



PYRO CHECK

Στα πλαίσια του μαθήματος

Διαδίκτυο των Πραγμάτων

Created By

Ηλίας Σαλτσιδης AM2022149, Γιώργος Ρετσιλας AM2022143



Contents

Περιγραφής εφαρμογής	2
Βιβλιογραφική Ανασκόπηση	2
Λειτουργικές & Τεχνικές Απαιτήσεις.....	3
Τεχνικές Απαιτήσεις	3
Λειτουργικές απαιτήσεις	4
Υλοποίηση & Παρουσίαση Επιπέδων	6
Κριτική Αξιολόγηση	9
Συμπεράσματα	10

Περιγραφής εφαρμογής

Το παρόν κείμενο είναι μια τεχνική αναφορά ενός ολοκληρωμένου συστήματος πυρασφάλειας βασισμένο στο IOT. Στόχος του είναι η έγκυρη αλλά και έγκαιρη πληροφόρηση σχετικά με εκδηλώσεις πυρκαγιάς αλλά και συγκέντρωση επικίνδυνων και εύφλεκτων αέριων στον χώρο

Το σύστημα δεν περιορίζεται μόνο στην ανίχνευση κινδύνου αλλά προσφέρει μια συνεχή ροή δεδομένων όπως είναι η θερμοκρασία, η υγρασία και αν υπάρχει καπνός ή άλλο είδος αερίου στον χώρο ώστε να συγκεντρώνονται και να απεικονίζονται στον χρήστη με διάφορες μορφές. Αποτελείται από δυο βασικούς αισθητήρες, έναν για την θερμοκρασία-υγρασία και έναν για τα αέρια του χώρου.

Το σύστημα σχεδιάστηκε ώστε να μπορεί να χρησιμοποιηθεί και σε χώρους με πολλαπλά δωμάτια είτε αυτοί είναι οικιακοί είτε επαγγελματικοί. Ο έλεγχος του συστήματος είναι πάρα πολύ εύκολος και μπορεί να γίνει και εκτός του χώρου που βρίσκεται η κεντρική μονάδα, καθιστώντας το αρκετά ευέλικτο και εύκολο στην χρήση

Βιβλιογραφική Ανασκόπηση

Υπάρχουν πάρα πολλά συστήματα πυρασφάλειας στην αγορά αυτήν την στιγμή, είτε αυτά είναι ολοκληρωμένα συστήματα με πολλούς αισθητήρες και κεντρικό σύστημα διαχείρισης που συνδέονται σε συστήματα κατάσβεσης, είτε αυτά είναι μεμονωμένα και συνδέοντα ξεχωριστά σε μια εφαρμογή και λειτουργούν όλα σαν ανεξάρτητες μονάδες. Επιπρόσθετα υπάρχουν και πολλές DIY εφαρμογές που κάνουν αυτήν την δουλειά εξίσου καλά ενσωματώνοντας πολλές φορές τεχνολογίες που μπορεί να τα καθιστά και καλύτερα από τις λύσεις που υπάρχουν στην αγορά αυτή την στιγμή.

Το σύστημα το οποίο έχουμε αναπτύξει δεν έχει σε τίποτα να ζηλέψει με τις λύσεις που υπάρχουν αυτήν την στιγμή στην αγορά διότι μπορεί να λειτουργήσει σαν μια αυτόνομη μονάδα πυρασφάλειας με ένα κεντρικό dashboard όπου μπορεί να γίνει η διαχείριση όλου του συστήματος από οπουδήποτε. Τα δεδομένα οπτικοποιούνται με τέτοιο τρόπο ώστε να επιτρέπουν στον χρήστη να



κατανοεί άμεσα την πληροφορία που προβάλλεται και να εξάγει συμπεράσματα μέσα σε δευτερόλεπτα. Σε περίπτωση ανίχνευσης κινδύνου, παρέχεται άμεση ειδοποίηση μέσω SMS και email, ώστε να εξασφαλίζεται έγκαιρη αντιμετώπιση χωρίς καθυστερήσεις που θα μπορούσαν να οδηγήσουν σε δυσάρεστες καταστάσεις.

Χρησιμοποιήθηκαν όσο το δυνατό νεότερες τεχνολογίες καθιστώντας το σύστημα αποδοτικό, αξιόπιστο και μελλοντικά επεκτάσιμο. Τα πρωτοκόλλα επιλέχθηκαν με γνώμονα την ευκολία αλλά και αποδοτικότητα που προσφέρουν, η συσκευή υλοποίησης επιλέχθηκε για τις τεχνολογίες και την ταχύτητα που προσφέρει. Οι γλώσσες προγραμματισμού που χρησιμοποιήθηκαν επιλέχθηκαν ώστε να εξασφαλίζουν σταθερή και χωρίς διακοπές επικοινωνία, καθώς και τη μέγιστη δυνατή συνδεσιμότητα.

Λειτουργικές & Τεχνικές Απαιτήσεις

Οι απαιτήσεις του συστήματος χωρίζονται σε δυο πολύ βασικές κατηγορίες έχουμε τις τεχνικές όπου περιγράφεται αναλυτικά το τι χρησιμοποιήθηκε για την ανάπτυξη της εφαρμογής και τις λειτουργικές όπου είναι το τι αναμένουμε από το σύστημα να κάνει κατά την διάρκεια της λειτουργία του.

Τεχνικές Απαιτήσεις

Το σύστημα αποτελείται από δύο βασικές κατηγορίες: το software και το hardware.

Στο Software/hardware επίπεδο του embedded συστήματος χρησιμοποιούμε ως κεντρική πλακέτα έναν ESP32, ο οποίος λειτουργεί ως το main hub του συστήματος. Ο ESP32 αξιοποιεί πρωτόκολλα Wi-Fi και σε αυτόν συνδέονται όλοι οι αισθητήρες του project.

Πιο συγκεκριμένα, χρησιμοποιούνται οι εξής αισθητήρες:

- AHT20 + BMP280, οι οποίοι συλλέγουν δεδομένα θερμοκρασίας, υγρασίας και ατμοσφαιρικής πίεσης, στέλνοντας τις μετρήσεις μέσω του πρωτοκόλλου I2C.
- MQ2, ο οποίος χρησιμοποιείται για την ανίχνευση καπνού και αερίων στον χώρο.

Για το firmware του συστήματος χρησιμοποιήθηκε το PlatformIO σε συνδυασμό με το Arduino Framework το οποίο αποτέλεσε τη βασική δομή ανάπτυξης του project.

Επιπλέον, αξιοποιήθηκαν οι παρακάτω βιβλιοθήκες:

- U8g2: Υπεύθυνη για τη διαχείριση και απεικόνιση δεδομένων στην οθόνη του ESP32.
- ArduinoJson: Χρησιμοποιήθηκε για τη μετατροπή και μεταφορά των μετρήσεων σε μορφή JSON.
- PubSubClient: Βιβλιοθήκη υπεύθυνη για την αποστολή δεδομένων μέσω του πρωτοκόλλου MQTT, επιτρέποντας στον ESP32 να κάνει Publish και Subscribe μηνύματα σε MQTT Broker.
- Adafruit AHTX0: Βιβλιοθήκη για την επικοινωνία με τον αισθητήρα AHT20 και τη μέτρηση θερμοκρασίας/υγρασίας υψηλής ακρίβειας.
- BMP280 Library: Χρησιμοποιείται για την ανάγνωση δεδομένων από τον αισθητήρα BMP280, ο οποίος μετρά ατμοσφαιρική πίεση και θερμοκρασία.
- LittleFS: Σύστημα αρχείων στην flash μνήμη του ESP32, το οποίο χρησιμοποιείται για αποθήκευση ρυθμίσεων, όπως Wi-Fi credentials, ώστε να παραμένουν αποθηκευμένα μετά από restart.

Στο backend, η ανάπτυξη πραγματοποιήθηκε με Python και πιο συγκεκριμένα χρησιμοποιήθηκε το framework FastAPI, το οποίο είναι υπεύθυνο για τη συνολική διαχείριση του συστήματος και την



υλοποίηση του server-side logic. Μέσω αυτού διαχειρίζονται τα API endpoints και η επικοινωνία με τη βάση δεδομένων.

Για τη βάση δεδομένων χρησιμοποιήθηκε η PostgreSQL, η οποία αποθηκεύει τις μετρήσεις των αισθητήρων, τα δεδομένα χρηστών, τις ρυθμίσεις του συστήματος και το ιστορικό ειδοποιήσεων/alerts.

Η επικοινωνία μεταξύ backend και ESP32 υλοποιήθηκε κυρίως μέσω MQTT Broker, χρησιμοποιώντας το HiveMQ Cloud, το οποίο επιτρέπει αξιόπιστη ανταλλαγή δεδομένων πραγματικού χρόνου μεταξύ των IoT συσκευών και του server.

Παράλληλα, χρησιμοποιήθηκαν και HTTP Requests / Responses για τις αρχικές ρυθμίσεις, το setup της συσκευής, καθώς και για λειτουργίες configuration που δεν απαιτούν συνεχή real-time σύνδεση.

Η επικοινωνία μεταξύ backend και frontend πραγματοποιείται μέσω HTTP Requests / Responses για την κλασική ανταλλαγή δεδομένων και μέσω WebSockets για την αποστολή live δεδομένων και real-time updates, όπως αισθητήρες και alerts, χωρίς ανάγκη συνεχών refresh από τον client.

Για το σύστημα ειδοποιήσεων χρησιμοποιήθηκαν οι εξής υπηρεσίες:

- SMTP για αποστολή ειδοποιήσεων μέσω email.
- Twilio API για αποστολή SMS alerts σε περιπτώσεις κρίσιμων συμβάντων ή συναγερμών.

Στο frontend τώρα η ανάπτυξη έγινε με το framework Next.js, το οποίο χρησιμοποιήθηκε για την κατασκευή του web interface της εφαρμογής.

Για το styling χρησιμοποιήθηκε το Tailwind CSS, το οποίο βοήθησε στη γρήγορη ανάπτυξη και οργάνωση του UI.

Η επικοινωνία για τα HTTP requests έγινε μέσω του Axios, το οποίο χρησιμοποιήθηκε για την ανταλλαγή δεδομένων με το backend.

Για τα live data χρησιμοποιήθηκαν WebSockets, ώστε να υπάρχει real-time ενημέρωση των δεδομένων και των alerts στο frontend.

Λειτουργικές απαιτήσεις

Το σύστημα διαθέτει αρκετές λειτουργικές απαιτήσεις, όπου οι περισσότερες αφορούν το firmware, καθώς εκεί πραγματοποιούνται οι βασικοί έλεγχοι και η κύρια λογική λειτουργίας του hub. Πιο συγκεκριμένα:

1. Το hub κατά το boot πρέπει να αρχικοποιεί το OLED display, το Wi-Fi Manager, να διαβάσει το claimed state από το NVS και να εμφανίζει την κατάσταση εκκίνησης (main.cpp, display_manager.cpp, backend_manager.cpp).
2. Πρέπει να συνδέεται σε αποθηκευμένο Wi-Fi, διαφορετικά να ανοίγει captive portal με όνομα Pyrocheck για provisioning (wifi_manager.cpp).
3. Το portal πρέπει να υποστηρίζει Wi-Fi scan, αποθήκευση SSID/password, reset credentials και restart της συσκευής (wifi_manager.cpp, portal.html).
4. Πρέπει να επιχειρεί registration στο backend κάθε 5 δευτερόλεπτα μέχρι αυτό να γίνει με επιτυχία και να συγχρονίζει is_claimed / user_id (main.cpp, backend_manager.cpp).
5. Πρέπει να συνδέεται σε MQTT over TLS, να στέλνει retained heartbeat isonline:true και να ορίζει LWT retained isonline:false (MqttManager.cpp).
6. Πρέπει να κάνει subscribe σε device config / commands topics αναμένοντας εντολές και να εφαρμόζει remote configuration live (MqttManager.cpp).



7. Πρέπει να δέχεται SET_CONFIG και να μηδενίζει το state όταν είναι UNCLAIMED ή λείπει valid user_id, διαδικασία που αφορά το setup των sensors (MqttManager.cpp).
8. Πρέπει να υποστηρίζει δυναμικό sensor configuration, μέχρι 4 AHT20-BMP280 και 5 MQ2 αισθητήρες (MqttManager.h).
9. Πρέπει να απορρίπτει μη επιτρεπτά pins:
 - MQ2 μόνο στα {32,33,34,35,36}
 - AHT/BMP μόνο σε επιτρεπτά I2C pin pairs (MqttManager.h, MqttManager.cpp).
10. Πρέπει να διαβάζει τους αισθητήρες περιοδικά βάσει του interval_ms από το config και να δημοσιεύει τις μετρήσεις στο topic pyrocheck/sensors/{mac}/{metric}/{sensor_id} (main.cpp).
11. Πρέπει να πραγματοποιεί I2C bus switch / re-init ανά sensor pair και retry initialization αισθητήρων κάθε 5 δευτερόλεπτα (sensors_manager.cpp).
12. Ο MQ2 πρέπει να έχει warmup 10s, sample averaging, noise filtering και threshold logic για smoke/gas detection (sensors_manager.cpp, main.cpp).
13. Πρέπει να παράγει alerts μέσω MQTT για temp/hum/mq2 με payload που περιέχει type, metric, sensor_id, value, threshold, ts (main.cpp).
14. Πρέπει να αποθηκεύει το claim state στο NVS ώστε να διατηρείται μετά από reboot (backend_manager.cpp).

Η εφαρμογή frontend πρέπει να καλύπτει τις παρακάτω λειτουργικές απαιτήσεις:

1. Η εφαρμογή πρέπει να επιτρέπει εγγραφή χρήστη με username, email και password (register-form.tsx, auth.ts).
2. Η εφαρμογή πρέπει να επιτρέπει σύνδεση χρήστη και να χρησιμοποιεί cookie session (login-form.tsx, auth.ts).
3. Η εφαρμογή πρέπει να προστατεύει τα Dashboard routes ώστε να είναι προσβάσιμα μόνο από authenticated χρήστες (middleware.ts).
4. Η εφαρμογή πρέπει να επιτρέπει claim συσκευής με validation MAC address (claim-device-form.tsx, hub-form.ts).
5. Η εφαρμογή πρέπει να εμφανίζει τα hubs του χρήστη (devices/page.tsx, hub.ts).
6. Η εφαρμογή πρέπει να εμφανίζει live status και live μετρήσεις μέσω WebSocket (ws-client.ts, dashboard-ws-bridge.tsx).
7. Η εφαρμογή πρέπει να επιτρέπει προσθήκη αισθητήρα με τύπο, pins και threshold (AddSensorDialog.tsx).
8. Η εφαρμογή πρέπει να επιτρέπει επεξεργασία αισθητήρα (AddSensorDialog.tsx, hub.ts).
9. Η εφαρμογή πρέπει να επιτρέπει αφαίρεση αισθητήρα με soft/hard delete (devices/page.tsx, hub.ts).
10. Η εφαρμογή πρέπει να επιτρέπει unclaim hub με confirmation (device-hub-card.tsx, hub.ts).
11. Η εφαρμογή πρέπει να εμφανίζει ιστορικό και summary δεδομένων ανά hub/sensor (history/page.tsx, history-utils.ts).
12. Η εφαρμογή πρέπει να εμφανίζει alerts στο history tab (history-alerts-tab.tsx, history/page.tsx).

Το backend/API πρέπει να καλύπτει τις παρακάτω λειτουργικές απαιτήσεις:

1. Το API πρέπει να υποστηρίζει auth endpoints για register/login/logout/verify (auth.py).
2. Το API πρέπει να δημιουργεί hub κατά το register από MAC του firmware (hub.py).
3. Το API πρέπει να υποστηρίζει claim hub και να κάνει publish config στο MQTT (hub.py, mqtt_manager.py).
4. Το API πρέπει να υποστηρίζει unclaim hub και καθαρισμό retained config (hub.py).



5. Το API πρέπει να υποστηρίζει add/edit/remove sensors με ownership checks (hub.py, schemas.py).
6. Το API πρέπει να παρέχει endpoints για my-hubs και sensors/{mac} (hub.py).
7. Το API πρέπει να κάνει ingest telemetry από MQTT topics και αποθήκευση SensorData (mqtt_manager.py, models.py).
8. Το API πρέπει να ενημερώνει hub status online/offline μέσω LWT/heartbeat (mqtt_manager.py, models.py).
9. Το API πρέπει να δημιουργεί και να αποθηκεύει alerts με severity (mqtt_manager.py, models.py).
10. Το API πρέπει να παρέχει Alerts API ανά hub (hub.py).
11. Το API πρέπει να παρέχει History API για raw και aggregated δεδομένα 15 ημερών (hub.py).
12. Το API πρέπει να υποστηρίζει WebSocket broadcast live events προς frontend (websocket.py, ws_manager.py).
13. Το API πρέπει να υποστηρίζει optional email/SMS notifications σε alerts (mqtt_manager.py).
14. Το backend πρέπει να κάνει application bootstrap, CORS setup, route registration και MQTT startup (server.py).

Υλοποίηση & Παρουσίαση Επιπέδων

Η αρχιτεκτονική που χρησιμοποιήθηκε στο σύστημα είναι παρόμοια με αυτή που χρησιμοποιούν εμπορικά συστήματα, καθιστώντας έτσι την εφαρμογή μας μια ολοκληρωμένη λύση.

Με το που συνδεθεί το ESP στο ρεύμα, ξεκινούν όλες οι διαδικασίες. Αρχικά προσπαθεί να συνδεθεί σε αποθηκευμένο δίκτυο WiFi· αν υπάρχει, κάνει μερικές προσπάθειες σύνδεσης. Αν δεν τα καταφέρει ή δεν υπάρχει αποθηκευμένο δίκτυο, ενεργοποιεί ένα captive portal, δηλαδή δημιουργεί το δικό του τοπικό WiFi. Ο χρήστης συνδέεται σε αυτό και μεταφέρεται σε ένα web interface, όπου επιλέγει το δίκτυό του και εισάγει τον κωδικό. Αν η σύνδεση είναι επιτυχής, το ESP λαμβάνει μια IP και προχωρά στο επόμενο στάδιο.

Στη συνέχεια, στέλνει request στο backend για να ελέγξει αν είναι καταχωρημένο σε κάποιον χρήστη. Αν δεν είναι, παραμένει σε αναμονή μέχρι να γίνει η καταχώρηση. Αν είναι ήδη καταχωρημένο, προχωρά και περιμένει να συνδεθεί σε MQTT broker. Μόλις συνδεθεί, στέλνει ένα heartbeat μήνυμα της μορφής: {"isonline": true, "ts": 11499} που δηλώνει ότι η συσκευή είναι online.

Έπειτα, ζητά από το backend το αρχείο ρυθμίσεων (config), στέλνοντας: {"mac_address": "4C:11:AE:AD:CF:7C"} Αν δεν υπάρχει config, περιμένει τον χρήστη να καταχωρήσει αισθητήρες. Αν υπάρχει, το λαμβάνει και ξεκινά την ανάγνωση των αισθητήρων.

Η εφαρμογή υποστηρίζει δύο τύπους αισθητήρων: τον AHT20 και τον BMP280, που βασίζονται στο πρωτόκολλο I2C, και τον MQ2, ο οποίος διαβάζεται αναλογικά. Στην περίπτωση πολλαπλών αισθητήρων I2C, το σύστημα εναλλάσσει σειριακά την ανάγνωσή τους, καθώς το I2C χρησιμοποιεί συγκεκριμένα pins, κάτι που προσθέτει μικρές καθυστερήσεις στο σύστημα.

Τα δεδομένα συλλέγονται κάθε 1 δευτερόλεπτο και αποστέλλονται κάθε 7 δευτερόλεπτα μέσω MQTT. Όταν ο χρήστης αρχικοποιήσει τους αισθητήρες, ορίζει τις τιμές ορίων για ειδοποιήσεις μόνο για τον αισθητήρα AHT20- BMP280, διότι ο mq2 έχει συγκεκριμένες τιμές για τις οποίες χτυπάει άρα δεν υπάρχει λόγος αρχικοποίησης. Έτσι, όταν κάποια μέτρηση ξεπεράσει το καθορισμένο όριο, το ESP στέλνει alert μήνυμα μέσω MQTT εκείνη την στιγμή και έπειτα αναλαμβάνει ο server.



Ο server είναι υλοποιημένος σε Python και αναλαμβάνει όλη την επικοινωνία και τη διαχείριση του συστήματος. Περιμένει από το ESP να στείλει αίτημα για register και απαντά μέσω HTTP. Επίσης, δέχεται αιτήματα μέσω MQTT για το config και μόλις τα λάβει, επιστρέφει τις κατάλληλες ρυθμίσεις.

Παράλληλα, ο server παρακολουθεί την κατάσταση της συσκευής, περιμένοντας heartbeat/status μηνύματα που δηλώνουν αν είναι online. Αν δεν λάβει τέτοιο μήνυμα για κάποιο χρονικό διάστημα, θεωρεί τη συσκευή offline.

Ο server δέχεται συνεχώς μετρήσεις μέσω MQTT στο topic `pyrocheck/sensors/`. Τα δεδομένα αυτά αποθηκεύονται και προωθούνται στο frontend. Όταν το ESP στείλει κάποιο alert, ο server το λαμβάνει, αποστέλλει email ειδοποίηση και το προωθεί επίσης στο frontend.

Για την απεικόνιση των live δεδομένων στο UI, χρησιμοποιείται συνεχής σύνδεση μέσω WebSocket, ώστε οι μετρήσεις να εμφανίζονται σε πραγματικό χρόνο. Επιπλέον, όταν ζητηθεί από το frontend, ο server επιστρέφει ιστορικά δεδομένα μετρήσεων καθώς και τα alerts.

Τέλος, υπάρχουν και APIs που χρησιμοποιούνται για τη διαχείριση των συσκευών, τη σύνδεση των χρηστών και γενικά για το authentication του συστήματος.

Όσον αφορά το UI τώρα το πρώτο πράγμα που βλέπει κάποιος με το που ανοίγει την ιστοσελίδα είναι το Login & Register Page (Figure 1)

Figure 1 Login

Figure 2 Add device



έπειτα με το που συνδεθεί τον πάει στο dashboard όπου εκεί έχει 4 επιλογές στην πρώτη (Figure 2) του έχει να προσθέσει συσκευή όπου εκεί με το mac address μπορεί να προσθέσει όσες συσκευές διαθέτει, στη δεύτερη (Figure 3) του έχει να δει τις συνδεδεμένες συσκευές και έπειτα να προσθέσει αισθητήρες αλλά και να τους διαχειριστεί.

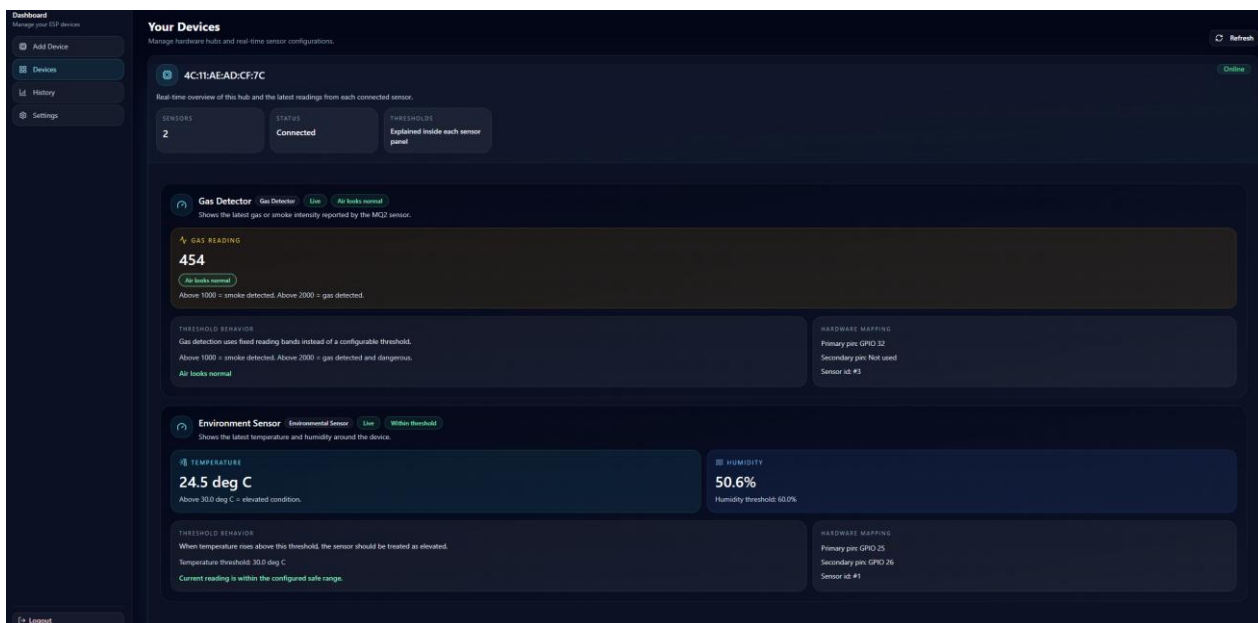


Figure 3 Devices

Στην τρίτη καρτέλα (Figure 4) έχουμε το ιστορικό όπου εκεί μπορεί να δει για της τελευταίες 15 ημέρες τα στατιστικά θερμοκρασίας υγρασίας αλλά και καπνού όπως επίσης και τα alerts.

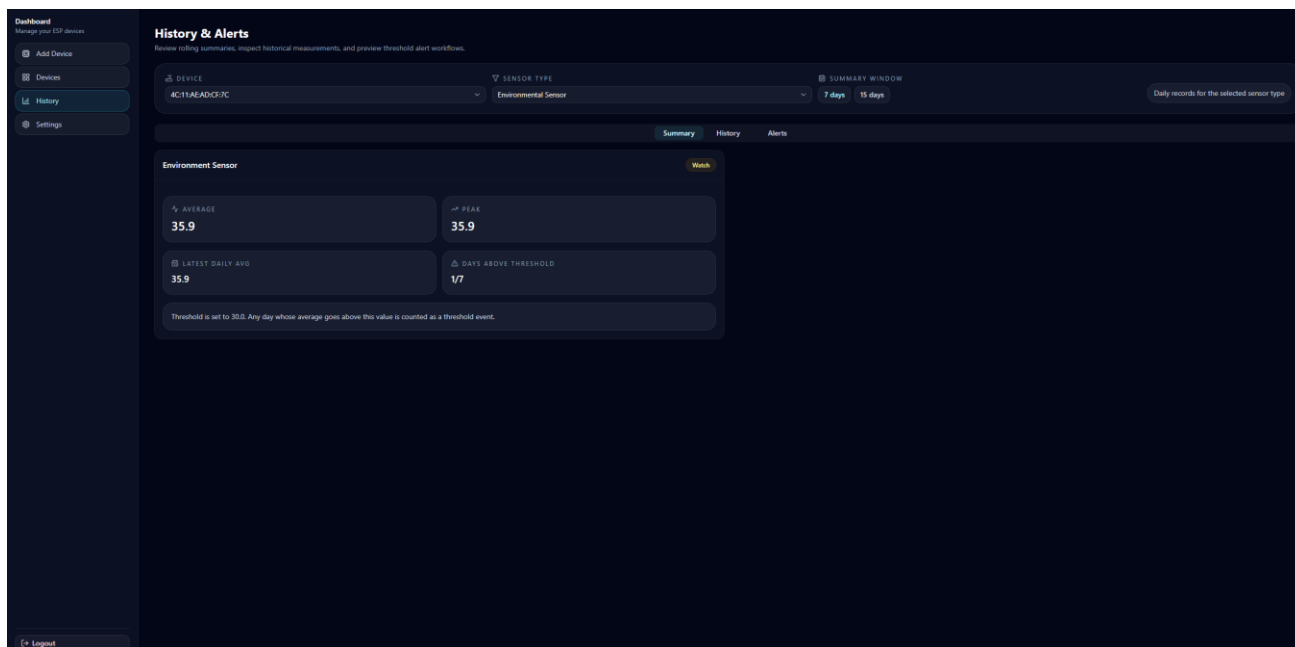


Figure 4 History

Και τέλος υπάρχει η καρτέλα των ρυθμίσεων (Figure 5) όπου εκεί δίνεται η επιλογή για αλλαγή ονόματος και τηλεφώνου.

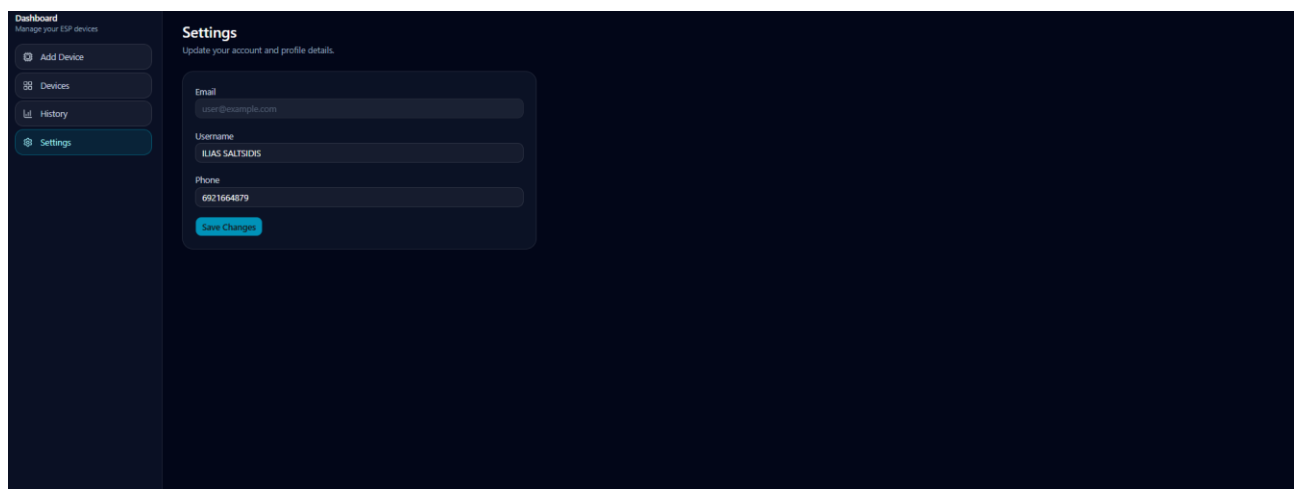


Figure 5 Settings

Κριτική Αξιολόγηση

Το σύστημα είναι σε θέση να αξιοποιηθεί και εμπορικά, καθώς όλες οι βασικές διαδικασίες έχουν αυτοματοποιηθεί. Το μόνο που απαιτείται από τον χρήστη είναι να προμηθευτεί μια συσκευή ESP με ήδη εγκατεστημένο το λογισμικό και στη συνέχεια να συνδέσει τους αντίστοιχους αισθητήρες.

Οι αισθητήρες τοποθετούνται στις κατάλληλες θύρες της συσκευής και ο χρήστης μπορεί να τις ορίσει και να τις παραμετροποιήσει μέσω του web interface. Από εκεί και πέρα, η συσκευή αναλαμβάνει αυτόματα τη σύνδεση, τη λήψη ρυθμίσεων και την αποστολή δεδομένων, χωρίς να απαιτείται περαιτέρω τεχνική παρέμβαση.

Η ανάπτυξη του συστήματος είναι πολύ σημαντική και υπάρχουν αρκετές δυνατότητες επέκτασης. Πέρα από την προσθήκη περισσότερων αισθητήρων — κάτι που ήδη υποστηρίζεται — μια επιπλέον ιδέα είναι η χρήση πολλαπλών ESP ως απομακρυσμένων αισθητήρων. Σε αυτή την προσέγγιση, κάθε ESP θα λειτουργεί ως ανεξάρτητη μονάδα συλλογής δεδομένων και θα συνδέεται σε ένα κεντρικό hub. Με αυτόν τον τρόπο, θα μπορούμε να καλύψουμε και να παρακολουθούμε πολύ μεγαλύτερους χώρους, χωρίς τους περιορισμούς μιας μόνο συσκευής. Μια τέτοια υλοποίηση θα επεκτείνει σημαντικά την αρχιτεκτονική του συστήματος και θα το καταστήσει κατάλληλο όχι μόνο για μικρές εγκαταστάσεις, αλλά και για μεγάλους χώρους ή επαγγελματικές εφαρμογές, όπως αποθήκες, βιομηχανικούς χώρους ή πολυώροφα κτίρια.

Μερικά από τα προβλήματα που μπορεί να προκύψουν είναι ότι μπορεί να υπάρξει πρόβλημα στην σύνδεση δηλαδή να υπάρχουν θέματα στο Wi-Fi ή ακόμα και με τον mqtt broker κάτι το οποίο μπορεί να προκαλέσει απώλεια δεδομένων αλλά και να χαθεί ένα σημαντικό alert, επίσης μπορεί να υπάρχουν καθυστερήσεις στις μετρήσεις ειδικά αν υπάρχουν πολλοί αισθητήρες τύπου I2C γιατί εκεί ο esp διαβάζει σειριακά τα δεδομένα. Κάποιο ακόμα πρόβλημα είναι στην αξιοπιστία των αισθητήρων, δηλαδή κάποιοι αισθητήρες να τύχει να πάρουν θόρυβο ή λανθασμένες τιμές.



Συμπεράσματα

Εν κατακλείδι η εφαρμογή είναι ένα ολοκληρωμένο σύστημα IOT με έμφαση στις καινούργιες τεχνολογίες και αξιοποίηση όσο γίνεται νέων πρωτοκόλλων. Το σύστημα είναι σε θέση να ανταπεξέλθει και σε πραγματική χρήση χωρίς κανένα πρόβλημα.

Μελλοντικά θα μπορούν να προστεθούν νέες δυνατότητες όπως η χρήση AI για την ανάλυση των δεδομένων αλλά και πρόβλεψη καταστάσεων ή κινδύνων. Παράλληλα θα μπορούσαν να προσθέτουν νέου τύπου αισθητήρες για πιο σωστή πρόβλεψη και λήψη αποφάσεων.

Βιβλιογραφία

- 1) <https://www.olympia-electronics.com/el/%CF%80%CF%81%CE%BF%CF%8A%CF%8C%CE%BD%CF%84%CE%B1/%CF%83%CF%85%CF%83%CF%84%CE%AE%CE%BC%CE%B1%CF%84%CE%B1-%CF%80%CF%85%CF%81%CE%B1%CE%BD%CE%AF%CF%87%CE%BD%CE%B5%CF%85%CF%83%CE%B7%CF%82>
- 2) https://www.skroutz.gr/c/1394/fire_detection.html
- 3) <https://www.instructables.com/Fire-Detection-and-Alert-System-Using-an-ESP32-Mic/>
- 4) <https://www.hackster.io/512816/esp32-smart-automatic-fire-extinguisher-with-dht11-4573ce>
- 5) <https://www.instructables.com/DIY-Arduino-Fire-Alarm-System-Homemade/>
- 6) <https://www.hackster.io/mrmodder/diy-arduino-fire-alarm-system-at-home-c70970>