ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Σχολή Τεχνολογίας

Τμήμα Ψηφιακών Συστημάτων

Ανάπτυξη Συστήματος IoT για την παρακολούθηση και τον έλεγχο των καιρικών συνθηκών

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Πλούταρχος Χίνης (ΑΜ: 3119218)

Επιβλέπων: ***Γεώργιος Αδάμ, καθηγητής***

ΛΑΡΙΣΑ, Φεβρουάριος 2024

UNIVERSITY OF THESSALY

School of Technology

Digital Systems Department

Development of an IoT System for weather conditions monitoring and control

BACHELOR THESIS

Ploutarchos Chinis (RN: 3119218)

Supervisor: ***Georgios Adam, professor***

LARISSA, February 2024

Υπεύθυνη Δήλωση περί Ακαδημαϊκής Δεοντολογίας και Πνευματικών Δικαιωμάτων

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα πτυχιακή εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον, και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.

Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο Δηλών

Πλούταρχος Χίνης
27/2/2024

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπων/πουσα	**Ονοματεπώνυμο Επιβλέποντα** Βαθμίδα/ιδιότητα επιβλέποντα, Τμήμα Ψηφιακών Συστημάτων, Πανεπιστήμιο Θεσσαλίας
Μέλος	**Ονοματεπώνυμο Μέλους 1** Βαθμίδα/ιδιότητα μέλους 1, Τμήμα/Ίδρυμα μέλους 1
Μέλος	**Ονοματεπώνυμο Μέλους 2** Βαθμίδα/ιδιότητα μέλους 2, Τμήμα/Ίδρυμα μέλους 2

Ημερομηνία έγκρισης: dd-mm-yyyy

Περίληψη

Η παρούσα πτυχιακή εργασία προτείνει τη δημιουργία ενός συστήματος Internet of Things (IoT) για την παρακολούθηση διαφόρων συνθηκών του περιβάλλοντος, σε συνδυασμό με την ενημέρωση του χρήστη μέσω εφαρμογής για φορητές συσκευές. Το σύστημα θα χρησιμοποιεί πολλαπλούς μικροελεγκτές ESP8266 εξοπλισμένους με αισθητήρα DHT22 για τη μέτρηση παραμέτρων περιβαλλοντικών συνθηκών, όπως η θερμοκρασία και η υγρασία. Μέσω του σωστού προγραμματισμού, οι αισθητήρες αυτοί θα συγκεντρώνουν σχετικά δεδομένα από το περιβάλλον της συσκευής. Έπειτα, τα συλλεγμένα δεδομένα θα μεταδίδονται μέσω MQTT. Οι χρήστες θα έχουν στη συνέχεια πρόσβαση σε αυτές τις πληροφορίες μέσω μιας εφαρμογής για φορητές συσκευές υλοποιημένη σε Flutter, όπου μπορούν να προβάλλουν τα δεδομένα και να προσαρμόσουν τις κλίμακες μέτρησης σύμφωνα με τις προτιμήσεις τους.

Abstract

This thesis proposes the creation of an Internet of Things (IoT) system to monitor various environmental conditions, combined with informing the user through a mobile application. The system will use multiple ESP8266 microcontrollers equipped with a DHT22 sensor to measure environmental parameters such as temperature and humidity. Through proper programming, these sensors will gather relevant data from the device's environment. The collected data will then be transmitted via MQTT. Users will then access this information through a mobile app implemented in Flutter, where they can view the data and adjust the measurement units according to their preferences.

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες προς τον κ. Γεώργιο Αδάμ για την εξαιρετική υποστήριξη ,την εκτίμηση και πάνω από όλα την υπομονή του κατά την εκπόνηση της πτυχιακής εργασίας μου.

Πλούταρχος Χίνης

27/2/2024

Περιεχόμενα

1 Εισαγωγή

Στην εποχή της ραγδαίας εξέλιξης των τεχνολογιών του Διαδικτύου των Αντικειμένων (IoT), η συλλογή δεδομένων έχει αναδειχθεί σε ζωτικό στοιχείο για ποικίλους τομείς της καθημερινότητας. Αντικατοπτρίζοντας την ανάγκη για προσεκτικό έλεγχο του περιβάλλοντος, η συγκεκριμένη τεχνολογία ανοίγει νέους ορίζοντες στην υγεία, την ασφάλεια, την έρευνα, και πολλούς άλλους τομείς. Τα είδη των δεδομένων που συλλέγονται διαφέρουν ανάλογα με τα προβλήματα που πρέπει να επιλυθούν. Συγκεκριμένα η μέτρηση της θερμοκρασίας και της υγρασίας μπορεί να δώσει λύση σε πάρα πολλά προβλήματα που αντιμετωπίζονται σε διάφορους τομείς όπως σε εργασιακά περιβάλλοντα, στη διατήρηση της φρεσκάδας των προϊόντων, σε χώρους καλλιέργειας και άλλα.

1.1 Σκοπός της πτυχιακής εργασίας

Η παρούσα πτυχιακή εργασία εστιάζει στην ανάπτυξη ενός ολοκληρωμένου συστήματος παρακολούθησης της θερμοκρασίας και της υγρασίας. Το σύστημα αξιοποιεί αισθητήρες DHT, οι οποίοι συλλέγουν μετρήσεις θερμοκρασίας και υγρασίας. Η επικοινωνία μεταξύ των αισθητήρων και του κεντρικού συστήματος πραγματοποιείται μέσω του πρωτοκόλλου MQTT, προσφέροντας ασύγχρονη και αξιόπιστη μεταφορά δεδομένων. Ο μικροελεγκτής ESP8266 διαδραματίζει καίριο ρόλο, λειτουργώντας ως γέφυρα μεταξύ των αισθητήρων και του ψηφιακού κόσμου.

Συμπληρωματικά, η εργασία περιλαμβάνει την ανάπτυξη μίας εφαρμογής σε Flutter. Η εφαρμογή λαμβάνει τα δεδομένα που συλλέγονται από τους αισθητήρες, τα επεξεργάζεται και τα απεικονίζει με εύληπτο τρόπο.

2 Θεωρητικό Υπόβαθρο

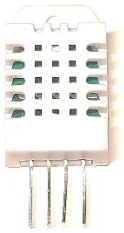
2.1 Διαδίκτυο των αντικειμένων (IOT)

Το Διαδίκτυο των αντικειμένων (Internet of Things - IoT) αναφέρεται στο δίκτυο φυσικών συσκευών που διαθέτουν ενσωματωμένους αισθητήρες, μικροελεγκτές και άλλες τεχνολογίες που τους επιτρέπει να συνδέονται και να ανταλλάσσουν δεδομένα με άλλες συσκευές και συστήματα μέσω διαδικτύου. Αυτή η τεχνολογία διευκολύνει την επικοινωνία και την αλληλεπίδραση μεταξύ συσκευών, επιτρέποντας τον ενιαίο συγχρονισμό δεδομένων και την αυτοματοποίηση διαφόρων εργασιών.

Κατά βάση, το IoT επεκτείνει τη σύνδεση στο διαδίκτυο σε πολλές άλλες συσκευές πέρα από τις παραδοσιακές όπως υπολογιστές και smartphones, σε μια ευρεία γκάμα καθημερινών αντικειμένων, συμπεριλαμβανομένων οικιακών συσκευών, οχημάτων, βιομηχανικού εξοπλισμού ακόμα και έξυπνων ρούχων. Αυτές οι διασυνδεδεμένες συσκευές μπορούν να επικοινωνήσουν μεταξύ τους και με διακομιστές, επιτρέποντας την ανταλλαγή δεδομένων και την αυτοματοποίηση διαφόρων εργασιών. [1]

2.2 Αισθητήρας DHT22

Ο αισθητήρας DHT22 συλλέγει μετρήσεις υγρασίας και θερμοκρασίας. Η ψηφιακή έξοδος που διαθέτει προσφέρει βαθμονομημένες τιμές με αποτέλεσμα να υπάρχει αυξημένη ακρίβεια και σταθερότητα. Ουσιαστικά πρόκειται για δυο διαφορετικά στοιχεία αισθητήρων ένα για την θερμοκρασία και ένα για την Υγρασία συνδεμένα με ένα μικροτσίπ 8-bit. Η κατανάλωση ενέργειας είναι πολύ μικρή καθιστώντας τον αισθητήρα να μπορεί να τροφοδοτηθεί από πηγές ενέργειας χαμηλής ισχύος όπως μπαταρίες. Επίσης λόγο του μικρού του μεγέθους και του χαμηλού κόστους χρησιμοποιείται σε μια πληθώρα από εφαρμογές. [3]



Εικόνα 1 Αισθητήρας DHT22

2.3 Πρωτόκολλο επικοινωνίας MQTT

Το πρωτόκολλο Message Queuing Telemetry Transport (MQTT) χρησιμοποιείται ευρέως για την επικοινωνία στο Internet of things (IoT) και την επικοινωνία μεταξύ μηχανών (M2M) σε δικτυακά περιβάλλοντα. Μπορεί να εκτελεστεί σε μικροελεγκτές 8-bit και συσκευές με περιορισμένη μνήμη RAM (π.χ. 256 kB) καθώς έχει πολύ ελαφριά κωδικοποίηση. Επιπλέον, το MQTT λειτουργεί αποδοτικά με χαμηλή κατανάλωση ενέργειας, χαμηλή χρήση εύρους ζώνης και είναι προσαρμόσιμο σε συνδέσεις μεταβλητής διαθεσιμότητας.

2.3.1 Θέματα MQTT

Το πρωτόκολλο MQTT, βασίζεται στην αρχή της δημοσίευσης μηνυμάτων και της συνδρομής σε θέματα. Πολλές συσκευές-πελάτες συνδέονται σε διακομιστές που φιλοξενούν πολλούς πόρους και υπηρεσίες. Οι λεπτομέρειες των πόρων και των υπηρεσιών αποθηκεύονται σε θέματα. Οι συσκευές μπορούν να συνδεθούν και να εγγραφούν σε θέματα που τις αφορούν. Οι πελάτες συνδέονται στο διακομιστή και δημοσιεύουν μηνύματα σε αυτά τα θέματα. Πολλές συσκευές-πελάτες μπορούν να εγγραφούν στα ίδια θέματα. Ένας διακομιστής μπορεί να δημοσιεύσει ένα μήνυμα την φορά το οποίο μπορεί να ληφθεί από πολλούς συνδρομητές. Τα θέματα δομούνται σε δέντρα, τα οποία χρησιμοποιούνται ως ιεραρχίες, χρησιμοποιώντας την κάθετο (/) ως διαχωριστικό. Αυτό επιτρέπει τη δημιουργία ομαδοποιημένων θεμάτων. Τα θέματα και τα δέντρα θεμάτων μπορούν να δημιουργηθούν εξ αρχής, αν και είναι πιο συνηθισμένο για ένα διακομιστή να δημιουργεί ένα θέμα κατόπιν αιτήματος (υπόκειται σε πολιτικές ασφαλείας) όταν μία συσκευή-πελάτης προσπαθεί για πρώτη φορά να δημοσιεύσει ή να εγγραφεί σε αυτό. Μια συνδρομή μπορεί να περιέχει ειδικούς χαρακτήρες, οι οποίοι επιτρέπουν στους πελάτες να εγγραφούν σε

πολλά θέματα ταυτόχρονα, εντός του ίδιου επιπέδου ή σε πολλά επίπεδα σε ένα δέντρο θεμάτων. [1]

2.3.2 Ποιότητα υπηρεσίας (QOS):

Τα επίπεδα ποιότητας υπηρεσίας (QOS) στο πρωτόκολλο MQTT ορίζουν το επίπεδο αξιοπιστίας και αναγνώρισης των μηνυμάτων μεταξύ του καταναλωτή και του διακομιστή. Ας εξετάσουμε τα τρία επίπεδα QOS:

QOS 0

Σε αυτό το επίπεδο, τα μηνύματα στέλνονται από τον πελάτη στο διακομιστή χωρίς να αναμένεται απόδειξη παραλαβής (acknowledgement). Δεν υπάρχει εγγύηση ότι το μήνυμα θα ληφθεί από τον παραλήπτη.

QOS 1

Σε αυτό το επίπεδο, η συσκευή-πελάτης αποστέλλει το μήνυμα στο διακομιστή και περιμένει αναγνώριση (acknowledgement) για την παραλαβή του από τον παραλήπτη. Αυτό εξασφαλίζει τουλάχιστον μία επιτυχημένη επικοινωνία.

QOS 2

Σε αυτό το επίπεδο, η συσκευή MQTT στέλνει το μήνυμα τουλάχιστον μία φορά στον διακομιστή με τέτοιο τρόπο ώστε να φτάσει στον συνδρομητή πριν διαγραφεί από την ουρά μηνυμάτων. Αυτό εξασφαλίζει μία ασφαλή παράδοση του μηνύματος.

Οι επιλογή επιπέδου QOS επηρεάζει την αξιοπιστία και την απόδοση του συστήματος MQTT. Η επιλογή του κατάλληλου επιπέδου QOS εξαρτάται από τις απαιτήσεις της εφαρμογής και την κρισιμότητα των δεδομένων. [1]

2.4 Flutter framework

Το πλαίσιο Flutter αποτελεί μια επαναστατική τεχνολογία για την ανάπτυξη εφαρμογών κινητών συσκευών, παρέχοντας στους προγραμματιστές μια πλήρη εργαλειοθήκη για τη δημιουργία εξαιρετικών εφαρμογών χωρίς να κάνουν συμβιβασμούς στην απόδοση και την κλιμάκωση. Ο πυρήνας του Flutter περιλαμβάνει διάφορες έννοιες που εστιάζουν στη βελτιστοποίηση της απόδοσης της εφαρμογής και της διεπαφής χρήστη. Χρησιμοποιώντας τη γλώσσα προγραμματισμού Dart, το Flutter στοχεύει στην παροχή ασύγκρι-

της παραγωγικότητας στους προγραμματιστές κατά την ανάπτυξη. Επίσης το Flutter, επιτρέπει στους προγραμματιστές να δημιουργούν και να διανέμουν εφαρμογές υψηλής απόδοσης σε πολλαπλές πλατφόρμες. [3]

```
import 'package:flutter/material.dart';

void main() {
  runApp(TempHumidApp());
}

class TempHumidApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Temp & Humidity'),
        ),
        body: Center(
          child: Text(
            'Temperature: 22°C, Humidity: 55%',
            style: TextStyle(fontSize: 18),
          ),
        ),
      ),
    );
  }
}
```

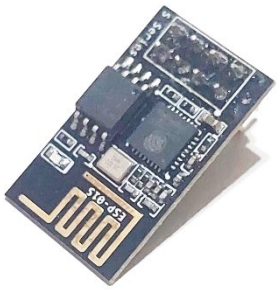
Παράδειγμα απλής εφαρμογής στο flutter

2.5 μικροελεγκτής ESP8266

Ο ESP8266 είναι ένας μικροελεγκτής μονού τσιπ και έχει αναπτύχθηκε από την Espressif. Μπορεί να χρησιμοποιηθεί ειδικά για την ανάπτυξη λύσεων Wi-Fi χαμηλής κατανάλωσης και εύκολης ενσωμάτωσης για φορητές ηλεκτρονικές συσκευές και εφαρμογές Internet of Things (IoT). [4]

Τα Κύρια χαρακτηριστικά του είναι τα εξής:

- Μικρό μέγεθος
- Ανθεκτικότητα σε υπερβολικές θερμοκρασίες (-40°C έως + 125°C)
- Προσφέρει την ευελιξία ενός System On Chip (SoC)
- Χαμηλή κατανάλωση ενέργειας



Εικόνα 2 μικροελεγκτής ESP8266 (01s)

3 Σχεδίαση και Υλοποίηση Συστήματος

3.1 Αρχιτεκτονική του συστήματος

Η αρχιτεκτονική του συστήματος βασίζεται σε ένα συνδυασμό υλικού και λογισμικού για τη συλλογή και τη μεταφορά δεδομένων από τους αισθητήρες DHT σε έναν κεντρικό διακομιστή MQTT. Ας εξετάσουμε αναλυτικά τα συστατικά της αρχιτεκτονικής:

Αισθητήρες

Οι αισθητήρες DHT είναι αυτόνομες συσκευές που χρησιμοποιούνται για τη μέτρηση θερμοκρασίας και υγρασίας σε διάφορα περιβάλλοντα. Κάθε αισθητήρας συνδέεται σε ένα μικροελεγκτή τύπου ESP01.

Μικροελεγκτής ESP01

Ο ESP01 είναι ένας μικροελεγκτής βασισμένος στο ESP8266 και λειτουργεί ως ο αποστολέας του συστήματος. Συλλέγει δεδομένα από τους αισθητήρες DHT και τα μεταφέρει μέσω του πρωτοκόλλου MQTT σε έναν κεντρικό διακομιστή.

Κεντρικός Διακομιστής MQTT

Ο κεντρικός διακομιστής MQTT λειτουργεί ως μεσολαβητής μεταξύ του ESP01 και της εφαρμογής android που λαμβάνει τα δεδομένα των αισθητήρων.

Εφαρμογή android

Η εφαρμογή είναι υπεύθυνη ώστε να λάβει τα δεδομένα και να τα προβάλει στον χρήστη σε μορφή που μπορεί να τα διαβάσει.

Η αρχιτεκτονική αυτή επιτρέπει τη συλλογή, τη μεταφορά και τη διανομή των δεδομένων αισθητήρων σε πραγματικό χρόνο, ενώ παράλληλα παρέχει ευελιξία.

Εικόνα 3 Σχεδιάγραμμα αρχιτεκτονικής συστήματος

3.2 Υλοποίηση του πρωτοκόλλου MQTT στο ESP8266

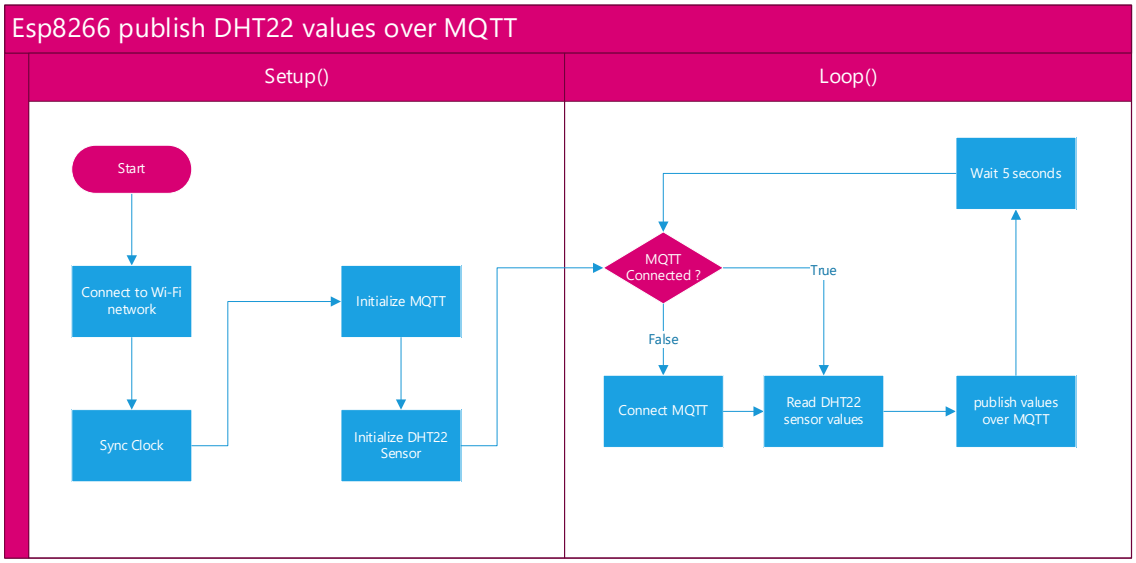
Το ESP-01S αποτελεί το πιο σημαντικό κομμάτι του συστήματος καθώς αυτό διαβιβάζει τις πληροφορίες στην ουρά μηνυμάτων ώστε να έχουν πρόσβαση οι συσκευές-συνδρομητές. Ο αλγόριθμος έχει ως εξής:

Αρχικοποίηση setup():

1. **Σύνδεση στο Wi-Fi**: Συνδέεται στο δίκτυο Wi-Fi χρησιμοποιώντας τα παρεχόμενα διαπιστευτήρια.
2. **Ρύθμιση του ρολογιού**: Χρησιμοποιεί μια διεύθυνση NTP(Network Time Protocol) για να συγχρονίσει το ρολόι της συσκευής.
3. **Ρύθμιση MQTT**: Ορίζει τον broker που πρέπει η συσκευή να συνδεθεί και τα θέματα(topics) για την ανάρτηση των δεδομένων του αισθητήρα.
4. **Ενεργοποίηση αισθητήρα DHT22**: Αρχικοποιεί τον αισθητήρα DHT22.

Επανάληψη loop():

1. **Διασύνδεση MQTT**: Αν η σύνδεση με τον broker MQTT ή με το δίκτυο διακοπεί, πραγματοποιούνται προσπάθειες για την επανασύνδεση στον broker.
2. **Ανάγνωση αισθητήρων DHT22**: Διαβάζει τη θερμοκρασία και την υγρασία από τον αισθητήρα DHT22.
3. **Ανάρτηση στοιχείων MQTT**: Αναρτά τα δεδομένα του αισθητήρα στα θέματα (topics) του προτοκόλου MQTT.
4. **Διακοπή για 5 δευτερόλεπτα**: Πραγματοποιεί μια διακοπή για 5 δευτερόλεπτα πριν επαναλάβει τον κύκλο εκ νέου.



Εικόνα 4 διάγραμμα ροής αλγόριθμου

3.3 Σύνδεση αισθητήρων DHT με το ESP8266

Τροφοδοσία (VCC)

Ο ακροδέκτης VCC του αισθητήρα DHT22 όπως και ο αντίστοιχός του ESP-01S συνδέθηκε στον θετικό πόλο της τροφοδοσίας, δηλαδή το θετικό πόλο των μπαταριών.

Γείωση (GND)

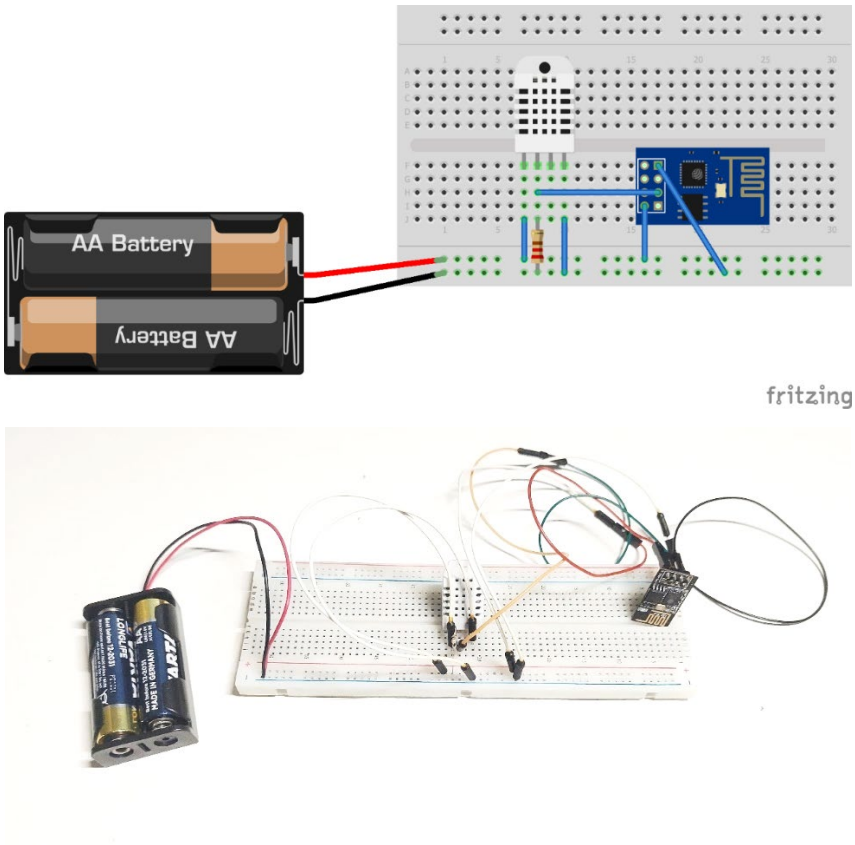
Ο ακροδέκτης GND του αισθητήρα DHT22 όπως και ο αντίστοιχος του ESP-01S συνδέθηκε στον αρνητικό πόλο της τροφοδοσίας, που αποτελεί τον αρνητικό των μπαταριών.

Σύνδεση Δεδομένων (DATA)

Ο ακροδέκτης δεδομένων (DATA) του αισθητήρα DHT22 συνδέθηκε σε έναν από τους ψηφιακούς ακροδέκτες εισόδου / εξόδου (GPIO) του ESP-01S, για παράδειγμα στον ακροδέκτη GPIO2 (4^{ο} pin στο ESP-01S).

Αντίσταση Pull-up (προαιρετικά)

Μια αντίσταση pull-up 4.7kΩ μπορεί να συνδεθεί ανάμεσα στον ακροδέκτη DATA και την τροφοδοσία (VCC), προκειμένου να διατηρήσει το σήμα DATA σε υψηλό (HIGH) επίπεδο όταν δεν γίνεται ανάγνωση από τον αισθητήρα ώστε να μην υπάρξουν λανθασμένες μετρήσεις.



Εικόνα 5 Διάταξη σύνδεσης esp8266 (01s) με dht22 πάνω σε breadboard

3.4 Ανάπτυξη εφαρμογής Flutter

Η εφαρμογή υλοποιήθηκε σε περιβάλλον Flutter και έχει σχεδιαστεί για να συνδέεται σε έναν μεσίτη MQTT, να εγγράφεται σε συγκεκριμένα θέματα (topics) και να εμφανίζει δεδομένα αισθητήρων σε πραγματικό χρόνο από αυτά τα θέματα σε ένα γράφημα.

3.4.1 Εισαγωγή απαραίτητων βιβλιοθηκών

d_chart: Πακέτο για τη δημιουργία διαδραστικών γραφημάτων.

flutter/material.dart: Η βιβλιοθήκη Material ui του Flutter για έτοιμα στοιχεία διεπαφής χρήστη (UI).

mqtt_client: Ένα πακέτο για την επικοινωνία με την χρήση MQTT.

hive_flutter και **hive**: Πακέτα για αποθήκευση δεδομένων σε τοπική μη σχεσιακή βάση δεδομένων.

3.4.2 Κύρια Συνάρτηση

Η συνάρτηση main() αρχικοποιεί την βάση δεδομένων Hive, για την τοπική αποθήκευση και ξεκινάει την εφαρμογή.

```
void main() async {
  await Hive.initFlutter();
  await Hive.openBox('topics');
  runApp(const MqttSense());
}
```

3.4.3 Εκκίνηση εφαρμογής

Αρχικοποίηση

Στη μέθοδο initState(), η σύνδεση MQTT με τον μεσίτη διαμορφώνεται χρησιμοποιώντας τη μέθοδο connectToMQTTBroker().

```
@override
  void initState() {
    super.initState();
    try {
      topics = box.getAt(0);
    } catch (e) {
      topics = [];
    }

    connectToMQTTBroker();
  }
```

Σύνδεση MQTT

Η μέθοδος connectToMQTTBroker() ρυθμίζει τον πελάτη MQTT με τα διαπιστευτήρια του μεσίτη και συνδέεται σε αυτόν. Επίσης, δημιουργεί callbacks για την κατάσταση σύνδεσης, την εγγραφή και τη λήψη μηνυμάτων.

```
void connectToMQTTBroker() async {
    client = MqttServerClient.withPort(mqttBroker, clientId, port);
    //credentials
    client.secure = true;

    client.autoReconnect = true;

    client.keepAlivePeriod = 100;

    client.onConnected = () {
      print('Connected to MQTT broker');
      subscribeToTopics();
    };

    client.onDisconnected = () {
      print('Disconnected from MQTT broker');
    };

    client.onSubscribed = (String topic) {
```

```
      print('Subscribed to topic: $topic');
    };

    try {
      await client.connect(username, password);
    } catch (e) {
      print('Error connecting to MQTT broker: $e');
    }
  }
```

Εγγραφή σε Θέματα(topics)

Η μέθοδος subscribeToTopics() εγγράφεται στα θέματα που είναι αποθηκευμένα στον πίνακα 'topics' χρησιμοποιώντας το πακέτο mqtt_client.

```
void subscribeToTopics() {
    if (topics == null) return;

    for (String topic in topics!) {
      client.subscribe(topic, MqttQos.atLeastOnce);
    }

    client.updates?.listen((List<MqttReceivedMessage<MqttMessage>> c) {
      final MqttPublishMessage message = c[0].payload as MqttPublishMessage;
      final String payload =
          MqttPublishPayload.bytesToStringAsString(message.payload.message);

      setState(() {
        if (sensorData[message.variableHeader!.topicName] == null) {
          sensorData[message.variableHeader!.topicName] = [];
        }
        sensorData[message.variableHeader!.topicName]!.add(
            TimeData(domain: DateTime.now(), measure: double.parse(payload)));
      });
    });
  }
```

3.4.4 Δημιουργία UI

Η μέθοδος build() κατασκευάζει την διεπαφή χρήστη (UI) της SensorPage. Περιλαμβάνει:

- **Widget γραφήματος** (DChartComboT) για να εμφανίζει δεδομένα αισθητήρων σε πραγματικό χρόνο.
- **Μια λίστα με τα εγγεγραμμένα θέματα** που εμφανίζονται ως πλακίδια, μαζί με τα πιο πρόσφατα δεδομένα αισθητήρων.
- **Κουμπί FloatingActionButton** για την προσθήκη νέων θεμάτων που καλεί την συνάρτηση _showAddTopicDialog();.

```
@override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
```

```
      title: const Text('Mqtt Sense'),
    ),
    body: Column(
      children: [
        if (topics!.isEmpty) ...[
          const Expanded(
            child: Center(
              child: Text('No topics added yet'),
            ),
          ),
        ] else ...[
          Container(
            height: 420,
            color: Colors.white,
            child: AspectRatio(
              aspectRatio: 16 / 9,
              child: DChartComboT(
                configRenderLine: ConfigRenderLine(
                  includeLine: true,
                  includePoints: true,
                  includeArea: true,
                ),
                domainAxis: const DomainAxis(
                  showLine: false,
                  gapAxisToLabel: 2,
                  labelStyle: LabelStyle(
                    color: Colors.black,
                    fontSize: 12,
                  ),
                  thickLength: 5,
                ),
                outsideBarLabelStyle: (group, timeData, index) => LabelStyle(
                  color: Colors.black,
                  fontSize: 12,
                ),
                layoutMargin: LayoutMargin(40, 30, 5, 30),
                measureAxis: MeasureAxis(
                  showLine: false,
                ),
                groupList: [
                  for (var topic in topics!)
                    if (sensorData[topic] != null)
                      TimeGroup(
                        color: Colors.primaries[
                            topics!.indexOf(topic) % Colors.primaries.length],
                        id: topic,
                        chartType: ChartType.line,
                        data: sensorData[topic] ?? [],
                      )
                ],
              ),
            ),
          ),
          //sliding
          Expanded(
            child: Scrollbar(
              child: ListView.builder(
                primary: false,
                itemCount: topics!.length,
```

```
                itemBuilder: (context, index) {
                  final topic = topics![index];
                  final data = sensorData[topic]?.last.measure ?? 'N/A';
                  return ListTile(
                    textColor: Colors.white,
                    tileColor: Colors.primaries[
                        topics!.indexOf(topic) % Colors.primaries.length],
                    title: Text('$topic: $data'),
                    // delete topic
                    trailing: IconButton(
                      icon: const Icon(Icons.delete),
                      onPressed: () {
                        setState(() {
                          topics!.remove(topic);
                          box.put(0, topics);
                          sensorData.remove(topic);
                        });
                        client.unsubscribe(topic);
                      },
                    ),
                  );
                },
              ),
            ),
          ),
        ),
      ],
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () {
        _showAddTopicDialog();
      },
      child: const Icon(Icons.add),
    ),
  );
 }
```

Διάλογος Προσθήκης Θέματος

Η μέθοδος _showAddTopicDialog() εμφανίζει ένα διάλογο για την εισαγωγή ενός νέου θέματος από τον χρήστη. Μετά την επιβεβαίωση, προσθέτει το θέμα στη λίστα των εγγεγραμμένων θεμάτων και ενημερώνει το UI αναλόγως.

3.4.5 state management

Η κατάσταση της εφαρμογής ελέγχεται με την χρήση της μεθόδου setState() ώστε να ενημερώνεται το UI κάθε φορά που λαμβάνονται νέα δεδομένα αισθητήρων ή όταν προστίθενται ή αφαιρούνται θέματα (MQTT topics).

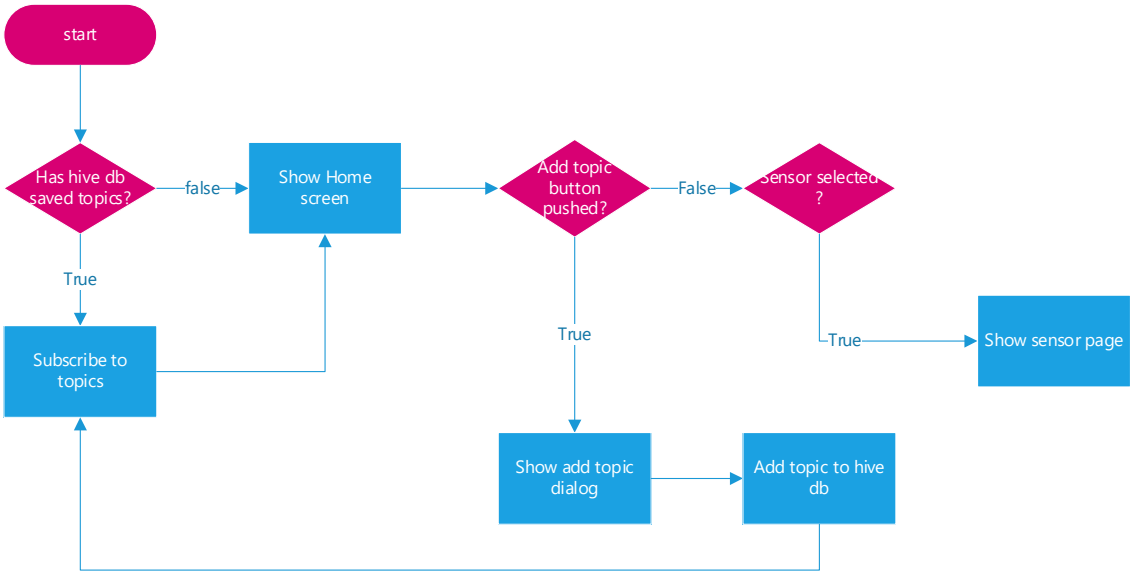
```
Future<void> _showAddTopicDialog() async {
    TextEditingController topicController = TextEditingController();

    return showDialog<void>(
      context: context,
```

```
    builder: (BuildContext context) {
      return AlertDialog(
        title: const Text('Add New Topic'),
        content: TextField(
          autofocus: true,
          controller: topicController,
          decoration: const InputDecoration(labelText: 'Topic'),
        ),
        actions: <Widget>[
          TextButton(
            onPressed: () {
              Navigator.of(context).pop();
            },
            child: const Text('Cancel'),
          ),
          TextButton(
            onPressed: () {
              String newTopic = topicController.text.trim();

              if (newTopic.isNotEmpty && !topics!.contains(newTopic)) {
                setState(() {
                  topics!.add(newTopic);
                  box.put(0, topics);
                });
                client.subscribe(newTopic, MqttQos.atMostOnce);
                Navigator.of(context).pop();
              }
            },
            child: const Text('Add'),
          ),
        ],
      );
    },
  );
}
```

Ο κώδικας υπάρχει ολοκληρωμένος στο παράρτημα 2



Εικόνα 6 διάγραμμα ροής εφαρμογής flutter

3.5 Πλοήγηση στην εφαρμογή

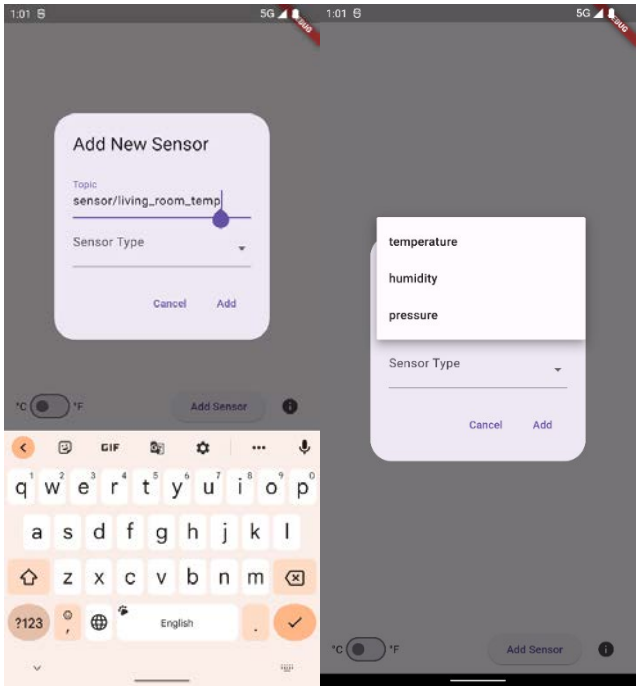
3.5.1 Αρχική οθόνη

Κατά την εκκίνηση της εφαρμογής, ο χρήστης παρατηρεί μια σύνοψη των αισθητήρων που είναι συνδεδεμένοι και τις πιο πρόσφατες τους τιμές. Μέσω αυτής της οθόνης, μπορεί να παρακολουθήσει γρήγορα την τρέχουσα κατάσταση του συστήματος. Επίσης παρέχεται η δυνατότητα επιλογής μονάδας μέτρησης (βαθμούς Κελσίου-C ή Φαρενάιτ-F)

Εικόνα 7 αρχική οθόνη

3.5.2 Προσθήκη αισθητήρα

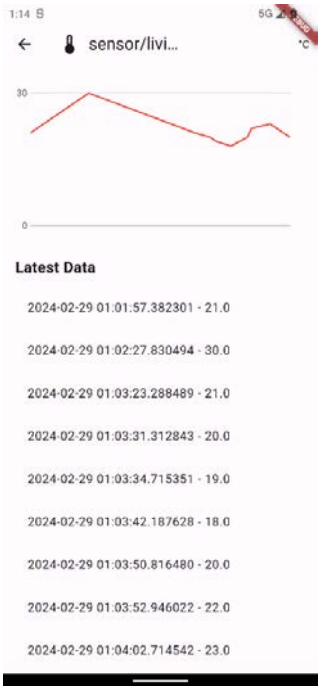
Σε αυτήν την ενότητα, ο χρήστης μπορεί να προσθέσει νέα θέματα MQTT για παρακολούθηση. Μέσω αυτής της λειτουργίας, είναι δυνατή η προσαρμογή της εφαρμογής στις ανάγκες του χρήστη. προσθέτονται νέοι αισθητήρες με εισαγωγή του θέματος MQTT και τον τύπο μετρήσεων (θερμοκρασία ή υγρασία).



Εικόνα 8 Διάλογος προσθήκης αισθητήρα

3.5.3 Οθόνη αισθητήρα

Σε αυτήν την οθόνη, ο χρήστης μπορεί να παρακολουθήσει τις πιο πρόσφατες μετρήσεις που έχουν ληφθεί από ένα συγκεκριμένο αισθητήρα. Αναγράφονται ο τύπος του αισθητήρα, η τιμή που μετρήθηκε και η χρονική στιγμή της μέτρησης, παρέχοντας έτσι μια ολοκληρωμένη εικόνα. Επιπλέων υπάρχει συνοπτικό διάγραμμα για αυτές τις τιμές



Εικόνα 9 οθόνη τελευταίων στοιχείων αισθητήρα

4 Πειραματικά αποτελέσματα

4.1 Δοκιμή αύξησης φόρτου

Σε αυτήν τη δοκιμή, θέσαμε ως στόχο την αξιολόγηση της απόδοσης του συστήματος υπό διαφορετικά επίπεδα φόρτου εργασίας. Με την χρήση του εργαλείου JMeter [5] και της επέκτασης (plugin) mqtt-jmeter [6], προσομοιώθηκε η αποστολή μεγάλου αριθμού αιτημάτων προς τον διακομιστή για μέτρηση της απόκρισής του σε διαφορετικά επίπεδα φορτίου. Οι μετρήσεις περιλαμβάνουν τους χρόνους απόκρισης του συστήματος κατά τη διάρκεια αυτών των φορτίων. Παρακάτω παρατίθενται τα χαρακτηριστικά των υπολογιστικών συστημάτων στα οποία έγινε η δοκιμή και η φιλοξενία του MQTT broker.

Χαρακτηριστικά συστήματος δοκιμών

Λειτουργικό σύστημα: Microsoft Windows 11 pro

Ram: 16Gb 3600 MHz

CPU: AMD Ryzen 5-2500

Χαρακτηριστικά συστήματος φιλοξενίας

Λειτουργικό σύστημα: Ubuntu server

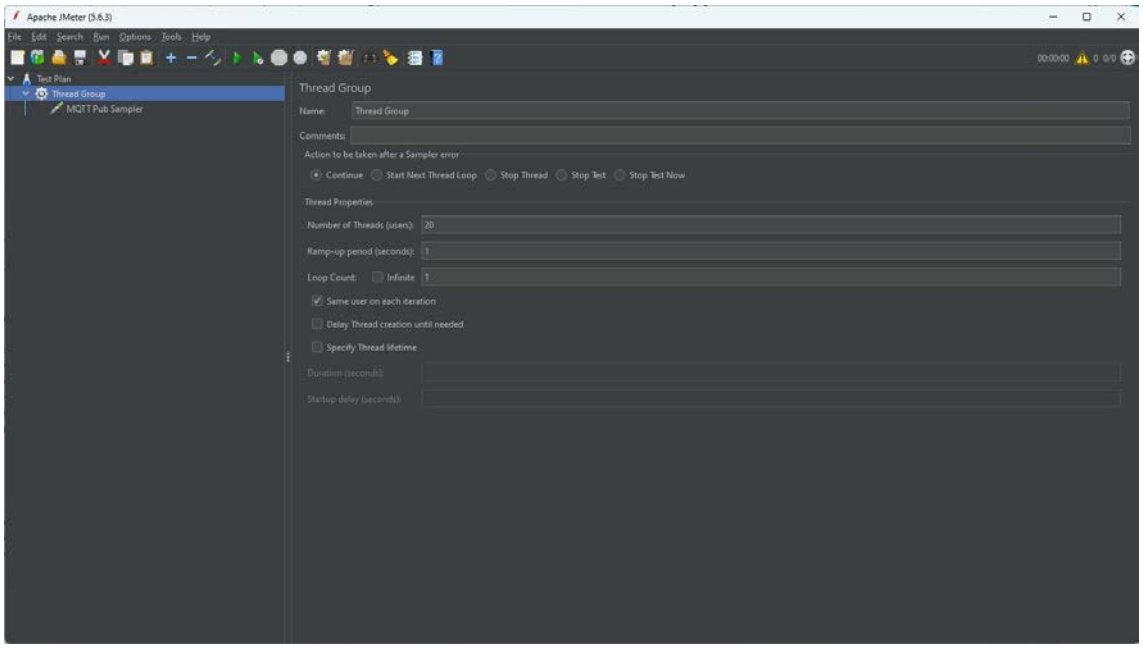
Ram: 8Gb 3600 MHz

CPU: Intel Core i7-9700

MQTT broker: EMQX

Για την διενέργεια όλων των ελέγχων χρησιμοποιήθηκε το παρακάτω δοκιμαστικό μήνυμα σε Json.

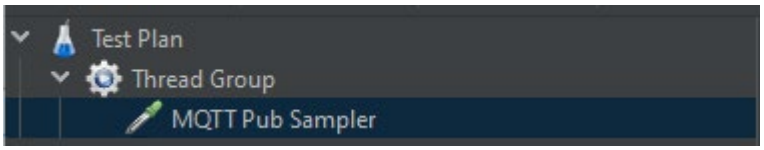
```
{
“temperature”: 20.3,
“humidity”: 64
}
```



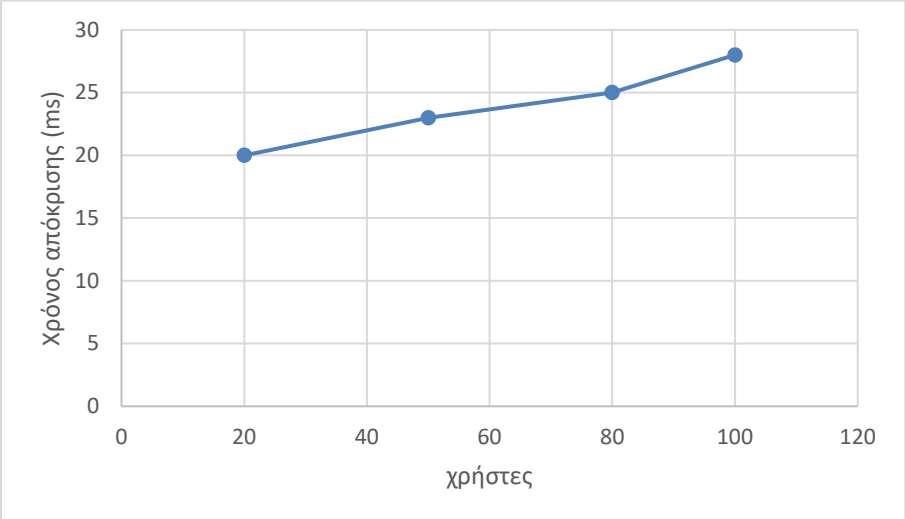
Εικόνα 10 Apache JMeter

4.1.1 Αύξηση φόρτου με εικονικές συσκευές-πελάτες

Ο έλεγχος πραγματοποιήθηκε με χρήση εικονικών συσκευών πελάτη, οι οποίες λειτούργησαν ως προσομοίωση πραγματικών συσκευών που συνδέονται στο σύστημα.



Εικόνα 11 δημιουργία νημάτων εικονικών χρηστών στο JMeter

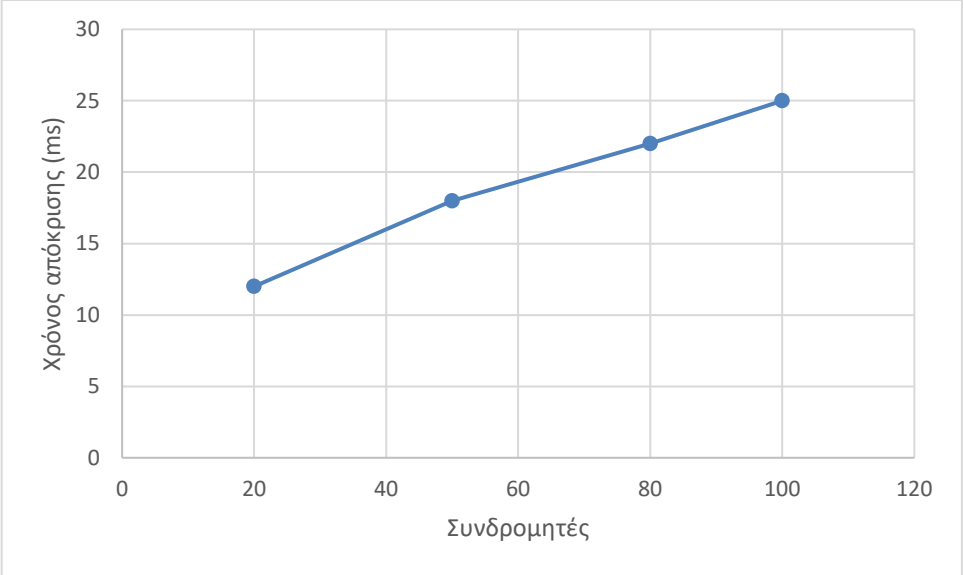


Ο χρόνος απόκρισης του συστήματος αυξάνεται με γραμμικό τρόπο όσο αυξάνεται ο αριθμός των συνδεδεμένων συσκευών. Ακόμα και με 100 συνδεδεμένες συσκευές, ο χρόνος απόκρισης παραμένει κάτω από 50ms, Άρα η απόδοση του συστήματος θεωρείται ικανοποιητική.

4.1.2 Αύξηση φόρτου με εικονικούς συνδρομητές

Ο έλεγχος πραγματοποιήθηκε με χρήση εικονικών συσκευών πελατών, οι οποίες λειτούργησαν ως προσομοίωση πραγματικών συνδρομητών που συνδέονται στο σύστημα.

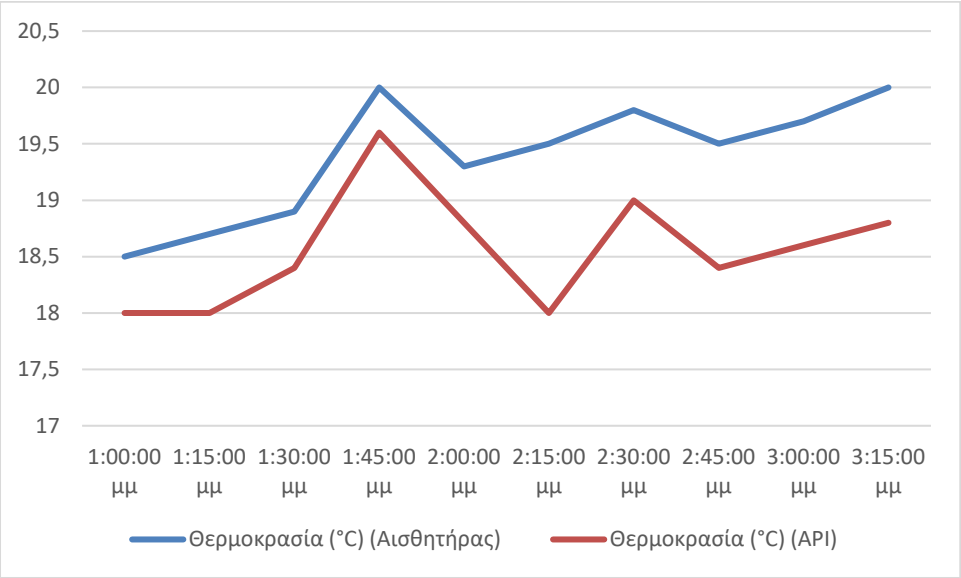
Εικόνα 12 δημιουργία νημάτων εικονικών συνδρομητών στο JMeter



Ο χρόνος απόκρισης και πάλι θεωρείται ικανοποιητικός αυτό όμως που διακρίνεται είναι ότι η έκδοση μηνυμάτων απαιτεί περισσότερους πόρους από την συνδρομή καθώς γίνεται broadcast.

4.2 Σύγκριση τιμών με OpenWeather

Για την επαλήθευση των μετρήσεων που πραγματοποιεί το προτεινόμενο σύστημα, υπήρξε σύγκριση των τιμών εξόδου με τις αντίστοιχες μετρήσεις από την υπηρεσία παροχής δεδομένων καιρού OpenWeather [7]. Αρχικά, δημιουργήθηκε ένα πρόγραμμα επικοινωνίας με το API του OpenWeather με την γλώσσα python (παράρτημα Γ) για να ανακτηθούν τα δεδομένα για την τρέχουσα τοποθεσία και να αποθηκευτούν σε αρχείο τύπου CSV. Κατόπιν, συγκρίναμε αυτά τα δεδομένα με τις μετρήσεις που πραγματοποιεί το προτεινόμενο σύστημα.



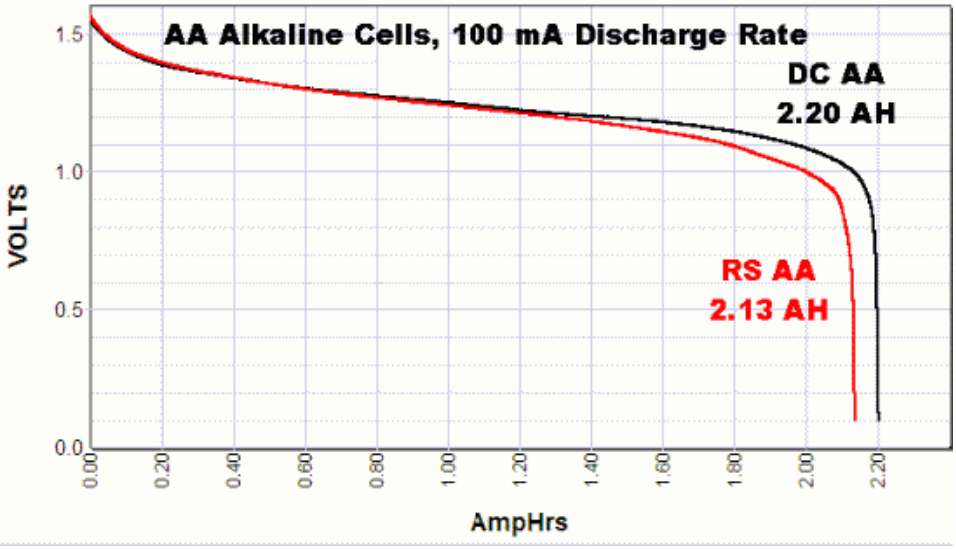
ΔΙΑΓΡΑΜΜΑ 1 θερμοκρασία που παρέχει ο αισθητήρας σε σχέση με την αυτήν του OpenWeather (οι μετρήσεις πραγματοποιήθηκαν στις 24/02/2024)

Τα αποτελέσματα έδειξαν ότι οι μετρήσεις που παρήχθησαν από το προτεινόμενο σύστημα εμφανίζουν συνολικά μια συμβατότητα και συμφωνία με τα δεδομένα που ανακτήθηκαν από το OpenWeather API. Ωστόσο, παρατηρήθηκαν κάποιες μικρές αποκλίσεις , οι οποίες μπορούν να αποδοθούν σε διαφορές στην τοποθεσία των αισθητήρων ή στις διαφορετικές συνθήκες μέτρησης. Συνολικά, όμως, τα δεδομένα επιβεβαίωσαν την αξιοπιστία και την ακρίβεια του προτεινόμενου συστήματος.

4.3 Δοκιμή πτώσης τάσης τροφοδοσίας

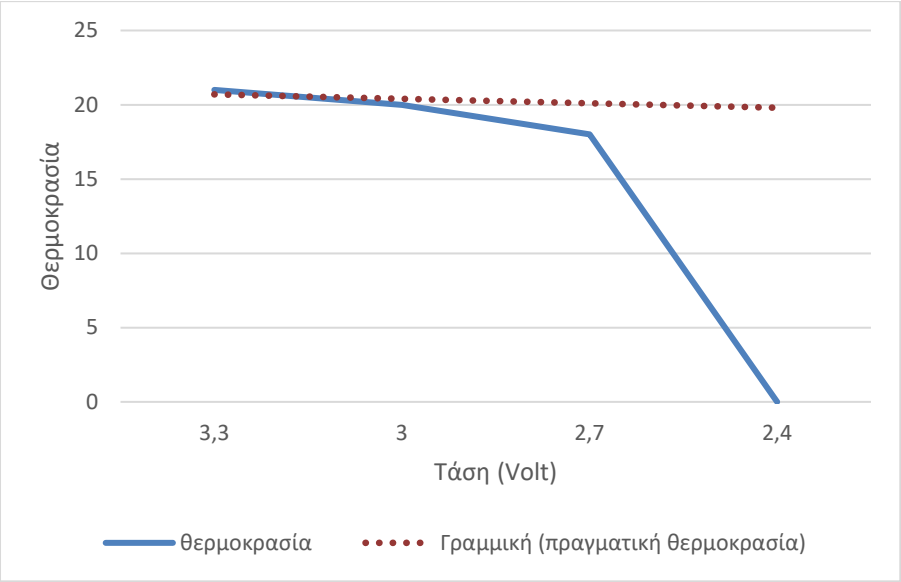
Σε αυτήν τη δοκιμή, αξιολογείται η αντοχή του συστήματος σε περιόδους χαμηλής τάσης τροφοδοσίας, κάνοντας χρήση ποτενσιόμετρου το οποίο είναι συνδεμένο πάνω στην

πηγή ενέργειας(μπαταρίες). Ταυτόχρονα μετρήθηκαν οι πραγματικές τιμές με αντίστοιχη διάταξη που τροφοδοτούνταν από σταθερή πηγή ενέργειας (τροφοδοτικό DC). Καταγράφεται η συμπεριφορά του συστήματος και το επίπεδο απόδοσης του κατά τη διάρκεια τέτοιων καταστάσεων. Η δοκιμή αυτή ήταν πολύ σημαντική, καθώς υπολογίστηκε η συνολική ένταση της διάταξης στα 24 mA ενώ η πηγή ενέργειας που επιλέχθηκε για την τροφοδοσία της διάταξης του αισθητήρα ήταν μπαταρίες (2 αλκαλικές τύπου AA). Οι μπαταρίες λόγο της κατασκευής τους είναι ευάλωτες σε πτώση της τάσης μετά από σχετικά σύντομο χρονικό διάστημα από την εγκατάσταση τους, καθώς αποφορτίζονται.



Εικόνα 13 διάγραμμα αποφόρτισης αλκαλικών μπαταριών τύπου AA [5]

Περίοδος τάσης	**Απόδοση (%)**
Κανονική Τάση (3.3 V)	100
Χαμηλή Τάση (2.7 V)	90
Πολύ Χαμηλή Τάση (<2.7 V)	0



ΔΙΑΓΡΑΜΜΑ 2 Διαβάθμισή τάσης και συσχέτιση με την θερμοκρασία που εξάγει ο αισθητήρας

Οι μπαταρίες ΑΑ έχουν χωρητικότητα ίση με 1 Ah (Αμπερώριο) έως ότου η τάση τους υποχωρήσει κάτω από 2,7V όπου το σύστημα δεν θα μπορεί πλέων να αποδώσει. Χρησιμοποιώντας τον τύπο $Time(H) = \frac{Capacity(Ah)}{Current(A)}$ υπάρχει η υπόθεση ότι σε ιδανικές συνθήκες η αυτονομία θα είναι $H = \frac{1}{0,24} = 4,16$ ώρες.

5 Συμπεράσματα

Στα πλαίσια της παρούσας πτυχιακής εργασίας, δημιουργήθηκε ένα σύστημα IoT για τον έλεγχο και την παρακολούθηση Θερμοκρασίας και υγρασίας. Αυτό περιελάβανε την ανάπτυξη μιας εφαρμογής ESP για τη συλλογή και μετάδοση των δεδομένων. Κατά τη διάρκεια της έρευνας και της ανάπτυξης, αντιμετωπίστηκαν προκλήσεις όπως η διασύνδεση των αισθητήρων με τη συσκευή ESP και η μετάδοση των δεδομένων στο cloud. Το σύστημα που μελετήθηκε θεωρείται αξιόπιστο εκτός από την περίπτωση όπου υπάρχει η απαίτηση τοποθέτησης σε απομακρυσμένα σημεία καθώς η αυτονομία αποδείχτηκε σχετικά μικρή.

Αν ξαναγινόταν η πτυχιακή σήμερα από την αρχή, θα γινόταν με παρόμοιο τρόπο. Αν και αντιμετωπίστηκαν προκλήσεις, η διαδικασία βοήθησε στην απόκτηση πολλών γνώσεων και δεξιοτήτων στον τομέα των IoT και της ανάπτυξης λογισμικού. Άξιζε τον κόπο, καθώς βοήθησε στην εμβάθυνση του ενδιαφέροντος για το θέμα και την ανάπτυξη της επιθυμίας για μελλοντικές επιστημονικές ή επαγγελματικές δραστηριότητες σε αυτόν τον τομέα.

Για τη συνέχιση της πτυχιακής εργασίας, θα προτεινόταν η ανάπτυξη επιπλέον λειτουργιών στην εφαρμογή Android και την επέκταση του συστήματος για την παρακολούθηση περισσότερων παραμέτρων περιβάλλοντος με περισσότερους αισθητήρες. Επίσης, η εξέταση των δυνατοτήτων για την ανάπτυξη μιας εφαρμογής iOS θα μπορούσε να επεκτείνει τη χρήση του συστήματος σε νέους χρήστες.

Βιβλιογραφία

[1] V. S. Chakravarthi, Internet of Things and M2M Communication, Springer Cham, 2022.

[2] O'Reilly Media, Inc., Environmental Monitoring with Arduino, O'Reilly Media, Inc., 2012.

[3] A. Biessek, Flutter for Beginners, BIRMINGHAM - MUMBAI: Packt Publishing, 2019.

[4] P. Seneviratne, ESP8266 Robotics Projects, Birmingham: Packt Publishing, 2017.

[5] "Apache JMeter," Apache, [Online]. Available: https://jmeter.apache.org/. [Accessed 18 2 2024].

[6] emqx, "mqtt-jmeter," [Online]. Available: https://github.com/emqx/mqtt-jmeter. [Accessed 2024 2 18].

[7] "OpenWeather," [Online]. Available: https://openweathermap.org/. [Accessed 20 2 2024].

[8] PowerStream engineers, "Discharge tests of AA Batteries, Alkaline and NiMH," [Online]. Available: https://www.powerstream.com/AA-tests.htm. [Accessed 20 2 2024].

Παράρτημα Α

Κώδικας esp-01s

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include "DHTesp.h"

const char* wifi_name = "{{your-wifi-ssid}}";
const char* wifi_password = "{{your-wifi-password}}";
const char* mqtt_service = "{{your-mqtt-server}}";
const char* mqtt_firstopic = "{{your-mqtt-topic}}";
const char* mqtt_sectopic = "{{your-second-mqtt-topic}}";
const char* mqtt_username = "{{your-mqtt-username}}";
const char* mqtt_pass = "{{your-mqtt-password}}";

BearSSL::WiFiClientSecure espClient;
static const char ca_cert[] PROGMEM = R"{{your-ca-certificate}}";

void setTime() {
    configTime(3 * 3600, 0, "pool.ntp.org", "time.nist.gov");
    Serial.print("Time sync started");
    time_t now = time(nullptr);
    while (now < 8 * 3600 * 2) {
        delay(500);
        Serial.print(".");
        now = time(nullptr);
    }
    Serial.println("");
    struct tm timeinfo;
    gmtime_r(&now, &timeinfo);
    Serial.print("Given Time: ");
    Serial.print(asctime(&timeinfo));
}

DHTesp dht;
PubSubClient client(espClient);

void setup() {
    Serial.begin(9600);
    BearSSL::X509List *serverTrustedCA = new BearSSL::X509List(ca_cert);
    espClient.setTrustAnchors(serverTrustedCA);

    // Connect to Wi-Fi
    WiFi.begin(wifi_name, wifi_password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Waiting for WiFi...");
    }
    Serial.println("WiFi Connected");
    setTime();

    // Set up MQTT
    client.setServer(mqtt_service, 8883);
```

```
    // Start DHT sensor
    dht.setup(2, DHTesp::DHT22);
}

void loop() {
    delay(dht.getMinimumSamplingPeriod());

    // Reconnect to MQTT if necessary
    if (!client.connected()) {
        reconnectToMqtt();
    }

    // Read sensor data
    float temperature = dht.getTemperature();
    float humidity = dht.getHumidity();
    char buf[6];
    char bufsectopic[6];
    dtostrf(temperature, 4, 2, buf);
    dtostrf(humidity, 4, 2, bufsectopic);
    client.publish(mqtt_firstopic, buf);
    client.publish(mqtt_sectopic, bufsectopic);
    delay(5000); // Publish every 5 seconds
}

void reconnectToMqtt() {
    // Loop until reconnected
    while (!client.connected()) {
        Serial.println("MQTT connection starting...");
        // Attempt to connect
        if (client.connect("admin", mqtt_username, mqtt_pass)) {
            Serial.println("MQTT Connected");
        } else {
            Serial.print("Failed, rc=");
            Serial.print(client.state());
            Serial.println(" Retrying in 5 seconds...");
            delay(5000);
        }
    }
}
```

Παράρτημα Β

Κώδικας flutter

```
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:hive/hive.dart';
import 'package:hive_flutter/hive_flutter.dart';
import 'package:mqtt_client/mqtt_client.dart';
import 'package:mqtt_client/mqtt_server_client.dart';
import 'package:d_chart/d_chart.dart';

void main() async {
  await Hive.initFlutter();
  await Hive.openBox<List>('topics');
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Sensor Monitoring App',
      home: SensorPage(),
    );
  }
}

class SensorPage extends StatefulWidget {
  @override
  _SensorPageState createState() => _SensorPageState();
}

class _SensorPageState extends State<SensorPage> {
  late MqttServerClient client;
  bool isCelsius = true;
  String mqttBroker = '3b2d113ecfca4337a34a02cb4f997004.s1.eu.hivemq.cloud';
  String username = 'admin';
  String password = 'User1234';
  Box box = Hive.box<List>('topics');
  // random client id
  String clientId = 'mqtt_sense_${DateTime.now().millisecondsSinceEpoch}';
  int port = 8883;

  List<SensorData> sensors = []; // List of added sensors

  @override
  void initState() {
    super.initState();
    sensors = box.get(0) != null
        ? box
            .get(0)
            .map<SensorData>((e) => SensorData.fromJson(e.toString()))
            .toList()
```

```
      : [];
  connectToMQTTBroker();
}

void connectToMQTTBroker() async {
  client = MqttServerClient.withPort(mqttBroker, clientId, port);

  client.secure = true;
  client.autoReconnect = true;

  client.onConnected = () {
    print('Connected to MQTT broker');
    subscribeToTopics();
  };

  client.onDisconnected = () {
    print('Disconnected from MQTT broker');
  };

  client.onSubscribed = (String topic) {
    print('Subscribed to topic: $topic');
  };

  try {
    await client.connect(username, password);
  } catch (e) {
    print('Error connecting to MQTT broker: $e');
  }
}

void subscribeToTopics() {
  for (var sensor in sensors) {
    client.subscribe(sensor.topic, MqttQos.atMostOnce);
  }

  client.updates?.listen((List<MqttReceivedMessage<MqttMessage>> c) {
    final MqttPublishMessage message = c[0].payload as MqttPublishMessage;
    final String payload =
        MqttPublishPayload.bytesToStringAsString(message.payload.message);

    setState(() {
      for (var sensor in sensors) {
        if (c[0].topic == sensor.topic) {
          sensor.data[DateTime.now()] = double.parse(payload);
        }
      }
    });
  });
}

void addSensor(String topic, SensorType sensorType) {
  setState(() {
    sensors.add(SensorData(topic: topic, data: [], type: sensorType));
    client.subscribe(topic, MqttQos.atMostOnce);
    box.put(0, sensors.map((e) => e.toJson()).toList());
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
```

```
      body: Column(
        children: [
          Expanded(
            child: GridView.builder(
              gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
                crossAxisCount: 2,
              ),
              itemCount: sensors.length,
              itemBuilder: (context, index) {
                return Material(
                  color: sensors[index].type == SensorType.temperature
                      ? Colors.red[50]
                      : sensors[index].type == SensorType.humidity
                          ? Colors.blue[50]
                          : Colors.green[50],
                  child: InkWell(
                    onTap: () {
                      Navigator.push(
                        context,
                        MaterialPageRoute(
                          builder: (context) =>
                              SensorDetailsPage(isCelsius: isCelsius),
                          settings: RouteSettings(arguments: sensors[index]),
                        ),
                      );
                    },
                    child: Column(
                      mainAxisAlignment: MainAxisAlignment.center,
                      children: [
                        Text('${sensors[index].topic}',
                            style: TextStyle(fontWeight: FontWeight.bold)),
                        Icon(
                          sensors[index].type == SensorType.temperature
                              ? Icons.thermostat
                              : sensors[index].type == SensorType.humidity
                                  ? Icons.water
                                  : Icons.bar_chart,
                          size: 50,
                          //beautify the icon
                          color: sensors[index].type == SensorType.temperature
                              ? Colors.red
                              : sensors[index].type == SensorType.humidity
                                  ? Colors.blue
                                  : Colors.green,
                        ),
                        //small gap
                        SizedBox(height: 10),
                        Text(
                            ' ${sensors[index].type == SensorType.temperature ? isCel-
sius ? '${sensors[index].latestData} °C' : '${sensors[index].latestData * 1.8 + 32} °F'
: sensors[index].latestData}'),
                      ],
                    ),
                  ),
                );
              },
            ),
          ),
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 15),
            child: Row(
```

```
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                Row(
                  children: [
                    Text('°C'),
                    Switch(
                      value: !isCelsius,
                      onChanged: (value) {
                        setState(() {
                          isCelsius = !value;
                        });
                      },
                    ),
                    Text('°F'),
                  ],
                ),
                Row(
                  children: [
                    Padding(
                      padding: const EdgeInsets.all(8.0),
                      child: ElevatedButton(
                        onPressed: () {
                          _showAddSensorDialog();
                        },
                        child: Text('Add Sensor'),
                      ),
                    ),
                    //about the app icon button
                    IconButton(
                      onPressed: () {
                        showAboutDialog(
                          context: context,
                          applicationName:
                              'Σύστημα IoT για την παρακολούθηση και τον έλεγχο των και-
ρικών συνθηκών',
                          applicationVersion: '1.0.0',
                          applicationIcon: Icon(Icons.info),
                          children: [
                            Text(
                                'Αυτή η εφαρμογή δημιουργήθηκε στα πλαίσια της πτυχιακής
εργασίας «Ανάπτυξη Συστήματος IoT για την παρακολούθηση και τον έλεγχο των καιρικών συν-
θηκών» του Πλούταρχου Χίνη φοιτητή του τμήματος ψηφιακών συστημάτων του πανεπιστήμιου
Θεσσαλίας '),
                          ],
                        );
                      },
                      icon: Icon(Icons.info),
                    ),
                  ],
                ),
              ],
            ),
          ),
        ],
      ),
    );
  }

  Future<void> _showAddSensorDialog() async {
    TextEditingController topicController = TextEditingController();
    SensorType? selectedType;
```

```
    return showDialog<void>(
      context: context,
      builder: (BuildContext context) {
        return StatefulBuilder(
          builder: (context, setState) {
            return AlertDialog(
              title: Text('Add New Sensor'),
              content: Column(
                mainAxisSize: MainAxisSize.min,
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  TextField(
                    controller: topicController,
                    decoration: InputDecoration(labelText: 'Topic'),
                  ),
                  DropdownButtonFormField<SensorType>(
                    value: selectedType,
                    onChanged: (SensorType? newValue) {
                      setState(() {
                        selectedType = newValue;
                      });
                    },
                    items: SensorType.values.map((type) {
                      return DropdownMenuItem<SensorType>(
                        value: type,
                        child: Text(type.toString().split('.').last),
                      );
                    }).toList(),
                    decoration: InputDecoration(labelText: 'Sensor Type'),
                  ),
                ],
              ),
              actions: <Widget>[
                TextButton(
                  onPressed: () {
                    Navigator.of(context).pop();
                  },
                  child: Text('Cancel'),
                ),
                TextButton(
                  onPressed: () {
                    String newTopic = topicController.text.trim();
                    if (newTopic.isNotEmpty && selectedType != null) {
                      addSensor(newTopic, selectedType!);
                    }
                    Navigator.of(context).pop();
                  },
                  child: Text('Add'),
                ),
              ],
            );
          },
        );
      },
    );
  }
}

class SensorData {
  final String topic;
```

```
  Map<DateTime, double> data;
  final SensorType type;

  SensorData(
      {required this.topic, required List<double> data, required this.type})
      : data = data.isNotEmpty ? {DateTime.now(): data.last} : {};

  double get latestData => data.isNotEmpty ? data.values.last : 0.0;

  void addData(double newData) {
    data[DateTime.now()] = newData;
  }

  //to json
  String toJson() {
    return jsonEncode({
      'topic': topic,
      'type': type.toString().split('.').last,
    });
  }

  //from json
  factory SensorData.fromJson(String json) {
    Map<String, dynamic> data = jsonDecode(json);
    return SensorData(
      topic: data['topic']!,
      data: [],
      type: data['type'] == 'temperature'
          ? SensorType.temperature
          : data['type'] == 'humidity'
              ? SensorType.humidity
              : SensorType.pressure,
    );
  }
}

enum SensorType { temperature, humidity, pressure }

class SensorDetailsPage extends StatelessWidget {
  final bool isCelsius;

  const SensorDetailsPage({super.key, required this.isCelsius});

  @override
  Widget build(BuildContext context) {
    SensorData sensor =
        ModalRoute.of(context)!.settings.arguments as SensorData;
    return Scaffold(
      appBar: AppBar(
        //add sensor icon and measurement type
        title: Row(
          children: [
            Icon(
              sensor.type == SensorType.temperature
                  ? Icons.thermostat
                  : sensor.type == SensorType.humidity
                      ? Icons.water
                      : Icons.bar_chart,
              size: 30,
            ),
            SizedBox(width: 10),
```

```
            SizedBox(
                width: 120,
                child: Text(sensor.topic, overflow: TextOverflow.ellipsis)),
          ],
        ),
        actions: [
          Text(
              ' ${sensor.type == SensorType.temperature ? isCelsius ? '°C' : '°F' : ''}
'
              ' ${sensor.type == SensorType.humidity ? '%' : ''}'
              ' ${sensor.type == SensorType.pressure ? 'hPa' : ''}'),
        ],
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [
            //d_chart
            AspectRatio(
              aspectRatio: 16 / 9,
              child: DChartLineT(
                layoutMargin: LayoutMargin(20, 20, 20, 20),
                domainAxis: DomainAxis(numericViewport: NumericViewport(0, 50)),
                allowSliding: true,
                groupList: [
                  TimeGroup(
                      color: sensor.type == SensorType.temperature
                          ? Colors.red
                          : sensor.type == SensorType.humidity
                              ? Colors.blue
                              : Colors.green,
                      data: [
                        for (var i = 0; i < sensor.data.length; i++)
                          TimeData(
                              domain: sensor.data.keys.elementAt(i),
                              measure: sensor.data.values.elementAt(i))
                      ],
                      id: '1')
                ],
              ),
            ),

            SizedBox(height: 20),

            //latest data list bulder
            Text(
              'Latest Data',
              style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
            ),
            SizedBox(height: 10),
            Expanded(
              child: ListView.builder(
                itemCount: sensor.data.length,
                itemBuilder: (context, index) {
                  final date = sensor.data.keys.elementAt(index);
                  final value = sensor.data.values.elementAt(index);
                  return ListTile(
                    title: Text(
                        '${date.toLocal()} - ${sensor.type != SensorType.temperature ?
isCelsius ? '$value °C' : '${value * 1.8 + 32} °F' : value}'),
```

```
                    );
                },
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

Παράρτημα Γ

Πρόγραμμα σε python το οποίο συλλέγει δεδομένα από αισθητήρα χρησιμοποιώντας MQTT και το API του Open weather map, ώστε να τα αποθηκεύσει σε αρχείο τύπου CSV.

```
import paho.mqtt.client as mqtt
import requests
import csv
from datetime import datetime
import time
import threading

# MQTT settings
mqtt_broker_address = "mqtt.example.com"
mqtt_topic = "sensors/temperature_humidity"

# OpenWeatherMap settings
api_key = "openweathermap_api_key"
city_name = "city_name"
units = "metric"  # or "imperial" for Fahrenheit

csv_file_path = "sensor_data.csv"

# Variables to store MQTT data
mqtt_temperature = None
mqtt_humidity = None

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe(mqtt_topic)

def on_message(client, userdata, msg):
    global mqtt_temperature, mqtt_humidity
    print(msg.topic+" "+str(msg.payload))
    data = msg.payload.decode().split(',')
    mqtt_temperature = float(data[0])
    mqtt_humidity = float(data[1])

def get_openweathermap_data():
    url = f"http://api.openweather-
map.org/data/2.5/weather?q={city_name}&units={units}&appid={api_key}"
    response = requests.get(url)
    data = response.json()
    temperature = data['main']['temp']
    humidity = data['main']['humidity']
    return temperature, humidity

def save_to_csv(api_temperature=None, api_humidity=None, mqtt_temperature=None, mqtt_hu-
midity=None):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

```
    with open(csv_file_path, 'a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow([timestamp, api_temperature, api_humidity, mqtt_temperature,
mqtt_humidity])

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect(mqtt_broker_address, 1883, 60)

# Start the MQTT client loop in a background thread
client.loop_start()

while True:
    # Fetch data from OpenWeatherMap
    api_temperature, api_humidity = get_openweathermap_data()

    # Save data to CSV
    save_to_csv(api_temperature=api_temperature, api_humidity=api_humidity, mqtt_temper-
ature=mqtt_temperature, mqtt_humidity=mqtt_humidity)

    # Reset MQTT data
    mqtt_temperature = None
    mqtt_humidity = None

    # Wait for a while before fetching data again
    time.sleep(60)
```