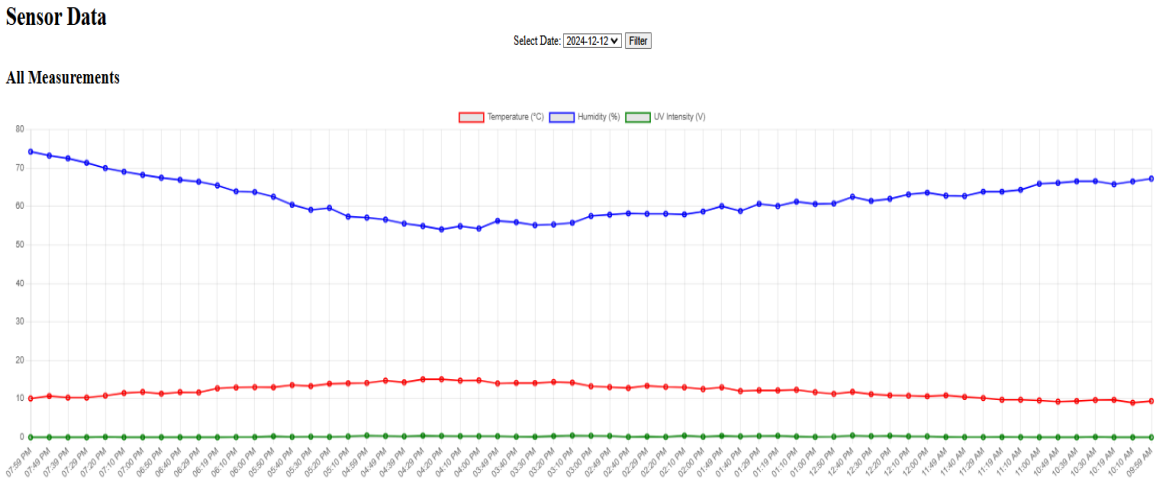
ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

# ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

# «Παρακολούθηση και ανάλυση περιβαλλοντικών συνθηκών έκθεσης προϊόντων»




**Φοιτητής**

ΜΙΧΑΗΛ ΜΑΥΡΙΔΗΣ

**517086**

**Επιβλέπων**

Δρ. Κυριάκος Τσιακμάκης

**Φεβρουάριος 2025**

Παρακολούθηση και ανάλυση περιβαλλοντικών συνθηκών έκθεσης προϊόντων

Κωδικός: 24304

Φοιτητής: Μαυρίδης Μιχαήλ

Εισηγητής: Δρ Κυριάκος Τσιακμάκης

Ημερομηνία ανάληψης Π.Ε. 31-10-2024

Ημερομηνία περάτωσης Π.Ε. 26-01-2025

*Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως πτυχιακή εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.*

*Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή* **Μαυρίδη Μιχαήλ** *που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.*

Η έγκριση της πτυχιακής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

# Περίληψη

Η παρούσα εργασία αφορά την ανάπτυξη συστήματος παρακολούθησης περιβαλλοντικών συνθηκών για προϊόντα κατά την έκθεσή τους στον ήλιο. Χρησιμοποιείται το ESP32 για τη συλλογή δεδομένων, όπως ο χρόνος έκθεσης στον ήλιο, η θερμοκρασία και η υγρασία, τα οποία αποστέλλονται σε server αναπτυγμένο σε Python με βάση δεδομένων MySQL. Η ασφάλεια των δεδομένων διασφαλίζεται μέσω blockchain, εξασφαλίζοντας ακεραιότητα και διαφάνεια. Το σύστημα παρέχει ανάλυση δεδομένων για τον συνολικό χρόνο έκθεσης και τις ακραίες συνθήκες, ειδοποιήσεις σε πραγματικό χρόνο για υπέρβαση ορίων, καθώς και αναφορές για την ποιότητα των προϊόντων. Στόχος είναι η βελτίωση της διαχείρισης προϊόντων, η λήψη τεκμηριωμένων αποφάσεων και η ενίσχυση της αξιοπιστίας μέσω προηγμένων τεχνολογιών.

« Monitoring and Analysis of Environmental Conditions for Product Exposure»

# Abstract

This study focuses on the development of a monitoring system for environmental conditions affecting products during sun exposure. An ESP32 is used to collect data such as exposure time, temperature, and humidity, which are sent to a server developed in Python with a MySQL database and a Bootstrap interface. Data security is ensured through blockchain technology, guaranteeing integrity and transparency. The system provides data analysis on total exposure time and extreme conditions, real-time notifications for threshold violations, and reports on product quality.

# Ευχαριστίες

Θέλω να ευχαριστήσω τους γονείς μου και τους φίλους μου για τη συμπαράσταση τους. Και τον επιβλέπων μου και τους συνεργάτες του για τη βοήθεια στους κώδικες.

# Περιεχόμενα

# Κατάλογος Σχημάτων

# Κεφάλαιο 1ο: Εισαγωγή

## 1.1 Εισαγωγή

Η παρούσα εργασία επικεντρώνεται στην ανάπτυξη ενός συστήματος για την παρακολούθηση και ανάλυση περιβαλλοντικών συνθηκών με τη χρήση τεχνολογιών Internet of Things (IoT). Σε έναν κόσμο όπου οι περιβαλλοντικές προκλήσεις αυξάνονται συνεχώς, η ανάγκη για αξιόπιστη και ακριβή καταγραφή δεδομένων είναι πιο επιτακτική από ποτέ. Το σύστημα που σχεδιάστηκε βασίζεται σε σύγχρονες τεχνολογίες χαμηλού κόστους, όπως το ESP32 και αισθητήρες μέτρησης περιβαλλοντικών παραμέτρων, και συνδυάζει την αποτελεσματικότητα με την ευκολία χρήσης. Σκοπός της εργασίας είναι να καταγραφούν και να αναλυθούν δεδομένα θερμοκρασίας, υγρασίας και έντασης UV, ώστε να παρέχεται η δυνατότητα παρακολούθησης περιβαλλοντικών συνθηκών σε πραγματικό χρόνο.

Για την υλοποίηση του συστήματος, συνδυάστηκαν διαφορετικές τεχνολογίες και εργαλεία. Ο κόμβος μέτρησης, βασισμένος στο ESP32, συλλέγει δεδομένα από αισθητήρες και τα αποστέλλει σε διακομιστή μέσω HTTP. Στη συνέχεια, τα δεδομένα αποθηκεύονται σε βάση δεδομένων MySQL και είναι διαθέσιμα για επεξεργασία και προβολή μέσω ιστοσελίδας που έχει σχεδιαστεί με τεχνολογίες HTML, CSS και JavaScript. Επιπλέον, η ασφάλεια των δεδομένων διασφαλίζεται με τη χρήση αλγορίθμων κρυπτογράφησης και επαλήθευσης, εξασφαλίζοντας την ακεραιότητα και την εμπιστευτικότητα της πληροφορίας.

Μία από τις κύριες λειτουργίες του συστήματος είναι η δυνατότητα ανίχνευσης υπερβάσεων προκαθορισμένων ορίων για θερμοκρασία και UV ένταση. Το σύστημα μπορεί να αποστέλλει ειδοποιήσεις σε πραγματικό χρόνο όταν οι μετρήσεις ξεπερνούν αυτά τα όρια, είτε μέσω email είτε με την αποθήκευση σχετικών ειδοποιήσεων στη βάση δεδομένων. Παράλληλα, η ιστοσελίδα παρέχει γραφήματα και πίνακες για την ανάλυση και την παρακολούθηση των δεδομένων, διευκολύνοντας τη λήψη αποφάσεων βασισμένων σε πραγματικά δεδομένα.

Η εργασία αυτή φιλοδοξεί να αποτελέσει ένα εργαλείο που μπορεί να χρησιμοποιηθεί τόσο για εκπαιδευτικούς σκοπούς όσο και για εφαρμογές σε πραγματικά περιβάλλοντα. Η σχεδίαση του συστήματος είναι τέτοια που επιτρέπει την εύκολη προσαρμογή και επέκταση, ενώ η έμφαση στην ασφάλεια και την επεκτασιμότητα καθιστούν το σύστημα ιδανικό για μελλοντική ανάπτυξη. Μέσα από αυτή την εργασία, γίνεται μια πρώτη προσπάθεια να αποδειχθεί η δυνατότητα χρήσης τεχνολογιών IoT για την αποτελεσματική και οικονομική παρακολούθηση του περιβάλλοντος, παρέχοντας ένα ισχυρό υπόβαθρο για περαιτέρω βελτιώσεις και καινοτομίες.

Ο κύριος στόχος της παρούσας εργασίας είναι η ανάπτυξη ενός ολοκληρωμένου και ασφαλούς συστήματος παρακολούθησης περιβαλλοντικών συνθηκών, το οποίο θα καταγράφει, θα αναλύει και θα

προβάλλει δεδομένα θερμοκρασίας, υγρασίας και UV έντασης σε πραγματικό χρόνο, ενώ παράλληλα θα παρέχει δυνατότητες ανίχνευσης υπερβάσεων ορίων και ειδοποιήσεων, με έμφαση στην αξιοπιστία, την επεκτασιμότητα και την ευκολία χρήσης.

Η συνεισφορά της παρούσας εργασίας είναι πολυδιάστατη, καθώς παρέχει ένα ολοκληρωμένο και προσαρμοστικό σύστημα για την παρακολούθηση περιβαλλοντικών συνθηκών. Πιο συγκεκριμένα, η εργασία συμβάλλει στα εξής:

1. **Ανάπτυξη Ολοκληρωμένου Συστήματος IoT**: Δημιουργείται ένα λειτουργικό σύστημα που συνδυάζει χαμηλού κόστους τεχνολογίες, όπως το ESP32 και αισθητήρες, με μια ασφαλή και αποτελεσματική υποδομή διακομιστή και βάσης δεδομένων. Το σύστημα είναι επεκτάσιμο και εύκολα προσαρμόσιμο για διάφορες περιβαλλοντικές εφαρμογές.
2. **Διασφάλιση Ασφάλειας Δεδομένων**: Ενσωματώνονται μηχανισμοί επαλήθευσης δεδομένων, όπως το hash verification, και προτείνονται μέθοδοι κρυπτογράφησης, διασφαλίζοντας την ακεραιότητα και την εμπιστευτικότητα της πληροφορίας από το στάδιο της συλλογής έως την αποθήκευση.
3. **Διαχείριση Ειδοποιήσεων σε Πραγματικό Χρόνο**: Παρέχεται η δυνατότητα ανίχνευσης υπερβάσεων προκαθορισμένων ορίων (θερμοκρασία και UV ένταση) και η αυτόματη αποστολή ειδοποιήσεων μέσω email ή αποθήκευσής τους στη βάση δεδομένων. Αυτή η λειτουργία καθιστά το σύστημα ιδανικό για άμεση απόκριση σε περιβαλλοντικές συνθήκες.
4. **Οπτικοποίηση και Ανάλυση Δεδομένων**: Αναπτύσσεται μια φιλική προς τον χρήστη ιστοσελίδα που παρέχει διαδραστικά γραφήματα και πίνακες δεδομένων, επιτρέποντας τη λεπτομερή ανάλυση των περιβαλλοντικών συνθηκών για συγκεκριμένες χρονικές περιόδους.
5. **Εφαρμογή σε Εκπαιδευτικά και Πραγματικά Περιβάλλοντα**: Το σύστημα μπορεί να χρησιμοποιηθεί ως εκπαιδευτικό εργαλείο για την κατανόηση των τεχνολογιών IoT και της σημασίας της ασφάλειας δεδομένων.

## 1.2 Δομή της εργασίας

Στο πρώτο κεφάλαιο παρουσιάζεται μια γενική εισαγωγή στην εργασία, περιγράφονται οι κύριοι στόχοι της και η συνεισφορά της στο πεδίο των συστημάτων IoT για την παρακολούθηση περιβαλλοντικών συνθηκών.

Στο δεύτερο κεφάλαιο παρουσιάζεται η βιβλιογραφική ανασκόπηση σχετικών συστημάτων και τεχνολογιών, με έμφαση στη διαχείριση ψυχρής αλυσίδας και τη χρήση τεχνολογιών IoT για τη μέτρηση και ανάλυση περιβαλλοντικών συνθηκών. Αναλύονται τρία σημαντικά έργα από τη διεθνή βιβλιογραφία.

Στο τρίτο κεφάλαιο γίνεται λεπτομερής περιγραφή της τεχνολογίας και του εξοπλισμού που χρησιμοποιήθηκε στην εργασία. Παρουσιάζονται οι αισθητήρες και το υλικό που επιλέχθηκαν, καθώς και οι πλατφόρμες ανάπτυξης και τα λογισμικά που υλοποιήθηκαν. Επιπλέον, περιγράφονται οι τεχνολογίες σχεδίασης της ιστοσελίδας και της οπτικοποίησης των δεδομένων.

Στο τέταρτο κεφάλαιο αναλύεται το σύστημα παρακολούθησης και ανάλυσης. Παρουσιάζονται τα επιμέρους συστατικά του, όπως ο κώδικας για τον αισθητήρα ESP32, ο server και οι μηχανισμοί ειδοποίησης. Περιγράφεται επίσης η ιστοσελίδα για την προβολή των δεδομένων και η βάση δεδομένων που χρησιμοποιήθηκε για την αποθήκευσή τους. Τέλος, γίνεται αναφορά στις στρατηγικές που εφαρμόστηκαν για τη διασφάλιση της ασφάλειας των δεδομένων.

Στο πέμπτο κεφάλαιο συνοψίζονται τα αποτελέσματα της εργασίας, παρουσιάζονται τα βασικά συμπεράσματα και διατυπώνονται προτάσεις για μελλοντική έρευνα και βελτιώσεις. Στο τέλος της εργασίας περιλαμβάνεται το παράρτημα με δείγματα κώδικα και αναφορές για την υλοποίηση.

# Κεφάλαιο 2ο: Βιβλιογραφική Ανασκόπηση

## 2.1 Η Επίδραση της Τεχνολογίας IoT στη Διαχείριση της Ψυχρής Αλυσίδας για Προμηθευτές Εφοδιαστικής Τροφίμων

Η εταιρεία Adapt Ideations εστιάζει στην ανάπτυξη καινοτόμων λύσεων για την παρακολούθηση και διαχείριση της ψυχρής αλυσίδας, αξιοποιώντας την τεχνολογία Internet of Things (IoT). Μέσω των υπηρεσιών και των προϊόντων της, επιδιώκει να προσφέρει στους πελάτες της τη δυνατότητα παρακολούθησης σε πραγματικό χρόνο, βελτιώνοντας τη διαφάνεια και την αποτελεσματικότητα στη διαχείριση ευπαθών προϊόντων, όπως τρόφιμα, φάρμακα και καλλυντικά. Η εταιρεία παρέχει προηγμένες συσκευές IoT με αισθητήρες θερμοκρασίας, υγρασίας και κραδασμών, οι οποίες συλλέγουν δεδομένα κατά τη μεταφορά και αποθήκευση προϊόντων. Τα δεδομένα αυτά αποστέλλονται σε cloud πλατφόρμες για ανάλυση και ανίχνευση πιθανών αποκλίσεων, επιτρέποντας την έγκαιρη λήψη διορθωτικών μέτρων. Το έργο της Adapt Ideations διασφαλίζει την ποιότητα και την ασφάλεια των προϊόντων, ενώ παράλληλα βοηθά τους πελάτες της να συμμορφώνονται με τις ρυθμιστικές απαιτήσεις και να μειώνουν τις απώλειες στη μεταφορά [1].

Η ψυχρή αλυσίδα, που αφορά τη διαχείριση θερμοκρασιών κατά τη μεταφορά και αποθήκευση ευαίσθητων προϊόντων όπως τρόφιμα, φάρμακα και άλλα ευπαθή αγαθά, αντιμετωπίζει διαρκώς αυξανόμενες απαιτήσεις για αξιοπιστία και αποτελεσματικότητα. Η τεχνολογία Internet of Things (IoT) παρέχει νέες δυνατότητες για την παρακολούθηση, τον έλεγχο και τη βελτίωση αυτών των διαδικασιών, μειώνοντας τις απώλειες προϊόντων και ενισχύοντας τη διαφάνεια σε κάθε στάδιο της εφοδιαστικής αλυσίδας.

Η ενσωμάτωση IoT στην ψυχρή αλυσίδα επιτρέπει τη συλλογή δεδομένων σε πραγματικό χρόνο, όπως θερμοκρασία, υγρασία, και επίπεδα δονήσεων, μέσω αισθητήρων που είναι εγκατεστημένοι σε φορτηγά, αποθήκες ή εμπορευματοκιβώτια. Αυτά τα δεδομένα αποστέλλονται σε cloud-based πλατφόρμες για ανάλυση και λήψη αποφάσεων, εξασφαλίζοντας ότι οι συνθήκες διατηρούνται εντός προκαθορισμένων ορίων. Με αυτό τον τρόπο, οι εταιρείες μπορούν να ανιχνεύουν άμεσα τυχόν αποκλίσεις και να λαμβάνουν διορθωτικά μέτρα πριν συμβούν απώλειες.

Ένα βασικό πλεονέκτημα της τεχνολογίας IoT είναι η δυνατότητα παρακολούθησης της κατάστασης του φορτίου σε πραγματικό χρόνο. Για παράδειγμα, αισθητήρες IoT μπορούν να παρακολουθούν συνεχώς τη θερμοκρασία μέσα σε ένα εμπορευματοκιβώτιο και να στέλνουν ειδοποιήσεις σε περιπτώσεις όπου η θερμοκρασία ξεπεράσει το επιτρεπτό όριο. Επιπλέον, η ανάλυση των δεδομένων που συλλέγονται επιτρέπει την πρόβλεψη πιθανών προβλημάτων, όπως βλάβες σε εξοπλισμό ψύξης, πριν αυτά επηρεάσουν την ποιότητα του φορτίου.

Η ενσωμάτωση IoT συμβάλλει επίσης στη βελτίωση της διαφάνειας, κάτι που είναι ιδιαίτερα σημαντικό για τους τελικούς καταναλωτές και τις ρυθμιστικές αρχές. Η ιχνηλασιμότητα των δεδομένων θερμοκρασίας και άλλων περιβαλλοντικών συνθηκών καθιστά δυνατή την επαλήθευση της ποιότητας των προϊόντων από την πηγή έως τον τελικό προορισμό. Αυτό είναι κρίσιμο σε τομείς όπως η μεταφορά τροφίμων ή φαρμάκων, όπου ακόμη και μικρές αποκλίσεις στη θερμοκρασία μπορούν να επηρεάσουν σημαντικά την ασφάλεια και την ποιότητα του προϊόντος.

Παρά τα σημαντικά οφέλη, η υιοθέτηση της τεχνολογίας IoT στην ψυχρή αλυσίδα παρουσιάζει και ορισμένες προκλήσεις. Αυτές περιλαμβάνουν το κόστος υλοποίησης, την ανάγκη για ειδικές υποδομές, και ζητήματα ασφάλειας δεδομένων. Η εξασφάλιση της εμπιστευτικότητας και της ακεραιότητας των δεδομένων που συλλέγονται από τις IoT συσκευές είναι υψίστης σημασίας, καθώς οι παραβιάσεις αυτών των δεδομένων μπορούν να έχουν σημαντικές οικονομικές και λειτουργικές συνέπειες.

Συνοψίζοντας, η τεχνολογία IoT έχει τη δυνατότητα να μεταμορφώσει την ψυχρή αλυσίδα, παρέχοντας ένα επίπεδο ακρίβειας, διαφάνειας και αποδοτικότητας που ήταν προηγουμένως αδύνατο. Με την κατάλληλη υποδομή και στρατηγική εφαρμογή, οι προμηθευτές εφοδιαστικής μπορούν να επωφεληθούν από τα πλεονεκτήματα του IoT, ελαχιστοποιώντας τις απώλειες και βελτιώνοντας τη συνολική ποιότητα των υπηρεσιών τους.

## 2.2 Κοστοβόρος Εφαρμογή Συστήματος Ιχνηλασιμότητας Θερμοκρασίας με Έξυπνα RFID και Υπηρεσίες IoT

Η εργασία [2] παρουσιάζει το σχεδιασμό και την επαλήθευση ενός συστήματος ιχνηλασιμότητας βασισμένου στην τεχνολογία RFID και τις υπηρεσίες Internet of Things (IoT). Στόχος του συστήματος είναι η αντιμετώπιση των προβλημάτων διασύνδεσης και κόστους υλοποίησης που παρατηρούνται συνήθως στα συστήματα ιχνηλασιμότητας. Η ενσωμάτωση αισθητήρων θερμοκρασίας στις ετικέτες RFID επιτρέπει την παρακολούθηση των συνθηκών των τροφίμων κατά τη μεταφορά. Η χρήση του IoT καθιστά δυνατή τη σύνδεση πολλαπλών συστημάτων στην ίδια πλατφόρμα, ενώ το σχήμα χρέωσης "Data as a Service" (DaaS) μειώνει την ανάγκη για μεγάλο αρχικό κόστος επένδυσης. Το σύστημα δοκιμάστηκε και επαληθεύτηκε σε δύο σενάρια: ένα σε εργαστηριακές συνθήκες με παρακολούθηση κολοκυθιών και ένα σε πραγματικές συνθήκες με παρακολούθηση πορτοκαλιών κατά τη μεταφορά τους από την Ισπανία στην Ιρλανδία.

Η παγκόσμια οικονομία έχει επηρεάσει τη βιομηχανία τροφίμων, αυξάνοντας την απόσταση που διανύουν τα τρόφιμα από τον παραγωγό στον τελικό καταναλωτή. Αυτή η αλλαγή έχει δημιουργήσει πολλές προκλήσεις όσον αφορά την ιχνηλασιμότητα των τροφίμων. Σκάνδαλα όπως αυτά με γενετικά

τροποποιημένα προϊόντα και επιμολύνσεις από μικροοργανισμούς, έχουν εντείνει τις ανησυχίες των καταναλωτών για την ποιότητα και τη διαφάνεια των προϊόντων. Η διατήρηση της ποιότητας και της ακεραιότητας των τροφίμων κατά την αλυσίδα εφοδιασμού είναι ζωτικής σημασίας για την πρόληψη ασθενειών και πιθανών επιθέσεων βιοτρομοκρατίας.

Για την αντιμετώπιση αυτών των προκλήσεων, έχουν αναπτυχθεί κανονισμοί, όπως το πρότυπο ISO 22005, που απαιτούν την ανίχνευση της προέλευσης και των χαρακτηριστικών των προϊόντων. Το πρότυπο αυτό ενθαρρύνει την καταγραφή και τη διαχείριση πληροφοριών καθ' όλη την αλυσίδα εφοδιασμού, διασφαλίζοντας την ποιότητα και τον έλεγχο .

Το σύστημα αποτελείται από μια πολυεπίπεδη δομή που συνδυάζει την τεχνολογία RFID με πλατφόρμες IoT. Κάθε ετικέτα RFID διαθέτει ενσωματωμένο αισθητήρα θερμοκρασίας, ενώ τα δεδομένα μεταφέρονται μέσω πύλης IoT στο cloud. Η πύλη είναι υπεύθυνη για τη διαχείριση δεδομένων σε πραγματικό χρόνο, ενώ η χρήση του μοντέλου DaaS επιτρέπει στους χρήστες να πληρώνουν μόνο για τα δεδομένα που καταναλώνουν, εξαλείφοντας το αρχικό κόστος εξοπλισμού.

Δύο σενάρια χρησιμοποιήθηκαν για την επαλήθευση του συστήματος:

1. Εργαστηριακή παρακολούθηση κολοκυθιών σε ελεγχόμενες θερμοκρασίες.
2. Πραγματική παρακολούθηση πορτοκαλιών κατά τη μεταφορά τους, όπου καταγράφηκαν οι θερμοκρασίες στις οποίες εκτέθηκαν .

Το σύστημα κατέγραψε τις θερμοκρασίες σε διάφορα στάδια, αποκαλύπτοντας ότι οι συνθήκες διατηρήθηκαν κατάλληλες εκτός από μερικές ώρες κατά τη μεταφορά. Στην περίπτωση των πορτοκαλιών, η παρακολούθηση έδειξε ότι τα προϊόντα διατηρήθηκαν σε κατάλληλες θερμοκρασίες κατά την πλειονότητα του ταξιδιού, με μικρές αποκλίσεις που δεν επηρέασαν την ποιότητα του προϊόντος. Και στις δύο περιπτώσεις, το σύστημα απέδειξε την ακρίβεια και την αποδοτικότητά του.

Το σύστημα που παρουσιάζει μια αποτελεσματική και οικονομική λύση για την παρακολούθηση της θερμοκρασίας στην αλυσίδα εφοδιασμού. Η χρήση τεχνολογιών RFID και IoT επιτρέπει την ανίχνευση των συνθηκών σε πραγματικό χρόνο, ενώ το μοντέλο DaaS καθιστά τη λύση προσιτή για μικρομεσαίες επιχειρήσεις. Μελλοντικές βελτιώσεις περιλαμβάνουν την επέκταση του συστήματος για τη διαχείριση περισσότερων παραμέτρων και την ενσωμάτωση με τεχνολογίες ανάλυσης δεδομένων.

## 2.3 Η IoT-ενεργοποιημένη Παρακολούθηση Ψυχρής Αλυσίδας σε Πραγματικό Χρόνο σε Λιμενικές Εγκαταστάσεις

Το άρθρο “Internet of Things Enabled Real-Time Cold Chain Monitoring in a Container Port” παρουσιάζει μια ολοκληρωμένη μελέτη για τη χρήση τεχνολογιών IoT στη διαχείριση ψυχρής αλυσίδας (Cold Chain Logistics) σε λιμενικούς χώρους. Εξετάζει πώς η παρακολούθηση σε πραγματικό χρόνο με τη χρήση αισθητήρων, RFID, και δικτύων WSN μπορεί να ενισχύσει τη διαχείριση θερμοκρασίας, υγρασίας, και άλλων κρίσιμων παραμέτρων που επηρεάζουν προϊόντα ευαίσθητα στη θερμοκρασία [3].

Οι λιμένες, ως κομβικοί σταθμοί για τη μεταφορά προϊόντων, είναι καίριοι για την ψυχρή αλυσίδα. Όταν διακόπτεται η ψυχρή αλυσίδα κατά τη μεταφορά και αποθήκευση, δημιουργούνται συνθήκες ευνοϊκές για ανάπτυξη βακτηρίων και υποβάθμιση προϊόντων. Το άρθρο προτείνει ένα IoT-enabled σύστημα που χρησιμοποιεί αισθητήρες και RFID για την παρακολούθηση θερμοκρασίας, υγρασίας, και άλλων παραμέτρων σε πραγματικό χρόνο. Το σύστημα υποστηρίζει ειδοποιήσεις μέσω SMS, email και οπτικών συναγερμών, ενώ τα δεδομένα αποθηκεύονται και αναλύονται για μελλοντική λήψη αποφάσεων* Η εισαγωγή του άρθρου τονίζει τη σημασία της διατήρησης της ψυχρής αλυσίδας για προϊόντα όπως τρόφιμα, φαρμακευτικά προϊόντα και χημικές ουσίες. Οι λιμενικές εγκαταστάσεις συχνά αντιμετωπίζουν προκλήσεις λόγω της φύσης των εργασιών τους, οι οποίες περιλαμβάνουν μεταφορά, αποθήκευση και μεταφόρτωση. Εξαιτίας αυτών των προκλήσεων, προκύπτει η ανάγκη για συστήματα που να μπορούν να παρακολουθούν τις συνθήκες του περιβάλλοντος και να διασφαλίζουν τη διατήρηση της ποιότητας των προϊόντων .

Μέθοδοι που προτείνεται στο άρθρο περιλαμβάνει τις εξής τεχνολογίες:

- **Δίκτυα Αισθητήρων (WSN):** Χρησιμοποιούνται για τη συλλογή δεδομένων από το περιβάλλον, όπως η θερμοκρασία και η υγρασία.
- **RFID:** Ενσωματώνεται για την παρακολούθηση προϊόντων και τον εντοπισμό τους σε πραγματικό χρόνο.
- **GSM Gateway:** Χρησιμοποιείται για την αποστολή δεδομένων από το σύστημα αισθητήρων στον server μέσω ασύρματων δικτύων. Το άρθρο περιλαμβάνει ένα μοντέλο προσομοίωσης που εξετάζει την αποτελεσματικότητα του συστήματος, με ιδιαίτερη έμφαση στην ταχύτητα μετάδοσης δεδομένων και την ακρίβεια των μετρήσεων .

Οι προσομοιώσεις δείχνουν ότι το προτεινόμενο σύστημα είναι σε θέση να μεταφέρει δεδομένα με υψηλή ταχύτητα και ακρίβεια. Η μέση καθυστέρηση στη μετάδοση δεδομένων (end-to-end latency) είναι σταθερή στα 0.015 δευτερόλεπτα, ενώ το throughput φτάνει τα 10 Kbps, επαρκές για τις ανάγκες της ψυχρής αλυσίδας. Τα δεδομένα που συλλέγονται περιλαμβάνουν όχι μόνο τις περιβαλλοντικές παραμέτρους αλλά και τον εκτιμώμενο χρόνο ζωής προϊόντων (shelf life) με βάση μαθηματικά μοντέλα όπως το Accelerated Shelf Life Test (ASLT) .

Το άρθρο καταλήγει ότι τος σύστημα τους μπορεί να βελτιώσει σημαντικά τη διαχείριση της ψυχρής αλυσίδας σε λιμενικές εγκαταστάσεις. Παρέχουν:

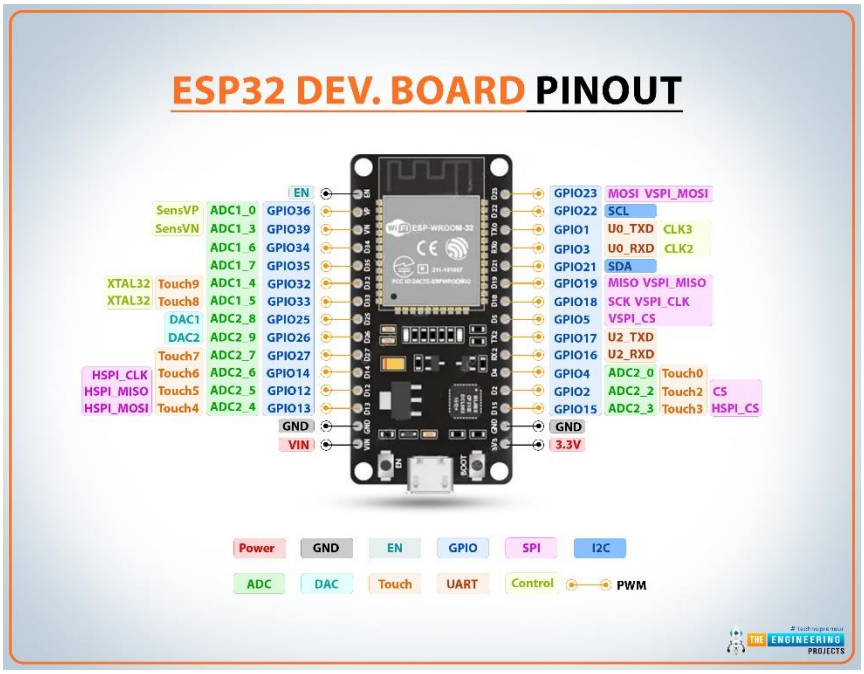
- **Αυξημένη διαφάνεια και λήψη αποφάσεων:** Μέσω πραγματικών δεδομένων και γραφικών εργαλείων ανάλυσης.
- **Μείωση κινδύνων:** Μέσω έγκαιρων ειδοποιήσεων σε περίπτωση ανωμαλιών.
- **Επεκτασιμότητα και προσαρμοστικότητα:** Εύκολη ενσωμάτωση σε υπάρχοντα συστήματα και προσαρμογή στις απαιτήσεις της βιομηχανίας. Παράλληλα, αναγνωρίζει ότι απαιτούνται περισσότερες εφαρμογές πεδίου για τη γενίκευση των αποτελεσμάτων .

# Κεφάλαιο 3ο: Τι χρησιμοποιήθηκε για την υλοποίηση της εργασίας

## 3.1 Αισθητήρες και Υλικό

### 3.1.1 ESP32

Το ESP32 είναι ένας ισχυρός και ευέλικτος μικροελεγκτής που έχει σχεδιαστεί για εφαρμογές IoT (Internet of Things). Πρόκειται για ένα σύστημα-on-chip (SoC) που διαθέτει διπύρηνο επεξεργαστή, Wi-Fi και Bluetooth, και μεγάλη γκάμα θυρών εισόδου/εξόδου (GPIO), καθιστώντας το ιδανικό για εφαρμογές όπως παρακολούθηση περιβάλλοντος, αυτοματισμούς και συστήματα ασφαλείας [4].




Εικόνα 3.1: esp32 pinout

[https://images.theengineeringprojects.com/image/main/2024/03/esp32-pinout.jpg]

Η ενσωμάτωση του ESP32 στο σύστημα που αναπτύχθηκε στην παρούσα εργασία ήταν κομβική για την αποστολή δεδομένων από τους αισθητήρες στον διακομιστή μέσω Wi-Fi. Η ισχυρή υπολογιστική ισχύς και η ενεργειακή αποδοτικότητα του ESP32 το καθιστούν κατάλληλο για έργα όπου απαιτείται συνεχής συλλογή δεδομένων σε πραγματικό χρόνο.

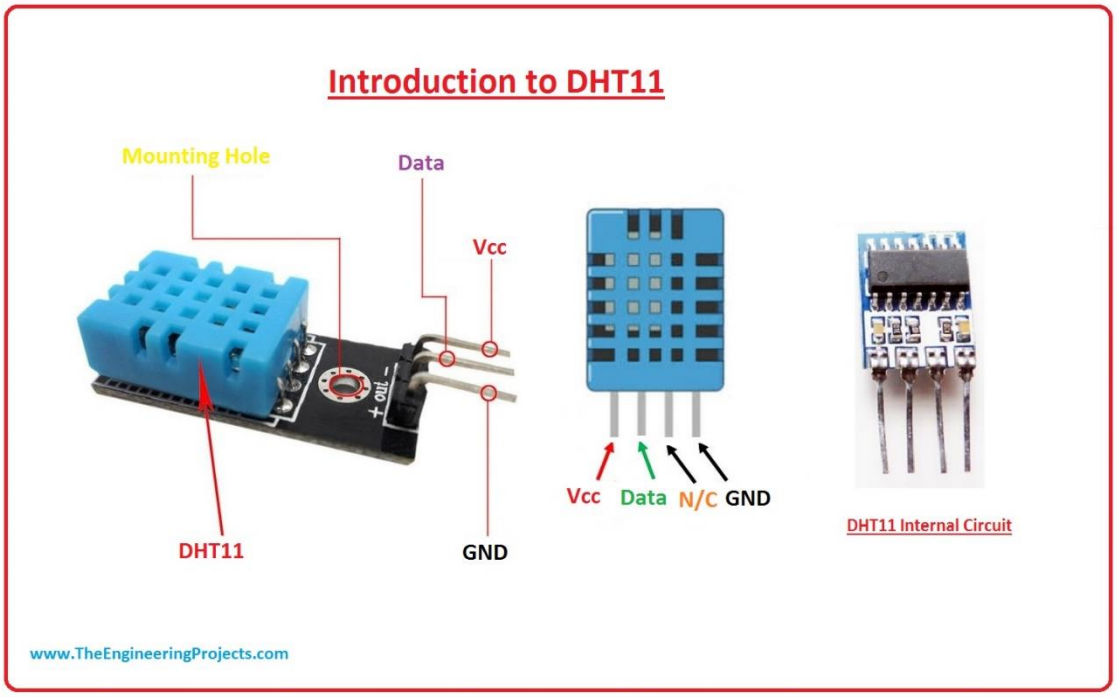
Το ESP32 υποστηρίζεται από πολλές πλατφόρμες ανάπτυξης, όπως το Arduino IDE, καθιστώντας εύκολη τη διαμόρφωση και τον προγραμματισμό του. Οι δυνατότητες Wi-Fi που διαθέτει επέτρεψαν την αποστολή δεδομένων μέσω HTTP στον Python server, ενώ οι είσοδοι/έξοδοι (GPIO) του χρησιμοποιήθηκαν για τη σύνδεση με αισθητήρες, όπως το DHT11 και ο UV αισθητήρας.

### 3.1.2 DHT11 για θερμοκρασία και υγρασία

Ο αισθητήρας DHT11 είναι ένας από τους πιο διαδεδομένους και προσιτούς αισθητήρες για τη μέτρηση θερμοκρασίας και υγρασίας, καθιστώντας τον ιδανικό για εφαρμογές που σχετίζονται με την παρακολούθηση περιβαλλοντικών συνθηκών. Διαθέτει ένα ενσωματωμένο θερμίστορ για τη μέτρηση θερμοκρασίας και έναν αισθητήρα αντίστασης για την υγρασία, προσφέροντας αξιόπιστες μετρήσεις σε χαμηλό κόστος [5].

Ο DHT11 παρέχει μετρήσεις θερμοκρασίας σε εύρος 0-50°C και υγρασίας σε εύρος 20-90%. Οι μετρήσεις είναι επαρκείς για πολλές εφαρμογές όπου η ακρίβεια δεν είναι η κύρια προτεραιότητα, όπως οικιακά συστήματα παρακολούθησης ή εκπαιδευτικά έργα. Η επικοινωνία του αισθητήρα γίνεται μέσω ενός ψηφιακού σήματος, καθιστώντας την ενσωμάτωσή του με μικροελεγκτές, όπως το ESP32, εξαιρετικά απλή.

Στο πλαίσιο της παρούσας εργασίας, ο αισθητήρας DHT11 χρησιμοποιήθηκε για τη συλλογή δεδομένων θερμοκρασίας και υγρασίας από το περιβάλλον. Ο αισθητήρας ήταν συνδεδεμένος στο ESP32 μέσω μιας από τις ψηφιακές εισόδους/εξόδους (GPIO), ενώ ο κώδικας προγραμματισμού για την ανάγνωση των δεδομένων υλοποιήθηκε στην πλατφόρμα Arduino IDE. Τα δεδομένα θερμοκρασίας και υγρασίας αποστέλλονταν στον διακομιστή μέσω Wi-Fi για αποθήκευση στη βάση δεδομένων και περαιτέρω ανάλυση.




Εικόνα 3.2: dht 11 pinout

[https://images.theengineeringprojects.com/image/main/2019/02/dht11.jpg]

Ο αισθητήρας ήταν τοποθετημένος στο εσωτερικό του πειραματικού κιβωτίου, όπου συλλέγονταν δεδομένα σε τακτά χρονικά διαστήματα (ανά ένα λεπτό). Αυτά τα δεδομένα ήταν ζωτικής σημασίας για την αξιολόγηση των περιβαλλοντικών συνθηκών και τη δημιουργία ειδοποιήσεων σε περίπτωση υπερβάσεων ορίων.

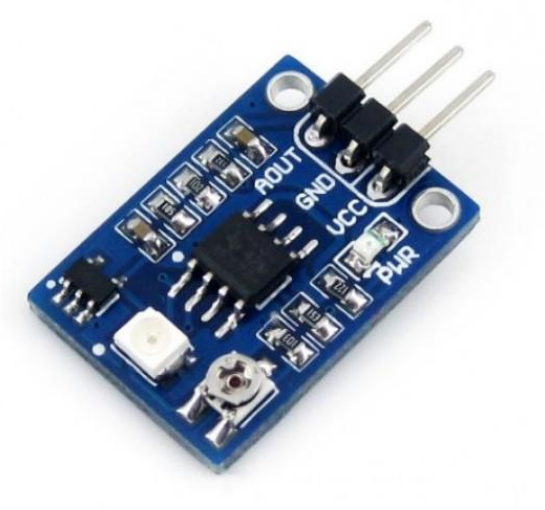
Ο DHT11 παρέχει έναν αξιόπιστο τρόπο μέτρησης των περιβαλλοντικών συνθηκών, ενώ η χαμηλή του κατανάλωση ενέργειας και η απλότητα στην εγκατάσταση τον καθιστούν ιδανικό για εφαρμογές IoT. Η ενσωμάτωσή του στο ESP32 επέτρεψε τη δημιουργία ενός ολοκληρωμένου συστήματος παρακολούθησης, χωρίς την ανάγκη για σύνθετο εξοπλισμό.

Παρά τα πλεονεκτήματά του, ο DHT11 έχει περιορισμούς ως προς την ακρίβεια και το εύρος μέτρησης. Για πιο απαιτητικές εφαρμογές, θα μπορούσε να εξεταστεί η χρήση πιο εξελιγμένων αισθητήρων, όπως ο DHT22 ή ο AM2302, οι οποίοι παρέχουν μεγαλύτερη ακρίβεια και ευρύτερα εύρη μέτρησης.

### 3.1.3 UV αισθητήρας (GUVA-S12SD)

Ο UV αισθητήρας GUVA-S12SD αποτελεί μια προηγμένη λύση για τη μέτρηση της έντασης της υπεριώδους ακτινοβολίας (UV). Η υπεριώδης ακτινοβολία, ως σημαντικός παράγοντας στην αλλοίωση προϊόντων και τη βιολογική φθορά, είναι κρίσιμη για την παρακολούθηση περιβαλλοντικών συνθηκών, ειδικά σε εφαρμογές όπως η μεταφορά τροφίμων, φαρμάκων και άλλων ευπαθών υλικών [6].

Ο GUVA-S12SD είναι ένας αισθητήρας βασισμένος σε φωτοδιόδους, ο οποίος ανιχνεύει την ένταση της UV ακτινοβολίας στο εύρος 240-370 nm (UV-A και UV-B). Η έξοδος του αισθητήρα είναι αναλογική, γεγονός που καθιστά δυνατή την απλή σύνδεσή του με μικροελεγκτές, όπως το ESP32. Η τάση εξόδου είναι ανάλογη της έντασης της UV ακτινοβολίας, με υψηλή ακρίβεια και γρήγορη απόκριση.




Εικόνα 3.3: Waveshare Αισθητήρας Υπεριώδους Ακτινοβολίας

[https://grobotronics.com/images/thumbnails/570/570/detailed/106/uvsern_grobo.jpg]

Στην παρούσα εργασία, ο αισθητήρας GUVA-S12SD χρησιμοποιήθηκε για τη μέτρηση της UV έντασης, προκειμένου να κατανοηθεί ο ρόλος της ηλιακής ακτινοβολίας στις περιβαλλοντικές συνθήκες όπου βρίσκονται τα ευαίσθητα προϊόντα. Ο αισθητήρας συνδέθηκε στην αναλογική είσοδο του ESP32 και οι μετρήσεις του μετατρέπονταν σε τιμές τάσης (volt), οι οποίες αποστέλλονταν μέσω Wi-Fi στον Python server.

Οι τιμές UV, μαζί με τις τιμές θερμοκρασίας και υγρασίας από τον DHT11, καταγράφονταν σε πραγματικό χρόνο, επιτρέποντας τη δημιουργία γραφημάτων και ειδοποιήσεων. Η ανίχνευση

υπερβάσεων ορίων UV ήταν ιδιαίτερα χρήσιμη για την παρακολούθηση πιθανών κινδύνων κατά την έκθεση προϊόντων σε ηλιακή ακτινοβολία.

Ο GUVA-S12SD παρέχει αξιόπιστες και σταθερές μετρήσεις της UV ακτινοβολίας, ενώ η απλή του συνδεσιμότητα και η μικρή κατανάλωση ενέργειας τον καθιστούν κατάλληλο για συστήματα IoT. Οι μετρήσεις του συμβάλλουν στη δημιουργία αναλυτικών δεδομένων που υποστηρίζουν τη βελτίωση της διαχείρισης ευαίσθητων προϊόντων.

Παρότι ο αισθητήρας GUVA-S12SD καλύπτει ευρύ φάσμα UV ακτινοβολίας, δεν παρέχει διαχωρισμό μεταξύ UV-A και UV-B, κάτι που θα μπορούσε να είναι χρήσιμο σε πιο εξειδικευμένες εφαρμογές. Επιπλέον, οι μετρήσεις επηρεάζονται από το περιβάλλον, όπως η παρουσία σύννεφων ή φυσικών εμποδίων.

### 3.1.4 RTC DS3231 για χρονοσήμανση

Ο RTC (Real Time Clock) DS3231 είναι ένας εξαιρετικά ακριβής αισθητήρας χρόνου που χρησιμοποιείται ευρέως σε εφαρμογές που απαιτούν ακριβή χρονομέτρηση. Χάρη στην ενσωμάτωσή του σε συστήματα IoT, επιτρέπει την ακριβή καταγραφή δεδομένων με χρονική σήμανση (timestamp), διασφαλίζοντας τη συνέπεια και την αξιοπιστία της συλλογής δεδομένων [7].

Ο DS3231 είναι ένας χρονόμετρος υψηλής ακρίβειας που βασίζεται σε κρύσταλλο (TCXO) και ενσωματώνει θερμοκρασιακή αντιστάθμιση για την ελαχιστοποίηση των αποκλίσεων λόγω μεταβολών της θερμοκρασίας. Έχει ενσωματωμένη μπαταρία για να διατηρεί τον χρόνο σε περίπτωση απώλειας ρεύματος, ενώ παρέχει ακριβείς μετρήσεις με απόκλιση λιγότερη από ±2 ppm (0.1728 δευτερόλεπτα την ημέρα).


Εικόνα 3.4: Real Time Clock Module I2C - DS3231

[https://grobotronics.com/images/thumbnails/570/570/detailed/126/htb1vmlfigvd8kjjssplq6yiefxaw_dd4c0822-d896-4fac-ac0b-00ab4a54d0c0_2000x_grobo.jpg]

Ο αισθητήρας επικοινωνεί μέσω I²C πρωτοκόλλου, κάνοντάς τον συμβατό με μικροελεγκτές όπως το ESP32. Υποστηρίζει διάφορες λειτουργίες, όπως χρονικά γεγονότα (alarms) και παρακολούθηση θερμοκρασίας, παρέχοντας έτσι μεγαλύτερη ευελιξία στις εφαρμογές.

Στην παρούσα εργασία, ο DS3231 χρησιμοποιήθηκε για την παροχή χρονοσήμανσης σε κάθε μέτρηση που συλλέγεται από τους αισθητήρες. Ο RTC συνδέθηκε με το ESP32 μέσω του I²C διαύλου, εξασφαλίζοντας την ακριβή χρονική σήμανση των δεδομένων θερμοκρασίας, υγρασίας, και UV. Η λειτουργία αυτή ήταν ζωτικής σημασίας για την αποθήκευση και ανάλυση των δεδομένων, καθώς επέτρεψε τη δημιουργία χρονολογικών γραφημάτων και την παρακολούθηση περιβαλλοντικών συνθηκών σε πραγματικό χρόνο.

Οι χρονοσημασμένες μετρήσεις ήταν απαραίτητες για την αναγνώριση μοτίβων και ανωμαλιών, ενώ αποτέλεσαν τη βάση για την ενεργοποίηση ειδοποιήσεων σε περίπτωση υπερβάσεων ορίων. Επιπλέον, ο RTC επέτρεψε την αξιοπιστία του συστήματος ακόμα και σε περιπτώσεις προσωρινής διακοπής της

Ο DS3231 παρέχει υψηλή ακρίβεια και αξιοπιστία στη χρονομέτρηση, ενώ η ενσωματωμένη μπαταρία διασφαλίζει τη συνεχή λειτουργία του ακόμα και σε περιπτώσεις διακοπών. Η χρήση του σε συνδυασμό με το ESP32 επιτρέπει τη δημιουργία συστημάτων IoT που απαιτούν ακριβείς χρονικές σφραγίδες για τις μετρήσεις τους.

Αν και ο DS3231 είναι εξαιρετικά ακριβής, εξαρτάται από την ποιότητα της μπαταρίας για τη διατήρηση του χρόνου σε μεγάλες περιόδους χωρίς ρεύμα. Επιπλέον, απαιτείται προγραμματισμός για τη σωστή ανάγνωση και χρήση των δεδομένων από τον I²C δίαυλο.

## 3.2 Πλατφόρμες Ανάπτυξης και Λογισμικό

### 3.2.1 Arduino IDE για προγραμματισμό ESP32

Το Arduino IDE είναι μια από τις πιο δημοφιλείς πλατφόρμες για προγραμματισμό μικροελεγκτών. Υποστηρίζει μια μεγάλη γκάμα συσκευών, συμπεριλαμβανομένου του ESP32, και παρέχει ένα εύχρηστο περιβάλλον ανάπτυξης που επιτρέπει στους προγραμματιστές να γράφουν, να δοκιμάζουν και να αναπτύσσουν κώδικα γρήγορα και αποτελεσματικά [8].

Το Arduino IDE χρησιμοποιεί τη γλώσσα προγραμματισμού C/C++, με τη βιβλιοθήκη Arduino API να απλοποιεί τη διαχείριση των περιφερειακών. Παρέχει ενσωματωμένο serial monitor για δοκιμές σε

πραγματικό χρόνο και προσφέρει υποστήριξη για πολλά πρόσθετα, όπως οι βιβλιοθήκες για αισθητήρες, δίκτυα και επικοινωνίες.

Στην παρούσα εργασία, το Arduino IDE χρησιμοποιήθηκε για τον προγραμματισμό του ESP32. Οι βιβλιοθήκες για τον αισθητήρα DHT11, τον αισθητήρα UV, και τον RTC DS3231 ενσωματώθηκαν εύκολα στο περιβάλλον ανάπτυξης, διευκολύνοντας τη διαχείριση των δεδομένων. Επιπλέον, οι δυνατότητες σύνδεσης με Wi-Fi ρυθμίστηκαν μέσω του IDE, επιτρέποντας τη μεταφορά δεδομένων στον διακομιστή.

### 3.2.2 Python για τον server

Η Python είναι μια από τις πιο δημοφιλείς γλώσσες προγραμματισμού παγκοσμίως, γνωστή για την ευχρηστία της και τη μεγάλη κοινότητα χρηστών. Παρέχει μια ευρεία γκάμα βιβλιοθηκών που υποστηρίζουν την ανάπτυξη εφαρμογών IoT, καθιστώντας την ιδανική για τη δημιουργία backend συστημάτων [9].

Η Python προσφέρει ισχυρά εργαλεία για τη διαχείριση δεδομένων, την ανάπτυξη API, και τη διασύνδεση με βάσεις δεδομένων. Η βιβλιοθήκη mysql.connector χρησιμοποιήθηκε για την επικοινωνία με τη MySQL, ενώ η Python ήταν το θεμέλιο για την υλοποίηση του server και της επικοινωνίας με τους κόμβους ESP32.

Ο Python server ήταν υπεύθυνος για την παραλαβή δεδομένων μέσω HTTP, την επαλήθευση της ακεραιότητας τους με χρήση hash verification, και την αποθήκευσή τους στη βάση δεδομένων MySQL. Επιπλέον, ο server παρείχε API για την παρουσίαση των δεδομένων μέσω της ιστοσελίδας, επιτρέποντας την άμεση οπτικοποίηση τους.

### 3.2.3 Flask framework για API και ιστοσελίδα

Το Flask είναι ένα μικρό αλλά ισχυρό framework για την ανάπτυξη web εφαρμογών. Είναι ελαφρύ, ευέλικτο και ιδανικό για τη δημιουργία RESTful API, καθώς και για την ανάπτυξη απλών ιστοσελίδων [10].

Το Flask παρέχει ευκολία στη δημιουργία web διακομιστών, με υποστήριξη για πολλές επεκτάσεις που διευκολύνουν την ανάπτυξη εφαρμογών. Είναι γραμμένο σε Python, καθιστώντας το ιδανικό για την ενοποίηση με άλλα Python-based συστήματα, όπως βάσεις δεδομένων και μηχανισμούς ανάλυσης δεδομένων.

Στην παρούσα εργασία, το Flask χρησιμοποιήθηκε για την ανάπτυξη του API που επικοινωνεί με τους κόμβους ESP32. Επιπλέον, μέσω του Flask δημιουργήθηκε η ιστοσελίδα που παρουσιάζει γραφήματα

και πίνακες δεδομένων. Η οργάνωση των API endpoints επέτρεψε την εύκολη ανάκτηση δεδομένων και την παρουσίασή τους σε πραγματικό χρόνο.

### 3.2.4 MySQL για αποθήκευση δεδομένων

Η MySQL είναι μία από τις πιο δημοφιλείς σχεσιακές βάσεις δεδομένων, γνωστή για την αξιοπιστία, την ταχύτητα και την ευκολία χρήσης της. Παρέχει δυνατότητες διαχείρισης μεγάλων όγκων δεδομένων, γεγονός που την καθιστά ιδανική για εφαρμογές IoT [11].

Η MySQL επιτρέπει την αποθήκευση, ανάκτηση, και διαχείριση δεδομένων με χρήση της γλώσσας SQL. Παρέχει υποστήριξη για συνδέσεις πολλών χρηστών και ενσωματώνεται εύκολα με γλώσσες προγραμματισμού, όπως η Python, μέσω βιβλιοθηκών όπως το mysql.connector.

Η MySQL χρησιμοποιήθηκε για την αποθήκευση όλων των δεδομένων που συλλέχθηκαν από τους αισθητήρες. Οι πίνακες της βάσης δεδομένων περιλάμβαναν πληροφορίες όπως θερμοκρασία, υγρασία, UV ακτινοβολία, και χρονικές σημάνσεις. Επιπλέον, η βάση δεδομένων διαχειριζόταν τις ειδοποιήσεις που ενεργοποιούνταν σε περιπτώσεις υπέρβασης των προκαθορισμένων ορίων.

## 3.3 Ιστοσελίδα και Οπτικοποίηση

### 3.3.1 HTML, CSS, JavaScript για τη σχεδίαση της ιστοσελίδας

Το Chart.js είναι μια βιβλιοθήκη JavaScript ανοιχτού κώδικα που παρέχει εργαλεία για τη δημιουργία διαδραστικών και προσαρμοσμένων γραφημάτων. Είναι ελαφριά, εύκολη στη χρήση, και υποστηρίζει πολλούς τύπους γραφημάτων, όπως γραμμικά, ραβδογράμματα και πίτες [12].

- Παρέχει πολλαπλούς τύπους γραφημάτων που μπορούν να προσαρμοστούν.
- Υποστηρίζει διαδραστικότητα, επιτρέποντας στον χρήστη να αλληλεπιδρά με τα δεδομένα.
- Εύκολη ενσωμάτωση με JavaScript και HTML.

Στην παρούσα εργασία, το Chart.js χρησιμοποιήθηκε για τη δημιουργία γραφημάτων που παρουσίαζαν δεδομένα θερμοκρασίας, υγρασίας και UV ακτινοβολίας ανά ημέρα. Η δυνατότητα δημιουργίας δυναμικών γραφημάτων επέτρεψε την άμεση οπτικοποίηση δεδομένων που συλλέχθηκαν από τους αισθητήρες. Οι χρήστες μπορούσαν να δουν συνολικά δεδομένα ή να επιλέξουν συγκεκριμένες ημερομηνίες για περαιτέρω ανάλυση.

### 3.3.2 Chart.js για γραφήματα

Το Chart.js είναι μια βιβλιοθήκη JavaScript ανοιχτού κώδικα που παρέχει εργαλεία για τη δημιουργία διαδραστικών και προσαρμοσμένων γραφημάτων. Είναι ελαφριά, εύκολη στη χρήση, και υποστηρίζει πολλούς τύπους γραφημάτων, όπως γραμμικά, ραβδογράμματα και πίτες [13].

- Παρέχει πολλαπλούς τύπους γραφημάτων που μπορούν να προσαρμοστούν.
- Υποστηρίζει διαδραστικότητα, επιτρέποντας στον χρήστη να αλληλεπιδρά με τα δεδομένα.

Στην παρούσα εργασία, το Chart.js χρησιμοποιήθηκε για τη δημιουργία γραφημάτων που παρουσίαζαν δεδομένα θερμοκρασίας, υγρασίας και UV ακτινοβολίας ανά ημέρα. Η δυνατότητα δημιουργίας δυναμικών γραφημάτων επέτρεψε την άμεση οπτικοποίηση δεδομένων που συλλέχθηκαν από τους αισθητήρες. Οι χρήστες μπορούσαν να δουν συνολικά δεδομένα ή να επιλέξουν συγκεκριμένες ημερομηνίες για περαιτέρω ανάλυση.

### 3.3.3 DataTables για τη διαχείριση δεδομένων

Η DataTables είναι μια δημοφιλής βιβλιοθήκη JavaScript που προσφέρει δυνατότητες για την παρουσίαση και τη διαχείριση δεδομένων σε πίνακες. Υποστηρίζει λειτουργίες όπως ταξινόμηση, φιλτράρισμα και αναζήτηση, κάνοντας τη διαχείριση δεδομένων πιο αποτελεσματική [14].

- Παρέχει λειτουργίες φιλτραρίσματος και ταξινόμησης για την εύκολη ανάλυση δεδομένων.
- Ενσωματώνεται εύκολα σε HTML πίνακες.
- Υποστηρίζει την εισαγωγή δεδομένων από API.

Στην ιστοσελίδα της εργασίας, η DataTables χρησιμοποιήθηκε για την παρουσίαση δεδομένων από τη βάση MySQL σε μορφή πίνακα. Οι χρήστες μπορούσαν να δουν τα δεδομένα των αισθητήρων ανά ημέρα, να τα φιλτράρουν, και να ταξινομήσουν τις εγγραφές ανάλογα με την ημερομηνία, τη θερμοκρασία, την υγρασία ή την UV ένταση. Η διαδραστικότητα που προσέφερε η DataTables βελτίωσε τη χρηστικότητα της ιστοσελίδας.

# Κεφάλαιο 4ο: Το σύστημα για την παρακολούθηση και ανάλυση περιβαλλοντικών συνθηκών έκθεσης προϊόντων

## 4.1 Εισαγωγή στη διαδικασία

Η παρακολούθηση και ανάλυση περιβαλλοντικών συνθηκών έκθεσης προϊόντων αποτελεί μια καινοτόμο διαδικασία που αξιοποιεί τις δυνατότητες των σύγχρονων τεχνολογιών του Διαδικτύου των Πραγμάτων (IoT), της ανάλυσης δεδομένων και της ασφάλειας με blockchain. Το σύστημα που αναπτύχθηκε για τον σκοπό αυτό, στοχεύει στη συλλογή, αποθήκευση, ανάλυση και παρουσίαση δεδομένων που αφορούν κρίσιμες παραμέτρους, όπως η θερμοκρασία, η υγρασία και η ένταση της υπεριώδους ακτινοβολίας (UV), με στόχο τη διασφάλιση της ποιότητας των προϊόντων και την υποστήριξη τεκμηριωμένων αποφάσεων.

Η διαδικασία ξεκινά με τη χρήση ενός αισθητήρα ESP32, ο οποίος είναι εξοπλισμένος με εξειδικευμένους αισθητήρες, όπως τον DHT11 για μέτρηση θερμοκρασίας και υγρασίας, και τον Waveshare UV Sensor (GUVA-S12SD) για την καταγραφή της υπεριώδους ακτινοβολίας. Ο ESP32 αναλαμβάνει την περιοδική συλλογή των δεδομένων, με συχνότητα ανά ένα λεπτό, ώστε να αποτυπώνεται μια λεπτομερής εικόνα των περιβαλλοντικών συνθηκών κατά τη διάρκεια της έκθεσης του προϊόντος.

Τα δεδομένα που συλλέγονται από τους αισθητήρες αποστέλλονται μέσω HTTP GET αιτημάτων σε έναν κεντρικό διακομιστή που έχει αναπτυχθεί σε Flask (Python). Το API του διακομιστή λαμβάνει τα δεδομένα, επαληθεύει την ακεραιότητά τους μέσω κρυπτογραφικών μεθόδων (hashing) και τα καταγράφει σε βάση δεδομένων MySQL. Στη βάση δεδομένων, αποθηκεύονται παράμετροι όπως η ταυτότητα του κόμβου (node ID), οι μετρήσεις θερμοκρασίας, υγρασίας, UV ακτινοβολίας, και το χρονικό στίγμα (timestamp) κάθε καταγραφής.

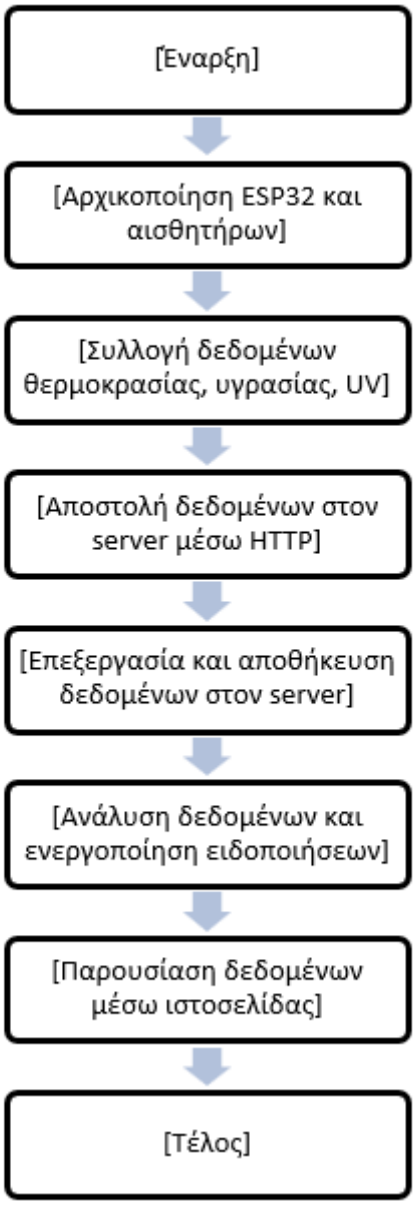
Για τη διασφάλιση της ακεραιότητας των δεδομένων και την αποφυγή μη εξουσιοδοτημένων παρεμβάσεων, το σύστημα ενσωματώνει τεχνολογία blockchain. Κάθε εγγραφή δεδομένων προστίθεται ως "block" σε μια αλυσίδα που καταγράφεται με τρόπο που εξασφαλίζει διαφάνεια και ανιχνευσιμότητα.

Η διαδικασία ολοκληρώνεται με την ανάλυση και την οπτικοποίηση των δεδομένων μέσω διαδραστικής ιστοσελίδας. Οι χρήστες μπορούν να επιλέξουν συγκεκριμένες ημερομηνίες από τα διαθέσιμα δεδομένα και να παρακολουθήσουν γραφήματα που απεικονίζουν τις περιβαλλοντικές συνθήκες σε πραγματικό χρόνο. Τα γραφήματα, που υλοποιούνται με Chart.js, παρέχουν λεπτομερείς αναλύσεις για κάθε μετρούμενη παράμετρο (θερμοκρασία, υγρασία, UV), καθώς και μια συνολική εικόνα όλων των δεδομένων.

Η ιστοσελίδα ενσωματώνει επίσης έναν διαδραστικό πίνακα δεδομένων (DataTable), όπου εμφανίζονται όλες οι καταγραφές με δυνατότητα ταξινόμησης και φιλτραρίσματος. Με αυτόν τον τρόπο, οι χρήστες έχουν πρόσβαση σε πλήρη ιστορικά δεδομένα, ενώ μπορούν να λάβουν τεκμηριωμένες αποφάσεις για τη βελτιστοποίηση της διαχείρισης των προϊόντων τους.

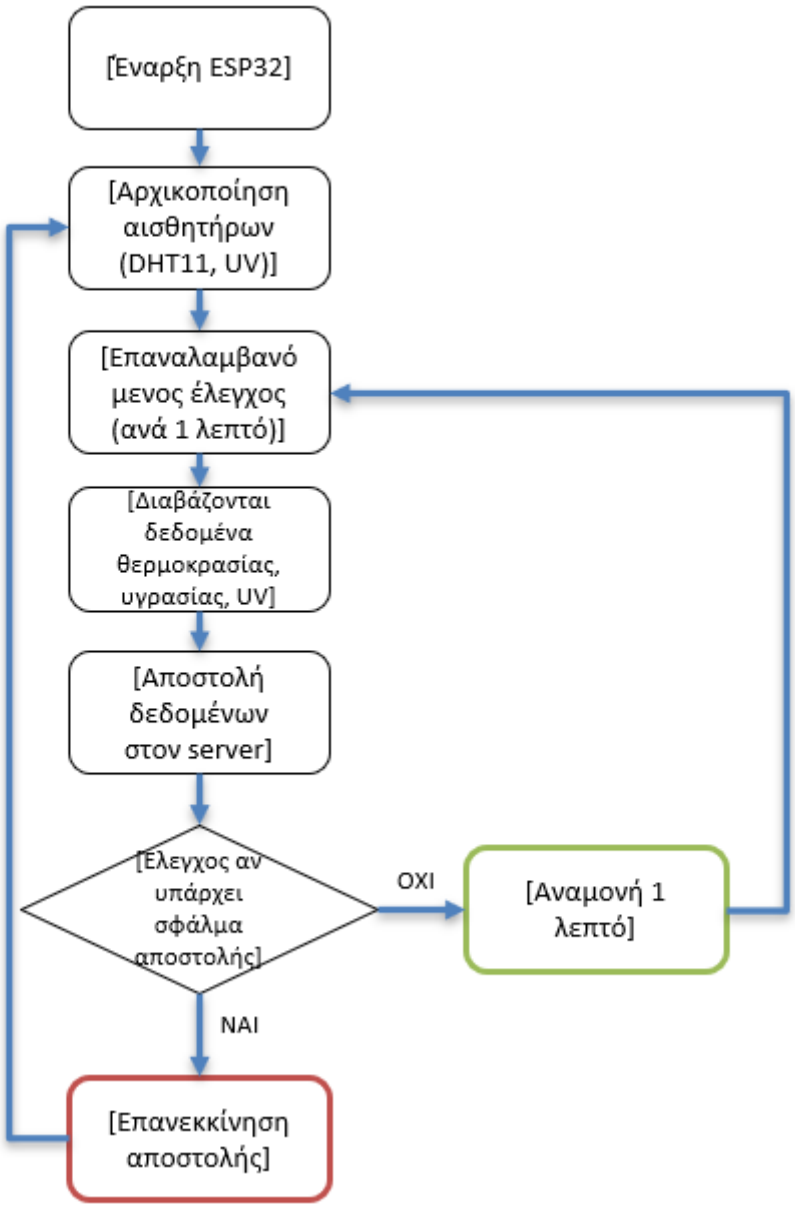
Η συνολική αρχιτεκτονική του συστήματος επιτυγχάνει την αυτοματοποίηση της συλλογής και ανάλυσης δεδομένων, προσφέροντας ένα εργαλείο υψηλής αξιοπιστίας και ευχρηστίας για επιχειρήσεις και ερευνητές. Η δυνατότητα λήψης real-time ειδοποιήσεων για κρίσιμα όρια συνθηκών εξασφαλίζει την έγκαιρη αντίδραση σε περιπτώσεις που η ποιότητα των προϊόντων απειλείται από ακραίες περιβαλλοντικές συνθήκες.

Η ανάπτυξη του συστήματος συνδυάζει την αξιοποίηση ανοιχτού λογισμικού, την εφαρμογή ασφαλών μεθόδων αποθήκευσης δεδομένων και την παρουσίαση φιλικών προς τον χρήστη διεπαφών, προσφέροντας μια ολοκληρωμένη λύση που μπορεί να προσαρμοστεί σε ποικίλες ανάγκες.



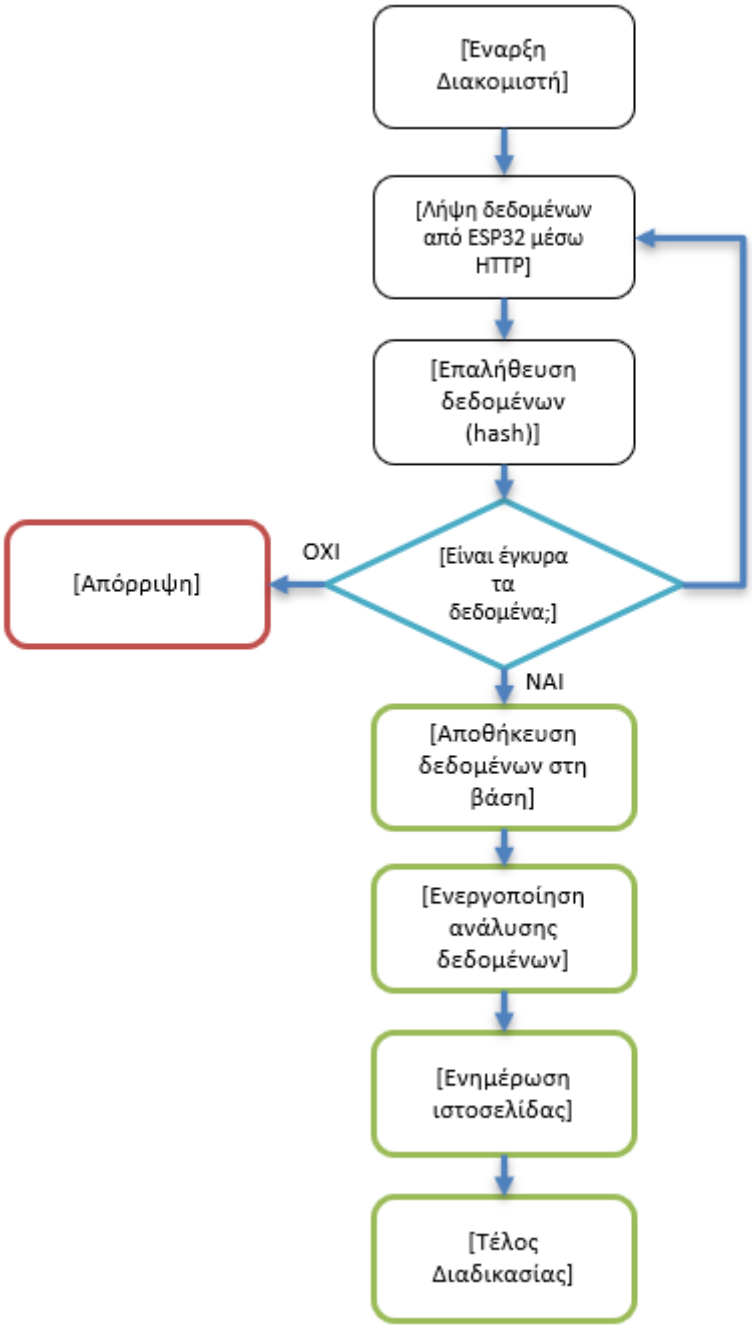

Εικόνα 4.1: Διάγραμμα του συστήματος

Το διάγραμμα στην Εικόνα 4.1 περιγράφει τη συνολική λειτουργία του συστήματος παρακολούθησης περιβαλλοντικών συνθηκών. Ξεκινά με την αρχικοποίηση του ESP32 και των αισθητήρων (θερμοκρασίας, υγρασίας και UV). Στη συνέχεια, τα δεδομένα συλλέγονται και αποστέλλονται στον server μέσω HTTP. Ο server επεξεργάζεται και αποθηκεύει τα δεδομένα στη βάση, ενώ πραγματοποιεί ανάλυση και, αν χρειαστεί, ενεργοποιεί ειδοποιήσεις. Τέλος, τα δεδομένα παρουσιάζονται μέσω ιστοσελίδας σε πραγματικό χρόνο.



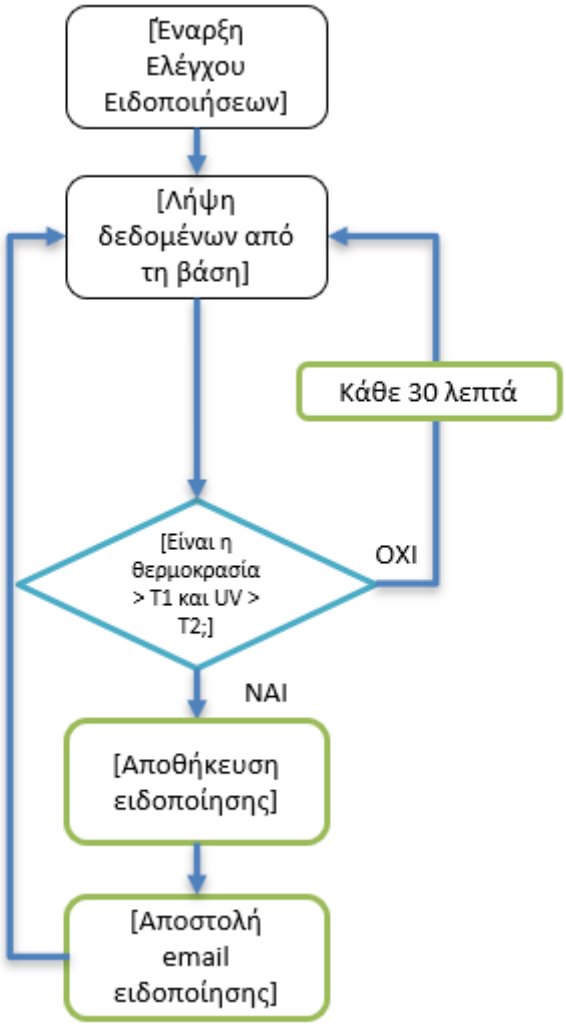

Εικόνα 4.2: Διάγραμμα λειτουργίας και διαδικασιών στο esp32

Το διάγραμμα στην Εικόνα 4.2 περιγράφει τη λειτουργία του ESP32. Η διαδικασία ξεκινά με την αρχικοποίηση του ESP32 και των συνδεδεμένων αισθητήρων (DHT11, UV). Το ESP32 εκτελεί έναν επαναλαμβανόμενο έλεγχο ανά λεπτό, όπου διαβάζει τις τιμές των αισθητήρων. Τα δεδομένα αποστέλλονται στον server μέσω HTTP. Αν διαπιστωθεί σφάλμα κατά την αποστολή, η διαδικασία επανεκκινείται. Διαφορετικά, το ESP32 περιμένει 1 λεπτό πριν επαναλάβει τον κύκλο.




Εικόνα 4.3: Διάγραμμα λειτουργίας και διαδικασιών στον python server

Το διάγραμμα στην Εικόνα 4.3 περιγράφει τη λειτουργία του διακομιστή. Ο server ξεκινά λαμβάνοντας δεδομένα από το ESP32 μέσω HTTP. Στη συνέχεια, επαληθεύει τα δεδομένα χρησιμοποιώντας hash για να διασφαλίσει την ακεραιότητά τους. Αν τα δεδομένα είναι έγκυρα, αποθηκεύονται στη βάση δεδομένων και ενεργοποιείται η διαδικασία ανάλυσης. Η ιστοσελίδα ενημερώνεται με τα νέα δεδομένα. Αν τα δεδομένα δεν είναι έγκυρα, απορρίπτονται.




Εικόνα 4.4: Διάγραμμα για τα Alerts

Στην Εικόνα 4.4 το διάγραμμα περιγράφει τη διαδικασία ελέγχου και αποστολής ειδοποιήσεων. Ο server λαμβάνει δεδομένα από τη βάση κάθε 30 λεπτά. Ελέγχει αν η θερμοκρασία και η UV υπερβαίνουν τα προκαθορισμένα όρια. Αν ναι, δημιουργείται μια ειδοποίηση, η οποία αποθηκεύεται στη βάση και αποστέλλεται μέσω email. Η διαδικασία συνεχίζεται σε κύκλους για συνεχή παρακολούθηση των συνθηκών.

### 4.1.1 Εξήγηση κώδικα esp32

Ο παρακάτω κώδικας αφορά την υλοποίηση του λογισμικού για τον κόμβο ESP32, ο οποίος χρησιμοποιεί αισθητήρες για τη συλλογή δεδομένων περιβαλλοντικών συνθηκών. Ο ESP32:

- Συνδέεται στο WiFi.
- Μετράει θερμοκρασία, υγρασία, και UV ένταση.
- Χρησιμοποιεί το RTC για την αποστολή χρονοσφραγισμένων δεδομένων.
- Δημιουργεί hash για την ασφάλεια των δεδομένων.
- Στέλνει δεδομένα στον server.

**Εισαγωγή Βιβλιοθηκών και Ορισμοί**

```
#include <WiFi.h>           // Βιβλιοθήκη για τη σύνδεση WiFi
#include <HTTPClient.h>     // Βιβλιοθήκη για HTTP αιτήματα
#include <Wire.h>           // I2C επικοινωνία
#include "RTClib.h"         // Βιβλιοθήκη για το RTC
#include "DHT.h"            // Βιβλιοθήκη για τον αισθητήρα DHT
#include "SHA256.h"         // Βιβλιοθήκη για την παραγωγή hash

// WiFi credentials
const char* ssid = "Your_SSID";      // Όνομα WiFi
const char* password = "Your_PASSWORD"; // Κωδικός WiFi

// Server URL
const char* serverUrl = "http://localhost:5000/api/data"; // URL του server

// Node ID
const char* nodeId = "Node_001"; // Μοναδικό αναγνωριστικό κόμβου ESP32

// DHT11 pin and type
#define DHTPIN 4  // Ακίδα σύνδεσης DHT
#define DHTTYPE DHT11  // Τύπος αισθητήρα DHT11
DHT dht(DHTPIN, DHTTYPE);
```

```
// UV sensor pin
#define UVPIN 36  // Ακίδα σύνδεσης UV αισθητήρα

// RTC Module
RTC_DS3231 rtc;  // RTC DS3231
```

**Επεξήγηση:**

- **WiFi**: Ρυθμίσεις για τη σύνδεση στο δίκτυο WiFi.
- **DHT11**: Χρησιμοποιείται για μέτρηση θερμοκρασίας και υγρασίας.
- **UVPIN**: Συνδέεται στον αισθητήρα UV.
- **RTC_DS3231**: Ρολόι πραγματικού χρόνου (Real Time Clock) για ακριβείς χρονοσφραγίδες.
- **nodeId**: Το αναγνωριστικό του κόμβου ESP32.

**setup**

```
void setup() {
  Serial.begin(115200);  // Έναρξη σειριακής επικοινωνίας
  WiFi.begin(ssid, password);  // Σύνδεση στο WiFi

  // Αναμονή για σύνδεση στο WiFi
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("WiFi connected!");

  // Αρχικοποίηση αισθητήρων
  dht.begin();  // DHT11
  pinMode(UVPIN, INPUT);  // UV αισθητήρας
```

```
  // Αρχικοποίηση RTC
  if (!rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }

  if (rtc.lostPower()) {
    Serial.println("RTC lost power, setting the time!");
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__))); // Ορισμός στην ώρα του compile
  }
}
```

**Επεξήγηση:**

- **WiFi**: Συνδέεται στο δίκτυο και εμφανίζει μηνύματα στη σειριακή κονσόλα.
- **Αισθητήρες**: Αρχικοποιεί τον DHT11 και τον UV αισθητήρα.
- **RTC**: Ελέγχει αν το RTC έχει χάσει την ώρα του και, αν χρειάζεται, την επαναφέρει.

**loop**

```
void loop() {
  // Read temperature and humidity
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  // Read UV intensity
  int uvAnalogValue = analogRead(UVPIN);
  float uvIntensity = uvAnalogValue * (3.3 / 4096.0); // Μετατροπή σε τάση
```

```
  // Get the current timestamp from RTC
  DateTime now = rtc.now();
  String timestamp = String(now.unixtime()); // Χρονική σφραγίδα σε μορφή Unix

  // Create data string
  String data = String("id=") + nodeId +
                "&temp=" + String(temperature, 2) +
                "&humidity=" + String(humidity, 2) +
                "&uv=" + String(uvIntensity, 2) +
                "&timestamp=" + timestamp;

  // Generate hash for security
  String hash = generateHash(data);

  // Append hash to data
  data += "&hash=" + hash;

  // Send data to server
  sendToServer(data);

  // Wait 1 minute before next reading
  delay(60000);
}
```

**Επεξήγηση:**

- **Μετρήσεις**:
    - Θερμοκρασία και υγρασία με τον DHT11.
    - Ένταση UV μετατρέπεται από αναλογική τιμή σε τάση.
- **Χρονοσφραγίδα**: Παράγεται με το RTC.

- **Συμβολοσειρά δεδομένων**: Δημιουργείται συμβολοσειρά που περιέχει τις μετρήσεις και το timestamp.
- **Hash**: Παράγεται hash για την ασφάλεια.
- **Αποστολή**: Καλείται η συνάρτηση αποστολής δεδομένων στον server.

**Αποστολή Δεδομένων**

```
void sendToServer(String data) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;  // Δημιουργία HTTP client
    String url = String(serverUrl) + "?" + data;

    http.begin(url);  // Έναρξη αιτήματος
    int httpResponseCode = http.GET();  // GET αίτημα

    if (httpResponseCode > 0) {
      Serial.println("Data sent successfully!");
      Serial.println(http.getString());
    } else {
      Serial.print("Error sending data: ");
      Serial.println(httpResponseCode);
    }

    http.end();  // Τέλος αιτήματος
  } else {
    Serial.println("WiFi not connected!");
  }
}
```

**Επεξήγηση:**

- **WiFi**: Ελέγχει αν είναι συνδεδεμένο.
- **HTTP Client**: Δημιουργεί σύνδεση και στέλνει GET αίτημα.
- **Αποτέλεσμα**: Εμφανίζει μηνύματα για την επιτυχία ή αποτυχία αποστολής.

**Δημιουργία Hash**

```
String generateHash(String data) {
 SHA256 hasher;  // Δημιουργία αντικειμένου SHA256
 hasher.update(data.c_str(), data.length());  // Ενημέρωση με τη συμβολοσειρά δεδομένων
 uint8_t* hashBytes = hasher.result();  // Υπολογισμός hash
 String hashString = "";

 for (int i = 0; i < 32; i++) {  // Μετατροπή hash σε συμβολοσειρά
  if (hashBytes[i] < 16) hashString += "0";
  hashString += String(hashBytes[i], HEX);
 }

 return hashString;  // Επιστροφή hash
}
```

**Επεξήγηση:**

- Χρησιμοποιεί τη βιβλιοθήκη SHA256 για να δημιουργήσει hash.
- Επιστρέφει το hash σε μορφή συμβολοσειράς.

### 4.1.2 Εξήγηση server κώδικα

Ο παρακάτω κώδικας αφορά την υλοποίηση ενός διακομιστή που χρησιμοποιεί το Flask framework για τη διαχείριση δεδομένων που λαμβάνονται από συσκευές ESP32. Ο διακομιστής προσφέρει λειτουργικότητες για:

- Λήψη δεδομένων από τους κόμβους (ESP32).
- Επαλήθευση της ακεραιότητας των δεδομένων.
- Αποθήκευση των δεδομένων στη βάση δεδομένων MySQL.
- Παροχή δεδομένων μέσω API.
- Απόκριση σε αιτήματα παρουσίασης μέσω ιστοσελίδας.

**Εισαγωγή Βιβλιοθηκών και Διαμόρφωση**

```
from flask import Flask, request, jsonify, render_template  # Flask framework για τον web server
import hashlib  # Για δημιουργία και επαλήθευση hash
import mysql.connector  # Για σύνδεση με MySQL
import json  # Για διαχείριση JSON δεδομένων


# Αρχικοποίηση του Flask app
app = Flask(__name__)


# Διαμόρφωση σύνδεσης με MySQL
db_config = {
    "host": "localhost",  # Διεύθυνση του server MySQL
    "user": "root",  # Χρήστης MySQL
    "password": "",  # Κωδικός MySQL (κενός εδώ)
    "database": "kivotia"  # Όνομα βάσης δεδομένων
}
```

```
# Μυστικό κλειδί για επαλήθευση hash
SECRET_KEY = "your_secret_key"
```

**Επεξήγηση:**

- **Βιβλιοθήκες**:
  - Flask: Χρησιμοποιείται για τη δημιουργία του web server.
  - hashlib: Για την κρυπτογραφική επαλήθευση των δεδομένων (hash).
  - mysql.connector: Χρησιμοποιείται για σύνδεση και αλληλεπίδραση με τη MySQL βάση δεδομένων.
  - json: Για τη διαχείριση JSON δεδομένων.
- **Διαμόρφωση**:
  - db_config: Περιλαμβάνει τις ρυθμίσεις σύνδεσης για τη βάση δεδομένων.
  - SECRET_KEY: Ένα μυστικό κλειδί για τη δημιουργία και επαλήθευση hash. Απαιτείται για την προστασία των δεδομένων από μη εξουσιοδοτημένες τροποποιήσεις.

**Επαλήθευση Hash**

```
def verify_hash(data, received_hash):
    """Επαληθεύει το hash χρησιμοποιώντας SHA256 και το μυστικό κλειδί."""
    hash_object = hashlib.sha256((data + SECRET_KEY).encode())  # Δημιουργία hash
    calculated_hash = hash_object.hexdigest()  # Υπολογισμός του hash
    return calculated_hash == received_hash  # Σύγκριση του υπολογισμένου με το παραληφθέν
```

**Επεξήγηση:**

- Αυτή η συνάρτηση επαληθεύει αν το hash που λαμβάνεται από τον ESP32 ταιριάζει με το hash που δημιουργείται στον server.
- **Διαδικασία**:
  1. Συνδυάζει τα δεδομένα με το SECRET_KEY.
  2. Δημιουργεί ένα SHA256 hash.
  3. Συγκρίνει το υπολογισμένο hash με αυτό που ελήφθη.

**Αποθήκευση στη Βάση Δεδομένων**

```
def save_to_database(active, node_id, temperature, humidity, uv_intensity, timestamp_from_node):
    """Αποθηκεύει δεδομένα στον πίνακα `sensordata`."""
    try:
        # Δημιουργία σύνδεσης με τη βάση
        connection = mysql.connector.connect(**db_config)
        cursor = connection.cursor()
        # Ερώτημα εισαγωγής δεδομένων
        query = """
        INSERT INTO sensordata (active, nodeid, temperature, humidity, uv_intensity, timestampfromnode)
        VALUES (%s, %s, %s, %s, %s, %s)
        """
        # Εκτέλεση του ερωτήματος
        cursor.execute(query, (active, node_id, temperature, humidity, uv_intensity, timestamp_from_node))
        connection.commit()  # Αποθήκευση αλλαγών στη βάση
        cursor.close()  # Κλείσιμο cursor
        connection.close()  # Κλείσιμο σύνδεσης
    except mysql.connector.Error as err:
        # Σε περίπτωση σφάλματος, εκτύπωση του μηνύματος
        print(f"Error: {err}")
        return False
    return True  # Επιστροφή επιτυχίας
```

**Επεξήγηση:**

- Αποθηκεύει τα δεδομένα που λαμβάνονται από τον ESP32 στον πίνακα sensordata της MySQL.
- Χρησιμοποιεί την παράμετρο active για να δηλώσει αν ο κόμβος είναι ενεργός.

- Κάθε εγγραφή περιλαμβάνει τον nodeid, θερμοκρασία, υγρασία, UV, και το χρονικό στίγμα (timestamp).

**Λήψη και Επεξεργασία Δεδομένων**

```
@app.route('/api/data', methods=['GET'])
def receive_data():
    try:
        # Ανάκτηση παραμέτρων από το query string
        node_id = request.args.get("id")
        temp = request.args.get("temp")
        humidity = request.args.get("humidity")
        uv = request.args.get("uv")
        timestamp = request.args.get("timestamp")
        received_hash = request.args.get("hash")

        # Έλεγχος για ελλιπείς παραμέτρους
        if not all([node_id, temp, humidity, uv, timestamp, received_hash]):
            return jsonify({"error": "Missing parameters"}), 400

        # Ανακατασκευή της αλφαριθμητικής ακολουθίας δεδομένων
        data = f"id={node_id}&temp={temp}&humidity={humidity}&uv={uv}&timestamp={timestamp}"

        # Επαλήθευση του hash
        if not verify_hash(data, received_hash):
            return jsonify({"error": "Hash verification failed"}), 403

        # Μετατροπή τύπων δεδομένων
```

```
    active = 1  # Ενεργή κατάσταση ως προεπιλογή
    node_id = int(node_id)  # Μετατροπή σε ακέραιο
    temp = float(temp)  # Μετατροπή σε δεκαδικό
    humidity = float(humidity)  # Μετατροπή σε δεκαδικό
    uv = float(uv)  # Μετατροπή σε δεκαδικό
    timestamp_from_node = timestamp  # Διατήρηση χρονικού στίγματος του ESP32

    # Αποθήκευση δεδομένων στη βάση
    if save_to_database(active, node_id, temp, humidity, uv, timestamp_from_node):
        return jsonify({"message": "Data saved successfully"}), 200
    else:
        return jsonify({"error": "Failed to save data"}), 500

  except Exception as e:
    # Διαχείριση εξαιρέσεων και επιστροφή του σφάλματος
    return jsonify({"error": str(e)}), 500
```

**Επεξήγηση:**

- **Λειτουργία**:
    - Λαμβάνει δεδομένα μέσω HTTP GET από τον ESP32.
    - Ελέγχει αν λείπουν υποχρεωτικές παράμετροι.
    - Επαληθεύει το hash για να διασφαλίσει την ακεραιότητα των δεδομένων.
    - Μετατρέπει τις παραμέτρους σε κατάλληλους τύπους δεδομένων.
    - Αποθηκεύει τα δεδομένα στη βάση.
    - Επιστρέφει απάντηση JSON.
- **Σφάλματα**:
    - Αν λείπουν δεδομένα ή η επαλήθευση αποτύχει, επιστρέφει κατάλληλο σφάλμα.

### 4.1.3 Εξήγηση κώδικας για τα alert

Ο παρακάτω κώδικας περιγράφει τη διαδικασία ανίχνευσης περιβαλλοντικών ειδοποιήσεων βάσει ορίων για θερμοκρασία και UV ένταση. Ενσωματώνει τις εξής λειτουργίες:

- Ανάκτηση δεδομένων από τη βάση για συγκεκριμένη ημέρα.
- Ανίχνευση ειδοποιήσεων βάσει προϋποθέσεων.
- Αποθήκευση ειδοποιήσεων στη βάση δεδομένων.
- Αποστολή email ειδοποιήσεων στους παραλήπτες.

**Διαμόρφωση Συστήματος**

```
import mysql.connector
from datetime import datetime, timedelta
import smtplib
from email.mime.text import MIMEText


# MySQL configuration
db_config = {
    "host": "localhost",
    "user": "root",
    "password": "",
    "database": "kivotia"
}


# Email configuration
EMAIL_HOST = "smtp.gmail.com"
EMAIL_PORT = 587
EMAIL_USER = "your_email@gmail.com"
```

```
EMAIL_PASS = "your_email_password"
ALERT_EMAIL = "alert_recipient@gmail.com"


# Thresholds
TEMP_THRESHOLD = 20.0  # Όριο για θερμοκρασία
UV_THRESHOLD = 0.5  # Όριο για UV ένταση
TIME_THRESHOLD_MINUTES = 30  # Ελάχιστη διάρκεια υπέρβασης για ειδοποίηση
```

**Επεξήγηση:**

- **Διαμόρφωση Βάσης**: Ρυθμίσεις σύνδεσης με τη MySQL.
- **Ρυθμίσεις Email**: Περιλαμβάνουν τον SMTP server, το email αποστολής και τον παραλήπτη ειδοποιήσεων.
- **Όρια Ειδοποίησης**:
    - TEMP_THRESHOLD: Θερμοκρασία πάνω από 20°C.
    - UV_THRESHOLD: UV ένταση πάνω από 0.5 V.
    - TIME_THRESHOLD_MINUTES: Ελάχιστη διάρκεια υπέρβασης (30 λεπτά).

**Ανάκτηση Δεδομένων**

```
def fetch_day_data(date):
    """Ανακτά δεδομένα για συγκεκριμένη ημέρα."""
    try:
        connection = mysql.connector.connect(**db_config)
        cursor = connection.cursor(dictionary=True)
        query = """
            SELECT nodeid, temperature, uv_intensity, timestampfromnode
            FROM sensordata
            WHERE DATE(timestampfromnode) = %s
            ORDER BY timestampfromnode
```

```
        """
        cursor.execute(query, (date,))
        data = cursor.fetchall()
        cursor.close()
        connection.close()
        return data
    except mysql.connector.Error as err:
        print(f"Error fetching data: {err}")
        return []
```

**Επεξήγηση:**

- **Λειτουργία**:
    - Συνδέεται στη βάση δεδομένων.
    - Ανακτά δεδομένα από τον πίνακα sensordata για τη συγκεκριμένη ημερομηνία.
    - Επιστρέφει τα δεδομένα σε μορφή λίστας λεξικών.
- **Χρήση**: Τα δεδομένα θα χρησιμοποιηθούν για την ανίχνευση ειδοποιήσεων.

**Ανίχνευση Ειδοποιήσεων**

```
def detect_alerts(data):
    """Ανιχνεύει ειδοποιήσεις βάσει ορίων."""
    alerts = []
    start_time = None
    duration_minutes = 0
    node_id = None

    for i, row in enumerate(data):
        temp = row["temperature"]
        uv = row["uv_intensity"]
```

```
    timestamp = datetime.strptime(row["timestampfromnode"], "%Y-%m-%d %H:%M:%S")

    if temp > TEMP_THRESHOLD and uv > UV_THRESHOLD:
        if not start_time:
            start_time = timestamp
            node_id = row["nodeid"]
        duration_minutes += 1
    else:
        if start_time and duration_minutes >= TIME_THRESHOLD_MINUTES:
            alerts.append({
                "nodeid": node_id,
                "temperature": temp,
                "uv_intensity": uv,
                "start_time": start_time,
                "end_time": timestamp - timedelta(minutes=1),
                "duration_minutes": duration_minutes
            })
        start_time = None
        duration_minutes = 0

# Αν υπάρχουν συνεχιζόμενες ειδοποιήσεις
if start_time and duration_minutes >= TIME_THRESHOLD_MINUTES:
    alerts.append({
        "nodeid": node_id,
        "temperature": temp,
        "uv_intensity": uv,
        "start_time": start_time,
        "end_time": timestamp,
```

```
            "duration_minutes": duration_minutes
        })

    return alerts
```

**Επεξήγηση:**

- **Λειτουργία**:
    - Ανιχνεύει περιόδους όπου θερμοκρασία και UV υπερβαίνουν τα όρια για περισσότερο από 30 λεπτά.
    - Ανιχνεύει και καταγράφει την έναρξη, τη λήξη και τη διάρκεια της υπέρβασης.
- **Επιστροφή**: Μια λίστα με τις ειδοποιήσεις που ανιχνεύθηκαν.

**Αποθήκευση Ειδοποιήσεων**

```
def save_alerts(alerts):
    """Αποθηκεύει ειδοποιήσεις στη βάση δεδομένων."""
    try:
        connection = mysql.connector.connect(**db_config)
        cursor = connection.cursor()
        query = """
            INSERT INTO alerts (nodeid, temperature, uv_intensity, start_time, end_time, duration_minutes)
            VALUES (%s, %s, %s, %s, %s, %s)
        """
        for alert in alerts:
            cursor.execute(query, (
                alert["nodeid"],
                alert["temperature"],
                alert["uv_intensity"],
```

```
            alert["start_time"],
            alert["end_time"],
            alert["duration_minutes"]
        ))
    connection.commit()
    cursor.close()
    connection.close()
  except mysql.connector.Error as err:
    print(f"Error saving alerts: {err}")
```

**Επεξήγηση:**

- **Αποθήκευση**: Οι ανιχνευθείσες ειδοποιήσεις αποθηκεύονται στον πίνακα alerts.
- **Στήλες**: Περιλαμβάνουν το Node ID, θερμοκρασία, UV ένταση, χρονικό διάστημα και διάρκεια.

**Αποστολή Email**

```
def send_email(alerts):
  """Αποστέλλει email για κάθε ειδοποίηση."""
  try:
    server = smtplib.SMTP(EMAIL_HOST, EMAIL_PORT)
    server.starttls()
    server.login(EMAIL_USER, EMAIL_PASS)

    for alert in alerts:
      subject = "Environmental Alert Detected"
      body = (
        f"Node ID: {alert['nodeid']}\n"
        f"Temperature: {alert['temperature']}°C\n"
```

```
            f"UV Intensity: {alert['uv_intensity']} V\n"
            f"Duration: {alert['duration_minutes']} minutes\n"
            f"Start Time: {alert['start_time']}\n"
            f"End Time: {alert['end_time']}\n"
        )
        msg = MIMEText(body)
        msg["Subject"] = subject
        msg["From"] = EMAIL_USER
        msg["To"] = ALERT_EMAIL

        server.sendmail(EMAIL_USER, ALERT_EMAIL, msg.as_string())

    server.quit()
except Exception as e:
    print(f"Error sending email: {e}")
```

**Επεξήγηση:**

- **Λειτουργία**: Δημιουργεί email με λεπτομέρειες κάθε ειδοποίησης και το αποστέλλει.
- **SMTP**: Χρησιμοποιεί τον SMTP server για την αποστολή.

**Εκτέλεση**

```
def main():
    date = datetime.now().strftime("%Y-%m-%d")  # Ημερομηνία τρέχουσας ημέρας
    data = fetch_day_data(date)  # Ανάκτηση δεδομένων
    alerts = detect_alerts(data)  # Ανίχνευση ειδοποιήσεων

    if alerts:
        save_alerts(alerts)  # Αποθήκευση ειδοποιήσεων
```

```
    send_email(alerts)  # Αποστολή ειδοποιήσεων


if __name__ == "__main__":
   main()
```

**Επεξήγηση:**

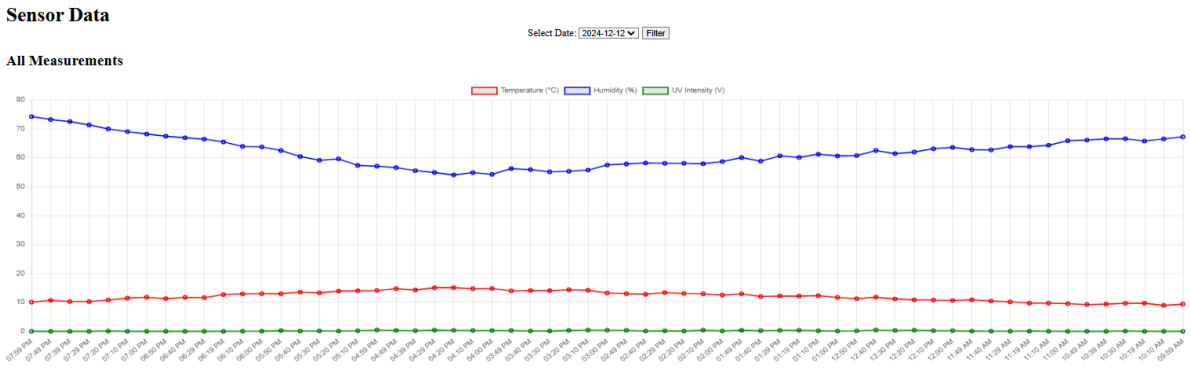
- Η κύρια συνάρτηση:
    - Ανακτά δεδομένα για την τρέχουσα ημέρα.
    - Ανιχνεύει ειδοποιήσεις.
    - Αποθηκεύει και αποστέλλει ειδοποιήσεις μέσω email.

## 4.2 Η ιστοσελίδα και οι μετρήσεις

Η ιστοσελίδα που αναπτύχθηκε αποτελεί ένα βασικό εργαλείο προβολής των δεδομένων που συλλέγονται από το σύστημα μέτρησης. Αποτελείται από τρεις κύριες λειτουργικές ενότητες:

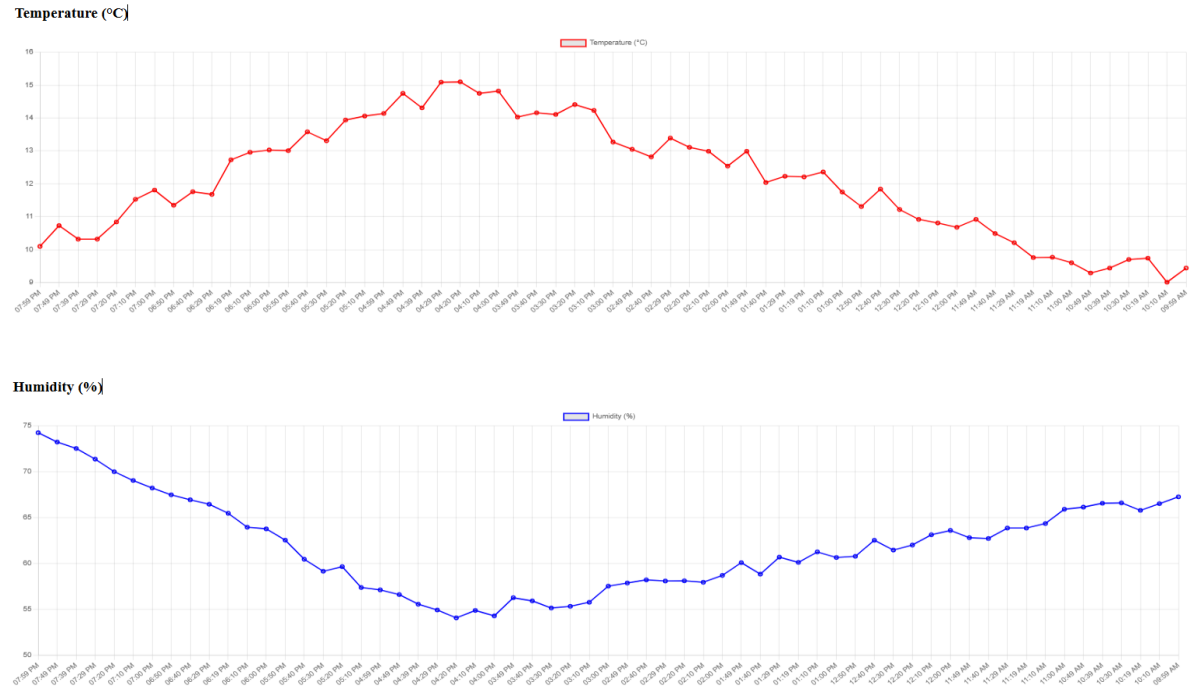
1. Προβολή γραφημάτων όλων των μετρήσεων (θερμοκρασία, υγρασία, UV) για την επιλεγμένη ημέρα.
2. Προβολή γραφημάτων ανά παράμετρο (θερμοκρασία, υγρασία ή UV) για την επιλεγμένη ημέρα.
3. Προβολή δεδομένων σε πίνακα DataTable, ο οποίος υποστηρίζει ταξινόμηση και φιλτράρισμα ανά ημέρα.

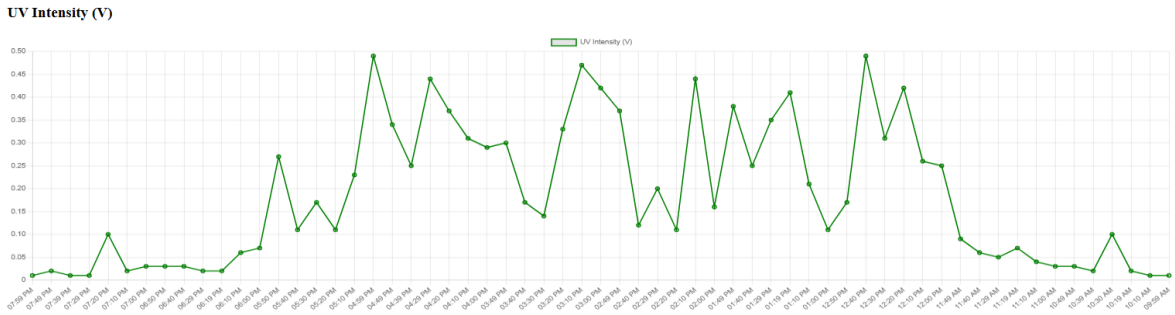
Ο χρήστης μπορεί να επιλέξει την ημερομηνία που επιθυμεί μέσω μιας λίστας ημερομηνιών, η οποία αντλείται δυναμικά από τη βάση δεδομένων. Με αυτόν τον τρόπο, παρέχεται μια ολοκληρωμένη επισκόπηση των περιβαλλοντικών συνθηκών που καταγράφηκαν.




Εικόνα 4.5: Ιστοσελίδα προβολής του γραφήματος όλων των δεδομένων ανά ημέρα για θερμοκρασία, υγρασία και UV– επιλογή της μέρας από λίστα

Στην Εικόνα 4.5, απεικονίζεται η κύρια λειτουργία της ιστοσελίδας, όπου ο χρήστης μπορεί να επιλέξει μια ημερομηνία από τη λίστα στο επάνω μέρος της σελίδας. Με την επιλογή της ημερομηνίας, φορτώνονται τα δεδομένα της συγκεκριμένης ημέρας και προβάλλονται σε ένα συνδυασμένο γράφημα. Στο γράφημα παρουσιάζονται τα δεδομένα θερμοκρασίας, υγρασίας και UV σε διαφορετικές χρωματικές γραμμές, παρέχοντας μια συνολική εικόνα των περιβαλλοντικών συνθηκών για την ημέρα.






Εικόνα 4.6: Ιστοσελίδα προβολής του γραφήματος δεδομένων ανά ημέρα  ανά παράμετρο

Στην Εικόνα 4.6, παρουσιάζεται η δυνατότητα προβολής των δεδομένων ανά παράμετρο (θερμοκρασία, υγρασία ή UV). Ο χρήστης μπορεί να επιλέξει μία από τις παραμέτρους και να δει το αντίστοιχο γράφημα. Αυτή η λειτουργία είναι χρήσιμη για την ανάλυση συγκεκριμένων παραμέτρων, καθώς επιτρέπει στον χρήστη να εξετάσει λεπτομερώς την πορεία μιας συγκεκριμένης μέτρησης κατά τη διάρκεια της ημέρας.

**Data Table**

Show 10 entries Search:

| Node ID | Temperature (°C) | Humidity (%) | UV Intensity (V) | Timestamp |
|---|---|---|---|---|
| 1 | 10.1 | 74.26 | 0.01 | Thu, 12 Dec 2024 17:59:58 GMT |
| 1 | 10.73 | 73.24 | 0.02 | Thu, 12 Dec 2024 17:49:59 GMT |
| 1 | 10.32 | 72.54 | 0.01 | Thu, 12 Dec 2024 17:39:58 GMT |
| 1 | 10.32 | 71.38 | 0.01 | Thu, 12 Dec 2024 17:29:59 GMT |
| 1 | 10.84 | 70.01 | 0.1 | Thu, 12 Dec 2024 17:20:01 GMT |
| 1 | 11.53 | 69.05 | 0.02 | Thu, 12 Dec 2024 17:10:01 GMT |
| 1 | 11.81 | 68.23 | 0.03 | Thu, 12 Dec 2024 17:00:01 GMT |
| 1 | 11.35 | 67.49 | 0.03 | Thu, 12 Dec 2024 16:50:01 GMT |
| 1 | 11.76 | 66.95 | 0.03 | Thu, 12 Dec 2024 16:40:02 GMT |
| 1 | 11.68 | 66.46 | 0.02 | Thu, 12 Dec 2024 16:29:59 GMT |

Showing 1 to 10 of 61 entries Previous 1 2 3 4 5 6 7 Next

Εικόνα 4.7: Προβολή πίνακα datatable δεδομένων ανά ημέρα

Στην Εικόνα 4.7, εμφανίζεται η λειτουργικότητα του πίνακα δεδομένων (DataTable). Ο πίνακας αυτός περιλαμβάνει όλες τις εγγραφές δεδομένων για την επιλεγμένη ημερομηνία, με στήλες για:

- Το Node ID
- Τη θερμοκρασία
- Την υγρασία
- Την ένταση UV
- Το χρονικό στίγμα (timestamp)

Ο πίνακας υποστηρίζει ταξινόμηση και φιλτράρισμα, διευκολύνοντας τον χρήστη να εντοπίσει συγκεκριμένα δεδομένα ή μοτίβα. Επιπλέον, η χρήση του DataTable παρέχει μια εύχρηστη και οπτικά ελκυστική μέθοδο παρουσίασης μεγάλου όγκου δεδομένων.

## 4.3 Οι πίνακες της βάσης δεδομένων

Η MySQL είναι ένα από τα πιο δημοφιλή συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων (RDBMS). Χρησιμοποιείται ευρέως για την αποθήκευση, ανάκτηση και διαχείριση δεδομένων σε διάφορες εφαρμογές, από απλές ιστοσελίδες έως περίπλοκα συστήματα. Στο παρόν έργο, η MySQL χρησιμοποιείται για την αποθήκευση και την ανάλυση των δεδομένων που συλλέγονται από αισθητήρες, καθώς και για τη διαχείριση ειδοποιήσεων που προκύπτουν από υπερβάσεις ορίων. Η δομή της βάσης είναι σχεδιασμένη ώστε να υποστηρίζει την αποθήκευση δεδομένων αισθητήρων, την ανίχνευση ειδοποιήσεων και την παρουσίαση των δεδομένων σε ιστοσελίδα.

Η βάση δεδομένων αποτελείται από δύο κύριους πίνακες: τον πίνακα sensordata για την αποθήκευση δεδομένων από αισθητήρες και τον πίνακα alerts για την αποθήκευση ειδοποιήσεων. Ακολουθεί η περιγραφή κάθε πίνακα:

### 4.3.1 Πίνακας sensordata

Ο πίνακας sensordata αποθηκεύει δεδομένα που συλλέγονται από τους αισθητήρες. Κάθε εγγραφή αντιπροσωπεύει μια μεμονωμένη μέτρηση από έναν κόμβο (ESP32).

**Δομή Πεδίου:**

- **id**: Ο μοναδικός αναγνωριστικός αριθμός για κάθε εγγραφή (πρωτεύον κλειδί).
- **active**: Δείκτης που υποδεικνύει αν ο κόμβος είναι ενεργός (1 για ενεργό).
- **nodeid**: Ο μοναδικός αριθμός ταυτοποίησης του κόμβου (ESP32).
- **temperature**: Η θερμοκρασία που μετρήθηκε από τον κόμβο (σε βαθμούς Κελσίου).
- **humidity**: Η υγρασία που μετρήθηκε από τον κόμβο (ως ποσοστό %).
- **uv_intensity**: Η ένταση UV που μετρήθηκε (σε Volt).
- **timestampfromnode**: Η χρονική σφραγίδα που καταγράφηκε από τον κόμβο κατά τη μέτρηση.
- **created_at**: Η χρονική σφραγίδα δημιουργίας της εγγραφής στη βάση δεδομένων.

**Χαρακτηριστικά:**

- Καταγράφει δεδομένα σε πραγματικό χρόνο για πολλαπλές παραμέτρους (θερμοκρασία, υγρασία, UV).
- Το πεδίο timestampfromnode διασφαλίζει ότι η εγγραφή έχει ακριβή χρονική σφραγίδα από τον αισθητήρα.

### 4.3.2 Πίνακας alerts

Ο πίνακας alerts αποθηκεύει τις ειδοποιήσεις που δημιουργούνται όταν υπερβαίνονται τα καθορισμένα όρια για θερμοκρασία ή UV ένταση. Κάθε εγγραφή αντιπροσωπεύει μια περίοδο υπέρβασης.

**Δομή Πεδίου:**

- **id**: Ο μοναδικός αναγνωριστικός αριθμός για κάθε ειδοποίηση (πρωτεύον κλειδί).
- **nodeid**: Ο αριθμός ταυτοποίησης του κόμβου που προκάλεσε την ειδοποίηση.
- **temperature**: Η θερμοκρασία που μετρήθηκε κατά την έναρξη της ειδοποίησης.
- **uv_intensity**: Η UV ένταση που μετρήθηκε κατά την έναρξη της ειδοποίησης.
- **start_time**: Η χρονική στιγμή που ξεκίνησε η υπέρβαση των ορίων.
- **end_time**: Η χρονική στιγμή που έληξε η υπέρβαση των ορίων.
- **duration_minutes**: Η συνολική διάρκεια της υπέρβασης, σε λεπτά.
- **created_at**: Η χρονική σφραγίδα δημιουργίας της εγγραφής στη βάση δεδομένων.

**Χαρακτηριστικά:**

- Καταγράφει τις υπερβάσεις θερμοκρασίας και UV έντασης.
- Περιλαμβάνει πληροφορίες για τη διάρκεια και τις ακριβείς χρονικές στιγμές της υπέρβασης.
- Είναι σχεδιασμένος να χρησιμοποιείται για ειδοποιήσεις, ανάλυση δεδομένων και στατιστική παρακολούθηση.

Η σχεδίαση της βάσης δεδομένων είναι απλή αλλά αποδοτική, υποστηρίζοντας τη συλλογή, την αποθήκευση και την ανάλυση δεδομένων σε πραγματικό χρόνο. Ο πίνακας sensordata λειτουργεί ως η κύρια πηγή πληροφοριών για τις μετρήσεις, ενώ ο πίνακας alerts αποθηκεύει σημαντικά συμβάντα που απαιτούν περαιτέρω ανάλυση ή ειδοποίηση. Με αυτήν τη δομή, η βάση δεδομένων παρέχει τη δυνατότητα τόσο για παρακολούθηση σε πραγματικό χρόνο όσο και για ιστορική ανάλυση.

## 4.4 Ασφάλεια συστήματος και δεδομένων

Η ασφάλεια συστήματος και δεδομένων αποτελεί έναν από τους πιο κρίσιμους παράγοντες για τη σωστή λειτουργία και αξιοπιστία του συστήματος μέτρησης περιβαλλοντικών δεδομένων. Δεδομένου ότι το σύστημα περιλαμβάνει την αποστολή δεδομένων από τους αισθητήρες στον server, την αποθήκευση σε βάση δεδομένων, και την προβολή τους μέσω ιστοσελίδας, είναι απαραίτητο να διασφαλίζεται η ακεραιότητα, η εμπιστευτικότητα και η διαθεσιμότητα των δεδομένων σε όλα τα στάδια της διαδικασίας.

### 4.4.1 Επαλήθευση Δεδομένων και Ακεραιότητα

Η ακεραιότητα των δεδομένων είναι καθοριστικής σημασίας για την αξιοπιστία του συστήματος. Η διαδικασία που εφαρμόστηκε για την επαλήθευση των δεδομένων περιλαμβάνει τα εξής:

- **Hash Verification (SHA-256):**

  Πριν τα δεδομένα αποσταλούν από τον κόμβο ESP32 στον server, δημιουργείται ένα κρυπτογραφικό hash χρησιμοποιώντας τον αλγόριθμο SHA-256. Το hash υπολογίζεται λαμβάνοντας υπόψη τα δεδομένα και ένα μυστικό κλειδί (secret key). Όταν τα δεδομένα φτάσουν στον server, το hash επαληθεύεται, ώστε να διασφαλιστεί ότι τα δεδομένα δεν έχουν αλλοιωθεί κατά τη μεταφορά.

- **Μηχανισμός Αντίστασης σε Επεμβάσεις:**

  Με τη χρήση του hash και του μυστικού κλειδιού, αποτρέπεται οποιαδήποτε μη εξουσιοδοτημένη παρέμβαση ή αλλοίωση στα δεδομένα από κακόβουλους τρίτους.

### 4.4.2 Ασφάλεια Βάσης Δεδομένων

Η βάση δεδομένων MySQL, που χρησιμοποιείται για την αποθήκευση των δεδομένων, προστατεύεται από τα εξής μέτρα ασφαλείας:

- **Περιορισμένη Πρόσβαση:**

  Η πρόσβαση στη βάση δεδομένων επιτρέπεται μόνο από τον server που φιλοξενεί την εφαρμογή. Οι χρήστες πρέπει να έχουν έγκυρα διαπιστευτήρια για να αποκτήσουν πρόσβαση.

- **Δικαιώματα Χρήστη:**

  Οι χρήστες της βάσης δεδομένων έχουν περιορισμένα δικαιώματα, ώστε να αποτρέπονται μη εξουσιοδοτημένες αλλαγές ή διαγραφές. Ο χρήστης που συνδέεται έχει μόνο δικαίωμα

ανάγνωσης και εγγραφής, αποφεύγοντας επικίνδυνες λειτουργίες, όπως η τροποποίηση της δομής της βάσης.

- **Κρυπτογράφηση Δεδομένων:**

  Ενσωματώθηκε η δυνατότητα κρυπτογράφησης συγκεκριμένων δεδομένων, ειδικά για την αποθήκευση ευαίσθητων πληροφοριών.

### 4.4.3 Ασφάλεια Email Ειδοποιήσεων

Για την αποστολή email ειδοποιήσεων χρησιμοποιείται ο SMTP server της Google (Gmail). Τα μέτρα που ελήφθησαν περιλαμβάνουν:

- **Χρήση TLS:**

  Όλες οι επικοινωνίες με τον SMTP server είναι κρυπτογραφημένες με το πρωτόκολλο TLS.

- **Ασφαλής Αποθήκευση Κωδικών:**

  Τα διαπιστευτήρια email αποθηκεύονται με ασφάλεια, ώστε να αποτρέπονται μη εξουσιοδοτημένες προσβάσεις.

- **Επαλήθευση Παραλήπτη:**

  Τα email αποστέλλονται μόνο σε καθορισμένους παραλήπτες για την αποφυγή διαρροής δεδομένων.

Η υλοποίηση των παραπάνω μέτρων ασφαλείας εξασφαλίζει ότι το σύστημα είναι ανθεκτικό σε επιθέσεις και διασφαλίζει την ακεραιότητα και την εμπιστευτικότητα των δεδομένων. Παρόλο που το παρόν σύστημα σχεδιάστηκε για πειραματική χρήση, οι αρχές ασφαλείας που υιοθετήθηκαν μπορούν να εφαρμοστούν και σε μεγαλύτερης κλίμακας υλοποιήσεις. Η μελλοντική επέκταση μπορεί να περιλαμβάνει περαιτέρω μέτρα, όπως αυθεντικοποίηση χρηστών και παρακολούθηση ασφαλείας σε πραγματικό χρόνο.

# Κεφάλαιο 5ο: Τα συμπεράσματα και οι βελτιώσεις

Η παρούσα εργασία είχε ως στόχο την ανάπτυξη ενός ολοκληρωμένου συστήματος για την παρακολούθηση και ανάλυση περιβαλλοντικών συνθηκών, με έμφαση στην ασφάλεια των δεδομένων και τη χρηστικότητα του συστήματος. Μέσα από τη διαδικασία ανάπτυξης και δοκιμής του συστήματος, προέκυψαν τα εξής βασικά συμπεράσματα: Το σύστημα λειτούργησε με επιτυχία, καταγράφοντας δεδομένα θερμοκρασίας, υγρασίας και UV έντασης. Η αποστολή δεδομένων από τους κόμβους ESP32 στον server και η αποθήκευσή τους στη βάση δεδομένων πραγματοποιήθηκε ομαλά. Η ενσωμάτωση μηχανισμών ασφαλείας, όπως το hash verification και τα κρυπτογραφημένα δεδομένα κατά τη μεταφορά, διασφάλισε την ακεραιότητα και την εμπιστευτικότητα των δεδομένων. Η ιστοσελίδα που αναπτύχθηκε παρείχε ένα φιλικό περιβάλλον για την ανάλυση των δεδομένων μέσω γραφημάτων και πίνακα δεδομένων, επιτρέποντας τη γρήγορη εξαγωγή χρήσιμων συμπερασμάτων. Η δυνατότητα ανίχνευσης και ειδοποίησης για υπερβάσεις ορίων λειτούργησε αποτελεσματικά, παρέχοντας ειδοποιήσεις σε πραγματικό χρόνο τόσο μέσω της βάσης δεδομένων όσο και μέσω email. Το σύστημα ήταν εύκολο στη χρήση και την εγκατάσταση, γεγονός που το καθιστά ιδανικό για πειραματικές και πρακτικές εφαρμογές. Η δομή του συστήματος επιτρέπει την εύκολη επεκτασιμότητα, τόσο σε επίπεδο αισθητήρων όσο και σε επίπεδο λειτουργιών, όπως η προσθήκη νέων παραμέτρων ή νέων ειδοποιήσεων.

Παρόλο που το σύστημα πέτυχε τους βασικούς στόχους του, υπάρχουν περιθώρια για βελτίωση και περαιτέρω ανάπτυξη. Η προσθήκη επιπλέον αισθητήρων, όπως για μέτρηση $CO_2$ ή σωματιδίων PM2.5, θα μπορούσε να επεκτείνει τη λειτουργικότητα του συστήματος σε περισσότερες εφαρμογές. Η ενσωμάτωση μηχανισμού αυθεντικοποίησης για την ιστοσελίδα και τα API θα ενισχύσει την ασφάλεια, περιορίζοντας την πρόσβαση σε εξουσιοδοτημένους χρήστες. Η χρήση HTTPS για την αποστολή δεδομένων από τον ESP32 στον server θα μπορούσε να προσφέρει επιπλέον επίπεδο ασφάλειας, ιδιαίτερα σε περιβάλλοντα με δημόσια δίκτυα. Η ενσωμάτωση με IoT πλατφόρμες, όπως το AWS IoT ή το Google Cloud IoT, θα επέτρεπε την επεξεργασία δεδομένων σε μεγαλύτερη κλίμακα και την αποθήκευσή τους σε cloud περιβάλλοντα. Η χρήση μηχανικής μάθησης για την ανάλυση των δεδομένων θα μπορούσε να προσφέρει προγνωστικές αναλύσεις, όπως πρόβλεψη υπερβάσεων ορίων ή ανίχνευση ανωμαλιών. Η δημιουργία αυτόματων αναφορών για τα δεδομένα και τις ειδοποιήσεις θα μπορούσε να διευκολύνει τη χρήση του συστήματος από τελικούς χρήστες. Η προσθήκη γραφημάτων σε πραγματικό χρόνο και διαδραστικών φίλτρων θα βελτίωνε την εμπειρία χρήστη. Η ενσωμάτωση ενός μηχανισμού buffer στον ESP32, για την αποθήκευση δεδομένων σε περίπτωση απώλειας σύνδεσης, θα μπορούσε να διασφαλίσει την ακεραιότητα των δεδομένων. Η εγκατάσταση του συστήματος σε πραγματικές συνθήκες, όπως βιομηχανικές μονάδες ή αγροτικές εκτάσεις, θα μπορούσε να αποδείξει την πρακτική του εφαρμογή και την αξιοπιστία του. Τέλος, η ανάπτυξη εφαρμογών για

φορητές συσκευές (mobile apps) θα διευκόλυνε την παρακολούθηση δεδομένων από τους χρήστες σε πραγματικό χρόνο. Σημειώνεται ότι στη διάρκεια της εργασίας αξιοποιήθηκε πλατφόρμα ai για την επεξεργασία και τη βελτίωση του συντακτικού και της μορφοποίησης του κειμένου.

Η εργασία αυτή έδειξε τη δυναμική που έχουν τα συστήματα IoT για την παρακολούθηση και ανάλυση περιβαλλοντικών δεδομένων. Η σχεδίαση ενός συστήματος που συνδυάζει χαμηλό κόστος, ευκολία χρήσης και επεκτασιμότητα αποτελεί ένα σημαντικό βήμα προς τη διαχείριση δεδομένων σε διάφορους τομείς. Με την εφαρμογή των προτάσεων βελτίωσης, το σύστημα μπορεί να γίνει πιο ισχυρό, ευέλικτο και λειτουργικό, επιτρέποντας την ενσωμάτωσή του σε επαγγελματικές και ερευνητικές εφαρμογές.

# ΒΙΒΛΙΟΓΡΑΦΙΑ


[1] https://www.adaptideations.com/the-impact-of-iot-technology-on-cold-chain-management-for-food-logistics-providers

[2] Urbano, O., Perles, A., Pedraza, C., Rubio-Arraez, S., Castelló, M. L., Ortola, M. D., & Mercado, R. (2020). Cost-Effective Implementation of a Temperature Traceability System Based on Smart RFID Tags and IoT Services. Sensors, 20(4), 1163. https://doi.org/10.3390/s20041163

[3] Cil, A.Y., Abdurahman, D. & Cil, I. Internet of Things enabled real time cold chain monitoring in a container port. J. shipp. trd. 7, 9 (2022). https://doi.org/10.1186/s41072-022-00110-z

[4] Espressif Systems, "ESP32: The Next Generation of IoT Development," διαθέσιμο από: https://www.espressif.com

[5] Adafruit Industries, "DHT11 Temperature and Humidity Sensor," διαθέσιμο από: https://www.adafruit.com

[6] Waveshare Electronics, "UV Sensor GUVA-S12SD," διαθέσιμο από: https://www.waveshare.com

[7] Maxim Integrated, "DS3231 Real-Time Clock," διαθέσιμο από: https://www.maximintegrated.com

[8] Arduino, "Arduino IDE," διαθέσιμο από: https://www.arduino.cc

[9] Python Software Foundation, "Python Programming Language," διαθέσιμο από: https://www.python.org

[10] Flask Documentation, "Flask: A Python Microframework," διαθέσιμο από: https://flask.palletsprojects.com

[11] Oracle, "MySQL: The World's Most Popular Open Source Database," διαθέσιμο από: https://www.mysql.com

[12] Chart.js, "Simple yet flexible JavaScript charting for designers & developers," διαθέσιμο από: https://www.chartjs.org

[13] DataTables, "Advanced tables, instantly," διαθέσιμο από: https://datatables.net

# ΠΑΡΑΡΤΗΜΑ Α

A1. Python Server

```
# Εισαγωγή των απαραίτητων βιβλιοθηκών
from flask import Flask, request, jsonify, render_template  # Flask framework για τον web server
import hashlib  # Για δημιουργία και επαλήθευση hash
import mysql.connector  # Για σύνδεση με MySQL
import json  # Για διαχείριση JSON δεδομένων

# Αρχικοποίηση του Flask app
app = Flask(__name__)

# Διαμόρφωση σύνδεσης με MySQL
db_config = {
    "host": "localhost",  # Διεύθυνση του server MySQL
    "user": "root",  # Χρήστης MySQL
    "password": "",  # Κωδικός MySQL (κενός εδώ)
    "database": "kivotia"  # Όνομα βάσης δεδομένων
}

# Μυστικό κλειδί για επαλήθευση hash
SECRET_KEY = "your_secret_key"

# Συνάρτηση για επαλήθευση hash
def verify_hash(data, received_hash):
    """Επαληθεύει το hash χρησιμοποιώντας SHA256 και το μυστικό κλειδί."""
    hash_object = hashlib.sha256((data + SECRET_KEY).encode())  # Δημιουργία hash
    calculated_hash = hash_object.hexdigest()  # Υπολογισμός του hash
    return calculated_hash == received_hash  # Σύγκριση του υπολογισμένου με το παραληφθέν

# Συνάρτηση για αποθήκευση δεδομένων στη βάση
def save_to_database(active, node_id, temperature, humidity, uv_intensity, timestamp_from_node):
    """Αποθηκεύει δεδομένα στον πίνακα `sensordata`."""
    try:
        # Δημιουργία σύνδεσης με τη βάση
        connection = mysql.connector.connect(**db_config)
        cursor = connection.cursor()
        # Ερώτημα εισαγωγής δεδομένων
        query = """
        INSERT INTO sensordata (active, nodeid, temperature, humidity, uv_intensity,
timestampfromnode)
        VALUES (%s, %s, %s, %s, %s, %s)
        """
        # Εκτέλεση του ερωτήματος
        cursor.execute(query, (active, node_id, temperature, humidity, uv_intensity,
timestamp_from_node))
```

```
        connection.commit()  # Αποθήκευση αλλαγών στη βάση
        cursor.close()  # Κλείσιμο cursor
        connection.close()  # Κλείσιμο σύνδεσης
    except mysql.connector.Error as err:
        # Σε περίπτωση σφάλματος, εκτύπωση του μηνύματος
        print(f"Error: {err}")
        return False
    return True  # Επιστροφή επιτυχίας

# Διαδρομή API για λήψη δεδομένων από ESP32
@app.route('/api/data', methods=['GET'])
def receive_data():
    try:
        # Ανάκτηση παραμέτρων από το query string
        node_id = request.args.get("id")
        temp = request.args.get("temp")
        humidity = request.args.get("humidity")
        uv = request.args.get("uv")
        timestamp = request.args.get("timestamp")
        received_hash = request.args.get("hash")

        # Έλεγχος για ελλιπείς παραμέτρους
        if not all([node_id, temp, humidity, uv, timestamp, received_hash]):
            return jsonify({"error": "Missing parameters"}), 400

        # Ανακατασκευή της αλφαριθμητικής ακολουθίας δεδομένων
        data = f"id={node_id}&temp={temp}&humidity={humidity}&uv={uv}&timestamp={timestamp}"

        # Επαλήθευση του hash
        if not verify_hash(data, received_hash):
            return jsonify({"error": "Hash verification failed"}), 403

        # Μετατροπή τύπων δεδομένων
        active = 1  # Ενεργή κατάσταση ως προεπιλογή
        node_id = int(node_id)  # Μετατροπή σε ακέραιο
        temp = float(temp)  # Μετατροπή σε δεκαδικό
        humidity = float(humidity)  # Μετατροπή σε δεκαδικό
        uv = float(uv)  # Μετατροπή σε δεκαδικό
        timestamp_from_node = timestamp  # Διατήρηση χρονικού στίγματος του ESP32

        # Αποθήκευση δεδομένων στη βάση
        if save_to_database(active, node_id, temp, humidity, uv, timestamp_from_node):
            return jsonify({"message": "Data saved successfully"}), 200
        else:
            return jsonify({"error": "Failed to save data"}), 500

    except Exception as e:
        # Διαχείριση εξαιρέσεων και επιστροφή του σφάλματος
```

```
        return jsonify({"error": str(e)}), 500

# Διαδρομή για την κύρια σελίδα
@app.route('/')
def index():
    """Εμφανίζει την κύρια σελίδα."""
    return render_template('index.html')

# Διαδρομή API για λήψη δεδομένων από τη βάση
@app.route('/api/get_data', methods=['GET'])
def get_data():
    """Ανακτά δεδομένα από τη βάση, με δυνατότητα φιλτραρίσματος ανά ημερομηνία."""
    try:
        # Ανάκτηση ημερομηνίας από το query string
        date = request.args.get('date')
        connection = mysql.connector.connect(**db_config)  # Σύνδεση με τη βάση
        cursor = connection.cursor(dictionary=True)  # Cursor για επιστροφή δεδομένων ως λεξικό
        if date:
            # Αν υπάρχει ημερομηνία, φιλτράρισμα των δεδομένων
            query = """
                SELECT nodeid, temperature, humidity, uv_intensity, timestampfromnode
                FROM sensordata
                WHERE DATE(timestampfromnode) = %s
                ORDER BY timestampfromnode DESC
            """
            cursor.execute(query, (date,))
        else:
            # Αν δεν υπάρχει ημερομηνία, επιστροφή όλων των δεδομένων
            query = """
                SELECT nodeid, temperature, humidity, uv_intensity, timestampfromnode
                FROM sensordata
                ORDER BY timestampfromnode DESC
            """
            cursor.execute(query)
        data = cursor.fetchall()  # Ανάκτηση όλων των εγγραφών
        cursor.close()  # Κλείσιμο cursor
        connection.close()  # Κλείσιμο σύνδεσης
        return jsonify(data)  # Επιστροφή δεδομένων σε μορφή JSON
    except mysql.connector.Error as err:
        # Σε περίπτωση σφάλματος, επιστροφή του μηνύματος σφάλματος
        return jsonify({"error": str(err)}), 500

# Διαδρομή API για λήψη μοναδικών ημερομηνιών
@app.route('/api/get_dates', methods=['GET'])
def get_dates():
    """Ανακτά μοναδικές ημερομηνίες από τη βάση δεδομένων."""
    try:
        connection = mysql.connector.connect(**db_config)  # Σύνδεση με τη βάση
```

```
        cursor = connection.cursor()
        # Ερώτημα για μοναδικές ημερομηνίες
        query = "SELECT DISTINCT DATE(timestampfromnode) AS date FROM sensordata ORDER BY date"
        cursor.execute(query)
        dates = [row[0].strftime('%Y-%m-%d') for row in cursor.fetchall()]  # Διαμόρφωση σε λίστα
        cursor.close()  # Κλείσιμο cursor
        connection.close()  # Κλείσιμο σύνδεσης
        return jsonify(dates)  # Επιστροφή ημερομηνιών σε μορφή JSON
    except mysql.connector.Error as err:
        # Σε περίπτωση σφάλματος, επιστροφή του μηνύματος σφάλματος
        return jsonify({"error": str(err)}), 500

# Εκκίνηση του Flask server
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

**Σελίδα προβολής των δεδομένων**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sensor Data</title>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <link rel="stylesheet" href="https://cdn.datatables.net/1.13.4/css/jquery.dataTables.min.css">
    <script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
    <script src="https://cdn.datatables.net/1.13.4/js/jquery.dataTables.min.js"></script>
    <style>
        #charts-container {
            width: 80%;
            margin: auto;
        }
        .chart-container {
            margin-bottom: 40px;
        }
        canvas {
            height: 450px !important;
        }
        #dataTable-container {
            width: 80%;
            margin: auto;
        }
        #date-filter {
```

```
            width: 80%;
            margin: auto;
            margin-bottom: 20px;
            text-align: center;
        }
    </style>
</head>
<body>
    <h1 style="width: 80%; margin: auto;">Sensor Data</h1>

    <div id="date-filter">
        <label for="date-select">Select Date:</label>
        <select id="date-select">
            <option value="">Loading dates...</option>
        </select>
        <button id="filterButton">Filter</button>
    </div>

    <div id="charts-container">
        <div class="chart-container">
            <h2>All Measurements</h2>
            <canvas id="allDataChart"></canvas>
        </div>
        <div class="chart-container">
            <h2>Temperature (°C)</h2>
            <canvas id="temperatureChart"></canvas>
        </div>
        <div class="chart-container">
            <h2>Humidity (%)</h2>
            <canvas id="humidityChart"></canvas>
        </div>
        <div class="chart-container">
            <h2>UV Intensity (V)</h2>
            <canvas id="uvChart"></canvas>
        </div>
    </div>

    <div id="dataTable-container">
        <h2>Data Table</h2>
        <table id="dataTable" class="display">
```

```
        <thead>
            <tr>
                <th>Node ID</th>
                <th>Temperature (°C)</th>
                <th>Humidity (%)</th>
                <th>UV Intensity (V)</th>
                <th>Timestamp</th>
            </tr>
        </thead>
        <tbody></tbody>
    </table>
</div>

<script>
    let allDataChart, temperatureChart, humidityChart, uvChart;

    // Fetch available dates from the server
    async function fetchAvailableDates() {
        const response = await fetch('/api/get_dates');
        return response.json();
    }

    // Fetch data from the server for a specific date
    async function fetchData(date) {
        const response = await fetch(`/api/get_data?date=${date}`);
        return response.json();
    }

    // Destroy a chart if it already exists
    function destroyChart(chart) {
        if (chart) {
            chart.destroy();
        }
    }

    // Populate the dropdown menu with available dates
    async function populateDateDropdown() {
        const dates = await fetchAvailableDates();
        const dateSelect = document.getElementById('date-select');
```

```
    // Clear existing options
    dateSelect.innerHTML = "";

    dates.forEach(date => {
        const option = document.createElement('option');
        option.value = date;
        option.textContent = date;
        dateSelect.appendChild(option);
    });

    // Automatically select the first date
    if (dates.length > 0) {
        dateSelect.value = dates[0];
        return dates[0]; // Return the first date
    }

    return null;
}

// Populate DataTable
async function populateTable(data) {
    const tableBody = $('#dataTable tbody');
    tableBody.empty();

    data.forEach(row => {
        tableBody.append(`
            <tr>
                <td>${row.nodeid}</td>
                <td>${row.temperature}</td>
                <td>${row.humidity}</td>
                <td>${row.uv_intensity}</td>
                <td>${row.timestampfromnode}</td>
            </tr>
        `);
    });

    $('#dataTable').DataTable();
}

// Populate Charts
```

```
        async function populateCharts(data) {
            const labels = data.map(row => {
                const timestamp = new Date(row.timestampfromnode);
                return timestamp.toLocaleTimeString([], { hour: '2-digit', minute: '2-digit' });
            });
            const temperatures = data.map(row => row.temperature);
            const humidities = data.map(row => row.humidity);
            const uvIntensities = data.map(row => row.uv_intensity);

            // Combined Chart
            destroyChart(allDataChart);
            const allCtx = document.getElementById('allDataChart').getContext('2d');
            allDataChart = new Chart(allCtx, {
                type: 'line',
                data: {
                    labels: labels,
                    datasets: [
                       { label: 'Temperature (°C)', data: temperatures, borderColor: 'red', borderWidth:
2 },
                         { label: 'Humidity (%)', data: humidities, borderColor: 'blue', borderWidth: 2
},
                              { label: 'UV Intensity (V)', data: uvIntensities, borderColor: 'green',
borderWidth: 2 }
                    ]
                },
                options: { responsive: true, maintainAspectRatio: false }
            });

            // Temperature Chart
            destroyChart(temperatureChart);
            const tempCtx = document.getElementById('temperatureChart').getContext('2d');
            temperatureChart = new Chart(tempCtx, {
                type: 'line',
                  data: { labels: labels, datasets: [{ label: 'Temperature (°C)', data: temperatures,
borderColor: 'red', borderWidth: 2 }] },
                options: { responsive: true, maintainAspectRatio: false }
            });

            // Humidity Chart
            destroyChart(humidityChart);
            const humCtx = document.getElementById('humidityChart').getContext('2d');
            humidityChart = new Chart(humCtx, {
```

```
                type: 'line',
               data: { labels: labels, datasets: [{ label: 'Humidity (%)', data: humidities, borderColor: 'blue', borderWidth: 2 }] },
                options: { responsive: true, maintainAspectRatio: false }
            });

            // UV Intensity Chart
            destroyChart(uvChart);
            const uvCtx = document.getElementById('uvChart').getContext('2d');
            uvChart = new Chart(uvCtx, {
                type: 'line',
                 data: { labels: labels, datasets: [{ label: 'UV Intensity (V)', data: uvIntensities, borderColor: 'green', borderWidth: 2 }] },
                options: { responsive: true, maintainAspectRatio: false }
            });
        }

        // Handle filter button click
        document.getElementById('filterButton').addEventListener('click', async () => {
            const date = document.getElementById('date-select').value;
            if (date) {
                const data = await fetchData(date);
                populateTable(data);
                populateCharts(data);
            }
        });

        // Initialize
        $(document).ready(async () => {
            const firstDate = await populateDateDropdown();
            if (firstDate) {
                const data = await fetchData(firstDate);
                populateTable(data);
                populateCharts(data);
            }
        });
    </script>
</body>
</html>
```