana: extended version, Martin Arlitt, Balachander Krishnamurthy, Jeffrey Mogul, 2005
[12]. What is CoAP protocol IoT | CoAP Architecture, message format
[13]. Internet of Things: Architectures, Protocols, and Applications, Pallavi Sethi, Smurti R. Sarangi.
[14]. The Constrained Application Protocol (CoAP), Z. Shelby, K. Hartke, C Bormann,
[15]. Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, UC Irvine, J. Gettys, Compaq/W3C, J. Mogul, Compaq, H. Frystyk, W3C/MIT, L. Masinter, Xerox, P. Leach, Microsoft, T. Berners-Lee, W3C/MIT, June 1999.
[16]. What Are IoT Devices: Definition, Types, and 5 Most Popular Ones for 2022, Nikita Duggal., 2022.
[17]. Internet of Things: A survey on the security of IoT frameworks, Mahmoud Ammar, Giovanni Russello, Bruno Crispo, 2017
[18]. https://en.wikipedia.org/wiki/Activity_tracker
[19]. Messaging technologies for the industrial internet and the internet of things whitepaper, A. Foster, PrismTech, 2015
[20]. Βλέπε [9].
[21]. https://www.cloudamqp.com/blog/what-is-amqp-and-why-is-it-used-in-rabbitmq.html , Lovisa Johansson.
[22]. IoT Frameworks and Complexity, Sunil Cheruvu, Anil Kumar, Ned Smith, David M. Wheeler, 2019
[23]. ARM. Arm mbed iot device platform. https://pelion.com/blog/product/iot-device-management-a-need-for-scaling-iot-eco-system/
[24]. ARM. Mbed device connector.
[25]. CoAP Protocol: Step-by-Step Guide, Francesco Azzola, 2018.
[26]. What is PostgreSQL?, https://www.postgresql.org/about/

[27]. What is Express JS? https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js
[28]. https://kaaproject.github.io/kaa/docs/v0.10.0/Welcome/
[29]. https://nodered.org/about/
[30]. What is Zetta? https://github.com/zettajs/zetta/wiki/Overview