



ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΤΑ

«ΕΞΥΠΝΕΣ ΤΕΧΝΟΛΟΓΙΕΣ ΜΕΤΡΗΣΗΣ ΓΙΑ ΕΞΟΙΚΟΝΟΜΙΣΗ ΕΝΕΡΓΕΙΑΣ»

(Smart Metering Technologies for Energy Savings)

«ΜΑΝΩΛΟΥΔΗΣ ΙΑΣΩΝ»

ΑΕΜ: 2539 Κατεύθυνση: Πληροφοριακά Συστήματα

Επίβλεπων Καθηγήτης:
ΔΗΜΗΤΡΗΣ ΒΡΑΚΑΣ, ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ

Βοηθός Επίβλεψης:
ΧΡΙΣΤΟΦΟΡΟΣ ΝΑΛΜΠΑΝΤΗΣ, ΥΠΟΨΗΦΙΟΣ ΔΙΔΑΚΤΟΡΑΣ

ΘΕΣΣΑΛΟΝΙΚΗ 2018

Περίληψη

Ο όρος “διατήρηση/εξοικονόμιση ενέργειας” αναφέρεται στην μείωση κατανάλωσης ενέργειας χρησιμοποιώντας λιγότερο συσκευές που κάνουν χρήση αυτής σε μεγάλη ποσότητα για την λειτουργία τους. Η σημασία της πράξης αυτής είναι ευρεία, τόσο για το συνολικό όφελος του πλανήτη όσο και για προσωπικό (οικονομικό) όφελος κάθε χρήστη. Ωστόσο η σημασία του όρου αυτού δεν σημαίνει απαραίτητα την στέρηση αγαθών και υπηρεσιών που οι συσκευές υψηλής κατανάλωσης προσφέρουν. Αντικείμενο της παρούσας εργασίας είναι το τεχνικό κομμάτι εγκατάστασης και επέκτασης ενός συστήματος παρακολούθησης ενέργειας, σε μικρότερη κλίμακα από ένα υπάρχον εμπορικό, για εκπαιδευτικούς λόγους. Παρουσιάζεται η υλοποίηση ενός κύκλου συσκευών Internet of Things, στα στάδια της συλλογής, αποθήκευσης και ανάκτησης δεδομένων ενώ τελικά πραγματοποιούνται πειράματα με χρήση του εγκατεστημένου συστήματος και κατασκευάζεται μια πειραματική εφαρμογή αναγνώρισης λειτουργίας Air-Condition, χρησιμοποιώντας τα παραχθέντα αποτελέσματα.

Abstract

Energy conservation refers to reducing energy consumption through using less of a high-consuming device in everyday life. This action is of great importance for planet Earth in general as well as for every individual human looking to save wealth. However, applying energy conservation in everyday life does not necessarily mean holding back in services provided by such machines. The subject of this thesis project is the technical part of installing and expanding an energy monitoring system of a smaller scale than an actual commercial one, for educational purpose. The “lifecycle” of an IoT device system is presented, regarding the stages of collecting, saving and accessing data. Finally, experiments were conducted using the installed monitoring system and a prototype AC-detection application was developed, using their results.

Ενχαριστίες

Πριν την παρουσίαση των αποτελεσμάτων της παρούσας εργασίας, αισθάνομαι την υποχρέωση να ευχαριστήσω ορισμένους από τους ανθρώπους που γνώρισα, συνεργάστηκα μαζί τους και έπαιξαν πολύ σημαντικό ρόλο στην πραγματοποίησή της.

Αρχικά, θα ήθελα να ευχαριστήσω τον κ. Δημήτρη Βράκα, επίκουρο καθηγητή του τμήματος πληροφορικής, για την ανάθεση της πτυχιακής εργασίας καθώς και για την παροχή των συσκευών που χρησιμοποιήθηκαν. Αισθάνομαι επίσης την ανάγκη να ευχαριστήσω θερμά τον κ. Χριστόφορο Ναλμπάντη, υποψήφιο διδάκτορα του τμήματος πληροφορικής, για την άριστη και στενή συνεργασία του κατά την διάρκεια εκπνόησης της εργασίας καθώς και για τις πολυάριθμες ώρες που αφιέρωσε στην καθοδήγηση και βοήθεια του συγγραφέα αυτής.

06/08/2018

Μανωλούδης Ιάσων

Περιεχόμενα

ΠΕΡΙΛΗΨΗ.....	VII
EXECUTIVE SUMMARY	IX
ΕΥΧΑΡΙΣΤΙΕΣ	XI
ΠΕΡΙΕΧΟΜΕΝΑ.....	XIII
ΛΙΣΤΑ ΣΧΗΜΑΤΩΝ	XV
ΛΙΣΤΑ ΠΙΝΑΚΩΝ	XVII
ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ	14
ΚΕΦΑΛΑΙΟ 2: ΜΕΤΡΗΣΕΙΣ ΚΑΙ ΚΑΤΑΝΑΛΩΣΗ ΕΝΕΡΓΕΙΑΣ.....	16
2.1 ΠΑΡΑΔΟΣΙΑΚΟΙ ΚΑΙ "ΕΞΥΠΝΟΙ" ΜΕΤΡΗΤΕΣ.....	16
2.1.1 ΟΙ ΜΕΤΡΗΤΕΣ ΕΩΣ ΣΗΜΕΡΑ	16
2.1.2 "ΕΞΥΠΝΟΙ" ΜΕΤΡΗΤΕΣ.....	20
2.2 SMART GRID & IOT.....	20
2.2.1 Ο ΡΟΛΟΣ ΤΟΥ ΙΟΤ ΣΕ ΔΙΚΤΥΑ ΣΠΙΤΙΩΝ ΚΑΙ ΓΕΝΙΚΟΤΕΡΑ ΚΤΗΡΙΩΝ.....	20
2.2.2 Ο ΡΟΛΟΣ ΤΟΥ ΙΟΤ ΣΤΙΣ ΕΠΙΧΕΙΡΗΣΕΙΣ ΠΑΡΟΧΗΣ ΕΝΕΡΓΕΙΑΣ.....	21
2.3 ΑΝΗΣΥΧΙΕΣ ΓΙΑ ΤΗΝ ΠΡΟΣΤΑΣΙΑ ΤΗΣ ΙΑΙΩΤΙΚΟΤΗΤΑΣ.....	23
2.4 ΓΕΝΙΚΕΣ ΠΑΡΑΤΗΡΗΣΕΙΣ.....	24
ΚΕΦΑΛΑΙΟ 3: ΣΥΛΛΟΓΗ, ΑΠΟΘΗΚΕΥΣΗ ΚΑΙ ΑΝΑΚΤΗΣΗ ΔΕΔΟΜΕΝΩΝ	24
3.1 ΣΥΛΛΟΓΗ ΔΕΔΟΜΕΝΩΝ.....	24
3.1.1 ΣΥΛΛΟΓΗ ΔΕΔΟΜΕΝΩΝ ΚΑΙ SMART HOMES.....	24
3.2 ΑΠΟΘΗΚΕΥΣΗ ΔΕΔΟΜΕΝΩΝ.....	24
3.2.1 ΑΠΟΘΗΚΕΥΣΗ ΔΕΔΟΜΕΝΩΝ ΚΑΙ SMART HOMES.....	24
3.3 ΑΝΑΚΤΗΣΗ ΔΕΔΟΜΕΝΩΝ (WEB-BASED APIs).....	26
3.3.1 ΑΥΞΗΣΗ ΤΟΥ ΟΓΚΟΥ ΔΕΔΟΜΕΝΩΝ.....	27
3.3.2 ΓΡΗΓΟΡΗ ΠΡΟΣΒΑΣΗ ΣΤΑ ΔΕΔΟΜΕΝΑ.....	27
3.3.3 ΑΝΑΠΤΥΞΗ ΝΕΩΝ ΠΡΟΪΟΝΤΩΝ ΚΑΙ ΥΠΗΡΕΣΙΩΝ.....	27
3.4 Η ΠΡΟΣΕΓΓΙΣΗ ΑΥΤΗΣ ΤΗΣ ΕΡΓΑΣΙΑΣ.....	28
ΚΕΦΑΛΑΙΟ 4: ΠΕΡΙΓΡΑΦΗ ΤΩΝ ΕΡΓΑΛΕΙΩΝ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΑΝ.....	30
4.1 Ο ΟΡΓΑΝΙΣΜΟΣ OPENENERGYMONITOR.....	30

4.1.1 Η ΣΥΣΚΕΥΗ EMONPI.....	30
4.1.2 Η ΣΥΣΚΕΥΗ EMONTX.....	32
4.2 ΤΟ ΛΟΓΙΣΜΙΚΟ LAMP.....	34
4.3 ΤΟ ΛΟΓΙΣΜΙΚΟ PHPMYADMIN.....	35
4.4 SUBLIME TEXT.....	36
4.5 ΤΟ ΕΡΓΑΛΕΙΟ NODE-RED.....	37
4.5.1 FLOW-BASED PROGRAMMING.....	37
ΚΕΦΑΛΑΙΟ 5: ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΩΝ ΣΥΣΤΗΜΑΤΩΝ.....	40
5.1 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΕΓΚΑΤΕΣΤΗΜΕΝΟΥ ΣΥΣΤΗΜΑΤΟΣ ΣΥΣΚΕΥΩΝ ΤΗΣ ΣΕΙΡΑΣ EMON.....	40
5.1.1 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ EMONPI.....	41
5.1.2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ MQTT PROTOCOL.....	41
5.2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΗΣ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ.....	42
5.3 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΩΝ ΕΦΑΡΜΟΓΩΝ NODE-RED.....	43
5.3.1 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΕΦΑΡΜΟΓΗΣ REAL-TIMEDB_INPUT.....	44
5.3.2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΕΦΑΡΜΟΓΗΣ REMOTEDB_INPUT.....	45
5.3.3 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΕΦΑΡΜΟΓΗΣ DETECT_AC.....	45
ΚΕΦΑΛΑΙΟ 6: ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΥΛΟΠΟΙΗΣΗΣ.....	48
6.1 ΕΓΚΑΤΑΣΤΑΣΗ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ ENERGY MONITORING (EMONPI, EMONTX & SENSORS).....	48
6.2 LOCAL & REMOTE LOGGING ΜΕΤΡΗΣΕΩΝ.....	50
6.2.1 LOCAL DATA LOGGING.....	50
6.2.2 REMOTE DATA LOGGING.....	51
6.3 ΔΗΜΙΟΥΡΓΙΑ ΤΗΣ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ.....	54
6.3.1 LIVINGROOMTEMP TABLE.....	55
6.3.2 VRMS TABLE.....	56
6.3.3 POWER TABLE.....	57
6.4 ΔΗΜΙΟΥΡΓΙΑ ΕΦΑΡΜΟΓΗΣ NODE-RED “REMOTEDB_INPUT”.....	57
6.5 ΔΗΜΙΟΥΡΓΙΑ ΕΦΑΡΜΟΓΗΣ NODE-RED “REAL-TIMEDB_INPUT”.....	60
6.6 ΚΑΤΑΣΚΕΥΗ API.....	62
6.6.1 ΥΛΟΠΟΙΗΣΗ PAGENATION.....	67
6.7 ΕΚΤΕΛΕΣΗ ΠΕΙΡΑΜΑΤΩΝ.....	68
6.8 ΥΛΟΠΟΙΗΣΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ NODE-RED “DETECT_AC”.....	72
ΚΕΦΑΛΑΙΟ 7: ΣΥΜΠΕΡΑΣΜΑΤΑ.....	75
ΠΑΡΑΡΤΗΜΑ I: ΑΝΑΦΟΡΕΣ	78
ΠΑΡΑΡΤΗΜΑ II: ΑΚΡΩΝΥΜΑ	81
ΠΑΡΑΡΤΗΜΑ III: ΓΛΩΣΣΑΡΙΟ	83
ΠΑΡΑΡΤΗΜΑ IV: ΚΩΔΙΚΑΣ.....	85

Λίστα Σχημάτων

ΕΙΚΟΝΑ 1: ΠΑΡΑΔΟΣΙΑΚΟΣ ΜΕΤΡΗΤΗΣ	17
ΕΙΚΟΝΑ 2: SMART METER.....	18
ΕΙΚΟΝΑ 3: ΕΓΚΑΤΑΣΤΑΣΗ ΕΞΥΠΙΝΟΥ ΜΕΤΡΗΤΗ ΣΕ ΣΠΙΤΙ.....	19
ΕΙΚΟΝΑ 4: ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΥΣΤΗΜΑΤΩΝ INTERNET OF THINGS.....	21
ΕΙΚΟΝΑ 5: ΣΥΛΛΟΓΗ, ΑΠΟΘΗΚΕΥΣΗ, ΑΝΑΚΤΗΣΗ ΔΕΔΟΜΕΝΩΝ	24
ΕΙΚΟΝΑ 6: ΕΠΙΠΤΩΣΕΙΣ ΧΡΗΣΗΣ API	27
ΕΙΚΟΝΑ 7: Η ΣΥΣΚΕΥΗ EMONPI	31
ΕΙΚΟΝΑ 8: ΓΡΑΦΙΚΗ ΑΝΑΠΑΡΑΣΤΑΣΗ ΤΙΜΩΝ ΣΤΟ EMONCMS.....	32
ΕΙΚΟΝΑ 9:EMONTX ΜΕ 4 ΑΙΣΘΗΤΗΡΕΣ CT KAI ENAN AC ADAPTER	33
ΕΙΚΟΝΑ 10: CT, TEMPERATURE, AC SENSORS	33
ΕΙΚΟΝΑ 11: SOFTWARE BUNDLE LAMP	35
ΕΙΚΟΝΑ 12: PHPMYADMIN	36
ΕΙΚΟΝΑ 13: SUBLIME TEXT	37
ΕΙΚΟΝΑ 14: ΕΦΑΡΜΟΓΗ ΣΕ NODE-RED	38
ΕΙΚΟΝΑ 15: ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΥΣΤΗΜΑΤΟΣ ΣΥΣΚΕΥΩΝ EMONPI & EMONTX	40
ΕΙΚΟΝΑ 16: ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ EMONPI.....	41
ΕΙΚΟΝΑ 17: TO MONTELO PUBLISH/SUBSCRIBE	42
ΕΙΚΟΝΑ 18: DATABASE SCHEMA	43
ΕΙΚΟΝΑ 19: FLOW REALTIMEDB_INPUT.....	44
ΕΙΚΟΝΑ 20: FLOW REMOTEDB_INPUT	45
ΕΙΚΟΝΑ 21: FLOW DETECT_AC	46
ΕΙΚΟΝΑ 22: ΕΓΚΑΤΑΣΤΑΣΗ ΕΜΟΝΤΧ ΚΑΙ ΑΙΣΘΗΤΗΡΩΝ	48
ΕΙΚΟΝΑ 23: ΕΓΚΑΤΑΣΤΑΣΗ ΕΜΟΝΠΙ ΩΣ GATEWAY	49
ΕΙΚΟΝΑ 24: SSH ΣΥΝΔΕΣΗ ΣΤΟ EMONPI.....	50
ΕΙΚΟΝΑ 25: LOCAL EMONCMS SERVER.....	51
ΕΙΚΟΝΑ 26: EMONCMS API KEYS.....	52
ΕΙΚΟΝΑ 27: ΧΡΗΣΗ R/W API KEY ΣΤΟ LOCAL EMONCMS	52
ΕΙΚΟΝΑ 28: Η ΔΙΑΔΙΚΑΣΙΑ LOG TO FEED	53
ΕΙΚΟΝΑ 29: ΓΡΑΦΙΚΗ ΠΑΡΑΣΤΑΣΗ ΤΙΜΩΝ VRMS	54
ΕΙΚΟΝΑ 30: ΑΠΕΛΕΥΘΕΡΩΣΗ PORT 3306	55
ΕΙΚΟΝΑ 31: ΠΡΩΤΟ ΕΠΙΠΕΔΟ CONFIGURATION EMONCMS IN NODE.....	58
ΕΙΚΟΝΑ 32: ΔΕΥΤΕΡΟ ΕΠΙΠΕΔΟ CONFIGURATION EMONCSM IN NODE.....	58
ΕΙΚΟΝΑ 33: MYSQL NODE CONFIGURATION.....	60
ΕΙΚΟΝΑ 34: MQTT IN NODE CONFIGURATION.....	61
ΕΙΚΟΝΑ 35: LOCALHOST:5000/ALL/LIVINGROOMTEMP URL.....	65
ΕΙΚΟΝΑ 36: LOCALHOST:5000/QUERY/<TABLE>?D1=VAL1&D2=VAL2 URL	68
ΕΙΚΟΝΑ 37: LOCALHOST:5000/AVG/<TABLE>/?D1=VAL1&D2=VAL2 URL	67
ΕΙΚΟΝΑ 38: LOCALHOST:5000/PAGINATE/<TABLE> URL	68
ΕΙΚΟΝΑ 39: ΜΕΤΡΗΣΕΙΣ VRMS ΧΩΡΙΣ AC	69
ΕΙΚΟΝΑ 40: ΜΕΤΡΗΣΕΙΣ ΘΕΡΜΟΚΡΑΣΙΑΣ ΧΩΡΙΣ AC	70
ΕΙΚΟΝΑ 41: ΘΕΡΜΟΚΡΑΣΙΑ ΜΕ AC	71
ΕΙΚΟΝΑ 42: VRMS ME AC	71
ΕΙΚΟΝΑ 43: ΑΥΤΟΜΑΤΟ TWEET ΤΗΣ ΕΦΑΡΜΟΓΗΣ DETECT_AC.....	73

Λίστα Πινάκων

ΠΙΝΑΚΑΣ 1: ΣΥΓΚΡΙΣΗ ΣΥΣΚΕΥΩΝ ΕΜΟΝΠΙ-ΕΜΟΝΤΧ	33
--	----

ΚΕΦΑΛΑΙΟ 1: Εισαγωγή

ΕΙΣΑΓΩΓΗ

Αντικείμενο της παρούσας εργασίας είναι το τεχνικό κομμάτι υλοποίησης συστήματος έξυπνων τεχνολογιών μέτρησης για την εξοικονόμιση ενέργειας.

Η εργασία δομείται σε κεφάλαια ως εξής:

- Στο Κεφάλαιο 2 γίνεται μια σύντομη περιγραφή στις μετρήσεις και κατανάλωση ενέργειας γενικότερα. Αναφέρονται οι τρόποι μέτρησης κατανάλωσης ενέργειας από το παρελθόν μέχρι τα σύγχρονα smard grid και ολοκληρώνεται με την παράθεση θεμάτων που αφορούν την προστασία της ιδιοτικότητας.
- Στο Κεφάλαιο 3 παρουσιάζονται σύντομα τα ειδικότερα θέματα που πραγματεύεται η πτυχιακή. Γίνεται αναφορά στη σημασία και τους τρόπους της συλλογής, αποθήκευσης και ανάκτησης δεδομένων ενώ παρουσιάζεται ο τρόπος εφαρμογής κάθε επιμέρους θέματος στα έξυπνα σπίτια.
- Στο κεφάλαιο 4 παρουσιάζονται τα διάφορα εργαλεία που χρησιμοποιήθηκαν για την εκπνόηση της εργασίας.
- Στο κεφάλαιο 5 δίνονται οι αρχιτεκνικές των συστήματων που χρησιμοποιήθηκαν καθώς και της βάσης δεδομένων αλλά και των εφαρμογών που υλοποιήθηκαν στα πλαίσια της εργασίας.
- Στο κεφάλαιο 6 καταγράφεται λεπτομερώς η συνολική υλοποίηση της εργασίας, παραθέτοντας εικόνες και επεξηγώντας όπου χρειάζεται.
- Το κεφάλαιο 7 αποτελεί τον επίλογο της εργασίας. Παραθέτεται η συνολική εικόνα, κριτική στα αποτελέσματα αυτής και ιδέες που μπορούν να τεθούν ως μελλοντικοί στόχοι υλοποίησης.
- Στο Παράρτημα I παρουσιάζονται αλφαριθμητικά η βιβλιογραφία και οι δικτυακοί τόποι που αναφέρονται στην εργασία.
- Στο Παράρτημα II παρουσιάζονται τα ακρωνύμια τα οποία χρησιμοποιούνται σε αυτή την εργασία για την διευκόλυνση του αναγνώστη.
- Στο Παράρτημα III παρουσιάζεται το γλωσσάριο ξενικών όρων οι οποίοι χρησιμοποιούνται σε αυτή την εργασία για την διευκόλυνση του αναγνώστη.
- Στο Παράρτημα IV παρουσιάζεται το ευρετήριο των όρων οι οποίοι χρησιμοποιούνται σε αυτή την εργασία για την διευκόλυνση του αναγνώστη.
- Στο Παράρτημα V παρουσιάζεται ο κώδικας και το documentation του API.

ΚΕΦΑΛΑΙΟ 2: Μετρήσεις και Κατανάλωση Ενέργειας

ΜΕΤΡΗΣΕΙΣ ΚΑΙ ΚΑΤΑΝΑΛΩΣΗ ΕΝΕΡΓΕΙΑΣ

Η ανάγκη καταμέτρησης της κατανάλωσης ηλεκτρικής ενέργειας γίνεται επιτακτική από τα μέσα του 19ου αιώνα, με την εφεύρεση του δυναμό από τους Anyos Jedlik το 1861, Werner von Siemens το 1867 [1]. Η πρώτη αξιοσημείωτη εφαρμογή του ηλεκτρισμού είναι ο φωτισμός. Με την εμφάνιση αυτού του νέου προϊόντος στον κόσμο, του ηλεκτρισμού, ήταν προφανές πως το κόστος του έπρεπε να καθοριστεί. Ο Oliver B. Shallenberger παρουσιάζει το AC αμπεριόμετρο το έτος 1888. Σταδιακά, η προοδευτική ανάπτυξη της τεχνολογίας μετρήσεων οδήγησε στη διαφώτιση των ζωών πολλών ανθρώπων [2]. Την πιο διαδεδομένη μονάδα μέτρησης αυτού μεχρι και σήμερα αποτελεί η Κιλοβατάρα (kWh) που αντιστοιχεί στην ενέργεια που παράγεται ή καταναλώνεται μέσα σε μία ώρα υπό σταθερή ισχύ ενός κιλοβάτ.

2.1 Παραδοσιακοί και “έξυπνοι” μετρητές

2.1.1 Οι μετρητές έως σήμερα

Οι ηλεκτρικές συσκευές που ανιχνεύουν και εκθέτουν την ενέργεια σε μορφή μετρήσεων κατανοητών από τον άνθρωπο ονομάζονται μετρητές ηλεκτρισμού (electricity meters). Παραδοσιακοί μετρητές χρησιμοποιούνται από τα τέλη του 19ου αιώνα, όπως προαναφέρθηκε. Ανταλλάσουν δεδομένα μεταξύ συσκευών σε ένα ηλεκτρονικό περιβάλλον που αφορούν τόσο την παραγωγή ενέργειας όσο και την διανομή αυτής. Στους περισσότερους από τους παραδοσιακούς μετρητές χρησιμοποιούνται δίσκοι αλουμινίου για την εύρεση χρήσης της ενέργειας [1]. Οι σημερινοί μετρητές είναι μεν ψηφιακοί ωστόσο παρουσιάζουν ακόμα σημαντικούς περιορισμούς. Ένας απλός μετρητής παρουσιάζεται παρακάτω στην εικόνα 2.1.



Εικόνα 1: Παραδοσιακός Μετρητής

Μερικοί από τους περιορισμούς των παραδοσιακών μετρητών [1] είναι:

- Αναξιοπιστία εκ φύσεως καθώς ο καταναλωτής είναι αναγκασμένος να περιμένει τον μηνιαίο λογιαργιασμό ηλεκτρικού ρεύματος προκειμένου να γνωρίζει την κατανάλωσή του
- Προκειμένου να διαβαστούν οι μετρήσεις, ένας μεγάλος αριθμός επιθεωρητών πρέπει να προσληφθεί
- Νέα είδη ωριαίας τιμολόγισης δεν είναι δυνατόν να εγκατασταθούν με τους υπάρχοντες μετρητές

2.1.2 “Έξυπνοι” μετρητές

2.1.2.1 “Έξυπνα” δίκτυα

Ένα έξυπνο δίκτυο αποτελεί την μοντέρνα εκδοχή του συμβατικού ηλεκτρικού δικτύου. Τα συμβατικά ηλεκτρικά δίκτυα καθίστανται αδύνατα σε σχέση με την ηλεκτρική διακύμανση φορτίου των συσκευών στο σπίτι. Η ραγδαία αύξηση του πληθυσμού που σημειώνεται τα τελευταία χρόνια αποτελεί επίσης ένδειξη πως τα ηλεκτρικά δίκτυα γίνονται όλο και πιο εύθραυντα. Όσο υψηλότερος ο πλυθησμός, τόσο μεγαλύτερο και το φορτίο στο δίκτυο. Ως έξυπνο δίκτυο λοιπόν ορίζουμε το δίκτυο αυτό του οποίου η αποδοτικότητα είναι αισθητά αυξημένη σε σχέση με τα συμβατικά χρησιμοποιώντας εξ αποστάσεως έλεγχο αυτού, η μέτρηση των καταναλώσεων σε μια επικοινωνία που υποστηρίζεται από την παροχή δεδομένων σε πραγματικό χρόνο (real time) στους καταναλωτές, τον προμηθευτή και αντιστρόφως. Στα λεγόμενα “έξυπνα” δίκτυα

χρησιμοποιούνται αυτοματοποιημένοι αισθητήρες. Οι αισθητήρες αυτοί είναι υπεύθυνοι για την αποστολή των μετρήσεων δεδομένων στις επιχειρήσεις κοινής οφέλειας (πάροχοι ενέργειας). Επιπλέον έχουν την δυνατότητα να προσδιορίζουν τυχών βλάβες, υπερφορτώσεις κ.α. ώστε να αποφεύγονται εν δυνάμει υπερθερμάνσεις των γραμμών του δικτύου. Με άλλα λόγια, κάνουν χρήση του χαρακτηριστικού της αυτοθεραπείας.

Η έννοια του έξυπνου μετρητή λοιπόν προέρχεται κυριολεκτικά από την ιδέα του έξυπνου δικτύου. Με τις εγκαταστάσεις των δικτύων αυτών έναντι των συμβατικών αναμένεται μείωση των εκπομπών του άνθρακα της τάξης του 5% μέχρι το έτος 2030, κάτι που μπορεί να έχει μεγάλη (θετική) επίπτωση στις κλιματικές αλλαγές [1]. Για τη βιώσιμη ανάπτυξη και τη δημιουργία νέων υποδομών δικτύων, τα Smart Grids συνιστώνται σε πολλές χώρες.

2.1.2.2 “Έξυπνοι” μετρητές

Οι έξυπνοι μετρητές είναι φιλικοί προς το περιβάλλον μετρητές ενέργειας που χρησιμοποιούνται για την μέτρηση της ηλεκτρικής ενέργειας σε kWh. Αποτελούν συσκευές που προσφέρουν άμεσο όφελος στους καταναλωτές που αναζητούν την μείωση του λογαριασμού ρεύματος. Αποτελούν ένα τμήμα των Υποδομών Προηγμένων Μετρητών (Advanced Meter Infrastructure, AMI) και είναι υπεύθυνοι για την αυτόματη αποστολή μετρήσεων στον προμηθευτή ενέργειας, όπως ορίζεται από την αρχιτεκτονική του AMI. Μία εικόνα ενός σύγχρονου έξυπνου μετρητή εμφανίζεται παρακάτω, στην εικόνα 2.2.



Eικόνα 2: Smart Meter

Η καταγραφή της κατανάλωσης ενέργειας γίνεται ανά ωριαία διαστήματα ή και διαστήματα μικρότερης χρονικής διάρκειας (έναντι των μηνιαίων μετρήσεων των συμβατικών μετρητών). Ένας Έξυπνος Μετρητής διαθέτει μη πτητική αποθήκευση δεδομένων, δυνατότητα απομακρυσμένης σύνδεσης ή αποσύνδεσης αυτού, ανίχνευση παραβίασης και διευκολύνσεις αμφίδρομης επικοινωνίας. Εκτελεί απομακρυσμένη αναφορά (remote reporting) των δεδομένων που συλλέγονται στον κεντρικό μετρητή. Αυτός με την σειρά του παρακολουθεί την λειτουργία του Smart Meter. Από λειτουργική άποψη, η χρήση της έξυπνης μετρητής επιτρέπει τη βελτιωμένη διαχείριση και έλεγχο του ηλεκτρικού δικτύου [2].

Μερικά από τα οφέλη των Smart Meters είναι:

- Χαμηλό λειτουργικό κόστος
- Τόσο οι καταναλωτές όσο και οι επιχειρήσεις κοινής οφέλειας εξοικονομούν χρόνο με την εύκολη αναφορά των μετρήσεων του μετρητή στους παρόχους ενέργειας
- Είναι επιτρεπτή η online πληρωμή λογαριασμού ηλεκτρικής ενέργειας
- Η κατανάλωση ρεύματος μπορεί να μειωθεί σημαντικά κατά τη διάρκεια high-traffic ωρών σύμφωνα με κάποια πολιτική
- Αρκετοί υποστηρίζουν την αυτόματη απενεργοποίηση συσκευών, όταν αυτές ανιχνεύονται αδρανείς για κάποιο χρονικό διάστημα [1].

Οι έξυπνοι μετρητές αντιλαμβάνονται το σύνολο της κατανάλωσης ενέργειας που επιτελούν οι κάτοικοι. Οι μετρήσεις αυτές δίνονται, μεταξύ άλλων, μία ευρύτερη κατανόηση στις επιχειρήσεις ενέργειας έτσι ώστε να μπορούν να τροποποιηθούν ανάλογα τα ενεργειακά τέλη των κατοίκων. Παρακατώ φαίνεται η εγκατάσταση έξυπνου μετρητή σε ένα συγχρονό σπίτι.



Εικόνα 3: Εγκατάσταση έξυπνου μετρητή σε σπίτι

Η παραπάνω εικόνα δείχνει τις καθημερινές δραστηριότητες των οικιακών συσκευών ενός σύγχρονου σπιτιού, που μετρήθηκαν από ένα Smart Meter. Ο έξυπνος μετρητής τοποθετείται έξω από το σπίτι και μετρώνται τα ωριαία δεδομένα κατανάλωσης ενέργειας. Μία τέτοια εγκατάσταση από μόνη της μπορεί να θεωρηθεί αρκετή για να μετατρέψει ένα απλό σπίτι σε ένα “έξυπνο σπίτι” (smart house).

Ωστόσο, με το μέγεθος της επίβλεψης (monitoring) που υφίσταται ένα σπίτι από την εγκατάσταση ενός smart meter είναι μόνο φυσικό να προκύψουν ανησυχίες και ζητήματα για την προστασία της ιδιοτικότητας των ατόμων.

2.2 Smart Grid & IoT

Οι επιχειρήσεις κοινής οφέλιας στις ανεπτυγμένες οικονομίες προσπαθούν να αξιοποιήσουν τις δυνατότητες του Internet of Things (IoT) για την βελτίωση της ανάπτυξης και της λειτουργίας των Smart Grids. Ωστόσο μία υποδομή Internet of Things δεν μπορεί να υπάρξει μόνη της. Για την υποστήριξή της είναι απαραίτητη η ύπαρξη πολυάριθμων υπηρεσιών πληροφορικής, κάτι που μεταφράζεται σε ευκαιρία για παρόχους υπηρεσιών πληροφορικής και σε οφέλη για παρόχους ενέργειας και τους καταναλωτές τους.

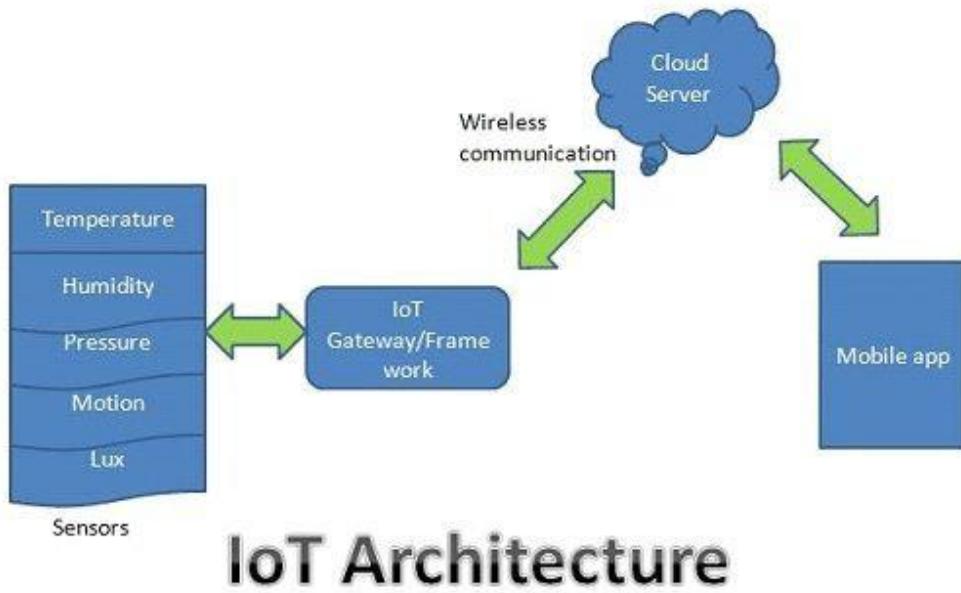
Από το 1999, όταν η ιδέα προτοπαρουσιάστηκε από το MIT [3], το Internet of Things έχει υιοθετηθεί από κυβερνήσεις σε Αμερική, Ιαπωνία, Κίνα, Κορέα και την Ευρωπαϊκή Ένωση (ΕΕ) για διάφορους σκοπούς. Ως παράδειγμα, όλοι οι πάροχοι ενέργειας της ΕΕ πρέπει υποχρεωτικά να χρησιμοποιούν Smart Meters. Η Αμερική έχει επίσης ξεκινήσει την μετάβαση σε τεχνολογίες Smart Grid ήδη από το 2007 [4].

Πρωτού συνεχίσουμε, είναι σημαντικό να ορίσουμε την σημασία της λεγόμενης “πύλης” Internet of Things (IoT Gateway). Ένα IoT Gateway μπορεί να είναι μία συσκευή ή ένα λογισμικό που χρησιμοποιείται για την σύνδεση των αισθητήρων και συσκευών του δικτύου Internet of Things με το cloud.

2.2.1 Ο ρόλος των IoT σε δίκτυα σπιτιών και γενικότερα κτηρίων

Τα IoT Gateways δίνουν πρόσβαση σε ένα ευρύ φάσμα συνδειμότητας σε πρωτόκολλα HAN ή BAN (Home Area Network ή Building Area Network) όπως το ZigBee, το Bluetooth, το Wi-Fi και το LAN. Συσκευές ή αισθητήρες μπορούν να συνδεθούν με την πύλη η οποία με τη σειρά της συνδέεται με το cloud. Αυτό επιτρέπει στον χρήστη να έχει πρόσβαση στα δεδομένα αισθητήρων από απόσταση μέσω των κινητών συσκευών του (smartphone, tablet, laptop κ.α.) από οποιαδήποτε θέση και ανά πάσα στιγμή.

Η δυνατότητα αυτή προέρχεται από την φύση της αρχιτεκτονικής ενός συστήματος Internet of Things, η οποία φαίνεται στην εικόνα 2.2.1. Η χρήση συστήματος IoT επιπλέον προσφέρει σημαντική βοήθεια στην ενεργειακή ανάλυση κάθε συσκευής συνδεδεμένης σε αυτό, διευκολύνοντας έτσι τον χρήστη να διαχειριστεί αποδοτικότερα τους χρόνους λειτουργίας/αδράνειας των συσκευών. Οι χρήστες μπορούν να έχουν πρόσβαση σε ιστορικά δεδομένα από το cloud (μετρήσεις παλαιότερων ημερών, μηνών, ετών) παίρνοντας έτσι σημαντικές πληροφορίες όπως ποια συσκευή κατανάλωσε περισσότερη ενέργεια και σε ποια περίοδο της ημέρας. Βάση αυτών βελτιστοποιείται η ενεργειακή τους κατανάλωση.



Εικόνα 4: Αρχιτεκτονική Συστήματος Internet of Things

2.2.2 Ο ρόλος των IoT στις επιχειρήσεις παροχής ενέργειας

Τα Internet of Things Gateways παρέχουν στις επιχειρήσεις κοινής ωφέλειας μια ευρύτερη εικόνα των προτύπων τους διανομής ενέργειας, επιτρέποντας υψηλή συνδεσιμότητα και ανάλυση των πόρων τους σε πραγματικό χρόνο.

Επιπλέον, επιτρέπει την ανάπτυξη μηχανισμού ζήτησης-απόκρισης (Demand-Response mechanism) για τους παρόχους υπηρεσιών κοινής ωφέλειας για τη βελτιστοποίηση της κατανομής ενέργειας με βάση τα καταναλωτικά πρότυπα.

Με την χρήση IoT Gateway γίνεται εύκολη η συλλογή δεδομένων από όλα τα συστήματα AMI (Advanced Meter Infrastructure) που είναι συνδεδεμένα με τον συγκεκριμένο πάροχο ενέργειας και παρέχονται αναλυτικά αποτελέσματα σε περιόδους χαμηλής και υψηλής κατανάλωσης ενέργειας. Κατά συνέπεια, οι πάροχοι ενέργειας μπορούν να χρησιμοποιήσουν τις πληροφορίες αυτές για να προβλέψουν τις ώρες αιχμής αλλά και να προσφέρουν δυναμικές επιλογές τιμολόγησης.

Τέλος, όπως προαναφέρθηκε, ένα σύστημα Internet of Things δίνει τη δυνατότητα προληπτικής συντήρησης του εαυτού του. Ειδοποιεί στις εταιρείες κοινής ωφέλειας σχετικά με τα ελαττώματα που εντοπίζονται σε αυτό, τα οποία θα χρειάζονταν μια άμεση αντιμετώπιση. Το IoT Gateway επιτρέπει τη διαλειτουργικότητα και παρέχει ένα ευρύ φάσμα πρωτοκόλλων που διασφαλίζουν τη συνδεσιμότητα με τα περισσότερα από τα στοιχεία του δικτύου.

2.3 Ανησυχίες για την προστασία της ιδιωτικότητας

Η έρευνα που γίνεται στον χώρο των έξυπνων μετρητών και της μείωσης κατανάλωσης ενέργειας έχει ως ιδανικό στόχο την εγκατάσταση έξυπνων δικτύων σε κάθε χώρο που αυτό είναι δυνατόν, σε βάρος των συμβατικών ηλεκτρικών δικτύων. Όπως αναφέρθηκε πιο πριν, με την επίτευξη αυτή είναι δυνατόν για τις εταιρείες παροχής ενέργειας να γνωρίζουν ανά πάσα στιγμή την αποσυντιθέμενη κατανάλωση συνολικής ενέργειας ενός σπιτιού ή κτηρίου, δηλαδή την ακριβή κατανάλωση ενέργειας κάθε συγκευής την συγκεκριμένη χρονική στιγμή. Η επιδίωξη αυτή προκαλεί ανησυχίες σχετικά με την διαφύλαξη των προσωπικών δεδομένων των χρηστών.

Σε περίπτωση που η παραπάνω ιδανική κατάσταση μπορέσει να υπάρξει, από τα δεδομένα λειτουργίας των συσκευών ενός χώρου μπορούν κατ' επέκταση να παραχθούν συμπεράσματα για τον τρόπο ζωής και την συμπεριφορά των ανθρώπων του παρατηρούμενου κτηρίου.

Ακόμα, τίθεται το θέμα της μεταφοράς των δεδομένων μεταξύ του μετρητή κατανάλωσης, του υπολογιστικού συστήματος που θα τρέχει τον αλγόριθμο αποσύνθεσης ενέργειας και των τερματικών που θα λαμβάνουν τα αποτελέσματα. Κρίνεται επιτακτική η ανάγκη διασφάλισης της κρυπτογράφησης και της διατήρησης της ακεραιότητας των δεδομένων.

Τέλος, υπάρχουν ενδείξεις πως, χρησιμοποιώντας ως δεδομένα την κατανάλωση ενέργειας μιας LCD οθόνης είναι δυνατό να αναδημιουργηθούν οι εικόνες που προβάλλονταν κατά τη διάρκεια της μέτρησης. Η έρευνα [1] μέχρι τώρα δεν έχει δείξει ότι μπορεί να αναγνωρίσει μαγνητοσκοπημένες εικόνες, μπορεί όμως να εκτιμήσει με καλή ακρίβεια ποιο ζωντανό τηλεοπτικό πρόγραμμα προβάλλεται τη συγκεκριμένη χρονική στιγμή.

2.4 Γενικές παρατηρήσεις

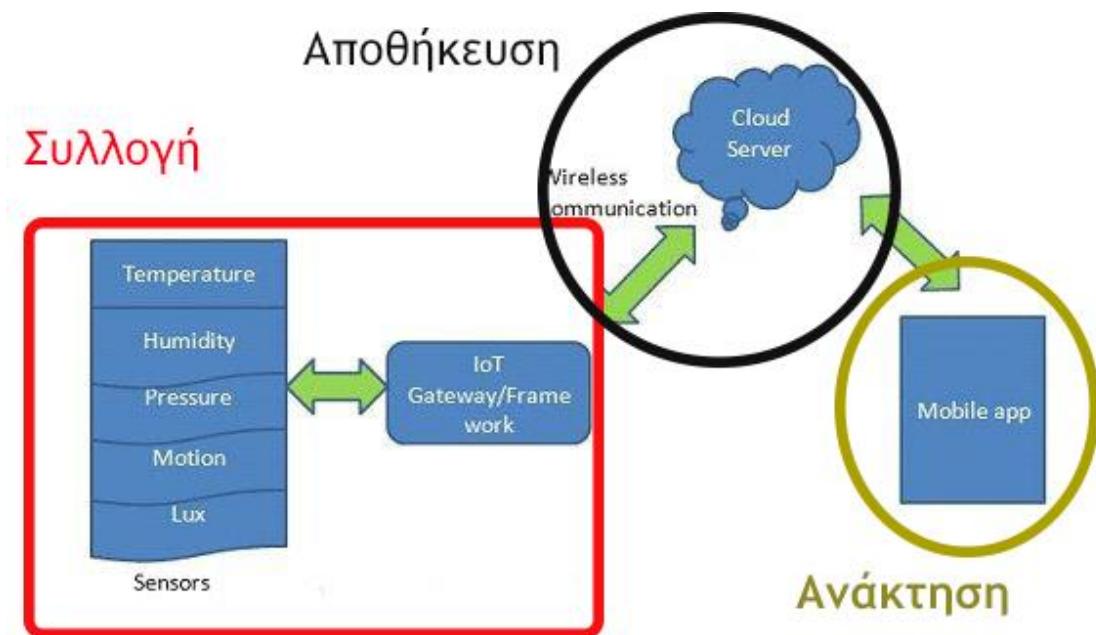
Παρατηρείται πως λόγω της μείζονος σημασίας που έχει λάβει το ενεργειακό πρόβλημα κατά την τελευταία δεκαετία, οι έξυπνοι μετρητές και ο τρόπος λειτουργίας τους αποκτά ενδιαφέρον τόσο για βιομηχανική όσο και για προσωπική χρήση. Η δυνατότητα της συλλογής χρήσιμης και διορατικής πληροφορίας ανοίγει το δρόμο για περαιτέρω έρευνα και πειραματισμό. Ταυτόχρονα συμβάλλει στη σωστή ενεργειακή σχεδίαση των χώρων και παρατήρηση των ενεργειακών αναγκών.

Συνοψίζοντας, στο πεδίο υπάρχει δυνατότητα για βελτίωση τόσο στο ερευνητικό σκέλος όσο και στο εφαρμοσμένο. Η εργασία αυτή πραγματεύεται το εφαρμοσμένο κομμάτι της συλλογής, αποθήκευσης και ανάκτησης των δεδομένων.

ΚΕΦΑΛΑΙΟ 3: Συλλογή, αποθήκευση και ανάκτηση δεδομένων

ΣΥΛΛΟΓΗ, ΑΠΟΘΗΚΕΥΣΗ ΚΑΙ ΑΝΑΚΤΗΣΗ ΔΕΔΟΜΕΝΩΝ

Όπως νωρίτερα συναντήσαμε στην εικόνα 2.2.1 τα βασικά στάδια ενός συστήματος Internet of Things και κατ'επέκταση ενός Smart Home είναι η συλλογή, η αποθήκευση και η ανάκτηση των συλλεχθέντων δεδομένων. Το τεχνικό μέρος, ο τρόπος υλοποίησης των σταδίων αυτών αποτελεί το ειδικότερο θέμα που πραγματεύεται η παρούσα εργασία. Τα στάδια αυτά υλοποιήθηκαν σε σκοπιά μικρότερη αυτής ενός εγκατεστημένου συστήματος Smart Grid για λόγους εκμάθησης, με την χρήση δύο συσκευών βασισμένων σε Arduino και Raspberry Pi. Στην εικόνα 3.1 παρουσάζεται αναλυτικά σε ποια μέρη της αρχιτεκτονικής συστημάτων Internet of Things αναφέρονται τα στάδια αυτά ενώ στο παρόν κεφάλαιο αναλύονται η έννοιες και η εφαρμογές σε Smart Homes των τριών αυτών όρων.



Εικόνα 5: Συλλογή, αποθήκευση, ανάκτηση δεδομένων

3.1 Συλλογή Δεδομένων

Ως συλλογή δεδομένων (Data Collection) ορίζεται η διαδικασία συλλογής και μέτρησης πληροφοριών σχετικά με στοχοθετημένες μεταβλητές με καθιερωμένο συστηματικό τρόπο, γεγονός που επιτρέπει σε κάποιον να απαντά σε σχετικές ερωτήσεις και να αξιολογεί τα αποτελέσματα [5]. Η συλλογή δεδομένων αποτελεί

στοιχείο της έρευνας σε όλους τους τομείς σπουδών, συμπεριλαμβανομένων των φυσικών και κοινωνικών επιστημών, των επιχειρήσεων και φυσικά της πληροφορικής. Ενώ οι μέθοδοι ποικίλλουν ανάλογα με την περίπτωση, η έμφαση στη διασφάλιση ακριβούς και ειλικρινούς συλλογής παραμένει η ίδια. Ο στόχος της συλλογής δεδομένων είναι να συλλεχθούν ποιοτικά στοιχεία που να επιτρέπουν στην ανάλυσή τους να οδηγήσει στη διατύπωση πειστικών και αξιόπιστων απαντήσεων στα ερωτήματα που τέθηκαν ή ακόμα και στην πρόβλεψη μελλοντικών καταστάσεων.

3.1.1 Συλλογή Δεδομένων και Smart Homes

Αναφορά στην μεγάλη σημασία της συλλογής δεδομένων που παράγονται από ένα έξυπνο σπίτι έχει ήδη προηγηθεί στο κεφάλαιο 2. Τα είδη των δεδομένων που συλλέγονται εξαρτώνται μόνο από την πληθώρα των εγκατεστημένων έξυπνων συσκευών στο σπίτι. Το γεγονός αυτό δημιουργεί μεν ένα αίσθημα ανασφάλειας το οποίο θα μπορούσε εν μέρη να αντισταθμιστεί αναλογιζόμενοι τις διευκολύνσεις που αυτές παρέχουν στην καθημερινή ζωή, όπως η διαχείριση όλων των συσκευών από ένα smartphone, η εξοικονόμιση ενέργειας κ.α.

Από όλες τις μετρήσεις που είναι δυνατόν να συλλεχθούν από ένα σπίτι, η παρούσα εργασία ξεχωρίζει αυτές της θερμοκρασίας, υγρασίας, VRMS και της παροχής ρεύματος και δουλέψει με αυτές.

3.2 Αποθήκευση δεδομένων

Ο όρος αποθήκευση δεδομένων αναφέρεται στην διατήρηση των αρχείων δεδομένων σε ασφαλή τοποθεσία έτσι ώστε να είναι διαθέσιμα για ανάκτηση οποιαδήποτε στιγμή. Αν τα δεδομένα θεωρούνται πλέον το σημαντικότερο περουσιακό στοιχείο ενός ιδιώτη ή μιας επιχείρησης [6] τότε το μέσο και ο τρόπος που χρησιμοποιείται για την αποθήκευσή τους είναι εξ ίσου σημαντικά.

Σημαντικοί παράγοντες που πρέπει να λαμβάνονται υπ'όψην κάτα την επιλογή τρόπου και μέσου αποθήκευσης δεδομένων είναι:

- **Επεκτασιμότητα:** η τεχνολογία αποθήκευσης δεδομένων που χρησιμοποιείται σε κάθε περίπτωση πρέπει να είναι καλά προσπαρμοσμένη στο μέγεθος των αναγκών του εκάστοτε χρήστη και να έχει την δυνατότητα να καλύψει την επερχόμενη αύξηση της ποσότητας δεδομένων του. Τα συστήματα αποθήκευσης πρέπει εξαρχής να είναι σχεδιασμένα να μπορούν να δεχτούν μεγαλύτερους όγκους δεδομένων χωρίς να χρειάζεται να υποστούν σημαντικές αλλαγές.
- **Κόστη:** αρκετοί παράγοντες που καθορίζουν το κόστος του εκάστοτε μέσου αποθήκευσης στον χρήστη πρέπει να λαμβάνονται υπόψην όπως η αρχική τιμή αγοράς του υλικού αποθηκευτικού μέσου (hardware), η συνεχής συντήρηση υλικού και λογισμικού και άλλα.
- **Επίδοση:** τα συστήματα αποθήκευσης πρέπει να είναι αποδοτικά στο να επιστρέφουν στον χρήστη τα δεδομένα με ταχύτητα.

3.2.1 Αποθήκευση Δεδομένων και Smart Homes (Cloud Storage)

Όπως φαίνεται και στην εικόνα 3.1, από τους πολυάριθμους τρόπους αποθήκευσης των συλλεχθέντων δεδομένων, τα συστήματα Internet of Things χρησιμοποιούν τη

λεγόμενη Αποθήκευση Νέφους (Cloud Storage).

Το Cloud Storage είναι μοντέλο υπολογιστικού νέφους στο οποίο αποθηκεύονται δεδομένα σε απομακρυσμένους διακομιστές (servers) οι οποίοι είναι προσβάσιμοι μέσω του διαδικτύου ή “νέφους” [7]. Τους servers αυτούς διαχειρίζονται, διατηρούν και λειτουργούν οι Πάροχοι Υπηρεσιών Αποθήκευσης Cloud (Cloud Storage service provider). Η χρήση αυτών βασίζεται σε τεχνικές εικονικοποίησης (virtualization).

Το Cloud Storage λειτουργεί χρησιμοποιώντας Data Center Virtualization. Παρέχει δηλαδή στους χρήστες του μία αρχιτεκτονική εικονικού χώρου αποθήκευσης (Virtual Machines, Vms) η οποία είναι κλιμακωτή ανάλογα με τις απαιτήσεις του συστήματος ή της εφαρμογής.

Πλεονεκτήματα που προσφέρονται μεταξύ άλλων από την αποθήκευση των μετρήσεων σε Cloud είναι:

- **Backup:** Το Cloud Storage επιτρέπει την αποθήκευση των δεδομένων σε ένα ασφαλές Server Room σε οποιοδήποτε μέρος του κόσμου. Ενοικιάζεται ο αναγκαίος χώρος από τον πάροχο και τα δεδομένα είναι πλέον Backed Up, που σημαίνει πως ακόμα και μετά την διαγραφή τους οι servers κρατούν ένα αντίγραφο αυτών διαβεβαιώνοντας τον χρήστη πως τα δεδομένα του είναι ασφαλή.
- **Πολυάριθμοι πάροχοι υπηρεσιών Cloud:** με τον μεγάλο αριθμό παρόχων υπηρεσιών Cloud που υπάρχουν είναι σίγουρο πως ο κάθε χρήστης εξασφαλισμένα θα βρει τον χώρο, το εύρος (bandwidth), τις υπηρεσίες backup, προστασίας κ.α. που θα είναι ιδανικά για την περίπτωσή του. Μεταξύ άλλων, γνωστά ονόματα παρόχων αποτελούν οι Google, Microsoft, Amazon και Dropbox. Τα περισσότερα μοντέλα υπηρεσιών των παρόχων αυτών χαρακτηρίζονται ως freemium services. Παρέχονται αρχικά δωρεάν υπηρεσίες με την δυνατότητα αναβάθμισης αυτών με την premium, επί πληρωμής, συνδρομητική την υπηρεσία.
- **Εύκολη πρόσβαση και διαμοιρασμός:** το γεγονός της online αποθήκευσης των δεδομένων δίνει επίσης την δυνατότητα εύκολου διαμοιρασμού αυτών σε πελάτες ή συνεργάτες που δεν έχουν απευθείας πρόσβαση σε αυτά.

Τα Cloud Services δεν θα είχανε κανένα νόημα χωρίς την ύπαρξη μιας μεθόδου για τον χειρισμό τους και την ανάκτηση δεδομένων από αυτά. Τη λειτουργία αυτή επιτελούν τα λεγόμενα APIs, που παρουσιάζονται στην επόμενη παράγραφο του κεφαλαίου.

3.3 Ανάκτηση δεδομένων (Web-Based APIs)

Όπως προαναφέρθηκε, τα Cloud Storages είναι “υπεύθυνα” για την αποθήκευση των μετρήσεων των διαφόρων αισθητήρων και συσκευών του συστήματος. Πώς όμως γίνεται ο χειρισμός αυτών.

Ο χειρισμός του Cloud Storage γίνεται με τη χρήση Web-Based APIs (Application Programming Interface). Ένα API είναι μια συλλογή μεθόδων και διαδικασιών που επιτρέπουν την πρόσβαση σε συγκεκριμένα χαρακτηριστικά και δεδομένα μίας τρίτης εφαρμογής ή υπηρεσίας, προκειμένου να χρησιμοποιηθούν για την ανάπτυξη εφαρμογών. Ουσιαστικά, με την χρήση των Web-Based APIs δίνεται η δυνατότητα στον πελάτη-χρήστη του Cloud Storage να διαχειρίζεται και να ανατρέχει την βάση δεδομένων του δηλαδή τον χώρο αποθήκευσης που αυτός δεσμεύει. Βάσεις δεδομένων που, στην περίπτωση των Smart Houses που πραγματεύεται αυτή η εργασία, θα

περιέχουν μετρήσεις των διαφόρων αισθητήρων και συσκευών.

Η σημασία των API καθίσταται μεγάλη καθώς η χρήση τους προσφέρει πλεονεκτήματα και στον “ιδιοκτήτη” τους αλλά και στον χρήστη. Για τον ιδιοκτήτη, όσο χρησιμοποιείται το API του σημαίνει πως τα προϊόντα και οι υπηρεσίες του κερδίζουν σε δημοσιότητα ενώ εξακολουθεί να διατηρεί την κυριότητα του κώδικα. Για τον χρήστη, τα API ανακουφίζουν, θα λέγαμε, τους developers από την δυσκολία και την προσπάθεια που θα κατέβαλαν να δημιουργήσουν ένα ολόκληρο λογισμικό από το μηδέν.

Γενικότερα, η επιπτώσεις της χρήσης των APIs μπορούν να συμπτυχθούν σε 3 κατηγορίες, όπως φαίνεται στην εικόνα 3.3.1



Εικόνα 6: Επιπτώσεις της χρήσης των APIs

3.3.1 Αύξηση του όγκου δεδομένων

Τα APIs έχουν συμβάλει σημαντικά στην ανάπτυξη της επιστήμης των δεδομένων αλλά και στην αύξηση του όγκου αυτών που χρησιμοποιούνται από εφαρμογές.

Διευκολύνουν δραστικά τη σύνδεση μίας εφαρμογής σε περισσότερες από μία πηγές δεδομένων (Data Sources) την ίδια στιγμή. Για την πρόσβαση σε αυτές, οι χρήστες αρκεί να “καλέσουν” το αντίστοιχο API για να προσφερθεί η πληροφορία που ζητήθηκε. Όλο και περισσότεροι άνθρωποι, προγραμματιστές και μη, χρησιμοποιούν τα APIs για τις ευκολίες που παρέχουν.

3.3.2 Γρήγορη πρόσβαση στα δεδομένα

Τα APIs μπορούν να βοηθήσουν στην ανάπτυξη εφαρμογών δεδομένων και μεγάλων δεδομένων (Big Data applications) καθώς προσφέρουν γρηγορότερη πρόσβαση στις αποθήκες δεδομένων. Το γεγονός αυτό οδηγεί σε ταχύτερη ανάκτηση, επεξεργασία και ανάλυση δεδομένων. Τέτοια API υπάρχουν ως ένα στρώμα μεταξύ κατανεμημένων εφαρμογών πληροφορικής και των χώρων αποθήκευσης.

3.3.3 Ανάπτυξη νέων προϊόντων και υπηρεσιών

Η εμβέλεια της χρησιμότητας των API δεν περιορίζεται στην επικοινωνία μιας εφαρμογής με κάποια αποθήκη/βάση δεδομένων.

Παραδειγματικά, μία νέα τεχνολογία στον κόσμο των API αποτελούν τα “Γνωστικά API” (cognitive APIs) και βρίσκουν ιδιαίτερη εφαρμογή στον κλάδο των αναλύσεων (analytics). Τα γνωστικά API δέχονται αιτήματα (requests) σε συγκεκριμένη μορφή από ένα σύστημα και παραδίδουν σε ένα άλλο. Το σύστημα-παραλήπτης παρέχει analytics ως απάντηση, η οποία παραδίδεται πίσω στο αιτούμενο σύστημα.

3.4 Η προσέγγιση αυτής της εργασίας

Κρίνοντας από προηγούμενες δουλειές στον τομέα και εξετάζοντας συσκευές και μεθόδους καταλληλότερες για εκμάθηση, η εργασία αυτή επικεντρώθηκε στην υλοποίηση των τριών προαναφερθέντων σταδίων της αρχιτεκτονικής του Internet of Things και κατ'επέκταση των συστημάτων Smart Homes.

Ως εκ τούτου, για την ευκολότερη λειτουργία και εγκατάσταση αισθητήρων στον χώρο χρησιμοποιήθηκαν δύο συσκευές του οργανισμού ανοιχτού κώδικα OpenEnergyMonitor [8].

Για την αναπαραγωγή remote data storage χρησιμοποιήθηκε local server σε προσωπικό υπολογιστή στον οποίο “στήθηκε” η απαραίτητη βάση δεδομένων.

Για την ανάκτιση των δεδομένων από αυτή κατασκευάστηκε ένα API με την χρήση της βιβλιοθήκης Flask και της γλώσσας προγραμματισμού Python.

Ακόμα, υλοποιήθηκε ένα πρότυπο σύστημα ειδοποίησης του χρήστη για την κατάσταση ηλεκτρικών συσκευών βασισμένο στο NodeRed, με περιορισμένες δυνατότητες. Ανάλυση αυτών και περισσότερων εργαλείων παρουσιάζεται στο κεφάλαιο 4.

ΚΕΦΑΛΑΙΟ 4: Περιγραφή των εργαλείων που χρησιμοποιήθηκαν

ΠΕΡΙΓΡΑΦΗ ΤΩΝ ΕΡΓΑΛΕΙΩΝ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΑΝ

Στο παρόν κεφάλαιο περιγράφονται τα εργαλεία που χρησιμοποιήθηκαν για την εκπνόηση της εργασίας. Ονοματικά, αυτά είναι:

- Η συσκευή emonPi του οργανισμού ανοιχτού κώδικα OpenEnergyMonitor ως gateway
- Η συσκευή emonTx του οργανισμού ανοιχτού κώδικα OpenEnergyMonitor για την χρήση των διάφορων αισθητήρων
- Το λογισμικό LAMP (Linux, Apache, MySQL, PHP) για την εγκατάσταση local server σε προσωπικό υπολογιστή
- Το λογισμικό phpMyAdmin για την διαχείριση της βάσης δεδομένων
- To Sublime Text, text editor για την σύνταξη του κώδικα
- To Node-Red για την αποστολή των μετρήσεων στην βάση δεδομένων και για τη δημιουργεία του συστήματος ειδοποίησεων μέσω Gmail και Tweeter

Αναλυτικά:

4.1 Ο οργανισμός OpenEnergyMonitor

Ο οργανισμός OpenEnergyMonitor ιδύθηκε από τους Trystan Lea και Glyn Hudson και έχει τη βάση του στο Eryri (Snowdonia) North Wales. Το project βασίστηκε στην επιθυμία για την ύπαρξη εργαλείων ανοιχτού κώδικα που θα βοηθήσουν τον κόσμο να γνωρίσει καλύτερα την ενεργειακή του κατανάλωση, τα ενεργειακά συστήματα αλλά και με την πρόκληση της βιώσιμης ενέργειας. Τα πιστεύω των ιδρυτών του περιλαμβάνουν τις αντιλήψεις πως στα επόμενα 20 χρόνια θα υπάρξει δραστικότατη αλλαγή στα συστήματα ενέργειας του κόσμου κατά την μεταβίβαση από την χρήση στερεών καυσίμων σε μία παροχή ενέργειας μηδενικής κατανάλωσης άνθρακα. Θεωρήθηκε απαραίτητο λοιπόν η συχέτιση του κόσμου με τις ενεργειακές τους καταναλώσεις προκειμένου να εξομαλυνθεί η μετάβαση αυτή όσο το δυνατόν περισσότερο [8]. Όλο το hardware και firmware που παρέχει ο οργανισμός είναι open source και διαθέσιμα στο GitHub [9].

4.1.1 Η συσκευή emonPi

Η συσκευή emonPi αποτελεί το monitor του συστήματος συσκευών της σειράς emon (περισσότερες από τις συσκευές που αναφέρονται σε αυτή την εργασία διατίθενται προς πώληση από τον οργανισμό που δεν ήταν αναγκαίο να χρησιμοποιηθούν). Το project της συσκευής έγινε εφικτό μέσω χρηματοδότησης του οργανισμού από την διάσημη πλατφόρμα crowdfunding, KickStarter, όπου συλλεχθήκαν συνολικά 24.723 Αγγλικές λίρες από 159 άτομα [10].

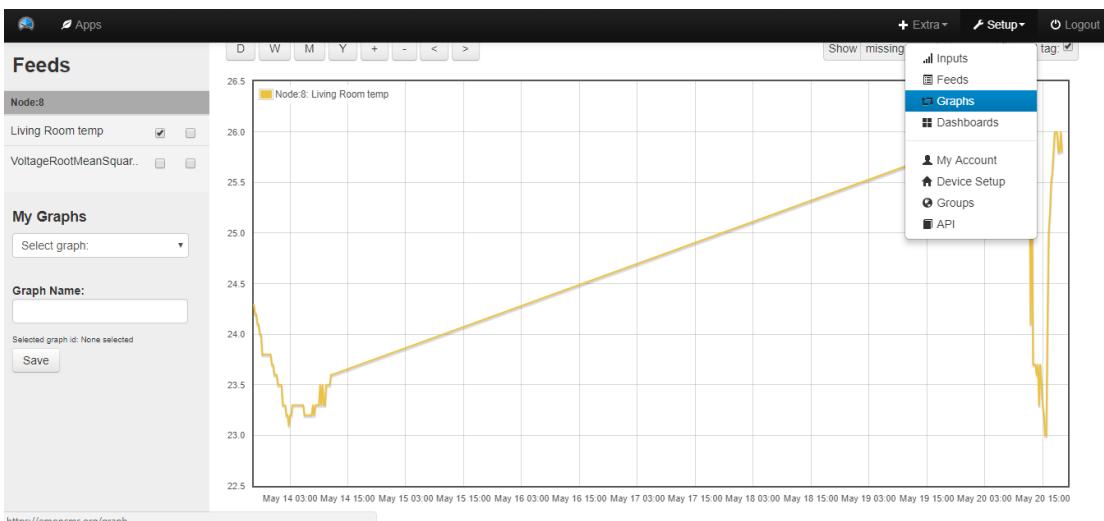


Εικόνα 7: Η συσκευή emonPi

Ο κόμβος (node) emonPi, της σειράς emon, είναι μία συσκευή παρακολούθησης ενέργειας αλλά και άλλων τιμών (θερμοκρασία, υγρασία κ.α.) βασισμένη σε ένα Raspberry Pi. Όπως συμβαίνει για οποιαδήποτε χρήση του Raspberry Pi έτσι και εδώ, προκειμένου το Raspberry να λειτουργήσει ως emonPi είναι απαραίτητη η ύπαρξη μίας SD card για τον προσδιορισμό των λειτουργιών του καθώς και ενός Raspberry Shield συνδεδεμένο σε αυτό, ονόματι emonPi Shield. Η κάρτα αυτή (EmonSD) είναι ήδη “στημένη” από τον οργανισμό και διατήθεται με την αγορά του προϊόντος.

Για την λειτουργία του emonPi απαραίτητη επίσης είναι, εκτός από την παροχή ρεύματος, η σύνδεση Ethernet αυτού με υπάρχον Router, μέσω της ακροδεξιάς θύρας που φαίνεται στην εικόνα 4.1.1.1. Το emonPi μπορεί να εκτελέσει και καθήκοντα κόμβου-αισθητήρα (sensor node) καθώς είναι δυνατή η προσκόμιση σε αυτό αισθητήρων CT, θερμοκρασίας και pulse sensor. Στην εκπνόηση της παρούσας πτυχιακής ωστόσο χρησιμοποιήθηκε αποκλειστικά ως gateway και προτιμήθηκε η συσκευή emonTx ως node sensor λόγω της μεγάλης απόστασης του ηλεκτρικού πίνακα του σπιτιού όπου εγκαταστάθηκε το σύστημα με το Router.

Τέλος, η χρήση του emonPi παρέχει remote αλλά και local data logging μέσω της υπηρεσίας EmonCMS [11]. Το emonCMS αποτελεί το web app του οργανισμού για την remote παρακολούθηση και όχι μόνο των δεδομένων που συλλέγονται από το σύστημα. Η ίδια υπηρεσία παρέχεται και τοπικά, εγκατεστημένη στον local server του Raspberry Pi.



Εικόνα 8: Γραφική αναπαράσταση τιμών στο EmonCMS

4.1.2 Η συσκευή emonTx

Όπως προαναφέρθηκε, η συσκευή emonTx αποτελεί ένα remote sensor node. Σε αυτή είναι δυνατόν να συνδεθούν αισθητήρες CT, temperature, humidity, AC. Τα δεδομένα αυτών μεταδίδονται στο emonPi με μία συχνότητα χαμηλής ισχύος των 433MHz. Η εμβέλεια της συχνότητας αυτής είναι σχεδόν ίση με αυτής ενός home Wi-Fi και μπορεί να επηρεαστεί από εμπόδια όπως χοντροί τοίχοι.

Το emonTx έχει την δυνατότητα monitoring έως και τεσσάρων κυκλωμάτων ταυτόχρονα, καθώς δέχεται μέχρι τέσσερις CT sensors, όπως φαίνεται στην εικόνα 4.1.2.1. Επιπλέον, ο AC adapter που χρησιμοποιείται για την τροφοδότησή του παρέχει δείγματα της AC τάσης για υπολογισμούς πραγματικής ισχύος.

Τέλος, μέχρι δύο emonTx μπορούν να είναι συνδεδεμένα ταυτόχρονα σε ένα emonPi. Στην εικόνα 4.1.2.2 φαίνονται με την σειρά, οι αισθητήρες CT, temperature και AC από τα αριστερά προς τα δεξιά ενώ στον πίνακα 1 παρουσιάζεται η αναλυτική σύγκριση των δύο συσκευών.



Eikόνα 9: To emonTx με 4 αισθητήρες CT και έναν AC adapter



Eikόνα 10: CT, temperature, AC sensor

EmonPi		EmonTx
Κύρια λειτουργεία	Monitor & gateway	Remote energy monitor
Αριθμός CT sensors	2	4

	EmonPi	EmonTx
Αριθμός voltage sensors	1	1
Αριθμός temperature sensors	0	1 τύπου RJ45
Παροχή ενέργειας	Μέσω 5V USB mini-B	Μέσω 9V AC/AC ή με χρήση 3 AAA μπαταριών
Τοπική αποθήκευση δεδομένων (emonCMS)	Ναι	Όχι
LCD οθόνη	Ναι	Όχι

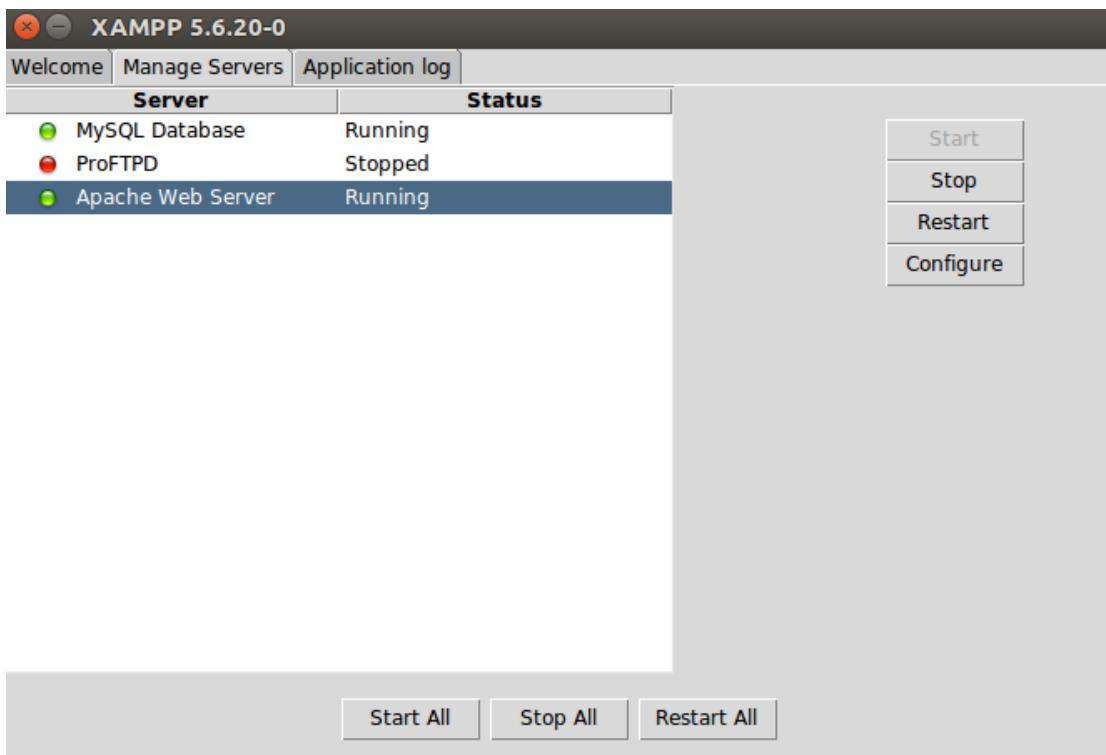
Πίνακας 1: Σύγκριση των emonPi – emonTx

4.2 Το λογισμικό LAMP

Το LAMP είναι ένα αρχέτυπο μοντέλο στοίβας υπηρεσιών ιστού. Το όνομά του αποτελεί ακρονύμιο των τεσσάρων open source συστατικών του: Linux operating system, Apache HTTP Server, MySQL relational database system (RDBMS) και PHP. Το LAMP είναι ιδιαίτερα χρήσιμο για την κατασεύνη δυναμικών ιστοσελίδων και web applications. Ισάξιες εκδόσεις για διαφορετικά λογισμικά αποτελούν τα WAMP, για windows και το MAMP για macOS.

Επίσημα το LAMP προοριζόταν ως εργαλείο ανάπτυξης και δοκιμής ιστοσελίδων τοπικά στον υπολογιστή χωρίς να είναι απαραίτητη η σύνδεση στο διαδίκτυο. Για να είναι δυνατή η χρήση του, πολλές σημαντικές λειτουργίες ασφάλειας έχουν απενεργοποιηθεί. Στην πράξη το LAMP ορισμένες φορές χρησιμοποιείται και για την φιλοξενία ιστοσελίδων. Το LAMP υποστηρίζει την δημιουργία και διαχείριση βάσεων δεδομένων τύπου MySQL.

Όταν το LAMP εγκατασταθεί στον τοπικό υπολογιστή διαχειρίζεται τον localhost ως ένα απομακρυσμένο κόμβο, ο οποίος συνδέεται με το πρωτόκολλο μεταφοράς αρχείων FTP. Η σύνδεση στον localhost μέσω του FTP μπορεί να γίνει με το όνομα χρήστη «newuser» και το κωδικό «wampp». Για την βάση δεδομένων MySQL υπάρχει ο χρήστης «root» χωρίς κωδικό πρόσβασης.



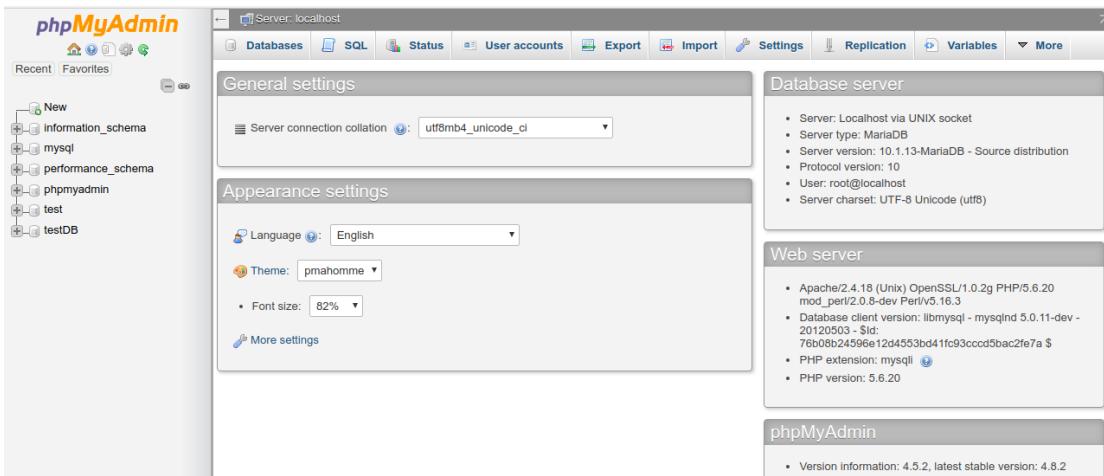
Εικόνα 11: To software bundle LAMP

4.3 Το λογισμικό phpMyAdmin

Το phpMyAdmin είναι ένα εργαλείο ελεύθερου λογισμικού γραμμένο σε PHP, το οποίο προορίζεται για τη διαχείριση βάσεων δεδομένων MySQL. Το phpMyAdmin υποστηρίζει ένα ευρύ φάσμα λειτουργιών στις υπηρεσίες MySQL και MariaDB. Συχνά χρησιμοποιούμενες λειτουργίες (διαχείριση βάσεων δεδομένων, πίνακες, στήλες, σχέσεις, ευρετήρια, χρήστες, δικαιώματα κ.λ.π.) μπορούν να εκτελεστούν μέσω του γραφικού περιβάλλοντος χρήστη, ενώ υπάρχει πάντα η δυνατότητα να εκτελεστεί άμεσα οποιοδήποτε query SQL. Από το 2000 μέχρι σήμερα παραμένει ένα από τα πιο δημοφιλή εργαλεία διαχείρισης MySQL, ειδικά για web hosting services.

Οι λόγοι επιλογής του phpMyAdmin έναντι άλλων εργαλείων διαχείρισης βάσεων δεδομένων παρατίθενται παρακάτω:

- Παρέχει web interface
- Δυνατότητα εισαγωγής δεδομένων από CSV
- Δυνατότητα εξαγωγής δεδομένων σε πολυάριθμες μορφές όπως CSV, SQL, XML, PDF, LaTeX κ.α.
- Η λειτουργία του σε διαφορετικά OS
- Η μεγάλη διευκόλυνση που παρέχει για την σύνταξη πολύπλοκων SQL queries

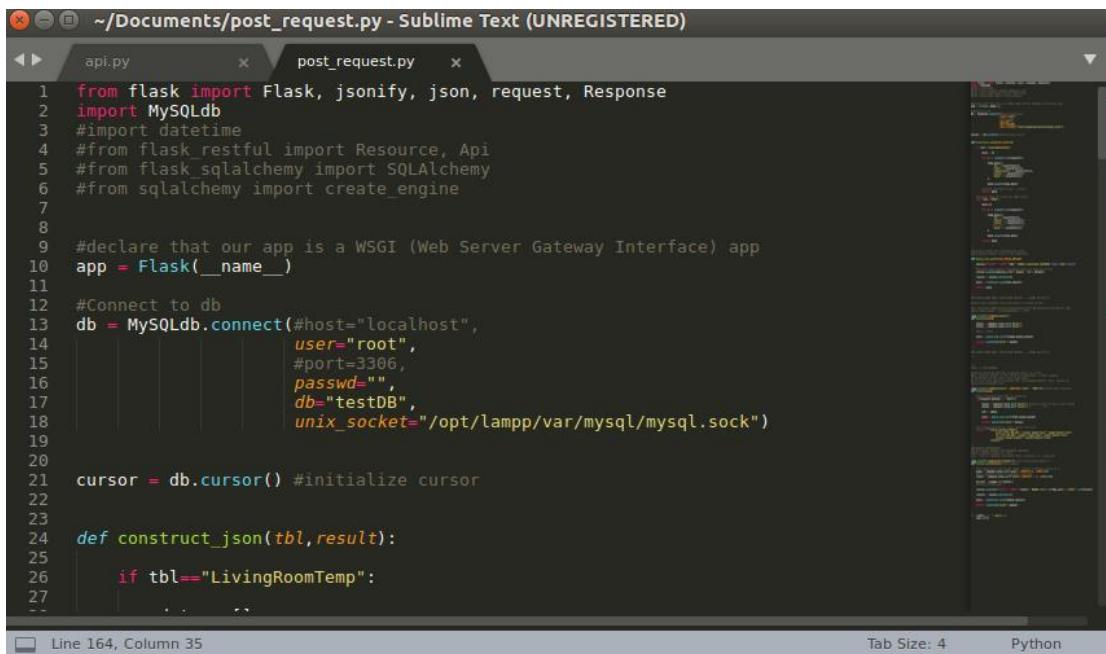


Εικόνα 12: phpMyAdmin

4.4 Sublime Text

To Sublime Text είναι ένα cross platform source code editor που υποστηρίζει τη σύνταξη προγραμμάτων στις περισσότερες χρησιμοποιούμενες γλώσσες προγραμματισμού αλλά και σε markup languages (π.χ. HTML). Λειτουργίες μπορούν να προστεθούν σε αυτό ως plugins από τους χρήστες, τα οποία είναι κυρίως κατασκευασμένα από την κοινότητα του Sublime Text.

Ενώ άλλα IDE μπορεί να προσφέρουν περισσότερες διευκολύνσεις στην σύνταξη κώδικα Python (π.χ. PyCharm) το Sublime Text προτιμήθηκε έναντι αυτών λόγω της οικειότητας του συγγραφέα με αυτό από προηγούμενα project



```

1  from flask import Flask, jsonify, json, request, Response
2  import MySQLdb
3  #import datetime
4  #from flask_restful import Resource, Api
5  #from flask_sqlalchemy import SQLAlchemy
6  #from sqlalchemy import create_engine
7
8
9  #declare that our app is a WSGI (Web Server Gateway Interface) app
10 app = Flask(__name__)
11
12 #Connect to db
13 db = MySQLdb.connect(host="localhost",
14                      user="root",
15                      port=3306,
16                      passwd="",
17                      db="testDB",
18                      unix_socket="/opt/lampp/var/mysql/mysql.sock")
19
20
21 cursor = db.cursor() #initialize cursor
22
23
24 def construct_json(tbl,result):
25
26     if tbl=="LivingRoomTemp":
27         ...
28

```

Line 164, Column 35 Tab Size: 4 Python

Εικόνα 13: SublimeText

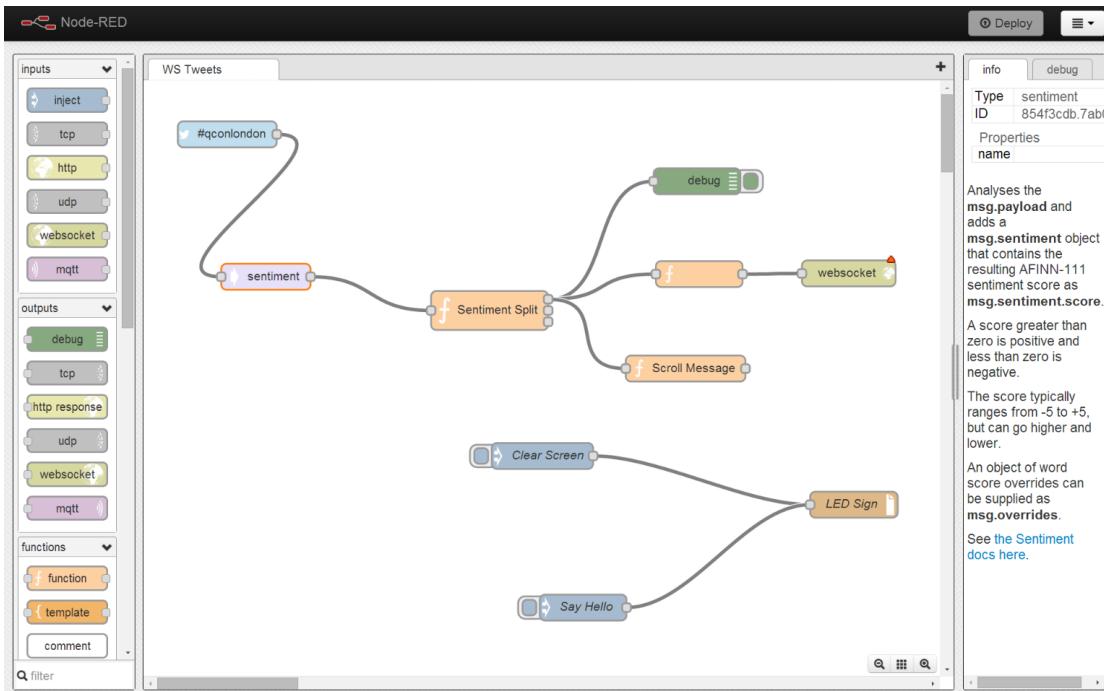
4.5 Το εργαλείο Node-Red

Το Node-Red είναι ένα flow-based εργαλείο προγραμματισμού αρχικά σχεδιασμένο από την ομάδα Emerging Technology Services του οργανισμού IBM [12] και μέρος του JS Foundation από το 2016 μέχρι και σήμερα. Από τον Σεπτέμβριο του 2013 και μετά είναι open source με τον κώδικά του διαθέσιμο στο Git Hub.

Αποτελεί ένα εργαλείο για την διασύνδεση hardware συσκευών, APIs και online υπηρεσιών. Είναι προσβάσιμο μέσω browser. Βασισμένο σε Node.js καθίσταται ιδανικό για χρήση σε συσκευές όπως το Raspberry Pi αλλά και για το cloud. Η εισαγωγή κόμβων στην ροή γίνεται με drag and drop και η σύνδεση αυτών μεταξύ τους με το ποντίκι. Οι ροές (flows) που δημιουργούνται στο Node-Red μπορούν να αποθηκευτούν σε μορφή JSON της οποίας η εισαγωγή σε και εξαγωγή από αυτό καθίσταται ιδιαίτερα εύκολη. Με αυτό τον τρόπο επιτρέπεται ο διαμοιρασμός flows μεταξύ των ατόμων των κοινοτήτων. Επιπλέον, δίνεται η δυνατότητα σύνταξης κώδικα JavaScript σε συγκεκριμένους κόμβους για οποιαδήποτε custom χρήση.

4.5.1 Flow-based programming

Επινοημένο από τον J. Paul Morrison το 1970, ο flow-based προγραμματισμός αποτελεί έναν τρόπο περιγραφής της συμπεριφοράς μίας εφαρμογής ως δίκτυο κόμβων (nodes). Κάθε node έχει σαφώς καθορισμένο σκοπό. Σε κάθε κόμβο δίνονται δεδομένα, επιτελούνται κάποιες λειτουργίες με αυτά και στη συνέχεια τα δεδομένα μεταβιβάζονται στον επόμενο κόμβο. Το δίκτυο είναι υπεύθυνο για την ροή των δεδομένων μεταξύ των κόμβων (εικόνα 14).



Εικόνα 14: Εφαρμογή σε Node-Red

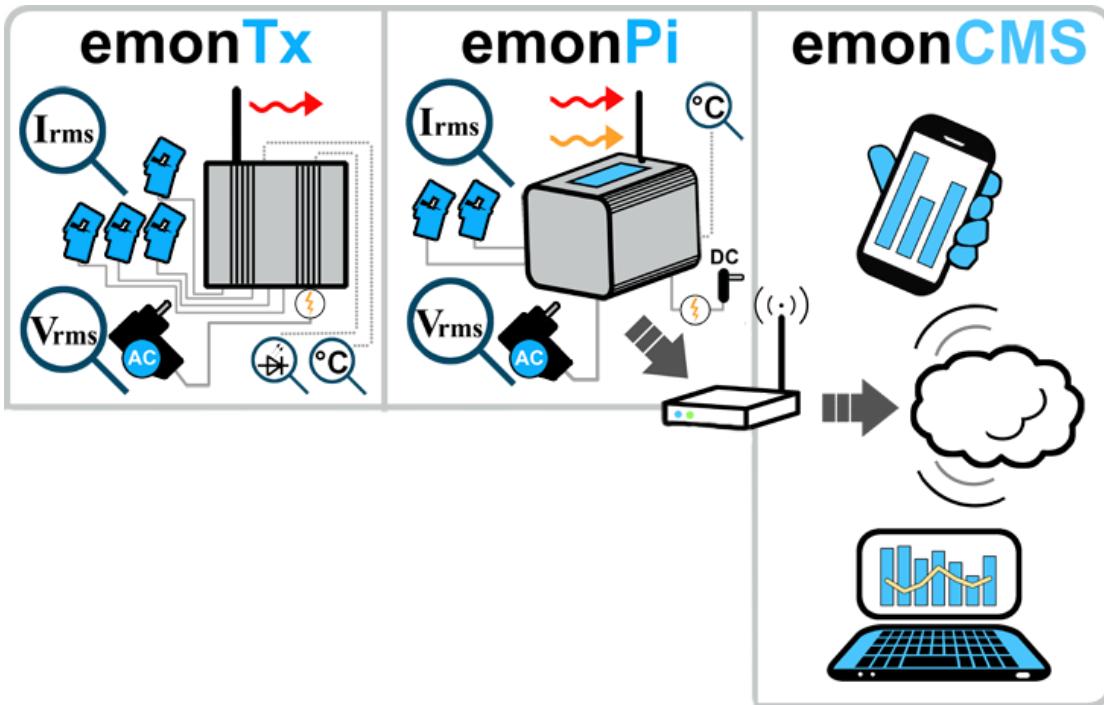
ΚΕΦΑΛΑΙΟ 5: Αρχιτεκτονική των συστημάτων

ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΩΝ ΣΥΣΤΗΜΑΤΩΝ

Κρίνεται αναγκαίο να επεξηγηθεί ο τρόπος λειτουργίας των συστημάτων που χρησιμοποιούνται στην παρούσα εργασία πριν επέλθει η ανάλυση του τρόπου υλοποίησης (βλ. Κεφάλαιο 6), για την πλήρη κατανόηση αυτού από τον αναγώστη.

5.1 Αρχιτεκτονική εγκατεστημένου συστήματος συσκευών της σειράς emon

Η αρχιτεκτονική που ακολουθείται σε ένα σύστημα συσκευών της σειράς emon, προφανώς παρουσιάζει, όπως θα περιμέναμε, μεγάλη ομοιότητα σε σχέση με αυτή της γενικής αρχιτεκτονικής του Internet of Things (ΑΡΙΘΜΟΣ ΕΙΚΟΝΑΣ IoT CYCLE ΚΕΦ 2). Όπως προαναφέρθηκε, στην παρούσα εργασία χρησιμοποιήθηκαν αποκλειστικά οι συσκεύες emonPi και emonTx και λόγω αυτού δεν γίνεται αναφορά στις υπόλοιπες υπάρχουσες.



Eikόνα 15: Αρχιτεκτονική συστήματος συσκευών emonPi & emonTx

Στην αριστερή πλευρά της εικόνας 578 παρουσιάζεται το emonTx με τους διάφορους αισθητήρες συνδεδεμένους σε αυτό (στην περίπτωσή μας χρησιμοποιούνται ένας CT (Irms), ένας temperature και ένας Vrms). Με το κόκκινο βελάκι αντιπροσωπεύεται η μετάδοση των μετρήσεων στο emonPi, που λειτουργεί (στην περίπτωσή μας) αποκλειστικά ως gateway. Ακολουθεί η σύνδεση αυτού μέσω Ethernet με υπάρχον Router και τέλος η αποστολή των δεδομένων στο cloud από όπου είναι πλέον δυνατή η ανάγνωση των μετρήσεων μέσω του web application emonCMS.

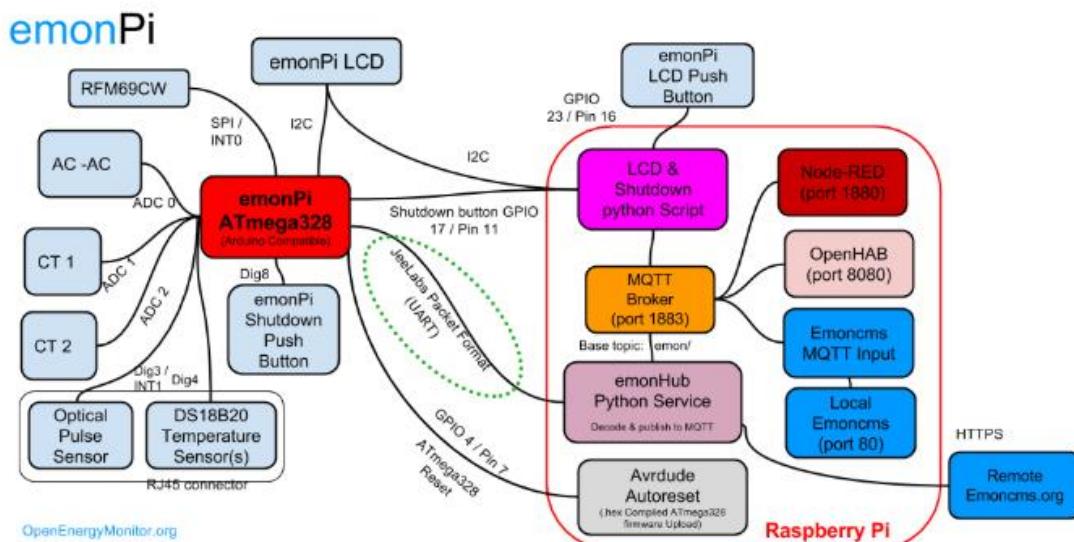
Μεγάλη σημασία αλλά και ενδιαφέρον παρουσιάζει η αρχιτεκτονική των συσκευών και ειδικότερα αυτή του emonPi όπου επεξηγείται ο τρόπος λειτουργίας του, οι

υπηρεσία (Python service) λήψης, αποκωδικοποίησης και αποστολής των δεδομένων στο cloud καθώς και το πρωτόκόλλο επικοινωνίας που χρησιμοποιείται.

5.1.1 Αρχιτεκτονική του emonPi

Από την αρχιτεκτονική της συσκευής emonPi, αν και δεν είναι ιδιαίτερα πολύπλοκη, αναλύονται μόνο τα μέρη που είναι απαραίτητα για την κατανόηση του τρόπου υλοποίησης της εργασίας. Στην EIKONA 1684 παρουσιάζεται ολοκληρωμένη.

Όπως έχει αναφερθεί στην παράγραφο 4.1.1, το Raspberry Pi στο οποίο βασίζεται η συσκευή emonPi έχει συνδεδεμένο (attached) σε αυτό το emonPi Shield. Το emonPi shield διαθέτει έναν μικροεπεξεργαστή ATmega328 [13] για τις λειτουργίες του, ο οποίος είναι και συμβατός με Arduino. Στην EIKONA 6514 παρουσιάζεται η αρχιτεκτονική της συσκευής emonPi όπου φαίνεται η επικοινωνία του μικροεπεξεργαστή με το Raspberry (με πράσινο διακεκομμένο).



Εικόνα 16: Αρχιτεκτονική του emonPi

Η επικοινωνία επιτυγχάνεται σειριακά μέσω του υπάρχοντος UART (Universal Asynchronous Receiver/Transmitter) του Raspberry Pi. Τα δεδομένα μεταδίδονται με μορφή JeeLib κωδικοποίησης [15] όπου αποκωδικοποιούνται με την υπηρεσία (service) emonHub.

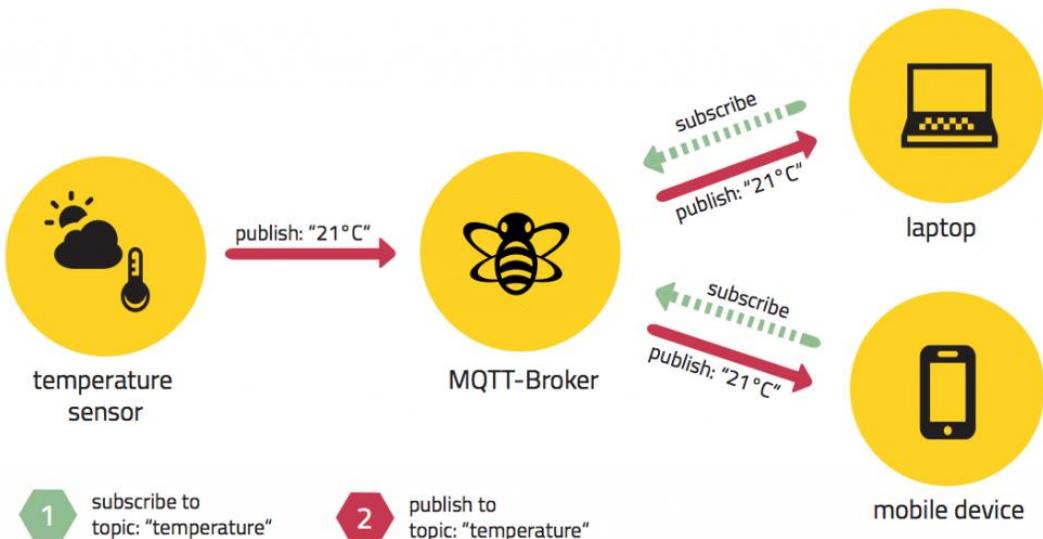
Το emonHub είναι ένα service γραμμένο σε Python το οποίο καθίσταται υπεύθυνο για την αποκωδικοποίηση των δεδομένων που λαμβάνει το Raspberry και την αποτύπωσή τους στον local Mosquitto MQTT server του emonPi καθώς και στο remote emonCMS, όπως φαίνεται στην εικόνα 16.

5.1.2 Αρχιτεκτονική του MQTT Protocol

Το MQTT Protocol είναι ένα Client Server publish/subscribe πρωτόκόλλο Machine to Machine (M2M) επικοινωνίας. Είναι χαρακτηριστικά “ελαφρύ” και ανοιχτού κώδικα και αποδίδει άριστα όταν πρόκειται για σειριακή μεταφορά δεδομένων, συγκριτικά καλύτερα από πρωτόκολλα όπως το HTTP [14].

Η μέθοδος publish/subscribe είναι μία εναλλακτική της παραδοσιακής αρχιτεκτονικής

client-server. Σε ένα μοντέλο client-server, ο client επικοινωνεί απευθείας με κάποιο endpoint. Το μοντέλο publish/subscribe αποσυνδέει πλέον τον client που στέλνει το μήνυμα (publisher) από τον client ή clients που το λαμβάνουν (subscribers). Οι publishers και subscribers δεν επικοινωνούν ποτέ απευθείας μεταξύ τους. Την μεταξύ τους σύνδεση χειρίζεται ένα τρίτο στοιχείο, ο broker. Σκοπός του broker είναι να φιλτράρει τα εισερχόμενα μηνύματα και να τα διανείμει σωστά στους subscribers που πρέπει.

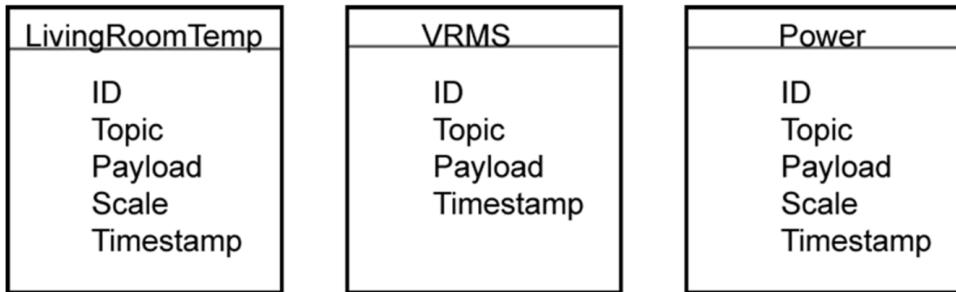


Eικόνα 17: Το μοντέλο publish/subscribe

Το φιλτράρισμα των μηνυμάτων και ο διαχωρισμός τους στους κατάλληλους subscribers καθορίζεται σύμφωνα με το σε ποιο topic είναι εγγεγραμμένος (*subscribed*) κάθε subscriber. Το topic είναι ένα μέρος του μηνύματος που μεταδίδεται, με σκοπό τον σωστό αυτό διαχωρισμό μηνυμάτων. Παρατηρούμε στην εικόνα 16 πως κάτω από τον MQTT Broker (με πορτοκαλί χρώμα) αναγράφεται “Base topic: emon/”. Αυτό είναι το βασικό “θέμα” του emonPi και ένα subscription στο topic emon/ θα έδινε στον client όλα τα δεδομένα που γίνονται publish με το topic αυτό, δηλαδή τις μετρήσεις όλων των αισθητήρων. Για κάτι πιο συγκεκριμένο π.χ. την απόκτηση των δεδομένων μόνο ενός συγκεκριμένου αισθητήρα θερμοκρασίας του emonTx/emonPi θα πρέπει να γίνει subscribe στο topic emon/temperature1. Τα ονόματα των topics γίνονται γνωστά διαβάζοντας την μορφή ολόκληρου του μηνύματος που μεταδίδεται, κάτι που σε αυτή την εργασία καθίσταται εφικτό με την χρήση του Node-Red.

5.2 Αρχιτεκτονική της Βάσης Δεδομένων

Η αρχιτεκτονική της βάσης δεδομένων που δημιουργήθηκε και χρησιμοποιήθηκε είναι αρκετά απλή καθώς δεν είναι απαραίτητο να υπάρχουν συνδέσεις μεταξύ των tables. Δημιουργήθηκε ένα table για κάθε αισθητήρα του οποίου λαμβάνονται οι μετρήσεις με ονομασίες LivingRoomTemp για τον αισθητήρα θερμοκρασίας, VRMS για τις μετρήσεις vrms και Power για τον CT sensor.



Eikόνα 18: Database Schema

Τα πεδία είναι κοινά για όλα τα tables και έχουν ως εξηγήσεις:

- ID: Primary Key κάθε table, set ως auto_increment, αποτελεί ένα unique identifier για κάθε καταχώρηση
- Topic: Τι είδους μέτρηση δείχνει η καταχώρηση. Για παράδειγμα στο table LivingRoomTemp το πεδίο topic έχει default τιμή LivingRoomTemperature, στο VRMS table, vrms κ.ο.κ.
- Payload: Η τιμή της μέτρησης (αριθμός).
- Scale: Η μονάδα μέτρησης. Στο LivingRoomTemp table το πεδίο scale έχει default τιμή “Celcius” ενώ στο table Power, “Ampere”.
- Timestamp: Η ακριβής ώρα της καταχώρησης. Default value CURRENT_TIMESTAMP ενώ έχει οριστεί για το πεδίο αυτό ON UPDATE CURRENT_TIMESTAMP.

5.3 Αρχιτεκτονική των εφαρμογών Node-Red

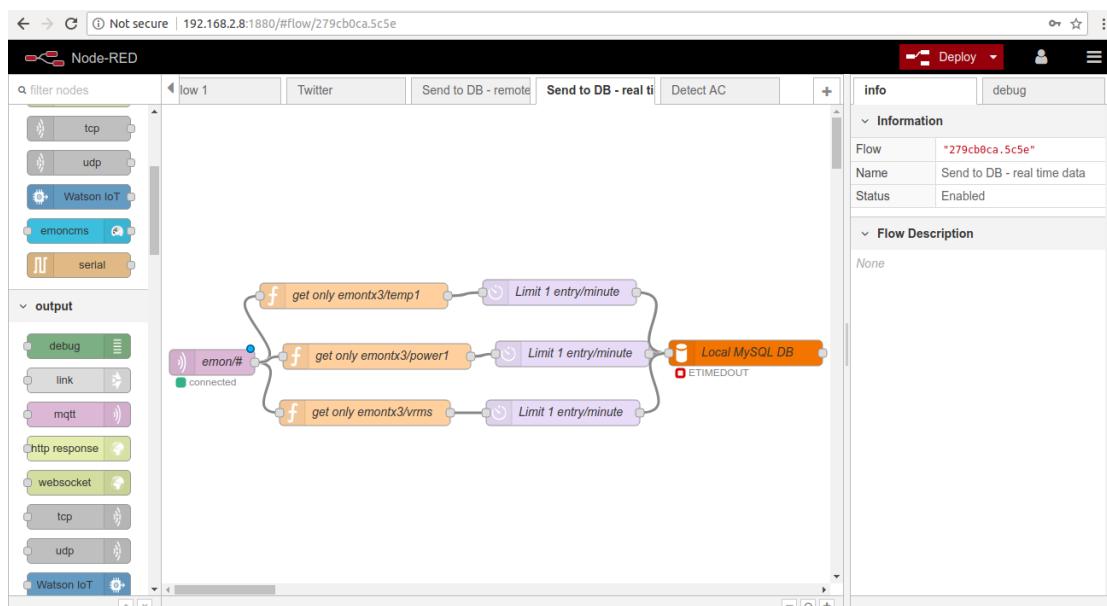
Με τη χρήση του εργαλείου Node-Red έχουν υλοποιηθεί, όπως προαναφέρθηκε, το κομμάτι τις αποστολής των μετρήσεων στην τοπική βάση δεδομένων (db) καθώς και αυτό της πειραματικής ανίχνευσης λειτουργίας του Air-Condition δωματίου σε δύο εφαρμογές. Κρίνεται σημαντικό να παρουσιαστούν οι αρχιτεκτονικές, τα flows, των εφαρμογών αυτών πρωτού επέλθει η αναλυτική παρουσίαση της συνολικής υλοποίησης της εργασίας, για την διασφάλιση της κατανόησης αυτής από τον αναγνώστη.

Μέσω του Node-Red, η αποστολή των δεδομένων στην τοπική βάση μπορεί να γίνει με δύο τρόπους: αποστολή μετρήσεων σε πραγματικό χρόνο (Real-Time), συλλέγοντας

τις τιμές απευθείας από το κατάλληλο MQTT topic ή ανάκτηση αυτών από τον remote server του emonCMS και αποθήκευσή τους. Στην τελευταία περίπτωση παρουσιάζεται μία καθυστέρηση στην ανανέωση των τιμών, σε σχέση με την ταχύτητα που αυτές συλλέγονται από τους αισθητήρες (κάθε 5 δευτερόλεπτα). Κατά την εκπνόηση της εργασίας υλοποιήθηκαν και οι δύο λειτουργίες. Σημειώνεται πως για την σωστή λειτουργία κάθε node είναι απαραίτητο το κατάλληλο configuration αυτού, στο οποίο υπάρχει πρόσβαση με διπλό αριστερό mouse click επάνω στο εκάστοτε node.

5.3.1 Αρχιτεκτονική εφαρμογής *Real-TimeDB input*

Στην εικόνα 19 παρουσιάζεται το flow της εφαρμογής αποθήκευσης δεδομένων σε πραγματικό χρόνο.



Εικόνα 19: Flow της εφαρμογής *Real-TimeDB input*

Αριστερά, με μοβ χρώμα, παρατηρούμε το επονομαζόμενο “MQTT In” node. Το node αυτό λαμβάνει απευθείας τις τιμές που αποστέλονται στον MQTT broker στον οποίο είναι συνδεδεμένο (καθορίζεται στο node configuration) και κάνει subscribe στα topic αυτού που του ορίζονται. Στην παρούσα περίπτωση, το MQTT In node είναι subscribed στο topic “*emon/#*”, εξού και η ονομασία του στην εικόνα. Το σύμβολο # αποτελεί wildcard character για το Node-Red, που στην παρούσα περίπτωση σημαίνει πως το node δέχεται όλα τα μηνύματα των οποίων τα topic ξεκινούν με “*emon/*” (στην παράγραφο 5.1.2 έχει αναφερθεί η σημασία του συγκεκριμένου topic).

Το *emon/#* node περνάει στην συνέχεια τα μηνύματα (messages) στους τρεις επόμενους κόμβους, τους function node. Οι function nodes είναι κόμβοι στους οποίους επιτρέπεται η σύνταξη κώδικα σε Javascript για την υλοποίηση οποιασδήποτε λειτουργίας. Όπως γίνεται αντιληπτό και από τα ονόματά τους, ο καθένας “φιλτράρει” τα εισερχόμενα μηνύματα ώστε να κρατηθούν μόνο αυτά της θερμοκρασίας, vrms και power των αισθητήρων που χρησιμοποιούνται αλλά και δημιουργούν τα κατάλληλα queries για την σωστή αποθήκευση αυτών στην τοπική βάση.

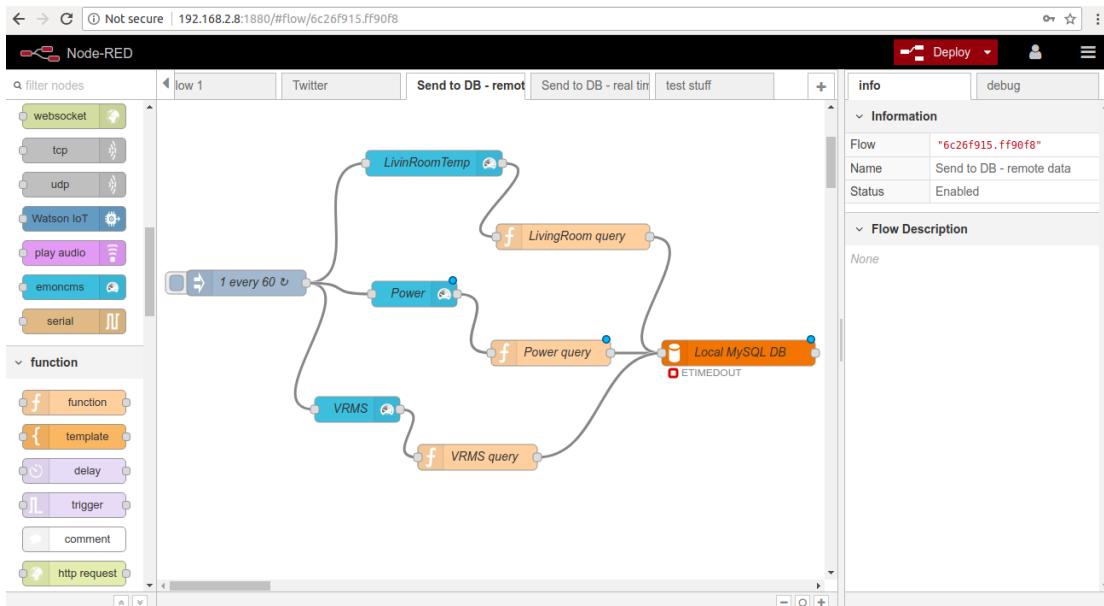
Έπειτα τα μηνύματα περνάνε από τρεις limiter nodes οι οποίοι επιτελούν ίδια

λειτουργία, αυτή της καθυστέρησης των μηνυμάτων κάτα ένα λεπτό ανά μήνυμα. Χρησιμοποιούνται για να διατηρηθεί η βάση δεδομένων σε λογικά μεγέθη.

Τέλος οι μετρήσεις αποστέλονται στην τοπική βάση δεδομένων με τον MySQL node.

5.3.2 Αρχιτεκτονική εφαρμογής *RemoteDB_input*

Στην εικόνα 15 παρουσιάζεται το flow της εφαρμογής αποθήκευσης δεδομένων που συλλέγονται από το remote emonCMS.



Εικόνα 20: Flow της εφαρμογής *RemoteDB_input*

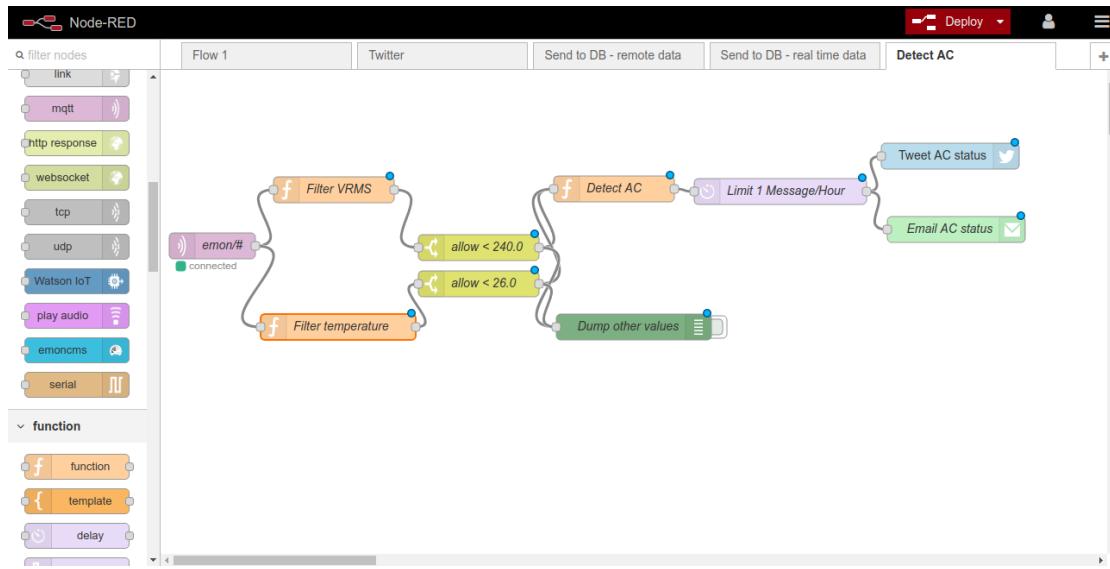
Ο πρώτος από αριστερά κόμβος αποτελεί ένα απλό injection node. Το injection node ενεργοποιεί το flow ανά ένα συγκεκριμένο, ορισμένο από τον χρήστη, χρονικό διάστημα. Στην παρούσα περίπτωση το διάστημα αυτό είναι της τάξης του ενός λεπτού.

Οι τρεις επόμενοι κόμβοι, με μπλε, ονομάζονται “emonCMS In” nodes. Η λειτουργία τους είναι η συλλογή της τελευταίας κάθε φορά τιμής του emonCMS feed (βλ. Κεφάλαιο 6) που τους έχει οριστεί. Χρησιμοποιούνται τρεις κόμβοι, ένας για κάθε υπάρχον feed που αντιστοιχεί σε έναν αισθητήρα. Η συλλογή αυτή γίνεται μία φορά ανά λεπτό, χάρη στο injection node.

Όπως και στην παράγραφο 5.3.1, τα function nodes κατασκευάζουν τα ανάλογα queries για την αποθήκευση των τιμών στα αντίστοιχα tables της βάσης.

5.3.3 Η αρχιτεκτονική της εφαρμογής *Detect_AC*

Στην εικόνα 16 παρουσιάζεται το flow της πειραματικής εφαρμογής ανίχνευσης λειτουργίας του Air-Condition δωματίου και ενημέρωση του χρήστη μέσω tweet και email.



Εικόνα 21: Flow της εφαρμογής Detect AC

Ο πρώτος κόμβος αποτελεί τον MQTT In, η λειτουργία του οποίου γνωστοποιήθηκε στην παράγραφο 5.3.1.

Στη συνέχεια τα topics “φιλτράρονται” με δύο function nodes που επιστρέφουν το msg μόνο αν το topic αυτού είναι emon/emontx3/vrms και emon/emontx3/temp1 αντίστοιχα.

Τα δύο επόμενα node ονομάζονται switch nodes. Τα node αυτά στέλνουν το msg object στο πρώτο destination (από πάνω προς τα κάτω) εάν ικανοποιείται η συνθήκη που φαίνεται στον τίτλο τους αλλιώς το στέλνουν στο δεύτερο destination που στην παρούσα περίπτωση είναι ένα απλό debug node χωρίς καμία λειτουργία.

Στη συνέχεια το function node Detect AC ορίζει ως msg.payload ένα μήνυμα ειδοποίησης πως το κλιματιστικό λειτουργεί και τέλος περνάει στα node που αντίστοιχα κάνουν tweet το μήνυμα και το αποστέλλουν και μέσω email. Για να μην υπάρχει συμφόριση μηνυμάτων σε email και twitter, το limiter node επιτρέπει ένα μήνυμα ανά ώρα.

ΚΕΦΑΛΑΙΟ 6: Περιγραφή της Υλοποίσης

ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΥΛΟΠΟΙΗΣΗΣ

Με βάση την κατανόηση των κεφαλαίων που προηγήθηκαν κρίνεται κατάλληλο να παρατεθεί η αναλυτική υλοποίηση της παρούσας πτυχιακής εργασίας. Το κεφάλαιο αυτό αποτελείται από αρκετά σε αριθμό μέρη καθώς αναφέρεται σε όλα τα στάδια της υλοποίησης, από την εγκατάσταση των συσκευών έως και την τελική εφαρμογή αναγνώρισης λειτουργίας Air-Condition.

6.1 Εγκατάσταση του συστήματος energy monitoring (emonPi, emonTx & sensors)

Το επόμενο βήμα από την απόκτηση των συσκευών που θα χρησιμοποιηθούν είναι φυσικά η εγκατάστασή τους στον χώρο. Η εγκατάσταση αυτή αποτελεί μία ιδιαίτερα επικίνδυνη πράξη που απαιτεί προσοχή και κατάλληλη προετοιμασία του χώρου αλλά και του χρήστη. Με τον όρο προετοιμασία εννοούνται:

- Κλείσιμο του γενικού διακόπτη του χώρου – διακοπή της παροχής ενέργειας
- Προμήθευση και χρήση μονοτικών (πλαστικών) γαντίων για την προστασία του διαχειριστή
- Αφαίρεση της προστατευτικής επιφάνειας του ηλεκτρικού πίνακα του χώρου εγκατάστασης



Εικόνα 22: Εγκατάσταση emonTx & αισθητήρων

Στην εικόνα 17 παρουσιάζεται εγκατεστημένη η συσκευή emonTx με τρεις αισθητήρες: στο καλώδιο κεντρικής παροχής ρεύματος του ηλεκτρικού πίνακα καταλήγει ο CT sensor ενώ με άσπρο και μαύρο καλώδιο φαίνονται οι αισθητήρες temperature και vrms αντίστοιχα. Μετά την επαναφορά του ρεύματος στον χώρο, αναμένεται η ένδειξη transmittion του emonTx, από το χαρακτηριστικό led που διαθέτει.



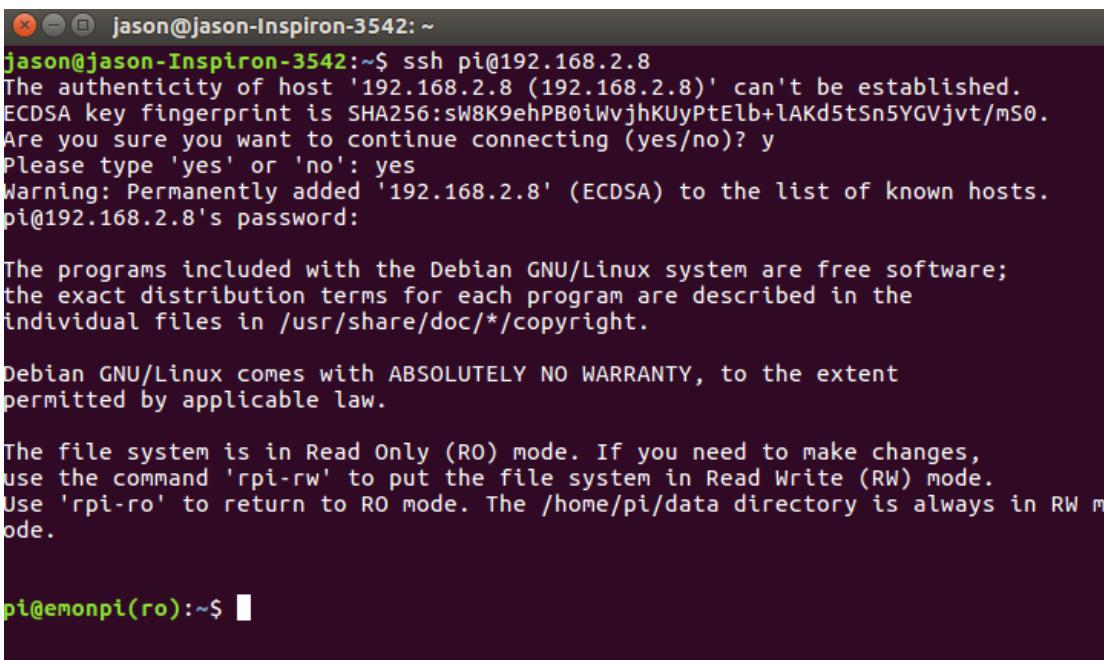
Εικόνα 23: Εγκατάσταση emonPi ως gateway

Στην εικόνα 18 φαίνεται η εγκατάσταση της συσκευής emonPi για χρήση αποκλειστικά ως gateway. Η εγκατάσταση αυτή τη φορά είναι αρκετά απλή με μοναδικό απαραίτητο το καλώδιο Ethernet και την σύνδεση αυτού σε μία LAN θύρα του router. Η led οθόνη του emonPi έπειτα παρουσιάζει την κατάσταση σύνδεσης, δηλαδή το αν υπάρχει συνδεδεμένο Ethernet και ποιοι αισθητήρες (και οι τιμές τους) βρίσκονται συνδεδεμένοι στο emonPi (στην παρούσα περίπτωση κανένας). Παρουσιάζει επίσης την διεύθυνση ip μέσω της οποίας υπάρχει πρόσβαση στον τοπικό server του emonPi.

Για την σωστή λειτουργία του συστήματος είναι απαραίτητο να γίνει factory reset του emonPi καθώς είναι προκαθορισμένη η δυνατότητα ύπαρξης μόνο ενός administrative account σε αυτό την φορά. Για την επαναφορά ρυθμίσεων καθώς και για πολλές άλλες λειτουργίες του emonPi απαιτείται η σύνδεση σε αυτό από προσωπικό υπολογιστή που βρίσκεται στο δίκτυο. Η σύνδεση επιτυγχάνεται μέσω του πρωτοκόλλου SSH (Secure Shell). Το SSH είναι ένα ασφαλές δικτυακό πρωτόκολλο το οποίο επιτρέπει τη μεταφορά δεδομένων μεταξύ δύο υπολογιστών.

Το SSH στο emonPi “τρέχει” στο port 22 και για την σύνδεση μέσω αυτού αρκεί η εντολή σε linux terminal “ssh pi@<IP ADDRESS>”, όπου <IP ADDRESS> η διεύθυνση ip που εμφανίζεται στην led οθόνη του emonPi. Τα credentials (username, password) που απαιτούνται για την σύνδεση, δίνονται από τον οργανισμό

OpenEnergyMonitor [16]. Μετά την σύνδεση, το factory reset επιτυγχάνεται με σύνδεση ως root και την εντολή `~/emonpi./factoryreset` για την εκτέλεση του script `factoryreset` που βρίσκεται στον φάκελο `emonpi`.



```
jason@jason-Inspiron-3542: ~$ ssh pi@192.168.2.8
The authenticity of host '192.168.2.8 (192.168.2.8)' can't be established.
ECDSA key fingerprint is SHA256:sW8K9ehPB0iWvjhKUyPtElb+lAKd5tSn5YGVjvt/mS0.
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' or 'no': yes
Warning: Permanently added '192.168.2.8' (ECDSA) to the list of known hosts.

pi@192.168.2.8's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

The file system is in Read Only (RO) mode. If you need to make changes,
use the command 'rpi-rw' to put the file system in Read Write (RW) mode.
Use 'rpi-ro' to return to RO mode. The /home/pi/data directory is always in RW m
ode.

pi@emonpi(ro):~$
```

Εικόνα 24: SSH σύνδεση στο emonPi

Το σύστημα είναι πλέον έτοιμο για την δημιουργία νέου administrative account και για το configuration τοπικού και απομακρυσμένου server.

6.2 Local & Remote logging μετρήσεων

Το επόμενο στάδιο μετά την εγκατάσταση του συστήματος είναι το logging δεδομένων στον τοπικό server του emonPi αλλά και στον απομακρυσμένο emoncms.org. Η διαδικασία που ακολουθείται για τα configuration αυτά είναι η εξής:

6.2.1 Local data logging

Κατά την εγκατάσταση των συσκευών στον χώρο, αποτελεί καλή τεχνική η σύνδεση της τροφοδοσίας οποιασδήποτε συσκευής αφού έχουν πρώτα συνδεθεί σε αυτές οι διάφοροι αισθητήρες. Θεωρώντας πως η ακολουθείται η τεχνική αυτή, οι μετρήσεις που λαμβάνονται είναι ορατές από την πρώτη στιγμή στον τοπικό server, υπό την επιλογή `Setup > Inputs`

Node	Key	Name	Process list	Updated	Value
emonpi	power1			3s	0
emonpi	power2			3s	0
emonpi	power1pluspower2			3s	0
emonpi	vrms			3s	0
emonpi	t1			3s	0
emonpi	t2			3s	0
emonpi	t3			3s	0
emonpi	t4			3s	0
emonpi	t5			3s	0
emonpi	t6			3s	0
emonpi	pulsecount			3s	0
emonpi	rssi			3s	0

Εικόνα 25: Local emonCMS server

Η πρόσβαση στον τοπικό server επιτυγχάνεται πληκτρολογώντας στον browser ως URL τη διεύθυνση IP που εμφανίζεται στην Led οθόνη του emonPi και δημιουργώντας administrative account, αφού έχει προηγηθεί factory reset της συσκευής, όπως προαναφέρθηκε.

Σημειώνεται πως οι μετρήσεις που παρουσιάζονται τοπικά δεν αποθηκεύονται πουθενά και παραμένουν ορατές μόνο για όσο χρόνικο διάστημα παραμένουν συνδεδεμένες στο δίκτυο οι συσκευές και οι αισθητήρες τους. Η αποθήκευσή τους είναι δυνατή στην SD card του Raspberry Pi με διαδικασία όμοια αυτής που παρουσιάζεται στην παράγραφο 6.2.2, μέχρι την εξάντληση του αποθηκευτικού χώρου.

Όπως φαίνεται στην εικόνα 25, οι μετρήσεις κάθε αισθητήρα εμφανίζονται στο κατάλληλο πλαίσιο με το όνομα αυτού. Παρουσιάζονται επίσης default σταθερές τιμές σε αρκετές σειρές. Οι σειρές αυτές αντιστοιχούν σε μη-κατειλημμένες θύρες αισθητήρων, εν δυνάμει νέους. Για παράδειγμα, όπως έχει αναφερθεί, το emonTx μπορεί να υποστηρίξει περισσότερους από έναν αισθητήρες θερμοκρασίας όμως χρησιμοποιήθηκε ένας για τις ανάγκες της εργασίας. Οι τιμές αυτού φαίνονται στην σειρά temperature1 ενώ οι υπόλοιπες σειρές temperature έχουν default τιμή 300. Οι μετρήσεις που απασχολούν την παρούσα εργασία είναι αυτές των power1, vrms, temperature1 για την παροχή ρεύματος, vrms και θερμοκρασία αντίστοιχα, ενώ πραγματική τιμή αποτελεί και αυτή του rss (Received Signal Strength Indicator).

6.2.2 Remote data logging

Για το logging τιμών στον remote emoncms server είναι απαραίτητη αρχικά η δημιουργία λογαριασμού στο emoncms.org. Με την δημιουργία νέου λογαριασμού, αυτόματα δίνονται στον χρήστη δύο API keys τα οποία βρίσκονται υπό την επιλογή Setup > Inputs > Input API Help. Η επιλογή αυτή φαίνεται στην εικόνα 26, τα keys της οποίας έχουν καλυφθεί για ευνόητους λόγους.

Input API

EIKONA 26: emoncms API keys

Το επόμενο βήμα απαιτεί την χρήση του Read & Write API key: αφού αυτό αντιγραφεί, είναι απαραίτητη η επικόλλησή του στο configuration file του emonHub (python service του emonPi, υπεύθυνο για το logging μετρήσεων, εικόνα 16). Αυτό είναι δυνατό να επιτευχθεί με δύο τρόπους: μεταβιβάση στο local emoncms, Setup > emonHub > επικόλληση του API key στο κατάλληλο σημείο του section [[emoncmsorg]] ή εύρεση του ίδιου configuration file στο emonPi με σύνδεση σε αυτό μέσω SSH και τις εντολές terminal

- \$ cd emonHub/conf , για browse στον κατάλληλο φάκελο
- \$ rpi-rw , για απόκτηση άδειας read/write στα αρχεία
- \$ nano emonhub.conf , για άνοιγμα και επεξεργασία του αρχείου emonhub.conf και επικόλληση του API key στο αντίστοιχο σημείο (ΕΙΚΟΝΑ 659).

```

pubchannels = ToRFM12,
subchannels = ToEmonCMS,
# emonhub/rx/10/values format
# Use with emoncms Nodes module
node_format_enable = 1
node_format_basetopic = emonhub/
# emon/emonbt/power1 format - use with Emoncms MQTT input
# http://github.com/emoncms/emoncms/blob/master/docs/RaspberryPi/MQTT.md
nodevar_format_enable = 1
nodevar_format_basetopic = emon/
[emoncmsorg]
Type = EmonHubEmoncmsHTTPInterface
[[init_settings]]
[[runtimesettings]]
pubchannels = ToRFM12,
subchannels = ToEmonCMS,
url = https://emoncms.org
apikey = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx2
senddata = 1 # Enable sending data to Emoncms.org
sendstatus = 1 # Enable sending WAN IP to Emoncms.org MyIP > https://emoncms.org/myip/list
sendinterval= 30 # Bulk send interval to Emocms.org in seconds
#####
##### Nodes #####
#####

[nodes]

```

EIKONA 27: Χρήση read/write API key στο local emoncms

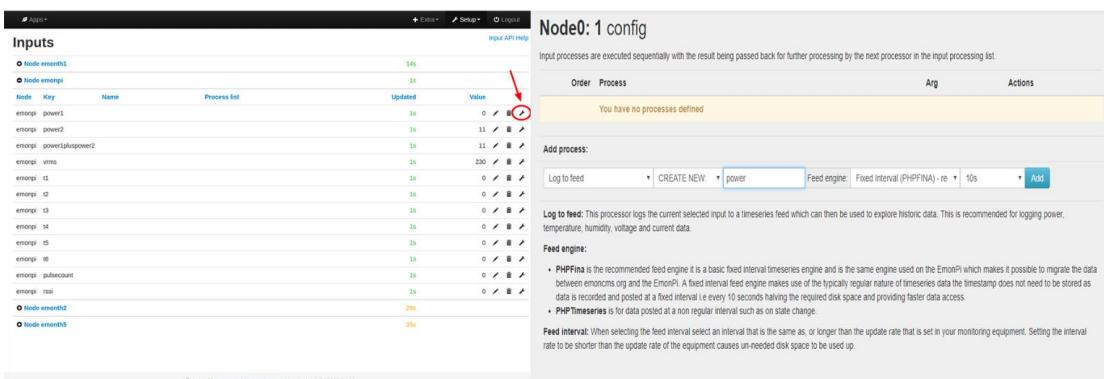
Τα ίδια inputs που φαίνονται στον local server θα εμφανίζονται πλέον και στο remote emoncms κάτω από την αντίστοιχη επιλογή, με μία μικρή καθυστέρηση σε σχέση με τον χρόνο ανανέωσης των τιμών του local server.

Ωστόσο οι τιμές αυτές επίσης δεν αποθηκεύονται πουθενά. Ακολουθεί η διαδικασία των κατάλληλων ρυθμίσεων προκειμένου να υπάρχει πρόσβαση σε ιστορικό μετρήσεων.

6.2.2.1 Log to Feed

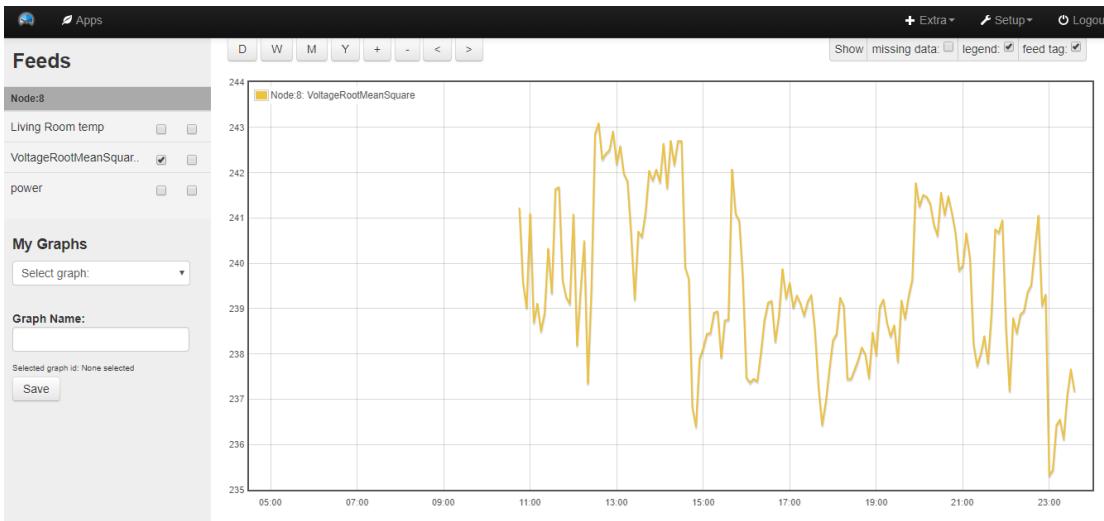
Η διαδικασία αυτή ονομάζεται Log to Feed και η χρήση της, πέρα από διατήρηση ιστορικού δεδομένων, προσφέρει και διαγραμματική απεικόνιση αυτών στο web app του emoncms.

Από την σελίδα των inputs επιλέγεται το εικονίδιο Input Config Spanner στα input που είναι επιθυμιτό να αποθηκεύονται. Στο αναδυόμενο παράθυρο, για την συγκεκριμένη λειτουργία προτείνεται όλες οι επιλογές πλην του ονόματος του Feed να διατηρήσουν τις default τιμές τους. Προσφαίρονται πολυάριθμες ακόμα επιλογές, οι λειτουργίες των οποίων δεν απασχολούν την παρούσα εργασία. Στην εικόνα 28 παρουσιάζονται τα δύο βήματα που προαναφέρθηκαν για την δημιουργία του feed power, που αποθηκεύει τις τιμές παροχής ρεύματος.



EIKONA 28: Η διαδικασία Log to Feed

Πλέον τα Feed έχουν δημιουργηθεί και είναι δυνατή ανάκτηση ιστορικού μετρήσεων σε κλίμακα ημέρας, εβδομάδος, μήνα και χρόνου καθώς και η γραφική αναπαράσταση αυτών. Στην εικόνα 29 φαίνεται η γραφική αναπαράσταση των τιμών (άξονας Y) vrms για τις αναγραφόμενες ώρες (άξονας X) μίας συγκεκριμένης ημέρας στον χώρο που βρίσκεται εγκατεστημένο το σύστημα.



EIKONA 29: Γραφική αναπαράσταση τιμών vrms

Η πρόσβαση στα Feed καθίσταται εφικτή με την επιλογή Setup > Feeds ενώ η εμφάνιση γραφικών αναπαραστάσεων με την επιλογή Setup > Feeds > Eye Icon. Τα Feed αυτά χρησιμοποιήθηκαν και για την δημιουργία της εφαρμογής Node-Red RemoteDB_input.

6.3 Δημιουργία της Βάσης Δεδομένων

Η αρχιτεκτονική της βάσης δεδομένων παρουσιάζεται στην ΕΙΚΟΝΑ ΒΔ, στην παράγραφο 5.2. Όπως έχει αναφερθεί, η βάση δεδομένων MySQL εγκαθιστάται στον προσωπικό υπολογιστή ως μέρος του software bundle του LAMP, στο port 3306. Κρίνεται σημαντικό να αναφερθεί πως γενικότερα, η εγκατάσταση MySQL server σε έναν υπολογιστή, είτε ως μέρος του LAMP είτε αυτόνομα, οδηγεί σε “κατάληψη” του port 3306.

Στην συχνή περίπτωση όπου υπάρχει ήδη εγκατεστημένος MySQL server και πραγματοποιείται εκ των υστέρων εγκατάσταση του LAMP, παρουσιάζεται πρόβλημα λόγω αδυναμίας της χρήσης του port 3306 από τον MySQL server του LAMP. Αν ο χρήστης δεν επιθυμεί την απεγκατάσταση του αυτόνομου MySQL server, όπως στην περίπτωση αυτής της εργασίας, η λύση στο πρόβλημα επέρχεται ανιχνεύοντας το PID και τερματίζοντας την υπάρχουνσα διεργασία mysql, πριν την ενεργοποίηση του LAMP. Η εύρεση του PID αυτής καθίσταται δυνατή με μία εκ των εντολών terminal:

- netstat -lp | grep 3306
- lsof -i TCP: 3306

για την εκτέλεση των οποίων είναι απαραίτητη η σύνδεση ως root. Ο τερματισμός της διεργασίας επιτυγχάνεται με την “kill <PID>”, όπου <PID> το PID που λαμβάνεται από τις παραπάνω εντολές. Στην εικόνα 30 παρουσιάζεται η ολοκληρωμένη διαδικασία “απελευθέρωσης” του port 3306 για την ομαλή εκκίνηση του LAMP.

```
root@jason-Inspiron-3542: /home/jason
[jason@jason-Inspiron-3542:~$ sudo su
[sudo] password for jason:
root@jason-Inspiron-3542:/home/jason# lsof -i TCP:3306
COMMAND   PID  USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
mysqld  10616 mysql  19u   IPv6 120713      0t0  TCP *:mysql (LISTEN)
root@jason-Inspiron-3542:/home/jason# kill 10616
```

Εικόνα 30: Απελευθέρωση του port 3306

Παρουσιάζεται ο κώδικας SQL της δημιουργίας των table της βάσης δεδομένων. Όπως είδαμε στην παράγραφο 5.2, τα πεδία των tables παρουσιάζουν αρκετές ομοιότητες. Για το λόγο αυτό παρόμοιες επεξηγήσεις σε διαφορετικά table δεν επαναλαμβάνονται.

6.3.1 *LivingRoomTemp table*

```
-- Database: `testDB`
-----
-- Table structure for table 'LivingRoomTemp'

CREATE TABLE `LivingRoomTemp` (
  `id` int(11) NOT NULL,
  `topic` text NOT NULL,
  `payload` float(10,1) NOT NULL,
  `scale` varchar(7) NOT NULL DEFAULT 'celsius',
  `timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON
  UPDATE CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Αρχικά παραθέτονται οι ονομασίες των πεδίων του table και οι τύποι αυτών με πεζά γράμματα. Στο int, η τιμή 11 αντιπροσωπεύει το πλήθος ψηφίων του αριθμού που θα εμφανίζονται. Δώθηκε το 11 ως μεγάλο νούμερο, δεδομένου ότι θα καλύπτει πάντα τις ανάγκες αυτής της εργασίας χωρίς να αποκοπεί ποτέ κάποιο ψηφίο.

Σχετικά με τον τύπο float, ένας ορισμός column ως float(M,D) θα επιτρέπει αποθήκευση τιμών μεγέθους έως και M ψηφία ενώ από αυτά, μόνο D θα εμφανίζονται μετά την υποδιαστολή, άν υπάρχει. Οι μετρήσεις θερμοκρασίας από default συλλέγονται με ακρίβεια ενός δεδαδικού ψηφίου, οπότε οι τιμές (10,1) καλύπτουν τις ανάγκες της παρούσας εργασίας.

Όσο αναφορά τον τύπο varchar του πεδίου scale, η παράμετρος 7 καθορίζει απλά το μέγιστο επιτρεπόμενο μήκος της συμβολοσειράς.

Σε όλα τα columns παρατηρείται ο περιορισμός NOT NULL. Όπως είναι κατανοητό από την ονομασία του, ο περιορισμός αυτός αποσκοπεί στο να μην γίνονται αποδεκτές

NULL τιμές σε κανένα από τα πεδία.

Τέλος, τα πεδία scale και timestamp έχουν ως default τιμές “celsius” και CURRENT_TIMESTAMP αντίστοιχα. Η συνάρτηση CURRENT_TIMESTAMP επιστρέφει την ημερομηνία και ώρα κάθε στιγμή. Ο περιορισμός ON UPDATE CURRENT_TIMESTAMP καταγράφει στην στήλη timestamp την ημερομηνία & ώρα σε κάθε νέα καταχώρηση στην βάση.

```
--  
--Indexes for table 'LivingRoomTemp'  
  
--  
ALTER TABLE `LivingRoomTemp`  
ADD PRIMARY KEY (`id`);  
  
--  
-- AUTO_INCREMENT for table 'LivingRoomTemp'  
  
--  
ALTER TABLE `LivingRoomTemp`  
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
```

Το πεδίο id ορίζεται ως primary key και αποτελεί unique identifier κάθε καταχώρησης. Έπειτα του αποδίδεται το χαρακτηριστικό autoincrement, για αυτόματη αύξηση κατά ένα αυτού, με κάθε νέα καταχώρηση.

6.3.2 VRMS table

```
--  
-- Table structure for table 'VRMS'  
  
--  
CREATE TABLE `VRMS` (  
  `id` int(11) NOT NULL,  
  `topic` varchar(4) NOT NULL DEFAULT 'VRMS',  
  `payload` float(10,1) NOT NULL,  
  `timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON  
  UPDATE CURRENT_TIMESTAMP  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;  
  
--  
-- Indexes for table 'VRMS'  
  
--  
ALTER TABLE `VRMS`  
ADD PRIMARY KEY (`id`);  
  
--  
-- AUTO_INCREMENT for table 'VRMS'  
  
--  
ALTER TABLE `VRMS`  
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
```

Τα πεδία των δύο table παρουσίαζουν πολλές ομοιότητες. Ως εκ τούτου, η επεξήγηση κώδικα της παραγράφου 6.3.1 επαρκεί πλήρως και για την κατανόηση του παρόντος table.

6.3.3 Power table

```
-- Table structure for table 'Power'

CREATE TABLE `Power` (
  `id` int(11) NOT NULL,
  `topic` text NOT NULL,
  `payload` float(10,1) NOT NULL,
  `scale` varchar(7) NOT NULL DEFAULT 'ampere',
  `timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-----
--Indexes for table 'Power'
--
ALTER TABLE `Power`
ADD PRIMARY KEY (`id`);

--
-- AUTO_INCREMENT for table 'Power'
--
ALTER TABLE `Power`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
```

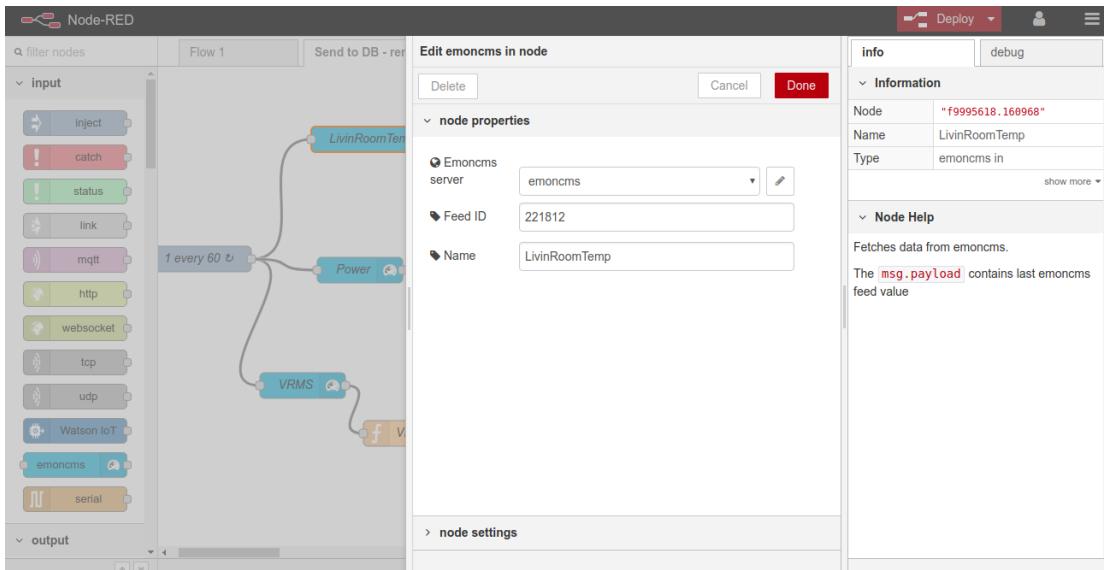
Τα πεδία των τριών table παρουσίαζουν πολλές ομοιότητες. Ως εκ τούτου, η επεξήγηση κώδικα της παραγράφου 6.3.1 επαρκεί πλήρως και για την κατανόηση του παρόντος table.

6.4 Δημιουργία εφαρμογής Node-Red RemoteDB_input

Στην παράγραφο αυτή παρουσιάζεται αναλυτικότερα η υλοποίηση της εφαρμογής Node-Red “RemoteDB_input”, η οποία συλλέγει δεδομένα από τον remote emoncms.org server, τα επεξεργάζεται και τα καταχωρεί στην τοπική βάση δεδομένων. Η αρχιτεκτονική, το flow, της εφαρμογής αυτής έχει ήδη παρουσιαστεί στην παράγραφο 5.3.2 .

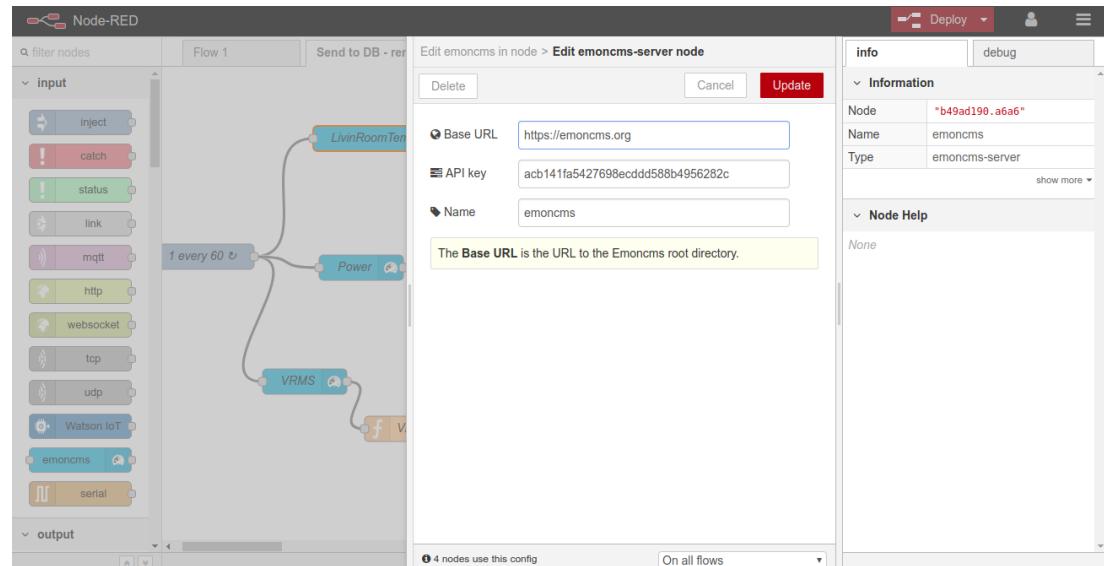
Όπως έχει προαναφερθεί, για την λειτουργία της εφαρμογής είναι απαραίτητη η ύπαρξη feeds (παράγραφος 6.2.2.1) στον remote server. Η ύπαρξή τους είναι απαραίτητη διότι τα “emonCMS In” nodes απαιτούν στοιχεία αυτών για το configuration τους.

Με την δημιουργία κάθε feed, ανατείθεται σε αυτό ένα Feed ID. Το ID του κάθε feed αποτελεί το κυριότερο στοιχείο σύμφωνα με το οποίο το node αναγνωρίζει από που να ανακτήσει δεδομένα.



EIKONA 31: Πρώτο επίπεδο configuration του emoncms In node

Η ολοκλήρωση του configuration ενός emonCMS In node επέρχεται με την επιλογή του edit, δίπλα από την ονομασία του Emoncms server. Ως Base URL δίνεται το URL που οδηγεί στο root directory του emoncms, το οποίο είναι και το κύριο url της ιστοσελίδας. Το API key που ζητείται στη συνέχεια, είναι το read only key, καθώς η εφαρμογή δεν μπορεί να επηρεάσει τα δεδομένα που βρίσκονται στον server, το οποίο δίνεται με την δημιουργία λογαριασμού στην ιστοσελίδα του emonCMS, στο Setup > Inputs > Input API help. Τέλος, η επιλογή Name tag είναι προεραυτική και καθορίζει μόνο τον τρόπο εμφάνισης της ονομασίας του server, στην πρώτη σελίδα του configuration.



EIKONA 32: Δεύτερο επίπεδο configuration του emoncms In node

Το configuration των επόμενων δύο emonCMS In nodes είναι ακριβώς ίδιο με μόνη διαφορά αυτή του Feed ID.

Σημειώνεται πως οποιαδήποτε πληροφορία σε μία εφαρμογή Node-Red μεταβιβάζεται

από node σε node ως ένα Javascript object, το msg. Το msg object έχει πολυάριθμα attributes τα σημαντικότερα των οποίων είναι τα payload, περιέχει το σώμα του μηνύματος, και topic. Στην περίπτωση του emonCMS In node, το msg.payload περιέχει την τελευταία καταχώρηση του συγκεκριμένου feed με το οποίο λειτουργεί.

Από τα emonCMS In nodes, οι μετρήσεις περνάνε σε τρεις function nodes, ένα για κάθε emoncms node, τα οποία είναι υπεύθυνα για την δημιουργία του κατάλληλου κάθε φορά query που θα αποθηκεύσει τα δεδομένα στη βάση. Το τελικό MySQL node από default απαιτεί τα queries να περιέχονται στο msg.topic του μηνύματος που αυτό δέχεται. Ο σύντομος κώδικας JavaScript που συντάχθηκε για αυτές τις λειτουργίες είναι:

```
//LivingRoom query node
var string = "livingRoom";
msg.topic = "INSERT INTO LivingRoomTemp (topic, payload ) VALUES (" + string+
",",
msg.payload+ ")";
return msg;
```

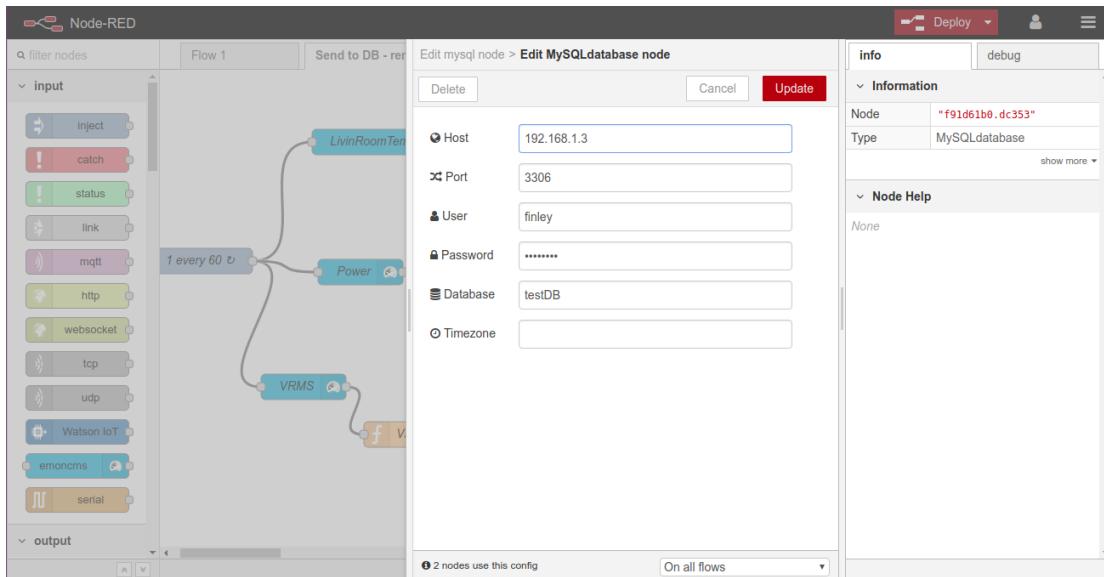
Καθώς το msg.topic είναι string, η τιμή του συντάσσεται αναλόγως. Το απλό αυτό query, καθορίζει την εισαγωγή, συγκεκριμένα στα columns topic και payload του table LivingRoomTemp, του string που περιέχει το “θέμα” της καταχώρησης, livingRoom και του msg.payload, που αποτελεί την τιμή της μέτρησης που ανακτήθηκε. Τέλος, επιστρέφει ολόκληρο το msg object προκειμένου να περάσει, σύμφωνα με το flow, στο MySQL node.

Όμοιος είναι και ο κώδικας των επόμενων δύο function nodes, “Power query” και “VRMS query” που παρατείθεται στη συνέχεια. Η επεξήγηση του κώδικα που προηγήθηκε καθίσταται επαρκής για την κατανόηση των υπολοίπων.

```
//Power query node
var string = "power";
msg.topic = "INSERT INTO Power (topic, payload ) VALUES (" + string+ ",",
msg.payload+ ")";
return msg;
//VRMS query node
msg.topic = "INSERT INTO VRMS (payload ) VALUES ("+msg.payload+ ")";
return msg;
```

Τέλος, το configuration του MySQL node απαιτεί το internal ip του localhost, που ανακτείται με την εντολή σε terminal \$ ifconfig -a, το port στο οποίο “μιλάει” η βάση δεδομένων MySQL (στην περίπτωσή μας 3306), το όνομα της βάσης και τα credentials ενός χρήστη της βάσης με πλήρη δικαιώματα, ισάξια του root. Για την παρούσα εργασία, ένας τέτοιος χρήστης δημιουργήθηκε με τη χρήση του phpMyAdmin, με το όνομα Finley.

Με την επιλογή deploy, η εφαρμογή ξεκινά την λειτουργία της.

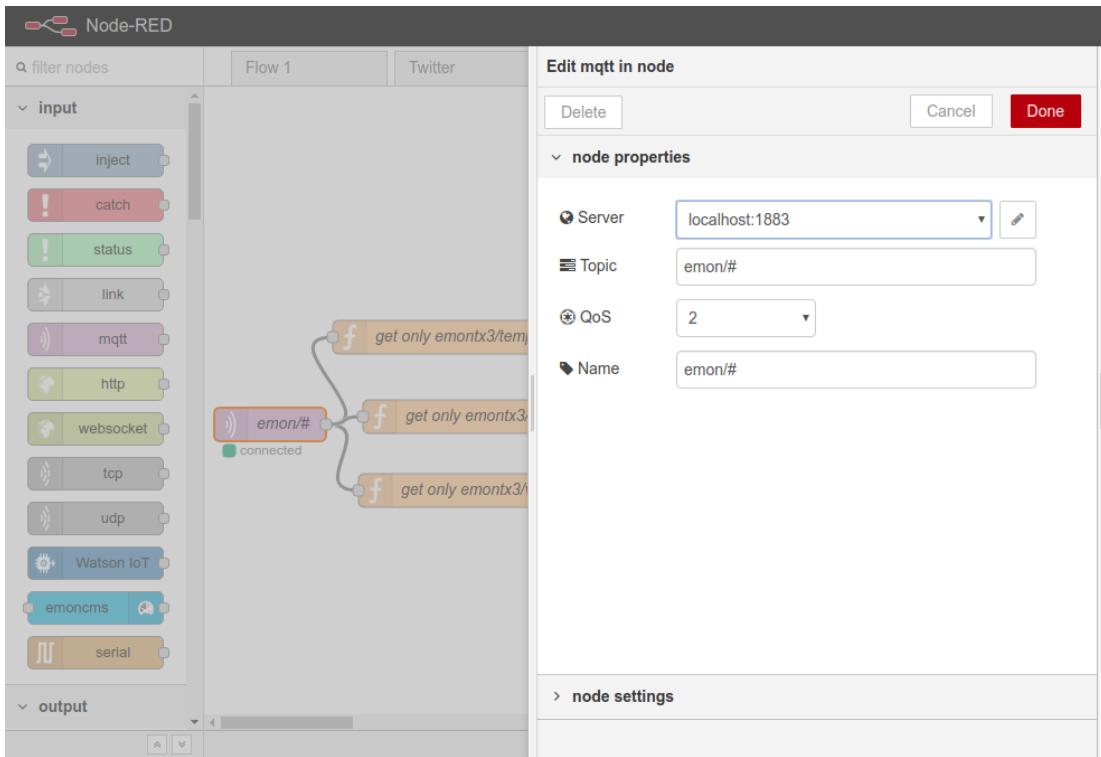


ΕΙΚΟΝΑ 33: MySQL node configuration

6.5 Δημιουργία της εφαρμογής Real-TimeDB_input

Παρουσιάζεται αναλυτικότερα η υλοποίηση της εφαρμογής Node-Red “Real-TimeDB_input”, η οποία συλλέγει δεδομένα σε πραγματικό χρόνο, τα επεξεργάζεται και τα καταχωρεί στην τοπική βάση δεδομένων. Η αρχιτεκτονική, το flow, της εφαρμογής αυτής έχει ήδη παρουσιαστεί στην παράγραφο 5.3.1.

Όπως φαίνεται στην εικόνα 19, αρχικά πραγματοποιείται η σύνδεση στον mqtt broker και το subscription στα κατάλληλα topics αυτού, με την χρήση ενός MQTT In node.



ΕΙΚΟΝΑ 34: MQTT In node configuration

Οι απαιτήσεις για το configuration αυτού δεν είναι ιδιαίτερα πολύπλοκες και περιλαμβάνουν: αρχικα τις πληροφορίες σύνδεσης του server όπου είναι εγκατεστημένος ο broker (στο emonPi “τρέχει” στο port 1883), το topic στο οποίο γίνεται subscribe και το επίπεδο QoS (Quality of Service). Η σημασία του topic που φαίνεται στην ΕΙΚΟΝΑ 8 έχει ήδη επεξηγηθεί στην παράγραφο 5.3.1 ενώ το επίπεδο QoS 2 εξασφαλίζει ότι κάθε μήνυμα θα παραληφθεί ακριβώς μία φορά. Το επίπεδο 2 μπορεί να είναι το αργότερο επίπεδο επικοινωνίας μεταξύ mqtt client και broker είναι όμως και το πιο ασφαλές. Κρίθηκε κατάλληλη η χρήση αυτού καθώς κάποιο duplicate message που πιθανόν να περνούσε με χρήση κατώτερων επιπέδων, θα έβλαπτε την ορθότητα αποτελεσμάτων της εφαρμογής.

Στη συνέχεια οι τρεις function nodes διασφαλίζουν το φιλτράρισμα μόνο των topics που είναι απαραίτητα για την παρούσα εργασία αλλά και κατασκευάζουν τα κατάλληλα queries για την εισαγωγή των μετρήσεων στη βάση, όπως και στην παράγραφο 6.4. Ο σύντομος κώδικας JavaScript που χρησιμοποιείται παρουσίαζει, όπως είναι φυσικό, ομοιότητες από node σε node. Για το λόγο αυτό, η αναλυτική επεξήγηση του κώδικα του πρώτου σε σειρά node καθίσταται επαρκής και για την πλήρη κατανόηση των υπολοίπων.

```
//get only emontx3/temp1 values
var room = "livingRoom";
var substring = "emon/emontx3/temp1";
if(msg.topic.includes(substring))
{
    msg.topic = "INSERT INTO LivingRoomTemp (topic, payload ) VALUES (" +

```

```
room+ "," +      msg.payload+ ");"
    return msg;
}
```

To string “substring” που δεσμεύεται στην δεύτερη σειρά του κώδικα αποτελέι το topic του αισθητήρα θερμοκρασίας. Ελέγχεται αν το string αυτό περιέχεται στο topic του μηνύματος που μεταδίδεται με την συνάρτηση includes() και αφού βρεθεί, κατασκευάζεται το κατάλληλο query, όμοιο με αυτά τις παραγράφου 6.4. Τέλος, ολόκληρο το msg object επιστρέφεται. Πατείθεται ο κώδικας των επόμενων δύο function nodes.

```
//get only emontx3/power1 values
var substring = "emon/emontx3/power1";
if(msg.topic.includes(substring))
{
    msg.topic = "INSERT INTO Power ( payload ) VALUES (" + msg.payload + ")";
    return msg;
}
//get only emontx3/vrms values
var substring = "emon/emontx3/vrms";
if(msg.topic.includes(substring))
{
    msg.topic = msg.topic = "INSERT INTO VRMS (payload) VALUES (" + msg.payload + ")";
    return msg;
}
```

Τα μηνύματα στην συνέχεια καθυστερούν τρεις limiter nodes ρυθμισμένοι να επιτρέπουν “πέρασμα” από αυτούς μόνο ενός μηνυύματος ανά λεπτό. Τα ενδιάμεσα μηνύματα απορρίπτονται (dropped). Η χρήση τους αποσκοπεί στην ελεγχόμενη αύξηση μεγέθους της τοπικής βάσης δεδομένων. Τέλος, καταλήγουν σε ένα MySQL node ο οποίος ακολουθεί ακριβώς ίδιο configuration με αυτόν της παραγράφου 6.4.

6.6 Κατασκεύη API

Για την κατασκευή του API για ανάκτηση των μετρήσεων από την τοπική βάση δεδομένων χρησιμοποιήθηκε το Flask web framework. Το Flask είναι ένα microframework (δεν απαιτεί κάποια συγκεκριμένα εργαλεία) γραμμένο σε Python και σχεδιασμένο για να υποστηρίζει την κατασκευή web applications, web services αλλά και web APIs. Συνεπώς, το API που δημιουργήθηκε στα πλαίσια αυτής της εργασίας είναι επίσης γραμμένο σε Python και κάνει χρήση της βιβλιοθήκης Flask καθώς και των component αυτής jsonify και request. Λειτουργώντας τοπικά, όπως στην περίπτωση της παρούσας εργασίας, το Flask “τρέχει” στο port 5000 του localhost.

Συνεπώς κάθε URL ξεκινάει με `localhost:5000/`.

Καθίσταται επίσης απαραίτητη η εγκατάσταση και χρήση της βιβλιοθήκης MySQLdb της Python. Επισημαίνεται πως η βιβλιοθήκη αυτή δεν υποστηρίζεται στην έκδοση 3.6 της Python τη στιγμή συγγραφής αυτής της εργασίας, παρά μόνο στην 2.7. Ωστόσο το API δημιουργήθηκε σε Python 3.6 καθώς η ρίζα του προβλήματος αυτού βρισκόταν στην μη ύπαρξη σημαντικών στοιχείων (dependencies) στην έκδοση 3.6 τα οποία εγκαταστάθηκαν με τις εντολές σε terminal `$sudo apt-get install python-dev libmysqlclient-dev` ακολουθούμενες από την εντολή εγκατάστασης της βιβλιοθήκης `$pip install mysqlclient`, δεδομένου ότι το `python-pip` είναι ήδη εγκατεστημένο.

Κατά τον σχεδιασμό του API αποφασίστηκε πως το είδος request που θα γίνονται στην τοπική βάση θα είναι μόνο GET requests καθώς και ότι το format στο οποίο επιστρέφονται τα δεδομένα θα είναι JSON. Συμφωνήθηκε επίσης η χρήση του Flask, όπως προαναφέρθηκε, γεγονός που οδηγεί την μορφή του API σε αυτή του REST (Representational State Transfer). Οι αποφάσεις αυτές καθόρισαν πλήρως την αρχιτεκτονική κατασκευής του API η οποία θα μπορούσε να είναι τελείως διαφορετική χρησιμοποιώντας την επίσης διαδεδομένη αρχιτεκτονική SOAP (Simple Object Access Protocol) ή/και το format δεδομένων XML. Ο κυριότερος λόγος για τον οποίο προτιμήθηκε REST έναντι της τεχνικής SOAP είναι πως η αρχιτεκτονική REST δεν μοχλεύει τόσο μεγάλο bandwidth όσο αυτή του SOAP, γεγονός που την κάνει καταληλότερη για χρήση στο Internet.

Όσο αναφορά τους λόγους προτίμησης JSON format έναντι του XML αυτοί είναι κυρίως πως

- το JSON είναι μικρότερο σε μέγεθος
- Συντομότερο στην σύνταξη/συγγραφή και ευκολότερο στην ανάγνωση και κατανόηση
- το JSON μπορεί να χρησιμοποιεί arrays

Τέλος, αξίζει να σημειωθεί πως η αναφορά στο SOAP ως αρχιτεκτονική δεν είναι απολύτως ακριβής καθώς, όπως φαίνεται και στην ολογράφως ονομασία του, το SOAP αποτελεί πρωτόκολλο. Συνεπώς η σύγκρισή τους καθίσταται αρκετά πολύπλοκη και δεν απασχολεί τα πλαίσια αυτής της εργασίας. Η χρήση του πρωτοκόλλου αυτού ωστόσο απαιτεί διαφορετική αρχιτεκτονική API από αυτή του REST εξους και ο λόγος που πραγματοποιήθηκε η προαναφερθείσα σύγκριση.

Το API που κατασκευάστηκε αποτελείται από έξι (6) διαφορετικές κλήσεις. Η επέκταση αυτού αποτελεί εργασία όχι ιδιαίτερης δυσκολίας και αφίνεται ως μελλοντικός στόχος αυτής της πτυχιακής. Ο ολοκληρωμένος κώδικας των κλήσεων και συναρτήσεων που κατασκευάστηκαν και χρησιμοποιήθηκαν, καθώς και το documentation αυτών, παρατείθεται στο παράρτημα ΑΡΙΘΜΟΣ ΠΑΡΑΡΤΗΜΑΤΟΣ ΚΩΔΙΚΑ. Στο παρόν κεφάλαιο παρουσιάζονται αναλυτική επεξήγηση της κάθε μεθόδου και κλήσης καθώς και εικόνες των αποτελεσμάτων που επιστρέφονται με τη χρήση αυτών.

```
def construct_json(tbl,result):
    """Prepare data to be displayed in JSON format
```

Η μέθοδος `construct_json` βοηθάει στην δημιουργία περισσότερο ευανάγνωστων JSON objects που εμφανίζονται στον browser μετά από χρήση κάποιας κλήσης. Χρησιμοποιείται σε κάθε μίας από τις επόμενες κλήσεις του API και αντιπροσωπεύει

την καλή τεχνική επαναχρησιμοποίησης κώδικα.

Η μέθοδος αυτή αντιστοιχίζει dictionary keys, τα οποία αντιπροσωπεύουν ονόματα columns της βάσης δεδομένων, στις κατάλληλες τιμές τους. Οι τιμές αυτές βρίσκονται στον δισδιάστατο πίνακα (2D array) “result”, που περνιέται ως όρισμα στην συνάρτηση και αντιπροσωπεύει το αποτέλεσμα της κλήσης της συνάρτησης cursor.fetchall(), που συλλέγει τα κατάλληλα δεδομένα με βάση κάποιο query που δίνεται πριν την κλήση της.

Η παράμετρος “tbl” είναι τύπου string και αποτελεί το όνομα του table της βάσης από το οποίο “τραβώνται” δεδομένα κάθε φορά. Τέλος, η συνάρτηση επιστρέφει μία λίστα ονόματι data, με το ενανάγνωστα πλέον κατασκευασμένο dictionary.

```
@app.route('/all/<table>')
def get_all(table):
    "Construct \"localhost:5000/all/<table>\" url functionality."
```

Ο κώδικας κάθε κλήσης του API ξεκινάει με @app.route(<URL>), όπου καθορίζεται το URL για το οποίο κατασκευάζεται η κλήση αυτή. Στη συνέχεια ορίζεται η συνάρτηση Python που περιέχει την λογική της εκάστοτε κλήσης. Κάθε φορά που πραγματοποιείται ένα GET request με το αναγραφόμενο URL, η συνάρτηση αυτή εκτελείται αυτόματα. Παρατηρείται πως στο ορισμένο URL το table βρίσκεται ανάμεσα σε ανισοτικά σύμβολα. Η σύνταξη αυτή καθιστά την περικλειόμενη λέξη ως άγνωστη μεταβλητή με αποτέλεσμα αυτή να αποτελέι παράμετρο, η τιμή της οποίας δίνεται από τον χρήστη στο URL. Η χρήση αυτού του είδους παραμετροποίησης απαιτεί η χρησιμοποιούμενη παράμετρος να δωθεί και ως όρισμα στην συνάρτηση.

Με την πραγματοποίηση του request αυτού, στη συνάρτηση κατασκευάζεται το κατάλληλο sql query για να επιστραφούν όλες οι καταχωρήσεις του table που δώθηκε ως παράμετρος, σε JSON format. Και εδώ, όπως και σε όλες τις υπόλοιπες κλήσης, γίνεται χρήση της συνάρτησης construct_json(table,result), όπως φαίνεται και στον κώδικα του παραρτήματος. Τα ονόματα που γίνονται αποδεκτά ως table names είναι μόνο αυτά των τριών tables που κατασκευάστηκαν στα πλαίσια της εργασίας, “LivingRoomTemp”, “Power”, “VRMS”.

Η συνάρτηση επιστρέφει το query result σε json format, σε ενανάγνωστη πλέον μορφή. Στην εικόνα 35 παρουσιάζεται το αποτέλεσμα της κλήσης για το table θερμοκρασίας, LivingRoomTemp.

```

{
  "data": [
    {
      "date": "Fri, 04 May 2018 21:58:11 GMT",
      "id": 1,
      "room": "livingRoom",
      "scale": "celsius",
      "temperature": 25.7
    },
    {
      "date": "Fri, 04 May 2018 21:58:16 GMT",
      "id": 2,
      "room": "livingRoom",
      "scale": "celsius",
      "temperature": 25.7
    },
    {
      "date": "Fri, 04 May 2018 21:59:13 GMT",
      "id": 3,
      "room": "livingRoom",
      "scale": "celsius",
      "temperature": 25.7
    },
    {
      "date": "Fri, 04 May 2018 22:00:13 GMT",
      "id": 4,
      "room": "livingRoom",
      "scale": "celsius",
      "temperature": 25.7
    },
    {
      "date": "Fri, 04 May 2018 22:01:13 GMT",
      "id": 5,
      "room": "livingRoom",
      "scale": "celsius",
      "temperature": 25.8
    },
    {
      "date": "Fri, 04 May 2018 22:02:13 GMT",
      "id": 6,
      "room": "livingRoom",
      "scale": "celsius",
      "temperature": 25.7
    }
  ]
}

```

ΕΙΚΟΝΑ 35: localhost:5000/all/LivingRoomTemp URL

```

@app.route('/today/<table>')
def get_today(table):
    """Construct "localhost:5000/today/<table>" URL functionality.

```

Η συγκεκριμένη κλήση επιστρέφει καταχωρήσεις ενός table μόνο αν αυτές περιλαμβάνουν timestamp ίδιο με την ημερομηνία πραγματοποίησης της, δηλαδή τις μετρήσεις που ανακτίθηκαν την ίδια μέρα. Πραγματοποιείται και πάλι κατασκευή του κατάλληλου sql query ενώ το table name δίνεται από τον χρήστη στο URL με τρόπο όμοιο αυτού που περιγράφηκε προηγουμένως. Κατάλληλα table names είναι και πάλι τα “LivingRoomTemp”, “Power”, “VRMS” ενώ η ανάκτηση της τρέχουσας ημερομηνίας πραγματοποιείται με χρήση του Datetime module της Python. Και αυτή, όπως και όλες οι κλήσεις που προηγήθηκαν και ακολουθούν, επιστρέφει το query result σε ευανάγνωστη json μορφή, με χρήση της συνάρτησης jsonify().

Για τα αποτελέσματα αυτής της κλήσης δεν παρατείθεται εικόνα καθώς η συγγραφή του κειμένου πραγματοποιήθηκε κατά ένα χρονικό διάστημα όπου δεν υπήρχε πρόσβαση στις συσκευές emonPi και emonTx.

```

@app.route('/query/<table>')
def query(table):
    """Construct "localhost:5000/query/<table>?startDate=value1&endDate=value2" URL
functionality

```

Στο παρόν URL συναντάται για πρώτη φορά στην παρούσα εργασία η χρήση παραμέτρων με την σύνταξη που αναγράφεται στο δοθέν κομμάτι του documentation. Προκειμένου να εκτελεστεί σωστά η λειτουργία της συνάρτησης, η σωστή σύνταξη του URL είναι ιδιαίτερα σημαντική, με το ? να δηλώνει πως από το σημείο εκείνο και μέτα ακολουθούν μόνο παράμετροι και με τις παραμέτρους φυσικά να ακολουθούν πάντα το αγγλικό σημείο στήξης. Σε αντίθεση με τον τρόπο παραμετροποίησης που απαντήθηκε στις προηγούμενες κλήσεις, στο συγκεκριμένο είναι απαραίτητο να αναγράφεται η τιμή αλλά και το όνομα της μεταβλητής, με την τιμή να ανατείθεται σε

αυτή με το ίσον (=).

Ο χρήστης καλείται να δώσει ως παραμέτρους στο URL μία αρχική και μία τελική ημερομηνία σε μορφή YYYY-MM-DD. Κατασκευάζεται έπειτα το κατάλληλο sql query ώστε να ανακτηθούν οι καταχωρήσεις της βάσης με ημερομηνία (timestamp) μεταξύ αυτών των δύο. Οι τιμές που δίνει ο χρήστης ανακτώνται στον κώδικα με χρήση της βιβλιοθήκης requests του package Flask ενώ οι καταχωρήσεις με timestamp ίδιο με το endDate δεν επιστρέφονται.

Στην EIKONA48 παρουσιάζονται τα αποτελέσματα διαστήματος μίας ημέρας, από τις 05/04 έως 06/04 για να σημειωθεί η απουσία αποτελεσμάτων με timestamp 2018-04-06



The screenshot shows a browser window with the URL `localhost:5000/query/LivingRoomTemp?startDate=2018-05-05&endDate=2018-05-06`. The page displays a JSON response with the following data:

```
{ "data": [ { "date": "Sat, 05 May 2018 01:15:25 GMT", "id": 11, "room": "livingRoom", "scale": "celsius", "temperature": 23.7 }, { "date": "Sat, 05 May 2018 01:15:30 GMT", "id": 12, "room": "livingRoom", "scale": "celsius", "temperature": 23.7 }, { "date": "Sat, 05 May 2018 20:13:32 GMT", "id": 13, "room": "livingRoom", "scale": "celsius", "temperature": 26.2 } ] }
```

EIKONA 36: localhost:5000/query/<table>?startDate=value1&endDate=value2 URL

```
@app.route('/avg/<table>')
def avg_between_dates(table):
    """Construct "localhost:5000/avg/<table>?date1=value1&date2=value2" URL
    functionality
```

Το συγκεκριμένο URL επιτελεί λειτουργία παρόμοια αυτής του προηγούμενου του καθώς οι παράμετροι ημερομηνιών δίνονται στο URL και ανακτώνται στο πρόγραμμα με παρόμοιο τρόπο. Η διαφορά εντωπίζεται στο επιστρέψιμο αποτέλεσμα της κλήσης, το οποίο είναι ο μέσος όρος τιμών των καταχωρήσεων του table που δίνεται, μεταξύ των δωθέντων ημερομηνιών. Το μέγεθος της τιμής μετράται σε βαθμούς κελσίου, Ampere ή VRMS ανάλογα με το table στο οποίο πραγματοποιείται η κλήση.

Το αποτέλεσμα της κλήσης αυτής για διάστημα 16 ημερών, από τις 04/05 έως τις 20/05 στο table LivingRoomTemp παρουσιάζεται στην εικόνα 37.

```
{
  "Average": [
    {
      "Temperature": 23.72874
    }
  ]
}
```

EIKONA 37: localhost:5000/avg/<table>?date1=value1&date2=value2 URL

```
@app.route('/avg/today/<table>')
def avg_today(table):
    """Construct 'localhost:5000/avg/today/<table>' URL functionality.
```

Συνδιάζοντας την λογική των προαναφερθέντων κλήσεων, το παρόν URL επιστρέφει τον μέσο όρο των μετρήσεων με timestamp ίδιο με την τρέχουσα ημερομηνία. Η ημερομηνία ανακτάται και πάλι από το Datetime module και τα μεγέθη της τιμής που επιστρέφεται είναι Celsius, Ampere, VRMS κατά αντιστοιχία με τα table.

Όπως και στην προηγούμενη κλήση έτσι και εδώ, παρά το γεγονός ότι επιστρέφεται μόνο μία τιμή και όχι πλήθως αυτών, το format JSON διατηρείται προκειμένου να υπάρχει συνοχή στο πρόγραμμα και να ακολουθηθούν οι κανόνες που συμφωνήθηκαν πριν την κατασκευή του.

Όπως και στη περίπτωση του localhost:5000/today/<table> url, έτσι και εδώ δεν παρατείθεται εικόνα καθώς η συγγραφή του κειμένου πραγματοποιήθηκε κατά ένα χρονικό διάστημα όπου δεν υπήρχε πρόσβαση στις συσκευές emonPi και emonTx.

6.6.1 Υλοποίηση pagination

Όπως παρατηρείται, είναι συχνό φαινόμενο η επιστροφή μεγάλου όγκου δεδομένων με κάποιες από τις παραπάνω κλήσεις API. Για ευκολότερη διάκριση αυτών αλλά και για αποφυγή υπερφόρτωσης του μέσου ανάκτησης (στην παρούσα περίπτωση browser) είναι καλή τεχνική η υλοποίηση pagination. Με τον όρο pagination εννοείται η εμφάνιση μέρους των αποτελεσμάτων κάθε φορά με την δυνατότητα αλλαγής “σελίδας” για εμφάνιση των υπόλοιπων. Στην παρούσα εργασία χρησιμοποιήθηκε όριο (limit) μετρήσεων οι 5 μετρήσεις ανά σελίδα ενώ οι αλλαγές σελίδας πραγματοποιούνται με χρήση κατάλληλης παραμέτρου στο URL, οι τιμές τις οποίας καθορίζονται από τον χρήστη. Το pagination υλοποιήθηκε για ένα εκ των προαναφερθέντων URL ενώ οδηγίες για την εφαρμογή του και σε άλλα δίνονται στο documentation κατάλληλων για pagination URLs (π.χ. του @app.route('/query/<table>')). Η υλοποίησή του σε όλες τις κατάλληλες κλήσεις είναι δουλειά που συμπεριλαμβάνεται στην εν δυνάμει μελλοντική επέκταση αυτής της εργασίας.

```
@app.route('/paginate/<table>')
def return_all(table):
    """Implements pagination. Construct "localhost:5000/paginate/<table>" URL
       functionality
```

```

    {
      "data": [
        {
          "date": "Thu, 10 May 2018 20:58:57 GMT",
          "id": 1,
          "topic": "VRMS",
          "value": 238.6
        },
        {
          "date": "Thu, 10 May 2018 20:59:10 GMT",
          "id": 2,
          "topic": "VRMS",
          "value": 239.2
        },
        {
          "date": "Thu, 10 May 2018 20:59:39 GMT",
          "id": 3,
          "topic": "VRMS",
          "value": 239.4
        },
        {
          "date": "Thu, 10 May 2018 20:59:44 GMT",
          "id": 4,
          "topic": "VRMS",
          "value": 239.4
        },
        {
          "date": "Thu, 10 May 2018 21:01:03 GMT",
          "id": 5,
          "topic": "VRMS",
          "value": 238.8
        }
      ]
    }

    {
      "data": [
        {
          "date": "Thu, 10 May 2018 21:01:19 GMT",
          "id": 6,
          "topic": "VRMS",
          "value": 239.4
        },
        {
          "date": "Thu, 10 May 2018 21:01:48 GMT",
          "id": 7,
          "topic": "VRMS",
          "value": 238.6
        },
        {
          "date": "Thu, 17 May 2018 11:38:27 GMT",
          "id": 8,
          "topic": "VRMS",
          "value": 242.0
        },
        {
          "date": "Thu, 17 May 2018 11:39:05 GMT",
          "id": 9,
          "topic": "VRMS",
          "value": 242.0
        },
        {
          "date": "Thu, 17 May 2018 11:40:05 GMT",
          "id": 10,
          "topic": "VRMS",
          "value": 242.0
        }
      ]
    }
  
```

Εικόνα 38: localhost:5000/paginate/<table> URL

Στην ΕΙΚΟΝΑ 34 παρατηρείται πως για page=2 τα id των μετρήσεων ξεκινούν από id=6, σηματοδοτώντας την σωστή λειτοργία του pagination. Υπάρχει η δυνατότητα αλλαγής της παραμέτρου limit για εμφάνιση περισσότερων/λιγότερων αποτελεσμάτων σε κάθε σελίδα.

6.7 Εκτέλεση πειραμάτων

Επισημαίνεται πως τα πειράματα αυτά εκτελέστικαν σε μικρή κλίμακα, αυτή μερικών ωρών μίας συγκεκριμένης ημέρας και συνεπώς δεν καθίστανται αντιπροσωπευτικά για όλες τις ημέρες του χρόνου. Πραγματοποιήθηκαν για την πειραματική υλοποίηση της εφαρμογής Node-Red DetectAC δίνοντας την βάση για την υλοποίηση παρόμοιας εφαρμογής με πάντοτε έγκυρα αποτελέσματα. Η πραγματοποίηση τέτοιας εφαρμογής θα έκανε χρήση data disaggregation σε αντίθεση με την παρούσα περίπτωση. Ωστόσο παρουσιάζει ιδιαίτερο ενδιαφέρον, επιδεικνύει τις δυνατότητες τις παρούσας εργασίας και παρατείθεται στην συνέχεια.

Αρχικά το σύστημα αφέθηκε να λειτουργεί στον χώρο χωρίς την χρήση κλιματιστικού για μερικά λεπτά. Στο διάστημα αυτό καταγράφηκαν οι τιμές θερμοκρασίας και vrms και δίνονται στις παρακάτω εικόνες

<input type="checkbox"/>		Edit		Copy		Delete	164	VRMS	239.4	2018-05-20 11:58:29
<input type="checkbox"/>		Edit		Copy		Delete	165	VRMS	240.0	2018-05-20 11:59:08
<input type="checkbox"/>		Edit		Copy		Delete	166	VRMS	240.0	2018-05-20 11:59:22
<input type="checkbox"/>		Edit		Copy		Delete	167	VRMS	239.1	2018-05-20 12:00:22
<input type="checkbox"/>		Edit		Copy		Delete	168	VRMS	239.2	2018-05-20 12:01:22
<input type="checkbox"/>		Edit		Copy		Delete	169	VRMS	240.0	2018-05-20 12:02:22
<input type="checkbox"/>		Edit		Copy		Delete	170	VRMS	239.8	2018-05-20 12:03:22
<input type="checkbox"/>		Edit		Copy		Delete	171	VRMS	240.1	2018-05-20 12:04:23
<input type="checkbox"/>		Edit		Copy		Delete	172	VRMS	240.5	2018-05-20 12:05:23
<input type="checkbox"/>		Edit		Copy		Delete	173	VRMS	240.3	2018-05-20 12:06:22
<input type="checkbox"/>		Edit		Copy		Delete	174	VRMS	240.8	2018-05-20 12:07:22
<input type="checkbox"/>		Edit		Copy		Delete	175	VRMS	240.4	2018-05-20 12:08:22
<input type="checkbox"/>		Edit		Copy		Delete	176	VRMS	240.2	2018-05-20 12:09:22
<input type="checkbox"/>		Edit		Copy		Delete	177	VRMS	239.9	2018-05-20 12:10:23
<input type="checkbox"/>		Edit		Copy		Delete	178	VRMS	240.3	2018-05-20 12:11:22
<input type="checkbox"/>		Edit		Copy		Delete	179	VRMS	239.8	2018-05-20 12:12:22

Εικόνα 39: Μετρήσεις vrms χωρις AC

<input type="checkbox"/>	 Edit	 Copy	 Delete	176	livingRoom	26.2	celsius	2018-05-20 11:59:08
<input type="checkbox"/>	 Edit	 Copy	 Delete	177	livingRoom	26.2	celsius	2018-05-20 11:59:22
<input type="checkbox"/>	 Edit	 Copy	 Delete	178	livingRoom	26.2	celsius	2018-05-20 12:00:22
<input type="checkbox"/>	 Edit	 Copy	 Delete	179	livingRoom	26.1	celsius	2018-05-20 12:01:23
<input type="checkbox"/>	 Edit	 Copy	 Delete	180	livingRoom	26.1	celsius	2018-05-20 12:02:23
<input type="checkbox"/>	 Edit	 Copy	 Delete	181	livingRoom	26.1	celsius	2018-05-20 12:03:22
<input type="checkbox"/>	 Edit	 Copy	 Delete	182	livingRoom	26.1	celsius	2018-05-20 12:04:23
<input type="checkbox"/>	 Edit	 Copy	 Delete	183	livingRoom	26.1	celsius	2018-05-20 12:05:23
<input type="checkbox"/>	 Edit	 Copy	 Delete	184	livingRoom	26.2	celsius	2018-05-20 12:06:22
<input type="checkbox"/>	 Edit	 Copy	 Delete	185	livingRoom	26.1	celsius	2018-05-20 12:07:22
<input type="checkbox"/>	 Edit	 Copy	 Delete	186	livingRoom	26.2	celsius	2018-05-20 12:08:22
<input type="checkbox"/>	 Edit	 Copy	 Delete	187	livingRoom	26.2	celsius	2018-05-20 12:09:22
<input type="checkbox"/>	 Edit	 Copy	 Delete	188	livingRoom	26.2	celsius	2018-05-20 12:10:23
<input type="checkbox"/>	 Edit	 Copy	 Delete	189	livingRoom	26.2	celsius	2018-05-20 12:11:22
<input type="checkbox"/>	 Edit	 Copy	 Delete	190	livingRoom	26.2	celsius	2018-05-20 12:12:23

Εικόνα 40: Μετρήσεις θερμοκρασίας χωρις AC

Παρατηρείται πως χωρίς την χρήση κλιματιστικού οι τιμές του VRMS κυμαίνονται από 239.0 μέχρι 240.9 ενώ η θερμοκρασία μένει σχεδόν σταθερή στους 26.2 βαθμούς κελσίου. Οι μετρήσεις με την χρήση του κλιματιστικού παρουσίαζονται παρακάτω. Παρατηρείται πως τα ID των μετρήσεων συνεχίζουν σε αύξουσα σειρά εξασφαλίζοντας την (σχετική) ορθότητα του πειράματος

<input type="checkbox"/>				190	livingRoom	26.2	celsius	2018-05-20 12:12:23
<input type="checkbox"/>				191	livingRoom	26.2	celsius	2018-05-20 12:13:22
<input type="checkbox"/>				192	livingRoom	26.2	celsius	2018-05-20 12:14:22
<input type="checkbox"/>				193	livingRoom	26.1	celsius	2018-05-20 12:15:23
<input type="checkbox"/>				194	livingRoom	25.8	celsius	2018-05-20 12:16:23
<input type="checkbox"/>				195	livingRoom	25.5	celsius	2018-05-20 12:17:23
<input type="checkbox"/>				196	livingRoom	25.0	celsius	2018-05-20 12:18:23
<input type="checkbox"/>				197	livingRoom	24.6	celsius	2018-05-20 12:19:23
<input type="checkbox"/>				198	livingRoom	24.2	celsius	2018-05-20 12:20:23
<input type="checkbox"/>				199	livingRoom	24.0	celsius	2018-05-20 12:21:23
<input type="checkbox"/>				200	livingRoom	24.0	celsius	2018-05-20 12:22:23
<input type="checkbox"/>				201	livingRoom	24.1	celsius	2018-05-20 12:23:23
<input type="checkbox"/>				202	livingRoom	24.2	celsius	2018-05-20 12:24:23
<input type="checkbox"/>				203	livingRoom	24.2	celsius	2018-05-20 12:25:23
<input type="checkbox"/>				204	livingRoom	24.0	celsius	2018-05-20 12:26:23
<input type="checkbox"/>				205	livingRoom	23.7	celsius	2018-05-20 12:27:23
<input type="checkbox"/>				206	livingRoom	23.8	celsius	2018-05-20 12:28:24
<input type="checkbox"/>				207	livingRoom	24.0	celsius	2018-05-20 12:29:23
<input type="checkbox"/>				208	livingRoom	24.1	celsius	2018-05-20 12:30:23
<input type="checkbox"/>				209	livingRoom	24.0	celsius	2018-05-20 12:31:23
<input type="checkbox"/>				210	livingRoom	23.7	celsius	2018-05-20 12:32:23

Εικόνα 41: θερμοκρασία με την χρήση AC

<input type="checkbox"/>				180	VRMS	237.6	2018-05-20 12:13:23
<input type="checkbox"/>				181	VRMS	238.2	2018-05-20 12:14:23
<input type="checkbox"/>				182	VRMS	238.4	2018-05-20 12:15:23
<input type="checkbox"/>				183	VRMS	239.2	2018-05-20 12:16:23
<input type="checkbox"/>				184	VRMS	238.6	2018-05-20 12:17:23
<input type="checkbox"/>				185	VRMS	239.0	2018-05-20 12:18:23
<input type="checkbox"/>				186	VRMS	238.0	2018-05-20 12:19:23
<input type="checkbox"/>				187	VRMS	238.0	2018-05-20 12:20:23
<input type="checkbox"/>				188	VRMS	240.2	2018-05-20 12:21:24
<input type="checkbox"/>				189	VRMS	238.9	2018-05-20 12:22:23
<input type="checkbox"/>				190	VRMS	239.3	2018-05-20 12:23:23
<input type="checkbox"/>				191	VRMS	238.4	2018-05-20 12:24:23
<input type="checkbox"/>				192	VRMS	238.5	2018-05-20 12:25:23
<input type="checkbox"/>				193	VRMS	238.1	2018-05-20 12:26:23
<input type="checkbox"/>				194	VRMS	240.2	2018-05-20 12:27:23
<input type="checkbox"/>				195	VRMS	240.1	2018-05-20 12:28:25
<input type="checkbox"/>				196	VRMS	239.8	2018-05-20 12:29:23
<input type="checkbox"/>				197	VRMS	237.9	2018-05-20 12:30:23
<input type="checkbox"/>				198	VRMS	238.1	2018-05-20 12:31:23
<input type="checkbox"/>				199	VRMS	237.7	2018-05-20 12:32:23

Εικόνα 42: vrms με την χρήση AC

Για τις τιμές VRMS παρατηρείται πως, με την εξαίρεση δύο μόνο μετρήσεων που οριακά ξεπερνούν την τιμή 240, οι τιμές πλέον κυμαίνονται αυστηρά κατώ από 240.0 ενώ, όπως είναι φυσικό, παρατηρείται μεγάλη πτώση της θερμοκρασίας, αυστηρά κάτω

από 26 βαθμούς κελσίου. Από τα timestamp των δύο table διαπιστώνεται πως οι μετρήσεις λήφθηκαν ταυτόχρονα.

Ορίζονται έτσι τα bottleneck τιμών ως ανστηρά < 240.0 για το VRMS, όταν λειτουργεί το AC και ανστηρά < 26.0 κελσίου. Τα όρια αυτά χρησιμοποιούνται στην εφαρμογή Node-Red Detect_AC.

6.8 Υλοποίηση της εφαρμογής Node-Red Detect_AC

Όπως και στις προηγούμενες εφαρμογές Node-Red, το flow της DetectAC έχει ήδη παρουσιαστεί στην παράγραφο 5.3.3 καθώς και οι λειτουργίες των switch και limiter nodes. Τα κομμάτια κώδικα JavaScript που χρησιμοποιούνται καθώς και τα configuration των πρώτων node είναι ιδιαίτερα απλά και η επεξήγηση των εφαρμογών Node-Red που προηγήθηκαν, επαρκεί πλήρως για την κατανόησή τους. Τα τμήματα JavaScript παραθέτονται στη συνέχεια

```
//Filter VRMS node
if(msg.topic=="emon/emonTx3/vrms")
{return msg;}
//Filter temperature node
if(msg.topic=="emon/emonTx3/temp1")
{return msg;}
//Detect AC node
msg.payload = "Note: LivingRoom AC on";
return msg;
```

Τα nodes Twitter In και email node απαιτούν ως configuration τα credentials ενός twitter account και ενός gmail account (του αποστολέα) αντίστοιχα, ενώ το τελευταίο χρειάζεται και τη διεύθυνση email του παραλήπτη. Τα αποτελέσματα της χρήσης της εφαρμογής στο twitter παρουσιάζονται στην εικόνα 43.



Εικόνα 43: Αυτόματο tweet από την εφαρμογή Detect_AC

ΚΕΦΑΛΑΙΟ 7: Συμπεράσματα

ΣΥΜΠΕΡΑΣΜΑΤΑ

Η εργασία αυτή αφορούσε το τεχνικό κομμάτι έξυπνων τεχνολογιών μέτρησης για την κατανάλωση ενέργειας αλλά και άλλων μετρήσεων ενός σπιτιού. Στόχος της ήταν η αρχικά η εξοικείωση με υπάρχον σύστημα energy monitoring και τελικά η διαμόρφωση και επέκταση αυτού με λειτουργίες κατάλληλες που αρμόζουν στο πλαίσιο του γενικότερου θέματος του Energy Dissagregation. Στα πλαίσια του στόχου αυτού μελετήθηκαν αρκετές δημοσιεύσεις και περισσότερες διαδικτυακές πηγές προκειμένου να επιτευχθούν μικρότεροι -πιο συγκεκριμένοι- στόχοι του αρχικού προβλήματος. Αυτοί ήταν:

1. Συλλογή δεδομένων: εγκαταστάθηκε και τροποποιήθηκε κατάλληλα υπάρχον σύστημα του οργανισμού OpenEnergyMonitor, ώστε η συλλογή δεδομένων από αυτό να καλύπτει τις ανάγκες της εργασίας.
2. Αποθήκευση δεδομένων: δημιουργήθηκε νέα βάση δεδομένων σε τοπικό επίπεδο με δυνατότητα online deployment αυτής. Κατασκευάστηκαν δύο εφαρμογές που με διαφορετικό τρόπο η κάθε μία πλέον αποστέλλουν τις μετρήσεις του εγκατεστημένου συστήματος και στην τοπική βάση.
3. Ανάκτηση δεδομένων: Κατασκευάστηκε εξ ολοκλήρου API σε γλώσσα Python, με τη χρήση της βιβλιοθήκης Flask. Με τη χρήση αυτού, δεδομένα επιστρέφονται από τη βάση ως response σε GET requests, σε μορφή JSON.
4. Πειραματισμός-Εκμετάλλευση των δυνατοτήτων της εργασίας: πραγματοποιήθηκαν πειράματα μικρής κλίμακας, όσο αναφορά το χρονικό περιθώριο, με παρακολούθηση της καταγραφής μετρήσεων με και χωρίς την λειτουργία AC στον χώρο. Με βάση τα αποτελέσματα αυτών, κατασκευάστηκε εφαρμογή αναγνώρισης της λειτουργίας AC και ειδοποίησης του χρήστη για αυτή, με δύο διαφορετικά μέσα.

Η διεκπεραίωση της εργασίας από την αρχή αυτής μέχρι το τέλος παρουσίασε πολυάριθμες δυσκολίες. Αρκετές από αυτές αναφέρονται σε σημεία του κειμένου. Τα σημαντικότερα θέματα κρίθηκε να είναι τα εξής:

Αρχικά, η μεγάλη έλλειψη γνώσεων σε ο,τι αφορά σχεδόν κάθε κομμάτι της εργασίας. Η πλήρης κατανόηση ενός υπάρχοντος project μεγέθους ίδιο με το project του οργανισμού OpenEnergyMonitor σίγουρα δεν υπήρξε εύκολη διαδικασία, καθώς η εργασία αυτή ήταν η πρώτη αφορμή έκθεσης του συγγραφέα σε τέτοιο. Η έλλειψη εξοικείωσης με τη γλώσσα προγραμματισμού υλοποίησης του project (Python) υπήρξε χρονοβόρο εμπόδιο τόσο στην κατανόηση αυτού όσο και στην δημιουργία του API και θα ήταν καλό να συμπεριληφθεί στα προαπαιτούμενα παρόμοιας εργασίας στο μέλλον. Ωστόσο τελικά, αυτή ήταν η αιτία απόκτησης πολλών πολύτιμων νέων γνώσεων και εμπειριών. Επίσης, η εξοικείωση με project πάνω σε Raspberry Pi κρίνεται εξ ίσου σημαντική καθώς η ύπαρξή της βοηθάει στην καλύτερη κατανόηση της λειτουργίας των συσκευών και την ευκολότερη χρήση αυτών σε ό,τι αφορά το λειτουργικό σύστημα Linux.

Επιπλέον, η εντυπωσιακά μικρή κοινότητα γύρω από τον οργανισμό OpenEnergyMonitor είναι κάτι που καθίσταται εμπόδιο στην ομαλή υλοποίηση. Η χρήση και η αναζήτηση στα forum του οργανισμού αρκετές φορές απέβη άκαρπη καθώς τα παλαιότερα θέματα συζήτησης είναι είτε πολύ υψηλότερου επιπέδου ή πολύ χαμηλότερου σε σχέση με ανάγκες που παρουσιάστηκαν κατά την εργασία. Το μικρό μέγεθος κοινότητας επίσης έχει ως αποτέλεσμα την ελάχιστη ή καμία ανταπόκριση σε περίπτωση χρήσης του forum για δημοσίευση.

Τελικά, ο στόχος της πτυχιακής εργασίας επιτεύχθει με αποτέλεσμα που κρίνεται πολύ καλό δεδομένων των γεγονότων και καταστάσεων. Η εργασία επιδέχεται επέκταση και τελειοποίηση, με δύο πρώτους μελλοντικούς στόχους να είναι αυτοί της επέκτασης του API και της βελτίωσης της εφαρμογής Detect_AC. Για τον δεύτερο, απαραίτητη είναι η πραγματοποίηση περισσότερων πειραμάτων προκειμένου το αποτέλεσμα να ανταποκρίνεται όσο το δυνατόν περισσότερο στην πραγματικότητα, καθ' όλη τη διάρκεια του ημερολογιακού έτους. Επίσης, όπως προαναφέρθηκε στο αντίστοιχο σημείο του κειμένου, η τελειοποίηση αυτής θα απαιτούσε χρήση τεχνικών Data Dissagregation.

ΠΑΡΑΡΤΗΜΑ I: Αναφορές

ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] U.Greveler, P.Glosekotterz, B. Justusy, D. Loehr. "Multimedia Content Identification Through Smart Meter Power Usage Profiles", Proceedings of the International Conference on Information and Knowledge Engineering (IKE); Athens: 1-8. Athens, 2012.
- [2] L. O. AlAbdulkarim and Z. Lukszo, —Smart Metering for the future energy systems in the Netherlands, in Fourth International Conference on Critical Infrastructures, 2009, pp. 1–7
- [3] Berhanu Regassa, Veronica Medina, Isabel M. Gomez, Octavio Rivera, and Jose A. Gomez -- Upgrading of Traditional Electric Meter Into Wireless Electric Meter Using ZigBee Technology

WEB SITES

- [1] <https://www.metering.com/top-stories/the-history-of-the-electricity-meter/> ,
accessed 19/7/2018
- [2] www.watthourmeters.com/history.html , accessed 16/7/2018
- [3] Postscapes, Tracking the Internet of Things, <http://postscapes.com/internet-of-things-history>, accessed 17/7/2018
- [4] EPA United States Environmental Protection Agency, Summary of the Energy Independence and Security Act, <http://www.epa.gov/lawsregulations/summary-energy-independence-andsecurity-act>, accessed 17/7/2018
- [5] The meaning of Data collection,
https://en.wikipedia.org/wiki/Data_collection ,accessed 17/7/2018
- [6] <https://www.slideshare.net/JohnWard23/importance-of-data-storage-and-backup-50883225> , accessed 17/7/2018
- [7] Definition of Cloud Storage <https://www.techopedia.com/definition/26535/cloud-storage> , accessed 18/7/2018
- [8] www.openenergymonitor.org ,accessed 18/7/2018
- [9] <https://github.com/openenergymonitor> , accessed 17/7/2018
- [10] <https://www.kickstarter.com/projects/openenergymonitor/emonpi-open-hardware-raspberry-pi-based-energy-mon>, accessed 18/7/2018
- [11] <https://emoncms.org/> , 18/7/2018
- [12] <https://emerging-technology.co.uk/>
- [13] <https://en.wikipedia.org/wiki/ATmega328>
- [14] <https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt>
- [15] <https://jeelabs.org/2011/06/09/rf12-packet-format-and-design/>
- [16] <https://guide.openenergymonitor.org/technical/credentials/>

ΠΑΡΑΡΤΗΜΑ II: Ακρόνυμα

ΑΓΓΛΙΚΟΙ ΟΡΟΙ

Ακρόνυμο	Επεξήγηση
kWh	Kilowatt hour
J	Joule
AC	Alternating Current
AMI	Advanced Meter Infrastructure
IoT	Internet of Things
MIT	Massachusetts Institute of Technology
HAN	Home Area Network
BAN	Building Area Network
LAN	Local Area Network
VM	Virtual Machine
API	Application Programming Interface
MHz	MegaHertz
VRMS	Voltage Root Mean Square

ΠΑΡΑΡΤΗΜΑ ΙΙΙ: Γλωσσάριο

ΓΛΩΣΣΑΡΙΟ

Όρος	Επεξήγηση
Altrernate Current	Εναλασσόμενο ρεύμα
Application Programming Interface	Διεπαφή για προγραμματισμό εφαρμογών
Cloud storage	Αποθήκευση δεδομένων σε σερβερ στο διαδίκτυο
Energy Monitoring	Παρακολούθηση της ενέργειας
Gateway	Πύλη
Internet of Things	Το διαδίκτυο των πραγμάτων
Node	Κόμβος
Pagenation	Σελιδοποίηση
Requests	Κλήσεις

ΠΑΡΑΡΤΗΜΑ IV: Κώδικας

ΚΩΔΙΚΑΣ

```

from flask import Flask, jsonify, json, request
import MySQLdb
import datetime
#from flask_restful import Resource, Api
#from flask_sqlalchemy import SQLAlchemy
#from sqlalchemy import create_engine

#declare that our app is a WSGI (Web Server Gateway Interface) app
app = Flask(__name__)

#Connect to db
db = MySQLdb.connect(#host="localhost",
                      user="root",
                      #port=3306,
                      passwd="",
                      db="testDB",
                      unix_socket="/opt/lampp/var/mysql/mysql.sock")

cursor = db.cursor() #initialize cursor

def construct_json(tbl,result):
    """Prepare data to be displayed in JSON format

    This module prepares data to be formated in JSON
    according to the table name passed as argument.
    Matches dictionary keys, representing database
    column names, to their appropriate values passed
    in the result array parameter.

    :type tbl: string
    :param tbl: the name of the database table

    :type result: 2D array
    :param result: the array returned from cursor.fetchall()

    :returns: a list with the constructed dictionary
    """

    data = []

    if tbl=="LivingRoomTemp":
        for i in range(0,len(result)):
            temp_data={

```

```
'id' : result[i][0],  
'room' : result[i][1],  
'temperature' : result[i][2],  
'scale' : result[i][3],  
'date' : result[i][4]  
}  
data.append(temp_data)  
elif tbl=="VRMS":  
    for i in range(0,len(result)):  
        temp_data={  
            'id' : result[i][0],  
            'topic' : result[i][1],  
            'value' : result[i][2],  
            'date' : result[i][3]  
        }  
        data.append(temp_data)  
return data
```

```
@app.route('/all/<table>')  
def get_all(table):  
    """Construct "localhost:5000/all/<table>" url functionality.
```

Executes appropriate sql query to return all entries in the corresponding table of the database in JSON format.
Valid table names: "LivingRoomTemp", "VRMS"

```
:type table: string  
:param table: the table name given in the URL  
  
:returns: the query result set in JSON format  
""  
  
try:  
    cursor.execute("SELECT * FROM "+table)  
except Exception as e:  
    return "Exception type: "+ str(e)  
result = cursor.fetchall()  
data = construct_json(table,result)  
return jsonify({'data' : data})
```

```
@app.route('/today/<table>')  
def get_today(table):  
    """Construct "localhost:5000/today/<table>" URL functionality.
```

Retrieves current date from Datetime module.
Executes appropriate sql query to return entries in the corresponding table of the database with today's timestamp,

in JSON format. Valid table names: "LivingRoomTemp", "VRMS".

```
:type table: string
:param table: the table name given in the URL

:returns: the query result set in JSON format
"""

now = datetime.datetime.now()
rightnow = now.strftime("%Y-%m-%d")
rnow = rightnow+'%' #attach a % behind date for query purposes
select_stmt = "SELECT * FROM " +table+ " WHERE timestamp
LIKE %(test)s"
try:
    cursor.execute(select_stmt, {'test': rnow})
except Exception as e:
    return "Exception type: "+ str(e)

result = cursor.fetchall()
data = construct_json(table,result)
return jsonify({'data' : data})

@app.route('/query/<table>')
def query(table):
    """Construct
    "localhost:5000/query/<table>?startDate=value1&endDate=value2" URL
    functionality
```

Retrieves dates given in the URL as parameters. Executes appropriate sql query
to return entries in the corresponding table of the database, between the given dates. Input values in YYYY-MM-DD format. Valid table names:
"LivingRoomTemp",
"VRMS". Entries corresponding to second date are not returned.
Uncomment commented parts to implement pagination.

```
:type table: string
:param table: the table name given in the URL

:returns: the query result set in JSON format
"""

date1 = request.args.get('startDate')
date2 = request.args.get('endDate')

#page = request.args.get('page', default=1, type=int)
#limit = request.args.get('limit', default = 5, type=int)

#my_int = ((page-1)*limit)+1
```

```
queryy="SELECT * FROM "+table+" WHERE timestamp  
BETWEEN %(d1)s AND %(d2)s" #AND id>="+str(my_int)+" LIMIT "+str(limit)
```

```
try:  
    cursor.execute(queryy,{'d1': date1, 'd2': date2})  
except Exception as e:  
    return "Exception type: "+ str(e)  
result = cursor.fetchall()  
data = construct_json(table,result)  
return jsonify({'data' : data})
```

```
@app.route('/paginate/<table>')  
def return_all(table):  
    """Implements pagination. Construct "localhost:5000/paginate/<table>" URL  
functionality
```

Executes appropriate sql query to return all entries in the corresponding table of the database, paginated. Pass "limit=.." & "page=.." parameters in URL to change the defaults and navigate through pages.
Default values: limit=5, page=1

:type table: string
:param table: the table name given in the URL

:returns: the query result set in JSON format
""

page = request.args.get('page', default=1, type=int)
limit = request.args.get('limit', default = 5, type=int)
my_int = ((page-1)*limit)+1 #appropriate id number to appear first in each
page

```
try:  
    cursor.execute("SELECT * FROM "+table+" WHERE  
id>="+str(my_int)+" LIMIT "+str(limit))  
except Exception as e:  
    return "Exception type: "+ str(e)  
result = cursor.fetchall()  
data = construct_json(table,result)  
return jsonify({'data': data})
```

```
@app.route('/avg/<table>')  
def avg_between_dates(table):  
    """Construct "localhost:5000/avg/<table>?date1=value1&date2=value2" URL  
functionality
```

Retrieves dates given in the URL as parameters. Executes appropriate sql

query

to return the average value of in the corresponding table of the database, between the given dates. Input values in YYYY-MM-DD format.
Valid table names: "LivingRoomTemp", "VRMS".

:type table: string

:param table: the table name given in the URL

:returns: the average value in JSON format. Value is either Temperature or VRMS

measurement, depending on the table name provided.

"

date1 = request.args.get('date1')

date2 = request.args.get('date2')

queryy="SELECT AVG(payload) FROM "+table+" WHERE timestamp BETWEEN %(d1)s AND %(d2)s"

try:

cursor.execute(queryy,{'d1': date1, 'd2': date2})

except Exception as e:

return "Exception type: "+ str(e)

result = cursor.fetchall()

data = []

if table=="LivingRoomTemp":

temp_data={

'Temperature':result[0][0]

}

elif table=="VRMS":

temp_data={

'VRMS': result[0][0]

}

data.append(temp_data)

return jsonify({'Average': data})

@app.route('/avg/today/<table>')

def avg_today(table):

"Construct 'localhost:5000/avg/today/<table>' URL functionality.

Retrieves current date from Datetime module.

Executes appropriate sql query to return the average value of the entries in the corresponding table of the database with today's timestamp. Valid table names: "LivingRoomTemp", "VRMS".

:type table: string

:param table: the table name given in the URL

:returns: the average value in JSON format. Value is either Temperature or VRMS

measurement, depending on the table name provided.

```
"""
now = datetime.datetime.now()
rightnow = now.strftime("%Y-%m-%d")
rnow = rightnow+'% #attach a % behind date for query purposes
select_stmt = "SELECT AVG(payload) FROM " +table+ " WHERE timestamp
LIKE %(test)s"
try:
    cursor.execute(select_stmt, {'test': rnow})
except Exception as e:
    return "Exception type: "+ str(e)
result = cursor.fetchall()
data = []
if table=="LivingRoomTemp":
    temp_data={
        'Temperature':result[0][0]
    }
elif table=="VRMS":
    temp_data={
        'VRMS': result[0][0]
    }
data.append(temp_data)

return jsonify({'Average_today': data})

if __name__ == '__main__':
    app.run()
```