



ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ
ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Ανάπτυξη Λογισμικού Διαχείρισης Ροών Δεδομένων από
Ασύρματους Κόμβους

Παναγιώτης Λιολιόπουλος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
Υπεύθυνοι
Αθανάσιος Λουκόπουλος
Αναπληρωτής Καθηγητής
Κωνσταντίνος Κολομβάτσος
Επίκουρος Καθηγητής

Λαμία, 2023



**ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ
ΒΙΟΙΑΤΡΙΚΗ**

**Ανάπτυξη Λογισμικού Διαχείρισης Ροών Δεδομένων από
Ασύρματους Κόμβους**

Παναγιώτης Λιολιόπουλος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

**Επιβλέποντες
Αθανάσιος Λουκόπουλος
Αναπληρωτής Καθηγητής
Κωνσταντίνος Κολομβάτσος
Επίκουρος Καθηγητής**

Λαμία, 2023

Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις ⁽¹⁾, που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάστηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.
2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.
3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια
4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: 27/06/2023

Ο Δηλ.

Λιολιόπουλος Παναγιώτης
(Υπογραφή)

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.

**Ανάπτυξη Λογισμικού Διαχείρισης Ροών Δεδομένων από
Ασύρματους Κόμβους**

Λιολιόπουλος Παναγιώτης

Τριμελής Επιτροπή:

Αθανάσιος Λουκόπουλος, Αναπληρωτής Καθηγητής (επιβλέπων)

Κωνσταντίνος Κολομβάτσος, Επίκουρος Καθηγητής (επιβλέπων)

Νικόλαος Τζιρίτας, Επίκουρος Καθηγητής

Περιεχόμενα

1	Εισαγωγή	1
1.1	Διαδίκτυο των Πραγμάτων	2
1.2	Πρόβλημα Παρακολούθησης Ποιότητας Αέρα Διασωστών	3
2	Μεθοδολογία	5
2.1	Σύστημα Παρατήρησης Ποιότητας του Αέρα	5
2.1.1	Περιγραφή Κόμβων	6
2.1.2	Μικροϋπολογιστής Raspberry Pi	12
2.1.3	Συνδεσμολογία	16
2.1.4	Περιγραφή Λογισμικού	18
2.2	Απομακρυσμένος server	21
2.2.1	Αρχιτεκτονική REST	24
2.2.2	Flask Web Framework	26
2.2.3	Βάση Δεδομένων Mongo	29
2.2.4	Δομή Web Server	32
2.3	Πλήρες Σύστημα Αισθητήρων	34
2.3.1	Σύστημα Αισθητήρων Libelium	35
2.3.2	MQTT Broker	36
2.4	Εργαλεία και Τεχνολογίες Ανάπτυξης Εφαρμογής	37
2.4.1	Docker	37
2.4.2	Postman	38
2.4.3	Ubuntu Virtual Private Server	39
2.5	Υλοποίηση Εφαρμογής	40
2.5.1	Δημιουργία Πόρων και Απόδοση Λειτουργικότητας	40
2.5.2	Σύστημα Πιστοποίησης Ταυτότητας	46
3	Αποτελέσματα	55
3.1	Πειραματικά Αποτελέσματα	55
3.2	Σύγκριση Αποτελεσμάτων Πλήρους Συστήματος	57
4	Συμπεράσματα-Συζήτηση	59

Κατάλογος Σχημάτων

2.1	Σύστημα Παρατήρησης Ποιότητας του Αέρα	6
2.2	Πίνακας Ευρωπαϊκού Δείκτη Ποιότητας Αέρα (European Environ- ment Agency, 2023). Οι τιμές είναι σε μονάδα $\mu g/m^3$	7
2.3	Διάγραμμα Λειτουργίας Αισθητήρα Ανίχνευσης Σωματιδίων PM2.5 και PM10.0 (DFRobot, 2023)	10
2.4	Αισθητήρας Ανίχνευσης Σωματιδίων PM1.0, PM2.5 και PM10.0 . . .	11
2.5	Διάγραμμα GPIO pins (Pinout, 2023)	15
2.6	Μοντέλο Client-Server	23
2.7	Μοντέλο αποθήκευσης Εγγράφων στην MongoDB.	31
2.8	Αναπαράσταση πρωτοκόλλου MQTT.	36
3.1	Πείραμα Μετρήσεων.	56
3.2	Αποτελέσματα αιτήσεων στα endpoints.	57
3.3	Αποτελέσματα μετρήσεων αισθητήρων Smart Spot.	58
3.4	Αποτελέσματα Κόμβων σε φυσιολογικές συνθήκες.	58

Κατάλογος Πινάκων

2.1	Πληροφορίες Αισθητήρων Ανίχνευσης Αερίων	8
2.2	Πίνακας Ευρωπαϊκού Δείκτη Ποιότητας Αέρα. Οι τιμές είναι σε μονάδα ppm.	12
2.3	Χαρακτηριστικά (Specs) Raspberry Pi 4 Model B (Raspberry Pi, 2023)	14
2.4	Βασικές Μέθοδοι Create Read Update Delete (CRUD) HTTP	22
2.5	Εγκατεστημένα Πακέτα Flask	27
2.6	Μέθοδοι CRUD στην βάση.	32
2.7	Διεπαφές μεταξύ web server και εφαρμογής.	33
2.8	Βασικά Χαρακτηριστικά Αισθητήρων Smart Spot.	35
2.9	Ρόλοι χρηστών και δυνατότητες.	47
2.10	Μέθοδοι κλάσης χρηστών.	52

Κατάλογος Συντομογραφιών

IoT	Internet of Things
IP	Internet Protocol
NO₂	Nitrogen Dioxide
SO₂	Sulfur Dioxide
O₃	Ozone
CO	Carbon Monoxide
PM	Particulate Matter
ppm	parts per million
ppmv	parts per million by volume
SoC	System on Chip
UART	Universal Asynchronous Receiver/Transmitter
I2C	Inter-Integrated Circuits
GPIO	General Purpose Input-Output
pip	Package Installer for Python
PyPI	Python Package Index
SCL	Serial Clock
SDA	Serial Data
JSON	JavaScript Object Notation
HTTP	HyperText Transfer Protocol
REST	Representational State Transfer
UI	User Interface
URL	Uniform Resource Locator
HTML	HyperText Markup Language
CRUD	Create Read Update Delete
CLI	Command Line Interface
DBMS	Database Management System
SQL	Structured Query Language
NoSQL	Not only SQL
XML	Extensible Markup Language
CGI	Common Gateway Interface
WSGI	Web Server Gateway Interface
MQTT	Message Queueing Telemetry Transport
API	Application Program Interface
AQI	Air Quality Index

Περίληψη

Οι περιβαλλοντικές καταστάσεις κρίσης - για παράδειγμα πυρκαγιές, κατολισθήσεις μεταλλείων και ηφαιστειακές εκρήξεις - αποτελούν σημαντικό πρόβλημα, καθώς εκθέτουν άνθρωπους σε άμεσο κίνδυνο και χρειάζεται η επέμβαση διασωστών, υπό παθογόνες συνθήκες. Σκοπός της εργασίας είναι η ανάπτυξη ενός συστήματος διαχείρισης ροών δεδομένων από ασύρματους κόμβους, το οποίο επιβλέπει την ποιότητα του αέρα στην περιοχή των διασωστών, συλλέγει και αποθηκεύει την πληροφορία αυτή και προσφέρει πρόσβαση σ' αυτήν τόσο στους διασώστες όσο και σε υπηρεσίες (πυροσβεστική, διασωστικά συνεργεία, δασονομία κ.λπ.). Η επίβλεψη της ποιότητας του αέρα γίνεται από ένα σύστημα κόμβων, αποτελούμενοι από αισθητήρες αερίων Carbon Monoxide (CO), Sulfur Dioxide (SO₂), Nitrogen Dioxide (NO₂) και Ozone (O₃) και σωματιδίων PM_{2.5} και PM_{10.0}, καθώς και έναν μικροϋπολογιστή Raspberry Pi, όπου η επικοινωνία μεταξύ τους γίνεται μέσω των πρωτοκόλλων I2C και UART. Για την συλλογή και αποθήκευση δεδομένων αναπτύσσεται web server με βάση δεδομένων και RESTful εφαρμογή, όπου ένας server δέχεται και μεταβιβάζει τα δεδομένα στην εφαρμογή, η οποία επικοινωνεί με την βάση για την αποθήκευση και ανάκληση των δεδομένων. Τέλος, η εφαρμογή προσφέρει ένα σύνολο από ασφαλή endpoints ως διεπαφή των χρηστών με τον web server και τα δεδομένα της βάσης. Τα αποτελέσματα πειράματος σε εξωτερικό χώρο με καύση οργανικής ύλης, μέτρηση δείκτης ποιότητας αέρα και ταυτόχρονης χρήσης της εφαρμογής, έδειξαν ικανοποιητική απόδοση, όμως χρειάζονται επιπλέον πειράματα, για την μελέτη της κλιμακωσιμότητας των κόμβων και αποφυγή biased αποτελεσμάτων.

Λέξεις-Κλειδιά: Αισθητήρες, Raspberry Pi, Δείκτης Ποιότητας Αέρα, Κόμβος, Web Server, Web Application, Endpoints

Abstract

Environmental crisis situations - for example fires, mine slides and volcanic eruptions - are a major problem, as they put people at immediate risk and require the intervention of rescuers under pathogenic conditions. The aim of this work is to develop a system for managing data streams from wireless nodes, which monitors the air quality in the area of the rescuers, collects and stores this information and provides access to it for both rescuers and services (fire brigade, rescue teams, forestry, etc.). Air quality monitoring is carried out by a system of nodes consisting of sensors for Carbon Monoxide (CO), Sulfur Dioxide (SO₂), Nitrogen Dioxide (NO₂) and Ozone (O₃) gases and PM_{2.5} and PM_{10.0} particles, as well as a Raspberry Pi microcomputer, communicating with each other via I2C and UART protocols. For data collection and storage, a web server with a database and a RESTful application is developed, where a server receives and transfers the data to the application, which communicates with the database to store and retrieve the data. Finally, the application provides a set of secure endpoints as the user interface to the web server and the database data. The results of an outdoor experiment with organic matter combustion, air quality indicator measurement and simultaneous use of the application showed satisfactory performance, but further experiments are needed to study the scalability of the nodes and avoid biased results.

Keywords: *Sensors, Raspberry Pi, Air Quality Index, Node, Web Server, Web Application, Endpoints*

Εισαγωγή

Οι σύγχρονες τεχνολογίες και εφαρμογές του Διαδικτύου των Πραγμάτων - Internet of Things (IoT) προσφέρουν πληθώρα λύσεων σε πολλά ζητήματα και εξελίσσουν την υπάρχουσα υποδομή. Πολλές ενέργειες που προηγουμένως χρειαζόταν να γίνουν χειροκίνητα έχουν αντικατασταθεί ή αυτοματοποιηθεί πλήρως από τεχνολογίες IoT. Για παράδειγμα, υπάρχουν πολλά ολοκληρωμένα συστήματα σπιτιών, τα οποία είναι υπεύθυνα για την ρύθμιση της θερμοκρασίας σε κάθε χώρο, για την παρακολούθηση της ασφάλειας του σπιτιού, την εξοικονόμηση ενέργειας κ.λπ. Πολλές πόλεις χρησιμοποιούν τεχνολογίες IoT για την επιβλέψη της κυκλοφορίας, της ποιότητας αέρα, της επιτήρησης κ.λπ. Οι βιομηχανίες χρησιμοποιούν τις τεχνολογίες αυτές για την επιτήρηση, την μεταφορά προϊόντων, την κατανάλωση ενέργειας κ.λπ. Γίνεται φανερό ότι αυτές οι τεχνολογίες αποτελούν σήμερα ακρογωνιαίο λίθο για την εξέλιξη και την ψηφιοποίηση της κοινωνίας και των ζητημάτων που την αφορούν.

Ένα τμήμα που χρίζει άμεσα την αξιοποίησή τους είναι οι φυσικές καταστάσεις κρίσεις. Συγκεκριμένα, πρόκειται για καταστάσεις στις οποίες κινδυνεύουν άμεσα ανθρώπινες ζωές, λόγω περιβαλλοντικών φαινομένων και καιρικών συνθηκών, ενώ ταυτόχρονα γίνεται επέμβαση διασωστικής ομάδας. Τέτοιες καταστάσεις αποτελούν πυρκαγιές, κατολισθήσεις ορυχείων, εκρήξεις ηφαιστειών κ.α. Σε μια τέτοια κατάσταση, υπάρχουν άνθρωποι οι οποίοι είναι παγιδευμένοι ή κινδυνεύουν άμεσα από τις συνεπείες μιας τέτοιας κατάστασης και είναι ανίκανοι να την αντιμετωπίσουν μόνοι. Σε αυτές τις καταστάσεις γίνεται άμεση επέμβαση από διασώστες (πυροσβεστική, διασωστικά συνεργεία κ.λπ.), οι οποίοι έρχονται αντιμέτωποι με αντίξοες συνθήκες. Ο μεγαλύτερος κίνδυνος που ελοχεύει είναι ο αέρας που αναπνέουν. Η ποιότητα του σε μεγάλο βαθμό είναι κακή και με την πάροδο του χρόνου, χειροτερεύει. Το αποτέλεσμα είναι να γίνεται παθογόνα τόσο για τους ανθρώπους σε κίνδυνο, όσο και για τους ίδιους τους διασώστες.

Με βάση στοιχεία του Παγκόσμιου Ταμείου για τη Φύση (WWF), περίπου 10.000 πυρκαγιές υπαίθρου εκδηλώνονται κάθε χρόνο στην Ελλάδα και καίγονται κατά μέσο όρο 386.812 στρέμματα, με το μεγαλύτερο ποσοστό να οφείλεται σε ανθρώπινη αμέλεια. Στοιχεία του Πυροσβεστικού Σώματος Ελλάδος αναφέρουν πως το 2022 υπήρχαν περίπου 25 εκατομμύρια καταστροφές στην Ελλάδα, με τις 5 εκατομμύρια να οφείλονται σε πυρκαγιές. Όσον αφορά τις κατολισθήσεις ορυχείων, την περασμένη δεκαετία υπολογίζεται, ότι περισσότεροι από 300.000 άνθρωποι ήρθαν αντιμέτωποι με κάποιου είδους κατολίσθηση (είτε τραυματίστηκαν, είτε πέθαναν, είτε έμειναν άστεγοι κ.λπ.). Παρόμοια και για την ηφαιστειακή δραστηριότητα, την περασμένη δεκαετία περισσότεροι από 300.000 άνθρωποι βρέθηκαν αντιμέτωποι με ηφαιστειακή δραστηριότητα (είτε τραυματίστηκαν, είτε πέθαναν, είτε έμειναν άστεγοι κ.λπ.). Συνεπώς, η ανάπτυξη τεχνολογιών για την επίβλεψή τους και την υποβοήθηση των διασωστών και των ανθρώπων σε κίνδυνο κρίνονται επιτακτικά.



(i) Πυρκαγιά



(ii) Έκρηξη Ηφαιστείου



(iii) Κατολίσθηση Μεταλλείου

1.1 Διαδίκτυο των Πραγμάτων

Το Διαδίκτυο των Πραγμάτων πρωτοαναφέρθηκε από τον Kevin Ashton το 1999 για να περιγράψει ένα σύστημα αντικειμένων συνδεδεμένα μεταξύ τους μέσω του Διαδικτύου. Ουσιαστικά, το IoT δεν αποτελεί τύποτα άλλο παρά ένα σύστημα αντικείμενων του πραγματικού κόσμου συνδεδεμένα μέσω του Διαδικτύου, για την συλλογή και αποθήκευση δεδομένων και επίβλεψη και ελέγχου ενός περιβάλλοντος. Από αισθητήρες μέχρι ολόκληρα ρομποτικά συστήματα, από οικιακές συσκευές και έπιπλα μέχρι βιομηχανικές μονάδες, το IoT προσφέρει πλήθος δυνατοτήτων για την ανάπτυξη εφαρμογών του σύγχρονου κόσμου. Το βασικό χαρακτηριστικό όλων αυτών των αντικειμένων είναι η δυνατότητα τους να συνδέονται στο Διαδίκτυο, μέσω διεύθυνση Internet Protocol (IP). Για να μπορέσει ένα αντικείμενο να δικτυωθεί, θα πρέπει να έχει μια μοναδική διεύθυνση, ώστε να είναι ορατό στο δίκτυο και να μπορούν άλλες συσκευές να επικοινωνούν μαζί της. Όπως φαίνεται, ένα σύστημα IoT συντελείται από πληθώρα παραγόντων. Εύλογα, γεννιέται το ερώτημα ποιές είναι αυτές.

Γενικά, το IoT αποτελεί έναν από τους πιο σύνθετους κλάδους της τεχνολογίας. Συγκεκριμένα, η άνοδος και η ραγδαία ανάπτυξη των εφαρμογών και τεχνολογιών IoT στυρίχθηκε στην συνεισφορά πολλών τεχνολογιών και εμπορικών τάσεων, οι οποίες επέτρεψαν την διασύνδεση πολλαπλών συσκευών, σε διαφορετικά μεγέθη, με έμφαση, όμως, σε μικρότερες συσκευές, όπως αισθητήρες και ολοκληρωμένα συστήματα. Επίσης, αυτές συνέβαλαν και στην μείωση του κόστους κατασκευής αλλά και στην πολυπλοκότητα της κατασκευής. Σύμφωνα με την Karen, οι τεχνολογίες αυτές αποτελούνται από την πανταχού παρούσα συνδεσιμότητα (*Ubiquitous Connectivity*), την ευρεία υιοθέτηση δικτύωσης μέσω IP, τα οικονομικά υπολογιστών, την σμίκρυνση των εξαρτημάτων, την πρόοδο της Ανάλυσης Δεδομένων και

την άνοδο του Υπολογιστικού Νέφους. Η πρώτη, όπως μαρτυρά και το όνομά της, προσφέρει την δυνατότητα διάχυτης δικτύωσης μέσω ασύρματων τεχνολογιών, με χαμηλό κόστος και υψηλή ταχύτητα. Με την ευρεία υιοθέτηση της IP δικτύωσης ολοένα και περισσότερες συσκευές και υπηρεσίες επικοινωνούν με το ίδιο παγκόσμιο πρωτόκολλο, χρησιμοποιώντας πληθώρα καλώς ορισμένων και ευρέως χρησιμοποιούμενων εργαλείων και υπηρεσιών. Η χρηματοδότηση εταιριών στην έρευνα και την ανάπτυξη εφαρμογών, έχουν προσφέρει μεγαλύτερη υπολογιστική δύναμη, με χαμηλότερο κόστος και κατανάλωση ενέργειας. Η σμίκρυνση των εξαρτημάτων παρέχει την δυνατότητα ενσωμάτωσης όχι μόνο των εξαρτημάτων αλλά και των τεχνολογιών τους σε μικροσκοπικά αντικείμενα, με αποτέλεσμα μικρά και φθηνά αντικείμενα, όπως συσκευές μαγειρικής με ενσωματωμένα συστήματα ή αισθητήρες, όπως αυτούς που χρησιμοποιήθηκαν. Η πρόοδος της ανάλυση δεδομένων ένα σύνολο βελτιωμένων δυνατοτήτων, όπως υπολογιστική ισχύ, αποθήκευση δεδομένων και υπηρεσίες νέφους, οι οποίες προσφέρουν διάφορες υπηρεσίες επεξεργασίας δεδομένων μεγάλου όγκου, από τα οποία μπορεί να προκύψει νέα γνώση. Τέλος, μικρές συσκευές μπορούν να αλληλεπιδράσουν μέσω του νέφους με ισχυρά συστήματα ελέγχου και ανάλυσης δεδομένων, καθώς το νέφος χρησιμοποιεί απομακρυσμένους πόρους για την επεξεργασία και διαχείριση των δεδομένων των συσκευών.

1.2 Πρόβλημα Παρακολούθησης Ποιότητας Αέρα Διασωστών

Όπως αναφέρθηκε προηγουμένως, ο μεγαλύτερος κίνδυνος των διασωστών και των ανθρώπων σε κίνδυνο είναι η ποιότητα του αέρα. Γενικά, η ποιότητα του αέρα καθορίζεται από την ποσότητα αερίων και σωματιδίων τα οποία βρίσκονται στην ατμόσφαιρα του χώρου. Σε φυσιολογικές τιμές, ο ατμοσφαιρικός αέρας αποτελείται κυρίως από άζωτο (N_2), οξυγόνο (O_2), αργό (Ar), νερό (H_2O) και διοξείδιο του άνθρακα (CO_2). Όπως είναι αναμενόμενο, μια περιβαλλοντική κατάσταση κρίσης προκαλεί αλλαγές στις τιμές αυτές και αυξάνει τιμές άλλων αερίων προϊόντων των πρώτων, καθώς και σωματιδίων, τα οποία είναι παθογόνα για τους ανθρώπους που βρίσκονται στην ζώνη κινδύνου. Αν αυτές οι αλλαγές δεν παρατηρηθούν εγκαίρως, ώστε να μπορέσουν να γίνουν αντίστοιχες ενέργειες, τότε το πλήθος των παθογόνων αερίων και σωματιδίων μπορεί να γίνει ασφυκτικά επικίνδυνο. Συνεπώς, η ανάπτυξη ενός συστήματος παρακολούθησης ποιότητας αέρα κρίνεται επιτακτική.

Σκοπός της παρούσας εργασίας είναι η ανάπτυξη ενός τέτοιου φορητού συστήματος, ικανό για να μπορεί να παρατηρεί και να ενημερώνει για την ποιότητα του αέρα στην περιοχή που βρίσκονται οι διασώστες, όχι μόνο τους ίδιους αλλά και υπηρεσίες που δύναται να το χρησιμοποιήσουν, όπως την πυροσβεστική, δασαρχεία, κέντρο ελέγχου διασωστικής ομάδας κ.λπ. Μέσω ενός συστήματος αισθητήρων δύναται η παρατήρηση της ποιότητας του αέρα, μέσω της συλλογής αερίων και σωματιδίων τα οποία ονομάζονται ρύποι. Αυτοί οι ρύποι στην συνέχεια αποστέλλονται σε μια βάση δεδομένων, όπου οι υπηρεσίες και οι διασώστες μπορούν να χρησιμοποιήσουν, μέσω μιας εφαρμογής ιστού, η οποία τους παρέχει την δυνατότητα αποστολής και εμφάνισης αποτελεσμάτων χρονολογικά, όσον αφορά τους ρύπους αλλά και την ποιότητα αέρα της περιοχής. Το σύστημα θα είναι φορητό, που σημαίνει ότι δεν θα πρέπει να επιβραδύνει τους διασώστες, ούτε να τους

εμποδίζει, καθώς και να μπορεί να αντέχει στις αντίξοες συνθήκες.

Το σύστημα αποτελείται από τρία μέρη. Το πρώτο μέρος είναι το σύστημα διαχείρισης ροής δεδομένων, το οποίο είναι υπεύθυνο για την λήψη και αποθήκευση των δεδομένων στην βάση, καθώς και τον διαμοιρασμό τους στους χρήστες (υπηρεσίες και διασώστες) που ζητούν δεδομένα από αυτό. Το δεύτερο μέρος είναι ένα σύστημα ασύρματων κόμβων, δηλαδή ένα σύνολο κόμβων τα οποία συλλέγουν πληροφορία για τους ρύπους, το καθένα σε δική του περιοχή, βάση της τοποθεσίας των διασωστών. Ο κάθε κόμβος αποτελείται από αισθητήρες και μικροϋπολογιστή, υπεύθυνα για την συλλογή και αποστολή των δεδομένων στο βασικό σύστημα. Τρίτο και τελευταίο μέρος είναι ένα σύνολο διαδραστικό τελικών σημείων, τα οποία ορίζουν την διεπαφή μεταξύ βασικού συστήματος και χρήστη. Μέσω αυτών, ο χρήστης μπορεί να αιτείται πληροφορίες για τους ρύπους και την ποιότητα αέρα μιας περιοχής.

Μεθοδολογία

Σε αυτό το κεφάλαιο της πτυχιακής εργασίας παρουσιάζονται αναλυτικά όλα τα μέρη που συνθέτουν το συνολικό λογισμικό: το σύστημα παρατήρησης της ποιότητας του αέρα, μέσω συλλογής δεδομένων από αισθητήρες και το σημείο επεξεργασίας των δεδομένων, δηλαδή μία εφαρμογή ιστού (web application), η οποία μέσω ενός web server διαμοιράζει τα δεδομένα και παρέχει και ένα Web API με τις σημαντικότερες εντολές. Πιο συγκεκριμένα, θα παρουσιαστεί ο εξοπλισμός, η συνδεσμολογία και ο κώδικας πίσω από το σύστημα συλλογής παρατηρήσεων, ο κώδικας, τα πακέτα, το λογισμικό, η αρχιτεκτονική και λοιπά προγράμματα της web εφαρμογής και του Web API, και οι διεπαφές χρήστη που προκύπτουν από τα endpoints και αναπαριστούν τα αποτελέσματα της ανάλυσης των δεδομένων.

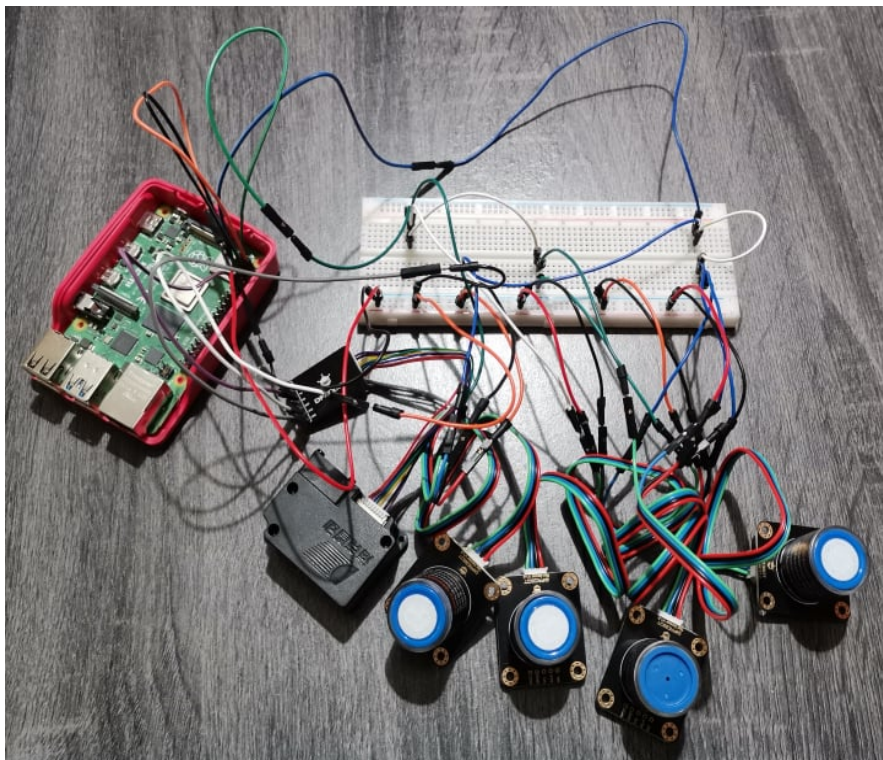
2.1 Σύστημα Παρατήρησης Ποιότητας του Αέρα

Το σύστημα παρατήρησης αποτελεί την ραχοκοκαλιά του τελικού συστήματος. Μέσω αυτού παρατηρείται σε πραγματικό χρόνο η ποιότητα του αέρα. Η ποιότητα του αέρα υπολογίζεται από το πλήθος των τοξικών ρύπων που υπάρχουν στην ατμόσφαιρα. Για την μέτρηση των ρύπων αυτών, χρησιμοποιούνται ειδικοί αισθητήρες, όπου αντιστοιχεί στον καθένα η μέτρηση για έναν ή περισσότερους ρύπους. Αυτές οι μετρήσεις από όλους τους αισθητήρες γίνονται μέσω ενός μικροϋπολογιστή, ο οποίος τροφοδοτεί τους αισθητήρες με τάση, τους παρέχει τον δίαυλο μετάδοσης δεδομένων και καταχωρεί και αποστέλλει τα δεδομένα σε έναν απομακρυσμένο server.

Όπως αναφέρθηκε και προηγουμένως, η αξιολόγηση της ποιότητας του αέρα γίνεται από μετρήσεις τιμών τοξικών αερίων και σωματιδίων. Τα αέρια και σωματίδια, τα οποία καθορίζουν την ποιότητα του αέρα, είναι τα: μονοξείδιο του άνθρακα (CO), διοξείδιο του θείου (SO₂), διοξείδιο του αζώτου (NO₂), όζον στην τροπόσφαιρα (O₃), σωματιδιακή ύλη μικρότερη από 2.5 μικρόμετρα (Particulate Mater (PM)_{2.5}) και σωματιδιακή ύλη μικρότερη από 10 μικρόμετρα (PM_{10.0}). Η επιλογή των σωματιδίων αυτών δεν είναι τυχαία. Αυτά τα αέρια υπάρχουν στην ατμόσφαιρα και παράγονται, σε απειρο-ελάχιστες ποσότητες μέσω βιολογικών φαινομένων. Εξ' αιτίας, όμως, της ρύπανσης, είτε ανθρώπινης είτε βιολογικής, παράγονται σε αυξημένες ποσότητες, σε περιοχές με υψηλή ρύπανση. Έτσι, καθίσταται εφικτή η μέτρησή τους.

Η μέτρηση των ρύπων στην ατμόσφαιρα γίνεται με την χρήση ενός υποσυστήματος αισθητήρων και ενός μικροϋπολογιστή. Συγκεκριμένα, ένα πλήθος αισθητήρων συνδέονται μέσω μιας συγκεκριμένης ηλεκτρονικής διάταξης και πρωτοκόλλου μετάδοσης δεδομένων με έναν μικροϋπολογιστή. Οι αισθητήρες μετράνε τους ρύπους συνεχώς και αποστέλλουν αυτές τις τιμές στον μικροϋπολογιστή. Αυτός αποστέλλει αυτές τις τιμές, αφού πρωτίστως τις επεξεργαστεί, στον απομακρυσμένο server. Αυτή η προεπεξεργασία περιλαμβάνει ένα στάδιο απαραίτητων υπολογισμών, όπως

χρόνο αναμονής για την λήψη των επόμενων δεδομένων απ' τους αισθητήρες, υπολογισμό μέσου όρου δεδομένων κ.λ.π. και της τροποποίησης των δεδομένων προς αποστολή (για παράδειγμα, αποστολή σε μορφή αρχείου JSON).



Σχήμα 2.1: Σύστημα Παρατήρησης Ποιότητας του Αέρα

Το σύστημα παρατήρησης δεν είναι ένα σύνολο αισθητήρων, μαζί με έναν μικρο-υπολογιστή, σε μια τοποθεσία, το οποίο λαμβάνει και αποστέλλει παρατηρήσεις. Στην πραγματικότητα, αποτελείται από πολλά τέτοια σύνολα αισθητήρων και μικρουπολογιστών. Κάθε τέτοιο σύνολο ονομάζεται κόμβος του συστήματος. Οι κόμβοι λαμβάνουν και αποστέλλουν παρατηρήσεις ανεξάρτητα. Κάθε κόμβος αποτελείται από έναν μικροεπεξεργαστή Raspberry Pi και ένα σύνολο 5 αισθητήρων, οι οποίοι συλλέγουν τις τιμές των ρύπων των προαναφερθέντων αερίων και σωματιδίων. Η εικόνα στο σχήμα 2.1 δείχνει έναν συναρμολογημένο τελικό κόμβο του συστήματος ανίχνευσης ποιότητας του αέρα.

2.1.1 Περιγραφή Κόμβων

Πριν την περιγραφή του συστήματος των αισθητήρων, είναι σημαντικό να αναφέρουμε κάποιες λεπτομέρειες σχετικά με το κάθε αέριο και σωματιδίο. Όπως αναφέρει ο Godish, το CO παράγεται από τεχνολογικές διαδικασίες ή κατά την καύση βιομάζας, είναι άοσμο, άχρωμο και άγευστο, ενώ είναι τοξικό για τα θυλαστικά, το SO₂ παράγεται κατά την καύση βιομάζας και καυσίμων ή κατά το λιώσιμο μετάλλων με περιεκτικότητα θείου, το NO₂ παράγεται μέσα απο πληθώρα ατμοσφαιρικών αντιδράσεων (βιολογικές, φωτοχημικές και αντιδράσεις με ρύπους), όμως, επειδή το άζωτο N₂ αποτελεί την συντριπτική πλειοψηφία των αερίων στην ατμοσφαίρα, εξ' αιτίας άλλων ενδιάμεσων ρύπων, προϊόντων και αντιδράσεων, παράγεται σε αυξημένες ποσότητες, και το O₃ παράγεται εξ' αιτίας φωτοχημικών αντιδράσεων

με αντιδρώντα τον ήλιο, NO_x , O_2 και μη μεθανικούς υδρογονάνθρακες (NMHCs) (Godish, 2005).

Τα σωματίδια $\text{PM}_{2.5}$ και $\text{PM}_{10.0}$ παράγονται είτε από ατμοσφαιρικές αντιδράσεις είτε από ανθρώπινους ρύπους. Λόγω του μικρού τους μεγέθους, παραμένουν στην ατμόσφαιρα για μέρες ή και εβδομάδες και, εκτός αυτού, τα σωματίδια $\text{PM}_{2.5}$ έχουν μεγάλη πιθανότητα να εναποθετηθούν στα κύτταρα και τους ιστούς των πνευμόνων, ενώ τα σωματίδια $\text{PM}_{10.0}$ μπορούν να εισέλθουν στην θωρακική περιοχή του αναπνευστικού συστήματος (Godish, 2005).

	Good	Fair	Moderate	Poor	Very poor	Extremely poor
Particles less than 2.5 μm ($\text{PM}_{2.5}$)	0-10	10-20	20-25	25-50	50-75	75-800
Particles less than 10 μm (PM_{10})	0-20	20-40	40-50	50-100	100-150	150-1200
Nitrogen dioxide (NO_2)	0-40	40-90	90-120	120-230	230-340	340-1000
Ozone (O_3)	0-50	50-100	100-130	130-240	240-380	380-800
Sulphur dioxide (SO_2)	0-100	100-200	200-350	350-500	500-750	750-1250

Σχήμα 2.2: Πίνακας Ευρωπαϊκού Δείκτη Ποιότητας Αέρα (European Environment Agency, 2023). Οι τιμές είναι σε μονάδα $\mu\text{g}/\text{m}^3$.

Στην εικόνα 2.2 παρουσιάζεται ο δείκτης ποιότητας του αέρα, όπως αυτός προκύπτει κάθε φορά από τις διάφορες τιμές των προαναφερθέντων χημικών ενώσεων. Ο δείκτης που προκύπτει, παίρνει τιμές από ένα σύνολο κατηγορηματικών τιμών. Αυτές είναι οι: «Καλός» («Good»), «Ικανοποιητικός» («Fair»), «Εντάξει» («Moderate»), «Κακός» («Poor»), «Πολύ Κακός» («Very Poor») και «Επικίνδυνος» («Extremely Poor»). Η τιμή του δείκτη ποιότητας αέρα καθορίζεται από το στοιχείο με την μεγαλύτερη συγκέντρωση. Η συγκέντρωση των σωματιδίων $\text{PM}_{2.5}$ και $\text{PM}_{10.0}$, καθώς και των αερίων, έχει γενικά μονάδες μέτρησης $\mu\text{g}/\text{m}^3$. Για την συγκέντρωση των αερίων, όμως, χρησιμοποιούμε μία άλλη μονάδα μέτρησης, τα *μέρη στο εκατομμύριο* - *parts per million (ppm)*. Τα *ppm* δεν αποτελούν καθιερωμένη μονάδα μέτρησης για επιστημονικές έρευνες, ωστόσο, τα πρότυπα που αφορούν την ατμοσφαιρική ρύπανση, για λόγους ευκολίας στους υπολογισμούς, έχει καθιερωθεί να χρησιμοποιούν τέτοιες μονάδες: πιο συγκεκριμένα, για τις μετρήσεις αερίων στην ατμόσφαιρα, χρησιμοποιούνται τα *μέρη ανά εκατομμύριο κατά όγκο* - *parts per million by volume (ppmv)*, αλλά για λόγους απλότητας και σαφήνειας, επιλέχθηκε η χρήση της μονάδας *ppm*. Αυτή η μονάδα μέτρησης προκύπτει από την σχέση 2.1 (Godish, 2005).

$$1 \text{ ppmv} = \frac{1 \mu\text{l gas}}{1 \text{ l air}} \quad (2.1)$$

Για την ανίχνευση των αερίων και των σωματιδίων, χρησιμοποιήθηκαν οι αισθητήρες της σειράς Gravity της εταιρίας DFRobot. Οι αισθητήρες αυτή της σειράς προσφέρουν σημαντικά πλεονεκτήματα, τα οποία δυνάμεθα να αξιοποιήσουμε στην ανάπτυξη του παρόντος συστήματος. Κατ' αρχάς, η σχεδίαση των εξαρτημάτων είναι αρθρωτή, παρέχοντας με αυτόν τον τρόπο την δυνατότητα στους

αισθητήρες να μπορούν, πρωτίστως να αποσυναρμολογούνται και να επανασυναρμολογούνται, και έπειτα να χρησιμοποιούν ο ένας τα εξαρτήματα του άλλου, χωρίς να αλλάζει το αρχικό αποτέλεσμα ή να υπάρχουν προβλήματα συμβατότητας, αναγνωρισιμότητας, επικοινωνίας κ.ο.κ. Δεύτερον, οι αισθητήρες έχουν μια τυποποιημένη διεπαφή, η οποία είναι plug and play, δηλαδή δεν απαιτεί την λήψη και εγκατάσταση κάποιων επιπλέον οδηγών και το σύστημα μπορεί να διαβάσει τους αισθητήρες κατά την σύνδεσή τους. Τρίτον, η εταιρία διαθέτει μια ηλεκτρονική εργαλειοθήκη ανοικτού κώδικα η οποία καλύπτει τις βασικές ανάγκες για επιλογή του πρωτοκόλλου μετάδοσης δεδομένων, κατωφλίου κινδύνου και βασικών ενεργειών Εισόδου/Εξόδου (E/E). Τέλος, η εταιρία χρησιμοποιεί χρωματική κωδικοποίηση των καλωδίων της, όπου το κάθε καλώδιο, ανάλογα με την ιδιότητά του, έχει το αντίστοιχο χρώμα.

Αισθητήρες Αερίων

Πίνακας 2.1: Πληροφορίες Αισθητήρων Ανίχνευσης Αερίων

Αέριο	Ανιχνεύσιμο Εύρος	Ανάλυση Εύρους Τιμών	Χρόνος Απόκρισης (T90)
CO	0-1000 ppm	1 ppm	$\leq 30s$
NO ₂	0-20 ppm	0.1 ppm	$\leq 30s$
SO ₂	0-20 ppm	0.1 ppm	$\leq 30s$
O ₃	0-10 ppm	0.1 ppm	$\leq 120s$

Οι αισθητήρες για την ανίχνευση των αερίων είναι ηλεκτροχημικοί αισθητήρες, εργοστασιακά βαθμονομημένοι. Κάθε τέτοιος αισθητήρας, απαρτίζεται από 3 μέρη: τον ανιχνευτή αερίου (*Sensor Probe*), έναν μετατροπέα σήματος των ηλεκτροχημικών παραγόντων (*Signal Conversion Board*) και τα καλώδια τεσσάρων pin (*4pin Cable*). Υποστηρίζουν τρία πρωτόκολλα μετάδοσης δεδομένων (αναλογικό, I2C και UART), τα οποία μπορούμε να εκμεταλλευτούμε, αναλόγως τις ανάγκες του προβλήματος, καθώς μας επιτρέπουν να εστιάσουμε σε άλλα προβλήματα, όπως η επιλογή του μικροϋπολογιστή και της ηλεκτρονικής διάταξης, αναλόγως και τις παραμέτρους αυτών των προβλημάτων, όπως κατανάλωση ρεύματος, μέγεθος, κόστος κ.λπ. ή, εφόσον δεν έχουμε έναν αλλά πολλούς αισθητήρες σε ένα κύκλωμα, ποιο πρωτόκολλο από αυτά να επιλέξουμε, ώστε να μπορούν να στέλνουν όλοι οι αισθητήρες τα δεδομένα τους, ανάλογα με τις ανάγκες του προβλήματος (συγχρονισμένα, στο ίδιο κανάλι κ.λπ.).

Η ανίχνευση των αερίων γίνεται μέσω ηλεκτροχημικών μεθόδων (οι οποίες, όμως, ξεφεύγουν από τα πλαίσια της παρούσας εργασίας και για αυτό τον λόγο δεν αναφέρονται). Διαθέτουν 32 τροποποιημένες διευθύνσεις I2C, έναν πλήρη αλγόριθμο αντιστάθμισης θερμοκρασίας και μια λειτουργία συναγερμού κατωφλίου. Είναι συμβατοί με τους περισσότερους μικροϋπολογιστές του εμπορίου, όπως το Arduino, το ESP32 και το Raspberry Pi. Το εύρος τάσης, στο οποίο λειτουργούν, κυμαίνεται από 3.3V DC, έως 5.5V DC. Το ρεύμα θα πρέπει να είναι μικρότερο από 5 mA. Οι θερμοκρασίες, στις οποίες λειτουργούν, ανήκουν στο εύρος $[-20, 50] ^\circ C$. Η υγρασία θα πρέπει να ανήκει στο εύρος $[15, 90] \% RH$, χωρίς συμπύκνωση. Το εύρος ζωής ξεπερνάει τα 2 χρόνια. Συνολικά, διατίθενται στο εμπόριο 12 διαφορετικοί ανιχνευτές αερίων, ανάλογα το αέριο: CO, O₂, NH₃, H₂S, NO₂, HCL, H₂,

PH₃, SO₂, O₃, CL₂, HF, οι οποίοι χρησιμοποιούνται με τον ίδιο ακριβώς τρόπο και τη μορφή δεδομένων. Οι διαστάσεις του είναι 37×32mm. Οι εικόνες 2.3i και 2.3ii παρουσιάζουν τα εξαρτήματα ενός τέτοιου αισθητήρα. Ο πίνακας 2.1 παραθέτει πληροφορίες σχετικά με τους αισθητήρες, για τα τέσσερα αέρια που μετράμε.



(i) Εξαρτήματα Αισθητήρα Αερίου



(ii) Αισθητήρας Ανίχνευσης Αερίων

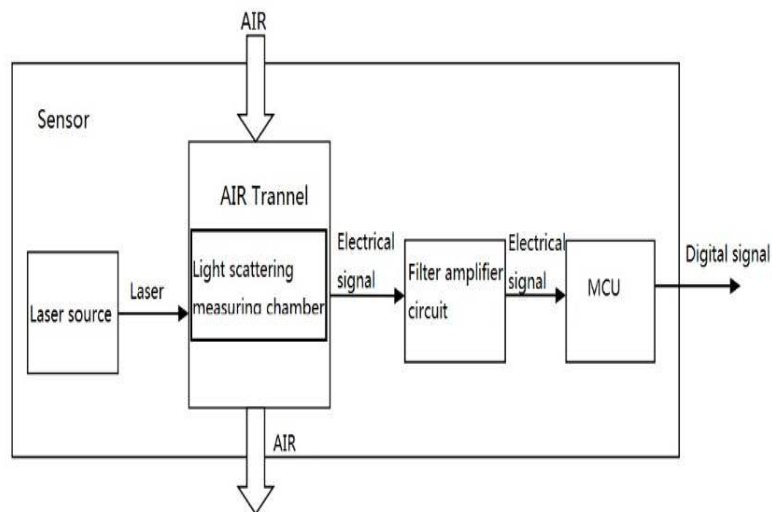
Εδώ θα πρέπει να αναφερθεί ο κύριος λόγος που επιλέξαμε ως μονάδα μέτρησης για την συγκέντρωση των αερίων τα ppm. Συγκεκριμένα, οι αισθητήρες ανίχνευσης αερίων που μόλις αναφέραμε καταγράφουν τις τιμές σε ppm. Οι τιμές αυτές είναι εργοστασιακά ρυθμισμένες και η καταμέτρησή τους σε οποιαδήποτε άλλη μονάδα μέτρησης, χρειάζεται να δηλωθεί από τον προγραμματιστή. Επειδή, λοιπόν, συμβαίνει αυτό και επειδή όλο και περισσότεροι οργανισμοί χρησιμοποιούν αυτήν την μετρική, η μονάδα μέτρησης της συγκέντρωσης των αερίων που επιλέχθηκε, είναι η συγκεκριμένη. Για την αλλαγή μονάδας μέτρησης, μπορεί να χρησιμοποιηθεί η εξίσωση 2.2. Η τιμή 24.45 είναι τα μικρολίτρα ενός αερίου που παράγεται από ένα μικρο-mole μιας ουσίας σε θερμοκρασία και πίεση αναφοράς (25 °C και 760 mmHg). Το M είναι το μοριακό βάρος της χημικής ένωσης.

$$1 \text{ ppmv} = \frac{24.45}{M * 1000} 1 \mu\text{g}/\text{m}^3 \quad (2.2)$$

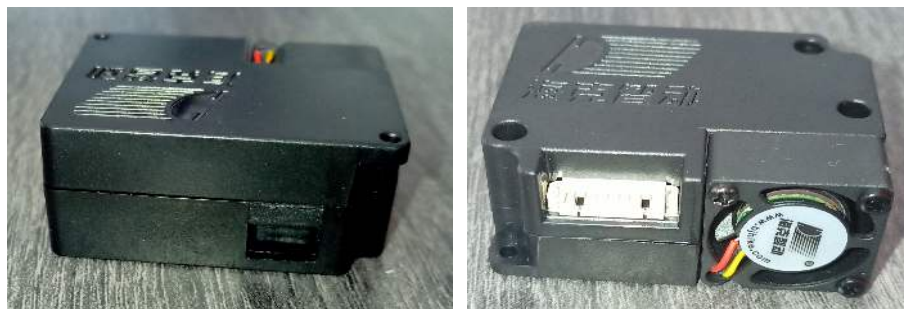
Αισθητήρες Σωματιδίων

Ο αισθητήρας ανίχνευσης σωματιδίων PM_{2.5} είναι ένας ψηφιακός αισθητήρας, που μετράει την συγκέντρωση σωματιδίων και μπορεί να χρησιμοποιηθεί για να ληφθεί ο αριθμός των αιωρούμενων σωματιδίων σε όγκο αέρα 0,3 έως 10 μικρόμετρα, δηλαδή όσο η συγκέντρωση των σωματιδίων, και να εξάγει με ψηφιακή διεπαφή, καθώς επίσης να εξάγει ποιοτικά δεδομένα, για τα σωματίδια που εισήχθησαν. Λειτουργεί με laser, και συγκεκριμένα ακολουθεί τις αρχές της θεωρίας σκέδασης φωτός. Συνοπτικά, χρησιμοποιεί την σκέδαση της ακτινοβολίας λέιζερ στα αιωρούμενα σωματίδια του αέρα, ενώ συλλέγει το σκεδαζόμενο φως σε μια συγκεκριμένη γωνία, για να λάβει την ένταση σκέδασης σε σχέση με την καμπύλη χρόνου, την οποία μετασχηματίζει στο πεδίο συχνοτήτων μέσω του μετασχηματισμού Fourier, και στη συνέχεια μέσω μιας σειράς πολύπλοκων αλγορίθμων προκύπτει ο αριθμός των σωματιδίων. Το σχήμα 2.3 είναι το διάγραμμα λειτουργίας του αισθητήρα. Ο αισθητήρας αποτελείται από 3 μέρη: τον αισθητήρα laser σκόνης (*Lazer Dust Sensor*), τον αντάπτορα

(Sensor Adapter) και το καλώδιο του αντάπτορα (Sensor Adapter Cable). Οι τιμές τάσης στις οποίες λειτουργεί είναι στο εύρος $[4.95, 5.05]$ V. Το μέγιστο ρεύμα που δέχεται είναι 120mA. Το μέγεθος (σε διάμετρο) των σωματιδίων διαιρείται σε 3 εύρη (λόγω του γεγονότος πως ο συγκεκριμένος αισθητήρας προορίζεται για μέτρηση και των 3 τύπων σωματιδίων): $[0.3, 1.0]$, $[1.0, 2.5]$, $[2.5, 10]$ (μm). Το εύρος τιμών είναι $[0, 999]$ $\mu m/m^3$. Ο χρόνος απόκρισης είναι $\leq 10s$. Η θερμοκρασία, κάτω απ' την οποία λειτουργεί, είναι $[-20, 50]$ $^{\circ}C$. Το εύρος της υγρασίας κάτω απ' την οποία λειτουργεί, είναι $[0, 99]$ %RH. Οι διαστάσεις του είναι 65x42x23 (mm). Εύρος ζωής μεγαλύτερο από 5 χρόνια. Το πρωτόκολλο μετάδοσης δεδομένων είναι το UART. Η εικόνα 2.4 περιέχει όλα τα εξαρτήματα του αισθητήρα ανίχνευσης σωματιδίων PM1.0, PM2.5 και PM10.0. Στις εικόνες 2.4i και 2.4ii βλέπουμε τον Αισθητήρα Lazer Σκόνης. Στις εικόνες 2.4iii και 2.4iv βλέπουμε τον αντάπτορα.



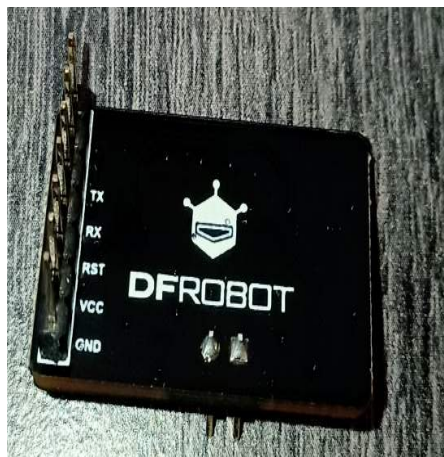
Σχήμα 2.3: Διάγραμμα Λειτουργίας Αισθητήρα Ανίχνευσης Σωματιδίων PM2.5 και PM10.0 (DFRobot, 2023)



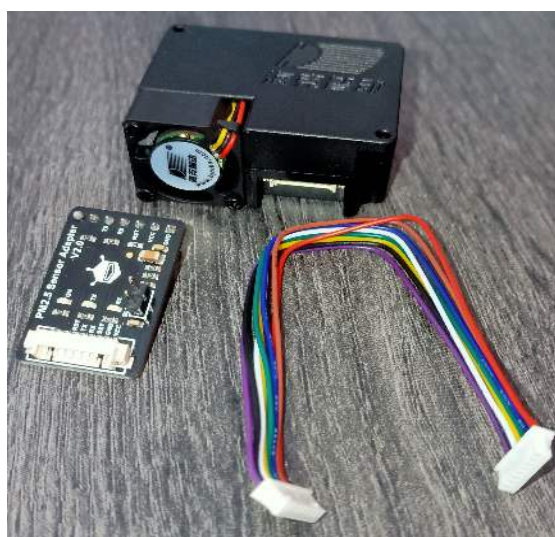
(i) Εμπρόσθια Όψη του Αισθητήρα Lazer(ii) Οπίσθια Όψη του Αισθητήρα Lazer Σκόνης



(iii) Εμπρόσθια Όψη του Αντάπτορα



(iv) Οπίσθια Όψη του Αντάπτορα



Σχήμα 2.4: Αισθητήρας Ανίχνευσης Σωματιδίων PM1.0, PM2.5 και PM10.0

Η επιλογή των αισθητήρων δεν είναι τυχαία. Οι συγκεκριμένοι αισθητήρες προσφέρουν πολλά πλεονεκτήματα για την ανάπτυξη ενός τέτοιου συστήματος. Πιο συγκεκριμένα, λόγω του μικρού μεγέθους και βάρους, οι διασώστες δεν επιβαρύνονται να φορέσουν κάποια ειδική στολή, που θα τους επιβραδύνει στο περπάτημα και στο σβήσιμο της φωτιάς. Ακόμη, οι συνθήκες λειτουργίας τους είναι ικανοποιητικές, καθώς οι θερμοκρασίες της συσκευασίας τους δεν ξεπερνάν την θερμοκρασία κάτω απ' την οποία λειτουργούν. Επίσης, η τάση λειτουργίας τους τους επιτρέπει να χρησιμοποιήσουν 5 V μπαταρία, οπότε το μέγεθος του συστήματος θα είναι σχετικά μικρό και άρα εύκολο να μεταφερθεί.

Το εύρος τιμών που καλύπτουν είναι επίσης ικανοποιητικό, για την συγκεκριμένη εφαρμογή. Συγκεκριμένα, όπως φαίνεται στον πίνακα 2.1, η ανάλυση του εύρους τιμών των αισθητήρων ανίχνευσης των αερίων NO₂, SO₂ και O₃ διαβάζει τιμές τάξης ενός δεκαδικού ψηφίου. Όπως φαίνεται στον πίνακα 2.2 οι τιμές των αερίων σε ppm είναι της τάξης των δύο δεκαδικών ψηφίων. Αν και αυτό φαίνεται να παρουσιάζει πρόβλημα αρχικά, εν τούτοις, αν αναλογιστούμε την ποιότητα του αέρα σε ένα δασικό περιβάλλον που πλήττεται από πυρκαγιά, εφαρμόζοντας στρογγυλοποίηση κατά ένα ψηφίο τις τιμές του πίνακα 2.2, τότε οι αισθητήρες είναι

ικανοί να διαχωρίσουν τις τιμές των αερίων που αντιστοιχούν στις τιμές της ποιότητας του αέρα σε αυτές τις συνθήκες. Με απλά λόγια, οι αισθητήρες δεν είναι ικανοί να ανιχνεύσουν κατά πόσο είναι καλή η ποιότητα του αέρα αλλά είναι ικανοί να διαχωρισούν την καλή απ' την κακή ποιότητα του αέρα και να ανιχνεύσουν πόσο κακή είναι, το οποίο και είναι το μείζον ζήτημα στο κομμάτι αυτό της εργασίας.

Πίνακας 2.2: Πίνακας Ευρωπαϊκού Δείκτη Ποιότητας Αέρα. Οι τιμές είναι σε μονάδα ppm.

Αέριο (Μοριακό Βάρος)	Καλό	Ικανοποι- ητικός	Εντάξει	Κακός	Πολύ Κακός	Επικίν- δυνος
NO ₂ (46)	0.00-0.02 ppm	0.02-0.05 ppm	0.05-0.06 ppm	0.06-0.12 ppm	0.12-0.18 ppm	0.18-0.53 ppm
SO ₂ (64)	0.00-0.04 ppm	0.04-0.08 ppm	0.08-0.13 ppm	0.13-0.19 ppm	0.19-0.29 ppm	0.29-0.48 ppm
O ₃ (48)	0.00-0.03 ppm	0.03-0.05 ppm	0.05-0.07 ppm	0.07-0.12 ppm	0.12-0.19 ppm	0.19-0.41 ppm

2.1.2 Μικροϋπολογιστής Raspberry Pi

Όπως αναφέρθηκε και προηγουμένως, για την προ-επεξεργασία και αποστολή των δεδομένων στον server, υπεύθυνος είναι ένας μικροϋπολογιστής Raspberry Pi. Ένα Raspberry Pi δεν αποτελεί τίποτα άλλο, παρά έναν φθηνό κλασσικό υπολογιστή γενικής χρήσης σε μέγεθος τσέπης, ο οποίος παρέχει όλες τις σημαντικές λειτουργίες ενός συμβατικού υπολογιστή, προσφέροντας, παράλληλα, την δυνατότητα ανάπτυξης έργων (*projects*), όπως εκπαιδευτικές εφαρμογές στην ρομποτική και IoT αλλά και σε ερευνητικά έργα, όπως συστήματα αισθητήρων (όπως το παρόν), μετεωρολογιακούς σταθμούς και γενικότερα, οτιδήποτε προσφέρει ένας μικροελεγκτής. Αυτός ο μικροϋπολογιστής κυκλοφόρησε για πρώτη φορά το 2012, στο Ηνωμένο Βασίλειο, ως εκπαιδευτικό εργαλείο από το ομόνυμο ίδρυμα. Εξ' αιτίας της τεράστιας απήχησης του και εξ' αιτίας του χαμηλού κόστους και των δυνατοτήτων που προσέφερε, η χρησιμότητά του επεκτάθηκε στα παραπάνω. Πλέον, κυκλοφορούν 15 μοντέλα στην αγορά, το καθένα με της δικιάς του γενιάς και παραλλαγής βάσει RAM (1GB, 2GB, 4GB, 8GB).

Στα πλαίσια της εργασίας, χρησιμοποιήσαμε ένα Raspberry Pi 4 Model B, το πιο πρόσφατο μοντέλο της τελευταίας γενιάς Raspberry Pi. Όπως και οι προγενέστεροί του, το μοντέλο αυτό έχει ένα σύστημα σε chip - *System on Chip* (SoC), το οποίο παρέχει στο Raspberry Pi την μονάδα επεξεργασίας γενικού σκοπού, την απόδοση των γραφικών και τις δυνατότητες Ε/Ε. Το SoC του είναι το Broadcom BCM2711, το οποίο περιέχει έναν τετραπύρην 64-bit επεξεργαστή Cortex-A72, ο οποίος λειτουργεί στα 1.8 GHz. Το μοντέλο έχει 8GB LPDDR4-3200 SDRAM, διαθέτει 2.4 GHz και 5.0 GHz IEEE 802.11ac wireless πρότυπο δικτύωσης, 40 GPIO Pins και τροφοδεύεται με 5V DC από USB Type-C αντάπτορα ή από GPIO header. Στον πίνακα 2.3 παραθέτονται πλήρως τα χαρακτηριστικά του Raspberry Pi 4 Model B. Να σημειωθεί, πως όλα τα μοντέλα Raspberry Pi δεν έρχονται με κάποιο μέσο διατήρησης των δεδομένων, όταν δεν παρέχεται τροφοδοσία ρεύματος, και συνεπώς δεν μπορούν να διατηρήσουν λειτουργικό σύστημα και προγράμματα εφαρμογών. Έχουν, όμως,

υποδοχές USB και Micro-SD, όπως φαίνεται και στον πίνακα 2.3. Η εγκατάσταση του λειτουργικού συστήματος και εφαρμογών είναι απαραίτητο να γίνει σε κάρτα Micro-SD. Η αποθήκευση δεδομένων, δύναται να γίνει είτε στην κάρτα, είτε σε εξωτερικό μέσο αποθήκευσης.

Ο συγκεκριμένος μικροϋπολογιστής, παρ' όλο που αποτελεί και υπολογιστή γενικού σκοπού, διαφέρει αρκετά από έναν συμβατικό υπολογιστή. Συγκεκριμένα, λόγω της φιλοσοφίας της σχεδίασης του (όλα τα εξαρτήματα σε μία πλακέτα), χρειάζεται, όπως αναφέρθηκε προηγουμένως, ένα SoC, που να ενσωματώνει όλες τις βασικές δυνατότητες ενός συμβατικού υπολογιστή σε ένα εξάρτημα στην πλακέτα. Το συγκεκριμένο SoC έχει σχεδιαστεί με βάσει την αρχιτεκτονική ARM (πιο συγκεκριμένα, ARM A72 core), η οποία δεν εντοπίζεται σε συμβατούς υπολογιστές. Η αρχιτεκτονική αυτή αποτελεί την φυσική λύση στο πρόβλημα επιλογής SoC, αφού προσφέρει ένα περιορισμένο σύνολο εντολών, με χαμηλή κατανάλωση ενέργειας, την δυνατότητα τροφοδοσίας ρεύματος τάσης 5V και εξαλείφει την ανάγκη ύπαρξης ψυκτικού μηχανισμού, αφού η χαμηλή κατανάλωση ρεύματος οδηγεί σε μικρή παραγωγή θερμότητας, ακόμη και στις πιο σύνθετες πράξεις. Το μειονέκτημά της είναι, ότι δεν είναι συμβατή με εφαρμογές, που έχουν αναπτυχθεί για συμβατικούς υπολογιστές, οι οποίοι έχουν x86 σετ εντολών στις αρχιτεκτονικές τους.

Εκτός από την αρχιτεκτονική, μια σημαντική διαφορά εντοπίζεται και στο λειτουργικό σύστημα. Οι περισσότεροι συμβατικοί υπολογιστές γενικού σκοπού εκτελούν είτε Microsoft Windows, είτε Apple Mac OS X. Και τα δύο λειτουργικά συστήματα είναι closed source, όσον αφορά τον κώδικά τους, αφού αυτός δεν βρίσκεται κάπου διαθέσιμος για τους χρήστες. Αντ' αυτού, ο κώδικας είναι «κρυμμένος» και οι χρήστες λαμβάνουν έτοιμο το τελικό προϊόν. Δεν έχουν την δυνατότητα να δουν τον κώδικα και να τον τροποποιήσουν. Το Raspberry Pi τρέχει σε *GNU/Linux* ή απλά Linux, ένα λογισμικό open source. Ο κώδικάς του είναι διαθέσιμος στους χρήστες και μπορούν οι ίδιοι να τον τροποποιήσουν, όπως επιθυμούν. Ο κώδικας για κάθε μέρος του λειτουργικού είναι διαθέσιμος και όλες οι αλλαγές είναι ορατές. Με βάση αυτό το σκεπτικό, έχουν αναπτυχθεί χιλιάδες εκδόσεις Linux, πολλές απ' τις οποίες ονομάζονται *διανομές (distributions)*, λόγω των μεγάλων διαφορών που έχουν (στην εμφάνιση, στην λειτουργικότητα, στις απαιτήσεις σε πόρους, στο συντακτικό των εντολών, κ.λπ.). Τα περισσότερα Raspberry Pi τρέχουν μια δική τους διανομή Linux, το Raspberry Pi Os (πρώην Raspbian). Συνοπτικά, είναι μια τροποποιημένη διανομή της διανομής Debian για αρχιτεκτονική ARM, η οποία έχει ρυθμιστεί κατάλληλα, ώστε να μπορεί να μεταφερθεί στο Raspberry Pi. Εκτός από το Raspberry Pi OS, υπάρχει η δυνατότητα να τρέξουν και σε άλλες διανομές, οι οποίες είναι ειδικά διαμορφωμένες και έχουν μεταφερθεί σε Raspberry Pi, όπως η διανομή Arch Linux, Debian Linux κ.λπ. Το πλεονέκτημα εδώ είναι ότι ο κώδικας είναι open source και ότι πολλές εφαρμογές που δεν είναι συμβατές με την αρχιτεκτονική ή το λειτουργικό σύστημα, έχουν εναλλακτικές, που μπορούν να τρέξουν στις διανομές Linux, είναι open source και δωρεάν, ενώ πολλές από αυτές έρχονται προ-εγκατεστημένες στις διανομές.

Πίνακας 2.3: Χαρακτηριστικά (Specs) Raspberry Pi 4 Model B (Raspberry Pi, 2023)

Χαρακτηριστικά	
Επεξεργαστής	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
Μνήμη	1GB, 2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)
Πρωτόκολλα Ασύρματης Επικοινωνίας	2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE, Gigabit Ethernet
Θύρες USB Ακροδέκτες	2 USB 3.0 ports; 2 USB 2.0 ports Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
Έξοδοι HDMI	2 × micro-HDMI ports (up to 4kp60 supported)
Σειριακή Έξοδος Οθόνης	2-lane MIPI DSI display port
Σειριακή Έξοδος Κάμερας	2-lane MIPI CSI camera port
Έξοδοι Ήχου Βίντεο	4-pole stereo audio and composite video port
Κωδικοποίηση Βίντεο	H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
Βιβλιοθήκες Γραφικών	OpenGL ES 3.1, Vulkan 1.0
Θύρα Micro SD	Micro-SD card slot for loading operating system and data storage
Τροφοδοσία Ρεύματος μέσω Θύρας	5V DC via USB-C connector (minimum 3A*)
Τροφοδοσία Ρεύματος μέσω Ακροδέκτη	5V DC via GPIO header (minimum 3A*)
Τροφοδοσία μέσω Ethernet	Power over Ethernet (PoE) enabled (requires separate PoE HAT)
Θερμοκρασία Λειτουργίας	Operating temperature: 0 – 50 degrees C ambient

Για την ανάπτυξη project, το Raspberry Pi διαθέτει ένα σύνολο ακροδεκτών (pins). Αυτά τα pins ονομάζονται *pins Γενικού-Σκοπού Εισόδου-Εξόδου (General Purpose Input-Output (GPIO) pins)*. Τα GPIO pins βρίσκονται πάνω αριστερά στην πλακέτα. Είναι 40 στο πλήθος, για το μοντέλο που χρησιμοποιούμε και γενικά στα πιο πρόσφατα μοντέλα (τα παλαιότερα μοντέλα είχαν 26), χωρισμένα σε δύο στήλες. Κάθε GPIO pin έχει τον δικό του σκοπό. Στο διάγραμμα του σχήματος 2.5 παρουσιάζονται τα pins με το όνομά τους (λευκά γράμματα) και τον ρόλο τους (αναγράφεται στις παρενθέσεις) και η αρίθμυσή τους (μαύρα γράμματα). Παρατηρούμε, ότι υπάρχουν τέσσερα pin για την τροφοδοσία του ρεύματος, δύο των 5V και δύο των 3.3V, οχτώ για την γείωση, έντεκα για τους διαύλους ανταλλαγής δεδομένων, τέσσερα για την διαμόρφωση παλμικού κώδικα και δεκατρία γενικού σκοπού. Τα GPIO pins που δεν έχουν συγκεκριμένο ρόλο, μπορούν να χρησιμοποιηθούν είτε ως πηγή τάσης 3.3V, είτε ως γείωση, είτε ως είσοδο. Τα υπόλοιπα είναι δεσμευμένα στην λειτουργία τους και οποιαδήποτε αλλαγή μπορεί να οδηγήσει σε

βλάβη του Raspberry Pi. Η αλλαγή λειτουργίας γίνεται με επεξεργασία του αρχείου config.txt, το οποίο περιέχει όλες τις παραμέτρους του hardware.



Σχήμα 2.5: Διάγραμμα GPIO pins (Pinout, 2023)

Όπως αναφέρθηκε προηγουμένως, το Raspberry Pi δεν έρχεται προεγκατεστημένο με κάποιο λειτουργικό σύστημα, όμως υπάρχουν πολλά ελεύθερα και συμβατά με αυτό. Στην εργασία αυτή, επιλέξαμε το επίσημο λειτουργικό σύστημα του ιδρύματος, το Raspberry Pi OS. Το συγκεκριμένο λειτουργικό σύστημα έρχεται με περισσότερα από 35000 πακέτα και πολλές εφαρμογές, που περιέχουν και οι κλασσικοί υπολογιστές. Η εγκατάστασή του έγινε με την χρήση του προγράμματος Raspberry Pi Imager και μιας Secure Digital (SD) κάρτας.

Γλώσσα Python

Ο μικροϋπολογιστής Raspberry Pi χρησιμοποιεί για την ανάπτυξη των εφαρμογών την γλώσσα Python. Η γλώσσα αυτή έρχεται προεγκατεστημένη στις περισσότερες διανομές Linux και αυτό ισχύει και για το Raspberry Pi OS. Είναι μια γλώσσα γενικού σκοπού υψηλού επιπέδου, που δημιουργήθηκε το 1990 από τον Guido Van Rossum. Οι γλώσσες υψηλού επιπέδου προσφέρουν ευκολία στην ανάπτυξη και κατανόηση του κώδικα από την πλευρά του προγραμματιστή, λόγω των ευέλικτων μεταβλητών, δομών και εντολών που προσφέρουν για την ανάπτυξη λογισμικού, οι οποίες προσφέρουν ευκολία στην ανάπτυξη, μεγαλύτερη απόδοση στον κώδικα καθώς και μειωμένο κόστος ανάπτυξης και αποσφαλμάτωσης. Επίσης, είναι μια διεργασμένη γλώσσα, δηλαδή ο κώδικας μηχανής παράγεται από την τρέχουσα γραμμή κατά την διάρκεια της εκτέλεσης. Το πλεονέκτημα που προσφέρει είναι εντοπισμός σφαλμάτων και καλύτερη αποσφαλμάτωση αλλά και η διακοπή, επεξεργασία και επανεκτέλεση του κώδικα, χωρίς την μεσολάβηση της μεταγλώττισης για ολόκληρο το πρόγραμμα. Παρ' όλα αυτά, ένα σημαντικό μειονέκτημα, είναι η ταχύτητα της εκτέλεσης. Αυτό, όμως, δεν μας περιορίζει στην συγκεκριμένη εφαρμογή, καθώς οι χρόνοι εκτέλεσης που απαιτούνται δεν επηρεάζονται από αυτήν την καθυστέρηση στην εκτέλεση. Το επιπλέον πλεονέκτημα της Python (και το οποίο είναι αυτό που έχει κερδίσει αρκετούς αρχάριους προγραμματιστές) είναι

απλότητα στην σύνταξη. Αυτή η απλότητα βοηθάει τον προγραμματιστή στο να εστιάσει στην ανάπτυξη του κώδικα όπως αυτός επιθυμεί, χωρίς να εμπίπτει στις ιδιαιτερότητες των γλωσσών. Οι τελευταίες ως επί το πλείστον είναι όσο το δυνατό λιγότερες, λόγω της καλής σχεδίασης της γλώσσας, η οποία επιλύει τα ζητήματα που μπορεί να προκύψουν και προσφέρει στον προγραμματιστή μια απλή και κατανοήτη διεπαφή. Η καλή σχεδίαση της γλώσσας μαζί με την απλή της σύνταξη προσφέρει την δυνατότητα ανάπτυξης κώδικα τόσο αντικειμενοστραφή όσο και συναρτιασιακό ή και τον συνδυασμό αυτών των δύο.

Πέραν των στοιχείων της δομής της γλώσσας, η γλώσσα Python είναι γλώσσα ελεύθερου λογισμικού, που σημαίνει ότι οποιοσδήποτε επιθυμεί μπορεί να την εγκαταστήσει δωρεάν, ενώ πληθώρα υλικού για την ανάπτυξη προγραμμάτων και πληροφοριών σχετικά με αυτή υπάρχουν διαθέσιμα. Το μεγαλύτερο πλεονέκτημα που προσφέρει αυτό είναι ο τεράστιος όγκος από βιβλιοθήκες που καλύπτουν τις βασικές ανάγκες της γλώσσας και προσφέρουν όλες τις απαραίτητες υποδομές και εργαλεία για την ανάπτυξη πληθώρας εφαρμογών, όπως εφαρμογές διαδικτύου, στις οποίες επικεντρωνόμαστε στην παρούσα εργασία. Όλες οι βιβλιοθήκες είναι ανοιχτού κώδικα και είναι διαθέσιμες προς όλους τους χρήστες της γλώσσας. Αυτό συμβάλλει στην ανάπτυξη ελεύθερου λογισμικού με την Python το οποίο έχει ως αποτέλεσμα ο όγκος να κλιμακώνεται και η ποιότητα των βιβλιοθηκών να βελτιώνεται, αλλά και οι τεχνολογίες οι οποίες στηρίζονται σε ανοιχτό λογισμικό, όπως οι τεχνολογίες διαδικτύου νέας γενιάς, να την στηρίζουν. Οι λόγοι αυτοί καθιστούν την Python την πρώτη επιλογή για την ανάπτυξη του κώδικα στο Raspberry Pi αλλά, όπως θα δούμε και στο επόμενο κεφάλαιο, και του server για την επικοινωνία μεταξύ κόμβων και πελατών.

Η εγκατάσταση των βιβλιοθηκών γίνεται με την χρήση ενός εργαλείου που ονομάζεται *Package Installer for Python (pip)*. Αυτό το εργαλείο επιτρέπει τον εντοπισμό και την εγκατάσταση βιβλιοθηκών και πακέτων από το αποθετήριο λογισμικού της Python (*Python Package Index (PyPI)*) μόνο με το όνομα του πακέτου που επιθυμούμε να εγκαταστήσουμε. Το pip έρχεται προεγκατεστημένο μαζί με την γλώσσα Python και συνεπώς, δεν χρειάζεται κάποια επιπλέον ενέργεια όσον αφορά την εγκατάσταση ή την παραμετροποίηση (εξαιρέσεις υπάρχουν, αλλά δεν θα μας απασχολήσουν στα πλαίσια της εργασίας). Παρέχει μια πληθώρα δυνατοτήτων, όπως την εγκατάσταση πολλαπλών βιβλιοθηκών, την εγκατάσταση βιβλιοθηκής από συμπιεσμένα αρχεία, την ενημέρωση πακέτων κ.λπ. Η εντολή 1 δείχνει την τυπική διαδικασία εγκατάστασης πακέτου μέσω pip. Με αυτή την εντολή, το pip εγκαθιστά την νεότερη διαθέσιμη έκδοση του πακέτου απ' το PyPI. Αυτήν θα χρησιμοποιήσουμε για την εγκατάσταση όλων των απαραίτητων πακέτων της εργασίας.

```
pip install <package_name>
```

Listing 1: Εγκατάσταση μέσω pip.

2.1.3 Συνδεσμολογία

Αν παρατηρήσουμε το διάγραμμα του σχήματος 2.5 στο υποκεφάλαιο 2.1.2, θα εντοπίσουμε πως υπάρχουν τρεις τύποι διαύλων μετάδοσης δεδομένων: *I2C*, *UART*

και SPI. Από αυτούς τους τρεις, θα μας απασχολήσουν μόνο οι I2C και UART (επειδή οι δύο δίαυλοι ορίζονται από πρωτόκολλα που είναι συνώνυμα, εδώ ο όρος δίαυλος και πρωτόκολλο θεωρούνται ως επί το πλείστον ταυτόσημοι). Και οι δύο αποτελούν διαύλους σειριακής επικοινωνίας. Στην επικοινωνία μικροϋπολογιστή και περιφερειακών συσκευών, στην ανταλλαγή δεδομένων, η ροή των δεδομένων γίνεται με δύο μορφές επικοινωνίας: την σειριακή και την παράλληλη. Στην παράλληλη, κάθε bit του διαύλου δεδομένων διοχετεύεται σε ξεχωριστό κανάλι (φυσικό καλώδιο), ενώ στην σειριακή επικοινωνία, τα bits πρέπει να διοχετευτούν σε ένα μόνο κανάλι (Παπάζογλου, 2018). Στην σειριακή επικοινωνία, χρειάζεται να γνωρίζουν τα δύο άκρα την έναρξη και την λήξη της μετάδοσης δεδομένων, καθώς και τα bits να αποστέλλονται σωστά. Για αυτό, χρειάζονται κάποια επιπλέον σήματα, τα οποία να πληροφορούν αυτές τις λεπτομέρειες στον δέκτη. Αν αυτά τα σήματα είναι στο ίδιο κανάλι με τα ανταλλασσόμενα ψηφία, τότε η επικοινωνία ονομάζεται ασύγχρονη. Αν χρησιμοποιείται ξεχωριστό κανάλι, τότε τα σήματα είναι παλμοί ρολογιού και η επικοινωνία ονομάζεται σύγχρονη.

Το πρωτόκολλο επικοινωνίας *Inter-Integrated Circuits (I2C)* ορίζει σύγχρονη σειριακή επικοινωνία, με την χρήση δύο καναλιών ή γραμμών σημάτων: την *Serial Clock (SCL)* και την *Serial Data (SDA)*. Η SDA δύναται να αξιοποιηθεί και από την συσκευή ελέγχου (*controller*) ή αλλιώς *master* και από την περιφερειακή συσκευή ή αλλιώς *slave*. Όμως, μόνο ο controller χειρίζεται την SCL. Ο controller είναι η συσκευή που είναι υπεύθυνη για την επικοινωνία με τα περιφερειακά. Ο slave είναι ένα περιφερειακό (*peripheral*) που στηρίζεται στον controller για την επικοινωνία και απλώς αποστέλλει και λαμβάνει δεδομένα. Σε ένα τυπικό σύστημα, υπάρχει ένας controller και ένα ή περισσότερα περιφερειακά. Κάθε περιφερειακό έχει την δική του ξεχωριστή διεύθυνση στον δίαυλο, ώστε ο controller να μπορεί να το ξεχωρίζει απ' τα υπόλοιπα στον δίαυλο (Hyde, 2022). Αυτό, γιατί το πρωτόκολλο επιτρέπει την ύπαρξη πολλαπλών διαύλων επικοινωνίας. Αυτό με την σειρά του επιτρέπει στα περιφερειακά να βρίσκονται σε ίδιες διευθύνσεις μεταξύ τους, αλλά σε διαφορετικούς διαύλους.

Το πρωτόκολλο επικοινωνίας *Καθολικός Ασύγχρονος Δέκτης Μεταδότης - Universal Asynchronous Receiver/Transmitter (UART)*, ορίζει (όπως λέει και το όνομά του) ασύγχρονη σειριακή επικοινωνία. Ο συγχρονισμός των δύο άκρων επικοινωνίας απαιτεί και τα δύο να έχουν ίδιο ρυθμό αποστολής bit (*baud rate*). Για την αναγνώριση των δεδομένων μεταξύ τους, τα δεδομένα χωρίζονται σε πλαίσια. Ένα πλαίσιο, εκτός από την πληροφορία που μεταδίδεται (7, 8 ή 9 bits), περιέχει το bit ισοτιμίας, κάποια bit για την έναρξη και λήξη του πακέτου, ενώ η κωδικοποίηση των bit της πληροφορίας μπορεί να είναι είτε *Most Significant Bit First*, είτε *Least Significant Bit First*. Κάθε πλαίσιο αρχίζει με ένα bit έναρξης, που αναπαρίσταται με χαμηλή τάση. Ακολουθώς βρίσκονται τα bits της πληροφορίας που μεταδίδεται. Για την λήξη του πακέτου, προηγείται ένα προαιρετικό bit ισοτιμίας και έπεται ένα με δύο bits για την λήξη του πακέτου. Μια διεπαφή αυτού του πρωτοκόλλου έχει δύο άκρα, το άκρο TX, για την αποστολή πακέτων, και το άκρο RX, για την λήψη πακέτων. Για την επικοινωνία μεταξύ δύο συσκευών με διεπαφή UART, χρειάζονται δύο γραμμές επικοινωνίας, μία για την αποστολή δεδομένων από την πρώτη συσκευή και λήψη από την δεύτερη και μία για την αποστολή δεδομένων από την δεύτερη συσκευή και λήψη από την πρώτη. Υπάρχουν δύο δυνατές υλοποιήσεις: είτε *full-duplex*, είτε *single-wire half-duplex*. Η πρώτη χρησιμοποιεί δύο φυσικά καλώδια, τα οποία συνδέουν το TX της μίας με το RX της άλλης. Στην δεύτερη, χρησιμοποιείται ένα καλώδιο, που συνδέει τα δύο άκρα TX, συν ένα pull-up κύκλωμα (Zhu, 2018). Η

υλοποίηση full-duplex είναι η πιο συνήθης και η πιο απλή και για αυτό τον λόγο επιλέχθηκε.

Όπως αναφέραμε στο υποκεφάλαιο 2.1, οι αισθητήρες ανίχνευσης αερίων διαθέτουν 3 διαφορετικά πρωτόκολλα μετάδοσης δεδομένων, ενώ ο αισθητήρας ανίχνευσης σωματιδίων PM2.5 και PM10, διαθέτει μόνο το πρωτόκολλο UART. Στην εργασία αυτή, χρησιμοποιήσαμε το πρωτόκολλο UART για τον αισθητήρα ανίχνευσης σωματιδίων και το πρωτόκολλο I2C για τους αισθητήρες ανίχνευσης αερίων. Ο πρώτος λόγος είναι, ότι το πρωτόκολλο I2C επιτρέπει την επικοινωνία πολλών περιφερειακών συσκευών μέσω ενός διαύλου, με μεγάλο πλήθος διευθύνσεων στον δίαυλο για τις συσκευές και με ευκολία στην υλοποίηση και την διαχείριση της ροής δεδομένων από το κάθε περιφερειακό. Ο δεύτερος λόγος είναι, γιατί όπως αναφέραμε και στο υποκεφάλαιο 2.1.2, το Raspberry Pi είναι ικανό να υποστηρίξει αυτά τα πρωτόκολλα ταυτόχρονα.

Έχοντας αναφέρει τα παραπάνω, μπορούμε να προχωρήσουμε στην συνδεσμολογία των αισθητήρων και του Raspberry Pi. Σε πρώτη φάση, τοποθετούμε στο Raspberry Pi 5 καλώδια dupont, ένα στο pin 4 (όπως φαίνεται στον πίνακα 2.5) για την τροφοδοσία των αισθητήρων με ρεύμα, ένα στο pin 6 για την γείωση, ένα στο pin 3 για την γραμμή SDA του πρωτοκόλλου I2C, ένα στο pin 5 για την γραμμή SCL του πρωτοκόλλου I2C και ένα στο pin 10 για την λήψη δεδομένων από τον αισθητήρα ανίχνευσης σωματιδίων. Στην συνέχεια, συναρμολογούμε τον κάθε αισθητήρα. Αυτή η διαδικασία είναι αρκετά απλή: για τους αισθητήρες ανίχνευσης αερίων, η συναρμολόγησή τους χρειάζεται μόνο να κουμπώσουμε τα καλώδια και το probe στο module. Για τον αισθητήρα ανίχνευσης σωματιδίων, χρειάζεται να κουμπώσουμε το καλώδιο στον αντάπτορα και τον αισθητήρα σκόνης. Η σύνδεση όλων των εξαρτημάτων μεταξύ τους γίνεται μέσω ενός breadboard. Χάρη στην χρωματική κωδικοποίηση, η υπόλοιπη διαδικασία σύνδεσης των αισθητήρων και του Raspberry Pi είναι αρκετά εύκολη: τα κόκκινα καλώδια αντιστοιχούν στην τροφοδοσία, τα μαύρα στην γείωση, τα πράσινα στην γραμμή SDA και τα μπλε στην γραμμή SCL. Με βάση αυτό, πραγματοποιείται η λοιπή συνδεσμολογία.

2.1.4 Περιγραφή Λογισμικού

Έχοντας περιγράψει όλα τα συστατικά ενός κόμβου, τα πρωτόκολλα που διέπουν την επικοινωνία μεταξύ τους καθώς και την συνδεσμολογία όλων αυτών, μπορούμε πλέον να περιγράψουμε και το λογισμικό της εφαρμογής για την επικοινωνία και συλλογή αερίων. Όπως φαίνεται στον κώδικα 2.1 ένας αισθητήρας αερίων, πριν την έναρξη της λειτουργίας του, χρειάζεται να ορίσει την λήψη δεδομένων σε παθητική (δηλαδή τα δεδομένα δεν αποστέλλονται συνεχώς από τον αισθητήρα, αλλά μετά από αίτημα του Raspberry Pi), να συμπεριλαβεί στους υπολογισμούς του την αντιστάθμιση της θερμοκρασίας και τέλος να γνωρίζει ποιο αέριο ανιχνεύει.

Listing 2.1: Αρχικοποίηση αισθητήρα αερίων.

```
# 1. Setting data transmission to passive.
Gas_Sensor.change_acquire_mode(Gas_Sensor.PASSIVITY)

# 2. Take into account the temperature for gas concentration.
Gas_Sensor.set_temp_compensation(Gas_Sensor.ON)
```

```
# 3. Get gas type (CO, SO2, NO2, O3), based on the probe.  
Gas_Sensor.read_gas_type()
```

Για την συλλογή των δεδομένων από τους αισθητήρες, όπως και για την παραμετροποίηση του κώδικα 2.1, χρησιμοποιήθηκαν δύο custom βιβλιοθήκες. Οι βιβλιοθήκες αυτές παρέχονται από την εταιρία. Και οι δύο βιβλιοθήκες είναι γραμμένες για διαφορετική γλώσσα προγραμματισμού (C++), λόγω του ότι αυτοί οι αισθητήρες προορίζονται σε μικροελεγκτές, που την χρησιμοποιούν (π.χ. Arduino), και υποστηρίζουν τις παραπάνω ιδιότητες. Η βιβλιοθήκη για τους αισθητήρες αερίων είναι γραμμένη και για την γλώσσα Python. Η βιβλιοθήκη για τον αισθητήρα σωματιδίων είναι γραμμένη μόνο σε C++. Συνεπώς, έπρεπε ο κώδικας να ξαναγραφτεί σε Python, καθώς αυτό ήταν πιο εύκολη επιλογή, μιας και ο κώδικας δεν ήταν μεγάλος σε έκταση.

Κατά την συλλογή των δεδομένων, ο κόμβος τα καταχωρεί σε μια δομή. Τα δεδομένα αποθηκεύονται σε μια δομή μαζί με επιπλέον πληροφορίες σχετικά με τον κόμβο, όπως το αναγνωριστικό του και η τοποθεσία του, η οποία θα αποσταλλεί στον server. Η πιο απλή δομή για την μετάδοση δεδομένων στο διαδίκτυο είναι το JSON. Όπως αναφέρει και ο Αβούρης, η δομή JavaScript Object Notation (JSON) είναι ένα πρότυπο σειριοποίησης δεδομένων και αποθήκευής τους ως ακολουθιών από χαρακτήρες. Επειδή η δομή τους είναι απλή και ανεξάρτητη συστήματος, το πρότυπο αυτό χρησιμοποιείται σε πληθώρα εφαρμογών μετάδοσης δεδομένων, όπως σε βάσεις δεδομένων και εφαρμογές Διαδικτύου. Η δομή του μοιάζει πολύ με ένα λεξικό στην Python, όμως υπάρχουν κάποιες διαφορές, όπως ότι τα κλειδιά είναι υποχρεωτικά συμβολοσειρές και οι τιμές είναι μόνο βασικοί τύποι και δομές δεδομένων (όχι δηλαδή αντικείμενα ή custom δομές), για αυτό τα δεδομένα αποθηκεύονται σε ένα λεξικό που πληρεί τους περιορισμούς μιας δομής JSON. Στο λεξικό-JSON καταχωρούνται ένα λεξικό με κλειδιά τους ρύπους και τιμές τις μετρήσεις των αισθητήρων, έναν ακέραιο για το αναγνωριστικό του κόμβου και μια λίστα με το γεωγραφικό μήκος και πλάτος του κόμβου. Οι τιμές των αερίων καθώς και το μήκος και το πλάτος του κόμβου είναι τύπου float64, με ένα δεκαδικό ψηφείο, ενώ οι τιμές των σωματιδίων είναι ακέραιοι (int64).

Listing 2.2: Λήψη παρατηρήσεων από αισθητήρες.

```
while(True):  
    #PM2.5 Sensor  
    PMSensor.read_PM01()  
    PMSensor.read_PM2_5()  
    PMSensor.read_PM10()  
  
    SensorValuesDict = {  
        "Emissions":  
        {  
            str(Gas_1.gastype): Gas_1.read_gas_concentration(),  
            str(Gas_2.gastype): Gas_2.read_gas_concentration(),  
            str(Gas_3.gastype): Gas_3.read_gas_concentration(),  
            str(Gas_4.gastype): Gas_4.read_gas_concentration(),  
            "PM1.0": PMSensor.PM1_Val,  
            "PM2.5": PMSensor.PM2_5_Val,  
            "PM10.0": PMSensor.PM10_Val  
        },  
    },
```

```
"ID": <serial_id>,  
"Coordinates": [<latitude>, <longitude>]  
}
```

Μετά την συλλογή των δεδομένων και την καταχώρησή τους στην δομή JSON, ο κόμβος επικοινωνεί με τον με server. Για την επικοινωνία αυτή, χρησιμοποιείται η βιβλιοθήκη requests. Η βιβλιοθήκη αυτή είναι μια HTTP εξ' ολοκλήρου γραμμένη σε Python και προσφέρει πολλές δυνατότητες στην παραγωγή κώδικα (Chandra, 2015). Η εντολή 2 δείχνει πως γίνεται η εγκατάσταση της requests. Το HTTP και οι μεθόδοί του αναφέρονται στο κεφάλαιο 2.2. Για την ώρα, αναφέρεται μόνο ότι γίνεται χρήση της μεθόδου POST για την καταχώρηση νέων δεδομένων στον server. Η βιβλιοθήκη requests προσφέρει όλες τις μεθόδους του HTTP, με όλες τις κεφαλίδες ως παραμέτρους, την δυνατότητα αποστολής και λήψης δεδομένων και παραμέτρων και η απάντηση που επιστρέφει ο server είναι αποκωδικοποιημένη. Αυτό είναι πολύ χρήσιμο, καθώς δεν χρειάζεται κατά την ανάπτυξη του κώδικα να συμπεριλάβουμε την επεξεργασία που χρειάζεται να υποστεί η απάντηση, αλλά μόνο την απάντηση και τα περιεχόμενά της, έτσι όπως ορίζονται από το HTTP.

```
python3 -m pip install requests
```

Listing 2: Εγκατάσταση Βιβλιοθήκης requests.

Στην συγκεκριμένη εφαρμογή, το Raspberry Pi κάνει POST στον server το αρχείο JSON που περιγράφηκε παραπάνω. Για να μπορέσει να εντοπίσει τον server και να συνδεθεί σε αυτόν, χρειάζεται δύο πράγματα. Πρώτον, χρειάζεται να ξέρει την διεύθυνσή του και την θύρα που ακούει το μηχάνημα που τον τρέχει. Χωρίς αυτά, το Raspberry Pi δεν μπορεί να γνωρίζει που να στείλει τα δεδομένα. Εφόσον, τα αποκτήσει και επιχειρήσει να συνδεθεί, ο server αναμένει και κάποιο αναγνωριστικό. Αυτό γιατί, όπως θα δούμε παρακάτω, ο server περιλαμβάνει και έναν μηχανισμό αυθεντικοποίησης, ώστε να εμποδίζει κακόβουλους χρήστες από το να αποκτήσουν πρόσβαση σ' αυτόν και να υποκλέψουν αρχεία, εμποδίσουν την πρόσβαση σε άλλους χρήστες κ.λπ. Το αναγνωριστικό περιλαμβάνει το όνομα του χρήστη και έναν κωδικό. Μόλις ολοκληρωθεί η επεξεργασία των στοιχείων και η ταυτοποίηση είναι επιτυχής, τα δεδομένα του κόμβου καταχωρούνται σε μία βάση δεδομένων. Έπειτα ο κόμβος αναμένει για να κάποιο χρονικό διάστημα και επαναλαμβάνει την διαδικασία.

Listing 2.3: Αποστολή δεδομένων στον server.

```
url = "http://<IP>:<Port>/post-data"  
basicAuthCredentials = ("<username>", "<password>")  
serverResponse = requests.post(url, json=SensorValuesDict,  
                                auth=basicAuthCredentials)  
if(serverResponse.status_code != 200):  
    print("Error: " + str(serverResponse.status_code))  
else:  
    print("Success!")  
  
sleep(<wait_time>)
```

2.2 Απομακρυσμένος server

Τα δεδομένα που συλλέγονται από το σύστημα που παρουσιάστηκε στο κεφάλαιο 2.1 θα πρέπει να καταχωρούνται κάπου αλλού και όχι στο Raspberry Pi. Αυτό, γιατί το Raspberry Pi είναι, όπως αναφέραμε και στο υποκεφάλαιο 2.1.2, ένας υπολογιστής με συγκεκριμένες δυνατότητες, πόρους και σκοπό, οπότε η ανάπτυξη ενός συστήματος διαχείρισης της ροής πληροφορίας σε αυτό θα μπορούσε να υποβαθμίσει την ικανότητα του να συλλέγει δεδομένα σε πραγματικό χρόνο. Επίσης, ένα άλλο θέμα που τίθεται είναι, ότι εδώ αναφερόμαστε σε ένα σύστημα από κόμβους και όχι σε έναν μοναδικό κόμβο. Άρα, επειδή συλλέγονται δεδομένα από πολλούς κόμβους, η υλοποίηση ενός τέτοιου συστήματος σε έναν από αυτούς θα προκαλούσε προβλήματα, όπως τον συντονισμό αυτών, την επιλογή του κατάλληλου κόμβου για τον ρόλο κ.λπ. Επιπλέον, λόγω της τοποθεσίας τους, θα είχαν πληθώρα προβλημάτων όσον αφορά την δικτύωσή τους (επικοινωνία απομακρυσμένων υπολογιστών σε αυτά, λόγω κακού σήματος στο δάσος με πυρκαγιά, αργές ταχύτητες αποστολής δεδομένων, πιο πιθανό να χαθούν πακέτα με κρίσιμη πληροφορία, όταν στέλνονται εκατοντάδες κ.λπ.). Γίνεται φανερό, λοιπόν, ότι χρειάζεται τα δεδομένα που συλλέγονται, να αποστέλλονται μία φορά από τον κάθε κόμβο ανά τακτά χρονικά διαστήματα σε έναν κεντροποιημένο υπολογιστή, ικανό να διαχειριστεί πολλαπλά αιτήματα και διαμοιρασμό πληροφορίας.

Στο κεφάλαιο αυτό παρουσιάζεται ένα τέτοιο σύστημα. Συγκεκριμένα, παρουσιάζεται η τεχνική υλοποίηση ενός web server, ο οποίος θα διαχειρίζεται την πληροφορία που λαμβάνει από κάθε κόμβο, καθώς και τα αιτήματά που δέχεται από απομακρυσμένους υπολογιστές (clients). Θα παρουσιαστεί ένα REST API που αναπτύχθηκε με σκοπό την βέλτιστη εξυπηρέτηση των clients, καθώς και την διαχείριση των δεδομένων από τους κόμβους. Επιπλέον, θα αναλυθεί ο τρόπος που αποθηκεύεται και οργανώνεται η πληροφορία που δέχεται από τους κόμβους, με την χρήση NoSQL βάσης δεδομένων. Ακόμη, θα παρουσιαστεί ένα πρωτόκολλο για την ασφάλεια των δεδομένων, με την δημιουργία ενός συστήματος χρηστών. Τέλος, θα παρουσιαστούν όλες οι επιμέρους τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση και ανάπτυξη του server.

Βασικές Έννοιες

Η διαδικασία επικοινωνίας δύο απομακρυσμένων συσκευών αποτελείται από πολλά επίπεδα. Από το επίπεδο της φυσικής μεταφοράς μέχρι το επίπεδο εφαρμογών με τις οποίες αλληλεπιδρούν οι χρήστες, οι τεχνολογίες και τα πρωτόκολλα που έχουν αναπτυχθεί είναι αμέτρητα. Στην παρούσα εργασία, δεν θα εστιάσουμε σε όλα, αλλά, χρησιμοποιώντας προϋπάρχουσες υποδομές και λογισμικά, θα εστιάσουμε σε αυτά που χρησιμοποιούνται για την ανάπτυξη εφαρμογών για την διαχείριση της ροής πληροφορίας από τους κόμβους που την συλλέγουν μέχρι τον τελικό χρήστη που λαμβάνει είτε την ακατέργαστη είτε την επεξεργασμένη πληροφορία. Το πρώτο και πιο σημαντικό πρωτόκολλο είναι το HTTP.

Η μεταφορά των δεδομένων σε επίπεδο εφαρμογών στον παγκόσμιο ιστό, γίνεται με την χρήση του *Πρωτοκόλλου Μεταφοράς Υπερ-κειμένου - HyperText Transfer Protocol (HTTP)*. Όπως αναφέρει και ο Tanenbaum, το HTTP είναι ένα απλό πρωτόκολλο αίτησης-απάντησης, το οποίο προσδιορίζει ποια μηνύματα μπορούν να στείλουν οι πελάτες στους διακομιστές και ποια μηνύματα μπορούν να λάβουν

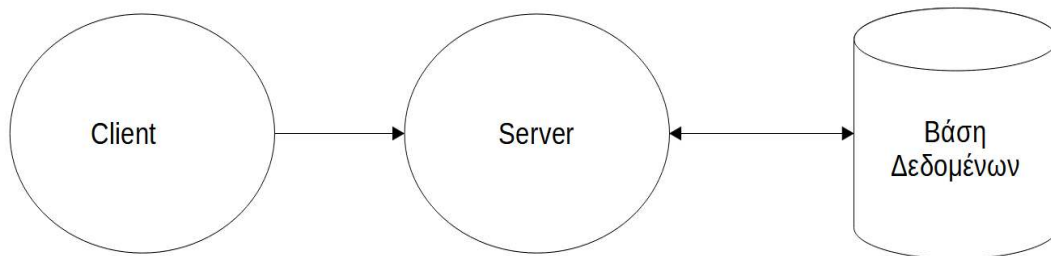
σε απάντηση. Οι αιτήσεις αποτελούν πακέτα που περιέχουν ερωτήματα και όλες τις απαραίτητες πληροφορίες τις οποίες θα χρειαστεί ο server, ενώ η απάντηση αποτελεί απαντητικό μήνυμα σε μία αίτηση (ένα αρχείο HTML, τον κώδικα προς εκτέλεση, αρχείο JSON κ.λπ.). Το πρωτόκολλο αυτό δεν χρησιμοποιείται μόνο στην επικοινωνία server και φυλλομετρητή ιστού, αλλά και στην επικοινωνία εφαρμογών και, πλέον με το IoT, συσκευών με έναν διακομιστή, μια δυνατότητα που θα μας χρειαστεί.

Η πιο απλή λογική είναι ένας web server που δέχεται αιτήματα και επιστρέφει ως απάντηση είτε ένα αρχείο HyperText Markup Language (HTML) με την πληροφορία, είτε κάποιες τιμές. Και στις δύο περιπτώσεις όμως, επιστρέφει μαζί και έναν κωδικό που αντιστοιχεί σε ένα μήνυμα επιτυχίας ή αποτυχίας. Το HTTP δεν καθορίζει μόνο τις αιτήσεις-απαντήσεις, αλλά υποστηρίζει και ένα σύνολο από βοηθητικές εντολές που ονομάζονται μέθοδοι. Στον πίνακα 2.4 παρουσιάζονται οι τέσσερις βασικές εντολές του HTTP, οι οποίες χρησιμοποιήθηκαν και στην ανάπτυξη του server. Αυτές οι εντολές χρησιμοποιούνται κατά κόρον σε όλες τις εφαρμογές που χρησιμοποιούν το πρωτόκολλο. Η εντολή GET χρησιμοποιείται για να ανακτήσουμε ένα πόρο, είτε αυτός είναι μια ιστοσελίδα (ένα αρχείο HTML), είτε ένα ή πολλά αρχεία (JSON, XML, RAW, κ.λπ.), είτε απλώς έναν κωδικό ότι όλα λειτουργούν. Η εντολή POST χρησιμοποιείται για την εισαγωγή νέων δεδομένων σε έναν πόρο, π.χ. μία νέα καταγραφή σε μία βάση δεδομένων. Η PUT ενημερώνει την τιμή ενός πόρου και η DELETE διαγράφει την τιμή ενός πόρου.

Πίνακας 2.4: Βασικές Μέθοδοι CRUD HTTP

Μέθοδος	Περιγραφή
GET	Ανάκτηση πόρου
POST	Προσάρτηση δεδομένων σε πόρο
PUT	Ενημέρωση δεδομένων πόρου
DELETE	Διαγραφή πόρου

Όπως ήδη έχει αναφερθεί, η διαχείριση και η μεταφορά των δεδομένων στηρίζεται σε ένα μοντέλο επικοινωνίας πελάτη-εξυπηρετητή/διακομιστή (client-server). Όπως μαρτυρά και το όνομά του, σε αυτό το μοντέλο τα δύο άκρα επικοινωνίας χωρίζονται σε δύο ρόλους, αναλόγως τις λειτουργίες τους. Το άκρο που είναι υπεύθυνο για την εγκαθίδρυση της επικοινωνίας και την παροχή δεδομένων και υπηρεσιών αποτελεί τον server. Το άκρο που αποστέλλει αιτήματα στον server και λαμβάνει πληροφορία από αυτόν ονομάζεται client. Ο server εκτελείται συνεχώς και αναμένει αιτήματα από τις διάφορες εργασίες client, που σε αόριστο χρονικό διάστημα στέλνουν αιτήματα. Αυτά τα αιτήματα τα αναζητεί σε μία βάση δεδομένων και επιστρέφει το αποτέλεσμα της αναζήτησης στο μήνυμα της απάντησης. Αυτό το μοντέλο αποτελεί το κυρίαρχο μοντέλο εφαρμογών παγκόσμιου ιστού και πλέον σχεδόν όλες (αν όχι όλες) οι εφαρμογές που υπάρχουν στον παγκόσμιο ιστό στηρίζονται σε αυτό. Στο σχήμα 2.6 αναπαρίσταται σχηματικά ο τρόπος λειτουργίας του μοντέλου.



Σχήμα 2.6: Μοντέλο Client-Server

Το μοντέλο αυτό παρέχει πολλά οφέλη στην υλοποίηση του συστήματος διαχείρισης και μεταφοράς της πληροφορίας. Κατ' αρχάς, διαχωρίζει την επεξεργασία της εφαρμογής σε πολλές μηχανές. Αυτό είναι πολύ χρήσιμο, αν αναλογιστούμε το πλήθος των συσκευών που επικοινωνούν και την ποικιλομορφία των ενεργειών των συμμετεχόντων. Για παράδειγμα, τα Raspberry Pis συλλέγουν τα δεδομένα για την πυρκαγιά και οι χρήστες ζητούν πληροφορίες σχετικά με αυτή. Αυτές οι πληροφορίες προκύπτουν από επεξεργασία των δεδομένων από τα Raspberry Pis. Αυτή η επεξεργασία δεν μπορεί να γίνει στα Raspberry Pis και να αποσταλλεί στους χρήστες, λόγω δυσκολιών στην ανάπτυξη της εφαρμογής και την επικοινωνία μεταξύ τους. Επομένως, χρειάζεται κάποιες ενέργειες, όπως η μορφοποίηση των δεδομένων και η αναπαράσταση της πληροφορίας να γίνεται από τα Raspberry Pi και τους χρήστες και η επεξεργασία και αποθήκευση των δεδομένων προς παραγωγή της πληροφορίας να γίνεται από τον server. Πέρα από αυτό, το μοντέλο client-server επιτρέπει τον ευκολότερο διαμοιρασμό των πόρων μεταξύ των client και του server. Συγκεκριμένα, η πληροφορία που παρέχουν οι clients δεν χρειάζεται να αποστέλλεται σε κάθε έναν ξεχωριστά, αλλά συγκεντρώνεται σε έναν κεντροποιημένο υπολογιστή, στον οποίο έχουν πρόσβαση και μπορούν να την ανακτήσουν ανεξάρτητα κάθε ένας. Τέλος, μειώνει την αντιγραφή δεδομένων με την αποθήκευση δεδομένων σε κάθε διακομιστή αντί για τον πελάτη, καθώς όπως και πριν, δεν χρειάζεται να έχουν όλοι οι πελάτες ένα αντίγραφο, αλλά όλη η πληροφορία συγκεντρώνεται σε έναν κεντροποιημένο υπολογιστή.

2.2.1 Αρχιτεκτονική REST

Για την οργάνωση και μεταφορά των δεδομένων μιας υπηρεσίας web, έχουν αναπτυχθεί διάφορα μοντέλα και αρχιτεκτονικές για διαφορετικούς σκοπούς. Η αρχιτεκτονική που χρησιμοποιήσαμε είναι η *Μεταφορά με Αναπαραστάσιμες Καταστάσεις - Representational State Transfer (REST)*. Η REST είναι μια αρχιτεκτονική που προσφέρει την δυνατότητα σε διαφορετικά είδη συστημάτων (όπως τα Raspberry Pi του συστήματος παρατήρησης της ποιότητας του αέρα και τον υπολογιστή της ομάδας διάσωσης που επιβλέπει την περιοχή της κρίσης) να ανταλλάσσουν μεταξύ τους δεδομένα. Αποτελεί αρχιτεκτονική υπηρεσιών ιστού (web services), δηλαδή ανοικτού προτύπου εφαρμογές πάνω στον ιστό, που αλληλεπιδρούν με άλλες εφαρμογές με σκοπό την ανταλλαγή δεδομένων. Ουσιαστικά, προσφέρει σε διαφορετικά συστήματα υπολογιστών την δυνατότητα ανταλλαγής δεδομένων μέσω του Ιστού. Τα web services που στηρίζονται στην αρχιτεκτονική REST, ονομάζονται RESTful.

Η REST αποτελεί μία από τα πιο δημοφιλή αρχιτεκτονές για την ανάπτυξη web services. Ιδρύθηκε το 2000 από τον Roy Fielding και έκτοτε αποτελεί την πρώτη επιλογή στην ανάπτυξη εφαρμογών ιστού. Με τις web services να εισβάλλουν όλο και περισσότερο στις καθημερινές διεργασίες, σε συνδυασμό με τις εφαρμογές κινητών συσκευών και IoT, η REST αποτελεί την πιο πολυχρησιμοποιούμενη αρχιτεκτονική, καθώς και αυτή που όλοι αναπτύσσουν και βελτιώνουν συνεχώς. Τα εργαλεία για την ανάπτυξη του, μπορούν να βρεθούν σε όλα τα σύγχρονα εργαλεία ανάπτυξης εφαρμογών ιστού. Στις γλώσσες προγραμματισμού που χρησιμοποιούνται κατά κόρον για την ανάπτυξη τέτοιων εφαρμογών, όπως την JavaScript και την Python, χρησιμοποιούνται frameworks. Η δημοφιλία αυτή, οφείλεται στις αρχές/περιορισμοί που διέπουν την αρχιτεκτονική REST:

1. Client-Server
2. Stateless
3. Cache
4. Uniform Interface
5. Layered System
6. Code-On-Demand

Ο περιορισμός «Client-Server» είναι αρκετά ξεκάθαρος. Αυτός ο περιορισμός ορίζει ότι το μοντέλο επικοινωνίας των συσκευών είναι το μοντέλο πελάτη-εξυπηρετητή. Αυτό με την σειρά του διακρίνει τον client και τις απαιτήσεις της διεπαφής του χρήστη με τις απαιτήσεις του αποθηκευτικού χώρου και κατ' αυτόν τον τρόπο παρέχει στην εφαρμογή πρωτίστως την δυνατότητα στην ανεξάρτητη ανάπτυξη και κλιμάκωση των δύο (μια δυνατότητα που είναι αναγκαία στην ανάπτυξη εφαρμογών πάνω από το Internet), καθώς και την δυνατότητα μεταφοράς μόνο της διεπαφής χρήσης σε πολλαπλές πλατφόρμες. Ως επι το πλείστον, οι clients επικοινωνούν με τον server με αιτήσεις και απαντήσεις μέσω του HTTP. Αυτό γιατί, όπως θα δούμε αργότερα, το πρωτόκολλο αυτό καλύπτει όλους τους περιορισμούς της αρχιτεκτονικής REST.

Ο περιορισμός το σύστημα να είναι χωρίς κατάσταση (stateless), σημαίνει ότι κάθε αίτηση HTTP περιέχει όλες τις πληροφορίες που χρειάζεται ο server, ώστε να

εκτελέσει το αίτημα, χωρίς αυτό να χρειάζεται πληροφορία αποθηκευμένη στον server αλλά και χωρίς να χρειάζεται επιπλέον αιτήματα. Αυτό αποδεσμεύει τον server από την δέσμευση πληροφοριών σχετικά με την συνδιάλεξη (session), όπως headers του HTTP. Επιπρόσθετα, διευκολύνει και τις απαντήσεις των client, καθώς ο server παρέχει αρκετές πληροφορίες στην απάντηση HTTP που ο client μπορεί να χρειαστεί. Όπως αναφέρει και ο Feilding, αυτό που προσφέρει αυτός ο περιορισμός είναι ορατότητα (visibility), αξιοπιστία (reliability) και επεκτασιμότητα (scalability). Η ορατότητα βελτιώνεται επειδή η παρακολούθηση του συστήματος αρκείται στην παρατήρηση μιας μεμονωμένης αίτησης για να προσδιορίσει την πλήρη φύση των αιτήσεων. Η αξιοπιστία του συστήματος είναι καλύτερη, επειδή η ανάκαμψη από μερικές αποτυχίες είναι πιο απλή. Η επεκτασιμότητα βελτιώνεται επειδή η μη αποθήκευση κατάστασης μεταξύ των αιτήσεων επιτρέπει στον server ευελιξία και ταχύτητα στην διαχείριση των πόρων και απλοποιεί περαιτέρω την υλοποίηση επειδή ο διακομιστής δεν χρειάζεται να διαχειρίζεται τη χρήση πόρων σε κάθε αίτηση. Επιτρέπει επίσης τη λειτουργία κρυφής αποθήκευσης (caching) της αρχιτεκτονικής REST, καθώς επιτρέπει στον server να αποφασίσει αν θα αποθηκεύσει την αίτηση στην κρυφή μνήμη ή όχι, με την εξέταση μόνο του συγκεκριμένου αιτήματος.

Η χρήση κρυφής μνήμης (Caching) είναι μια τεχνική που επαναχρησιμοποιεί ένα αντίγραφο μιας απάντησης, η οποία είναι πολύ πιθανό να ξαναπροκύψει για ένα παρόμοιο αίτημα. Αποτελεί μεσάζων μεταξύ του client και του server και για αυτό τον λόγο μπορεί να υλοποιηθεί είτε στον server, είτε στον client είτε κάπου ενδιάμεσα, όπως έναν proxy server. Η κρυφή μνήμη προσφέρει σημαντικά πλεονεκτήματα στις εφαρμογές RESTful, όπως είναι η μείωση των αναγνώσεων από την βάση δεδομένων και τον χρόνο επεξεργασίας, η βελτίωση της επεκτασιμότητας, των επιδόσεων και της απόδοσης, μέσω της μείωσης της μέσης καθυστέρησης των αλληλεπιδράσεων, ωστόσο, εάν δεν γίνεται σωστή διαχείριση, ο client θα λαμβάνει αποτελέσματα διαφορετικά από τα αναμενόμενα. Για την προσωρινή αποθήκευση στην κρυφή μνήμη, χρησιμοποιούνται κεφαλίδες HTTP. Συνήθως, όταν χρησιμοποιείται μια ασφαλής μέθοδος σε ένα URL, υπεύθυνος για την κρυφή μνήμη είναι ο αντίστροφος διακομιστής μεσολάβησης (reverse proxy server), ο οποίος αποθηκεύει στην γρήγορη μνήμη του ίδιου τα δεδομένα για μελλοντική χρήση. Θα αναφερόμαστε αργότερα στους reverse proxy servers, όταν θα περιγράψουμε τον Nginx.

Ο τέταρτος περιορισμός της αρχιτεκτονικής REST είναι αυτός που την διακρίνει απ' τις υπόλοιπες. Πιο συγκεκριμένα, ορίζει ότι η επικοινωνία μεταξύ client, server και των ενδιάμεσων θα πρέπει να είναι ομοιόμορφη. Αυτό επιτυγχάνει την απλοποίηση της συνολικής αρχιτεκτονικής, καθώς και την βελτίωση της ορατότητας των ενεργειών που πραγματοποιούνται. Η αρχιτεκτονική REST το πετυχαίνει αυτό χρησιμοποιώντας ένα λογικό σύστημα URI με ομοιόμορφους τρόπους ανάκτησης και χειρισμού δεδομένων. Όπως αναφέρει ο Feilding μια ομοιόμορφη διεπαφή ορίζεται από τέσσερις περιορισμούς: ταυτοποίηση πόρων, χειρισμός πόρων μέσω αναπαραστάσεων, αυτοπεριγραφικά μηνύματα και υπερμέσα ως βάση της κατάστασης της εφαρμογής. Η περαιτέρω ανάλυσή τους ξεφεύγει από τα πλαίσια της παρούσας εργασίας.

Για την κλιμάκωση σε ακόμη μεγαλύτερο επίπεδο, η αρχιτεκτονική REST έχει τον περιορισμό του πολυεπίπεδου συστήματος (Layered System). Ο περιορισμός αυτός ορίζει ότι μεταξύ των client και server θα πρέπει να υπάρχουν επίπεδα, όπου το κάθε επίπεδο θα προσφέρει υπηρεσίες στο επόμενο. Το κάθε επίπεδο γνωρίζει μόνο το γειτονικό επίπεδο με το οποίο αλληλεπιδρά. Αυτό προσφέρει απλότητα στο σύστημα και ανεξαρτησία από τα υπόλοιπα υποστρώματα, το οποίο με την

σειρά του βελτιώνει την εξελιξιμότητα και την επαναχρησιμοποίηση καθώς και ασφάλεια στις υπηρεσίες. Το βασικό μειονέκτημα που παρουσιάζει είναι ο φόρτος και η καθυστέρηση στην επεξεργασία δεδομένων, το οποίο όμως ελλατώνεται με την χρήση κρυφής μνήμης.

Ο τελευταίος περιορισμός προσθέτει την δυνατότητα στον client να λαμβάνει, όχι μόνο τα δεδομένα αλλά και τον κώδικα για την επεξεργασία τους. Αυτός ο κώδικας εκτελείται τοπικά στον client. Τα πλεονεκτήματα που προσφέρει είναι ότι η ανάπτυξη του client είναι ανεξάρτητη του κώδικα της επεξεργασίας των δεδομένων και άρα οποιαδήποτε αλλαγή σε αυτόν δεν θα επηρεάσει τον ήδη υπάρχοντα client, οι δυνατότητες επεκτασιμότητας και διαμορφωσης, η απόδοση και οι επιδόσεις από την οπτική τπυ χρήστη. Βέβαια, η υλοποίηση γίνεται πιο πολύπλοκη και φυσικά η ορατότητα περιορίζεται. Επειδή το API της εφαρμογής είναι Web API, οι clients επικοινωνούν με τον server μέσω του φυλλομετρητή τους, οπότε αυτός ο περιορισμός περισσεύει, καθώς δεν είναι δυνατή η αποστολή και εκτέλεση κώδικα μέσω του φυλλομετρητή.

2.2.2 Flask Web Framework

Για την ανάπτυξη του server χρησιμοποιήθηκε το web framework *Flask*. Ένα web framework είναι ένα σύνολο από εργαλεία λογισμικού που χρησιμοποιούνται στην ανάπτυξη και εκτέλεση Web-based διεπαφών χρήστη - User Interface (UI) (Kourie, 2008). Αυτά τα UI αναπτύσσονται από τους προγραμματιστές και προσφέρονται στους χρήστες, μέσω των φυλλομετρητών τους. Ο κώδικας για τα UI εκτελείται είτε στον server και το αποτέλεσμα παρουσιάζεται στον χρήστη (server-side render) είτε εκτελείται εξ' ολοκλήρου στον φυλλομετρητή του χρήστη (client-side render). Τα web frameworks αναπτύχθηκαν για να προσφέρουν τα εργαλεία για την ανάπτυξη ιστοσελίδων και υπάγονται σε δύο κατηγορίες, ανάλογα με το που χρησιμοποιούνται: σε client -side frameworks, αν χρησιμοποιούνται σε client-side rendered ιστοσελίδες ή σε server-side frameworks, αν χρησιμοποιούνται σε server-side rendered ιστοσελίδες. Το Flask αποτελεί server-side framework, καθώς χρησιμοποιείται για την υλοποίηση ιστοσελίδων που εκτελούνται στον server. Τα web frameworks αυτά προσφέρουν δυνατότητες όπως ασφάλεια, αντιστοίχιση Uniform Resource Locator (URL), σύστημα web template, caching κ.λπ. (Curie, 2019).

Το Flask αποτελεί ένα open source μικρό-framework (micro framework), που παρέχει τις περισσότερες από αυτές τις δυνατότητες (όπως το σύστημα web template και αντιστοίχιση URL), όμως, εξ' αιτίας της δομής του, μπορεί πολύ εύκολα να επεκταθεί και να υλοποιήσει και τις υπόλοιπες, μέσω πρόσθετων (extensions). Τα extensions αυτά είναι open source (όπως αναφέραμε και στο υποκεφάλαιο 2.1.2), που σημαίνει, ότι οποιοσδήποτε επιθυμεί μπορεί να δημιουργήσει δικά του extensions ή να χρησιμοποιήσει προϋπάρχοντα. Με αυτόν τον τρόπο, το Flask παραμένει απλό και αποδοτικό, παρέχοντας την βασική υποστήριξη και παρέχει ταυτόχρονα την δυνατότητα επεκτάσης. Επίσης, το Flask παρέχει την δυνατότητα στον προγραμματιστή να χρησιμοποιήσει στο πρόγραμμα οποιαδήποτε προσέγγιση επιθυμεί, π.χ. χρήση Structured Query Language (SQL) ή NoSQL βάσης δεδομένων, χωρίς αυστηρούς περιορισμούς στην σχεδίαση, όπως χρήση ενός μοντέλου Model-View-Control (MVC). Αυτή η απλότητα σε συνεργασία με την επεκτασιμότητα και ελευθερία στις επιλογές του προγραμματιστή, καθιστούν το Flask την κατάλληλη επιλογή για την ανάπτυξη του server.

```
pip install flask
```

Listing 3: Εγκατάσταση Flask.

Πίνακας 2.5: Εγκατεστημένα Πακέτα Flask

Πακέτο	Περιγραφή
Werkzeug	Web Server Gateway Interface (WSGI) υλοποίηση για Python.
Jinja2	Γλώσσα προτύπων απόδοσης ιστοσελίδων.
MarkupSafe	Πρόσθετο της Jinja για την αποφυγή injection επιθέσεων στην βάση.
ItsDangerous	Εξασφάλιση ακεραιότητας δεδομένων.
Click	Framework για την ανάπτυξη command line εφαρμογών, όπως το flask.
Blinker	Υποστήριξη πακέτων πηρύνα.

Η εγκατάσταση του Flask είναι πολύ απλή. Το Flask γενικά, είναι ένα πακέτο για την Python, το οποίο διατίθεται στο PyPI. Στο κεφάλαιο 2.1.2 αναφέραμε το εργαλείο `pip` και πως μέσω αυτού γίνεται η εγκατάσταση όλων των πακέτων της Python που υπάρχουν στο PyPI. Συνεπώς, η εγκατάστασή του γίνεται με την κλήση της εντολής `pip`, όπως φαίνεται στην εντολή 3. Πάραλληλα με το Flask, γίνεται εγκατάσταση και κάποιων επιπλέον πακέτων, που χρησιμοποιούνται για την ολοκληρωμένη ανάπτυξη του web server. Στον πίνακα 2.5 παραθέτονται αυτά τα πακέτα, καθώς και πληροφορίες σχετικά με τις λειτουργίες τους.

Εκτός από το βασικό framework, το Flask προσφέρει και ένα Command Line Interface (CLI), μέσω του οποίου μπορούν να εκτελεστούν οι εφαρμογές του Flask. Αυτό προσφέρει ευκολία στην εκτέλεση αλλά και ταυτόχρονα την δυνατότητα ανάπτυξης και διαχείρισης πολλαπλών εφαρμογών στο ίδιο πρόγραμμα. Μαζί με το CLI, το Flask παρέχει και έναν server για την ανάπτυξη της εφαρμογής. Ο server αυτός τρέχει συνεχώς τον κώδικα για την εφαρμογή Flask στην τοπική διεύθυνση (localhost) και την προκαθορισμένη θύρα 5000. Ο server είναι διαθέσιμος για την ανάπτυξη της εφαρμογής και δεν ενδείκνυται για την κυκλοφορία της εφαρμογής. Αυτό, γιατί η σχεδιάσή του είναι τέτοια που δεν επιτρέπει την είσοδο περισσότερων του ενός client στα endpoints την φορά, καθώς η είσοδος πολλαπλών client μειώνει κατακόρυφα την απόδοση, την ασφάλεια και την σταθερότητα του. Για την εκτέλεσή του, χρησιμοποιείται η εντολή 4.

```
flask --app <app_name> run --debug
```

Listing 4: Εκτέλεση Εφαρμογής Flask.

Η παραπάνω εντολή εκτελεί τον server για μια εφαρμογή Flask. Το flask υποδηλώνει την κλήση του προγράμματος γραμμής εντολών flask. Η παράμετρος `-app`

με το όρισμα `<app_name>` ορίζει ότι η εφαρμογή που θα εκτελεστεί είναι αυτή με όνομα «`app_name`». Το `run` είναι εντολή η οποία εκκινεί τον Flask server και η παραμέτρος `-debug` ορίζει ότι ο server θα εκτελεστεί σε λειτουργία αποσφαλμάτωσης. Η τελευταία είναι προαιρετική και χρησιμοποιείται, ώστε αλλαγές που συμβαίνουν στον κώδικα της εφαρμογής την ώρα της εκτέλεσής της, να εφαρμόζονται στον server της εφαρμογής, ενώ εκτελείται.

Για την ανάπτυξη των εφαρμογών, το Flask προσφέρει ένα απλό και ολοκληρωμένο API. Ο προγραμματιστής μπορεί να αναπτύξει την εφαρμογή του πολύ εύκολα και σε πολύ λίγες γραμμές. Στον κώδικα 2.4, παρουσιάζεται η ανάπτυξη μιας εφαρμογής, όπου στους clients που συνδέονται εμφανίζεται ένα μήνυμα σε HTML. Με την εντολή `Flask(__name__)` δημιουργείται μια εφαρμογή Flask με παράμετρο ονόματος το όνομα της εφαρμογής. Με τον διακοσμητή (decorator) `route` της εφαρμογής `app`, η συνάρτηση `hello_world` δεσμεύεται να εκτελέσει τις ενέργειες για τον πόρο «`http://<IP>:<Port>/`». Όταν ένας client προσπελαύνει τον πόρο, τότε θα του εμφανίζεται μια σελίδα HTML με το κείμενο «Hello World».

Listing 2.4: Δημιουργία και εκτέλεση εφαρμογής στο Flask.

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

Εκτελώντας την εντολή 4 αυτή την φορά με `app_name` το `app`, το πρόγραμμα θα εκτελέσει τον Flask server για την εφαρμογή `app`, του κώδικα 2.4. Η πρόσβαση σε αυτόν γίνεται από τον φυλλομετρητή και την URL «`http://localhost:5000/`». Αυτή η διεύθυνση όμως δεν είναι ορατή σε κανένα άλλο μηχάνημα, γιατί είναι η τοπική διεύθυνση του υπολογιστή. Αν επιχειρήσουμε την οποιαδήποτε αλλαγή στον κώδικα, τότε η σελίδα θα ανανεωθεί αυτόματα και θα εμφανιστεί το ανανεωμένο περιεχόμενό της.

Το Flask προσφέρει μεγάλη ευκολία όσον αφορά την ανάπτυξη πόρων που χρησιμοποιούν μεθόδους του HTTP. Πιο συγκεκριμένα, όπως φαίνεται στον κώδικα 2.5 η δημιουργία πόρων μιας εφαρμογής καθώς και η ανάθεση λειτουργικότητας σε αυτούς γίνεται με τον decorator `route` της εφαρμογής, όπου δέχεται ως όρισμα εκτός από το όνομα του πόρου και ποιες μεθόδους HTTP μπορεί να εκτελέσει. Μέσω της κλάσης `request` παρέχονται πληροφορίες για το αίτημα του client, ανάμεσά τους και η μέθοδος με την οποία κλήθηκε ο αντίστοιχος πόρος.

Listing 2.5: Δημιουργία πολλαπλών πόρων μαζί με μεθόδους HTTP.

```
from flask import Flask, request

app = Flask(__name__)

@app.route("/", method=["GET"])
def hello_world():
    return "<p>Hello, World!</p>"
```

```
@app.route("/hello", method=["GET", "POST"])
def hello_world():
    if request.method == "GET":
        return "<p>Hello, GET!</p>"
    else:
        return "<p>Hello, POST!</p>"
```

Όπως φαίνεται και στα παραδείγματα, η παραγωγή κώδικα HTML από την Python αποτελεί κακή πρακτική, λόγω της κλιμάκωσης του κώδικα και της δυσκολίας ανάγνωσης του αλλά και ανάπτυξης του κώδικα της HTML. Πράγματι, ο κώδικας για μια τύπικης ιστοσελίδας είναι πολύ μεγαλύτερος σε μέγεθος από μερικές γραμμές. Γι' αυτό τον λόγο το Flask έχει συμπεριλάβει το πακέτο Jinja2, το οποίο επιτρέπει την δημιουργία κώδικα Python μέσα σένα αρχείο HTML. Με αυτόν τον τρόπο, η ανάπτυξη λειτουργικού κώδικα σε HTML μπορεί να γίνει απευθείας στο αρχείο. Για την εκτέλεση του κώδικα και την φόρτωση της σελίδας, το Flask τροποποιεί κατάλληλα το Jinja2 αυτόματα, κατά την διάρκεια εκτέλεσης του κώδικα, μέσω της συνάρτησης `render_template`.

Listing 2.6: Πόρος που φορτώνει ιστοσελίδα από αρχείο HTML.

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/", method=["GET"])
def hello_world():
    return render_template("<html_filename>.html"<, parameter=value, ...>)
```

2.2.3 Βάση Δεδομένων Mongo

Όπως αναφέρθηκε στην αρχή του κεφαλαίου, ο server χρειάζεται να αποθηκεύει κάπου τις παρατηρήσεις των αισθητήρων που λαμβάνει. Επειδή τα δεδομένα είναι οργανωμένα σε μια δομή (δομή JSON), όπως είδαμε στο κεφάλαιο 2.1.2, είναι αναγκαίο να χρησιμοποιηθεί ένα σύστημα βάσης δεδομένων για την αποδοτική και εύκολη αποθήκευση και διαχείριση της πληροφορίας των αισθητήρων. Ο σκοπός αυτού του συστήματος είναι η αποθήκευση, προστασία και επεξεργασία των δεδομένων, παρέχοντας εξελιγμένους μηχανισμούς πρόσβασης, δικαιωμάτων των χρηστών και ενημέρωσης των δεδομένων (Παπαδόπουλος, 2020). Ένα σύστημα βάσεων δεδομένων αποτελείται από μία ή περισσότερες βάσεις δεδομένων, όπου εκεί καταχωρούνται τα δομημένα δεδομένα που σχετίζονται μεταξύ τους, και το σύστημα διαχείρισης βάσεων δεδομένων Database Management System (DBMS), δηλαδή το λογισμικό που καλύπτει όλες τις απαραίτητες λειτουργίες για την αποθήκευση, ανάκτηση, προστασία, αναζήτηση κ.λπ. των δεδομένων. Γίνεται φανερό, ότι το DBMS είναι υπεύθυνο για την διεπαφή με τον χρήστη και τον έλεγχο και την διαχείριση των δεδομένων. Για τον λόγο αυτό, πλέον θα αναφερόμαστε σε αυτό όταν θέλουμε να περιγράψουμε το σύστημα βάσεων δεδομένων.

Γενικά, ένα DBMS προσφέρει πληθώρα δυνατοτήτων για την αποθήκευση και διαχείριση οργανωμένων δεδομένων. Κατ' αρχάς, προσφέρουν την δυνατότητα

περιγραφής των δεδομένων, δηλαδή παροχής πληροφοριών για την δομή και οργάνωσή τους και άρα την δυνατότητα τροποποίησης της δομής τους. Εκτός αυτού, τα δεδομένα δεν σχετίζονται με τις λειτουργίες του DBMS, που σημαίνει ότι οποιαδήποτε αλλαγή στην δομή του ενός δεν θα επηρεάσει την δομή του άλλου αλλά και ότι οι λειτουργίες είναι ανεξάρτητες των προγραμματιστικών εργαλείων (π.χ. γλωσσών προγραμματισμού). Επίσης, προσφέρουν αποδοτικότερη διαχείριση των δεδομένων, προστασία και δικαιώματα χρήστη, δυνατότητα ταυτόχρονης προσπέλασης των βάσεων δεδομένων, καθώς και την δυνατότητα επεκτασιμότητας των λειτουργιών του (Παπαδόπουλος, 2020).

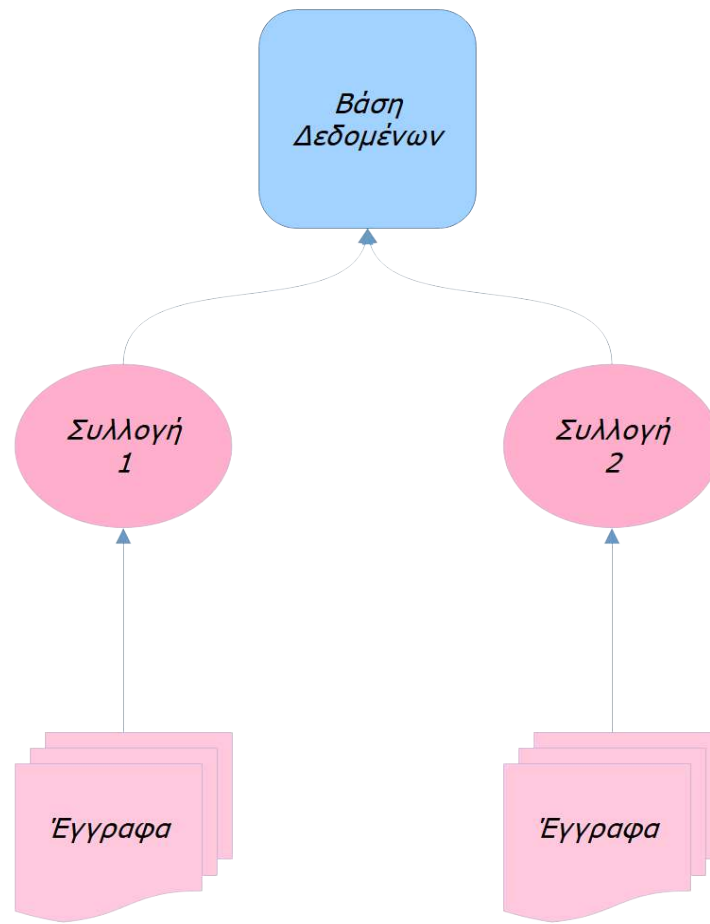
Listing 2.7: Σύνδεση σε *MongoClient*.

```
from pymongo import MongoClient

client = MongoClient("<host_address>", <port_number>)
```

Οι παραπάνω απαιτήσεις ικανοποιούνται επαρκώς από σχεσιακά DBMS, όμως, παρά τα γενικά οφέλη που προσφέρουν, οι ανάγκες της εφαρμογής πάνε ένα ακόμα βήμα παραπέρα τα κριτήρια επιλογής. Συγκεκριμένα, το πλήθος των ερωτημάτων που υποβάλλονται για τον δείκτη ποιότητας αέρα σε ένα τέτοιο σύστημα είναι μεγάλο, αν σκεφτούμε τις υπηρεσίες (πυροσβεστική, ομάδα διάσωσης κ.λπ.) και πιθανούς πολίτες που μπορεί να βρίσκονται εντός της ζώνης κινδύνου, καθώς και ότι μπορεί να υπάρχει κατοικήσιμη περιοχή κοντά, μέρος της οποίας να πρέπει να εκκενωθεί. Επίσης, καθώς τα δεδομένα που συγκεντρώνονται δεν έχουν σχήμα, αφού είναι απλώς μια συλλογή JSON, είναι απαραίτητη η υποστήριξη ευέλικτου σχήματος βάσης δεδομένων. Τέλος, επειδή η κατάσταση τέτοιων φαινομένων είναι κρίσιμη, χρειάζεται μεγάλη ανοχή σε σφάλματα. Για τον λόγο αυτό, το DBMS που επιλέξαμε είναι η MongoDB.

Η MongoDB συγκαταλέγεται στις *Not only SQL (NoSQL) DBMS*. Τα συστήματα αυτά χαρακτηρίζονται από εκτεταμένα αντίγραφα και διαμερισμό των δεδομένων σε (γεωγραφικά) κατανεμημένους κόμβους, το μοντέλο της τελικής συνέπειας των αντιγράφων και ευέλικτη κατάσταση του συστήματος (Cielen, 2016). Υπάρχουν τέσσερις τύποι βάσεων δεδομένων NoSQL: ζεύγος κλειδί-τιμή, προσανατολισμένα σε στύλες, εγγράφων και γράφων. Η MongoDB είναι τύπου εγγράφων. Σε αυτόν τον τύπο, τα δεδομένα αναπαριστώνται ως έγγραφο, όπως Extensible Markup Language (XML) ή JSON. Στα έγγραφα αυτά, τα δεδομένα μπορούν να αποθηκεύονται με ευέλικτο σχήμα, δηλαδή το κάθε έγγραφο μπορεί να περιέχει διαφορετικό πλήθος και είδος χαρακτηριστικών, αρκεί τα δεδομένα να είναι αρκετά δομημένα. Όταν θα αναφερόμαστε στα δεδομένα, θα αναφερόμαστε σε αυτά ως έγγραφα. Τα έγγραφα οργανώνονται σε συλλογές, οι οποίες οργανώνονται στην βάση. Σχηματικά, αυτό αναπαρίσταται στο σχήμα 2.7.



Σχήμα 2.7: Μοντέλο αποθήκευσης Εγγράφων στην MongoDB.

Η MongoDB μπορεί να ενσωματωθεί σε έναν κώδικα Python, μέσω του πακέτου PyMongo. Η εγκατάστασή του φαίνεται στον κώδικα 5. Το πακέτο αυτό είναι ειδικά σχεδιασμένο, ώστε ένα πρόγραμμα γραμμένο σε Python, να μπορεί να επικοινωνεί με μια βάση MongoDB. Για να συνδεθεί το πρόγραμμα στο DBMS, χρειάζεται να δημιουργήσει ένα αντικείμενο υπόστασης της κλάσης MongoClient. Το αντικείμενο αυτό αρχικοποιείται με τις παραμέτρους host και port, με προεπιλεγμένες τιμές 'localhost' και 27017. Αν το DBMS τρέχει τοπικά, τότε οι προεπιλεγμένες τιμές είναι και οι αντίστοιχες τιμές της βάσης. Αν το DBMS τρέχει σε απομακρυσμένο μηχάνημα, τότε θα πρέπει να περαστούν η διεύθυνση του host και η θύρα στην οποία τρέχει. Ο κώδικας 2.7 αναπαριστά την εγκαθίδρυση μιας σύνδεσης με μια βάση MongoDB μέσα σε κώδικα Python.

```
python3 -m pip install pymongo
```

Listing 5: Εγκατάσταση Πακέτου PyMongo.

Πίνακας 2.6: Μέθοδοι CRUD στην βάση.

Μέθοδος	Ενέργεια
insert_one	Εισάγει ένα νέο έγγραφο στην συλλογή.
insert_many	Εισάγει μία λίστα από έγγραφα στην συλλογή.
update_one	Ενημερώνει ένα έγγραφο στην λίστα, το οποίο ταιριάζει σε ένα φίλτρο επιλογής.
update_many	Ενημερώνει ένα ή περισσότερα έγγραφα, τα οποία ταιριάζουν στο φίλτρο επιλογής.
delete_one	Διαγράφει ένα έγγραφο, το οποίο ταιριάζει σε ένα φίλτρο επιλογής.
delete_many	Διαγράφει ένα ή περισσότερα έγγραφα, τα οποία ταιριάζουν σε ένα φίλτρο επιλογής.
find_one	Επιστρέφει ένα έγγραφο, το οποίο ταιριάζει σε ένα φίλτρο επιλογής.
find	Επιστρέφει ένα ή περισσότερα έγγραφα, τα οποία ταιριάζουν σε ένα φίλτρο επιλογής.

Μέσα στον κώδικα, η PyMongo προσφέρει μια απλή διεπαφή για την πρόσβαση σε μία βάση και κατ' επέκταση στις συλλογές και τα έγγραφά της. Για να διαβάσει κανείς μια βάση, χρειάζεται από το αντικείμενο MongoClient πρόσβαση με στυλ ιδιότητας, δηλαδή κάθε βάση αποτελεί ιδιότητα του αντικειμένου MongoClient. Η πρόσβαση με στυλ ιδιότητας στο αντικείμενο MongoClient επιστρέφει το αντικείμενο της βάσης. Αντίστοιχα, για την προσπέλαση μιας συλλογής μίας βάσης, η διαδικασία είναι η ίδια, αυτή την φορά όμως από το αντικείμενο της βάσης. Η λογική αυτή είναι και η ίδια λογική για την δημιουργία μιας νέας βάσης ή συλλογής. Όμως, για να αποθηκευτεί η νέα δομή (βάση ή συλλογή) θα πρέπει να καταχωρηθεί και ένα νέο έγγραφο στην νέα συλλογή. Για την εισαγωγή νέων εγγράφων, ενημέρωση υπάρχοντων, διαγραφή και αναζήτηση, η PyMongo διαθέτει ένα σύνολο απλών μεθόδων. Στον πίνακα 2.6 παραθέτονται οι βασικότερες μέθοδοι CRUD σε μία συλλογή.

2.2.4 Δομή Web Server

Κάθε web εφαρμογή χρειάζεται να εκτελείται συνεχώς, ώστε να δέχεται και να απαντάει σε δυνητικές αιτήσεις πελατών. Αυτό γίνεται με την χρήση ενός web server. Ένας web server αποτελείται από το υλικό και το λογισμικό (Mozilla). Το υλικό αποτελεί τον χώρο όπου αποθηκεύονται τα αρχεία (HTML, CSS, JSON, XML, JavaScript αρχεία, εικόνες και έγγραφα) και υποστηρίζει την φυσική μεταφορά των

αρχείων στο Internet. Το λογισμικό είναι υπεύθυνο για την διαχείριση αιτήσεων και εύρεση και επιστροφή των κατάλληλων αρχείων, αν ο server είναι στατικός, ή την εκτέλεση κώδικα, αν είναι δυναμικός. Για την επικοινωνία του με τους client χρησιμοποιεί το HTTP και για αυτό και μερικές φορές ονομάζονται και HTTP servers. Οι στατικοί servers είναι αρκετά απλοί, γιατί δεν χρειάζεται να κάνουν τίποτα άλλο παρά μόνο να δέχονται αιτήματα, να ψάχνουν αρχεία και να επιστρέφουν την κατάλληλη απάντηση. Οι δυναμικοί servers είναι πιο σύνθετοι, καθώς είναι υπεύθυνοι για την εκτέλεση εφαρμογών από μέρους τους και την διατήρηση δεδομένων, που σχετίζονται με αυτές. Αυτές οι εφαρμογές αποτελούν τον λεγόμενο *application server*, ενώ για την διαχείριση των δεδομένων διατηρούν μία *βάση δεδομένων*.

Gunicorn

Η εκτέλεση web εφαρμογών από τους web servers εξαρτάται από την σχεδίαση των τελευταίων. Αυτό, όμως, έχει θέσει ζήτημα διασύνδεσης μεταξύ web server και application server, όσον αφορά το πως ο τελευταίος θα ενημερώνει τον πρώτο για την εκτέλεση προγραμμάτων ή ανάκτηση αρχείων. Λύση σε αυτό αποτελούν API διεπαφές ειδικά διαμορφωμένες, οι οποίες προσφέρουν την δυνατότητα στον προγραμματιστή να καθορίζει πως η εφαρμογή αλληλεπιδρά με τον web server. Ως παράδειγμα, αναφέρουμε την πρώτη διεπαφή που αναπτύχθηκε, την *Κοινή Διασύνδεση Πύλης Δικτύου* ή *Common Gateway Interface (CGI)*. Συνοπτικά, αυτή η διεπαφή προσφέρει στους web servers την δυνατότητα επικοινωνίας με back-end προγράμματα και σενάρια που δέχονται εισόδους και παράγουν HTML ως απαντήσεις και αυτό προσφέρει την δυνατότητα ο κώδικας των back-end προγραμμάτων να είναι γραμμένος σε διαφορετική γλώσσα από την γλώσσα που είναι γραμμένος ο web server. Στον πίνακα 2.7 παραθέτονται μερικές από τις πιο δημοφιλείς διεπαφές. Ομοίως με την CGI, και αυτές προσφέρουν την δυνατότητα επικοινωνίας με back-end προγράμματα και σενάρια, μόνο που η αρχιτεκτονικές τους διαφέρουν σημαντικά και λύνουν προβλήματα που υφίσταται στην CGI, για αυτό και τείνουν να την αντικαταστήσουν. Παρ' όλα αυτά, η ανάλυση αυτών των διεπαφών βρίσκεται εκτός του πλαισίου της εργασίας αυτής και για αυτό δεν πραγματοποιείται.

Πίνακας 2.7: Διεπαφές μεταξύ web server και εφαρμογής.

Διεπαφή
CGI
mod_python
FastCGI
SCGI
mod_wsgi

Αν και η CGI αποτελεί την πιο ευρέως χρησιμοποιημένη διεπαφή σε web servers, εντούτοις η ανάπτυξη back-end προγραμμάτων σε python και η εκτέλεσή τους από web server με την χρήση αυτής αλλά και των υπολοίπων θεωρείται παρωχημένο (PEP 333). Και αυτό, γιατί το ίδιο PEP υποστηρίζει ότι η επιλογή μεταξύ κατάλληλου framework και web server περιορίζεται από την διεπαφή που υποστηρίζει το εκάστοτε framework και web servers. Γι' αυτό προτείνει μια απλή, τυποποιημένη διεπαφή, η οποία υλοποιείται πάνω στις προϋπάρχουσες διεπαφές του web server και ενσωματώνεται στα frameworks, την *Python Web Server Gateway Interface (WSGI)*.

Υπάρχουν πολλοί web servers οι οποίοι υποστηρίζουν την WSGI. Για την συγκεκριμένη εργασία, χρησιμοποιήθηκε ο *Gunicorn*. Ο συγκεκριμένος web server προσφέρει πολλά πλεονεκτήματα: είναι αρκετά απλός στην παραμετροποίηση και προσφέρει πολλαπλούς «εργάτες (workers)» για την διαρύθμιση της αποδόσης. Εκτός αυτού, προσφέρει ευκολία στην εγκατάσταση, καθώς δεν έχει επιπλέον παραμέτρους ή μεταγλώττιση, και ευκολία στην ενσωμάτωση στην πλατφόρμα που φιλοξενεί τον server. Επίσης παρέχει υποστήριξη ασύγχρονων worker, με την χρήση του πακέτου *gevent*. Το μειονέκτημά του είναι ότι δεν τρέχει σε λειτουργικό σύστημα Windows, όμως αυτό δεν αποτελεί πρόβλημα, καθώς ο server εκτελείται σε Ubuntu VPS. Αν και υπάρχουν πολύ πιο αποδοτικοί web servers (uWSGI), εντούτοις είναι πολύ πιο περίπλοκοι στην παραμετροποίηση και έχουν περισσότερες απαιτήσεις (όπως την ανάγκη χρήσης μεταγλωττιστή).

Nginx

Όπως αναφέρθηκε και προηγουμένως, ο web server που χρησιμοποιείται είναι ο *Gunicorn*. Όμως, για τις απαιτήσεις μια τέτοιας εφαρμογής, δεν αρκεί ένας απλός web server, ο οποίος μπορεί να διαχειρίζεται καλά αρκετά αιτήματα. Χρειάζεται ένας web server αρκετά ισχυρός, ώστε να χειρίζεται προβλήματα που μπορεί να προκύψουν, όπως την συμφόρηση δικτύου, ισοζύγιο φόρτου κ.λπ., καθώς και να μπορεί να κλιμακώνεται εύκολα και με πολλές εφαρμογές. Αυτό, λειτουργεί και ως μέτρο ασφαλείας σε επιθέσεις distributed denial-of-service (DDOS). Ο συγκεκριμένος server (όπως και οι υπόλοιποι servers που προσφέρουν WSGI) δεν είναι ικανός να διαχειριστεί αυτά τα ζητήματα, λόγω του ότι επικεντρώνεται στο να λύσει αυτό το ζήτημα. Γι' αυτό χρειάζεται ένας web server που να μεσολαβεί μεταξύ WSGI server και clients.

Υπάρχουν διάφοροι HTTP servers που προσφέρουν αυτές τις επιπλέον υπηρεσίες, με διαφορετικές αρχιτεκτονικές. Ο HTTP server που επιλέξαμε να χρησιμοποιήσουμε είναι ο *Nginx (Engine-X)*. Ο συγκεκριμένος αποτελεί τον πιο χρησιμοποιούμενο πλέον web server για όλους τους τύπους σελίδων. Αυτό γιατί αποτελεί server ανοικτού κώδικα υψηλής απόδοσης (high performance), με πληθώρα άλλων δυνατοτήτων, όπως reverse proxying, διαμοιρασμό φόρτου, υποστήριξη SSL/TLS (Secure Sockets Layer και Transport Layer Security), κρυφή μνήμη κ.λπ. Ο Nginx είναι ασφαλής, σταθερός και πολύ εύκολος στην διαχείριση, καθώς επίσης και πολύ πιο αποδοτικός σε σχέση με άλλους γνωστούς web servers, όπως ο Apache (Reese, 2008). Από τις επιπλέον δυνατότητες που προσφέρει ο Nginx, το ενδιαφέρον μας στρέφεται στο reverse proxying. Συγκεκριμένα, με αυτή την δυνατότητα, ο Nginx μπορεί να επικοινωνεί με άλλον web server χρησιμοποιώντας το HTTP, χωρίς να χρειάζεται κάποια επιπλέον διεπαφή μεταξύ τους, με τον Nginx να λειτουργεί ως client, και τον άλλο web server, ο οποίος επικοινωνεί με τον application server, να θεωρεί πως λαμβάνει αίτημα HTTP από κάποιον client, μην γνωρίζοντας την ύπαρξη του Nginx.

2.3 Πλήρες Σύστημα Αισθητήρων

Για την ευρύτερη υλοποίηση τέτοιων συστημάτων, υπάρχουν εταιρίες που διαθέτουν ολοκληρωμένες λύσεις IoT. Οι εταιρίες αυτές παρέχουν λογισμικό, υπηρεσίες και εξοπλισμό, ειδικά διαμορφωμένο σύμφωνα με τις ανάγκες των πεδίων IoT

που καλύπτουν, όπως τις έξυπνες πόλεις, την γεωργία και την κτηνοτροφία, την βιομηχανία, το περιβάλλον, κ.λπ. Ένα τέτοιο σύστημα χρησιμοποιήθηκε για να συγκριθεί με το σύστημα που υλοποιήθηκε στην παρούσα εργασία. Το σύστημα αυτό είναι προϊόν της εταιρίας Libelium, η οποία προσφέρει έτοιμες IoT λύσεις, σε θέματα που επιδέχονται προσέγγιση IoT, όπως λύσεις στην στάθμευση σε έξυπνες πόλεις, επίβλεψη συνθηκών ανάπτυξης φυτών στην γεωργία, επίβλεψης ποιότητας αέρα σε πόλεις και φυσικά περιβάλλοντα κ.λπ.

2.3.1 Σύστημα Αισθητήρων Libelium

Το μοντέλο αισθητήρων της Libelium που χρησιμοποιήθηκε, είναι το Smart Spot. Το συγκεκριμένο μοντέλο είναι κατάλληλο για την παρατήρηση διάφορων περιβαλλοντικών παραγόντων, όπως είναι η ποιότητα του αέρα, η θερμοκρασία, η υγρασία, ο θόρυβος και επίσης χρησιμοποιείται και ως καιρικός σταθμός. Είναι ολοκληρωμένο και περιλαμβάνει, πέραν των αισθητήρων που απαιτούνται για τα παραπάνω, hardware και software, που επιτρέπουν την συλλογή και αποστολή των μετρήσεων, χωρίς κάποια επιπλέον αγορά ή προηγούμενη εγκατάσταση λογισμικού από τον χρήστη. Χάρη σ' αυτό, προσφέρει ευκολία στην εγκατάσταση, την διαχείριση, την παραμετροποίηση και την επικοινωνία. Το σύστημα, μπορεί να βρει εφαρμογή και εξ' αποστάσεως, λόγω των δυνατοτήτων τροφοδοσίας (σύνδεση με φωτοβολταϊκά ή με μπαταρίες υψηλής χωρητικότητας). Ο πίνακας 2.8 δείχνει τα σημαντικότερα στοιχεία που περιέχει το συγκεκριμένο σύστημα.

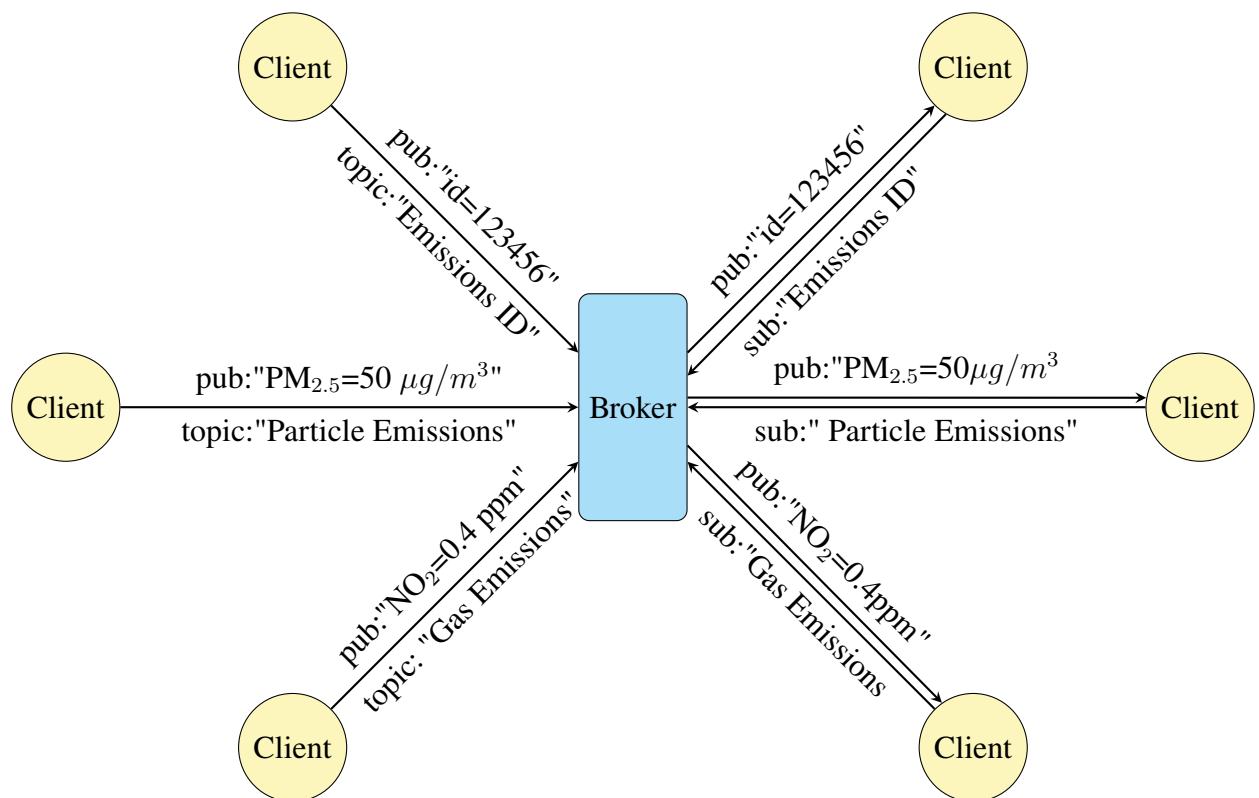
Εκτός από τα πολλαπλά οφέλη που παρέχουν, οι αισθητήρες αυτοί είναι πιστοποιημένοι από την Ευρωπαϊκή Ένωση. Συγκεκριμένα, η *εντολή 2008/50/EC για την περιβαλλοντική ποιότητα αέρα και καθαρότερο αέρα στην Ευρώπη* ορίζει τα κριτήρια για την ποιότητα του περιβαλλοντικού αέρα. Πάνω σε αυτή την εντολή, έρχεται το πρότυπο *CEN/TS 17660-1:2021*, το οποίο ορίζει τα κριτήρια για την αξιολόγηση της απόδοσης των συστημάτων αισθητήρων. Με βάση αυτό εγκρίθηκε, κατηγοροποιήθηκε και ταξινομήθηκε στην κλάση 1.

Πίνακας 2.8: Βασικά Χαρακτηριστικά Αισθητήρων Smart Spot.

Χαρακτηριστικό	
Λειτουργικό Σύστημα	FreeRTOS
CPU	Dual Core 240MHz
RAM	16MB
Δίκτυο	WiFi, NB-IoT
Απομακρυσμένη Διαχείριση	Homard platofrm
Κατανάλωση Ενέργειας	180 – 300 mA Active
Τάση	5 V
Μέγεθος	300x220x36, 7 cm
Θερμοκρασία Λειτουργίας	[−30, 60] °C
Βάρος	1, 8 kg
Αέρια	NO ₂ , H ₂ S, CO, NO, SO ₂ , O ₃ , NH ₃ , CO ₂
Σωματίδια	PM _{1.0} , PM _{2.5} , PM _{10.0}
Παράμετροι Αέρα	Θερμοκρασία, Υγρασία, Πίεση

2.3.2 MQTT Broker

Τα σύγχρονα συστήματα IoT βρίσκουν εφαρμογή σε διάφορα μέρη. Συγκεκριμένα, με την άνοδο των έξυπνων πόλεων και της επιτήρησης τομέων, όπως το περιβάλλον, η γεωργία και η κτηνοτροφία, η βιομηχανία κ.λπ. η ανάγκη για έγκυρα και ακριβή συστήματα αυξάνεται. Οι συνθήκες όμως, κάτω από τις οποίες λειτουργούν τα συστήματα αυτά διαφέρουν σημαντικά. Για παράδειγμα, στις έξυπνες πόλεις, όπου υπάρχουν εκατομύρια ή δισεκατομύρια ηλεκτρονικές συσκευές που συνδέονται στο Internet ή άλλα δίκτυα και οι οποίες λειτουργούν για διαφορετικούς και πολλαπλούς σκοπούς, χρειάζεται διαχείριση της πληροφορίας σε μεγάλα δίκτυα και αποφυγή προβλημάτων, όπως καθυστερήσεις, άνισο μοιρασμό φόρτου κ.λπ. Από την άλλη, σε ένα μέρος, όπως ένας βιότοπος, όπου υπάρχουν ελάχιστες συσκευές αλλά αυτές βρίσκονται σε πολύ μακρινή απόσταση από τις συσκευές με τις οποίες επικοινωνούν, χρειάζονται διαχείριση της πληροφορίας σε μακρινά δίκτυα, όπου αυτή δεν θα εξασθενεί ή θα χάνεται. Με βάση τους διάφορους περιορισμούς που τίθενται κάθε φορά, έχουν αναπτυχθεί πολλά και διαφορετικά πρωτόκολλα επικοινωνίας, υπεύθυνα για τα αντίστοιχα σενάρια. Ένα από αυτά τα πρωτόκολλα είναι και το MQTT.



Σχήμα 2.8: Αναπαράσταση πρωτοκόλλου MQTT.

Το πρωτόκολλο *Message Queueing Telemetry Transport (MQTT)* συνδέει συσκευές και δίκτυα με εφαρμογές με middleware και εφαρμογές (Yassein, 2017). Προσφέρει την δυνατότητα σε πολλαπλές συσκευές (πελάτες) να συνδέονται σε μια κεντροποιημένη συσκευή (broker) η οποία είναι υπεύθυνη για τον διαμοιρασμό των μηνυμάτων μεταξύ των πελατών. Στηρίζεται στο μοντέλο publish-subscribe, δηλαδή ο publisher (κάποιος από τους clients) δημοσιεύει μηνύματα σε topics και οι πελάτες που εγγράφονται στα topics αυτά (subscribers) μπορούν να διαβάσουν αυτά

τα μηνύματα. Αυτή η αρχιτεκτονική επιτρέπει την επικοινωνία πολλά-προς-πολλά, καθώς και την επικοινωνία μεταξύ μηχανής-σε-μηχανή, μηχανής-σε-διακομιστή και διακομιστή-σε-διακομιστή. Η μετάδοση στηρίζεται στο TCP/IP αλλά μπορεί να επεκταθεί και στο TLS/SSL. Το σχήμα 2.8 αναπαριστά την δομή του πρωτοκόλλου MQTT.

Υπάρχουν πολλές υλοποιήσεις, όσον αφορά τόσο τον client, όσο και τον broker. Η πιο συνηθισμένη είναι η *Mosquitto*. Συνολικά, ένα σύστημα Mosquitto απαρτίζεται από τρία μέρη: τον mosquitto broker, τις λειτουργίες publish και subscribe για τους πελάτες και την μια βιβλιοθήκη για την υλοποίηση του πελάτη. Η φιλοσοφία του είναι η χρήση σε ελαφριά συστήματα μηνυμάτων, με συσκευές με περιορισμένους πόρους. Υποστηρίζει το νεότερο πρωτόκολλο MQTT (MQTT v3.1), είναι ανοιχτού λογισμικού, προσφέρει δυναμικά topics, υποστηρίζει web sockets, bridges και προσφέρει ασφάλεια μέσω αυθεντικοποίησης. Βέβαια, δεν επιτρέπει την δυνατότητα δημιουργίας cluster, αλλά αυτό δεν επηρεάζει την υλοποίηση.

2.4 Εργαλεία και Τεχνολογίες Ανάπτυξης Εφαρμογής

Η ανάπτυξη της εφαρμογής έγινε με την βοήθεια εργαλείων, στοχευμένα για την υλοποίηση τέτοιων εφαρμογών. Το καθένα προσφέρει διάφορες υπηρεσίες και λειτουργίες, για ανάπτυξη, ασφάλεια, διαχείριση και έλεγχο της αναπτυσσόμενης εφαρμογής. Η χρήση τους είναι πολύ απλή και εύκολη στην κατανόηση, οπότε ο χρήστης γρήγορα εξοικειώνεται και μπορεί να αναπτύξει μεγάλης διάστασης περιβάλλοντα. Ένα μεγάλο πλεονέκτημα που προσφέρουν είναι ότι είναι ανοιχτού λογισμικού, ενώ σε πολλά από αυτά υπάρχει ακόμη μια τεράστια κοινότητα, η οποία συντηρεί και ενημερώνει τις εφαρμογές. Οι λόγοι αυτοί, καθιστούν τα παρακάτω εργαλεία εξαιρετικά σημαντικά για την ορθή ανάπτυξη της εφαρμογής.

2.4.1 Docker

Η πρώτη τεχνολογία πάνω στην οποία βασίστηκε η ανάπτυξη του web server είναι το Docker. Συγκεκριμένα, το Docker αποτελεί μια πλατφόρμα ανοιχτού λογισμικού, η οποία προσφέρει την δυνατότητα δημιουργίας απομονωμένων από το σύστημα περιβαλλόντων εκτέλεσης εφαρμογών. Τα περιβάλλοντα αυτά ονομάζονται «containers». Τα containers προσφέρουν σημαντικά πλεονεκτήματα για την ανάπτυξη και λειτουργία μιας εφαρμογής, όπως είναι η ευκολία στη μεταφορά και την αναπαραγωγή, καθώς και η εκτέλεση εφαρμογών σε διαφορετικές πλατφόρμες (Windows, Linux, MacOS κ.λπ.). Για την δημιουργία ενός container χρειάζεται να δημιουργηθεί μια «εικόνα». Οι εικόνες είναι πρότυπα αρχείων για την δημιουργία container. Μια εικόνα περιλαμβάνει όλες τις παραμέτρους για την εκτέλεση μιας εφαρμογής, όπως τον πηγαίο κώδικα της, τις βιβλιοθήκες που χρειάζεται, το λειτουργικό σύστημα κ.α. Το παραγόμενο εκτελέσιμο προϊόν μιας εικόνας είναι το container.

Όπως αναφέρθηκε και πριν, το Docker προσφέρει πολλά πλεονεκτήματα. Συγκεκριμένα, το βασικότερο πλεονεκτήματα είναι η απομόνωση. Η εκτέλεση της εφαρμογής γίνεται σε ένα απομονωμένο περιβάλλον container, χωρίς αυτό να έχει γνώση του λειτουργικού συστήματος και των αρχείων του. Αυτό πρακτικά, σημαίνει ότι οι εφαρμογές που εκτελούνται σε διαφορετικά containers μπορούν να λειτουργούν ανεξάρτητα, χωρίς να έρχονται σε επαφή με τα περιβάλλοντα των άλλων

εφαρμογών. Με αυτόν τον τρόπο καθίσταται δυνατή η εκτέλεση πολλαπλών εφαρμογών στον ίδιο διακομιστή. Δεδομένων των παραπάνω, προκύπτει ένα επιπλέον σημαντικό πλεονέκτημα: η φορητότητα. Όπως αναφέρθηκε και προηγουμένως, οι εικόνες του Docker αποτελούν στην ουσία ένα μορφότυπο για το container και δεν είναι τίποτα άλλο παρά ένα αρχείο με όλη αυτή την πληροφορία. Εφόσον δεν υπάρχει πρόβλημα με το περιβάλλον εκτέλεσης των container, αυτό σημαίνει ότι μπορούν να τρέξουν σε οποιοδήποτε σύστημα υποστηρίζει το Docker. Έτσι, ο χρήστης μπορεί να μεταφέρει το container σε οποιοδήποτε σύστημα υποστηρίζει το Docker, χωρίς να ανησυχεί για την συμβατότητα της εφαρμογής ή των απαιτήσεών της με το περιβάλλον εκτέλεσης.

Εκτός αυτών των σημαντικών πλεονεκτημάτων, το Docker προσφέρει κάποια επιπλέον χρήσιμα χαρακτηριστικά, που διευκολύνουν την ανάπτυξη εικόνων και container. Συγκεκριμένα, το πρώτο χαρακτηριστικό είναι η ευκολία στην κατανομή και την αναπαραγωγή εφαρμογών. Προηγουμένως, αναφέρθηκε ότι οι εικόνες είναι απλά αρχεία, τα οποία περιέχουν τις απαιτήσεις ενός container. Αυτά τα αρχεία μπορούν να διαμοιραστούν μεταξύ προγραμματιστών ή να αναρτηθούν σε αποθετήρια. Αυτό διευκολύνει τη συνεργασία μεταξύ προγραμματιστών, καθώς επίσης διευκολύνει την γρήγορη και αποτελεσματική ανάπτυξη των εφαρμογών, αφού δεν χρειάζεται να δημιουργείται νέα εικόνα για εφαρμογές με ίδιες ή παρόμοιες απαιτήσεις. Εκτός από την ευκολία στην κατανομή και αναπαραγωγή εφαρμογών, ένα σημαντικό χαρακτηριστικό του Docker είναι η δυνατότητα αυτοματοποίησης των container και των εικόνων. Για την δημιουργία εικόνων, χρησιμοποιούνται αρχεία διαμόρφωσης, όπως το Dockerfile, τα οποία επιτρέπουν τον πλήρη καθορισμό μιας εικόνας, καθώς και των εντολών που θα εκτελεστούν κατά την δημιουργία και εκτέλεση του container. Αυτό προσδίδει στις διαδικασίες εγκατάστασης και ανάπτυξης της εφαρμογής τις ιδιότητες της επαναληψιμότητας και της αυτοματοποίησης αυτών. Τέλος, η κοινότητα που υποστηρίζει το Docker είναι ενεργή, καθώς συνεχώς νέες δυνατότητες υποστηρίζονται και γίνεται συντήριση της πλατφόρμας. Μέσω της κοινότητας προσφέρεται πληθώρα πληθώρα εργαλείων, βιβλιοθηκών και πρότυπων, που μπορούν να βοηθήσουν στην ανάπτυξη και τη διαχείριση εφαρμογών.

2.4.2 Postman

Για τον έλεγχο διαπίστωσης ορθής λειτουργίας των endpoints χρησιμοποιήθηκε το λογισμικό Postman. Το Postman είναι από τα πιο δημοφιλή εργαλεία που χρησιμοποιούνται στην ανάπτυξη λογισμικού και δημιουργία διεπαφών προγραμματισμού εφαρμογών - Application Program Interface (API). Αποτελεί ένα ευέλικτο περιβάλλον που παρέχει δυνατότητες ανάπτυξης, ελέγχου, και τεκμηρίωσης API. Στην ουσία, το Postman λειτουργεί ως ένας HTTP-client, μέσω του οποίου οι προγραμματιστές δύναται να πραγματοποιούν αιτήσεις (requests) σε μια υπηρεσία API και να λάβουν απαντήσεις (responses) από αυτήν. Χρησιμοποιείται κατά κόρρον για δοκιμή, επαλήθευση και επικοινωνία με τις διάφορες API μιας εφαρμογής ή ενός έργου. Το Postman παρέχει έναν ευέλικτο περιβάλλον για τη δημιουργία, οργάνωση και εκτέλεση αιτήσεων σε APIs. Χάρη στον ενσωματωμένο επεξεργαστή αιτήσεων, προσφέρει την δυνατότητα να οριστούν ο τύπος της αίτησης (όπως GET, POST, PUT ή DELETE), τα δυνητικά headers και παραμέτροι, και τα δεδομένα που αποστέλλονται στον διακομιστή.

Από τα πιο σημαντικά χαρακτηριστικά του Postman είναι η δυνατότητα δημιουργίας και οργάνωσης συλλογών (collections) αιτημάτων. Το Postman προσφέρει την δυνατότητα στους προγραμματιστές να οργανώνουν και να αποθηκεύουν τις αιτήσεις τους σε συλλογές διαφόρων κατηγοριών, για εύκολη πρόσβαση και διαχείριση. Αυτό είναι ιδιαίτερα χρήσιμο κατά την ανάπτυξη και έλεγχο πολλαπλών λειτουργιών του API, που υλοποιείται. Επιπλέον, το Postman παρέχει ένα εκτεταμένο σύνολο εργαλείων για τη διαχείριση και την αυτοματοποίηση των δοκιμών. Για την ακρίβεια, επιτρέπει στον προγραμματιστή να δημιουργήσει σενάρια δοκιμής (test scripts) προς επαλήθευση της ακρίβειας των αποτελεσμάτων, να παρακολουθεί τα αποτελέσματα των αιτημάτων και να αυτοματοποιεί την εκτέλεσή τους.

2.4.3 Ubuntu Virtual Private Server

Η εκτέλεση του web server γίνεται σε έναν απομακρυσμένο Virtual Private Server (VPS). Ένας VPS είναι μια εικονική μηχανή (Virtual Machine), της οποίας η λειτουργία υποστηρίζεται από έναν φυσικό υπολογιστή, γνωστού ως οικοδεσπότη (host). Ο host αναλαμβάνει τη δημιουργία και διαχείριση πολλών Virtual Machines μέσω ειδικού λογισμικού, το οποίο ονομάζεται επόπτης (hypervisor). Κάθε Virtual Machine εκτελείται ως ανεξάρτητος server, παρέχοντας στον χρήστη την αίσθηση ότι διαθέτει έναν πραγματικό, αποκλειστικό server. Τα βασικότερα πλεονεκτήματα ενός Virtual Private Server συνοψίζονται παρακάτω.

- **Ευελιξία:** Εξ' αιτίας της φύσης τους, οι VPS μπορούν να προσαρμόζονται στις απαιτήσεις των χρηστών. Κάθε χρήστης μπορεί να επιλέξει τους πόρους (επεξεργαστή, μνήμη, αποθηκευτικό χώρο κ.λπ.) που χρειάζεται και, καθώς οι απαιτήσεις αυξάνονται (ή μειώνονται), να προσαρμόσει το σύστημα αναλόγως.
- **Οικονομία:** Ένας VPS είναι συνήθως πιο οικονομικός από έναν πραγματικό φυσικό server (dedicated servers), καθώς παρέχει τα ίδια πλεονεκτήματα με μειωμένο κόστος. Αυτό οφείλεται στο ότι ο VPS δεν δεσμεύει ολόκληρο το hardware αλλά συγκεκριμένους πόρους.
- **Ιδιωτικότητα και ασφάλεια:** Επειδή κάθε VPS είναι ανεξάρτητος από τους υπόλοιπους, οι ενέργειες των υπόλοιπων χρηστών δεν τους επηρεάζουν. Επίσης, είναι δυνατό να εφαρμοστεί περιορισμός πρόσβασης και εγκατάσταση επιπλέον λογισμικών ασφαλείας, ώστε να αποτραπεί η είσοδος σε κακόβουλους τρίτους.
- **Επεκτασιμότητα:** Προηγούμενως, αναφέρθηκε η δυνατότητα επεκτασιμότητας του VPS. Αυτό είναι αρκετά εύκολο, και δεν απαιτεί ιδιαίτερες ενέργειες. Οι χρήστες μπορούν να δεσμεύσουν επιπλέον πόρους (επεξεργαστή, αποθηκευτικό χώρο, μνήμη κ.λπ.), ενώ τα δεδομένα δεν χρειάζεται να μεταφερθούν σε νέο server.
- **Πλήρης έλεγχος:** Οι χρήστες έχουν άμεσο έλεγχο και πρόσβαση στον server. Αυτό τους επιτρέπει την διαμόρφωση και την διαχείριση, ανάλογα με τις ανάγκες τους, εγκαθιστώντας λογισμικό, ρυθμίζοντας τους διακομιστές εφαρμογών και προσαρμόζοντας τις ρυθμίσεις ασφαλείας.

2.5 Υλοποίηση Εφαρμογής

Έχοντας περιγράψει όλα τα βασικά συστατικά που χρησιμοποιήθηκαν στην ανάπτυξη του συστήματος παρατήρησης, μπορούμε να περιγράψουμε την κατασκευή του συστήματος και τον τρόπο λειτουργίας του. Αρχικά, παρουσιάζεται η ανάπτυξη της βασικής δομής της εφαρμογής, μέσω της χρήσης των πακέτων Flask και PyMongo. Έπειτα, παρουσιάζεται το σύστημα ασφαλείας της εφαρμογής, μέσω πιστοποίησης ταυτότητας με χρήση ρόλων και δικαιωμάτων για κάθε πόρο. Τέλος, περιγράφεται η διαδικασία εγκατάστασης του server σε απομακρυσμένο μηχάνημα VPS και προώθησης στο Διαδίκτυο.

2.5.1 Δημιουργία Πόρων και Απόδοση Λειτουργικότητας

Όπως περιγράφηκε και στην ενότητα 2.2.2, για την δημιουργία μιας web εφαρμογής με την χρήση του Flask, το πρώτο πράγμα που χρειάζεται να δημιουργήσουμε, είναι ένα αντικείμενο Flask. Αφού δημιουργηθεί, το επόμενο βήμα είναι ο ορισμός τελικών σημείων (endpoints), με την απόδοση URLs, και η προσθήκη λειτουργικότητας, με την χρήση συναρτήσεων. Ο πρώτος πόρος που δημιουργήθηκε είναι ο πόρος για την λήψη και αποθήκευση των εγγράφων JSON στην βάση. Για να αξιοποιηθεί η βάση MongoDB εντός του προγράμματος, απαιτείται η σύνδεση μέσω ενός πελάτη Mongo στο DBMS. Αυτό γίνεται, με τον τρόπο που περιγράφηκε στην ενότητα 2.2.3. Ο παρακάτω κώδικας αντιστοιχεί στην διαδικασία που περιγράφηκε.

Listing 2.8: Δημιουργία εφαρμογής και πόρου για την λήψη και αποθήκευση στην βάση.

```
from flask import Flask, request, render_template
from pymongo import MongoClient

# Create Flask app.
app = Flask(__name__)

# Connect to Mongo client and access "Data" db.
client = MongoClient("mongodb:<IP>:27017/Data")
db = client["Data"] # The "Data" DB

try:
    Emissions = db["Emissions"]
except TypeError:
    db.create_collection(
        "Emissions",
        timeseries=dict(timeField="timestamp", metaField="metadata",
            granularity="seconds")
    )
    Emissions = db["Emissions"]

@app.route('/insert-data', methods=["POST"])
def get_data():
    # Data got from Pi.
    post_data = request.json
```

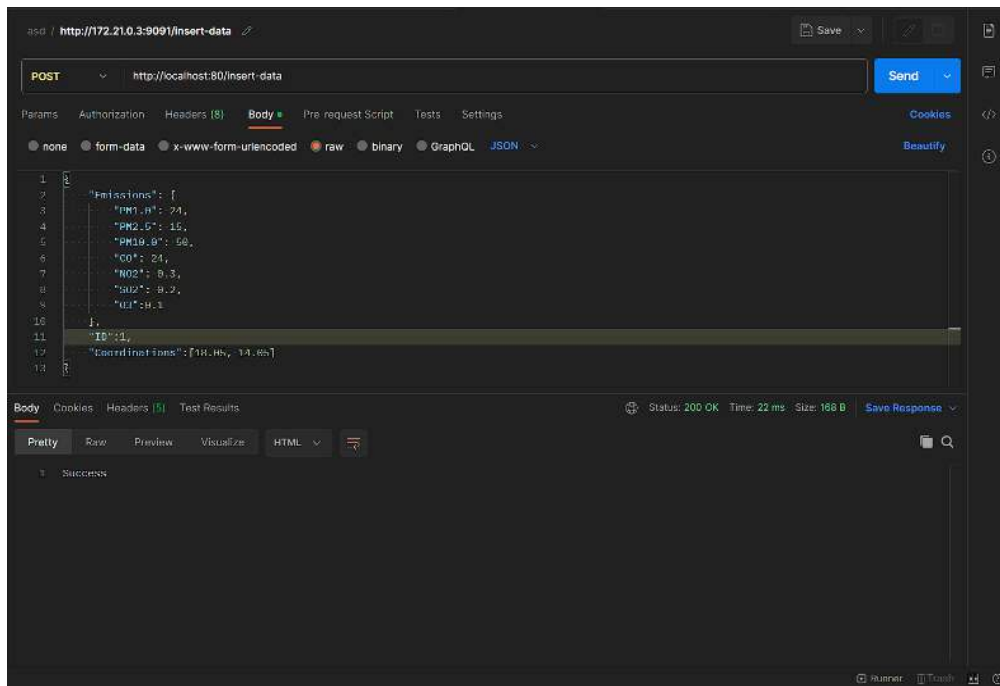
```
# Data added to time series.
Emissions.insert_one(
    dict(
        metadata=dict(
            sensorId=post_data.get("ID"),
            position=dict(Longitude=post_data.get("Coordinates")[0],
                          Latitude=post_data.get("Coordinates")[1]
            ),
            timestamp=datetime.now(timezone.utc).isoformat().replace("+00:00",
                              "Z"),
            Emissions=post_data.get("Emissions")
        )
    )

return "Success"
```

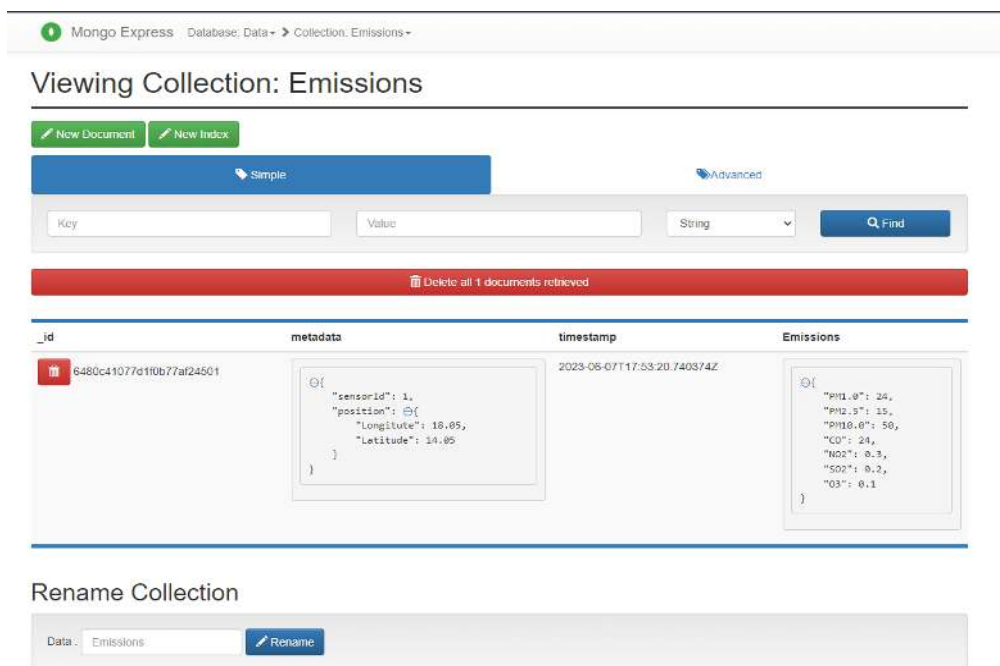
Μετά την εκτέλεση της εφαρμογής με την χρήση του Docker container σε τοπικό nginx server και βάση, γίνεται χρήση του περιβάλλοντος Postman για τον έλεγχο της αποστολής των δεδομένων. Στην εικόνα 2.9i φαίνονται τα δεδομένα που αποστάλθηκαν με το Postman καθώς και το μήνυμα επιτυχίας. Δίνοντας την τοπική διεύθυνση και την θύρα 8081, εμφανίζεται το mongo express, μία διεπαφή χρήστη, όπου παρουσιάζονται οι προϋπάρχουσες βάσεις του DBMS, καθώς και η βάση με την συλλογή «Data» που μόλις δημιουργήθηκε. Στην εικόνα 2.9ii παρουσιάζεται το έγγραφο καταχωρημένο στην συλλογή «Emissions» που αντιστοιχεί στα δεδομένα που στάλθηκαν μέσω Postman.

Όπως φαίνεται στον παραπάνω κώδικα, τα δεδομένα αποθηκεύονται σε συλλογή χρονοσειράς. Οι χρονοσειρές (time series) αποτελούν ακολουθίες δεδομένων με χρονολογική σειρά, δηλαδή εκτός από τις τιμές τους καταχωρείται και η χρονική στιγμή που αυτές καταγράφηκαν. Τα δεδομένα συλλέγονται ως επί το πλείστον περιοδικά, με τα διαστήματα να ποικίλουν, από έτος μέχρι δευτερόλεπτα. Η δομή αυτή, προσφέρει πλεονέκτηματα, όσον αφορά την οπτικοποίηση των δεδομένων και, κυρίως, την πρόβλεψη των τιμών. Αν και σε αυτές τις περιπτώσεις κρίσεις δεν είναι εύκολο να γίνει πρόβλεψη των τιμών, η οπτικοποίηση των δεδομένων είναι αρκετά χρήσιμη. Εκτός αυτού, επειδή οι κόμβοι καταγράφουν παρατηρήσεις σε περιοδικά διαστήματα, θα ήταν ευκολότερο και καλύτερο στην διαχείριση να καταχωρηθούν σε αυτή την μορφή.

Το endpoint /inser-data είναι υπεύθυνος για να δέχεται δεδομένα και να τα αποθηκεύει στην βάση. Το αίτημα που έρχεται στον server περιέχει, όπως φαίνεται και στην εικόνα 2.9i, περιέχει ένα αρχείο JSON. Η συνάρτηση αυτή φορτώνει το JSON αρχείο σε μια μεταβλητή. Στην συνέχεια, με την χρήση της εντολής insert_one προσθέτει στην συλλογή «Emissions» ένα νέο έγγραφο, στο οποίο καταχωρεί τις τιμές του αρχείου που έλαβε. Επίσης, καταχωρεί τον χρόνο που το δεδομένο αυτό αποθηκεύεται στην βάση, σε Συντονισμένη Παγκόσμια Ώρα (Universal Time Coordinated - UTC), καθώς και σε χρόνο Zulu (Z).



(i) Αποστολή Δοκιμαστικών Δεδομένων μέσω Postman.



(ii) Έγγραφο Δοκιμαστικών Δεδομένων.

Μετά την αποστολή και αποθήκευση των παρατηρήσεων από τους κόμβους, ακολουθεί η ανάκτηση των δεδομένων. Συγκεκριμένα, θα πρέπει ο πελάτης/τελικός χρήστης να μπορεί να προσπελαίνει τις τιμές, σε περίπτωση που θέλει να τις αξιοποιήσει (οπτικοποίηση, πρόβλεψη κ.λπ.), καθώς και να λαμβάνει την τιμή του πιο πρόσφατου δείκτη ποιότητας αέρα. Ο κώδικας 2.9 περιέχει τρία endpoints, που αντιστοιχούν στις παραπάνω λειτουργίες. Όπως φαίνεται, το ένα endpoint ανακαλεί απ' την βάση το πιο πρόσφατο έγγραφο και από τις τιμές που είναι καταχωρημένες υπολογίζει τον δείκτη ποιότητας αέρα, ενώ τα άλλα δύο βρίσκουν και επιστρέφουν είτε σε JSON μορφή είτε σε ιστοσελίδα τα N πιο πρόσφατα δεδομένα.

Listing 2.9: Δημιουργία πόρων για την επιστροφή δείκτη ποιότητας αέρα, η πιο πρόσφατων εγγραφών και διεπαφής χρήστη για αναζήτηση.

```
from flask import jsonify
from bson.json_utils import dumps

# Sends the AQI, based on the latest.
@app.route('/aqi', methods=["GET"])
def send_aqi():
    """Return the European Air Quality Index estimation based on
    concentration values for up to five pollutants (ug/m3)"""

    EAQI_INDEX = {
        0: "Good",
        1: "Fair",
        2: "Moderate",
        3: "Poor",
        4: "Very Poor",
        5: "Extremely Poor",
        6: "No Data"
    }

    ceil_pol = [[10, 20, 25, 50, 75, 800], # PM2.5
                 [20, 40, 50, 100, 150, 1200], # PM10.0
                 [40, 90, 120, 230, 340, 1000], # NO2
                 [50, 100, 130, 240, 380, 800], # O3
                 [100, 200, 350, 500, 750, 1250]] # SO2

    # Query for the latest particle values.
    # Returns Cursor object, something like
    # [{"metadata":..., "timestamp":..., "Emissions":...}]
    emission = Emissions.find().sort([("timestamp", -1)]).limit(1)
    emission = emission[0]
    c_pm2_5 = emission['Emissions']['PM2.5']
    c_pm10 = emission['Emissions']['PM10.0']
    c_no2 = emission['Emissions']['NO2']*1000*46/24.45
    c_o3 = emission['Emissions']['O3']*1000*48/24.45
    c_so2 = emission['Emissions']['SO2']*1000*64/24.45
    c_pol_list = [c_pm2_5, c_pm10, c_no2, c_o3, c_so2]

    ch_list = []

    for c_pol in c_pol_list:
        pol_i = 0
        for ceil in ceil_pol[c_pol_list.index(c_pol)]:
            if c_pol < ceil:
                pol_i = ceil_pol[c_pol_list.index(c_pol)].index(ceil)
                break
        ch_list.append(pol_i)

    index = max(ch_list)
    return jsonify(
        timestamp=emission["timestamp"],
```

```
        sensorId=emission["metadata"]["sensorId"],
        position=dict(
            Longitude=emission["metadata"]["position"]["Longitude"],
            Latitude=emission["metadata"]["position"]["Latitude"]
        ),
        AQI=EAQI_INDEX[index]
    )

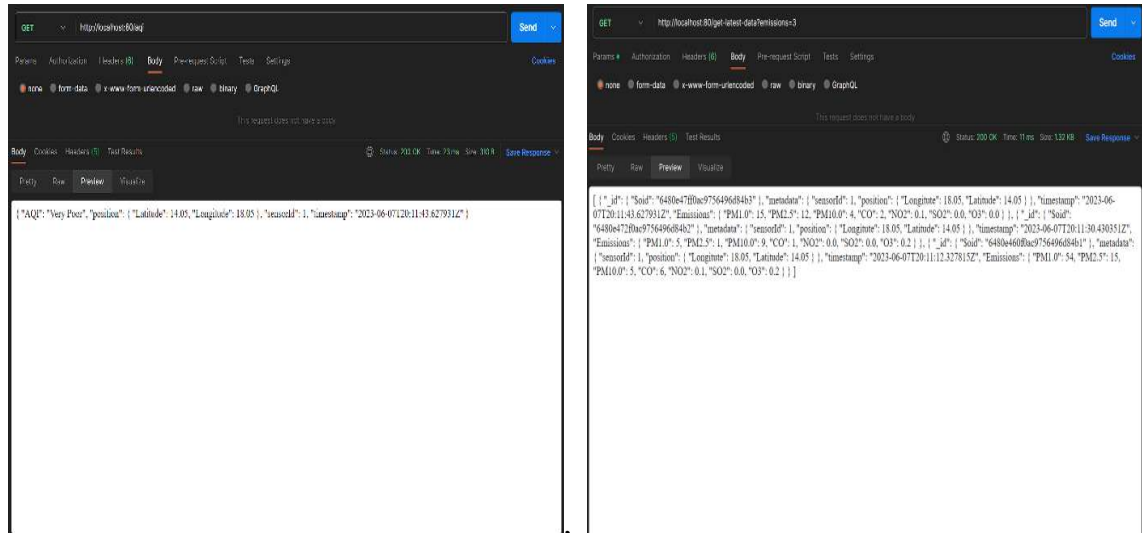
@app.route('/get-latest-data', methods=["GET"])
def send_latest():
    num_of_latest_emissions = int(request.args.get("emissions")) # Get
    # the num-of-rows from a URL like:
    # "http://whatever/get-latest-data?emissions=5"
    data = dumps(list(Emissions.find().sort([('timestamp',
        -1)]).limit(num_of_latest_emissions)), indent=2)

    return data

# This is what the user see and use.
@app.route('/render-data', methods=["POST", "GET"])
def render_site():
    if request.method == "POST":
        num_rows = int(request.form['num_rows'])
        data = Emissions.find().sort([('timestamp', -1)]).limit(num_rows)
        return render_template('index.html', data=data, num_rows=num_rows)

    return render_template('index.html')
```

Το πρώτο endpoint ξεκινάει ένα λεξικό που αντιστοιχίζει μια τιμή (0, 1, ..., 6) με την τιμή του δείκτη ποιότητας αέρα («Good», «Fair», ..., «No Data»). Αν και δεν αναφέρθηκε, η τιμή «No Data» σημαίνει πώς δεν υπάρχουν δεδομένα για μία περιοχή. Μαζί με το λεξικό, περιέχει έναν διδιάστατο πίνακα, τον `ceil_pol`, με στοιχεία τις τιμές των ρύπων (οριζοντίως αναπαριστούνται οι ρύποι και καθέτως οι τιμές του δείκτη). Το πρώτο βήμα είναι η ανάκτηση του πιο πρόσφατου εγγράφου στην βάση. Εδώ, υπάρχει η εξής ιδιαιτερότητα: η MongoDB επιστρέφει ένα αντικείμενο `Cursor`, το οποίο είναι επαναλήψιμο (`iterable`), ακόμη και στην περίπτωση που ζητείται μόνο ένα έγγραφο. Συνεπώς, η ανάκτηση του πιο πρόσφατου εγγράφου γίνεται με δύο εντολές: το ερώτημα και την ανάκτηση από το αντικείμενο `Cursor`. Το επόμενο βήμα είναι ο καθορισμός της τιμής των ρύπων. Συγκεκριμένα, για κάθε ρύπο, ελέγχεται σε ποιο εύρος ανήκει η τιμή, χρησιμοποιώντας τον πίνακα `ceil_pol` (π.χ. για τα σωματίδια $PM_{2.5}$ με τιμή $22 \mu g/m^3$, η τιμή του ανήκει στο εύρος [20, 25]). Αφού διαπιστωθούν τα εύρη τιμών για κάθε ρύπο, υπολογίζεται ποιος ρύπος έχει την μεγαλύτερη τιμή και με βάση αυτή υπολογίζεται ο δείκτης. Πριν γίνει η σύγκριση, είναι σημαντικό να γίνει η μετατροπή των τιμών των αερίων από *ppm* σε $\mu g/m^3$. Χωρίς αυτό, οι τιμές των αερίων σε *ppm* είναι σημαντικά μικρότερες και συνεπώς, οι τιμές των αερίων πάντα θα υπολείπονται.



- (iii) Endpoint ανάκτησης Δείκτη Ποιότητας Αέρα. (iv) Endpoint ανάκτησης N πιο πρόσφατα καταχωρημένων εγγράφων, μέσω URL.

MongoDB Time Series Example

Enter number of rows to display:

Time Series:

- 2023-06-07T20:11:43.627931Z - {PM10: 15, PM25: 12, PM100: 4, CO: 2, NO2: 0.1, SO2: 0.0, O3: 0.0} - P1_id: 1 - Coordinates: 18.05, 14.05
- 2023-06-07T20:11:30.430351Z - {PM10: 5, PM25: 1, PM100: 9, CO: 1, NO2: 0.0, SO2: 0.0, O3: 0.2} - P1_id: 1 - Coordinates: 18.05, 14.05
- 2023-06-07T20:11:12.327815Z - {PM10: 54, PM25: 15, PM100: 5, CO: 6, NO2: 0.1, SO2: 0.0, O3: 0.2} - P1_id: 1 - Coordinates: 18.05, 14.05

- (v) Διεπαφή Χρήση ανάκτησης N πιο πρόσφατα καταχωρημένων εγγράφων.

Το δεύτερο και το τρίτο endpoint έχουν πρακτικά την ίδια λειτουργικότητα. Συγκεκριμένα, και τα δύο αναζητούν και επιστρέφουν τα N πιο πρόσφατα καταχωρημένα έγγραφα της συλλογής. Οι διαφορές τους είναι πως στο δεύτερο endpoint το πλήθος N των εγγράφων δίνεται ως όρισμα από τον χρήστη μέσω του URL, ως όρισμα αιτήματος HTTP. Στο τρίτο endpoint υπάρχει διεπαφή χρήστη που υλοποιείται μέσω HTML και ο χρήστης χρησιμοποιεί στοιχεία της HTML για την καταχώρηση του πλήθους N . Επίσης, στο δεύτερο endpoint τα δεδομένα επιστρέφουν ως μια λίστα με έγγραφα JSON (N το πλήθος) ενώ στο τρίτο, τα δεδομένα εμφανίζονται κατάλληλα διαμορφωμένα πίσω στην σελίδα HTML της διεπαφής χρήστη.

Στις εικόνες 2.9iii, 2.9iv, 2.9v παρουσιάζονται οι απαντήσεις, που επιστρέφουν τα endpoints. Οι εικόνες 2.9iii και 2.9iv έχουν απαντήσεις σε μορφή JSON, ενώ η εικόνα 2.9v παρουσιάζει μια διαδραστική σελίδα για την επικοινωνία του χρήστη με την εφαρμογή.

2.5.2 Σύστημα Πιστοποίησης Ταυτότητας

Μέχρι στιγμής, η εφαρμογή δουλεύει ικανοποιητικά. Επικοινωνεί με τους τελικούς χρήστες και τους κόμβους, αποθηκεύει και προσπελάζει δεδομένα και παρέχει εύκολο περιβάλλον για την αλληλεπίδραση του χρήστη. Όμως, καθώς οι δυνητικοί χρήστες αυξάνονται, αυξάνεται και ο κίνδυνος χρήσης της εφαρμογής και από κακόβουλους χρήστες. Συνεπώς, χρειάζεται η ανάπτυξη ενός συστήματος που θα προσφέρει ασφάλεια στην εισόδο των χρηστών.

Η πιο εύκολη και απλή υλοποίηση είναι μέσω του HTTP. Το HTTP προσφέρει ένα framework, με όλα τα απαραίτητα στοιχεία ασφάλειας. Συγκεκριμένα, σε έναν πελάτη που δεν έχει πιστοποιηθεί, ο server επιστρέφει κωδικό 401 (Μη Πιστοποιημένος) και τον παραπέμπει στα βήματα της διαδικασίας της πιστοποίησης. Από την άλλη, για την αποστολή των πιστοποιητικών, ο πελάτης χρησιμοποιεί τις κεφαλίδες του HTTP. Ο server ελέγχει εκ νέου αυτά τα στοιχεία και αν ο πελάτης πιστοποιηθεί, τότε του επιστρέφεται ο κωδικός 200 (Εντάξει), αλλιώς πάλι ο κωδικός 401. Υπάρχουν διάφορα σχήματα τα οποία υλοποιούν αυτήν την ιδέα. Το πιο απλό σχήμα είναι το Basic. Σ' αυτό, τα στοιχεία της πιστοποίησης είναι ένα αναγνωριστικό που αντιστοιχεί σε έναν χρήστη μαζί με έναν κωδικό. Σε κάθε αίτημα, ο χρήστης δίνει τα στοιχεία αυτά και ο server ελέγχει αν ο χρήστης είναι καταχωρημένος στο σύστημα και επιστρέφει την αντίστοιχη απάντηση. Τα στοιχεία αυτά κωδικοποιούνται, αλλά δεν είναι κρυπτογραφημένα και γι' αυτό πρέπει να χρησιμοποιείται το πρωτόκολλο HTTPS/TLS. Προς το παρόν, το τελευταίο δεν έχει ενσωματωθεί στην εφαρμογή.

Δημιουργία Βάσης και πρώτων Χρηστών

Το πρωταρχικό κομμάτι για την υλοποίηση του συστήματος είναι η δημιουργία μιας βάσης δεδομένων με τους χρήστες. Συγκεκριμένα, εκτός από την βάση δεδομένων και την συλλογή με τους ρύπους, δημιουργήθηκε μια ακόμη βάση με μια συλλογή μόνο με χρήστες. Στην βάση αυτή αποθηκεύονται σε μορφή εγγράφων τα αναγνωριστικά των χρηστών καθώς και οι κωδικοί τους. Οι κωδικοί εισάγονται κρυπτογραφημένοι, που σημαίνει πως αν κάποιος αποκτήσει πρόσβαση στην βάση, δεν θα μπορεί να γνωρίζει τους κωδικούς των χρηστών, μόνο ψάχνοντας τα έγγραφα στην συλλογή. Για την δημιουργία της βάσης και των πρώτων χρηστών χρησιμοποιήθηκε το CLI της MongoDB, όπως φαίνεται στον κώδικα 6.

```
mongo mongo:27017
> use Users
switched to db Users
> db.createCollection("Users")
{"ok": 1}
> db.Users.insertOne({"id": "<userID>", "password": "<encrypted_password>"})
{
  "acknowledged": true,
  "insertedId": ObjectId("<object_id>")
}
```

Listing 6: Δημιουργία βάσης και συλλογής χρηστών της εφαρμογής και απλού χρήστη.

Σε αυτό το σημείο να αναφερθεί πως η MongoDB έχει την δυνατότητα υλοποίησης συστήματος ασφαλείας με χρήστες, όμως αυτό αφορά χρήστες που έχουν πρόσβαση και ρόλους στις βάσεις του DBMS και δεν έχει καμία σχέση με τους χρήστες της εφαρμογής. Δηλαδή, αν ένας χρήστης της MongoDB με δικαιώματα εγγραφής και ανάγνωσης στην βάση δοκιμάσει να συνδεθεί σε ένα endpoint της εφαρμογής, η εφαρμογή δεν θα του επιτρέψει την είσοδο (εκτός και αν υπάρχει χρήστης με ίδια ακριβώς στοιχεία).

Ο χρήστης που δημιουργήθηκε με αυτή την μέθοδο έχει πρόσβαση σε κάθε endpoint αυτής της εφαρμογής. Παρ' όλα αυτά, μπορεί κάποιος χρήστης να αποκτήσει πρόσβαση καταλάθος σε ένα endpoint που δεν θα έπρεπε να είχε, π.χ. ένας χρήστης δοκιμάζει να προσπελάσει το endpoint /insert-data, σκοπός του οποίου είναι μόνο η αποστολή δεδομένων από τους κόμβους. Αυτό μπορεί να προκαλέσει διάφορα προβλήματα τόσο στην εφαρμογή, όσο και στον web server. Για τον λόγο αυτό, χρησιμοποιήθηκε ένα επιπλέον μέτρο ασφάλειας: η χρήση ρόλων. Κάθε χρήστης μπορεί να προορίζεται μόνο για συγκεκριμένες ενέργειες (π.χ. οι κόμβοι μόνο αποστέλλουν δεδομένα, κανείς άλλος δεν αποστέλλει δεδομένα, οι διασώστες αιτούνται το Air Quality Index (AQI) κ.λπ.). Συνεπώς, πρέπει να ορίσουμε τους πιθανούς χρήστες και τους ρόλους τους. Στον πίνακα 2.9 παρουσιάζεται αυτή η κατηγοριοποίηση.

Πίνακας 2.9: Ρόλοι χρηστών και δυνατότητες.

Ρόλος	Δυνατότητες
Κόμβος	Αποστολή Δεδομένων
Πελάτης Δεδομένων	Ανάκτηση Δεδομένων
Πελάτης Δείκτη Ποιότητας Αέρα	Ανάκτηση Δείκτη Ποιότητας Αέρα
Προηγμένος Πελάτης	Ανάκτηση Δεδομένων, Ανάκτηση Δείκτη Ποιότητας Αέρα
Διαχειριστής	Ανάκτηση Δεδομένων, Ανάκτηση Δείκτη Ποιότητας Αέρα, Αποστολή Δεδομένων

Με βάση την παραπάνω κατηγοριοποίηση, προκύπτουν πέντε κατηγορίες χρηστών: οι κόμβοι, οι πελάτες που έχουν πρόσβαση απευθείας στα έγγραφα, οι πελάτες που έχουν πρόσβαση στον δείκτη ποιότητας αέρα, οι πελάτες που έχουν πρόσβαση και στα δύο και οι πελάτες που έχουν πρόσβαση σε όλα. Οι τελευταίοι έχουν επιπλέον δυνατότητες, οι οποίες θα αναφερθούν αργότερα. Ο παρακάτω κώδικας δείχνει πως θα εισαχθεί ένας χρήστης με ρόλους στην βάση χρηστών. Επειδή αυτοί οι ρόλοι δεν υπάρχουν καταχωρημένοι εντός της βάσης, αποθηκεύονται σε μία λίστα όλες δυνατότητές του (ακόμη και στην περίπτωση της μίας, όπως π.χ. των κόμβων).

```
mongo mongo:27017
> use Users
switched to db Users
> db.Users.insertOne({"id":"<userID>", "password":"<encrypted_password>",
"roles":["<role1>", ...]})
{
  "acknowledged": true,
  "insertedId": ObjectId("<object_id>")
}
```

Listing 7: Εισαγωγή στην βάση χρήστη με ρόλο.

Υλοποίηση στην Εφαρμογή

Έχοντας δημιουργήσει την βάση και έχοντας κάνει την διαδικασία εισαγωγής χρηστών μέσω CLI της MongoDB, ήρθε η ώρα της δημιουργίας του συστήματος ταυτοποίησης στην εφαρμογή. Αρχικά, χρειάζεται να γίνει η εγκατάσταση των κατάλληλων επεκτάσεων. Όπως αναφέρθηκε και στο κεφάλαιο 2.2.2, το Flask ενσωματώνει πολύ βασικές δυνατότητες, αλλά όχι ένα σύστημα διαχείρισης και ταυτοποίησης χρηστών. Παρ' όλα αυτά, μας δίνει την ελευθερία να το κάνουμε, με πληθώρα επεκτάσεων και τρόπων. Όπως αναφέρθηκε και προηγουμένως, η ταυτοποίηση γίνεται με την χρήση κεφαλίδων του HTTP. Η διαδικασία αυτή, μπορεί να γίνει μέσω του framework του Flask, όμως είναι αρκετά δύσκολη στην υλοποίηση. Εκτός αυτού, υπάρχουν αρκετές έτοιμες υλοποιήσεις, οι οποίες προσφέρονται ως extensions του Flask. Ένα τέτοιο extension χρησιμοποιήθηκε για την εφαρμογή.

Για την ανάπτυξη της εφαρμογής, χρησιμοποιήθηκε το πακέτο `werkzeug`, το οποίο υπάρχει προεγκατεστημένο στο Flask, καθώς και το extension `flask_httpauth`. Το πακέτο `werkzeug` αποτελεί μια ολοκληρωμένη βιβλιοθήκη ανάπτυξης web εφαρμογών στην Python. Παρέχει πληθώρα δυνατοτήτων που μπορεί να χρειαστούν στην ανάπτυξη μιας web εφαρμογής, όπως URL Mapping, Header Utilities, Cookies, Proxying κ.λπ. Εντός αυτών των εργαλείων βρίσκονται και κάποιες βασικές συναρτήσεις για ασφάλεια. Συγκεκριμένα, οι συναρτήσεις αυτές είναι η `generate_password_hash` και η `check_password_hash`. Η πρώτη συνάρτηση είναι υπεύθυνη για την δημιουργία κατακερματισμένων κωδικών πρόσβασης προς αποθήκευση. Η δεύτερη δέχεται δύο ορίσματα, έναν κατακερματισμένο κωδικό πρόσβασης και έναν απλό κωδικό πρόσβασης και ελέγχει αν ο κατακερματισμένος κωδικός πρόσβασης που προέκυψε προηγουμένως είναι ίδιος με τον απλό κωδικό πρόσβασης. Η πρώτη χρησιμοποιείται κατά την δημιουργία και εισαγωγή ενός χρήστη στην βάση, όπου ο κωδικός δεν αποθηκεύεται αυτούσιος, αλλά κρυπτογραφείται για λόγους ασφαλείας. Η δεύτερη χρησιμοποιείται κατά την είσοδο του χρήστη στην εφαρμογή, όπου εκεί γίνεται η ταυτοποίηση του.

Εκτός της βιβλιοθήκης `werkzeug`, αναφέρθηκε και το extension `flask_httpauth`. Το συγκεκριμένο πακέτο προσφέρει ένα σύνολο απλών εργαλείων για την ταυτοποίησης μέσω HTTP. Η γενική ιδέα είναι πως, αφού επιλέξουμε τα endpoints τα οποία επιθυμούμε να έχουν μέτρα ασφαλείας, τα δηλώνουμε με τον αντίστοιχο διακοσμητή και χρησιμοποιούμε μια συνάρτηση επαλήθευσης. Αυτό φαίνεται καλύτερα στον κώδικα 2.10. Γενικά, οι τρόποι ταυτοποίησης με το HTTP είναι πολλοί.

Το extension αυτό υλοποιεί τους τρεις πιο βασικούς τρόπους ταυτοποίησης: την Basic, την Digest και μέσω Token. Όπως αναφέρθηκε στην αρχή του κεφαλαίου, η επιλεγμένη ταυτοποίηση είναι η Basic. Για την υλοποίησή της χρειάζεται να υπάρχουν καταχωρημένοι (σε κάποια δομή δεδομένων, σε μία βάση δεδομένων κ.λπ.) οι χρήστες μαζί με τους κωδικούς τους και κάθε φορά που ένας χρήστης προσπαθεί να συνδεθεί σε ένα endpoint, να γίνεται αναζήτηση του αναγνωριστικού του και του κωδικού πρόσβασης που αντιστοιχεί σε αυτό το αναγνωριστικό και να ελέγχεται ο κωδικός που καταχωρήθηκε από τον χρήστη αν ταυτίζεται με τον κωδικό που βρέθηκε (αν βρέθηκε) για τον χρήστη αυτό.

Listing 2.10: Ταυτοποίηση endpoint μέσω `http_auth` extension και Basic ταυτοποίησης. Πηγή: `flask-HTTPAuth`.

```
from flask import Flask
from flask_httpauth import HTTPBasicAuth
from werkzeug.security import generate_password_hash, check_password_hash

app = Flask(__name__)
auth = HTTPBasicAuth()

users = {
    "john": generate_password_hash("hello"),
    "susan": generate_password_hash("bye")
}

@auth.verify_password
def verify_password(username, password):
    if username in users and \
        check_password_hash(users.get(username), password):
        return username

@app.route('/')
@auth.login_required
def index():
    return "Hello, {}".format(auth.current_user())

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

Σε αυτό το σημείο, πρέπει να αναφερθεί πως οι χρήστες θα πρέπει να συνδέονται κάθε φορά που ο επιθυμούν να προσπελάσουν έναν διαφορετικό πόρο. Αυτό, γιατί με βάση την αρχιτεκτονική REST, ο server είναι stateless, δηλαδή η πληροφορία που αποστέλλεται σε κάθε αίτηση είναι η αναγκαία και δεν χρειάζεται να αποθηκεύει τίποτα ο sever. Αυτό, ως έναν βαθμό συμβαίνει, αλλά οι σύγχρονοι φυλλομετρητές αποθηκεύουν τα στοιχεία αυτά, για όσο διάστημα εκτελούνται, και τα αποστέλουν κάθε φορά που ο χρήστης προσπελάει έναν διαφορετικό πόρο. Συνεπώς, αν και τα στοιχεία ταυτοποίησης αποστέλλονται σε κάθε αίτημα HTTP, ο τελικός χρήστης έχει την εντύπωση πως παραμένει συνδεδεμένος.

Η υλοποίηση ρόλων με αυτό το extension είναι πολύ απλή: στον διακοσμητή `login_required` προστίθεται το όρισμα `role` και οι τιμές του είναι είτε ένα αλφαριθμητικό, είτε μία λίστα αλφαριθμητικών είτε μία λίστα με αλφαριθμητικά και λίστες αλφαριθμητικών. Αναλόγως την περίπτωση, επιλέγουμε και την κατάλληλη

υλοποίηση. Για την συγκεκριμένη εφαρμογή, χρησιμοποιείται η τρίτη προσέγγιση για δύο λόγους: η δομή των ρόλων και το πλήθος των χρηστών. Η δομή των ρόλων ορίζει πως ένας ρόλος μπορεί να αποτελείται από μια λίστα ενεργειών. Για παράδειγμα, ο ρόλος *Πελάτη Δεδομένων* μόνο από την ενέργεια λήψη δεδομένων, ενώ ο ρόλος *Προηγμένου Πελάτη* αποτελείται από μια λίστα με ενεργείες την λήψη δεδομένων και την λήψη δείκτη ποιότητας αέρα. Το πλήθος των χρηστών ορίζει τους χρήστες με διαφορετικούς ρόλους που έχουν πρόσβαση στον ίδιο πόρο. Για παράδειγμα, ο χρήστης με ρόλο *Πελάτη Δεδομένων* έχει πρόσβαση στον πόρο `/get-latest-data`, στον οποίο όμως έχει πρόσβαση και ο χρήστης με ρόλο *Προηγμένου Πελάτη*. Ο κώδικας 2.11 αναπαριστά την υλοποίηση αυτού του σχήματος.

Listing 2.11: Χρήση ρόλων με το extension *Flask HTTPAuth*. Πηγή: *flask-httpauth*

```
@auth.get_user_roles
def get_user_roles(user):
    return user.get_roles()

@app.route('/admin')
@auth.login_required(role=['user', ['moderator', 'contributor']])
def admins_only():
    return "Hello {}, you are a user or a
           moderator/contributor!".format(auth.current_user())
```

Όπως φαίνεται στον κώδικα 2.10, χρειάζεται μια συνάρτηση για την πιστοποίηση του κωδικού του χρήστη που συνδέεται, καθώς και ένας διακοσμητής κάτω από κάθε πόρο ορισμένο από το Flask. Μέσω των διακοσμητών του extension, η συνάρτηση πιστοποίησης συσχετίζεται με τους αντίστοιχους πόρους και καλείται κάθε φορά που ένας χρήστης πραγματοποιεί σύνδεση. Για την συγκεκριμένη εφαρμογή, η υλοποίησή της είναι αρκετά πιο σύνθετη. Κατ' αρχάς, επειδή τα στοιχεία είναι αποθηκευμένα σε μια βάση δεδομένων αντί για μια δομή δεδομένων, χρειάζεται πρώτα να ανακληθούν τα σωστά στοιχεία απ' την βάση και επείτα να πραγματοποιηθεί ο έλεγχος. Όμως, επειδή χρησιμοποιούνται και ρόλοι, τα δεδομένα που χρησιμοποιούνται στην ταυτοποίηση και είσοδο στους πόρους αυξάνονται. Συνεπώς, η αποθήκευση αυτών σε διαφορετικές δομές/μεταβλητές θα έκανε το σύνολο το πράξεων ταυτοποίησης αρκετά πιο πολύπλοκο. Για τον λόγο αυτό, αναπτύχθηκε μια ειδικά διαμορφωμένη κλάση χρηστών, υπεύθυνη για την αποθήκευση και την ευέλικτη διαχείριση των ζητημάτων της εισόδου. Η κλάση αυτή φαίνεται στον κώδικα 2.12.

Listing 2.12: Κλάση χρηστών.

```
class User:
    db = users_db

    def __init__(self, username=None, password=None, roles=None):
        self.username = username
        self.password = password
        self.roles = roles # A list of roles.

    @staticmethod
    def check_password(password, password_hash):
```

```
        return check_password_hash(password_hash, password)

    @classmethod
    def get_by_username(cls, username):
        user_dict = cls.db["Users"].find_one({"username": username})
        if user_dict is not None:
            return cls(user_dict["username"], user_dict["password"],
                       user_dict["roles"])
        else:
            return None

    def get_roles(self):
        return self.roles

    def update_username(self, username):
        old_username = self.username
        self.username = username
        self.update_user_in_db(old_username)

    def update_password(self, password):
        self.password = password
        self.update_user_in_db(self.username)

    def update_roles(self, roles):
        self.roles = roles
        self.update_user_in_db(self.username)

    def update_user(self, username, password, roles):
        old_username = self.username
        self.username = username
        self.password = password
        self.roles = roles
        self.update_user_in_db(old_username)

    def update_user_in_db(self, old_username):
        self.db["Users"].update_one({"username": old_username},
                                     {'$set': {"username": self.username,
                                               "password":
                                                   generate_password_hash(self.password),
                                               "roles": self.roles
                                               }})

    def insert_user_in_db(self):
        self.db["Users"].insert_one(dict(
            username=self.username,
            password=generate_password_hash(self.password),
            roles=self.roles
        ))
```

Η κλάση αυτή αποτελείται από τρεις μεταβλητές υπόστασης, οι οποίες αποθηκεύουν το αναγνωριστικό χρήστη (username), τον κωδικό (password) και τους ρόλους (roles) των χρηστών. Χρησιμοποιεί μία στατική μέθοδο wrapper για τον

έλεγχο του κωδικού πρόσβασης και μία classmethod για την ανάκληση του χρήστη από την βάση, με βάση το αναγνωριστικό που εισήγαγε κατά την σύνδεση. Οι μέθοδοί του και οι ενέργειές τους φαίνονται στον πίνακα 2.10. Οι περισσότερες μέθοδοι χρησιμοποιούνται για την δημιουργία και την ενημέρωση καταχωρήσεων στην βάση, σε συνδυασμό με μία απλή διεπαφή χρήστη, μέσω μιας σελίδας HTML. Προηγουμένως αναφέρθηκε, πως οι χρήστες με ρόλο διαχειριστή έχουν επιπλέον δυνατότητες. Αυτές είναι να μπορούν να διαχειρίζονται τους χρήστες της βάσης, δηλαδή την εισαγωγή και την ενημέρωση.

Πίνακας 2.10: Μέθοδοι κλάσης χρηστών.

Μέθοδος	Ενέργειες
get_roles	Επιστρέφει λίστα ρόλων συνδεδεμένου χρήστη
instert_user_in_db	Εισαγωγή νέου χρήστη σε βάση δεδομένων. Ο κωδικός κρυπτογραφείται πριν την καταχώρηση
update_user_in_db	Ενημέρωση στοιχείων χρήστη στην βάση δεδομένων. Καλείται σε κάθε ενημέρωση χρήστη. Ο κωδικός κρυπτογραφείται πριν την καταχώρηση
update_user	Ενημέρωση στοιχείων χρήστη
update_username	Ενημέρωση αναγνωριστικού χρήστη
update_password	Ενημέρωση κωδικού πρόσβασης χρήστη
update_roles	Ενημέρωση ρόλων χρήστη

Το περιβάλλον για την διαχείριση των χρηστών μέσω της σελίδας HTML παρουσιάζεται στις εικόνες 2.9vi, 2.9vii, 2.9viii, 2.9ix, 2.9x και 2.9xi. Με την είσοδό του στην σελίδα, εμφανίζεται στον διαχειριστή μια σελίδα με πέντε κουμπία (εικόνα 2.9vi). Το κάθε κουμπί προκαλεί μια ενέργεια και αλλάζει την εκάστοτε μορφή της σελίδας. Το κουμπί *Create User* αλλάζει την σελίδα κατάλληλα, για την δημιουργία νέου χρήστη. Εμφανίζει κάτω από τα αρχικά κουμπία, πέντε κουμπία επιλογών, για τους ρόλους των χρηστών, δύο πεδία κειμένου, για την εισαγωγή αναγνωριστικού χρήστη και κωδικού και ένα κουμπί υποβολής των στοιχείων. Το κουμπί *Update Username* τροποποιεί την σελίδα, ώστε να ενημερωθεί το αναγνωριστικό του χρήστη. Κάτω από τα αρχικά κουμπία εμφανίζονται δύο πεδία για την εισαγωγή του αναγνωριστικού και του νέου αναγνωριστικού, καθώς και ένα κουμπί υποβολής. Το κουμπί *Update Password* τροποποιεί την σελίδα, ώστε να ενημερωθεί ο κωδικός του χρήστη. ομοίως με πριν, εμφανίζονται δύο πεδία, για την εισαγωγή αναγνωριστικού χρήστη και νέου κωδικού πρόσβασης, και ένα κουμπί υποβολής. Το κουμπί *Update Role* τροποποιεί την σελίδα, ώστε να ενημερωθεί ο ρόλος του χρήστη. Κάτω απ' τα αρχικά κουμπία εμφανίζονται ένα πεδίο, για το αναγνωριστικό χρήστη, πέντε κουμπία επιλογής, για την επιλογή ρόλου, και το κουμπί υποβολής. Τέλος, το κουμπί *Show Users* εμφανίζει για όλους τους χρήστες το αναγνωριστικό τους, τον κρυπτογραφημένο κωδικό πρόσβασης και τον ρόλο τους.

Create User Update Username Update Password Update Role Show Users

(vi)

Create User Update Username Update Password Update Role Show Users

Select values:

☐ Admin
☐ AQI Client
☐ Data Client
☐ Advanced Client
☐ Pi

Enter a username:

Enter a password:

Submit

(vii)

Create User Update Username Update Password Update Role Show Users

Enter a username:

Enter new username:

Submit

(viii)

Create User Update Username Update Password Update Role Show Users

Enter a username:

Enter new password:

Submit

(ix)

Create User Update Username Update Password Update Role Show Users

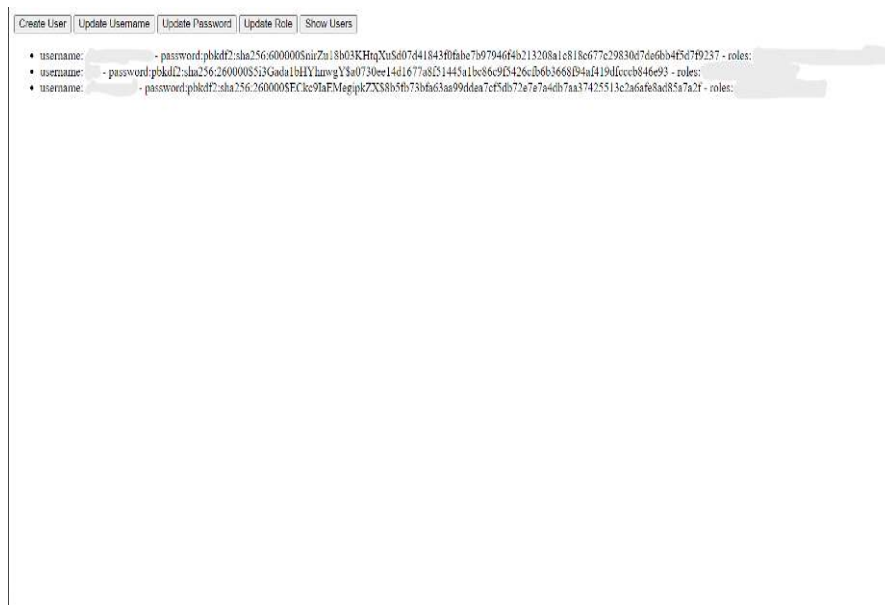
Enter a username:

Select values:

☐ Admin
☐ AQI Client
☐ Data Client
☐ Advanced Client
☐ Pi

Submit

(x)



(xi)

Αποτελέσματα

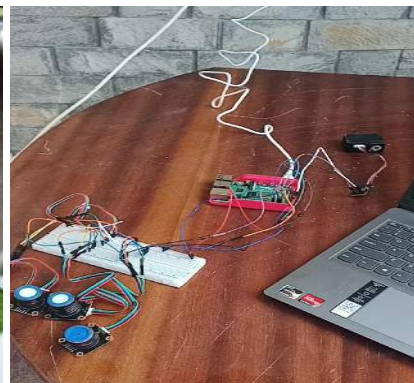
3.1 Πειραματικά Αποτελέσματα

Για την επαλήθευση της καλής λειτουργίας της εφαρμογής, πραγματοποιήθηκε πείραμα καταγραφής ποιότητας αέρα από τους κόμβους και αποστολής και καταχώρησης στην βάση του web server. Συγκεκριμένα, σε ειδικά διαμορφωμένες συνθήκες, πραγματοποιήθηκαν δοκιμές, όπου οι αισθητήρες ενός κόμβου κατέγραφαν τον μολυσμένο άνεμο που κατευθυνόταν προς αυτά σε μια περίοδο ενός δευτερολέπτου και στην συνέχεια έστελναν αίτημα στον web server, σύμφωνα με τον κώδικα που παρουσιάστηκε στο κεφάλαιο 2.1.4. Οι τιμές που κατέγραφαν την εκάστοτε χρονική στιγμή, καθώς και οι αντίστοιχες απαντήσεις που λάμβαναν από τον web server, εμφανίζονται στο τερματικό.

Προηγουμένως, αναφέρθηκε πως το πείραμα έγινε σε ειδικές συνθήκες. Συγκεκριμένα, για την παραγωγή μολυνστικού αέρα, χρειάζεται καύση οργανικής ύλης (κεφάλαιο 2.1). Η καύση της ύλης πρέπει να πραγματοποιηθεί σε ένα πλαίσιο, το οποίο να μην επιτρέπει την διαρροή της φωτιάς εκτός του χώρου της καύσης. Η καύση επίσης, θα πρέπει να γίνει σε ανοιχτό χώρο, καθώς ο αέρας που παράγεται, θα πρέπει να μπορεί να διαχέεται και να μην περιορίζεται στο πλαίσιο, για προφανείς λόγους ασφαλείας. Οι κόμβοι χρειάζονται τροφοδοσία ρεύματος και δυνατότητα σύνδεσης στο διαδίκτυο, ώστε να λειτουργήσουν και να μπορούν να επικοινωνούν με τον web server αντίστοιχα. Συνεπώς, ο εξωτερικός χώρος θα πρέπει να παρέχει με κάποιον τρόπο αυτές τις απαιτήσεις. Με βάση αυτές τις ανάγκες, επιλέχθηκαν τα κατάλληλα υλικά.



(i) Ψησταριά που χρησιμοποιήθηκε.



(ii) Εξοπλισμός.

```

Ambient CO concentration is :26.0 ppm
Ambient O3 concentration is :0.2 ppm
PM1.0 value: 361 ug/m3
PM2.5 value: 428 ug/m3
PM10.0 value: 970 ug/m3
http://      /insert-data
POSTING...
Success!
Ambient CO concentration is :24.0 ppm
Ambient O3 concentration is :0.2 ppm
PM1.0 value: 420 ug/m3
PM2.5 value: 420 ug/m3
PM10.0 value: 1116 ug/m3
http://      /insert-data
POSTING...
Success!
Ambient CO concentration is :32.0 ppm
Ambient O3 concentration is :0.2 ppm
PM1.0 value: 420 ug/m3
PM2.5 value: 408 ug/m3
PM10.0 value: 1200 ug/m3
http://      /insert-data
POSTING...
Success!
Ambient CO concentration is :26.0 ppm
Ambient O3 concentration is :0.2 ppm
PM1.0 value: 341 ug/m3
PM2.5 value: 424 ug/m3
PM10.0 value: 1100 ug/m3
http://      /insert-data
POSTING...

```

(iii) Αποτελέσματα.

Σχήμα 3.1: Πείραμα Μετρήσεων.

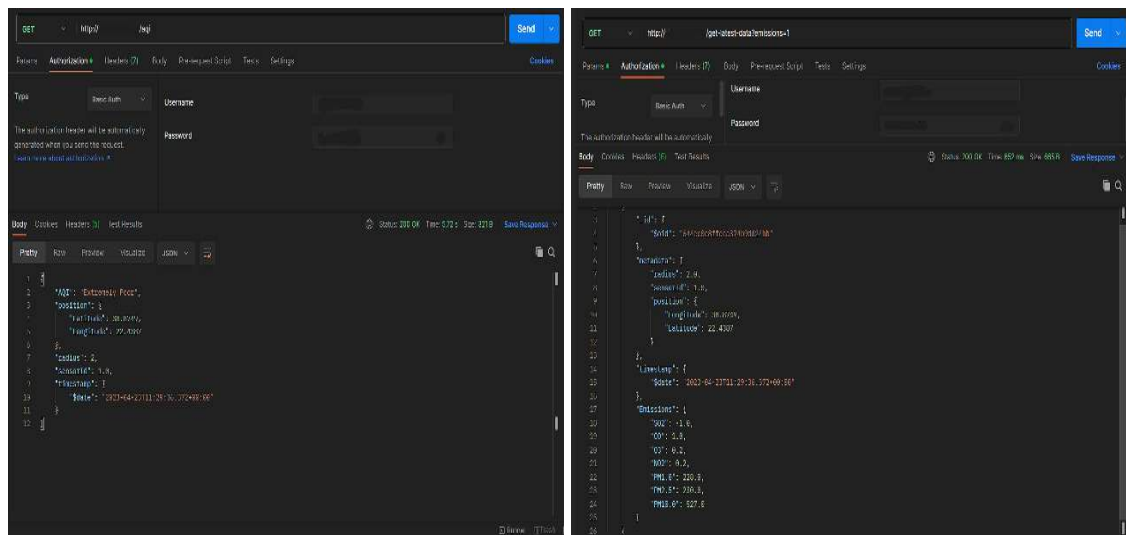
Το πείραμα διεξήχθη σε εξωτερικό χώρο (αυλή) με πρόσβαση στο διαδίκτυο και κοντινή πηγή τροφοδοσίας ρεύματος, καθώς δεν χρησιμοποιήθηκαν μπαταρίες. Για το πείραμα χρησιμοποιήθηκαν τα εξής: οι κόμβοι, όπως περιγράφηκαν στο κεφάλαιο 2.1, ένας φορητός υπολογιστής, μια ψησταριά και φύλλα. Τα φύλλα αποτελούν την οργανική ύλη προς καύση. Η ψησταριά αποτελεί το πλαίσιο στο οποίο γίνεται η καύση της οργανικής ύλης. Συγκεκριμένα, εξ' αιτίας της δομής, επιτρέπει στην φωτιά να μην διαχέεται εκτός αυτής, όμως ο αέρας που παράγεται μπορεί να διαχέεται δυνητικά προς όλες τις κατευθύνσεις. Οι κόμβοι καταγράφουν τις τιμές των αερίων του αέρα που παράγεται από την καύση. Τροφοδοτούνται από κοντινή πηγή ρεύματος και υπάρχει Wi-Fi για την σύνδεση τους στο διαδίκτυο. Ο φορητός υπολογιστής χρησιμοποιείται για την απομακρυσμένη πρόσβαση στους κόμβους μέσω του πρωτοκόλλου SSH και την εμφάνιση των αποτελεσμάτων της συλλογής και αποστολής στον server. Οι εικόνες 3.1i, 3.1ii και 3.1iii παρουσιάζουν τις συνθήκες του πειράματος καθώς και το αποτέλεσμα.

Τα αποτελέσματα φαίνονται στην εικόνα 3.1iii. Οι τιμές των αερίων είναι αρκετά υψηλές σε ppm (0.2 ppm), γεγονός που υποδεικνύει ότι η ποιότητα του αέρα πλησιάζει την τιμή «Πολύ Κακή». Επίσης, παρατηρείται υπερβολικά μεγάλη τιμή για τους αισθητήρες σωματιδίων. Αυτό εν μέρει στηρίζεται στο γεγονός, πως τα κάρβουνα που καίγονται γίνονται σκόνη, η οποία μεταφέρεται μαζί με τον αέρα. Λόγω του ύψους των τιμών αυτών, η ποιότητα του αέρα έχει τιμή «Επικίνδυνη». Η τιμή αυτή είναι biased, καθώς όπως αναφέρθηκε προηγουμένως, τα σωματίδια οφείλονται στα κάρβουνα τα οποία γίνονται σκόνη και μεταφέρονται μαζί με τον αέρα. Εξ' αιτίας αυτού, η πραγματική τιμή για την ποιότητα του αέρα θα είναι «Πολύ Κακή».

Στην δεύτερη φάση, γίνεται έλεγχος στην βάση, για την επαλήθευση καταχώρησης των δεδομένων. Συγκεκριμένα, γίνεται αίτημα προς τον πόρο `get-latest-data` με πλήθος `emission` ίσο με 1 και αντίστοιχα προς τον πόρο `aqi`. Το αποτέλεσμα της αίτησης προς τον πρώτο κόμβο θα πρέπει να έχει τιμές κοντινές με τις τιμές της εικόνας 3.1iii και το αποτέλεσμα της αίτησης προς τον δεύτερο κόμβο θα πρέπει να επιστρέψει δείκτη με τιμή «Επικίνδυνη». Όπως φαίνεται στις εικόνες 3.2i και 3.2ii, το αποτέλεσμα είναι το αναμενόμενο. Παρ' όλο που οι τιμές των σωματιδίων διαφέρουν πάρα πολύ, οι τιμές των αερίων είναι ίδιες. Αυτό συμβαίνει, γιατί η σκόνη που παράγεται από τα κάρβουνα δεν παράγεται με τον ίδιο ρυθμό, αλλά μεταβάλλεται. Επίσης, εξ' αιτίας της χαόδους πορείας που ακολουθεί ο αέρας, κάποια κομμάτια σκόνης μπορεί να μετακινηθούν προς άλλες κατευθύνσεις. Συ-

3.2. ΣΥΓΚΡΙΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΠΛΗΡΟΥΣ ΣΥΣΤΗΜΑΤΟΣ

νεπώς, αυτή η διαφορά είναι φυσιολογική. Στην εικόνα 3.2i η ποιότητα του αέρα είναι η αναμενόμενη και προκύπτει εξ' αιτίας του bias που αναφέρθηκε πριν και το οποίο επαληθεύεται λόγω των διαφορών στις τιμές της εικόνας 3.2ii και της εικόνας 3.1iii.



η ώρα καταγραφής είναι απογευματίνη (η καταγραφή έγινε στην Ελλάδα, αλλά η ώρα που καταχωρείται είναι του Ισημερινού). Αυτό σημαίνει ότι δεν υπήρχε πολύ κίνηση εκείνη την στιγμή και άρα δεν υπήρχαν εξωτερικοί ρύποι από αυτοκίνητα. Επίσης, οι αυξημένες τιμές NO₂ οφείλονται στην ανθρώπινη δραστηριότητα, από την χρήση ηλεκτρικών συσκευών οι οποίες παράγουν αυτές τις ποσότητες. Αυτό επιβεβαιώνεται και από το γεγονός, ότι δεν παρατηρήθηκε αύξηση σε καμία άλλη τιμή συγκέντρωσης αερίων και σωματιδίων.

```

1  {
2    "TimeInstant": "2023-04-30T18:10:00Z",
3    "period": 5,
4    "status": "connected",
5    "no2-a4": 94.830001831054688,
6    "ox-a431": 29.280000686645508,
7    "pm10": 15.006196975708008,
8    "pm2": 10.773552894592285,
9    "pm1": 7.3032760620117188
10 }
```

Σχήμα 3.3: Αποτελέσματα μετρήσεων αισθητήρων Smart Spot.

Σε παρόμοιο πείραμα που έγινε σε εσωτερικό χώρο, με την χρήση του συστήματος παρατήρησης ποιότητας αέρα, που περιγράφηκε στην παρούσα εργασία, τα αποτελέσματα ήταν παρόμοια. Η βασική διαφορά είναι οι μονάδες μέτρησης της συγκέντρωσης των αερίων, όπου, όπως αναφέρθηκε και στο κεφάλαιο 2.1.1, οι μονάδες μέτρησης της συγκέντρωσης για τους αισθητήρες του συστήματος είναι ppm, ενώ οι αισθητήρες Smart Spot μετράνε την συγκέντρωση σε $\mu g/m^3$. Επίσης, μια επιπλέον διαφορά ήταν στην ακρίβεια των μετρήσεων. Συγκεκριμένα, όπως αναφέρθηκε στο ίδιο κεφάλαιο, η ακρίβεια των αισθητήρων είναι της τάξης ενός δεκαδικού ψηφίου και δεν μπορεί να καταγράψει πόσο «καλή» είναι η ποιότητα του αέρα. Τέλος, οι αισθητήρες Smart Spot δεν συμπεριελάμβαναν στις μετρήσεις όλες τις συγκεντρώσεις των αερίων, λόγω δυσκολίας στην παραμετροποίηση και κατανόηση του τρόπου λειτουργίας τους, καθώς η ιστοσελίδα ήταν στα ισπανικά και δεν υπήρχε μέχρι εκείνη την ημερομηνία κάποια αγγλική έκδοση.

```

{
  metadata: {
    sensorId: 1,
    position: {
      longitude: 18.5,
      latitude: 51.2
    }
  },
  timestamp: '2023-06-23T20:59:00.840649Z',
  Emissions: {
    CO: 0,
    SO2: 0,
    O3: 0.0,
    NO2: 0.0,
    'PM1.0': 6,
    'PM2.5': 8,
    'PM10.0': 15
  }
}
```

```

{
  sensorId: 1,
  position: {
    longitude: 18.5,
    latitude: 51.2
  },
  AQI: 'Good',
  timestamp: '2023-06-23T20:59:00.842415Z'
}
```

(i)
(ii)

Σχήμα 3.4: Αποτελέσματα Κόμβων σε φυσιολογικές συνθήκες.

Συμπεράσματα-Συζήτηση

Τα αποτελέσματα του πειράματος έδειξαν ότι το συνολικό σύστημα λειτουργεί ορθά και είναι ικανό να διαχειριστεί πληροφορία σε πραγματικό χρόνο. Τα δεδομένα από τον κομβό που χρησιμοποιήθηκαν καταχωρήθηκαν στην βάση σε πραγματικό χρόνο, η οποία βρίσκεται σε ένα απομακρυσμένο σύστημα VPS. Η βάση και η εφαρμογή μπορούν να χρησιμοποιηθούν από πελάτες ταυτόχρονα με την επικοινωνία των κόμβων και την εισαγωγή νέων δεδομένων. Οι κόμβοι λαμβάνουν πραγματικές τιμές, οι οποίες είναι οι αναμενόμενες και αποστέλλονται στην βάση χωρίς προβλήματα. Τα endpoints ανακαλούν την πληροφορία από την βάση και την στέλνουν στον χρήστη που τα καλεί. Συνεπώς, το σύστημα είναι λειτουργικά αξιόπιστο. Σημαντικό ζήτημα, όμως, που πρέπει να μελετηθεί είναι η ικανότητα και η αξιοπιστία του συστήματος όσον αφορά την σύνδεση πολλαπλών τόσο κόμβων όσο και χρηστών ταυτόχρονα. Οι δυνατότητα της εφαρμογής, της βάσης ή του server να ανταποκριθούν σε πολλαπλά αιτήματα ταυτόχρονα, χωρίς την δημιουργία προβλημάτων, όπως υπερφόρτωση δικτύου, είναι ένα ζήτημα που χρειάζεται περισσότερα πειράματα, τα οποία θα πρέπει να γίνουν και υπό πραγματικές συνθήκες, όχι δηλαδή σε προστατευμένο περιβάλλον, όπου, για παράδειγμα, η φωτιά μένει σταθερή αλλά σε ένα περιβάλλον όπου οι συνθήκες αλλάζουν (η πυρκαγιά εξαπλώνεται ή συρρικνώνεται, η λάβα από το ηφαίστειο κινείται και είναι ακόμη ζεστή κ.λπ.), με την χρήση της εφαρμογής από πολλά άτομα και υπηρεσίες. Στους μελλοντικούς στόχους της εργασίας έγκειται η πραγματοποίηση πειραμάτων, τα οποία θα πληρούν τις παραπάνω προδιαγραφές.

Όσον αφορά τους αισθητήρες Smart Spot, οι συγκεκριμένοι αισθητήρες έχουν πολύ μεγαλύτερη ακρίβεια στις τιμές των συγκεντρώσεων, καθώς και αρκετές παραμέτρους τις οποίες μπορούν να επιλέξουν (περίοδο συλλογής, χρονοσφραγίδα, αέρια και σωματίδια που θα μετρηθούν, τοποθεσία κ.λπ.). Βέβαια, αυτό είναι αναμενόμενο, καθώς προορίζονται για εμπορική χρήση. Παρ' όλα αυτά, το αποτέλεσμα του βασικού σκοπού τους δεν διαφέρει από αυτό ενός κόμβου, όπως αναπτύχθηκε στην παρούσα εργασία. Κατά συνέπεια, μπορούμε να πούμε πως οι κόμβοι που αναπτύχθηκαν στην συγκεκριμένη πτυχιακή εργασία έχουν παραπλήσιες επιδόσεις με αισθητήρες διαθέσιμους στο εμπόριο. Βέβαια, όπως αναφέρθηκε προηγουμένως, χρειάζεται να πραγματοποιηθούν επιπλέον πειράματα, ώστε να εντοπιστούν πιθανά προβλήματα που μπορεί να μην ήταν δυνατόν να εντοπιστούν, καθώς και να γίνουν έλεγχοι σύγκρισης απόδοσης των δύο σε διάφορα περιβάλλοντα, ώστε να εξασφαλισθεί ότι οι κόμβοι δεν διαφέρουν σημαντικά σε απόδοση από εμπορικούς αισθητήρες.

Bibliography

- [1] Rakesh Vidya Chandra and Bala Subrahmanyam Varanasi. *Python requests essentials: learn how to integrate your applications seamlessly with web services using Python requests*. OCLC: 915143465. Birmingham, UK: Packt Publishing, 2015. isbn: 978-1-78439-541-4.
- [2] Davy Cielen, Arno Meysman, and Mohamed Ali. *Introducing data science: big data, machine learning, and more, using Python tools*. OCLC: ocn910773263. Shelter Island, NY: Manning Publications, 2016. 300 pp. isbn: 978-1-63343-003-7.
- [3] Dasari Hermitha Curie et al. «Analysis on Web Frameworks». In: *Journal of Physics: Conference Series* 1362.1 (Nov. 1, 2019), p. 012114. issn: 1742-6588, 1742-6596. doi: 10.1088/1742-6596/1362/1/012114. url: <https://iopscience.iop.org/article/10.1088/1742-6596/1362/1/012114> (visited on 05/25/2023).
- [4] *Directive 2008/50/EC of the European Parliament and of the Council of 21 May 2008 on ambient air quality and cleaner air for Europe*. May 21, 2008. url: <http://data.europa.eu/eli/dir/2008/50/oj/eng> (visited on 06/13/2023).
- [5] Roy T. Fielding et al. «Reflections on the REST architectural style and "principled design of the modern web architecture" (impact paper award)». In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE'17: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. Paderborn Germany: ACM, Aug. 21, 2017, pp. 4–14. isbn: 978-1-4503-5105-8. doi: 10.1145/3106237.3121282. url: <https://dl.acm.org/doi/10.1145/3106237.3121282> (visited on 05/24/2023).
- [6] Roy Thomas Fielding. «Architectural Styles and the Design of Network-based Software Architectures». Doctoral dissertation. Irvine: University of California, 2000.
- [7] Thad Godish. *Air quality*. 4th ed. Boca Raton: Lewis Publishers, 2004. 460 pp. isbn: 978-1-56670-586-8.
- [8] Miguel Grinberg. *Flask web development: developing web applications with Python*. 2nd edition. OCLC: on1030283575. Sebastopol, California: O'Reilly, 2018. 291 pp. isbn: 978-1-4919-9173-2.
- [9] *HTTP authentication - HTTP | MDN*. Apr. 10, 2023. url: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication> (visited on 06/13/2023).
- [10] Randall Hyde. *The book of PC: a guide for adventurers*. San Francisco: No Starch Press, 2023. 409 pp. isbn: 978-1-71850-246-8.
- [11] Rob J. Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. 2nd edition. Lexington, Ky.: Otexts, online, open-access textbook, 2018. 382 pp. isbn: 978-0-9875071-1-2.

- [12] Kevin Ashton. «That'Internet of Things' Thing». In: *RFID Journal* 22 (2009), p. 1.
- [13] Marek Kubika. *HOWTO Use Python in the web*. Sept. 28, 2010.
- [14] Dirk Merkel. «Docker: Lightweight Linux Containers for Consistent Development and Deployment». In: *Linux J.* 2014.239 (Nov. 2014). Place: Houston, TX Publisher: Belltown Media. issn: 1075-3583.
- [15] Clément Nedelcu. *Nginx HTTP Server: adopt Nginx for your web applications to make the most of your infrastructure and serve pages faster than ever*. 2. Aufl. Birmingham: Packt Publ, 2013. 327 pp. isbn: 978-1-78216-232-2.
- [16] *Overview*. Postman Learning Center. url: <https://learning.postman.com/docs/introduction/overview/> (visited on 06/24/2023).
- [17] *PEP 3333 – Python Web Server Gateway Interface v1.0.1* | [peps.python.org](https://peps.python.org/pep-3333/). url: <https://peps.python.org/pep-3333/> (visited on 06/13/2023).
- [18] *pip documentation v23.1.2*. url: <https://pip.pypa.io/en/stable/> (visited on 06/13/2023).
- [19] *Raspberry Pi GPIO Pinout*. url: <https://pinout.xyz/> (visited on 06/20/2023).
- [20] *Reference Guide — virtualenv 16.7.9 documentation*. url: <https://virtualenv.pypa.io/en/16.7.9/reference.html> (visited on 06/20/2023).
- [21] Kunal Relan. *Building REST APIs with Flask: create Python web services with MySQL*. OCLC: 1120770697. New York: Apress, 2019. isbn: 978-1-4842-5022-8.
- [22] Hannah Ritchie, Pablo Rosado, and Max Roser. «Natural Disasters». In: *Our World in Data* (Dec. 7, 2022). url: <https://ourworldindata.org/natural-disasters> (visited on 06/22/2023).
- [23] School of Computing Universiti Utara Malaysia Kedah, Malaysia and Haroon Shakirat Oluwatosin. «Client-Server Model». In: *IOSR Journal of Computer Engineering* 16.1 (2014), pp. 57–71. issn: 22788727, 22780661. doi: 10.9790/0661-16195771. url: <http://www.iosrjournals.org/iosr-jce/papers/Vol16-issue1/Version-9/J016195771.pdf> (visited on 05/24/2023).
- [24] S. E. Schwartz and P. Warneck. «Units for use in atmospheric chemistry (IUPAC Recommendations 1995)». In: *Pure and Applied Chemistry* 67.8 (Jan. 1, 1995), pp. 1377–1406. issn: 1365-3075, 0033-4545. doi: 10.1351/pac199567081377. url: <https://www.degruyter.com/document/doi/10.1351/pac199567081377/html> (visited on 05/24/2023).
- [25] *Smart Spot - IoT device for environmental parameter sensing*. Marketplace EIT Urban Mobility. url: <https://marketplace.eiturbanmobility.eu/products/hopu-smart-spot/> (visited on 06/13/2023).
- [26] ANDREW S. TANEMBAUM. *DISTRIBUTED SYSTEMS*. OCLC: 1373659118. S.l.: MAARTEN VAN STEEN, 2023. isbn: 978-90-815406-3-6.
- [27] Andrew S. Tanenbaum. *Computer networks*. 4th ed. Upper Saddle River, NJ: Prentice Hall PTR, 2003. 891 pp. isbn: 978-0-13-066102-9.
- [28] Eben Upton and Gareth Halfacree. *Raspberry Pi user guide*. 4th edition. OCLC: ocn944380277. Chichester, West Sussex, UK: John Wiley & Sons, 2016. 293 pp. isbn: 978-1-119-26436-1.

- [29] Guido Van Rossum. *The Python Language Reference*.
- [30] Iwan Vosloo and Derrick G. Kourie. «Server-centric Web frameworks: An overview». In: *ACM Computing Surveys* 40.2 (Apr. 2008), pp. 1–33. issn: 0360-0300, 1557-7341. doi: 10.1145/1348246.1348247. url: <https://dl.acm.org/doi/10.1145/1348246.1348247> (visited on 05/25/2023).
- [31] Werkzeug — Werkzeug Documentation (2.3.x). url: <https://werkzeug.palletsprojects.com/en/2.3.x/> (visited on 06/13/2023).
- [32] Jon Westfall. *Set up and manage your virtual private server: making system administration accessible to professionals*. New York, NY: Apress, 2021. 243 pp. isbn: 978-1-4842-6965-7. doi: 10.1007/978-1-4842-6966-4.
- [33] What is a web server? - Learn web development | MDN. Feb. 23, 2023. url: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server (visited on 06/13/2023).
- [34] What is Postman? Postman API Platform. Postman API Platform. url: <https://www.postman.com/product/what-is-postman/> (visited on 06/24/2023).
- [35] Joseph M. Willi, Gavin P. Horn, and Daniel Madrzykowski. «Characterizing a Firefighter’s Immediate Thermal Environment in Live-Fire Training Scenarios». In: *Fire Technology* 52.6 (Nov. 2016), pp. 1667–1696. issn: 0015-2684, 1572-8099. doi: 10.1007/s10694-015-0555-1. url: <http://link.springer.com/10.1007/s10694-015-0555-1> (visited on 05/24/2023).
- [36] Yu Yan Jiang et al. «An integrated numerical simulator for thermal performance assessments of firefighters’ protective clothing». In: *Fire Safety Journal* 45.5 (Aug. 2010), pp. 314–326. issn: 03797112. doi: 10.1016/j.firesaf.2010.06.003. url: <https://linkinghub.elsevier.com/retrieve/pii/S0379711210000500> (visited on 05/24/2023).
- [37] Yifeng Zhu. *Embedded systems with arm cortex-m microcontrollers in assembly language and c*. 3rd edition. Ballston Spa, NY: E-Man Press LLC, 2017. isbn: 978-0-9826926-6-0.
- [38] Απόστολος Ν. Παπαδόπουλος et al. *Συστήματα Βάσεων Δεδομένων Θεωρία & Πρακτική Εφαρμογή*. 2η έκδοση. Ελλάδα: NewTechPub, 2020. 670 pp. isbn: 978-960-578-070-8.
- [39] Μανής, Γεώργιος. *Εισαγωγή στον Προγραμματισμό με Αρωγό τη Γλώσσα Python*. Κάλλιπος. Αθήνα: Ανοικτές Ακαδημαϊκές Εκδόσεις, 2015. 200 pp. isbn: 978-960-603-415-2. url: <https://hdl.handle.net/11419/2745>.
- [40] Νικόλαος Αβούρης et al. *Python Εισαγωγή στους Υπολογιστές*. 4η. Ηράκλειο, Ελλάδα: Πανεπιστημιακές Εκδόσεις Κρήτης, 2021. 543 pp. isbn: 978-960-524-529-0.
- [41] Παναγιώτης Παπάζογλου. *Μικροεπεξεργαστές Αρχές & Εφαρμογές*. 1η Έκδοση. Ελλάδα: Εκδόσεις Τζιόλα, 2018. 823 pp. isbn: 978-960-418-508-5.
- [42] . url: https://www.wwf.gr/ti_kanoume/fysh/dasi/pyrkagies/ (visited on 06/22/2023).
- [43] *Σύνολα Δεδομένων - Πυροσβεστικό Σώμα Ελλάδος*. url: <https://www.fireservice.gr/el/synola-dedomenon> (visited on 06/22/2023).

