Live Streaming and Presentation of Sensors’ Data

Μοίρας Αλέξανδρος   
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών  
Εθνικό Μετσόβιο ΠολυτεχνείοΑθήνα, Ελλάδα  
el18081@mail.ntua.gr

Παντελαίος Δημήτριος   
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών  
Εθνικό Μετσόβιο ΠολυτεχνείοΑθήνα, Ελλάδα  
el18049@mail.ntua.gr

Κώδικας Εργασίας:

<https://github.com/dpantelaios/Analysis-and-Design-of-Information-Systems>

*Abstract*—Tα τελευταία χρόνια αυξάνονται με ραγδαίους ρυθμούς οι ηλεκτρονικές συσκευές που επιτρέπουν την σύνδεση και την ανταλλαγή δεδομένων. Για αυτό το λόγο οδηγηθήκαμε στην δημιουργία του Διαδικτύου των πραγμάτων ή αλλιώς Internet of Things(IoT), το οποίο αποτελεί ένα δίκτυο επικοινωνίας πληθώρας συσκευών και παρέχει την δυνατότητα λήψης, συγκέντρωσης, επεξεργασίας και απεικόνισης δεδομένων που παράγονται από τις edge συσκευές, αλλά και διαχείρισης της λειτουργίας τους[1]. Εμπνευσμένοι από τα διάφορα ήδη υπάρχοντα IoT συστήματα, θα δημιουργήσουμε ένα live streaming σύστημα το οποίο θα αποτελεί prototype ενός πραγματικού IoT συστήματος.

Keywords—Apache Kafka, Kafka streams, telegraf, grafana, influxdb, websockets, IoT, sensors

# Introduction

Σκοπός της εργασίας είναι η δημιουργία μιας εφαρμογής ΙοΤ, κατά την οποία θα παράγουμε δεδομένα από αισθητήρες, θα λαμβάνονται από κάποιους κόμβους όπου θα γίνεται η απαιτούμενη επεξεργασία τους και με την σειρά τους τα επεξεργασμένα δεδομένα θα αποστέλλονται στον αντίστοιχο κόμβο, όπου θα πραγματοποιούνται η αποθήκευση τους σε μια timeseries database και η αποστολή τους για live παρουσίαση στον κατάλληλο κόμβο.

# STEPS TO SET UP SYSTEm

Αρχικά κάνουμε git clone το repository που έχει παρατεθεί παραπάνω και περιλαμβάνει τον κώδικα της εργασίας, προκειμένου να τον κατεβάσουμε στο σύστημα από το οποίο θα γίνει η εκτέλεση του. Εναλλακτικά μπορούμε να μεταφερθούμε στο repository που περιέχει τον κώδικα και να τον κατεβάσουμε χειροκίνητα πατώντας Code Download Zip και στην συνέχεια να τον κάνουμε extract στο σύστημα που έγινε η αποθήκευση.

To project εκτελείται αποκλειστικά σε docker container, οπότε η μοναδική εγκατάσταση λογισμικού που θα πρέπει να κάνουμε είναι η εγκατάσταση της πλατφόρμας docker Desktop από [docker Desktop Download](https://www.docker.com/products/docker-desktop/).

Στην συνέχεια, προκειμένου να θέσουμε σε λειτουργία όλα τα docker containers που χρησιμοποιεί η εφαρμογή πρέπει, αφού βρισκόμαστε ήδη εντός του directory που κατεβάσαμε, να ακολουθηθούν με την σειρά τα ακόλουθα βήματα:

* cd Messaging\_Broker\_Layer
* docker compose up
* cd Presentation Layer
* docker compose up
* cd Data-Storage Layer
* docker compose up
* cd Live Streaming Layer Java
* docker compose build
* docker compose up
* cd Device Layer
* docker compose build
* docker compose up

Θα χρειαστούμε 5 διαφορετικά terminal, ένα για κάθε Compose Project που θέλουμε να θέσουμε σε λειτουργία. Για κάθε Docker Compose Project που κάνουμε compose up, θα υπάρχει αναμονή λίγα δευτερόλεπτα μέχρι να ολοκληρωθούν οι απαραίτητες διαδικασίες initialization των containers του, ώστε να μπορούμε να συνεχίσουμε στα επόμενα βήματα. Συγκεκριμένα οι εξαρτήσεις έχουν ως εξής: Πρώτα πρέπει να σηκωθεί το Messaging Broker Layer παράλληλα με το Presentation Layer, έπειτα μπορούν να σηκωθούν τα Data Storage και Live Streaming layers, και αφού σηκωθούν όλα και ολοκληρωθεί το initialization τους, μόνο τότε μπορεί να σηκωθεί το device layer και να τεθεί η εφαρμογή σε πλήρη λειτουργία.

Μόλις ολοκληρωθούν τα παραπάνω βήματα θα μεταβούμε στην διεύθυνση <http://localhost:3000/> και θα γίνει σύνδεση με username και password την λέξη admin για να εισέλθουμε στο περιβάλλον του Grafana.

Από εκεί και πέρα μπορούμε να δούμε την live απεικόνιση των δεδομένων πατώντας στο link:

<http://localhost:3000/d/grS74cbVz/live-streaming?orgId=1>

Τέλος μπορούμε να δούμε την απεικόνιση των δεδομένων που είναι αποθηκευμένα στην βάση πατώντας το link:

<http://localhost:3000/d/1Dr8HObVk/stored-data?orgId=1>

Εναλλακτικά μπορεί να γίνει περιήγηση στο Grafana πατώντας Dashboards Browse και ανοίγοντας τα δύο αποθηκευμένα dashboards που αντιστοιχούν στην live απεικόνιση και στην απεικόνιση των δεδομένων από την βάση.

# COMPONENT, SOFTWARE

Η εφαρμογή αποτελείται από 8 components τα οποία παρουσιάζονται παρακάτω:

1. Το component device\_layer είναι υπεύθυνο για την παραγωγή δεδομένων ανά 15 λεπτά για 7 αισθητήρες, ανά ημέρα για 2 αισθητήρες, 4-6 δεδομένα την ημέρα τυχαία επιλεγμένα για τον αισθητήρα ανίχνευσης κίνησης, καθώς και ετεροχρονισμένα δεδομένα για τον αισθητήρα που παράγει μετρήσεις κατανάλωσης νερού ανά 15 λεπτά. Τα ετεροχρονισμένα δεδομένα που παράγονται ανά 6 ώρες και αντιστοιχούν σε ημερομηνία 2 μέρες πίσω θεωρούνται late processed, ενώ αυτά που παράγονται ανά 30 ώρες και αντιστοιχούν σε 10 μέρες πίσω θεωρούνται late rejected.

Oι αισθητήρες που προσομοιώθηκαν τα αντίστοιχα όρια των πιθανών τιμών τους παρουσιάζονται παρακάτω:

TABLE I . DEVICES

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sensor | Sensor Name | Interval | Value Type | Value Range |
| Θερμοκρασία | TH1 | 15 min | float | 12-35 |
| Θερμοκρασία | TH2 | 15 min | float | 12-35 |
| Ενέργεια κλιματιστικών | HVAC1 | 15 min | float | 0-100 |
| Ενέργεια κλιματιστικών | HVAC2 | 15 min | float | 0-200 |
| Ενέργειας υπόλοιπων ηλεκτρικών συσκευών | MiAC1 | 15 min | float | 0-150 |
| Ενέργειας υπόλοιπων ηλεκτρικών συσκευών | MiAC2 | 15 min | float | 0-200 |
| Κατανάλωση νερού | W1 | 15 min | float | 0-1 |
| Συνολική ενέργεια | Etot | 1 day | float | 2600x24 – 1000 – 2600x24 + 1000 |
| Συνολική κατανάλωση νερού | Wtot | 1 day | float | 100-120 |
| Ανίχνευση κίνησης | Mov1 | - | int | 0-1 |

Fig1: Αισθητήρες που προσομοιώθηκαν

Tα δεδομένα που παράγονται στο device layer αποστέλλονται στα αντίστοιχα topic που παρουσιάζονται στον παρακάτω πίνακα:

TABLE ΙΙ . ΤΟPICS

|  |  |  |  |
| --- | --- | --- | --- |
| Topic | Αισθητήρες | Partitions | Replication Factor |
| 15min | Αισθητήρες με interval 15 λεπτών, μαζί με τα ετεροχρονισμένα δεδομένα του αισθητήρα κατανάλωσης νερού | 7 | 3 |
| day | Αισθητήρες με interval 1 μέρα | 2 | 3 |
| movementSensor | Αισθητήρας ανίχνευσης κίνησης | 1 | 3 |

Fig2:Τα topic που στέλνονται τα δεδομένα κάθε sensor

Στα δεδομένα του κάθε αισθητήρα που στέλνουμε στον kafka χρησιμοποιούμε ως key το όνομα του αισθητήρα, έτσι ώστε να στέλνονται τα δεδομένα του στο ίδιο partition (εκμεταλλευόμαστε το γεγονός ότι το destination partition, εφόσον υπάρχει key, επιλέγεται βάσει του hash του key επομένως messages με το ίδιο key θα καταλήγουν πάντα στο ίδιο partition. Αυτό μας είναι απαραίτητο καθώς το windowing στον χρόνο (θα αναλυθεί στο live streaming layer) θέλουμε να γίνεται σε επίπεδο ενός partition για κάθε αισθητήρα, ώστε να έχουμε total ordering στα records του και να αποφύγουμε out-of-order messages που θα δημιουργούσαν προβλήματα, σε ακραίες περιπτώσεις προχωρώντας απότομα τον χρόνο του stream και οδηγώντας στο κλείσιμο παραθύρων που έχουν ακόμα να λάβουν δεδομένα.

1. Tα component kafka και zookeeper χρησιμοποιούνται για την επικοινωνία μεταξύ των services. Ο zookeeper λειτουργεί ως συντονιστής των kafka clusters και διαχειρίζεται τα metadata τους. Συγκεκριμένα χρησιμοποιούμε 3 kafka-brokers χρησιμοποιώντας έναν από αυτούς ως τον bootstrap server στον οποίο συνδέονται οι consumers και οι producers, ο οποίος γνωστοποιεί σε αυτούς την παρουσία των υπολοίπων brokers. Mε το component kafka-init δημιουργούμε topics με συγκεκριμένα ονόματα στα οποία τα services μπορούν να γράφουν και να διαβάζουν δεδομένα. Ο αριθμός των partitions ανά topic επιλέχθηκε με βάση τον αριθμό των αισθητήρων-διακριτών δεδομένων που γράφουν-γράφονται σε κάθε topic ώστε να επιτύχουμε τον μέγιστο δυνατό βαθμό παραλληλίας. Θέλουμε ιδανικά κάθε αισθητήρας να γράφει σε δικό του partition ώστε για τους διάφορους ανεξάρτητους αισθητήρες να μπορούν να καταναλωθούν δεδομένα ταυτόχρονα από τους consumers που βλέπουν τα partitions. Π.χ. για τους 15min αισθητήρες έχουμε 7 συσκευές που γράφουν εκεί (th1, th2, hvac1, hvac2, miac1, miac2, w1) επομένως επιλέγουμε να χρησιμοποιήσουμε 7 partitions. Το replication factor 3 μας εγγυάται ότι ακόμα και αν κάποιος broker αποτύχει, ένας από τους άλλους δύο θα αναλάβει τα partitions στα οποία ήταν ηγέτης ο πεσών broker. Με αυτόν τον τρόπο εξασφαλίζουμε το scalability του broker, καθώς επιπλέον αισθητήρες στο device layer απλώς θα οδηγήσουν σε μία trivial αύξηση των partitions των κατάλληλων topics, τα οποία έχουν τη δυνατότητα να μοιραστούν σε 3 διαφορετικούς brokers οι οποίοι δύνανται να χειριστούν χιλιάδες partitions χωρίς πρόβλημα. Επομένως αύξηση των devices δεν αναμένεται να έχει δυσμενείς επιπτώσεις στον broker μας. Ακόμη το kafka χρησιμοποιείται στο παρασκήνιο για την backup αποθήκευση κάποιων aggregated δεδομένων και κάποιων states (changelog topics) κατά την επεξεργασία των δεδομένων στο live streaming layer, το οποίο θα αναλυθεί παρακάτω.
2. To component live\_streaming\_layer java διαβάζει από τα δεδομένα των αισθητήρων από τα αντίστοιχα topic και πραγματοποιεί την απαιτούμενη επεξεργασία σε κάθε αισθητήρα.

TABLE III AGGREGATIONS

|  |  |
| --- | --- |
| Αισθητήρας | Είδος επεξεργασίας |
| TH1 | Μέσος όρος τη μέρα |
| TH2 | Μέσος όρος τη μέρα |
| HVAC1 | Άθροισμα τη μέρα |
| HVAC2 | Άθροισμα τη μέρα |
| MiAC1 | Άθροισμα τη μέρα |
| MiAC2 | Άθροισμα τη μέρα |
| W1 | Άθροισμα τη μέρα συμπεριλαμβανομένων των late processed events |
| Etot | Μέγιστο κάθε μέρα |
| Wtot | Μέγιστο κάθε μέρα |
| Mov1 | Συνολικό άθροισμα |
| Etot | Διαφορά μεταξύ διαδοχικών ημερών |
| Wtot | Διαφορά μεταξύ διαδοχικών ημερών |
| Hvac1, Hvac2, MiAC1, MiAC1, Etot | Etot – (Hvac1\_daily\_sum + Hvac2\_daily\_sum + MiAC1\_daily\_sum + MiAC1\_daily\_sum) |
| W1, Wtot | Wtot – W1\_daily\_sum |

Fig3:Τύπος επεξεργασίας για κάθε αισθητήρα

Στην συνέχεια τόσο τα αρχικά δεδομένα όσο και αυτά που προήλθαν από επεξεργασία αποστέλλονται στον kafka στα αντίστοιχα topics που παρουσιάζονται παρακάτω:

TABLE IV TOPICS

|  |  |
| --- | --- |
| Topic | Δεδομένα |
| raw | Oλα τα δεδομένα 15λεπτων αισθητήρων πριν την επεξεργασία |
| aggDay15min | Oλα τα δεδομένα 15λεπτων αισθητήρων μετά την επεξεργασία (μέσος όρος ή άθροισμα) |
| aggDayDiff | Oλα τα δεδομένα ημερήσιων αισθητήρων μετά την επεξεργασία (διαφορά μεταξύ διαδοχικών ημερών) |
| leaks | Το αποτέλεσμα της καθημερινής διαφοράς της συνολικής κατανάλωσης ενέργειας από το άθροισμα των τιμών κατανάλωσης ενέργειας που στέλνουν οι αντίστοιχοι αισθητήρες ανά 15 λεπτά και η αντίστοιχη διαφορά για την κατανάλωση νερού |
| totalMovements | Συνολικές ανιχνεύσεις κίνησης μέχρι την δεδομένη ημερομηνία |
| lateRejected | Τα δεδομένα του αισθητήρα νερού w1 που ήρθαν καθυστερημένα κατά 10 μέρες |

Fig4: Τα δεδομένα που στέλνονται σε κάθε topic

Για την αναγνώριση των late rejected events χρησιμοποιήσαμε έναν transformer ο οποίος αποθηκεύει την μέγιστη ημερομηνία που έχει συναντήσει μέχρι στιγμής και αν η ημερομηνία του νέου δεδομένου είναι τουλάχιστον 10 μέρες παλαιότερη αναγνωρίζεται ως late rejected και φιλτράρεται εκτός από την επεξεργασία των δεδομένων και αποστέλλεται σε ξεχωριστό topic στον message broker.

Αντίστοιχα δημιουργήθηκε ένας transformer που αποθηκεύει πόσες ανιχνεύσεις κίνησης έχουν γίνει μέχρι εκείνη την στιγμή και προσθέτει 1 για κάθε νέα ανίχνευση που λαμβάνει.

Tέλος για τον υπολογισμό των διαρροών γίνεται inner join των stream που υπολογίζουν το sum με τα αντίστοιχα stream που περιέχουν τις καθημερινές συνολικές μετρήσεις κατανάλωσης ενέργειας και νερού αντίστοιχα.

1. Το component influxdb αποτελεί την timeseries βάση δεδομένων στην οποία αποθηκεύονται τόσο το αρχικά όσο και τα επεξεργασμένα δεδομένα.

Το component telegraf λαμβάνει τα δεδομένα από τα κατάλληλα topic σε μορφή json και κάνει τις απαραίτητες αλλαγές τα στέλνει σε μορφή influx τόσο στην βάση για την εισαγωγή τους, όσο και μέσω web sockets στο Grafana για την απεικόνιση τους.

Μία από τις βασικές αλλαγές που κάνει το telegraf είναι να παίρνει την ημερομηνία της κάθε εγγραφής που λαμβάνει από τα field σε millisecond και να την μεταφέρει στο πεδίο timestamp, προκειμένου να αποθηκευτεί η εγγραφή με την σωστή ημερομηνία στην βάση δεδομένων και όχι με την ημερομηνία εισαγωγής της σε αυτή.

Ακόμη στα δεδομένα που είναι διαθέσιμο το όνομα του αισθητήρα στον οποίο αναφέρονται, μεταφέρεται στο όνομα του measurement μαζί με το αντίστοιχο prefix που αναφέρεται στον τύπο των δεδομένων, ώστε να είναι ευδιάκριτα τα ονόματα στην βάση δεδομένων και να αντιπροσωπεύουν τα δεδομένα που περιέχουν.

Τέλος στις διαρροές προστέθηκε το είδος την κατανάλωσης στην οποία αναφέρεται, το οποίο λαμβάνεται και πάλι από το αντίστοιχο πεδίο στα fields της κάθε εγγραφής.

Στον παρακάτω πίνακα παρουσιάζονται τα ονόματα των measurements, όσο και τα δεδομένα που αντιπροσωπεύουν.

TABLE V ΜEASUREMENT NAMES

|  |  |
| --- | --- |
| Όνομα measurement | Δεδομένα που περιλαμβάνει |
| SensorName | Μη επεξεργασμένα δεδομένα αντίστοιχου αισθητήρα |
| “aggDay15min\_” + sensorName | Ημερήσιος μέσος όρος ή άθροισμα αντίστοιχου 15λεπτου αισθητήρα |
| “aggDayDiff\_” + sensorName | Διαφορά τιμών διαδοχικών ημερών αντίστοιχου ημερήσιου αισθητήρα |
| leakType + “\_leak” | Διαρροή κατανάλωσης ενέργειας ή νερού |
| “Total\_moves” | Συνολικός αριθμός ανιχνεύσεων κίνησης μέχρι την δεδομένη ημερομηνία |
| “Late\_rejected” + sensorName | Τα Late rejected δεδομένα του αντίστοιχου αισθητήρα |

Fig5: Όνομα measurement και τα δεδομένα που περιέχει

1. Το component grafana-server2 είναι υπεύθυνο για την απεικόνιση των δεδομένων σε διαγράμματα και πίνακες. Αρχικά δημιουργούμε ένα API key, με το οποίο θα μπορεί το component telegraf να στέλνει δεδομένα μέσω web sockets για live απεικόνιση. Ακόμη γίνεται σύνδεση της βάσης δεδομένων influxdb, προκειμένου να λαμβάνουμε δεδομένα και από εκεί για απεικόνιση.

Δημιουργούμε ένα dashboard που περιλαμβάνει panels για live απεικονίσεις. Πιο συγκεκριμένα περιλαμβάνει time series απεικονίσεις για τους αισθητήρες TH1, W1, Wtot, τα αποτελέσματα των aggregation για TH1, W1, την διαρροή του νερού, αλλά και πίνακες για τον TH1, την διαρροή νερού και τα Late rejected events.

Ακόμη δημιουργούμε ένα dashboard που περιλαμβάνει αντίστοιχα panels, όπως στην live παρουσίαση, για απεικονίσεις δεδομένων από την βάση.

Στο τέλος κάνουμε mount το image του component, ώστε να μην χρειάζεται να τα δημιουργούμε όλα ξανά από την αρχή κάθε φορά, αλλά να μπορούμε να έχουμε πρόσβαση με εύκολο τρόπο απλώς πατώντας στα link που περιλαμβάνονται στα βήματα για να σετάρουμε την εφαρμογή μας.

Diagram

Description automatically generatedDiagram

Description automatically generatedFig6: Component diagram της εφαρμογής μας

Fig7: Deployment diagram της εφαρμογής μας

Παραπάνω στις εικόνες 6 και 7 παρουσιάζονται τόσο τo component diagram, το οποίο περιλαμβάνει όλα τα components αλλά και το πως συνδέονται μεταξύ τους, όσο και το deployment diagram το οποίο παρουσιάζει την δομή του συστήματος, αλλά και τα components που υπάρχουν σε κάθε κόμβο[2].

Το software που χρησιμοποιήθηκε

* Python3.10.10 για την παραγωγή των δεδομένων των αισθητήρων. Η Python είναι διερμηνευόμενη, γενικού σκοπού και υψηλού επιπέδου, γλώσσα προγραμματισμού και 3.10.10 είναι η τελευταία της έκδοση την δεδομένη στιγμή[3].
* Kafka για την μεταφορά δεδομένων ανάμεσα στα services και την επεξεργασία τους μέσω streams. Kafka είναι μια distributed event streaming πλατφόρμα που χρησιμοποιείται για επεξεργασία ροών δεδομένων[4],[5].

Πιο συγκεκριμένα για την επεξεργασία των δεδομένων γίνεται χρήση των Kafka streams τα οποία αποτελούν μια βιβλιοθήκη ανάπτυξης εφαρμογών, όπου η είσοδος και η έξοδος αποθηκεύονται σε clusters του kafka[6].

* telegraf για την λήψη των δεδομένων από τον kafka και την αποστολή τους στην κατάλληλη μορφή, τόσο για την είσοδο τους στην time series βάση δεδομένων, όσο και στην αποστολή τους μέσω websockets στο grafana για την live απεικόνιση τους.

To telegraf είναι ένας ανοιχτού-κώδικα server agent που βασίζεται σε plugins και χρησιμοποιείται για την συλλογή και μεταφορά metrics[7].

* influxdb για την αποθήκευση των metrics που στέλνει το telegraf.

InfluxDB είναι μια Τime Series πλατφόρμα δεδομένων που χρησιμοποιείται κυρίως σε εφαρμογές Internet of Things, analytics και cloud[8].

* Grafana για την απεικόνιση των δεδομένων τόσο live όταν προέρχονται μέσω websockets από το telegraf, όσο και χωρίς αυτόματη ενημέρωση όταν προέρχονται από την influxdb

Grafana είναι ένα εργαλείο ανοιχτού-κώδικα συμβατό με διαφορετικούς τύπους υπολογιστών και λειτουργικών συστημάτων και χρησιμοποιείται για analytics και για οπτικοποίηση των δεδομένων. Παρέχει πολλών ειδών διαγράμματα, γραφικές παραστάσεις και ειδοποιήσεις για το web, όταν συνδέεται με υποστηριζόμενες πηγές δεδομένων[9].

# RESULTS-CONCLUSION

Tα αποτελέσματα της εφαρμογής φαίνονται στο τελικό στάδιο, όπου τα δεδομένα μπαίνουν στην βάση δεδομένων, αλλά και στην απεικόνιση τους από το Grafana τόσο live μέσω websocket, όσο και χωρίς αυτόματη ενημέρωση από την time series βάση δεδομένων influxdb.

Στις παρακάτω εικόνες παρουσιάζονται ενδεικτικά τα περιεχόμενα κάποιον measurements στην βάση δεδομένων.

Graphical user interface, text, application, chat or text message

Description automatically generated

Fig 8: Late rejected δεδομένα του 15λεπτου αισθητήρα w1 στην βάση δεδομένων.

Text

Description automatically generated

Fig 9: Μη επεξεργασμένα δεδομένα αισθητήρα th1 στην βάση δεδομένων.

Graphical user interface, text, application, chat or text message

Description automatically generated

Fig 10: Μη επεξεργασμένα ημερήσια δεδομένα αισθητήρα etot στην βάση δεδομένων.

Text

Description automatically generated

Fig 11: Hμερήσιος μέσος όρος δεδομένων του 15λεπτου αισθητήρα th1 στην βάση δεδομένων.

Graphical user interface, text

Description automatically generated

Fig 12: Hμερήσιο άθροισμα δεδομένων του 15λεπτου αισθητήρα hvac1 στην βάση δεδομένων.

H απεικόνιση των παραπάνω δεδομένων γίνεται στο Grafana, συνδέοντας το με την time series βάση δεδομένων.

Chart, line chart

Description automatically generatedChart

Description automatically generatedΚάποια από τα διαγράμματα και τους πίνακες που προέκυψαν παρουσιάζονται παρακάτω:

Fig13: Απεικόνιση των χωρίς επεξεργασία δεδομένων του αισθητήρα th1

Graphical user interface

Description automatically generated

Fig14: Αντίστοιχη table view απεικόνιση για τον 15λεπτο αισθητήρα th1

Chart

Description automatically generated

Fig15: Απεικόνιση ημερήσιου μέσου όρου δεδομένων του 15λεπτου αισθητήρα th1

Table

Description automatically generated with medium confidence

Fig16: Πίνακας με τα late rejected events του 15λεπτου αισθητήρα w1

Fig17: Πίνακας με τα μη επεξεργασμένα δεδομένα του ημερήσιου αισθητήρα wtot

Graphical user interface

Description automatically generated

Fig18: Πίνακας με τις ημερήσιες τιμές της διαρροής του νερού που προκύπτει αφαιρώντας από την συνολική ημερήσια κατανάλωση του νερού το άθροισμα των τιμών του 15λεπτου αισθητήρα κατανάλωσης νερού για την ίδια μέρα

Από όλα τα παραπάνω αντιλαμβανόμαστε την χρησιμότητα των IoT εφαρμογών, καθώς δίνεται η δυνατότητα με εύκολο και γρήγορο τρόπο να υπάρχει απεικόνιση των δεδομένων που προέρχονται από τους αισθητήρες και να υπάρχει γρήγορος εντοπισμός αν υπάρχει βλάβη σε κάποιον αισθητήρα και γρήγορη ενημέρωση για τις συνθήκες που επικρατούν στον χώρο, είτε μέσω παρατήρησης των raw μετρήσεων των αισθητήρων είτε μέσω των επεξεργασμένων δεδομένων, όπως είναι η διαρροή ενέργειας, διαρροή νερού ή και το ημερήσιο aggregation που εφαρμόζεται σε κάθε αισθητήρα.

# SCALABILITY

………..

# Acknowledgment

Θα θέλαμε να ευχαριστήσουμε την καθηγήτρια Vassiliki (Verena) Κantere και τον Παρασκευά Κερασιώτη για την πολύτιμη και άμεση βοήθεια τους, καθ’όλη την διάρκεια της εργασίας.

##### References

1. <https://el.wikipedia.org/wiki/Διαδίκτυο_των_πραγμάτων>
2. <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/>
3. <https://el.wikipedia.org/wiki/Python>
4. <https://el.wikipedia.org/wiki/Apache_Kafka>
5. <https://kafka.apache.org/>
6. <https://kafka.apache.org/documentation/streams/>
7. <https://www.influxdata.com/time-series-platform/telegraf/>
8. <https://www.influxdata.com/>
9. <https://en.wikipedia.org/wiki/Grafana>