Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Ηλεκτρικων Βιομηχανικων Διαταξεων και Συστηματων Αποφασεων

Ανάπτυξη Εφαρμογής Smart Home για την Διαχείριση IoT Συσκευών στο Edge

Διπλωματικη Εργασια

του

ΧΡΙΣΤΑΚΗ ΚΩΝΣΤΑΝΤΙΝΟΥ

Επιβλέπων: Ευάγγελος Μαρινάκης
Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2024

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Ηλεκτρικων Βιομηχανικων Διαταξεων και Συστηματων Αποφασεων

Ανάπτυξη Εφαρμογής Smart Home για την Διαχείριση IoT Συσκευών στο Edge

Διπλωματικη Εργασια

του

ΧΡΙΣΤΑΚΗ ΚΩΝΣΤΑΝΤΙΝΟΥ

Επιβλέπων: Ευάγγελος Μαρινάκης
Επίκουρος Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 15η Ιουλίου 2024.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Ευάγγελος Μαρινάκης	Ιωάννης Ψαρράς	Δημήτριος Ασκούνης
Επίκουρος Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2024

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Ηλεκτρικων Βιομηχανικων Διαταξεων και Συστηματων Αποφασεων

(Υπογραφή)

.....
Κωνσταντίνος Χριστάκης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

10 Ιουλίου 2024

Περίληψη

Τα τελευταία χρόνια η χρήση συσκευών Internet of Things σε οικιακό περιβάλλον γίνεται όλο και πιο διαδεδομένη. Τέτοιες συσκευές παράγουν τεράστια ποσότητα δεδομένων, η επεξεργασία των οποίων είναι απαραίτητη για την αξιοποίηση τους από τον τελικό χρήστη. Ο στόχος της παρούσας Διπλωματικής Εργασίας είναι η ανάπτυξη μιας Cloud πλατφόρμας για τον έλεγχο συσκευών έξυπνων σπιτιών και την συλλογή, την διαχείριση και την παρουσίαση των δεδομένων που συλλέγουν. Αντί μιας κεντροποιημένης αρχιτεκτονικής, γίνεται χρήση του μοντέλου Edge Computing με στόχο την μείωση του όγκου των δεδομένων που μεταφέρονται μέσω διαδικτύου και την γρηγορότερη απόκριση των συσκευών. Επιπλέον, παρουσιάζονται δύο εναλλακτικοί τρόποι επικοινωνίας του έξυπνου σπιτιού με την Cloud πλατφόρμα (MQTT και WebSocket).

Λέξεις Κλειδιά

Διαδίκτυο των πραγμάτων, Επεξεργασία, Αυτοματισμός οικίας, Έξυπνο σπίτι, raspberry pi, MQTT, pub/sub, WebSocket, IoT, Edge

Abstract

In recent years, the use of Internet of Things devices in the home environment has become more and more widespread. Such devices generate a huge amount of data, the processing of which is essential for its use by the end user. The goal of this Diploma Thesis is the development of a Cloud platform for controlling smart home devices and collecting, managing and presenting of the data they collect. Instead of a centralized architecture, Edge Computing model is used in order to reduce the amount of data transferred over the Internet and and to facilitate faster response time of the devices. In addition, two alternative ways for the smart home to communicate with the Cloud platform are demonstrated (MQTT and WebSocket).

Keywords

Internet of Things, processing, home automation, Smart home, raspberry pi, MQTT, pub/sub, WebSocket, IoT, Edge

στους γονείς μου

Ευχαριστίες

Θα ήθελα καταρχήν να ευχαριστήσω τον καθηγητή κ. Ευάγγελο Μαρινάκη για την επίβλεψη αυτής της διπλωματικής εργασίας και για την ευκαιρία που μου έδωσε να την εκπονήσω στο εργαστήριο Συστημάτων Αποφάσεων και Διοίκησης. Επίσης ευχαριστώ ιδιαίτερα τον Ιωάννη Παπία και τον Φίλιππο Σερέπα για τον χρόνο που διέθεσαν, την πολύτιμη καθοδήγησή τους και την εξαιρετική συνεργασία που είχαμε. Τέλος θα ήθελα να ευχαριστήσω τους γονείς μου για την συνεχή στήριξη και την ηθική συμπαράσταση που μου προσφέρουν όλα αυτά τα χρόνια.

Αθήνα, Ιούλιος 2024

Κωνσταντίνος Χριστάκης

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

Το Διαδικτύο των Πραγμάτων (IoT) έχει γνωρίσει μια εντυπωσιακή ανάπτυξη μέσα στην τελευταία δεκαετία. Ενδεικτικά, ο αριθμός των εγκατεστημένων συνδεδεμένων συσκευών IoT αναμένεται να αυξηθεί από περίπου 40 δισεκατομμύρια το 2023 σε 49 δισεκατομμύρια έως το 2026. Στην αρχή, οι πρώτες εφαρμογές του IoT αφορούσαν κυρίως την διαχείριση της εφοδιαστικής αλυσίδας (τεχνολογία RFID) και την αύξηση της αποδοτικότητας στην βαριά βιομηχανία (Machine-to-Machine Communication) [2]. Όμως, τα τελευταία χρόνια, η μείωση του κόστους των έξυπνων συσκευών, έχει οδηγήσει στην ταχεία υιοθέτησή τους από τους απλούς καταναλωτές για την διευκόλυνση της καθημερινότητάς τους. Τέτοιες συσκευές, διασυνδεδεμένες στο τοπικό δίκτυο του σπιτιού, δημιουργούν ένα Έξυπνο Σπίτι. Σε ένα Έξυπνο Σπίτι, συσκευές IoT, όπως έξυπνοι θερμοστάτες, μπρίζες, θερμοσίφωνες και οικιακές συσκευές, επιτρέπουν στον χρήστη να ελέγχει την λειτουργία τους εξ αποστάσεως και να παρακολουθεί την κατανάλωση ενέργειάς τους. Σήμερα, η μετατροπή ενός σπιτιού σε έξυπνο, μέσω της εγκατάστασης αυτών των συσκευών, γίνεται κυρίως για λόγους διευκόλυνσης της καθημερινότητας των κατοίκων. Στο εγγύς μέλλον, όμως, τέτοιες μετατροπές μπορεί να γίνουν αναγκαίες, λόγω της δυνατότητας ευέλικτης κατανάλωσης των συσκευών που προσφέρουν. Αυτό συμβαίνει γιατί, τελευταία, η κλιματική αλλαγή και τα προβλήματα που δημιουργεί γίνονται όλο και πιο έντονα. Παράλληλα, η ενεργειακή κρίση που έχει δημιουργηθεί λόγω της γεωπολιτικής κατάστασης στην Ευρώπη συνεχίζεται, καθιστώντας την ανάγκη για πράσινη μετάβαση πιο επίκαιρη από ποτέ. Σε αυτό το πλαίσιο, τα περισσότερα κράτη στρέφονται προς τις Ανανεώσιμες Πηγές Ενέργειας (ΑΠΕ) φωτοβολταικά, αιολικά [3]. Η ιδιαιτερότητα αυτών είναι πως η ποσότητα ενέργειας που παράγουν παρουσιάζει μεγάλες μεταβολές, ανάλογα με τις περιβαλλοντικές συνθήκες και την ώρα της ημέρας. Η ικανότητα, λοιπόν, των σπιτιών να προσαρμόζουν την κατανάλωσή τους και να εκμεταλλεύονται αποτελεσματικά την ενέργεια που παράγεται από τις ΑΠΕ, θα είναι καθοριστικής σημασίας για την επιτυχή πράσινη μετάβαση [4]. Η ανάγκη ανάπτυξης μιας εύχρηστης εφαρμογής που θα διαχειρίζεται ένα Smart Home είναι φανερή.

1.1 Υπάρχουσες λύσεις

Μέχρι στιγμής έχουν αναπτυχθεί αρκετές προτάσεις για την διαχείριση ενός έξυπνου σπιτιού. Οι πιο δημοφιλείς εξ αυτών έχουν υλοποιηθεί από τις μεγαλύτερες εταιρείες τεχνολογίας. Όμως, η κλειστή φύση του πηγαίου κώδικα αυτών των προτάσεων δημιουργεί ανη-

συχίες σε ορισμένους καταναλωτές σχετικά με την ιδιωτικότητα των δεδομένων τους. Ακόμη, τέτοιες προτάσεις παρουσιάζουν προβλήματα επεκτασιμότητας και παραμετροποίησης.

Ο τεράστιος όγκος δεδομένων που παράγεται από τις έξυπνες συσκευές παρουσιάζει σημαντικές ευκαιρίες για καινοτομία, αλλά θέτει επίσης προκλήσεις όσον αφορά τη διαχείριση και την επεξεργασία των δεδομένων [5]. Η πιο διαδεδομένη προσέγγιση που υιοθετούν οι παραπάνω λύσεις είναι η διασύνδεση αυτών των συσκευών με το Cloud και η ενσωμάτωση τους με άλλες υπηρεσίες που προσφέρουν αυτές οι εταιρείες. Όμως, η μεταφορά αυτού του τεράστιου όγκου δεδομένων μπορεί να δημιουργήσει συμφόρηση στο οικιακό δίκτυο, ενώ ο χρόνος απόκρισης του συστήματος στις εντολές του χρήστη ενδέχεται να είναι υψηλός.

Μια εναλλακτική προσέγγιση, που αντιμετωπίζει τα παραπάνω ζητήματα και κερδίζει σταδιακά έδαφος, ονομάζεται Edge Computing. Σύμφωνα με αυτή, ένα μέρος της επεξεργασίας αυτών των δεδομένων γίνεται πιο κοντά στις πηγές που τα παράγουν και στον τελικό χρήστη.

1.2 Αντικείμενο της διπλωματικής

Αντικείμενο της διπλωματικής είναι η ανάπτυξη μιας εφαρμογής Smart Home που θα διαχειρίζεται τις συσκευές ενός έξυπνου σπιτιού. Η εφαρμογή θα πρέπει να είναι σε θέση να πραγματοποιεί όλες τις βασικές λειτουργίες που προσφέρονται απο τις ήδη εδραιωμένες υπηρεσίες. Εκτός αυτών όμως, απαραίτητη προϋπόθεση είναι ο σχεδιασμός της με βάση το μοντέλο Edge Computing. Θα πρέπει, δηλαδή, να είναι σε θέση να επεξεργάζεται τα δεδομένα που παράγουν οι συσκευές, πριν αυτά σταλούν στο Cloud. Παράλληλα, η ευελιξία και η επεκταστιμότητα της πλατφόρμας είναι καίριας σημασίας. Αναλυτικότερα, ο σχεδιασμός της πλατφόρμας θα πρέπει να επιτρέπει την προσθήκη νέων εφαρμογών για την επεξεργασία των δεδομένων στο Edge. Επιπλέον, η προσθήκη μιας νέας συσκευής στην λίστα των υποστηριζόμενων συσκευών από την πλατφόρμα θα πρέπει να γίνεται με εύκολο τρόπο και να μην απαιτεί τον ανασχεδιασμό της. Ο απαραίτητος εξοπλισμός θα πρέπει να είναι χαμηλού κόστους, μικρού μεγέθους και εύκολος στην εγκατάσταση. Επίσης, θα πρέπει να δίνεται η δυνατότητα απομακρυσμένου ελέγχου των συσκευών ακόμη και όταν ο χρήστης βρίσκεται εκτός της οικίας του. Όμως, η εφαρμογή απευθύνεται σε χρήστες που δεν έχουν ιδιαίτερες τεχνικές γνώσεις. Για αυτόν τον λόγο τεχνικές όπως port forwarding, static public IPs, που εκτός από πολύπλοκες θέτουν και σε κίνδυνο έναν αρχάριο χρήστη, θα πρέπει να αποφευχθούν. Τέλος, η υλοποίηση θα πρέπει να γίνει κατά κύριο λόγο με λογισμικά ανοιχτού κώδικα.

1.3 Οργάνωση του τόμου

Η εργασία αυτή είναι οργανωμένη σε πέντε κεφάλαια: Στο Κεφάλαιο 2 δίνεται το θεωρητικό υπόβαθρο των βασικών τεχνολογιών που σχετίζονται με τη διπλωματική αυτή. Αρχικά, περιγράφεται ο όρος Διαδίκτυο των Πραγμάτων (Internet of Things) και οι δύο κύριες προσεγγίσεις για την διαχείριση του όκγου των δεδομένων που παράγονται από τις συσκευές IoT (Cloud και Edge Computing). Στην συνέχεια, παρουσιάζονται τα δύο εναλλακτιά πρωτόκολλα επικοινωνίας που χρησιμοποιεί η Πλατφόρμα (MQTT και WebSocket). Στο Κεφάλαιο 3

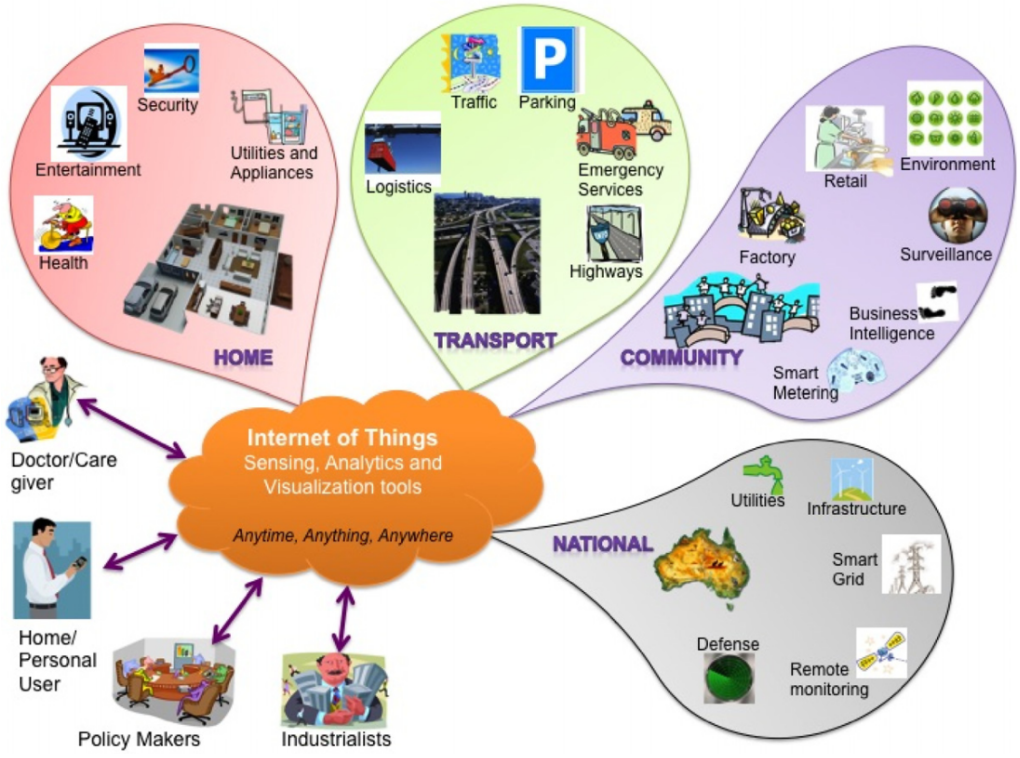
παρουσιάζεται η αρχιτεκτονική και η σχεδίαση του συστήματος, δηλαδή η περιγραφή των υποσυστημάτων και των εφαρμογών του, καθώς και λεπτομέρειες σχετικά με το υλικό, το λογισμικό και τα προγραμματιστικά εργαλεία που χρησιμοποιήθηκαν. Η περιγραφή της υλοποίησης του συστήματος, με ανάλυση των βασικών αλγορίθμων δίνεται στο Κεφάλαιο 4. Στο Κεφάλαιο 5 παρουσιάζεται ο έλεγχος καλής λειτουργίας του συστήματος. Τέλος, στο Κεφάλαιο 6 αναφέρεται η συνεισφορά αυτής της διπλωματικής εργασίας, καθώς και μελλοντικές επεκτάσεις.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

2.1 Internet of Things

Ο όρος Internet of Things (Διαδίκτυο των Πραγμάτων) αναφέρεται στο δίκτυο συνδεδεμένων συσκευών που ενσωματώνουν αισθητήρες, λογισμικό και άλλες τεχνολογίες, επιτρέποντας την συλλογή δεδομένων και την ανταλλαγή τους με άλλες συσκευές και συστήματα μέσω του Διαδικτύου. Το είδος αυτών των συσκευών ποικίλλει έντονα, από αισθητήρες, οικιακές συσκευές μέχρι και βιομηχανικά μηχανήματα. Οι συσκευές IoT δημιουργούν τεράστιες ποσότητες δεδομένων και γι' αυτό η αποτελεσματική αξιοποίηση αυτών των δεδομένων, απαιτεί ισχυρές δυνατότητες επεξεργασίας και ανάλυσης. Μια κυρίαρχη προσέγγιση για την διαχείριση αυτών των δεδομένων είναι η σύνδεση του IoT με το Cloud [6].



Εικόνα 2.1: *Σχηματική απεικόνιση του Internet of Things*

2.2 Cloud Computing

Το υπολογιστικό νέφος (Cloud) αποτελεί τη ραχοκοκαλιά της σύγχρονης ψηφιακής υποδομής, παρέχοντας μια ευέλικτη και κλιμακούμενη πλατφόρμα για την αποθήκευση, διαχείριση και επεξεργασία τεράστιων ποσοτήτων δεδομένων και εφαρμογών. Στην ουσία, πρόκειται για την παροχή υπολογιστικών υπηρεσιών μέσω του διαδικτύου σε χρήστες κατόπιν αιτήσεως. Αντί να βασίζεται σε τοπικούς εξυπηρετητές ή προσωπικές συσκευές, το Cloud Computing αξιοποιεί ένα δίκτυο απομακρυσμένων εξυπηρετητών που στεγάζονται σε κέντρα δεδομένων σε ολόκληρο τον κόσμο. Αυτό το κατανεμημένο μοντέλο επιτρέπει στους χρήστες να έχουν απρόσκοπτη πρόσβαση σε υπολογιστικούς πόρους από οποιαδήποτε τοποθεσία με σύνδεση στο διαδίκτυο, χωρίς την ανάγκη σημαντικών επενδύσεων σε υλικό και υποδομές [7].

Στα πλαίσια του IoT, ενώ το Cloud Computing έχει συμβάλει στην παροχή κλιμακούμενων και προσβάσιμων υπολογιστικών πόρων, αντιμετωπίζει επίσης σημαντικούς περιορισμούς. Μια βασική πρόκληση είναι ο τεράστιος όγκος δεδομένων που παράγεται από τις συσκευές IoT, ο οποίος μπορεί γρήγορα να υπερβεί τις παραδοσιακές υποδομές νέφους. Με δισεκατομμύρια διασυνδεδεμένες συσκευές που παράγουν συνεχώς δεδομένα στην άκρη του δικτύου, η μετάδοση όλων αυτών των πληροφοριών σε κεντρικούς διακομιστές νέφους μπορεί να οδηγήσει σε αυξημένη συμφόρηση του δικτύου. Επιπλέον, η εξάρτηση από τα κέντρα δεδομένων νέφους για την επεξεργασία και την αποθήκευση εισάγει πιθανά προβλήματα καθυστέρησης, ιδίως σε εφαρμογές που απαιτούν αποκρίσεις σε πραγματικό χρόνο ή αποκρίσεις χαμηλής καθυστέρησης [8].

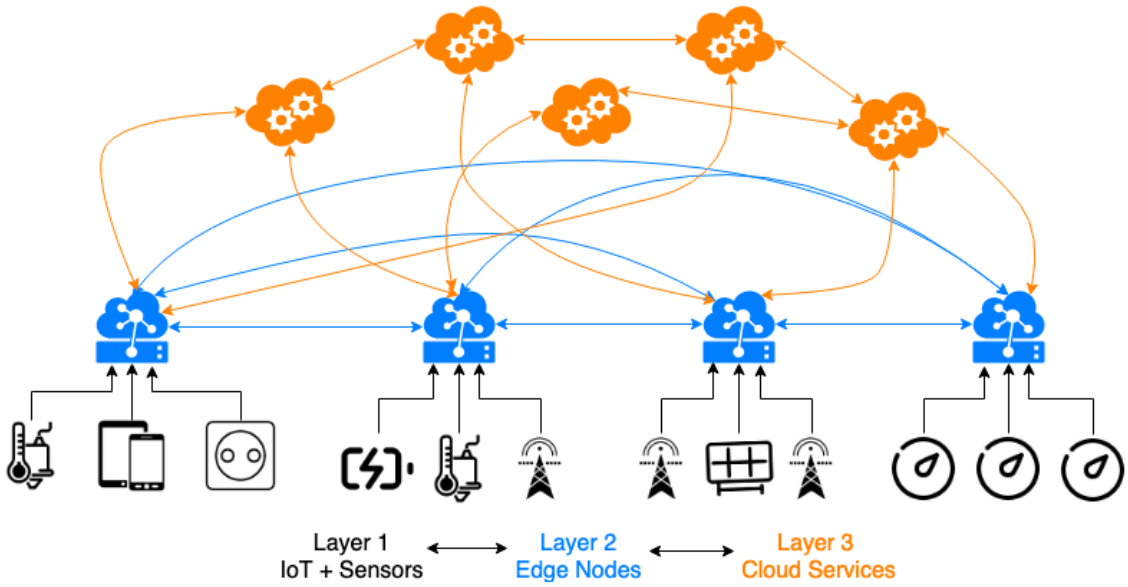
2.3 Edge Computing

2.3.1 Περιγραφή

Το μοντέλο Edge Computing προσπαθεί να αντιμετωπίσει τους προαναφερόμενους περιορισμούς του Cloud Computing, ιδιαίτερα στον τομέα του IoT. Η βασική ιδέα αυτού του νέου μοντέλου είναι η επεξεργασία των δεδομένων στην άκρη του δικτύου. Με άλλα λόγια, είναι η μετακίνηση υπηρεσιών που παραδοσιακά προσφέρονταν από τους εξυπηρετητές στο Cloud σε υπολογιστικούς πόρους πιο κοντά στο σημείο όπου παράγονται τα δεδομένα, δηλαδή κοντά στις συσκευές IoT και στους αισθητήρες. Μερικά οφέλη αυτής της προσέγγισης είναι η μείωση του χρόνου απόκρισης των συσκευών και η μεγαλύτερη αξιοπιστία της μεταξύ τους επικοινωνίας. Επιπλέον, μειώνεται ο όγκος των δεδομένων που μεταφέρονται μέσω του δικτύου στο Cloud.

2.3.2 Σχέση με Cloud Computing

Αξίζει να τονιστεί, πως το νέο αυτό μοντέλο δεν έχει ως στόχο την αντικατάσταση του παραδοσιακού μοντέλου Cloud Computing. Αντιθέτως, ο στόχος του είναι να το συμπληρώσει και να συμβάλει στην άμβλυνση των ανεπαρκειών του, καθώς και οι δύο προσεγγίσεις είναι απαραίτητες. Για παράδειγμα, στον τομέα του Smart Home, οι Edge κόμβοι αναλαμβάνουν



Εικόνα 2.2: *Σχηματική απεικόνιση αρχιτεκτονικής μοντέλου Edge Computing τριών επιπέδων [1]*

την συγκέντρωση, την μετατροπή και την προεπεξεργασία ετερογενών δεδομένων από διαφορετικού είδους συσκευές και τα μεταφορτώνουν στο Cloud. Η εις βάθος ανάλυση των δεδομένων, όμως, γίνεται σε εξυπηρητητές στο Cloud, καθώς οι ανάγκες σε υπολογιστικούς πόρους είναι πολύ μεγαλύτερες αν χρειαστεί να εξάγουμε πιο ουσιώδη συμπεράσματα. Τελικά, η ενσωμάτωση του Edge Computing με το Cloud Computing διαμορφώνει μια συμβιωτική σχέση, ενδυναμώνοντας τα οικοσυστήματα IoT με την ευελιξία και την ανταπόκριση που απαιτούνται για την πλήρη αξιοποίηση των δυνατοτήτων των διασυνδεδεμένων συσκευών και υπηρεσιών στην ψηφιακή εποχή [9].

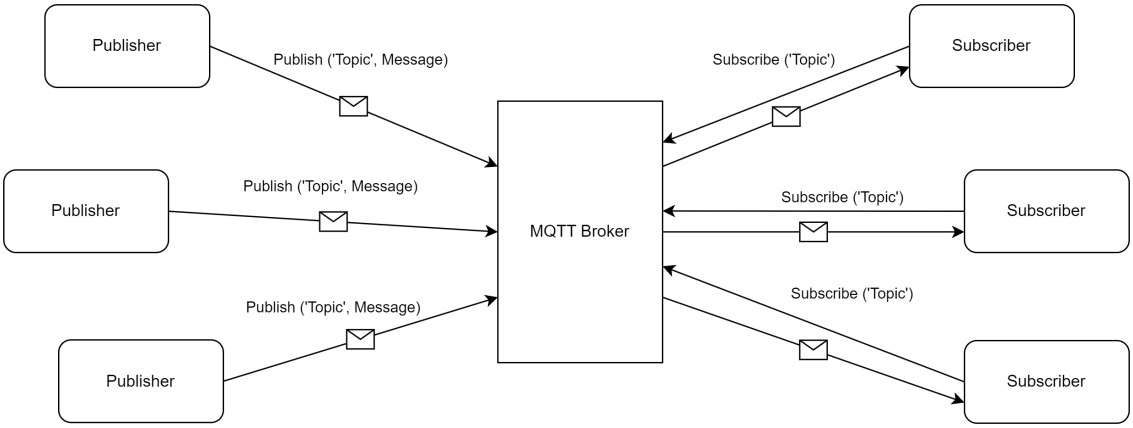
2.4 MQTT

Η διαλειτουργικότητα είναι καίριας σημασίας για τη δυνατότητα επικοινωνίας και αλληλεπίδρασης μεταξύ ετερογενών συσκευών και συστημάτων IoT. Τα πρότυπα και τα πρωτόκολλα διαδραματίζουν ζωτικό ρόλο στη διασφάλιση της συμβατότητας και της διαλειτουργικότητας σε διάφορες πλατφόρμες, συσκευές και εφαρμογές IoT. Ένα από τα πιο δημοφηλή τέτοια πρωτόκολλα είναι το MQTT (Message Queuing Telemetry Transport). Το MQTT είναι ένα ελαφρύ πρωτόκολλο ανταλλαγής μηνυμάτων που έχει σχεδιαστεί με γνώμονα την αποδοτικότητα ως προς τη χωρητικότητα δικτύου, την κατανάλωση ενέργειας και της μνήμης της συσκευής. Υλοποιείται κυρίως πάνω στο Πρωτόκολλο Ελέγχου Μεταφοράς (TCP) [10].

2.4.1 Publish/Subscribe

Το πρωτόκολλο λειτουργεί με βάση το μοντέλο επικοινωνίας Publish/Subscribe. Σε αυτό το μοντέλο υπάρχουν τρία κύρια είδη οντοτήτων: Αποστολείς (Publishers), Παραλήπτες (Subscribers) και ένας Διαμεσολαβητής (Broker). Οι Publishers είναι υπεύθυνοι για την αποστολή μηνυμάτων, οι Subscribers λαμβάνουν μηνύματα και ο Broker είναι ένας κεντρικός

εξυπηρετητής που λαμβάνει όλα τα δημοσιευμένα μηνύματα και στη συνέχεια τα δρομολογεί στους κατάλληλους Subscribers [11].



Σχήμα 2.1: *Αρχιτεκτονική Δικτύου MQTT*

2.4.2 Topic

Τα μηνύματα στο MQTT δημοσιεύονται σε topics (θέματα), τα οποία λειτουργούν ως ιεραρχικές διευθύνσεις τις οποίες οι Subscribers μπορούν να χρησιμοποιούν για να φιλτράρουν και να εγγραφούν σε συγκεκριμένους τύπους μηνυμάτων. Πιο συγκεκριμένα, τα topics είναι συμβολοσειρές που χρησιμοποιούν οι γλιεντς για να καθορίσουν το περιεχόμενο που τους ενδιαφέρει να λάβουν ή να δημοσιεύσουν. Είναι δομημένα ως μια σειρά επιπέδων που χωρίζονται με κάθετους (/), σχηματίζοντας μια δομή που μοιάζει με δέντρο. Για παράδειγμα, ένα όνομα θέματος θα μπορούσε να μοιάζει με "sensors/temperature" ή "home/living_room/lights/status". Αυτή η ιεραρχική δομή επιτρέπει το ευέλικτο και λεπτομερές φιλτράρισμα των μηνυμάτων από τους συνδρομητές. Οι συνδρομητές μπορούν να επιλέξουν να εγγραφούν σε συγκεκριμένα θέματα ή ακόμη και να χρησιμοποιήσουν χαρακτήρες μπαλαντέρ για να αντιστοιχούν σε πολλαπλά θέματα ταυτόχρονα. Υπάρχουν δύο κύριοι χαρακτήρες μπαλαντέρ που χρησιμοποιούνται στο MQTT:

1. **Μπαλαντέρ ενός επιπέδου (+):** Το σύμβολο συν (+) χρησιμοποιείται για την αντιστοίχιση ακριβώς ενός επιπέδου σε μια ιεραρχία θεμάτων. Για παράδειγμα, εάν ένας συνδρομητής εγγραφεί στο "sensors/+", θα λαμβάνει μηνύματα από οποιοδήποτε θέμα που αρχίζει με "sensors/" ακολουθούμενο από ένα επιπλέον επίπεδο, όπως "sensors/temperature" ή "sensors/humidity". Ωστόσο, δεν θα λαμβάνει μηνύματα από θέματα με περισσότερα από ένα πρόσθετα επίπεδα, όπως "sensors/temperature/bedroom".

2. **Μπαλαντέρ πολλαπλών επιπέδων (#):** Το σύμβολο της δίεσης (#) χρησιμοποιείται για την αντιστοίχιση οποιουδήποτε αριθμού επιπέδων σε μια ιεραρχία θεμάτων, συμπεριλαμβανομένου του μηδενός. Όταν ένας συνδρομητής εγγράφεται σε ένα θέμα που περιέχει το μπαλαντέρ πολλαπλών επιπέδων, θα λαμβάνει μηνύματα από αυτό το θέμα και από κάθε θέμα που ταιριάζει με το μοτίβο πριν από το μπαλαντέρ. Για

παράδειγμα, η εγγραφή στο "home/+" θα ταιριάζει με τα "home/living_room", "home/kitchen", και ούτω καθεξής. Ωστόσο, η εγγραφή στο "home/#" θα ταιριάζει με όλα τα θέματα που αρχίζουν με "home/", ανεξάρτητα από το πόσα επίπεδα περιέχουν.

Τα ονόματα θεμάτων παίζουν καθοριστικό ρόλο στην αποτελεσματική και στοχευμένη επικοινωνία μεταξύ των πελατών MQTT, διασφαλίζοντας ότι τα μηνύματα παραδίδονται μόνο στους ενδιαφερόμενους παραλήπτες.

2.4.3 Επίπεδα Quality of Service

Το MQTT υποστηρίζει διαφορετικά επίπεδα ποιότητας υπηρεσίας (Quality of Service) για να ικανοποιεί τις διάφορες απαιτήσεις εφαρμογών:

- **QoS 0:** Παράδοση το πολύ μία φορά. Τα μηνύματα παραδίδονται στον διαμεσολαβητή χωρίς επιβεβαίωση ή εγγυημένη παράδοση στους συνδρομητές.
- **QoS 1:** Παράδοση τουλάχιστον μία φορά. Ο διαμεσολαβητής εξασφαλίζει ότι τα μηνύματα παραδίδονται στους συνδρομητές τουλάχιστον μία φορά, με πιθανό αποτέλεσμα την ύπαρξη διπλών μηνυμάτων.
- **QoS 2:** Παράδοση ακριβώς μία φορά. Αυτό το επίπεδο εξασφαλίζει ότι κάθε μήνυμα παραδίδεται ακριβώς μία φορά στους συνδρομητές, χρησιμοποιώντας μια πιο σύνθετη διαδικασία επιβεβαίωσης και χειραψίας.

Όταν το επίπεδο ποιότητας υπηρεσίας αυξάνεται, η αξιοπιστία της παράδοσης των μηνυμάτων αυξάνεται. Ωστόσο, αυτή η αυξημένη αξιοπιστία επιφέρει πρόσθετη επιβάρυνση από την άποψη του εύρους ζώνης του δικτύου και της επεξεργαστικής ισχύος. Επιπλέον, τα υψηλότερα επίπεδα QoS οδηγούν συνήθως σε αυξημένη καθυστέρηση λόγω των πρόσθετων βημάτων επικοινωνίας και επαλήθευσης μεταξύ αποστολέα και παραλήπτη. Θα πρέπει λοιπόν, η επιλογή επιπέδου QoS να καθορίζεται από την εφαρμογή με βάση τις απαιτήσεις της σε ρυθμό διεκπεραίωσης μηνυμάτων και την ανοχή της σε απώλεια δεδομένων.

2.5 WebSocket

Το WebSocket είναι ένα πρωτόκολλο επικοινωνίας που επιτρέπει την πραγματικού χρόνου, αμφίδρομη επικοινωνία μεταξύ ένος πελάτη και ενός διακομιστή μέσω μίας μονής, μακρόβιας TCP σύνδεσης. Σε αντίθεση με το HTTP, το οποίο ακολουθεί ένα μοντέλο request-response και που απαιτεί την καθιέρωση μιας νέας σύνδεσης για κάθε αίτημα, το WebSocket επιτρέπει την αμφίδρομη ανταλλαγή δεδομένων χωρίς την επιβάρυνση που επιφέρουν τα αιτήματα για επαναλαμβανόμενες συνδέσεις. Αυτό επιτρέπει την αποτελεσματική επικοινωνία χαμηλής καθυστέρησης μεταξύ εφαρμογών και διακομιστών, καθιστώντας το ιδανικό για διαδραστικές εφαρμογές. Με τη δυνατότητα υποστήριξης ασύγχρονων μηνυμάτων και στις δύο κατευθύνσεις, το WebSocket ενισχύει την ανταπόκριση και την διαδραστικότητα των εφαρμογών, παρέχοντας μια απρόσκοπτη εμπειρία χρήστη σε διάφορες συσκευές και πλατφόρμες [12].

Κεφάλαιο 3

Αρχιτεκτονική

Στο κεφάλαιο αυτό γίνεται η περιγραφή της αρχιτεκτονικής του συστήματος και ο διαχωρισμός του στα επιμέρους υποσυστήματα. Επιπλέον, περιγράφονται οι επιλογές σχετικά με το Hardware και το Software της πλατφόρμας.

3.1 Ανάλυση - περιγραφή αρχιτεκτονικής

Η αρχιτεκτονική του συστήματος έχει σχεδιαστεί για την αξιοποίηση των δυνατοτήτων του μοντέλου Edge Computing για την επεξεργασία δεδομένων IoT. Στον πυρήνα της, η αρχιτεκτονική περιλαμβάνει τέσσερα κύρια στοιχεία:

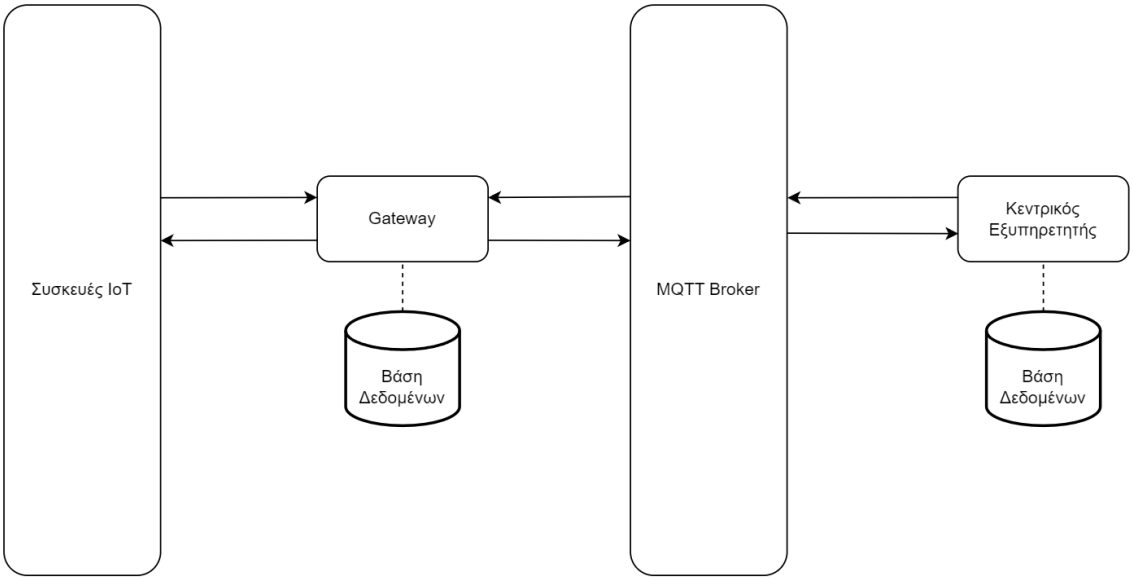
1. **Συσκευές IoT:** Οι συσκευές IoT είναι εγκατεστημένες στην οικία, αποτελούν τις πηγές δεδομένων του συστήματος και έχουν την δυνατότητα να δέχονται εντολές για τον εξ αποστάσεως χειρισμό τους. Κάθε συσκευή IoT συνδέεται με το Gateway της οικίας.

2. **Gateway:** Αυτό το στοιχείο διαδραματίζει κρίσιμο ρόλο στο σύστημα κάνοντας εφικτή την λειτουργία του με βάση το μοντέλο Edge Computing [13]. Είναι εγκατεστημένο σε κάθε οικία και λειτουργεί ως κεντρικό σημείο σύνδεσης όλων των συσκευών IoT. Είναι υπεύθυνο για την απρόσκοπτη επικοινωνία του Κεντρικού Εξυπηρετητή και των διαφορετικών συσκευών IoT, ανεξάρτητα από τα πρωτόκολλα επικοινωνίας ή τις μορφές των παραγόμενων δεδομένων τους. Διαθέτει μια βάση δεδομένων, καθώς είναι υπεύθυνο και για την συλλογή και αποθήκευση των δεδομένων πριν αποσταλλούν στον Κεντρικό Εξυπηρετητή. Ακόμη, έχει μικρή επεξεργαστική ισχύ και συνεπώς μικρή κατανάλωση ρεύματος καθιστώντας όμως δυνατή την εκτέλεση μια αρχικής επεξεργασίας των δεδομένων καθώς και το φιλτράρισμά τους (π.χ. με βάση το είδος της συσκευής ή με βάση την χρονική στιγμή των μετρήσεων). Με αυτόν τον τρόπο, βελτιστοποιείται η διαδικασία μετάδοσης δεδομένων και εξασφαλίζεται η αποδοτική χρήση των πόρων του δικτύου.

3. **MQTT Broker**: Αυτό το στοιχείο είναι υπεύθυνο για την επικοινωνία μεταξύ των Gateways και του Κεντρικού Εξυπηρετητή στο Cloud. Αποτελεί τον κόμβο από τον οποίο περνάνε όλα τα μηνύματα πρωτοκόλλου MQTT που ανταλλάσσονται στο σύστημα.

4. **Κεντρικός Εξυπηρετητής:** Αυτό είναι το στοιχείο της πλατφόρμας με το οποίο αλληλεπιδρά ο τελικός χρήστης. Αναλαμβάνει κυρίως τον ρόλο της αποστολής εντολών στα Gateways. Τέτοιες εντολές μπορεί να είναι εντολές για τον έλεγχο μια συσκεύης ή

ένα αίτημα για αποστολή συγκεκριμένων δεδομένων. Διαθέτει, επίσης, μια βάση δεδομένων στην οποία είναι αποθηκευμένες όλες οι πληροφορίες σχετικά με τις οικίες που διαχειρίζεται η πλατφόρμα, καθώς και πληροφορίες για τα είδη των έξυπνων συσκευών που υποστηρίζονται από αυτή. Επιπλέον, σε αυτή την βάση βρίσκονται αποθηκευμένα και τα δεδομένα που παράγονται από τις εγκατεστημένες έξυπνες συσκευές σε κάθε οικία. Τέλος, προσφέρει μια ιστοσελίδα μέσω της οποίας ο χρήστης μπορεί να ελέγχει τις συσκευές της οικίας του, να ενημερώνεται για την κατάστασή τους και να παρακολουθεί, μέσω γραφικών παραστάσεων, τα δεδομένα που παράγουν οι έξυπνες συσκευές.



Σχήμα 3.1: *Σχηματική αναπαράσταση της Αρχιτεκτονικής*

3.2 Επιλογές Hardware-Software

3.2.1 Συσκευές IoT

Στα πλαίσια της παρούσας Διπλωματικής Εργασίας χρησιμοποιήθηκαν τρεις διαφορετικές συσκεύες.

1. **Shelly Plug S:** Ένας έξυπνος αντάπτορας ρεύματος. Υποστηρίζει μέγιστο φορτίο 12 Ampere και μέγιστη ισχύ 2500 Watt. Προσφέρει την δυνατότητα του εξ αποστάσεως ελέγχου της ροής ρεύματος μέσω ενός διακόπτη. Τα δεδομένα μου συλλέγει περιέχουν πληροφορίες σχετικά με την στιγμιαία ισχύ σε Watt, την συνολική κατανάλωση ενέργειας σε Watt-hours και την εσωτερική θερμοκρασία της συσκεύης σε βαθμούς Κελσίου και βαθμούς Fahrenheit. Ανήκει στην πρώτη γενιά συσκεύων Shelly. [1]

2. **Shelly TRV:** Μια θερμοστατική βαλβίδα θερμαντικού σώματος. Επιτρέπει τον έλεγχο της ροής του νερού μέσα στο θερμαντικό σώμα μέσω μιας βαλβίδας. Ο χρήστης μπορεί να ελέγχει απ' ευθείας την θέση της βαλβίδας (0% έως 100% ανοιχτή) ή να θέσει

[1] https://kb.shelly.cloud/knowledge-base/shelly-plug-s#ShellyPlugS-Specification

Εικόνα 3.1: *Συσκευή Shelly Plug S*

την επιθυμητή θερμοκρασία δωματίου (4°C έως 31°C), με την θέση της βαλβίδας να ελέγχεται πλήρως από την συσκευή. Τροφοδοτείται από μια ενσωματομένη μπαταρία με δυνατότητα επαναφόρτισης μέσω ενός καλωδίου USB-C. Η χωρητικότητά της είναι 6500 mAh με εκτιμώμενη διάρκεια ζωής τα 2 χρόνια. Τα δεδομένα μου συλλέγει περιέχουν πληροφορίες σχετικά με την θέση της βαλβίδας και την στιγμιαία θερμοκρασία του χώρου. Ανήκει στην πρώτη γενιά συσκεύων Shelly. [2]

Εικόνα 3.2: *Συσκευή TRV*

3. **Shelly Plus Plug S:** Τα χαρακτηριστικά αυτής της συσκευής είναι τα ίδια με αυτά της Plug S, με την μόνη διαφορά πως αυτή ανήκει στην δεύτερη γενιά συσκευών Shelly. [3]

Παρόλο που όλες οι συσκεύες είναι κατασκευασμένες από την ίδια εταιρεία, το γεγονός πως ανήκουν σε διαφορετικές γενιές, δημιουργεί προβλήματα ετερογένειας που η πλατφόρμα πρέπει να είναι σε θέση να αντιμετωπίσει.

Όλες οι συσκευές που χρησιμοποιήθηκαν, υποστηρίζουν σύνδεση στο διαδίκτυο μέσω WiFi και υποστηρίζουν το πρωτόκολλο MQTT. Οι λειτουργίες των συσκευών είναι προσπελάσιμες είτε μέσω ενός HTTP API, είτε μέσω μηνυμάτων MQTT. Αξίζει να αναφέρουμε πως

[2] https://www.shelly.com/en-gr/products/product-overview/shelly-trv#node-lx8fbk6d6hx50
[3] https://www.shelly.com/en-gr/products/product-overview/shelly-plus-plug-s#node-28ep5fbiw4drm

Εικόνα 3.3: *Συσκευή Shelly Plus Plug S*

στις συσκευές της πρώτης γενιάς δεν υποστηρίζονται πλήρως όλες οι λειτουργίες μέσω MQTT. Μια αρκετά χρήσιμη προσθήκη στις συσκευές της δεύτερης γενιάς είναι η δυνατότητα παραμετροποίησης των λειτουργιών της συσκευής μέσω μιας scripting γλώσσας βασισμένης στην JavaScript. Ενδεικτικά, προσφέρει πρόσβαση σε όλους τους αισθητήρες της συσκευής, εξερχόμενες συνδέσεις πρωτοκόλλων HTTP και WebSockets καθώς και έναν πλήρη client πρωτοκόλλου MQTT. Η απουσία αυτής της δυνατότητας στις συσκευές της πρώτης γενίας έκανε πιο δύσκολη την ανάπτυξη της πλατφόρμας αλλά αντικατοπτρίζει κάλυτερα την πραγματικότητα του τομέα των οικιακών συσκευών IoT, καθώς πολλές από αυτές δεν είναι παραμετροποιήσιμες.

Για την επικοινωνία με το Gateway αξιοποιήθηκαν και οι δύο δυνατότητες. Οι εντολές για την αλλαγή της κατάστασης κάθε συσκευής χρησιμοποιούν το HTTP API. Αφενός γιατί υποστηρίζεται από όλες τις συσκευές πλήρως και αφετέρου επειδή το μοντέλο αίτηση/απόκριση του πρωτοκόλλου HTTP ταιριάζει καλύτερα στην δεδομένη περίπτωση χρήσης, μιας και ο τελικός χρήστης θα πρέπει να ενημερώνεται αμέσως μετά από κάθε εντολή που στέλνει για την κατάσταση της. Αντιθέτως, η λήψη των δεδομένων των συσκευών γίνεται μέσω MQTT. Το μοντέλο Publish/Subscribe ταιριάζει καλύτερα σε αυτή την περίπτωση, επειδή τα δεδομένα αποστέλλονται στο Gateway την στιγμή που αυτά δημιουργούνται. Με αυτό τον τρόπο αποφεύγονται άσκοπες polling HTTP αιτήσεις που υπερφορτώνουν το δίκτυο και μεταφέρουν πληροφορίες που δεν υπόκεινται σε αλλαγές συχνά.

Τέλος, όλες οι συσκευές υποστηρίζουν το πρωτόκολλο mDNS. Αυτό σημαίνει πως για την επικοινωνία με το Gateway δεν είναι απαραίτητη η αντιστοίχισή τους με την IP που κατέχουν στο τοπικό δίκτυο. Αντ' αυτού η επικοινωνία μπορεί να γίνει με το μοναδικό ID κάθε συσκευής, που λειτουργεί ως hostname στο τοπικό δίκτυο. Έτσι, η απόδοση μιας στατικής διεύθυνσης IP σε κάθε συσκευή δεν είναι απαραίτητη. Το γεγονός αυτό είναι ιδιαίτερα σημαντικό, γιατί η πλατφόρμα απευθύνεται σε χρήστες που ενδέχεται να μην έχουν την απαράιτητη εξοικείωση με τέτοιες ρυθμίσεις. Στον παρακάτω πίνακα συνοψίζονται όλες οι υποστηριζόμενες λειτουργίες για κάθε συσκευή.

Πίνακας 3.1: *Υποστηριζόμενες λειτουργίες κάθε συσκευής*

Λειτουργία	**Shelly Plug S**	**Shelly TRV**	**Shelly Plus Plug S**
WiFi	Ναι	Ναι	Ναι
MQTT	Ναι	Ναι	Ναι
Πλήρης έλεγχος μέσω MQTT	Όχι	Όχι	Ναι
HTTP API	Ναι	Ναι	Ναι
Scripting Γλώσσα	Όχι	Όχι	Ναι
mDNS	Ναι	Ναι	Ναι

3.2.2 Gateway

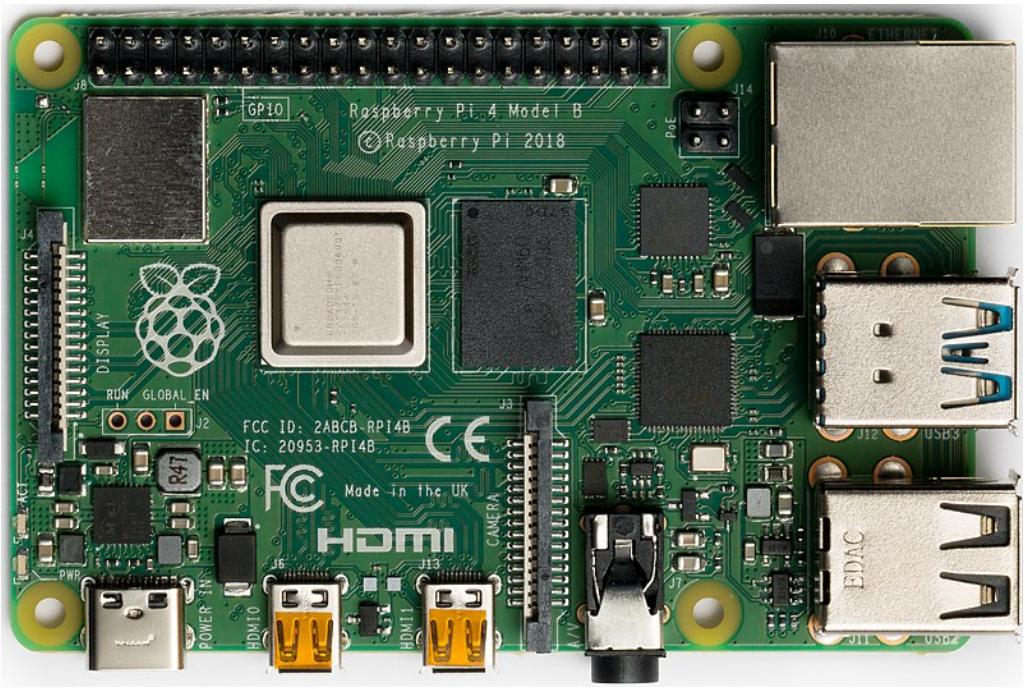
Από άποψη υλικού, για τον ρόλο του Gateway, επιλέχθηκε το Raspberry Pi 4 Model B 2GB[4]. Η κύρια μονάδα επεξεργασίας του Raspberry Pi, το BCM2711, είναι ένα System on Chip (SoC). Αυτό σημαίνει πως όλα τα διάφορα βασικά εξαρτήματα για τη λειτουργία ενός υπολογιστή είναι ενσωματωμένα σε ένα ενιαίο τσιπ, συμπεριλαμβανομένων της CPU (Κεντρική Μονάδα Επεξεργασίας), της GPU (Μονάδα Επεξεργασίας Γραφικών), της RAM (Μνήμη Τυχαίας Πρόσβασης) και άλλων περιφερειακών που απαιτούνται. Αυτή η προσέγγιση οφείλεται για το μικρό μέγεθος (όσο μια πιστωτική κάρτα) και για το χαμηλό κόστος του συγκεκριμένου υπολογιστή (τη στιγμή συγγραφής της παρούσας διπλωματικής εργασίας $\tilde{6}5$ ευρώ). Από άποψη συνδεσιμότητας το Raspberry Pi διαθέτει θύρες USB, έξοδο HDMI για εικόνα, έξοδο ήχου και μια θύρα Ethernet. Ο υπολογιστής τροφοδοτείται από ένα τροφοδοτικό που αποδίδει, ως μέγιστη ισχύ, 15.3 Watt. Αυτή η χαμηλή κατανάλωση ενέργειας το κάνει ιδανικό για την χρήση του ως Gateway, καθώς επιτρέπει την συνεχόμενη λειτουργία του χωρίς ιδιαίτερο κόστος. Το Raspberry Pi είναι σε θέση τα τρέξει τα περισσότερα σύγχρονα λειτουργικά συστήματα όπως MS Windows και διάφορες διανομές Linux. Στα πλαίσια της συγκεκριμένης Εργασίας επιλέχθηκε η διανομή Raspberry Pi OS η οποία είναι ειδικά σχεδιασμένη για να λειτουργεί με τον συγκεκριμένο υπολογιστή [14].

Για την επικοινωνία με τις εγκατεστημένες συσκευές IoT χρησιμοποιήθηκαν τρεις τεχνολογίες. Αρχικά, για την εισροή των παραγόμενων δεδομένων, η οποία όπως προαναφέρθηκε γίνεται μέσω MQTT, αξιοποιήθηκε ο Eclipse Mosquitto καθώς υποστηρίζει πλήρως την έκδοση 3.1.1 του MQTT, έχει μικρό αντίκτυπο στους πόρους του συστήματος, είναι εύκολος ως προς την ρύθμιση και αποτελεί Ελεύθερο Λογισμικό και Λογισμικό Ανοιχτού Κώδικα (FOSS) [15].

Επιπλέον, για την αποστολή εντολών στις συσκευές αναπτύχθηκε ένα API με χρήση της γλώσσας προγραμματισμού Go[5]. Ο ρόλος του είναι να μεταφράζει τα μηνύματα που δέχεται από τον Κεντρικό Εξυπηρετητή σε αιτήματα HTTP και να τα δρομολογεί στην κατάλληλη συσκευή. Παράλληλα, αναλαμβάνει την δρομολόγηση των αποκρίσεων HTTP, καθώς και την αποστολή των συλλεγμένων δεδομένων, πίσω στον Κεντρικό Εξυπηρετητή.

Για τις ανάγκες της αποθήκευσης των δεδομένων που αποστέλλονται στο Gateway χρησιμοποιήθηκε η SQLite3[6]. Πρόκειται για ένα ευέλικτο και αποδοτικό σύστημα διαχείρισης

[4]https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/
[5]https://go.dev
[6]https://www.sqlite.org/



Εικόνα 3.4: *Raspberry Pi 4 Model B*

σχεσιακών βάσεων δεδομένων (RDBMS), βασισμένο στην γλώσσα προγραμματισμού C. Φημίζεται για την απλότητα, τη φορητότητα και την αυτοτελή φύση του. Είναι συμβατό με την γλώσσα SQL και χαρακτηρίζεται από τον ελαφρύ σχεδιασμό του. Σε αντίθεση με τα RDBMS που λειτουργούν με το μοντέλο πελάτη-εξυπηρετητή, η προσπέλαση μιας βάσης δεδομένων SQLite3 δεν απαιτεί την συνεχή εκτέλεση μιας διεργασίας παρασκηνίου, εξοικονομώντας έτσι υπολογιστικούς πόρους. Αντ' αυτού, κάθε εφαρμογή που θέλει να την προσπελάσει, καλεί η ίδια μεθόδους της βιβλιοθήκης SQLite3. Το μέγεθος της βιβλιοθήκης αυτής είναι συνήθως κάτω από 1 MB (διαφέρει ανάλογα την πλατφόρμα). Τα δεδομένα και η δομή μιας βάσης δεδομένων SQLite3 αποθηκεύονται σε ένα μόνο αρχείο. Αυτό επιτρέπει την εύκολη μεταφορά της από μηχάνημα σε μηχάνημα και την φόρτωση ολόκληρης της βάσης στη μνήμη, αυξάνοντας την ταχύτητα προσπέλασης των δεδομένων της. Μερικά αρνητικά αυτού του συστήματος διαχείρισης σχεσιακών βάσεων δεδομένων είναι πως δεν επιτρέπει παραπάνω από μια εντολή εγγραφής σε μια δεδομένη χρονική στιγμή, ενώ δεν διαθέτει έλεγχο ταυτοχρονισμού. Στην υπόθεση χρήσης μας αυτό δεν αποτελεί πρόβλημα, καθώς τα δεδομένα από κάθε συσκευή δημιουργούνται και αποστέλλονται περιοδικά (περίπου κάθε 10 δευτερόλεπτα), ενώ κάθε εντολή εγγραφής διαρκεί μερικά χιλιοστά του δευτερολέπτου. Συνεπώς, δεν υπάρχει ανάγκη παράλληλων εγγραφών. Ένα επιπλέον αρνητικό είναι η απουσία μερικών τύπων δεδομένων που υποστηρίζουν άλλα σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων, που όμως στην υλοποίηση αυτής της πλατφόρμας δεν φάνηκαν χρήσιμοι [16].

Τέλος, για την αξιοποίηση του πρωτοκόλλου mDNS των συσκευών IoT θα χρησιμοποιηθεί το Λογισμικό Ανοιχτού Κώδικα Avahi[7]. Έχει σχεδιαστεί για να είναι ελαφρύ και αποδοτικό, με ελάχιστο αντίκτυπο στους πόρους του συστήματος. Αυτό το καθιστά κατάλληλο για χρήση σε συσκευές με περιορισμένους πόρους, όπως το Raspberry Pi.

[7]https://avahi.org/

3.2.3 MQTT Broker

Από άποψη υλικού, ο MQTT Broker γίνεται ηοστ σε κάποιο μηχάνημα στο Cloud. Και εδώ χρησιμοποιήθηκε ο Eclipse Mosquitto για τους λόγους που αναφέρθηκαν πιο πάνω.

3.2.4 Κεντρικός Εξυπηρετητής

Από άποψη υλικού, ο Κεντρικός Εξυπηρετητής γίνεται ηοστ σε κάποιο μηχάνημα στο ΄λουδ. Για την διαδικτυακή εφαρμογή που προσφέρει στους τελικούς χρήστες χρησιμοποιήθηκε το JavaScript runtime environment NodeJS[8]. Αυτό επιτρέπει την εκτέλεση κώδικα ΘααΣςριπτ - μιας γλώσσας που αρχικά σχεδιάστηκε για να κάνει τις ιστοσελίδες διαδραστικές, επιτρέποντας στους προγραμματιστές να χειρίζονται δυναμικά το DOM (Document Object Model) της HTML και να ανταποκρίνονται σε ενέργειες του χρήστη, όπως κλικ και πληκτρολογήσεις - έξω από ένα πρόγραμμα περιήγησης. Με αυτό τον τρόπο, γίνεται εφικτή η ανάπτυξη εφαρμογών από την πλευρά του σερερ χρησιμοποιώντας την ίδια γλώσσα που χρησιμοποιείται για το client-side scripting. Το NodeJS, επιπλέον, παρέχει ένα πλούσιο σύνολο βιβλιοθηκών και modules, καθιστώντας το κατάλληλο για ένα ευρύ φάσμα εργασιών. Αυτό το γεγονός, καθώς και οι δυνατότητες της ΘααΣςριπτ στις ασύγχρονες διεργασίες υπήρξαν βασικοί λόγοι για τους οποίους επιλέχθηκε το συγκεκριμένο εργαλείο. Μερικές από τις βιβλιοθήκες που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής είναι οι εξής:

- Express.js[9]: Ένα Web Framework που κάνει πιο εύκολη την ανάπτυξη της εφαρμογής και τον χειρισμό των αιτημάτων/αποκρίσεων HTTP.
- MQTT.js[10]: Μία βιβλιοθήκη που προσφέρει λειτουργίες γλιεντ του πρωτοκόλλου MQTT.
- ws[11]: Μία βιβλιοθήκη που προσφέρει λειτουργίες γλιεντ του πρωτοκόλλου WebSocket.
- ejs[12]: Μια βιβλιοθήκη που επιτρέπει την ενσωμάτωση κώδικα JavaScript απευθείας σε αρχεία HTML, καθιστώντας εύκολη τη δημιουργία ιστοσελίδων δυναμικά, με βάση δεδομένα που βρίσκονται στον Εξυπηρετητή.
- Highcharts[13]: Αυτή η βιβλιοθήκη επιτρέπει την δημιουργία αναπαραστάσεων δεδομένων μέσω μια πληθώρας διαφορετικών διαγραμμάτων. Επιπλέον, προσφέρει έναν εύκολο τρόπο για την εξαγωγή των δεδομένων σε μορφή CSV, XLS.

Για την βάση δεδομένων του Εξυπηρετητή χρησιμοποιήθηκε η SQLite3. Πρακτικά, υπάρχει η δυνατότητα να χρησιμοποιηθεί κάποιο πιο πλήρες RDBMS, όπως Postgres[14] ή MariaDB[15], καθώς δεν υπάρχουν οι περιορισμοί σε ενέργεια και υπολογιστική ισχύ που συναντήθηκαν στην περίπτωση του Gateway. Στα πλαίσια όμως της εκπόνησης της Διπλωματικής Εργασίας, επιλέχθηκε η επαναχρησιμοποίηση της SQLite3 για λόγους απλοποίησης

[8] https://nodejs.org/en
[9] https://expressjs.com/
[10] https://www.npmjs.com/package/mqtt
[11] https://www.npmjs.com/package/ws
[12] https://ejs.co/
[13] https://www.highcharts.com/
[14] https://www.postgresql.org/
[15] https://mariadb.org/

της υλοποίησης και με δεδομένο πως δεν δημιούργηθηκε κανένα προβλήματα στην απόδοση του υποσυστήματος.

Κεφάλαιο 4

Υλοποίηση

Στο κεφάλαιο αυτό παρουσιάζεται η παραμετροποίηση των εργαλείων που χρησιμοποιήθηκαν, πληροφορίες που αφορούν τις λειτουργίες και τα παραγόμενα δεδομένα των συσκευών. Περιγράφεται, επιπλέον, αναλυτικά η υλοποίηση των λειτουργιών κάθε επιμέρους στοιχείου της πλατφόρμας.

4.1 Συσκεύες IoT

Σε αυτή την ενότητα παρουσιάζεται ο τρόπος με τον οποίο επιτυγχάνεται η επικοινωνία μεταξύ των συσκευών και του Gateway. Παρουσιάζονται η μορφή των παραγόμενων δεδομένων που αποστέλλονται μέσω MQTT και τα endpoints που προσφέρουν οι συσκευές για τον έλεγχο της κατάστασής τους.

4.1.1 Plug S

- **Δεδομένα μέσω MQTT:** Η συσκευή κάνει Publish μηνύματα που περιέχουν τα δεδομένα των αισθητήρων της σε δύο topics. Τα μηνύματα αποστέλλονται περιοδικά κάθε 10 δευτερόλεπτα ή όταν ανιχνευτεί από τους αισθητήρες της συσκευής μεγάλη μεταβολή στις μετρήσεις τους.

  Αρχικά, αποστέλλει μηνύματα στο topic `shellies/{device_id}/relay/0/energy`.

  Τα μηνύματα αυτά περιέχουν μόνο έναν αριθμό που αναφέρεται στην συνολική κατανάλωση ενέργειας σε Watt-hours.

  Επιπλέον, αποστέλλει μηνύματα στο topic `shellies/{device_id}/relay/0/power`.

  Τα μηνύματα αυτά περιέχουν μόνο έναν αριθμό που αναφέρεται στην στιγμιάια ισχύ σε Watt.

  Όπως γίνεται αντιληπτό, τα δεδομένα δεν περιέχουν πληροφορία για την χρονική στιγμή στην οποία αναφέρονται. Αυτή η έλλειψη θα αντιμετωπιστεί από το Gateway κατά την παραλαβή των μηνυμάτων.

- **HTTP API endpoints:** Η συσκευή προσφέρει την δυνατότητα ελέγχου του διακόπτη ρεύματος, δεχόμενη αιτήσεις HTTP στο endpoint:

```
http://{device_id}.local/relay/0?turn={action}
```

Η μεταβλητή action αναφέρεται στην επιθυμητή θέση του διακόπτη και μπορεί να είναι κάποια από τις τρεις τιμές {on, off, toggle}. Η απόκριση HTTP είναι της μορφής:

```
{"ison": false,
"has_timer":false,
"timer_started_at": 0,
"timer_duration": 0.00,
"timer_remaining": 0.00,
"overpower": false,
"source": "http"}
```

4.1.2 Plug Plus S

Η δυνατότητα ελέγχου του διακόπτη ρεύματος προσφέρεται με τον ίδιο ακριβώς τρόπο με την συσκευή της προηγούμενης γενιάς.

Για την αποστολή των δεδομένων μέσω μέσω MQTT θα γίνει χρήση της scripting γλώσσας που προσφέρει η συσκευή. Με το παρακάτω script υλοποιείται η αποστολή μηνυμάτων που περιέχουν πληροφορίες για την στιγμιαία ισχύ.

```
let id = Shelly.getDeviceInfo().id;
Timer.set(10000, true, function () {
    Shelly.call(
        "switch.getstatus",
        {
            id: 0,
        },
        function (result, error_code, error_message) {
            let unixtime = Shelly.getComponentStatus("sys").unixtime;
            let mqtt_mess = {
                value: result.apower,
                timestamp: unixtime
            };
            MQTT.publish("shellies/"+id+"/relay/0/power", JSON.stringify(mqtt_mess), 0, false);
        }
    );
});
```

Κάθε 10 δευτερόλεπτα το script λαμβάνει την τιμή της στιγμιαίας κατανάλωσης, από τον αισθητήρα καθώς και την χρονική στιγμή στην οποία αναφέρεται ως χρονοσφραγίδα Unix. Στην συνέχεια τα συνθέτει σε μια αναπαράσταση JSON και την αποστέλλει μέσω MQTT. Το όνομα του topic έχει επιλεχθεί ώστε να είναι της ίδιας μορφής με το αντίστοιχο όνομα topic της συσκευής της προηγούμενης γενίας.

Αντίστοιχα, με το παρακάτω script υλοποιείται η αποστολή μηνυμάτων που περιέχουν πληροφορίες για την συνολική κατανάλωση ενέργειας:

```
let id = Shelly.getDeviceInfo().id;
Timer.set(10000, true, function () {
    Shelly.call(
        "switch.getstatus",
        {
            id: 0,
        },
```

```
        function (result, error_code, error_message) {
            let unixtime = Shelly.getComponentStatus("sys").unixtime;
            let mqtt_mess = {
                value: result.aenergy.total,
                timestamp: unixtime
            };
            MQTT.publish("shellies/"+id+"/relay/0/energy", JSON.stringify(mqtt_mess), 0, false);
        }
    );
});
```

Με τα παραπάνω παραδείγματα γίνεται εύκολα αντιληπτή η ευελιξία που προσφέρει η scripting γλώσσα.

4.1.3 TRV

- **Δεδομένα μέσω MQTT:** Η συσκευή κάνει Publish μηνύματα που περιέχουν τα δεδομένα των αισθητήρων της καθώς και των ρυθμίσεων της στο topic `shellies/{device_id}/info`. Τα μηνύματα αποστέλλονται είτε όταν ανιχνευτεί από τους αισθητήρες της συσκευής μεταβολή στις μετρήσεις τους είτε όταν δεχθεί μια νέα εντολή από τον χρήστη. Τα μηνύματα αυτά είναι σε μορφή JSON. Ενδεικτικά:


```
{"wifi_sta":{"connected":true,"ssid":"Kwstas","ip":"192.168.1.4","rssi":-45},
"cloud":{"enabled":false,"connected":false},
"mqtt":{"connected":true},
"time":"03:52",
"unixtime":1715043127,
"serial":0,"has_update":false,
"mac":"60A423DB07A6",
"cfg_changed_cnt":3,"actions_stats":{"skipped":0},
"thermostats":[{"pos":68.9,"target_t":{"enabled":false,"value":22.0,"value_op":8.0,"units":"C"},"
    tmp":{"value":22.7,"units":"C","is_valid":true},"schedule":false,"schedule_profile":1,"
    boost_minutes":0,"window_open":false}],
"calibrated":true,
"bat":{"value":82,"voltage":3.739},
"charger":false,
"update":{"status":"unknown","has_update":false,"new_version":"20240306-093529/v2.2.3@8f5e4729","
    old_version":"20240306-093529/v2.2.3@8f5e4729","beta_version":null},
"ram_total":97280,"ram_free":22488,"fs_size":65536,"fs_free":59564,"uptime":84297,
"fw_info":{"device":"shellytrv-60A423DB07A6","fw":"20240306-093529/v2.2.3@8f5e4729"},
"ps_mode":0,"dbg_flags":0}
```


  Από την πληθώρα των πληροφοριών η πλατφόρμα θα χρησιμοποιήσει την μέτρηση για την θερμοκρασία περιβάλλοντος, το επίπεδο της ενέργειας της μπαταρίας, την επιθυμητή θερμοκρασία περιβάλλοντος που έχει ορίσει ο χρήστης μέσω του θερμοστάτη καθώς και την θέση της βαλβίδας. Η συγκεκριμένη συσκευή, σε αντίθεση με την Plug S που ανήκει στην ίδια γενιά, αποστέλλει μαζί με τα δεδομένα των αισθητήρων της και πληροφορίες για την κατάσταση του συστήματος της συσκευής συμπεριλαμβανομένης και της χρονικής στιγμής που αποστέλλεται το μήνυμα ως χρονοσφραγίδα Unix.

- **HTTP API endpoints:** Αρχικά, η συσκευή προσφέρει την δυνατότητα ελέγχου της

θέσης της βαλβίδας, δεχόμενη αιτήσεις HTTP στο endpoint:

```
http://{device_id}.local/thermostats/0?pos={value}
```

Η μεταβλητή value αναφέρεται στην επιθυμητή θέση της βαλβίδας και μπορεί να είναι οποιοσδήποτε ακέραιος αριθμός στο διάστημα [0, 100]. Η απόκριση HTTP είναι της μορφής:

```
{"pos":50.0,
"target_t":{"enabled":false,"value":11.0,"value_op":8.0,"units":"C"},
"tmp":{"value":22.7,"units":"C","is_valid":true},
"schedule":false,
"schedule_profile":1,
"boost_minutes":0,
"window_open":false}
```

Επιπλέον, η συσκευή προσφέρει την δυνατότητα ελέγχου μέσω θερμοστάτη, δεχόμενη αιτήσεις HTTP στο endpoint:

```
http://{device_id}.local/settings/thermostats/0?target_t={value}
```

Η μεταβλητή value αναφέρεται στην επιθυμητή θερμοκρασία δωματίου σε βαθμούς Κελσίου και μπορεί να είναι οποιοσδήποτε ακέραιος αριθμός στο διάστημα [4, 31]. Η απόκριση HTTP είναι της μορφής:

```
{"target_t":{"enabled":true,"value":22.0,"value_op":8.0,"units":"C","accelerated_heating":true},
"schedule":false,
"schedule_profile":1,
"schedule_profile_names":["Livingroom","Livingroom 1","Bedroom","Bedroom 1","Holiday"],
"schedule_rules":["0600-0123456-23","2300-0123456-18"],
"temperature_offset":0.0,
"ext_t":{"enabled":false, "floor_heating": false},
"t_auto":{"enabled":true},
"boost_minutes":30,
"valve_min_percent":0.00,
"valve_min_report":1.00,
"force_close":false,
"calibration_correction":true,
"extra_pressure":false,
"open_window_report":false}
```

4.2 Gateway

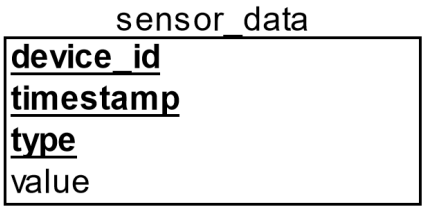
4.2.1 Βάση δεδομένων

Η βάση δεδομένων περιέχει έναν μόνο πίνακα (sensor_data), με 4 στήλες:

- device_id: Το μοναδικό αναγνωριστικό κάθε συσκευής. Παίρνει τιμές τύπου TEXT. Π.χ. shellytrv60A423DB07A6, shellyplusplugs3ce90e2fbe5c
- timestamp: Η χρονοσφραγίδα Unix στην οποία δημιουργήθηκαν τα δεδομένα της εγγραφής. Παίρνει τιμές τύπου INTEGER. Π.χ. 1707948139, 1707948146

- type: Ο τύπος του αισθητήρα στον οποίο αναφέρεται η εγγραφή. Παίρνει τιμές τύπου TEXT. Π.χ. energy, power, temperature
- value: Η τιμή της μέτρησης του αισθητήρα. Παίρνει τιμές τύπου REAL. Π.χ. 4.0, 21.3

Οι στήλες device_id, timestamp και type αποτελούν το πρωτεύον κλειδί του πίνακα. Αυτό σημαίνει πως για να εισέλθει μια καινούργια εγγραφή στον πινάκα θα πρέπει να διαφέρει τουλάχιστον σε ένα από τα τρία αυτά πεδία, από όλες τις άλλες που είναι ήδη καταχωρημένες.



Σχήμα 4.1: *Σχεσιακό διάγραμμα βάσης*

4.2.2 Ingest/Upload scripts

Σε αυτή την υποενότητα παρουσιάζονται συνοπτικά ο κώδικας που αφορά την συλλογή, την αποθήκευση και την αποστολή στο Cloud των παραγόμενων δεδομένων, εστιάζοντας στα σημαντικότερα μέρη τους.

- Ingest Scripts: Ο σκοπός αυτών των scripts είναι η συλλογή των παραγόμενων δεδομένων, η επεξεργασία τους και τέλος η εγγραφή τους στην βάση δεδομένων του Gateway. Υπάρχουν 2 τέτοια scripts (ingest_smart_plugs, ingest_trv) ένα για κάθε τύπο συσκευής. Για αρχή, δημιουργείται μια σύνδεση με τον MQTT Broker του Gateway. Όταν αυτή είναι επιτυχής γίνεται εγγραφή (subscribe) στο κατάλληλο topic. Για την περίπτωση των smart plugs αυτό είναι το shellies/+/relay/0/+ ενώ για την περίπτωση του TRV είναι το shellies/shellytrv-60A423DB07A6/info. Για κάθε μήνυμα που λαμβάνεται, το script πρέπει να το επεξεργαστεί κατάλληλα, να κρατήσει μόνο τις χρήσιμες πληροφορίες που εμπεριέχονται σε αυτό και να προσθέσει, ενδεχομένως, τιμές που απουσιάζουν (π.χ. timestamp, τύπος αισθητήρα, id συσκευής). Για το script που αφορά τις έξυπνες μπρίζες αυτή διαδικασία γίνεται στο παρακάτω κομμάτι κώδικα:

```
local topicParts = splitString(msg.topic, "/")
local device_id = topicParts[2]
local sensor_type = topicParts[5]

if msg.payload:sub(1, 1) == "{" then
    -- Decode JSON payload
    local success, data = pcall(cjson.decode, msg.payload)
    if success then
        local timestamp = data.timestamp
        local sensorValue = tonumber(data.value)
```

```
        insertData(device_id, sensorValue, sensor_type, timestamp)
    else
        print("Failed to decode JSON payload:", data)
    end
else
    local sensorValue = tonumber(msg.payload)
    insertData(device_id, sensorValue, sensor_type)
end
```

Αρχικά, χρησιμοποιώντας την ιεραρχική δομή των ονομάτων των MQTT topics, αντλείται από το όνομα του topic στο οποίο στάλθηκε το μήνυμα, το μοναδικό αναγνωριστικό της συσκευής που το απέστειλε, καθώς και ο τύπος του αισθητήρα που αφορά η μέτρηση. Έπειτα, ελέγχεται αν το περιεχόμενο του μηνύματος είναι απλώς μια αριθμητική τιμή ή είναι σε μορφή JSON. Στην πρώτη περίπτωση το μήνυμα προέρχεται από την έξυπνη μπρίζα της πρώτης γενιάς και δεν περιέχει πληροφορία για την χρονική στιγμή που έγινε η μέτρηση. Ως εκ τούτου καλείται η συνάρτηση που προσθέτει τα δεδομένα στην βάση με τρία ορίσματα, το μοναδικό αναγνωριστικό της συσκευής, την τιμή της μέτρησης και τον τύπο του αισθητήρα. Στην δεύτερη περίπτωση το μήνυμα προέρχεται απο την έξυπνη μπρίζα της δεύτερης γενιάς και περιέχει πληροφορία για την χρονική στιγμή που έγινε η μέτρηση. Αφού αποκωδικοποιηθεί η αναπαράσταση JSON των δεδομένων και εξαχθεί η τιμή της μέτρησης και η χρονική στιγμή που έγινε, καλείται η συνάρτηση που προσθέτει τα δεδομένα στην βάση με τέσσερα ορίσματα, το μοναδικό αναγνωριστικό της συσκευής, την τιμή της μέτρησης, τον τύπο του αισθητήρα και την χρονική στιγμή της μέτρησης ως χρονοσφραγίδα Unix.

Το script που αφορά αυτή τη διαδικασία της θερμοστατικής βαλβίδας παρατίθεται στο ακόλουθο κομμάτι κώδικα:

```
local success, data = pcall(cjson.decode, msg.payload)
if success then
    local timestamp = data.unixtime
    local batteryLevelValue = tonumber(data.bat.value)
    local currentTempValue = tonumber(data.thermostats[1].tmp.value)
    local valvePositionValue = tonumber(data.thermostats[1].pos)
    local targetTempValue = tonumber(data.thermostats[1].target_t.value)

    insertData('shellytrv-60A423DB07A6', batteryLevelValue, 'battery percentage', timestamp)
    insertData('shellytrv-60A423DB07A6', currentTempValue, 'temperature', timestamp)
    insertData('shellytrv-60A423DB07A6', targetTempValue, 'target temperature', timestamp)
    insertData('shellytrv-60A423DB07A6', valvePositionValue, 'valve position', timestamp)

else
    print("Failed to decode JSON payload:", data)
end
```

Αρχικά, γίνεται η αποκωδικοποίηση της αναπαράστασης JSON των δεδομένων και εξάγονται οι πληροφορίες που αφορούν τις ζητούμενες μετρήσεις, όπως η στάθμη της μπαταρίας, η θερμοκρασία δωματίου, η επιθυμητή θερμοκρασία δωματίου, η θέση της βαλβίδας, καθώς και η χρονική στιγμή που έγιναν οι παραπάνω μετρήσεις. Ύστερα,

για κάθε μία από αυτές καλείται η συνάρτηση που προσθέτει τα δεδομένα στην βάση με τέσσερα ορίσματα, το μοναδικό αναγνωριστικό της συσκευής, την τιμή της μέτρησης, τον τύπο του αισθητήρα και την χρονική στιγμή της μέτρησης ως χρονοσφραγίδα Unix.

Η συνάρτηση που αναλαμβάνει την προσθήκη των δεδομένων στην βάση είναι κοινή και για τα 2 scripts και παρουσιάζεται παρακάτω:

```
function insertData(device_id, value, type, --[[optional]] timestamp)
    timestamp = timestamp or os.time()
    local success, error_code = con:execute(string.format(
        "INSERT INTO sensor_data (device_id, timestamp, value, type) VALUES ('%s', %d, %f, '%s')",
        device_id,
        timestamp,
        value, type))

    if not success then
        print("Error executing SQL statement. Error code:", error_code)
    end
end
```

Το τέταρτο όρισμα, δηλαδή η χρονοσφραγίδα, είναι προαιρετικό. Αν δεν υπάρχει η συνάρτηση χρησιμοποιεί την τωρινή χρονοσφραγίδα του συστήματος ως την χρονοσφραγίδα της μέτρησης θεωρώντας, δηλαδή, πως η μέτρηση αναφέρεται στην τωρινή χρονική στιγμή. Μια τέτοια θεώρηση έχει αρκέτα ικανοποιητική ακρίβεια, καθώς δεν υπάρχει ιδιαίτερη καθυστερήση από την στιγμή που σταλθεί το MQTT μήνυμα από την συσκευή μέχρι την λήψη του από το Gateway. Σε κάθε περίπτωση, οι μικρές αποκλίσεις που τυχόν παρατηρηθούν είναι αποδεκτές για την περίπτωση χρήσης που εξετάζεται.

- Upload Script: Ο σκοπός του συγκεκριμένου script είναι η μεταφόρτωση των δεδομένων που είναι αποθηκευμένα στο Gateway, στην βάση δεδομένων του Κεντρικού Εξυπηρετητή. Αρχικά, γίνεται η σύνδεση με την βάση και διαβάζονται όλα τα δεδομένα που έχουν χρονοσφραγίδα ως και μία ώρα πίσω από την τωρινή χρονική στιγμή. Στην συνέχεια, τα δεδομένα σειριοποιούνται σε μορφή JSON και αποστέλλονται στον Κεντρικό Εξυπηρετητή μέσω MQTT. Για κάθε εγγραφή στη βάση δεδομένων αποστέλλεται και ένα μήνυμα MQTT. Το script εκτελείται περιοδικά, μέσω ενός cronjob, κάθε μία ώρα. Επίσης εκτελείται όποτε ο χρήστης ζητήσει πληροφορίες σχετικά με τις μετρήσεις μιας συσκευής από τον Κεντρικό Εξυπηρετητή.

4.2.3 MQTT API

Το συγκεκριμένου API είναι υπεύθυνο για την αμφίδρομη επικοινωνία μεταξύ των έξυπνων συσκευών και του Κεντρικού Εξυπηρετητή. Η επικοινωνία βασίζεται εξ ολοκλήρου στο πρωτόκολλο MQTT.

Το API φροντίζει να είναι μόνιμα συνδεδεμένο με τον MQTT Broker του Cloud. Μετά από μια επιτυχή σύνδεση γίνεται εγγραφή στο MQTT topic `{gateway_uuid}/+`. Το όνομα του topic αποτελείται από 2 επίπεδα ιεραρχίας. Το πρώτο αναφέρεται στο μοναδικό αναγνωριστικό κάθε Gateway και αποσκοπεί στην ορθή δρομολόγηση των αιτήματων. Το δεύτερο είναι ένα

μοναδικό αναγνωριστικό αίτησης, που αποσκοπεί στην σωστή δρομολόγηση της απάντησης. Το όνομα του topic στο οποίο στέλνεται η απάντηση είναι το `{gateway_uuid}/{request_uuid}/res`. Υπάρχουν δύο διαφορετικά είδη μηνυμάτων. Το πρώτο είδος αποτελούν τα μηνύματα που αιτούνται την μεταφορά των δεδομένων της βάσης του Gateway. Το payload αυτών των μηνυμάτων είναι η συμβολοσειρά "sendData". Κατά την λήψη ενός τέτοιου μηνύματος εκτελείται το Upload Script που περιγράφηκε παραπάνω. Το δεύτερο είδος μηνυμάτων είναι τα μηνύματα που αιτούνται την εκτέλεση ενός HTTP αιτήματος. Το payload αυτών των μηνυμάτων είναι ένας σύνδεσμος URL. Για παράδειγμα, το payload ενός τέτοιου μηνύματος μπορεί να είναι `"http://shellyplusplugs-3ce90e2fbe5c.local/relay/0?turn=toggle"`. Όπως προαναφέρθηκε, όλα τα URL κάνουν χρήση του πρωτοκόλλου mDNS. Κατά την λήψη ενός τέτοιου μηνύματος, εκτελείται το ζητούμενο HTTP αίτημα. Αν ο κωδικός της απόκρισης επιβεβαιώνει την ορθή εκτέλεση του, αποστέλλεται στο response MQTT topic το σώμα της HTTP απόκρισης. Διαφορετικά, δεν αποστέλλεται καμία απάντηση. Ενδεικτικά, παρατίθεται η συνάρτηση που εκτελείται κατά την λήψη ενός μηνύματος:

```
func onMessageReceived(client mqtt.Client, message mqtt.Message) {
  responseTopic := message.Topic() + "/res"
  payload := string(message.Payload())
  fmt.Printf("Received message: %s\n", payload)
  if payload == "sendData" {
    cmd := exec.Command("/home/kostas/edge-pi-scripts/upload_data.lua")
    cmd.Stdout = os.Stdout
    cmd.Stderr = os.Stderr

    err := cmd.Run()
    if err != nil {
      fmt.Println("Error executing Lua script:", err)
      os.Exit(1)
    }
  } else {
    urlToHit := payload

    // Make an HTTP request
    response, err := http.Get(urlToHit)
    if err != nil {
      fmt.Printf("Error making HTTP request: %v\n", err)
      return
    }
    defer response.Body.Close()

    if response.StatusCode == http.StatusOK {
      fmt.Println("HTTP request successful")
      b, err := io.ReadAll(response.Body)
      if err != nil {
        fmt.Printf("Error reading HTTP response body: %v\n", err)
        return
      }
      fmt.Println(string(b))

      // Send a response message
```

```
      client.Publish(responseTopic, 0, false, string(b))
    } else {
      fmt.Printf("HTTP request failed with status code %d\n", response.StatusCode)
    }
  }
}
```

4.2.4 Websocket API

Ο σκοπός του συγκεκριμένου API είναι ίδιος με αυτόν του MQTT API. Η διαφορά στην συγκεκριμένη περίπτωση είναι ότι η επικοινωνία βασίζεται εν μέρει στο MQTT και κυρίως στο πρωτόκολλο WebSocket. Το API φροντίζει να είναι μόνιμα συνδεδεμένο με τον MQTT Broker του Cloud. Μετά από μια επιτυχή σύνδεση γίνεται εγγραφή στο MQTT topic `{gateway_uuid}`. Το όνομα του topic, τώρα, αποτελείται μόνο από ένα επίπεδο ιεραρχίας, το οποίο αναφέρεται στο μοναδικό αναγνωριστικό κάθε Gateway. Τα είδη των μηνυμάτων είναι τα ίδια με αυτά στο MQTT API. Επιπλέον, ίδιος παραμένει και ο τρόπος χειρισμού των μηνυμάτων που αιτούνται την μεταφορά των δεδομένων της βάσης του Gateway. Η προσέγγιση χειρισμού, όμως, των μηνυμάτων που αιτούνται την εκτέλεση ενός HTTP αιτήματος διαφέρει. Αρχικά, ο Κεντρικός Εξυπηρετητής στέλνει ένα μήνυμα μέσω MQTT. Το payload αυτών των μηνυμάτων είναι ένα μοναδικό αναγνωριστικό της αίτησης. Κατά την λήψη ένος τέτοιου μηνύματος, το API θα δημιουργήσει μια αμφίδρομη σύνδεση πρωτοκόλλου WebSocket με τον Κεντρικό Εξυπηρετητή, χρησιμοποιώντας το μοναδικό αναγνωριστικό της αίτησης. Δηλαδή, κάθε τέτοια σύνδεση είναι μοναδική και αφορά μία μόνο αίτηση. Στην συνέχεια, ο Κεντρικός Εξυπηρετητής στέλνει μέσω αυτής της σύνδεσης τον σύνδεσμο URL στον οποίο επιθυμεί να γίνει το HTTP αίτημα. Το API εκτελεί αυτό το αίτημα και αν ο κωδικός της απόκρισης επιβεβαιώνει την ορθή εκτέλεση του, αποστέλλει μέσω της ανοιχτής WebSocket σύνδεσης το σώμα της HTTP απόκρισης. Διαφορετικά, δεν αποστέλλεται καμία απάντηση. Σε κάθε περίπτωση, το API κλείνει την σύνδεση. Ενδεικτικά, παρατίθεται η συνάρτηση που εκτελείται κατά την λήψη ενός μηνύματος:

```
func onMessageReceived(client mqtt.Client, message mqtt.Message) {
  payload := string(message.Payload())
  fmt.Printf("Received message: %s\n", payload)
  if payload == "sendData" {
    cmd := exec.Command("/home/kostas/edge-pi-scripts/upload_data.lua")
    cmd.Stdout = os.Stdout
    cmd.Stderr = os.Stderr

    err := cmd.Run()
    if err != nil {
      fmt.Println("Error executing Lua script:", err)
      os.Exit(1)
    }
  } else {
    ctx, cancel := context.WithTimeout(context.Background(), time.Minute)
    defer cancel()
    c, _, err := websocket.Dial(ctx, "ws://192.168.1.2:8080/"+payload, nil)
    if err != nil {
```

```
      log.Fatal(err)
    }
    defer c.CloseNow()
    _, msg, err := c.Read(ctx)
    msgString := string(msg[:])
    if err != nil {
      log.Fatal(err)
    }
    fmt.Println(msgString)
    urlToHit := msgString

    // Make an HTTP request
    response, err := http.Get(urlToHit)
    if err != nil {
      fmt.Printf("Error making HTTP request: %v\n", err)
      return
    }
    defer response.Body.Close()

    if response.StatusCode == http.StatusOK {
      fmt.Println("HTTP request successful")
      b, err := io.ReadAll(response.Body)
      if err != nil {
        fmt.Printf("Error reading HTTP response body: %v\n", err)
        return
      }
      fmt.Println(string(b))

      //Send a response message
      err = c.Write(ctx, websocket.MessageText, b)
      if err != nil {
        fmt.Println("Error sending response message", err)
      }
    } else {
      fmt.Printf("HTTP request failed with status code %d\n", response.StatusCode)
    }
    c.Close(websocket.StatusNormalClosure, "")
  }
}
```

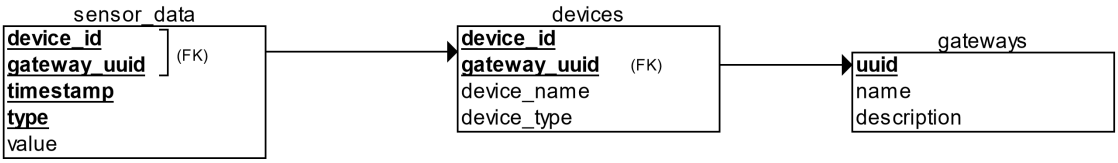
4.3 Κεντρικός Εξυπηρετητής

4.3.1 Βάση δεδομένων

Η βάση δεδομένων του Κεντρικού Εξυπηρετητή περιέχει τρείς πίνακες:

1. gateways: Στον συγκεκριμένο πίνακα αποθηκεύονται πληροφορίες σχετικά με τις οικίες που είναι εγγεγραμένες στην πλατφόρμα. Ο πίνακας περιέχει 3 στήλες.

    - uuid: Το μοναδικό αναγνωριστικό κάθε Gateway. Κάθε οικία έχει ένα μόνο Gateway και κάθε Gateway βρίσκεται σε μόνο μία οικία. Άρα αυτό το πεδίο



Σχήμα 4.2: *Σχεσιακό διάγραμμα βάσης Κεντρικού Εξυπηρετητή*

χαρακτηρίζει μοναδικά κάθε οικία. Παίρνει τιμές τύπου TEXT. Π.χ. 2dec771f-091a-409b-ab4c-25174904f6da

- name: Το όνομα που έχει δώσει ο χρήστης στην οικία. Παίρνει τιμές τύπου TEXT. Π.χ. Εξοχικό
- description: Λοιπές πληροφορίες σχετικά με την οικία που έχει συμπεριλάβει ο χρήστης. Παίρνει τιμές τύπου TEXT.

Η στήλη uuid αποτελεί το πρωτεύον κλειδί του πίνακα.

2. devices: Στον συγκεκριμένο πίνακα αποθηκεύονται πληροφορίες σχετικά με τις έξυπνες συσκευές που είναι εγκατεστημένες σε κάθε οικία. Ο πίνακας περιέχει 4 στήλες.

   - device_id: Το μοναδικό αναγνωριστικό κάθε συσκευής. Παίρνει τιμές τύπου TEXT. Π.χ. shellytrv60A423DB07A6, shellyplusplugs3ce90e2fbe5c
   - device_name: Το όνομα που έχει δώσει ο χρήστης στην συσκευή. Παίρνει τιμές τύπου TEXT. Π.χ. Μπρίζα σαλόνι
   - device_type: Ο τύπος της συσκευής. Παίρνει τιμές τύπου TEXT. Π.χ. plug, trv
   - gateway_uuid: Το μοναδικό αναγνωριστικό της οικίας στην οποία είναι εγκατεστημένη η συσκευή.

   Οι στήλες device_id, gateway_uuid αποτελούν το πρωτεύον κλειδί του πίνακα. Η στήλη gateway_uuid είναι foreign key και αναφέρεται στην στήλη uuid του πίνακα gateways.

3. sensor_data: Στον συγκεκριμένο πίνακα αποθηκεύονται τα δεδομένα που συλλέγονται από τις συσκευές. Ο πίνακας περιέχει 5 στήλες. Είναι πανομοιότυπος με τον πίνακα στην βάση δεδομένων του Gateway με την διαφορά ότι εδώ προστίθεται και η στήλη gateway_uiid. Οι στήλες device_id, gateway_uuid, timestamp και type αποτελούν το πρωτεύον κλειδί του πίνακα. Οι στήλες device_id και gateway_uuid αποτελούν foreign key που αναφέρεται στις αντίστοιχες στήλες του πίνακα devices.

4.3.2 Δικτυακή Εφαρμογή (Frontend)

Η δικτυακή εφαρμογή προσφέρεται ως ο βασικός τρόπος αλληλεπίδρασης του χρήστη με την Πλατφόρμα. Αποτελείται από τέσσερεις σελίδες:

1. Αρχική Σελίδα: Σε αυτή τη σελίδα παρουσιάζονται οι πληροφορίες σχετικά με τις οικίες που είναι εγγεγραμμένες στην Πλατφόρμα. Ο χρήστης επιλέγει την οικία με την οποία θέλει να αλληλεπιδράσει και οδηγείται στην επόμενη σελίδα. Το αρχείο που αντιστοιχεί σε αυτή την σελίδα είναι το `landing_page.ejs`.

2. Συσκευές: Σε αυτή τη σελίδα παρουσιάζονται όλες οι έξυπνες συσκευές που είναι εγκατεστημένες στην επιλεγμένη οικία. Ο χρήστης έχει την δυνατότητα να κατευθυνθεί, είτε στην σελίδα ελέγχου μιας συσκευής είτε στη σελίδα όπου παρουσιάζονται οι μετρήσεις των αισθητήρων της. Το αρχείο που αντιστοιχεί σε αυτή την σελίδα είναι το `devices.ejs`

3. Έλεγχος συσκευής: Σε αυτή τη σελίδα ο χρήστης μπορεί να ελέγχει την επιλεγμένη συσκευή. Κάθε διαφορετικός τύπος συσκευών έχει διαφορετική σελίδα ελέγχου. Οι έξυπνες μπρίζες δεν διαθέτουν σελίδα ελέγχου καθώς η μόνη λειτουργία που υποστηρίζουν είναι το άνοιγμα και κλείσιμο του διακόπτη. Αυτή η λειτουργία προσφέρεται από την σελίδα συσκευών. Το αρχείο που αντιστοιχεί σε αυτή την σελίδα είναι το `trv_controls.ejs`

4. Δεδομένα συσκευής: Σε αυτή τη σελίδα ο χρήστης παρακολουθεί τα δεδομένα που αποστέλλονται στην πλατφόρμα από την συσκευή. Και εδώ, κάθε διαφορετικός τύπος συσκευών έχει διαφορετική σελίδα δεδομένων. Επίσης, μέσω αυτής της σελίδας ο χρήστης έχει την δυνατότητα να κατεβάσει τα δεδομένα που αντιστοιχούν στην επιλεγμένη συσκευή. Τα αρχεία που αντιστοιχούν σε αυτή την σελίδα είναι τα `trv_data.ejs`, `plug_data.ejs`

Ο τρόπος με τον οποίο γίνονται render οι σελίδες, καθώς και τα δεδομένα που απαιτούνται για να υλοποιηθεί αυτό παρουσιάζονται στην επόμενη υποενότητα.

4.3.3 Backend API Πλατφόρμας (MQTT)

Για την υποστήριξη των λειτουργιών της δικτυακής εφαρμογής αναπτύχθηκε ένα backend API. Σε αυτή την υποενότητα περιγράφεται η υλοποίηση του, στην περίπτωση κατά την οποία η επικοινωνία μεταξύ των έξυπνων σπιτιών και του Κεντρικού Εξυπρετητή γίνεται αποκλειστικά μέσω MQTT.

Αρχικά, το API φροντίζει ώστε να διατηρείται μια μόνιμη σύνδεση με τον MQTT Broker που βρίσκεται στο Cloud. Μετά από μια επιτυχή σύνδεση, γίνεται η εγγραφή στο MQTT Topic `download_data`. Σε αυτό το topic αποστέλλονται από κάθε Gateway τα δεδομένα των μετρήσεων των έξυπνων συσκευών. Το περιεχόμενο ενός τέτοιου μηνύματος έχει την ακόλουθη μορφή:

```
{"device_id":"shellyplug-s-C1C6FE",
"timestamp":1719070733,
"gateway_uuid":"fc04af0f-1a6a-49b9-aa81-95ef25296b44",
"type":"power",
"value":21.19}
```

Κατά την λήψη του, αποσειριοποιείται το περιεχόμενο του μηνύματος και στην συνέχεια γίνεται η εισαγωγή των δεδομένων στην βάση του Κεντρικού Εξυπηρετητή. Η συνάρτηση που εκτελείται κατά την λήψη ενός μηνύματος είναι η παρακάτω:

```
client.on('message', (topic, message) => {
    if (topic == 'download_data') {
        const data = JSON.parse(message.toString());
        db.run('INSERT INTO sensor_data (device_id, value, timestamp, type, gateway_uuid) VALUES (?, ?,
     ?, ?, ?)', [data.device_id, data.value, data.timestamp, data.type, data.gateway_uuid], (err) => {
```

```
            if (err) {
                console.error(err);
            } else {
                console.log('Data inserted');
            }
        });
    }
});
```

Τα routes του API είναι:

- '/': Υποστηρίζει την μέθοδο HTTP GET. Αρχικά, διαβάζει από την βάση δεδομένων πληροφορίες σχετικά με τις οικίες που είναι εγγεγραμμένες στην πλατφόρμα, χρησιμοποιώντας τον πίνακα gateways. Στη συνέχεια, κάνει render την αρχική σελίδα της δικτυακής εφαρμογής με βάση αυτές.

```
app.get('/', (req, res) => {
    db.all('SELECT * FROM gateways', (err, rows) => {
        if (err) {
            console.error(err);
            res.status(500).send('Internal Server Error');
        } else {
            res.render('landing_page', { data: rows });
        }
    });
});
```

- '/devices': Υποστηρίζει την μέθοδο HTTP GET. Δέχεται το query parameter gateway που είναι το μοναδικό αναγνωριστικό κάθε Gateway/οικίας. Αρχικά, διαβάζει από την βάση δεδομένων πληροφορίες σχετικά με τις συσκευές που είναι εγκατεστημένες στην οικία, χρησιμοποιώντας τον πίνακα devices. Στη συνέχεια, κάνει render την σελίδα Devices της δικτυακής εφαρμογής με βάση αυτές.

```
app.get('/devices', (req, res) => {
    const gatewayId = req.query.gateway
    // Fetch data from the database
    db.all("SELECT * from devices where gateway_uuid = ?", [gatewayId], (err, rows) => {
        if (err) {
            console.error(err);
            res.status(500).send('Internal Server Error');
        } else {
            // Render the landing page with the fetched data
            res.render('devices', { devices: rows });
        }
    });
});
```

- '/device_data': Υποστηρίζει την μέθοδο HTTP GET. Τα query parameters που δέχεται είναι το μοναδικό αναγνωριστικό κάθε Gateway/οικίας, το μοναδικό αναγνωριστικό της συσκευής και ο τύπος της συσκευής. Αρχικά, διαβάζει από την βάση δεδομένων πληροφορίες σχετικά με τις μετρήσεις των αισθητήρων της συσκευής, προσπελάζοντας

τον πίνακα sensor_data. Στην συνέχεια, κάνει render την σελίδα Device Data που ανιστοιχεί στον τύπο της συσκευής με βάση αυτές τις πληροφορίες.

```
app.get('/device_data', (req, res) => {
    const gatewayId = req.query.gateway_uuid
    const deviceId = req.query.device_id
    const deviceType = req.query.device_type
    db.all("SELECT type, json_group_array(json_object('value', sensor_data.value, 'timestamp',
     sensor_data.timestamp)) as data FROM sensor_data WHERE gateway_uuid = ? AND device_id = ? AND
      timestamp >= unixepoch('now', '-1 days') AND timestamp <= unixepoch('now') GROUP BY type ",
     [gatewayId, deviceId], (err, rows) => {
        if (err) {
            console.error(err);
            res.status(500).send('Internal Server Error');
        } else {
            data_object = {}
            rows.forEach(row => {
                row.data = JSON.parse(row.data)
                data_object[row.type] = row.data
            })
            res.render(`${deviceType}_data`, {
                data_object: data_object
            });
        }
    });
});
```

- '/device_controls': Υποστηρίζει την μέθοδο HTTP GET. Τα query parameters που δέχεται είναι το μοναδικό αναγνωριστικό κάθε Gateway/οικίας, το μοναδικό αναγνωριστικό της συσκευής και ο τύπος της συσκευής. Κάνει render την σελίδα Device Controls που ανιστοιχεί στον τύπο της συσκευής με βάση τα query parameters.

```
app.get('/device_controls', (req, res) => {
    const deviceType = req.query.device_type
    res.render(`${deviceType}_controls`, { gateway_uuid: req.query.gateway_uuid, device_id: req.
     query.device_id });
});
```

- '/mqtt_to_http': Μέσω αυτου του route δίνεται μια εντολή σε ένα Gateway και επιστρέφεται η απόκρισή του. Υποστηρίζει την μέθοδο HTTP POST. Το API αναμένει η πληροφορία που εμπεριέχεται στο body της αίτησης POST να είναι της μορφής application/json. Ειδικότερα, το JSON που αποστέλλεται θα πρέπει να έχει δύο πεδία. Το πεδίο command είναι η εντολή που θα πρέπει να εκτελεστεί από το Gateway. Μπορεί να είναι είτε το URL στο οποίο θα γίνει μια HTTP αίτηση, είτε η συμβολοσειρά "sendData". Το gateway_uuid είναι το μοναδικό αναγνωριστικό του Gateway και παράλληλα, όπως προαναφέρθηκε, το όνομα του MQTT topic στο οποίο πρέπει να κατευθυνθεί η εντολή. Το API, αφού εξάγει τις παραπάνω πληροφορίες από το body της αίτησης, δημιουργεί ένα μοναδικό αναγνωριστικό (req_uuid) για την αίτηση. Επιπλέον, στέλνει ένα MQTT μήνυμα με payload το περιεχόμενο του πεδίου command στο topic `{gateway_uuid}/{req_uuid}`. Αν το περιεχόμενο αυτό είναι η συμβολοσειρά "sendData", τότε

αποκρίνεται με HTTP status code 200. Σε αντίθετη περίπτωση θα πρέπει να επιστραφεί η απόκριση της έξυπνης συσκευής στο αίτημα που εμπεριέχεται στο πεδίο command. Για να γίνει αυτό, το API εγγράφεται στο MQTT topic `{gateway_uuid}/{req_uuid}/res`, αναμένοντας την απόκριση στην εντολή. Αν ληφθεί ένα μήνυμα στο response topic μέσα σε διάστημα 5 δευτερολέπτων από την εγγραφή, η εντολή θεωρείται πως έχει επιτύχει και το route αποκρίνεται με HTTP status code 200, ενσωματώνοντας το payload του μηνύματος στο body της HTTP απόκρισης. Σε αντίθετη περίπτωση, η εντολή θεωρείται ότι έχει αποτύχει και το route αποκρίνεται με HTTP status code 500.

```
app.post('/mqtt_to_http', (req, res) => {
    const topic = `${req.body.gateway_uuid}/${uuidv4()}`
    const message = req.body.command
    const route_client = mqtt.connect(mqtt_server_url);

    if (message != 'sendData') {
        // Set the timeout for the response
        const timeout = 5000; // 5 seconds
        res.setTimeout(timeout, () => {
            res.status(500).send('Timeout Error');
            route_client.end();
        });
        route_client.on('message', (response_topic, message) => {
            if (`${topic}/res` == response_topic) {
                res.send(message.toString());
                route_client.end();
            }
        });
        route_client.subscribe(`${topic}/res`);
        route_client.publish(topic, message);
    }

    else {
        route_client.publish(topic, message);
        res.send('Request sent');
        route_client.end();
    }
});
```

4.3.4 Backend API Πλατφόρμας (WebSocket)

Σε αυτή την υποενότητα περιγράφεται η υλοποίηση του Backend API, όταν η επικοινωνία μεταξύ των έξυπνων σπιτιών και του Κεντρικού Εξυπρετητή γίνεται κυρίως μέσω WebSocket, εστιάζοντας στις διαφορές μεταξύ αυτής και της προσέγγισης που περιγράφηκε στην προηγούμενη υποενότητα.

Η πρώτη διαφορά είναι πως προστίθεται στις λειτουργίες του API η δυνατότητα εξυπηρέτησης αιτήσεων WebSocket.

Η δεύτερη διαφορά έγκειται στον τρόπο με τον οποίο χειρίζονται οι HTTP αιτήσεις στο route '/mqtt_to_http'. Αν το περιεχόμενο του πεδίου command είναι η συμβολοσειρά "sendData", τότε στέλνει ένα MQTT μήνυμα με payload τη συμβολοσειρά στο topic `{gateway_uuid}`

και αποκρίνεται με HTTP status code 200. Σε αντίθετη περίπτωση, δηλαδή αν το περιεχόμενο είναι ένα URL, δημιουργεί ένα μοναδικό αναγνωριστικό για την αίτηση (req_uuid), το ενσωματώνει σε ένα MQTT μήνυμα και το στέλνει στο topic `{gateway_uuid}`. Παράλληλα, δημιουργεί μια εγγραφή στην μνήμη με κλειδί το μοναδικό αναγνωριστικό της αίτησης και τιμή το περιεχόμενο του πεδίου command και το route response object. Το τελευταίο χρησιμοποιείται για να δοθεί η απόκριση από διαφορετικό σημείο του κώδικα. Μέρος αυτής της διαδικασίας περιγράφεται στον παρακάτω κώδικα:

```
app.post('/mqtt_to_http', (req, res) => {
    const topic = req.body.gateway_uuid
    const message = req.body.url
    if (message == 'sendData') {
        client.publish(topic, message);
        res.send('Request sent');
    }
    else {
        const request_uuid = uuidv4()
        res.setTimeout(5000, () => {
            res.status(500).send('Timeout Error');
            delete inflight_http_requests[request_uuid]
        });
        inflight_http_requests[request_uuid] = {
            message: message,
            res: res
        }
        client.publish(topic, request_uuid);
    }
});
```

Στη συνέχεια, το API αναμένει μια νέα σύνδεση WebSocket με αναγνωριστικό το `{req_uuid}`. Αν αυτό συμβεί, στέλνει μέσω αυτής της νέας σύνδεσης την τιμή της εγγραφής με κλειδί `{req_uuid}` και αναμένει στην σύνδεση την απάντηση από το Gateway. Όταν την λάβει, διαγράφει την εγγραφή της αίτησης με το αναγνωριστικό `{req_uuid}` και αποκρίνεται με HTTP status code 200, ενσωματώνοντας την απάντηση στο body της HTTP απόκρισης. Η παραπάνω διαδικασία περιγράφεται στον παρακάτω κώδικα:

```
wss.on('connection', function connection(ws, http_request) {
    request_uuid = http_request.url.split('/')[1]
    ws.on('error', console.error);
    // When the gateway sends a message, send it to the client, if the request is still alive.
    ws.on('message', function message(data) {
        if (inflight_http_requests.hasOwnProperty(request_uuid)) {
            inflight_http_requests[request_uuid].res.send(data)
            delete inflight_http_requests[request_uuid]
        }
    });
    // If the request is still alive, send the message to the gateway
    if (inflight_http_requests.hasOwnProperty(request_uuid)) {
        ws.send(inflight_http_requests[request_uuid].message)
    }
});
```

Αν η παραπάνω διαδικασία δεν ολοκληρωθεί μέσα σε 5 δευτερόλεπτα η εντολή θεωρείται αποτυχημένη, διαγράφεται η εγγραφή της αίτησης και το route αποκρίνεται με HTTP status code 500.

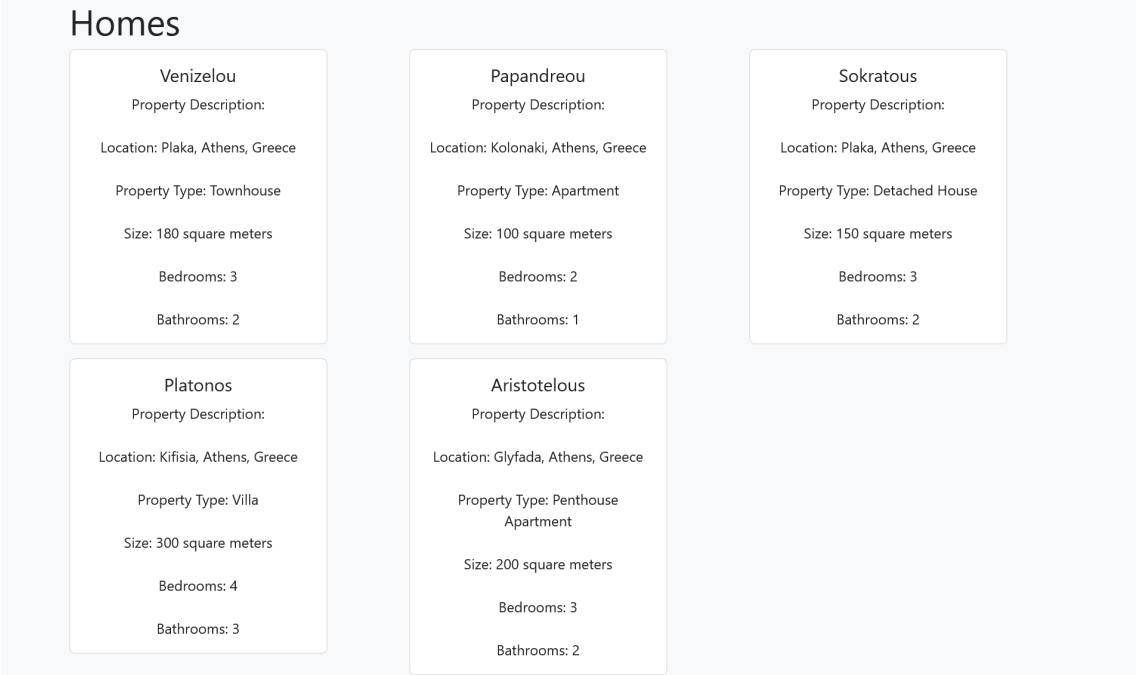
Κεφάλαιο 5

Επίδειξη

Στο κεφάλαιο αυτό γίνεται η επίδειξη της καλής λειτουργίας της πλατφόρμας, παρουσιάζοντας όλες τις λειτουργίες της.

5.1 Χρήση

Αρχικά, ο χρήστης συνδέεται στην ιστοσελίδα από την οποία προσφέρεται η δικτυακή εφαρμογή, μέσω του `'/'` endpoint. Η αρχική σελίδα αυτής φαίνεται παρακάτω:

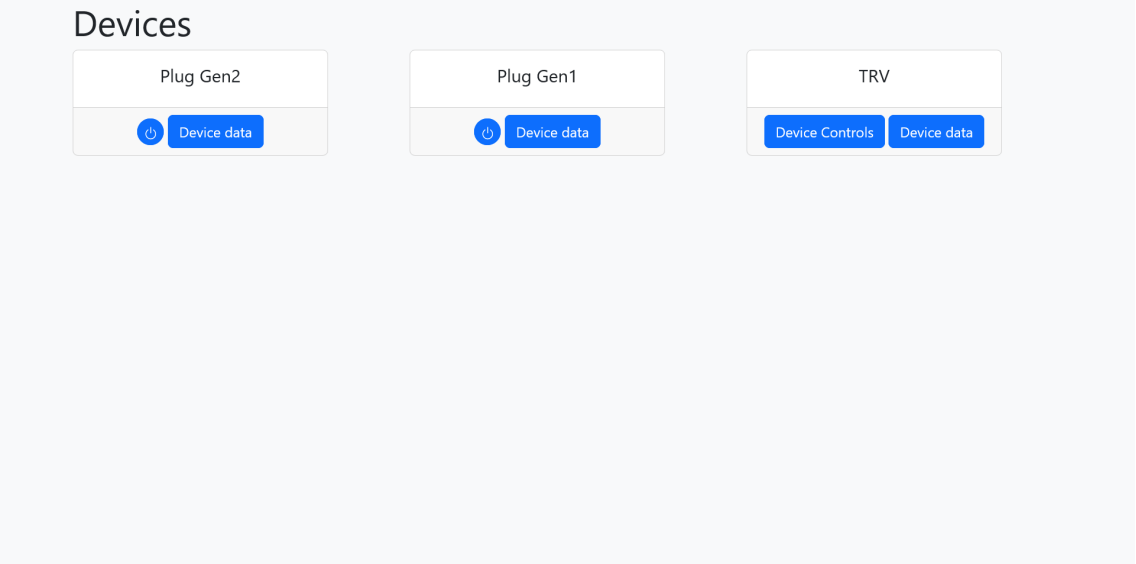


Εικόνα 5.1: *Αρχική σελίδα δικτυακής εφαρμογής*

Στη σελίδα αυτή απεικονίζονται όλα τα Έξυπνα Σπίτια που είναι εγγεγραμμένα στην πλατφόρμα. Απεικονίζονται επίσης, με λεπτομέρεια, όλες οι πληροφορίες που διαθέτει η πλατφόρμα για το κάθε ένα.

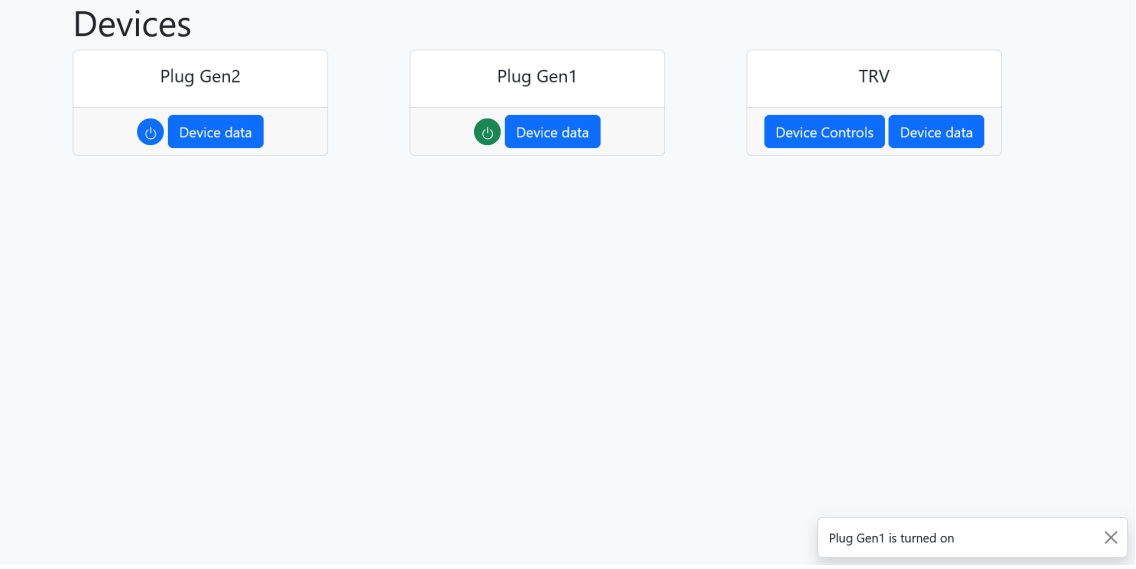
Στην συνέχεια ο χρήστης επιλέγει το σπίτι με το οποίο θέλει να αλληλεπιδράσει και ανακατευθύνεται, μέσω του `'/devices'` endpoint στην σελίδα που φαίνεται στην Εικόνα 5.2. Σε αυτή παρουσιάζονται όλες οι συσκευές που είναι εγκατεστημένες στο σπίτι του. Για κάθε

συσκευή ο χρήστης έχει την επιλογή να κατευθυνθεί στην σελίδα ελέγχου τους ή στην σελίδα που παρουσιάζονται τα δεδομένα τους.



Εικόνα 5.2: *Σελίδα Devices*

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο όμως, οι συσκευές τύπου έξυπνης μπρίζας δεν διαθέτουν σελίδα ελέγχου καθώς υποστηρίζουν μόνο μια λειτουργία ελέγχου, το άνοιγμα και κλείσιμο του διακόπτη τους. Ο χρήστης επιλέγει να αλλάξει την θέση του διακόπτη της συσκευής πρώτης γενιάς. Όταν πατήσει το αντίστοιχο κουμπί, αποστέλλεται μέσω του `'/mqtt_to_http'` endpoint η εντολή στο Gateway. Αυτό με την σειρά του, εκτελεί την εντολή και επιστρέφει την απόκριση που έλαβε από την έξυπνη συσκευή, πίσω στον Κεντρικό Εξυπηρετητή. Η εφαρμογή χρησιμοποιεί το πεδίο `ison` της απόκρισης και ενημερώνει την ιστοσελίδα.

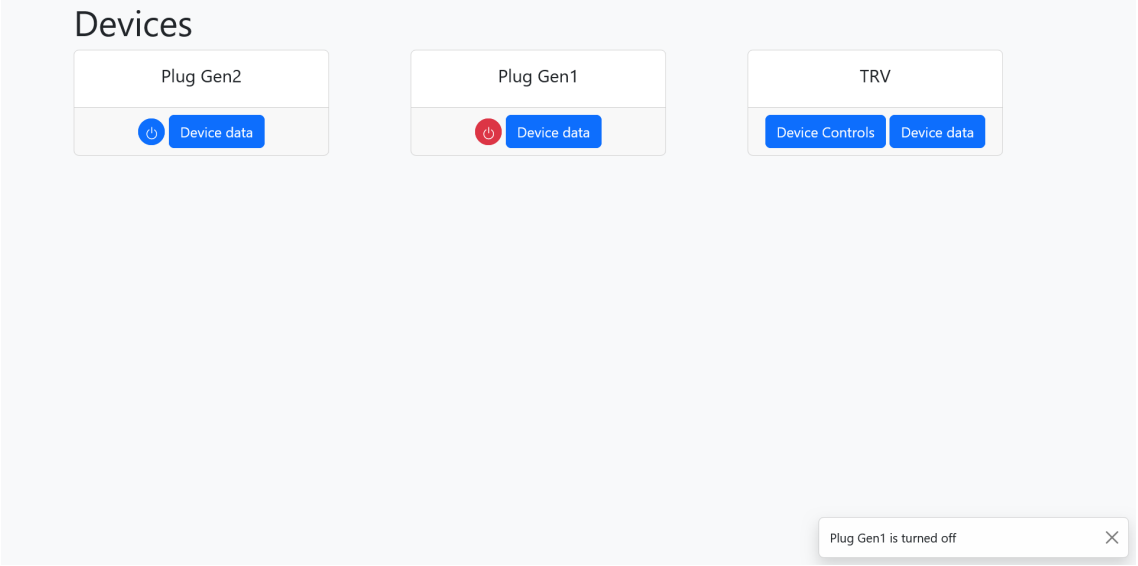


Εικόνα 5.3: *Ενημέρωση στοιχείων ιστοσελίδας (πράσινο)*

Εμφανίζεται ένα μήνυμα (toast) στο κάτω μέρος της ιστοσελίδας, που ενημερώνει τον

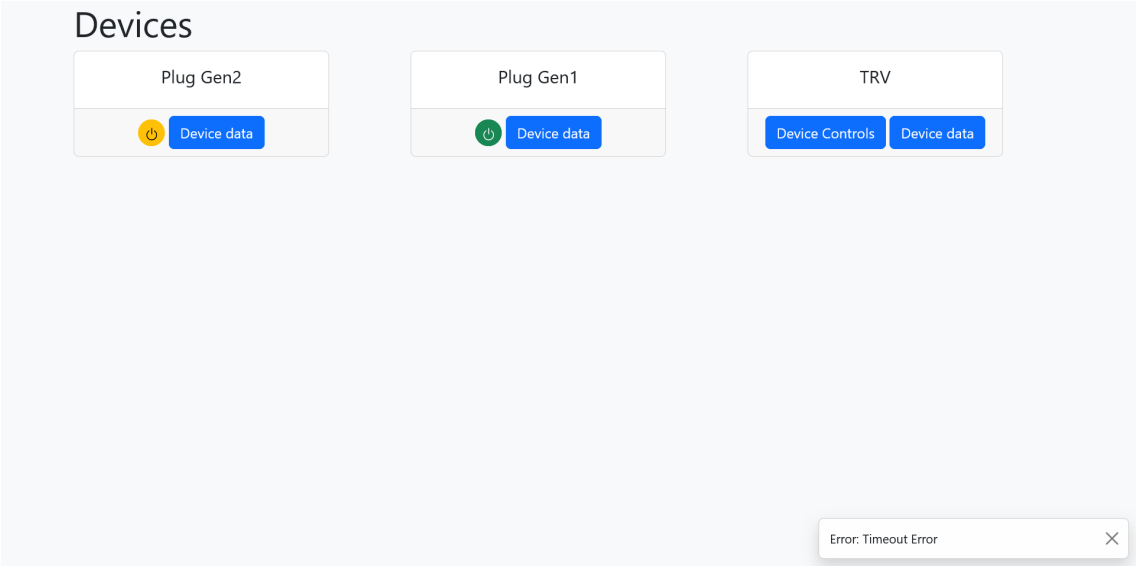
χρήστη για το αποτέλεσμα της ενέργειάς του. Παρατηρούμε, επίσης πως το κουμπί έχει γίνει πράσινο, για να υποδηλώσει πως ο διακόπτης είναι πλέον στην κλειστή θέση.

Αμέσως μετά, ο χρήστης επιλέγει να εκτελέσει ξανά την ίδια ενέργεια. Όπως παρατηρούμε στην Εικόνα 5.4 το κουμπί πλέον έχει κόκκινο χρώμα για να υποδηλώσει πως ο διακόπτης είναι στην ανοιχτή θέση. Εμφανίζεται επίσης το μήνυμα που ενημερώνει τον χρήστη για το αποτέλεσμα της ενέργειάς του.



Εικόνα 5.4: *Ενημέρωση στοιχείων ιστοσελίδας (κόκκινο)*

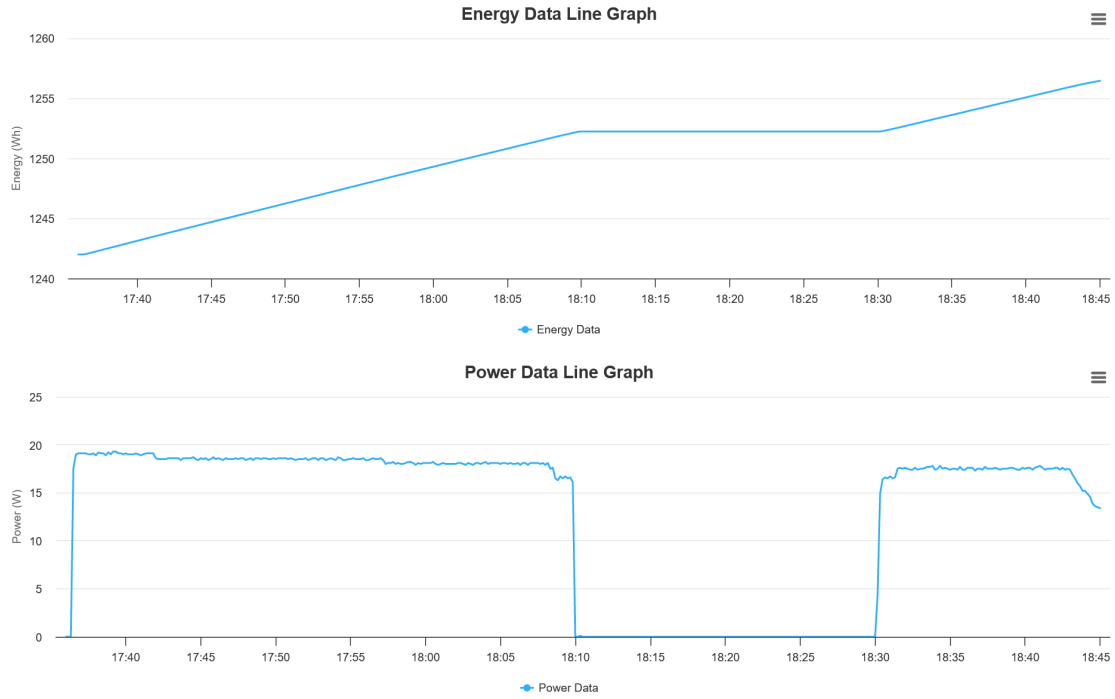
Τέλος, ο χρήστης επιλέγει να αλλάξει θέση στον διακόπτη της μπρίζας δεύτερης γενιάς, η οποία για τους λόγους της επίδειξης έχει αποσυνδεθεί από το τοπικό δίκτυο. Ενημερώνεται για την αποτυχία της εκτέλεσης της εντολής και το χρώμα του κουμπιού γίνεται πορτοκαλί για να υποδηλώσει πως υπήρξε κάποιο σφάλμα κατά την εκτέλεση αυτής.



Εικόνα 5.5: *Ενημέρωση στοιχείων ιστοσελίδας (σφάλμα)*

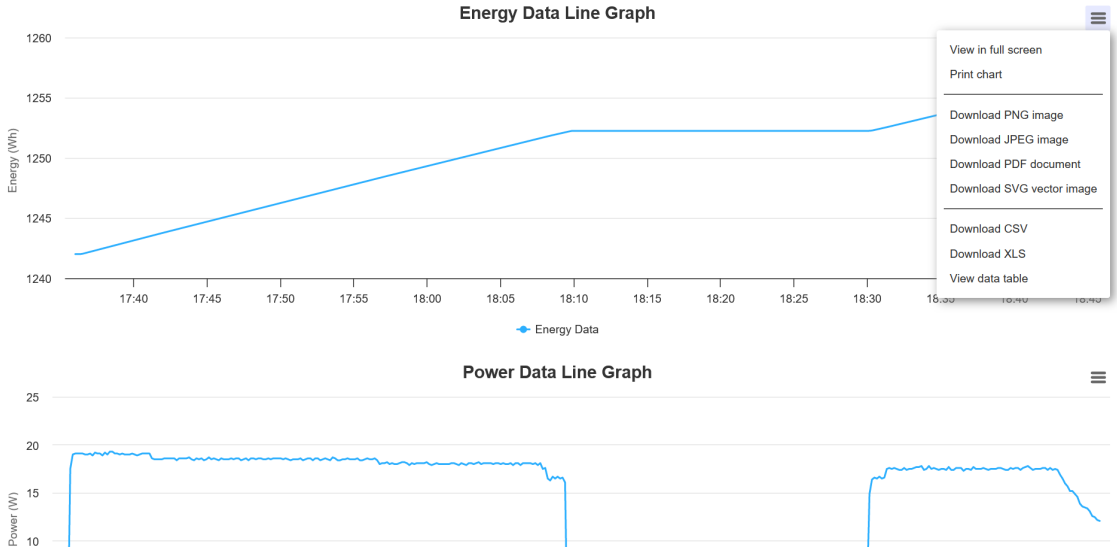
Ο χρήσης επιθυμεί να μελετήσει τα δεδομένα της έξυπνης μπρίζας που έχει συλλέξει η

πλατφόρμα. Χρησιμοποιώντας το αντίστοιχο κουμπί, αρχικά, η εφαρμογή χρησιμοποιεί το `'/mqtt_to_http'` endpoint με την εντολή `'sendData'`, ώστε να ενημερωθεί η βάση του Κεντρικού Εξυπηρετητή με τα πιο πρόσφατα δεδομένα. Το Gateway λαμβάνει το μήνυμα και αποστέλλει τα δεδομένα που δεν έχει διαβιβάσει. Μόλις ολοκληρωθεί αυτή η διαδικασία, μέσω του `'/device_data'` endpoint, ο χρήστης ανακατευθύνεται στην σελίδα των δεδομένων. Λόγω του τύπου της συσκευής που επέλεξε, το αρχείο που επιστρέφει είναι το endpoint το `plug_data.ejs`. Η σελίδα των δεδομένων παρουσιάζεται παρακάτω στην Εικόνα 5.6:



Εικόνα 5.6: *Σελίδα δεδομένων μπρίζας*

Στην σελίδα παρουσιάζονται τα διαγράμματα για κάθε τύπο αισθητήρα που διαθέτει η συσκευή, δηλαδή η συνολική κατανάλωση ενέργειας και η στιγμιαία ισχύς. Ο χρήστης μπορεί να αλληλεπιδράσει με τα διαγράμματα, κάνοντας hover πάνω τους και να δει την τιμή για κάθε χρονική στιγμή που αποτυπώνεται. Επιπλέον, χρησιμοποιώντας το εικονίδιο του μενού, που βρίσκεται στην πάνω δεξιά γωνία κάθε διαγράμματος, ο χρήστης έχει την δυνατότητα να κατεβάσει το διάγραμμα ως εικόνα ή να εξάγει δεδομένα που παρουσιάζονται σε αυτό σε μορφή αρχείου υπολογιστικών φύλλων.



Εικόνα 5.7: *Μενού διαγράμματος*

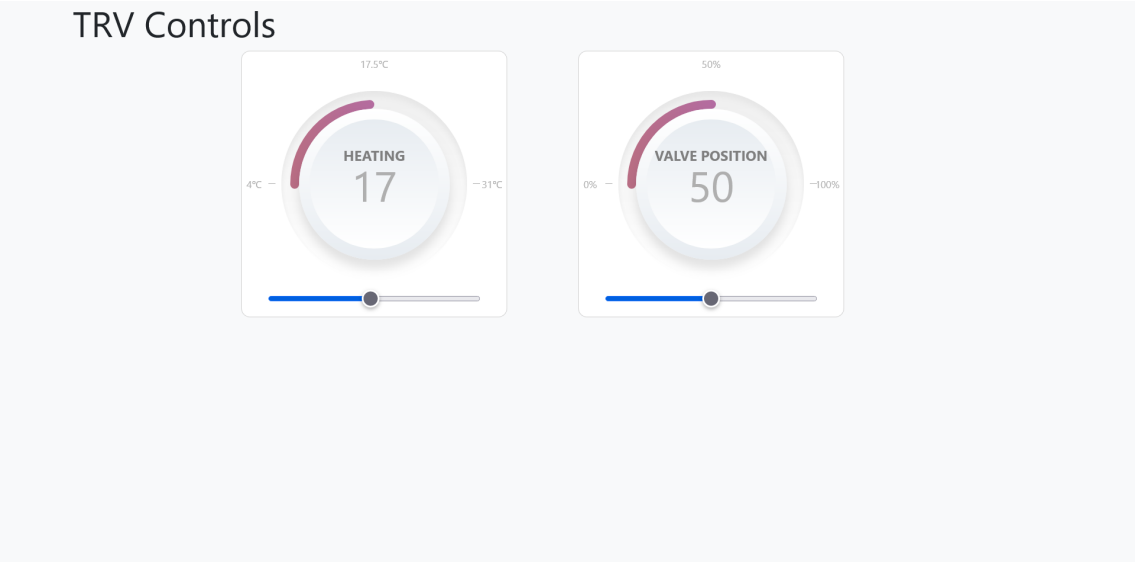
Ενδεικτικά, έγινε η εξαγωγή των δεδομένων ισχύος σε ένα αρχείο μορφής csv. Παρακάτω παρουσιάζονται οι πρώτες 50 γραμμές του αρχείου:

```
"DateTime","Power Data"
"2024-06-24 17:36:00",0
"2024-06-24 17:36:10",0
"2024-06-24 17:36:20",0
"2024-06-24 17:36:30",17.5
"2024-06-24 17:36:40",19
"2024-06-24 17:36:50",19.1
"2024-06-24 17:37:00",19.1
"2024-06-24 17:37:10",19.1
"2024-06-24 17:37:20",19.1
"2024-06-24 17:37:30",19
"2024-06-24 17:37:40",19
"2024-06-24 17:37:50",19.1
"2024-06-24 17:38:00",18.9
"2024-06-24 17:38:10",19.2
"2024-06-24 17:38:20",19.1
"2024-06-24 17:38:30",19.1
"2024-06-24 17:38:40",18.9
"2024-06-24 17:38:50",19.2
"2024-06-24 17:39:00",19
"2024-06-24 17:39:10",19.3
"2024-06-24 17:39:20",19.3
"2024-06-24 17:39:30",19.1
"2024-06-24 17:39:40",19.1
"2024-06-24 17:39:50",19
"2024-06-24 17:40:00",19.1
"2024-06-24 17:40:10",19
"2024-06-24 17:40:20",19
"2024-06-24 17:40:30",19
"2024-06-24 17:40:40",19.1
"2024-06-24 17:40:50",19
```

```
"2024-06-24 17:41:00",18.9
"2024-06-24 17:41:10",19
"2024-06-24 17:41:20",19.1
"2024-06-24 17:41:30",19.1
"2024-06-24 17:41:40",19.1
"2024-06-24 17:41:50",19.1
"2024-06-24 17:42:00",18.6
"2024-06-24 17:42:10",18.5
"2024-06-24 17:42:20",18.5
"2024-06-24 17:42:30",18.5
"2024-06-24 17:42:40",18.5
"2024-06-24 17:42:50",18.6
"2024-06-24 17:43:00",18.6
"2024-06-24 17:43:10",18.6
"2024-06-24 17:43:20",18.6
"2024-06-24 17:43:30",18.6
"2024-06-24 17:43:40",18.4
"2024-06-24 17:43:50",18.6
"2024-06-24 17:44:00",18.6
```

Παρατηρούμε πως πράγματι κάθε εγγραφή απέχει από την επόμενη δέκα δευτερόλεπτα, όπως ακριβώς είχαμε ορίσει στο script της συσκευής και πως δεν έχει χαθεί καμία.

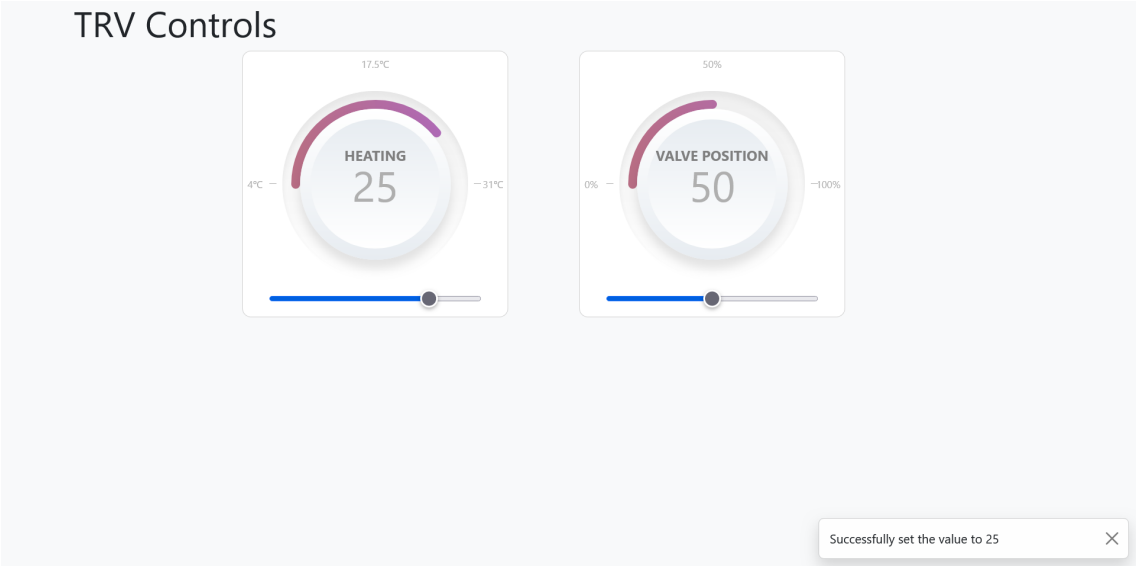
Συνεχίζοντας, ο χρήστης επιθυμεί να ελέγξει την συσκευή TRV. Από την σελίδα Devices, επιλέγει το κουμπί TRV Data και μέσω του `'/devices'` endpoint ανακατευθύνεται στην παρακάτω σελίδα:



Εικόνα 5.8: *Σελίδα ελέγχου TRV*

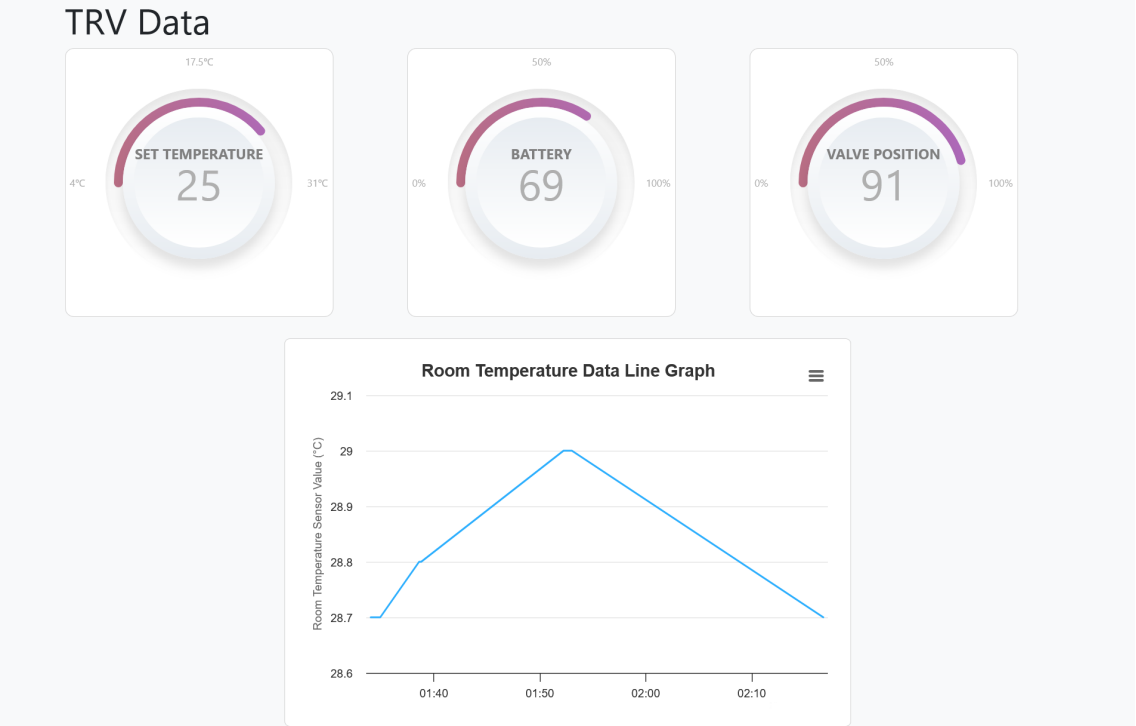
Σε αυτή βρίσκονται δύο κυκλικά dials. Το ένα επιτρέπει τον έλεγχο της θέσης της βαλβίδας χειροκίνητα, ενώ με το άλλο ορίζεται η επιθυμητή θερμοκρασία δωματίου με τον έλεγχο της βαλβίδας να τον αναλαμβάνει η συσκευή. Το input του χρήστη γίνεται μέσω του slider κάτω από κάθε διάγραμμα. Η ένδειξη του dial προσαρμόζεται ανάλογα ώστε να αντικατοπτρίζει την θέση του slider. Όταν γίνει η επιλογή αποστέλλεται μέσω του `'/mqtt_to_http'` endpoint η εντολή στο Gateway. Αυτό με την σειρά του, εκτελεί την εντολή και επιστρέφει

την απόκριση που έλαβε από την έξυπνη συσκευή, πίσω στον Κεντρικό Εξυπηρετητή. Η εφαρμογή ελέγχει την απόκριση και ενημερώνει τον χρήστη μέσω ενός μηνύματος σχετικά με το αποτέλεσμά της.



Εικόνα 5.9: *Απόκριση στη σελίδα ελέγχου TRV*

Τέλος, ο χρήστης επιθυμεί να δει την τωρινή κατάσταση της συσκευής TRV και τα δεδομένα που έχει συλλέξει η πλατφόρμα. Χρησιμοποιώντας το αντίστοιχο κουμπί, αρχικά, η εφαρμογή χρησιμοποιεί το `'/mqtt_to_http'` endpoint, ώστε να ενημερωθεί η βάση του Κεντρικού Εξυπηρετητή με τα πιο πρόσφατα δεδομένα. Μόλις τελειώσει αυτή η διαδικασία, μέσω του `'/device_data'` endpoint, ο χρήστης ανακατευθύνεται στην σελίδα των δεδομένων. Λόγω του τύπου της συσκευής που επέλεξε, το αρχείο που επιστρέφει το endpoint είναι το `trv_data.ejs`. Η σελίδα των δεδομένων παρουσιάζεται στην Εικόνα 5.10.



Εικόνα 5.10: *Σελίδα δεδομένων TRV*

Στην σελίδα παρουσιάζονται οι πιο πρόσφατες τιμές για τις μετρήσεις της επιθυμητής θερμοκρασίας δωματίου, της στάθμης της μπαταρίας και της θέσης της βαλβίδας. Επιπλέον, παρουσιάζεται το ιστορικό της θερμοκρασίας του δωματίου στο οποίο βρίσκεται η συσκευή μέσω ενός διαγράμματος. Το εικονίδιο του μενού, που βρίσκεται στην πάνω δεξία γωνία κάθε διαγράμματος, προσφέρει στον χρήστη τις ίδιες δυνατότητες, όπως και στα διαγράμματα δεδομένων των συσκευών τύπου plug.

Κεφάλαιο 6

Επίλογος

6.1 Σύνοψη

Σε αυτή την Διπλωματική Εργασία, μελετήθηκε και αναπτύχθηκε μια εφαρμογή Smart Home που διαχειρίζεται τις συσκευές ενός έξυπνου σπιτιού. Παρουσιάστηκαν δύο εναλλακτικές προσεγγίσεις στην επικοινωνία μεταξύ του Gateway και του Κεντρικού Εξυπηρετητή, μία εξολοκλήρου μέσω MQTT και μία κυρίως μέσω WebSocket. Οι προϋποθέσεις που τέθηκαν στο πρώτο κεφάλαιο, σχετικά με τα χαρακτηριστικά της πλατφόρμας ικανοποιούνται σε μεγάλο βαθμό. Ακολούθως παρουσιάζονται μερικές από αυτές και εξηγείται πως ικανοποιούνται.

- **Edge Computing:** Το Gateway συλλέγει τα δεδομένα που παράγουν οι συσκευές και δίνει την δυνατότητα επεξεργασίας πριν αυτά προωθηθούν στον Κεντρικό Εξυπηρετητή.
- **Επεκτασιμότητα:** Στο πρώτο κεφάλαιο τέθηκαν δύο προϋποθέσεις σχετικά με την επεκτασιμότητα της πλατφόρμας. Όσον αφορά την προσθήκη υποστήριξης ενός νέου τύπου συσκεύης, η διαδικασία είναι απλή, καθώς απαιτείται μόνο η δημιουργία ενός script για την εισαγωγή των δεδομένων στην βάση του Gateway και δύο ιστοσελίδες, μία για την παρουσίαση των συλλεγμένων δεδομένων και μία για τον έλεγχο της συσκευής. Όσον αφορά την προσθήκη νέων εφαρμογών για την επεξεργασία των δεδομένων στο Edge, το Raspberry Pi 4 και το βασισμένο στο Linux λειτουργικό σύστημά του, είναι ικανά να διαχειριστούν ένα μεγάλο αριθμό ήδη υπάρχοντων εφαρμογών, ενώ η ανάπτυξη νέων καθίσταται εύκολη.
- **Απομακρυσμένος έλεγχος χωρίς Port Forwarding:** Η επικοινωνία του Gateway με τον Κεντρικό εξυπηρετητή δεν γίνεται μέσω της αποστολής HTTP αιτήσεων. Αντ' αυτού γίνεται με την ανταλλαγή μηνυμάτων MQTT διαμέσου του MQTT Broker. Κάθε Gateway εγγράφεται σε ένα topic και δέχεται εκεί τις εντολές που προορίζονται για αυτό. Έτσι, ο Κεντρικός Εξυπηρετητής δεν χρειάζεται να γνωρίζει την IP κάθε Gateway και συνεπώς, αποφεύγονται τεχνικές όπως Port Forwarding και static public IPs.

6.2 Μελλοντικές Επεκτάσεις

Παρακάτω παρουσιάζονται πιθανές μελλοντικές επεκτάσεις στην πλατφόρμα καθώς και μερικές λειτουργίες που εξετάστηκαν αλλά λόγω προβλημάτων κατά την υλοποίησή τους δεν

αναπτύχθηκαν περαιτέρω.

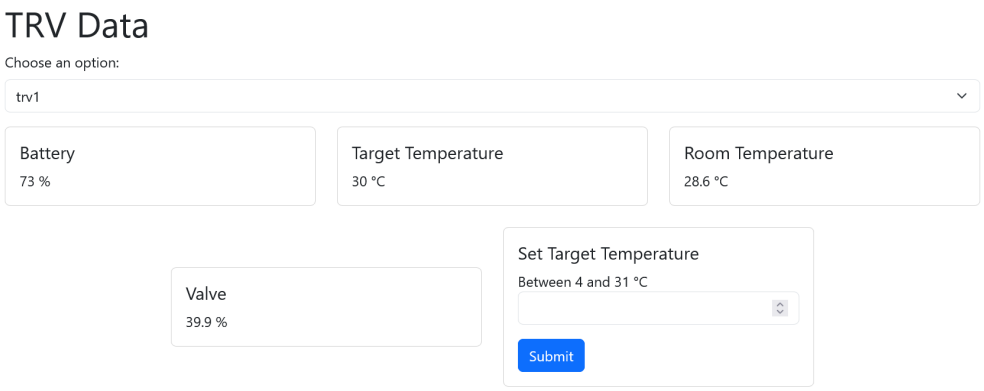
6.2.1 Δικτυακή Εφαρμογή από το Gateway

Στην υλοποίηση που περιγράφηκε η Δικτυακή Εφαρμογή της Πλατφόρμας προσφέρεται από τον Κεντρικό Εξυπηρετητή. Μια πιο ελαφρυά έκδοσή της θα μπορούσε να προσφέρεται αποκλειστικά από το Gateway. Με αυτόν τον τρόπο, μια πιθανή απώλεια σύνδεσης με το διαδίκτυο, δεν θα εμπόδιζε τον χρήστη από τον έλεγχο και την παρακολούθηση των συσκευών του. Προφανώς, απαραίτητη προϋπόθεση για την πρόσβαση του χρήστη σε αυτή την Εφαρμογή είναι το Gateway και η συσκευή του χρήστη να είναι συνδεδεμένα στο ίδιο τοπικό δίκτυο. Ένα επιπλέον θετικό αυτής της προσθήκης είναι η μείωση του χρόνου απόκρισης των έξυπνων συσκευών στις εντολές του χρήστη, καθώς δεν χρειάζεται το αίτημα να αποσταλλεί εκτός τοπικού δικτύου. Όμως, μια τέτοια εφαρμογή θα είχε πρόσβαση μόνο στις συσκευές που είναι συνδεδεμένες στο αντίστοιχο Gateway. Επιπλέον, λόγω της περιορισμένης μνήμης που διαθέτει το Raspberry Pi, η εφαρμογή δεν θα μπορεί να διατηρεί και να προβάλλει ιστορικό δεδομένων μεγάλου χρονικού διαστήματος. Συμπερασματικά, παρόλο που μια τέτοια προσέγγιση δεν μπορεί να αντικαταστήσει την Δικτυακή Εφαρμογή που προσφέρεται από τον Κεντρικό Εξυπηρετητή, η ευελιξία και η διαθεσιμότητα που προσδίδει στην Πλατφόρμα δικαιολογούν την ανάπτυξή της.

6.2.2 Χρήση Έξυπνης Συσκευής ως Gateway

Η συσκευή Shelly Plus Plug S, διαθέτει μια βάση δεδομένων key/value μπορώντας με αυτό τον τρόπο να λειτουργήσει ως HTTP Server υποστηρίζοντας μέχρι πέντε endpoints ανά script, να εκτελέσει αιτήματα HTTP σε τρίτες συσκευές, ενώ όπως προαναφέρθηκε, υποστηρίζει πλήρως και το πρωτόκολλο MQTT. Για αυτόν τον λόγο, εξετάστηκε η δυνατότητα αντικατάστασης του Raspberry Pi και χρήση αυτής της συσκευής ως Gateway. Όμως οι περιορισμοί στους πόρους της συσκευής οδήγησαν σε μερικά προβλήματα κατά την υλοποίηση αυτής της προσέγγισης. Αρχικά, η βάση δεδομένων που προσφέρει η συσκευή έχει δυνατότητα μόνο 50 εγγραφών. Το μέγεθος της τιμής κάθε εγγραφής περιορίζεται στα 255 bytes. Όπως γίνεται εύκολα αντιληπτό, τα δεδομένα που καταγράφονται στο Gateway θα πρέπει να συμπιεστούν κατά πολύ. Δηλαδή, αντί για αυτοτελείς μετρήσεις με περίοδο 10 δευτερολέπτων, θα έπρεπε να εφαρμόζεται μια συνάρτηση συνάθροισης, όπως η συνάρτηση μέση τιμής, για κάθε νεά μέτρηση που δημιουργείται. Τα αποθηκευμένα δεδομένα, λοιπόν, θα αναφέρονταν στην μέση τιμή της μέτρησης για ένα μεγάλο σχετικά χρονικό διάστημα, π.χ. για μια ημέρα. Επίσης, το μέγεθος των αιτήσεων που δέχεται ο HTTP Server περιορίζεται στα 3072 bytes. Τέλος, σε αντίθεση με το Raspberry Pi το οποίο υποστηρίζει μια πληθώρα εφαρμογών για την περαιτέρω επεξεργασία των δεδομένων, η συγκεκριμένη προσέγγιση περιορίζει πολύ αυτή την δυνατότητα, αφού η συσκευή δεν υποστηρίζει third party εφαρμογές. Παρ' όλους του περιορισμούς έγινε μια απόπειρα πειραματικής υλοποίησης αυτής της προσθήκης σε συνδυασμό και με την προηγούμενη. Όμως προέκυπταν σε τυχαία βάση πρόβληματα στον τρόπο με τον οποίο η συσκευή έκανε resolve τα τοπικά hostnames μέσω του πρωτοκόλλου mDNS. Μέτα την αναφορά του προβλήματος στην εταιρεία που κατασκευάζει την συσκευή παραχώρηθηκε ένα πειραματικό firmware, που έλυσε το παραπάνω

πρόβλημα. Δημιουργήθηκε, λοιπόν ένα endpoint που θα επέτρεπε στον χρήστη την παρακολούθηση και τον έλεγχο των συσκευών TRV που έχει εγκαταστήσει. Το αποτέλεσμα είναι πλήρως λειτουργικό και φαίνεται στην Εικόνα 6.1:



Εικόνα 6.1: *Σελίδα που προσφέρεται μέσω της Έξυπνης Συσκευής*

Ο χρήστης επιλέγει με ποια συσκευή θέλει να αλληλεπιδράσει, παρακολουθεί τα δεδομένα της και θέτει την επιθυμητή θερμοκρασία δωματίου. Η σχεδίαση της σελίδας είναι πολύ πιο απλή από αυτή που παρουσιάστηκε στο προηγούμενο κεφάλαιο, λόγω των περιορισμένων πόρων. Επιπλέον, στην βάση της συσκευής αποθηκεύονται μόνο οι τελευταίες μετρήσεις των αισθητήρων του TRV. Μια εγγραφή στη βάση παρουσιάζεται παρακάτω:

```
key: trv1
value: {
  "hostname": "shellytrv-60a423db07a6",
  "valve_position": 39.9,
  "target_t": 30,
  "temperature": 28.6,
  "battery": 73
}
```

Συμπερασματικά, η συγκεκριμένη προσθήκη είναι εφικτή υπό την προϋπόθεση πως θα γίνουν υποχωρήσεις στον όγκο των δεδομένων που θα αποθηκεύονται στο Gateway, καθώς και στην δυνατότητα επεξεργασίας τους. Στα πλαίσια αυτής της Διπλωματικής Εργασίας αυτή η υποχώρηση δεν ήταν αποδεκτή και έτσι δεν εξετάστηκε περαιτέρω.

6.2.3 ML/AI

Η ταχεία ανάπτυξη του κλάδου της τεχνητής νοημοσύνης και της μηχανικής μάθησης προσφέρει πολλές δυνατότητες για την εξαγωγή συμπερασμάτων από τα παραγόμενα δεδομένα των έξυπνων συσκευών [17, 18]. Η προσθήκη μιας τέτοιας εφαρμογής αποτελεί μια λογική συνέχεια της υλοποίησης που παρουσιάστηκε. Ειδικότερα, λόγω της επεκτασιμότητας που προσφέρει το Raspberry Pi, μια ελαφριά εφαρμογή τεχνητής νοημοσύνης

θα μπορούσε να λειτουργεί στο Edge και να ενημερώνει σε πραγματικό χρόνο τον χρήστη για τυχόν ασυνήθιστα μεγάλη κατανάλωση των συσκευών του μέσα στην ημέρα. Ακόμη, η μεγάλη επεξεργαστική ισχύς που προσφέρει το Cloud, επιτρέπει την χρήση πιο εξελιγμένων και απαιτητικών μοντέλων. Ένα τέτοιο παράδειγμα είναι μια εφαρμογή που θα παρατηρούσε την καθημερινότητα του χρήση, θα μάθανε τις συνήθειές του και θα μπορούσε να κάνει κατάλληλες ενέργειες χωρίς την επέμβαση του χρήστη.

6.2.4 OTA ενημερώσεις

Οι παραπάνω προσθήκες δεν είναι απαραίτητο να εμπεριέχονται στην αρχική έκδοση της πλατφόρμας. Η προσθήκη νέων λειτουργιών και η αναβάθμιση των ήδη υπαρχόντων μπορεί να γίνεται μέσω ενημερώσεων λογισμικού. Αυτές που αφορούν τον Κεντρικό Εξυπηρετητή γίνονται ήδη χωρίς την αλληλεπίδραση του χρήστη. Αυτές που αφορούν το Edge, δηλαδή οι προσθήκες που αφορούν τις έξυπνες συσκευές και το Gateway, θα πρέπει να προσφέρονται από απόσταση (OTA) με την ελάχιστη δυνατή αλληλεπίδραση του χρήστη. Προς αυτόν τον σκοπό προτείνεται η ανάπτυξη ενός API που θα λειτουργεί στο Gateway και θα επιτρέπει στους σχεδιαστές της πλατφόρμας να αναπτύσσουν νέες λειτουργίες και να τις προωθούν μέσω του διαδικτύου στις κατάλληλες συσκευές.

6.2.5 Αυτόνομο δίκτυο συσκευών

Στην υλοποίηση που περιγράφηκε, οι έξυπνες συσκευές είναι συνδεδεμένες στο τοπικό οικιακό δίκτυο. Αυτό σημαίνει πως στην επικοινωνία αυτών με το Gateway, παρεμβάλεται ο δρομολογητής γεγονός το οποίο μπορεί δυνητικά να οδηγήσει σε υπερφόρτωση του τοπικού δικτύου. Επιπλέον, οποιαδήποτε βλάβη στον δρομολογητή θα δημιουργούσε πρόβλημα στην επικοινωνία των συσκευών με την πλατφόρμα. Μια προσθήκη που θα αντιμετώπιζε τα παραπάνω προβλήματα είναι η δημιουργία ενός αυτόνομου δικτύου για την επικοινωνία των συσκευών. Επιπλέον, το πρωτόκολλο επικονωνίας αυτού του δικτύου μπορεί να είναι είτε το HTTP, είτε κάποιο διαφορετικό αρκεί να υποστηρίζεται από τις εγκατεστημένες συσκευές. Ένα τέτοιο παράδειγμα είναι η επικοινωνία μεταξύ των συσκευών και του Gateway μέσω Bluetooth [19].

Βιβλιογραφία

[1] Inés Sittón-Candanedo, Ricardo Alonso, Óscar García, Lilia Muñoz και Sara Rodríguez. *Edge Computing, IoT and Social Computing in Smart Energy Scenarios*. *Sensors*, 19:3353, 2019.

[2] Priya Suresh, J Vijay Daniel, Velusamy Parthasarathy και RH Aswathy. *A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment*. *2014 International conference on science engineering and management research (ICSEMR)*, σελίδες 1–8. IEEE, 2014.

[3] N.L. Panwar, S.C. Kaushik και Surendra Kothari. *Role of renewable energy sources in environmental protection: A review*. *Renewable and Sustainable Energy Reviews*, 15(3):1513–1524, 2011.

[4] Jamshid Aghaei και Mohammad Iman Alizadeh. *Demand response in smart electricity grids equipped with renewable energy sources: A review*. *Renewable and Sustainable Energy Reviews*, 18:64–72, 2013.

[5] Mohsen Marjani, Fariza Nasaruddin, Abdullah Gani, Ahmad Karim, Ibrahim Abaker Targio Hashem, Aisha Siddiqa και Ibrar Yaqoob. *Big IoT data analytics: architecture, opportunities, and open research challenges*. *ieee access*, 5:5247–5261, 2017.

[6] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic και Marimuthu Palaniswami. *Internet of Things (IoT): A vision, architectural elements, and future directions*. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[7] Michael Armbrust, Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica και Matei Zaharia. *A View of Cloud Computing*. *Commun. ACM*, 53:50–58, 2010.

[8] Keyan Cao, Yefan Liu, Gongjie Meng και Qimeng Sun. *An Overview on Edge Computing Research*. *IEEE Access*, 8:85714–85728, 2020.

[9] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li και Lanyu Xu. *Edge Computing: Vision and Challenges*. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[10] Ghyzlane Cherradi, Adil Bouziri και Azedine Boulmakoul. *Smart Data Collection Based on IoT Protocols*. 2016.

[11] Sasu Tarkoma. *Publish/subscribe systems: design and principles*. John Wiley & Sons, 2012.

[12] Alexey Melnikov και Ian Fette. *The WebSocket Protocol*. RFC 6455, 2011.

[13] Udhaya Kumar Dayalan, Rostand A. K. Fezeu, Nitin Varyani, Timothy J. Salo και Zhi Li Zhang. *VeerEdge: Towards an Edge-Centric IoT Gateway*. *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, σελίδες 690–695, 2021.

[14] Wikipedia. *Raspberry Pi — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Raspberry%20Pi&oldid=1218999355, 2024. [Online; accessed 17-April-2024].

[15] Roger A. Light. *Mosquitto: server and client implementation of the MQTT protocol*. *Journal of Open Source Software*, 2(13):265, 2017.

[16] Kevin P. Gaffney, Martin Prammer, Larry Brasfield, D. Richard Hipp, Dan Kennedy και Jignesh M. Patel. *SQLite: past, present, and future*. *Proc. VLDB Endow.*, 15(12):3535–3547, 2022.

[17] Haochen Hua, Yutong Li, Tonghe Wang, Nanqing Dong, Wei Li και Junwei Cao. *Edge computing with artificial intelligence: A machine learning perspective*. *ACM Computing Surveys*, 55(9):1–35, 2023.

[18] Massimo Merenda, Carlo Porcaro και Demetrio Iero. *Edge machine learning for ai-enabled iot devices: A review*. *Sensors*, 20(9):2533, 2020.

[19] Antonio Cilfone, Luca Davoli, Laura Belli και Gianluigi Ferrari. *Wireless mesh networking: An IoT-oriented perspective survey on relevant technologies*. *Future internet*, 11(4):99, 2019.

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

π.χ.	παραδείγματος χάριν
IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
TCP	Transmission Control Protocol
HTTP	Hypertext Transfer Protocol
RDBMS	Relational Database Management System
QoS	Quality of Service
SoC	System on Chip
mDNS	multicast Domain Name System
API	Application Programming Interface
ML	Machine Learning
AI	Artificial Intelligence
OTA	over the air

Απόδοση ξενόγλωσσων όρων

Απόδοση	Ξενόγλωσσος όρος
νέφος	cloud
άκρη	edge
προώθηση θυρών	port forwarding
αναγνωριστικό	identifier
έξυπνο σπίτι	smart home
εκδότης	publisher
συνδρομητής	subscriber
διαμεσολαβητής	broker
θέμα	topic
ποιότητα υπηρεσίας	quality of service
υλικό	hardware
λογισμικό	software
εξυπηρετητής	server
αίτηση	request
απόκριση	response