ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΠΜΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΔΙΚΤΥΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΔΙΑΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑ ΚΑΙ ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΙ ΡΟΩΝ ΔΕΔΟΜΕΝΩΝ ΣΤΟ ΔΙΑΔΙΚΤΥΟ ΤΩΝ ΑΝΤΙΚΕΙΜΕΝΩΝ

Ηλίας Λάμπρου

Επιβλέπων: ***Γρηγόριος Δουμένης PH D.***
Επίκουρος καθηγητής

Άρτα, Μάιος 2021

INTEROPERABILITY AND DATA STREAM TRANSFORMATIONS IN THE IOT

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, 17/6/2021

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

- Επιβλέπων καθηγητής
  Γρηγόριος Δουμένης,
  PhD, Επίκουρος Καθηγητής

  Μέλος επιτροπής
  Χρήστος Γκόγκος,
  PhD, Αναπληρωτής Καθηγητής
- Μέλος επιτροπής
  Μαργαρίτη Σπυριδούλα,
  PhD, ΕΔΙΠ

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα μεταπτυχιακή εργασία είναι εκ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Λάμπρου Ηλίας

Υπογραφή

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα διπλωματική εκπονήθηκε στο τμήμα Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Ιωαννίνων για το μεταπτυχιακό πρόγραμμα Πληροφορικής και Δικτύων. Για την εκπόνησή αυτής θα ήθελα να ευχαριστήσω θερμά τον επιβλέπων καθηγητή κ. Γρηγόριο Δουμένη για τις πολύτιμες συμβουλές και οδηγίες.

Επίσης θα ήθελα να ευχαριστήσω όλους του συμφοιτητές μου του μεταπτυχιακού τμήματός μου για τη συμπαράστασή τους και ιδιαίτερα τους Αριστοτέλη Ζαχάρη και Σακκά Σοφία για την πολύτιμη βοήθεια.

ΠΕΡΙΛΗΨΗ

Στη διπλωματική αυτή εργασία γίνεται αρχικά μια ανάλυση των δυνατοτήτων των πλατφορμών IoT και ορίζονται τα κριτήρια αξιολόγησης που θα πρέπει να λάβουμε υπόψιν πριν την επιλογή μιας πλατφόρμας IoT για ένα έργο μας.

Περιγράφονται η δομή, τα χαρακτηριστικά και η αρχιτεκτονική τριών ανοιχτού κώδικα πλατφορμών IoT και αναλύεται ο τρόπος εγκατάστασης αλλά και ο τρόπος δημιουργίας ενός έργου IoT.

Ένα πολύ σημαντικό πράγμα που πρέπει να λάβουμε υπόψη στη σχεδίαση ενός έργου IoT είναι η επιλογή της πλατφόρμας και αυτό γιατί αυτή η επιλογή είναι δεσμευτική σε μεγάλο βαθμό, ώστε να μην μπορούμε στο μελλον να μεταφέρουμε το έργο μας σε άλλη πλατφόρμα.

Λόγω του ότι οι συσκευές υποστηρίζουν διαφορετικούς τύπους συνδεσιμότητας, διαφορετικά πρωτόκολλα αλλά και διαφορετικές μορφές μηνυμάτων η μεταφορά ενός έργου σε άλλη πλατφόρμα καθίσταται ιδιαίτερα δύσκολη.

Η εργασία αυτή ασχολείται με το να προτείνει μια λύση στο πρόβλημα που δημιουργείται όταν υπάρχει ανάγκη μεταφοράς ενός έργου IoT από μια πλατφόρμα σε μια άλλη.

Αναζητάμε τους τρόπους με τους οποίους μπορεί να γίνει η μεταφορά ενός έργου σε άλλη πλατφόρμα με την μικρότερη παρέμβαση στο υπάρχον υλικό και γι αυτό προτείνεται η λύση της IoT Gateway, της οποίας στη συνέχεια αναλύονται τα βασικά της χαρακτηριστικά και η αρχιτεκτονική.

Η εργασία εστιάζει στην ανάγκη χρήσης μιας IoT Gateway μεταξύ δύο MQTT brokers και αναλύονται οι λόγοι για τους οποίους είναι απαραίτητη η χρήση μιας τέτοιας πύλης μεταξύ συσκευών που χρησιμοποιούν το ίδιο πρωτόκολλο, τον ίδιο τύπο σύνδεσης αλλά και την ίδια μορφή μηνυμάτων.

Παρουσιάζεται ο τρόπος εγκατάστασης και παραμετροποίησης μιας υπάρχουσας IoT Gateway της IoT πλατφόρμας Thingsboard. Η IoT Gateway χρησιμοποιείται και δοκιμάζεται στη μεταφορά ένα υπάρχοντος IoT έργου στην πλατφόρμα Thingsboard.

Ερευνάται ο τρόπος με τον οποίο μπορεί να δημιουργηθεί μια IoT Gateway και υλοποιείται με κώδικα Java μια IoT Gateway η οποία μπορεί να μεταφέρει έργα IoT μεταξύ πλατφορμών που υποστηρίζουν το πρωτόκολλο MQTT.

Μέσα από δοκιμές γίνεται προσπάθεια να υποστηριχθούν πολλές διαφορετικές μορφές μηνυμάτων ώστε η IoT Gateway να μπορεί να χρησιμοποιηθεί σε ένα μεγάλο εύρος πλατφορμών IoT.

Τέλος γίνεται εξαγωγή πολύτιμων συμπερασμάτων.

Λέξεις-κλειδιά: Ιντερνετ των πραγμάτων, Πλατφόρμες, Πύλες

ABSTRACT

In this diplomatic assignment is firstly made an analysis of the capabilities of the IoT platforms and the criteria of evaluation are set, which we must take into consideration before selecting an IoT platform for one of our projects.

The structure, the characteristics and the architecture of three open source IoT platforms are described and the way of installation is analyzed, as well as the way of creating an IoT project.

One very important thing to take into consideration in designing an IoT project is the selection of the platform and that is because this option is binding to a great degree and binds us to be unable to transfer our work to another platform in the future.

This paper deals with proposing a solution to the problem that arises when there is a need to transfer an IoT project from one platform to another. Due to the fact that devices support different types of connectivity, different protocols but also different forms of messages, the transfer of a project to another platform becomes very difficult. We are looking for ways in which a project can be transferred to another platform with the least intervention in the existing material and therefore the solution of IoT Gateway is proposed, whose basic characteristics and architecture are then analyzed.

The assignment focuses on the need to use an IoT Gateway between two MQTT brokers and analyzes the reasons why it is necessary to use such a gateway between devices that use the same protocol, the same type of connection and the same message format. The way to set up and configure an existing IoT Gateway offered by the Thingsboard platform is analyzed. The IoT Gateway is used and tested to transfer an existing IoT project to the Thingsboard platform. The way to create an IoT Gateway is investigated and one that can transfer IoT projects between platforms that support the MQTT protocol is developed with Java code.

Through testing, an attempt is made to support many different message formats so that our IoT Gateway can be used on a wide range of IoT platforms.

Finally, valuable conclusions are drawn.

Keywords: IoT, Platforms, Gateways, MQTT, Java

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1- ΕΙΣΑΓΩΓΗ

1.1 ΠΛΑΤΦΟΡΜΕΣ IoT - ΓΕΝΙΚΑ

Το Διαδίκτυο των αντικειμένων είναι ένα δίκτυο συνδεδεμένων συσκευών και εφαρμογών οι οποίες μπορούν να συλλέγουν πληροφορίες και να τις μεταφέρουν σε άλλα σημεία του δικτύου για αποθήκευση, επεξεργασία και διανομή σε άλλες συσκευές με στόχο την ενημέρωση, ειδοποίηση ή τη δράση.

Ενώ το Iot ξεκίνησε από ανάγκη σύνδεσης συσκευών στο Cloud για την ανταλλαγή μηνυμάτων και πολύ σύντομα δημιουργήθηκαν επιπλέον ανάγκες για:

- Μαζική αποθήκευση δεδομένων
- Authentication και ασφάλεια
- Εφαρμογές επεξεργασίας δεδομένων
- Διαχείριση των συσκευών IoT
- Οπτικοποίηση δεδομένων

Πλατφόρμα IoT είναι σύνολο υπηρεσιών (SaaS) και τεχνολογιών που επιτρέπουν την υλοποίηση εφαρμογών παρέχοντας λύσεις για όλες τις παραπάνω ανάγκες.

Οι εφαρμογές οι οποίες σχετίζονται με το IoT και μπορεί να αφορούν τόσο τις συσκευές όσο και την αποθήκευση ή επεξεργασία των δεδομένων σε μια πλατφόρμα IoT ονομάζονται έργα IoT.

Όταν αναφερόμαστε σε πλατφόρμες IoT, χρησιμοποιούμε μια τεχνική ορολογία όπως πρωτόκολλα μεταφοράς, μηχανισμοί κανόνων, βάσεις δεδομένων κ.λπ. Αν και αυτοί οι όροι είναι σημαντικοί και αξίζουν προσεκτικής μελέτης, δεν δείχνουν με σαφήνεια πώς μια πλατφόρμα IoT μπορεί να μας βοηθήσει.

Θα αναλύσουμε τις βασικές εργασίες που πρέπει να εκτελέσει ένα έργο IoT, επισημαίνοντας τις λειτουργίες που πρέπει να καλύπτει η πλατφόρμα IoT.

Ένα έργο IoT πρέπει:

Να λαμβάνει δεδομένα πραγματικού κόσμου μέσω αισθητήρων.

Να μπορεί να αναλύσει δεδομένα τοπικά (edge computing)

Να συνδέεται στο cloud για μετάδοση δεδομένων και λήψη εντολών

Να αποθηκεύει δεδομένα στο cloud

Να αναλύει δεδομένα στο cloud και να δημιουργεί πληροφορίες

Να μπορεί να δίνει εντολές βάσει αυτών των πληροφοριών στα “πράγματα” να εκτελέσουν εργασίες.

Να παρέχει ασφάλεια τόσο στη μεταφορά των δεδομένων αλλά και στη συντήρησή τους.

Ξεκινώντας ένας χρήστης τη δημιουργία ενός έργου IoT δεν μπορεί να προσδιορίσει απόλυτα τις μελλοντικές του ανάγκες. Αυτό μπορούμε εύκολα να το καταλάβουμε αν σκεφτούμε πως ήταν το IoT πριν από 6-7 χρόνια. Το σημαντικότερο κομμάτι τότε ήταν το ανέβασμα των δεδομένων και η αποθήκευσή τους.

Σήμερα, θεωρούμε πολύ βασικό στοιχείο μιας πλατφόρμας την ανάλυση των δεδομένων και την επιβολή κανόνων. Είναι πολύ πιθανό πιθανό οι χρήστες μετά από καιρό να διαπιστώσουν κάποια κενά στην πλατφόρμα που επέλεξαν αρχικά και να αναγκαστούν να οδηγηθούν στη χρήση περισσότερων από μία πλατφόρμες ή τη μεταφορά τους σε άλλη πλατφόρμα που να πληροί όλες τις προδιαγραφές της εργασίας τους.

Αυτό που κάνει αυτή την εργασία πολύ σημαντική είναι το ότι έρχεται να προλάβει τέτοιες καταστάσεις και να βοηθήσει πέρα από την εξ' αρχής σωστή επιλογή της κατάλληλης πλατφόρμας, στην υποβολή λύσεων.

Θα γίνει ανάλυση στο θέμα της διαλειτουργικότητας και θα προταθεί η λύση μιας υπάρχουσας IoT Gateway για τη σύνδεση ενός υπάρχοντος έργου σε μια πλατφόρμα αλλά και η υλοποίηση με κώδικα μιας IoT Gateway.

Οι πλατφόρμες IoT προσφέρουν στους πελάτες ανταγωνιστικά πλεονεκτήματα και χαρακτηριστικά ώστε να τους ενθαρρύνουν για την επιλογή τους. Ένα κριτήριο αξιολόγησης μιας πλατφόρμας IoT, είναι όπως θα δούμε και στη συνέχεια η παροχή πολλών υπηρεσιών και εφαρμογών.

Λόγω της αυξανόμενης χρήσης του IoT, ο αριθμός των προσφερόμενων IoT πλατφορμών αυξάνεται με νέες πλατφόρμες να εισέρχονται στο χώρο κάθε μέρα. Παρατηρούμε ότι υπάρχει μια ετερογένεια σε αυτές τις πλατφόρμες λόγω των διαφορετικών προτύπων και προσεγγίσεων, που οδηγεί σε προβλήματα κατανόησης της αρχιτεκτονικής τους.

Το βασικό πρόβλημα που προκύπτει είναι η εύρεση της κατάλληλης πλατφόρμας IoT για ένα συγκεκριμένο πεδίο εφαρμογής. Παρόλο που οι πλατφόρμες IoT παρέχουν παρόμοιες ή ακόμη και ίδιες πολλές φορές εφαρμογές οι υποκείμενες τεχνολογίες διαφέρουν. Δεδομένου ότι δεν υπάρχει μια γενική αρχιτεκτονική, οι χρήστες θα πρέπει πρέπει να ελέγξουν τις περιγραφές των πλατφορμών ώστε να κατανοήσουν σε βάθος την κάθε αρχιτεκτονική.

1.2 ΣΥΣΚΕΥΈΣ ΚΑΙ ΣΥΝΔΕΣΙΜΌΤΗΤΑ

Η σύνδεση των συσκευών στο Διαδίκτυο είναι σίγουρα ένα ουσιαστικό κομμάτι του Διαδικτύου των πραγμάτων. Υπάρχει ένας μεγάλος αριθμός επιλογών σχετικά με τον τρόπο που θα συνδεθεί μια συσκευή στο IoT.

Η επιλογή της συνδεσιμότητας μιας συσκευής έχει να κάνει με τα παρακάτω:

- Χαμηλό κόστος

- Χαμηλή κατανάλωση ενέργειας
- Χαμηλός ρυθμός δεδομένων
- Επεκτασιμότητα
- Αξιοπιστία
- Τοποθεσία της συσκευής
- Το πρωτόκολλο σύνδεσης

Λαμβάνοντας υπόψην τα παραπάνω ο χρήστης IoT μπορεί να επιλέξει ένα από τους παρακάτω τύπους δικτύωσης για τις συσκευές του:

1. **Cellular**
2. **Satellite**
3. **WiFi / Ethernet**
4. **Bluetooth**
5. **RFID**
6. **Zigbee**
7. **NFC**
8. **LPWAN**

Με ποιους όμως τρόπους θα συνδεθούν αυτές οι συσκευές με μια πλατφόρμα IoT ή μεταξύ τους;

Ασφαλώς για κάποια είδη συνδεσιμότητας η άμεση σύνδεση με μια πλατφόρμα θα ήταν αδύνατη χωρίς να υπάρχει μεσολάβηση στην επικοινωνία άλλης συσκευής. Τέτοιες συσκευές είναι οι IoT Gateways οι οποίες θα αναλυθούν παρακάτω.

Γενικά μπορούμε πούμε ότι το IoT αποτελείται από τρία βασικά στοιχεία, τα οποία είναι οι συσκευές, οι gateways και οι πλατφόρμες.

Μεταξύ αυτών μπορούμε να βρούμε 4 γενικούς τρόπους σύνδεσης: [32]

Συσκευή σε συσκευή (D2D) - άμεση σύνδεση μεταξύ δύο IoT συσκευών οι οποίες μπορούν να μοιράζονται πληροφορίες αμέσως χωρίς μεσάζοντες. Για παράδειγμα, τα βιομηχανικά ρομπότ και οι αισθητήρες συνδέονται μεταξύ τους άμεσα για να συντονίσουν τις ενέργειές τους και να εκτελέσουν τη συναρμολόγηση εξαρτημάτων πιο αποτελεσματικά. Αυτός ο τύπος σύνδεσης δεν είναι ακόμη πολύ κοινός, επειδή οι περισσότερες συσκευές δεν είναι σε θέση να χειριστούν τέτοιες διαδικασίες.

Συσκευή με IoT Gateway - σύνδεση μεταξύ αισθητήρων και IoT Gateway. Οι Gateways είναι πιο ισχυρές υπολογιστικές συσκευές από τους αισθητήρες. Έχουν δύο βασικές λειτουργίες: να μπορούν να επικοινωνούν με τις συσκευές υποστηρίζοντας όσο το δυνατόν περισσότερους τρόπους σύνδεσης και πρωτόκολλα. Μπορούν να επεξεργάζονται και να αναδιαμορφόνουν τα δεδομένα από αισθητήρες και να τα δρομολογούν προς μία πλατφόρμα IoT για ανάλυση και αποθήκευση.

Πύλη προς πλατφόρμα IoT - μετάδοση δεδομένων από πύλη προς το μια πλατφόρμα IoT. Εδώ για να προσδιοριστεί ποιο πρωτόκολλο θα χρησιμοποιηθεί, πρέπει να γίνει ανάλυση της κυκλοφορίας δεδομένων (συχνότητα σφαλμάτων και συμφόρησης, απαιτήσεις ασφάλειας και πόσες παράλληλες συνδέσεις χρειάζονται).

Μεταξύ Πλατφορμών IoT - μεταφορά πληροφοριών εντός κέντρων δεδομένων ή σύννεφων. Τα πρωτόκολλα για αυτόν τον τύπο σύνδεσης θα πρέπει να είναι εύκολο να αναπτυχθούν και να ενσωματωθούν σε υπάρχουσες εφαρμογές, να έχουν υψηλή διαθεσιμότητα, χωρητικότητα και αξιόπιστη αποκατάσταση καταστροφών.

Αναλύοντας την δομή των συσκευών μπορούμε να πούμε ότι αποτελούνται από τρία μέρη:

1. Sensor and/or actuator

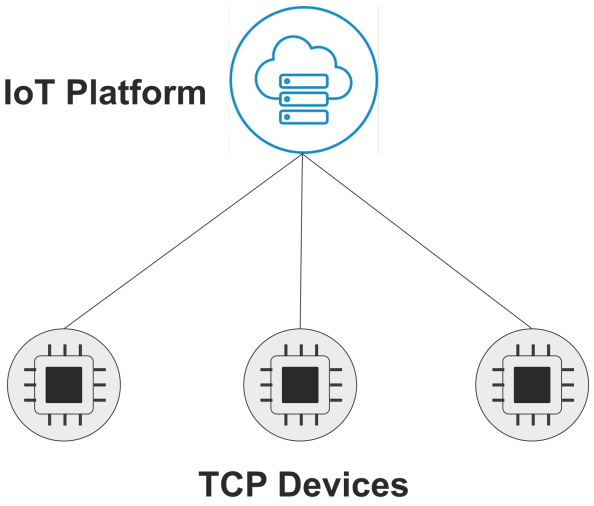
2. CPU, MEMORY, I/O

3. Physical connection (wired/wireless)

Όσον αφορά τις συνδέσεις μπορούμε πάλι να τις κατατάξουμε σε δύο βασικές κατηγορίες

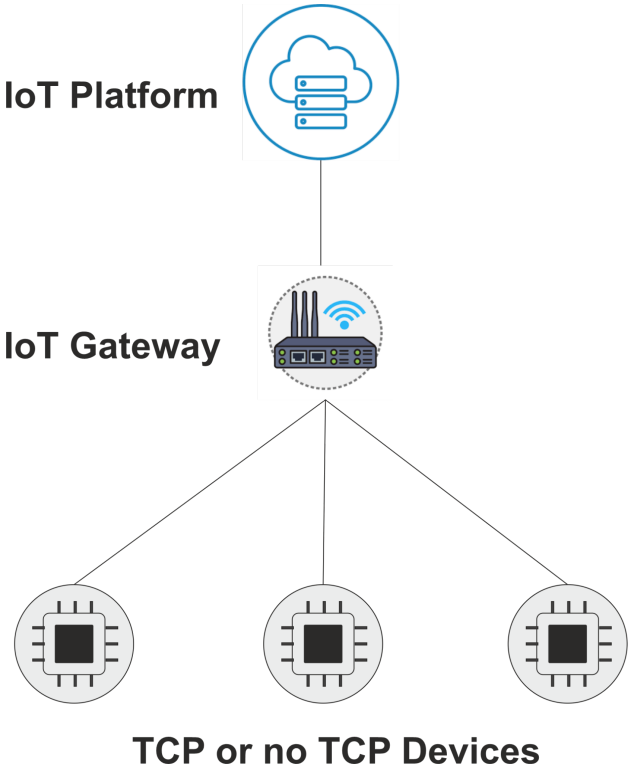
1. **TCP**
2. **no TCP**

Σε περίπτωση που μια συσκευή υποστηρίζει TCP και μπορεί να συνδεθεί άμεσα με μία πλατφόρμα μπορεί να χαρακτηριστεί ως IoT Ready.



Εικόνα 1.1: Άμεση σύνδεση TCP συσκευών σε πλατφόρμα

Ενώ στην περίπτωση που δεν είναι TCP ή δεν είναι IoT Ready θα είναι απαραίτητη μία IoT Gateway:



Εικόνα 1.2: Σύνδεση συσκευών σε πλατφόρμα μέσω IoT Gateway

Υπάρχουν διάφορα πρωτόκολλα με τα οποία επικοινωνούν οι συσκευές τις πλατφόρμες IoT. Στην εργασία αυτή η ανάλυση και οι δοκιμές θα γίνουν με το πρωτόκολλο MQTT, το οποίο είναι ένα απλό, «ελαφρύ» πρωτόκολλο μεταφοράς δεδομένων που χρησιμοποιείται πάνω από το TCP/IP και βασίζεται στο μοντέλο publish/subscribe.

Ο MQTT broker είναι ένας server, ο οποίος λαμβάνει όλα τα μηνύματα που κάνουν δημοσίευση “publish” οι clients και τα προωθεί μόνο στους clients που έχουν ζητήσει να τα λαμβάνουν κάνοντας “subscribe”.

Τα μηνύματα που ανταλλάσσονται αποτελούνται από δύο μέρη:

1. Topic: Είναι το θέμα του μηνύματος και πολλές φορές περιέχει και πληροφορίες για τη συσκευή ή τον αισθητήρα.
2. Payload: Είναι το κυρίως μέρος του μηνύματος που περιέχει πληροφορίες για τις τιμές των αισθητήρων. Μπορεί να είναι ένας αριθμός, ένα αλφαριθμητικό ή ένα κείμενο μορφής Json ή xml.

1.3 ΣΤΟΧΟΣ ΤΗΣ ΕΡΓΑΣΙΑΣ

Η παρούσα εργασία ασχολείται όχι με το πως θα συνδεθεί αρχικά μια συσκευή σε μια πλατφόρμα αλλά διερευνά και αναλύει τους τρόπους με τους οποίους μπορεί ένα υπάρχον έργο IoT με πολλές συσκευές να μεταφερθεί σε άλλη πλατφόρμα από αυτή που μελετήθηκε στον αρχικό σχεδιασμό. Αλλά και για όσους ξεκινούν ένα έργο IoT προτείνει μια αρχιτεκτονική που μπορεί να ακολουθηθεί ώστε μελλοντικά να είναι πιο εύκολη η αλλαγή μιας πλατφόρμας.

Προτείνεται μια γενική λύση η οποία δεν αφορά όλους τους τύπους συσκευών IoT αλλά εστιάζει μόνο σε έργα IoT τα οποία χρησιμοποιούν συσκευές TCP με δυνατότητα σύνδεσης σε τοπικό MQTT Broker.

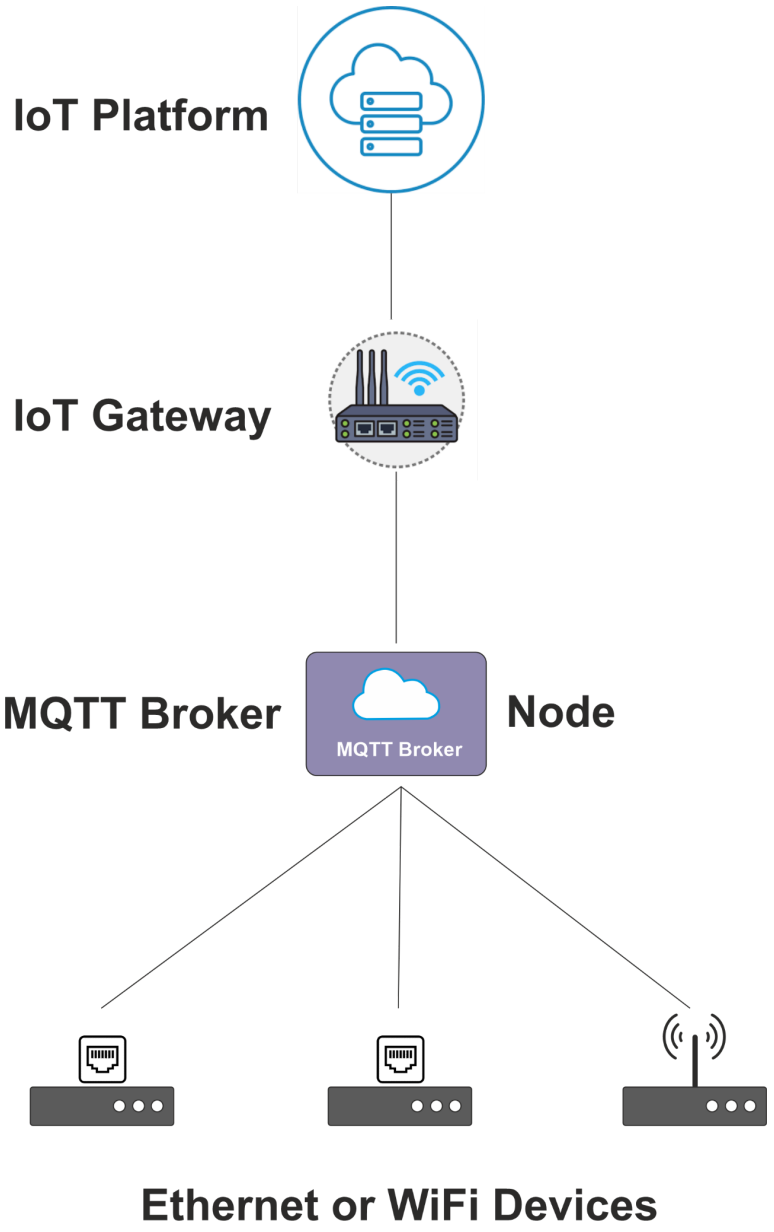
Βασικά στην εργασία αυτή ασχολούμαστε με μια κατηγορία έργων IoT στα οποίο όλες οι συσκευές είναι IoT Ready και συνδέονται σε έναν MQTT broker ο οποίες βρίσκεται τοπικά κοντά στο χώρο των συσκευών.

Οι συσκευές μπορούν να υποστηρίζουν είτε σύνδεση WiFi είτε σύνδεση Ethernet.

Ο MQTT broker συνδέεται σε μια IoT Gateway η οποία στη συνέχεια συνδέεται σε μία IoT πλατφόρμα.

Διερευνούμε το πως μπορεί σε αυτή την ειδική περίπτωση να γίνει μια αλλαγή πλατφόρμας χωρίς να χρειαστεί να γίνει καμία αλλαγή στις συσκευές και στον broker.

Προτείνουμε τη λύση της IoT Gateway η οποία δεν επηρεάζει καθόλου την υπάρχουσα υποδομή υλικού.



Εικόνα 1.3: Διάγραμμα σύνδεσης συσκευών, MQTT broker, IoT Gateway σε πλατφόρμα IoT

Στην εικόνα 1.3 βλέπουμε πως παρεμβάλλεται η IoT Gateway μεταξύ του MQTT broker και της πλατφόρμας.

Όπως θα δούμε παρακάτω στην εργασία για την μεταφορά ενός έργου με την παραπάνω δομή σε άλλη πλατφόρμα θα χρειαστεί είτε η αλλαγή της IoT Gateway είτε η εκ νέου παραμετροποίηση της ώστε να υποστηρίξει τη νέα πλατφόρμα.

Το ερώτημα που γεννιέται τώρα είναι γιατί από τη στιγμή που σχεδόν όλες οι πλατφόρμες, όπως θα δούμε παρακάτω στην ανάλυση των πλατφορμών, υποστηρίζουν το πρωτόκολλο MQTT, και όλες οι συσκευές του IoT έργου μας υποστηρίζουν και αυτές το ίδιο πρωτόκολλο, να είναι αναγκαία η χρήση της Gateway.

Ο βασικός λόγος είναι γιατί παρόλο που το πρωτόκολλο είναι ίδιο αλλάζει η μορφή και ο τύπος των μηνυμάτων.

Στο πρωτόκολλο MQTT διακρίνουμε διάφορους τύπους μηνυμάτων:

- **Json**
- **xml**
- **csv**
- **Byte array**
- **ή απλή αποστολή και λήψη ενός String, int, double etc..**

Δεν υποστηρίζουν όμως όλες οι πλατφόρμες όλους τους παραπάνω τύπους. Χτίζοντας για παράδειγμα ένα έργο με xml μηνύματα δεν είμαστε σίγουροι ότι αν χρειαστεί να μεταφέρουμε το έργο σε άλλη πλατφόρμα θα υποστηρίζεται ο ίδιος τύπος μηνυμάτων.

Στην παρούσα εργασία το πρόβλημα της μεταφοράς του έργου περιορίζεται ακόμη πιο πολύ και εξετάζουμε μόνο την περίπτωση που και η αρχική αλλά και η νέα πλατφόρμα υποστηρίζουν τον τύπο μηνυμάτων Json.

Δεν εξετάζουμε για παράδειγμα περιπτώσεις όπου η αρχική πλατφόρμα λαμβάνει μηνύματα xml και η νέα στην οποία θέλουμε να μεταφέρουμε το έργο λαμβάνει Json.

Ο τύπος Json υποστηρίζεται από όλες τις πλατφόρμες που αναφέρονται ή αναλύονται παρακάτω αλλά και από το σύνολο των πλατφορμών του IoT.

Παρόλο όμως που ακόμη και ο τύπος των μηνυμάτων είναι ίδιος, οι πλατφόρμες χρησιμοποιούν διαφορετική μορφή στη σύνταξη των μηνυμάτων με αποτέλεσμα ένα μήνυμα Json που στέλνεται σε μια πλατφόρμα να μην μπορεί να αναγνωριστεί από άλλη πλατφόρμα.

Η μη υιοθέτηση μιας ενιαίας μορφής μηνυμάτων Json από όλες τις πλατφόρμες καθιστά απαραίτητη της χρήση μιας IoT Gateway η οποία θα μπορεί να προσαρμόζει τα μηνύματα από μια μορφή σε μια άλλη.

Αυτό αποτελεί και το βασικό στόχο της εργασίας.

Αρχικά θα αναλύσουμε αλλά και θα χρησιμοποιήσουμε μια IoT Gateway - MQTT to MQTT σε ένα υπάρχων IoT έργο. Η IoT Gateway δίνεται από την πλατφόρμα στην οποία θέλουμε να μεταφέρουμε το έργο μας ώς εργαλείο και αφορά μόνο τη συγκεκριμένη πλατφόρμα μη μπορώντας να τη χρησιμοποιήσουμε σε κάποια άλλη.

Στη συνέχεια θα δημιουργήσουμε μια δική μας IoT Gateway - MQTT to MQTT η οποία θα μπορεί να χρησιμοποιηθεί γενικότερα για την εξασφάλιση διαλειτουργικότητας σε έργα IoT.

Θεωρήθηκε σημαντικό για την εργασία η παρουσίαση ορισμένων βασικών χαρακτηριστικών που διαθέτουν οι πλατφόρμες IoT ώστε να γίνει σωστή επιλογή μιας πλατφόρμας αλλά και να δούμε τους λόγους που πολλές φορές χρειάζεται η μετάβαση ενός έργου από μια πλατφόρμα σε μια άλλη. Στο παράρτημα ένα υπάρχει μια ανάλυση των βασικών χαρακτηριστικών τριών πλατφορμών IoT.

1.4 ΔΟΜΗ ΚΑΙ ΑΝΑΛΥΣΗ ΠΕΡΙΕΧΟΜΕΝΟΥ ΤΗΣ ΕΡΓΑΣΙΑΣ

Στο κεφάλαιο 2 θα κάνουμε μια ανάλυση των βασικών κριτηρίων επιλογής μιας πλατφόρμας. Θα ορίσουμε μια αρχιτεκτονική αναφοράς που ακολουθείται από το σύνολο των πλατφορμών και θα παρέχει μια πιο αφηρημένη εικόνα για τα συστατικά των IoT platforms. Αυτή η αρχιτεκτονική αναφοράς θα βοηθήσει να αναλύσουμε μια γενική ορολογία που θα χρησιμεύσει ως ενιαία βάση γνώσεων. Θα γίνει μια αναλυτική περιγραφή των IoT Gateways και θα ασχοληθούμε με την αρχιτεκτονική τους, τη χρησιμότητά τους και τις δυνατότητες που παρέχουν.

Στο κεφάλαιο 3: θα γίνει εγκατάσταση σε δικό μας server της IoT Gateway που μας παρέχει η πλατφόρμα Thingsboard.

Θα πραγματοποιηθούν δοκιμές και θα γίνει μεταφορά ενός υπάρχοντος έργου IoT που αφορά την ενεργειακή διαχείριση κτιρίων στην πλατφόρμα Thingsboard στην οποία θα κατασκευαστεί ένα Dashboard για απομακρυσμένο έλεγχο.

Στο κεφάλαιο 4 θα χρησιμοποιήσουμε τις γνώσεις και την εμπειρία που αποκτήσαμε από τη χρήση της gateway του κεφαλαίου τέσσερα ώστε να δημιουργήσουμε εξ' αρχής μια δική μας IoT gateway η οποία θα έχει την δυνατότητα να κάνει μετατροπές μηνυμάτων από MQTT to MQTT.

Η IoT gateway θα δοκιμαστεί για τη λειτουργία της σε μια διαφορετική πλατφόρμα IoT. Επιλέχθηκε για αυτές τις δοκιμές η πλατφόρμα Adafruit.io η οποία υποστηρίζει μεγάλη παραμετροποίηση στα μηνύματα που λαμβάνει και στέλνει, και μας δίνει τη δυνατότητα να εξετάσουμε ένα μεγάλο πλήθος περιπτώσεων μετατροπής μηνυμάτων.

Στο κεφάλαιο 5 θα αναφερθούν τα συμπεράσματα που απορρέουν από τις δοκιμές των δύο IoT Gateways.

Στο παράρτημα ένα θα αναλύσουμε τρεις ανοικτού κώδικα πλατφόρμες IoT.

Αρχικά την πλατφόρμα Thingspeak της οποίας θα αναλύσουμε τα βασικά της χαρακτηριστικά και τις δυνατότητες που προσφέρει στους χρήστες.

Στη συνέχεια θα γίνει μια ανάλυση της πλατφόρμας Thinger.io.

Και τέλος θα γίνει και μια μικρή αναφορά στην πλατφόρμα Thingsboard στην οποία θα μεταφέρουμε το IoT έργο μας χρησιμοποιώντας την IoT Gateway που μας παρέχει.

Σκοπός αυτών των αναλύσεων είναι να δούμε τις διαφορετικές αρχιτεκτονικές και λύσεις των πλατφορμών IoT.

Ειδικά για την πλατφόρμα thinger.io θα περιγράψουμε και τον τρόπο με τον οποίο μπορεί να γίνει εγκατάσταση σε δικό μας server και στη συνέχεια θα δημιουργήσουμε και ένα IoT έργο για να δουμε την υλοποίηση των δυνατοτήτων που αναφέρει η πλατφόρμα ότι μας παρέχει.

Στο παράρτημα Β υπάρχει ο κώδικας σε Java που αναπτύχθηκε για για την υλοποίηση της IoT Gateway της εργασίας

ΚΕΦΑΛΑΙΟ 2

2.1 ΒΑΣΙΚΑ ΣΤΟΙΧΕΙΑ ΕΠΙΛΟΓΗΣ ΜΙΑΣ ΠΛΑΤΦΟΡΜΑΣ IoT

Τα Βασικά κριτήρια επιλογής μιας πλατφόρμας μπορούμε να τα ομαδοποιήσουμε στις παρακάτω 6 κατηγορίες και υποκατηγορίες αυτών:

α. **Συσκευές**

- Εύκολη σύνδεση συσκευών
- Υποστήριξη μεγάλου πλήθους συσκευών

β. **Επικοινωνία**

- Ευκολία στη αποστολή - λήψη μηνυμάτων
- Υποστήριξη πολλών διαφορετικών πρωτοκόλλων όπως MQTT, HTTP (Rest Api), CoAP
- Υποστήριξη μορφοποίησης μηνυμάτων με διαφορετικούς τρόπους
- Παροχή έτοιμου κώδικα σε διάφορες γλώσσες προγραμματισμού για βασικές συσκευές όπως Arduino, raspberry, ESP etc.

γ. **Δεδομένα**

- Αποθήκευση δεδομένων
- Χρόνος μεταξύ αποστολής δεδομένων (μηνυμάτων)
- Υποστήριξη δυνατοτήτων επεξεργασίας των δεδομένων στο Cloud
- Υποστηριξη εξωτερικών βάσεων

δ. **Πλατφόρμα**

- Δημιουργία Dashboard και δυνατότητα οπτικοποίησης των δεδομένων (visualization)
- Υποστήριξη ειδοποιήσεων στο κινητό μέσω web service
- Υποστήριξη αποστολής emails, sms κλπ
- Δημιουργία κανόνων (Rule management) από το χρήστη με τη χρήση διαδικτυακών εργαλείων που παρέχονται από την πλατφόρμα.
- Παροχή επιπλέον έτοιμων εφαρμογών για κινητά τηλέφωνα με δυνατότητα παραμετροποίησης αυτών από τους χρήστες
- Παροχή έτοιμου κώδικα για Developes για ανάπτυξη WEB και MOBILE εφαρμογών .

- Καθυστέρηση εμφάνισης τιμών στο DashBoard καθώς και καθυστέρηση αποστολής εντολών στις συσκευές
- Προγραμματισμός επιπλεων συμβάντων μέσω της πλατφόρμας, όπως εντολές προς τις συσκευές βάση χρόνου ή λαμβάνοντας υπόψη τιμές από αισθητήρες άλλων συνδεδεμένων συσκευών.
- Διαχείριση συσκευών
- Στατιστικά στοιχεία για τις συσκευές τα δεδομένα και την επικοινωνία
- Είδος χρήσης (ερασιτεχνική ή επαγγελματική).
- Κλιμάκωση

ε. **Ασφάλεια**

- Ασφάλεια σύνδεσης συσκευών
- Ασφάλεια επικοινωνίας. Υποστήριξη πρωτοκόλλων SSL/TLS, AES κλπ.

στ. **Κόστος**

- Δυνατότητα ελεύθερης χρήσης
- Περιορισμοί δοκιμαστικής χρήσης
- Κόστος επιχειρηματικής χρήσης.

ζ. **Υποστήριξη**

- Forum / Community
- Υποστήριξη πελατών

Στη συνέχεια αυτής της εργασία θα γίνει μια περεταίρω ανάλυση των παραπάνω κριτηρίων ώστε να μπορέσουμε με βάση αυτά τα κριτήρια να συγκρίνουμε τις διάφορες IoT πλατφόρμες που θα αναλυθούν παρακάτω και να εντοπίσουμε και τα σημεία που θα κάνουν ένα χρήστη να αλλάξει πλατφόρμα IoT ή να χρησιμοποιήσει μια IoT Gateway για να μεταφέρει τα έργα του σε νέα πλατφόρμα.

α. Συσκευές

Κάθε πλατφόρμα μπορεί να συνδέεται με συσκευές οι οποίες είτε περιλαμβάνουν αισθητήρες ή συστήματα ελέγχου ή και τα δύο. Στην αγορά υπάρχουν συσκευές οι οποίες λειτουργούν με συγκεκριμένες πλατφόρμες (πχ. έξυπνοι διακόπτες) αλλά και ένα πλήθος συσκευών όπως Arduino, Raspberry, ESP8266 οι οποίες μπορούν να προγραμματιστούν από τους χρήστες ώστε να μπορούν να στέλνουν δεδομένα στις διάφορες πλατφόρμες.

Η πλατφόρμα θα πρέπει να παρέχει στο χρήστη ένα εύκολο στη χρήση διαχειριστή συσκευών (device manager) ώστε να ελέγχει τη σύνδεση και τη σωστή λειτουργία αυτών.

Σημαντική επίσης είναι η δυνατότητα να μπορεί ο χρήστης να δει τα αρχεία καταγραφής σύνδεσης και σφαλμάτων και να μπορεί να λάβει πληροφορίες για τον αριθμό των συσκευών που δημοσίευσαν δεδομένα σε ένα συγκεκριμένο θέμα (Cloud Pub / Sub). Η παραπάνω λειτουργία βοηθάει και στη αποσφαλμάτωση του κώδικα των συσκευών αλλά και στον εντοπισμό συσκευών που δε λειτουργούν σωστά.

Σε μερικές πλατφόρμες υπάρχουν πρόσθετες επιλογές οι οποίες προσπαθούν να προστατεύσουν το χρήστη από συσκευές που υπερβαίνουν ένα προκαθορισμένο χρεωστικό όριο δεδομένων, δημιουργώντας κατάλληλες ειδοποιήσεις.

Θα εξεταστεί το πόσο εύκολα μπορεί ένας χρήστης να συνδέσει μία συσκευή σε μια IoT πλατφόρμα και αν αυτή βοηθάει προς την κατεύθυνση αυτή με οδηγίες και γενικά με τα κατάλληλα tutorials.

Ο μέγιστος αριθμός συσκευών που μπορούν να συνδεθούν είναι ασφαλώς ένα ακόμη σημαντικό κριτήριο, δεδομένου ότι οι χρήστες ξεκινώντας με το IoT ποτέ δε μπορούν να ξέρουν τις μελλοντικές ανάγκες που μπορεί να προκύψουν στην επιχείρησή τους. Ασφαλώς για αυτούς είναι καλύτερα να επιλέξουν μια πλατφόρμα με υποστήριξη μεγάλου αριθμού συσκευών.

Δεν θα πρέπει όμως να ξεχνάμε και το ερασιτεχνικό κομμάτι. Η επανάσταση του Arduino έφερε ένα μεγάλο αριθμό ανθρώπων σε επαφή με την ηλεκτρονική τεχνολογία και το IoT. Εκατομμύρια ερασιτέχνες στον κόσμο θα θελήσουν και αυτοί κάποια στιγμή να χρησιμοποιήσουν μια πλατφόρμα. Θα λέγαμε ότι αυτό δεν είναι απλώς πρόβλεψη αλλά μονόδρομος. Κάποιες δυνατότητες οι οποίες ίσως να είναι χρήσιμες σε επαγγελματίες, ίσως να μην λαμβάνονται υπόψη από τους ερασιτέχνες.

Σκοπός του άρθρου αυτού δεν είναι να προσφέρει βοήθεια σε όσους θέλουν να χρησιμοποιήσουν μια πλατφόρμα για επαγγελματική χρήση αλλά να βοηθήσει στη σωστή επιλογή ένα πλήθος άλλων ανθρώπων, όπως φοιτητές, μαθητές, ερασιτέχνες ηλεκτρονικούς αλλά και απλούς χρήστες, οι οποίοι θα αγοράζουν στο μέλλον έτοιμες συσκευές που θα παρέχουν επιλογές σύνδεσης με πολλές πλατφόρμες αφήνοντας τον χρήστη να επιλέξει με ποια θα συνδεθεί.

β. Επικοινωνία

Η επικοινωνία μιας συσκευής με με μια IoT πλατφόρμα παρόλο που θα έπρεπε να είναι το εύκολο κομμάτι, όπως φαίνεται γίνεται όλο και πιο περίπλοκο λόγω του ότι οι ίδιες οι πλατφόρμες δεν χρησιμοποιούν έναν συγκεκριμένο πρωτόκολλο αλλά ούτε την ίδια μορφή μηνυμάτων.

Εδώ θα βρούμε πλατφόρμες που υποστηρίζουν επικοινωνία μέσω HTTP και πλατφόρμες που υποστηρίζουν επιπλέον πρωτόκολλα όπως το MQTT και CoAP. Διαφοροποιούνται όμως στη μορφή των μηνυμάτων. Αυτό ασφαλώς αποτελεί σοβαρή δέσμευση για τους χρήστες και ειδικά για τις επιχειρήσεις που θα χρησιμοποιούν μεγάλο αριθμό συσκευών. Το κόστος σε χρήμα αλλά και σε χρόνο σε περίπτωση απόφασης από μια εταιρία να αλλάξει πλατφόρμα θα είναι υψηλό.

Θεωρούμε ότι εάν μια πλατφόρμα παρέχει τη δυνατότητα προσαρμογής των μηνυμάτων ώστε οι συσκευές να μπορούν να συνδέονται σε αυτή χωρίς να χρειαστεί να αλλαχτεί το firmware, αυτό θα είναι ένα πολύ σοβαρό κριτήριο επιλογής αυτής της πλατφόρμας.

Πολλές πλατφόρμες παρέχουν έτοιμα παραδείγματα με κώδικα σε διάφορες γλώσσες και για συσκευές όπως Raspberry και Arduino. Κατά πόσο όμως οι χρήστες είναι ικανοί να χρησιμοποιήσουν αυτούς τους κώδικες. Θεωρούμε ότι είναι βασικό κριτήριο η ύπαρξη ενός ολοκληρωμένου οδηγού που θα συμπεριλαμβάνει ακόμη και βίντεο ώστε να βοηθηθούν οι χρήστες και να πετύχουν σε μικρό χρόνο και χωρίς προβλήματα την επικοινωνία.

γ. Δεδομένα

Εδώ θα δούμε πλατφόρμες IoT οι οποίες δεν παρέχουν καθόλου δυνατότητες αποθήκευσης των τιμών που στέλνονται από τους αισθητήρες, αλλά απλώς κρατάνε την τελευταία τιμή. Επίσης υπάρχουν περιορισμοί και στο χρόνο και στον όγκο των δεδομένων που μπορούν να αποθηκεύουν.

Οι περισσότερες χρησιμοποιούν το μέγεθος των δεδομένων και ως κριτήριο κόστους.

Ορισμένες αναλόγως με το επιλεγμένο από το χρήστη πακέτο απορρίπτουν μηνύματα τα οποία έρχονται σε μικρό χρονικό διάστημα από το προηγούμενο εμποδίζοντας έτσι το μεγάλο traffic.

Στο θέμα των δεδομένων είναι απαραίτητο αρχικά η πλατφόρμα να επιτρέπει ένα φιλτράρισμα των δεδομένων ώστε να εμποδίζεται η καταγραφή τιμών εκτός αποδεκτών ορίων. Η αυτόματη διαγραφή παλαιότερων τιμών μετά από επιλογή του χρήστη είναι επίσης πλεονέκτημα.

Ένα από τα βασικότερα όμως σημεία είναι η επεξεργασία των δεδομένων στο cloud, η ανάλυση αυτών των δεδομένων και η εξαγωγή πληροφοριών οι οποίες θα μπορούν να παρουσιαστούν στο χρήστη με γραφήματα.

Πολλές πλατφόρμες χρησιμοποιούν τη ροή δεδομένων IoT για προηγμένες αναλύσεις, οπτικοποιήσεις, μηχανική εκμάθηση και άλλα, ώστε να βελτιωθεί η λειτουργική αποδοτικότητα, να γίνει πρόβλεψη προβλημάτων και να δημιουργηθούν μοντέλα που περιγράφουν καλύτερα και βελτιστοποιουν κάθε επιχείρηση.

Πολλές βάσεις δεδομένων όπως SQL, NoSQL and Hybrid υποστηρίζονται από τις διάφορες πλατφόρμες IoT ενώ σημαντικό είναι να υπάρχει και υποστήριξη εξωτερικών βάσεων δεδομένων

δ. Πλατφόρμα

Η δημιουργία Dashboards και η δυνατότητα οπτικοποίησης των δεδομένων (visualization) είναι ένα από τα βασικότερα κριτήρια επιλογής μιας πλατφόρμας. Εδώ θα αξιολογηθεί η παραμετρικότητα και οι δυνατότητες της οπτικοποίησης των τιμών. Θα μετρηθούν οι χρόνοι καθυστέρησης από την αποστολή της τιμής από ένα αισθητήρα μέχρι την εμφάνιση της τιμής στο dashboard αλλά και η καθυστέρηση κάθε φορά που δίνεται μία εντολή από την πλατφόρμα μέχρι να φτάσει στον αισθητήρα.

Ορισμένες πλατφόρμες διαθέτουν apps τα οποία εγκαθίστανται σε περιβάλλον Android ή iOS και λειτουργούν ως service παρέχοντας ειδοποιήσεις για τα συμβάντα. Αλλά το σημαντικότερο για μια πλατφόρμα είναι η δυνατότητα αποστολής emails και sms για πολύ σημαντικά συμβάντα και καταστάσεις.

Αρκετές από τις IoT πλατφόρμες της αγοράς υποστηρίζουν τη λειτουργία "Rule management". Μέσα από αυτή τη λειτουργία, μπορεί ο χρήστης να επιβάλλει κάποιους κανόνες που αφορούν την αποστολή ειδοποιήσεων ή και την αποθήκευση τιμών στη βάση. Αυτοί οι κανόνες είναι εντολές βασικά μορφής If-Then οι οποίες λαμβάνουν υπόψη καταστάσεις από διαφορετικούς αισθητήρες αλλά κάνουν και φιλτράρισμα τιμών. Είναι βασικό ο χρήστης να μπορεί εύκολα να ορίσει τέτοιους κανόνες μέσα από εργαλεία που θα παρέχονται από την πλατφόρμα.

Ορισμένες πλατφόρμες IoT παρέχουν έτοιμα apps για Android και iOS τα οποία επεκτείνουν και φέρνουν κοντά στο χρήστη την εικόνα του Dashboard, δίνοντάς του ένα εύκολο έλεγχο καταστάσεων και τη δυνατότητα να παρακολουθεί τα δεδομένα του από τους αισθητήρες όπου και αν βρίσκεται.

Αρκετές πλατφόρμες IoT δίνουν και έτοιμους κώδικες από apps ώστε να βοηθήσουν τους developers στην ανάπτυξη εφαρμογών που θα είναι συμβατές με αυτές. Μεγάλες εταιρείες όπως η Google, Microsoft και Apple έχουν ενσωματώσει στις προγραμματιστικές τους πλατφόρμες ένα πλήθος βιβλιοθηκών για την υποστήριξη των σημαντικότερων IoT πλατφορμών της αγοράς.

ε. Ασφάλεια

Οι σύνδεσμοι επικοινωνίας με το cloud συχνά βασίζονται σε πρωτόκολλα όπως το Transport Layer Security (TLS) και το Secure Socket Layer (SSL) για την προστασία του απορρήτου των μηνυμάτων. Το πρωτόκολλο TLS και ο προκάτοχός του, το πρωτόκολλο SSL, χρησιμοποιούνται ευρέως για την ασφαλή μεταφορά δεδομένων μεταξύ του πελάτη (κόμβος IoT) και του διακομιστή μέσω μηχανισμών ελέγχου ταυτότητας, κρυπτογράφησης και ακεραιότητας.

Το TLS ή Internet Engineering Task Force (IETF), είναι ένα πρότυπο για ασφαλή επικοινωνία, χρησιμοποιείται για την ασφάλεια επικοινωνιών HTTP μέσω συνδέσμων που βασίζονται σε TCP. Το Datagram Layer Transport Security (DLTS) εκτελεί παρόμοια λειτουργικότητα για τους συνδέσμους UDP.

Η έκδοση TLS 1.2 του πρωτοκόλλου ασφαλείας καθίσταται το de facto πρότυπο για τη σύνδεση ενσωματωμένων συστημάτων σε ένα δίκτυο. Αυτός είναι ο λόγος για τον οποίο οι υπηρεσίες cloud όπως το AWS IoT απαιτούν από τη συσκευή IoT να πραγματοποιηθεί έλεγχος ταυτότητας χρησιμοποιώντας έναν μηχανισμό για συγκεκριμένη συσκευή.

Οι προγραμματιστές IoT χρησιμοποιούν επίσης συνήθως το wolfSSL, μια ελαφριά βιβλιοθήκη SSL / TLS βασισμένη σε C-γλώσσα, η οποία στοχεύει στα ενσωματωμένα περιβάλλοντα, RTOS και γενικά σε περιβαλλοντα με περιορισμένους πόρους κυρίως λόγω του μικρού μεγέθους, της ταχύτητας και των δυνατοτήτων φορητότητας που παρέχουν.

Παρόλο που η ασφάλεια είναι σημαντικό κριτήριο για την επιλογή μιας πλατφόρμας θα δυσκολευτούμε να βρούμε μια πλατφόρμα IoT που να μην παρέχει τουλάχιστον κρυπτογράφηση με το πρωτόκολλο SSL/TLS.

στ. Κόστος

Στο θέμα του κόστους διαπιστώσαμε μεγάλες τιμολογιακές διαφορές.

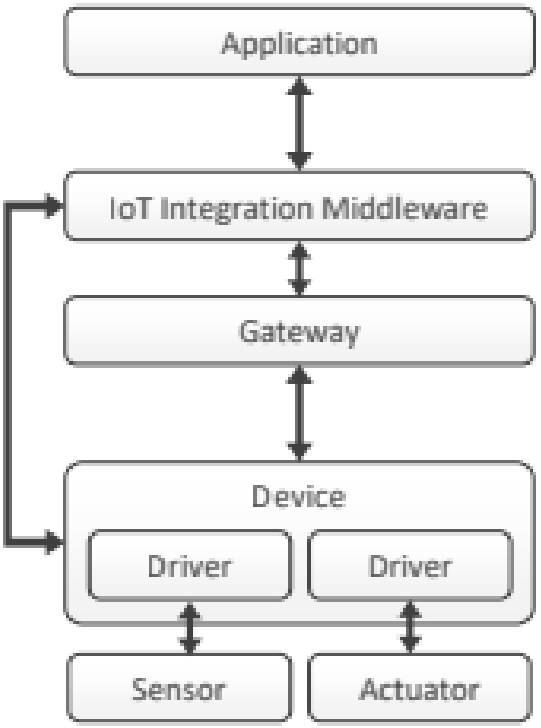
Οι πλατφόρμες IoT περιλαμβάνουν διάφορα πακέτα τιμολόγησης τα οποία σχετίζονται βασικά με τον όγκο των δεδομένων αλλά και με τις επιπλέον παροχές τους στην επεξεργασία αυτών των δεδομένων και των εφαρμογών που παρέχουν.

Σημαντικό είναι ασφαλώς να υπάρχει μια δοκιμαστική περίοδος λειτουργίας της πλατφόρμας ώστε ο χρήστης να μπορεί να δοκιμάσει τις δυνατότητές της.

Ορισμένες πλατφόρμες δίνουν και ένα πακέτο εντελώς ελεύθερο με περιορισμούς στον αριθμό των συσκευών και στον αριθμό των μηνυμάτων. με αυτό τον τρόπο γίνονται δελεαστικές για χρήση από πανεπιστήμια, σχολεία και γενικά ερασιτέχνες.

2.2 Η ΒΑΣΙΚΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΚΑΙ ΤΑ ΜΕΡΗ ΜΙΑΣ ΠΛΑΤΦΟΡΜΑΣ IoT

Αναλύοντας τα βασικά χαρακτηριστικά των IoT πλατφορμών μπορούμε να κατασκευάσουμε το διάγραμμα της εικόνας 2.1, στο οποίο διακρίνονται από τη μια τα μέρη που συνθέτουν την πλατφόρμα και από την άλλη ο τρόπος επικοινωνίας και η αλληλεπίδραση μεταξύ τους.



Εικόνα 2.1 Αρχιτεκτονική πλατφόρμας IoT

Sensor (Αισθητήρας) :

είναι μία συσκευή που ανιχνεύει ένα φυσικό μέγεθος όπως για παράδειγμα θερμοκρασία, υγρασία, πίεση κλπ. και παράγει από αυτό ηλεκτρικά σήματα. Τα σήματα αυτά οδηγούνται σε άλλες συσκευές που περιέχουν μικροελεγκτές οι οποίοι τα μετατρέπουν σε αριθμούς ή σε ειδικές περιπτώσεις ο ίδια η συσκευή του αισθητήρα εμπεριέχει μικροελεγκτή και το κατάλληλο ηλεκτρονικό κύκλωμα για την αποστολή των τιμών του αισθητήρα στην IoT πλατφόρμα.

Actuator (Ενεργοποιητής) :

είναι μία συσκευή η οποία μπορεί να δράσει, ελέγξει ή να χειριστεί το φυσικό περιβάλλον. Για παράδειγμα μπορεί να ελέγξει κάποια κίνηση ή να προκαλέσει ένα φωτεινό ή ηχητικό σήμα.

Οι ενεργοποιητές χρειάζονται συνήθως και μια πηγή τροφοδοσίας η οποία παρέχεται από τη συσκευή στην οποία είναι συνδεδεμένοι.

Λαμβάνουν ένα ηλεκτρικό σήμα και δημιουργούν μια κατάσταση όπως κίνηση, φως, ήχο, ενεργοποίηση κυκλωμάτων κλπ.

Μέσω των ενεργοποιητών μπορεί ένα λογισμικό να επηρεάζει το περιβάλλον, ενώ με τους αισθητήρες συμβαίνει το αντίθετο. Μπορεί δηλαδή το φυσικό περιβάλλον να επηρεάσει ένα λογισμικό.

Device (συσκευή)

Σε αυτή μπορεί είναι συνδεδεμένοι αισθητήρες ή ενεργοποιητές ή και τα δύο μέσω καλωδίων ή ασύρματα ή μπορεί ακόμη και να ενσωματώνει αυτά τα στοιχεία.

Για την επεξεργασία δεδομένων από αισθητήρες και για τον έλεγχο των ενεργοποιητών, περιέχουν μικροεπεξεργαστές και λογισμικό. Σε πολλές περιπτώσεις κάνουν και επεξεργασία των δεδομένων που λαμβάνουν από τους αισθητήρες.

Αποτελούν το σημείο εισόδου του φυσικού περιβάλλον στον ψηφιακό κόσμο. Οι συσκευές είτε είναι αυτόνομες είτε συνδέονται άλλη συσκευή..

Gateway (Πύλη)

Οι περισσότερες συσκευές λόγω της προσπάθειας μείωσης του όγκου αλλά και της κατανάλωσής τους δεν διαθέτουν το απαιτούμενο υλικό για την αποστολή δεδομένων στο cloud αλλά συνδέονται σε αυτό μέσω ενδιάμεσων πυλών (gateways).

Επίσης σε πολλές περιπτώσεις οι συσκευές δεν διαθέτουν τα κατάλληλα πρωτόκολλα για απευθείας σύνδεση.

Η πύλη χρησιμοποιείται για την αντιστάθμιση τέτοιων περιορισμών και την παροχή των απαιτούμενων τεχνολογιών καθώς και λειτουργιών μετάφρασης μεταξύ διαφορετικών πρωτοκόλλων και προώθησης επικοινωνίας μεταξύ συσκευών και άλλων συστημάτων.

Μια πύλη είναι, επομένως, υπεύθυνη για την υποστήριξη των απαιτούμενων τεχνολογιών και πρωτοκόλλων επικοινωνίας και προς τις δύο κατευθύνσεις και για τη μετάφραση των δεδομένων εάν αυτό είναι απαραίτητο.

Πολλές επιχειρήσεις προσθέτουν συνεχώς συσκευές οι οποίες μπορεί να υποστηρίζουν διαφορετικούς τρόπους διασύνδεσης, διαφορετικά πρωτόκολλα και διαφορετικές απαιτήσεις ισχύος.

Η πύλη γίνεται ο ενδιάμεσος σταθμός ο οποίος μπορεί από τη μια να επικοινωνεί με τέτοιου είδους συσκευές και από την άλλη με το cloud.

Επίσης σε πολλές των περιπτώσεων μπορεί να εκτελεί και κάποιες επιπλέον λειτουργίες επεξεργασίας δεδομένων.

Οι πύλες IoT τείνουν να γίνουν ένα βασικό συστατικό για την οικοδόμηση ενός ισχυρού IoT δημιουργώντας το Edge computing, το οποίο κατανέμει το φορτίο σε ένα σύστημα εκτελώντας την επεξεργασία δεδομένων στην πηγή δεδομένων, ή σε ένα κόμβο, αντί να βασίζεται σε έναν κεντρικό διακομιστή για το μεγαλύτερο μέρος της εργασίας.

IoT Integration Middleware

Το IoT Integration Middleware είναι υπεύθυνο για τη λήψη δεδομένων από τις συνδεδεμένες συσκευές και την επεξεργασία των ληφθέντων δεδομένων.

Μπορεί να αξιολογεί τα εισερχόμενα δεδομένα και με βάση των κανόνων συνθηκών δράσης (condition-action rules) να παρέχει τα ληφθέντα δεδομένα σε συνδεδεμένες εφαρμογές και να ελέγχει συσκευές με την αποστολή εντολών προς τους ενεργοποιητές.

Υπάρχει και η περίπτωση της επικοινωνίας μιας συσκευής απευθείας με το IoT integration middleware, αλλά για να συμβεί αυτό πρέπει η συσκευή να υποστηρίζει την κατάλληλη τεχνολογία επικοινωνίας,και το αντίστοιχο πρωτόκολλο μεταφοράς, όπως HTTP ή MQTT.

Επίσης και η μορφοποίηση των δεδομένων θα πρέπει να είναι σε μια συμβατή μορφή όπως JSON ή XML.

Σε διαφορετική περίπτωση η επικοινωνία της συσκευής με το IoT Integration Middleware θα πρέπει να γίνει μέσω Gateway.

Συνήθως, σε ένα IoT Integration Middleware μπορούμε να έχουμε πρόσβαση με χρήση API, π.χ. REST API που βασίζονται σε HTTP.

Application

Οι εφαρμογές είναι ειδικό λογισμικό που χρησιμοποιεί το IoT Integration Middleware ώστε να λάβει τα δεδομένα των αισθητήρων ή για να ελέγξει φυσικές ενέργειες μέσω των ενεργοποιητών.

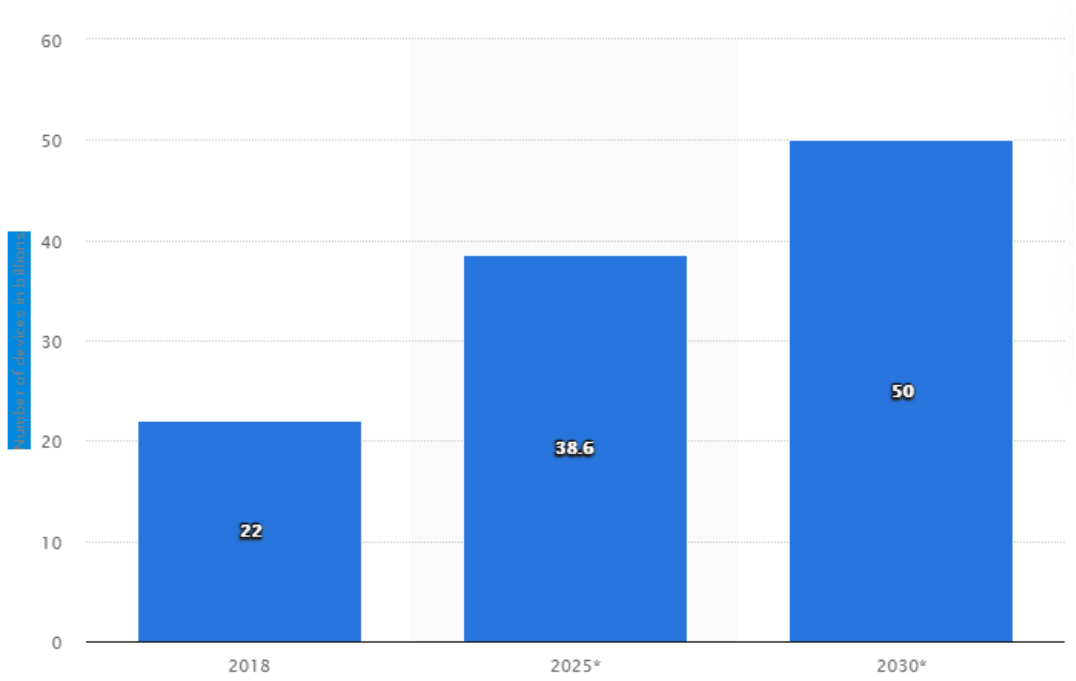
Στην εργασία αυτή το ειδικό λογισμικό που θα ελέγχει την ενεργειακή κατάσταση ενός κτιρίου είναι μία εφαρμογή συνδεδεμένη με το IoT Integration Middleware.

ΚΕΦΑΛΑΙΟ 3

3.1 IOT GATEWAYS - ΓΕΝΙΚΑ

Στις μέρες μας αυξάνεται η επιθυμία να συνδεθούν τα πάντα στο Διαδίκτυο. Η σύνδεση αυτή δεν έχει να κάνει μόνο με την αποστολή πληροφοριών σε διακομιστές για επεξεργασία και αποθήκευση, αλλά και με τον πλήρη έλεγχο των φυσικών συσκευών μέσω του διαδικτύου.

Οι άνθρωποι θα συνεχίσουν να συνδέουν τις συσκευές τους στο Διαδίκτυο όλο και σε μεγαλύτερο αριθμό και έως το 2030 αναμένεται περισσότερες από 50 δισεκατομμύρια έξυπνες συσκευές να συνδεθούν στο Internet



Εικόνα 3.1: Οι συνδεδεμένες συσκευές στο διαδίκτυο των πραγμάτων από το 2018 μέχρι το 2030

Μέχρι το τέλος του 2018, υπήρχαν περίπου 22 δισεκατομμύρια συνδεδεμένες συσκευές διαδικτύου (IoT) σε όλο τον κόσμο.

Ένα αυξανόμενο μερίδιο των ηλεκτρονικών συσκευών που παράγονται σε όλο τον κόσμο κατασκευάζονται με σύνδεση στο Διαδίκτυο.

Οι προβλέψεις δείχνουν ότι έως το 2030 περίπου 50 δισεκατομμύρια από αυτές τις συσκευές IoT θα χρησιμοποιούνται σε όλο τον κόσμο, δημιουργώντας ένα τεράστιο δίκτυο διασυνδεδεμένων συσκευών που καλύπτουν τα πάντα, από smartphone έως συσκευές κουζίνας.

Το Διαδίκτυο των πραγμάτων ή το IoT αναφέρεται απλώς στο γενικό δίκτυο που δημιουργήθηκε από δισεκατομμύρια συσκευές και μηχανές συμβατές με το Διαδίκτυο που μοιράζονται δεδομένα και πληροφορίες σε όλο τον κόσμο [21].

Το IoT σχεδιάστηκε για να διαδραματίζει σπουδαίο ρόλο βελτιώνοντας την ποιότητα της ζωής και τις εφαρμογές του υπάρχουν σε πολλές από τις καθημερινές μας εμπειρίες, όπως η μεταφορά, η υγειονομική περίθαλψη και ο βιομηχανικός αυτοματισμός. Το IoT έχει τη δυνατότητα να μετατρέψει μια απλή φυσική συσκευή σε έξυπνη, χρησιμοποιώντας την ενσωματωμένη τεχνολογία για σύνδεση στο διαδίκτυο και την υπολογιστική ισχύ.

Χρησιμοποιώντας τους διαθέσιμους αισθητήρες και ενεργοποιητές οι συσκευές μπορούν πλέον να ανταλλάσουν πληροφορίες τόσο με άλλες συσκευές αλλά και με ανθρώπους.

Αυτό θα έχει ως αποτέλεσμα να έχουμε μια μεγάλη έκρηξη που θα προέρχεται από συσκευές που συνδέονται με τον ιστό που δεν είχαν συνδεθεί στο παρελθόν, δεν υπήρχαν, ή χρησιμοποιούν πλέον τη σύνδεσή τους ως βασικό χαρακτηριστικό [20].

Τι είναι η IoT Gateway?

Με απλά λόγια, μια πύλη IoT είναι μια φυσική συσκευή ή μια εικονική πλατφόρμα που συνδέει αισθητήρες, μονάδες IoT και έξυπνες συσκευές στο cloud [23,24].

Οι πύλες χρησιμεύουν και για την ασύρματη πρόσβασης ώστε να παρέχουν στις συσκευές IoT πρόσβαση στο Διαδίκτυο.

Παρόλο που ο παραπάνω ορισμός δείχνει η IoT Gateway να μοιάζει με ένα απλό δρομολογητή, επιτρέποντας την επικοινωνία μεταξύ διαφορετικών πρωτοκόλλων και συσκευών.

Η IoT Gateway μπορεί συλλέγει τεράστια δεδομένα από πολλές συνδεδεμένες συσκευές και αισθητήρες, από οποιοδήποτε τοπικό σύστημα IoT.

Οι πύλες IoT λαμβάνουν επίσης πληροφορίες από το cloud, που αποστέλλονται πίσω σε συσκευές επιτρέποντας την αυτόνομη διαχείριση συσκευών στο πεδίο.

Η πύλη μπορεί να προ επεξεργάζεται τα δεδομένα προτού τα μεταβιβάσει σε πλατφόρμες cloud, έτσι μειώνεται η επεξεργασία στο Cloud.[27]

Χρησιμοποιείται συνήθως μεταξύ συσκευών με σύνδεση M2M μεταξύ τους και για τη σύνδεση συσκευών clients (π.χ. πελατών) με το Διαδικτύου.

Ο βασικός στόχος της πύλης είναι να διευθετήσει την ετερογένεια μεταξύ διαφορετικών δικτύων τελικού σημείου και του Διαδικτύου, να ενισχύσει τη διαχείριση των δικτύων τελικού σημείου και να γεφυρώσει το παραδοσιακό Διαδίκτυο με το δίκτυο τελικού σημείου [22].

Οι ειδικές εφαρμογές υλικού IoT gateways χρησιμοποιούνται για τη σύνδεση των συσκευών και χρηστών στο δίκτυο, επιτρέποντας τη μετατροπή δεδομένων μεταξύ των πρωτοκόλλων επικοινωνίας.

Η πύλη υποστηρίζει διαφορετικούς τύπους κόμβων αισθητήρων, πολλαπλά πρωτόκολλα επικοινωνίας, ασύρματα ή ενσύρματα, και παρέχει ένα σύνολο ενοποιημένων πληροφοριών για την εφαρμογή ή τον χρήστη, καθιστώντας αυτά μόνο υπεύθυνα για την επεξεργασία δεδομένων.

Η κύρια πρόκληση για τη δημιουργία μιας πύλης IoT είναι η έλλειψη προτύπων, καθώς κάθε κόμβος αισθητήρα μπορεί να επικοινωνεί με διαφορετικό πρωτόκολλο που δεν είναι συμβατό για άλλους.

Αυτό καθιστά την ανάπτυξη μιας Gateway γενικού σκοπού μια περίπλοκη εργασία. Αυτός είναι και ο λόγος που δημιουργούνται πύλες για συγκεκριμένες εφαρμογές και υποστηρίζουν περιορισμένο αριθμό πρωτοκόλλων [20].

Ωστόσο, όλες έχουν τις ίδιες βασικές απαιτήσεις:

- **υλικό χαμηλού κόστους**
- **εύκολη εφαρμογή και επεκτασιμότητα και υποστήριξη επιπέδου εφαρμογής**

Το Διαδίκτυο των πραγμάτων (IoT) θα περιλαμβάνει νέες συσκευές ειδικά σχεδιασμένες για συμβατότητα IoT και συστήματα που είναι ήδη σε ισχύ σήμερα και λειτουργούν εκτός των δικτύων IoT.

Ωστόσο, για τη δημιουργία δικτύων cloud διασυνδεδεμένων συσκευών απαιτείται ένα μέσο για τη σύνδεση συσκευών που δεν βασίζονται σε IP χωρίς να χρειάζεται να επιβαρυνθεί το κόστος μιας πλήρους διεπαφής Ethernet ή WiFi.

Αυτό μπορεί να επιτευχθεί με τη χρήση πυλών IoT που γεφυρώνουν αυτές τις συσκευές στο Διαδίκτυο στο πλαίσιο πραγματικών εφαρμογών [22].

Η πύλη IoT παίζει πολύ σημαντικό ρόλο στο Διαδίκτυο των πραγμάτων. Είναι ένας βασικός εξοπλισμός για την ενσωμάτωση όλων των συσκευών στο διαδίκτυο.

Όχι μόνο μπορεί να καλύψει τις ανάγκες πρόσβασης στην επικοινωνία μικρής απόστασης, την υλοποίηση και τη σύνδεση δημόσιου δικτύου στην τοπική περιοχή, αλλά μπορεί επίσης να ολοκληρώσει τις λειτουργίες προώθησης και ελέγχου, ανταλλαγής σηματοδοσίας και κωδικοποίησης και αποκωδικοποίησης, και η διαχείριση τερματικών, ο έλεγχος ταυτότητας και άλλες λειτουργίες εξασφαλίζουν την ποιότητα και την ασφάλεια. [25].

Σε τι διαφέρουν οι πύλες IoT από τους δρομολογητές;

Σε αντίθεση με τους δρομολογητές, οι πύλες IoT μπορούν να ενσωματώσουν δεδομένα από συσκευές που επικοινωνούν με διάφορα πρωτόκολλα δικτύου, Wi-Fi, LoRA, Ethernet, Bluetooth, ZigBee και πολλά άλλα.

Οι βιομηχανικές πύλες IoT προσφέρουν επίσης περισσότερες βιομηχανικές διεπαφές από τους δρομολογητές, συμπεριλαμβανομένων των RS485, RS232, USB, I2C, SPI.

Προσφέρουν επίσης προσαρμόσιμο υλικο ή λογισμικό όταν απαιτείται για την προσαρμογή της πύλης στην απαιτούμενη εφαρμογή. [27]

Πρωτόκολλα υποστήριξης των Gateways

Οι IoT Gateways έχουν τη δυνατότητα να υποστηρίξουν διαφορετικά πρωτόκολλα επικοινωνίας μεταξύ των συνδεδεμένων σε αυτές συσκευών και της πλατφόρμας IoT.

Αυτό σημαίνει πως πάνω σε μια IoT Gateway μπορούν να συνδεθούν συστήματα Modbus, MQTT ή HTTP συσκευές, συσκευές bluetooth κλπ.

Παρακάτω βλέπουμε ορισμένα από τα πρωτόκολλα που υποστηρίζονται και το είδος των συσκευών που μπορούν να συνδεθούν σε μία IoT Gateway: [13]

- **Σύνδεση MQTT** για έλεγχο, διαμόρφωση και συλλογή δεδομένων από συσκευές IoT που είναι συνδεδεμένες σε εξωτερικούς brokers MQTT χρησιμοποιώντας υπάρχοντα πρωτόκολλα.
- **Σύνδεση OPC-UA** για τη συλλογή δεδομένων από συσκευές IoT που είναι συνδεδεμένες σε διακομιστές OPC-UA.
- **Σύνδεση Modbus** για τη συλλογή δεδομένων από συσκευές IoT που συνδέονται μέσω πρωτοκόλλου Modbus.
- **Σύνδεση BLE** για τη συλλογή δεδομένων από συσκευές IoT που είναι συνδεδεμένες χρησιμοποιώντας Bluetooth χαμηλής ενέργειας.
- **HTTP σύνδεση** για τη συλλογή δεδομένων από συσκευές IoT που υποστηρίζουν το πρωτόκολλο HTTP .
- **Σύνδεση CAN** για τη συλλογή δεδομένων από συσκευές IoT που συνδέονται μέσω πρωτοκόλλου CAN.
- **Σύνδεση BACnet** για τη συλλογή δεδομένων από συσκευές IoT που συνδέονται μέσω πρωτοκόλλου BACnet.
- **Σύνδεση ODBC** για τη συλλογή δεδομένων από βάσεις δεδομένων ODBC.
- **REST σύνδεση** για τη δημιουργία endpoints και τη συλλογή δεδομένων από εισερχόμενα αιτήματα HTTP.
- **Σύνδεση SNMP** για τη συλλογή δεδομένων από διαχειριστές SNMP.
- **Προσαρμοσμένη σύνδεση** για τη συλλογή δεδομένων από συσκευές IoT που συνδέονται με διαφορετικά πρωτόκολλα. (Μπορεί δηλαδή ο χρήστης να δημιουργήσει μια ειδική εφαρμογή για το πρωτόκολλο που απαιτείται.

3.2 Η ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΜΙΑΣ IoT GATEWAY

Για να κατανοήσουμε πραγματικά τη λειτουργία των IoT Gateways, σε τι διαφέρουν από τους δρομολογητές και σε τι από τους αισθητήρες και τις συσκευές αναλύουμε παρακάτω τα βασικά στοιχεία από τα οποία αποτελείται μια IoT Gateway.

Hardware - Το hardware πύλης IoT περιλαμβάνει έναν μικροεπεξεργαστή ή έναν ελεγκτή, αναλόγως με την ταχύτητα επεξεργασίας και τη μνήμη που απαιτείται, μια μονάδα σύνδεσης (Cellular, Ethernet, Wi-Fi, Bluetooth κ.λπ.), αισθητήρες IoT και κύκλωμα.

Λειτουργικό σύστημα - Το λειτουργικό σύστημα είναι λογισμικό που σχετίζεται με το Hardware της πύλης και άλλες εφαρμογές στη συσκευή. Η επιλογή ενός λειτουργικού συστήματος όπως Java, Linux, RTOS κ.λπ. εξαρτάται από την εφαρμογή της πύλης.

Αφαιρούμενο υλικό - Το επίπεδο αφαίρεσης επιτρέπει στο λογισμικό να αναπτύσσεται και να ελέγχεται ανεξάρτητα από το υλικό. Αυτό προσθέτει γενικά ευελιξία καθώς και ευελιξία στο σχεδιασμό εφαρμογών και διευκολύνει τις ενημερώσεις λογισμικού και την εξέλιξη.

Προγράμματα οδήγησης αισθητήρων και ενεργοποιητών - Αυτό το επίπεδο χρησιμεύει ως διεπαφή μεταξύ της συσκευής και των αισθητήρων και των μονάδων.

Διαχείριση και διαμόρφωση συσκευών - Οι πύλες IoT πρέπει να παρακολουθούν όλες τις συνδεδεμένες συσκευές και αισθητήρες με τους οποίους επικοινωνούν. Αυτό το επίπεδο παρακολουθεί και διαχειρίζεται τις διαμορφώσεις, τις ρυθμίσεις και τις ιδιότητες των αισθητήρων και των συνδεδεμένων συσκευών στο σύστημα του.

Ασφάλεια - Η ασφάλεια αποτελεί βασικό στοιχείο στην αρχιτεκτονική πύλης. Αυτό το επίπεδο διασφαλίζει ότι οι πύλες έχουν αξιόπιστες ταυτότητες, ισχυρή κρυπτογράφηση και σχήματα ελέγχου ταυτότητας κρυπτογράφησης. Παρέχει μια ασφαλή εκκίνηση για προστασία των συσκευών από εισβολή και διασφάλιση της ακεραιότητας και της εμπιστευτικότητας των δεδομένων.

Ενημερώσεις υλικολογισμικού μέσω του αέρα (FOTA) - Η ενημέρωση του υλικολογισμικού της συσκευής και η δυνατότητα επιδιόρθωσης και επιδιορθώσεων ασφαλείας για την άμυνα από τις συνεχώς εξελισσόμενες απειλές είναι υψίστης σημασίας για τη διατήρηση της ακεραιότητας της συσκευής. Αυτό το επίπεδο διασφαλίζει ότι η διαχείριση των ενημερώσεων Firmware Over The Air (FOTA) γίνεται με ασφάλεια και αποτελεσματικότητα για τη διατήρηση της μνήμης της συσκευής, της ισχύος και του εύρους ζώνης δικτύου.

Πρωτόκολλα επικοινωνίας - Τα πρωτόκολλα μιας IoT Gateway επιλέγονται ανάλογα με την ποσότητα και τη συχνότητα των δεδομένων που κοινοποιούνται στο cloud. Οι πύλες

πρέπει να συνδεθούν μέσω μιας κυψελοειδούς μονάδας (5G / 4G / 3G), Ethernet και / ή Wi-Fi, αλλά το υποκείμενο επίπεδο πρωτοκόλλου επικοινωνίας είναι συνήθως πρωτόκολλο TCP IP.

Διαχείριση δεδομένων - Οι πύλες IoT διαχειρίζονται δεδομένα από αισθητήρες και συνδεδεμένες συσκευές και δεδομένα που προέρχονται από το σύννεφο. Το επίπεδο διαχείρισης δεδομένων ελέγχει τη ροή, το φιλτράρισμα και την αποθήκευση δεδομένων και παρέχει έλεγχο κυκλοφορίας δεδομένων για την ελαχιστοποίηση των καθυστερήσεων και τη διασφάλιση της πιστότητας της συσκευής.

Διαχείριση συνδεσιμότητας Cloud - Αυτό το επίπεδο είναι υπεύθυνο για απρόσκοπτη, ασφαλή συνδεσιμότητα με πλατφόρμες cloud και έλεγχο ταυτότητας συσκευής και cloud.

Προσαρμοσμένες εφαρμογές λογισμικού - Οι πύλες IoT ενσωματώνουν προσαρμοσμένο λογισμικό για τη διαχείριση συγκεκριμένων αναγκών εφαρμογών. Αυτό το επίπεδο αλληλεπιδρά με όλα τα άλλα επίπεδα για αποτελεσματική, ασφαλή και αποτελεσματική διαχείριση αναγκών δεδομένων ειδικά για την εφαρμογή IoT.

Μεταφορά δεδομένων Gateway - Αυτό το επίπεδο ελέγχει τη σύνδεση της πύλης στο Διαδίκτυο χρησιμοποιώντας είτε μόντεμ 5G / 4G / 3G / GPRS ή μονάδα IoT, ethernet ή Wi-Fi. Αναλύει επίσης και καθορίζει ποια δεδομένα πρέπει να κοινοποιούνται στο cloud και ποια δεδομένα πρέπει να αποθηκεύονται προσωρινά για επεξεργασία εκτός σύνδεσης για εξοικονόμηση ενέργειας επεξεργασίας και χρεώσεων προγράμματος δεδομένων.[27]

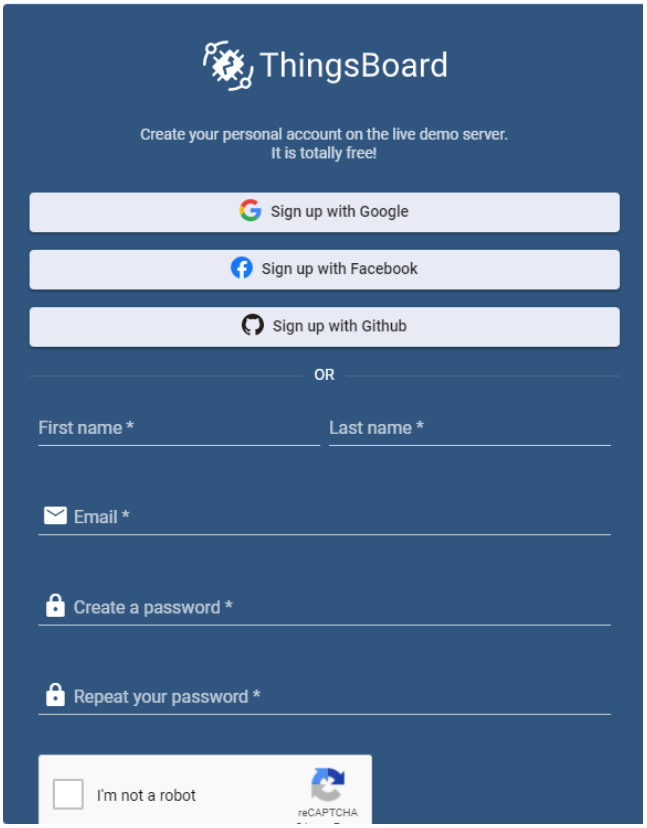
ΚΕΦΑΛΑΙΟ 4

4.1 Η IoT GATEWAY ΤΗΣ ΠΛΑΤΦΟΡΜΑΣ THINGSBOARD

Εγκατάσταση και λειτουργία της IoT GateWay που διαθέτει η πλατφόρμα ThingsBoard

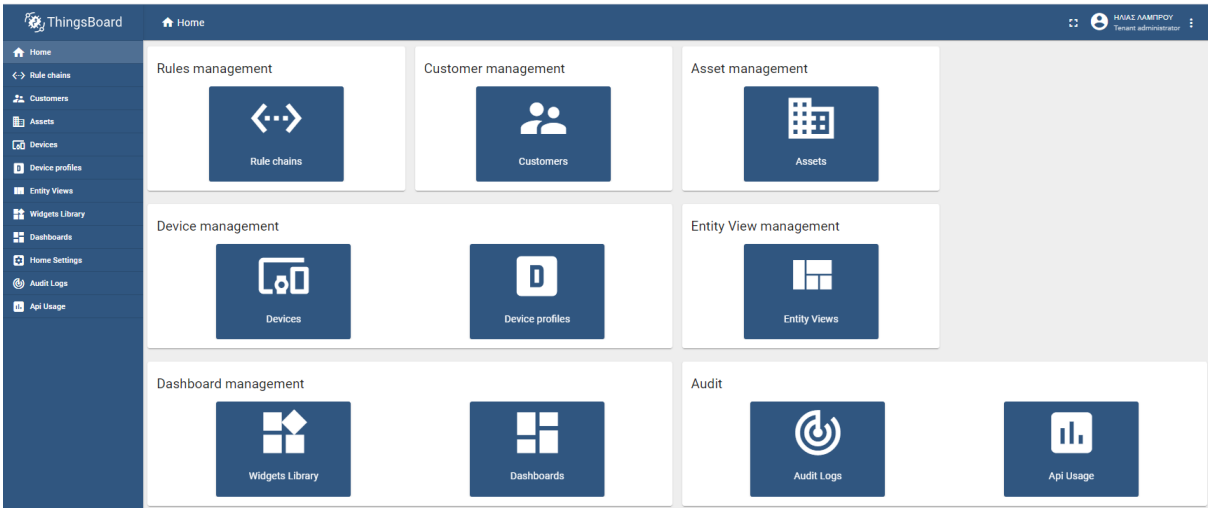
Για την εγκατάσταση ακολουθήθηκαν οι οδηγίες που δίνονται από την πλατφόρμα: https://thingsboard.io/docs/iot-gateway/getting-started/

Αρχικά κάναμε ένα λογαριασμό στην demo έκδοση της πλατφόρμας (εικόνα 4.1)
https://demo.thingsboard.io



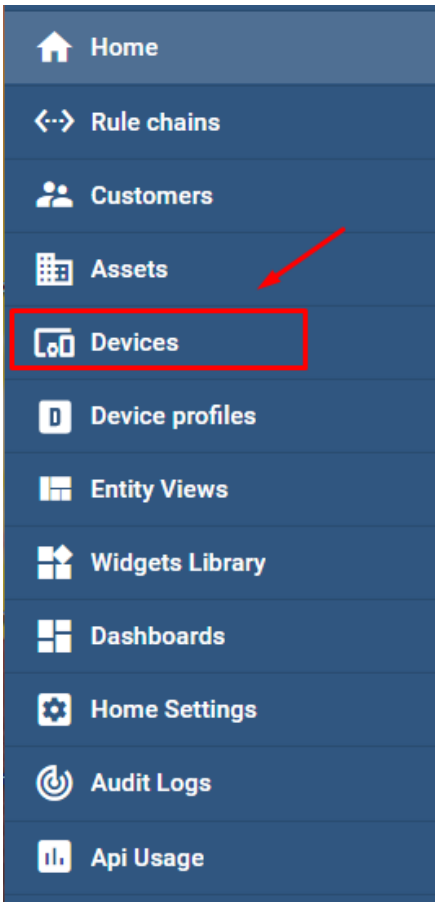
Εικόνα 4.1 Το παράθυρο δημιουργίας λογαριασμού στην πλατφόρμα Thingsboard

Μετά τη δημιουργία του λογαριασμού εισερχόμαστε στο περιβαλλον της πλατφόρμα ως Tenant Administrator



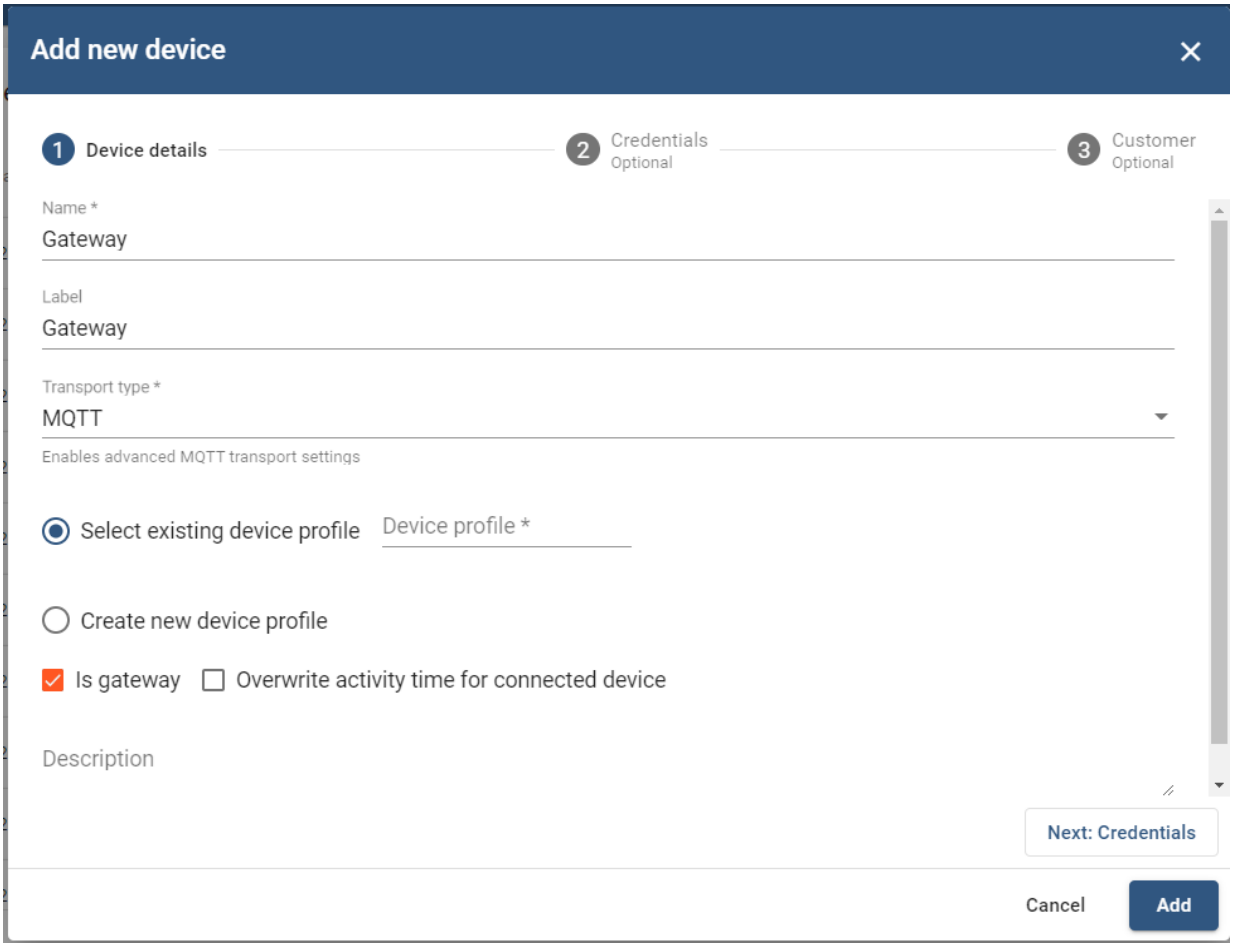
Εικόνα 4.1 Το γραφικό περιβάλλον της πλατφόρμας Thingsboard

Στο μενού της πλατφόρμας επιλέγουμε “Devices” για να εισάγουμε την Gateway που θέλουμε να δημιουργήσουμε ως “Device”



Εικόνα 4.3 Το μενού Home της πλατφόρμας Thingsboard

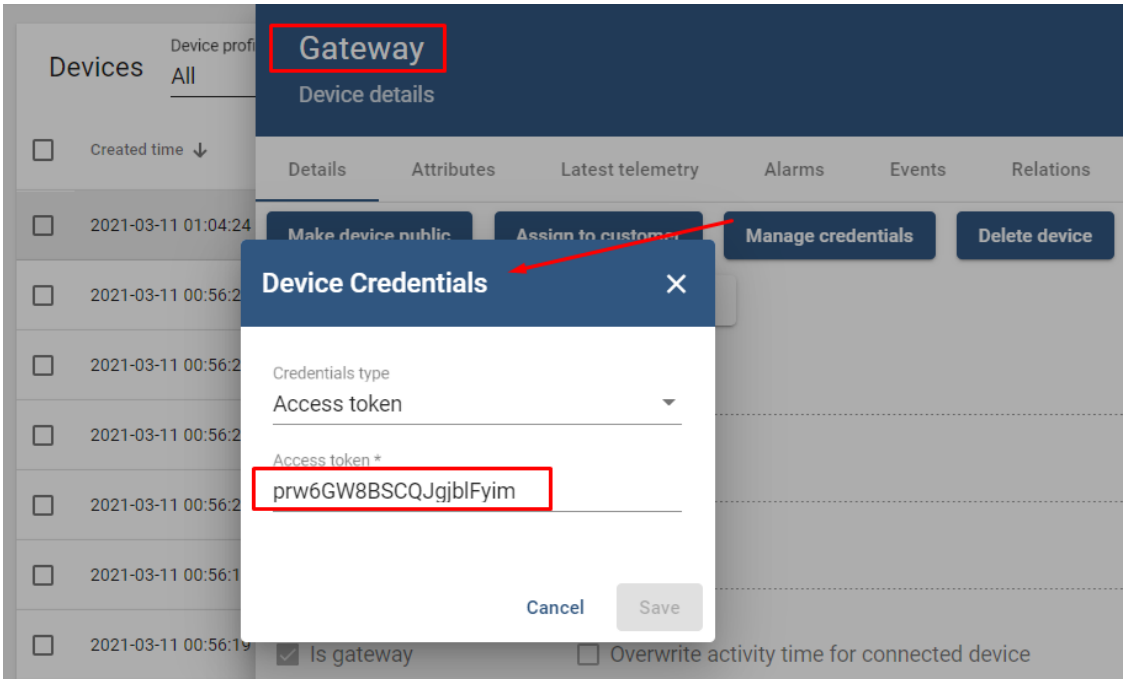
Θα δούμε να υπάρχουν κάποια έτοιμα “Devices” από την πλατφόρμα ως παραδείγματα. Θα πιέσουμε το εικονίδιο “+” για να εισάγουμε τη δική μας Gateway ως καινούργια συσκευή (Add new Device).



Εικόνα 4.4 Το παράθυρο προσθήκης νέας συσκευής στην πλατφόρμα Thingsboard

Αφού κάνουμε τις παραπάνω ρυθμίσεις προσθέτουμε την Gateway στις υπάρχουσες συσκευές. Ως “Transport type” επιλέξαμε το πρωτόκολλο MQTT.

Επιλέγοντας τη νέα συσκευή εμφανίζεται το παράθυρο “Device details” στο οποίο επιλέγοντας “Manage credentials” μπορούμε να δούμε και να αντιγράψουμε το “Access token” της συσκευής.

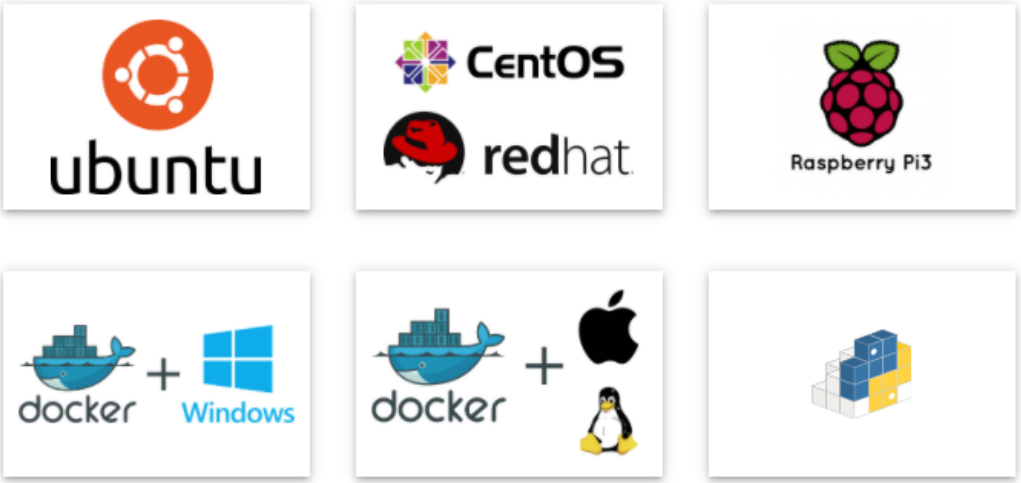


Εικόνα 4.5 Το access token που χρησιμοποιήσαμε για τη σύνδεση της Gateway

Κάθε συσκευή που δημιουργούμε στην πλατφόρμα παίρνει αυτόματα ένα μοναδικό Access token το οποίο μπορούμε στη συνέχεια να το αλλάξουμε.

Στη συνέχεια θα εγκαταστήσουμε την Gateway στο περιβάλλον του υπολογιστή μας:

Η "Gateway" της πλατφόρμας Thingsboard διατίθεται σε διάφορες εκδόσεις :



Εικόνα 4.6 Οι διαθέσιμες εκδόσεις της πλατφόρμας Thingsboard

a. Έκδοση για ubuntu
b. Έκδοση για CentOS or RHEL
c. Έκδοση για Raspberry η οποία είναι στην ουσία μια έκδοση για Ubuntu Server 18.04 LTS
d. Έκδοση για Windows με χρήση Docker

e. Έκδοση για Mac OS ή Linux ή με χρήση Docker
f. και μία έκδοση ως Python Module

Για την πραγματοποίηση των δοκιμών της παρούσας εργασίας χρησιμοποιήσαμε ένα ηλεκτρονικό υπολογιστή με λειτουργικό Windows 10 και επιλέξαμε την έκδοση Windows+Docker.

Στον παρακάτω πίνακα βλέπουμε τα τεχνικά χαρακτηριστικά του Η/Υ της δοκιμής

CPU	Ryzen 7 3800X - 4,5GHz
Cores/Threads	8/16
Ram	32GB
OS	Windows 10 Pro

Ακολουθώντας τις οδηγίες:
https://thingsboard.io/docs/iot-gateway/install/docker-installation/
το πρώτο που έπρεπε να κάνουμε ήταν η εγκατάσταση του Docker:
https://docs.docker.com/engine/install/

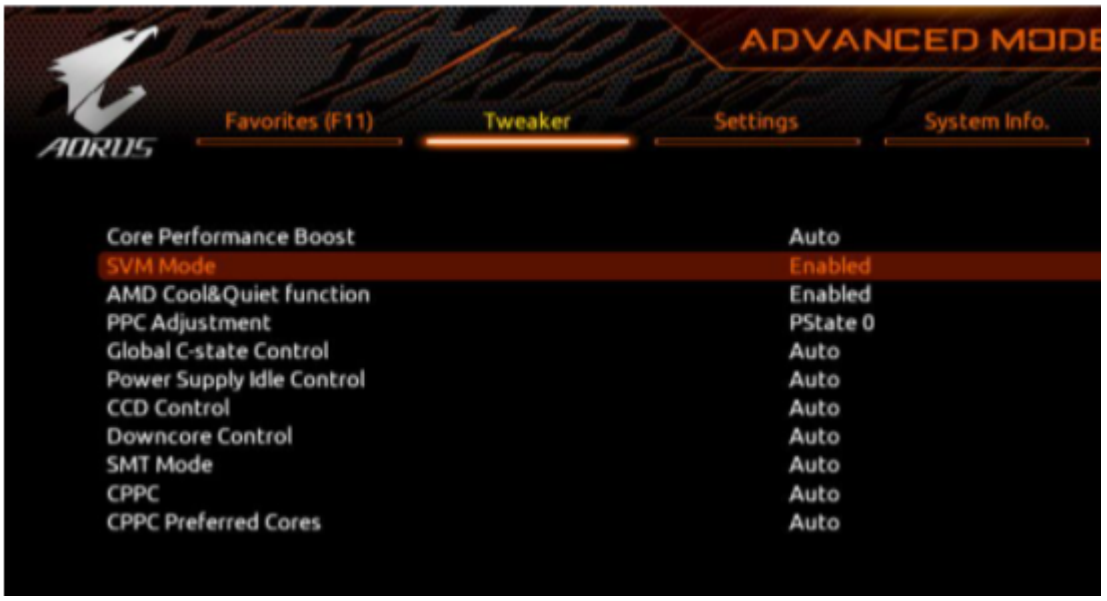
Μέσα από την επίσημο ιστότοπο του Docker επιλέξαμε την εγκατάσταση για Docker Desktop for Windows και στη συνέχεια από το Docker Hub κατεβάσαμε το Docker Desktop for Windows.

Εικόνα 4.7 Οι διαθέσιμες επιλογές για την εγκατάσταση του Docker Desktop for Windows

Αφού κατέβηκε το αρχείο δεν έγινε αμέσως εγκατάσταση αλλά πρώτα ενεργοποιήθηκε το “Hardware Virtualization” στη μητρική πλακέτα του υπολογιστή.

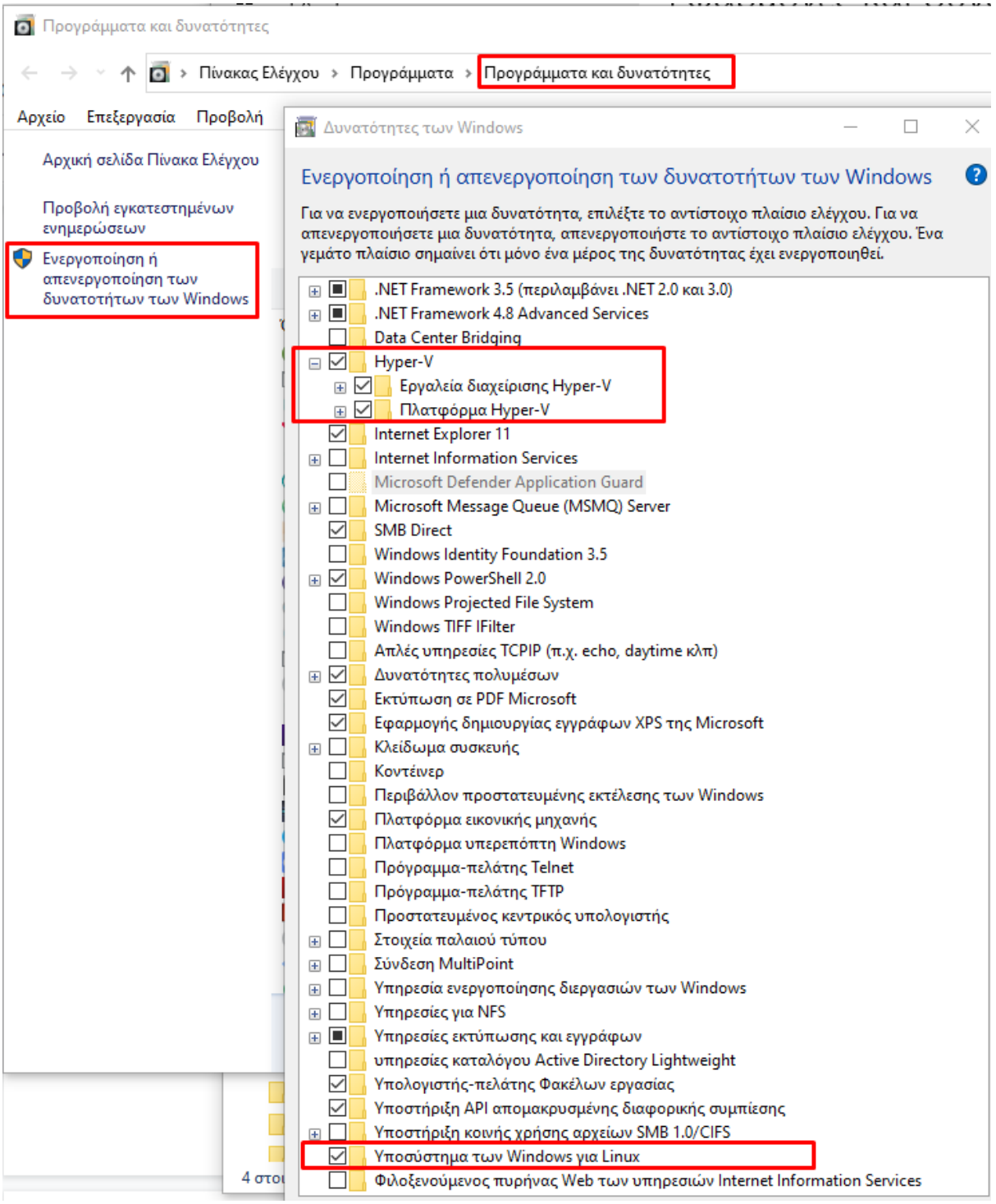
Για να γίνει αυτό μέσα από το BIOS της μητρικής για την πλατφόρμα της AMD ενεργοποιήθηκε η επιλογή:

Advanced CPU Settings / SVM Mode



Εικόνα 4.8 Η επιλογή SVM Mode έπρεπε να ενεργοποιηθεί για την υποστήριξη του hardware virtualization

Και μετά την επανεκκίνηση έγιναν οι παρακάτω ρυθμίσεις στο λειτουργικό σύστημα Windows 10 pro.

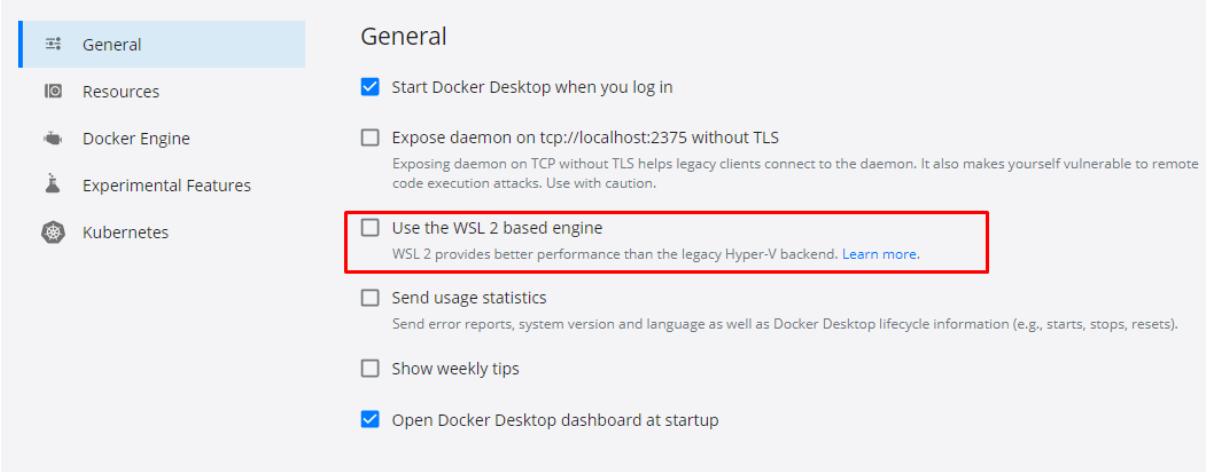


Εικόνα 4.9 Οι πρόσθετες δυνατότητες των Windows που ενεργοποιήθηκαν

Μέσα από το παράθυρο “Ενεργοποίηση ή απενεργοποίηση των δυνατοτήτων των Windows” ενεργοποιήθηκαν η “Πλατφόρμα Hyper-V” και τα “Εργαλεία διαχείρισης Hyper-V” καθώς και το “Υποσύστημα των Windows για Linux”

Μετά την ενεργοποίηση του Hyper-V έγινε επανεκκίνηση και στη συνέχεια τρέξαμε το αρχείο για την εγκατάσταση του Docker.

Μετά την εγκατάσταση του Docker κάναμε τις παρακάτω ρυθμίσεις:



Εικόνα 4.10 Η παραμετροποίηση του Docker

Πολύ βασικό για τα Windows Pro η μη ενεργοποίηση του WSL 2 based engine

Στη συνέχεια μέσα από "Command line" εκτελέσαμε την παρακάτω εντολή για την εγκατάση του "container" για την Gateway:

docker run -it -v C:/Users/Baal/tb-gateway/config:/thingsboard_gateway/config -v C:/Users/Baal/tb-gateway/extensions:/thingsboard_gateway/extensions -v C:/Users/Baal/tb-gateway/logs:/thingsboard_gateway/logs --name tb-gateway --restart always thingsboard/tb-gateway

Στο φάκελο του χρήστη των Windows δημιουργήθηκε ο φάκελος "tb-gateway" ο οποίος περιέχει τα απαραίτητα αρχεία για να μπορέσουμε να ρυθμίσουμε την Gateway.

Ανοίγοντας τώρα το παράθυρο του Docker βλεπουμε το "container" tb_gateway να είναι σε κατάσταση "RUNNING"

Ρυθμίσεις της Gateway

Για να τρέξουμε την Gateway από το command line δίνουμε την παρακάτω εντολή:

docker start tb-gateway

Για σταμάτημα:

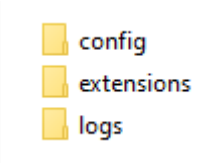
docker stop tb-gateway

Στην εντολή του docker που δώσαμε παραπάνω ορίσαμε τρεις βασικούς φακέλους στο περιβαλλον windows οι οποίοι κάνουν mount τρεις αντίστοιχους φακέλους της gateway.

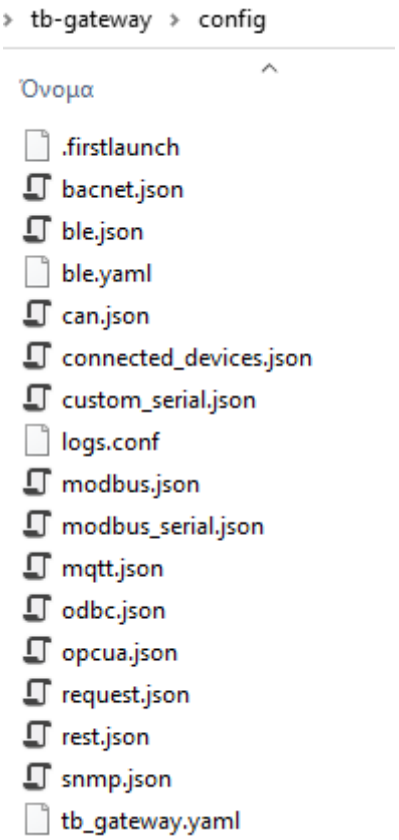
Στους φακέλους αυτούς βρίσκονται τα βασικά αρχεία με τα οποία θα ρυθμίσουμε την Gateway.

https://thingsboard.io/docs/iot-gateway/configuration/

Υπάρχουν βασικά τρεις φάκελοι:



Ο φάκελος που μας ενδιαφέρει στην παρούσα εργασία είναι είναι ο φάκελος “config” ο οποίος περιέχει τα δύο βασικά αρχεία που θα πρέπει να ρυθμίσουμε να να πετύχουμε τόσο την επικοινωνία με τον τοπικό μας broker όσο και τη σωστή μετατροπή των μηνυμάτων.



Εικόνα 4.11 Η λίστα με τα αρχεία παραμετροποίησης της IoT Gateway

Το αρχείο **tb_gateway.yaml**

είναι το κύριο αρχείο διαμόρφωσης της gateway και χρησιμοποιείται για να ορίσει την διεύθυνση της πλατφόρμας Thingsboard με την οποία θα συνδεθεί η Gateway.

Παρακάτω βλέπουμε πως διαμορφώθηκε το το αρχείο **tb_gateway.yaml** για τις ανάγκες της παρούσας εργασίας:

```
thingsboard:
  host: demo.thingsboard.io
  port: 1883
#  remoteShell: false
#  remoteConfiguration: false
  security:
    accessToken: 5MK2sOGmCQlHmZMBu1c9
  qos: 1
storage:
  type: memory
  read_records_count: 10
  max_records_count: 1000
#  type: file
#  data_folder_path: ./data/
#  max_file_count: 10
#  max_read_records_count: 10
#  max_records_per_file: 10000
connectors:
  -
    name: MQTT Broker Connector
    type: mqtt
    configuration: mqtt.json
```

Στο πεδίο host διαμορφώσαμε την gateway έτσι ώστε να συνδέεται με την ThingsBoard instance ***demo.thingsboard.io***

Στο πεδίο port ορίσαμε η σύνδεσης στον broker να γίνεται στην πόρτα 1883 που είναι η καθορισμένη για συνδέσεις χωρίς κρυπτογράφηση TLS.

Όπως είχε αναφερθεί και αρχικά η πλατφόρμα βλέπει την Gateway σαν ένα Device και για την ασφαλή σύνδεση χρησιμοποιεί ένα AccessToken. Το AccessToken το οποίο αντιγράψαμε παραπάνω όταν προσθέσαμε την Gateway στις συσκευές της πλατφόρμας θα το εισάγουμε εδώ στο αντίστοιχο πεδίο *accessToken*.

Βασικά υποστηρίζονται τρια επίπεδα ασφάλειας.

Access Token	Βασικό επίπεδο ασφάλειας μόνο με Access Token
TLS+Access Token	Ασφάλεια με κρυπτογράφηση TLS με κλειδί που περιέχεται στην gateway και Access Token
TLS_Private Key	Ασφάλεια με κρυπτογράφηση TLS με

	κλειδί που θα καθορίσει ο χρήστης και Access Token

security:
accessToken**: PUT_YOUR_GW_ACCESS_TOKEN_HERE**

caCert**: /etc/thingsboard-gateway/mqttserver.pub.pem**

Σε περίπτωση που θέλουμε να εισάγουμε δικό μας κλειδί θα πρέπει να καθορίσουμε τη διαδρομή στην οποία θα έχουμε αποθηκευμένο το αρχείο στο πεδίο "caCert" όπως φαίνεται στο παραπάνω παράδειγμα

Τα εισερχόμενα μηνύματα στη Gateway μπαίνουν σε μια ουρά μέχρι να αποσταλούν και πρέπει να καθορίσουμε αν αυτά θα αποθηκεύονται στο δίσκο η στη μνήμη.
Στο πεδίο *storage* καθορίσαμε η αποθήκευση αυτή να γίνεται στη μνήμη ορίζοντας type : **memory**.
Στο πεδίο read_records_count δώσαμε την τιμή **10** ώστε να διαβάζονται κάθε φορά 10 μηνύματα πριν αποσταλλούν στη πλατφόρμα.
Στο πεδίο max_records_count ορίζουμε τον μέγιστο αριθμό μηνυμάτων που θα μπορεί η gateway να αποθηκευσει στην τιμή **1000**.

Η gateway από τη μια πλευρά συνδέεται με την πλατφόρμα Thingsboard αλλά από την άλλη μπορεί να κάνει διαφορετικούς τύπους συνδέσεων, από τους οποίους ενεργοποιήσαμε μόνο την MQTT σύνδεση και απενεργοποιήσαμε τις υπόλοιπες.
Κάθε σύνδεση διαθέτει και ένα αρχείο που καθορίζει τις επιμέρους ρυθμίσεις της.

Για τη σύνδεση MQTT το αρχείο που καθορίζει αυτές τις ρυθμίσεις είναι το αρχείο mqtt.json.
Στη συνέχεια θα αναλύσουμε το αρχείο mqtt.json και θα κάνουμε τις κατάλληλες ρυθμίσεις ώστε να συνδεθεί η gateway με τον broker που επικοινωνεί η συσκευή της δοκιμής και να μπορεί να γίνει η μετατροπή των μηνυμάτων στη μορφή που απαιτεί η πλατφόρμα Thingsboard ώστε να γίνει δυνατή στη συνέχεια η κατασκευή του dashboard.
https://thingsboard.io/docs/iot-gateway/config/mqtt/?MqttConverterTypeConfig=json

Το αρχείο mqtt.json περιέχει ένα Json Object το οποίο περιέχει επιμέρους Json Objects και Json Arrays.

Παρακάτω διακρίνουμε τη γενική δομή του αρχείου:

- Section "broker"
    - Subsection "security"
- Section "mapping"
    - topicfilter

    - Subsection “converter”
- Section “connectRequests”
- Section “disconnectRequest”
- Section “attributeUpdates”
- Server side RPC commands

A. Section “broker”

Το πρώτο επιμέρους Json object του αρχείου έχει το label “broker” και περιέχει τις βασικές ρυθμίσεις που αφορούν τον broker με τον οποίο θα συνδεθεί η gateway.

```
{
 "broker": {
  "name":"Default Local Broker",
  "host":"peatrib322.cloud.shiftr.io",
  "port":1883,
  "clientId": "ThingsBoard_gateway",
  "security": {
   "type": "basic",
   "username": "peatrib322",
   "password": "6973403792"
  }
 },
```

Στο δικό μας παράδειγμα τροποποιήσαμε το αρχείο ώστε η σύνδεση να γίνει στον broker της πλατφορματ shiftr.io (peatrib322.cloud.shiftr.io)

Η πόρτα για μη ασφαλείς συνδέσεις ορίστηκε στο 1883.

Για κάθε MQTT συνδεση είναι απαραίτητο ένα client ID. Δώσαμε ως client ID το όνομα ThingsBoard_gateway ώστε από την πλευρά του broker να ξεχωρίζουμε ποια σύνδεση είναι η gateway και ποια ή ποιες οι συσκευές.

Στο τμήμα security η πλατφόρμα μας δίνει τρεις επιλογές:

a. Βασική ασφάλεια: Για τη σύνδεση απαιτείται ένα username και password τα οποία δίνονται από τον broker.
b. Σύνδεση με ανωνυμία: Πολλοί brokers επιτρέπουν την ανώνυμη σύνδεση. Σε αυτή την περίπτωση το πεδίο “security” διαμορφώνεται ως παρακάτω:

```
"security": {
    "type": "anonymous"
  }
```

c. Σύνδεση με προχωρημένη ασφάλεια: Σε αυτή την περίπτωση απαιτούνται ένα CA certificate αρχείο, ένα αρχείο που θα περιέχει ένα Private Key και ένα Certificate αρχείο και θα πρέπει να καθορίσουμε τις διαδρομές των φακέλων στους οποίους έχουμε αποθηκευμένα αυτά τα αρχεία ώστε να χρησιμοποιηθούν από τη gateway για τη σύνδεση στον broker

Παρακάτω βλέπουμε ένα παράδειγμα πως διαμορφώνεται το τμήμα security σε αυτή την περίπτωση:

```
"security":{
  "caCert": "/etc/thingsboard-gateway/ca.pem",
  "privateKey": "/etc/thingsboard-gateway/privateKey.pem",
  "cert": "/etc/thingsboard-gateway/certificate.pem"
 }
```

B. Section “mapping”

Το τμήμα αυτό περιέχει τις ρυθμίσεις για να γίνει η μετατροπή των μηνυμάτων που έρχονται από την πλευρά του broker πριν αποσταλούν στον broker της πλατφόρμας Thingsboard.

Είναι στην πραγματικότητα ένα Json Array του οποίου το κάθε στοιχείο καθορίζει δύο πράγματα, το topic το οποίο θα γίνει subscribe από την gateway προς την πλευρά του broker. Το topic καθορίζεται από την τιμή του πεδίου “topicFilter” και τον τρόπο με τον οποίο θα γίνει η μετατροπή του εισερχόμενου μηνύματος στην μορφή που απαιτεί το Thingsboard, η οποία μορφή καθορίζεται από το Json του πεδίου “converter”.

Subsection “topicFilter”:

Parameter	Default value	Description
topicFilter	/sensor/data	Topic address for subscribing.

Εικόνα 4.12 Το πεδίου “topicFilter” καθορίζει το topic το οποίο που θα γίνει subscribe από την Gateway

Στο πεδίο αυτό εισάγουμε το topic στο οποίο θέλουμε η gateway να κάνει subscribe από την πλευρά του broker ώστε να λαμβάνει τις ενημερώσεις των τιμών.

Το topic μπορεί να είναι είτε ένα αποκλειστικό topic είτε ένα topic το οποίο περιλαμβάνει τους ειδικούς χαρακτήρες ‘+’ και ‘#’ για να γίνει subscribe σε ένα πολλαπλό topic.

Στις περιπτώσεις που από την πλευρά του broker είναι συνδεδεμένες πολλές παρόμοιες συσκευές το αναγνωριστικό συσκευής Device ID εισάγεται ως ένα τμήμα του topic. Κάτι τέτοιο συμβαίνει και με τη συσκευή του παραδείγματος που εξετάζουμε σε αυτή την εργασία.

Σε άλλες περιπτώσεις το αναγνωριστικό συσκευής εισάγεται μέσα στο μήνυμα Payload.

Αναλόγως την περίπτωση θα πρέπει να διαμορφώσουμε κατάλληλα αυτό το πεδίο.

όπως φαίνεται στον παρακάτω πίνακα:

Example Name	Topic	Topic Filter	Payload	Comments
Example 1	/sensor/data	/sensor/data	{"serialNumber": "SN-001", "sensorType": "Thermometer", "sensorModel": "T1000", "temp": 42, "hum": 58}	Device Name is part of the payload
Example 2	/sensor/SN-001/data	/sensor/+/data	{ "sensorType": "Thermometer", "sensorModel": "T1000", "temp": 42, "hum": 58}	Device Name is part of the topic

Εικόνα 4.13 Παραδείγματα παραμετροποίησης της IoT Gateway

Στο example1 το αναγνωριστικό της συσκευής δεν περιέχεται στο topic, ενώ στο example 2 περιέχεται σε αυτό. Προσθέτοντας το '+' στη θέση του αναγνωριστικού 'SN=001' η gateway θα κάνει subscribe στα 'sensor data' από όλες τις συνδεδεμένες στον broker συσκευές.

Βέβαια στη συνέχεια κατα τη δημιουργία του προς αποστολή στο Thingsboard μηνύματος θα πρέπει να διαβαστεί το αναγνωριστικό συσκευής από το topic του εισερχόμενου μηνύματος και να εισαχθεί στο προς αποστολή Json payload.

Subsection "converter":

Το τμήμα αυτό περιέχει τη διαμόρφωση για την επεξεργασία των μηνυμάτων.

Υποστηρίζονται δύο τύποι μετατροπής οι οποίοι θα αναλυθούν παρακάτω:

a. Json
b. Custom

a. Μετατροπή με χρήση Json μετατροπέα ο οποίος είναι και ο προκαθορισμένος από την gateway.

   Στον παρακάτω πίνακα βλέπουμε την περιγραφή της χρήσης του κάθε πεδίου.

Parameter	Default value	Description
type	**json**	Provides information to connector that default converter will be uses for converting data from topic.
deviceNameJsonExpression	**${serialNumber}**	Simple JSON expression, uses for looking device name in the incoming message (parameter "serialNumber" will be used as device name).
deviceTypeJsonExpression	**${sensorType}**	Simple JSON expression, uses for looking device type in the incoming message (parameter "sensorType" will be used as device type).
timeout	**60000**	Timeout for triggering "Device Disconnected" event
attributes		This subsection contains parameters of the incoming message, that will be interpreted as attributes for the device.
... type	**string**	Type of incoming data for a current attribute.
... key	**model**	Attribute name, that will sends to ThingsBoard instance.
... value	**${sensorModel}**	Simple JSON expression, uses for looking value in the incoming message, that will send to ThingsBoard instance as value of key parameter.
timeseries		This subsection contains parameters of the incoming message, that will be interpreted as telemetry for the device.
... type	**double**	Type of incoming data for a current telemetry.
... key	**temperature**	Telemetry name, that will sends to ThingsBoard instance.
... value	**${temp}**	Simple JSON expression, uses for looking value in the incoming message, that will send to ThingsBoard instance as value of key parameter.

Εικόνα 4.14 Στην εικόνα βλέπουμε τις δυνατότητες παραμετροποίησης του τμήματος "converter":

Για το example 1 που αναφέραμε παραπάνω όπου το αναγνωριστικό συσκευή δεν περιλαμβάνεται στο topic η μορφή ενός στοιχείου του Json Array του πεδίου "mapping" έχει ως παρακάτω:

```
{
  "topicFilter": "/sensor/data",
  "converter": {
    "type": "json",
    "deviceNameJsonExpression": "${serialNumber}",
    "deviceTypeJsonExpression": "${sensorType}",
    "timeout": 60000,
    "attributes": [
      {
        "type": "string",
        "key": "model",
        "value": "${sensorModel}"
      }
    ],
    "timeseries": [
      {
        "type": "double",
        "key": "temperature",
        "value": "${temp}"
      },
      {
        "type": "double",
        "key": "humidity",
        "value": "${hum}"
      }
    ]
  }
}
```

Εικόνα 4.15 Το τμήμα "mapping

Στο πεδίο "deviceNameJsonExpression" εισάγουμε το όνομα του αναγνωριστικού της συσκευής που στέλνει το μήνυμα. Σε περίπτωση που το όνομα αυτό είναι ένα πεδίο του εισερχόμενου μηνύματος Json εισάγουμε αυτό σαν μεταβλητή με τη μορφή

"${serialNumber}

όπου το αλφαριθμικό "serialNumber" θα πρέπει να αντικατασταθεί με το πεδίο του payload που περιέχει το αναγνωριστικό συσκευής.

Η gateway σε αυτή την περίπτωση θα διαβάσει την τιμή που περιέχεται στο πεδίο "serialNumber" του εισερχόμενου json μηνύματος και θα τη βάλει στη θέση της μεταβλητής ${serialNumber} πριν κάνει την αποστολή.

Στο πεδίο μπορούμε και εκεί να ορίσουμε μια μεταβλητή για τον τύπο του device εφόσον υπάρχει μέσα στο payload του εισερχόμενου μηνύματος ή να βάλουμε ένα απλώς ένα όνομα αντί για τη μεταβλητή **${sensorType}.**

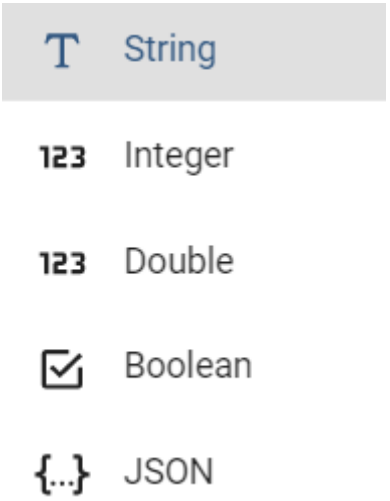
Όπως αναφέραμε παραπάνω οι τιμές που στέλνονται στο Thingsboard μπορεί να είναι είτε timeseries είτε attributes.

Πεδίο "attributes":

Σε περίπτωση που θέλουμε κάποιες τιμές από το εισερχόμενο μήνυμα να τις στείλουμε στο Thingsboard ως device attributes τις προσθέτουμε σε αυτό το πεδίο. Διαφορετικά διαγράφουμε εντελώς αυτό το πεδίο.

Στο πεδίο "**type**" καθορίζουμε αν η τιμή που θα αποσταλεί θα είναι τύπου "string", "int", "double", "boolean" κλπ.

Στον παρακάτω πίνακα βλέπουμε τις αποδεκτές τιμές από την πλατφόρμα:



Εικόνα 4.16 Οι υποστηριζόμενοι τύποι δεδομένων

Στο πεδίο "key" δίνουμε το όνομα με το οποίο θα εμφανίζεται η τιμή στην πλατφόρμα ως attribute.

Στο πεδίο **value** δίνουμε ως όνομα μεταβλητής το όνομα του πεδίου που περιέχει την τιμή στο εισερχόμενο Json όπως κάναμε και πριν για το αναγνωριστικό συσκευής.

Για παράδειγμα εάν η τιμή που θέλουμε να στείλουμε ως attribute στην πλατφόρμα βρίσκεται στο εισερχόμενο μήνυμα Json στο πεδίο "sensorModel" ορίζουμε την τιμή του πεδίου value ως ${sensorModel} όπως βλέπουμε και στο παραπάνω παράδειγμα.

Πεδίο "timeseries":

Η πλατφόρμα επιτρέπει όπως είδαμε παραπάνω την αποστολή τιμών ως timeseries. Σε περίπτωση που θέλουμε κάποιες τιμές από το εισερχόμενο μήνυμα να τις στείλουμε στο Thingsboard ως timeseries τις προσθέτουμε σε αυτό το πεδίο και ακολουθούμε την ίδια διαμόρφωση του πεδίου με τον τρόπο που εφαρμόσαμε για το πεδίο "attributes".

Στην εικόνα 4.17 παρακάτω βλέπουμε ένα παράδειγμα για το πως θα διαμορφωθεί το παρακάτω εισερχόμενο μήνυμα:

Εισερχόμενο μήνυμα

```
topic:   /sensor/data
payload: {
           “serialNumber”: “SN-001”,
           “sensorType”: “Thermometer”,
           “sensorModel”: “T1000”,
           “temp”: 42,
           “hum”: 58
         }
```

Εικόνα 4.17: Διαμόρφωση εισερχόμενου μηνύματος

```
{
    "topicFilter": "/sensor/data",
    "converter": {
      "type": "json",
      "deviceNameJsonExpression": "${serialNumber}",
      "deviceTypeJsonExpression": "${sensorType}",
      "timeout": 60000,
      "attributes": [
        {
          "type": "string",
          "key": "model",
          "value": "${sensorModel}"
        }
      ],
      "timeseries": [
        {
          "type": "double",
          "key": "temperature",
          "value": "${temp}"
        },
        {
          "type": "double",
          "key": "humidity",
          "value": "${hum}"
        }
      ]
    }
  }
```

Εικόνα 4.18: Διαμόρφωση ενός στοιχείου του πίνακα “mapping”

Για το example 2 που αναφέραμε παραπάνω όπου το topic του εισερχόμενου μηνύματος περιλαμβάνει το αναγνωριστικό συσκευής εισάγουμε το topic με τον χαρακτήρα ‘+’ στη θέση του αναγνωριστικού συσκευής

Ειδικότερα στο πεδίο **deviceNameTopicExpression** η μεταβλητή ${serialNumber} αντικαθίσταται από το αλφαριθμιτικό "(?<=sensor\/)(.*?)(?=\/data)”

"**deviceNameTopicExpression**": "(?<=sensor\/)(.*?)(?=\/data)"

Σε αυτό το παράδειγμα το topic έγινε subscribe ως πολλαπλό topic με τη μορφή /sensor/+/data. Σε κάθε εισερχόμενο μήνυμα στη θέση του '+' θα υπάρχει το αναγνωριστικό συσκευής. Για παράδειγμα αν το μήνυμα στάλθηκε από τη συσκευή SN-001
το εισερχόμενο topic του μηνύματος θα είναι /sensor/SN-001/data.
Με την παραπάνω σύνταξη η gateway θα θέσει ως τιμή του πεδίου **deviceNameTopicExpression** το τμήμα του topic που βρίσκεται ανάμεσα από από τα αλφαριθμητικά "sensor/" και "/data" και θα δώσει ως τιμή στο πεδίο **deviceNameTopicExpression** το αλφαριθμητικό SN=001.

Τα υπόλοιπα πεδία διαμορφώνονται όπως και στο example 1 και όπως βλέπουμε στην παρακάτω εικόνα η τιμή του deviceTypeJsonExpression καθορίζεται σε μια συγκεκριμένη τιμή από εμάς κατά τη διαμόρφωση του mapping:

```
{
  "topicFilter": "/sensor/+/data",
  "converter": {
    "type": "json",
    "deviceNameTopicExpression": "(?<=sensor\/)(.*?)(?=\/data)",
    "deviceTypeTopicExpression": "Thermometer",
    "timeout": 60000,
    "attributes": [
      {
        "type": "string",
        "key": "model",
        "value": "${sensorModel}"
      }
    ],
    "timeseries": [
      {
        "type": "double",
        "key": "temperature",
        "value": "${temp}"
      },
      {
        "type": "double",
        "key": "humidity",
        "value": "${hum}"
      }
    ]
  }
}
```

Εικόνα 4.19 Η διαμόρφωση του mapping

Πεδίο “connectRequests”:

Το ThingsBoard κάθε φορά που ενημερώνονται κάποια χαρακτηριστικά “attributes” της συσκευής επιτρέπει την αποστολή εντολών RPC προς τη συσκευή. Αλλά για να σταλούν, η πλατφόρμα πρέπει να γνωρίζει εάν η συσκευή προορισμού είναι συνδεδεμένη και ποια πύλη ή περίοδος σύνδεσης χρησιμοποιείται για τη σύνδεση της συσκευής εκείνη τη στιγμή.

Εάν η συσκευή στέλνει συνεχώς δεδομένα τηλεμετρίας, το ThingsBoard γνωρίζει ήδη πώς να προωθεί ειδοποιήσεις. Εάν η συσκευή συνδέεται μόνο με τον broker MQTT και περιμένει εντολές / ενημερώσεις, η πλατφόρμα δεν γνωρίζει αν η συσκευή είναι συνδεδεμένη στο broker. Σε αυτή την περίπτωση πρέπει η συσκευή να στείλει ένα μήνυμα στην gateway για να ενημερώσει ότι είναι συνδεδεμένη στο broker.

Και εδώ υπάρχουν δύο περιπτώσεις. Η συσκευή να στείλει το αναγνωριστικό συσκευής ως μέρος του topic ή ως πεδίο του Json payload.

Παρακάτω βλέπουμε πως μπορεί να διαμορφωθεί το πεδίο connectRequests και στις δύο περιπτώσεις.

```
"connectRequests": [
  {
    "topicFilter": "sensors/connect",
    "deviceNameJsonExpression": "${$.SerialNumber}"
  },
  {
    "topicFilter": "sensor/+/connect",
    "deviceNameTopicExpression": "(?<=sensor\/)(.*?)(?=\/connect)"
  }
]
```

Εικόνα 4.20 Η διαμόρφωση του τμήματος connectRequests

Πεδίο "disconnectRequest":

Στο πεδίο αυτό διαμορφώνουμε ένα μήνυμα που στέλνεται από τη συσκευή στη gateway για να ενημερωθεί η πλατφόρμα Thingsboad για αποσύνδεση της συσκευής. Αυτό το κάνουμε εάν η συσκευή περιμένει ενημερώσεις τιμών και προκειται να αποσυνδεθεί.

Η λειτουργία αυτή είναι προεραιτική.

Το πεδίο διαμορφώνεται παρόμοια με το πεδίο connectRequests

```
"disconnectRequests": [
  {
    "topicFilter": "sensors/disconnect",
    "deviceNameJsonExpression": "${SerialNumber}"
  },
  {
    "topicFilter": "sensor/+/disconnect",
    "deviceNameTopicExpression": "(?<=sensor\/)(.*?)(?=\/disconnect)"
  }
]
```

Εικόνα 4.21Η διαμόρφωση του τμήματος disconnectRequests

Πεδίο "attributesUpdates":

Και αυτό το πεδίο είναι προαιρετικό.

Στο πεδίο αυτό διαμορφώνουμε τα μηνύματα που θα στέλνονται από την πλατφόρμα προς τη συσκευή κάθε φορά που ένα "κοινόχρηστο χαρακτηριστικό συσκευής" shared attribute ενημερώνεται.

Παρακάτω βλέπουμε τον πίνακα παραμέτρων που πρέπει να διαμορφώσουμε:

Parameter	Default value	Description
deviceNameFilter	**SmartMeter.***	Regular expression device name filter, uses to determine, which function to execute.
attributeFilter	**uploadFrequency**	Regular expression attribute name filter, uses to determine, which function to execute.
topicExpression	**sensor/${deviceName}/${attributeKey}**	JSON-path expression uses for creating topic address to send a message.
valueExpression	**{\"${attributeKey}\":\"${attributeValue}\"}**	JSON-path expression uses for creating the message data that will send to topic.

Ακολουθεί ένα παράδειγμα της διαμόρφωσης του πεδίου:

```
"attributeUpdates": [
  {
    "deviceNameFilter": "SmartMeter.*",
    "attributeFilter": "uploadFrequency",
    "topicExpression": "sensor/${deviceName}/${attributeKey}",
    "valueExpression": "{\"${attributeKey}\":\"${attributeValue}\"}"
  }
]
```

Εικόνα 4.22 Η διαμόρφωση του τμήματος attributeUpdates

Το πεδίο topicExpression καθορίζει τη μορφή που θα έχει το topic του μηνύματος που θα αποσταλεί από τη gateway προς τον broker όταν θα λάβει μήνυμα από την πλατφόρμα Thingsboard ότι μια attribute έχει ενημερωθεί.

Οι μεταβλητές ${deviceName} και ${attributeKey} θα αντικατασταθούν με τις αντίστοιχες τιμές που έχουν στην πλατφόρμα.

Το πεδίο valueExpression καθορίζει τη μορφή που θα έχει το payload του μηνύματος που θα αποσταλεί.

Η μεταβλητή ${attributeValue} περιέχει την ενημερωμένη τιμή που πρέπει να αποσταλεί.

Ισχύουν και εδώ οι επιλογές να εισαχθεί το αναγνωριστικό συσκευής στο topic ή στο payload όπως και πριν.

Πεδίο "serverSideRpc":

Η πλατφόρμα Thingsboard μπορεί να στέλνει RPC εντολές είτε απευθείας στην συνδεδεμένη σε αυτή συσκευή είτε σε συσκευή συνδεδεμένη στην gateway.

Στο πεδίο αυτό διαμορφώνουμε τα μηνύματα που θα στέλνονται από την gateway στον broker κάθε φορά που η πλατφόρμα Thingsboard θα στέλνει μια RPC εντολή στην gateway.

Εδώ η πλατφόρμα υποστηρίζει αποστολή RPC:

a. Με απάντηση από τη συσκευή, όπου θα περιμένει την απάντηση.
b. Χωρίς απάντηση από τη συσκευή.

Στην παρακάτω εικόνα βλέπουμε πως μπορούμε να διαμορφώσουμε το πεδίο είτε για την περίπτωση που πρέπει η συσκευή να απαντήσει είτε στην περίπτωση που δεν χρειάζεται απάντηση.

Αυτό καθορίζεται από την τιμή του πεδίου **methodFilter** σε "echo" ή "no-reply"

```
"serverSideRpc": [
  {
    "deviceNameFilter": ".*",
    "methodFilter": "echo",
    "requestTopicExpression": "sensor/${deviceName}/request/${methodName}/${requestId}",
    "responseTopicExpression": "sensor/${deviceName}/response/${methodName}/${requestId}",
    "responseTimeout": 10000,
    "valueExpression": "${params}"
  },
  {
    "deviceNameFilter": ".*",
    "methodFilter": "no-reply",
    "requestTopicExpression": "sensor/${deviceName}/request/${methodName}/${requestId}",
    "valueExpression": "${params}"
  }
]
```

Εικόνα 4.23 Η διαμόρφωση του τμήματος serverSideRpc

Το πεδίο deviceNameFilter είναι ένα φίλτρο ώστε να δεχόμαστε RPC request για ορισμένες συσκευές.

Μόλις η gateway λάβει ένα RPC request θα κάνει publish στο topic requestTopicExpression τα περιεχόμενα του πεδίου "valueExpession". Στην πρώτη περίπτωση που η πλατφόρμα θα περιμένει απάντηση θα πρέπει να καθοριστεί και το πεδίο responseTopicExpression το οποίο θα γίνει από την gateway subscribe στον broker ώστε να μπορεί να λάβει την απάντηση από τη συσκευή.

Το πεδίο ResponseTimeout καθορίζει το χρόνο για τον οποίο η πλατφόρμα θα περιμένει τη συσκευή να απαντήσει σε χιλιοστά του δευτερολέπτου.

Συνολικά το αρχείο mqtt.json θα έχει την παρακάτω μορφή:

```
{
 "broker": {
  "name":"Default Local Broker",
  "host":"peatrib322.cloud.shiftr.io",
  "port":1883,
  "clientId": "ThingsBoard_gateway",
  "security": {
   "type": "basic",
   "username": "peatrib322",
   "password": "6973403792"
  }
```

```
},
"mapping": [
  {
    "topicFilter": "topic_1",
    "converter": {
      "type": "json",
      "deviceNameJsonExpression": "${NodeID}",
      "deviceTypeJsonExpression": "${Type}",
      "timeout": 60000,
      "attributes": [
        {
          "type": "string",
          "key": "model",
          "value": "Wemos ESP32"
        }
      ],
      "timeseries": [
        {
          "type": "double",
          "key": "temperature",
          "value": "${Temperature}"
        },
        {
          "type": "int",
          "key": "humidity",
          "value": "${Humidity}"
        },
            {
          "type": "double",
          "key": "coolantTemp",
          "value": "${CoolantTemp}"
        },
            {
          "type": "boolean",
          "key": "OnOff",
          "value": "${OnOff}"
        },
            {
          "type": "boolean",
          "key": "HiLow",
          "value": "${HiLow}"
        },
            {
          "type": "int",
```

```
        "key": "msgCount",
        "value": "${msgCount}"
      }
    ]
  }
},
{
  "topicFilter": "/sensor/+/data",
  "converter": {
    "type": "json",
    "deviceNameTopicExpression": "(?<=sensor\/)(.*?)(?=\/data)",
    "deviceTypeTopicExpression": "Thermometer",
    "timeout": 60000,
    "attributes": [
      {
        "type": "string",
        "key": "model",
        "value": "${sensorModel}"
      }
    ],
    "timeseries": [
      {
        "type": "double",
        "key": "temperature",
        "value": "${temp}"
      },
      {
        "type": "double",
        "key": "humidity",
        "value": "${hum}"
      }
    ]
  }
},
{
  "topicFilter": "/custom/sensors/+",
  "converter": {
    "type": "custom",
    "extension": "CustomMqttUplinkConverter",
    "extension-config": {
        "temperatureBytes" : 2,
        "humidityBytes" :  2,
        "batteryLevelBytes" : 1
    }
```

```
        }
      }
    ],
    "connectRequests": [
      {
        "topicFilter": "sensor/connect",
        "deviceNameJsonExpression": "${SerialNumber}"
      },
      {
        "topicFilter": "sensor/+/connect",
        "deviceNameTopicExpression": "(?<=sensor\/)(.*?)(?=\/connect)"
      }
    ],
    "disconnectRequests": [
      {
        "topicFilter": "sensor/disconnect",
        "deviceNameJsonExpression": "${SerialNumber}"
      },
      {
        "topicFilter": "sensor/+/disconnect",
        "deviceNameTopicExpression": "(?<=sensor\/)(.*?)(?=\/disconnect)"
      }
    ],
    "attributeUpdates": [
      {
        "deviceNameFilter": "SmartMeter.*",
        "attributeFilter": "uploadFrequency",
        "topicExpression": "sensor/${deviceName}/${attributeKey}",
        "valueExpression": "{\"${attributeKey}\":\"${attributeValue}\"}"
      }
    ],
    "serverSideRpc": [
      {
        "deviceNameFilter": ".*",
        "methodFilter": "echo",
        "requestTopicExpression":
"sensor/${deviceName}/request/${methodName}/${requestId}",
        "responseTopicExpression":
"sensor/${deviceName}/response/${methodName}/${requestId}",
        "responseTimeout": 10000,
        "valueExpression": "${params}"
      },
      {
        "deviceNameFilter": ".*",
```

```
            "methodFilter": "no-reply",
            "requestTopicExpression":
        "sensor/${deviceName}/request/${methodName}/${requestId}",
            "valueExpression": "${params}"
          }
        ]
      }
```

b. **Custom**

Η gateway έχει τη δυνατότητα να μετατρέψει και εισερχόμενα από τον broker μηνύματα τα οποία δεν είναι σε Json format αλλά είναι μια ακολουθία από bytes.

Σε αυτή την περίπτωση το πεδίο converter διαμορφώνεται όπως παρακάτω:

```
"converter": {
    "type": "custom",
    "extension": "CustomMqttUplinkConverter",
    "extension-config": {
      "temperatureBytes" : 2,
      "humidityBytes" :  2,
      "batteryLevelBytes" : 1
    }
  }
```

Το πεδίο type ορίζεται σε “custom”

Το πεδίο CustomMqttUplinkConverter περιέχει το όνομα από την custom converter class και χρησιμοποιείται από την πλατφόρμα.

Το πεδίο extension-config περιέχει τη διαμόρφωση του Custom Converter. Στο παράδειγμα η διαμόρφωση καθορίζεται σε πέντε bytes από τα οποία τα δύο πρώτα είναι η θερμοκρασία, τα επόμενα δύο η υγρασία και το τελευταίο το επίπεδο ενέργειας της μπαταρίας.

4.2 ΜΕΤΑΦΟΡΑ ΕΝΟΣ ΥΠΑΡΧΟΝΤΟΣ IoT ΕΡΓΟΥ ΣΤΗΝ ΠΛΑΤΦΟΡΜΑ THINGSBOARD

Ένας από τους στοχους της παρούσας διπλωματικής εργασίας είναι η μεταφορά ενός υπάρχοντος έργου στην πλατφόρμα Thingsboard με χρήση της gateway που αναλύσαμε παραπάνω.

Το έργο που θα μεταφερθεί αφορά τον περιβαντολογικό έλεγχο κτιρίων και αποτελείται από τρία μέρη τα οποία αποτέλεσαν αντικείμενα πτυχιακών εργασιών του τμήματος ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗ Τ.Ε του ΤΕΙ Ηπείρου.

Θα μπορούσαμε να χωρίσουμε το έργο σε τέσσερα μέρη:

a. Η συσκευή με τους αισθητήρες και τους ενεργοποιητές
b. Ο Broker
c. Η βάση δεδομένων
d. Η ιστοσελίδα διαχείρισης του έργου

Από τα παραπάνω μέρη του έργου αυτά που εμπλέκονται στην παρούσα εργασία είναι η συσκευή με τους αισθητήρες και τους ενεργοποιητές και ο MQTT broker

Η συσκευή με τους αισθητήρες κατασκευάστηκε το 2016 από το Λισγάρα Βασίλειο ως μέρος της πτυχιακής του εργασίας στο ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗ Τ.Ε της σχολής ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ του ΤΕΙ Ηπείρου.[11

ΑΣΥΡΜΑΤΟ ΣΥΣΤΗΜΑ ΠΕΡΙΒΑΝΤΟΛΟΓΙΚΟΥ ΕΛΕΓΧΟΥ ΚΤΙΡΙΩΝ

Το εξυπνο σύστημα διαχείρισης αποτελείται από ένα μικροελεγκτή Texas Instruments MSP-EXP430F5529LP , που διαχειρίζεται την επικοινωνία με ένα MQTT broker, καθώς και τη λήψη και αποστολή των τιμών θερμοκρασίας και υγρασίας από τους αισθητήρες. Επισης, λαμβάνει εντολές για τη διαχείριση της κλιματιστικής μονάδας, που πραγματοποιείται μέσω ενός συνδεδεμένου ηλεκτρομαγνητικού διακόπτη.

Αναλυτικά περιλαμβάνει:

Ένα αισθητήρα θερμοκρασίας και υγρασίας χώρου (DHT11)

Ένα αισθητήρα θερμοκρασία ψυκτικού υγρού (DS18B20)

Δύο Relay 5V για τη διαχείριση των καταστάσεων (On/Off και Hi/Low) της κλιματιστικής μονάδας.

Εικόνα 4.24 Τα βασικά μέρη του έξυπνου συστήματος περιβαντολλογικού ελέγχου

Οι τιμές των αισθητήρων αυτών στέλνονται σε ένα broker που είναι εγκατεστημένος στο τμήμα Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Ιωαννίνων.

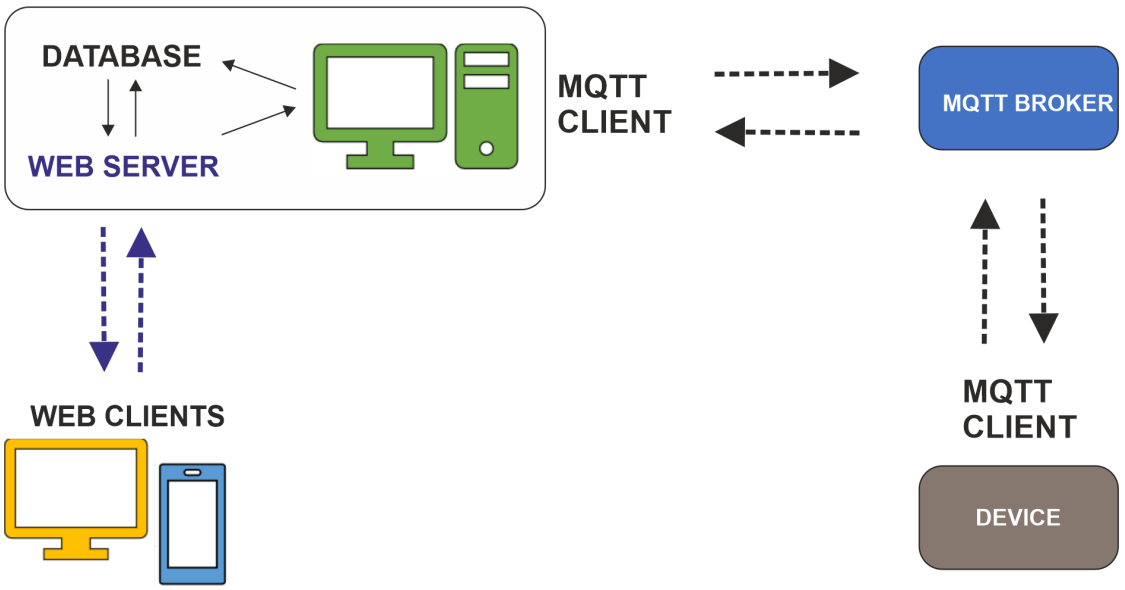
Ταυτόχρονα το σύστημα μπορεί να λαμβάνει εντολές από την ιστοσελίδα διαχείρισης που αφορούν την ON-OFF λειτουργία και τη μετάβαση από την ψύξη στη θέρμανση.

Ο MQTT Broker αποτελεί μέρος της εργασίας του Ιωάννη Μασκλαβάνου του ΤΜΗΜΑΤΟΣ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ Τ.Ε της σχολής ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ του ΤΕΙ Ηπείρου με τίτλο ΣΥΣΤΗΜΑ ΔΙΑΧΕΙΡΙΣΗΣ ΚΑΙ ΑΠΟΜΑΚΡΥΣΜΕΝΟΥ ΕΛΕΓΧΟΥ ΑΙΣΘΗΤΗΡΩΝ. [12]

Σκοπός της παρούσας εργασίας είναι τα μηνύματα τα οποία αποστέλλονται από τη συσκευή στον MQTT Broker να αποστέλλονται στη συνέχεια από τον MQTT Broker στην Gateway και από αυτή στην πλατφόρμα Thingsboard.

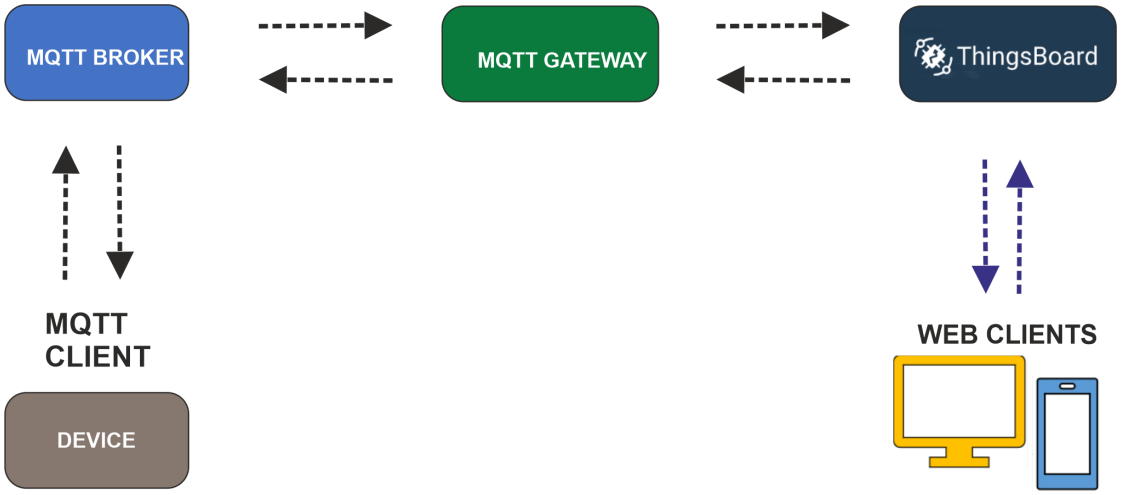
Αλλά και η εντολές “ON-OFF” και “High-Low” που αποστέλλονται από την ιστοσελίδα διαχείρισης να αποστέλλονται πλέον από το Dashboard της πλατφόρμας Thingsboard και να φτάνουν στη συσκευή αφού πρώτα περάσουν από τη gateway και μετά από τον MQTT broker

Τα παρακάτω διαγράμματα δείχνουν τον τρόπο σύνδεσης των μερών του έργου πριν και μετά την προσθήκη της gateway και της νέας πλατφόρμας Thingsboard.



Εικόνα 4.24: Στο διάγραμμα βλέπουμε τον τρόπο συνδέσεων των διαφόρων μερών του έργου πριν την προσθήκη της gateway

Το πρώτο βήμα για τη μεταφορά του έργου στη νέα πλατφόρμα με χρήση της Gateway είναι η διαμόρφωση του αρχείο



Εικόνα 4.25: Στο διάγραμμα βλέπουμε τον τρόπο σύνδεσης της Gateway και της πλατφόρμας Thingsboard στο έργο.

Δεν έγινε καμία αλλαγή στο device καθώς και στον MQTT Broker, ενώ δεν θα χρησιμοποιηθεί πλέον η database και ο web server αφού τόσο η αποθήκευση όσο και ο έλεγχος θα γίνονται μέσα από την πλατφόρμα Thingsboard.

Το πρώτο πράγμα λοιπόν που πρέπει να πετύχουμε είναι την ταυτόχρονη σύνδεση της Gateway τόσο με τον υπάρχων MQTT Broker όσο και με την πλατφόρμα Thingsboard.

Αρχικά διαμορφώνουμε το αρχείο **tb_gateway.yaml**

Διαμόρφωση αρχείου tb_gateway.yaml

Στο πεδίο host εισάγουμε το url του υπολογιστή που έχει γίνει deploy η πλατφόρμα και στο πεδίο accessToken: εισάγουμε το access token που μας δίνει η πλατφόρμα για τη Gateway όπως αναφέρθηκε παραπάνω.

```
thingsboard:
  host: demo.thingsboard.io
  port: 1883
#  remoteShell: false
#  remoteConfiguration: false
  security:
    accessToken: 5MK2sOGmCQlHmZMBu1c9
  qos: 1
storage:
  type: memory
  read_records_count: 10
  max_records_count: 1000
#  type: file
#  data_folder_path: ./data/
#  max_file_count: 10
#  max_read_records_count: 10
#  max_records_per_file: 10000
connectors:
  -
    name: MQTT Broker Connector
    type: mqtt
    configuration: mqtt.json
```

Διαμόρφωση αρχείου mqtt.json

Στη συνέχεια θα διαμορφώσουμε το αρχείο mqtt.json ώστε να πετύχουμε σύνδεση με τον υπάρχων MQTT Broker.

Το πεδίο broker διαμορφώνεται ως παρακάτω:

```
  "broker": {
    "name":"Default Local Broker",
```

```
  "host":"peatrib322.cloud.shiftr.io",
  "port":1883,
  "clientId": "ThingsBoard_gateway",
  "security": {
   "type": "basic",
   "username": "peatrib322",
   "password": "6973403792"
  }
 },
```

Στο πεδίο host έχουμε εισάγει την ip του MQTT Broker .
Έχουμε επιλέξει σύνδεση χωρίς ασφάλεια TLS οπότε χρησιμοποιούμε την πόρτα 1883.

Στη συνέχεια θα διαμορφώσουμε το πεδίο mapping για να καθορίσουμε τα topics που θα γίνουν subscribe από την gateway προς τον MQTT Broker, καθώς και τη διαμόρφωση των εισερχομένων στην Gateway μηνυμάτων από αυτόν.

Η συσκευή στέλνει συνολικά όλες τις πληροφορίες στο topic m2mce/hvac/ce/meas του broker. Οι πληροφορίες αυτές περιέχονται σε ένα Json object της παρακάτω μορφής

```
{
        "NodeID": "00:11:22:33:44:55",
        "Type": "ControlSensor/Sensor",
        "Humidity":<int> ,
        "Temperature": <int>,
        "CoolantTemp": <int>,
        "OnOff": <Boolean>,
        "HiLow": <Boolean>,
        “msgCount”:<int>
 }
```

Από ότι βλέπουμε στο παραπάνω Json το αναγνωριστικό της συσκευής υπάρχει μέσα στο payload και όχι στο topic. Αυτό σημαίνει ότι εδώ έχουμε να κάνουμε με την πρώτη περίπτωση διαμόρφωσης που αναφέραμε στη σελίδα 46, δηλαδή θα εισάγουμε το topic στο mapping χωρίς να χρησιμοποιήσουμε τους χαρακτήρες ‘+’ και ‘#’.

Τα δεδομένα θα ανέβουν στην πλατφόρμα σαν telemetry data οπότε το πρώτο στοιχείο του Json Array mapping θα έχει ως παρακάτω:

```
{
   "topicFilter": "m2mce/hvac/ce/meas",
   "converter": {
    "type": "json",
```

```
  "deviceNameJsonExpression": "${NodeID}",
  "deviceTypeJsonExpression": "${Type}",
  "timeout": 60000,
  "attributes": [
    {
      "type": "string",
      "key": "model",
      "value": "Wemos ESP32"
    }
  ],
  "timeseries": [
    {
      "type": "double",
      "key": "temperature",
      "value": "${Temperature}"
    },
    {
      "type": "int",
      "key": "humidity",
      "value": "${Humidity}"
    },
        {
      "type": "double",
      "key": "coolantTemp",
      "value": "${CoolantTemp}"
    },
        {
      "type": "boolean",
      "key": "OnOff",
      "value": "${OnOff}"
    },
        {
      "type": "boolean",
      "key": "HiLow",
      "value": "${HiLow}"
    },
        {
      "type": "int",
      "key": "msgCount",
      "value": "${msgCount}"
    }
  ]
 }
}
```

Παρατηρούμε όχι τα πεδία
NodeID
Type
Humidity
Temperature
CoolantTemp
OnOff
HiLow
msgCount

έχουν εισαχθεί ώς μεταβλητές στο αρχείο διαμόρφωσεις με τη μορφή
${JsonLabel }

Αφού όλα τα δεδομένα των αισθητήρων θα αποστέλλονται στον broker με ένα μόνο Json δεν χρειάζεται να προσθέσουμε άλλο στοιχείο στον πίνακα mapping του αρχείου διαμόρφωσης.

Η τελική μορφή του αρχείου mqtt.json μετά τη διαμόρφωση έχει ως παρακάτω:

```
{
 "broker": {
  "name":"Default Local Broker",
  "host":"peatrib322.cloud.shiftr.io",
  "port":1883,
  "clientId": "ThingsBoard_gateway",
  "security": {
   "type": "basic",
   "username": "peatrib322",
   "password": "6973403792"
  }
 },
 "mapping": [
  {
   "topicFilter": "m2mce/hvac/ce/meas",
   "converter": {
    "type": "json",
    "deviceNameJsonExpression": "${NodeID}",
    "deviceTypeJsonExpression": "${Type}",
    "timeout": 60000,
    "attributes": [
     {
      "type": "string",
```

```
          "key": "model",
          "value": "Wemos ESP32"
        }
      ],
      "timeseries": [
        {
          "type": "double",
          "key": "temperature",
          "value": "${Temperature}"
        },
        {
          "type": "int",
          "key": "humidity",
          "value": "${Humidity}"
        },
            {
          "type": "double",
          "key": "coolantTemp",
          "value": "${CoolantTemp}"
        },
            {
          "type": "boolean",
          "key": "OnOff",
          "value": "${OnOff}"
        },
            {
          "type": "boolean",
          "key": "HiLow",
          "value": "${HiLow}"
        },
            {
          "type": "int",
          "key": "msgCount",
          "value": "${msgCount}"
        }
      ]
    }
  }
],
"connectRequests": [
],
"disconnectRequests": [
],
"attributeUpdates": [
```

```
    ],
    "serverSideRpc": [
    ]
}
```

Στη συνέχεια είμαστε έτοιμοι να κάνουμε τη δοκιμή της gateway.

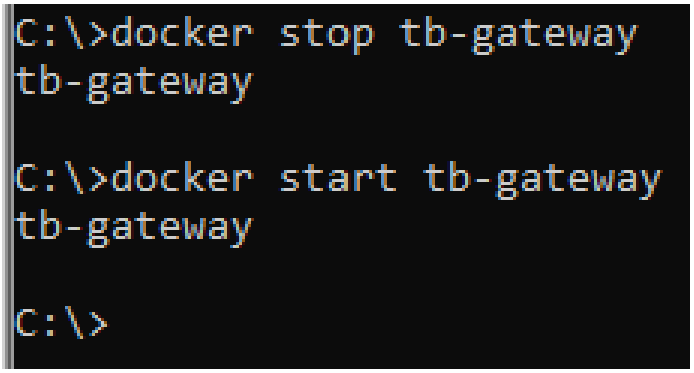
από το command line δίνουμε την εντολή:

docker stop tb-gateway

και αμέσως μετά την εντολή

docker start tb-gateway

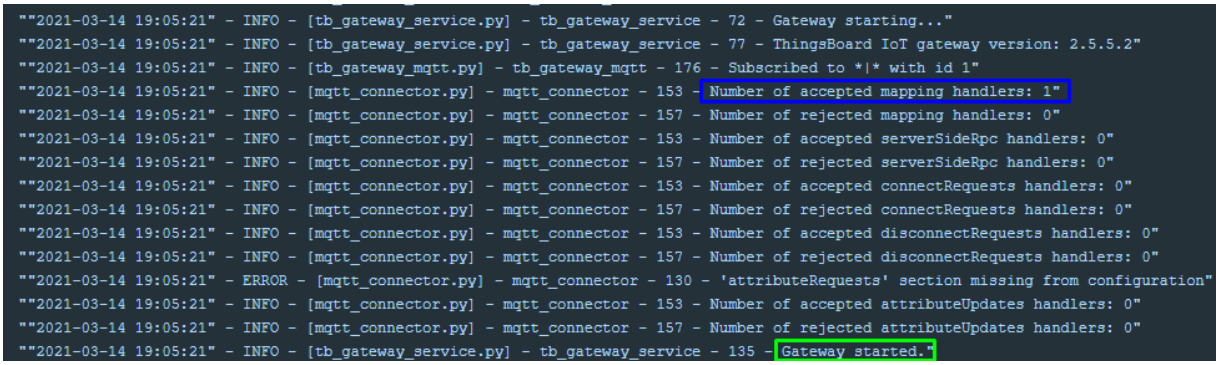
ώστε η Gateway να κάνει επανεκκίνηση και να πάρει τις νέες τιμές των δύο αρχείων που διαμορφώσαμε.



Εικόνα 4.46 Οι εντολές μέσα από το command line για το ξεκίνημα.σταμάτημα του Docker

Ασφαλώς το ίδιο θα μπορούσαμε να κάνουμε και μέσα από την εφαρμογή Docker Desktop

Μετά την εκκίνηση της gateway μπορούμε να δούμε το αρχείο Log μέσα από το Docker Desktop



Εικόνα 4.27: Το αρχείο Log της gateway για τη σύνδεσή της με την πλατφόρμα

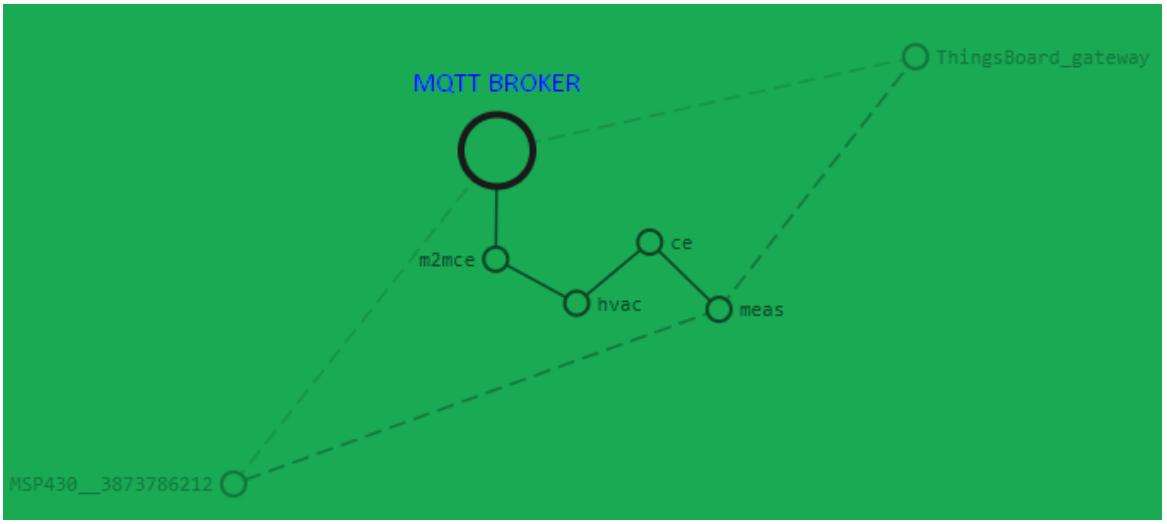
Στην εικόνα 4.28 βλέπουμε πληροφορίες για τη σύνδεση με την πλατφόρμα. Μπορούμε να δούμε πόσα από τα στοιχεία των Json Arrays : mapping, serverSideEpc, connectRequests, disconectRequests, attributesUpdates έχουν αναγνωριστεί

```
""2021-03-14 19:05:22" - INFO - [mqtt_connector.py] - mqtt_connector - 227 - MQTT Broker Connector connected to
peatrib322.cloud.shiftr.io:1883 - successfully."
""2021-03-14 19:05:22" - INFO - [mqtt_connector.py] - mqtt_connector - 269 - Connector "MQTT Broker Connector" subscribe to
m2mce/hvac/ce/meas"
""2021-03-14 19:05:22" - INFO - [tb_device_mqtt.py] - tb_device_mqtt - 141 - connection SUCCESS"
""2021-03-14 19:05:22" - INFO - [mqtt_connector.py] - mqtt_connector - 317 - "MQTT Broker Connector" subscription success to topic
m2mce/hvac/ce/meas, subscription message id = 1"
""2021-03-14 19:05:22" - INFO - [tb_gateway_mqtt.py] - tb_gateway_mqtt - 176 - Subscribed to *|* with id 2"
```

Εικόνα 4.28: Το αρχείο Log της gateway για τη σύνδεσή της με τον MQTT Broker

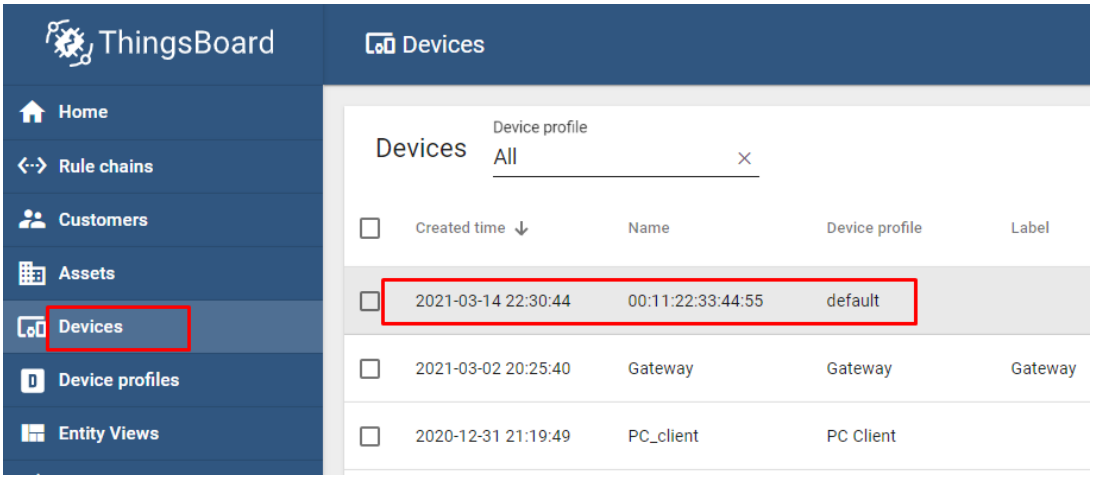
Στην εικόνα 4.29 του αρχείου Log βλέπουμε πληροφορίες για τη σύνδεση με τον MQTT Broker.

Βλέπουμε στην πρώτη γραμμή ότι η σύνδεση επετεύχθει και στη συνέχεια μέσα στο κόκκινο πλαίσιο πληροφορίες για τα topics που έγιναν subscribe.



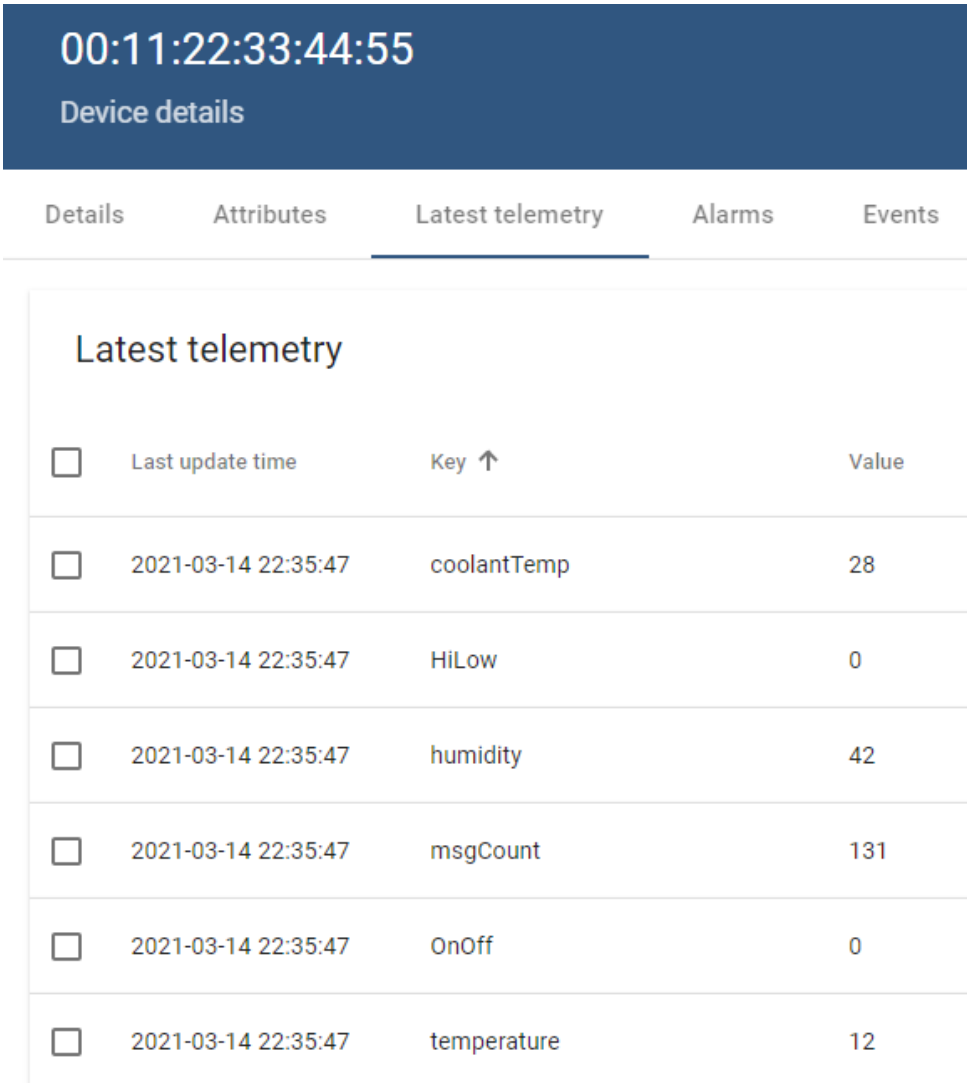
Εικόνα 4.29: Στην κονσόλα του MQTT Broker βλέπουμε ότι είναι συνδεδεμένος τόσο με τη συσκευή όσο και με την Gateway

Στη συνέχεια αφού κάνουμε login στην πλατφόρμα Thingsboard στα Devices παρατηρούμε ότι έχει προστεθεί αυτόματα το όνομα της συσκευής μας.



Εικόνα 4.30 Οι εγκατεστημένες συσκευές στην πλατφόρμα Thingsboard

Στη συνέχεια ανοίγουμε τα settings της συσκευής μας και βλέπουμε ότι στα "Latest telemetry" υπάρχουν οι τιμές που έχουν αποσταλεί από τη συσκευή, αλλά επίσης έχουν δημιουργηθεί και τα Keys που ορίσαμε για τα telemetry data στο αρχείο διαμόρφωσης mqtt.json της Gateway



Εικόνα 4.31 Οι τιμές που έχουν αποσταλεί από τη συσκευή εμφανίζονται στο παράθυρο “Latest telemetry”

Η συσκευή Απομακρυσμένου Ελέγχου Κλιματικών Εγκαταστάσεων στέλνει κάθε ένα λεπτό τις τιμές των αισθητήρων στον Broker.

Παρακάτω βλέπουμε τη μορφή ενός τέτοιου μηνύματος που στάλθηκε προς τον Broker από τη συσκευή:

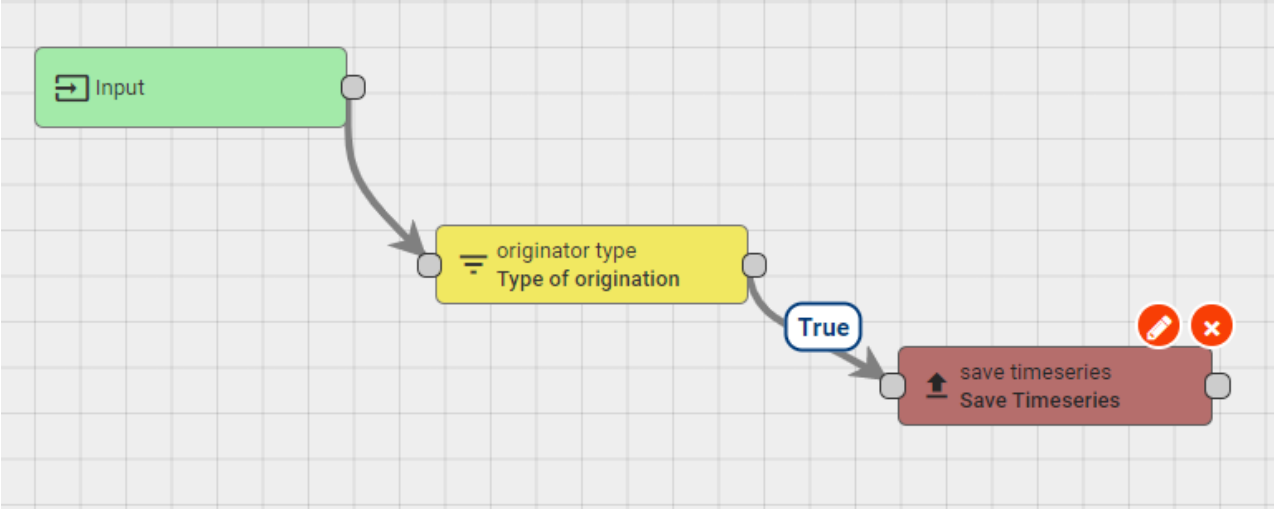
{
"NodeID":"00:11:22:33:44:55",
"Type":"ControlSensor/Sensor",
"Humidity":54,
"Temperature":18,
"CoolantTemp":25,
"OnOff":0,
"HiLow":1,
"msgCount":6

}

Για να δούμε αυτό το μήνυμα πως λαμβάνεται από την πλατφόρμα

Από το κεντρικό μενού επιλέγουμε Rule chains / Root Rule Chain / Open rule chain

Στον Node Red editor δημιουργούμε μια απλή σύνδεση όπου φιλτράροντας τα εισερχόμενα μηνύματα ώστε να λαμβάνουμε μόνο από devices τα οδηγούμε σε ένα Node το οποίο τα αποθηκεύει στην πλατφόρμα.

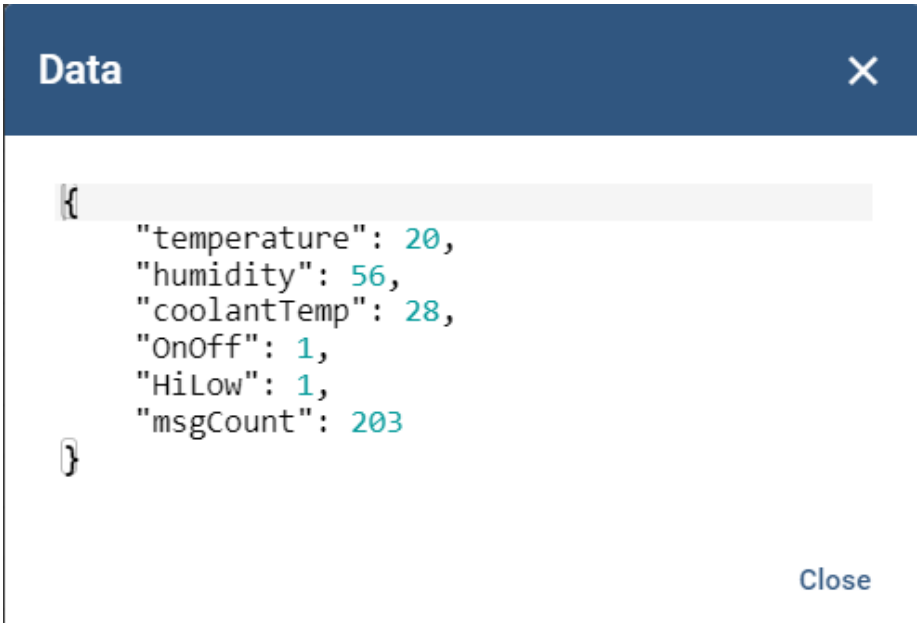


Εικόνα 4.32 Ο Node Red editor της πλατφόρμας Thingsboard

Κάνοντας “edit” το Node “Save Timeseries” μπορούμε να δούμε όλα τα εισερχόμενα στην πλατφόρμα μηνύματα από την Gateway.

Παρατηρούμε ότι τα μηνύματα POST_TELEMETRY είναι αποδεκτά από το Node.

Κάνοντας click σε κάποιο από αυτά μπορούμε να δούμε το εισερχόμενο στην πλατφορμα μήνυμα



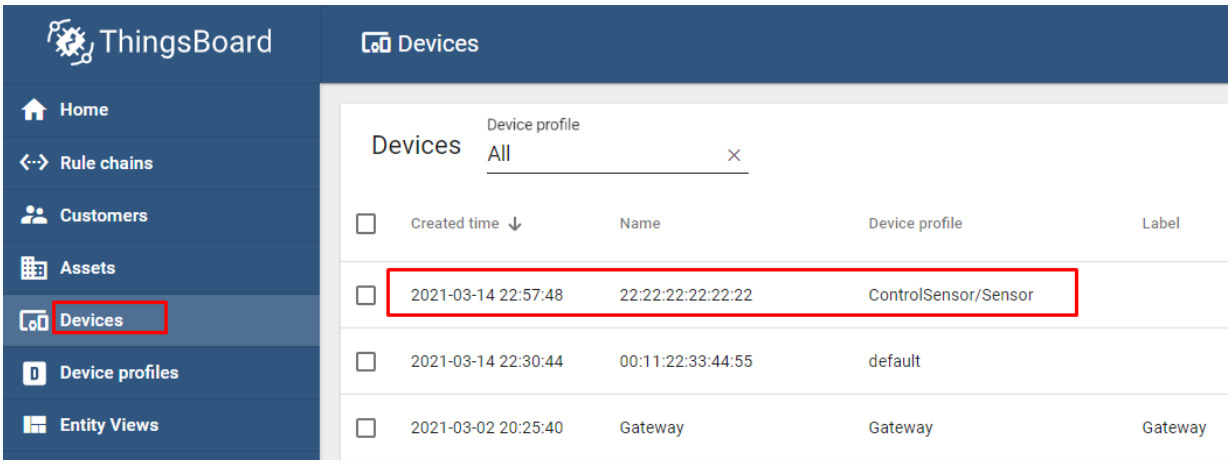
Εικόνα 4.33: Στην εικόνα βλέπουμε τη μορφή του εισερχόμενου στη πλατφόρμα μηνύματος. Παρατηρούμε ότι έχει τη διαμόρφωση που ορίσαμε στο αρχείο διαμόρφωσης mqtt.json

Κάνοντας click στα Metadata εμφανίζεται το παράθυρο Metadata

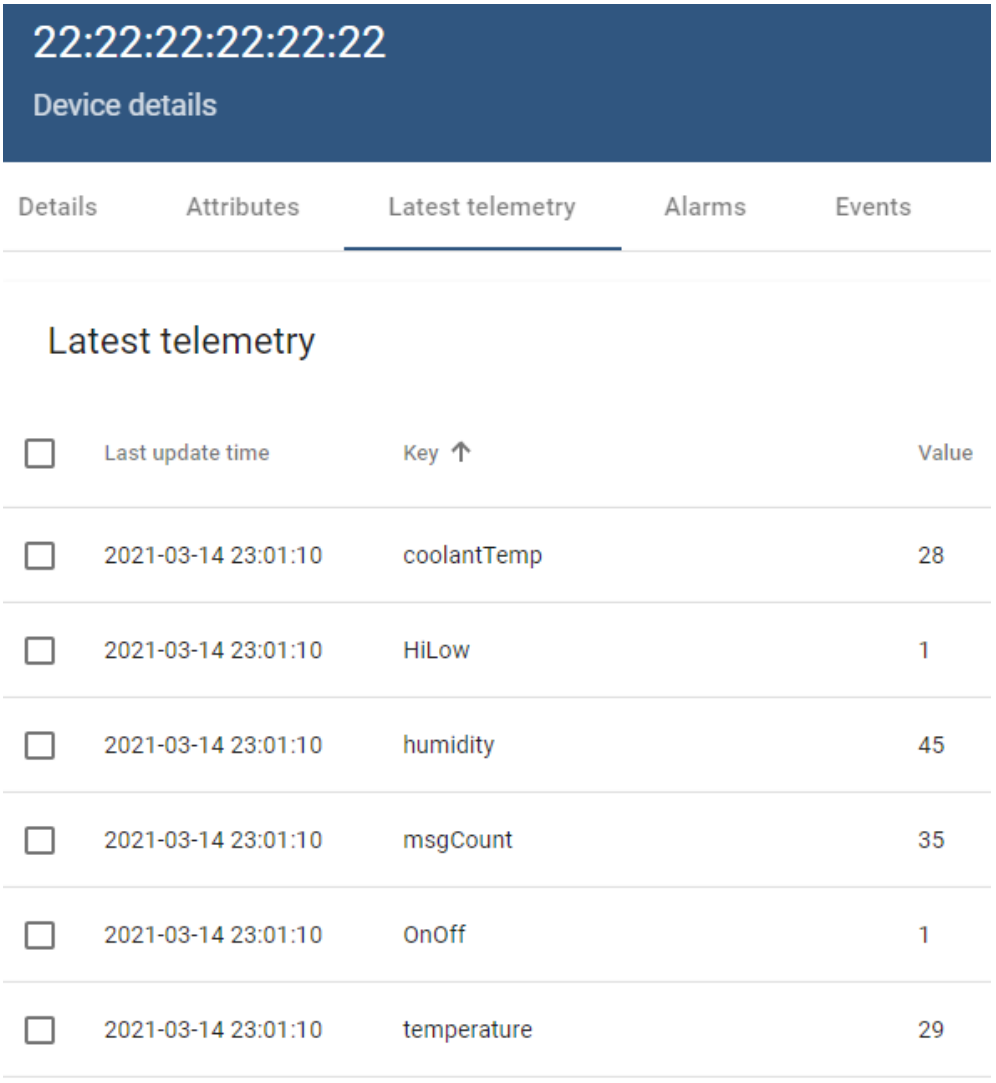
Εικόνα 4.34: Στην εικόνα βλέπουμε το όνομα της συσκευής του εισερχόμενου μηνύματος καθώς και τον τύπο. Όπως αναφέρθηκε παραπάνω το όνομα της συσκευής βρισκόταν στο payload του απεσταλμένου από τη συσκευή μηνύματος. Αυτό δείχνει ότι η Gateway λειτουργεί κανονικά και έχει την ικανότητα να ξεχωρίζει τις συσκευές.

Για να το δούμε καλύτερα αυτό βγάζουμε τη συσκευή εκτός λειτουργίας και αλλάζουμε στον κώδικα του απεσταλμένου Json το όνομα του πεδίου NodeID σε “22:22:22:22:22:22” ώστε να προσομοιώσουμε την κατάσταση όπου η πρώτη συσκευή αποσυνδέθηκε και συνδέθηκε μία άλλη με διαφορετικό NodeID.

Μετά τη σύνδεση της συσκευής στον MQTT Broker ελέγχουμε τα Devices στην πλατφόρμα Thingsboard και βλέπουμε τη νέα συσκευή.



Εικόνα 4.35: Με τη σύνδεση μιας νέα συσκευής στη gateway η πλατφόρμα τη δημιουργεί και την προσθέτει αυτόματα στις υπάρχουσες συσκευές.



Εικόνα 4.36: Οπως και πριν μπορούμε να δούμε στο tab “Latest Telemetry” τα πεδία που είχαμε ορίσει να αποθηκεύονται οι τιμές στο αρχείο mqtt.json

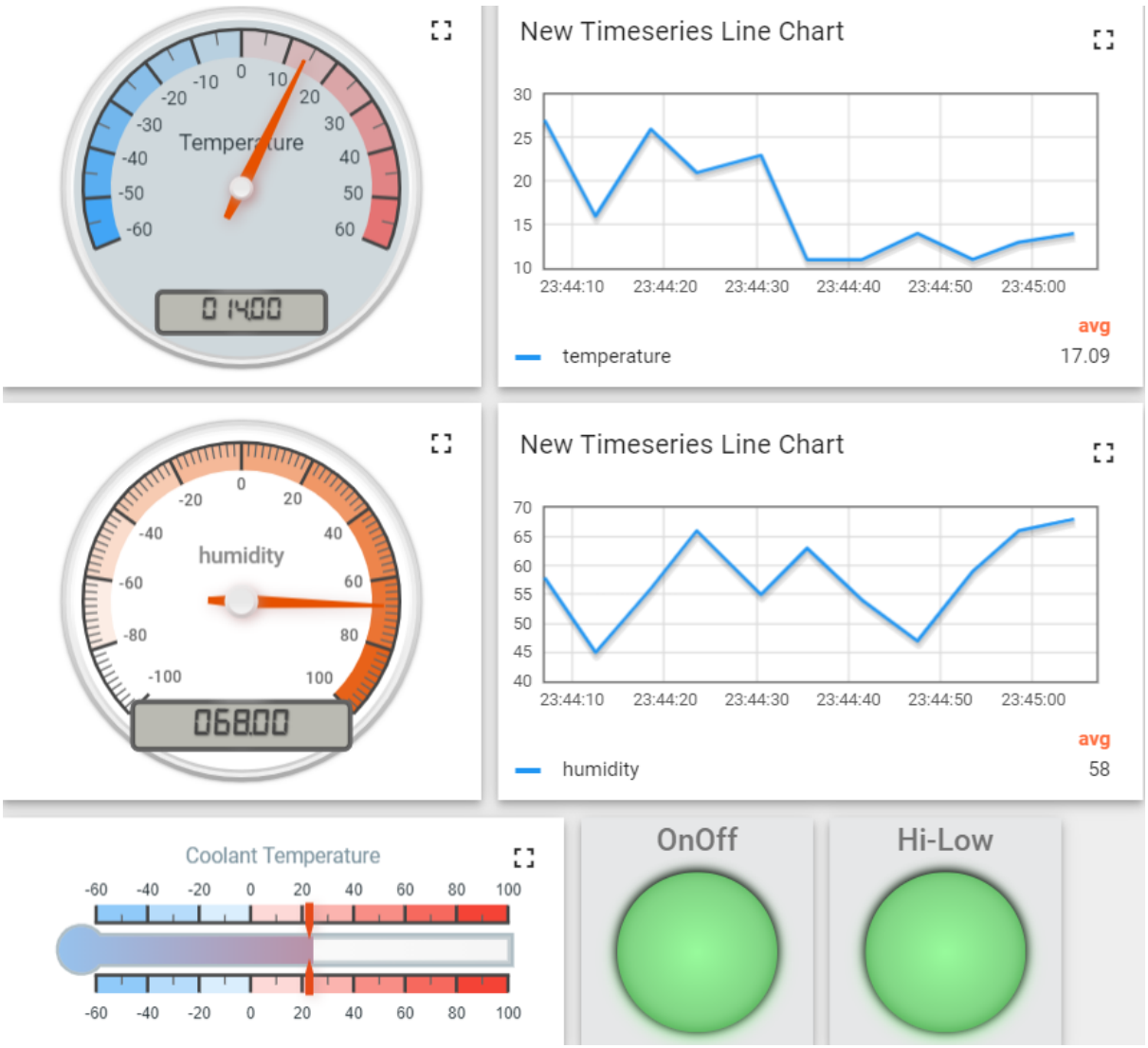
Εικόνα 4.37: Στα Metadata βλέπουμε το όνομα της νέας συσκευής να περιλαμβάνεται το εισερχόμενο μήνυμα

Δημιουργία ενός απλού Dashboard για την παρακολούθηση των τιμών.

Το επόμενο βήμα είναι η δημιουργία απλού dashboard για την παρακολούθηση των τιμών.

Από το κεντρικό μενού επιλέγουμε “Dashboards” και στη συνέχεια κάνοντας click στο εικονίδιο ‘+’ επιλέγουμε “Create new dashboard” και δημιουργούμε την Dashboard με το όνομα “Control”.

Ανοίγουμε για επεξεργασία την Dashboard και δημιουργούμε την παρακάτω Dashboard ακολουθώντας τις οδηγίες από το Documentation της πλατφόρμας.



Εικόνα 4.38: Το Dashboard που δημιουργήσαμε για το έργο μέσα από τον Dashboard editor

ΚΕΦΑΛΑΙΟ 5

5.1 ΔΗΜΙΟΥΡΓΙΑ ΜΕ ΚΩΔΙΚΑ JAVA ΜΙΑΣ IoT GATEWAY MQTT -MQTT

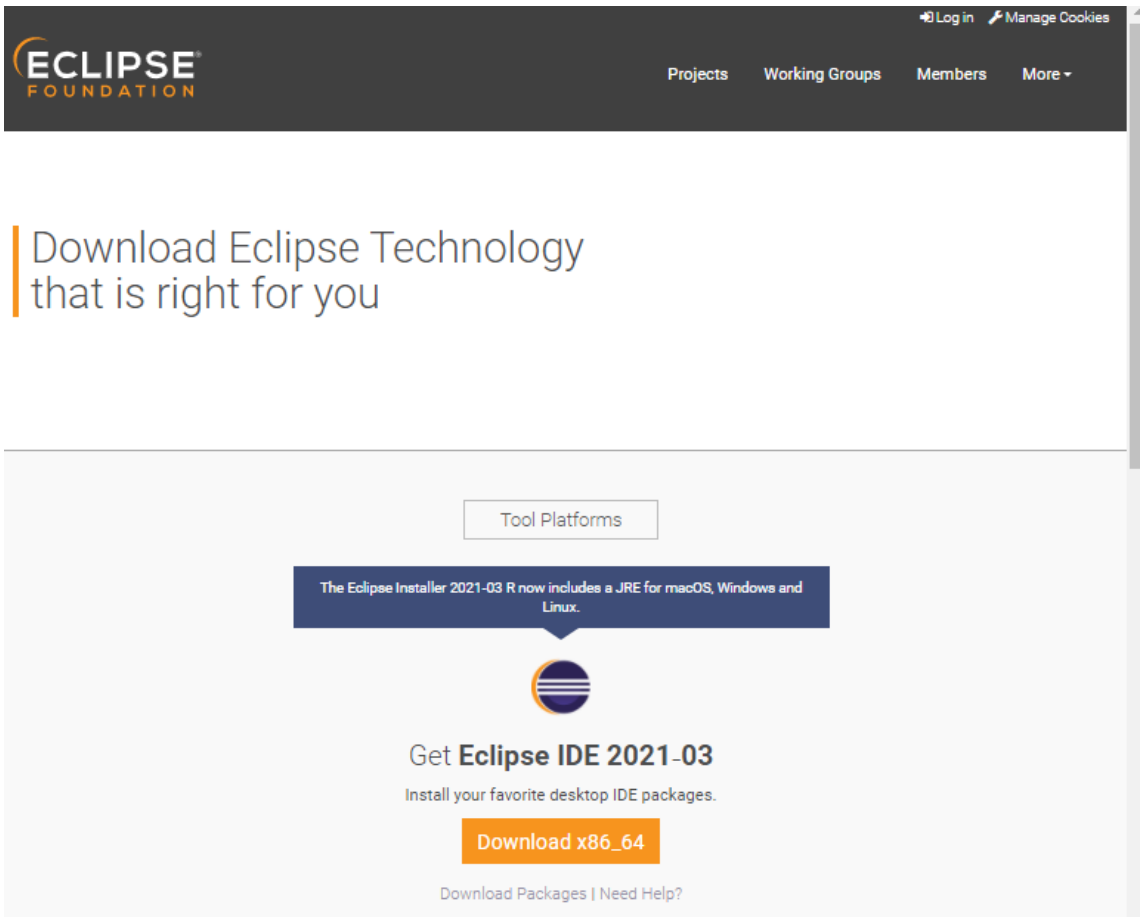
Η παραπάνω gateway που χρησιμοποιήσαμε αποσκοπεί αποκλειστικά στη σύνδεση της δικής της πλατφόρμας Thingsboard με άλλους brokers.

Σε αυτό το μέρος θα ασχοληθούμε με την κατασκευή μιας gateway η οποία θα μπορεί να παραμετροποιηθεί και να συνδέει οποιαδήποτε πλατφόρμα με άλλες που υποστηρίζουν το πρωτόκολλο MQTT.

Θα ακολουθήσουμε την ίδια λογική με την gateway της πλατφόρμας Thingsboard και θα δημιουργήσουμε ένα κώδικα ο οποίος θα μπορεί ταυτόχρονα να πραγματοποιεί δύο συνδέσεις με το πρωτόκολλο MQTT και να μορφοποιεί τα μηνύματα που στέλνονται εκατέρωθεν.

Η ανάπτυξη της gateway θα γίνει σε περιβάλλον windows σε γλώσσα Java και θα χρησιμοποιηθεί η πλατφόρμα ανάπτυξης λογισμικού Eclipse IDE 4.18.0 (Java ver 15.0.1)

Αρχικά κατεβάσαμε την έκδοση Eclipse IDE 2021-03 από την επίσημη ιστοσελίδα της εφαρμογής : https://www.eclipse.org/downloads/



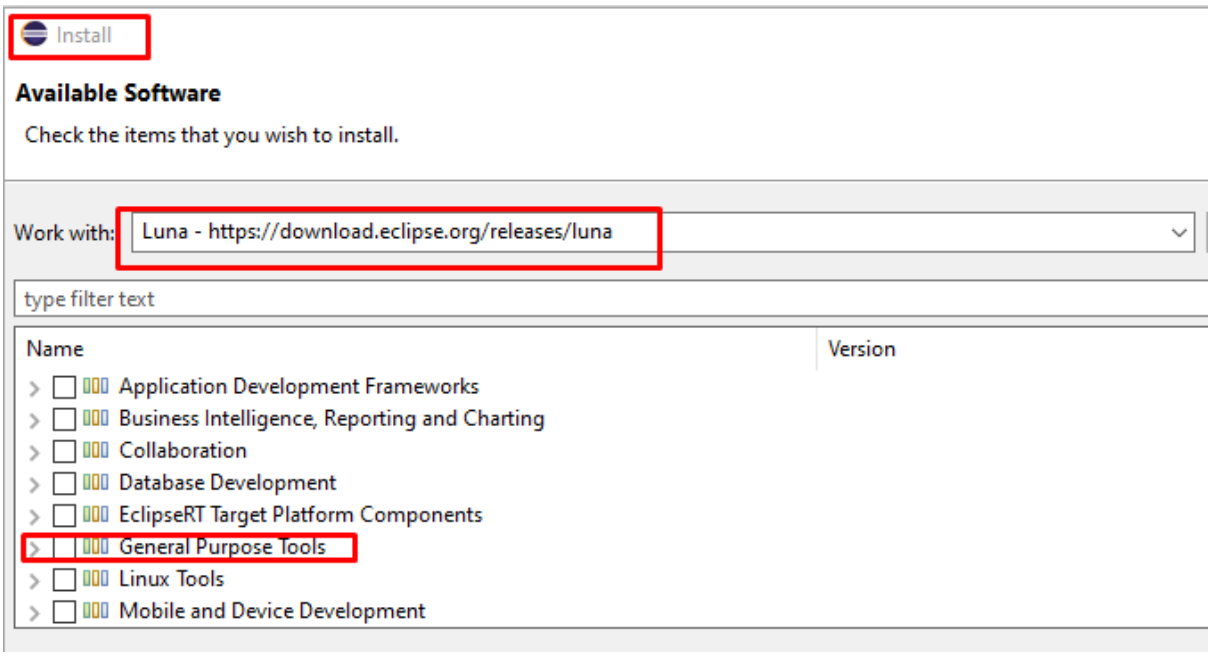
Εικόνα5.1

Μετά την εγκατάσταση του Eclipse IDE το επόμενο βήμα ήταν η εγκατάσταση του WindowBuider ενός πρόσθετου πακέτου για την υποστήριξη γραφικού περιβάλλοντος.

Τρέξαμε το Eclipse IDE και μέσα από το κεντρικό μενού επιλέξαμε

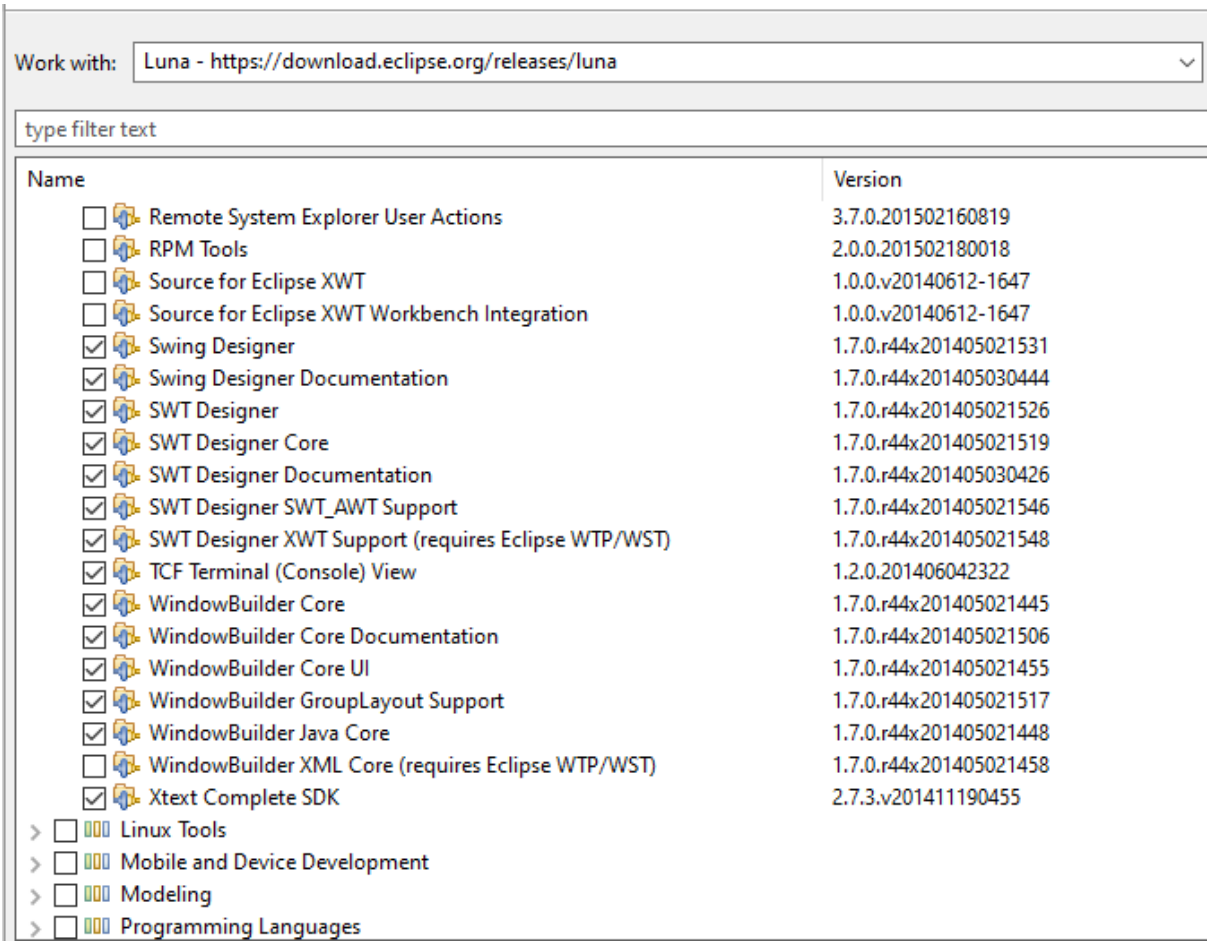
Help/Install New Software…

Εμφανίζεται το παράθυρο Install όπου εισάγουμε το παρακάτω link για την εγκατάσταση του πακέτου.



Εικόνα 5.2: Στο πεδίο "Work with" εισάγουμε το link από το οποίο θα κατέβουν οι προσθήκες που θα επιλέξουμε να εγκατασταθούν στην πλατφόρμα για τις ανάγκες της εργασίας.

Ανοίγουμε τα General Purpose Tools και επιλέγουμε τα πακέτα της εικόνας 5.2



Εικόνα 5.3: Στην εικόνα βλέπουμε τα πακέτα που προστέθηκαν για την υποστήριξη του γραφικού περιβαλλοντος.

Μετά την εγκατάσταση των πακέτων και την επανεκκίνηση της εφαρμογής ξεκινήσαμε ένα νέο project επιλέγοντας από το κεντρικό μενου:

File / New / Other...

Στο παράθυρο “Select a wizard” επιλέγουμε WindowsBuilder/ Swing Designer/ Application Window και δημιουργήσαμε το project: MQTT_GATEWAY.

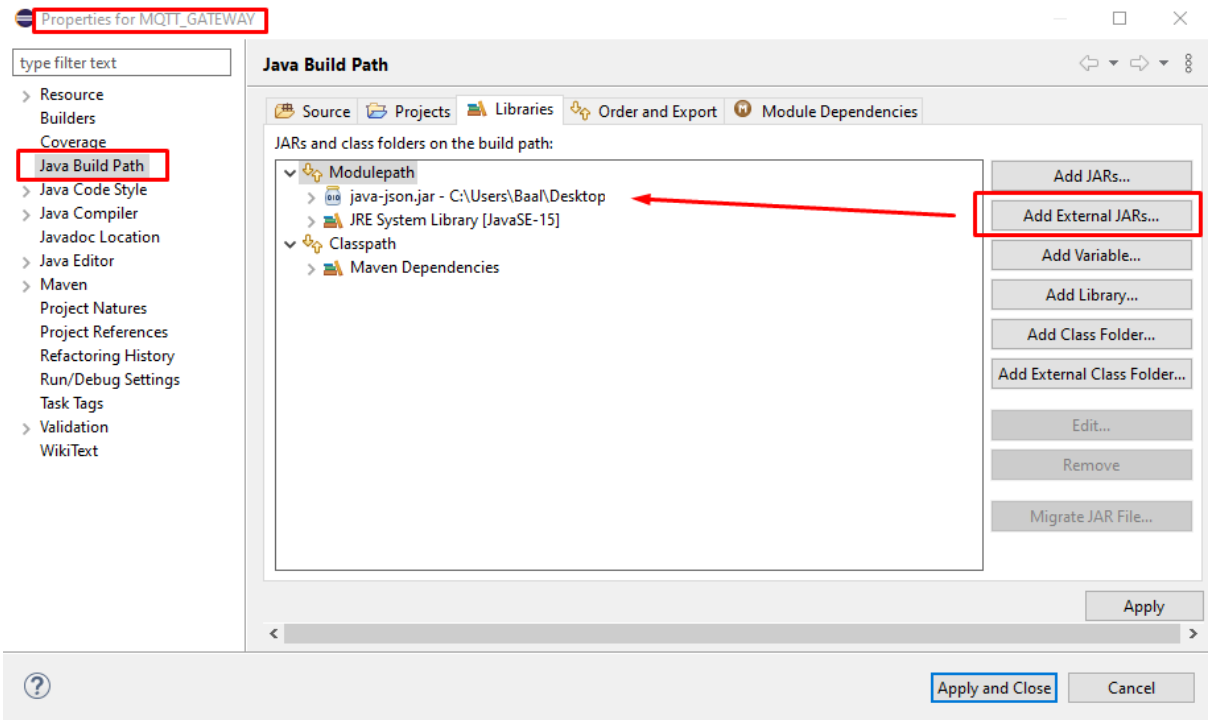
Μετά τη δημιουργία του project έγινε εγκατάσταση μιας βιβλιοθήκης για υποστήριξη Json από το παρακάτω link:

http://www.java2s.com/Code/Jar/j/Downloadjavajsonjar.htm

Για την εγκατάσταση του αρχείου java-json.jar που περιέχει η παραπάνω βιβλιοθήκη επιλέξαμε από το κεντρικό μενού:

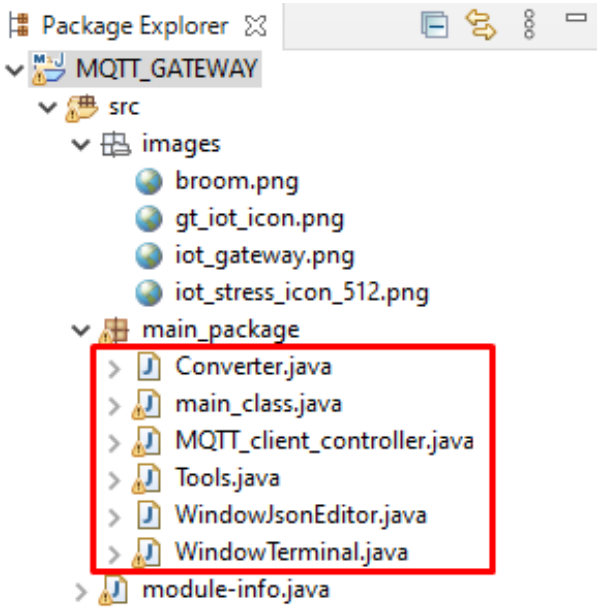
Project > Build Path > Configure build path

και στο εμφανιζόμενο παράθυρο από το tab Libraries δώσαμε την επιλογή Add External Libraries και επιλέξαμε το αρχείο .jar

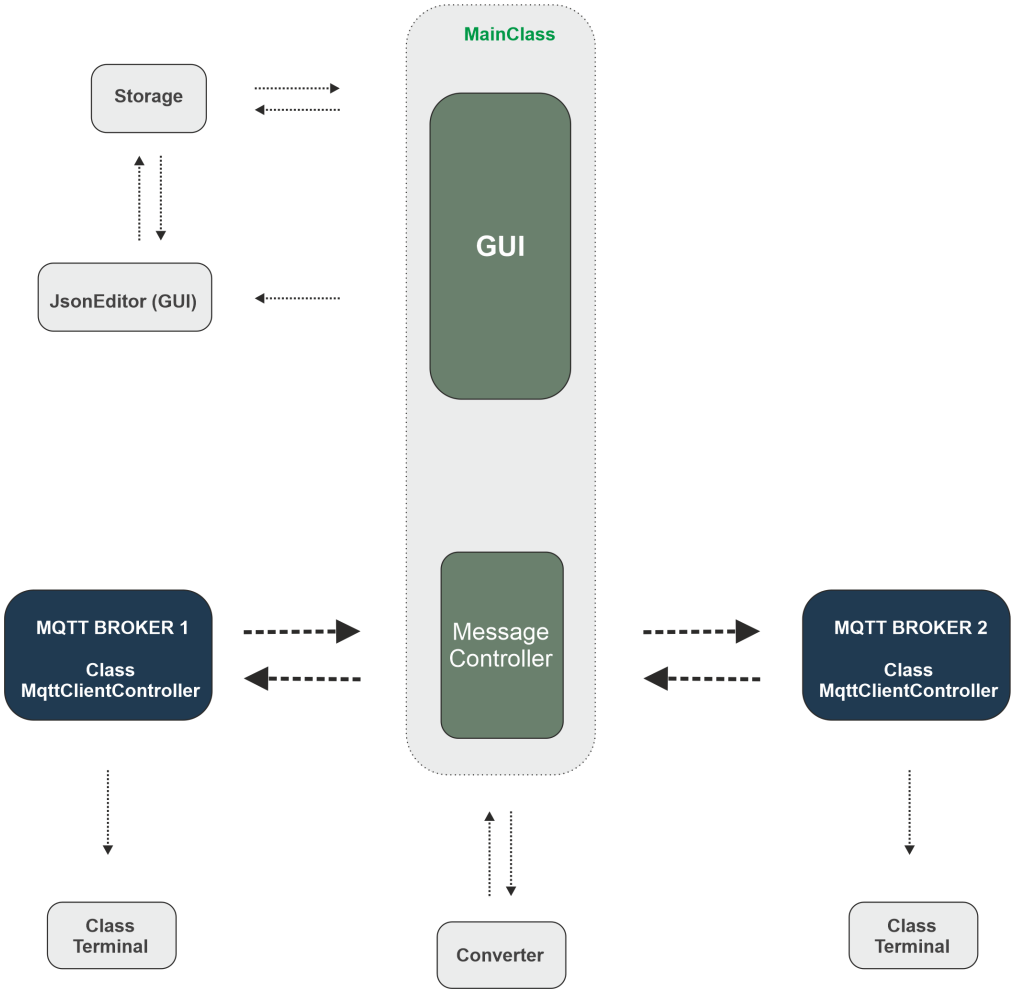


Εικόνα 5.4: Ο διαχειριστής βιβλιοθηκών της πλατφόρμας Eclipse

Η gateway θα διαθέτει ένα μικρό γραφικό περιβάλλον μέσα από το οποίο θα μπορούμε να εισάγουμε τις παραμέτρους σύνδεσης των δύο MQTT broker, να κάνουμε εύκολα τη μορφοποίηση των αρχείων που θα καθορίζουν τον τρόπο μετατροπής των μηνυμάτων καθως και να βλέπουμε μέσα από τερματικά την κίνηση και μετατροπή των μηνυμάτων.



Εικόνα 5.5 Τα αρχεία της IoT Gateway



Εικόνα 5.6: Στο διάγραμμα βλέπουμε την εσωτερική αρχιτεκτονική με την οποία δημιουργήθηκε ο κώδικας της gateway.

Αναπτύχθηκαν έξι συνολικά κλάσεις ο κώδικας των οποίων βρίσκεται στο παράρτημα της παρούσας εργασίας.

Class MainClass:

Η κλάση αυτή είναι η πρώτη κλάση της οποίας ο κώδικας εκτελείται με την εκτέλεση της εφαρμογής.

Περιέχει το γραφικό περιβάλλον αλλά ταυτόχρονα περιέχει και τον κώδικα που ελέγχει τα μηνύματα που έρχονται από τον κάθε broker και αποστέλλονται στον άλλον αφού πρωτα σταλούν σε άλλες κλάσεις για να υποστούν επεξεργασία.

Class MQTT_client_controller:

Η κλάση αυτή περιέχει τις απαραίτητες ρουτίνες για σύνδεση σε broker.

Class Converter:

Είναι η βασικότερη κλάση της εφαρμογής και περιέχει όλες τις συναρτήσεις για την ανάγνωση και αναδημιουργία των Json μηνυμάτων που στέλνονται μεταξύ των brokers που είναι συνδεδεμένοι με την gateway.

Class Tools:

Περιέχει βοηθητικές ρουτίνες εγγραφής και ανάγνωσης αρχείων

Class WindowJsonEditor:

Περιέχει έναν απλό editor Json με δυνατότητα ανίχνευσης λαθών στη μορφή του Json κειμένου που εισάγεται από τον χρήστη

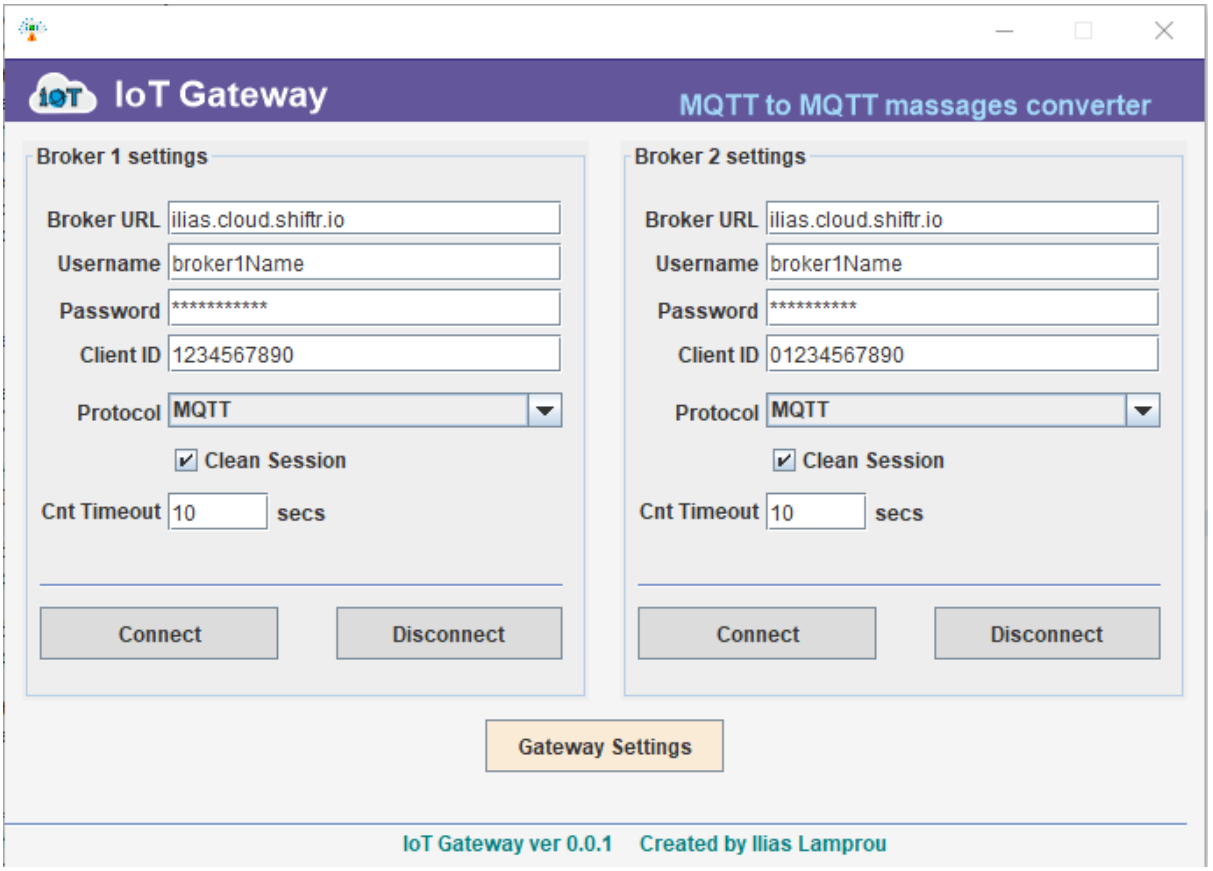
Class WindowTerminal:

Περιέχει τον κώδικα για τα δύο τερματικά που εμφανίζουν την επικοινωνία της Gateway με τους Brokers

Ο κώδικας της IoT Gateway είναι διαθέσιμος στο GitHub:
https://github.com/iliaslmp/MQTT_GATEWAY.git

Το γραφικό περιβάλλον της εφαρμογής.

Τρέχοντας την gateway μέσα από το περιβάλλον Windows 10 ανοίγει το βασικό παράθυρο της εφαρμογής.



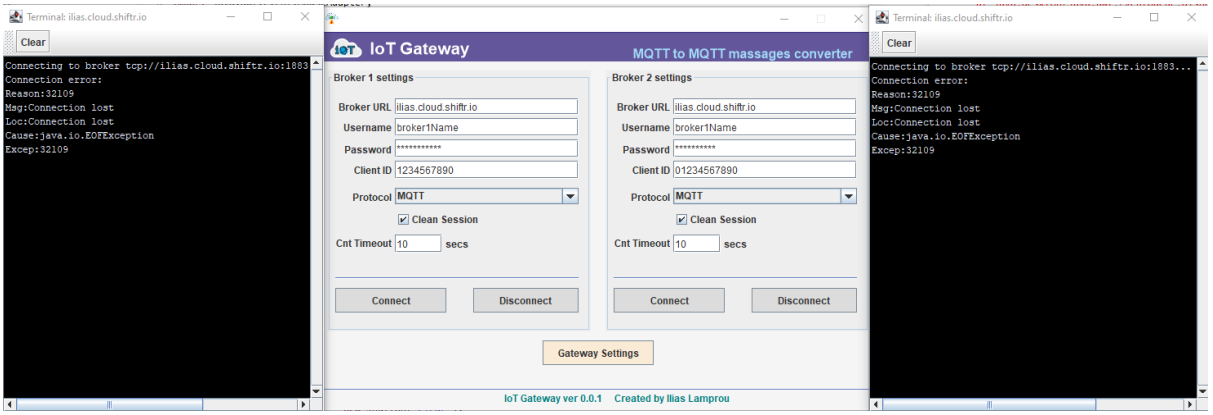
Εικόνα 5.7: Το γραφικό περιβάλλον της εφαρμογής

Αυτό βασικά χωρίζεται σε δύο μέρη τα οποία περιέχουν τα πεδία για την εισαγωγή των παραμέτρων σύνδεσης στους δύο brokers.

Στο κάτω μέρος το μπουτόν “Gateway Settings” θα ανοίξει το παράθυρο του Json Editor που αναφέραμε προηγουμένως ώστε να γίνει η παραμετροποίηση για την μετατροπή των μηνυμάτων.

Ο χρήστης αφού εισάγει τα στοιχεία που απαιτούνται για τη σύνδεση σε κάθε broker μπορεί στη συνέχεια να δοκιμάσει αυτή τη σύνδεση ξεχωριστά για τον καθένα.

Πιέζοντας το μπουτόν “Connect” θα γίνει αυτόματη αποθήκευση των παραμέτρων του broker και θα ανοίξει ένα παράθυρο Terminal στον οποίο ο συγκεκριμένος broker θα εμφανίζει πληροφορίες σύνδεσης καθώς και τα εισερχομενα και εξερχόμενα μηνύματα.



Εικόνα 5.8: Στην εικόνα βλέπουμε το βασικό παράθυρο της εφαρμογής καθώς και τους δύο terminals (έναν για κάθε broker)

Εικόνα 5.9: Στην εικόνα βλέπουμε τον Json Editor μέσα από τον οποίο θα γίνει η μορφοποίηση των μηνυμάτων.

Αρχείο μορφοποίησης JsonConverter.txt

Μέσα από το γραφικό περιβάλλον ανοίγουμε το παράθυρο JsonEditor. Στο παράθυρο αυτό περιέχεται ένα Json με βάση το οποίο θα γίνει η μετατροπή των μηνυμάτων σε συμβατή από κάθε πλατφόρμα μορφοποίηση.

Για κάθε πλατφόρμα θα απαιτείται μια μορφοποίηση του συγκεκριμένου Json.

```
{
        "broker1": [
                {       "subTopic": "sensor/data",
                        "pubTopic": "username/data",
                        "pubSettings":{
                                "qos": 0,
                                "retained" :1
                        },
                        "pubData":{
                         "temperature": "${temp}"
                        }
                },
                {
                        "subTopic": "sensor/+/data",
                        "pubTopic": "username/pressure",
                        "pubSettings":{
                                "qos": 0,
                                "retained" :1
                        },
                        "pubData":{
                         "device" :  "${topic=sensor/*/data}",
                         "pressure" :  "${pressure}"
                        }
                }
        ],

        "broker2": [
        {
                        "subTopic": "username/+",
                        "pubTopic": "test",
                        "pubSettings":{
                                "qos": 0,
                                "retained" :1
                        },
                        "pubData":{
                         "humidity" :  "${topic=temperature/*/me}",
                         "preasure" :  "${preasure}"
```

```
            }
        }

    ]

    }
```

Υποστηρίξαμε στην ανάπτυξη της Gateway ένα παρόμοιο τρόπο μορφοποίησης του αρχείου με αυτόν της πλατφόρμας Thingsboard.

Στη βασική μορφή του αρχείου υπάρχουν δύο Json Arrays με labels broker1 και broker2.

Κάθε ένας από αυτούς τους πίνακες περιέχουν στοιχεία τα οποία είναι επιμέρους Json Objects με τις παραμέτρους μορφοποίησης για κάθε topic ξεχωριστά.

Υπάρχουν δύο βασικές περιπτώσεις, η gateway να κάνει subscribe σε απλό topic ή σε πολλαπλό.

Για κάθε μία περίπτωση θα χρειαστεί διαφορετική μορφοποίηση.

Απλό topic (χωρίς χαρακτήρες + ή #)

```
{       "subTopic": "sensor/data",
        "pubTopic": "username/data",
        "pubSettings":{
                "qos": 0,
                "retained" :1
        },
        "pubData":{
          "temperature": "${temp}"
        }
},
```

Εικόνα 5.10: Μορφοποίηση απλού topic

Για να μορφοποιήσουμε ένα απλό topic πρέπει να καθορίσουμε το topic στο οποίο η gateway θα κάνει subscribe στον broker 1 (πορτοκαλί πλαίσιο).

Στη συνέχεια το topic του Broker στο οποίο θα στείλει το νέο διαμορφωμένο μήνυμα (κόκκινο πλαίσιο).

Στο πεδίο pubSettings καθορίζουμε το QoS σε 0,1,2 (ή σε άλλη τιμή εάν θέλουμε να είναι το ίδιο με το εισερχόμενο μήνυμα) καθώς και το retained του μηνύματος.

Τέλος στο πεδίο pubData καθορίζουμε τη μορφή του μηνύματος που θα αποσταλεί. Αυτό μπορεί να είναι Json ή Text ή μόνο μια τιμή.

Το παράδειγμα της εικόνας 5.10 είναι ένα Json

{“temperature”: “${temp}”}

με ένα μόνο πεδίο temperature, στο οποίο θα τοποθετήσει ως τιμή την τιμή του πεδίου temp από το εισερχόμενο Json.

Ακολουθούμε δηλαδή την ίδια ακριβώς λογική και μορφή με τη μορφοποίηση του Thingsboard.

Σύνθετο topic (με χαρακτήρες + ή #)

```
{
    "subTopic": "sensor/+/data",
    "pubTopic": "username/pressure",
    "pubSettings":{
            "qos": 0,
            "retained" :1
    },
    "pubData":{
      "device" :  "${topic=sensor/*/data}",
      "pressure" :  "${pressure}"
    }
```

Εικόνα 5.11: Μορφοποίηση πολλαπλού topic

Στην εικόνα 5.11 βλέπουμε τη μορφή της διαμόρφωσης ενός πολλαπλού topic.

Η gateway θα κάνει subscribe στον broker 1 το topic sensor/+/data και τα μηνύματα που θα λαμβάνει θα έχουν ως topic στη θέση του + το όνομα της συσκευής και θα είναι της μορφής:

sensor/deviceName/data

Το εξερχόμενο μήνυμα προς τον broker 2 θα σταλεί στο topic /username/pressure (κόκκινο πλαίσιο) και θα είναι μορφής json το οποίο θα έχει και ένα πεδίο στο οποίο θα πρέπει να εισαχθεί η συσκευή από το topic του εισερχόμενου μηνύματος.

Για να γίνει αυτό ορίζουμε το πεδίο ως:

"device" : "${topic=sensor/*/data}"

αυτό θα έχει ως αποτέλεσμα να εισαχθεί στο πεδίο device το κείμενο του topic που βρίσκεται μεταξύ των "**sensor/**" και "**/data**" το οποίο θα πρέπει να είναι το όνομα της συσκευής.

Περισσότερη ανάλυση στον τρόπο μορφοποίησης και των δυνατοτήτων θα υπάρξει στα παραδείγματα παρακάτω.

5.2 ΔΟΚΙΜΗ ΤΗΣ ΛΕΙΤΟΥΡΓΙΑΣ ΤΗΣ IoT GATEWAY ΜΕ ΔΙΑΦΟΡΟΥΣ MQTT CLIENTS ΚΑΙ BROKERS

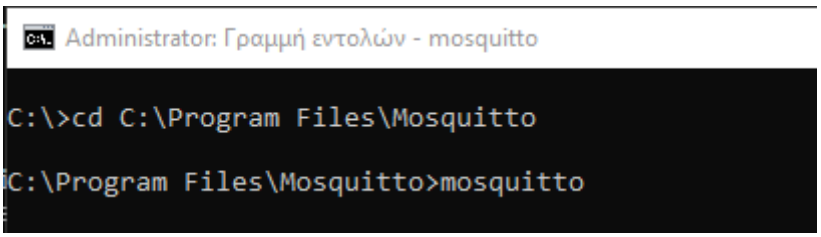
Για να δοκιμάσουμε την Gateway χρησιμοποιήσαμε δύο brokers ένα τοπικό στον οποίο συνδέονται οι συσκευές και ένα broker στο cloud που περιλαμβάνεται σε μια πλατφόρμα.

Τα μηνύματα που θα δοκιμάσουμε να στείλουμε από τη συσκευή έχουν ακριβως την ίδια μορφή με τα μηνύματα που στέλναμε παραπάνω και στην πλατφόρμα Thingsboard.

Τοπικός broker
Ως τοπικός κόμβος χρησιμοποιήθηκε ο broker mosquitto.
Κατεβάσαμε και εγκαταστήσαμε την έκδοση για Windows από την επίσημη ιστοσελίδα:
https://mosquitto.org/download/

Εικόνα 5.12 Οι διαθέσιμες εκδόσεις του mosquitto broker για Windows

Μετά την εγκατάσταση τρέξαμε τον Broker από τη γραμμή εντολών χωρίς να ασχοληθούμε με την παραμετροποίηση του Broker, οπότε δεν θα απαιτηθεί στη συνέχεια κάποιο username και password για να επιτευχθεί η σύνδεση.



Εικόνα 5.13 Τρέχοντας τον Mosquitto broker από το command line

Για να σιγουρέψουμε ότι τρέχει ο broker και η θύρα είναι ανοιχτή δώσαμε την εντολή netstat -an για να δούμε τις ανοιχτές πόρτες.

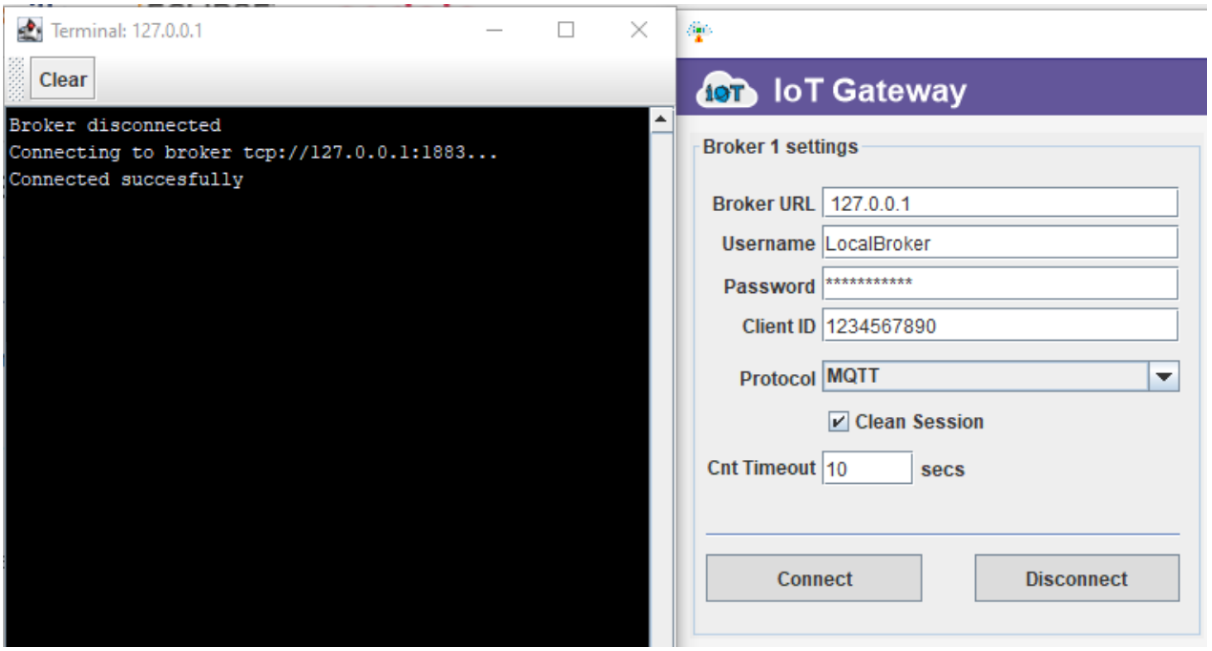
Όπως βλέπουμε στην εικόνα 5.14 η πόρτα 1883 είναι ανοιχτή και έχει αποδοθεί στον broker η IP 127.0.0.1

```
C:\Program Files\Mosquitto>netstat -an

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1309           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:2179           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:5040           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:7680           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49664          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49665          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49666          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49667          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49668          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:51357          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:59098          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:59102          0.0.0.0:0              LISTENING
  TCP    127.0.0.1:1883         0.0.0.0:0              LISTENING
  TCP    127.0.0.1:4800         0.0.0.0:0              LISTENING
  TCP    127.0.0.1:4801         0.0.0.0:0              LISTENING
  TCP    172.18.16.1:139        0.0.0.0:0              LISTENING
  TCP    192.168.1.6:139        0.0.0.0:0              LISTENING
  TCP    192.168.1.6:57390      35.189.87.168:443      ESTABLISHED
  TCP    192.168.1.6:57724      92.123.88.16:443       CLOSE_WAIT
  TCP    192.168.1.6:57725      93.184.220.29:80       CLOSE_WAIT
  TCP    192.168.1.6:58050      198.41.30.198:443      CLOSE_WAIT
  TCP    192.168.1.6:58526      140.82.113.25:443      ESTABLISHED
```

Εικόνα 5.14: Το αποτέλεσμα της εντολής netstat -an

Στη συνέχεια εισάγουμε τις παραμέτρους του mosquitto broker στην Gateway και πιέζουμε το μπουτόν “Connect”.

Ανοίγει αυτόματα το παράθυρο “Terminal” (εικόνα 5.15) και βλέπουμε ότι η σύνδεση ήταν επιτυχής.



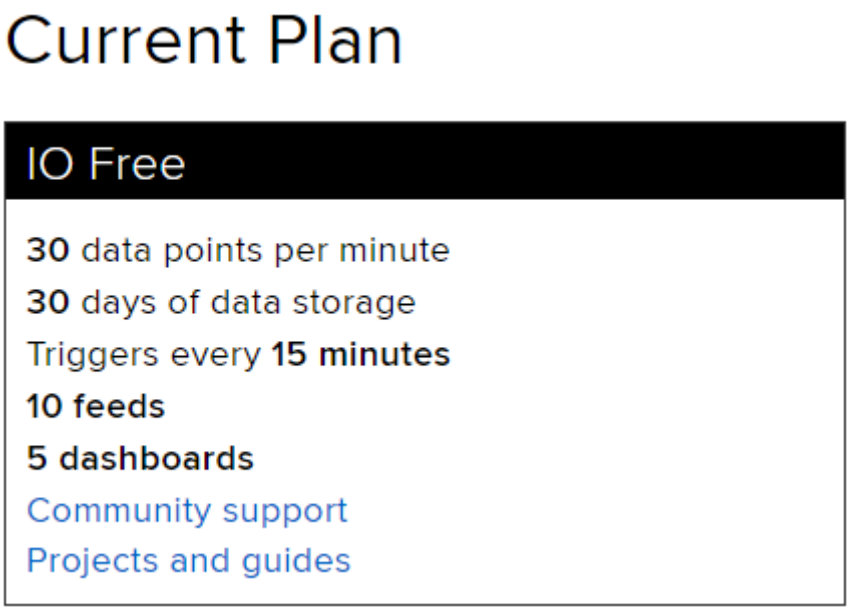
Εικόνα 5.15: Στην εικόνα βλέπουμε τις παραμέτρους σύνδεσης για τον τοπικό mosquitto broker.

Μετά την επιτυχή σύνδεση με τον τοπικό broker θα επιχειρήσουμε σύνδεση με τον broker της πλατφόρμας Adafruit.

Ο λόγος που επιλέχθηκε αρχικά αυτή η πλατφόρμα για τη δοκιμή είναι γιατί υποστηρίζει ένα συγκεκριμένο format στα topics και απαιτεί να στέλνονται μόνο οι τιμές των αισθητήρων χωρίς να εμπεριέχονται σε μηνύματα Json.

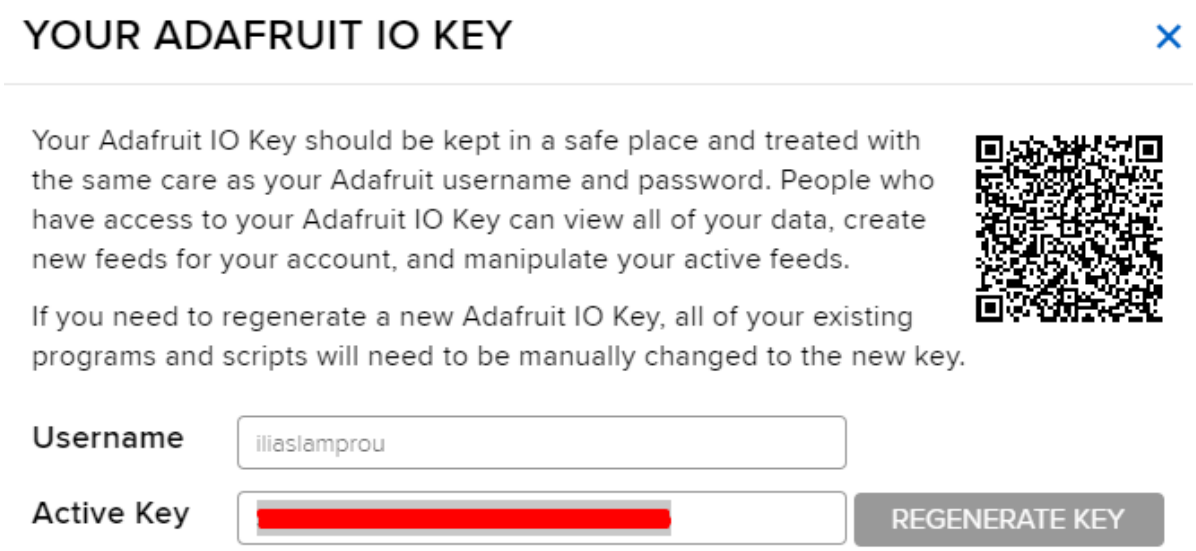
Αυτό σημαίνει ότι τα Json μηνύματα της συσκευής μας θα πρέπει να μετατραπούν από την Gateway σε τιμές και τα πεδία του Json μηνύματος να μετατραπούν σε topics

Αρχικά δημιουργούμε ένα λογαριασμό στην πλατφόρμα στη διεύθυνση https://io.adafruit.com/ και επιλέγουμε το ελεύθερο πακέτο το οποίο μας δίνει τις παρακάτω δυνατότητες οι οποίες είναι αρκετές για τις ανάγκες της δοκιμής.



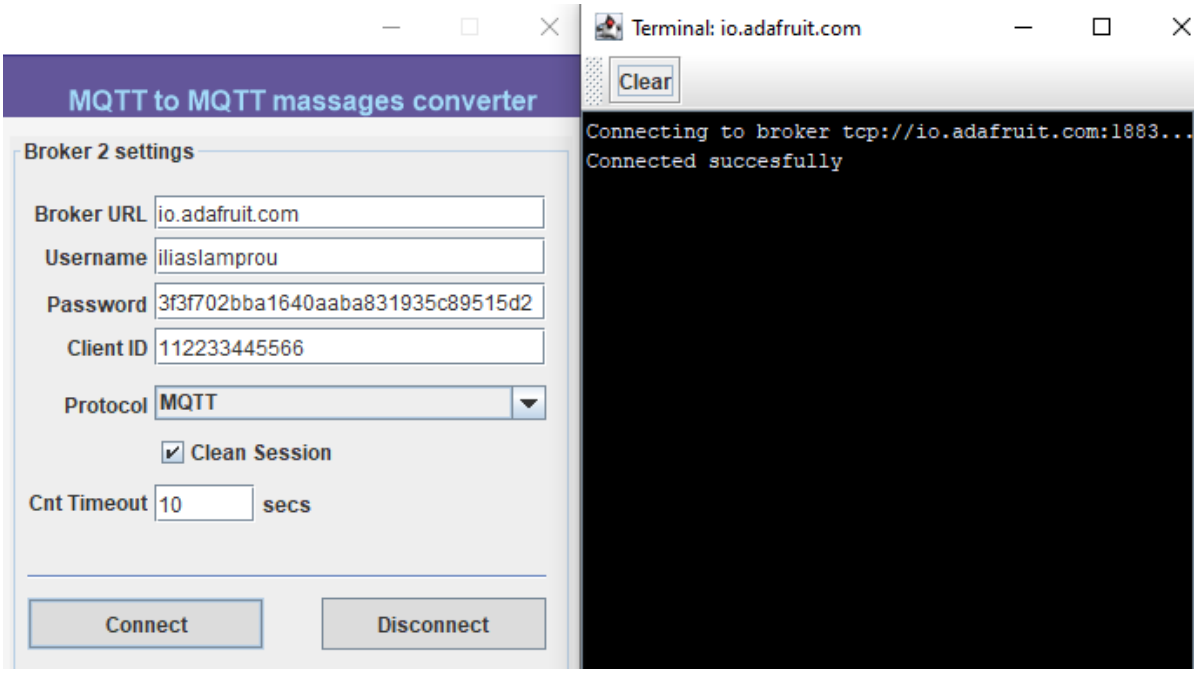
Εικόνα 5.16: Στην εικόνα βλέπουμε τις δυνατότητες που μας παρέχει το ελεύθερο πακέτο της πλατφόρμας adafruit.io

Μετά τη δημιουργία του λογαριασμού μέσα από το κεντρικό μενού της πλατφόρμας επιλέγουμε "My Key" και βλέπουμε το Username και το Active key που μας δίνει η πλατφόρμα για να πραγματοποιήσουμε την σύνδεση σε αυτή (εικόνα 5.16).



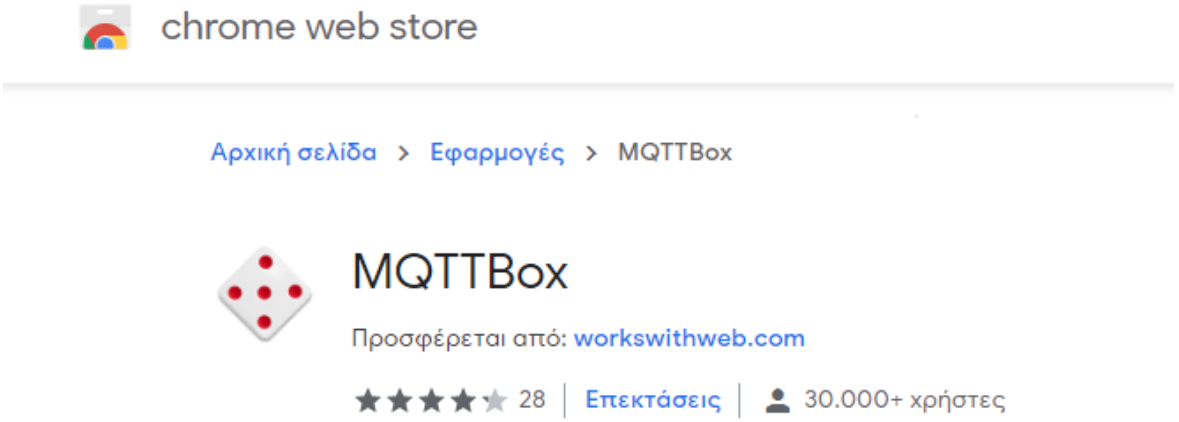
Εικόνα 5.17 Το παράθυρο με τις παραμέτρους σύνδεσης των συσκευών στην πλατφόρμα adafruit.io

Εισάγουμε τις παραμέτρους της πλατφόρμας adafruit.io στην gateway και πιέζουμε σύνδεση.



Εικόνα 5.18 Τα αποτελέσματα της προσπάθειας σύνδεσης στον MQTT broker εμφανίζονται στο τερματικό

Θα πραγματοποιήσουμε στη συνέχεια μερικές αποστολές μηνυμάτων τόσο από τη συσκευή μας όσο και από ένα επιπλέον MQTT client.



Εικόνα 5.19: Απο το chrome web store κατεβάζουμε τον MQTT Client MQTTBox για να τον χρησιμοποιήσουμε για αποστολή και λήψη μηνυμάτων προς τον τοπικό mosquitto broker ώστε να δοκιμάσουμε μορφές μηνυμάτων πέρα από αυτές που υποστηρίζονται από τη συσκευή μας η οποία περιέχει το project διαχείρισης ενέργειας.

Όπως αναφέρθηκε παραπάνω η πλατφόρμα έχει ένα συγκεκριμένο τρόπο με τον οποίο δέχεται τις τιμές από τις συσκευές.

Για παράδειγμα για να στείλουμε την θερμοκρασία στην πλατφόρμα θα πρέπει το topic να έχει τη μορφή username/feeds/temperature και το payload να είναι μόνο η τιμή που έχει η θερμοκρασία.

Αυτό σημαίνει ότι για να σταλεί το μήνυμα Json της μορφής
{“temperature”: 34.2}
από τη συσκευή μας στην πλατφόρμα Adafruit θα πρέπει η Gateway να στείλει απλώς την τιμή 34.2 ως payload στο topic username/feeds/temperature.

Σε περιπτωση που η συσκευή που κάνει την αποστολή συμπεριλαμβάνει τις τιμές των αισθητήρων σε ένα Json, τότε η Gateway θα πρέπει όχι απλώς να μορφοποιήσει το μήνυμα αλλά και να το διασπάσει και να το στείλει σε διαφορετικά topics.

Θα πραγματοποιήσουμε στη συνέχεια δύο δοκιμές για να ελέγξουμε τη λειτουργία της Gateway σε αυτές τις δύο καταστάσεις

Παράδειγμα 1:
Μετατροπή ενός Json εισερχόμενου μηνύματος που περιέχει ένα πεδίο με μια τιμή, σε άλλο το οποίο περιέχει ως payload μόνο την τιμή, ενώ στο topic θα περιέχει και το πεδίο ή μόνο αυτό.
Στο παράδειγμα αυτό θα μορφοποιησουμε το αρχείο JsonConveretr.txt ώστε η gateway να δέχεται ένα μήνυμα Json από ένα απλό topic του broker 1 και να στέλνει ένα μήνυμα το οποίο θα περιέχει μόνο τιμές σε ένα προκαθορισμένο topic του broker2.

Το μήνυμα θα σταλεί από τον MQTTBox client στον broker1. Η gateway αφού το μορφοποιήσει θα το αποστείλει σε ένα νέο topic στον broker2 που είναι ο broker της πλατφόρμας Adafruit.io

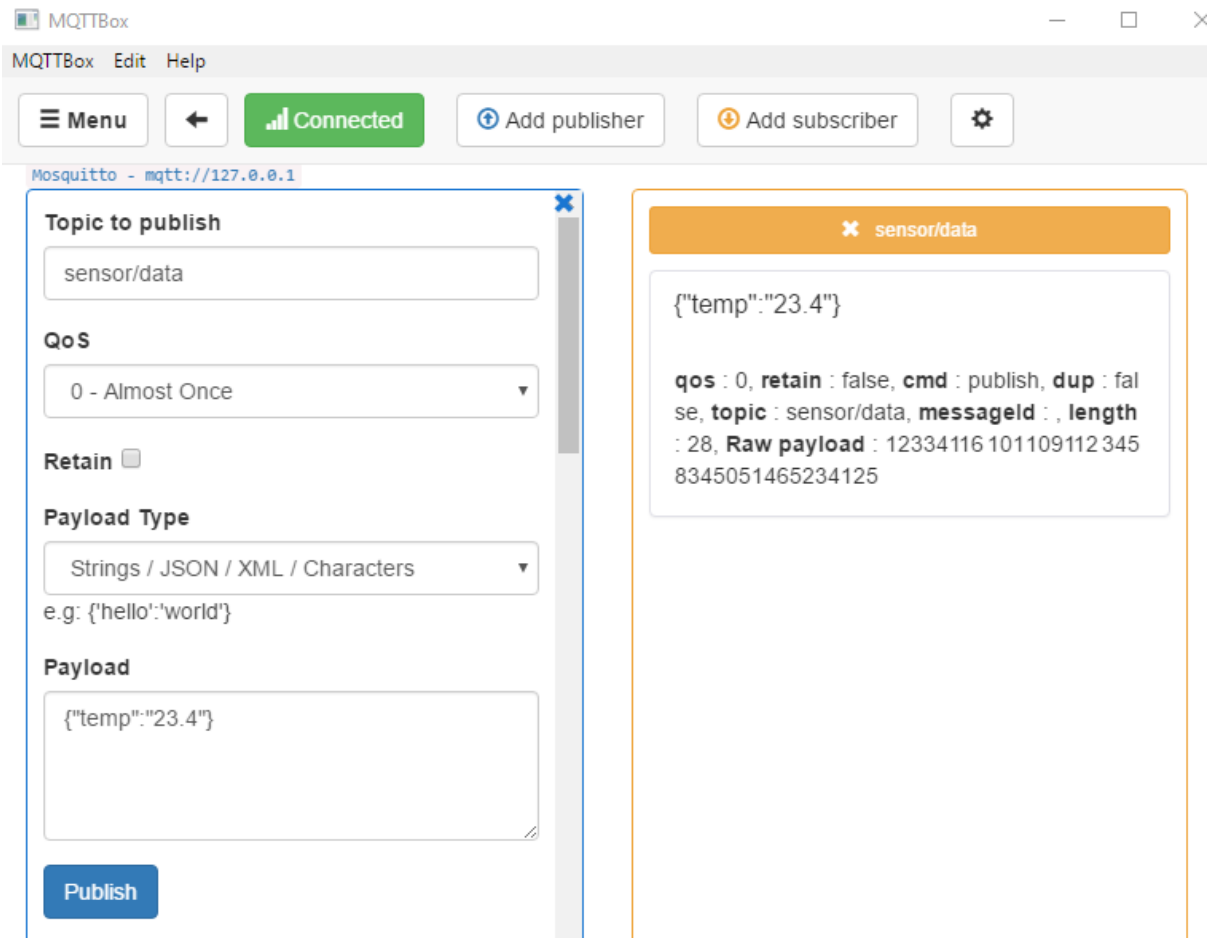
Οπότε έχουμε:

Εισερχόμενο μήνυμα από broker1:
topic: sensor/data
payload: {“temp”: 23.4}

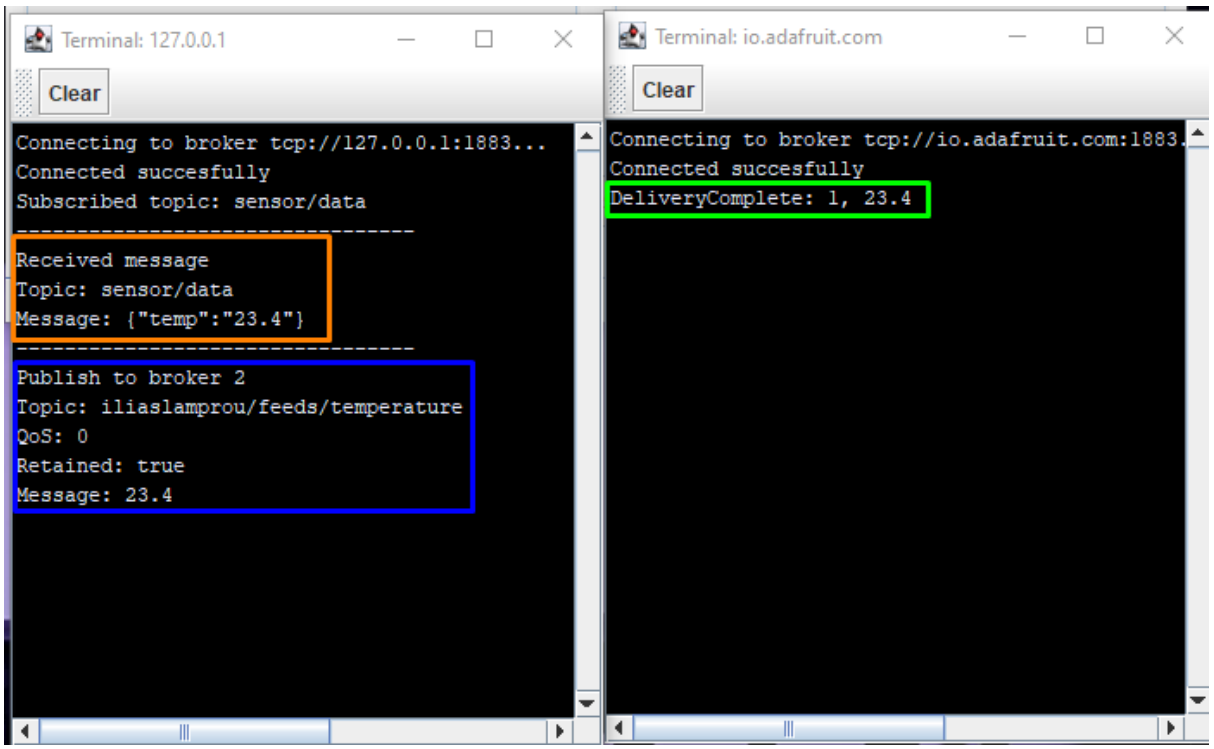
Εξερχόμενο μήνυμα προς broker2:
topic: iliaslamprou/feeds/temperature
payload: “23.4”

Το αρχείο JsonConverter διαμορφώνεται όπως παρακάτω:

```
{
    "broker1": [
        {       "subTopic": "sensor/data",
                "pubTopic": "iliaslamprou/feeds/temperature",
                "pubSettings":{
                        "qos": 0,
                        "retained" :1
                },
                "pubData":{
                  "temperature": "${temp}"
                }
        }

    ],

    "broker2": [
    ]

    }
```



Εικόνα 5.20: Το παράθυρο του MqttBox Client με το οποίο έγινε η αποστολή του μηνύματος.



Εικόνα 5.21: Τα τερματικά των δύο brokers μετά την αποστολή ενός μηνύματος στον broker1.

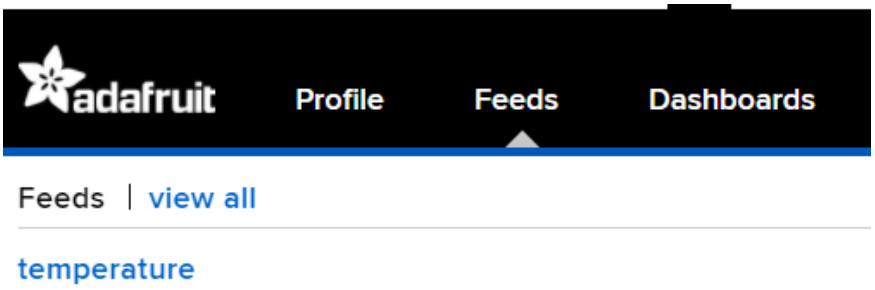
Στην εικόνα 5.21 βλέπουμε στα τερματικά το αποτέλεσμα της αποστολής του παραπάνω μηνύματος στον broker1.

Στο πορτοκαλί πλαίσιο είναι το μήνυμα που έλαβε ο broker 1

Στο μπλε πλαίσιο βλέπουμε το μήνυμα που θα αποσταλεί στον broker2 μετά τη διαμόρφωσή του σύμφωνα με τη μορφοποίηση που καθορίσατε στο αρχείο JsonConverter.txt.

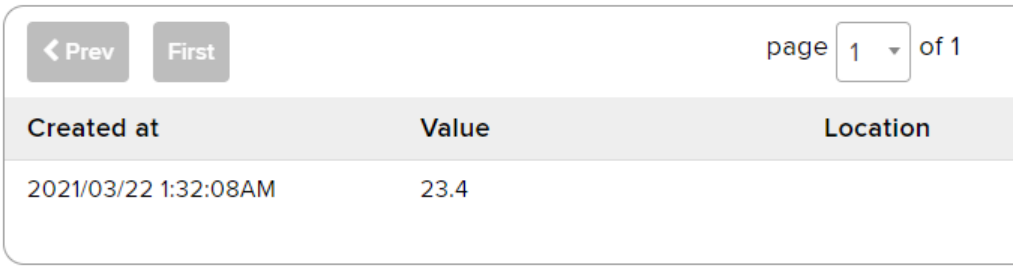
Στο πράσινο πλαίσιο του broker 2 βλέπουμε ότι το μήνυμα παραδόθηκε στην πλατφόρμα με επιτυχία.

Στη σελίδα τώρα της πλατφόρμας adafruit.io μέσα από το κεντρικό μενού επιλέγουμε Feeds (εικόνα 5.22).



Εικόνα 5.22: Στην εικόνα βλέπουμε το topic στο οποίο έγινε η αποστολή να εμφανίζεται στην πλατφόρμα αυτόματα.

Κάνοντας κλικ στο temperature εμφανίζεται το παράθυρο με τα στατιστικά του topic όπου βλέπουμε και την τιμή που στείλαμε (εικόνα 5.23).



Εικόνα 5.23: Στην εικόνα βλέπουμε την τιμή που στείλαμε στο topic temperature καθώς και την ώρα που έγινε η λήψη του μηνύματος.

Παράδειγμα 2:

Διάσπαση εισερχόμενου Json μηνύματος με πολλές τιμές αισθητήρων, σε πολλά μικρότερα μηνύματα που θα αποσταλούν σε διαφορετικά topics.

Στο παράδειγμα αυτό θα μορφοποιησουμε το αρχείο JsonConveretr.txt ώστε η gateway να δέχεται ένα μήνυμα Json το οποίο θα περιέχει τιμές από πολλούς αισθητήρες, αλλά στη συνέχεια ορισμένες από αυτές τις τιμές θα πρέπει να σταλούν σε διαφορετικά topics στην πλατφόρμα.

Θα χρησιμοποιήσουμε ως αποστολέα τη συσκευή από το project ενεργειακής κτιριακής διαχείρισης.

Όπως αναφέρθηκε παραπάνω η συσκευή στέλνει στον broker1 το παρακάτω Json το οποίο περιέχει και το όνομα της συσκευής κάτι που δεν θα χρειαστούμε στην Adafruit.io.

```
{
"NodeID":"00:11:22:33:44:55",
"Type":"ControlSensor/Sensor",
"Humidity":54,
"Temperature":18,
"CoolantTemp":25,
"OnOff":0,
"HiLow":1,
"msgCount":6
}
```

Ενώ η Gateway θα λάβει το παραπάνω Json θα πρέπει να στείλει στην πλατφόρμα τα παρακάτω μηνύματα:

1. Topic: iliaslamprou/feeds/humidity
   Payload: “54”
2. Topic: iliaslamprou/feeds/temperature
   Payload: “18”
3. Topic: iliaslamprou/feeds/CoolantTemp

Payload: “25”

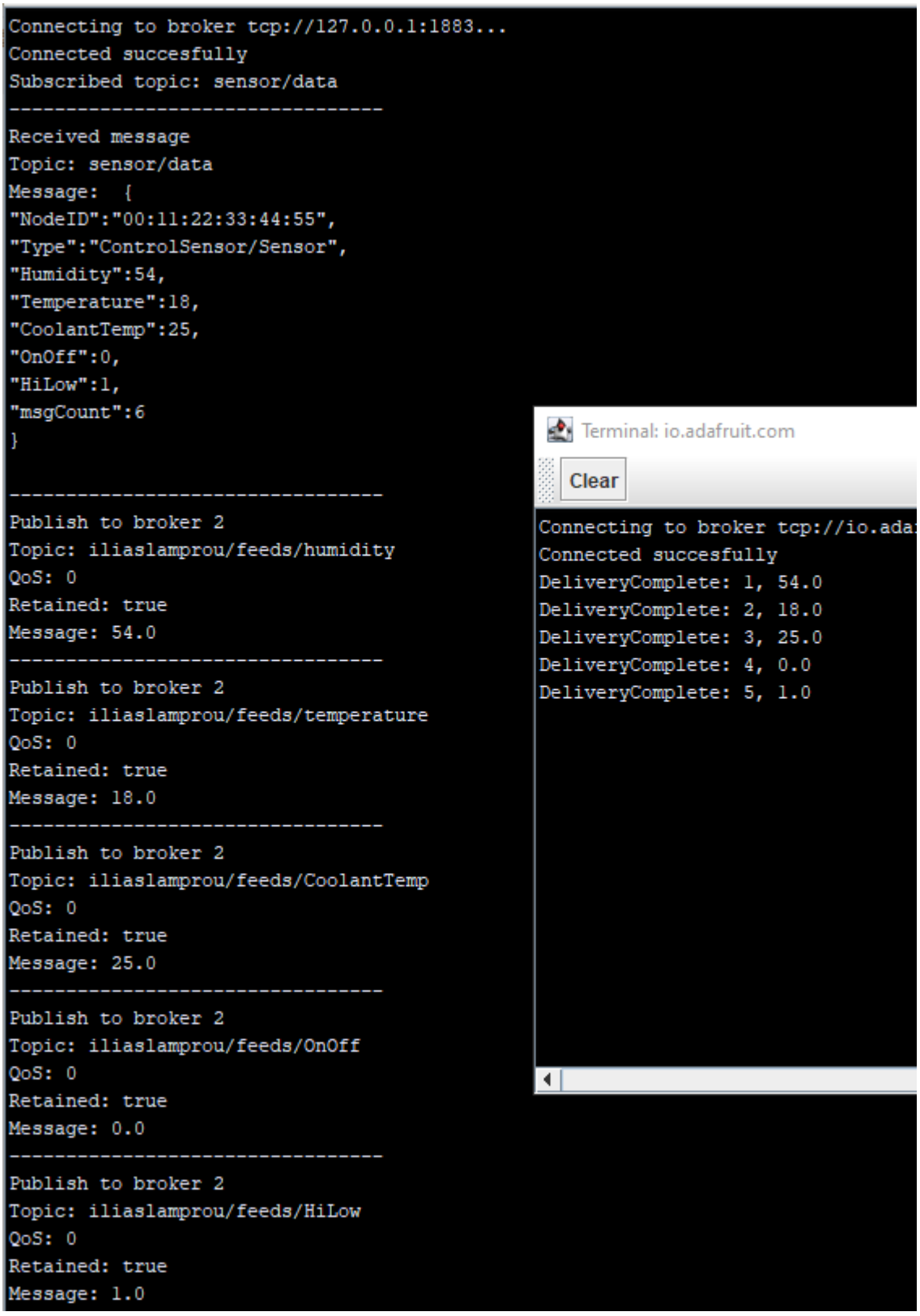
4. Topic: iliaslamprou/feeds/OnOff
   Payload: “0”
5. Topic: iliaslamprou/feeds/HiLow
   Payload: “1”

Το αρχείο JsonConverter για αυτή την περίπτωση μορφοποιείται όπως παρακάτω:

```
{
	"broker1": [
		{	"subTopic": "sensor/data",
			"pubTopic": "iliaslamprou/feeds/humidity",
			"pubSettings":{
				"qos": 0,
				"retained" :1
			},
			"pubData":"${Humidity}"
		},
		{	"subTopic": "sensor/data",
			"pubTopic": "iliaslamprou/feeds/temperature",
			"pubSettings":{
				"qos": 0,
				"retained" :1
			},
			"pubData":"${Temperature}"
		},
		{	"subTopic": "sensor/data",
			"pubTopic": "iliaslamprou/feeds/CoolantTemp",
			"pubSettings":{
				"qos": 0,
				"retained" :1
			},
			"pubData":"${CoolantTemp}"
		},
		{	"subTopic": "sensor/data",
			"pubTopic": "iliaslamprou/feeds/OnOff",
			"pubSettings":{
				"qos": 0,
				"retained" :1
			},
			"pubData":"${OnOff}"
		},
		{	"subTopic": "sensor/data",
```

```
                "pubTopic": "iliaslamprou/feeds/HiLow",
                "pubSettings":{
                        "qos": 0,
                        "retained" :1
                },
                "pubData":"${HiLow}"
        }

],

"broker2": [
]

}
```


Μετά την αποστολή του μηνύματος βλέπουμε τη διάσπαση του εισερχόμενου μηνύματος σε 5 διαφορετικά μηνύματα (εικόνα 5.24)



Εικόνα 5.24: Το εισερχόμενο στην Gateway μήνυμα διασπάστηκε και στάλθηκε στην πλατφόρμα ως διαφορετικά 5 μηνύματα σε διαφορετικά topics.

Feed Name	Key	Last value
CoolantTemp	coolanttemp	25.0
HiLow	hilow	1.0
humidity	humidity	54.0
OnOff	onoff	0.0
temperature	temperature	18.0

Εικόνα 5.25: Οι τελευταίες τιμές που στάλθηκαν από τη συσκευή έχουν πλέον αποθηκευτεί στην πλατφόρμα.

Παράδειγμα 3:

Εισαγωγή του ονόματος της συσκευής ως μέρος του εξερχόμενο topic.

Η πλατφόρμα Adafruit.io δεν μας επιτρέπει να εισάγουμε περισσότερα από τρία πεδία σε ένα topic. Για παράδειγμα δε μπορούμε να έχουμε ένα topic με τέσσερα πεδία όπως το παρακάτω

/iliaslamprou/feeds/deviceName/temperature

ώστε να εισάγουμε τη συσκευή ως ένα μέρος του topic.

Γι' αυτό θα εισάγουμε τη συσκευή ως μέρος του τρίτου πεδίου από το topic και το τελικό topic θα έχει τη μορφή:

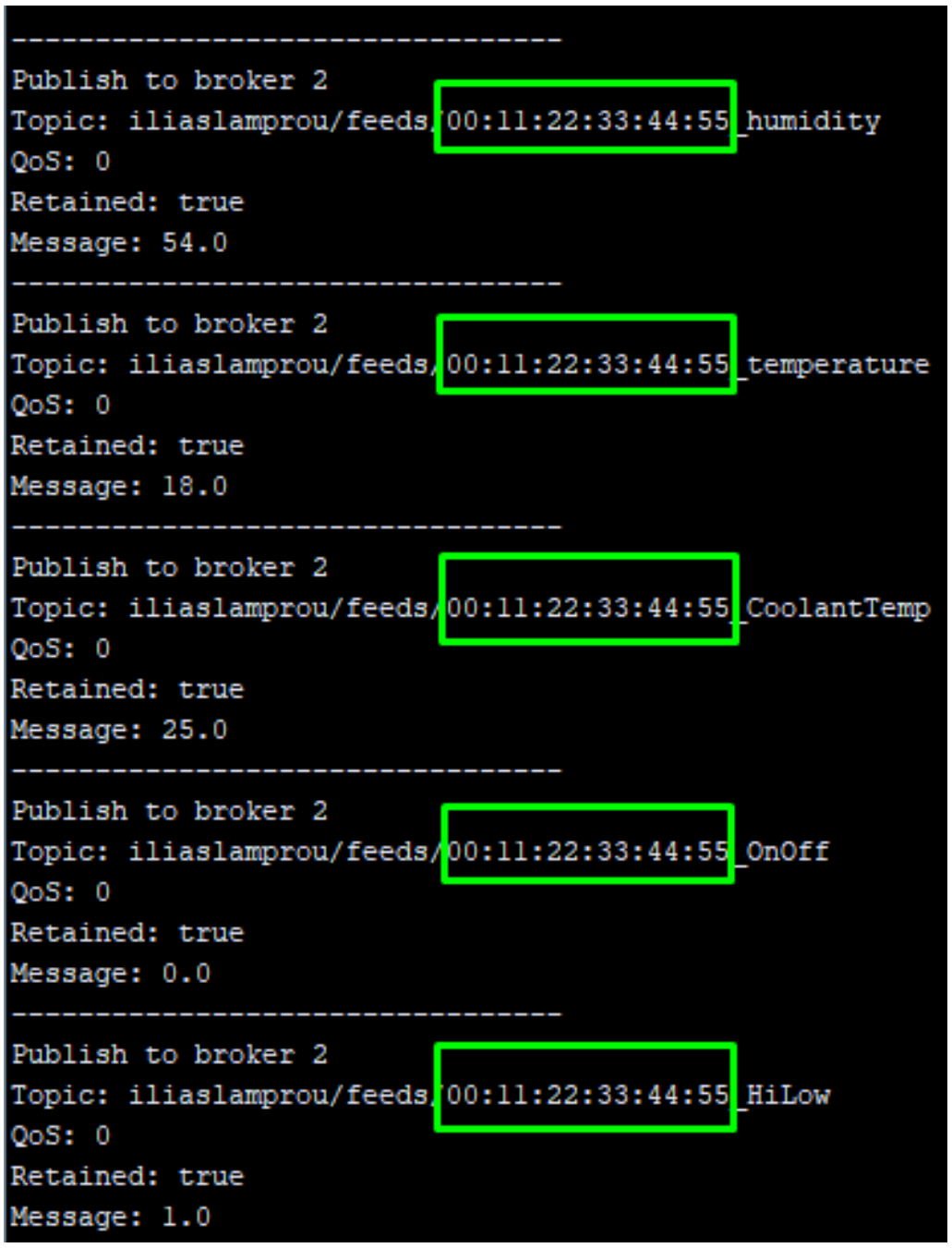
/iliaslamprou/feeds/deviceName_temperature

Για να γίνει αυτό θα εισάγουμε το όνομα της συσκευής ως μεταβλητή της μορφής ${deviceName} ως μέρος του εξερχόμενου topic

Στο αρχείο JsonConverter.txt κάνουμε τις παρακάτω αλλαγές στα πέντε πεδία subTopic

"pubTopic": "iliaslamprou/feeds/${NodeID}_humidity",
"pubTopic": "iliaslamprou/feeds/${NodeID}_temperature",
"pubTopic": "iliaslamprou/feeds/${NodeID}_CoolantTemp",
"pubTopic": "iliaslamprou/feeds/${NodeID}_OnOff",
"pubTopic": "iliaslamprou/feeds/${NodeID}_HiLow",

Μετά την αποστολή του μηνύματος στον terminal, βλέπουμε την εισαγωγή του ονόματος της συσκευής από το πεδίο NodeID του εισερχόμενου Json μηνύματος, στα topics των προς αποστολή μηνυμάτων:



Εικόνα 5.26: Το όνομα της συσκευής προστίθεται στο topic.

Feed Name	Key	Last value
00:11:22:33:44:55_CoolantTemp	00-11-22-33-44-5...	25.0
00:11:22:33:44:55_HiLow	00-11-22-33-44-5...	1.0
00:11:22:33:44:55_humidity	00-11-22-33-44-5...	54.0
00:11:22:33:44:55_OnOff	00-11-22-33-44-5...	0.0
00:11:22:33:44:55_temperature	00-11-22-33-44-5...	18.0
CoolantTemp	coolanttemp	25.0
HiLow	hilow	1.0
humidity	humidity	54.0
OnOff	onoff	0.0
temperature	temperature	18.0

Εικόνα 5.27: Στην πλατφόρμα έχει προστεθεί στο topic το όνομα της συσκευής. Έτσι μπορούν διαφορετικές συσκευές να στέλνουν στην πλατφόρμα τιμές και να κάνουμε διαχωρισμό αυτών με βάση το topic.

Παράδειγμα 4:

Μετατροπή ενός Json εισερχόμενου μηνύματος σε ένα νέο Json εξερχόμενο μήνυμα

Σε αυτό το παράδειγμα θα εκμεταλλευτούμε μια πολύ σημαντική δυνατότητα που μας δίνει η συγκεκριμένη πλατφόρμα που είναι η δημιουργία group τιμών.

Στόχος μας είναι κάθε συσκευή που συνδέεται στον τοπικό broker να στέλνει τις τιμές της σε ένα διαφορετικό group της πλατφόρμας το οποίο θα έχει μάλιστα και το όνομα της συσκευής

Έστω τώρα ότι η gateway λαμβάνει το παρακάτω μήνυμα από τη συσκευή μας

```
{
"NodeID":"00:11:22:33:44:55",
"Type":"ControlSensor/Sensor",
"Humidity":54,
"Temperature":18,
"CoolantTemp":25,
"OnOff":0,
"HiLow":1,
"msgCount":6
}
```

Θα πρέπει να στείλει το παρακάτω μήνυμα:

topic: iliaslamprou/groups/deviceName/json
payload: {

```
"feeds": {
    "humidity": "54",
    "temperature": "18",
    "coolan_ttemp": "25",
    "on_off": "0",
    "hi_low": "1"
}
```

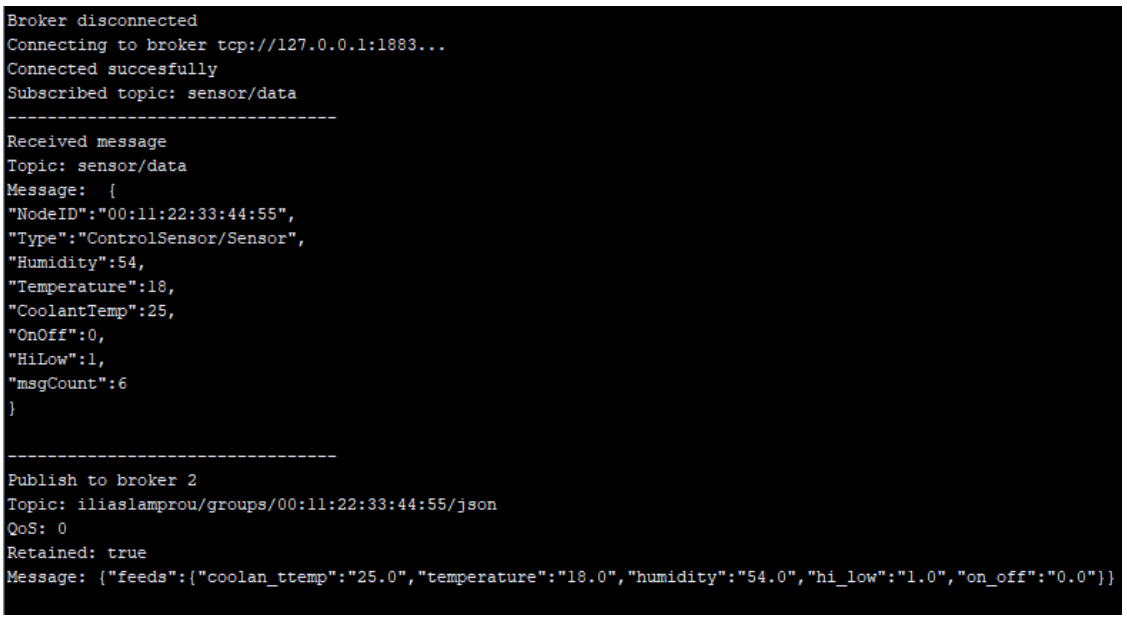
}

Το deviceName θα πρέπει κάθε φορά να αντικαθίσταται από το όνομα της συσκευής, καθώς και οι τιμές των αισθητήρων θα πρέπει να αντικαθίστανται από αυτές του εισερχόμενου μηνύματος.

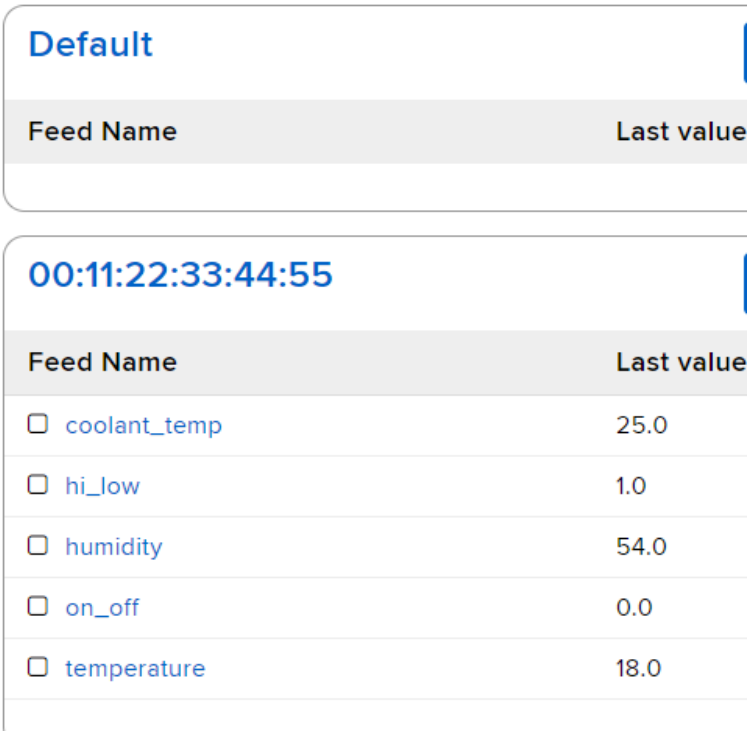
Το αρχείο JsonConverter.txt μορφοποιείται ως παρακάτω:

```
{
    "broker1": [
        {   "subTopic": "sensor/data",
            "pubTopic": "iliaslamprou/groups/${NodeID}/json",
            "pubSettings":{
                "qos": 0,
                "retained" :1
            },
            "pubData":{
                "feeds": {
                    "humidity": "${Humidity}",
                    "temperature": "${Temperature}",
                    "coolan_ttemp": "${CoolantTemp}",
                    "on_off": "${OnOff}",
                    "hi_low": "${HiLow}"
                }
            }

        }
    ],

    "broker2": [
    ]

    }
```

Μετά την αποστολή του μηνύματος από τη συσκευή στο terminal βλέπουμε ότι η μορφοποίηση του προς αποστολή μηνύματος έχει γίνει σωστά.



Εικόνα 5.28: Στην εικόνα βλέπουμε το μήνυμα που εισήχθη στην Gateway καθώς στην τελευταία γραμμή το μήνυμα που εστάλη στον broker της πλατφόρμας Adafruit.io μετά τη μορφοποίηση.



Εικόνα 5.29: Στην πλευρά της πλατφόρμας δημιουργήθηκε αυτόματα ένα group με το όνομα της συσκευής που έστειλε το μήνυμα και πέντε feeds που περιέχουν τις τιμές των αισθητήρων.

Παράδειγμα 5:

Τροποποίηση εισερχόμενων μηνυμάτων όπου το όνομα της συσκευής είναι μέρος του topic το οποίο περιέχει χαρακτήρες '+' και '#'.

Για παράδειγμα ας πούμε ότι έχουμε διάφορες συσκευές που στέλνουν μηνύματα σε ένα topic της μορφής.

sensor/deviceName/data

Η gateway θα πρέπει να κάνει subscribe σε ένα topic της μορφής

sensor/+/data

και στη συνέχεια να λάβει το όνομα της συσκευής από τα topics των εισερχόμενων μηνυμάτων και να τα μεταφέρει είτε στο payload είτε στο topic του εξερχόμενου μηνύματος.

Για να το πετύχουμε αυτό στο προσθέτουμε είτε στο pubTopic είτε στο payload την παρακάτω έκφραση:

${topic=sensor/*/data}

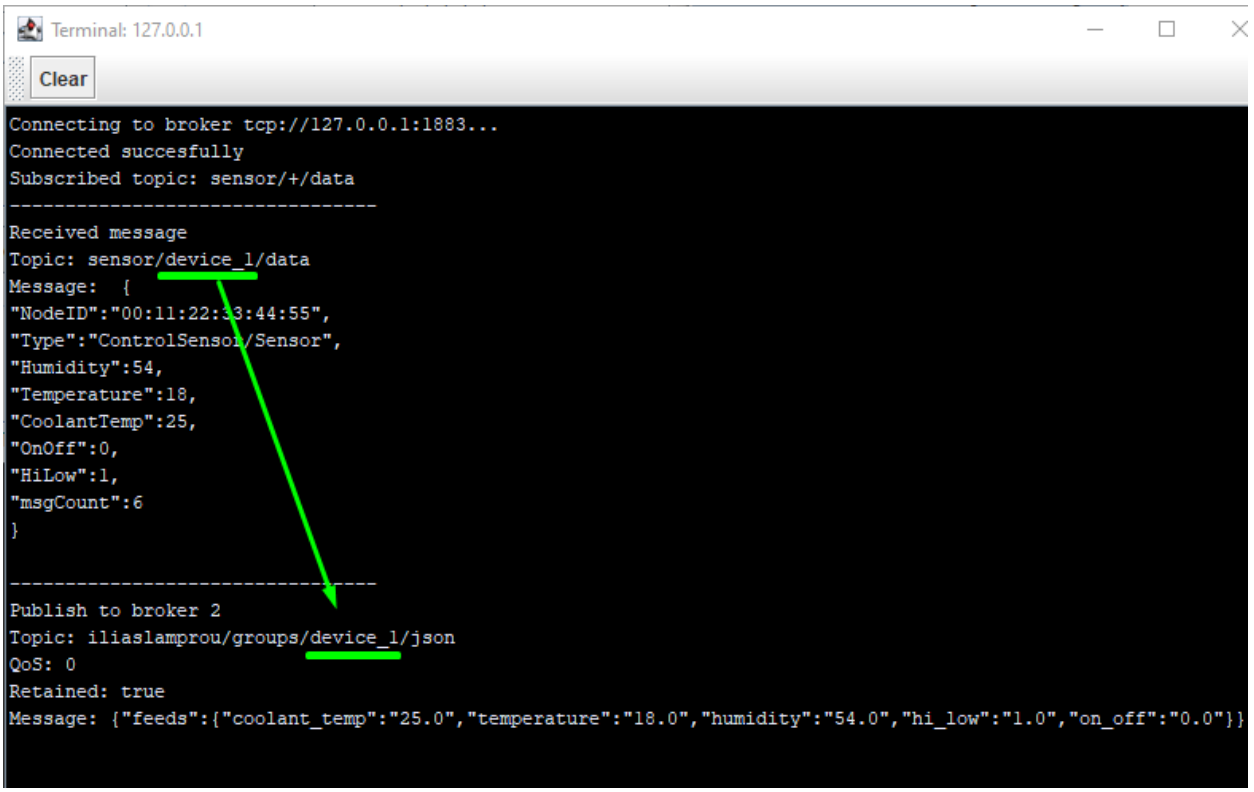
Η λέξη topic στην παραπάνω έκφραση δηλώνει ότι η μεταβλητή αυτή θα λάβει τιμές από το topic και όχι από το payload του εισερχόμενου μηνύματος όπως γινόταν στα προηγούμενα παραδείγματα.

Ότι κείμενο υπάρχει στο εισερχόμενο topic μεταξύ των αλφαριθμητικών "sensor/" και "/data" θα εισαχθεί το εξερχόμενο topic ή payload.

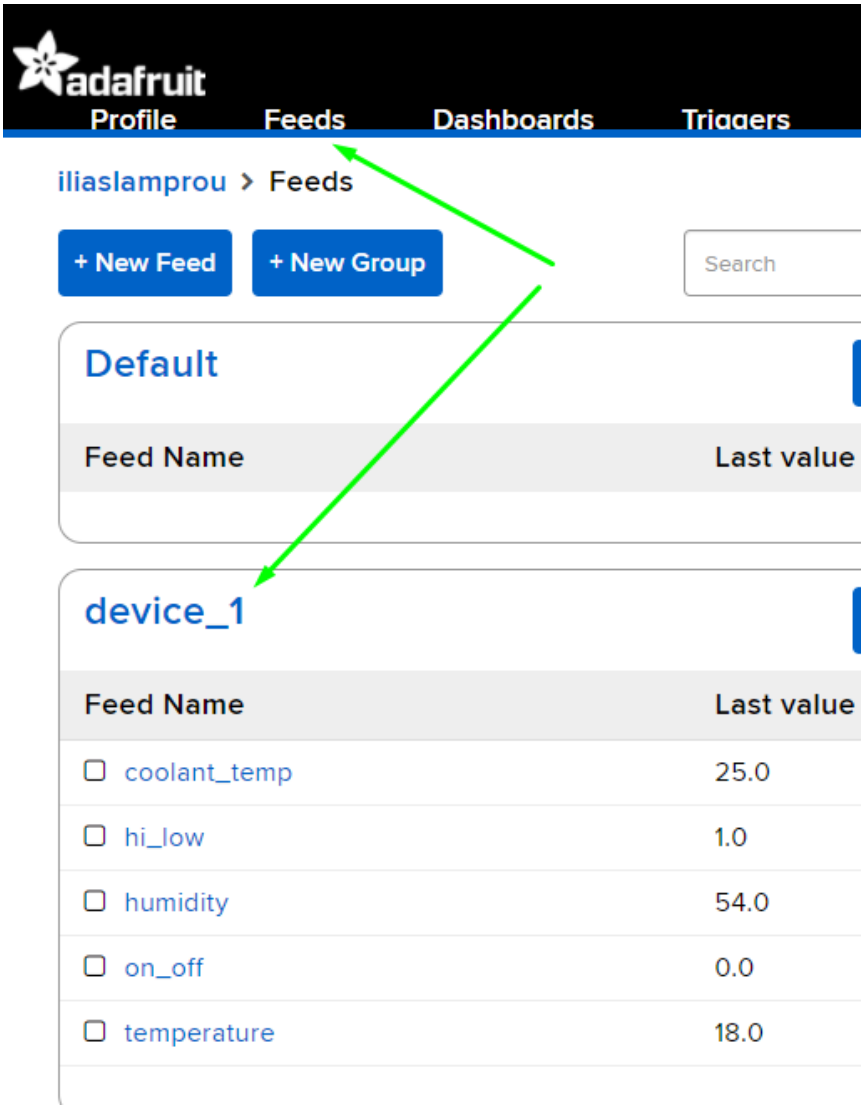
Για το παράδειγμά μας το αρχείο ConverterJson.txt διαμορφώνεται όπως παρακάτω:

```
{
    "broker1": [
        {   "subTopic": "sensor/+/data",
            "pubTopic": "iliaslamprou/groups/${topic=sensor/*/data}/json",
            "pubSettings":{
                "qos": 0,
                "retained" :1
            },
            "pubData":{
                "feeds": {
                    "humidity": "${Humidity}",
                    "temperature": "${Temperature}",
                    "coolant_temp": "${CoolantTemp}",
                    "on_off": "${OnOff}",
                    "hi_low": "${HiLow}"
                }
            }

        }
    ],
```

```
"broker2": [
]

}
```



Εικόνα 5.30: Στο terminal του broker βλέπουμε τη μεταφορά του ονόματος της συσκευής από το εισερχόμενο topic στο topic του εξερχόμενου από την gateway μηνύματος.



Εικόνα 5.31: Στην πλατφόρμα Adafruit.io δημιουργήθηκε ένα group με το όνομα της συσκευής και feeds τα πεδία του Json μηνύματος που στάλθηκε από τη συσκευή.

Αν θέλαμε να εισάγουμε το όνομα της συσκευής στο payload θα μπορούσαμε να κάνουμε την παρακάτω διαμόρφωση:

```
"pubData":{
        "feeds": {
            "device": "${topic=sensor/*/data}",
            "humidity": "${Humidity}",
            "temperature": "${Temperature}",
            "coolant_temp": "${CoolantTemp}",
            "on_off": "${OnOff}",
            "hi_low": "${HiLow}"
            }
    }
```

Παράδειγμα 6:

Διαμόρφωση του αρχείου jsonConverter.txt ώστε η gateway να δέχεται μηνύματα από την πλατφόρμα και να τα στέλνει στη συσκευή.

Για να το πετύχουμε αυτό αυτό θα πρέπει στο αρχείο JsonConverter να εισάγουμε πληροφορίες μορφοποίησης στο πεδίο broker2.

Ισχύουν ακριβώς τα ίδια που αναφέρθηκαν παραπάνω για τον broker1. Η Η gateway μας είναι 100% αμφίδρομη υποστηρίζοντας τον ίδιο τρόπο μορφοποίησης μηνυμάτων και από τις δύο πλευρές.

Για να στείλουμε ένα μήνυμα πχ για ενεργοποίηση της συσκευής, πρέπει να στείλουμε το παρακάτω Json:

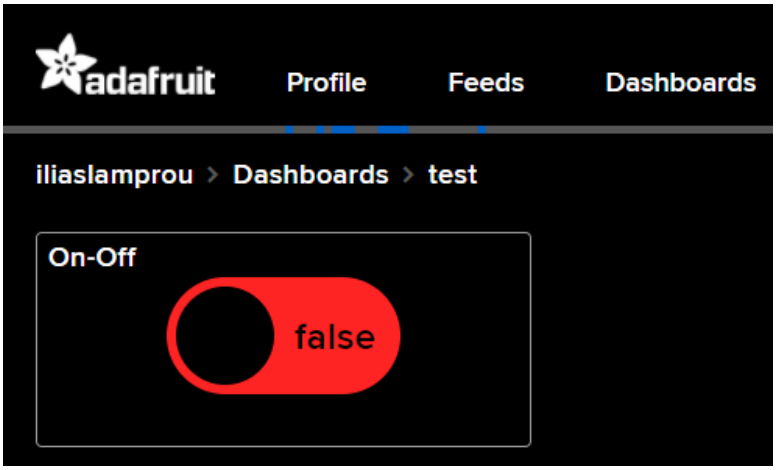
```
{
 "NodeNo": “deviceName”,
 "OnOff": , true",
 "msgCount": 1
}
```

αν παραλείψουμε το msgCount γιατί κάτι τέτοιο δεν μπορεί να σταλεί από την πλατφόρμα προκύπτει το παρακάτω μήνυμα:

```
{
 "NodeNo": “2”,
 "OnOff": , true"
}
```

Στην πλατφόρμα Adafruit.io δημιουργούμε αρχικά ένα νέο feed με όνομα **on_off_cmd.** Αυτό είναι το feed του οποίου την τιμή θα αλλάζουμε από την πλατφόρμα. (Η πλατφόρμα δεν επιτρέπει να αλλάζουμε feeds που εμπεριέχονται μέσα σε groups.)

Στη συνέχεια δημιουργούμε ένα Dashboard και οποίο εισάγουμε ένα Switch.



Εικόνα 5.32: Το Dashboard που δημιουργήσαμε και το απαραίτητο switch για το παράδειγμα.

Το μήνυμα που θα στέλνει η πλατφόρμα κάθε φορά που θα αλλάζουμε την κατάσταση στο switch θα είναι της παρακάτω μορφής θα είναι ένα “true” ή “false”.

Το topic που θα πρέπει να κάνουμε subscribe για να λάβουμε το μήνυμα είναι το:
iliaslamprou/feeds/on_off_cmd

Όπως βλέπουμε έχουμε μια περίπτωση όπου το εισερχόμενο topic περιέχει το device, ενώ το payload περιέχει μόνο μια τιμή true-false..

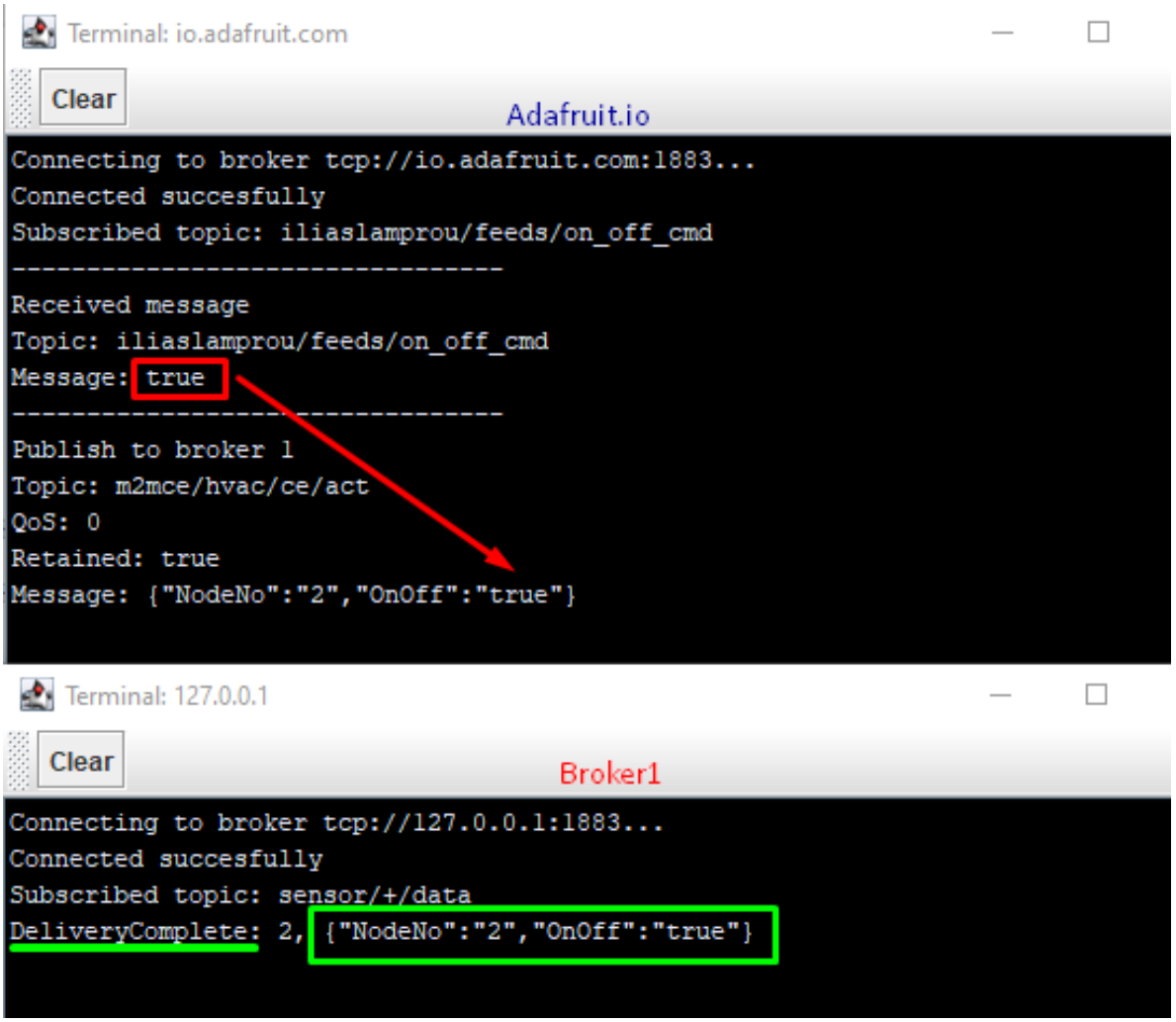
Από την άλλη πλευρά θα πρέπει να στείλουμε στη συσκευή το μήνυμα:
{
"NodeNo": “deviceID”,
"OnOff": , value"
}
στο topic: m2mce/hvac/ce/act

Η gateway να περάσει στο payload ως τιμή του διακόπτη On-Off ολόκληρο το εισερχόμενο μήνυμα. Για να γίνει αυτό ως τιμή βάζουμε τον παρακάτω τύπο μεταβλητής.
${receivedMessage}

Τροποποιούμε το αρχείο jsonConvereter.txt προσθέτοντας τώρα πληροφορίες και στο πεδίο broker2 όπως παρακάτω:

```
{
    "broker1": [
        {   "subTopic": "sensor/+/data",
            "pubTopic": "iliaslamprou/groups/${topic=sendor/*/data}/json",
            "pubSettings":{
                "qos": 0,
                "retained" :1
            },
            "pubData":{
                "feeds": {
                    "humidity": "${Humidity}",
                    "temperature": "${Temperature}",
                    "coolant_temp": "${CoolantTemp}",
                    "on_off": "${OnOff}",
                    "hi_low": "${HiLow}"
                }
            }

        }
    ],

    "broker2": [
        {   "subTopic": "iliaslamprou/feeds/on_off_cmd",
            "pubTopic": "m2mce/hvac/ce/act",
```

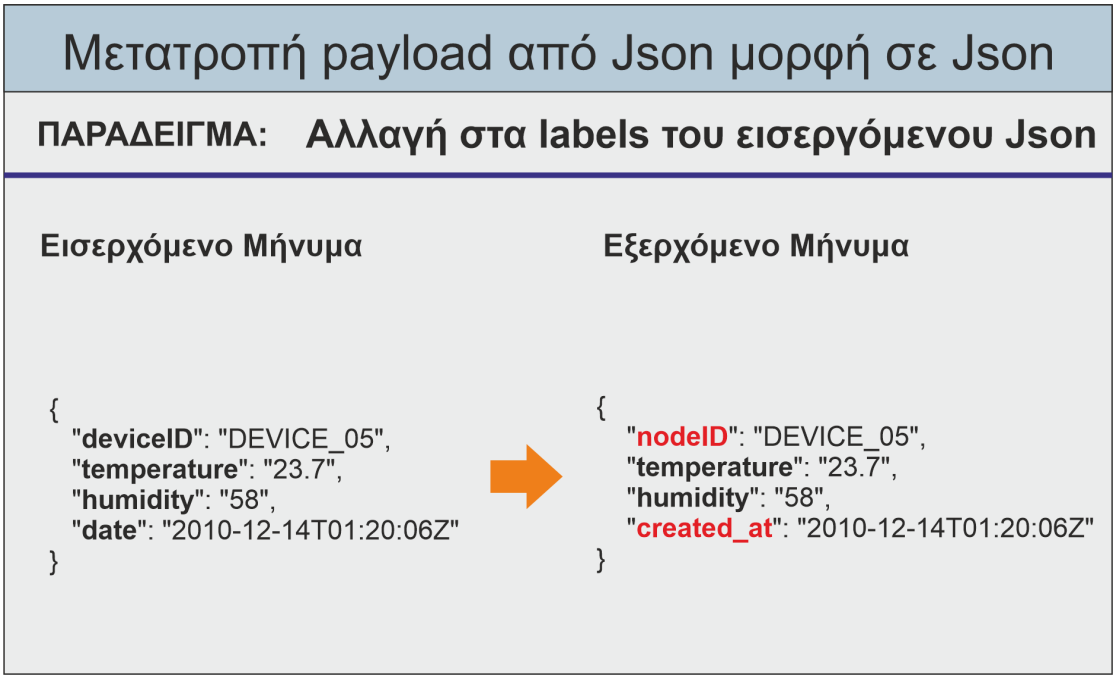
```
                "pubSettings":{
                        "qos": 0,
                        "retained" :1
                },
                "pubData":{
                          "NodeNo": "2",
                          "OnOff": "${receivedMessage}"
                         }


        }
    ]

    }
```



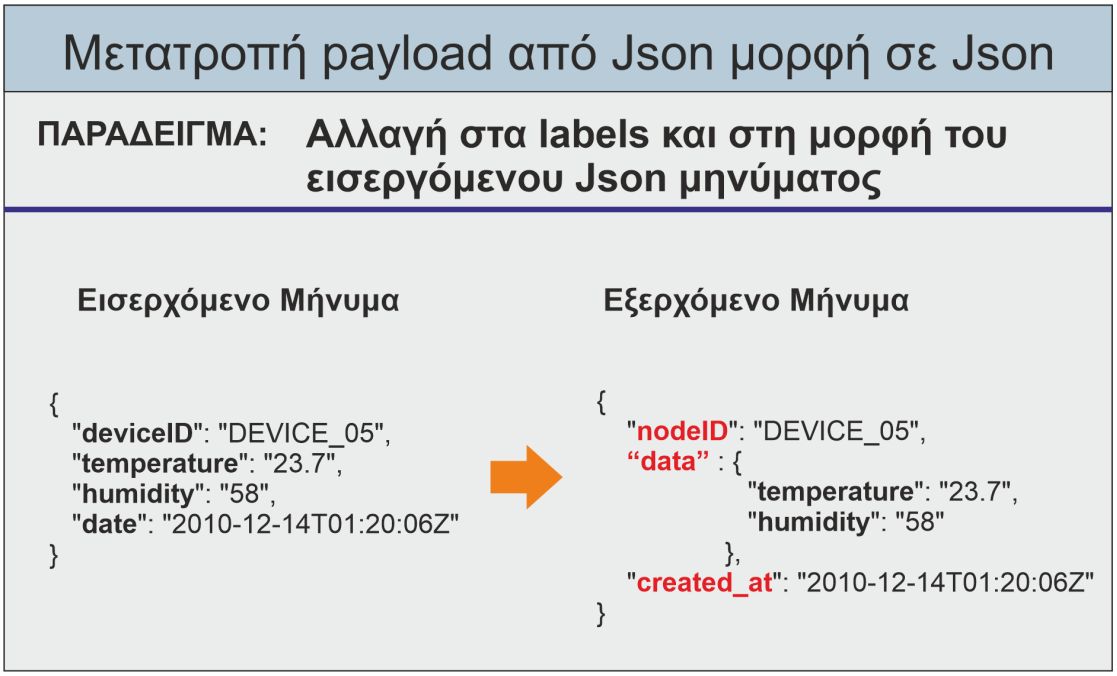
Εικόνα 5.33: Αλλάζοντας της κατάσταση του switch βλέπουμε στο terminal του broker2 (Adafruit.io) το εισερχόμενο μήνυμα καθώς και τη μετατροπή του στη μορφή που θέλουμε να αποσταλεί στη συσκευή.

Στο δεύτερο terminal του broker1 στο οποίο είναι συνδεδεμένη η συσκευή μας βλέπουμε την αναφορά αποστολής του μηνύματος στη συσκευή μας.

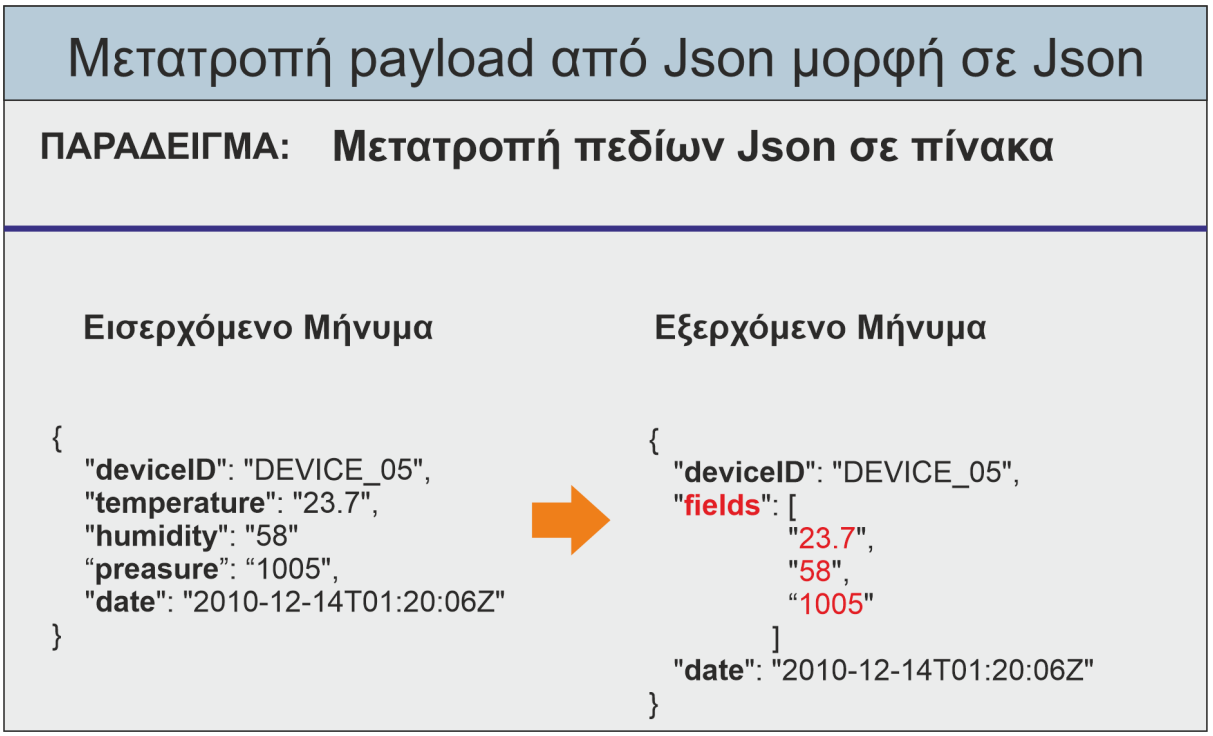
Ακολουθεί μια σειρά εικόνων με τις περισσότερες των περιπτώσεων μετατροπής μηνυμάτων.



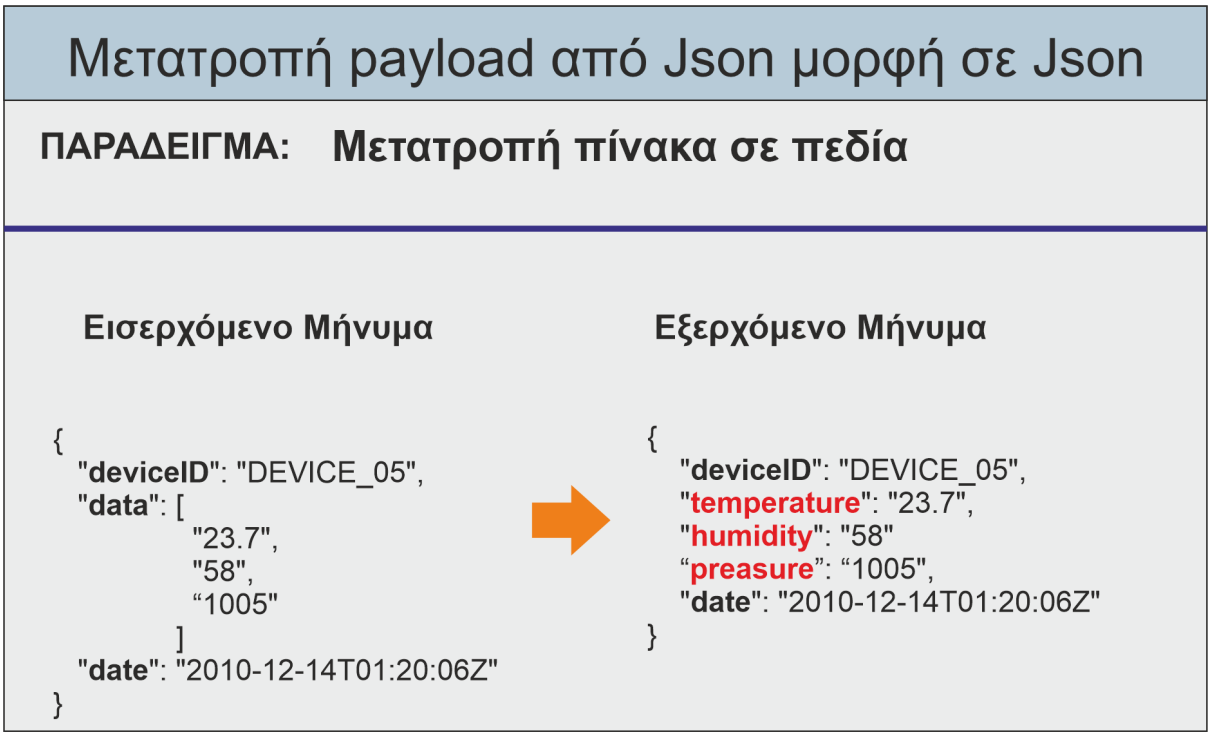
Εικόνα 5.34 Παράδειγμα 1 - Αλλαγή στα labels του Json



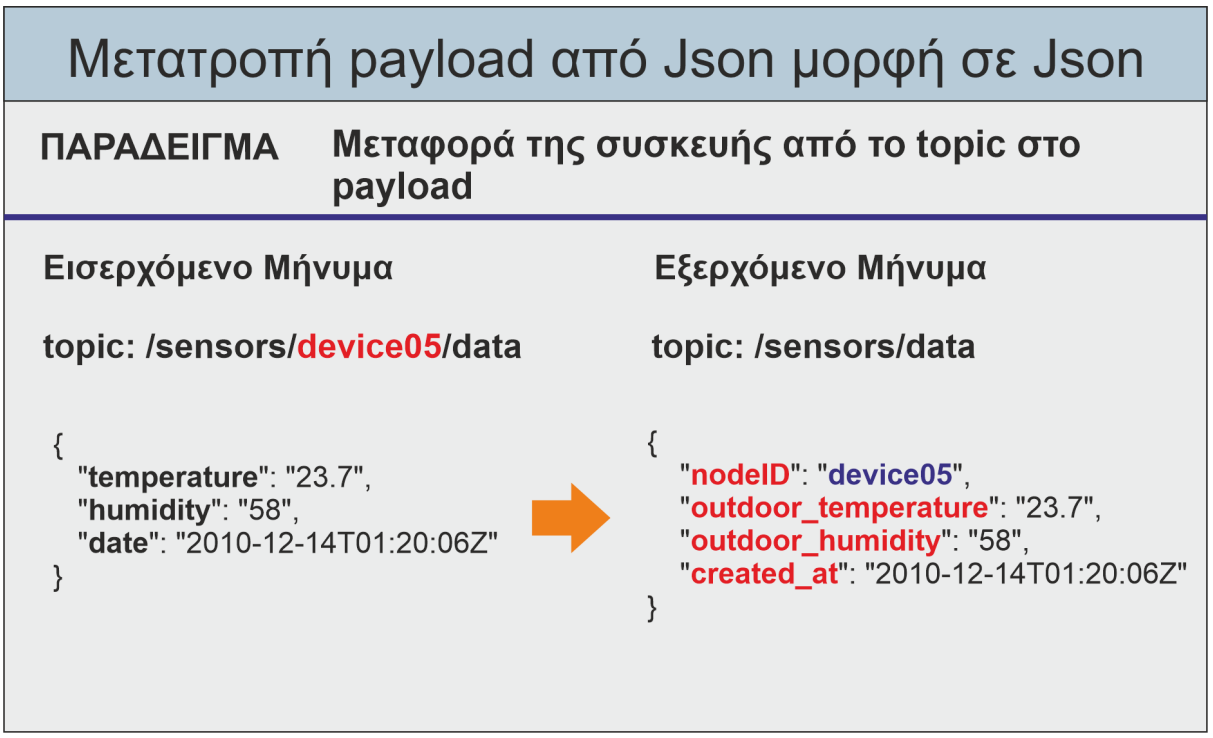
Εικόνα 5.35 Παράδειγμα 2 - Json μέσα σε Json



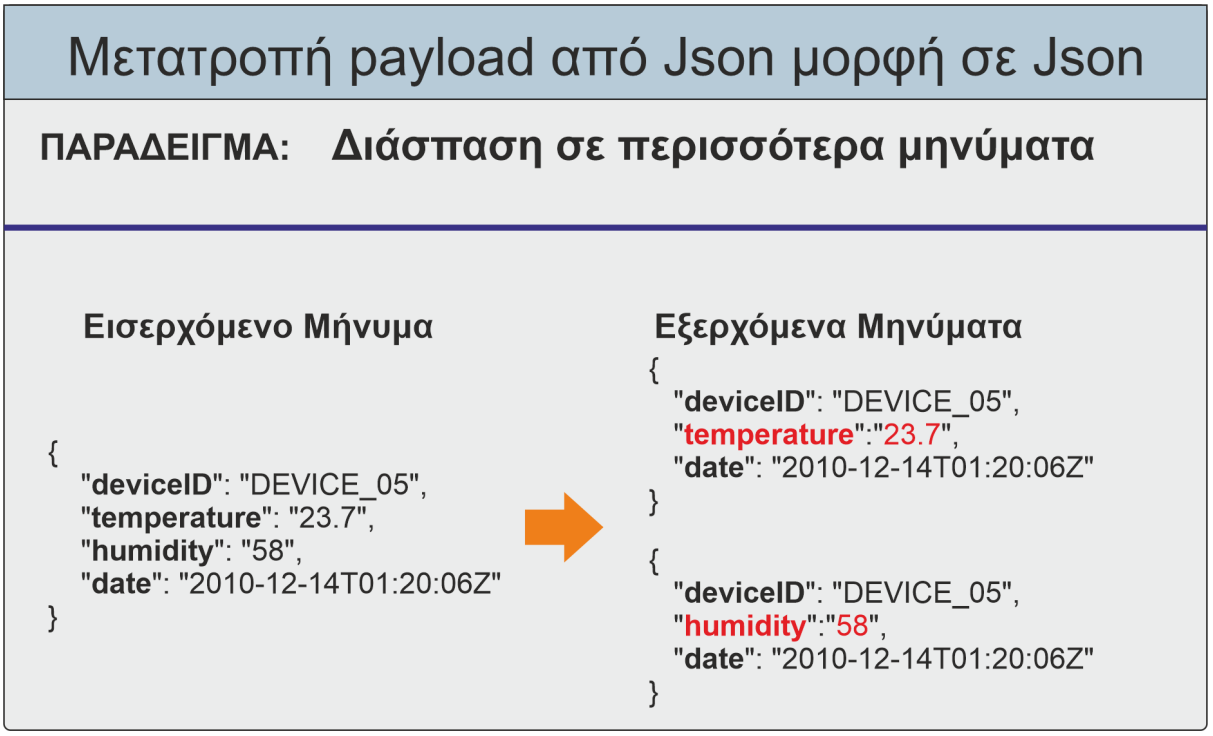
Εικόνα 5.36 Παράδειγμα 3 - Δημιουργία πίνακα



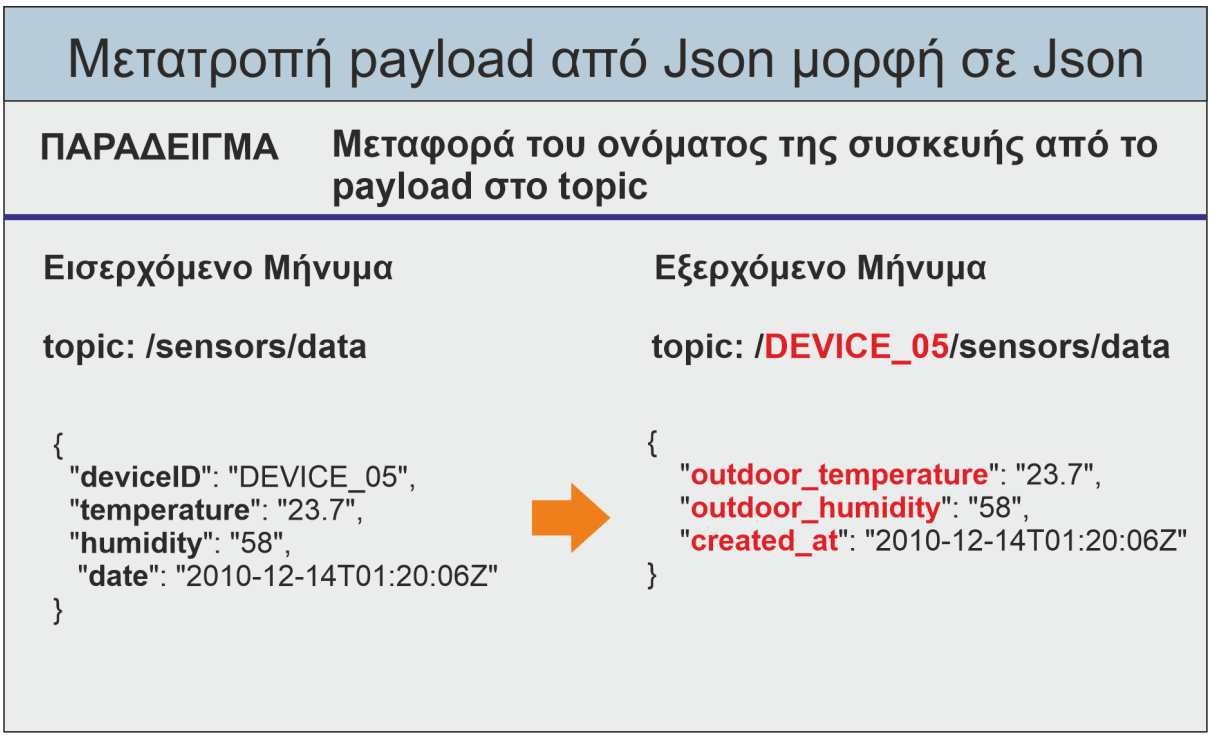
Εικόνα 5.37 Παράδειγμα 4 - Εισαγωγή πίνακα τιμών σε Json



Εικόνα 5.38 Παράδειγμα 5 - Μεταφορά αναγνωριστικού συσκευής από το topic του εισερχομένου μηνύματος στο payload του εξερχομένου



Εικόνα 5.39 Παράδειγμα 6 - Διάσπαση Json



Εικόνα 5.40 Παράδειγμα 5 - Μεταφορά αναγνωριστικού συσκευής από το payload του εισερχομένου μηνύματος στο topic του εξερχομένου

5.3 ΜΕΤΑΦΟΡΑ ΕΝΟΣ ΕΡΓΟΥ ΕΝΕΡΓΕΙΑΚΗΣ ΔΙΑΧΕΙΡΙΣΗΣ ΚΤΙΡΙΟΥ ΣΤΗΝ ΠΛΑΤΦΟΡΜΑ ADAFRUIT.IO

Τέλος θα δημιουργήσουμε ένα απλό dashboard στην πλατφόρμα adafruit.io και θα κάνουμε μια ολοκληρωμένη από εκεί διαχείριση του έργου μας μέσω της gateway .

Δημιουργούμε ένα dashboard μέσα από το οποίο θα παρακολουθούμε τις τιμές των

- Humidity
- Temperature
- CoolantTemp

Ενώ θα μπορούμε να παρακολουθούμε και να ελέγχουμε την κατάσταση των

- OnOff
- HiLow

Όλες οι τιμές θα στέλνονται σε ένα group με όνομα Device_1.

```
{
        "NodeID":"00:11:22:33:44:55",
        "Type":"ControlSensor/Sensor",
        "Humidity":52,
        "Temperature":13,
        "CoolantTemp":24,
        "OnOff":"false",
        "HiLow":"true",
        "msgCount":6
}
```

Οι τιμές της θερμοκρασίας και της υγρασίας θα εμφανίζονται μέσω δύο “Gauge widgets”, ενώ θα υπάρχουν στο dashboard και δύο “chart widgets” τα οποία θα εμφανίζουν τις τιμές του τελευταίου 24ώρου.

Οι τιμές των OnOff και HiLow θα εμφανίζονται μέ δύο “Led widgets” αλλά θα υπάρχουν και δύο “Switch widgets” για να γίνεται ο χειρισμός μέσα από την πλατφόρμα.

Για να μπορεί να διαβάσει η Gateway τις τιμές των διακοπτών κάθε φορά που ο χρήστης της πλατφόρμας αλλάζει την κατάστασή του θα πρέπει να κάνει subscribe στο topic:
iliaslamprou/groups/**device_1**/json

Τα μηνύματα που θα λαμβάνει θα είναι της μορφής:

{"feeds":{"hi-low":"true/false"}}
ή
{"feeds":{"on-off":"true/false"}}
αναλόγως τον διακόπτη του οποίου αλλάζει η κατάσταση.

Στη συσκευή θα πρέπει κάθε φορά να στέλνεται ένα μήνυμα της μορφής:

```
{  "NodeNo": "2",
   "OnOff": "true/false"
}
```

ή

```
{  "NodeNo": "2",
   "HiLow": "true/false"
}
```

Εδώ έχουμε μία περίπτωση που δεν περιλαμβάνεται στα παρακάτω παραδείγματα.

Η πλατφόρμα κάθε φορά που αλλάζουμε την κατάσταση σε κάποιον από τους διακόπτες στέλνει στο ίδιο topic δύο Json μηνύματα τα οποία διαφέρουν σε ένα πεδίο. Η Gateway θα πρέπει να ξεχωρίσει τα μηνύματα με βάση το πεδίο αυτό και να στείλει δύο διαφορετικά Json μηνύματα στη στη συσκευή.

Για να γίνει αυτό διαμορφώνουμε το αρχείο JsonConverter όπως παρακάτω:

```
{
    "broker1": [
        {       "subTopic": "sensor/+/data",
                "pubTopic": "iliaslamprou/groups/${topic=sendor/*/data}/json",
                "pubSettings":{
                        "qos": 0,
                        "retained" :1
                },
                "pubData":{
                        "feeds": {
                                "humidity": "${Humidity}",
                                "temperature": "${Temperature}",
                                "coolant_temp": "${CoolantTemp}",
                                "on_off": "${OnOff}",
                                "hi_low": "${HiLow}"
                        }
                }

        }
    ],
```

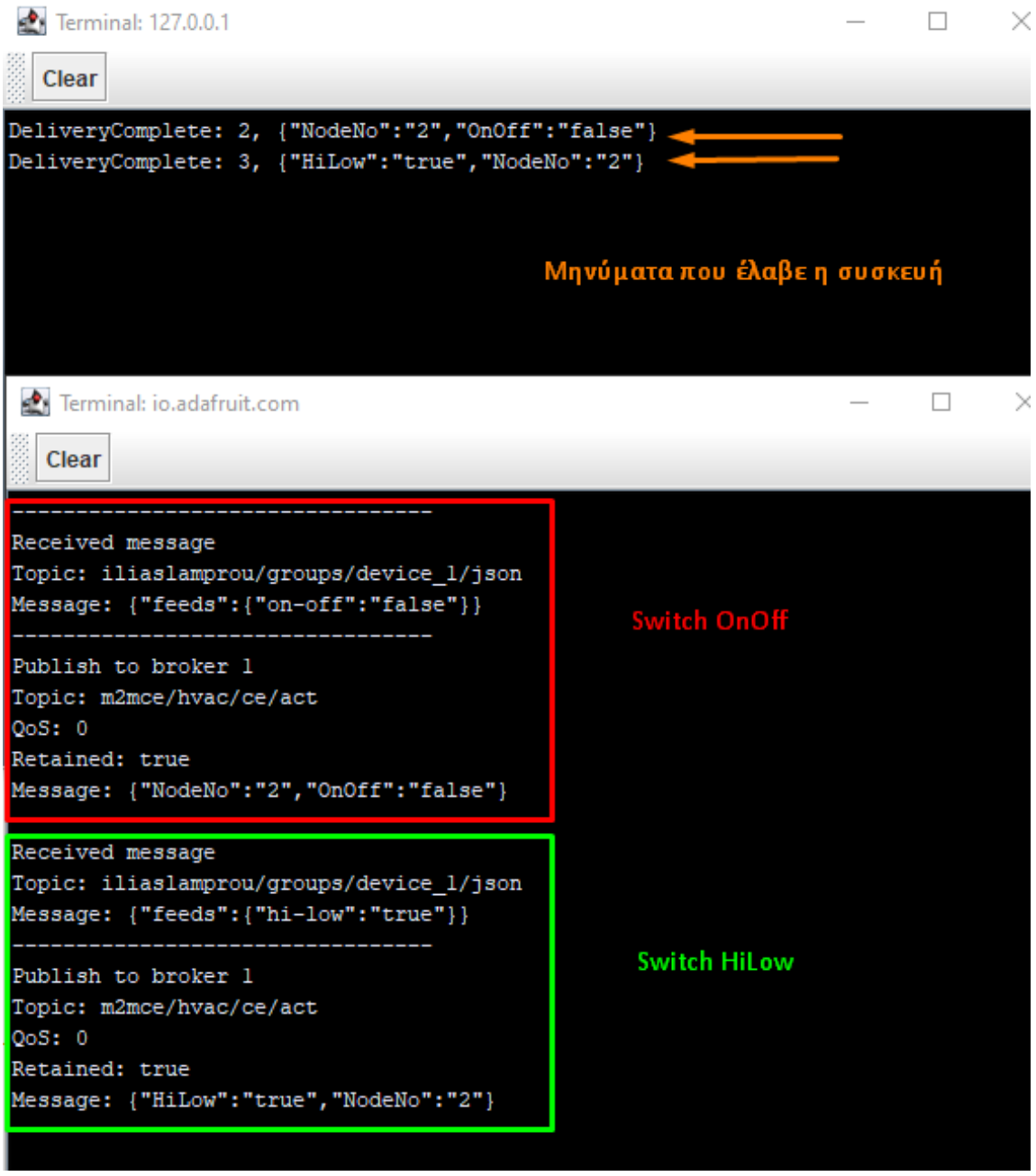
```
"broker2": [
        {       "subTopic": "iliaslamprou/groups/device_1/json",
                "pubTopic": "m2mce/hvac/ce/act",
                "pubSettings":{
                        "qos": 0,
                        "retained" :1
                },
                "pubData":{
                         "NodeNo": "2",
                         "OnOff": "${feeds/on-off}"
                        }


        },
        {       "subTopic": "iliaslamprou/groups/device_1/json",
                "pubTopic": "m2mce/hvac/ce/act",
                "pubSettings":{
                        "qos": 0,
                        "retained" :1
                },
                "pubData":{
                         "NodeNo": "2",
                         "HiLow": "${feeds/hi-low}"
                        }


        }
]

}
```

Στο αρχείο JsonConverter.txt στο JsonArray broker2 έχουμε εισάγει δυο στοιχεία τα οποία περιέχουν τις ρυθμίσεις για την τροποποίηση των μηνυμάτων των δύο διακοπτών.

Παρατηρούμε ότι το topic στο οποίο θα κάνει subscribe η Gateway είναι ίδιο.

Η Gateway κάνει μία μόνο φορά subscribe τα διπλά topics αλλά μπορεί και λαμβάνει κανονικά όλα τα διαφορετικά μηνύματα που ορίζονται στο JsonArray broker2.

Δοκιμάζοντας τη λειτουργία των διακοπτών βλέπουμε ότι η Gateway λειτουργεί κανονικά αποστέλλοντας διαφορετικά μηνύματα στη συσκευή κάθε φορά που ενεργοποιούμε διαφορετικό διακόπτη.



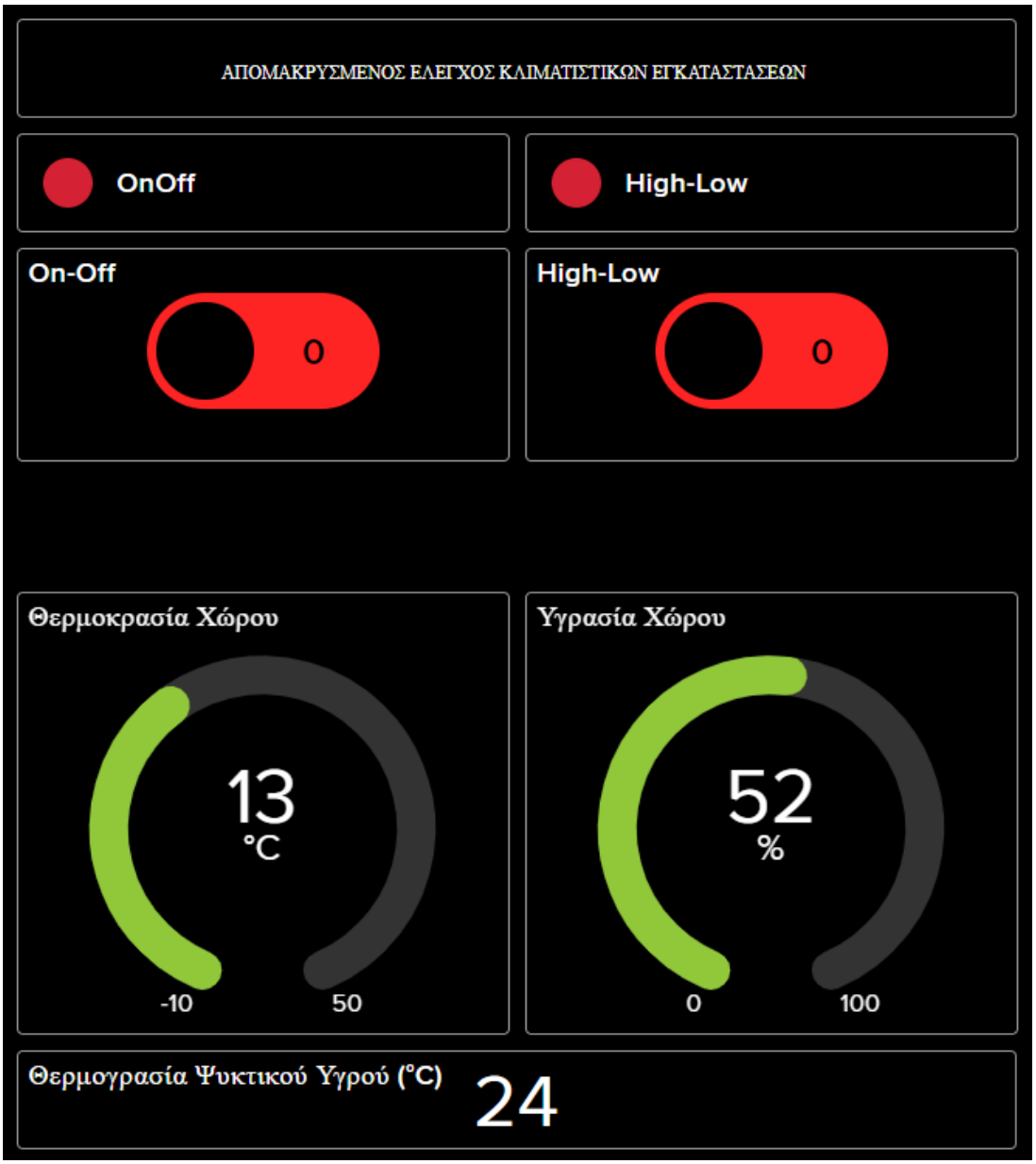
Εικόνα 5.41: Η εικόνα των τερματικών μετά την αλλαγή κατάστασης των διακοπτών.

Στην εικόνα 5.41 βλέπουμε το αποτέλεσμα της αλλαγής κατάστασης των διακοπτών στα δύο τερματικά.

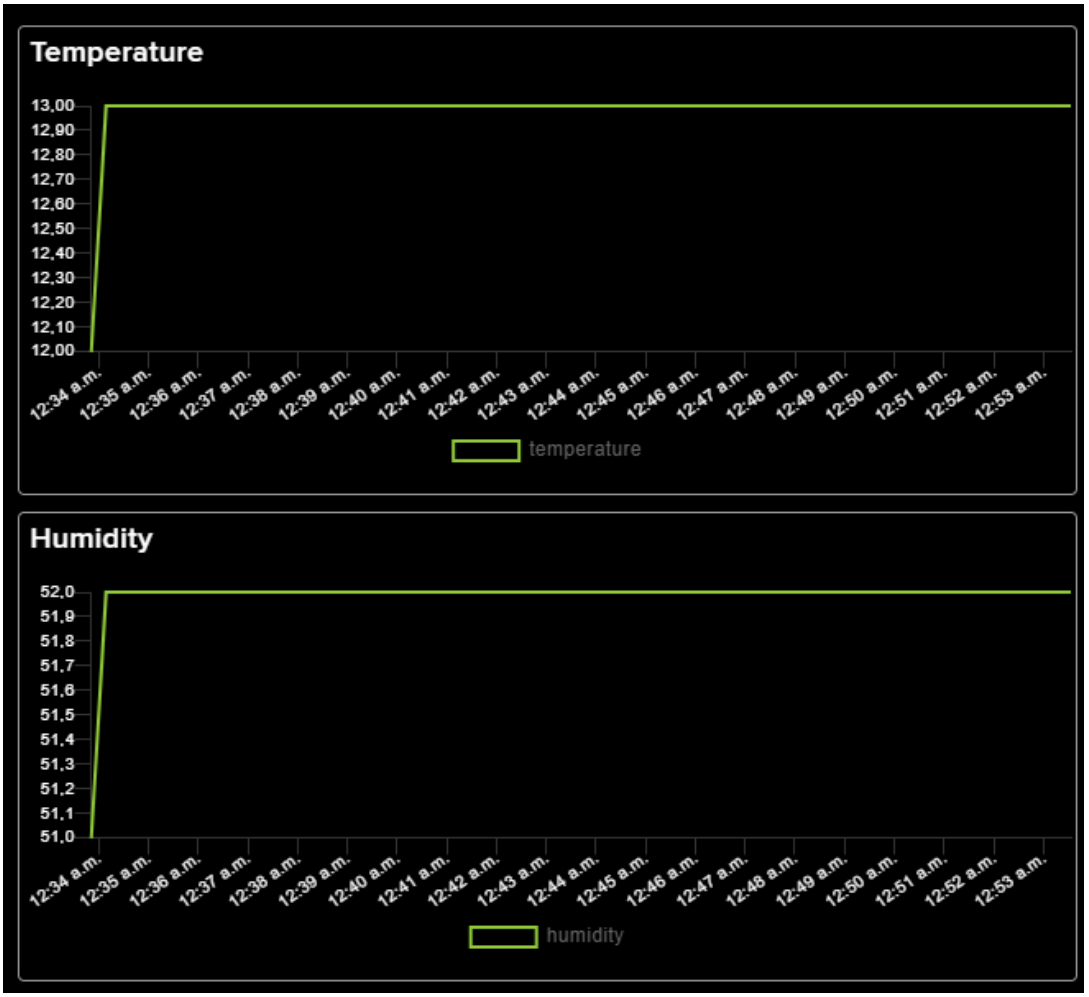
Στο τερματικό της πλατφόρμας adafruit.io μέσα στο κόκκινο πλαίσιο βλέπουμε κάτω από τη γραμμή "Receive message" το topic στο οποίο στέλνεται το μήνυμα αλλαγής κατάστασης του διακόπτη OnOff καθώς και το ίδιο το μήνυμα. Βλέπουμε δηλαδή το μήνυμα που στέλνει η πλατφόρμα στην Gateway.

Αμέσως μετά κάτω από τη γραμμή "Publish to broker1" βλέπουμε το μήνυμα που στέλνει η Gateway στον broker που είναι συνδεδεμένη η συσκευή. Το μήνυμα αυτό εμπεριέχει την τιμή true/false του εισερχομένου μηνύματος αλλά έχει τροποποιηθεί με βάση το αρχείο JsonConverter.txt

Στο πράσινο πλαίσιο βλέπουμε τα αντίστοιχα μηνύματα που λαμβάνει και στέλνει η Gateway κάθε φορά που αλλάζει η κατάσταση του διακόπτη HiLow.

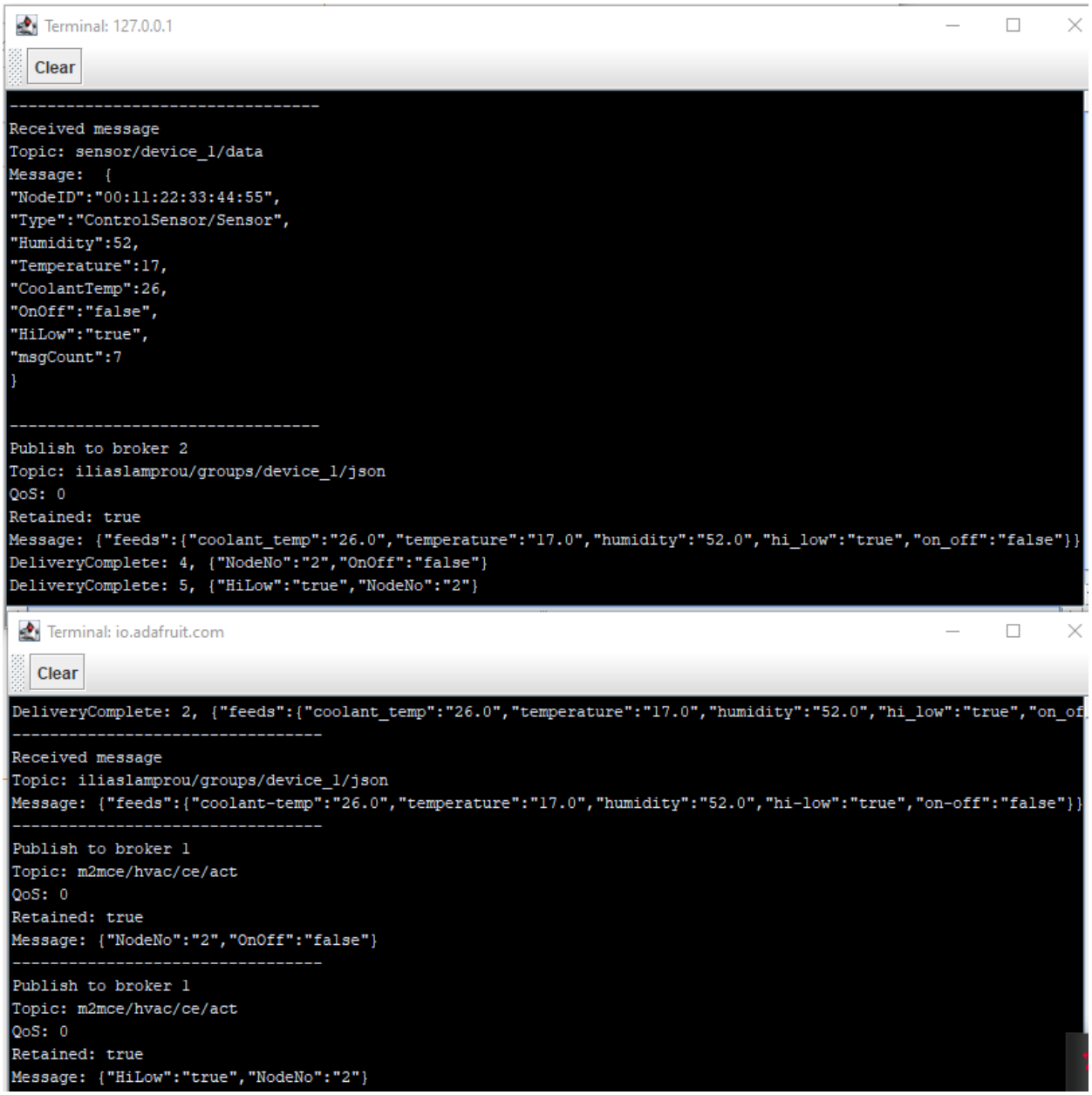


Εικόνα 5.42: Το dashboard για το έργο απομακρυσμένης διαχείρισης κλιματιστικών εγκαταστάσεων.



Εικόνα 5.43: Τα charts για την θερμοκρασία και την υγρασία χώρου.

Τέλος δοκιμάζουμε και την αποστολή δεδομένων από τη συσκευή προς την πλατφόρμα κάτι που το είχαμε κάνει και παραπάνω στέλνοντας όλα τα δεδομένα σε ένα group με το όνομα της συσκευής μας.



Εικόνα 5.44: Στην εικόνα βλέπουμε το αποτέλεσμα της αποστολής δεδομένων από τη συσκευή προς την πλατφόρμα. Η Gateway παραλαμβάνει το μήνυμα, το τροποποιεί με βάση τις ρυθμίσεις του αρχείου JsonConverter.txt και το στέλνει στην πλατφόρμα.

Η πλατφόρμα το λαμβάνει και απαντά στη Gateway που έχει κάνει subscribe στα topics των διακοπτών.

ΚΕΦΑΛΑΙΟ 6

ΣΥΜΠΕΡΑΣΜΑΤΑ

Μέσα από αυτή τη διπλωματική εργασία είδαμε τα βασικά χαρακτηριστικά και την αρχιτεκτονική ορισμένων ενδεικτικών πλατφορμών IoT όπως η πλατφόρμα Thingspeak, η πλατφόρμα Thinger.io, η πλατφόρμα Thingsboard, η πλατφόρμα Adafruit, καθώς επίσης και τον broker Mosquitto.

Εκτός από την πλατφόρμα Thingspeak στην οποία μείναμε μόνο στα βασικά χαρακτηριστικά και την αρχιτεκτονική της, σε όλες τις άλλες κάναμε και δοκιμές με πραγματικά έργα IoT ώστε να εξερευνήσουμε σε βάθος τον τρόπο λειτουργίας τους, τις δυνατότητες που πραγματικά παρέχουν, αλλά και την ευκολία με την οποία μπορεί να συνδεθεί ένα υπάρχων έργο IoT σε αυτές.

Παρόλο που οι πλατφόρμες αυτές υποστήριζαν διάφορα πρωτόκολλα επικοινωνίας ασχοληθήκαμε με το πρωτόκολλο MQTT το οποίο το υποστήριζαν όλες οι πλατφόρμες.

Η βασική παρατήρηση η οποία όμως προέκυψε μετά τη λειτουργία και δοκιμή των πλατφορμών πάνω σε ένα πραγματικό project ήταν ότι ναι μεν υποστηρίζεται το πρωτόκολλο MQTT, από την άλλη δεν υπάρχει μια κοινή γραμμή στον τρόπο διαμόρφωσης των Topics και Payloads αλλά κάθε πλατφόρμα έχει υιοθετήσει ένα ξεχωριστό τρόπο στην διαμόρφωση μηνυμάτων MQTT.

Για παράδειγμα ενώ ο Broker Mosquitto μας επιτρέπει να εισάγουμε σχεδόν οποιασδήποτε μορφής topics, από την άλλη αυτό δεν ίσχυε στην πλατφόρμα Adafruit.io της οποίας ο broker μας υποχρέωνε το topic να ξεκινά με συγκεκριμένη μορφή. Το ίδιο συμβαίνει και με την πλατφόρμα Thingspeak της οποίας τα topics πρέπει να περιλαμβάνουν το κανάλι (channel ID), ενώ το payload πρέπει να περιλαμβάνει το πεδίο "field". [14]

Παρόλο που έχουμε ένα κοινό πρωτόκολλο παρατηρήσαμε ότι αν θελήσουμε μια συσκευή να επικοινωνήσει με διαφορετικές πλατφόρμες, αυτό είναι αδύνατο λόγω της διαφοροποίησης διαμόρφωσης των μηνυμάτων.

Ενώ σύμφωνα με το πρωτόκολλο MQTT μια απλή αλλαγή των παραμέτρων σύνδεσης θα αρκούσε για να μεταβεί μια συσκευή από ένα broker σε άλλον, αυτό δεν αρκεί και θα χρειαστεί και αλλαγή του κώδικα μέσα στη συσκευή ώστε να επιτευχθεί μια μεταφορά ενός υπάρχοντος έργου. Αν μία επιχείρηση διαθέτει μεγάλο πλήθος συσκευών αυτό θα απαιτούσε μεγάλο χρηματικό κόστος αλλά και κόστος σε χρόνο.

Λόγω αυτής της διαφοροποίησης ο χρήστης μιας πλατφόρμας δεσμεύεται να δημιουργήσει και να προσαρμόσει τα έργα του ώστε να είναι συμβατά σε μια συγκεκριμένη πλατφόρμα. Αν στο μέλλον χρειαζόταν να αλλάξει πλατφόρμα αυτό θα σήμαινε αυτόματα

και επαναπρογραμματισμό όλων των συσκευών και σχεδίασει από την αρχή του IoT έργου του.

Η δεύτερη παρατήρηση έχει να κάνει με τα Rule Engines. Η επεξεργασία **on the cloud** είναι μια πολύ σημαντική δυνατότητα των πλατφορμών IoT. Παρατηρήσαμε ότι όλες οι πλατφόρμες (εκτός τον Mosquitto broker που δεν είναι πλατφόρμα), έχουν εισάγει αυτή τη δυνατότητα στα χαρακτηριστικά τους.

Και εδώ όμως παρατηρήσαμε σημαντικές διαφορές στον τρόπο που υλοποιεί η κάθε πλατφόρμα αυτή τη δυνατότητα.

Είδαμε για παράδειγμα ότι η πλατφόρμα Thingspeak επιτρέπει την εισαγωγή κώδικα Matlab μέσα από τον οποίο μπορούν οι χρήστες της πλατφόρμας να κάνουν υπολογισμούς τιμών, αλλά και να δημιουργήσουν διαγράμματα με στατιστικά αποτελέσματα.

Από την άλλη οι πλατφόρμες Thinger.io και Thingsboard δεν υποστηρίζουν αυτή τη λειτουργία αλλά καλύπτουν την επεξεργασία των δεδομένων εισάγοντας στις πλατφόρμες τους εφαρμογές τρίτων οι οποίες όμως δεν είναι συμβατές μεταξύ τους.

Ακόμη και στην περίπτωση του Node-Red [16] το οποίο υποστηρίζεται και από τις δύο πλατφόρμες πάλι έχει διαφοροποιήσεις γιατί κάθε πλατφόρμα προσπαθεί να κάνει προσαρμογή στα μέτρα της με αποτέλεσμα τα έργα που θα δημιουργηθούν να μην είναι συμβατά μεταξύ τους.

Αυτό ενισχύει ακόμη περισσότερο αυτό που αναφέραμε παραπάνω στην πρώτη παρατήρηση. Ο χρήστης δηλαδή δεσμεύεται σε μία πλατφόρμα και δεν μπορεί εύκολα να μεταφέρει το έργο του σε άλλη πλατφόρμα.

Ενώ το πρώτο μέρος λοιπόν της παρούσας διπλωματικής βοήθησε στην εξαγωγή του παραπάνω συμπεράσματος το δεύτερο μισό αφορούσε τη λύση που θα μπορούσαμε να δώσουμε σε αυτό το πρόβλημα.

Η λύση που προτείνεται είναι η χρήση μίας IoT Gateway μεταξύ των πρωτοκόλλων MQTT η οποία θα μπορεί να μεταφέρει ένα έργο IoT από μια πλατφόρμα σε μια άλλη χωρίς να χρειαστεί σε ορισμένες περιπτώσεις καμία αλλαγή στον κώδικα των συσκευών ένώ σε κάποιες άλλες ίσως χρειαστεί αλλαγή μόνο των παραμέτρων σύνδεσης.

Είδαμε ότι η λειτουργία MQTT to MQTT έχει ήδη υιοθετηθεί από την IoT Gateway της πλατφόρμας Thingsboard πράγμα που δείχνει πόσο σημαντική είναι και επιβεβαιώνει τα παραπάνω συμπεράσματα.

Κάθε μέρα εισέρχονται στον κόσμο του IoT νέες πλατφόρμες, αυτό σημαίνει ότι θα έχουμε και προσφορά οικονομικότερων λύσεων αλλά και περισσοτέρων δυνατοτήτων. Κάθε επιχείρηση που ξεκινά ένα IoT έργο θα πρέπει εξ' αρχής να γνωρίζει ότι στο μέλλον ίσως θα χρειαστεί να μεταφερθεί το έργο της σε άλλη πλατφόρμα και να το αναπτύσει λαμβάνοντας σοβαρά υπόψιν αυτή τη παράμετρο.

Η λύση μιας IoT Gateway MQTT to MQTT είναι αυτή αυτή που φαίνεται να λύνει αυτό το πρόβλημα, αλλά για να γίνει αυτό ακόμη πιο εύκολα θα πρέπει να εισαχθούν στο IoT έργο και οι δυνατότητες του edge computing. Δηλαδή να μην υπάρχει απευθείας σύνδεση των συσκευών με την πλατφόρμα αλλά να υπάρχει ένα τοπικός broker ο οποίος θα συνδέεται από τη μια πλευρά με την πλατφόρμα και από την άλλη με το πλήθος των συσκευών IoT.

Τα δεδομένα θα περνούν μέσα από τον broker και στη συνέχεια θα στέλνονται στην πλατφόρμα. Με αυτόν τον τρόπο η προσθήκη μιας Gateway είναι σχετικά εύκολη υπόθεση.

Αυτό το είδαμε στο κεφάλαιο 6 όπου αναλύσαμε την IoT Gateway της πλατφόρμας ThingsBoard αλλά και προσαρμόσαμε πάνω σε αυτή ένα υπάρχων έργο IoT.

Είδαμε ότι η IoT Gateway έχει κατασκευαστεί έτσι ώστε να υποστηρίζει μεγάλη ποικιλία προσαρμογών μεταξύ των topics και payloads πράγμα που μας βοήθησε στο να καταφέρουμε να προσαρμόσουμε το έργο μας στη συγκεκριμένη πλατφόρμα.

Ένα άλλο που παρατηρήσαμε εδώ είναι η δυνατότητα της IoT Gateway να συνδέεται απευθείας με συσκευές και μάλιστα διαφορετικών πρωτοκόλλων. Αυτό θα μπορούσε να κάνει περιττό τον ενδιάμεσο broker και να χρησιμοποιηθεί η ίδια η IoT Gateway ώς κοινός κόμβος για τις συσκευές και μεσολαβητής με την πλατφόρμα.

Σε αυτή την περίπτωση η αλλαγή πλατφόρμας θα μπορούσε να γίνει αλλάζοντας κάθε φορά την IoT Gateway με την αντίστοιχη που διαθέτει η κάθε πλατφόρμα.

Αρκούν όμως μόνο οι παραπάνω προϋποθέσεις για την μεταφορά ενός έργου από μια πλατφόρμα σε μια άλλη;

Η απάντηση είναι όχι. Είδαμε παραπάνω ότι ένα σημαντικό κομμάτι πλέον του IoT είναι η επεξεργασία των δεδομένων στην πλατφόρμα. Η απλή μεταφορά του έργου δεν μας καλύπτει αφού η νέα πλατφόρμα δεν θα μπορούσε να κάνει την επεξεργασία και θα απαιτούσε εκ νέου προγραμματισμό.

Η λύση στο πρόβλημα είναι η τοπική επεξεργασία. Δηλαδή τα δεδομένα πρώτα να συλλέγονται σε ένα τοπικό κόμβο και στη συνέχεια μέσω της IoT Gateway να αποστέλλονται στην πλατφόρμα η οποία θα παίζει τώρα το ρόλο μόνο της διανομής και αποθήκευσης.

Διαθέτουν όμως όλες οι IoT πλατφόρμες Gateways;

Όπως είδαμε από τις πλατφόρμες που αναλύσαμε μόνο μία μας δίνει αυτή τη δυνατότητα.

Το σημαντικότερο κομμάτι αυτής της διπλωματικής εργασίας ήταν να προτείνει ένα τρόπο ανάπτυξης μιας IoT Gateway η οποία να μην είναι προσαρμοσμένη σε μια συγκεκριμένη IoT πλατφόρμα, αλλά να μπορεί να προσαρμοστεί σε οποιαδήποτε υποστηρίζει το πρωτόκολλο MQTT.

Μετά την κατασκευή της IoT Gateway έγιναν δοκιμές σε Mosquitto broker καθώς και στην IoT πλατφόρμα Adafruit.io. Οι δοκιμές είχαν μεγάλη επιτυχία και έγινε εφικτή η μεταφορά ενός έτοιμου IoT έργου στην πλατφόρμα Adafruit.io μέσα σε μικρό χρόνο.

Η τεχνική που χρησιμοποιήθηκε στη δική μας IoT πλατφόρμα ήταν παρόμοια με αυτή της πλατφόρμας Thingsboard. Ο χρήστης δηλαδή έπρεπε απλά να παραμετροποιήσει ένα αρχείο Json.

Αυτό που παρατηρήσαμε ήταν ότι έπρεπε να προστεθούν πολύ περισσότερες περιπτώσεις στις μετατροπές των μηνυμάτων Json, ώστε να γίνει τελικά η νέα IoT Gateway ευέλικτη και ικανή να υποστηρίξει πλήθος από τις υπάρχουσες πλατφόρμες.

Η IoT Gateway που δημιουργήσαμε έχει τη δυνατότητα μεγαλύτερης ανάπτυξης και υποστήριξης και άλλων πρωτοκόλλων.

Μέχρι να θεσπιστεί ένας ενιαίος τρόπος στην μορφή των μηνυμάτων που θα ανταλλάσσονται μεταξύ των Brokers η IoT Gateway αποτελεί σίγουρα μια σημαντική λύση στο πρόβλημα μεταφορά έργων από πλατφόρμα σε πλατφόρμα.

ΠΑΡΑΡΤΗΜΑ Α

1. Η ΠΛΑΤΦΟΡΜΑ THINGSPEAK

Α. Γενικά

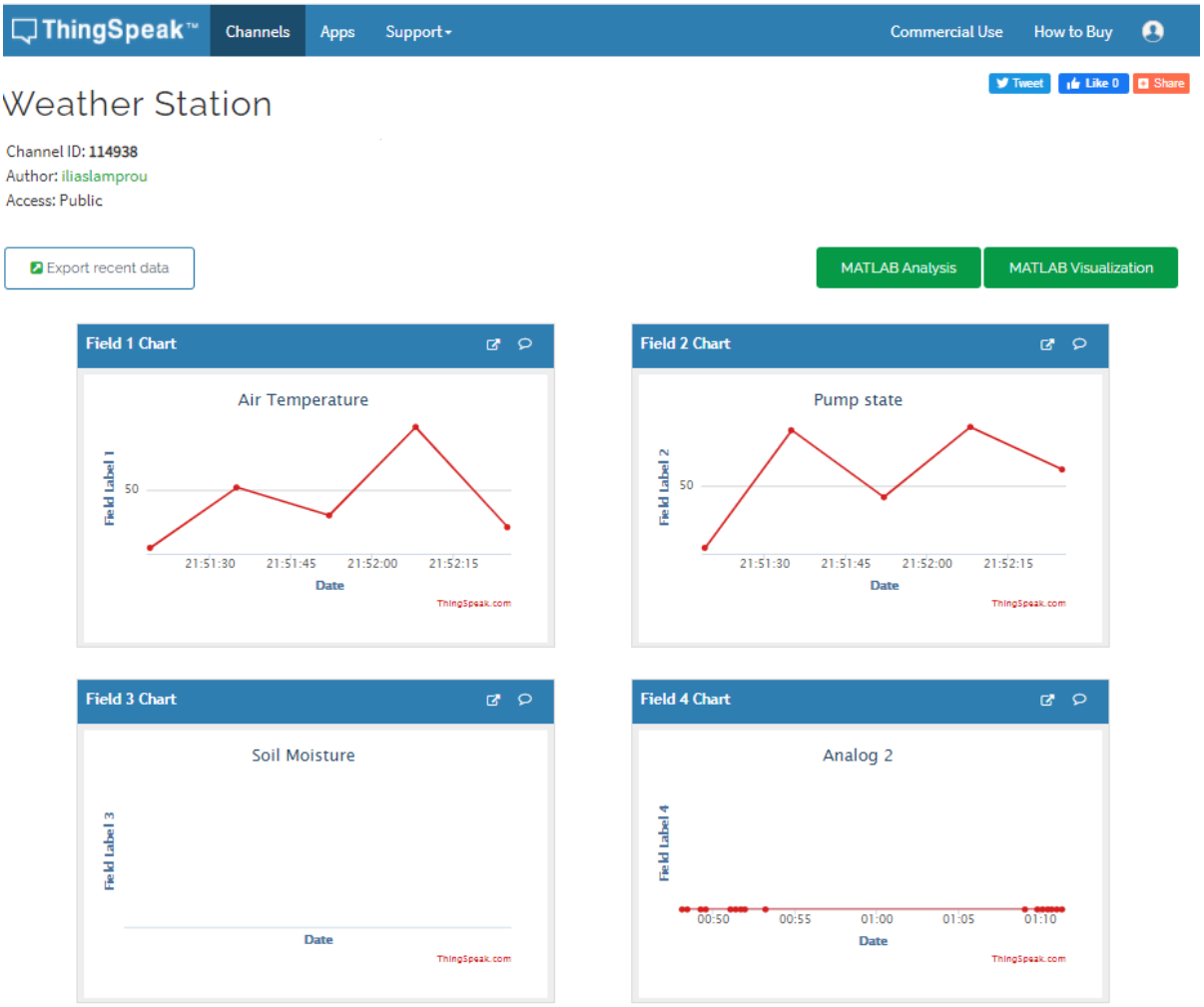
Πρώτη πλατφόρμα που θα αναλύσουμε ως προς τα χαρακτηριστικά της αλλά και θα δοκιμάσουμε τη λειτουργία της κατασκευάζοντας ένα απλό project είναι η Open Source πλατφόρμα IoT ThingSpeak της Mathworks (https://thingspeak.com/).

Επιτρέπει τη σύνδεση απεριόριστου πλήθους συσκευών, ο αριθμός των οποίων περιορίζεται από το τιμολογιακό πακέτο.

Ο τρόπος που διαχειρίζεται τις συσκευές διαφέρει σημαντικά από τις άλλες πλατφόρμες. Κάθε συσκευή θεωρείται ως ένα κανάλι αποστολής δεδομένων (channel) και μπορεί να στείλει μέχρι οκτώ τιμές από αισθητήρες.

Κάθε channel έχει το δικό του ξεχωριστό dashboard.

Για να εξηγήσουμε καλύτερα τα παρακάτω θα αναφέρουμε ένα παράδειγμα. Έστω ότι έχουμε μια συσκευή που κάνει κάποιες μετρήσεις για τον καιρό και θέλει να στείλει τα δεδομένα από τέσσερις αισθητήρες στην πλατφόρμα. Θα πρέπει να δημιουργήσουμε στην πλατφόρμα ένα κανάλι που αντιστοιχεί στη συσκευή μας με πέντε (fields) που αντιστοιχούν στους αισθητήρες μας. Ο κάθε αισθητήρας θα στέλνει τις τιμές σε ένα ξεχωριστό field. Οι τιμές αυτές καταγράφονται και μπορούν να εμφανιστούν σε ένα Dashboard είτε ώς τιμές είτε ως γραφήματα με καθοριζόμενο από τον χρήστη βάθος χρόνου.



Εικόνα Α.1.1 Το dashboard της πλατφόρμας Thingspeak

Το κάθε κανάλι μπορεί να είναι είτε ιδιωτικό είτε δημόσιο. Στο δημόσιο μπορεί να έχει πρόσβαση οποιοσδήποτε έχει το link του καναλιού ενώ για το ιδιωτικό απαιτείται είτε σύνδεση είτε η χρήση κλειδιού που μπορεί να παραχθεί από την πλατφόρμα.

Μεγάλο πλεονέκτημα αποτελεί το γεγονός ότι ο διαχειριστής μπορεί να ορίσει να εμφανίζονται διαφορετικοί αισθητήρες στο δημόσιο από ότι στο ιδιωτικό Dashboard.

Μειονέκτημα είναι το γεγονός ότι δεν επιτρέπει στο ίδιο πάνελ να εμφανίζονται τιμές από διαφορετικές συσκευές. Κάθε συσκευή έχει και ένα ξεχωριστό πάνελ καθώς και link για πρόσβαση σε αυτό.

Άλλο μειονέκτημα είναι η έλλειψη πολλών έτοιμων widgets αλλά και η έλλειψη ελευθερίας σχεδίασης του πάνελ. Δίνει όμως τη δυνατότητα μέσω ενσωματωμένης εφαρμογή να κατασκευάσει ο χρήστης δικά του widgets με κώδικα JavaScript.

Το δυνατό όμως σημείο της πλατφόρμας είναι η δυνατότητα ανάλυσης των δεδομένων μέσω κώδικα Matlab που μπορεί να γράψουμε online μέσα από την εφαρμογή (editor) που μας παρέχεται για αυτό το λόγο.

Με τον κώδικα μπορούμε να εμφανίσουμε κάθε είδους γραφήματα μας επιτρέπει να δημιουργήσουμε η γλωσσα Matlab.

Στην εικόνα Α.1.2 φαίνεται ο κώδικας Matlab που δημιουργήσαμε μέσα από τον editor της πλατφόρμας. Ενώ στην εικόνα Α.1.3 βλέπουμε το αποτέλεσμα.

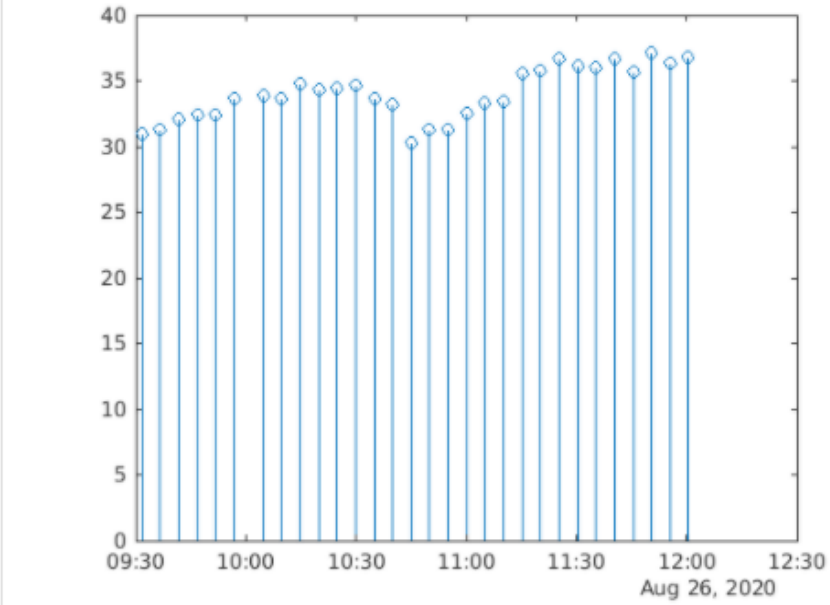
Η πλατφόρμα μας δίνει τη δυνατότητα σε αυτό το αποτέλεσμα να έχουμε πρόσβαση μέσω ενός δημοσίου link: (https://thingspeak.com/apps/matlab_visualizations/363019)

MATLAB Code

```
readChannelID = 114938;
fieldID = 1;

% Channel Read API Key
% If your channel is private, then enter the read API
% Key between the '' below:
readAPIKey = '';

%% Read Data %%

[data, time] = thingSpeakRead(readChannelID, 'Field', fieldID, 'NumPoints', 30,
    'ReadKey', readAPIKey);

%% Visualize Data %%

stem(time, data);
```

Εικόνα Α.1.2 Επεξεργασία δεδομένων με κώδικα Matlab στην πλατφόρμα Thingspeak

MATLAB Plot Output



Εικόνα Α.1.3 Δημιουργία γραφήματος μετά την επεξεργασία δεδομένων με κώδικα Matlab

Η πλατφόρμα, για το κάθε γράφημα που δημιουργούμε με τον κώδικα, μας δίνει την επιλογή να το εμφανίζουμε σε οποιοδήποτε dashboard συσκευής (channel). Έτσι μπορούμε να ξεπεράσουμε το πρόβλημα που αναφέραμε παραπάνω της μη προβολής αισθητήρων απο διαφορετικές συσκευές στο ίδιο πάνελ.

Αν κρίνουμε από την ευκολία με την οποία δημιουργούμε νέα γραφήματα μέσα από τον κώδικα κρίνουμε αυτή τη λειτουργία της πλατφόρμας εντυπωσιακή.

Συνολικά οι δυνατότητες που προσφέρονται με αυτή τη λειτουργία είναι:

- Μετατροπή, συνδυασμός και υπολογισμός νέων δεδομένων
- Δημιουργία προγραμμάτων για εκτέλεση σε συγκεκριμένες ώρες
- Οπτική κατανόηση των σχέσεων μεταξύ των δεδομένων χρησιμοποιώντας ενσωματωμένες συναρτήσεις
- Συνδυασμό δεδομένων από πολλά κανάλια για δημιουργία μιας πιο εξελιγμένης ανάλυσης

Στο θέμα της αποθήκευσης τιμών η πλατφόρμα επιτρέπει ανα κανάλι την αποστολή μηνυμάτων ανα 15 δευτερόλεπτα στο free mode και ανα ένα δευτερόλεπτο στα υπόλοιπα πακέτα. Αυτό σημαίνει πως αν κάποια συσκευή στείλει ένα μήνυμα με διαφορά χρόνου από το προηγούμενο μικρότερη από τις παραπάνω τιμές αυτό θα αγνοηθεί.

Δεν υπάρχει κάποιο φιλτράρισμα τιμών κατα την αποθήκευση αλλά μόνο κατα την εμφάνιση. Αυτό σημαίνει πως οι ίδιοι οι αισθητήρες θα πρέπει να φροντίσουν να μην στέλνουν εσφαλμένες και εντός ορίων τιμές. Δίνει όμως τη δυνατότητα μέσω κώδικα Matlab για διαγραφή αυτών των τιμών από τον server.

Πέρα από τις εφαρμογές που αναφέρθηκαν παραπάνω:

Matlab Analysis: Για μορφοποίηση τιμών

Matlab Visualisation: Για εμφάνιση γραφημάτων

Plugins για τη δημιουργία widgets με χρήση κώδικα JavaScript για τη βελτίωση της οπτικοποίησης.

η πλατφόρμα διαθέτει μερικές ακόμη χρήσιμες εφαρμογές η λειτουργία και η χρησιμότητα των οποίων αξίζει να διερευνηθούν.

ThingTweet : Είναι μια εφαρμογή που χρησιμοποιεί η πλατφόρμα για να στείλει ειδοποιήσεις στο χρήστη μέσω Twitter.

Υποστηρίζονται και άλλοι τρόποι ειδοποιήσεων όπως η αποστολή email αλλά θα χρειαστεί ο χρήστης να δημιουργήσει κώδικα μέσω της εφαρμογής Matlab Analysis.

TimeControl: επιτρέπει την αποστολή συμβάντων σε καθορισμένου χρόνου και λειτουργεί σαν χρονοδιακόπτης.

React: Το React λειτουργεί με τις εφαρμογές ThingHTTP, ThingTweet και MATLAB Analysis για την εκτέλεση ενεργειών όταν τα δεδομένα καναλιού πληρούν μια συγκεκριμένη προϋπόθεση. Για παράδειγμα, μπορούμε να έχουμε μια εφαρμογή για κινητά να αναφέρει το γεωγραφικό πλάτος και μήκος σας σε ένα κανάλι ThingSpeak. Όταν η θέση μας βρίσκεται σε

μια συγκεκριμένη απόσταση από το σπίτι σας, ζητάμε από το ThingHTTP να ανάψει τα φώτα του καθιστικού σας.

TalkBack: Επιτρέπει την εκτέλεση εντολών σε ουρά. Οι εντολές στέλνονται στην πλατφόρμα και μπαίνουν σε ουρά. Η συσκευή στέλνει αίτημα στη πλατφόρμα να διαβάσει την πρώτη που βρίσκεται στην ουρα εντολή και την εκτελή. Μετά το διάβασμα της εντολής αυτή διαγράφεται και είναι στη συσκευή διαθέσιμη η επόμενη. Με αυτόν τον τρόπο δεν χάνονται εντολές από εσφαλμένη επικοινωνία και είμαστε σίγουροι ότι θα εκτελεστούν με τη σωστή σειρά..

ThingsHttp: Επιτρέπει την επικοινωνία μεταξύ συσκευών, ιστότοπων και υπηρεσιών ιστού χωρίς να χρειάζεται να εφαρμοστεί το πρωτόκολλο σε επίπεδο συσκευής. Ο χρήστης μπορεί να καθορίζετε ενέργειες στο ThingHTTP, οι οποίες μπορούν να ενεργοποιηθούν χρησιμοποιώντας άλλες εφαρμογές ThingSpeak ™, όπως TweetControl, TimeControl και React.

Στο επικοινωνιακό κομμάτι υποστηρίζει επικοινωνία μέσω RestFull Api και MQTT. Το RestFull Api επιτρέπει μηνύματα HTTP τύπου GET, POST, PUT και DELETE για αποστολή, ανάγνωση ή διαγραφή τιμών στο server.

Διαθέτει πλούσιο Documentation και παραδείγματα για τον τρόπο επικοινωνίας. Χρησιμοποιεί μια εντελώς διαφορετική μορφή αποστολής μηνυμάτων η οποία δεν μπορεί να τροποποιηθεί. Αυτό σημαίνει ότι για να μεταφερθεί μια συσκευή από άλλη πλατφόρμα στο ThingSpeak θα πρέπει να τροποποιηθεί το firmware ώστε η μορφή των μηνυμάτων να είναι συμβατή με την πλατφόρμα. Το πρόβλημα είναι ότι και στη σύνδεση MQTT χρησιμοποιεί και πάλι εντελώς διαφορετική μορφή και καθίσταται έτσι δύσκολο σε ανεξάρτητες MQTT εφαρμογές για κινητά ή desktop PCs να μπορέσουν να παραμετροποιηθούν ώστε να υποστηρίξουν την επικοινωνία.

Όσον αφορά τη υποστήριξη των χρηστών με Android και Ios APPs η πλατφόρμα δεν παρέχει κάποιο App αλλά υπάρχουν πάρα πολλά αξιόλογα Apps τα οποία δημιουργήθηκαν από ανεξάρτητους χρήστες και υποστηρίζουν την επικοινωνία και την οπτικοποίηση της πλατφόρμας.

Μέσα από την ιστοσελίδα βρήκαμε επίσης ένα πλήθος από παραδείγματα για σύνδεση στη πλατφόρμα μέσω Arduino, Raspberry, ESP8266 και Google assistant με χρήση κώδικα C++ και Python. Πολύ σημαντικό είναι ακόμη το ότι περιέχεται και έτοιμη βιβλιοθήκη για Arduino που απλοποιεί τη σύνδεση αυτών των συσκευών.

Στο θέμα του χρόνου απόκρισης του dashboard στις ενημερώσεις τιμών από τις συσκευές παρουσιάστηκε μικρή καθυστέρηση μέχρι και τριών-τεσσάρων δευτερολέπτων ορισμένες φορές αλλά για αυτό το θέμα θα υπάρξει ειδική ανάλυση στον έλεγχο της λειτουργίας της πλατφόρμας με πραγματικές συσκευές.

ε. Ασφάλεια

Η πλατφόρμα υποστηρίζει ασφαλείς συνδέσεις μέσω SSL/TLS τόσο στην HTTP επικοινωνία, όσο και στην MQTT.

Η σύνδεση των συσκευών είναι πάρα πολύ απλή και δεν απαιτεί κάποια MAC address η αναγνωριστικό συσκευής.

Ο χρήστης μπορεί να δημιουργήσει μέσα από την πλατφόρμα δύο κλειδιά τα οποία είναι αλφαριθμητικά μήκους μήκους 16 χαρακτήρων. Αυτό θα γίνει για κάθε κανάλι ξεχωριστά αφού όπως αναφέρθηκε παραπάνω κάθε κανάλι αποτελεί και μια συσκευή. Το ένα από τα δύο κλειδιά ονομάζεται ReadApiKey και το χρησιμοποιούν οι συσκευές για να διαβάσουν τιμές από την πλατφόρμα όταν το κανάλι έχει δηλωθεί ως private, ενώ στην περίπτωση που έχει δηλωθεί public αγνοείται. Το δεύτερο κλειδί λέγεται WriteApiKey και αποστέλλεται κάθε φορά που γίνεται ανέβασμα τιμών τιμών από κάθε συσκευή για να γίνεται η αυθεντικοποίηση. Ο χρήστης που έχει το λογαριασμό στην πλατφόρμα μπορεί οποιαδήποτε στιγμή να αλλάξει αυτά τα κλειδιά κάνοντας αδύνατη την επικοινωνία από τις αντίστοιχες συσκευές συσκευές.

Μπορούμε να έχουμε περισσότερα από ένα κλειδιά για ανάγνωση τιμών αλλά μόνο ένα για εγγραφή.

Δεν υπάρχουν άλλα επίπεδα πρόσβασης στο Dashboard πέρα από τη διαβάθμιση σε Public και Private που αναφέραμε παραπάνω.

στ. Κόστος

Το ThingSpeak διατίθεται ως δωρεάν υπηρεσία για μη εμπορικά μικρά έργα (<3 εκατομμύρια μηνύματα / έτος ή ~ 8.200 μηνύματα / ημέρα). Για μεγαλύτερα έργα ή εμπορικές εφαρμογές, προσφέρονται τέσσερις διαφορετικοί τύποι ετήσιων αδειών: Standard, Academic, Student και Home. Το ThingSpeak αγοράζεται σε μονάδες, όπου μία μονάδα επιτρέπει την επεξεργασία και αποθήκευση 33 εκατομμυρίων μηνυμάτων σε περίοδο ενός έτους (~ 90.000 μηνύματα / ημέρα). Μία μονάδα παρέχει επίσης τη δυνατότητα δημιουργίας σταθερού αριθμού καναλιών στο ThingSpeak.

Στην εικόνα Α.1.4 βλέπουμε το ελεύθερο πακέτο που προσφέρει η πλατφόρμα. Η προσφορά ενός πακέτου χωρίς χρονικά όρια αποτελεί ένα από τα μεγαλύτερα πλεονεκτήματα της πλατφόρμας σε σχέση με όλες τις άλλες πλατφόρμες της αγοράς πράγμα που την κάνει δελεαστική για ερασιτέχνες, μαθητές και φοιτητές, ενώ η δυνατότητα της χρήσης του Matlab για ανάλυση των δεδομένων την κάνει δημοφιλή δημοφιλή για χρήση σε πανεπιστήμια.

Βλέπουμε ότι οι περιορισμοί έχουν να κάνουν με τον αριθμό μηνυμάτων ανα έτος, τον αριθμό τον καναλιών (συσκευών), το χρόνο διάρκειας των υπολογισμών του κώδικα του Matlab που θα εισάγουμε, το είδος της υποστήριξης, τον αριθμό των ταυτόχρονων subscriptions στη σύνδεση MQTT και τέλος προστίθεται μια καθυστέρηση 15 δευτερολέπτων μεταξύ των μηνυμάτων. Αυτό σημαίνει πως για μια τιμή ενός αισθητήρα δε μπορεί να ενημερώνεται η βάσει σε χρονικά διαστήματα μικρότερα από αυτά τα 15 δευτερόλεπτα.

	FREE For small non-commercial projects
Scalable for larger projects	✖ No. Annual usage is capped.
Number of messages	3 million/year (~8.200/day)[2]
Message update interval limit	Every 15 seconds
Number of channels	4
MATLAB Compute Timeout	20 seconds
Number of simultaneous MQTT subscriptions	Limited to 3
Private channel sharing	Limited to 3 shares
Technical Support	Community Support

Εικόνα Α.1.4: Οι δυνατότητες του ελεύθερου πακέτο

Αναλυτικά όλα τα υπόλοιπα πακέτα παρουσιάζονται στην εικόνα Α.1.5 και εικόνα Α.1.6.

	FREE For time-limited commercial evaluation of the service	STANDARD For all commercial, government and revenue generating activities	ACADEMIC For academic use by faculty, staff, or researchers at degree-granting institutions[1]
Scalable for larger projects	✖ No. Annual usage is capped.	✔	✔
Number of messages	3 million/year (~8.200/day)[2]	33 million/year per unit (~90.000/day per unit)[2]	33 million/year per unit (~90.000/day per unit)[2]
Message update interval limit	Every 15 seconds	Every second	Every second
Number of channels	4	250 per unit	250 per unit
MATLAB Compute Timeout	20 seconds	60 seconds	60 seconds
Number of simultaneous MQTT subscriptions	Limited to 3	50 per unit	50 per unit
Private channel sharing	Limited to 3 shares	Unlimited	Unlimited
Technical Support	Community Support	Standard MathWorks support	Standard MathWorks support

Εικόνα Α.1.5 Τα πακέτα Free, Standard και Academic που προσφέρονται από την πλατφόρμα Thingspeak

	STUDENT For students at degree-granting institutions[1]	HOME For personal use only[1]
Scalable for larger projects	✔	✔
Number of messages	33 million/year per unit (~90.000/day per unit)[2]	33 million/year per unit (~90.000/day per unit)[2]
Message update interval limit	Every second	Every second
Number of channels	10 per unit	10 per unit
MATLAB Compute Timeout	20 seconds	20 seconds
Number of simultaneous MQTT subscriptions	50 per unit	50 per unit
Private channel sharing	Unlimited	Unlimited
Technical Support	Community Support	Community Support

Εικόνα Α.1.6 Τα πακέτα Student και Home που προσφέρονται από την πλατφόρμα Thingspeak

Το κόστος των πακέτων τον Σεπτέμβριο του 2020 έχει ως εξής:

- Standard: 600 €/year
- Academic: 250 €/year
- Student: 110 €/year
- Home: 150 €/year

ζ. Υποστήριξη

Για να υποστηριχθούν οι χρήστες υπάρχουν δύο τρόποι. Είτε απευθείας μέσα από το κέντρο υποστήριξης της MathWorks είτε μέσα από το forum της κοινότητας. Αυτό έχει να κάνει το πακέτο που θα επιλέξει ο χρήστης. Τα πακέτα Home και Student υποστηρίζονται μόνο από την κοινότητα πράγμα που αποτελεί σοβαρό μειονέκτημα. [3]

2 Η ΠΛΑΤΦΟΡΜΑ THINGER.IO

A. Overview

Η δεύτερη πλατφόρμα που θα αναλύσουμε ως προς τα χαρακτηριστικά της αλλά και θα δοκιμάσουμε τη λειτουργία της κατασκευάζοντας ένα απλό project είναι η Open Sourse πλατφόρμα Thinger.io

Το Thinger.io είναι μια πλατφόρμα IoT ανοιχτού κώδικα που αναπτύχθηκε από την INTERNET OF THINGER SL, μια ισπανική εταιρεία με στόχο την παροχή μιας αποτελεσματικής, συνεπούς και εύχρηστης τεχνολογίας για το IoT.

Είναι μια πλατφόρμα cloud IoT που παρέχει κάθε απαραίτητο εργαλείο για την παραγωγή κλιμακούμενα και διαχειρίσιμα συνδεδεμένα προϊόντα IoT.

α. Συσκευές

Η πλατφόρμα επιτρέπει τη σύνδεση κάθε είδους συσκευής ή Web Client.

Βασικά υποστηρίζει τα παρακάτω είδη συσκευών:

- Arduino
- Linux/Raspberry PI
- MQTT clients
- Low Power devices or Edge devices όπως Sigfox, LoraWAN, TheThingsNetwork κλπ.
- Http devices

Το πρώτο βήμα για να ξεκινήσουμε οποιοδήποτε έργο IoT με το Thinger.io (εκτός από μη συνδεδεμένες συσκευές όπως το Sigfox) είναι η δημιουργία ενός προφίλ συσκευής, το οποίο θα συσχετίζει τη συσκευή υλικού με το λογαριασμό χρήστη.

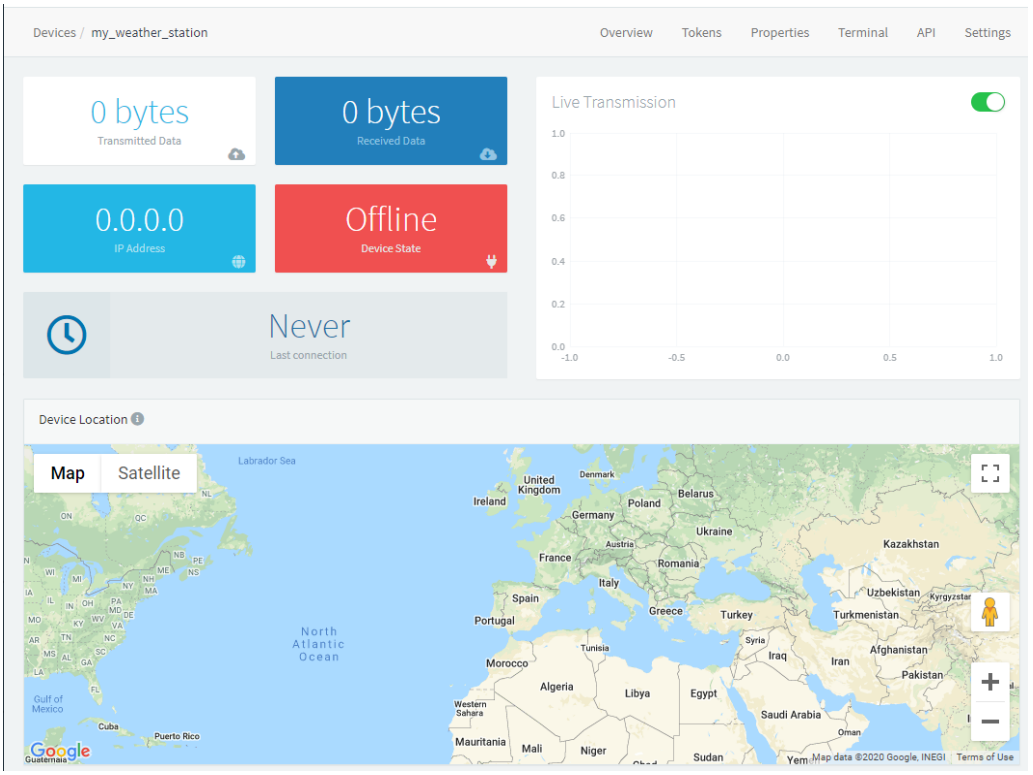
Κάθε συσκευή έχει το δικό της αναγνωριστικό / διαπιστευτήριο (Device credentials), επομένως μια παραβιασμένη συσκευή δεν θα επηρεάσει άλλες συσκευές. Όλοι οι κωδικοί πρόσβασής των συσκευών στο διακομιστή αποθηκεύονται με ασφάλεια χρησιμοποιώντας το PBKDF2 (Password-Based Key Derivation Function 2) SHA256 32 bytes που δημιουργείται με PRNG (pseudorandom number generator) και έτσι παρέχεται προστασία από επιθέσεις brute force.

Για να δούμε τις επιμέρους λειτουργίες προσπαθήσαμε να εισάγουμε μια συσκευή στο σύστημα μέσα από την πλατφόρμα. Κατα την εισαγωγή μας ζητήθηκε ο τύπος της συσκευής, ένα device ID το οποίο το καθορίζουμε εμείς και δεν χρειάζεται να έχει σχέση με κάποιο

χαρακτηριστικό της συσκευής (πχ. MAC address), μια περιγραφή και ένας κωδικός που θα χρησιμοποιηθεί για τη σύνδεση της συσκευής (device credentials).

Το device ID πρέπει να είναι διαφορετικό για κάθε συσκευή που θα εισάγουμε.

Μετά την εισαγωγή εμφανίστηκε μια εντυπωσιακή οθόνη προβολής των χαρακτηριστικών της συσκευής όπως βλέπουμε στην εικόνα Α.2.1.



Εικόνα Α.2.1 Το Dashboard συσκευών της πλατφόρμας Thinger.io

Μέσα από το dashboard της συσκευής μπορούμε να δούμε πάρα πολλές πληροφορίες που σχετίζονται με αυτή όπως τον όγκο των δεδομένων που έστειλε ή έλαβε η συσκευή, αν είναι online ή offline, την IP της, την τοποθεσία της και το χρόνο από την τελευταία σύνδεση.

Στο dashboard της συσκευής θα δούμε να συμπεριλαμβάνεται και ένα τερματικό. Αυτή η λειτουργία έχει σχεδιαστεί για να λειτουργεί ως κοινό τερματικό προγράμματος, επιτρέποντας την εκτύπωση μηνυμάτων εντοπισμού σφαλμάτων ή εκτέλεσης από το πρόγραμμα συσκευής, αλλά λειτουργεί μέσω διαδικτύου μέσω πόρου συσκευής.

Όλες οι αλληλεπιδράσεις με τις συνδεδεμένες συσκευές σας, πρέπει να ελεγχθούν μέσω της πλατφόρμας. Από προεπιλογή, όταν επικοινωνούμε με τις συσκευές μας μέσω της κονσόλας Thinger.io, υποφέρουμε σιωπηρά όλα τα αιτήματά σας στην πλατφόρμα με ένα διακριτικό πρόσβασης που αποκτήσαμε από το όνομα χρήστη και τον κωδικό πρόσβασής. Αυτό το είδος εξουσιοδότησης παρέχει πρόσβαση σε όλους τους πόρους του λογαριασμού μας, οπότε μπορούμε να διαμορφώσουμε συσκευές, κάδους κ.λπ. Ωστόσο, αυτή η εξουσιοδότηση λήγει αρκετά συχνά (αλλά ανανεώνεται αυτόματα από το πρόγραμμα περιήγησής σας) και δεν μπορεί να χρησιμοποιηθεί για την παραχώρηση πρόσβασης στις

συσκευές μας σε άλλους χρήστες ή πλατφόρμες, καθώς θα παρέχει πρόσβαση σε όλους τους λογαριασμούς μας.

Για αυτό το λόγο είναι δυνατή η δημιουργία ειδικών διακριτικών πρόσβασης (tokens) για την παραχώρηση πρόσβασης στις συσκευές μας και ακόμη και την παραχώρηση πρόσβασης σε συγκεκριμένους πόρους των συσκευών μας. Επιπλέον, είναι δυνατό να οριστεί η χρονική διάρκεια του token, ενεργοποιώντας μια ημερομηνία λήξης. Με αυτόν τον τρόπο, μπορούμε να παρέχουμε πρόσβαση σε ορισμένους από τους πόρους της συσκευής μας, όπως το IFTTT, μια εξωτερική ιστοσελίδα, ένα κινητό τηλέφωνο ή οποιαδήποτε άλλη υπηρεσία.

β. Επικοινωνία

Για να έχουμε πρόσβαση στους πόρους μιας συσκευής (αισθητήρες, μεταβλητές) χρησιμοποιείται η διεπαφή Rest API. Κάθε πόρος έχει ένα αναγνωριστικό, που σχετίζεται με το όνομα πόρου που ορίζεται στον κωδικα μας. Στη πλατφόρμα Thinger.io, μπορουν να οριστούν 4 διαφορετικοί τύποι πόρων, έναν για εισαγωγή (αποστολή δεδομένων στη συσκευή), έναν για έξοδο (η συσκευή θα στείλει πληροφορίες), ένα για είσοδο / έξοδο (μπορούμε να στείλουμε και να λάβουμε πληροφορίες σε μία κλήση), και ακόμη ένα πόρο επανάκλησης, τον οποίο μπορούμε απλώς να εκτελέσουμε χωρίς αποστολή ή λήψη πληροφοριών. Τα δεδομένα εισόδου και εξόδου, από την οπτική γωνία του API, μπορούν να είναι οποιοδήποτε έγγραφο JSON.

Οι πλατφόρμα χρησιμοποιεί μια μέθοδο POST για την αποστολή τιμών στο συσκευή ή χρησιμοποιώντας μια μέθοδο GET κάνουμε ανάγνωση πληροφοριών από τη συσκευή.

Γενικά τα πρωτόκολλα επικοινωνίας που υποστηρίζει η πλατφόρμα είναι το HTTP (Rest Api) και το MQTT. Δεν υποστηρίζεται το πρωτόκολλο CoAP.

Η πλατφόρμα παρέχει έτοιμο κώδικα και οδηγούς για εύκολη σύνδεση συσκευών όπως Arduino, Linux/Raspberry PI, SigFox καθώς και γενικότερα παραδείγματα για επικοινωνία μέσω HTTP και MQTT (ver 3.1 or 3.1.1 + Web Sockets).

Η βιβλιοθήκη για Arduino που παρέχει κάνει πολύ εύκολη την επικοινωνία ακόμη και για τους αρχάριους.

Όσον αφορά τη μορφή των μηνυμάτων χρησιμοποιεί μηνύματα Json τα οποία έχουν την παρακάτω μορφή:

REST API + ?authorization= + Token

και αποστέλλονται με τη μέθοδο Post.

Αυτό σημαίνει πως αν κάποιος χρήστης θέλει από μια πλατφόρμα να μεταφερθεί στο Thinger.io θα πρέπει να αλλάξει τους κώδικες των συσκευών του, αφού η πλατφόρμα δεν διαθέτει κάποιο εργαλείο ή κάποια ευελιξία με την υποστήριξη πολλών μορφών μηνυμάτων.

γ. Δεδομένα

Η μηχανή αποθήκευσης δεδομένων του Thinger.io μπορεί να ρυθμιστεί ώστε να χρησιμοποιεί διαφορετικά συστήματα βάσεων δεδομένων, όπως MongoDB, DynamoDB ή InfluxDB (χρησιμοποιείται από προεπιλογή σε ιδιωτικές χρήσεις της πλατφόρμας).

Η προεπιλεγμένη διαμόρφωση της μηχανής αποθήκευσης του Thinger.io χρησιμοποιεί το σύστημα InfluxDB, καθώς παρέχει μερικές ενδιαφέρουσες δυνατότητες όπως τη

συγκέντρωση δεδομένων ή τη χρήση ετικετών δεδομένων για την ταξινόμηση δεδομένων βάσει συγκεκριμένων κριτηρίων.

Δεν υπάρχουν περιορισμοί στον όγκο των δεδομένων ή στο ρυθμό αποστολής αφού πρόκειται για Open Source προϊόν. Αυτοί η περιορισμοί θα πρέπει να τεθούν από τον provider που θα φιλοξενήσει την πλατφόρμα.

Η εξαγωγή των δεδομένων από τη βάση μπορεί να γίνει ολικά ή μερικά σε διαφορετικές μορφές αρχείων και να γίνει επεξεργασία αυτών εκτός σύνδεσης.

Data format: CSV, ARFF or JSON format file
Time format: Timestamp or ISO date format
Δεν υπάρχει υποστήριξη εξωτερικών βάσεων.

δ. Πλατφόρμα

Το thinger.io διαθέτει ένα πολύ εύκολο και εύχρηστο Dashboard.

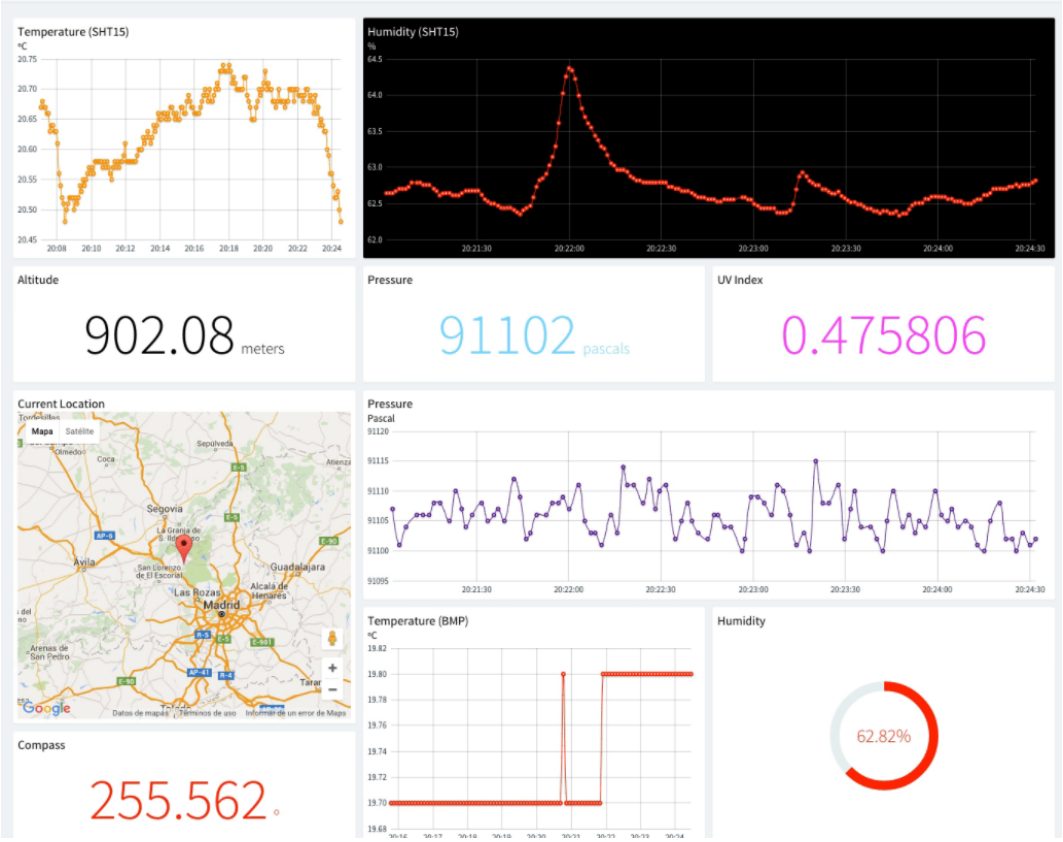
Μπορούμε μέσω αυτού να διαμορφώσουμε τους πίνακες ελέγχου με διαφορετικά widgets, να αλλάξουμε τη διάταξη, τη διάσταση, το χρώμα και τις πηγές δεδομένων και να δημιουργήσουμε πολύτιμες πληροφορίες για την επιχείρηση ή τις εργασίες μας.

Οι πίνακες εργαλείων μπορούν να εμφανίζουν πληροφορίες σε πραγματικό χρόνο από τις συσκευές (χρησιμοποιώντας υποδοχές διαδικτύου μέσω του διακομιστή για ελάχιστη καθυστέρηση) ή να χρησιμοποιούν ιστορικές πληροφορίες που είναι αποθηκευμένες. Είναι δυνατή η ανεξάρτητη διαμόρφωση της πηγής δεδομένων για κάθε widget του πίνακα ελέγχου.

Μια πολύ σημαντική λειτουργία της πλατφόρμας, είναι η δυνατότητα που μας δίνει να διαμορφώσουμε δυναμικά το διάστημα δειγματοληψίας του κάθε πόρου.

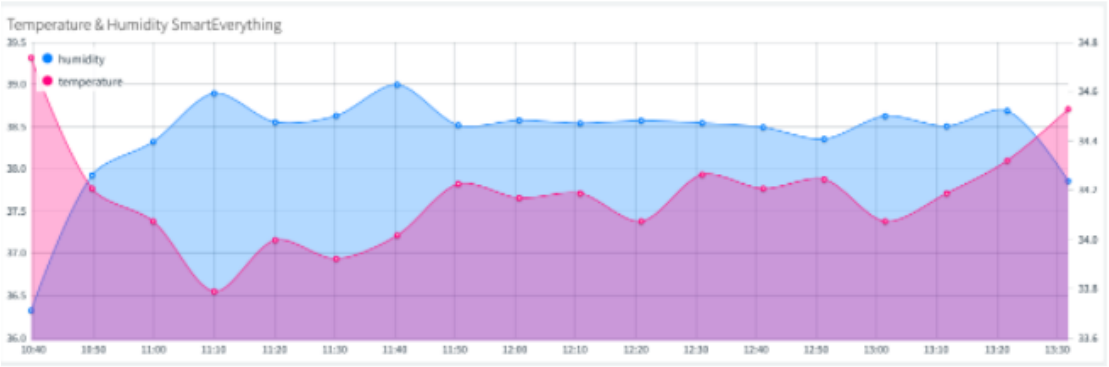
Οι πίνακες ελέγχου δεν είναι μόνο για την εμφάνιση δεδομένων, αλλά μπορούμε επίσης να ενεργοποιούμε σε πραγματικό χρόνο διακόπτες και ρυθμιστικά που είναι συνδεδεμένα με τις συνδεδεμένες συσκευές.

Στην εικόνα Α.2.2 βλέπουμε ένα παράδειγμα των δυνατοτήτων του Visualization της πλατφόρμας.



Εικόνα Α.2.2 Το Dashboard της πλατφόρμας Thinger.io

Εντυπωσιακά είναι τα γραφήματα χρονοσειρών με δυνατότητα απεικόνισης πολλών μεταβλητών στο ίδιο γράφημα.



Εικόνα Α.2.3: Multiple charts

Ένα πολύτιμο εργαλείο είναι το ***Data Aggregation:***

Η εμφάνιση ακατέργαστων δεδομένων απευθείας από έναν χώρο αποθήκευσης μπορεί να είναι δύσκολη όταν υπάρχουν πολλά σημεία δεδομένων και ειδικά εάν αυτά είναι πολύ θορυβώδη ή ακανόνιστα.

Το ***Data Aggregation*** επιτρέπει τη συγκέντρωση δεδομένων χρησιμοποιώντας διαφορετικά στατιστικά στοιχεία, όπως διάμεσος, μέσος όρος, ελάχιστες και μέγιστες τιμές, μετρητής σημείων δεδομένων ανά περίοδο και αθροίσματος δεδομένων. Το aggregation

(συνάθροιση) μπορεί να εφαρμοστεί σε διαφορετικά διαστήματα που κυμαίνονται από πέντε λεπτά έως μία εβδομάδα. Αυτή η παραμετροποίηση μπορεί να γίνει μέσα από τη φόρμα του widget και επίσης στη γραφική παράσταση των χρονοσειρών.

Η επόμενη εικόνα Α.2.4 δείχνει τέσσερις διαφορετικές αναπαραστάσεις του ίδιου συνόλου δεδομένων και χρονικού διαστήματος, χρησιμοποιώντας διαφορετικούς αλγόριθμους:

Εικονα Α.2.4: Εφαρμογή ***Data Aggregation*** *σε χρονοσειρές*

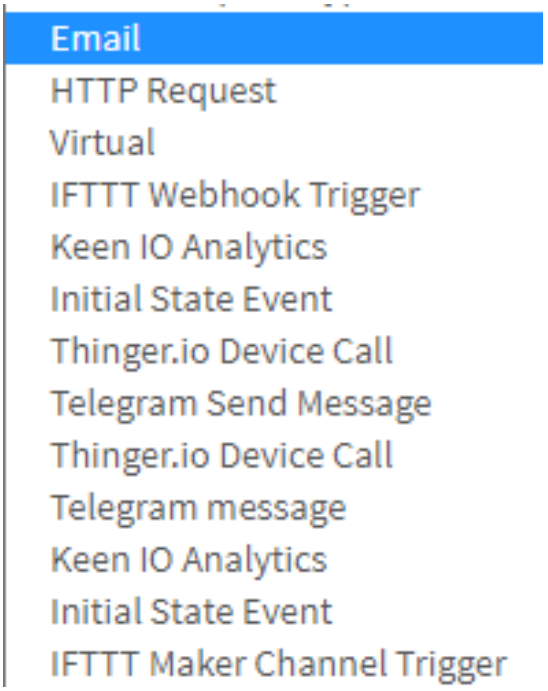
Πέρα από τα βασικά widgets όπως μπουτόν, led, progress bar, donut chart, instruments, value displays κλπ, ένα πολύ ενδιαφέρον widget είναι ***HTML widget***.

Αυτό το widget επιτρέπει τη δημιουργία προσαρμοσμένων διεπαφών αναπαράστασης δεδομένων εάν προγραμματιστεί με τυπικές γλώσσες ιστού όπως HTML, CSS, JS. Είναι επίσης σε θέση να αντιπροσωπεύει δεδομένα από συσκευές Thinger.io ή κάδους δεδομένων (data buckets) ή να εμφανίζει δεδομένα από πηγές τρίτων στον ίδιο πίνακα ελέγχου.

Η αποστολή ειδοποιήσεων σε τρίτους υποστηρίζεται από το Thinger.io με την εφαρμογή **ENDPOINT.** Είναι ένα τρόπος να ενεργοποιηθεί μια υπηρεσία μια διαδικασία ή οτιδήποτε άλλο από τον server προς κάποιον προορισμό.

Ένα ENDPOINT μπορεί κληθεί από μια συσκευή και να εκτελεστεί από τον σερωερ η αποστολή ενός email, SMS, κλήση REST API ή να ενεργοποιηθεί κάποια αλληλεπίδραση με IFTTT.

Η παρακάτω εικόνα Α.2.5 μας δείχνει τους διαφορετικούς τύπους EndPoints που μπορούμε να δημιουργήσουμε:



Εικόνα Α.2.5 Οι τύποι EndPoints της πλατφόρμας Thinger.io

Για παράδειγμα αν μια συνδεδεμένη συσκευή θέλει να στείλει ένα email ή ένα μήνυμα σε κάποιον χρήστη αφαλώς είναι δύσκολο να το κάνει απευθείας λόγω των μικρών της δυνατοτήτων. Μέ τη δημιουργία του αντίστοιχου ENDPOINT στο Thinger.io η συσκευή στέλνει απλώς ένα αίτητα με HTTP και η πλατφόρμα στέλνει το μήνυμα.

Ενδιαφέρον παρουσιάζει το **HTTP Endpoint.** Ένα τελικό σημείο HTTP είναι ένας γενικός τύπος τελικού σημείου που μπορεί να χρησιμοποιηθεί για την αλληλεπίδραση με οποιαδήποτε άλλη υπηρεσία ιστού ή εφαρμογή ιστού. Αυτό το τελικό σημείο μπορεί να διαμορφωθεί ώστε να κάνει οποιοδήποτε αίτημα HTTP, ρυθμίζοντας τη μέθοδο (GET, POST, PUT, PATCH ή DELETE), τη διεύθυνση URL, τις κεφαλίδες και το σώμα.

Το σώμα μπορεί να είναι είτε προσαρμοσμένο σώμα με συγκεκριμένο περιεχόμενο είτε JSON payload με τις πληροφορίες που αποστέλλονται από τη συσκευή.

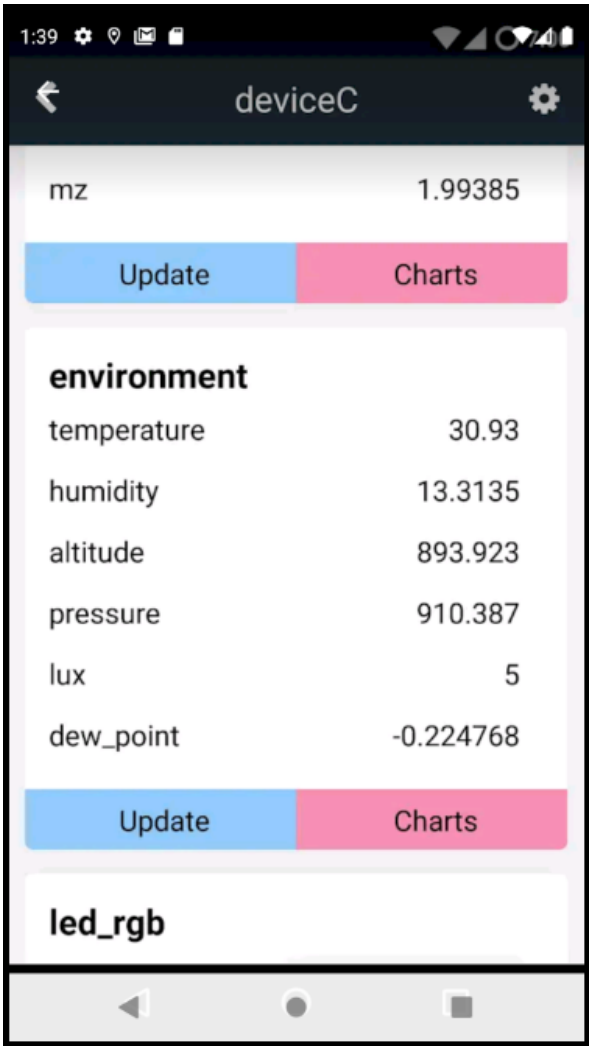
Το **Telegram Bot Endpoint** επίσης δίνει τη δυνατότητα αποστολής ειδοποιήσεων σε συσκευές Android, iOS και σε οποιαδήποτε άλλη υποστηρίζει την ανταλλαγή μηνυμάτων μέσω telegram. Το Telegram είναι μια από τις πιο δημοφιλείς πλατφόρμες άμεσων μηνυμάτων σήμερα.

Ενα σοβαρό μειωνέκτημα του Thinger.io είναι η απουσία ενός “Rule management” για τη δημιουργία κανόνων από τον χρήστη και την αποστολή ειδοποιήσεων συνδυάζοντας τιμές και συμβάντα από τις συσκευές.

Η πλατφόρμα διαθέτει για τους χρήστες Android και iOS μια εφαρμογή μέσα από την οποία μπορούν να έχουν ένα Visualization πάντα μαζί τους. Χωρίς η εφαρμογή να εντυπωσιάζει ιδιαίτερα (εικόνα Α.2.6) δίνει τη δυνατότητα να βλέπουμε tokes από διαφορετικούς χρήστες.

Ο κώδικας του App είναι διαθέσιμος στους developers μέσω του GitHub

(https://github.com/thinger-io/Mobile-App)



Εικόνα Α.2.6 Μια οθόνη από το app της πλατφόρμας Thinger.io

Η πλατφόρμα διαθέτει αρκετά plug-ins τα οποία μπορούν να επεκτείνουν τις λειτουργίες της, όπως:

- Node-RED
- sigfox
- THE THINGS NETWORK
- Grafana

ενώ πολλά ακόμη πρόκειται να προστεθούν σύντομα.

ε. Ασφάλεια

Στο θέμα της ασφάλειας η πλατφόρμα υποστηρίζει τα πρωτόκολλα SSL/TLS τόσο για την HTTP όσο και για την MQTT σύνδεση.

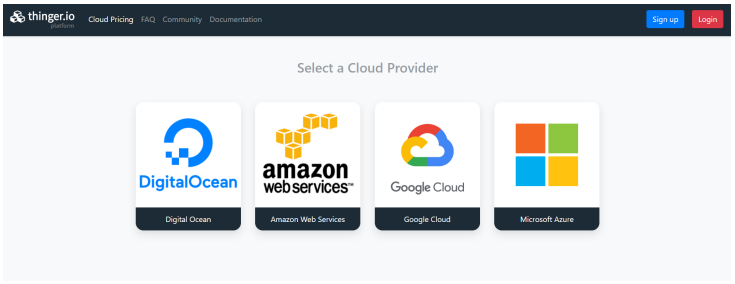
Η ασφάλεια της σύνδεσης των συσκευών γίνεται, όπως αναφέρθηκε παραπάνω με τη χρήση είτε του token είτε με τη χρήση των Device credentials και Device ID.

Το Thinger.io μπορεί να υποστηρίξει πολλούς λογαριασμούς χρηστών που μπορούν να ελέγχονται από τον λογαριασμό διαχείρισης. Αυτή η δυνατότητα επιτρέπει τη δημιουργία ενός δικτύου χρηστών στο οποίο κάθε λογαριασμός χρήστη να έχει τους δικούς του πόρους, όπως συσκευές, πίνακες εργαλείων, κάδο δεδομένων ή ακόμη και τα δικά του πρόσθετα.

στ. Κόστος

Το thinger.io είναι Open Sourse.

Private Instances μπορούν να διατεθούν από διάφορους παρόχους όπως Digital Ocean, Amazon Web Services, Google Cloud, ή Azure (εικόνα Α.2.7).

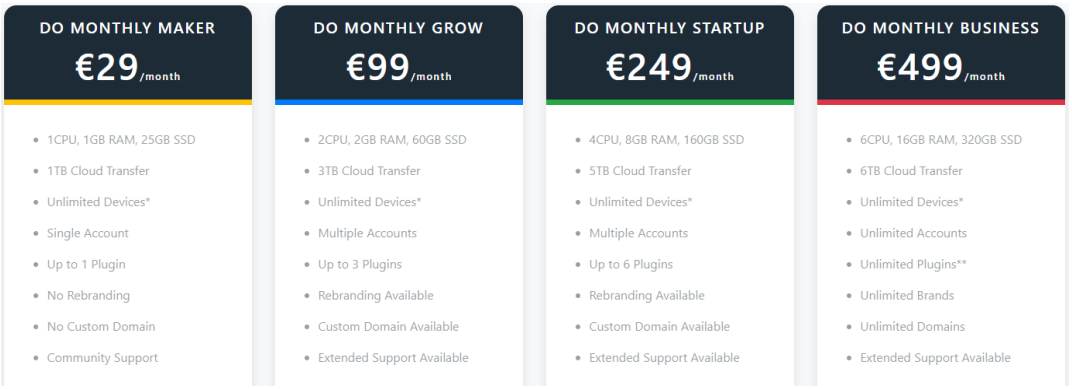


Εικόνα Α.2.7 Οι πάροχοι φιλοξενίας της πλατφόρμας Thinger.io

Οι ιδιωτικές cloud instances μπορούν να αναπτυχθούν με διαφορετικές άδειες, ανάλογα με τις απαιτήσεις του έργου, όπως η απόδοση του κεντρικού υπολογιστή και το εύρος ζώνης.

Το AWS και το Digital Ocean κάνουν αυτόματα deploy μια ιδιωτική εγκατάσταση του thinger.io και έχουν την τιμολογιακή πολιτική που φαίνεται στην εικόνα Α.2.8 και εικόνα Α.2.9.

Εικόνα Α.2.8: Τιμολογιακή πολιτική του AWS



Εικόνα Α.2.9: Τιμολογιακή πολιτική του Digital Ocean

Οι λογαριασμοί Freemium μέσα από την ιστοσελίδα thinger.io είναι ιδανικοί για εκμάθηση και δοκιμή της πλατφόρμας Thinger.io με λίγους περιορισμούς, ωστόσο, για να έχει ο χρήστης την καλύτερη απόδοση και αξιοπιστία αυτής της πλατφόρμας και την πρόσβαση σε ορισμένες προηγμένες δυνατότητες που είναι απαραίτητες για επαγγελματική χρήση, είναι απαραίτητο να αναπτύξει ένα ιδιωτικό Thinger.

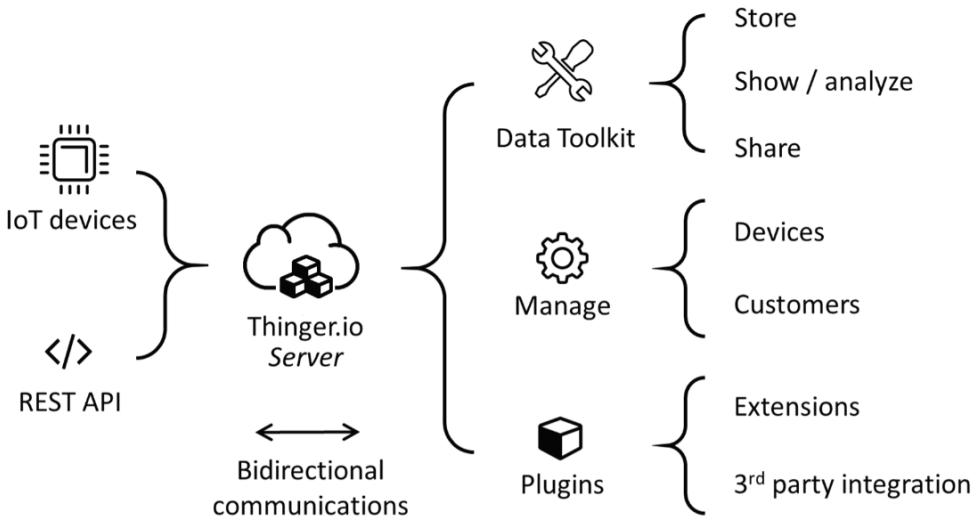
ζ. Υποστήριξη

Οι χρήστες που δεν έχουν αγοράσει κάποια ιδιωτική εγκατάσταση ή έχουν αγοράσει το φθηνότερο πακέτο υποστηρίζονται από το φόρουμ της κοινότητας Thinger.io και το φόρουμ του αποθετηρίου github.

Για επαγγελματική υποστήριξη και για τους χρήστες που χρειάζονται εμπορικές πληροφορίες υπάρχει επικοινωνία μέσω του email: info@thinger.io.

Ασφαλώς υποστήριξη υπάρχει και με την αγορά προνομιούχων πακέτων.

Αρχιτεκτονική [4] [6]



Εικόνα Α.2.10: Διάγραμμα της αρχιτεκτονικής του Thinger.io.

Η πλατφόρμα Thinger.io διαμορφώνεται από δύο βασικά μέρη, ένα Backend (που είναι ο πραγματικός διακομιστής IoT) και ένα Frontend που βασίζεται στον Ιστό που απλοποιεί την εργασία με όλες τις δυνατότητες χρησιμοποιώντας οποιονδήποτε υπολογιστή ή smartphone. Παρακάτω βλέπουμε τα κύρια χαρακτηριστικά που παρέχει αυτή η πλατφόρμα για τη δημιουργία έργων IoT.

Connect devices: Είναι πλήρως συμβατή με κάθε είδους συσκευή, ανεξάρτητα από τον επεξεργαστή, το δίκτυο ή τον κατασκευαστή. Το Thinger.io επιτρέπει τη δημιουργία αμφίδρομων επικοινωνιών με συσκευές Linux, Arduino, Raspberry Pi ή MQTT και ακόμη και με τεχνολογίες αιχμής όπως Sigfox ή LoRaWAN ή άλλους πόρους δεδομένων API διαδικτύου.

Store Device Data: Μόνο μερικά κλικς αρκούν για να δημιουργήσει ο χρήστης έναν κάδο δεδομένων για αποθήκευση δεδομένων IoT με κλιμακωτό, αποδοτικό και προσιτό τρόπο, κάτι που επιτρέπει επίσης τη συγκέντρωση δεδομένων σε πραγματικό χρόνο.

Display Real-time or **Stored Data:** Υπάρχει η δυνατότητα εμφάνισης δεδομένων σε πραγματικό χρόνο ή αποθηκευμένα σε πολλά widget, όπως χρονοσειρές, γραφήματα ντόνατ, μετρητές ή ακόμη και προσαρμοσμένες παραστάσεις για να δημιουργήσετε καταπληκτικούς πίνακες ελέγχου μέσα σε λίγα λεπτά.

Trigger events and data values: Μπορεί να γίνει ενεργοποίηση συμβάντων χρησιμοποιώντας μια ενσωματωμένη μηχανή κανόνων Node-RED

Extend with custom features Η πλατφόρμα μπορεί να επεκταθεί με προσαρμοσμένες λειτουργίες με πολλά πρόσθετα για να προσαρμοστούν έργα IoT στο λογισμικό της εταιρείας σας ή σε οποιαδήποτε άλλη υπηρεσία Διαδικτύου τρίτων.

Custom the appearance Μπορεί να γίνει προσαρμογή της εμφάνισης χάρη στο πλήρως επαναπροσδιορισμένο frontend, που επιτρέπει την εισαγωγή της επωνυμίας της εταιρείας και των λογότυπων. [4] [6]

Ο σέρβερ μπορεί να κάνει αμφίδρομες συνδέσεις από τη μια πλευρά με εξωτερικές συσκευές και από την άλλη με εσωτερικά εργαλεία τα οποία επιτρέπουν την αποθήκευση και διαχείριση τιμών, τη διαχείριση πελατών και συσκευών καθώς και έναν αριθμό plugins τα οποία επεκτείνουν τις δυνατότητες της πλατφόρμας.

Αναλύοντας περαιτέρω αυτή τη δομή της εικόνας 18 βλέπουμε ότι η πλατφόρμα μπορεί να χωριστεί σε 4 βασικά μέρη.

1. IoT devices και REST API:

Το πρώτο κομμάτι της πλατφόρμας περιλαμβάνει τις εξωτερικές συνδέσεις που μπορούν να πραγματοποιηθούν σε αυτή.

Όπως αναφέρθηκε και παραπάνω μπορούν να συνδεθούν και να επικοινωνήσουν με την πλατφόρμα διαφορετικού τύπου συσκευές με διαφορετικά πρωτόκολλα επικοινωνίας. [6]

Εδώ υποστηρίζεται η αρχιτεκτονική REST.

Είναι αρχιτεκτονικό στυλ για distributed hypermedia systems και παρουσιάστηκε για πρώτη φορά από τον Roy Fielding το 2000.

Το REST έχει 6 βασικούς περιορισμούς που πρέπει να ικανοποιηθούν εάν μια διεπαφή πρέπει να αναφέρεται ως RESTful. Αυτές οι αρχές παρατίθενται παρακάτω.

Client–server:

Διαχωρίζοντας τις περιπτώσεις διεπαφής χρήστη από τις περιπτώσεις αποθήκευσης δεδομένων, βελτιώνεται τη φορητότητα του περιβάλλοντος εργασίας χρήστη σε πολλές πλατφόρμες και βελτιώνεται η επεκτασιμότητα απλοποιώντας τα στοιχεία του διακομιστή.

Stateless:

Κάθε αίτημα από πελάτη σε διακομιστή πρέπει να περιέχει όλες τις απαραίτητες πληροφορίες για την κατανόηση του αιτήματος και δεν μπορεί να εκμεταλλευτεί οποιοδήποτε αποθηκευμένο περιβάλλον στον διακομιστή. Συνεπώς, η κατάσταση συνεδρίας διατηρείται αποκλειστικά στον πελάτη.

Cacheable:

Οι περιορισμοί της προσωρινής μνήμης απαιτούν τα δεδομένα εντός μιας απόκρισης σε ένα αίτημα να επισημαίνονται έμμεσα ή ρητά ως προσωρινά αποθηκευμένα ή μη προσωρινά αποθηκευμένα. Εάν μια απόκριση είναι προσωρινά αποθηκευμένη, τότε μια κρυφή μνήμη πελάτη έχει το δικαίωμα να επαναχρησιμοποιήσει αυτά τα δεδομένα απόκρισης και αργότερα για ισοδύναμα αιτήματα.

Uniform interface:

Εφαρμόζοντας την αρχή μηχανικής λογισμικού στη διεπαφή συστατικών στοιχείων, απλοποιείται η συνολική αρχιτεκτονική του συστήματος και βελτιώνεται η ορατότητα των αλληλεπιδράσεων. Προκειμένου να επιτευχθεί μια ομοιόμορφη διεπαφή, απαιτούνται πολλοί αρχιτεκτονικοί περιορισμοί για την καθοδήγηση της συμπεριφοράς των συστατικών.

Το REST ορίζεται από τέσσερις περιορισμούς διεπαφής:

- Αναγνώριση πόρων
- Χειρισμός πόρων μέσω παραστάσεων
- Αυτο-περιγραφικά μηνύματα
- Υπερμέσα ως μηχανή κατάστασης εφαρμογής.

Layered system

Το στυλ συστήματος με στρώσεις επιτρέπει σε μια αρχιτεκτονική να αποτελείται από ιεραρχικά στρώματα περιορίζοντας τη συμπεριφορά των συστατικών έτσι ώστε κάθε στοιχείο να μην μπορεί να "δει" πέρα από το άμεσο επίπεδο με το οποίο αλληλεπιδρούν.

Code on demand (optional)

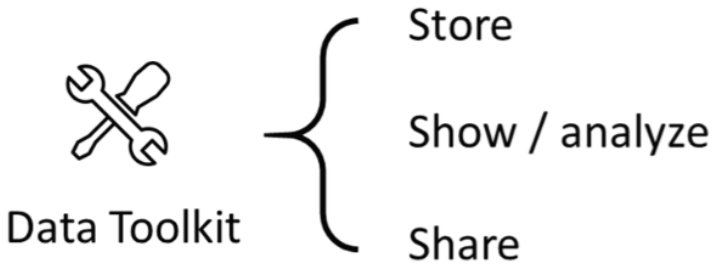
Το REST επιτρέπει την επέκταση της λειτουργικότητας του πελάτη με λήψη και εκτέλεση κώδικα με τη μορφή μικροεφαρμογών ή σεναρίων. Αυτό απλοποιεί τους πελάτες μειώνοντας τον αριθμό των δυνατοτήτων που απαιτούνται να εφαρμοστούν.

[5]

2. **Server:**

Ο Server αποτελεί το δεύτερο μέρος της αρχιτεκτονικής της πλατφόρμας. Επιτρέπει τόσο αμφίδρομες συνδέσεις με τις συσκευές και το REST API αλλά και αμφίδρομες εσωτερικές συνδέσεις με τα εργαλεία που περιλαμβάνει η πλατφόρμα.

3. **Data Toolkit**

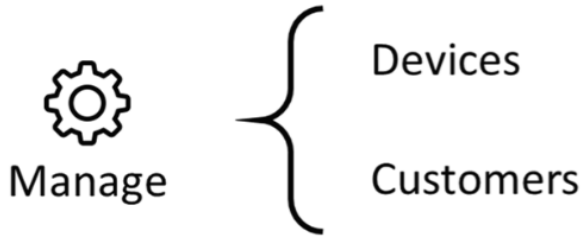


Εικόνα Α.2.11 Τα διαθέσιμα εργαλεία του Data Toolkit της πλατφόρμας Thinger.io

Το Data Toolkit περιλαμβάνει ένα σύνολο εργαλείων που αφορούν την αποθήκευση, την εμφάνιση-ανάλυση των δεδομένων και τον διαμοιρασμό.

ο Thinger.io παρέχει ένα ευέλικτο σύστημα αποθήκευσης cloud που επιτρέπει τη μεταφόρτωση αρχείων στο διακομιστή IoT προκειμένου να παρέχει υποστήριξη σε άλλες δυνατότητες πλατφόρμας όπως το Widget HTML ή το σύστημα OTA. Οι πληροφορίες θα αποθηκευτούν σε μια μη πτητική μνήμη του κεντρικού υπολογιστή διακομιστή, οπότε θέλει προσοχή το το μέγεθος των αρχείων, ειδικά όταν γίνεται κοινή χρήση με άλλους λογαριασμούς χρηστών, για να αποφευχθεί η κορεσμός του τοπικού χώρου αποθήκευσης διακομιστή.

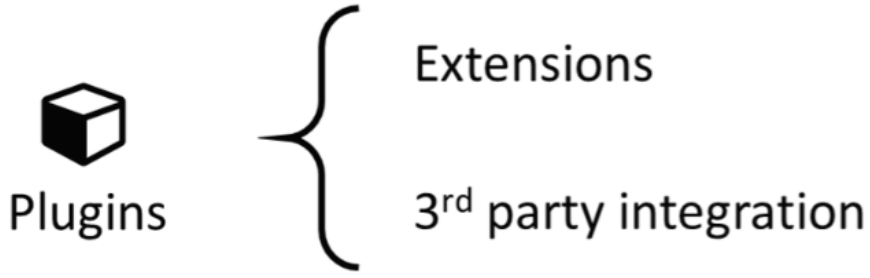
4. **Manage:**



Εικόνα Α.2.12 Οι διαθέσιμες επιλογές από το Manage της πλατφόρμας Thinger.io

Περιλαμβάνει ένα σύστημα διαχείρισης των συσκευών και ένα σύστημα διαχείρισης των πελατών.

5. **Plugins:**



Εικόνα Α.2.13 Οι διαθέσιμες επιλογές από το Plugins της πλατφόρμας Thinger.io

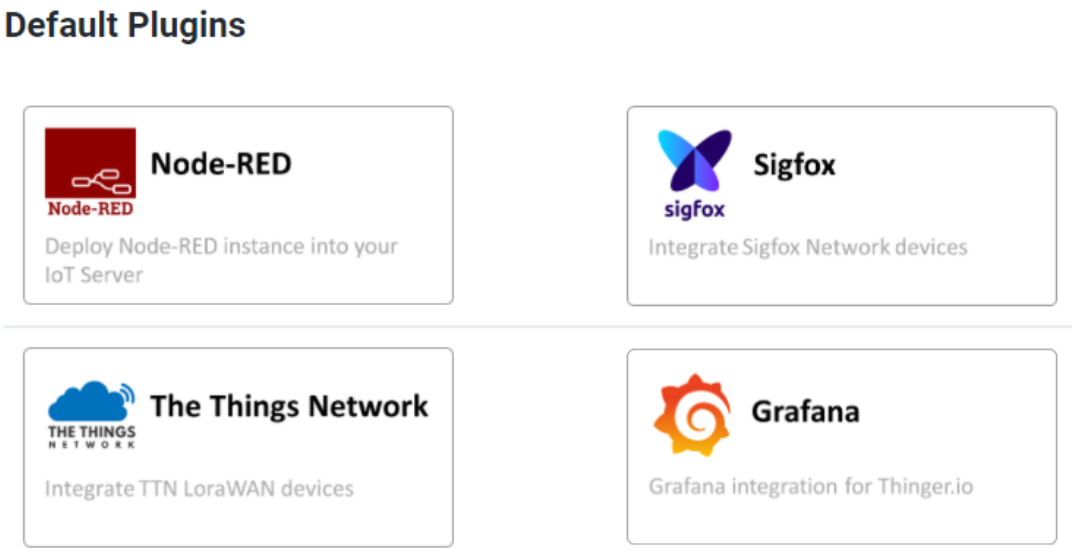
Οι προσθήκες είναι επεκτάσεις που επιτρέπουν τη συμπλήρωση της πλατφόρμας Thinger.io με πρόσθετες δυνατότητες ή τη δημιουργία ενοποιήσεων με υπηρεσίες διαδικτύου τρίτων, αλγόριθμους ανάλυσης δεδομένων ή προσαρμοσμένο λογισμικό.

Ο πυρήνας της πλατφόρμας Thinger.io IoT έχει σχεδιαστεί για να είναι λιτός και ελαφρύς, ώστε να μεγιστοποιεί την ευελιξία και να ελαχιστοποιεί τη μάθηση και την κοινή διαμόρφωση των δικτύων IoT. Οι προσθήκες προσφέρουν προσαρμοσμένες λειτουργίες που μπορούν να αναπτυχθούν κατ 'απαίτηση, επιτρέποντας σε κάθε χρήστη να συμπληρώσει τον διακομιστή IoT ως συγκεκριμένες ανάγκες του.
Για οδηγίες και πληροφορίες σχετικά με την εγκατάσταση, την αναβάθμιση, την αντιμετώπιση προβλημάτων και τη διαχείριση των προσθηκών Thinger.io, ανατρέξτε στην ενότητα "Διαχείριση προσθηκών". Για να μάθετε πώς να χρησιμοποιείτε οποιαδήποτε υπάρχουσα προσθήκη, απλώς βρείτε παρακάτω μια λίστα με κάθε μία.

Οι προσθήκες είναι επεκτάσεις που επιτρέπουν τη συμπλήρωση της πλατφόρμας Thinger.io με πρόσθετες δυνατότητες ή τη δημιουργία ενοποιήσεων με υπηρεσίες διαδικτύου τρίτων, αλγόριθμους ανάλυσης δεδομένων ή προσαρμοσμένο λογισμικό.

Ο πυρήνας της πλατφόρμας Thinger.io IoT έχει σχεδιαστεί για να είναι λιτός και ελαφρύς, ώστε να μεγιστοποιεί την ευελιξία και να ελαχιστοποιεί τη μάθηση και την κοινή διαμόρφωση των δικτύων IoT. Οι προσθήκες προσφέρουν προσαρμοσμένες λειτουργίες που μπορούν να αναπτυχθούν κατ 'απαίτηση, επιτρέποντας σε κάθε χρήστη να συμπληρώσει τον διακομιστή IoT ως συγκεκριμένες ανάγκες του.

Παρακάτω (εικόνα Α.2.14)βλέπουμε μερικά από τα προκαθορισμένα plugins της πλατφόρμας:



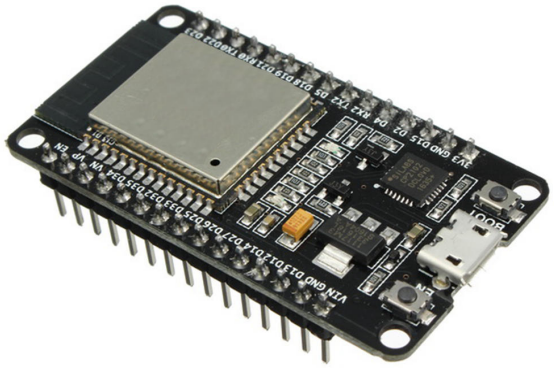
Εικόνα Α.2.14 Τα Plugins που παρέχει η πλατφόρμα Thinger.io

Η 3rd Party Integration επιτρέπει την πρόσβαση σε λειτουργίες ή δεδομένα για χρήση σε ιστότοπο ή άλλες εφαρμογές. Με αυτό τον τρόπο υπάρχει μια επικοινωνία της πλατφόρμα με άλλες εφαρμογές.

6 . Δημιουργία ενός project για δοκιμή της πλατφόρμας ThingerIO

Μετά την ανάλυση των βασικών χαρακτηριστικών και της αρχιτεκτονικής της κάθε πλατφόρμας, θα γίνει υλοποίηση ενος project διαχείρισης περιβαλλοντικών δεδομένων κτιριακών εγκαταστάσεων.

Η συσκευή που θα χρησιμοποιηθεί για την αποστολή των κτιριακών δεδομένων είναι ένα module ESP32. [8]



Εικόνα Α.2.15: ESP32 development Module

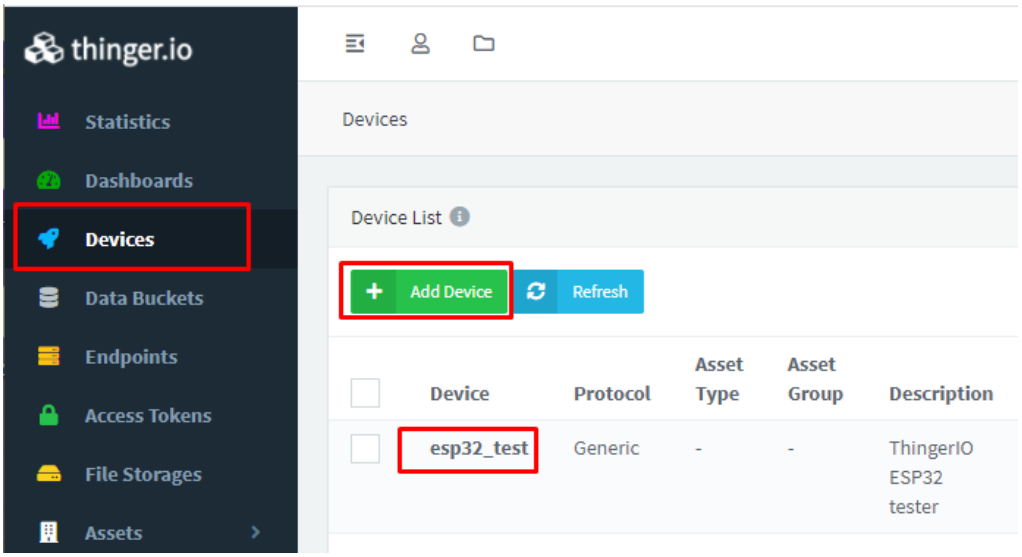
Το ESP32 είναι ένα Module χαμηλού κόστους και χαμηλής κατανάλωσης. Χρησιμοποιεί μικροελεγκτή με ενσωματωμένο Wi-Fi και Bluetooth διπλής λειτουργίας (Classic & BLE).

Η σειρά ESP32 χρησιμοποιεί μικροεπεξεργαστή Tensilica Xtensa LX6 σε παραλλαγές διπλού πυρήνα και μονού πυρήνα και περιλαμβάνει ενσωματωμένους διακόπτες κεραίας, RF balun, ενισχυτή ισχύος, ενισχυτή λήψης χαμηλού θορύβου, φίλτρα και μονάδες διαχείρισης ισχύος. Το ESP32 δημιουργήθηκε και αναπτύχθηκε από την Espressif Systems, μια κινεζική εταιρεία με έδρα τη Σαγκάη και κατασκευάζεται από την TSMC χρησιμοποιώντας τη διαδικασία των 40 nm. Είναι διάδοχος του μικροελεγκτή ESP8266.

Περιέχει μεγάλο αριθμό pins, εκ' των οποίων περισσότερα από είκοσι μπορούν να χρησιμοποιηθούν για τη σύνδεση ψηφιακών και αναλογικών αισθητήρων, ενώ υποστηρίζει συνδέσεις SPI, I2C, I2S και UART [9]

Για τις δοκιμές δεν θα χρησιμοποιηθούν πραγματικοί αισθητήρες αλλά το module ESP32 θα στέλνει είτε τυχαίες τιμές είτε τιμές προερχόμενες από προσομοιωτή.

Το πρώτο βήμα ήταν η δημιουργία μιας νέας συσκευής στην πλατφόρμα, όπου εισαγάγαμε ένα device με όνομα ESP232_test. (εικόνα Α.2.16)



Εικόνα Α.2.16 Ο διαχειριστής συσκευών της πλατφόρμας Thinger.io

Αυτά που ζητήθηκαν από την πλατφόρμα να εισάγουμε ήταν:

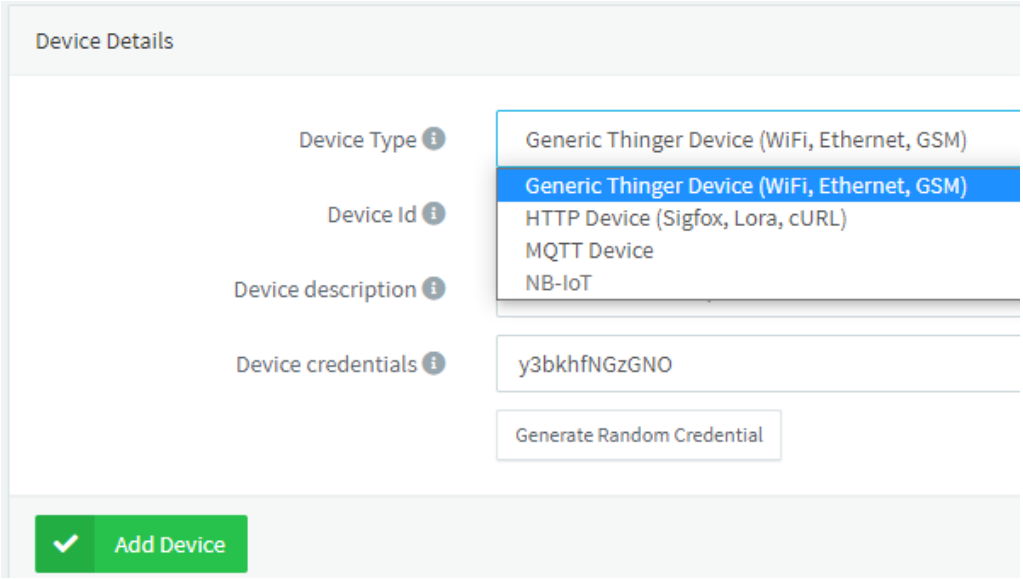
Ο τύπος της συσκευής, όπου επιλέχθηκε ο τύπος “Generic Thinger Device (WiFi, Ethernet, GSM) (εικόνας Α.2.17)

Το Device ID, το οποίο αποτελεί το όνομα της συσκευής στην πλατφόρμα.

Μια σύντομη περιγραφή

Το Device credential, το οποίο είναι ένα αλφαριθμητικό που παίζει το ρόλο κωδικού ασφαλείας σύνδεσης για τη συγκεκριμένη συσκευή και θα πρέπει να αποστέλλεται από τη συσκευή κατα τη διάρκεια της σύνδεσής της. Αυτό σημαίνει ότι θα πρέπει να εισαχθεί στον κώδικα επικοινωνίας της συσκευής ή να υπάρχει η δυνατότητα να γίνεται εισαγωγή από τον χρήστη της συσκευής.

Ο χρήστης της πλατφόρμας (διαχειριστής) μπορεί να αλλάξει αν χρειαστεί αυτόν τον κωδικό και να εμποδίσει τη συγκεκριμένη συσκευή να έχει πλέον τη δυνατότητα να συνδεθεί.



Εικόνα Α.2.17 Οι υποστηριζόμενοι τύποι συνδέσεων συσκευών της πλατφόρμας Thinger.io

Παρακάτω βλέπουμε τα βασικά μέρη του κώδικα C++ που χρησιμοποιήθηκε για τη σύνδεση και τη δημιουργία του project ενεργειακής κτιριακής διαχείρισης.

Αρχικά εισάγουμε το username, device_ID και device_gredential για να μπορέσει η πλακέτα να συνδεθεί στην πλατφόρμα, καθώς και τα απαραίτητα SSID και WiFi password για να συνδεθεί στο τοπικό δίκτυο WiFi.

```
//---------------------------------------- Για το Thinger.io
#include <ThingerESP32.h>
#define USERNAME "iliaslamprou"                 //web
#define DEVICE_ID "esp32_test"                       // web
#define DEVICE_CREDENTIAL "wHh2yqFwnw9yrl"     // web

ThingerESP32 thing(USERNAME, DEVICE_ID, DEVICE_CREDENTIAL);


//---------------------------------------- WiFi
#define SSID "WiFi NETWORK"
#define SSID_PASSWORD "***********"
```

Ακολουθεί ο υπόλοιπος κώδικας

```
//----------- Για ανάγνωση της εσωτερικής θερμοκρασίας συσκευής
#ifdef __cplusplus
extern "C" {
#endif
uint8_t temprature_sens_read();
#ifdef __cplusplus
}
#endif
uint8_t temprature_sens_read();

//---------------------------------------------- Sensors
int humidity=0;
int temperature=0;
int CoolantTemp=0;
int OnOff=0;
int HiLow=0;
int esp_Temp=0;
long WiFi_RSSI=0;

unsigned long lastTimeSend=0;
```

```
//================================================ setup
void setup() {
  Serial.begin(9600);
  thing.add_wifi(SSID, SSID_PASSWORD);
  // digital pin control example
  thing["led_on_off"] << digitalPin(4);
  thing["led_hi_low"] << digitalPin(17);
  //--- αναστροφή κατάστασης του ενσωματωμένου LED
   thing["led_test"] << [](pson& in){
    if(in.is_empty()){
        // We send back the pin value to thinger platform
        in = (bool) !digitalRead(LED_BUILTIN);
    }
    else{
        // This code is called whenever the "led" resource change
        digitalWrite(LED_BUILTIN, in ?  LOW : HIGH);
    }
  };
  // resource output example (i.e. reading a sensor value)
  thing["millis"] >> outputValue(millis());
  thing["indoor_humidity"] >> outputValue(humidity);
  thing["indoor_temperature"] >> outputValue(temperature);
  thing["coolant_temperature"] >> outputValue(CoolantTemp);
  thing["on_off"] >> outputValue(OnOff);
  thing["hi_low"] >> outputValue(HiLow);
  thing["esp_Temp"] >> outputValue(esp_Temp);
  thing["wifi_rssi"] >> outputValue(WiFi_RSSI);

  //-- Άλλος τρόπος να ανέβουν τιμές
  /*thing["dht11"] >> [](pson& out){
    out["humidity"] = random(100);
    out["celsius"] = random(50);
    out["fahrenheit"] = random(50);
  };
  */
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(4,OUTPUT);   // OnOff
  pinMode(17,OUTPUT);  // HiLow
  pinMode(18,INPUT);   // Test Button
  lastTimeSend=millis();
  // more details at http://docs.thinger.io/arduino/
}
```

```
//============================================== loop
void loop() {
  thing.handle();

  humidity=50+random(20);
  temperature=30+random(10);
  CoolantTemp=getDS18B20_temperature();
  OnOff=digitalRead(4);
  HiLow=digitalRead(17);
  esp_Temp=getESP32_internalTemperature();
  WiFi_RSSI=getWiFiRSII();

//--- Παράδειγμα ανεβάσματος τιμών στα properties
/*  if(millis()-lastTimeSend>5000){    //must be flow controlled

    //create a pson with new values
    pson data;
    data["longitude"]=random(10);
    data["latitude"]=random(10);
    thing.set_property("location", data, true);
    lastTimeSend=millis();
    }
*/
//--- Παράδειγμα διαβάσματος τιμών από τα properties
/*    pson data;
    thing.get_property("My_Property", data);//retrieving data from the platform
    float lng=data["longitude"];
    float lat=data["latitude"];
    Serial.print("L: ");
    Serial.print(lng);
    Serial.print(" , l: ");
    Serial.println(lat);
    lastTimeSend=millis();
*/
 //--- Παράδειγμα EndPoint
 if(millis()-lastTimeSend>60000){
    pson data;
    data["temperature"] = random(20+random(10));
    data["humidity"] = random(30+random(50));
    thing.call_endpoint("EndPoint_Email_Test", data);
    Serial.println("Send email");
    lastTimeSend=millis();
 }
}
```

```
//============================== getESP32_internalTemperature
// Ανάγνωση εσωτερικής θερμοκρασίας του ESP32
int getESP32_internalTemperature(){
  int esp_temp=(temprature_sens_read() - 32) / 1.8;  // convert to Celcius
  //Serial.printf("Internal Temperature: %dC°\n",esp_temp);
  return esp_temp;
}


//============================================== getWiFiRSII
// Ανάγνωση της στάθμης του σήματος WiFi
int getWiFiRSII(){
    long rssi = 0;
    long averageRSSI = 0;
    int points=10;

    for (int i=0;i < points;i++){
        rssi += WiFi.RSSI();
        delay(20);
    }
  averageRSSI = rssi/points;
 // Serial.printf("WiFi RSSI=%ddBm\n",averageRSSI);
  return averageRSSI;
}

//============================================== getDS18B20_temoerature
// Ανάγνωση θερμοκρασίας
int getDS18B20_temperature(){
  return 20+random(5);
}
```

Στη συνέχεια έγινε η σύνδεση της πλακέτας με την πλατφόρμα και ακολούθησε η δημιουργία ενός Dashboard για το project ενεργειακής διαχείρισης.

Η δημιουργία του Dashboard περιοριζόταν από ένα μικρό αριθμό widgets τα οποία είχαν και πολύ μικρό βαθμό παραμετροποίησης.

Στα θετικά ήταν η δυνατότητα εξαγωγής της πλατφόρμα σαν αρχείο κειμένου και η εισαγωγή του σε άλλο Dashboard ακόμη και διαφορετικού χρήστη.

Η free έκδοση του Thinger.io δεν επιτρέπει τη δημιουργία Data Bucket με αποτέλεσμα ο server να γίνει Deploy ώστε να εξεταστούν κάποιες λειτουργίες που ήταν αδύνατο να εξεταστούν στο cloud server της πλατφόρμας.

Στην εικόνα Α.2.18 βλέπουμε το Dashboard που δημιουργήθηκε για το project δοκιμής.

Στο πάνω μέρος ο διακόπτης On-Off ενεργοποιεί ή όχι το σύστημα κλιματισμού και την ένδειξη λειτουργίας βλέπουμε στο ενδεικτικό On-Off που βρίσκεται δεξιά του.

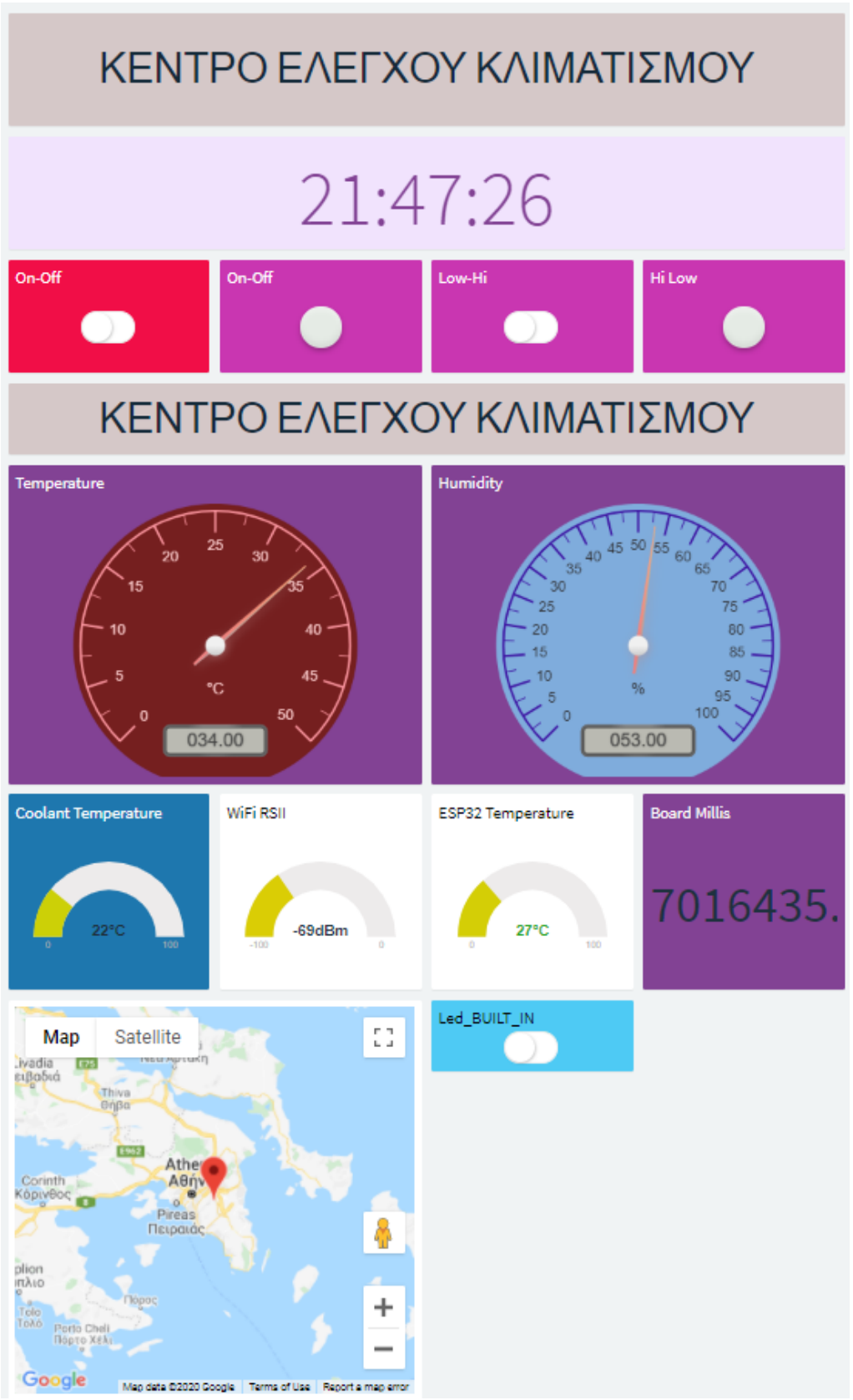
Ο διακόπτης Hi-Low αλλάζει την κατάσταση της θέρμανσης από Low σε High και αντιστρόφως και στο Led δεξιά του βλέπουμε αυτή την κατάσταση.

Οι διακόπτες με την εκκίνηση του Dashboard παίρνουν αυτόματα τις πραγματικές καταστάσεις των συσκευών που ελέγχουν. Αν δηλαδή η συσκευή εκείνη τη στιγμή είναι σε λειτουργία τότε ο διακόπτης θα μετακινηθεί αυτόματα στη θέση ON.

Στα δύο όργανα στη μέση του πάνελ βλέπουμε την θερμοκρασία και υγρασία του εσωτερικού χώρου.

Ενώ ακριβώς από κάτω στα όργανα βλέπουμε τη θερμοκρασία του ψυκτικού υγρού, το σήμα WiFi, τη θερμοκρασία της συσκευής και έναν αριθμό που μεταβάλλεται μετά από κάθε refresh τιμών.

Τέλος στο κάτω μέρος προστέθηκε και ένας διακόπτης για να κάνουμε test λειτουργίας την συσκευή



Εικόνα Α.2.18 Το dashboard για το έργο που δημιουργήθηκε στην πλατφόρμας Thinger.io

7. Deploying the Thinger.io

Σε αυτή την ενότητα θα παρουσιαστεί όλη η διαδικασία μιας εγκατάστασης του server Thinger.io σε ένα τοπικό υπολογιστή..

Η πλατφόρμα όπως αναφέραμε παραπάνω είναι Open Source και υποστηρίζει το server deploying.

Για το deploy επιλέχθηκε μια κεντρική μονάδα με τα παρακάτω χαρακτηριστικά.

- ***Intel core i5 - τετραπύρηνος***
- ***8GB ram***
- ***128 SSD***
- ***Linux Ubuntu 20.04***

Τα βήματα που ακολουθούμε είναι τα εξής:

Βήμα 1:

Αρχικά γίνεται εγκατάσταση του **Docker Engine.** Ακολουθήθηκαν οι οδηγίες από τον ιστότοπο: https://docs.docker.com/engine/install/ubuntu/

Εγκατάσταση του **Docker repository**

```
$ sudo apt-get update
```

```
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common
```

Add Docker's **official GPG key**:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Επαλήθευση κλειδιού με fingerprint: 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88, ελέγχοντας τους τελευταίους οκτώ χαρακτήρες δίνοντας την παρακάτω εντολή.

```
$ sudo apt-key fingerprint 0EBFCD88
```

```
pub   rsa4096 2017-02-22 [SCEA]
      9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid           [ unknown] Docker Release (CE deb) <docker@docker.com>
sub   rsa4096 2017-02-22 [S]
```

Ακολουθεί η εγκατάσταση του stable repository:

```
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

Κάνουμε update το apt package index και εγκαθιστούμε την τελευταία έκδοση του Docker Engine

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Στη συνέχεια ελέγχουμε τις διαθέσιμες εκδόσεις του Docker Engine που περιλαμβάνονται στο repository με την παρακάτω εντολή:

```
apt-cache madison docker-ce
```

και λαμβάνουμε την παρακάτω λίστα:

```
 docker-ce | 5:19.03.12~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
 docker-ce | 5:19.03.11~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
 docker-ce | 5:19.03.10~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
 docker-ce | 5:19.03.9~3-0~ubuntu-focal | https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
```

Επιλέγουμε την τελευταία έκδοση: 5:19.03.12~3-0~ubuntu-focal
και την κάνουμε εγκατάσταση με την παρακάτω εντολή:

```
$ sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-cli=<VERSION_STRING> containerd.io
```

όπου αντικαθιστούμε το VERSION_STRING με την έκδοση:
5:19.03.12~3-0~ubuntu-focal

Ελέγχουμε αν όλα πήγαν καλά με την εντολή:

```
$ sudo docker run hello-world
```

Και περιμένουμε την απάντηση:
Hello from Docker!

Βήμα 2:
Εγκατάσταση του **Docker compose** ακολουθώντας τα βήματα από το link: [10]
https://docs.docker.com/compose/install/

Αρχικά κατεβάζουμε την stable έκδοση του Docker Compose

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.26.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Εφαρμογή εκτελέσιμων δικαιωμάτων

```
sudo chmod +x /usr/local/bin/docker-compose
```

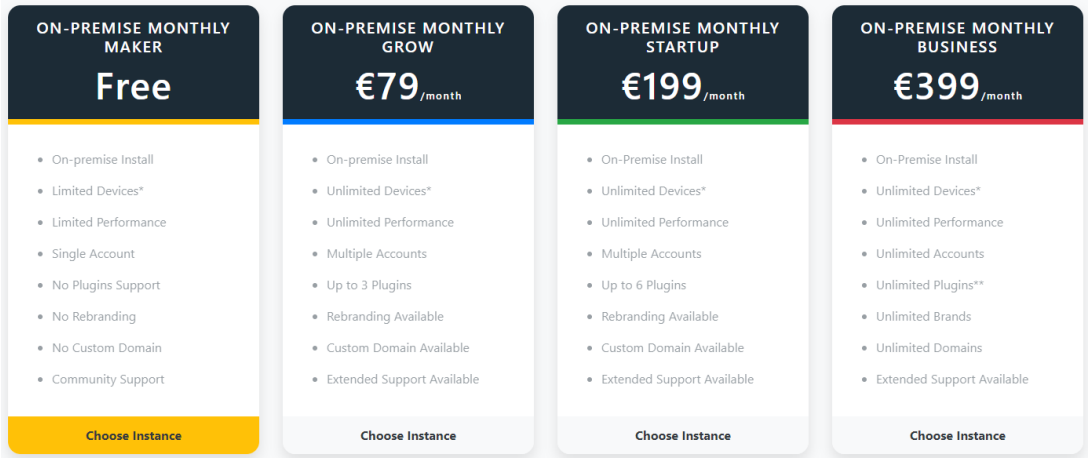
Ελέγχουμε την εγκατάσταση:

```
$ docker-compose --version

ildocker-compose version 1.26.2, build 1110ad01
```

Βήμα 3:

Στη συνέχεια πριν κατεβάσουμε το docker-compose θα πρέπει να έχουμε λάβει ένα license key.

Στον ιστότοπο:
https://pricing.thinger.io/#!/on-premise
βλέπουμε τα πακέτα που μας παρέχει το thinger.io για να κάνουμε το deploy .



Εικόνα Α.2.19 Τα διαθέσιμα πακέτα της πλατφόρμας Thinger.io

Στα πακέτα που παρέχονται παρατηρούμε ότι το free πακέτο περιλαμβάνει πολύ λίγες δυνατότητες, ενώ το αμέσως επόμενο έχει σχετικά υψηλή τιμή.

Επιλέγουμε το free πακέτο το οποίο έχει το πολύ βασικό μειονέκτημα ότι υποστηρίζει ένα χρήστη μόνο και κανένα plug-in.

Ακολουθούμε τα βήματα και αφού δώσουμε τα απαραίτητα στοιχεία ολοκληρώνουμε τον οδηγό και λαμβάνουμε στο email που δώσαμε ένα license key

Βήμα 4:

Στη συνέχεια προχωράμε στην εγκατάσταση των αρχείων του docker-compose που σχετίζονται με τα license που μας έχουν αποσταλεί.

Δίνουμε την παρακάτω εντολή αντικαθιστώντας τη λέξη LICENSE με το 96bytes κλειδί.

curl https://subscriptions.thinger.io/v1/docker-compose.yml?token={LICENSE} -o docker-compose.yml

Αφού γίνει η εγκατάσταση ελέγχουμε αν τα docker-compose αρχεία έχουν κατέβει σωστά.

cat docker-compose.yml

Θα πρέπει να λάβουμε ως απάντηση ένα αρχείο που ξεκινά όπως παρακάτω:

docker-compose.yml

```
version: '3.7'
services:

  # Thinger.io server
  thinger:
    image: thinger/server:latest
    container_name: thinger
    user: root
    volumes:
      # for controlling docker engine
      - /var/run/docker.sock:/var/run/docker.sock
      # folder for storing thinger generated data (maxmind, certificates, plugins..
      - /data/thinger:/data
    entrypoint:
      - thinger
      - -v1
      - --runpath=/data
    environment:
      - TOKEN=ce84fc9f089227a4828f66c470d9319533611ee37adc41676ba1ef5a92bd1dca0f225
    network_mode: host
    restart: always
    logging:
      driver: "json-file"
      options:
        max-size: "200k"
        max-file: "10"
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost/v1/server/healthcheck"]
      interval: 1m
      timeout: 5s
      retries: 3
      start_period: 2m
```

Εικόνα Α.2.20 Το αρχείο παραμετροποίησης του Docker

Στη συνέχεια ξεκινάμε τον Server δίνοντας την επόμενη εντολή:

```
sudo  docker-compose up -d
```

Εάν όλα πήγαν καλά μετά ίσως από μερικά λεπτά θα δούμε αρχικά μια λίστα ενημερωτικών μηνυμάτων στο τερματικό και στο τέλος θα δούμε τα παρακάτω:

Creating mongodb ... done
Creating ouroboros ... done
Creating influxd ... done
Creating thinger ... done

Βήμα 5:

Για να συνδεθούμε στο server που μολις εγκαταστήσαμε

μπορούμε μέσα από τον ίδιο υπολογιστή που εγκαταστήσαμε να δώσουμε στον browser την ip

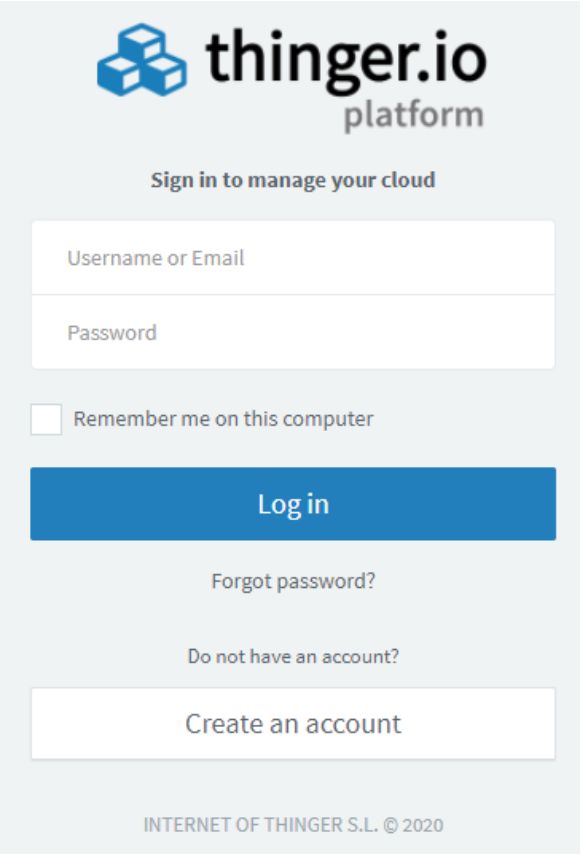
https://127.0.0.1 ή **https://localhost**

ή από κάποιον άλλο υπολογιστή ή άλλη συσκευή που είναι συνδεδεμένα στο ίδιο τοπικό δίκτυο να δώσουμε την IP του υπολογιστή που φιλοξενεί τον server Thinger.io (πχ **https://192.168.1.4**).

Ο browser και στις δύο περιπτώσεις θα απορρίψει τη σύνδεση ως μη ασφαλή και θα μας προτρέψει να χρησιμοποιήσουμε self-signed certificate. Μετά την αποδοχή θα γίνει η σύνδεση και θα δούμε τη σελίδα login (εικόνα Α.2.21)

Αρχικά θα πρέπει να δημιουργήσουμε ένα χρήστη και στη συνέχεια να εισέλθουμε στην πλατφόρμα.

Δυστυχώς το free πακέτο του Thinger.io επιτρέπει μόνο τη δημιουργία ενός χρήστη.



Εικόνα Α.2.21

8. Δημιουργία ενός project για δοκιμή της πλατφόρμας Thinger.io

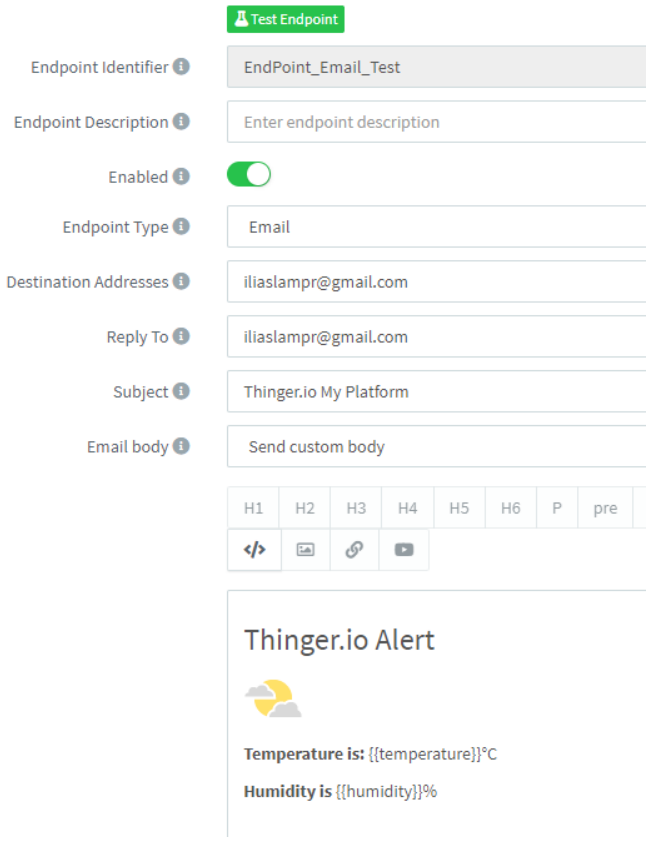
```
#define THINGER_SERVER "192.168.1.4"

#include <ThingerESP32.h>
#define USERNAME "ilias"
#define DEVICE_ID
"esp32_weather_station"
#define DEVICE_CREDENTIAL
"Ni!5Q9LNbtG7a$"

#define SSID "ILIAS NETWORK"
#define SSID_PASSWORD "6973403792"

ThingerESP32 thing(USERNAME,
DEVICE_ID, DEVICE_CREDENTIAL);
```

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);

  thing.add_wifi(SSID,
SSID_PASSWORD);

  // digital pin control example
  thing["led"] <<
digitalPin(LED_BUILTIN);

  // resource output example (i.e. reading a
sensor value)
  thing["millis"] >> outputValue(millis());

  thing["indoor_humidity"] >>
outputValue(random(100));
  thing["indoor_temperature"] >>
outputValue(random(50));
```

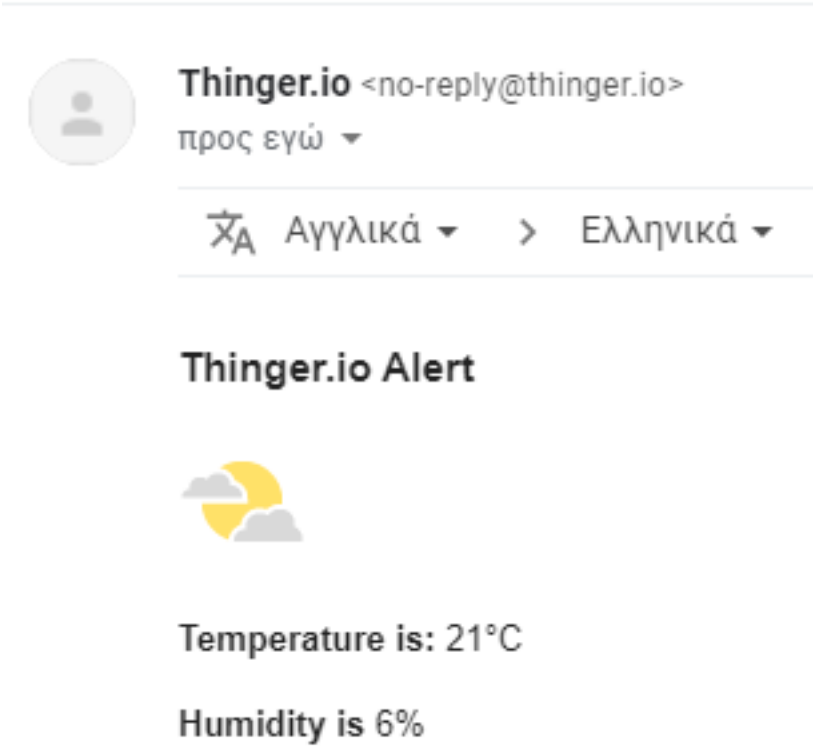
Δοκιμή αποστολής email μέσα από την πλατφόρμα

Arduino code

```
//--- Παράδειγμα EndPoint
if(millis()-lastTimeSend>60000){
   pson data;
   data["temperature"] = temperature;
   data["humidity"] = humidity;
   thing.call_endpoint("EndPoint_Email_Test", data);
   Serial.println("Send email");
   lastTimeSend=millis();
}
```

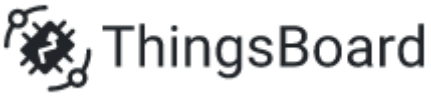


Εικόνα Α.2.22 : Δημιουργία EndPoint στην πλατφόρμα



Εικόνα Α.2.23 : Στην εικόνα βλέπουμε το email που έστειλε η πλατφόρμα μετά την ενεργοποίηση του **EndPoint**

3 Η ΠΛΑΤΦΟΡΜΑ THINGSBOARD



A. Overview

Η τρίτη πλατφόρμα που θα αναλύσουμε ως προς τα χαρακτηριστικά της είναι η Open Source πλατφόρμα Thingsboard.

Η Thingsboard Inc. είναι μια αμερικάνικη εταιρεία που ιδρύθηκε το 2016. Το RnD κέντρο της βρίσκεται στο Κίεβο της Ουκρανίας. Η εταιρία αυτή είναι και ο κύριος διαχειριστής της πλατφόρμας ανοικτού κώδικα Thingsboard.

Η πλατφόρμα Thingsboard έχει σχεδιαστεί για να προσφέρει ανάπτυξη και διαχείριση IoT έργων.

Επιτρέπει την δημιουργία υποδομής διακομιστή (server) είτε στο cloud ή σε χώρο του χρήστη.

Η πλατφόρμα Thingsboard προσφέρει τις παρακάτω δυνατότητες:

• Δυνατότητα δημιουργίας συσκευών, στοιχείων και πελατών και σχέσεων μεταξύ τους,

• Συλλογή και οπτικοποίηση δεδομένων από στοιχεία και συσκευές

• Ανάλυση εισερχόμενης τηλεμετρίας και ενεργοποίηση συναγερμών με πολύπλοκη επεξεργασία συμβάντων,

• Έλεγχος συσκευών με απομακρυσμένες κλήσεις διαδικασίας (RPC)

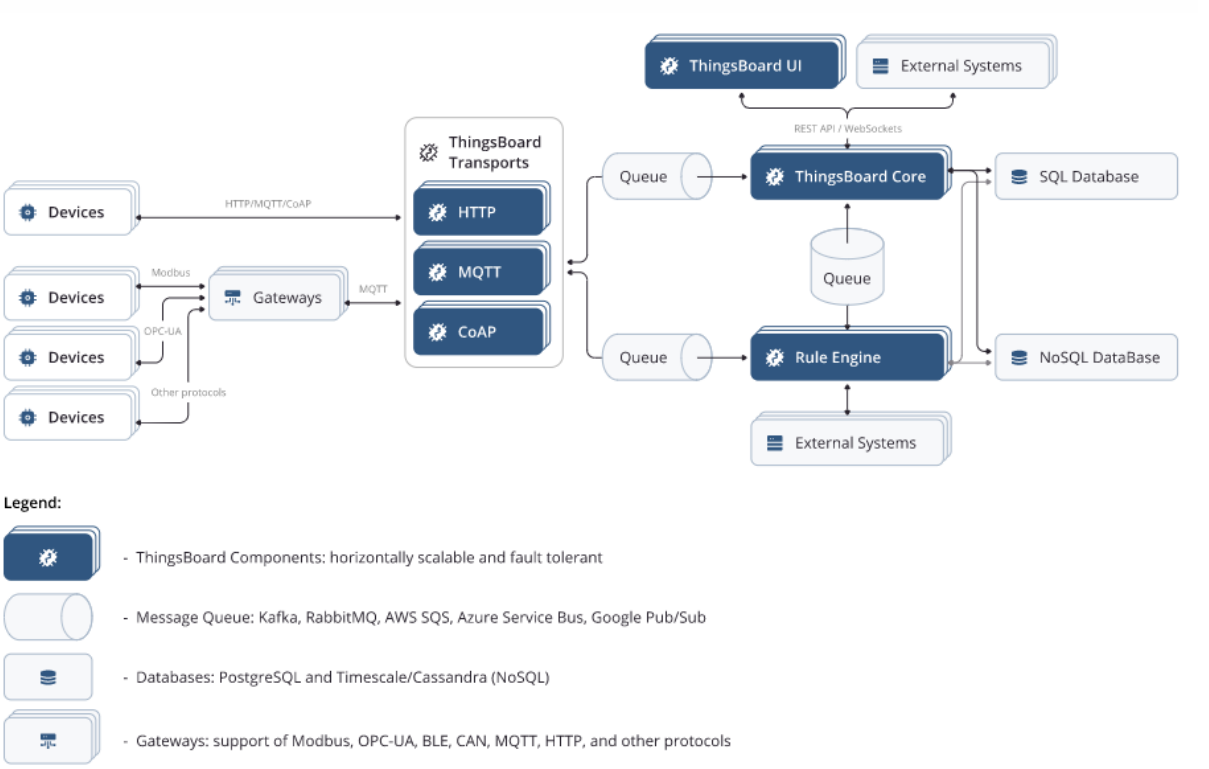
• Δημιουργία ροής εργασίας βασισμένης σε συμβάντα κύκλου ζωής συσκευής, REST API, αιτήματος RPC, κτλ,

• Σχεδιασμός δυναμικών και ανταποκρινόμενων ταμπλό για παρουσίαση τηλεμετρίας συσκευών ή στοιχείων και πληροφοριών,

• Δυνατότητα χρήσης συγκεκριμένων λειτουργιών ανά περίπτωση χρησιμοποιώντας προσαρμόσιμες αλυσίδες κανόνων,

• Προώθηση δεδομένων συσκευών σε άλλα συστήματα

και πολλές άλλες αντίστοιχες δυνατότητες. [31]



Εικόνα Α.3.1: Αρχιτεκτονική Thingsboard [29]

Στη παραπάνω εικόνα, απεικονίζεται σε γράφημα η αρχιτεκτονική της πλατφόρμας.

Τα κύρια στοιχεία της περιλαμβάνονται στα μπλε κουτάκια και είναι τα εξής:

- Μεταφορείς (Thingsboard Transports)

Για κάθε εφαρμογή ή υλικολογισμικό μιας συσκευής, το Thingsboard παρέχει APIs (Application Programming Interface) – το API μπορεί να περιγραφεί ως ένα ενδιάμεσο λογισμικό που επιτρέπει την επικοινωνία μεταξύ δύο εφαρμογών – βασισμένα σε πρωτόκολλα MQTT, HTTP και CoAP.

Υπάρχουν ξεχωριστά στοιχεία του server για κάθε API και είναι μέρος του ‘’Transport Layer’’ του Thingsboard. Το στοιχείο που αφορά το πρωτόκολλο MQTT προσφέρει επίσης τα ‘’Gateway APIs’’ με τα οποία δύναται ο χρήστης να χειρίζεται πολλαπλές συσκευές και/ή αισθητήρες.

Οι Μεταφορείς λειτουργούν ως εξής: Η συσκευή στέλνει το μήνυμα που το παραλαμβάνει ο Μεταφορέας, αναλύεται και προωθείται στην Ουρά Μηνυμάτων (Message Queue).

Η παράδοση του μηνύματος αναγνωρίζεται μόνον αφού το αντίστοιχο μήνυμα αναγνωριστεί από την Ουρά Μηνυμάτων.

- Πυρήνας (Thingsboard Core)

Ο Πυρήνας του Thingsboard είναι υπεύθυνος για τον χειρισμό κλήσεων REST API και

συνδρομών WebSocket. Επιπλέον, είναι υπεύθυνος για την αποθήκευση των πιο πρόσφατων πληροφοριών σχετικά με ενεργές συσκευές και για την παρακολούθηση της συνδεσιμότητας τους. Εδώ γίνεται και η χρήση του Συστήματος Παραγόντων (Actor System) προκειμένου να γίνει εφαρμογή παραγόντων για τις κύριες οντότητες, τους ‘’Ενοίκους’’ (Tenants) και τις ‘’Συσκευές’’ (Devices). Στην συστάδα μπορούν να συμμετέχουν και κόμβοι της πλατφόρμας, όπου κάθε κόμβος είναι υπεύθυνος για ορισμένα μέρη των εισερχόμενων μηνυμάτων.

- Μηχανή Κανόνων (Thingsboard Rule Engine)

Η Μηχανή Κανόνων του Thingsboard είναι ίσως το σημαντικότερο μέρος του συστήματος και είναι υπεύθυνο για την επεξεργασία των εισερχόμενων μηνυμάτων.

Όπως και ο Πυρήνας, χρησιμοποιεί το Actor System για να εφαρμόσει παράγοντες για τις κύριες οντότητες: αλυσίδες κανόνων (rule chains) και κόμβοι κανόνων (rule nodes).

Επίσης όπως και ο Πυρήνας, οι κόμβοι της Μηχανής Κανόνων μπορούν να συμπεριλαμβάνονται στην συστάδα όπου κάθε κόμβος είναι υπεύθυνος για ορισμένα μέρη των εισερχόμενων μηνυμάτων.

Η μεγάλη ποικιλία στην χρήση της πλατφόρμας Thingsboard οφείλεται στις πολλαπλές στρατηγικές διαθέσιμες από την Μηχανή Κανόνων, για τον έλεγχο της σειράς ή της επεξεργασίας μηνυμάτων και των κριτηρίων για την αναγνώριση τους.

Η Μηχανή Κανόνων έχει δύο λειτουργίες: κοινή και απομονωμένη. Στην κοινή επεξεργάζεται μηνύματα που ανήκουν σε πολλαπλούς tenants, ενώ στην απομονωμένη για έναν συγκεκριμένο.

- Διαδικτυακό Περιβάλλον Χρήστη (Thingsboard Web UI)

Το Thingsboard παρέχει ένα ελαφρύ στοιχείο σχεδιασμένο με Express.js Framework για να φιλοξενεί στατικό web UI περιεχόμενο. Εκεί περιέχεται και δέσμη εφαρμογών. Μόλις η εφαρμογή φορτωθεί, αρχίζει να χρησιμοποιεί το REST API και το WebSockets API που παρέχονται από τον Πυρήνα [31]

Με το ThingsBoard, μπορούν να εκτελεστούν ενέργειες για :

• Δ**ήλωση** συσκευών, στοιχείων και πελατών και καθορισμός των μεταξύ τους σχέσεων.

• **Συλλογή και οπτικοποίηση δεδομένων** από συσκευές και αντικείμενα.

• **Ανάλυση της εισερχόμενης τηλεμετρίας** και ενεργοποίηση των συναγερμών με σύνθετη επεξεργασία συμβάντων.

• ΄**Ελεγχος των συσκευών** χρησιμοποιώντας κλήσεις απομακρυσμένης διαδικασίας.

• **Κατασκευή ροών εργασίας** βάση συμβάντος κύκλου ζωής της κάθε συσκευής.

• **Σχεδίαση δυναμικών και ανταποκρινόμενων πινάκων ελέγχου** για την λειτουργική παρουσίαση της τηλεμετρίας των συσκευών ή αντικειμένων με τις εκάστοτε πληροφορίες για τους πελάτες.

• **Ενεργοποίηση λειτουργιών** για συγκεκριμένες περιπτώσεις χρησιμοποιώντας προσαρμοσμένες αλυσίδες κανόνων.

• **Αποστολή των συλλεγόμενων δεδομένων** σε άλλα συστήματα. [30]

Από πλευρά Αρχιτεκτονικής το ThingsBoard είναι σχεδιασμένο να είναι :

• **Κλιμακωμένο** : οριζόντια κλιμάκωση πλατφόρμας, που βασίζεται σε κορυφαίες

τεχνολογίες ανοιχτού κώδικα.

• **Ανθεκτικό σε σφάλματα** : κανένα σημείο αποτυχίας, κάθε κόμβος στο σύμπλεγμα είναι πανομοιότυπος.

• **Ισχυρό και αποδοτικό** : ένας μοναδικός κόμβος διακομιστή μπορεί να χειριστεί δεκάδες ή ακόμα και εκατοντάδες χιλιάδες συσκευές ανάλογα με τη χρήση. Ένα ThingsBoard cluster μπορεί να χειριστεί εκατομμύρια συσκευές.

• **Προσαρμόσιμο** : η προσθήκη νέων λειτουργιών είναι εύκολη με τα προσαρμοσμένα widget και τους κόμβους μηχανών κανόνων.

• **Ανθεκτικό** : δεν χάνονται ποτέ τα δεδομένα.

Επισκόπηση οντοτήτων

Το ThingsBoard παρέχει τη διεπαφή χρήστη και τα REST APIs για την παροχή και τη διαχείριση πολλών τύπων οντοτήτων και των σχέσεών τους στην IoT εφαρμογή.

Οι υποστηριζόμενες οντότητες είναι :

• **Tenants** : κάθε Tenant μπορεί να αντιμετωπίζετε και ως ξεχωριστή επιχείρηση, άτομο ή οργανισμό που κατέχει ή παράγει συσκευές και αντικείμενα. Ο Tenant μπορεί να έχει πολλαπλούς χρήστες, διαχειριστές και εκατομμύρια πελάτες.

• **Πελάτες** : ο πελάτης είναι επίσης ξεχωριστή επιχείρηση, άτομο ή οργανισμός που αγοράζει ή χρησιμοποιεί συσκευές. Ο πελάτης μπορεί να έχει πολλούς χρήστες και εκατομμύρια συσκευές.

• **Χρήστες** : οι χρήστες μπορούν να περιηγούνται σε πίνακες ελέγχου και να διαχειρίζονται οντότητες.

• **Συσκευές** : βασικές οντότητες IoT που μπορούν να παράγουν δεδομένα τηλεμετρίας και να χειρίζονται εντολές remote procedure calls (RPC). Για παράδειγμα αισθητήρες ή ενεργοποιητές.

• **Αντικείμενα** : αφηρημένες οντότητες IoT που ενδέχεται να σχετίζονται με άλλες συσκευές και περιουσιακά στοιχεία. Για παράδειγμα εργοστάσιο, πεδίο ή όχημα.

• **Συναγερμοί** : συμβάντα που εντοπίζουν Cητήματα από τα αντικείμενα, τις συσκευές. [30]

ΠΑΡΑΡΤΗΜΑ Β

Ο ΚΩΔΙΚΑΣ ΣΕ ΓΛΩΣΣΑ JAVA ΤΗΣ IoT GATEWAY

Class MainClass

Η κλάση αυτή είναι η πρώτη κλάση της οποίας ο κώδικας εκτελείται με την εκτέλεση της εφαρμογής.

Περιέχει το γραφικό περιβάλλον αλλά ταυτόχρονα περιέχει και τον κώδικα που ελέγχει τα μηνύματα που έρχονται από τον κάθε broker και αποστέλλονται στον άλλον αφού πρωτα σταλούν σε άλλες κλάσεις για να υποστούν επεξεργασία.

```
package main_package;

import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.net.URL;
import java.util.ArrayList;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSeparator;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.border.TitledBorder;

import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.json.JSONException;
import org.json.JSONObject;

import main_package.Converter.ConverterItem;
import main_package.MqttClientController.OnMqttClientReceiveCallbackInterface;

/* ------------------- INFO ----------------------------------
 * Updated : 20/03/2021
```

```
 * Για compile τα dependencies  Run>Run as...> maven build...> clean verify
 */

//================================================== MainClass
public class MainClass {

  JsonEditor windowJsonEditor = null;
  private static final String APP_NAME = "IoT Gateway";

  // -------- GUI Variables --------------------------------
  // -------------------------------------------------------
  private JFrame frame;
  private JTextField textField_url1;
  private JTextField textField_username1;
  private JTextField textField_password1;
  private JTextField textField_clientID1;
  private JTextField textField_timeout1;
  private JTextField textField_url2;
  private JTextField textField_username2;
  private JTextField textField_password2;
  private JTextField textField_clientID_2;
  private JTextField textField_timeout2;

  public static MqttClientController mqtt_client_controller1 = null;
  public static MqttClientController mqtt_client_controller2 = null;

  public static ArrayList<ConverterItem> broker1_settings = null;
  public static ArrayList<ConverterItem> broker2_settings = null;

  public static Terminal windowTerminal1 = null;
  public static Terminal windowTerminal2 = null;

  final String broker1SettingsFilename = "broker1_settings.json";
  final String broker2SettingsFilename = "broker2_settings.json";

  class BrokerSettings {
        String brokerURL = "";
        String username = "";
        String password = "";
        String clientID = "1234567890";
        int protocol = 0;
        boolean cleanSession = true;
        int timeout = 10;
        int keepAlive = 60;
  }

  BrokerSettings broker1 = null;
  BrokerSettings broker2 = null;

  // ========================================== onReceiveFromBroker1 callback
  private OnMqttClientReceiveCallbackInterface onReceiveFromBroker1 = new
OnMqttClientReceiveCallbackInterface() {

@Override
public void onReceive(String topic, MqttMessage message) {
  mqtt_client_controller1.writeToOutput("----------------------");
  mqtt_client_controller1.writeToOutput("Received message");
  mqtt_client_controller1.writeToOutput("Topic: " + topic);
```

```
        mqtt_client_controller1.writeToOutput("Message: " + message);

        if (broker1_settings == null) System.out.println("Error: Can\'t read JsonConverter.txt file ");
        if (mqtt_client_controller2 == null)
                System.out.println("Error: Broker2 is not running");

        if ((broker1_settings == null) || (mqtt_client_controller2 == null))       return;

        else {
                ArrayList<MqttClientController.PubMessage> pubList =
                        Converter.getConvertedPubMessageList(topic,
                                message.toString(), broker1_settings);
                  for (int i = 0; i < pubList.size(); i++) {
                  MqttClientController.PubMessage pubMessage = pubList.get(i);
                  if (pubMessage != null) {
                                int qos = pubMessage.qos;
                                if ((qos < 0) || (qos > 2))
                                qos = message.getQos();
                                mqtt_client_controller1.writeToOutput("----------------------");
                                mqtt_client_controller1.writeToOutput("Publish to broker 2");
                                String newPubTopic = Converter.getConvertedTopic(topic, pubMessage.topic,
                                        message.toString());
                                mqtt_client_controller1.writeToOutput("Topic: " + newPubTopic);
                                mqtt_client_controller1.writeToOutput("QoS: " + pubMessage.qos);
                                mqtt_client_controller1.writeToOutput("Retained: " + (pubMessage.retained == 1));
                                mqtt_client_controller1.writeToOutput("Message: " + pubMessage.message);

                                System.out.println("newPubTopic=" + newPubTopic);

                                if (pubMessage != null) {
                                        mqtt_client_controller2.mqttPublish(newPubTopic,
                                                pubMessage.qos, pubMessage.retained == 1,
                                                  pubMessage.message);
                                }
                  } else      System.out.println("Error: Can\'t retrieve data from file JsonConverter.txt");
        }

        }
        }
};


        // ========================================================
        // onReceiveFromBroker1 callback
        private OnMqttClientReceiveCallbackInterface onReceiveFromBroker2 = new
OnMqttClientReceiveCallbackInterface() {

        @Override
        public void onReceive(String topic, MqttMessage message) {
        mqtt_client_controller2.writeToOutput("---------------------------------");
        mqtt_client_controller2.writeToOutput("Received message");
        mqtt_client_controller2.writeToOutput("Topic: " + topic);
        mqtt_client_controller2.writeToOutput("Message: " + message);

        if (broker2_settings == null) System.out.println("Error: Can\'t read JsonConverter.txt file ");
        if (mqtt_client_controller1 == null)       System.out.println("Error: Broker1 is not running");

        if ((broker2_settings == null) || (mqtt_client_controller1 == null))       return;
```

```
    else {
            ArrayList<MqttClientController.PubMessage> pubList =
                    Converter.getConvertedPubMessageList(topic,message.toString(), broker2_settings);
            for (int i = 0; i < pubList.size(); i++) {
                    MqttClientController.PubMessage pubMessage = pubList.get(i);
            if (pubMessage != null) {
                    int qos = pubMessage.qos;
                    if ((qos < 0) || (qos > 2))
                    qos = message.getQos();

                    mqtt_client_controller2.writeToOutput("---------------------------------");
                    mqtt_client_controller2.writeToOutput("Publish to broker 1");
                    String newPubTopic = Converter.getConvertedTopic(topic, pubMessage.topic,
                            message.toString());
                    mqtt_client_controller2.writeToOutput("Topic: " + newPubTopic);
                    mqtt_client_controller2.writeToOutput("QoS: " + pubMessage.qos);
                    mqtt_client_controller2.writeToOutput("Retained: " + (pubMessage.retained == 1));
                    mqtt_client_controller2.writeToOutput("Message: " + pubMessage.message);

                    System.out.println("newPubTopic=" + newPubTopic);

                    if (pubMessage != null) {
                            mqtt_client_controller1.mqttPublish(newPubTopic, pubMessage.qos,
                                    pubMessage.retained == 1,
                                      pubMessage.message);
                    }
            } else
                    System.out.println("Error: Can\'t retrieve data from file JsonConverter.txt");
            }

    }
    }
};

    // ======================================================================== Main
    // ========================================================================
    // ========================================================================
    public static void main(String[] args) {

            EventQueue.invokeLater(new Runnable() {
                    public void run() {
                            try {
                                    MainClass window = new MainClass();
                                    window.frame.setVisible(true);
                            } catch (Exception e) {
                                    e.printStackTrace();
                            }
                    }
            });
    }

    // ========================================================================
    // main_class
    public MainClass() {
            broker1 = loadBrokerSettings(broker1SettingsFilename);
            broker2 = loadBrokerSettings(broker2SettingsFilename);
            initialize();
```

```
		mqtt_client_controller1 = new MqttClientController(onReceiveFromBroker1);
		mqtt_client_controller2 = new MqttClientController(onReceiveFromBroker2);

	}

	// ======================================================================
	// initialize GUI
	// ======================================================================
	// ======================================================================
	// ======================================================================
	// ======================================================================
	private void initialize() {
		frame = new JFrame();
		frame.setBackground(new Color(253, 245, 230));
		frame.setResizable(false);
		frame.getContentPane().setBackground(new Color(245, 245, 245));
		frame.setBounds(100, 0, 700, 500);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.getContentPane().setLayout(null);
		URL url = getClass().getResource("/images/iot_stress_icon_512.png");
		frame.setIconImage(new ImageIcon(url).getImage());

		JPanel panel = new JPanel();
		panel.setBounds(0, 0, 694, 36);

		frame.getContentPane().add(panel);
		URL url1 = getClass().getResource("/images/gt_iot_icon.png");
		ImageIcon imageIcon = new ImageIcon(url1);
		Image image = imageIcon.getImage(); // transform it
		Image newimg = image.getScaledInstance(700, 36, java.awt.Image.SCALE_SMOOTH); // scale it the
smooth way
		imageIcon = new ImageIcon(newimg); // transform it back
		JLabel label = new JLabel(new ImageIcon(MainClass.class.getResource("/images/iot_gateway.png")));
		label.setBounds(0, 0, 694, 36);// x axis, y axis, width, height
		label.setVisible(true);
		panel.setLayout(null);
		panel.add(label);

		// ------ Broker 1 settings ----------------------
		// -----------------------------------------------
		// -----------------------------------------------
		// -----------------------------------------------
		JPanel panel_serverSettings1 = new JPanel();
		panel_serverSettings1.setBounds(10, 47, 323, 318);
		panel_serverSettings1.setBorder(
				new TitledBorder(null, "Broker 1 settings", TitledBorder.LEADING,
TitledBorder.TOP, null, null));
		panel_serverSettings1.setToolTipText("");
		frame.getContentPane().add(panel_serverSettings1);
		panel_serverSettings1.setLayout(null);

		JLabel lblNewLabel_url1 = new JLabel("Broker URL");
		lblNewLabel_url1.setHorizontalAlignment(SwingConstants.RIGHT);
		lblNewLabel_url1.setBounds(10, 34, 69, 20);
		panel_serverSettings1.add(lblNewLabel_url1);

		textField_url1 = new JTextField();
		textField_url1.setBounds(83, 34, 225, 20);
```

```
textField_url1.setHorizontalAlignment(SwingConstants.LEFT);
textField_url1.setText(broker1.brokerURL);
panel_serverSettings1.add(textField_url1);
textField_url1.setColumns(10);

JLabel lblNewLabel_un1 = new JLabel("Username");
lblNewLabel_un1.setHorizontalAlignment(SwingConstants.RIGHT);
lblNewLabel_un1.setBounds(10, 59, 69, 20);
panel_serverSettings1.add(lblNewLabel_un1);

textField_username1 = new JTextField();
textField_username1.setBounds(83, 58, 225, 22);
textField_username1.setText(broker1.username);
panel_serverSettings1.add(textField_username1);
textField_username1.setColumns(10);

textField_password1 = new JTextField();
textField_password1.setColumns(10);
textField_password1.setBounds(83, 84, 225, 22);
textField_password1.setText(broker1.password);
panel_serverSettings1.add(textField_password1);

JLabel lblNewLabel_ps1 = new JLabel("Password");
lblNewLabel_ps1.setHorizontalAlignment(SwingConstants.RIGHT);
lblNewLabel_ps1.setBounds(10, 86, 69, 20);
panel_serverSettings1.add(lblNewLabel_ps1);

textField_clientID1 = new JTextField();
textField_clientID1.setColumns(10);
textField_clientID1.setBounds(83, 110, 225, 22);
textField_clientID1.setText(broker1.clientID);
panel_serverSettings1.add(textField_clientID1);

JLabel lblNewLabel_clientID1 = new JLabel("Client ID");
lblNewLabel_clientID1.setHorizontalAlignment(SwingConstants.RIGHT);
lblNewLabel_clientID1.setBounds(10, 110, 69, 22);
panel_serverSettings1.add(lblNewLabel_clientID1);

JCheckBox CheckBox_cleanSeason1 = new JCheckBox("Clean Session");
CheckBox_cleanSeason1.setBounds(83, 170, 226, 23);
CheckBox_cleanSeason1.setSelected(broker1.cleanSession);
panel_serverSettings1.add(CheckBox_cleanSeason1);

textField_timeout1 = new JTextField();
textField_timeout1.setText(broker1.timeout + "");
textField_timeout1.setColumns(10);
textField_timeout1.setBounds(83, 200, 58, 22);
textField_timeout1.addKeyListener(new java.awt.event.KeyAdapter() {
        @Override
        public void keyReleased(java.awt.event.KeyEvent evt) {
        }

        @Override
        public void keyPressed(KeyEvent e) {
        }

        @Override
        public void keyTyped(KeyEvent e) {
```

```
                        String filterStr = "0123456789";
                        char c = (char) e.getKeyChar();
                        if (filterStr.indexOf(c) < 0) {
                                e.consume();
                        }
                }
});
panel_serverSettings1.add(textField_timeout1);

JLabel lblNewLabel_timeout1 = new JLabel("Cnt Timeout");
lblNewLabel_timeout1.setHorizontalAlignment(SwingConstants.RIGHT);
lblNewLabel_timeout1.setBounds(10, 200, 69, 20);
panel_serverSettings1.add(lblNewLabel_timeout1);

JLabel lblNewLabel_secs1 = new JLabel("secs");
lblNewLabel_secs1.setHorizontalAlignment(SwingConstants.LEFT);
lblNewLabel_secs1.setBounds(145, 200, 77, 22);
panel_serverSettings1.add(lblNewLabel_secs1);

JSeparator separator1 = new JSeparator();
separator1.setBounds(10, 252, 298, 2);
panel_serverSettings1.add(separator1);

JButton Button_connect1 = new JButton("Connect");
Button_connect1.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
        mqtt_client_controller1.mqttDisconnect();
        broker1.brokerURL = textField_url1.getText().toString().trim();
        broker1.username = textField_username1.getText().toString().trim();
        broker1.password = textField_password1.getText().toString().trim();
        broker1.clientID = textField_clientID1.getText().toString().trim();
        int timeout = 10;
        try {
                timeout = Integer.parseInt(textField_timeout1.getText().toString().trim());
                } catch (NumberFormatException nfe) {
                }
                broker1.timeout = timeout;
                broker1.cleanSession = CheckBox_cleanSeason1.isSelected();
                if (broker1.brokerURL.length() > 0) {
                // mqttConnect(bUrl,bUn,bPs,bClientID,timeout,cleanSession);
                mqtt_client_controller1.showTerminal();
                broker1_settings = Converter.getConverterItemList(1, "");
                mqtt_client_controller1.mqttConnect(broker1.brokerURL, broker1.username,
                        broker1.password,     broker1.clientID, broker1.timeout,
                          broker1.cleanSession,Converter.getTopicList(broker1_settings));

                }
                saveBrokerSettings(broker1, broker1SettingsFilename);
        }
});
Button_connect1.setBackground(new Color(220, 220, 220));
Button_connect1.setBounds(10, 265, 136, 30);
panel_serverSettings1.add(Button_connect1);

JButton Button_disconnect1 = new JButton("Disconnect");
Button_disconnect1.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                mqtt_client_controller1.mqttDisconnect();
```

```
            }
});
Button_disconnect1.setBackground(new Color(220, 220, 220));
Button_disconnect1.setBounds(179, 265, 129, 30);
panel_serverSettings1.add(Button_disconnect1);

String[] protocolList = { "MQTT", "MQTTS /TLS" };
JComboBox comboBox = new JComboBox(protocolList);
comboBox.setBounds(83, 143, 225, 20);
comboBox.setMaximumSize(comboBox.getPreferredSize()); // added code
comboBox.setAlignmentX(Component.CENTER_ALIGNMENT);// added code
panel_serverSettings1.add(comboBox);

JLabel label_protocol = new JLabel("Protocol");
label_protocol.setHorizontalAlignment(SwingConstants.RIGHT);
label_protocol.setBounds(10, 143, 69, 22);
panel_serverSettings1.add(label_protocol);

// ------ Broker 2 settings ---------------------
// -----------------------------------------------
// -----------------------------------------------
// -----------------------------------------------

JPanel panel_serverSettings_2 = new JPanel();
panel_serverSettings_2.setLayout(null);
panel_serverSettings_2.setToolTipText("");
panel_serverSettings_2.setBorder(
            new TitledBorder(null, "Broker 2 settings", TitledBorder.LEADING, TitledBorder.TOP,
                    null, null));
panel_serverSettings_2.setBounds(351, 47, 323, 318);
frame.getContentPane().add(panel_serverSettings_2);

JLabel lblNewLabel_url_2 = new JLabel("Broker URL");
lblNewLabel_url_2.setHorizontalAlignment(SwingConstants.RIGHT);
lblNewLabel_url_2.setBounds(10, 34, 69, 20);
panel_serverSettings_2.add(lblNewLabel_url_2);

textField_url2 = new JTextField();
textField_url2.setText(broker2.brokerURL);
textField_url2.setHorizontalAlignment(SwingConstants.LEFT);
textField_url2.setColumns(10);
textField_url2.setBounds(83, 34, 225, 20);
panel_serverSettings_2.add(textField_url2);

JLabel lblNewLabel_un_2 = new JLabel("Username");
lblNewLabel_un_2.setHorizontalAlignment(SwingConstants.RIGHT);
lblNewLabel_un_2.setBounds(10, 59, 69, 20);
panel_serverSettings_2.add(lblNewLabel_un_2);

textField_username2 = new JTextField();
textField_username2.setText(broker2.username);
textField_username2.setColumns(10);
textField_username2.setBounds(83, 58, 225, 22);
panel_serverSettings_2.add(textField_username2);

textField_password2 = new JTextField();
textField_password2.setText(broker2.password);
textField_password2.setColumns(10);
```

```
        textField_password2.setBounds(83, 84, 225, 22);
        panel_serverSettings_2.add(textField_password2);

        JLabel lblNewLabel_ps_2 = new JLabel("Password");
        lblNewLabel_ps_2.setHorizontalAlignment(SwingConstants.RIGHT);
        lblNewLabel_ps_2.setBounds(10, 86, 69, 20);
        panel_serverSettings_2.add(lblNewLabel_ps_2);

        textField_clientID_2 = new JTextField();
        textField_clientID_2.setText(broker2.clientID);
        textField_clientID_2.setColumns(10);
        textField_clientID_2.setBounds(83, 110, 225, 22);
        panel_serverSettings_2.add(textField_clientID_2);

        JLabel lblNewLabel_clientID_2 = new JLabel("Client ID");
        lblNewLabel_clientID_2.setHorizontalAlignment(SwingConstants.RIGHT);
        lblNewLabel_clientID_2.setBounds(10, 110, 69, 22);
        panel_serverSettings_2.add(lblNewLabel_clientID_2);

        JCheckBox checkBox_cleanSeason2 = new JCheckBox("Clean Session");
        checkBox_cleanSeason2.setSelected(broker2.cleanSession);
        checkBox_cleanSeason2.setBounds(83, 170, 226, 23);
        panel_serverSettings_2.add(checkBox_cleanSeason2);

        textField_timeout2 = new JTextField();
        textField_timeout2.setText("" + broker2.timeout);
        textField_timeout2.setColumns(10);
        textField_timeout2.setBounds(83, 200, 58, 22);
        panel_serverSettings_2.add(textField_timeout2);

        JLabel lblNewLabel_timeout_2 = new JLabel("Cnt Timeout");
        lblNewLabel_timeout_2.setHorizontalAlignment(SwingConstants.RIGHT);
        lblNewLabel_timeout_2.setBounds(10, 200, 69, 20);
        panel_serverSettings_2.add(lblNewLabel_timeout_2);

        JLabel lblNewLabel_secs_1 = new JLabel("secs");
        lblNewLabel_secs_1.setHorizontalAlignment(SwingConstants.LEFT);
        lblNewLabel_secs_1.setBounds(145, 200, 77, 22);
        panel_serverSettings_2.add(lblNewLabel_secs_1);

        JSeparator separator_2 = new JSeparator();
        separator_2.setBounds(10, 252, 298, 2);
        panel_serverSettings_2.add(separator_2);

        JButton button_connect_2 = new JButton("Connect");
        button_connect_2.setBackground(new Color(220, 220, 220));
        button_connect_2.setBounds(10, 265, 136, 30);
        panel_serverSettings_2.add(button_connect_2);
        button_connect_2.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {
        mqtt_client_controller2.mqttDisconnect();
        broker2.brokerURL = textField_url2.getText().toString().trim();
        broker2.username = textField_username2.getText().toString().trim();
        broker2.password = textField_password2.getText().toString().trim();
        broker2.clientID = textField_clientID_2.getText().toString().trim();
        int timeout = 10;
        try {
```

```
		timeout = Integer.parseInt(textField_timeout2.getText().toString().trim());
		} catch (NumberFormatException nfe) {
		}
		broker2.timeout = timeout;
		broker2.cleanSession = checkBox_cleanSeason2.isSelected();
		if (broker2.brokerURL.length() > 0) {
			// mqttConnect(bUrl,bUn,bPs,bClientID,timeout,cleanSession);
			mqtt_client_controller2.showTerminal();
			broker2_settings = Converter.getConverterItemList(2, "");
			mqtt_client_controller2.mqttConnect(broker2.brokerURL, broker2.username,
				broker2.password,	broker2.clientID, broker2.timeout,
					broker2.cleanSession,
					  Converter.getTopicList(broker2_settings));

			}
			saveBrokerSettings(broker2, broker2SettingsFilename);
		}
});

JButton button_disconnect_2 = new JButton("Disconnect");
button_disconnect_2.setBackground(new Color(220, 220, 220));
button_disconnect_2.setBounds(179, 265, 129, 30);
button_disconnect_2.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent e) {
			mqtt_client_controller2.mqttDisconnect();
		}
});
panel_serverSettings_2.add(button_disconnect_2);

JComboBox<Object> comboBox_1 = new JComboBox<Object>(protocolList);
comboBox_1.setMaximumSize(new Dimension(84, 20));
comboBox_1.setAlignmentX(0.5f);
comboBox_1.setBounds(83, 143, 225, 20);
panel_serverSettings_2.add(comboBox_1);

JLabel label_protocol_1 = new JLabel("Protocol");
label_protocol_1.setHorizontalAlignment(SwingConstants.RIGHT);
label_protocol_1.setBounds(10, 143, 69, 22);
panel_serverSettings_2.add(label_protocol_1);

JButton btnGatewaySettings = new JButton("Gateway Settings");
btnGatewaySettings.setBackground(new Color(250, 235, 215));
btnGatewaySettings.setBounds(274, 376, 136, 30);
btnGatewaySettings.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent e) {
			// JOptionPane.showMessageDialog(null, "java is fun");
			if (windowJsonEditor == null) {
				windowJsonEditor = new JsonEditor();
				windowJsonEditor.showWindow();
			} else
				windowJsonEditor.show();

		}
});
frame.getContentPane().add(btnGatewaySettings);

JLabel lblNewLabel = new JLabel("IoT Gateway ver 0.0.1     Created by Ilias Lamprou");
lblNewLabel.setForeground(new Color(0, 128, 128));
```

```
		lblNewLabel.setBackground(new Color(0, 0, 139));
		lblNewLabel.setBounds(227, 435, 287, 23);
		frame.getContentPane().add(lblNewLabel);

		JSeparator separator1_1 = new JSeparator();
		separator1_1.setBounds(0, 435, 674, 2);
		frame.getContentPane().add(separator1_1);

	/*	JButton btnPublish = new JButton("Publish");
		btnPublish.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {
		mqtt_client_controller2.mqttPublish("iliaslamprou/feeds/temperature", 0, true, "55.4");
		}
		});
		btnPublish.setBackground(new Color(250, 235, 215));
		btnPublish.setBounds(441, 376, 136, 30);
		frame.getContentPane().add(btnPublish);
*/
	}

	// ======================================================
	BrokerSettings loadBrokerSettings(String filename) {
		// System.out.println("========loadBrokerSettings");
		String brokerData = StorageTools.readFile(filename);
		// System.out.println("============brokerData="+brokerData);
		return getBrokerSettingsFromJson(brokerData);

	}

	// ======================================================
	void saveBrokerSettings(BrokerSettings bs, String filename) {
		String brokerData = getBrokerSettingsAsJson(bs);
		// System.out.println("========Save brokerData:"+brokerData);
		StorageTools.stringToFile(brokerData, filename);
	}

	// ======================================================
	String getBrokerSettingsAsJson(BrokerSettings bs) {
		JSONObject jsObject = new JSONObject();
		try {
			jsObject.put("brokerURL", bs.brokerURL);
			jsObject.put("username", bs.username);
			jsObject.put("password", bs.password);
			jsObject.put("clientID", bs.clientID);
			jsObject.put("protocol", bs.protocol);
			jsObject.put("cleanSession", bs.cleanSession);
			jsObject.put("timeout", bs.timeout);
			jsObject.put("keepAlive", bs.keepAlive);
		} catch (JSONException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		return jsObject.toString();
	}

	// ======================================================
	BrokerSettings getBrokerSettingsFromJson(String jsonText) {
```

```
		BrokerSettings brokerSettings = new BrokerSettings();
		try {
			JSONObject jsObject = new JSONObject(jsonText);
			brokerSettings.brokerURL = jsObject.getString("brokerURL");
			brokerSettings.username = jsObject.getString("username");
			brokerSettings.password = jsObject.getString("password");
			brokerSettings.clientID = jsObject.getString("clientID");
			brokerSettings.protocol = jsObject.getInt("protocol");
			brokerSettings.cleanSession = jsObject.getBoolean("cleanSession");
			brokerSettings.timeout = jsObject.getInt("timeout");
			brokerSettings.keepAlive = jsObject.getInt("keepAlive");
		} catch (JSONException e) {
			e.printStackTrace();
		}
		return brokerSettings;
	}
}
```

Class MqttClientController

Η κλάση MqttClientController περιέχει τον βασικό κώδικα για τη δημιουργία ενός MQTT Client και τις βασικές συναρτήσεις για σύνδεση και έλεγχο της αποστολή και λήψης μηνυμάτων.

Στη MainClass δημιουργούνται δύο αντικείμενα αυτής της κλάσης τα οποία κάνουν το καθένα σύνδεση με δυο διαφορετικούς Brokers.

Η επικοινωνία αυτών των αντικειμένων με τη MainClass γίνεται μέσω μιας callback function η οποία εισάγεται στην κλάση σαν παράμετρος.

```
package main_package;
import java.util.ArrayList;
import java.util.List;

import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.eclipse.paho.client.mqttv3.MqttPersistenceException;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;



public class MqttClientController {

  private String brokerURL = "";
  private String username= "";
  private String password= "";
  private String clientID= "1234567890";
  private MqttClient mqttClient=null;

  private Terminal windowTerminal = null;


  public static interface OnMqttClientReceiveCallbackInterface {
      void onReceive(String topic, MqttMessage message);

            }

   private OnMqttClientReceiveCallbackInterface onMqttClientReceiveCallbackInterface=null;




//=================================================================
 public static class PubMessage {
            String topic="";
            String message="";
            int qos=0;
```

```
                int retained=0;
        }

        //=================================================================
MQTT_client_controller
        public MqttClientController(OnMqttClientReceiveCallbackInterface callback) {
                //output=output_;
                onMqttClientReceiveCallbackInterface=callback;
        }


         //================================================================= writeToOutput
                public void writeToOutput(String message) {
                                if (windowTerminal!=null) {
                                        windowTerminal.println(message);
                                }
                                //else
                                        System.out.println(message);
                        }


                //=============================================================
                public void showTerminal() {
                                if (windowTerminal==null) {
                                        windowTerminal= new Terminal();
                                        windowTerminal.showWindow();
                                }
                                else windowTerminal.show();
                }

        //================================================================= mqttConnect
        public boolean mqttConnect(String brokerURL_,String username_, String password_,
                String clientID_, int timeout_, boolean cleanSession_, List<String> topicList){

                brokerURL = brokerURL_;
                username = username_;
                password=password_;

                if (windowTerminal!=null) windowTerminal.setTitle(brokerURL_);

                clientID="";
                if (clientID_.length()!=0) clientID=clientID_;
                else clientID     = "iot_tester"+System.currentTimeMillis();
                MemoryPersistence persistence = new MemoryPersistence();
            try {
                 //System.out.println("clientId="+clientID);
                 String broker = "tcp://"+brokerURL+":1883";
                 mqttClient= new MqttClient(broker,clientID, persistence);

                 MqttConnectOptions connOpts = new MqttConnectOptions();
                connOpts.setCleanSession(cleanSession_);
                 //options.setMqttVersion(MqttConnectOptions.MQTT_VERSION_3_1);
                //options.setMqttVersion(MqttConnectOptions.MQTT_VERSION_3_1_1);
                connOpts.setMqttVersion(MqttConnectOptions.MQTT_VERSION_DEFAULT);

                 if (username.length() > 0) connOpts.setUserName(username);
                 if (password.length() > 0) connOpts.setPassword(password.toCharArray());
                 connOpts.setConnectionTimeout(timeout_*1000);
```

```
                connOpts.setAutomaticReconnect(false);

                //System.out.println("Connecting to broker: "+broker);
                writeToOutput("Connecting to broker "+broker+"...");

                mqttClient.connect(connOpts);
                System.out.println("Connected");
                writeToOutput("Connected succesfully");

        mqttSubscribeList(topicList);

                mqttClient.setCallback(new MqttCallback() {
                        @Override
                                public void connectionLost(Throwable cause) {
                                    writeToOutput( "Error: connectionLost");
                                        }

                                @Override

public void messageArrived(String topic, MqttMessage message) throws Exception {
                Thread.sleep(100);
                new Thread("Tread name"){
                        public void run(){
                                System.out.println("Thread: " + getName() + " running");
                                if (onMqttClientReceiveCallbackInterface!=null)
                                        onMqttClientReceiveCallbackInterface.onReceive(topic,message);

                                    }
                            }.start();

                //              new MainClass.PublishRunnable(topic,message).run();


                }


  @Override
  public void deliveryComplete(IMqttDeliveryToken arg0) {
                try {
                        writeToOutput("DeliveryComplete: "+ arg0.getMessageId()+", "+
                        arg0.getMessage().toString());
                } catch (MqttException e) {
                        e.printStackTrace();
                }
  }
  });


   } catch(MqttException me) {
                writeToOutput("Connection error: ");
                writeToOutput("Reason:"+ me.getReasonCode());
                writeToOutput("Msg:"+ me.getMessage());
                writeToOutput("Loc:"+ me.getLocalizedMessage());
                writeToOutput("Cause:"+ me.getCause());
                writeToOutput("Excep:"+ me.getReasonCode());
            me.printStackTrace();
   }
                return false;
```

```
		}

	//================================================================== executePost
		public void mqttDisconnect() {
		if (mqttClient!=null) {
			if (mqttClient.isConnected()) {
				 try {
					mqttClient.disconnect();
					if (!mqttClient.isConnected())   {
						System.out.println("Disconnected");
						 writeToOutput("Broker disconnected");
					}
				} catch (MqttException e) {
					e.printStackTrace();
					writeToOutput("Disconnection error:");
					writeToOutput(""+e.getMessage().toString());
					}
				}
			}

		}


//================================================================== executePost
	public void mqttPublish(String topic, int qos, boolean retained, String  payload) {
		try {
			MqttMessage message = new MqttMessage(payload.getBytes());
			if ((qos<=2) & (qos>=0)) message.setQos(qos); else message.setQos(0);
			message.setRetained(retained);
			mqttClient.publish(topic, message);
	//		writeToOutput("topic="+topic+"    message="+message);
	//		writeToOutput("Send succesfully");
		} catch (MqttPersistenceException e) {
			writeToOutput("Publishing error: "+e.getMessage());
			e.printStackTrace();
		} catch (MqttException e) {
			e.printStackTrace();
			writeToOutput("Publishing error: "+e.getMessage());
				}
			}


//================================================================== executePost
				public void mqttSubscribe(String topic) {
				try {
					mqttClient.subscribe(topic);
					writeToOutput("Subscribed topic: "+topic);
				} catch (MqttException e) {
					writeToOutput("Error subscribed topic: "+topic);
					writeToOutput(e.getMessage().toString());
					e.printStackTrace();
				}
				}

//================================================================== executePost
				public void mqttSubscribeList(List<String> topicList) {
				if (topicList==null) return;
```

```
                                        for (int i=0;i<topicList.size();i++) {
                                                String topic = topicList.get(i);
                                                try {
                                                mqttClient.subscribe(topic);
                                                writeToOutput("Subscribed topic: "+topic);
                                        } catch (MqttException e) {
                                                writeToOutput("Error subscribed topic: "+topic);
                                                writeToOutput(e.getMessage().toString());
                                                e.printStackTrace();
                                        }
                                        }



                                        }

//============================================================== executePost
                                        public void mqttUnsubscribe(String topic) {
                                        try {
                                                mqttClient.unsubscribe(topic);
                                                writeToOutput("Unsubscribed topic: "+topic);
                                        } catch (MqttException e) {
                                                writeToOutput("Error unsubscribed topic: "+topic);
                                                writeToOutput(e.getMessage().toString());
                                                e.printStackTrace();
                                        }

                                        }

        }
```

Class StorageTools

Η κλάση αυτή περιέχει δύο συναρτήσεις που χρησιμοποιούνται για το Load/Save των αρχεων broker1_settings.json και broker2_settings.json που περιέχουν τις παραμέτρους που ορίζει ο χρήστης για τους Brokers καθώς και για το αρχείο μορφοποίησης jsonConverter.txt το οποίο περιέχει ένα Json αντικείμενο με τις μορφοποιήσεις που πρέπει να γίνονται στα μηνύματα από τον ένα broker στον άλλον.

Ακολουθεί ο κώδικας της κλάσης StorageTools

```
package main_package;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class StorageTools {

  //============================================================
  public static String stringToFile( String text, String fileName ){
  try{
```

```
    File file = new File( fileName );
    if ( ! file.exists( ) ){
       file.createNewFile( );
    }
    FileWriter fw = new FileWriter( file.getAbsoluteFile( ) );
    BufferedWriter bw = new BufferedWriter( fw );
    bw.write( text );
    bw.close( );
    return "OK"+"\n"+file.getAbsolutePath();
  }
  catch( IOException e )
  {
  System.out.println("Error: " + e);
  e.printStackTrace( );
  return e.getMessage().toString();
  }
 } //End method stringToFile


//=============================================================

  public static String readFile(String filename){
    String content = "";
    File file = new File(filename); // For example, foo.txt
    System.out.println("file: "+file.getAbsolutePath());
    FileReader reader = null;
    try {
       reader = new FileReader(file);
       char[] chars = new char[(int) file.length()];
       reader.read(chars);
       content = new String(chars);
       reader.close();
    } catch (IOException e) {
       e.printStackTrace();
       System.out.println(e.getMessage().toString());

    } finally {
       if(reader != null){
          try {
                                        reader.close();
                                    } catch (IOException e) {
                                        e.printStackTrace();
                                        System.out.println(e.getMessage().toString());
                                    }
       }
    }
    return content;
  }

}
```

Class Converter

Είναι μία από τις βασικότερες κλάσεις γιατί περιέχει όλες τις συναρτήσεις με τις οποίες θα γίνεται η μετατροπή των μηνυμάτων.

Ακολουθεί ο κώδικας της παρακάτω κλάσης με ανάλυση της λειτουργίας κάθε μιας συνάρτησης ξεχωριστά.

```
package main_package;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;


public  class Converter {

  static String jsonConverterFilename = "jsonConverter.txt";

  static class ConverterItem  {
            public String subTopic="";
            String pubTopic="";
            int pubQoS=0;
            int pubRetained=0;
            String pubData="";
  }

  public static String errorWhileConvertingMessage="";



  static final String INCOMMING_MESSAGE_ID = "${receivedMessage}";
  static final String TOPIC_SPECIAL_CHAR_ID = "*";

  //=====================================================
  public static ArrayList<ConverterItem> getConverterItemList(int brokerID,
                  String jSettings){
            errorWhileConvertingMessage="";
            ArrayList<ConverterItem> list = new ArrayList<ConverterItem>();
            String brokerLabel="broker1";
            if (brokerID==2) brokerLabel="broker2";
            String jsonSettings =jSettings;
            if (jsonSettings.length()==0) jsonSettings =
                        StorageTools.readFile(jsonConverterFilename);
            System.out.println("========== jsonSettings: "+jsonSettings);
            if (jsonSettings!=null) {
                  try {
                        JSONObject mainJsonObject = new JSONObject(jsonSettings);
                        JSONArray brokerJA = mainJsonObject.getJSONArray(brokerLabel);
                        System.out.println("========== brokerJA: "+brokerJA.toString());
                        for (int i=0;i<brokerJA.length();i++) {
```

```
                                        JSONObject jsonItem = brokerJA.getJSONObject(i);
                                        ConverterItem item = new ConverterItem();
                                        item.subTopic = jsonItem.getString("subTopic");
                                        item.pubTopic = jsonItem.getString("pubTopic");
                                        JSONObject pubSettings = jsonItem.getJSONObject("pubSettings");
                                        item.pubQoS = pubSettings.getInt("qos");
                                        item.pubRetained = pubSettings.getInt("retained");
                                        try {
                                         item.pubData = jsonItem.getJSONObject("pubData").toString();
                                         } catch (JSONException e) {
                                                e.printStackTrace();
                                                item.pubData = jsonItem.getString("pubData");
                                        }

                                        list.add(item);
                                }
                        } catch (JSONException e) {
                                e.printStackTrace();
                                System.out.println("========= error: "+e.getMessage().toString());
                                errorWhileConvertingMessage=e.getMessage().toString();

                        }
                }
                System.out.println("========= list.size=: "+list.size());
                return list;
        }

        //=========================================================
        public static String getCovertedStatusMessage(String jsonSettings) {
                getConverterItemList(1,jsonSettings);
                if (errorWhileConvertingMessage.length()>0) return errorWhileConvertingMessage;
                else {
                        getConverterItemList(2,jsonSettings);
                        if (errorWhileConvertingMessage.length()>0) return errorWhileConvertingMessage;
                }
                return "";
        }


        //=======================================================
        public static List<String> getTopicList(ArrayList<ConverterItem> list){
                List<String> topicList =new ArrayList<String>();
                for (int i=0;i<list.size();i++){
                        ConverterItem item = list.get(i);
                        topicList.add(item.subTopic);
                }
                topicList = topicList.stream().distinct().collect(Collectors.toList());
                return topicList;
        }

        //=======================================================

        public static MqttClientController.PubMessage getConvertedPubMessage(String topic, String message,
                ArrayList<ConverterItem> list) {
                for (int i=0;i<list.size();i++) {
                        ConverterItem item = list.get(i);
                        //if (topic.equals(item.subTopic)){
                        System.out.println("topic="+topic);
```

```
                if (isTopicAccepted(topic,item.subTopic)) {
                        String pubTopic=item.pubTopic.trim();
                        System.out.println("pubTopic ="+topic);
                        if (pubTopic.length()>0) {
                                MqttClientController.PubMessage pubMessage =
                                        new MqttClientController.PubMessage();
                                pubMessage.topic = pubTopic;
                                pubMessage.qos= item.pubQoS;
                                pubMessage.retained= item.pubRetained;
                                pubMessage.message=
                                        getConvertedPayload(topic,message,item.pubData);
                                return pubMessage;
                        }
                        else return null;
                }
                else System.out.println("not accepted topic="+topic);
        }

        return null;


}

//=====================================================

public static ArrayList<MqttClientController.PubMessage> getConvertedPubMessageList(String topic,
                        String message, ArrayList<ConverterItem> list) {
        ArrayList<MqttClientController.PubMessage> pubList = new
                        ArrayList<MqttClientController.PubMessage>();
        for (int i=0;i<list.size();i++) {
                ConverterItem item = list.get(i);
                //if (topic.equals(item.subTopic)){
                System.out.println("topic="+topic);
                if (isTopicAccepted(topic,item.subTopic)) {
                        String pubTopic=item.pubTopic.trim();
                        System.out.println("pubTopic ="+topic);
                        if (pubTopic.length()>0) {
                                MqttClientController.PubMessage pubMessage =
                                        new MqttClientController.PubMessage();
                                pubMessage.topic = pubTopic;
                                pubMessage.qos= item.pubQoS;
                                pubMessage.retained= item.pubRetained;
                                pubMessage.message=
                                        getConvertedPayload(topic,message,item.pubData);
                                if (pubMessage.message.length()>0) pubList.add(pubMessage);
                        }
                }
                //else System.out.println("not accepted topic="+topic);
        }

        return pubList;


}


//=========================================================
```

```
static String getConvertedTopic(String subTopic, String pubTopic, String message) {
	String cTopic = pubTopic;

	ArrayList<String> labelList = new ArrayList<String>();
		int pos=-1;
		do {
			pos= cTopic.indexOf("${",pos+1);
			if (pos!=-1) {
				int lPos = cTopic.indexOf("}",pos+1);
				if (lPos!=-1) {
					String variable = cTopic.substring(pos+2, lPos);
					labelList.add(variable);
				}
				else break;
			}

		} while (pos>=0);
		for (int i=0;i<labelList.size();i++) {
			System.out.println("======== labelList."+i+"="+ labelList.get(i));
		}
	//	System.out.println("======== message ="+ message);
		for (int i=0;i<labelList.size();i++) {
			String jsonLabel=labelList.get(i);

			String valueFromIncommingMessage = "";
			if (jsonLabel.startsWith("topic=")) valueFromIncommingMessage =
				getTopicValue(subTopic,jsonLabel);
			else valueFromIncommingMessage = getJsonValue(message,jsonLabel);

			System.out.println("======== label "+i+"=   "+jsonLabel);
			System.out.println("======== valueFromIncommingJson=
				"+valueFromIncommingMessage);


			cTopic = cTopic.replace("${"+jsonLabel+"}",valueFromIncommingMessage);


		}

//		System.out.println("======== final cPayload ="+ cPayload);

	return cTopic;
}


//=============================================================

static String getConvertedPayload(String topic, String message, String payload) {
	String cPayload = payload;
	// περίπτωση 1.  επιστροφή του εισερχόμενου μηνύματος είτε αποκλειστικά είτε μαζί με
	// προσθήκη κειμένου
	//                                          είτε δημιουργώντας ένα json
	if (payload.contains(INCOMMING_MESSAGE_ID)){
		cPayload= cPayload.replace(INCOMMING_MESSAGE_ID,message);

	 }
```

```
                // περίπτωση 2.  αντικατάσταση τιμών στο προς αποστολή payload από τιμές που βρίσκονται στο εισερχόμενο μήνυμα.
                //                                          το εισερχόμενο payload πρέπει να είναι json
                else {
                //System.out.println("payload not contains(INCOMMING_MESSAGE_ID)");
                        ArrayList<String> labelList = new ArrayList<String>();
                        int pos=-1;
                        do {
                                pos= payload.indexOf("${",pos+1);
                                if (pos!=-1) {
                                        int lPos = payload.indexOf("}",pos+1);
                                        if (lPos!=-1) {
                                                String variable = payload.substring(pos+2, lPos);
                                                labelList.add(variable);
                                        }
                                        else break;
                                }

                        } while (pos>=0);

                //              System.out.println("======== payload ="+ payload);
                //              System.out.println("========= message ="+ message);
                                for (int i=0;i<labelList.size();i++) {
                                        String jsonLabel=labelList.get(i);

                                        String valueFromIncommingMessage = "";
                                        if (jsonLabel.startsWith("topic=")) valueFromIncommingMessage =
                                                getTopicValue(topic,jsonLabel);
                                        else valueFromIncommingMessage = getJsonValue(message,jsonLabel);


                                        System.out.println("==>>>====== label "+i+"=   "+jsonLabel);
                                        System.out.println("==>>>====== valueFromIncommingJson=
                                                "+valueFromIncommingMessage);

                                        if (valueFromIncommingMessage.equals("CLEAR_JSON_FIELD")) {
                                                cPayload = "";
                                                }
                                        else
                                cPayload = Payload.replace("${"+jsonLabel+"}",valueFromIncommingMessage);


                                }

        //                      System.out.println("======== final cPayload ="+ cPayload);
                }
                return cPayload;
        }


         //=================================================================
         public static String getJsonValue(String messageOnJsonFormat, String path){
                System.out.println("========= >>>>> getJsonValue   path= "+ path);
            if (path.length()==0) return "";
            ArrayList<String> labels = new ArrayList<>();
            String s= path;
            if (!s.endsWith("/")) s+='/';
            String part = "";
```

```
    for (int i=0; i<s.length();i++){
       char c = s.charAt(i);
       if ((c=='/') & (part.length()>0)) {
          labels.add(part.trim());
          part="";
       }
       else {
          part+=c;
       }
    }

    System.out.println("========= >>>>> labels.size= "+ labels.size());

    System.out.println("========= >>>>> messageOnJsonFormat= "+ messageOnJsonFormat);
    String jsonText = messageOnJsonFormat;
    try {
       for (int i=0;i<labels.size();i++) {

          String label = labels.get(i);
            System.out.println("========= >>>>> i="+i+" label  ="+ label);
          jsonText=getJsonLabelPart(label,jsonText);
          System.out.println("========= >>>>> jsonText  ="+ jsonText);
       }

} catch (JSONException e) {
                System.out.println("========= >>>>> error  ="+ e.getMessage().toString());
       return "";
}

    System.out.println("========= >>>>> jsonText final  ="+ jsonText);
    return jsonText;

}


//======================================================================
public static String getJsonLabelPart(String jsonLabel, String jsonText) throws JSONException {

          System.out.println("========= getJsonLabelPart > jsonLabel="+jsonLabel+"     jsonText ="+
                   jsonText);

    if (jsonLabel.endsWith("]")) {     // is array
          System.out.println("========= getJsonLabelPart > isArray");
       int sPos = jsonLabel.lastIndexOf("[");
       String arrayIndexAsText = jsonLabel.substring(sPos + 1,jsonLabel.length()-1);
       int arrayIndex = -1;
       try{arrayIndex= Integer.parseInt(arrayIndexAsText); }catch(NumberFormatException e){}
       if (arrayIndex >= 0) {
          String label_org = "";
          if (sPos > 0) label_org = jsonLabel.substring(0, sPos);
          if (label_org.length() > 0) {
             JSONObject jsonObject = new JSONObject(jsonText);
             JSONArray jsonArray = jsonObject.getJSONArray(label_org);
             if (arrayIndex < jsonArray.length()) {
                String jsonItem = jsonArray.getString(arrayIndex);
                return jsonItem;
             }
          }
```

```
            else {
                JSONArray jsonArray = new JSONArray(jsonText);
                if (arrayIndex < jsonArray.length()) {
                    String jsonItem = jsonArray.getString(arrayIndex);
                    return jsonItem;
                }
            }
        }
    }
    else {
            String jsonPart="";
        JSONObject jsonObject = new JSONObject(jsonText);
        System.out.println("======== getJsonLabelPart > jsonObject="+jsonObject.toString());

        try {
            jsonPart = jsonObject.getJSONObject(jsonLabel).toString();
        }
            catch (JSONException e) {
                                try {
                                    jsonPart = jsonObject.getString(jsonLabel);
                                    System.out.println("======== getJsonLabelPart 1 > jsonPart="+jsonPart);
                                } catch (JSONException e2) {
                                    System.out.println("======== getJsonLabelPart 2 > jsonPart="+jsonPart);
                                    try {
                                            jsonPart = ""+jsonObject.getDouble(jsonLabel);
                                    } catch (JSONException e3) {
                                            System.out.println("======== getJsonLabelPart  > Error3");
                                            return "CLEAR_JSON_FIELD";
                                    }
                                }
                                System.out.println("======== getJsonLabelPart 1 > jsonPart final ="+jsonPart);
                                //return jsonPart;
                            }
        return jsonPart;
            }

        return "";
    }

//=================================================================
public static String getTopicValue(String topic, String pattern) {
            int equalPos =   pattern.indexOf("=");
            try {
            if ((equalPos>=0) & (pattern.length()>equalPos+1)) {
                        String clearedPattern = pattern.substring(equalPos+1);
                          int specialCharPos =   clearedPattern.indexOf(TOPIC_SPECIAL_CHAR_ID);
                          if (specialCharPos>=0) {
                                    String prefix = "";
                                    if (specialCharPos>0)
                                            prefix=clearedPattern.substring(0,specialCharPos+1);
                                            String suffix =  clearedPattern.substring(specialCharPos+
                                           TOPIC_SPECIAL_CHAR_ID.length());
                                    int topicPos_start = topic.indexOf(prefix)+prefix.length();
                                    String value = "";
                                    if (suffix.length()>0) {
                                              int topicPos_end = topic.indexOf(suffix);
                                              if (topicPos_end>=0) value = topic.substring(topicPos_start,
                                            topicPos_end);
```

```
                                        }
                                                else value = topic.substring(topicPos_start);
                                        return value;

                                }
                }
                } catch (Exception e) {}
                return "";
        }


        //=========================================================
        static ArrayList<String> getTopicPartList(String topic) {
                ArrayList<String> list= new ArrayList<String>();
                String restTopic=topic;
                int pos =-1;
                do {
                        pos = restTopic.indexOf('/');
                        if (pos==-1) list.add(restTopic);
                        else {
                                String firstPart = restTopic.substring(0,pos);
                                list.add(firstPart);
                                restTopic=restTopic.substring(pos+1);
                        }

                } while (pos>=0);
                return list;
        }
        //=========================================================
        static boolean isTopicAccepted(String topic, String topicMask) {
                        if (topicMask.endsWith("#")) {
                                String firstPartOfMask = topicMask.substring(0,topicMask.length()-1);
                                if (firstPartOfMask.length()<=topic.length()) {
                                        String firstPartOfTopic = topic.substring(0,firstPartOfMask.length());
                                        if (firstPartOfMask.endsWith(firstPartOfTopic)) return true;
                                }
                        }
                        else if (topicMask.contains("+")) {
                                ArrayList<String> topicList  =getTopicPartList(topic);
                                ArrayList<String> maskList  =getTopicPartList(topicMask);
                                for (int i=0;i<maskList.size();i++) {
                                        if (i==topicList.size()) return false;
                                        String maskPart=maskList.get(i);
                                        String topicPart=topicList.get(i);
                                        if (!((topicPart.equals(maskPart))|| (maskPart.equals("+")))) return
                                                false;

                                }
                                return true;

                        }
                        else if (topic.equals(topicMask)) return true;



                return false;
        }
}
```

Class JsonEditor

Η κλάση αυτή περιλαμβάνει ένα απλό JsonEditor με γραφικό περιβάλλον για να βοηθήσει το χρήστη να κάνει τροποποιήσεις στο αρχείο JsonConverter.txt.

Η διαφορά από ένα απλό editor είναι ότι η κλάση περιέχει και τη δυνατότητα εύρεσης λαθών στα Json αντικείμενα που εισάγει ο χρήστης βοηθώντας στο debugging.

```
package main_package;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Font;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JToolBar;
import javax.swing.border.BevelBorder;

public class JsonEditor  {

static JTextArea editArea=null;
static  JFrame f=null;
   public static void main(String [] args) {
     showWindow();
   }

   public static void show() {
            if (f!=null) f.setVisible(true);
   }

   public static void showWindow() {
            JToolBar toolBar =  new JToolBar("TaskBar", JToolBar.HORIZONTAL);
            JButton btnSave = new JButton("Save");
            btnSave.addMouseListener(new MouseAdapter() {
                     @Override
                     public void mouseClicked(MouseEvent e) {
                              String jsonText= editArea.getText().toString().trim();
                              String result =
                               StorageTools.stringToFile(jsonText,Converter.jsonConverterFilename);
                              JOptionPane.showMessageDialog(null, result);
                     }
            });

            toolBar.add(btnSave);
            JButton btnCheck = new JButton("Check");
            btnCheck.addMouseListener(new MouseAdapter() {
                     @Override
                     public void mouseClicked(MouseEvent e) {
                              String jsonText= editArea.getText().toString().trim();
```

```
                                String error = Converter.getCovertedStatusMessage(jsonText);
                                if (error.length()==0) JOptionPane.showMessageDialog(null, "Json is valid");
                                else  JOptionPane.showMessageDialog(null, error);
                        }
                });
                toolBar.add(btnCheck);
        JPanel panel = new JPanel(new BorderLayout());
        editArea = new JTextArea(20,80);
        editArea.setBorder(new BevelBorder(BevelBorder.RAISED, null, null, null, null));
        Font font = new Font(
                Font.MONOSPACED,
                Font.PLAIN,
                editArea.getFont().getSize());
        editArea.setFont(font);
        panel.add(new JScrollPane(editArea,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                        JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS));
        f = new JFrame("Editpad");
        f.setTitle("JsonEditor");

        Container cp = f.getContentPane();
        cp.setLayout(new BorderLayout());
        f.getContentPane().add(toolBar, BorderLayout.NORTH);
        f.getContentPane().add(panel);
        f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
          f.pack();
        f.setVisible(true);

        String jsonText = StorageTools.readFile(Converter.jsonConverterFilename);
       editArea.setText(jsonText);

    }


}
```

Class Terminal

Η κλάση περιλαμβάνει έναν terminal σε γραφικό περιβάλλον στον οποίο προβάλλονται τα εισερχόμενα και εξερχόμενα μηνύματα από κάθε broker.

```
package main_package;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Font;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JToolBar;
import javax.swing.border.BevelBorder;
import java.awt.Color;

public class Terminal  {
   JTextArea editArea=null;
    JFrame f=null;
     public  void main(String [] args) {
       showWindow();
     }
     public  void show() {
              if (f!=null) f.setVisible(true);
     }
     public void println(String text) {
              editArea.append(text+"\n");
     }
     public void print(String text) {
              editArea.append(text);
     }
     public void setTitle(String title) {
               f.setTitle("Terminal: "+title);
     }

     //=========================================================
     public  void showWindow() {
              JToolBar toolBar =  new JToolBar("TaskBar", JToolBar.HORIZONTAL);
              JButton btnSave = new JButton("Clear");
              btnSave.addMouseListener(new MouseAdapter() {
                                       @Override
                                       public void mouseClicked(MouseEvent e) {
                                                editArea.setText("");
                                       }
                             });

              toolBar.add(btnSave);
```

```
                JPanel panel = new JPanel(new BorderLayout());
                editArea = new JTextArea(20,80);
                editArea.setForeground(new Color(230, 230, 250));
                editArea.setBackground(Color.BLACK);
                editArea.setEditable(false);
                editArea.setCaretColor(Color.RED);
                editArea.addFocusListener(new FocusListener() {

                  @Override
                  public void focusGained(FocusEvent e) {
                    editArea.getCaret().setVisible(true);
                  }
                                        @Override
                                        public void focusLost(FocusEvent e) {
                                        }
                });
                editArea.setBorder(new BevelBorder(BevelBorder.RAISED, null, null, null, null));
                Font font = new Font(
                      Font.MONOSPACED,
                      Font.PLAIN,
                      editArea.getFont().getSize());
                editArea.setFont(font);
                panel.add(new
JScrollPane(editArea,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,JScrollPane.HORIZONTAL_SCROLLBAR_AL
WAYS));

                f = new JFrame("Editpad");
                Container cp = f.getContentPane();
                cp.setLayout(new BorderLayout());
                f.getContentPane().add(toolBar, BorderLayout.NORTH);
                f.getContentPane().add(panel);
                f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                  f.pack();
                f.setVisible(true);
              }
            }
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. *Jasmin Guth, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Lukas Reinfurt (2016 November)*
   Comparison of IoT Platform Architectures: A Field Study based on a Reference Architecture.
   In *2016 Cloudification of the Internet of Things (CIoT)*
   https://www.iaas.uni-stuttgart.de/publications/INPROC-2016-48-Comparison-of-IoT-Platform-Architectures.pdf

2. *Jasmin Guth1, Uwe Breitenbücher1, Michael Falkenthal1, Paul Fremantle2, Oliver Kopp3, Frank Leymann1, Lukas Reinfurt1*
   A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences
   In 2018 Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives
   https://www.iaas.uni-stuttgart.de/publications/INBOOK-2018-01-A-Detailed-Analysis-of-IoT-Platform-Architectures-Concepts-Similarities-and-Differences.pdf

3. **Thinkspeak Platform Documentation**
   https://www.mathworks.com/help/thingspeak/

4. **Thinger.io platform documentation**
   https://docs.thinger.io/

5. **Guiding Principles of REST**
   https://restfulapi.net

6. *Alvaro Luis Bustamante,Miguel A. Patricio,José M. Molina*
   Thinger.io: An Open Source Platform for DeployingData Fusion Applications in IoT Environments
   https://www.researchgate.net/publication/331490834_Thingerio_An_Open_Source_Platform_for_Deploying_Data_Fusion_Applications_in_IoT_Environments/figures

7. *Daniyal Akhtar Qureshi*
   PERFORMANCE EVALUATION OF IoT PLATFORMS IN GREEN ICT APPLICATIONS
   Computer Science and Engineering, master's level (120 credits) 2018

http://www.diva-portal.se/smash/get/diva2:1259803/FULLTEXT01.pdf

8. **ESP32 documentation**
   https://docs.espressif.com/projects/esp-idf/en/latest/esp32/

9. **ESP 32 wikipedia**
   https://en.wikipedia.org/wiki/ESP32

10. **Install Docker Engine on Ubuntu**
    https://docs.docker.com/engine/install/ubuntu/

11. *Λισγάρας Βασίλειος (2016)*
    Πτυχιακή εργασία: Αυτόματο σύστημα περιβαντολογικού ελέγχου κτιρίων.
    https://apothetirio.lib.uoi.gr/xmlui/bitstream/handle/123456789/5879/1439.pdf?sequence=1

12. *Ιωάννης Μασκλαβάνος (Jun 2016)*
    Πτυχιακή εργασία: ΣΥΣΤΗΜΑ ΔΙΑΧΕΙΡΙΣΗΣ ΚΑΙ ΑΠΟΜΑΚΡΥΣΜΕΝΟΥ ΕΛΕΓΧΟΥ ΑΙΣΘΗΤΗΡΩΝ.
    https://apothetirio.lib.uoi.gr/xmlui/bitstream/handle/123456789/5881/1441.pdf?sequence=1

13. **What is an IoT Gateway - Thingsboard documentation**
    https://thingsboard.io/docs/iot-gateway/what-is-iot-gateway/

14. **MQTT API - Thingspeak**
    https://www.mathworks.com/help/thingspeak/mqtt-api.html

15. **Node-Red Low-code programming for event-driven applications**
    https://nodered.org/

16. **Ο κώδικας της IoT Gateway που δημιουργήθηκε για την παρούσα εργασία στο GitHub:**
    https://github.com/iliaslmp/MQTT_GATEWAY.git

17. *André Glória, Francisco Cercas, Nuno Souto*
    Design and implementation of an IoT gateway to create smart environments
    The 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2017)
    https://www.researchgate.net/publication/317548714_Design_and_implementation_of_an_IoT_gateway_to_create_smart_environments

18. *Joe Folkens (Texas Instruments - 2014)*

Building a gateway to the Internet of Things
https://www.ti.com/lit/wp/spmy013/spmy013.pdf?ts=1619722232262&ref_url=https%253A%252F%252Fwww.google.com%252F

19. *Thomas Zachariah, Noah Klugman, Bradford Campbell, Joshua Adkins, Neal Jackson, and Prabal Dutta*
The Internet of Things Has a Gateway Problem
In HotMobile '15: Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications, February 2015 Pages 27–32
https://www.cs.virginia.edu/~bjc8c/papers/zachariah15gateway.pdf

20. *André Glória, Francisco Cercas, Nuno Souto (December 2017)*
Design and implementation of an IoT gateway to create smart environments
The 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2017)
https://www.researchgate.net/publication/317548714_Design_and_implementation_of_an_IoT_gateway_to_create_smart_environments

21. IoT connected devices worldwide 2030
Published by Lionel Sujay Vailshery, Jan 22, 2021
https://www.statista.com/statistics/802690/worldwide-connected-devices-by-access-technology/

22. *Byungseok Kang, Hyunseung Choo*
An experimental study of a reliable IoT gateway
In ICT Express 4 (2018) 130-133
https://reader.elsevier.com/reader/sd/pii/S2405959516301485?token=C201D8BB59FD9125A00648CA6ECA009590999A8CA5632FC0B503E1B9DDD7C6242881D8D560B33F017C4E6067FC61391B&originRegion=eu-west-1&originCreation=20210414102206

23. *Rodger Lea, Michael Blackstock*
City Hub: A Cloud-Based IoT Platform for Smart Cities
Published in: 2014 IEEE 6th International Conference on Cloud Computing Technology and Science
https://ieeexplore.ieee.org/document/7037764

24. *Geng Yang, Li Xie, Matti Mäntysalo, Xiaolin Zhou, Zhibo Pang, Li Da Xu, Sharon Kao-Walter, Qiang Chen, Li-Rong Zheng*
A Health-IoT Platform Based on the Integration of Intelligent Packaging, Unobtrusive Bio-Sensor, and Intelligent Medicine Box
Published in: IEEE Transactions on Industrial Informatics (Volume: 10, Issue: 4, Nov. 2014)
https://ieeexplore.ieee.org/document/6747344

25. Research and Application of the IOT Gateway Based on the Real-Time Specification for Java
https://www.i-scoop.eu/internet-of-things-guide/iot-gateways/

26. *Raghunath Nambiar (September 14, 2017)*
Introducing TPC Express Benchmark IoT (TPCx-IoT) Industry's First Standard for Internet of Things
http://www.tpc.org/tpcx-iot/documents/introducing_tpcx-iot.pdf

27. Web Link
https://www.thalesgroup.com/en/markets/digital-identity-and-security/iot/inspired/iot-gateway

28. *Jinpo Fan, Zhiqiang Wang, Changchun Li*
Design and Implementation of IoT Gateway Security System
2019 International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM)
https://sci-hub.se/https://ieeexplore.ieee.org/document/8950784

29. Thingsboard Documentation
https://thingsboard.io/docs/reference/

30. *Διπλωματική Εργασία του Νεστορίδη Θεόδωρου (Mar 10, 2020)*
Αρχιτεκτονική ασφάλειας συστημάτων στο διαδίκτυο των αντικειμένων
http://ikee.lib.auth.gr/record/317350/files/GRI-2020-27070.pdf

31. *Διπλωματική Εργασία του ΛΙΒΑΝΟΥ ΘΕΟΔΩΡΟΥ (2019-2020)*
Βελτιστοποίηση Ποιότητας Επιστημονικών Μετρήσεων με Χρήση Δικτύων Αισθητήρων
https://nemertes.lis.upatras.gr/jspui/bitstream/10889/14444/1/%CE%9A%CE%B5%CE%AF%CE%BC%CE%B5%CE%BD%CE%BF.pdf

32. *Natallia Sakovich (Aug 22, 2018)*
Internet of Things (IoT) Protocols and Connectivity Options
https://www.sam-solutions.com/blog/internet-of-things-iot-protocols-and-connectivity-options-an-overview/