ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Τίτλος Πτυχιακής Εργασίας	Σύστημα Ποτίσματος IoT με συνδυασμό Μετρήσεων Εδάφους και Μετεωρολογικών Δεδομένων IoT Irrigation System Combining Soil Measurements and Meteorological Data
Ονοματεπώνυμο Φοιτητή	ΚΩΝΣΤΑΝΤΙΝΟΣ ΧΑΛΒΑΝΤΖΗΣ
Πατρώνυμο	ΠΑΝΑΓΙΩΤΗΣ
Αριθμός Μητρώου	Π21185
Επιβλέπων	Δέσποινα Πολέμη, Καθηγήτρια

Ημερομηνία Παράδοσης:Φεβρουάριος 2026

Copyright ©

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω θερμά τον καθηγητή **Δουληγέρη Χρήστο** για την πολύτιμη καθοδήγησή του καθ' όλη τη διάρκεια της εκπόνησης της εργασίας. Σε κάθε δυσκολία ή εμπόδιο που προέκυπτε, δεν περιοριζόταν στο να δώσει απλώς μια έτοιμη λύση· αντίθετα, με κατεύθυνε με τρόπο ουσιαστικό και παιδαγωγικό, θέτοντας τις σωστές ερωτήσεις και υποδεικνύοντας τα κατάλληλα βήματα, ώστε να μπορώ να αναλύω το πρόβλημα, να πειραματίζομαι και τελικά να καταλήγω μόνος μου στη λύση.

Θα ήθελα, επίσης, να ευχαριστήσω θερμά την καθηγήτρια **Δέσποινα Πολέμη** για την υποστήριξη και τη συμβολή της στην εκπόνηση της εργασίας. Οι παρατηρήσεις, οι συμβουλές και η επιστημονική της καθοδήγηση συνέβαλαν ουσιαστικά στη βελτίωση της δομής, του περιεχομένου και της συνολικής ποιότητας του τελικού αποτελέσματος.

Περίληψη

Η παρούσα πτυχιακή εργασία παρουσιάζει τη σχεδίαση και υλοποίηση ενός ολοκληρωμένου συστήματος ποτίσματος IoT, το οποίο συνδυάζει μετρήσεις εδάφους με μετεωρολογικά δεδομένα για τη λήψη απόφασης ποτίσματος και την απομακρυσμένη παρακολούθηση. Η λύση αποτελείται από δύο βασικούς κόμβους: (α) ένα edge device με ESP32-S3 που συλλέγει δεδομένα από αισθητήρα υγρασίας εδάφους, αισθητήρα ροής νερού και αισθητήρα στάθμης δοχείου (υπερηχητικό), ελέγχει αντλία/ρελέ και παρέχει τοπικό web dashboard καθώς και απεικόνιση σε LCD 20×4, και (β) έναν backend κόμβο σε Raspberry Pi Zero 2 W που λειτουργεί ως MQTT broker (Mosquitto), αποθηκεύει ιστορικά δεδομένα σε InfluxDB και τα οπτικοποιεί μέσω Grafana. Για τη μετεωρολογική πληροφορία χρησιμοποιείται το OpenWeatherMap Forecast API (ανά 3 ώρες), από το οποίο εξάγονται θερμοκρασία/υγρασία αέρα και εκτιμήσεις βροχόπτωσης (π.χ. 3h/12h/24h), ώστε να αποφεύγεται άσκοπο πότισμα όταν αναμένεται βροχή. Το σύστημα υποστηρίζει αυτόματη λειτουργία βάσει κανόνων (thresholds) και μηχανισμούς ασφαλείας (όρια χρόνου λειτουργίας, cooldown, emergency stop), ενώ παρέχει χειροκίνητο πότισμα μέσω web endpoints και καταγραφή των συμβάντων σε ιστορικό.
Σημειώνεται ότι σε παλαιότερη έκδοση του πρωτοτύπου είχε χρησιμοποιηθεί τοπικός αισθητήρας περιβάλλοντος, όμως στην τελική υλοποίηση τα δεδομένα θερμοκρασίας/υγρασίας αέρα λαμβάνονται από το OpenWeatherMap, απλοποιώντας το hardware χωρίς να επηρεάζεται ο βασικός στόχος του συστήματος.

Λέξεις Κλειδιά: *IoT, έξυπνη άρδευση, υγρασία εδάφους, μετεωρολογικά δεδομένα, OpenWeatherMap, MQTT, ESP32-S3, InfluxDB, αισθητήρες, Grafana, αυτοματισμός*

Abstract

This thesis presents the design and implementation of a complete IoT irrigation system that combines soil measurements with meteorological data to support watering decisions and remote monitoring. The solution follows a two-node architecture: (a) an ESP32-S3 edge device that reads a soil moisture sensor, a water flow sensor, and a water tank level sensor (ultrasonic), controls a pump/relay, and provides both a local web dashboard and a 20×4 LCD status display; and (b) a Raspberry Pi Zero 2 W backend that operates as an MQTT broker (Mosquitto), stores time-series data in InfluxDB, and visualizes historical measurements and events through Grafana dashboards.
Meteorological information is obtained from the OpenWeatherMap Forecast API (3-hour intervals), enabling the system to derive air temperature/humidity and rainfall forecasts (e.g., 3h/12h/24h) and to prevent unnecessary irrigation when rain is expected. The firmware implements rule-based control (thresholds) and safety mechanisms (maximum runtime, cooldown periods, emergency stop), while the web interface exposes endpoints for real-time monitoring and manual watering commands.
An earlier prototype included a local environmental sensor; however, the final version retrieves air temperature and humidity from the weather API, simplifying the hardware while preserving the system's core functionality.

Keywords: *IoT, smart irrigation, soil moisture, weather forecast, OpenWeatherMap, ESP32-S3, MQTT, sensors, InfluxDB, Grafana, automated watering*

Πίνακας Περιεχομένων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Εισαγωγή

Η ορθολογική διαχείριση του νερού αποτελεί κρίσιμο ζήτημα τόσο σε οικιακές καλλιέργειες όσο και σε μικρής κλίμακας συστήματα άρδευσης. Στην πράξη, το πότισμα γίνεται συχνά είτε “με το μάτι” είτε με σταθερό χρονοπρογραμματισμό, προσεγγίσεις που δεν λαμβάνουν επαρκώς υπόψη τις πραγματικές ανάγκες του φυτού και τις μεταβολές του περιβάλλοντος. Η υπερβολική άρδευση οδηγεί σε σπατάλη νερού, πιθανή ασφυξία του ριζικού συστήματος και αυξημένη κατανάλωση ενέργειας, ενώ η ανεπαρκής άρδευση προκαλεί στρες στα φυτά και μειώνει την ανάπτυξή τους. Παράλληλα, παράγοντες όπως η βροχόπτωση, η θερμοκρασία και η σχετική υγρασία αέρα επηρεάζουν άμεσα το ρυθμό εξάτμισης/διαπνοής και, κατ’ επέκταση, την απαιτούμενη ποσότητα νερού.

Οι τεχνολογίες Internet of Things (IoT) δίνουν τη δυνατότητα συλλογής μετρήσεων σε πραγματικό χρόνο, αυτοματισμού ενεργειών και απομακρυσμένης παρακολούθησης μέσω δικτύου. Με τη χρήση κατάλληλων αισθητήρων και μικροελεγκτών, είναι εφικτό να καταγράφεται η κατάσταση του εδάφους (π.χ. υγρασία), να ελέγχεται η παροχή νερού (αντλία/βαλβίδα) και να αποθηκεύονται ιστορικά δεδομένα για καλύτερη εποπτεία και τεκμηρίωση της λειτουργίας. Επιπλέον, η αξιοποίηση μετεωρολογικών δεδομένων και προβλέψεων μπορεί να βελτιώσει τη λήψη αποφάσεων, καθώς επιτρέπει την αναβολή ή μείωση του ποτίσματος όταν αναμένεται βροχή, περιορίζοντας άσκοπες ενεργοποιήσεις του συστήματος.

Σκοπός της παρούσας πτυχιακής εργασίας είναι η ανάπτυξη ενός συστήματος ποτίσματος IoT που συνδυάζει μετρήσεις εδάφους με μετεωρολογικά δεδομένα, με στόχο την αυτοματοποίηση της άρδευσης και την παροχή εργαλείων παρακολούθησης σε πραγματικό χρόνο και ιστορικά. Η λύση υλοποιείται με αρχιτεκτονική δύο κόμβων: ένα edge device βασισμένο σε ESP32-S3 το οποίο συλλέγει δεδομένα από αισθητήρα υγρασίας εδάφους, αισθητήρα ροής νερού και αισθητήρα στάθμης δοχείου, ελέγχει την αντλία μέσω ρελέ και παρέχει τοπική απεικόνιση κατάστασης σε LCD καθώς και web dashboard· και έναν backend κόμβο σε Raspberry Pi, όπου λειτουργεί MQTT broker, αποθηκεύονται τα δεδομένα σε βάση χρονοσειρών και προβάλλονται σε dashboards. Για τη μετεωρολογική πληροφορία αξιοποιείται το OpenWeatherMap Forecast API, από το οποίο αντλούνται παράμετροι όπως η προβλεπόμενη βροχόπτωση και περιβαλλοντικές τιμές, οι οποίες ενσωματώνονται σε απλούς κανόνες απόφασης ποτίσματος.

Η εργασία επικεντρώνεται στη σχεδίαση της αρχιτεκτονικής, στην υλοποίηση του υλικού και του λογισμικού του edge device, στην ενσωμάτωση της επικοινωνίας και αποθήκευσης δεδομένων, καθώς και στην παρουσίαση της λειτουργίας του πρωτοτύπου μέσα από ενδεικτικά σενάρια χρήσης. Αξίζει να σημειωθεί ότι σε προηγούμενη εκδοχή του πρωτοτύπου είχε χρησιμοποιηθεί τοπικός αισθητήρας θερμοκρασίας/υγρασίας αέρα, όμως στην τελική υλοποίηση τα δεδομένα αυτά λαμβάνονται από το OpenWeatherMap, απλοποιώντας το υλικό χωρίς να επηρεάζεται η βασική λειτουργία του συστήματος.

Η δομή της εργασίας είναι η εξής: στο Κεφάλαιο 1 παρουσιάζεται η περιγραφή του συστήματος και οι βασικές προδιαγραφές/απαιτήσεις. Στο Κεφάλαιο 2 αναλύεται η αρχιτεκτονική, η ροή δεδομένων και η λογική απόφασης ποτίσματος. Στο Κεφάλαιο 3 περιγράφεται η υλοποίηση του edge device (ESP32-S3), συμπεριλαμβανομένων των αισθητήρων, του ελέγχου ποτίσματος και της διεπαφής. Στο Κεφάλαιο 4 παρουσιάζεται η υποδομή backend (MQTT, αποθήκευση, οπτικοποίηση). Στο Κεφάλαιο 5 περιγράφεται η λειτουργία του πρωτοτύπου και παρατίθενται ενδεικτικά σενάρια χρήσης και παραδείγματα ιστορικών δεδομένων. Τέλος, στα Συμπεράσματα συνοψίζονται τα βασικά σημεία και προτείνονται πιθανές μελλοντικές επεκτάσεις.

1 State of the Art

Στη βιβλιογραφία παρατηρούνται πολλές υλοποιήσεις “smart irrigation” που ξεκινούν από την κλασική λογική **threshold-based**: συλλογή μετρήσεων από αισθητήρες (κυρίως υγρασίας εδάφους) και ενεργοποίηση αντλίας/βαλβίδας μέσω ρελέ όταν η τιμή αποκλίνει από προκαθορισμένα όρια. Ένα χαρακτηριστικό παράδειγμα είναι η εργασία των Darshna et al. (2015), όπου χρησιμοποιείται μικροελεγκτής ATmega328, αισθητήρας υγρασίας εδάφους και RTD αισθητήρας θερμοκρασίας, με ON/OFF έλεγχο αντλίας μέσω ρελέ [28]. Η προσέγγιση αυτή αποτυπώνει καθαρά το βασικό μοτίβο “sensor → threshold → actuator”, το οποίο παραμένει ιδιαίτερα δημοφιλές λόγω απλότητας, χαμηλού κόστους και εύκολης υλοποίησης. Ωστόσο, αναδεικνύεται και ο βασικός περιορισμός: η απόφαση βασίζεται αποκλειστικά σε τοπικές μετρήσεις, χωρίς αξιοποίηση εξωτερικής πληροφορίας (π.χ. επικείμενη βροχόπτωση), κάτι που οι ίδιοι οι συγγραφείς αναφέρουν ως πιθανή μελλοντική επέκταση [28].

Ένα από τα πιο κρίσιμα σημεία στις σύγχρονες υλοποιήσεις είναι η **αξιοπιστία των αισθητήρων υγρασίας**, ειδικά όταν χρησιμοποιούνται low-cost capacitive sensors. Η βιβλιογραφία τονίζει ότι οι ακατέργαστες τιμές δεν είναι “καθολικές”, καθώς επηρεάζονται από το υπόστρωμα, την αλατότητα, τη θέση/βάθος εγκατάστασης και πιθανό drift με τον χρόνο. Για τον λόγο αυτό, δίνεται έμφαση στη **βαθμονόμηση (calibration)** και στην προσαρμογή thresholds ανά εγκατάσταση. Πρόσφατες εργασίες παρουσιάζουν μεθοδολογίες βαθμονόμησης για low-cost capacitive αισθητήρες και τεκμηριώνουν ότι η σωστή χαρτογράφηση “dry/wet” σημείων αναφοράς βελτιώνει σημαντικά τη χρησιμότητα του αισθητήρα σε εφαρμογές άρδευσης [26]. Αντίστοιχα, προσεγγίσεις με Arduino-based capacitive sensors σε πλαίσιο smart agriculture δείχνουν ότι η διαδικασία calibration είναι αναγκαία προϋπόθεση ώστε το threshold να ανταποκρίνεται σε πραγματικές συνθήκες και όχι σε “τυπικές” τιμές εργαστηρίου [27]. Πέρα από το calibration, είναι συχνή η υιοθέτηση πρακτικών σταθεροποίησης των μετρήσεων, όπως φιλτράρισμα (moving average/median), απόρριψη ακραίων τιμών (outliers) και χρονικά “παράθυρα” μετά από πότισμα, ώστε να αποφεύγονται ψευδείς ενεργοποιήσεις λόγω μεταβατικών φαινομένων (π.χ. στιγμιαία διαβροχή γύρω από τον αισθητήρα).

Στο επίπεδο λογικής ελέγχου, πολλές υλοποιήσεις εξελίσσονται πέρα από το απλό ON/OFF threshold. Ένα τυπικό βήμα ωρίμανσης είναι η χρήση **band control / hysteresis**, δηλαδή καθορισμός δύο ορίων (ένα για ενεργοποίηση και ένα για απενεργοποίηση), ώστε να μειώνεται το “oscillation” και να αποφεύγονται συνεχόμενα μικρά ποτίσματα από θορυβώδη μέτρηση. Η έννοια της υστέρησης συνδέεται και με τη δυναμική του νερού στο έδαφος κατά την άρδευση, όπου η απόκριση της υγρασίας δεν είναι στιγμιαία και επηρεάζεται από τη διάταξη/στρωματοποίηση του εδάφους και τον τρόπο εφαρμογής νερού [25]. Έτσι, πρακτικές όπως cooldown periods, μέγιστος χρόνος λειτουργίας αντλίας (max runtime) και έλεγχοι ασφαλείας θεωρούνται απαραίτητα στοιχεία για να μετατραπεί ένα prototype σε σύστημα με ανθεκτικότητα σε αστοχίες αισθητήρων ή ενεργοποιητών.

Ιδιαίτερα εμφανής στη σύγχρονη βιβλιογραφία είναι η στροφή προς **υβριδικό έλεγχο (soil + weather)**, όπου η πρόγνωση καιρού λειτουργεί ως επιπλέον “φίλτρο” στη λήψη απόφασης. Η πρακτική λογική είναι σαφής: αν αναμένεται βροχόπτωση στο άμεσο χρονικό παράθυρο, το σύστημα μπορεί να καθυστερήσει ή να αποφύγει ένα πότισμα, μειώνοντας τη σπατάλη νερού. Η πρόσβαση σε πρόγνωση υλοποιείται συχνά μέσω δημόσιων weather APIs, όπως η υπηρεσία “5 Day / 3 Hour Forecast” της OpenWeather [19] (και εφόσον χρησιμοποιηθεί, το One Call API 3.0 [20]). Ενδεικτικά, η εργασία *Weather Integrated SMART Irrigation System* (2025, IEEE) περιγράφει διάκριση λειτουργιών όπως sensor-only (π.χ. indoor) και weather-assisted (π.χ. outdoor), αξιοποιώντας πρόγνωση βροχής μέσω OpenWeatherMap ώστε να αναβάλλεται το πότισμα όταν προβλέπεται βροχόπτωση [31]. Αντίστοιχα, οι Asokan et al. (2025) ενσωματώνουν weather forecast στο decision loop, εδραιώνοντας το μοντέλο “soil measurement + forecast constraint” ως σύγχρονη πρακτική κατεύθυνση [29]. Επιπλέον, οι Tiwari et al. (2025) πηγαίνουν ένα βήμα παραπέρα, προτείνοντας AI-driven decision making που συνδυάζει αισθητήρες και forecast δεδομένα για πιο δυναμική και προσαρμοστική απόφαση άρδευσης [30]. Ένα κοινό τεχνικό συμπέρασμα σε τέτοιες προσεγγίσεις είναι η

ανάγκη για **fallback συμπεριφορά**: όταν το API ή το δίκτυο δεν είναι διαθέσιμο, το σύστημα πρέπει να παραμένει λειτουργικό επιστρέφοντας σε sensor-only λογική, ώστε να μην εισάγεται νέο σημείο αστοχίας [31].

Παράλληλα με τη λογική ελέγχου, η βιβλιογραφία δείχνει ότι πολλά σύγχρονα συστήματα “smart irrigation” ωριμάζουν αρχιτεκτονικά προς ολοκληρωμένες **IoT λύσεις** με τηλεμετρία, ιστορικότητα και οπτικοποίηση. Σε επίπεδο edge συσκευών, πλατφόρμες όπως ESP32/ESP32-S3 προσφέρουν Wi-Fi δυνατότητες και υποστήριξη για IoT εφαρμογές [1]–[4], ενώ πρακτικά components όπως web interfaces πάνω στον μικροελεγκτή υλοποιούνται με ελαφριές βιβλιοθήκες (π.χ. asynchronous web server) για τοπική παρακολούθηση και έλεγχο [11]. Για ανταλλαγή δεδομένων, το MQTT αποτελεί μία από τις πιο διαδεδομένες επιλογές λόγω publish/subscribe μοντέλου και χαμηλού overhead, με τυποποίηση στο MQTT 3.1.1 [5]. Στην πράξη, brokers όπως ο Mosquitto χρησιμοποιούνται ευρέως ως κεντρικό σημείο δρομολόγησης μηνυμάτων, ενώ η τεκμηρίωση/εργαλεία του οικοσυστήματος διευκολύνουν την ανάπτυξη και τη διάγνωση [6]–[8]. Στο επίπεδο client, βιβλιοθήκες όπως PubSubClient είναι επίσης συχνές σε microcontroller περιβάλλοντα για σύνδεση σε MQTT brokers [9], [10].

Για την αποθήκευση ιστορικών δεδομένων και την ανάλυση χρονοσειρών, βάσεις όπως η InfluxDB επιτρέπουν αποτελεσματική καταγραφή telemetry (π.χ. moisture, temperature, events) και ερωτήματα/aggregations μέσω InfluxQL και line protocol [14]–[16]. Στη συνέχεια, η Grafana χρησιμοποιείται ως επίπεδο οπτικοποίησης με dashboards, επιτρέποντας διαγράμματα, thresholds και συνοπτικούς δείκτες (stats) που κάνουν τη λειτουργία του συστήματος “παρατηρήσιμη” (observable) [17], [18]. Αυτή η μετάβαση από “απλό αυτοματισμό” σε “monitorable σύστημα” είναι κρίσιμη, επειδή επιτρέπει αξιολόγηση με αντικειμενικές μετρικές: πόσες φορές ενεργοποιήθηκε η αντλία, πώς ανταποκρίθηκε η υγρασία μετά από κάθε κύκλο, αν υπάρχουν ανωμαλίες (π.χ. πότισμα χωρίς αύξηση moisture), και κατά πόσο οι αποφάσεις ευθυγραμμίζονται με εξωτερικές συνθήκες (π.χ. forecast βροχής).

Ένα ακόμη στοιχείο που εμφανίζεται σε πιο πρακτικές/εφαρμοσμένες υλοποιήσεις είναι η **ποσοτικοποίηση της κατανάλωσης νερού**. Σε αντίθεση με απλές λύσεις που καταγράφουν μόνο την ενεργοποίηση αντλίας, η ενσωμάτωση flow sensing επιτρέπει μέτρηση mL ανά κύκλο και συνολική κατανάλωση, δίνοντας τη δυνατότητα πραγματικής σύγκρισης ρυθμίσεων (dose, thresholds, cooldowns) και αποδοτικότητας. Χαρακτηριστικός τύπος αισθητήρα ροής σε low-cost prototypes είναι ο YF-S201 (Hall effect flow sensor), όπου η τεκμηρίωση/datasheet χρησιμοποιείται για αντιστοίχιση παλμών σε παροχή/όγκο και για ρυθμίσεις στον υπολογισμό [22]. Παράλληλα, πρακτικές όπως monitoring στάθμης δοχείου/δεξαμενής (για αποφυγή dry-run της αντλίας) ενισχύουν την αξιοπιστία, καθώς προσθέτουν “safety constraints” στο control loop.

Συνοψίζοντας, η εξέλιξη που αποτυπώνεται στη βιβλιογραφία κινείται από απλό threshold-based control [28] προς υβριδικές προσεγγίσεις που ενσωματώνουν forecast δεδομένα [19], [29]–[31] και, σε πιο σύνθετες περιπτώσεις, AI-driven decision making [30]. Παράλληλα, η ωρίμανση των υλοποιήσεων συνδέεται με την υιοθέτηση πρακτικών IoT τεχνολογιών: MQTT επικοινωνία [5]–[10], backend συλλογή/αποθήκευση σε time-series DB [14]–[16] και οπτικοποίηση σε dashboards [17], [18], ώστε η λειτουργία να είναι τεκμηριωμένη και αξιολογήσιμη. Η παρούσα εργασία ευθυγραμμίζεται με αυτή τη σύγχρονη κατεύθυνση, αξιοποιώντας πρόγνωση βροχής ως φίλτρο απόφασης [19] και δίνοντας έμφαση στη σωστή μεταχείριση low-cost αισθητήρων μέσω calibration [26], [27], καθώς και στη μετρήσιμη τεκμηρίωση λειτουργίας μέσω monitoring/ιστορικότητας και καταγραφής κατανάλωσης.

2 Περιγραφή Συστήματος και Προδιαγραφές

2.1 Συνολική εικόνα λύσης (Edge + Backend)

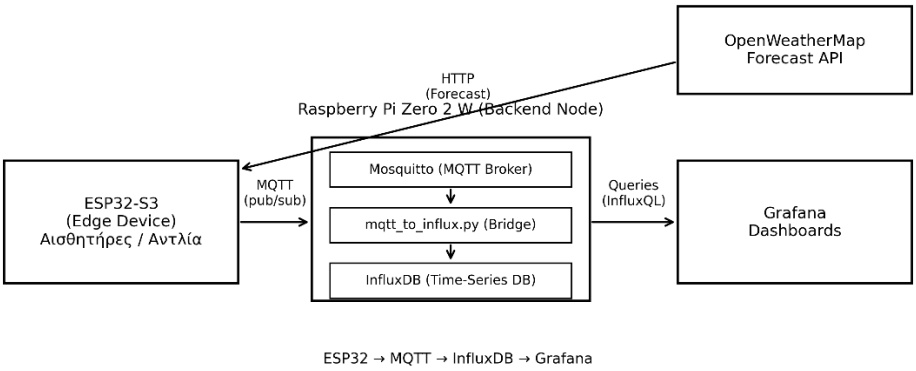
Το σύστημα ποτίσματος που υλοποιήθηκε βασίζεται σε αρχιτεκτονική δύο επιπέδων, με στόχο τον διαχωρισμό της συλλογής/ελέγχου (edge) από την αποθήκευση και την οπτικοποίηση (backend). Ο πρώτος κόμβος είναι ένα edge device βασισμένο σε ESP32-S3, το οποίο αναλαμβάνει τη λήψη μετρήσεων από αισθητήρες, τη λήψη αποφάσεων ποτίσματος και την ενεργοποίηση της αντλίας μέσω ρελέ. Ο δεύτερος κόμβος είναι ένας backend server σε Raspberry Pi Zero 2 W, ο οποίος φιλοξενεί τον MQTT broker, αποθηκεύει τα δεδομένα σε βάση χρονοσειρών και παρέχει οπτικοποίηση ιστορικών μετρήσεων μέσω dashboard.

Στο επίπεδο του edge device, το ESP32-S3 συλλέγει μετρήσεις από:

- **Αισθητήρα υγρασίας εδάφους**, ώστε να αποτυπώνεται η κατάσταση του υποστρώματος (dry/wet).
- **Αισθητήρα ροής νερού (flow meter)**, ώστε να καταγράφεται η κατανάλωση νερού κατά τη διάρκεια ποτίσματος.
- **Αισθητήρα στάθμης δοχείου (υπερηχητικό)**, ώστε να υπάρχει ένδειξη διαθέσιμου νερού και να αποφεύγεται πότισμα όταν το δοχείο είναι χαμηλό. Παράλληλα, υπάρχει **LCD 20×4 (I2C)** για τοπική απεικόνιση βασικών ενδείξεων (π.χ. υγρασία εδάφους, κατάσταση ποτίσματος, στάθμη δοχείου, συνοπτικό status).

Για την ενσωμάτωση της μετεωρολογικής πληροφορίας χρησιμοποιείται το **OpenWeatherMap Forecast API**, από το οποίο λαμβάνονται προβλέψεις (ανά 3 ώρες) και εξάγονται χρήσιμες παράμετροι όπως αναμενόμενη βροχόπτωση σε χρονικά παράθυρα (π.χ. 3h/12h/24h) καθώς και περιβαλλοντικές τιμές που αξιοποιούνται σε επίπεδο ενημέρωσης/παρακολούθησης. Με αυτόν τον τρόπο, το σύστημα μπορεί να αναβάλει ή να περιορίσει το πότισμα όταν αναμένεται βροχή, μειώνοντας τη σπατάλη νερού.

Η επικοινωνία μεταξύ edge device και backend γίνεται μέσω **MQTT**, όπου το ESP32 δημοσιεύει περιοδικά τις μετρήσεις του και (όπου χρειάζεται) κατάσταση/συμβάντα (events). Στον backend κόμβο, ένας μηχανισμός “γεφύρωσης” (bridge) καταγράφει τα εισερχόμενα MQTT μηνύματα σε **InfluxDB**, ώστε να διατηρείται ιστορικό για ανάλυση και παρακολούθηση. Τέλος, μέσω **Grafana** εμφανίζονται γραφήματα και πίνακες με τις μετρήσεις και τα συμβάντα (π.χ. πότε έγινε πότισμα, πόσο νερό καταναλώθηκε, πώς κινήθηκε η υγρασία εδάφους κ.λπ.).



Εικόνα 2-1: Διάγραμμα αρχιτεκτονικής δύο κόμβων (ESP32-S3 edge → MQTT → InfluxDB → Grafana) και παράλληλη λήψη μετεωρολογικών δεδομένων από OpenWeatherMap.

2.2 Σενάρια χρήσης (Use Cases)

Η λειτουργία του συστήματος περιγράφεται καλύτερα μέσα από βασικά σενάρια χρήσης, τα οποία αντιστοιχούν σε πραγματικές ανάγκες κατά τη φροντίδα φυτών.

1. **UC-1: Αυτόματο πότισμα βάσει υγρασίας εδάφους**
    1. Το ESP32-S3 λαμβάνει περιοδικά μέτρηση υγρασίας εδάφους.
    2. Αν η τιμή βρίσκεται κάτω από προκαθορισμένο όριο “ξηρού εδάφους”, το σύστημα εξετάζει αν επιτρέπεται πότισμα (π.χ. δεν βρίσκεται σε cooldown).
    3. Αν δεν υπάρχει περιορισμός, ενεργοποιείται το ρελέ και ξεκινά η αντλία.
    4. Κατά το πότισμα καταγράφεται ροή/κατανάλωση και ενημερώνεται η κατάσταση (status).
    5. Το πότισμα τερματίζεται είτε όταν συμπληρωθεί προκαθορισμένη διάρκεια/δόση είτε όταν επιτευχθεί ασφαλές όριο, και το σύστημα επιστρέφει σε κατάσταση αναμονής.
2. **UC-2: Αναβολή ποτίσματος λόγω πρόβλεψης βροχής**
    1. Το σύστημα ενημερώνεται από το OpenWeatherMap για την πιθανότητα/ποσότητα βροχόπτωσης στο προσεχές διάστημα.
    2. Αν η πρόβλεψη δείχνει αναμενόμενη βροχή πάνω από ένα όριο (π.χ. σε 12h/24h), το σύστημα περιορίζει ή αναβάλλει το αυτόματο πότισμα.
    3. Καταγράφεται στο ιστορικό ότι “αποφεύχθηκε πότισμα λόγω πρόβλεψης βροχής” (ως event) για λόγους διαφάνειας/ιχνηλασιμότητας.
3. **UC-3: Χειροκίνητη ενεργοποίηση ποτίσματος (Manual Watering)**
    1. Ο χρήστης δίνει εντολή ποτίσματος από το web dashboard του ESP32 (ή από αντίστοιχο endpoint).
    2. Το σύστημα ελέγχει βασικές προϋποθέσεις ασφαλείας (π.χ. στάθμη δοχείου, emergency stop).
    3. Ενεργοποιεί την αντλία για προκαθορισμένη διάρκεια/δόση και καταγράφει την κατανάλωση νερού.
4. **UC-4: Ασφαλής λειτουργία σε σφάλμα (Fail-safe)**
    1. Αν εντοπιστεί μη έγκυρη μέτρηση αισθητήρα (π.χ. out-of-range) ή απώλεια κρίσιμης πληροφορίας, το σύστημα περνά σε “ασφαλή” συμπεριφορά.
    2. Εφαρμόζεται προστασία όπως: τερματισμός ποτίσματος, κλείδωμα αυτόματης ενεργοποίησης, ή απαίτηση χειροκίνητης επιβεβαίωσης.
    3. Εμφανίζεται σχετική ένδειξη σε LCD/dashboard και καταγράφεται event για διάγνωση.

2.3 Απαιτήσεις

Οι απαιτήσεις του συστήματος προκύπτουν από τα σενάρια χρήσης που περιγράφηκαν και αποτυπώνουν τόσο τις λειτουργίες που πρέπει να υποστηρίζει (λειτουργικές απαιτήσεις) όσο και τα ποιοτικά χαρακτηριστικά που πρέπει να ικανοποιεί (μη λειτουργικές απαιτήσεις). Στόχος είναι η υλοποίηση ενός πρακτικού πρωτοτύπου που να λειτουργεί με συνέπεια, να είναι επεκτάσιμο και να μπορεί να χρησιμοποιηθεί εύκολα σε πραγματικές συνθήκες.

2.3.1 Λειτουργικές απαιτήσεις

FR-1: Μέτρηση υγρασίας εδάφους
Το σύστημα πρέπει να μετρά την υγρασία του εδάφους σε τακτά χρονικά διαστήματα και να χρησιμοποιεί τη μέτρηση αυτή ως κύριο κριτήριο για την ενεργοποίηση ποτίσματος.

FR-2: Έλεγχος ποτίσματος μέσω ενεργοποιητή
Το σύστημα πρέπει να μπορεί να ενεργοποιεί/απενεργοποιεί αντλία (ή αντίστοιχο μηχανισμό παροχής νερού) μέσω ρελέ, με σαφή κατάσταση λειτουργίας (IDLE/WATERING/ERROR).

FR-3: Καταγραφή κατανάλωσης νερού
Κατά τη διάρκεια ποτίσματος πρέπει να καταγράφεται ροή/κατανάλωση νερού μέσω αισθητήρα ροής, ώστε να υπάρχει μέτρηση “πόσο ποτίστηκε” και όχι μόνο “πότε ποτίστηκε”.

FR-4: Εκτίμηση στάθμης δοχείου
Το σύστημα πρέπει να υπολογίζει τη στάθμη διαθέσιμου νερού στο δοχείο μέσω υπερηχητικού αισθητήρα και να μπορεί να αποτρέψει πότισμα σε περίπτωση χαμηλής στάθμης.

FR-5: Λήψη μετεωρολογικών δεδομένων και εξαγωγή δεικτών βροχής
Το σύστημα πρέπει να λαμβάνει προβλέψεις καιρού από OpenWeatherMap (Forecast API) και να υπολογίζει δείκτες/παράγωγα μεγέθη (π.χ. αναμενόμενη βροχή σε 12h/24h ή boolean “will_rain”), ώστε να επηρεάζεται η απόφαση ποτίσματος.

FR-6: Αυτόματη λειτουργία βάσει κανόνων
Το σύστημα πρέπει να εφαρμόζει κανόνες απόφασης που συνδυάζουν την υγρασία εδάφους με τη μετεωρολογική πληροφορία, ώστε να ενεργοποιεί πότισμα μόνο όταν χρειάζεται και να αποφεύγει άσκοπες ενεργοποιήσεις.

FR-7: Χειροκίνητος έλεγχος
Το σύστημα πρέπει να παρέχει χειροκίνητη ενεργοποίηση/τερματισμό ποτίσματος μέσω web interface/endpoints, ανεξάρτητα από την αυτόματη λειτουργία (manual override).

FR-8: Τοπική απεικόνιση κατάστασης
Το σύστημα πρέπει να εμφανίζει βασικές ενδείξεις σε LCD (π.χ. υγρασία, στάθμη, κατάσταση ποτίσματος, συνοπτικό status/error).

FR-9: Απομακρυσμένη παρακολούθηση και ιστορικό
Οι μετρήσεις και τα συμβάντα (events) πρέπει να δημοσιεύονται μέσω MQTT και να αποθηκεύονται σε βάση χρονοσειρών (InfluxDB), ώστε να υπάρχει ιστορικό για οπτικοποίηση σε Grafana.

FR-10: Καταγραφή συμβάντων και σφαλμάτων
Το σύστημα πρέπει να καταγράφει σημαντικά συμβάντα, όπως έναρξη/λήξη ποτίσματος, αποφυγή ποτίσματος λόγω πρόβλεψης βροχής, χαμηλή στάθμη δοχείου και σφάλματα αισθητήρων ή επικοινωνίας.

2.3.2 Μη λειτουργικές απαιτήσεις (αξιοπιστία, ασφάλεια, συντηρησιμότητα)

NFR-1: Αξιοπιστία και σταθερότητα λειτουργίας
Το σύστημα πρέπει να λειτουργεί με σταθερό τρόπο, διατηρώντας τη σωστή κατάσταση (state) και να αποφεύγει ανεπιθύμητες συνεχείς ενεργοποιήσεις (on/off).

NFR-2: Ασφαλής λειτουργία (safety)
Πρέπει να υπάρχουν μηχανισμοί προστασίας όπως μέγιστη διάρκεια ποτίσματος, χρονικό διάστημα “cooldown” μεταξύ ποτισμάτων και emergency stop, ώστε να μειώνεται ο κίνδυνος υπερ-ποτίσματος ή βλάβης εξοπλισμού.

NFR-3: Ανθεκτικότητα σε αποτυχίες επικοινωνίας
Σε περίπτωση απώλειας σύνδεσης (Wi-Fi/MQTT/API), το σύστημα πρέπει να συνεχίζει να λειτουργεί με ασφαλή τρόπο (π.χ. περιορισμός αυτόματου ποτίσματος ή fallback σε συντηρητικούς κανόνες) και να επανέρχεται όταν αποκατασταθεί η σύνδεση.

NFR-4: Επεκτασιμότητα
Η αρχιτεκτονική πρέπει να επιτρέπει μελλοντικές επεκτάσεις (π.χ. περισσότερες ζώνες ποτίσματος, επιπλέον αισθητήρες), χωρίς να απαιτείται πλήρης ανασχεδιασμός.

NFR-5: Συντηρησιμότητα και καθαρή δομή λογισμικού
Ο κώδικας του συστήματος πρέπει να είναι οργανωμένος σε λειτουργικές ενότητες (modules), ώστε να διευκολύνεται η συντήρηση και η προσθήκη νέων λειτουργιών.

NFR-6: Χαμηλό κόστος και πρακτική υλοποίηση
Η λύση πρέπει να χρησιμοποιεί διαθέσιμο και οικονομικό εξοπλισμό (microcontroller, αισθητήρες, Raspberry Pi), ώστε να είναι εφαρμόσιμη και σε οικιακό περιβάλλον.

NFR-7: Βασική προστασία ευαίσθητων πληροφοριών
Κρίσιμες παράμετροι (π.χ. Wi-Fi credentials, API key) πρέπει να αποθηκεύονται/διαχειρίζονται με τρόπο που αποφεύγει άσκοπη έκθεση (π.χ. σε ξεχωριστό αρχείο ρυθμίσεων), ιδιαίτερα όταν ο κώδικας διαμοιράζεται.

3 Αρχιτεκτονική και Σχεδιασμός

3.1 Τοπολογία δικτύου και ρόλοι κόμβων

Η προτεινόμενη λύση υλοποιείται ως ένα κατανεμημένο σύστημα δύο κόμβων, όπου ο έλεγχος ποτίσματος πραγματοποιείται στο edge, ενώ η αποθήκευση και η οπτικοποίηση γίνονται στο backend. Η επικοινωνία βασίζεται σε τοπικό δίκτυο (Wi-Fi) και σε πρωτόκολλο δημοσίευσης/εγγραφής (publish/subscribe) μέσω MQTT, το οποίο είναι κατάλληλο για IoT εφαρμογές λόγω μικρού overhead και απλής διαχείρισης ασύγχρονης ανταλλαγής μηνυμάτων.

Edge κόμβος (ESP32-S3)
Ο ρόλος του ESP32-S3 είναι “real-time” και περιλαμβάνει:

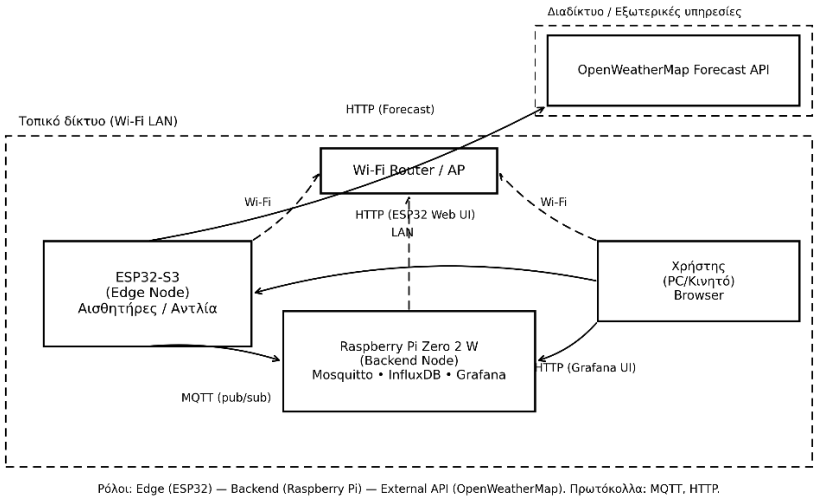
- Συλλογή μετρήσεων από αισθητήρες (υγρασία εδάφους, ροή νερού, στάθμη δοχείου).
- Εκτέλεση της λογικής απόφασης ποτίσματος (κανόνες, ασφαλιστικές δικλείδες).
- Ενεργοποίηση/απενεργοποίηση αντλίας μέσω ρελέ.
- Έκθεση τοπικής διεπαφής χρήστη (web dashboard) για παρακολούθηση και χειροκίνητες εντολές.
- Τοπική ένδειξη κατάστασης μέσω LCD 20×4.

Backend κόμβος (Raspberry Pi Zero 2 W)
Ο ρόλος του Raspberry Pi είναι να λειτουργεί ως “hub” υπηρεσιών:

- Φιλοξενία MQTT broker (Mosquitto) για λήψη των δεδομένων από το ESP32.
- Αποθήκευση μετρήσεων/συμβάντων σε InfluxDB (time-series database).
- Οπτικοποίηση ιστορικού και κατάστασης μέσω Grafana dashboards.

Εξωτερική πηγή δεδομένων (OpenWeatherMap)
Το σύστημα αξιοποιεί μετεωρολογικά δεδομένα μέσω OpenWeatherMap Forecast API. Η λήψη της πρόβλεψης ενσωματώνεται στη ροή λήψης αποφάσεων, επιτρέποντας την “ενημερωμένη” αναβολή ποτίσματος όταν αναμένεται βροχή. Σημειώνεται ότι σε παλαιότερη έκδοση είχε εξεταστεί η χρήση τοπικού αισθητήρα περιβάλλοντος, όμως στην τελική υλοποίηση τα αντίστοιχα δεδομένα προέρχονται από το API, μειώνοντας την πολυπλοκότητα του hardware.

Συνολικά, η τοπολογία μπορεί να ιδωθεί ως: **ESP32-S3 (Wi-Fi) → Mosquitto (MQTT) → InfluxDB → Grafana**, ενώ παράλληλα το ESP32 λαμβάνει **Forecast δεδομένα** από το OpenWeatherMap και εκθέτει **local web UI** για χειρισμό και άμεση παρακολούθηση.



Εικόνα 3-1: Διάγραμμα τοπολογίας δικτύου και ρόλων

3.2 Ροή δεδομένων (Data Flow) από αισθητήρες έως οπτικοποίηση

Η ροή δεδομένων του συστήματος σχεδιάστηκε ώστε να καλύπτει τρεις βασικές ανάγκες: (α) λήψη μετρήσεων, (β) λήψη απόφασης και ενέργεια ποτίσματος, (γ) αποθήκευση και οπτικοποίηση ιστορικών δεδομένων.

3.2.1 Ροή μετρήσεων (telemetry)

- Το ESP32 εκτελεί περιοδικά ανάγνωση των αισθητήρων (υγρασία εδάφους, στάθμη δοχείου, ροή νερού).
- Οι τιμές ελέγχονται για εγκυρότητα (π.χ. out-of-range), ενδεχομένως εξομαλύνονται/φιλτράρονται, και συσχετίζονται με χρονική σήμανση.
- Οι μετρήσεις δημοσιεύονται σε MQTT topics ως telemetry (π.χ. “soil”, “tank”, “flow”, “system status”), ώστε να είναι διαθέσιμες τόσο για αποθήκευση όσο και για πραγματικού χρόνου παρακολούθηση.

3.2.2 Ροή μετεωρολογικών δεδομένων (weather feed)

- Το ESP32 (ή η υποδομή του συστήματος, ανά σχεδίαση) ζητά περιοδικά πρόβλεψη καιρού από το OpenWeatherMap Forecast API.

- Από τα επιστρεφόμενα δεδομένα εξάγονται χρήσιμες παράμετροι για τον έλεγχο ποτίσματος, όπως αναμενόμενη βροχή στο προσεχές παράθυρο (π.χ. 12h/24h) και συνοπτικές περιβαλλοντικές τιμές.

- Τα weather-derived δεδομένα μπορούν είτε να δημοσιεύονται ως ξεχωριστό telemetry (ώστε να φαίνονται και στο ιστορικό) είτε να χρησιμοποιούνται άμεσα στη λήψη απόφασης.

3.2.3 Ροή εντολών και ελέγχου (control loop)

- Το σύστημα αξιολογεί κανόνες ποτίσματος:

    - κύριο κριτήριο: υγρασία εδάφους σε σχέση με thresholds,
    - περιορισμοί: στάθμη δοχείου, cooldown, μέγιστος χρόνος ποτίσματος,
    - τροποποίηση απόφασης: αναμενόμενη βροχόπτωση (αναβολή/μείωση).

- Αν ικανοποιούνται οι προϋποθέσεις, ενεργοποιείται το ρελέ και ξεκινά το πότισμα.

- Κατά τη διάρκεια ποτίσματος, το flow meter επιτρέπει την καταγραφή κατανάλωσης νερού, ενώ η κατάσταση (WATERING) προβάλλεται σε LCD και web dashboard.

- Στο τέλος, το σύστημα τερματίζει το πότισμα, καταγράφει event (start/stop, κατανάλωση, λόγος τερματισμού) και επιστρέφει σε IDLE, θέτοντας (αν απαιτείται) cooldown.Μοντέλο δεδομένων.

3.2.4 Ροή αποθήκευσης και οπτικοποίησης (persistence & visualization)

- Ο MQTT broker στο Raspberry Pi δέχεται τα μηνύματα του ESP32.

- Ένας μηχανισμός bridge λαμβάνει τα μηνύματα και τα μετασχηματίζει/καταγράφει σε InfluxDB (time-series).

- Η Grafana συνδέεται στην InfluxDB και προβάλλει dashboards με γραφήματα υγρασίας εδάφους, στάθμης δοχείου, κατανάλωσης νερού, συμβάντων ποτίσματος και βασικών δεικτών καιρού.

- Με τον τρόπο αυτό ο χρήστης βλέπει τόσο την τρέχουσα κατάσταση όσο και ιστορικές τάσεις (π.χ. πώς πέφτει η υγρασία με τον χρόνο και τι επίδραση είχε ένα πότισμα).

3.3 Μοντέλο δεδομένων

Το σύστημα ανταλλάσσει και αποθηκεύει δεδομένα με τρεις κύριους τρόπους: (α) μέσω του τοπικού web API του ESP32 (JSON), (β) μέσω MQTT (τηλεμετρία και εντολές), και (γ) μέσω αποθήκευσης σε InfluxDB (ιστορικό). Η ενοποίηση αυτών των επιπέδων επιτρέπει τόσο “real-time” παρακολούθηση (dashboard/LCD) όσο και ιστορική ανάλυση (Grafana).

3.3.1 Web API του ESP32 — JSON από το GET /data

Το ESP32 παρέχει endpoint GET /data που επιστρέφει ένα ενιαίο JSON αντικείμενο με όλα τα δεδομένα αισθητήρων/κατάστασης, ώστε το web dashboard να ανανεώνεται περιοδικά. Το σχήμα είναι ιεραρχικό και περιλαμβάνει: *state, message, uptime* και επιμέρους ομάδες: *soil, environment, flow, relay, weather, tank*.

Ενδεικτικό σχήμα (skeleton):

```
{
  "state": "IDLE",
  "message": "OK",
  "uptime": 123456,
  "soil": {
    "moisture": 45.2,
    "raw_adc": 1820
  },
  "environment": {
    "temperature": 18.5,
    "humidity": 65.0
  },
  "flow": {
    "grand_total_ml": 2500,
    "last_watering_ml": 45,
    "total_count": 55,
    "auto_count": 50,
    "manual_count": 5
  },
  "relay": {
    "is_on": false,
    "duration_s": 3
  },
  "weather": {
    "...": "..."
  },
  "tank": {
    "distance_cm": 18.2,
    "level_percent": 72,
    "status": "OK"
  }
}
```

3.3.2 MQTT μοντέλο — Topics τηλεμετρίας και εντολών

Η επικοινωνία telemetry γίνεται με MQTT προς broker Mosquitto στο Raspberry Pi (port 1883). Το ESP32 δημοσιεύει περιοδικά δεδομένα (κάθε ~30s), κάνει auto-reconnect (~5s) όταν χρειάζεται και χρησιμοποιεί retained messages ώστε το τελευταίο state/measurement να είναι διαθέσιμο άμεσα σε νέους subscribers.

Topics δημοσίευσης (Publish):

- smartwatering/system/state — κατάσταση (IDLE/WATERING/ERROR)
- smartwatering/soil/moisture — υγρασία εδάφους (%)
- smartwatering/environment/temperature — θερμοκρασία αέρα *(από Weather API)*
- smartwatering/environment/humidity — υγρασία αέρα *(από Weather API)*
- smartwatering/flow/grand_total_ml — συνολική κατανάλωση (mL)
- smartwatering/flow/total_count — αριθμός ποτισμάτων
- smartwatering/weather/rain_3h / rain_12h / rain_24h — εκτίμηση βροχής (mm)
- smartwatering/weather/will_rain — boolean ένδειξη
- smartwatering/weather/temperature / humidity — μετεωρολογικές τιμές

Topics εγγραφής (Subscribe) για εντολές:

- smartwatering/system/command — εντολές τύπου "water", "start", "stop"

Έτσι διαχωρίζεται καθαρά:

- **telemetry** (μετρήσεις/κατάσταση → publish)
- **control** (εντολές χρήστη/συστήματος → subscribe)

3.3.3 Αποθήκευση InfluxDB — Measurements και fields

Στον backend κόμβο χρησιμοποιείται InfluxDB 1.8 με database smartwatering και retention policy forever για διατήρηση ιστορικού.

Η εγγραφή των δεδομένων γίνεται μέσω Python bridge mqtt_to_influx.py (ως systemd service), το οποίο κάνει subscribe στο smartwatering/# και μετατρέπει τα MQTT topics σε points στην InfluxDB, βάσει αντιστοίχισης (topic → measurement/field/type).

Πίνακας 3.1: Σύνοψη Measurements και Fields (InfluxDB)

Measurement	Fields (ενδεικτικά)
soil	moisture
environment *(Weather API)*	temperature, humidity
flow	grand_total_ml, last_watering_ml, total_count, auto_count, manual_count
tank	distance_cm, level_percent
relay	is_on
Weather *(Weather API)*	rain_12h, rain_24h, will_rain
system	state, ip, uptime

3.4 Σχεδιασμός λογικής απόφασης ποτίσματος

Η λογική απόφασης ποτίσματος σχεδιάστηκε ως ένας απλός, προβλέψιμος “κανόνας ελέγχου” (rule-based control), ο οποίος συνδυάζει τη **μέτρηση υγρασίας εδάφους** με την **πρόβλεψη βροχόπτωσης** από το OpenWeatherMap. Στόχος είναι να ποτίζει μόνο όταν υπάρχει πραγματική ανάγκη (ξηρό έδαφος), αλλά να αποφεύγει το πότισμα όταν αναμένεται βροχή σύντομα, ώστε να μειώνεται η σπατάλη νερού.

3.4.1 Είσοδοι (inputs) του αλγορίθμου

Οι βασικές είσοδοι που χρησιμοποιούνται για τη λήψη απόφασης είναι:

- **Υγρασία εδάφους (%)**, η οποία προκύπτει από μέτρηση ADC με αντιστασιακό αισθητήρα και μετατροπή σε ποσοστό βάσει βαθμονόμησης (π.χ. ADC 3000 → 0% “ξηρό”, ADC 700 → 100% “υγρό”), με averaging/median filter για μείωση θορύβου.
- **Πρόβλεψη βροχής** σε συγκεκριμένο χρονικό παράθυρο (κυρίως 12h), όπως εξάγεται από το OpenWeatherMap Forecast API.
- **Κατάσταση συστήματος και χρονισμοί ασφαλείας**, όπως cooldown μεταξύ κύκλων, μέγιστη διάρκεια αντλίας, emergency stop κ.λπ.
- **(Υποστηρικτικά)** στάθμη δοχείου και ροή νερού χρησιμοποιούνται για επίβλεψη και αποφυγή λανθασμένων ενεργοποιήσεων/καταγραφών (π.χ. χαμηλή στάθμη, μηδενική ροή).

3.4.2 Παράμετροι και χρονισμοί (tunable parameters)

Οι βασικές παράμετροι που καθορίζουν τη συμπεριφορά είναι συγκεντρωμένες σε κεντρικό αρχείο ρυθμίσεων, ώστε να αλλάζουν εύκολα χωρίς τροποποίηση της συνολικής λογικής:

- **Όριο υγρασίας (soil threshold):** 30% (αν κάτω από αυτό θεωρείται “ξηρό”).
- **Όριο βροχής (rain threshold):** αν προβλέπεται βροχή **> 2 mm** στις επόμενες **12 ώρες**, το πότισμα ακυρώνεται.
- **Δόση ποτίσματος:** ~100 mL ανά κύκλο, με **10 δευτερόλεπτα** λειτουργίας αντλίας.
- **Διάβασμα αισθητήρων:** κάθε **2 δευτερόλεπτα** (soil/tank κ.λπ.).
- **Έλεγχος “χρειάζεται πότισμα;”:** κάθε **1 λεπτό**.
- **Cooldown μετά από πότισμα: 10 δευτερόλεπτα** πριν επιτραπεί νέος κύκλος.
- **Ανανέωση καιρού:** περίπου κάθε **1 ώρα**.
- **Δημοσίευση MQTT:** κάθε **30 δευτερόλεπτα**.

3.4.3 Κανόνες απόφασης (decision rules)

Η απόφαση ποτίσματος βασίζεται στα ακόλουθα βήματα:

1. **Έλεγχος ξηρότητας εδάφους:**
   Αν soil_moisture < 30%, τότε το σύστημα προχωρά σε έλεγχο καιρού.

2. **Έλεγχος πρόβλεψης βροχής:**
   Γίνεται ερώτηση στο Weather API: “αναμένεται βροχή > 2 mm στις επόμενες 12 ώρες;”
   - Αν **ΝΑΙ** → **ακύρωση** ποτίσματος (skip).
   - Αν **ΟΧΙ** → **έναρξη** ποτίσματος.
3. **Εκτέλεση ποτίσματος (dose-based):**
   Ενεργοποίηση αντλίας μέσω ρελέ για ~10 sec (στόχος ~100 mL). Κατά τη διάρκεια, καταγράφεται ροή/κατανάλωση μέσω flow meter.
4. **Μετά το πότισμα:**
   Καταγραφή στατιστικών (π.χ. κατανάλωση, counts), και ενεργοποίηση cooldown 10 sec.

Safety fallback:
Αν δεν υπάρχουν έγκυρα δεδομένα καιρού (π.χ. αποτυχία API, parse error), το σύστημα **ποτίζει κανονικά** όταν το έδαφος είναι ξηρό, ώστε να μην “αδικήσει” το φυτό λόγω προσωρινής απώλειας internet/API.

3.4.4 Μηχανή καταστάσεων (state machine)

Η υλοποίηση οργανώνεται ως state machine, ώστε η συμπεριφορά να είναι καθαρή και να αποφεύγονται ασαφείς μεταβάσεις:

- **IDLE:** αναμονή, περιοδική συλλογή αισθητήρων και χρονικός έλεγχος ανά 1 λεπτό.
- **CHECKING:** έλεγχος συνθηκών (υγρασία < 30% + κανόνας βροχής).
- **WATERING:** ενεργό πότισμα (6 sec) και ταυτόχρονη μέτρηση ροής.
- **COMPLETED:** καταγραφή/ενημέρωση counters και cooldown 10 sec.
- **ERROR:** κατάσταση σφάλματος/κλειδώματος (π.χ. emergency stop), όπου απαιτείται παρέμβαση/επανεκκίνηση ανάλογα με το σενάριο.

3.4.5 Ασφαλιστικές δικλείδες και προστασίες

Για αποφυγή επικίνδυνων ή ανεπιθύμητων ενεργοποιήσεων, εφαρμόζονται συγκεκριμένοι μηχανισμοί ασφαλείας:

- **Μέγιστος χρόνος λειτουργίας αντλίας:** 15 sec → αυτόματο emergency stop αν ξεπεραστεί.
- **Debounce ρελέ:** 50 ms μεταξύ εντολών.
- **Ελάχιστος χρόνος OFF μεταξύ κύκλων:** 10 sec (συνδυάζεται με cooldown).
- **Emergency stop:** διαθέσιμο μέσω web interface, MQTT ή εσωτερικής λογικής.
- **Watchdog timer:** επανεκκίνηση σε περίπτωση “κολλήματος”.
- **Ανίχνευση σφαλμάτων αισθητήρων:** NaN/out-of-range → χρήση cached τιμής/ασφαλής συμπεριφορά.

3.5 Διεπαφές ελέγχου και καταγραφές συστήματος

Για να είναι το σύστημα λειτουργικό στην πράξη (όχι μόνο “να ποτίζει”), σχεδιάστηκαν σαφείς διεπαφές επικοινωνίας και μηχανισμοί καταγραφής. Οι διεπαφές εξυπηρετούν δύο βασικούς στόχους: (α) **παρακολούθηση** της κατάστασης και των μετρήσεων σε πραγματικό χρόνο, και (β) **έλεγχο** του ποτίσματος (αυτόματα ή χειροκίνητα) με ασφαλή τρόπο. Παράλληλα, η καταγραφή συμβάντων (events) και μετρήσεων εξασφαλίζει ιστορικό, ιχνηλασιμότητα και διευκολύνει τη διάγνωση προβλημάτων (debugging).

3.5.1 Διεπαφή Web (ESP32)

Το ESP32 λειτουργεί ως τοπικός web server, προσφέροντας ένα απλό dashboard στο οποίο ο χρήστης μπορεί να δει την τρέχουσα κατάσταση και βασικές μετρήσεις, καθώς και να εκτελέσει χειροκίνητες εντολές. Η επιλογή τοπικής web διεπαφής μειώνει την πολυπλοκότητα, καθώς δεν απαιτείται εξωτερική εφαρμογή για βασικό χειρισμό, ενώ είναι ιδιαίτερα χρήσιμη σε περιπτώσεις αρχικής ρύθμισης ή troubleshooting.

Η web διεπαφή οργανώνεται γύρω από συγκεκριμένα endpoints:

- **GET /**: προβάλλει το dashboard (HTML) με τις τρέχουσες ενδείξεις.
- **GET /data**: επιστρέφει τα δεδομένα σε JSON μορφή, ώστε το dashboard να κάνει περιοδική ανανέωση χωρίς επαναφόρτωση της σελίδας.
- **POST /water**: εκτελεί χειροκίνητο πότισμα (manual watering) με προκαθορισμένη “δόση” (διάρκεια/όγκο).
- **POST /stop**: ενεργοποιεί emergency stop, τερματίζοντας άμεσα το πότισμα και κλειδώνοντας/περιορίζοντας ενέργειες ανάλογα με τη λογική ασφαλείας.

Στη σχεδίαση δόθηκε έμφαση στο να είναι η διεπαφή “λειτουργική” και όχι βαριά: εμφανίζει τα κρίσιμα μεγέθη (soil moisture, tank level, water usage, system state) και παρέχει καθαρές ενέργειες χρήστη (water/stop). Επιπλέον, οι περιβαλλοντικές τιμές (θερμοκρασία/υγρασία αέρα) εμφανίζονται ως ενημερωτική πληροφορία από το Weather API, χωρίς να απαιτείται τοπικός αισθητήρας περιβάλλοντος στην τελική υλοποίηση.

3.5.2 Διεπαφή MQTT (Telemetry & Commands)

Για τη συνεχή ροή δεδομένων προς το backend και για την απομακρυσμένη διασύνδεση, χρησιμοποιείται MQTT. Το MQTT διευκολύνει την αποστολή τηλεμετρίας (publish) σε τακτά χρονικά διαστήματα και επιτρέπει την ύπαρξη πολλαπλών subscribers (π.χ. bridge προς InfluxDB, εργαλεία παρακολούθησης, μελλοντικές επεκτάσεις) χωρίς να αλλάζει το edge device.

Η επικοινωνία χωρίζεται σε δύο κατηγορίες:

(α) Telemetry (Publish):
Το ESP32 δημοσιεύει περιοδικά μετρήσεις και κατάσταση, όπως υγρασία εδάφους, στάθμη δοχείου, ροή/κατανάλωση, δείκτες καιρού (π.χ. αναμενόμενη βροχή σε 12h/24h) και system state. Στόχος είναι να υπάρχει σταθερή ροή δεδομένων, που να τροφοδοτεί το ιστορικό και τα dashboards.

(β) Commands (Subscribe):
Το ESP32 εγγράφεται σε topic εντολών, ώστε να μπορεί να δεχθεί:

- εντολή για χειροκίνητο πότισμα,
- εντολή για άμεσο τερματισμό (stop),
- ενδεχομένως ειδικές εντολές συντήρησης/ελέγχου (ανάλογα με την υλοποίηση).

Σε επίπεδο σχεδίασης, ο τρόπος αυτός είναι πιο “καθαρός” από ένα μοντέλο όπου όλα περνούν μόνο από web requests, επειδή:

- το backend μπορεί να δώσει εντολές χωρίς να χρειάζεται άμεση πρόσβαση στο local web UI,
- οι εντολές και η τηλεμετρία ακολουθούν ενιαίο μηχανισμό και μπορούν να καταγράφονται.

Για τη σταθερότητα του συστήματος υιοθετούνται πρακτικές όπως reconnect πολιτική, περιοδική δημοσίευση “state” και (όπου χρειάζεται) retained μηνύματα για να είναι διαθέσιμη η τελευταία γνωστή κατάσταση σε νέους consumers.

3.5.3 Καταγραφές, συμβάντα (events) και ιστορικό

Η καταγραφή αποτελεί κομβικό μέρος του συστήματος, γιατί επιτρέπει:

- να αποδεικνύεται τι συνέβη (πότε πότισε, πόσο πότισε, γιατί δεν πότισε),
- να γίνεται διάγνωση σφαλμάτων (sensor out-of-range, API failure, manual stop),
- να παρουσιάζεται το σύστημα ως ολοκληρωμένο (monitoring + control + traceability).

Στη σχεδίαση διαχωρίζονται δύο επίπεδα καταγραφής:

(α) Μετρήσεις (metrics / time-series):
Ό,τι μετριέται σε συνεχόμενη ροή (soil moisture, tank level, flow, rainfall indicators, system uptime) αποθηκεύεται ως χρονοσειρά, ώστε να απεικονίζεται με γραφήματα και να συσχετίζεται χρονικά (π.χ. πτώση υγρασίας → πότισμα → άνοδος).

(β) Συμβάντα (events):
Τα events είναι “σημεία” που εξηγούν συμπεριφορά και αποφάσεις. Ενδεικτικά:

- **Start watering / Stop watering** (μαζί με διάρκεια/όγκο/λόγο τερματισμού),
- **Skip watering λόγω βροχής** (όταν ο κανόνας πρόβλεψης βροχής ακυρώνει πότισμα),
- **Low tank / Tank empty** (αποτροπή ποτίσματος λόγω χαμηλής στάθμης),
- **Emergency stop ενεργοποίηση** (από web/MQTT ή safety rule),
- **Sensor/API error** (invalid readings, αποτυχία λήψης forecast, κ.λπ.).

Τα παραπάνω events είναι ιδιαίτερα σημαντικά για να “δικαιολογείται” η λειτουργία του συστήματος: δεν αρκεί να φαίνεται μόνο ότι “έγινε πότισμα”, αλλά και *γιατί έγινε* ή *γιατί δεν έγινε*. Επιπλέον, με το ιστορικό σε InfluxDB και την οπτικοποίηση σε Grafana, ο χρήστης μπορεί να δει τόσο την εξέλιξη των μετρήσεων όσο και τα χρονικά σημεία των ενεργειών (events), σχηματίζοντας πλήρη εικόνα λειτουργίας.

4 Υλοποίηση Edge Device (ESP32-S3)

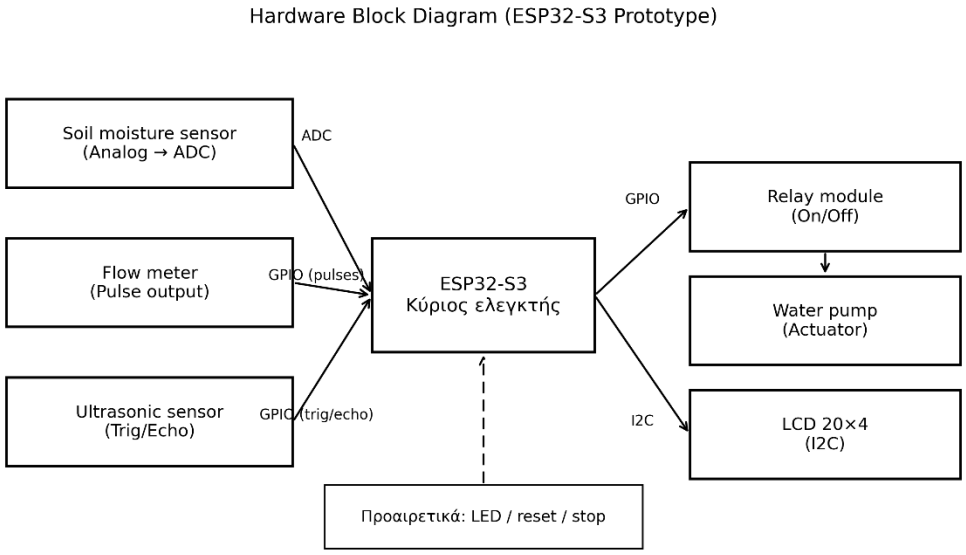
4.1 Υλικό και συνδεσμολογία

Το edge device του συστήματος βασίζεται σε ESP32-S3 και έχει ως στόχο τη συλλογή μετρήσεων από αισθητήρες, την εκτέλεση της λογικής απόφασης ποτίσματος και τον έλεγχο της αντλίας μέσω ρελέ. Παράλληλα παρέχει τοπική ένδειξη κατάστασης μέσω LCD 20×4 (I2C) και τοπικό web dashboard για monitoring/χειρισμό. [1]–[4]

4.1.1 Συστατικά (BOM – βασικά μέρη)

Η υλοποίηση περιλαμβάνει τα εξής βασικά υποσυστήματα:

- **ESP32-S3** (κύριος ελεγκτής)
- **Soil moisture sensor (analog)** για μέτρηση υγρασίας εδάφους (ADC input)
- **Flow meter** για καταγραφή κατανάλωσης νερού (pulse output)
- **Ultrasonic sensor** για εκτίμηση στάθμης δοχείου (trig/echo)
- **Relay module** για on/off έλεγχο αντλίας
- **Water pump** (ή αντίστοιχος ενεργοποιητής)
- **LCD 20×4 με I2C backpack** για τοπική απεικόνιση



Εικόνα 4-1: Hardware block diagram

4.1.2 Συνδεσμολογία αισθητήρων και ενεργοποιητών

Η συνδεσμολογία ακολουθεί πρακτική “χαμηλής πολυπλοκότητας”, ώστε κάθε αισθητήρας να συνδέεται σε κατάλληλο τύπο pin:

- **Soil moisture (analog)** → σε **ADC pin** του ESP32-S3
  Χρησιμοποιείται αναλογική ανάγνωση και μετατροπή σε ποσοστό μέσω βαθμονόμησης (dry/wet calibration).
- **Flow meter (pulse)** → σε **digital pin / interrupt-capable pin**
  Κάθε παλμός αντιστοιχεί σε ποσότητα νερού (ρυθμισμένη μέσω calibration factor). Μετρώντας pulses/δευτερόλεπτο ή pulses/κύκλο, υπολογίζεται συνολικός όγκος (mL).
- **Ultrasonic (trig/echo)** → σε **2 digital pins**
  Υπολογίζεται απόσταση (cm) και μετατρέπεται σε εκτίμηση στάθμης/ποσοστού δοχείου, με όρια (min/max) και κατηγορίες κατάστασης (OK/LOW/EMPTY).
- **Relay module** → σε **digital output pin**
  Οδηγεί την αντλία on/off. Εφαρμόζονται προστασίες όπως max runtime, cooldown και emergency stop.
- **LCD 20×4 (I2C)** → **SDA/SCL** pins του ESP32-S3
  Χρησιμοποιείται για άμεση τοπική ένδειξη (soil moisture, tank status, watering state, basic alerts).

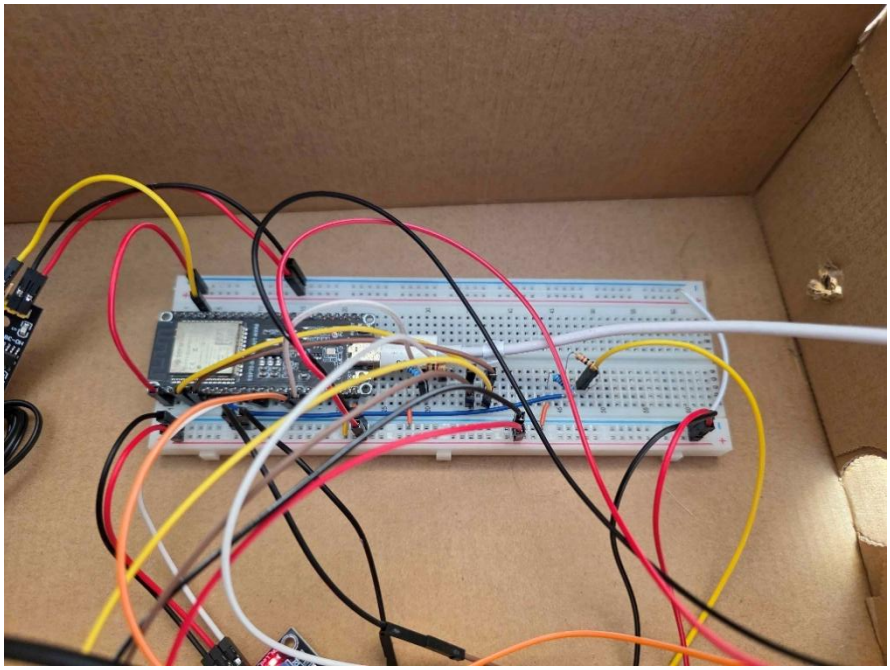
Πίνακας 4.1: Σύνοψη συνδεσμολογίας

Sensor/Module	ESP32 pin	Type	Notes
Soil moisture (Analog)	GPIO1	ADC	0–3.3V, calibrate thresholds
Flow meter (Pulse)	GPIO7	Digital IN	Interrupt / pulses→mL
Ultrasonic TRIG	GPIO9	Digital OUT	Trigger pulse (10μs)
Ultrasonic ECHO	GPIO10	Digital IN	Αν χρειάζεται level shifting
Relay IN	GPIO8	Digital OUT	Active LOW/ HIGH (ανά module)
LCD SDA	GPIO5	I2C	κοινό bus
LCD SCL	GPIO4	I2C	κοινό bus

4.1.3 Τροφοδοσία και πρακτικές ασφάλειας

Η τροφοδοσία του ESP32 γίνεται από σταθερή πηγή (USB/5V regulator), ενώ η αντλία τροφοδοτείται από ξεχωριστή γραμμή σύμφωνα με τις προδιαγραφές της (π.χ. 5V/12V). Το relay λειτουργεί ως ηλεκτρική απομόνωση της ισχύος της αντλίας από τον μικροελεγκτή. Για τη σταθερότητα της λειτουργίας λαμβάνονται μέτρα όπως:

- κοινή γείωση όπου απαιτείται (GND reference),
- αποφυγή θορύβου σε analog readings (soil sensor),
- χρονικά όρια λειτουργίας αντλίας (max runtime) και cooldown,
- emergency stop (software) ώστε να διακοπεί άμεσα το πότισμα.



Εικόνα 4-2: Φωτογραφία breadboard/συνδεσμολογίας

4.1.4 Τοπική απεικόνιση (LCD)

Η LCD χρησιμοποιείται για να φαίνεται άμεσα:

- soil moisture (%), rain forecast 12h (mm)
- temperature (°C), humidity (%) από OpenWeatherMap API
- tank level % / status (OK, LOW!)
- μηνύματα κατάστασης (π.χ. "WATERING...", "RAIN EXPECTED", "LOW TANK", "SOIL DRY!", "IDLE")

Έτσι, το σύστημα παραμένει χρήσιμο ακόμα και χωρίς πρόσβαση σε υπολογιστή ή Grafana.

Εικόνα 4-3: Φωτογραφία LCD. (1)

Εικόνα 4-4: Φωτογραφία LCD. (2)

Εικόνα 4-5: Φωτογραφία LCD. (3)

4.2 Δομή λογισμικού (αρθρωτή αρχιτεκτονική & βασικές ρυθμίσεις)

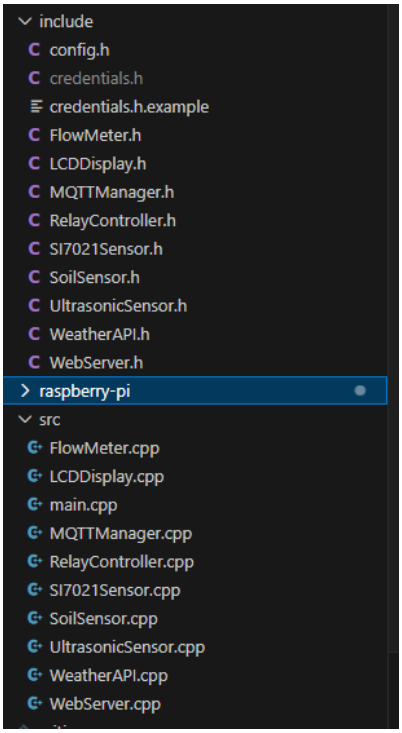
Η υλοποίηση του edge device στο ESP32-S3 σχεδιάστηκε με αρθρωτή (modular) λογική, ώστε κάθε βασική λειτουργία (αισθητήρες, έλεγχος αντλίας, καιρός, επικοινωνία, διεπαφές) να είναι όσο γίνεται ανεξάρτητη. Με αυτόν τον τρόπο, το σύστημα γίνεται ευκολότερο στη συντήρηση, επιτρέπει γρήγορες αλλαγές στις παραμέτρους και μπορεί να επεκταθεί με επιπλέον αισθητήρες ή νέες λειτουργίες χωρίς να απαιτείται ανασχεδιασμός του συνολικού κώδικα.

4.2.1 Δομή έργου και αρχεία ρυθμίσεων

Σε επίπεδο κώδικα, το έργο οργανώνεται σε βασικές ενότητες που αντιστοιχούν στα υποσυστήματα του συστήματος. Ενδεικτικά, υπάρχουν modules για:

- **Ανάγνωση αισθητήρων** (soil moisture, flow meter, ultrasonic tank level),
- **Έλεγχο ενεργοποιητή** (relay/αντλία και προστασίες),
- **Ενσωμάτωση μετεωρολογικών δεδομένων** (λήψη και επεξεργασία forecast από OpenWeatherMap),
- **Επικοινωνία** (MQTT publish/subscribe [5]),
- **Web server / dashboard** (endpoints για monitoring και χειρισμό),
- **LCD UI** (τοπική απεικόνιση κατάστασης).

Η κεντρική λογική του προγράμματος βρίσκεται στο κύριο αρχείο του firmware (π.χ. main.cpp), το οποίο καλεί τις επιμέρους λειτουργίες των modules και διαχειρίζεται τη “ροή” εργασιών.



Εικόνα 4-6: Φωτογραφία project tree

Ιδιαίτερη σημασία έχει η ύπαρξη αρχείων ρυθμίσεων, ώστε κρίσιμες παράμετροι να ορίζονται συγκεντρωμένα και να μπορούν να τροποποιηθούν χωρίς αλλαγές σε πολλά σημεία. Στο έργο είναι χρήσιμο να υπάρχουν (ή να οριστούν) αρχεία όπως:

- **config.h**: περιέχει παραμέτρους λειτουργίας (thresholds, χρονισμοί, calibration constants).
- **credentials.h.example** (ή αντίστοιχο): δείχνει τη μορφή των credentials (Wi-Fi SSID/PASS, API key OpenWeatherMap, MQTT broker IP), χωρίς να αποκαλύπτει πραγματικές τιμές.
- **credentials.h**: το πραγματικό αρχείο credentials, το οποίο δεν πρέπει να διαμοιράζεται δημόσια.

Με αυτή τη διάκριση, διασφαλίζεται ότι το έργο μπορεί να αναπαραχθεί ή να παρουσιαστεί χωρίς έκθεση ευαίσθητων πληροφοριών (π.χ. κωδικών Wi-Fi ή API key).

```
123  // Βαθμονόμηση ADC → Ποσοστό υγρασίας (0-100%)
124  // ⚠️ RESISTIVE SENSOR: Υψηλό ADC (≥3000) = ΞΗΡΟ, Χαμηλό ADC (≤700) = ΥΓΡΟ
125  // Range: ADC 700 (wet) → 3000 (dry) για πρακτική χρήση σε γλάστρα
126  #define SOIL_ADC_DRY          3000  // ADC value σε πολύ ξηρό έδαφος (0% moisture)
127  #define SOIL_ADC_WET          700   // ADC value σε πολύ υγρό έδαφος (100% moisture)
128
129  // Κατώφλια ενεργοποίησης ποτίσματος (Chamaedorea elegans - μικρή γλάστρα)
130  #define SOIL_MOISTURE_MIN     30.0  // % - Αν soil < 30% → ξεκινά πότισμα
131  #define SOIL_MOISTURE_TARGET  35.0  // % - Στόχος υγρασίας (σταματά όταν φτάσει εδώ)
132
133  // Φίλτρο θορύβου (median filter)
134  #define SOIL_FILTER_SIZE      5     // Χρησιμοποιεί τις τελευταίες 5 μετρήσεις
```

Εικόνα 4-7: Απόσπασμα από config.h με ενδεικτικές παραμέτρους

4.2.2 Κύκλος λειτουργίας (setup/loop) και χρονοπρογραμματισμός εργασιών

Η λειτουργία του συστήματος ακολουθεί το κλασικό μοντέλο firmware σε μικροελεγκτή: αρχικοποίηση στο setup() και επαναλαμβανόμενη εκτέλεση στο loop(). Στο setup() πραγματοποιούνται οι βασικές αρχικοποιήσεις υποσυστημάτων, ενώ στο loop() εκτελούνται περιοδικά εργασίες με βάση χρονισμούς (timers), ώστε να αποφεύγεται blocking συμπεριφορά και να παραμένει το σύστημα “responsive”.

Αρχικοποίηση (setup()):
Κατά την εκκίνηση εκτελούνται ενδεικτικά τα παρακάτω βήματα:

1. Αρχικοποίηση σειριακής επικοινωνίας (debug).
2. Αρχικοποίηση αισθητήρων (ADC/interrupt pins/ultrasonic pins).
3. Αρχικοποίηση LCD και εμφάνιση αρχικού status.
4. Σύνδεση στο Wi-Fi.
5. Εκκίνηση web server και δήλωση των endpoints.
6. Εκκίνηση MQTT client και σύνδεση στον broker.
7. Αρχικοποίηση παραμέτρων κατάστασης (state machine) και counters.

```
void setup() {
  ''

  Serial.println("📦 Αρχικοποίηση modules...\n");

  // Αρχικοποίηση I2C bus ΠΡΩΤΑ (shared από SI7021 και LCD)
  Wire.begin(PIN_I2C_SDA, PIN_I2C_SCL);
  Serial.printf("🔌 I2C Bus: SDA=GPIO%d, SCL=GPIO%d\n\n", PIN_I2C_SDA, PIN_I2C_SCL);

  // 1. Relay Controller
  if (!relay.begin()) {
    Serial.println("❌ ΣΦΑΛΜΑ: Relay initialization failed!");
  }

  // 2. Soil Sensor
  if (!soilSensor.begin()) {
    Serial.println("❌ ΣΦΑΛΜΑ: Soil sensor initialization failed!");
  }

  // 3. SI7021 Sensor - DISABLED (χρήση Weather API data)
  // if (!si7021.begin()) {
  //   Serial.println("⚠️  ΠΡΟΕΙΔΟΠΟΙΗΣΗ: SI7021 initialization failed!");
  // }
  Serial.println("ℹ️  SI7021: Disabled - Using Weather API data");

  // 4. Flow Meter
  if (!flowMeter.begin()) {
    Serial.println("❌ ΣΦΑΛΜΑ: Flow meter initialization failed!");
  }

  // 5. Ultrasonic Tank Sensor
  if (!tankSensor.begin()) {
    Serial.println("⚠️  ΠΡΟΕΙΔΟΠΟΙΗΣΗ: Tank sensor initialization failed!");
    Serial.println("   Το σύστημα θα λειτουργεί χωρίς μέτρηση στάθμης νερού");
  }

  // 6. LCD Display
  if (!lcdDisplay.begin()) {
    Serial.println("⚠️  ΠΡΟΕΙΔΟΠΟΙΗΣΗ: LCD Display initialization failed!");
    Serial.println("   Το σύστημα θα λειτουργεί χωρίς LCD");
  }

  Serial.println();
```

Εικόνα 4-8: Απόσπασμα του setup() που δείχνει σειρά αρχικοποιήσεων (Wi-Fi, MQTT, web server, sensors, LCD).(1)

```
// ═══════════════════════════════════════════════════════════
// WIFI  & NETWORK
// ═══════════════════════════════════════════════════════════

#if ENABLE_WEATHER_CHECK
  initWiFi(); // WiFi required για weather API

  // 5. Weather API
  if (!weatherAPI.begin()) {
    Serial.println("⚠️  ΠΡΟΕΙΔΟΠΟΙΗΣΗ: Weather API initialization failed!");
    Serial.println("   Το σύστημα θα λειτουργεί χωρίς έλεγχο καιρού");
  }

  // 6. Web Server
  #if ENABLE_WEB_SERVER
    if (!webServer.begin(&soilSensor, &si7021, &flowMeter, &relay, &weatherAPI, &tankSensor,
                         &stateName, &stateMessage, &emergencyStop, &manualWateringRequest)) {
      Serial.println("⚠️  ΠΡΟΕΙΔΟΠΟΙΗΣΗ: Web Server initialization failed!");
    }
  #endif

  // 7. MQTT
  #if ENABLE_MQTT
    if (!mqttManager.begin(&soilSensor, &si7021, &flowMeter, &relay, &weatherAPI,
                           &stateMessage)) {
      Serial.println("⚠️  ΠΡΟΕΙΔΟΠΟΙΗΣΗ: MQTT initialization failed!");
    }
  #endif

#else
  // initWiFi(); // Uncomment αν θέλεις WiFi χωρίς weather
#endif

// ═══════════════════════════════════════════════════════════
// WATCHDOG
// ═══════════════════════════════════════════════════════════
```

FlowMeter flowMeter

Generate Copilot summary

Εικόνα 4-9: Απόσπασμα του setup() που δείχνει σειρά αρχικοποιήσεων (Wi-Fi, MQTT, web server, sensors, LCD).(2)

Επαναλαμβανόμενη λειτουργία (loop()):
Στο loop() δεν εκτελείται συνεχώς “ένα μεγάλο κομμάτι κώδικα”, αλλά μικρές περιοδικές εργασίες με διαφορετικές συχνότητες. Ενδεικτικά:

- **Ανάγνωση αισθητήρων (κάθε ~2 sec):** ενημέρωση soil moisture, tank level, flow counters.
- **Έλεγχος “χρειάζεται πότισμα;” (κάθε ~1 min):** εκτέλεση της rule-based λογικής (soil threshold + rain forecast), με εφαρμογή cooldown και ασφαλιστικών περιορισμών.
- **Ενημέρωση καιρού (κάθε ~1 hour):** κλήση OpenWeatherMap και υπολογισμός παραγώγων (3h/12h/24h rain, will_rain).
- **Δημοσίευση MQTT (κάθε ~30 sec):** αποστολή telemetry και system state, ώστε το backend να ενημερώνεται και να καταγράφει ιστορικό.
- **Web server handling (συνεχώς):** εξυπηρέτηση requests για /data, /water, /stop.
- **LCD update (ανά κατάλληλο διάστημα):** ανανέωση ενδείξεων, χωρίς υπερβολικά συχνό refresh.

Η παραπάνω προσέγγιση λειτουργεί ως ένας απλός “χρονοπρογραμματιστής” (scheduler) που βασίζεται σε timestamps (millis()), αποφεύγοντας καθυστερήσεις (delay()) που θα επηρέαζαν την απόκριση του συστήματος ή την εξυπηρέτηση web/MQTT.

```
void loop() {
  unsigned long now = millis();

  // ═══════════════════════════════════════════════════════════
  // WATCHDOG RESET
  // ═══════════════════════════════════════════════════════════

  #if ENABLE_WATCHDOG
    esp_task_wdt_reset();
  #endif

  // ═══════════════════════════════════════════════════════════
  // UPDATE MODULES
  // ═══════════════════════════════════════════════════════════

  relay.update();        // Έλεγχος safety timeout
  flowMeter.update();    // Υπολογισμός flow rate

  // Weather update (αν ενεργοποιημένο)
  #if ENABLE_WEATHER_CHECK
    if (weatherAPI.needsUpdate()) {
      weatherAPI.update();
    }
  #endif

  // Web Server update
  #if ENABLE_WEB_SERVER
    webServer.update();
  #endif

  // MQTT update
  #if ENABLE_MQTT
    mqttManager.update();
  #endif

  // ═══════════════════════════════════════════════════════════
  // ΔΙΑΒΑΣΜΑ SENSORS (κάθε 5 δευτερόλεπτα)
  // ═══════════════════════════════════════════════════════════

  if (now - lastSensorRead >= SENSOR_READ_INTERVAL) {
    updateSensors();
    lastSensorRead = now;
  }
```

Εικόνα 4-10: Απόσπασμα του loop() που δείχνει την κατανομή των εργασιών. (1)

```
// LCD Display update - smart status message
const char* lcdStatus;
if (currentState == SystemState::WATERING) {
  lcdStatus = ">>> WATERING... <<<";
} else if (tankSensor.getLastLevel() >= 0 && tankSensor.getLastLevel() <= TANK_LOW_LEVEL) {
  lcdStatus = "!! LOW TANK !!";
} else if (soilSensor.getLastMoisture() < SOIL_MOISTURE_MIN) {
  lcdStatus = "SOIL DRY!";
} else if (weatherAPI.getRain12h() >= 2.0) {
  lcdStatus = "RAIN EXPECTED";
} else if (currentState == SystemState::ERROR) {
  lcdStatus = "ERROR!";
} else {
  lcdStatus = "IDLE";
}

lcdDisplay.update(
  soilSensor.getLastMoisture(),
  weatherAPI.getTemperature(),
  weatherAPI.getHumidity(),
  weatherAPI.getRain12h(),
  tankSensor.getLastLevel(),
  lcdStatus
);

// ═══════════════════════════════════════════════════════════
// STATE MACHINE
// ═══════════════════════════════════════════════════════════

// Update stateName για το web interface
switch (currentState) {
  case SystemState::IDLE: stateName = "IDLE"; break;
  case SystemState::CHECKING: stateName = "CHECKING"; break;
  case SystemState::WATERING: stateName = "WATERING"; break;
  case SystemState::COMPLETED: stateName = "COMPLETED"; break;
  case SystemState::ERROR: stateName = "ERROR"; break;
}
```

Εικόνα 4-11: Απόσπασμα του loop() που δείχνει την κατανομή των εργασιών. (2)

```
switch (currentState) {

  case SystemState::IDLE:
    // Έλεγχος manual watering request από web interface
    if (manualWateringRequest) {
      manualWateringRequest = false;
      currentState = SystemState::WATERING;
      stateMessage = "Manual watering...";
      flowMeter.reset();
      flowMeter.enable();  // Ενεργοποίηση interrupt
      Serial.println("💧 MANUAL WATERING - Ξεκίνημα χειροκίνητου ποτίσματος!");
      break;
    }

    // Αναμονή - έλεγχος κάθε 1 λεπτό
    if (now - lastCheck >= CHECK_INTERVAL_MS) {
      currentState = SystemState::CHECKING;
      lastCheck = now;
    }
    break;

  case SystemState::CHECKING:
    checkWateringNeeded();
    break;

  case SystemState::WATERING:
    // Έλεγχος emergency stop
    if (emergencyStop) {
      relay.turnOff();
      flowMeter.disable();  // Απενεργοποίηση interrupt
      flowMeter.finishWatering(false);  // Αποθήκευση (auto watering)
      flowMeter.reset();
      currentState = SystemState::IDLE;
      stateMessage = "Emergency stop!";
      emergencyStop = false;
      Serial.println("⚠️  EMERGENCY STOP - Τερματισμός ποτίσματος!");
      break;
    }
    runWateringCycle();
    break;
```

Εικόνα 4-12: Απόσπασμα του loop() που δείχνει την κατανομή των εργασιών. (3)

```
  case SystemState::COMPLETED:
    // Cooldown - περίμενε 10 δευτερόλεπτα
    if (now - wateringStartTime >= RELAY_MIN_OFF_TIME_MS) {
      currentState = SystemState::IDLE;
      stateMessage = "Επιστροφή σε IDLE";
      Serial.println("✅ Cooldown complete → IDLE\n");
    }
    break;

  case SystemState::ERROR:
    // Error state - χρειάζεται manual unlock
    if (relay.isRelayLocked()) {
      Serial.println("🚨 ERROR STATE: Relay locked!");
      Serial.printf("   Reason: %s\n", relay.getLockReason().c_str());
      Serial.println("   Κάνε reset το ESP32 για unlock\n");
    }
    delay(5000);
    break;
}

// ═══════════════════════════════════════════════════════════
// PRINT STATUS (κάθε 10 δευτερόλεπτα)
// ═══════════════════════════════════════════════════════════

static unsigned long lastStatusPrint = 0;
if (now - lastStatusPrint >= 10000) {
  printStatus();
  lastStatusPrint = now;
}

// Small delay
delay(50);
}
```

Εικόνα 4-13: Απόσπασμα του loop() που δείχνει την κατανομή των εργασιών. (4)

Πίνακας 4.2: Συγκεντρωτικός πίνακας χρόνων

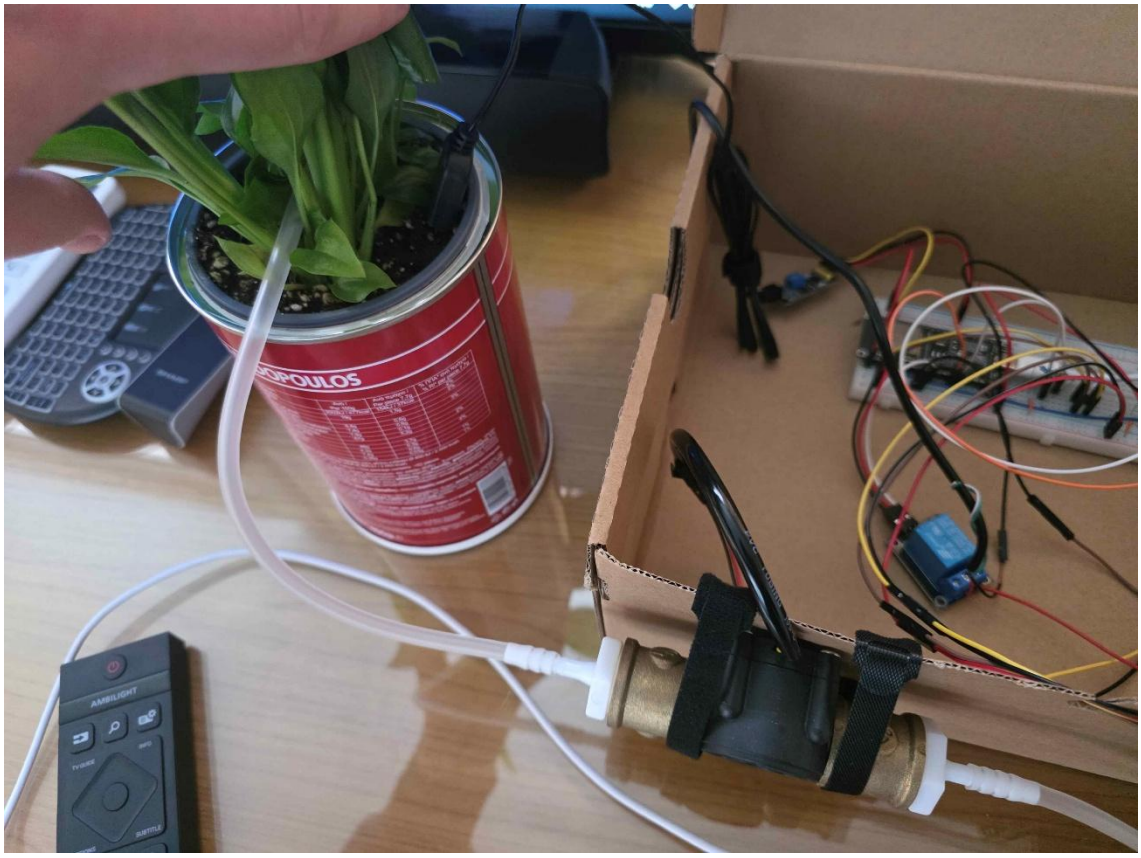
Εργασία	Περίοδος	Σκοπός
Ανάγνωση αισθητήρων (soil/tank/flow)	~2 s	Ενημέρωση μετρήσεων
Έλεγχος απόφασης ποτίσματος	~1 min	Thresholds + καιρός
MQTT publish	~30 s	Telemetry/state → backend
Ενημέρωση καιρού (OpenWeatherMap)	~1 h	Δείκτες βροχής (rain_12h/24h, will_rain)
Ενημέρωση LCD	~1–2 s / on-change	Τοπική ένδειξη κατάστασης
Web dashboard refresh (/data)	~1–5 s	Real-time εμφάνιση στον browser
Cooldown μετά από πότισμα	~10 s	Αποφυγή συνεχόμενων κύκλων
Max runtime αντλίας (safety stop)	~15 s	Προστασία αντλίας/συστήματος

Διαχείριση κατάστασης (state machine):
Παράλληλα με τα timers, η συμπεριφορά του ποτίσματος οργανώνεται σε καταστάσεις (IDLE/CHECKING/WATERING/COMPLETED/ERROR). Αυτό καθιστά σαφές πότε επιτρέπεται η ενεργοποίηση του ρελέ, πότε γίνεται ενημέρωση counters και πότε εφαρμόζονται περιορισμοί (cooldown, max runtime). Με τον τρόπο αυτό αποφεύγονται ασαφείς μεταβάσεις και μειώνονται τα σφάλματα, ειδικά όταν υπάρχουν ταυτόχρονα web requests, MQTT commands και αυτόματος έλεγχος.

Σημείωση για τα περιβαλλοντικά δεδομένα:
Σε προηγούμενη εκδοχή του πρωτοτύπου είχε εξεταστεί η χρήση τοπικού αισθητήρα περιβάλλοντος. Στην τελική υλοποίηση, θερμοκρασία και υγρασία αέρα αντλούνται από το OpenWeatherMap Forecast API, οπότε το firmware διατηρεί την έννοια του “environment” ως λογική ενότητα δεδομένων, αλλά χωρίς πρόσθετο hardware αισθητήρα.

4.3 Μέτρηση υγρασίας εδάφους και βαθμονόμηση (ADC thresholds)

Η μέτρηση υγρασίας εδάφους αποτελεί το βασικότερο σήμα εισόδου του συστήματος, καθώς από αυτήν εξαρτάται η ενεργοποίηση ή μη του ποτίσματος. Για τον λόγο αυτό, η υλοποίηση δίνει έμφαση σε: (α) σταθερή ανάγνωση του αναλογικού σήματος (ADC), (β) μετατροπή της τιμής σε εύχρηστη κλίμακα (ποσοστό), και (γ) βαθμονόμηση (calibration) ώστε το “ξηρό” και το “υγρό” έδαφος να αντιστοιχούν σε ρεαλιστικά όρια. [26]–[27]



Εικόνα 4-14: Υγρασία εδάφους με σένσορα

4.3.1 Ανάγνωση αναλογικού σήματος (ADC)

Ο αισθητήρας υγρασίας εδάφους δίνει αναλογική έξοδο, η οποία συνδέεται σε ADC pin του ESP32-S3. Σε κάθε κύκλο μέτρησης διαβάζεται η ακατέργαστη τιμή ADC (raw_adc), η οποία εξαρτάται από την αγωγιμότητα/χωρητικότητα του εδάφους και μπορεί να επηρεάζεται από θόρυβο ή μικρές μεταβολές.

Για να περιοριστεί ο θόρυβος, η ανάγνωση δεν βασίζεται σε ένα μόνο δείγμα, αλλά σε πολλαπλά samples (π.χ. N δειγματοληψίες) που συνδυάζονται σε μία πιο σταθερή τιμή (μέσος όρος ή median). Έτσι μειώνονται οι απότομες διακυμάνσεις, και αποφεύγονται λάθος ενεργοποιήσεις ποτίσματος.

```
bool SoilSensor::begin() {
  // Ρύθμιση ADC pin
  pinMode(adcPin, INPUT);

  // ESP32-S3 ADC configuration
  analogReadResolution(ADC_RESOLUTION);
  analogSetAttenuation(ADC_ATTENUATION);

  DEBUG_PRINTLN("✅ SoilSensor: Αρχικοποιήθηκε επιτυχώς");
  DEBUG_PRINTF("   Pin: GPIO %d (ADC)\n", adcPin);
  DEBUG_PRINTF("   Resolution: %d-bit (0-%d)\n", ADC_RESOLUTION, (1 << ADC_RESOLUTION) - 1);
  DEBUG_PRINTF("   Calibration: Dry=%d, Wet=%d\n", adcDry, adcWet);

  // Αρχικό διάβασμα για fill το buffer
  for (int i = 0; i < SOIL_FILTER_SIZE; i++) {
    readRawADC();
    delay(10);
  }

  return true;
}

uint16_t SoilSensor::readRawADC() {
  // Διάβασμα πολλαπλών samples για averaging
  uint32_t sum = 0;

  for (int i = 0; i < ADC_SAMPLES; i++) {
    sum += analogRead(adcPin);
    delayMicroseconds(100); // Μικρή καθυστέρηση μεταξύ samples
  }

  uint16_t avgADC = sum / ADC_SAMPLES;

  // Median filter
  uint16_t filteredADC = medianFilter(avgADC);

  lastRawADC = filteredADC;
  lastReadTime = millis();

  return filteredADC;
}
```

Εικόνα 4-15: Απόσπασμα του soilSensor

4.3.2 Βαθμονόμηση “ξηρού” και “υγρού” εδάφους (dry/wet points)

Η τιμή ADC από μόνη της δεν είναι “καθολικά” ερμηνεύσιμη. Το ίδιο sensor μπορεί να δίνει διαφορετικές τιμές ανάλογα με:

- το είδος του εδάφους (γλάστρα/κήπος, σύσταση),
- το βάθος τοποθέτησης,
- τη θερμοκρασία και τη φθορά του αισθητήρα,
- τη συγκεκριμένη τροφοδοσία/ADC συμπεριφορά.

Για αυτό εφαρμόζεται βαθμονόμηση με δύο βασικά σημεία:

- **Dry point (ADC_DRY):** τιμή ADC που αντιστοιχεί σε “ξηρό” έδαφος.
- **Wet point (ADC_WET):** τιμή ADC που αντιστοιχεί σε “πολύ υγρό” έδαφος.

Η διαδικασία βαθμονόμησης είναι πρακτική:

1. Ο αισθητήρας τοποθετείται σε στεγνό χώμα και καταγράφεται η σταθεροποιημένη τιμή ADC ως ADC_DRY.
2. Ο αισθητήρας τοποθετείται σε πλήρως υγρό χώμα (ή σε νερό, εφόσον αυτό χρησιμοποιείται ως “άνω όριο”) και καταγράφεται η τιμή ADC ως ADC_WET.
3. Οι τιμές αυτές περνούν στο αρχείο ρυθμίσεων (π.χ. config.h) ώστε η μετατροπή σε ποσοστό να αντανακλά το συγκεκριμένο setup.

```
// Βαθμονόμηση ADC → Ποσοστό υγρασίας (0-100%)
// ⚠️ RESISTIVE SENSOR: Υψηλό ADC (≥3000) = ΞΗΡΟ, Χαμηλό ADC (≤700) = ΥΓΡΟ
// Range: ADC 700 (wet) → 3000 (dry) για πρακτική χρήση σε γλάστρα
#define SOIL_ADC_DRY          3000  // ADC value σε πολύ ξηρό έδαφος (0% moisture)
#define SOIL_ADC_WET          700   // ADC value σε πολύ υγρό έδαφος (100% moisture)
```

Εικόνα 4-16: Δήλωση των ADC_DRY και ADC_WET στο αρχείο ρυθμίσεων ή στο module του αισθητήρα.

4.3.3 Μετατροπή ADC σε ποσοστό υγρασίας (%)

Για να χρησιμοποιηθεί η μέτρηση στη λογική απόφασης και να εμφανίζεται στο dashboard/LCD, η τιμή ADC μετατρέπεται σε ποσοστό υγρασίας 0–100%. Η μετατροπή βασίζεται στη γραμμική αντιστοίχιση μεταξύ ADC_DRY και ADC_WET:

- Όταν η τιμή είναι κοντά στο ADC_DRY, η υγρασία τείνει στο **0%** (ξηρό).
- Όταν η τιμή είναι κοντά στο ADC_WET, η υγρασία τείνει στο **100%** (υγρό).

Στην πράξη, η μετατροπή περιλαμβάνει:

- γραμμικό mapping,
- “clamping” ώστε οι τιμές να μένουν στο [0, 100],
- ενδεχομένως μικρή εξομάλυνση (smoothing) για σταθερή εμφάνιση.

Η έξοδος αποθηκεύεται ως soil_moisture_percent και χρησιμοποιείται άμεσα στους κανόνες ποτίσματος (π.χ. “αν < 30% τότε θεωρείται ξηρό”).

4.3.4 Χειρισμός ακραίων τιμών και σταθερότητα μέτρησης

Επειδή οι αναλογικές μετρήσεις μπορεί να έχουν σφάλματα, είναι σημαντικό να υπάρχει βασικός χειρισμός περιπτώσεων όπως:

- **out-of-range τιμές** (π.χ. ADC πολύ κοντά σε 0 ή στο μέγιστο),
- **αστάθεια** λόγω κακής επαφής του αισθητήρα,
- **παροδικές μεταβολές** όταν μόλις έγινε πότισμα.

Στο σύστημα, τέτοιες περιπτώσεις αντιμετωπίζονται με απλές πρακτικές:

- χρήση πολλαπλών δειγμάτων και averaging/median,
- απόρριψη μη έγκυρων τιμών (και χρήση προηγούμενης έγκυρης μέτρησης),
- σταθερή συχνότητα μέτρησης (π.χ. κάθε 2 sec) ώστε να μην αλλάζει η συμπεριφορά.

Με αυτόν τον τρόπο, η ένδειξη υγρασίας παραμένει σταθερή και αξιόπιστη για χρήση στους κανόνες ποτίσματος, μειώνοντας τις πιθανότητες ψευδών ενεργοποιήσεων.

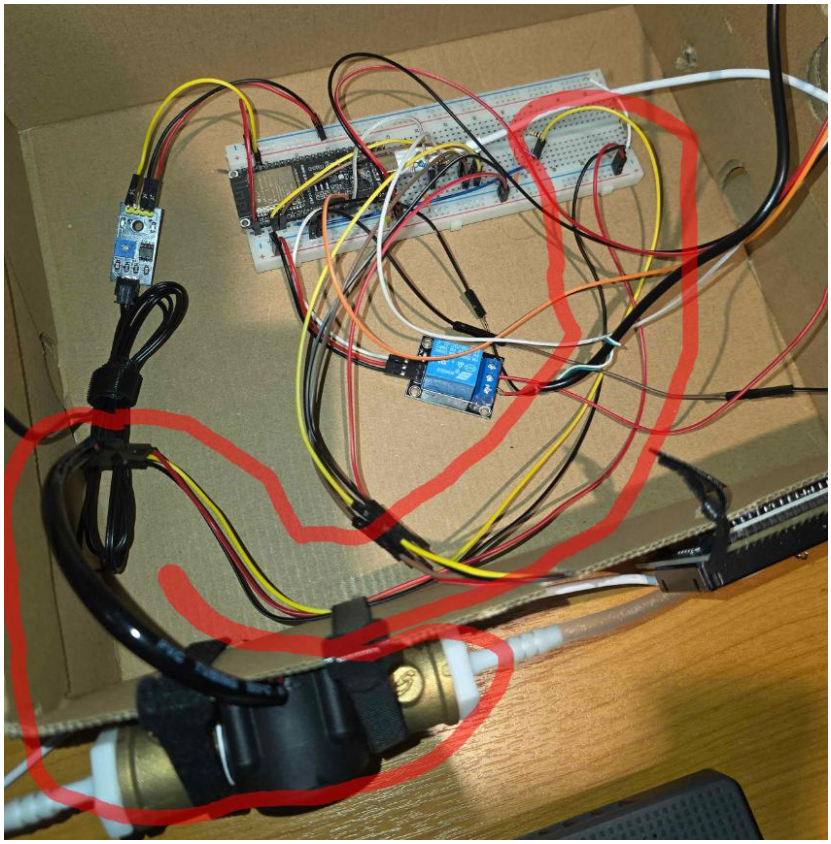
4.4 Μέτρηση κατανάλωσης νερού (Flow Meter) και στατιστικά

Η μέτρηση κατανάλωσης νερού προσθέτει μια σημαντική “ποιοτική” πληροφορία στο σύστημα: δεν καταγράφεται μόνο ότι “έγινε πότισμα”, αλλά και **πόσο νερό** καταναλώθηκε. Αυτό επιτρέπει καλύτερη παρακολούθηση, πιο τεκμηριωμένο ιστορικό στο Grafana και δυνατότητα μελλοντικής βελτίωσης της στρατηγικής ποτίσματος (π.χ. στόχος όγκου αντί για σταθερό χρόνο λειτουργίας). [22]

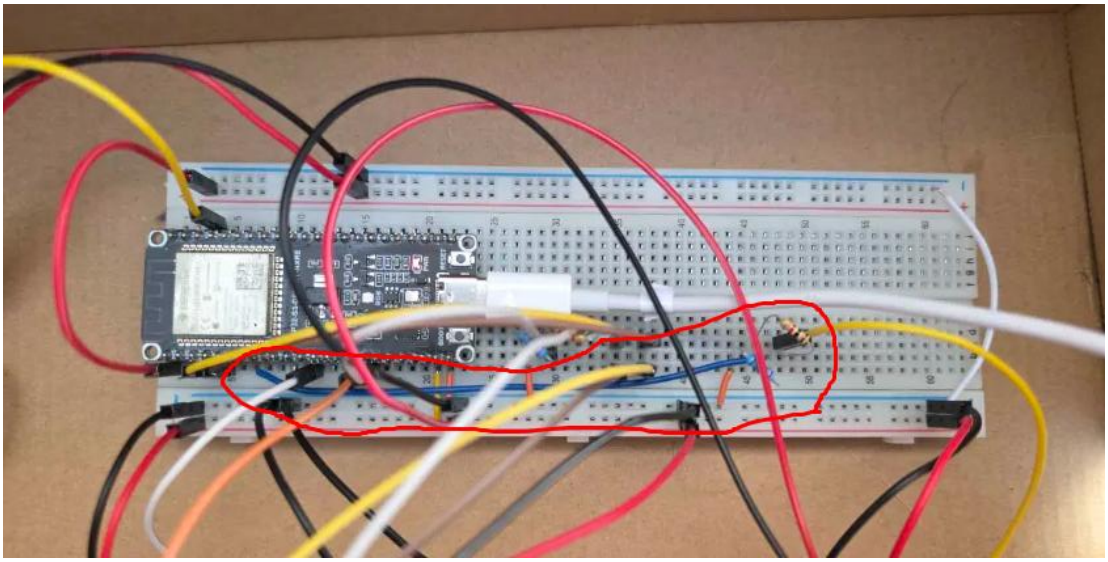
4.4.1 Αρχή λειτουργίας αισθητήρα ροής

Ο αισθητήρας ροής (flow meter) λειτουργεί συνήθως με παλμούς: καθώς περνά νερό, δημιουργούνται ηλεκτρικοί παλμοί στην έξοδο. Ο αριθμός παλμών ανά μονάδα χρόνου σχετίζεται με την παροχή (flow rate), ενώ το άθροισμα των παλμών σε έναν κύκλο σχετίζεται με τον συνολικό όγκο.

Στην υλοποίηση, το flow meter συνδέεται σε ψηφιακό pin του ESP32-S3 και η μέτρηση βασίζεται στην καταμέτρηση παλμών (pulse counting). Για τη σωστή λειτουργία απαιτείται κατάλληλη βαθμονόμηση, δηλαδή ένας συντελεστής που μετατρέπει “pulses → mL”.



Εικόνα 4-17: Διάγραμμα σύνδεσης flow meter. (1)



Εικόνα 4-18: Διάγραμμα σύνδεσης flow meter. (2)

4.4.2 Καταμέτρηση παλμών με interrupts

Για να καταγράφονται με ακρίβεια οι παλμοί, χρησιμοποιείται μηχανισμός διακοπών (interrupts). Κάθε παλμός προκαλεί κλήση ενός interrupt handler, ο οποίος αυξάνει έναν μετρητή (counter). Η χρήση interrupts είναι προτιμότερη από “polling” (συνεχές διάβασμα pin), γιατί:

- μειώνει την πιθανότητα να χαθούν παλμοί,
- λειτουργεί αξιόπιστα ακόμη και όταν το loop() εκτελεί άλλες εργασίες,
- είναι πιο αποδοτική σε υπολογιστικό κόστος.

Κατά τη διάρκεια ποτίσματος, ο counter αυξάνεται και στο τέλος του κύκλου ο αριθμός παλμών μετατρέπεται σε mL, ώστε να υπολογιστεί το πόσο νερό δόθηκε στον συγκεκριμένο κύκλο.

4.4.3 Μετατροπή παλμών σε όγκο (pulses → mL) και βαθμονόμηση

Ο υπολογισμός του όγκου βασίζεται σε έναν συντελεστή βαθμονόμησης, ο οποίος εκφράζει πόσοι παλμοί αντιστοιχούν σε συγκεκριμένη ποσότητα νερού. Η βαθμονόμηση μπορεί να γίνει εμπειρικά: εκτελείται πότισμα γνωστής διάρκειας ή συλλέγεται νερό σε δοχείο, και συγκρίνεται ο πραγματικός όγκος με τους καταμετρημένους παλμούς. Από αυτό προκύπτει ένας παράγοντας μετατροπής.

Στη λογική του συστήματος, ο όγκος που προκύπτει:

- αποθηκεύεται ως **last_watering_ml** (όγκος τελευταίου ποτίσματος),
- προστίθεται σε έναν συσσωρευτικό μετρητή **grand_total_ml** (συνολική κατανάλωση).

Με τον τρόπο αυτό, το ιστορικό μπορεί να δείξει τόσο “ανά κύκλο” όσο και “συνολικά”.

4.4.4 Στατιστικά ποτισμάτων και κατηγοριοποίηση (auto/manual)

Πέρα από την κατανάλωση, το σύστημα διατηρεί μετρητές (counters) για στατιστική παρακολούθηση της λειτουργίας, όπως:

- **total_count**: συνολικός αριθμός ποτισμάτων,
- **auto_count**: ποτίσματα που εκτελέστηκαν αυτόματα από τη λογική κανόνων,
- **manual_count**: ποτίσματα που ξεκίνησαν χειροκίνητα από τον χρήστη.

Η κατηγοριοποίηση επιτρέπει να φαίνεται πόσο “δουλεύει” το σύστημα αυτόνομα και πόσο συχνά χρειάζεται παρέμβαση χρήστη. Επιπλέον, οι μετρητές αυτοί μπορούν να παρουσιαστούν σε dashboards (Grafana) ή στο τοπικό web UI ως συνοπτικό στατιστικό.

```
41  bool FlowMeter::begin() {
42    // Ρύθμιση pin ως input με pull-down για να αποφύγουμε θόρυβο
43    pinMode(flowPin, INPUT_PULLDOWN);
44
45    DEBUG_PRINTLN("💦 FlowMeter: Initializing...");
46    DEBUG_PRINTF("   Pin: GPIO %d\n", flowPin);
47    DEBUG_PRINTF("   Calibration: %.3f pulses/mL\n", pulsesPerML);
48
49    // DON'T attach interrupt here - θα το ενεργοποιούμε μόνο όταν τρέχει η αντλία
50    // attachInterrupt(digitalPinToInterrupt(flowPin), pulseISR, RISING);
51
52    // Reset counters
53    reset();
54
55    DEBUG_PRINTLN("✅ FlowMeter: Αρχικοποιήθηκε επιτυχώς (interrupt disabled)");
56
57
58    return true;
59  }
```

Εικόνα 4-19: Απόσπασμα του FlowMeter. (1)

```
61  void FlowMeter::update() {
62    unsigned long now = millis();
63    unsigned long elapsed = now - lastUpdateTime;
64
65    // Update κάθε 1 δευτερόλεπτο
66    if (elapsed >= 1000) {
67      // Υπολογισμός νέων pulses από την τελευταία update
68      unsigned long newPulses = pulseCount - lastPulseCount;
69
70      // Μετατροπή pulses → mL
71      float newML = (float)newPulses / pulsesPerML;
72      totalML += newML;
73
74      // Υπολογισμός flow rate (ml/s)
75      float elapsedSeconds = elapsed / 1000.0;
76      currentFlowRate = newML / elapsedSeconds;
77
78      // Debug output (μόνο αν υπάρχει ροή)
79      if (newPulses > 0) {
80        DEBUG_PRINTF("💦 Flow: +%.1f mL (%.1f ml/s) | Total: %.1f mL\n",
81                     newML, currentFlowRate, totalML);
82      }
83
84      // Update για επόμενη φορά
85      lastPulseCount = pulseCount;
86      lastUpdateTime = now;
87    }
88  }
89
90  void FlowMeter::reset() {
91    pulseCount = 0;
92    lastPulseCount = 0;
93    totalML = 0.0;
94    currentFlowRate = 0.0;
95    lastUpdateTime = millis();
96
97    DEBUG_PRINTLN("💦 FlowMeter: Reset counters");
98  }
99
100 void FlowMeter::enable() {
101   pulseCount = 0;
102   lastPulseCount = 0;
103   lastPulseTime = micros();  // Reset debounce timer
104   attachInterrupt(digitalPinToInterrupt(flowPin), pulseISR, RISING);
105   DEBUG_PRINTLN("💦 FlowMeter: Interrupt enabled");
106 }
```

Εικόνα 4-20: Απόσπασμα του FlowMeter. (2)

```
108  void FlowMeter::disable() {
109    detachInterrupt(digitalPinToInterrupt(flowPin));
110    // Περίμενε λίγο για να καταλαγιάσει ο θόρυβος από την αντλία
111    delay(100);
112    DEBUG_PRINTLN("💦 FlowMeter: Interrupt disabled");
113  }
114
115  void FlowMeter::finishWatering(bool isManual) {
116    // Αποθηκεύει το τρέχον πότισμα ως τελευταίο
117    if (totalML > 0) {
118      lastWateringML = totalML;
119      grandTotalML += totalML;
120      totalWateringCount++;
121
122      if (isManual) {
123        manualWateringCount++;
124        DEBUG_PRINTF("💦 Manual watering finished: %.1f mL | Total: %.1f mL | Count: %u (A:%u M:%u)\n",
125                     lastWateringML, grandTotalML, totalWateringCount, autoWateringCount, manualWateringCount);
126      } else {
127        autoWateringCount++;
128        DEBUG_PRINTF("💦 Auto watering finished: %.1f mL | Total: %.1f mL | Count: %u (A:%u M:%u)\n",
129                     lastWateringML, grandTotalML, totalWateringCount, autoWateringCount, manualWateringCount);
130      }
131    }
132  }
133
134  void FlowMeter::setCalibration(float pulses_per_ml) {
135    pulsesPerML = pulses_per_ml;
136
137    DEBUG_PRINTLN("✅ FlowMeter: Calibration updated");
138    DEBUG_PRINTF("   Pulses/mL: %.3f\n", pulses_per_ml);
139  }
140
141  // ═══════════════════════════════════════════════════════════
142  // ISR (Interrupt Service Routine)
143  // ═══════════════════════════════════════════════════════════
144
145  void IRAM_ATTR FlowMeter::pulseISR() {
146    // Debounce: minimum 3ms μεταξύ pulses (max 333 Hz - αρκετό για YF-S201 max ~450 Hz at highest flow)
147    unsigned long now = micros();
148    if (now - lastPulseTime > 3000) {  // 3000 microseconds = 3ms
149      pulseCount++;
150      lastPulseTime = now;
151    }
152  }
153
```

Εικόνα 4-21: Απόσπασμα του FlowMeter. (3)

4.4.5 Ενσωμάτωση στη διεπαφή (dashboard/MQTT) και αποθήκευση ιστορικού

Τα δεδομένα ροής και κατανάλωσης ενσωματώνονται:

- στο **GET /data** JSON του ESP32, ώστε το web dashboard να εμφανίζει συνολική κατανάλωση και τελευταίο πότισμα,
- σε **MQTT topics**, ώστε το backend να αποθηκεύει και να οπτικοποιεί ιστορικά τις τιμές,
- στο **InfluxDB**, ώστε να φαίνονται με γραφήματα (π.χ. κατανάλωση ανά ημέρα/ανά κύκλο ή συνολικός μετρητής).

Αυτό δίνει στο σύστημα ένα σημαντικό πλεονέκτημα: οι αποφάσεις ποτίσματος μπορούν να τεκμηριωθούν ποσοτικά, καθώς το “πόσο νερό δόθηκε” είναι μετρήσιμο και όχι υποθετικό.

4.5 Εκτίμηση στάθμης δοχείου (Ultrasonic) και κατηγοριοποίηση κατάστασης

Η παρακολούθηση της στάθμης νερού στο δοχείο αποτελεί κρίσιμη λειτουργία για την ασφαλή λειτουργία του συστήματος, καθώς αποτρέπει την ενεργοποίηση της αντλίας όταν δεν υπάρχει επαρκές νερό (π.χ. αποφυγή λειτουργίας “στεγνής” αντλίας). Για τον σκοπό αυτό χρησιμοποιείται υπερηχητικός αισθητήρας απόστασης, ο οποίος μετρά την απόσταση από το σημείο τοποθέτησης μέχρι την επιφάνεια του νερού και από αυτήν εξάγεται η στάθμη/ποσοστό πλήρωσης.

Εικόνα 4-22: Ultrasonic Sensor

4.5.1 Αρχή λειτουργίας υπερηχητικού αισθητήρα

Ο υπερηχητικός αισθητήρας λειτουργεί με εκπομπή ενός παλμού (trigger) και μέτρηση του χρόνου επιστροφής του ηχητικού κύματος (echo). Ο χρόνος αυτός μετατρέπεται σε απόσταση με βάση την ταχύτητα του ήχου. Στην πράξη:

- αποστέλλεται ένα σύντομο σήμα στο pin **TRIG**,
- λαμβάνεται σήμα στο pin **ECHO** με διάρκεια ανάλογη της απόστασης,
- ο χρόνος μετατρέπεται σε **distance_cm**.

Η απόσταση αποτελεί την “πρωτογενή” μέτρηση που στη συνέχεια μεταφράζεται σε στάθμη δοχείου.

4.5.2 Υπολογισμός απόστασης (trigger/echo) και σταθεροποίηση μέτρησης

Η μέτρηση υπερηχητικού μπορεί να επηρεαστεί από:

- μικρές αναταράξεις στην επιφάνεια του νερού,
- θόρυβο/ανακλάσεις στο δοχείο,
- μη ιδανική γωνία τοποθέτησης.

Για αυτό, η υλοποίηση συνήθως εφαρμόζει πρακτικές σταθεροποίησης όπως:

- πολλαπλές μετρήσεις σε μικρό χρονικό διάστημα και λήψη μέσου όρου/median,
- απόρριψη ακραίων τιμών (outliers),
- χρονικό διάστημα ανάμεσα σε μετρήσεις (π.χ. κάθε 2 sec μαζί με τα υπόλοιπα sensors).

Η τελική “καθαρή” τιμή αποθηκεύεται ως distance_cm και χρησιμοποιείται για μετατροπή σε επίπεδο/ποσοστό.

4.5.3 Μετατροπή απόστασης σε ποσοστό στάθμης (level_percent)

Η απόσταση που επιστρέφει ο αισθητήρας μετατρέπεται σε “ποσοστό πλήρωσης” με βάση τη γεωμετρία του δοχείου και τη θέση τοποθέτησης του αισθητήρα. Για τη μετατροπή ορίζονται δύο σημεία αναφοράς:

- **MIN_DISTANCE (full tank):** η ελάχιστη απόσταση όταν το δοχείο είναι “γεμάτο” (η επιφάνεια του νερού είναι κοντά στον αισθητήρα).
- **MAX_DISTANCE (empty tank):** η μέγιστη απόσταση όταν το δοχείο είναι “άδειο” (η επιφάνεια είναι χαμηλά ή δεν υπάρχει νερό).

Με βάση αυτά, η μετατροπή γίνεται γραμμικά:

- απόσταση κοντά στο MIN → **100%**
- απόσταση κοντά στο MAX → **0%**

Στην υλοποίηση εφαρμόζεται επίσης clamping στο [0, 100], ώστε να αποφεύγονται μη ρεαλιστικές τιμές.

4.5.4 Κατηγοριοποίηση κατάστασης δοχείου (OK/LOW/EMPTY)

Για να χρησιμοποιηθεί η στάθμη στην απόφαση ποτίσματος και να παρουσιαστεί εύκολα στον χρήστη, το σύστημα δεν κρατά μόνο ποσοστό, αλλά και μια “κατάσταση” (status). Ενδεικτικά:

- **OK:** στάθμη πάνω από ένα κατώφλι (π.χ. > 20%)
- **LOW:** στάθμη χαμηλή, προειδοποίηση (π.χ. 5%–20%)
- **EMPTY:** στάθμη πολύ χαμηλή/μηδενική, αποτροπή ποτίσματος (π.χ. < 5%)

Η κατάσταση αυτή εμφανίζεται στην LCD και στο web dashboard (π.χ. “LOW TANK”), ενώ μπορεί να χρησιμοποιηθεί και ως κανόνας ασφαλείας: όταν το δοχείο είναι EMPTY, το σύστημα δεν επιτρέπει ενεργοποίηση αντλίας, ακόμη και αν το έδαφος είναι ξηρό.

```cpp
bool UltrasonicSensor::begin() {
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);

    digitalWrite(trigPin, LOW);

    DEBUG_PRINTLN("🔪 UltrasonicSensor: Initializing...");
    DEBUG_PRINTF("   TRIG Pin: GPIO %d\n", trigPin);
    DEBUG_PRINTF("   ECHO Pin: GPIO %d\n", echoPin);
    DEBUG_PRINTF("   Empty distance: %.1f cm\n", emptyDistance);
    DEBUG_PRINTF("   Full distance: %.1f cm\n", fullDistance);

    // Μία αρχική μέτρηση
    readDistance();

    DEBUG_PRINTLN("✅ UltrasonicSensor: Αρχικοποιήθηκε επιτυχώς");
    return true;
}

float UltrasonicSensor::readDistance() {
    float distance = measureDistance();
    float filtered = medianFilter(distance);

    lastDistance = filtered;
    return filtered;
}

float UltrasonicSensor::readLevel() {
    readDistance();
    lastLevel = distanceToPercent(lastDistance);

    DEBUG_PRINTF("🔪 Tank: %.1f cm → %.1f%%\n", lastDistance, lastLevel);

    return lastLevel;
}

void UltrasonicSensor::setCalibration(float empty, float full) {
    emptyDistance = empty;
    fullDistance = full;

    DEBUG_PRINTLN("✅ UltrasonicSensor: Calibration updated");
    DEBUG_PRINTF("   Empty: %.1f cm\n", empty);
    DEBUG_PRINTF("   Full: %.1f cm\n", full);
}
```

Εικόνα 4-23: Απόσπασμα του UltrasonicSensor. (1)

```cpp
float UltrasonicSensor::measureDistance() {
    // Trigger pulse
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    // Measure echo pulse duration
    // Timeout μετά από 6ms (~100cm max - αρκετό για μικρό δοχείο)
    unsigned long duration = pulseIn(echoPin, HIGH, 6000);

    if (duration == 0) {
        // Timeout - επιστροφή μεγάλης τιμής
        return 999.0;
    }

    // Υπολογισμός απόστασης
    // Ταχύτητα ήχου: 343 m/s = 0.0343 cm/μs
    // Διαίρεση με 2 γιατί ο ήχος πάει και γυρίζει
    float distance = (duration * 0.0343) / 2.0;

    return distance;
}

float UltrasonicSensor::medianFilter(float value) {
    // Χωρίς filter - απευθείας τιμή για instant response
    // Αν υπάρχει timeout (999), κράτα προηγούμενη τιμή
    if (value > 500) {
        return filterBuffer[0];  // Κράτα παλιά αν timeout
    }

    filterBuffer[0] = value;
    filterFilled = true;
    return value;
}
```

Εικόνα 4-24: Απόσπασμα του UltrasonicSensor. (2)

```
118  float UltrasonicSensor::distanceToPercent(float distance) {
119      // Μικρότερη απόσταση = περισσότερο νερό = υψηλότερο %
120      // Μεγαλύτερη απόσταση = λιγότερο νερό = χαμηλότερο %
121
122      if (distance <= fullDistance) {
123          return 100.0;  // Γεμάτο
124      }
125      if (distance >= emptyDistance) {
126          return 0.0;    // Άδειο
127      }
128
129      // Linear interpolation
130      float percent = 100.0 * (emptyDistance - distance) / (emptyDistance - fullDistance);
131
132      return constrain(percent, 0.0, 100.0);
133  }
134
```

Εικόνα 4-25: Απόσπασμα του UltrasonicSensor. (3)

4.5.5 Ενσωμάτωση σε UI και ιστορικό

Η μέτρηση στάθμης και η κατάσταση δοχείου:

- εμφανίζονται στο GET /data ώστε το dashboard να παρουσιάζει ποσοστό και status,
- δημοσιεύονται μέσω MQTT ώστε να αποθηκεύονται στο InfluxDB και να φαίνεται ιστορικό (π.χ. πότε το δοχείο άδειασε),
- προβάλλονται σε Grafana panel ως time-series (level_percent) και ως συμβάν/σήμανση (LOW/EMPTY).

Έτσι, το σύστημα παρέχει τόσο “real-time” ενημέρωση όσο και ιστορική εικόνα για το πότε απαιτήθηκε αναπλήρωση νερού.

4.6 Έλεγχος αντλίας/βαλβίδας (Relay Controller) και μηχανισμοί ασφαλείας

Ο έλεγχος της αντλίας αποτελεί τη βασική “ενέργεια” του συστήματος και υλοποιείται μέσω ρελέ (relay module) που ενεργοποιείται από ψηφιακή έξοδο του ESP32-S3. Επειδή η αντλία είναι συσκευή ισχύος και η λανθασμένη λειτουργία μπορεί να προκαλέσει υπερ-πότισμα ή/και βλάβη (π.χ. λειτουργία χωρίς νερό), η υλοποίηση περιλαμβάνει σαφείς μηχανισμούς ασφάλειας και κανόνες προστασίας.

4.6.1 Βασικός μηχανισμός on/off μέσω ρελέ

Το ρελέ λειτουργεί ως “διακόπτης” μεταξύ του ESP32 και του κυκλώματος τροφοδοσίας της αντλίας. Το ESP32 δίνει σήμα HIGH/LOW στο pin ελέγχου του ρελέ και έτσι:

- **Relay ON → αντλία ενεργή (WATERING)**
- **Relay OFF → αντλία ανενεργή (IDLE)**

Η ενεργοποίηση γίνεται είτε από την αυτόματη λογική (rule-based control) είτε χειροκίνητα (web/MQTT command). Σε κάθε περίπτωση, το σύστημα ενημερώνει την κατάσταση (state) και καταγράφει το γεγονός (event).

4.6.2 Δόση ποτίσματος και τρόπος τερματισμού

Στο πρωτότυπο, το πότισμα υλοποιείται ως “δοσολογία” (dose-based watering), δηλαδή ένα πότισμα αντιστοιχεί σε συγκεκριμένη διάρκεια λειτουργίας αντλίας (π.χ. 10 sec) με στόχο έναν περίπου σταθερό όγκο (π.χ. ~100 mL). Με αυτόν τον τρόπο:

- το σύστημα είναι προβλέψιμο,
- οι κύκλοι ποτίσματος είναι μικροί,
- αποφεύγεται μεγάλης διάρκειας ανεξέλεγκτη λειτουργία.

Ο τερματισμός ποτίσματος μπορεί να συμβεί:

- όταν ολοκληρωθεί η προκαθορισμένη διάρκεια/δόση,
- όταν ενεργοποιηθεί προστασία (π.χ. max runtime),
- όταν υπάρχει ανίχνευση κατάστασης που δεν επιτρέπει συνέχιση (π.χ. πολύ χαμηλή στάθμη δοχείου).

4.6.3 Μηχανισμοί ασφάλειας (safety guards)

Για να αποφευχθεί ανεπιθύμητη ή επικίνδυνη λειτουργία, εφαρμόζονται οι παρακάτω μηχανισμοί:

(α) Μέγιστος χρόνος λειτουργίας αντλίας (max runtime)
Ανεξάρτητα από το τι ζητά ο χρήστης ή η αυτόματη λογική, υπάρχει ανώτατο όριο λειτουργίας (π.χ. 10 sec). Αν ξεπεραστεί, η αντλία τερματίζεται άμεσα και το σύστημα σημειώνει σφάλμα/προστασία. Αυτό προστατεύει από:

- κολλημένο state,
- λανθασμένη εντολή,
- απρόβλεπτη συμπεριφορά σε σφάλμα αισθητήρα/κώδικα.

(β) Cooldown μεταξύ ποτισμάτων
Μετά από κάθε πότισμα εφαρμόζεται cooldown (π.χ. 10 sec), ώστε να αποφεύγεται συνεχής ενεργοποίηση σε περιπτώσεις ασταθούς μέτρησης ή μεταβατικής κατάστασης εδάφους.

(γ) Debounce/σταθερότητα εντολών ρελέ
Μικρή καθυστέρηση/προστασία μεταξύ διαδοχικών αλλαγών κατάστασης του ρελέ (π.χ. debounce 50 ms) αποτρέπει “τρεμόπαιγμα” (relay chatter).

(δ) Έλεγχος προϋποθέσεων πριν το πότισμα (pre-checks)
Πριν ενεργοποιηθεί η αντλία, γίνεται έλεγχος κρίσιμων συνθηκών, όπως:

- κατάλληλη κατάσταση δοχείου (όχι EMPTY),
- μη ύπαρξη ενεργού emergency stop,
- μη ενεργός cooldown,
- ύπαρξη έγκυρων μετρήσεων (όπου απαιτείται).

(ε) Watchdog timer
Για αυξημένη ανθεκτικότητα, ο watchdog επιτρέπει στο σύστημα να επανεκκινήσει αν “κολλήσει” ή αν δεν εκτελεί σωστά τον κύκλο λειτουργίας, μειώνοντας το ρίσκο παρατεταμένης μη ελεγχόμενης κατάστασης.

4.6.4 Emergency stop και “κλείδωμα” λειτουργίας

Το emergency stop αποτελεί κρίσιμη λειτουργία ασφαλείας και μπορεί να ενεργοποιηθεί:

- από MQTT command (.../command),
- από εσωτερική προστασία (π.χ. υπέρβαση max runtime).

Με την ενεργοποίηση:

1. Το ρελέ απενεργοποιείται άμεσα (αντλία OFF).
2. Το σύστημα ενημερώνει την κατάσταση (π.χ. ERROR ή STOPPED).
3. Καταγράφεται event (γιατί έγινε stop, πηγή εντολής).
4. Εφόσον έχει εφαρμοστεί “lock”, απαιτείται συγκεκριμένη ενέργεια για επαναφορά (π.χ. reset flag ή επανεκκίνηση), ώστε να μην ξεκινήσει ξανά αυτόματα χωρίς επίβλεψη.

```cpp
bool RelayController::begin() {
  // Ρύθμιση pin ως έξοδος
  pinMode(relayPin, OUTPUT);

  // Αρχικά OFF (ασφάλεια!)
  RELAY_OFF();
  currentState = RelayState::OFF;
  lastStateChange = millis();
  lastActionTime = millis();

  DEBUG_PRINTLN("✅ RelayController: Αρχικοποιήθηκε επιτυχώς");
  DEBUG_PRINTF("   Pin: GPIO %d\n", relayPin);
  DEBUG_PRINTF("   Trigger: %s\n", RELAY_TRIGGER_LOW ? "LOW (active low)" : "HIGH (active high)");

  return true;
}
```

Εικόνα 4-26: Απόσπασμα του RelayController. (1)

```cpp
bool RelayController::turnOn() {
  // ═══════════════════════════════════════════════════════
  // ΕΛΕΓΧΟΙ ΑΣΦΑΛΕΙΑΣ
  // ═══════════════════════════════════════════════════════

  // Έλεγχος 1: Locked;
  if (isLocked) {
    DEBUG_PRINTLN("⛔ RelayController: LOCKED - Δεν μπορεί να ανοίξει!");
    DEBUG_PRINTF("   Λόγος: %s\n", lockReason.c_str());
    return false;
  }

  // Έλεγχος 2: Ήδη ON;
  if (currentState == RelayState::ON) {
    DEBUG_PRINTLN("⚠️  RelayController: Ήδη ON");
    return true; // Όχι error, απλά ήδη ON
  }

  // Έλεγχος 3: Debouncing
  unsigned long now = millis();
  if (now - lastActionTime < RELAY_DEBOUNCE_MS) {
    DEBUG_PRINTLN("⚠️  RelayController: Debounce - πολύ σύντομα!");
    return false;
  }

  // Έλεγχος 4: Ελάχιστος χρόνος OFF
  if (now - lastStateChange < RELAY_MIN_OFF_TIME_MS) {
    unsigned long remaining = RELAY_MIN_OFF_TIME_MS - (now - lastStateChange);
    DEBUG_PRINTF("⚠️  RelayController: Πολύ σύντομα! Περίμενε %lu ms\n", remaining);
    return false;
  }

  // ═══════════════════════════════════════════════════════
  // ΑΝΟΙΓΜΑ RELAY
  // ═══════════════════════════════════════════════════════

  RELAY_ON();
  currentState = RelayState::ON;
  onStartTime = now;
  lastStateChange = now;
  lastActionTime = now;
  totalCycles++;

  DEBUG_PRINTLN("✅ RelayController: Relay ON (Αντλία ξεκίνησε)");
  DEBUG_PRINTF("   Cycle #%u\n", totalCycles);

  return true;
}
```

Εικόνα 4-27: Απόσπασμα του RelayController. (2)

```
bool RelayController::turnOff() {
  // Αν ήδη OFF, επιστροφή
  if (currentState == RelayState::OFF) {
    return true;
  }

  // Κλείσιμο relay
  RELAY_OFF();

  // Υπολογισμός χρόνου που ήταν ON
  if (currentState == RelayState::ON) {
    unsigned long duration = millis() - onStartTime;
    totalOnTime += duration;

    DEBUG_PRINTLN("✅ RelayController: Relay OFF (Αντλία σταμάτησε)");
    DEBUG_PRINTF("   Διάρκεια: %.2f δευτερόλεπτα\n", duration / 1000.0);
    DEBUG_PRINTF("   Συνολικός χρόνος: %.2f λεπτά\n", totalOnTime / 60000.0);
  }

  currentState = RelayState::OFF;
  lastStateChange = millis();
  lastActionTime = millis();

  return true;
}

void RelayController::forceOn() {
  // Παρακάμπτει όλα τα safety checks - για manual watering μόνο!
  if (isLocked) {
    DEBUG_PRINTLN("⛔ RelayController: LOCKED - unlock first!");
    return;
  }

  RELAY_ON();
  currentState = RelayState::ON;
  onStartTime = millis();
  lastStateChange = millis();
  lastActionTime = millis();
  totalCycles++;

  Serial.println("⚡ RelayController: FORCE ON!");
}
```

Εικόνα 4-28: Απόσπασμα του RelayController. (3)

```
void RelayController::forceOff() {
  // Άμεσο κλείσιμο χωρίς checks
  RELAY_OFF();

  if (currentState == RelayState::ON) {
    unsigned long duration = millis() - onStartTime;
    totalOnTime += duration;
  }

  currentState = RelayState::OFF;
  lastStateChange = millis();
  lastActionTime = millis();

  Serial.println("⚡ RelayController: FORCE OFF!");
}

void RelayController::emergencyStop(const String& reason) {
  DEBUG_PRINTLN("🚨 EMERGENCY STOP!");
  DEBUG_PRINTF("   Λόγος: %s\n", reason.c_str());

  // Άμεσο κλείσιμο
  RELAY_OFF();

  // Lock
  isLocked = true;
  lockReason = reason;
  currentState = RelayState::LOCKED;
  lastStateChange = millis();

  DEBUG_PRINTLN("   Relay LOCKED - Χρειάζεται unlock() για ξεκλείδωμα");
}
```

Εικόνα 4-29: Απόσπασμα του RelayController. (4)

```
171  bool RelayController::unlock() {
172    if (!isLocked) {
173      DEBUG_PRINTLN("⚠️  RelayController: Δεν είναι locked");
174      return true;
175    }
176
177    isLocked = false;
178    lockReason = "";
179    currentState = RelayState::OFF;
180
181    DEBUG_PRINTLN("🔓 RelayController: Ξεκλειδώθηκε");
182
183    return true;
184  }
185
186  void RelayController::update() {
187    // Έλεγχος μόνο αν είναι ON
188    if (currentState != RelayState::ON) {
189      return;
190    }
191
192    // Έλεγχος MAX_RUNTIME_MS
193    unsigned long duration = millis() - onStartTime;
194
195    if (duration > MAX_RUNTIME_MS) {
196      DEBUG_PRINTLN("⚠️  RelayController: MAX RUNTIME EXCEEDED!");
197      DEBUG_PRINTF("   Διάρκεια: %lu ms (max: %lu ms)\n", duration, MAX_RUNTIME_MS);
198
199      // Emergency stop
200      emergencyStop("Max runtime exceeded (" + String(MAX_RUNTIME_MS / 1000) + "s)");
201    }
202  }
203
204  // ═══════════════════════════════════════════════════════════
205  // GETTERS
206  // ═══════════════════════════════════════════════════════════
207
208  unsigned long RelayController::getOnDuration() const {
209    if (currentState == RelayState::ON) {
210      return millis() - onStartTime;
211    }
212    return 0;
213  }
```

Εικόνα 4-30: Απόσπασμα του RelayController. (5)

```
215  bool RelayController::canTurnOn() const {
216    if (isLocked) return false;
217    if (currentState == RelayState::ON) return false;
218
219    unsigned long now = millis();
220
221    // Debounce check
222    if (now - lastActionTime < RELAY_DEBOUNCE_MS) {
223      return false;
224    }
225
226    // Min off time check
227    if (now - lastStateChange < RELAY_MIN_OFF_TIME_MS) {
228      return false;
229    }
230
231    return true;
232  }
233
234  unsigned long RelayController::getMinOffTimeRemaining() const {
235    if (currentState == RelayState::ON) {
236      return 0; // Ήδη ON
237    }
238
239    unsigned long now = millis();
240    unsigned long elapsed = now - lastStateChange;
241
242    if (elapsed >= RELAY_MIN_OFF_TIME_MS) {
243      return 0; // Μπορεί να ανοίξει
244    }
245
246    return RELAY_MIN_OFF_TIME_MS - elapsed;
247  }
248
```

Εικόνα 4-31: Απόσπασμα του RelayController. (6)

4.6.5 Καταγραφή ενεργειών ποτίσματος (events) και ενημέρωση διεπαφών

Κάθε κύκλος ποτίσματος συνοδεύεται από ενημέρωση των διεπαφών:

- Στο GET /data ενημερώνονται state/is_on, counters και όγκος τελευταίου ποτίσματος.
- Στην LCD εμφανίζεται άμεσα η αλλαγή κατάστασης (WATERING, STOP, LOW TANK).
- Σε MQTT δημοσιεύονται state transitions και σχετικές μετρήσεις (π.χ. start/stop, total_count, last_watering_ml).
- Στο backend αποθηκεύονται τα δεδομένα στην InfluxDB και απεικονίζονται στο Grafana.

Έτσι, η λειτουργία του ενεργοποιητή παραμένει διαφανής και “ιχνηλατήσιμη”, κάτι που είναι σημαντικό για την παρουσίαση ενός ολοκληρωμένου IoT συστήματος.

4.7 Ενσωμάτωση μετεωρολογικών δεδομένων (OpenWeatherMap)

Το σύστημα αξιοποιεί μετεωρολογικά δεδομένα από το **OpenWeatherMap Forecast API** (πρόβλεψη ανά 3 ώρες) ώστε να βελτιώνει τη λήψη απόφασης ποτίσματος και να αποφεύγει άσκοπες ενεργοποιήσεις όταν αναμένεται βροχή. Στην τελική υλοποίηση, οι τιμές περιβάλλοντος (θερμοκρασία/σχετική υγρασία αέρα) εμφανίζονται ως “environment” στο dashboard, αλλά **προέρχονται από το API** και όχι από τοπικό αισθητήρα. Σε παλαιότερη εκδοχή του πρωτοτύπου είχε εξεταστεί/χρησιμοποιηθεί αισθητήρας περιβάλλοντος, όμως αφαιρέθηκε καθώς τα απαιτούμενα δεδομένα καλύπτονται από την υπηρεσία καιρού. [19]

4.7.1 Λήψη πρόβλεψης και επεξεργασία (parsing)

Ανά τακτά χρονικά διαστήματα (π.χ. ανά ώρα), το ESP32 στέλνει αίτημα HTTP προς το OpenWeatherMap και λαμβάνει τη λίστα πρόβλεψης (3ωρα βήματα). Από την απόκριση εξάγονται τα πεδία που ενδιαφέρουν το σύστημα, όπως:

- προβλεπόμενη βροχόπτωση (όπου παρέχεται),
- θερμοκρασία αέρα,
- σχετική υγρασία αέρα,
  και αποθηκεύονται σε “cache” μαζί με χρονική σήμανση τελευταίας ενημέρωσης, ώστε να χρησιμοποιούνται τόσο στην απόφαση ποτίσματος όσο και στην απεικόνιση στο dashboard/LCD.

4.7.2 Δείκτες βροχής και σύνδεση με τον κανόνα ποτίσματος

Για να χρησιμοποιηθεί πρακτικά η πρόβλεψη, το σύστημα υπολογίζει συνοπτικούς δείκτες από τα 3ωρα βήματα, όπως:

- **βροχή 12 ωρών (rain_12h):** άθροισμα προβλεπόμενης βροχόπτωσης στο επόμενο 12ωρο,
- **βροχή 24 ωρών (rain_24h):** αντίστοιχα στο επόμενο 24ωρο,
- **will_rain:** λογική ένδειξη (true/false) όταν η προβλεπόμενη βροχή ξεπερνά ένα όριο (π.χ. > 2 mm στο 12ωρο).

Οι δείκτες αυτοί ενσωματώνονται στον κανόνα απόφασης (βλ. Κεφάλαιο 2.4): όταν το έδαφος είναι “ξηρό” (κάτω από το όριο υγρασίας), το σύστημα επιτρέπει πότισμα **μόνο αν** δεν αναμένεται βροχή πάνω από το προκαθορισμένο κατώφλι. Αν αναμένεται βροχή, το πότισμα παραλείπεται και καταγράφεται σχετικό συμβάν (π.χ. “skip λόγω πρόβλεψης βροχής”), ώστε να είναι εμφανές στο ιστορικό.

Σε περίπτωση αποτυχίας λήψης/επεξεργασίας καιρού (π.χ. προσωρινή απώλεια σύνδεσης), εφαρμόζεται ασφαλής συμπεριφορά: το σύστημα δεν “μπλοκάρει” τη λειτουργία του, αλλά συνεχίζει να βασίζεται κυρίως στη μέτρηση υγρασίας εδάφους και στους μηχανισμούς ασφάλειας (cooldown, max runtime, emergency stop) μέχρι να αποκατασταθεί η ενημέρωση καιρού.

```cpp
bool WeatherAPI::willRainSoon(int hoursAhead, float thresholdMM) {
  if (!cachedData.isValid) {
    DEBUG_PRINTLN("⚠️  WeatherAPI: No valid data, assuming no rain");
    return false;
  }

  // Απλή λογική βάσει 12h forecast (για μικρές δόσεις ~100ml)
  bool rainExpected = false;

  if (cachedData.rain3h > 1.0) {
    rainExpected = true;
    DEBUG_PRINTF("🌧️ Rain forecast (3h): %.1f mm > 1.0 mm threshold\n", cachedData.rain3h);
  } else if (cachedData.rain12h > 2.0) {
    rainExpected = true;
    DEBUG_PRINTF("🌧️ Rain forecast (12h): %.1f mm > 2.0 mm threshold\n", cachedData.rain12h);
  }

  return rainExpected;
}
```

Εικόνα 4-32: Απόσπασμα του WeatherAPI - Λογική απόφασης βάσει πρόβλεψης.

```cpp
bool WeatherAPI::parseWeatherJSON(const String& json) {
  // Allocate JSON document
  JsonDocument doc;

  DeserializationError error = deserializeJson(doc, json);

  if (error) {
    DEBUG_PRINTF("❌ JSON parse error: %s\n", error.c_str());
    DEBUG_PRINTF("   Input length: %d bytes\n", json.length());
    return false;
  }

  // Check for API errors first
  if (doc.containsKey("cod")) {
    String cod = doc["cod"].as<String>();
    if (cod != "200") {
      DEBUG_PRINTF("❌ API Error (cod): %s\n", cod.c_str());
      if (doc.containsKey("message")) {
        DEBUG_PRINTF("   Message: %s\n", doc["message"].as<String>().c_str());
      }
      return false;
    }
  }

  // Έλεγχος αν υπάρχει list
  if (!doc["list"].is<JsonArray>()) {
    DEBUG_PRINTLN("❌ JSON missing 'list' field");
    DEBUG_PRINTLN("   Available fields:");
    for (JsonPair p : doc.as<JsonObject>()) {
      DEBUG_PRINTF("   - %s\n", p.key().c_str());
    }
    return false;
  }

  // Συγκέντρωση δεδομένων από τα επόμενα intervals
  float rain3h = 0.0;
  float rain12h = 0.0;
  float rain24h = 0.0;
  float avgTemp = 0.0;
  float avgHum = 0.0;
  int count = 0;
```

Εικόνα 4-33: Απόσπασμα του WeatherAPI - Parsing JSON & υπολογισμός βροχής. (1)

```
  JsonArray list = doc["list"].as<JsonArray>();

  for (JsonObject item : list) {
    // Temperature & Humidity
    avgTemp += item["main"]["temp"].as<float>();
    avgHum += item["main"]["humidity"].as<float>();

    // Rain (αν υπάρχει)
    float intervalRain = 0.0;
    if (item["rain"]["3h"].is<float>()) {
      intervalRain = item["rain"]["3h"].as<float>();
    }

    // Αθροισμός βροχής ανά χρονικό παράθυρο
    if (count == 0) rain3h += intervalRain;        // Πρώτη 3ωρη
    if (count < 4) rain12h += intervalRain;        // Πρώτες 4x3h = 12h
    rain24h += intervalRain;                       // Όλες 8x3h = 24h

    // Description (από το πρώτο interval)
    if (count == 0 && item["weather"].is<JsonArray>()) {
      cachedData.description = item["weather"][0]["description"].as<String>();
    }

    count++;
  }

  // Υπολογισμός μέσων όρων
  if (count > 0) {
    avgTemp /= count;
    avgHum /= count;
  }

  // Αποθήκευση
  cachedData.isValid = true;
  cachedData.rain3h = rain3h;
  cachedData.rain12h = rain12h;
  cachedData.rain24h = rain24h;
  cachedData.rainMM = rain24h;  // Χρησιμοποιούμε 24h για decision
  cachedData.willRain = (rain24h >= 2.0);
  cachedData.temperature = avgTemp;
  cachedData.humidity = avgHum;

  return true;
}
```

Εικόνα 4-34: Απόσπασμα του WeatherAPI - Parsing JSON & υπολογισμός βροχής. (2)

```
bool WeatherAPI::shouldWaterNow() {
  #if !ENABLE_WEATHER_CHECK
    return true; // Weather check disabled, always OK to water
  #endif

  // Update αν χρειάζεται
  if (needsUpdate()) {
    update();
  }

  // Αν δεν έχουμε έγκυρα δεδομένα, επίτρεψε πότισμα (safety fallback)
  if (!cachedData.isValid) {
    DEBUG_PRINTLN("⚠️  Weather data invalid or no update yet, proceeding with watering (safety mode)");
    return true;
  }

  // Έλεγχος αν αναμένεται βροχή
  if (willRainSoon()) {
    DEBUG_PRINTLN("🚫 SKIPPING watering: Rain forecast detected!");
    return false;
  }

  DEBUG_PRINTLN("✅ Weather OK for watering");
  return true;
}
```

Εικόνα 4-35: Απόσπασμα του WeatherAPI - Decision engine με safety fallback.

4.8 Web Dashboard στο ESP32

Για την άμεση παρακολούθηση και τον χειροκίνητο έλεγχο του συστήματος, το ESP32 λειτουργεί ως τοπικός web server και παρέχει ένα απλό web dashboard. Η επιλογή αυτή επιτρέπει στον χρήστη να βλέπει σε πραγματικό χρόνο βασικές μετρήσεις (υγρασία εδάφους, στάθμη δοχείου, κατανάλωση νερού, κατάσταση ποτίσματος, δείκτες καιρού) χωρίς να απαιτείται εξωτερική εφαρμογή. Επιπλέον, μέσω του dashboard ο χρήστης μπορεί να εκκινήσει χειροκίνητο πότισμα ή να ενεργοποιήσει άμεσα το emergency stop.

4.8.1 Endpoints και λειτουργικότητα

Η διεπαφή βασίζεται σε συγκεκριμένα endpoints, τα οποία έχουν σχεδιαστεί ώστε να καλύπτουν monitoring και control:

- **GET /**: Επιστρέφει την κύρια σελίδα του dashboard (HTML/JS).
- **GET /data**: Επιστρέφει τα τρέχοντα δεδομένα σε JSON μορφή. Το dashboard κάνει περιοδική ανανέωση ζητώντας το /data, ώστε να ενημερώνονται οι ενδείξεις χωρίς ανανέωση της σελίδας.
- **POST /water**: Εκτελεί χειροκίνητο πότισμα με προκαθορισμένη δόση (π.χ. 10 sec / ~100 mL). Πριν την εκτέλεση εφαρμόζονται οι βασικοί έλεγχοι ασφάλειας (π.χ. emergency flag, κατάσταση δοχείου).
- **POST /stop**: Τερματίζει άμεσα την αντλία και ενεργοποιεί emergency stop (διακοπή ποτίσματος και καταγραφή συμβάντος).

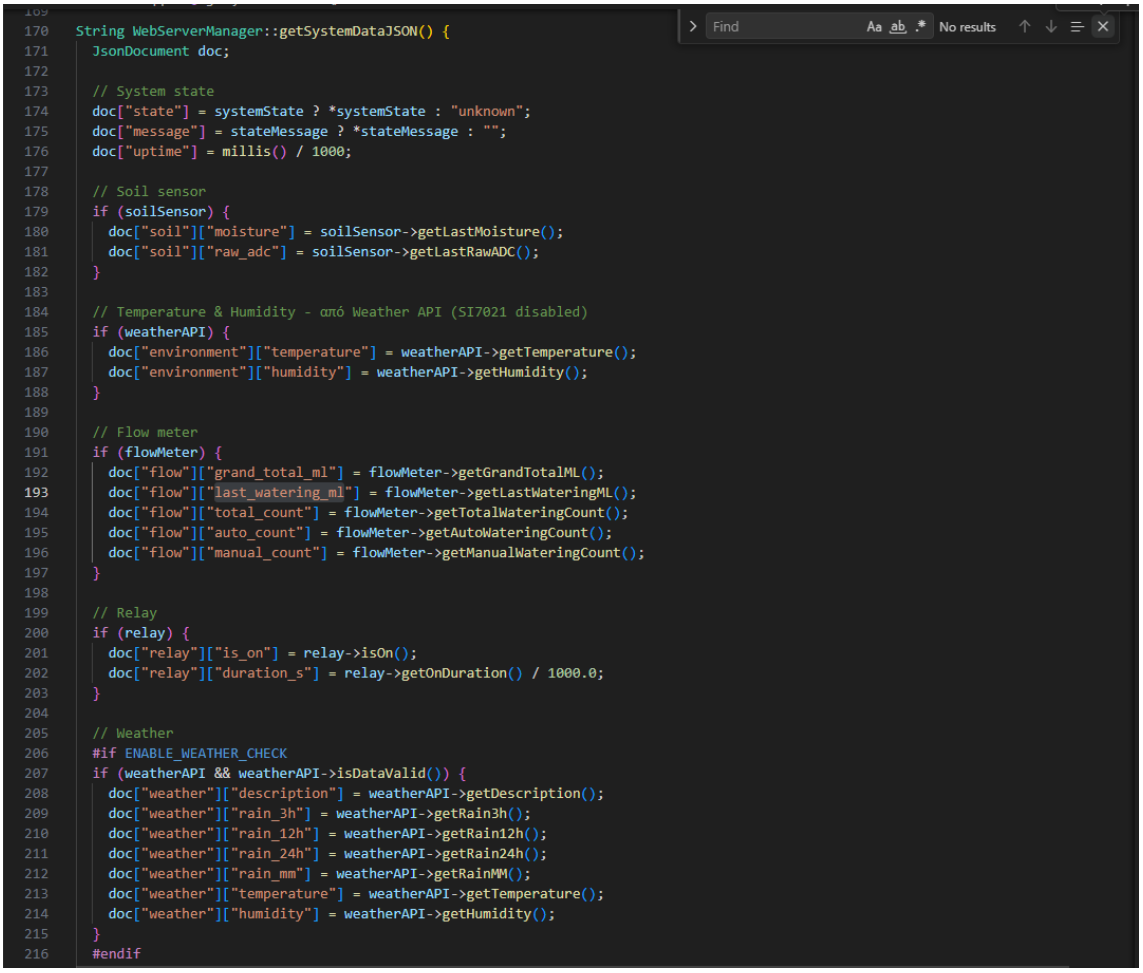
Με την ύπαρξη αυτών των endpoints, το web UI λειτουργεί ως “τοπικός πίνακας ελέγχου” για το σύστημα, ενώ παράλληλα η ίδια πληροφορία μπορεί να καταγράφεται στο backend μέσω MQTT/InfluxDB για ιστορικό.

4.8.2 Περιεχόμενο dashboard και σύνδεση με ιστορικά δεδομένα

Το dashboard εμφανίζει συνοπτικά τις βασικές πληροφορίες λειτουργίας, όπως:

- **Κατάσταση συστήματος** (IDLE/WATERING/ERROR) και μήνυμα/ένδειξη.
- **Υγρασία εδάφους** (ποσοστό και/ή raw ADC).
- **Στάθμη δοχείου** (ποσοστό + κατηγορία OK/LOW/EMPTY).
- **Κατανάλωση νερού** (συνολική και/ή τελευταία δόση).
- **Δείκτες καιρού** (π.χ. αναμενόμενη βροχή 12h/24h, will_rain), καθώς και θερμοκρασία/υγρασία αέρα από OpenWeatherMap.

Σε επίπεδο παρουσίασης, το dashboard προσφέρει και σύνδεση προς τα ιστορικά δεδομένα (Grafana), ώστε ο χρήστης να μπορεί να μεταβεί από το “τρέχον” status στο ιστορικό (π.χ. μεταβολή υγρασίας με τον χρόνο, συμβάντα ποτίσματος, κατανάλωση). Αυτό ενισχύει τον πρακτικό χαρακτήρα του συστήματος, επειδή δίνει τόσο real-time εικόνα όσο και δυνατότητα αναδρομικής παρακολούθησης.



```
String WebServerManager::getSystemDataJSON() {
  JsonDocument doc;

  // System state
  doc["state"] = systemState ? *systemState : "unknown";
  doc["message"] = stateMessage ? *stateMessage : "";
  doc["uptime"] = millis() / 1000;

  // Soil sensor
  if (soilSensor) {
    doc["soil"]["moisture"] = soilSensor->getLastMoisture();
    doc["soil"]["raw_adc"] = soilSensor->getLastRawADC();
  }

  // Temperature & Humidity - από Weather API (SI7021 disabled)
  if (weatherAPI) {
    doc["environment"]["temperature"] = weatherAPI->getTemperature();
    doc["environment"]["humidity"] = weatherAPI->getHumidity();
  }

  // Flow meter
  if (flowMeter) {
    doc["flow"]["grand_total_ml"] = flowMeter->getGrandTotalML();
    doc["flow"]["last_watering_ml"] = flowMeter->getLastWateringML();
    doc["flow"]["total_count"] = flowMeter->getTotalWateringCount();
    doc["flow"]["auto_count"] = flowMeter->getAutoWateringCount();
    doc["flow"]["manual_count"] = flowMeter->getManualWateringCount();
  }

  // Relay
  if (relay) {
    doc["relay"]["is_on"] = relay->isOn();
    doc["relay"]["duration_s"] = relay->getOnDuration() / 1000.0;
  }

  // Weather
  #if ENABLE_WEATHER_CHECK
  if (weatherAPI && weatherAPI->isDataValid()) {
    doc["weather"]["description"] = weatherAPI->getDescription();
    doc["weather"]["rain_3h"] = weatherAPI->getRain3h();
    doc["weather"]["rain_12h"] = weatherAPI->getRain12h();
    doc["weather"]["rain_24h"] = weatherAPI->getRain24h();
    doc["weather"]["rain_mm"] = weatherAPI->getRainMM();
    doc["weather"]["temperature"] = weatherAPI->getTemperature();
    doc["weather"]["humidity"] = weatherAPI->getHumidity();
  }
  #endif
```

Εικόνα 4-36: Απόσπασμα του WebServer - REST API endpoint. (1)

```
  // Water Tank (Ultrasonic)
  if (tankSensor) {
    float level = tankSensor->getLastLevel();
    doc["tank"]["distance_cm"] = tankSensor->getLastDistance();
    doc["tank"]["level_percent"] = level;
    // Status based on level
    if (level >= 50) {
      doc["tank"]["status"] = "OK";
    } else if (level >= 20) {
      doc["tank"]["status"] = "LOW";
    } else {
      doc["tank"]["status"] = "CRITICAL";
    }
  }

  String output;
  serializeJson(doc, output);
  return output;
}
```

Εικόνα 4-37: Απόσπασμα του WebServer - REST API endpoint. (2)

```
server.on("/water", HTTP_POST, [this](AsyncWebServerRequest *request) {
  if (relay && flowMeter) {
    Serial.println("💧 MANUAL WATER - 10 seconds!");

    // Enable flow meter και reset
    flowMeter->reset();
    flowMeter->enable();

    // Πότισμα 10 δευτερόλεπτα (~100ml)
    relay->forceOn();
    delay(WATERING_DURATION_MS);
    relay->forceOff();

    delay(100);

    // Update flow meter
    flowMeter->update();
    float waterUsed = flowMeter->getTotalML();

    // Disable flow meter
    flowMeter->disable();

    // Αποθήκευση στο grand total και last watering (manual)
    flowMeter->finishWatering(true);

    Serial.printf("✅ Done! Used: %.1f mL\n", waterUsed);

    // Response
    char response[128];
    snprintf(response, sizeof(response),
             "{\"status\":\"ok\",\"message\":\"Ποτίστηκε! Χρήση: %.1f mL\",\"success\":true,\"water_ml\":%.1f}",
             waterUsed, waterUsed);
    request->send(200, "application/json", response);
  } else {
    request->send(500, "application/json", "{\"status\":\"error\",\"message\":\"Relay/FlowMeter not available\",\"success\":fals
  }
});
```

Εικόνα 4-38: Απόσπασμα του WebServer - Manual watering με feedback.

4.9 LCD 20×4 (I2C) και τοπική απεικόνιση κατάστασης

Εκτός από την απομακρυσμένη παρακολούθηση μέσω web dashboard και backend dashboards, το σύστημα παρέχει και **τοπική απεικόνιση** μέσω LCD 20×4 με διασύνδεση I2C. Η LCD λειτουργεί ως “άμεσος πίνακας κατάστασης” (status display), χρήσιμος σε περιπτώσεις όπου ο χρήστης βρίσκεται κοντά στη συσκευή, κατά τη ρύθμιση του πρωτοτύπου ή όταν δεν είναι διαθέσιμη κάποια άλλη διεπαφή.

Η χρήση I2C μειώνει τον αριθμό των απαιτούμενων pin, καθώς η LCD επικοινωνεί με το ESP32 μέσω των γραμμών SDA/SCL. Η οθόνη ενημερώνεται περιοδικά (ή όταν αλλάζει σημαντικά η κατάσταση), ώστε να μην επιβαρύνεται άσκοπα η λειτουργία του συστήματος.

4.9.1 Περιεχόμενο οθόνης και μηνύματα κατάστασης

Η LCD εμφανίζει συνοπτικές πληροφορίες που επιτρέπουν γρήγορη κατανόηση της λειτουργίας. Ενδεικτικά, μπορεί να προβάλλει:

- Υγρασία εδάφους (%) και πρόβλεψη βροχής 12 ωρών (mm)
- Θερμοκρασία (°C) και υγρασία αέρα (%) μέσω OpenWeatherMap API
- Στάθμη δοχείου νερού (%) και κατάσταση (OK / LOW!)
- Μήνυμα κατάστασης συστήματος: "IDLE", "WATERING...", "RAIN EXPECTED", "LOW TANK", "SOIL DRY!", "ERROR"

Η πληροφορία είναι σκόπιμα συνοπτική, ώστε να χωρά σε 4 γραμμές και να δίνει το “τι γίνεται τώρα” χωρίς να απαιτείται πρόσβαση στον browser ή στο Grafana.

4.9.2 Λογική ενημέρωσης (update strategy)

Για να παραμένει σταθερή και χρήσιμη, η LCD ενημερώνεται με βάση δύο αρχές:

1. **Περιοδική ενημέρωση με χρονισμό** (π.χ. κάθε 1–2 sec ή λίγο πιο αραιά), ώστε να ανανεώνονται οι βασικές τιμές χωρίς “flicker”.
2. **Άμεση ενημέρωση σε σημαντικά συμβάντα**, όπως αλλαγή state (έναρξη/λήξη ποτίσματος), ενεργοποίηση emergency stop ή μεταβολή tank status (OK → LOW/EMPTY).

Με αυτόν τον τρόπο, το σύστημα δεν σπαταλά χρόνο “γράφοντας συνέχεια” στην οθόνη, αλλά κρατά την πληροφορία επίκαιρη και σταθερή.

```
62  void LCDDisplay::update(float soilMoisture, float temperature, float humidity,
63                          float rain12h, float tankLevel, const char* statusMsg) {
64      if (!initialized) return;
65
66      unsigned long now = millis();
67
68      // Throttle updates - μόνο κάθε 5 sec
69      if (now - lastUpdate < updateInterval) return;
70      lastUpdate = now;
71
72      // Περιοδικό reinit για σταθερότητα (κάθε 1 λεπτό)
73      if (now - lastReinit >= reinitInterval) {
74          reinit();
75          lastReinit = now;
76      }
77
78      // Έλεγχος I2C connection πριν γράψουμε
79      if (!checkConnection()) {
80          Serial.println("⚠️ LCD: I2C connection lost, reinitializing...");
81          reinit();
82          if (!initialized) return;
83      }
84
85      // Line 1: Soil + Rain 12h
86      // "Soil:42.5% Rain:1.2mm"
87      lcd->setCursor(0, 0);
88      char line1[21];
89      snprintf(line1, 21, "Soil:%-5.1f Rain:%4.1f", soilMoisture, rain12h);
90      lcd->print(line1);
91
92      // Line 2: Temp + Humidity
93      // "18.5°C    Hum:65.2%"
94      lcd->setCursor(0, 1);
95      char line2[21];
96      snprintf(line2, 21, "%-5.1f%cC   Hum:%-5.1f", temperature, (char)223, humidity);
97      lcd->print(line2);
98
```

Εικόνα 4-39: Απόσπασμα του LCDDisplay. (1)

```
98
99      // Line 3: Tank level + status
100     // "Tank:75%  OK" or "Tank:15%  LOW!"
101     lcd->setCursor(0, 2);
102     char line3[21];
103     if (tankLevel < 0) {
104         snprintf(line3, 21, "Tank: --            ");
105     } else if (tankLevel <= TANK_LOW_LEVEL) {
106         snprintf(line3, 21, "Tank:%-3.0f%%  LOW!    ", tankLevel);
107     } else {
108         snprintf(line3, 21, "Tank:%-3.0f%%  OK      ", tankLevel);
109     }
110     lcd->print(line3);
111
112     // Line 4: Status message
113     // "IDLE" / "WATERING..." / "RAIN EXPECTED" / "SOIL DRY!"
114     lcd->setCursor(0, 3);
115     char line4[21];
116     snprintf(line4, 21, "%-20s", statusMsg);
117     lcd->print(line4);
118 }
119
```

Εικόνα 4-40: Απόσπασμα του LCDDisplay. (2)

4.10 Επικοινωνία MQTT (publish/subscribe) και πολιτική reconnect/retained

Για την αποστολή τηλεμετρίας (μετρήσεις/κατάσταση) προς το backend και για τη λήψη εντολών ελέγχου, το σύστημα χρησιμοποιεί το πρωτόκολλο **MQTT**. Η επιλογή MQTT είναι κατάλληλη για IoT εφαρμογές, καθώς προσφέρει ελαφριά επικοινωνία, δυνατότητα πολλαπλών subscribers και σαφή διαχωρισμό μεταξύ δεδομένων (telemetry) και εντολών (commands). Ο MQTT broker εκτελείται στον backend κόμβο (Raspberry Pi) μέσω Mosquitto [6]–[8], ενώ το ESP32 λειτουργεί ως client.

4.10.1 Δημοσίευση τηλεμετρίας (topics & δεδομένα)

Το ESP32 δημοσιεύει περιοδικά τις βασικές μετρήσεις και την κατάσταση λειτουργίας του συστήματος. Στόχος είναι το backend να μπορεί να αποθηκεύει ιστορικό (InfluxDB) και να τροφοδοτεί dashboards (Grafana), ενώ παράλληλα να υπάρχει συνεχής ενημέρωση της “τελευταίας κατάστασης”.

Ενδεικτικά δημοσιεύονται:

- **Κατάσταση συστήματος** (IDLE/WATERING/ERROR),
- **Υγρασία εδάφους** (ποσοστό και/ή raw ADC),
- **Στάθμη δοχείου** (ποσοστό + status),
- **Κατανάλωση νερού** (last_watering_ml, grand_total_ml και counters),
- **Δείκτες καιρού** (π.χ. rain_12h, rain_24h, will_rain) και περιβαλλοντικές τιμές από OpenWeatherMap.

Για μεγαλύτερη χρηστικότητα σε consumers που “συνδέονται αργότερα”, η κατάσταση (state) και ορισμένες βασικές τιμές μπορούν να δημοσιεύονται ως **retained**, ώστε ο broker να κρατά το τελευταίο μήνυμα διαθέσιμο.

4.10.2 Λήψη εντολών και ανθεκτικότητα σύνδεσης (commands, reconnect)

Πέρα από τηλεμετρία, το ESP32 μπορεί να λαμβάνει εντολές από το MQTT, μέσω ενός command topic. Αυτό επιτρέπει στο σύστημα να ελέγχεται όχι μόνο από το τοπικό web dashboard, αλλά και από backend/εργαλεία αυτοματισμού (ή μελλοντικά από κάποια εφαρμογή).

Ενδεικτικές εντολές:

- **“water” / “start”**: χειροκίνητη ενεργοποίηση ποτίσματος (με τους ελέγχους ασφάλειας).
- **“stop”**: άμεση διακοπή (emergency stop).

Κατά τη λήψη εντολής, εφαρμόζονται οι ίδιοι μηχανισμοί ασφαλείας που ισχύουν και στο web control (π.χ. έλεγχος tank status, emergency flag, cooldown, max runtime). Έτσι αποφεύγεται το ενδεχόμενο “επικίνδυνης” εντολής από λάθος ή κακό χειρισμό.

Για τη σταθερότητα του συστήματος, υλοποιείται πολιτική **reconnect** σε περίπτωση απώλειας σύνδεσης με τον broker. Το ESP32 ελέγχει περιοδικά τη σύνδεση MQTT και, αν αυτή χαθεί, επιχειρεί επανασύνδεση ανά συγκεκριμένο χρονικό διάστημα. Με αυτόν τον τρόπο το σύστημα παραμένει λειτουργικό σε πραγματικές συνθήκες, όπου το Wi-Fi ή ο broker μπορεί προσωρινά να μην είναι διαθέσιμα.

```
98  void MQTTManager::publishAll() {
99    if (!mqttClient.connected()) return;
100
101   DEBUG_PRINTLN("📡 MQTT: Publishing data...");
102
103   // System status
104   if (systemState) {
105     publish(MQTT_TOPIC_STATUS, *systemState);
106   }
107
108   // Soil moisture
109   if (soilSensor) {
110     publish(MQTT_TOPIC_SOIL, String(soilSensor->getLastMoisture(), 1));
111   }
112
113   // Temperature
114   if (si7021 && si7021->isReady()) {
115     publish(MQTT_TOPIC_TEMP, String(si7021->getLastTemperature(), 1));
116     publish(MQTT_TOPIC_HUMIDITY, String(si7021->getLastHumidity(), 1));
117   }
118
119   // Flow
120   if (flowMeter) {
121     publish(MQTT_TOPIC_FLOW, String(flowMeter->getGrandTotalML(), 0));
122     publish(MQTT_TOPIC_FLOW_SESSION, String(flowMeter->getLastWateringML(), 0));
123     publish(MQTT_TOPIC_FLOW_COUNT, String(flowMeter->getTotalWateringCount()));
124   }
125
126   // Weather
127   #if ENABLE_WEATHER_CHECK
128   if (weatherAPI && weatherAPI->isDataValid()) {
129     // Temperature and Humidity from API
130     publish(MQTT_TOPIC_WEATHER_TEMP, String(weatherAPI->getTemperature(), 1));
131     publish(MQTT_TOPIC_WEATHER_HUM, String(weatherAPI->getHumidity(), 1));
132
133     // Rain forecasts
134     publish(MQTT_TOPIC_WEATHER_RAIN3H, String(weatherAPI->getRain3h(), 1));
135     publish(MQTT_TOPIC_WEATHER_RAIN12H, String(weatherAPI->getRain12h(), 1));
136     publish(MQTT_TOPIC_WEATHER_RAIN24H, String(weatherAPI->getRain24h(), 1));
137     publish(MQTT_TOPIC_WEATHER_WILL, weatherAPI->willRainSoon() ? "true" : "false");
138   }
139   #endif
140
141   DEBUG_PRINTLN("   ✅ Published");
142 }
```

Εικόνα 4-41: Απόσπασμα του MQTTManager. (1)

```
171   
172   void MQTTManager::mqttCallback(char* topic, byte* payload, unsigned int length) {
173     // Convert payload to string
174     String message;
175     for (unsigned int i = 0; i < length; i++) {
176       message += (char)payload[i];
177     }
178   
179     DEBUG_PRINTF("📡 MQTT: Received [%s]: %s\n", topic, message.c_str());
180   
181     // Handle command
182     if (mqttInstance && String(topic) == MQTT_TOPIC_COMMAND) {
183       mqttInstance->handleCommand(message);
184     }
185   }
186   
187   void MQTTManager::handleCommand(const String& command) {
188     if (command == "water" || command == "start") {
189       DEBUG_PRINTLN("   → Starting watering via MQTT");
190       if (relay) {
191         relay->turnOn();
192       }
193   
194     } else if (command == "stop") {
195       DEBUG_PRINTLN("   → Stopping watering via MQTT");
196       if (relay) {
197         relay->turnOff();
198       }
199   
200     } else {
201       DEBUG_PRINTF("   ⚠️ Unknown command: %s\n", command.c_str());
202     }
203   }
204
```

Εικόνα 4-42: Απόσπασμα του MQTTManager. (2)

4.11 Εξέλιξη πρωτοτύπου: μετάβαση σε Weather API για δεδομένα περιβάλλοντος

Κατά την εξέλιξη του πρωτοτύπου εξετάστηκε αρχικά η χρήση τοπικού αισθητήρα περιβάλλοντος (θερμοκρασία/σχετική υγρασία αέρα) ως επιπλέον πηγή δεδομένων. Ωστόσο, στην τελική υλοποίηση έγινε σχεδιαστική επιλογή να αφαιρεθεί ο τοπικός αισθητήρας και να λαμβάνονται οι αντίστοιχες περιβαλλοντικές τιμές από το **OpenWeatherMap**, μαζί με την πρόβλεψη βροχόπτωσης που χρησιμοποιείται ήδη για τη λήψη απόφασης ποτίσματος. [19]

Η αλλαγή αυτή έγινε για τους παρακάτω λόγους:

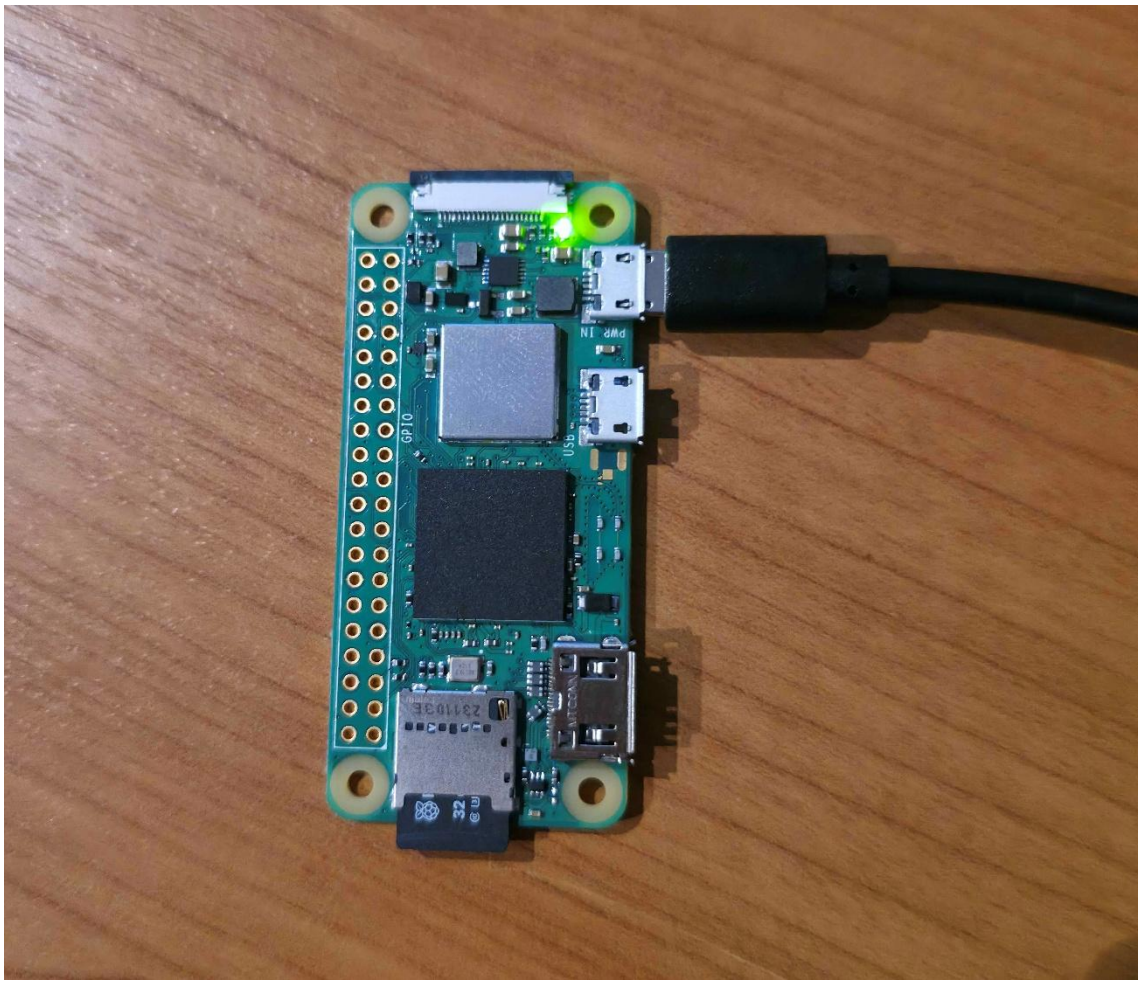
- **Απλοποίηση hardware**: λιγότερα εξαρτήματα, λιγότερες καλωδιώσεις και μικρότερη πολυπλοκότητα στο κύκλωμα.
- **Μείωση πιθανών σημείων αστοχίας**: ένας επιπλέον αισθητήρας και η αντίστοιχη επικοινωνία (π.χ. I2C) μπορούν να δημιουργήσουν προβλήματα σε πραγματικές συνθήκες.
- **Επαρκής κάλυψη απαιτήσεων**: η βασική απόφαση ποτίσματος εξαρτάται κυρίως από την υγρασία εδάφους και την πρόβλεψη βροχής, ενώ θερμοκρασία/υγρασία αέρα χρησιμοποιούνται κυρίως ως συμπληρωματική πληροφορία παρακολούθησης.
- **Ενιαία πηγή περιβαλλοντικών δεδομένων**: το Weather API παρέχει ταυτόχρονα πρόβλεψη βροχής και περιβαλλοντικές τιμές, επιτρέποντας συνεκτική απεικόνιση στο dashboard και στο ιστορικό.

Συνεπώς, το σύστημα διατηρεί την έννοια “environment” στη δομή δεδομένων (για λόγους οργάνωσης και απεικόνισης), αλλά οι τιμές προέρχονται από την υπηρεσία καιρού. Η επιλογή αυτή αντανακλά μια πρακτική προσέγγιση σχεδίασης: αξιοποίηση μιας ήδη υπάρχουσας εξωτερικής πηγής δεδομένων, χωρίς να επιβαρύνεται το πρωτότυπο με πρόσθετο υλικό που δεν είναι απαραίτητο για τον βασικό στόχο του.

5 Υλοποίηση Backend (Raspberry Pi Zero 2 W)

5.1 Mosquitto MQTT Broker (βασική ρύθμιση, persistence, logging)

Ο backend κόμβος (Raspberry Pi Zero 2 W) φιλοξενεί τον MQTT broker **Mosquitto**, ο οποίος λειτουργεί ως κεντρικό σημείο ανταλλαγής μηνυμάτων μεταξύ του ESP32 και των υπηρεσιών αποθήκευσης/οπτικοποίησης. Ο broker επιτρέπει στο ESP32 να δημοσιεύει τηλεμετρία (μετρήσεις και κατάσταση) και να λαμβάνει εντολές (commands), με χαμηλό overhead και αξιόπιστο μηχανισμό publish/subscribe. Η επιλογή Mosquitto είναι κατάλληλη για IoT σενάρια, καθώς είναι ελαφρύς και σταθερός, ειδικά σε συσκευές περιορισμένων πόρων όπως το Raspberry Pi Zero 2 W. [6]–[8], [21]



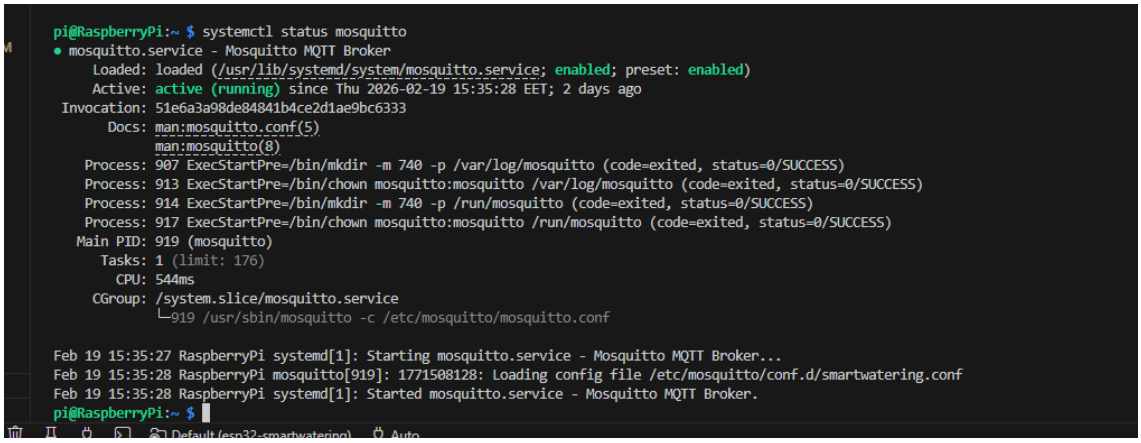
Εικόνα 5-1: Raspberry Pi Zero 2 W

5.1.1 Εγκατάσταση και ενεργοποίηση υπηρεσίας

Η εγκατάσταση πραγματοποιείται μέσω του package manager και ο broker εκκινεί ως system service, ώστε να ξεκινά αυτόματα μετά από επανεκκίνηση. Ενδεικτικά:

```
sudo apt update
sudo apt install -y mosquitto mosquitto-clients
sudo systemctl enable mosquitto
```

```
sudo systemctl start mosquitto
sudo systemctl status mosquitto
```



Εικόνα 5-2: Command systemctl status mosquitto

5.1.2 Βασική ρύθμιση, persistence, logging και δοκιμή publish/subscribe

Η παραμετροποίηση του Mosquitto γίνεται σε αρχείο ρυθμίσεων (π.χ. /etc/mosquitto/mosquitto.conf ή /etc/mosquitto/conf.d/*.conf). Σε επίπεδο πρωτοτύπου, τα κρίσιμα σημεία είναι:

- **listener/port** (π.χ. 1883 στο τοπικό δίκτυο),
- **persistence** για διατήρηση κατάστασης/retained όπου απαιτείται,
- **logging** για διάγνωση.

Για επιβεβαίωση λειτουργίας, γίνεται δοκιμή με mosquitto_sub και mosquitto_pub:

Terminal 1

mosquitto_sub -h localhost -t "smartwatering/#" -v

Terminal 2

mosquitto_pub -h localhost -t "smartwatering/system/state" -m "TEST"

raspberry-pi > mosquitto.conf

```
# Mosquitto MQTT Broker Configuration
# Smart Watering System

# Listener - accept connections from any IP
listener 1883
protocol mqtt

# Allow anonymous connections (local network)
allow_anonymous true

# Logging
log_dest syslog
log_type error
log_type warning
log_type notice
log_type information

# Connection messages
connection_messages true

# Persistence
persistence true
persistence_location /var/lib/mosquitto/

```

Εικόνα 5-3: Mosquitto Configuration file.

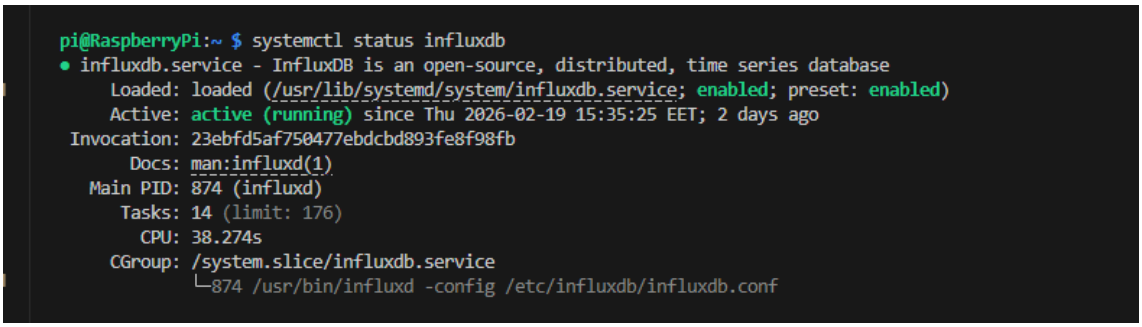
5.2 InfluxDB 1.8 (βάση δεδομένων χρονοσειρών)

Για την αποθήκευση ιστορικών δεδομένων επιλέγεται η **InfluxDB 1.8**, καθώς είναι σχεδιασμένη για δεδομένα τύπου telemetry που καταφθάνουν συνεχώς με χρονική σήμανση (time-series). Το ESP32 δεν γράφει απευθείας στη βάση· τα δεδομένα φτάνουν πρώτα στο Mosquitto και στη συνέχεια καταγράφονται στην InfluxDB μέσω μηχανισμού “γεφύρωσης” (bridge), ο οποίος περιγράφεται στην επόμενη ενότητα. [14]–[16]

5.2.1 Εγκατάσταση και βασική λειτουργία ως υπηρεσία

Η InfluxDB εγκαθίσταται στο Raspberry Pi και εκκινεί ως system service. Στη ρύθμιση του πρωτοτύπου αρκεί να τεκμηριώνεται ότι:

- η υπηρεσία είναι ενεργή,
- η βάση είναι προσβάσιμη τοπικά (τυπικά στο port 8086),
- υπάρχει δυνατότητα χρήσης InfluxQL μέσω shell/CLI.



Εικόνα 5-4: Command systemctl status influxdb.

5.2.2 Δημιουργία database/retention και δομή αποθήκευσης

Δημιουργείται μία βάση, π.χ. smartwatering, και ορίζεται κατάλληλη retention policy (στο prototype συνήθως “forever” ή πολύ μεγάλη διάρκεια), ώστε να διατηρείται ιστορικό για οπτικοποίηση. Ενδεικτικές εντολές InfluxQL:

```
CREATE DATABASE smartwatering;
CREATE RETENTION POLICY "forever" ON "smartwatering" DURATION INF
REPLICATION 1 DEFAULT;
SHOW DATABASES;
SHOW RETENTION POLICIES ON smartwatering;
```

Σε επίπεδο μοντελοποίησης, η InfluxDB αποθηκεύει measurements (π.χ. soil, flow, tank, weather, system) με fields όπως moisture, grand_total_ml, level_percent, rain_12h, state κ.λπ. Αυτό επιτρέπει στη Grafana να δημιουργεί time-series panels και να συσχετίζει γεγονότα ποτίσματος με την εξέλιξη μετρήσεων.

5.3 Γέφυρα MQTT → InfluxDB (mqtt_to_influx.py)

Παρότι το ESP32 δημοσιεύει δεδομένα μέσω MQTT, η InfluxDB δεν “ακούει” απευθείας MQTT. Για να αποθηκεύονται οι μετρήσεις ως χρονοσειρές, υλοποιήθηκε ένας μηχανισμός γέφυρας (bridge) σε Python, ο οποίος κάνει **subscribe** στα MQTT topics του συστήματος, μετασχηματίζει τα μηνύματα σε points (measurement/field/timestamp) και τα γράφει στην InfluxDB. Με αυτόν τον τρόπο επιτυγχάνεται πλήρης διαδρομή δεδομένων:

ESP32 → Mosquitto (MQTT) → mqtt_to_influx.py → InfluxDB → Grafana

Ο bridge λειτουργεί συνεχώς στον backend κόμβο και αποτελεί “το κομμάτι” που μετατρέπει την τηλεμετρία σε αποθηκεύσιμο ιστορικό.

5.3.1 Ροή λειτουργίας του bridge (από topic σε time-series)

Η λειτουργία του bridge μπορεί να περιγραφεί σε τέσσερα βήματα:

1. **Σύνδεση στον MQTT broker** (Mosquitto) με client id και πολιτική reconnect.
2. **Subscribe στο namespace** του project (π.χ. smartwatering/#) ώστε να λαμβάνονται όλες οι μετρήσεις και τα status updates.
3. **Επεξεργασία μηνύματος (on_message callback)**:
    - ανάγνωση του topic,
    - parsing του payload (π.χ. μετατροπή σε float/int/bool όπου χρειάζεται),
    - αντιστοίχιση topic → measurement + field.
4. **Εγγραφή στην InfluxDB**:
    - δημιουργία point με measurement, fields, πιθανώς tags,
    - χρήση timestamp (συνήθως “now” του backend),
    - write στη βάση smartwatering.

5.3.2 Αντιστοίχιση MQTT topics σε InfluxDB measurements/fields (mapping)

Για να είναι καθαρή η αποθήκευση, ο bridge εφαρμόζει συγκεκριμένη αντιστοίχιση. Η λογική είναι ότι:

- το “τι” μετράμε ορίζεται ως **measurement** (π.χ. soil, tank, flow, weather, system)
- το “ποιο πεδίο” μέσα σε αυτό ορίζεται ως **field** (π.χ. moisture, level_percent, grand_total_ml, rain_12h, state)

Παράδειγμα αντιστοίχισης (ενδεικτικό, όπως το έχεις εσύ στα topics σου):

- smartwatering/soil/moisture → measurement: soil, field: moisture
- smartwatering/tank/level_percent → measurement: tank, field: level_percent
- smartwatering/flow/grand_total_ml → measurement: flow, field: grand_total_ml
- smartwatering/weather/rain_12h → measurement: weather, field: rain_12h

- smartwatering/system/state → measurement: system, field: state

Προαιρετικά, μπορείς να χρησιμοποιείς **tags** για καλύτερο φιλτράρισμα (ιδιαίτερα αν μελλοντικά μπουν πολλές συσκευές), π.χ.:

- device="esp32_1", location="balcony".

Πίνακας 5.1: Συγκεντρωτικός πίνακας topic MQTT

MQTT Topic	Measurement.Field	Type	Περιγραφή
`smartwatering/soil/moisture`	`soil.moisture`	Float	Υγρασία εδάφους (%)
~~`smartwatering/environment/temperature`~~	~~`environment.temperature`~~	~~Float~~	~~Θερμοκρασία SI7021 (°C)~~
~~`smartwatering/environment/humidity`~~	~~`environment.humidity`~~	~~Float~~	~~Υγρασία αέρα SI7021 (%)~~
`smartwatering/flow/grand_total_ml`	`flow.grand_total_ml`	Float	Σύνολο νερού (mL)
`smartwatering/flow/session_ml`	`flow.session_ml`	Float	Νερό τρέχοντος ποτίσματος (mL)
`smartwatering/flow/total_count`	`flow.total_count`	Int	Πλήθος ποτισμάτων
`smartwatering/tank/level_percent`	`tank.level_percent`	Float	Στάθμη δεξαμενής (%)
`smartwatering/relay/is_on`	`relay.is_on`	Bool	Κατάσταση αντλίας
`smartwatering/weather/temperature`	`weather.temperature`	Float	Θερμοκρασία API (°C)
`smartwatering/weather/rain_3h`	`weather.rain_3h`	Float	Πρόβλεψη βροχής 3ω (mm)
`smartwatering/weather/will_rain`	`weather.will_rain`	Bool	Αναμένεται βροχή;
`smartwatering/system/state`	`system.state`	String	Κατάσταση συστήματος

5.3.3 Υλοποίηση σε Python: μετατροπές τύπων, αξιοπιστία και batching

Στην υλοποίηση, υπάρχει πρακτικό θέμα: τα payloads από MQTT έρχονται ως strings. Άρα ο bridge πρέπει να κάνει:

- **μετατροπές τύπων** (π.χ. "45.2" → float, "1" → int, `"true" → boolean),
- **χειρισμό σφαλμάτων** (π.χ. άκυρο payload → skip ή log),
- **σταθερή εγγραφή** χωρίς να “μπουκώνει” η βάση.

Σε επίπεδο αξιοπιστίας, είναι καλή πρακτική να υπάρχει:

- reconnect policy για MQTT,
- try/except γύρω από writes προς Influx,
- (προαιρετικά) buffering/batching: αντί να γράφεις κάθε μήνυμα μόνο του, να γράφεις ανά μικρό batch (π.χ. κάθε 1–2 sec ή κάθε N μηνύματα). Αυτό μειώνει overhead και κάνει πιο “ήρεμη” τη λειτουργία.

5.3.4 Εκτέλεση του bridge ως systemd service

Για να τρέχει μόνιμα και να ξεκινά αυτόματα με το Raspberry Pi, το mqtt_to_influx.py ορίστηκε ως **systemd service**. Αυτό έχει δύο μεγάλα πλεονεκτήματα:

- επιβίωση μετά από reboot,
- εύκολος έλεγχος κατάστασης/logs (systemctl, journalctl).

raspberry-pi > mqtt-influx.service

```
[Unit]
Description=MQTT to InfluxDB Bridge for Smart Watering
After=network.target mosquitto.service influxdb.service
Wants=mosquitto.service influxdb.service

[Service]
Type=simple
User=pi
ExecStart=/usr/bin/python3 /opt/mqtt_to_influx.py
Restart=always
RestartSec=10
StandardOutput=journal
StandardError=journal

[Install]
WantedBy=multi-user.target

```

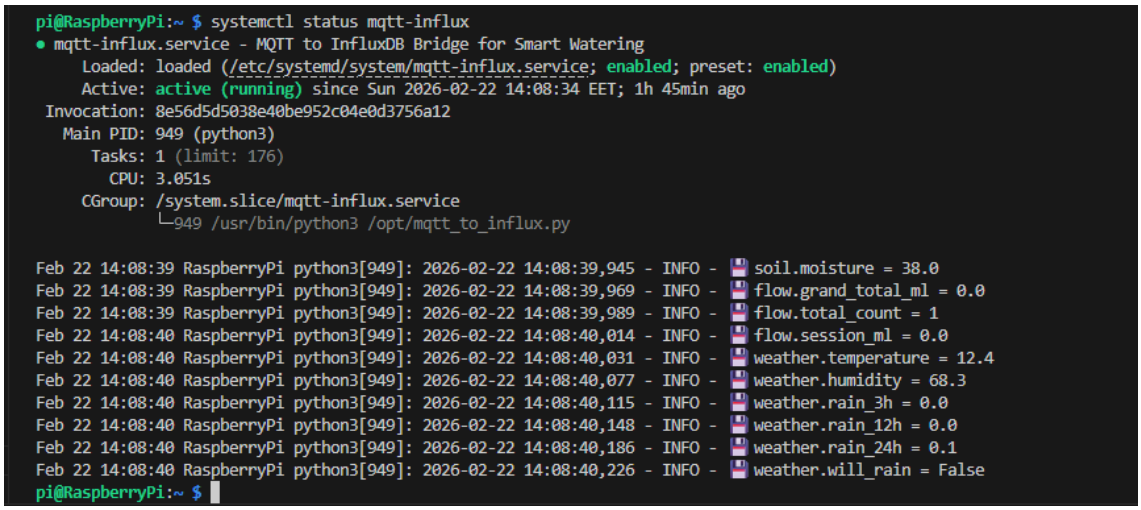
Εικόνα 5-5: MQTT-influx service file.

Ενεργοποίηση/εκκίνηση:

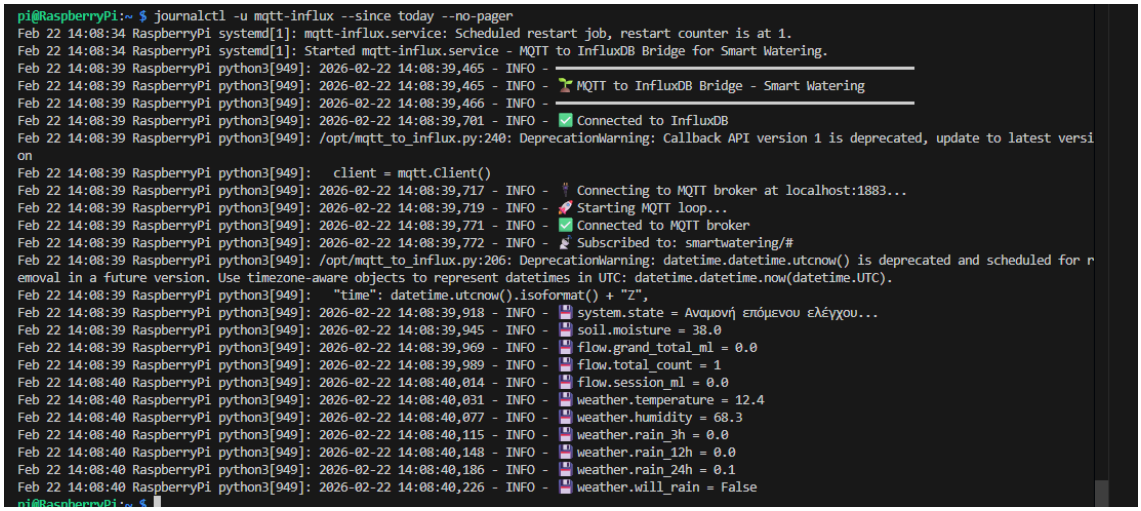
```
sudo systemctl daemon-reload
sudo systemctl enable mqtt-influx
sudo systemctl start mqtt-influx
sudo systemctl status mqtt-influx
```

Logs:

```
journalctl -u mqtt-influx -f
```



Εικόνα 5-6: Command systemctl status mqtt-influx.



Εικόνα 5-7: Command journalctl -u mqtt-influx --since today --no-pager..

5.3.5 Έλεγχος σωστής λειτουργίας (end-to-end test)

Για να επιβεβαιωθεί ότι η αλυσίδα δουλεύει, αρκούν 2 πολύ πρακτικά checks:

1. **MQTT φτάνει στον broker**
   Με mosquitto_sub βλέπεις live τα topics:

```
mosquitto_sub -h localhost -t "smartwatering/#" -v
```

2. **InfluxDB γράφει δεδομένα**
   Με ένα απλό query βλέπεις ότι υπάρχουν εγγραφές (π.χ. soil):

```
SELECT * FROM soil ORDER BY time DESC LIMIT 5;
```

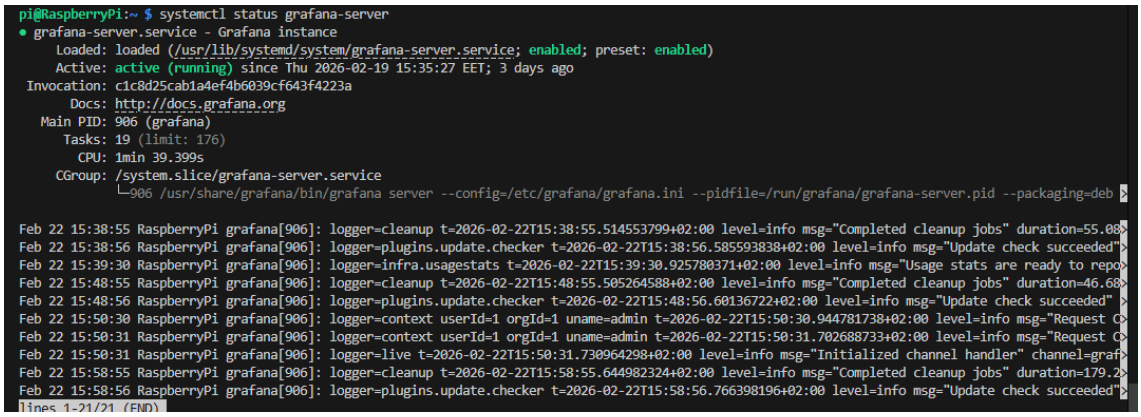
5.4 Grafana Dashboard

Για την οπτικοποίηση των μετρήσεων και την παρακολούθηση της λειτουργίας του συστήματος σε πραγματικό χρόνο αλλά και ιστορικά, χρησιμοποιείται το **Grafana**. Το Grafana συνδέεται στην InfluxDB και επιτρέπει τη δημιουργία dashboards με γραφήματα, δείκτες (gauges), πίνακες και χρονικές σειρές, ώστε να είναι εύκολη η κατανόηση της συμπεριφοράς του συστήματος (π.χ. πότε έγινε πότισμα, πώς μεταβλήθηκε η υγρασία εδάφους, ποια ήταν η στάθμη δοχείου, τι δείχνει η πρόβλεψη βροχής). [17]–[18]

5.4.1 Εγκατάσταση και σύνδεση με InfluxDB (Data Source)

Το Grafana εγκαθίσταται στον backend κόμβο και εκκινεί ως υπηρεσία. Στη συνέχεια, γίνεται ρύθμιση ενός **Data Source** τύπου InfluxDB (InfluxQL για InfluxDB 1.8), με βάση την οποία το Grafana θα εκτελεί queries στη βάση smartwatering. Στην ρύθμιση δηλώνονται ενδεικτικά:

- URL της InfluxDB (π.χ. http://localhost:8086),
- Database: smartwatering,
- Query language: InfluxQL (για 1.8),
- (προαιρετικά) user/password αν χρησιμοποιούνται.



Εικόνα 5-8: Command systemctl status Grafana-server.

5.4.2 Σχεδίαση dashboards και βασικά panels

Το dashboard **ESP32 Smart Watering System** οργανώνεται ώστε να απαντά σε δύο πρακτικά ερωτήματα:

1. **Τι συμβαίνει τώρα;** (τρέχουσα κατάσταση / άμεση εποπτεία)
2. **Τι συνέβη πριν;** (ιστορικό / τάσεις / επιβεβαίωση ότι το πότισμα “έπιασε”)

Στη συγκεκριμένη υλοποίηση χρησιμοποιείται χρονικό παράθυρο **Last 1 hour** και **refresh ~10s** (ανάλογα με το publish interval των αισθητήρων).

Παρακάτω περιγράφονται τα panels όπως είναι:

1. (α) Soil Moisture (%) – Χρονική σειρά υγρασίας εδάφους
Γράφημα time series της μεταβλητής soil.moisture (υγρασία %), ώστε να φαίνεται:

- η σταδιακή πτώση της υγρασίας με τον χρόνο,
- η απότομη αύξηση μετά από κύκλο ποτίσματος,
- και γενικά η άμεση επίδραση του συστήματος στο έδαφος.

Επιπλέον, χρησιμοποιούνται threshold bands (χρωματικές ζώνες) για "dry / ok / wet" οπτικοποίηση.

Ενδεικτικό query (InfluxQL):

- `SELECT mean("moisture") FROM "soil" WHERE $timeFilter GROUP BY time($__interval) fill(null)`

Ρύθμιση: Unit = percent (0–100), refresh 10s.

2. (β) Temperature (Weather API) – Θερμοκρασία περιβάλλοντος
Time series για τη weather.temperature (°C). Χρήσιμο για συσχέτιση με τις ανάγκες ποτίσματος (π.χ. υψηλότερη θερμοκρασία → ταχύτερη εξάτμιση).

Query:

- `SELECT mean("temperature") FROM "weather" WHERE $timeFilter GROUP BY time($__interval) fill(null)`

Unit: °C.

3. (γ) Humidity (Weather API) – Σχετική υγρασία αέρα
Time series για τη weather.humidity (%). Βοηθάει να "διαβάζεις" τις συνθήκες περιβάλλοντος που επηρεάζουν την απώλεια νερού από το φυτό.

Query:

- `SELECT mean("humidity") FROM "weather" WHERE $timeFilter GROUP BY time($__interval) fill(null)`

Unit: % (RH).

4. (δ) Current Soil Moisture – Μετρητής (Gauge)
Gauge που δείχνει την τρέχουσα υγρασία εδάφους (last value), με χρωματική κλίμακα (π.χ. κόκκινο/πορτοκαλί/πράσινο) για άμεση "με μια ματιά" αξιολόγηση.

Query:

- `SELECT last("moisture") FROM "soil" WHERE $timeFilter`

5. (ε) Water Usage (mL) – Συνολική κατανάλωση νερού
Μεγάλο Stat panel που εμφανίζει τη συνολική κατανάλωση σε mL (π.χ. grand total). Είναι το βασικό KPI για το "πόσο πότισε συνολικά" το σύστημα στο επιλεγμένο διάστημα/συνολικά.

Query (ενδεικτικά):

- `SELECT last("grand_total_ml") FROM "flow" WHERE $timeFilter`

Unit: mL.

6. (στ) Rain Forecast (mm) – Πρόβλεψη/δείκτες βροχής
Time series με τις μεταβλητές weather.rain_3h, weather.rain_12h, weather.rain_24h (mm), ώστε να φαίνεται αν "έρχεται" βροχή και πώς αυτό μπορεί να επηρεάσει τη στρατηγική ποτίσματος (ιδίως σε μελλοντική επέκταση με κανόνες "skip watering if rain").

Queries:

- `SELECT last("rain_3h") FROM "weather" WHERE $timeFilter`
- `SELECT last("rain_12h") FROM "weather" WHERE $timeFilter`
- `SELECT last("rain_24h") FROM "weather" WHERE $timeFilter`

Unit: mm.

7. (ζ) Current Weather – Cards για τρέχοντα καιρικά
Δύο Stat panels για:

- Temp (τρέχουσα θερμοκρασία)
- Humidity (τρέχουσα υγρασία)

Στόχος: άμεση εικόνα "τώρα" χωρίς να διαβάζεις το γράφημα.

Queries:

- `SELECT last("temperature") FROM "weather" WHERE $timeFilter`
- `SELECT last("humidity") FROM "weather" WHERE $timeFilter`

8. (η) Soil Moisture Summary – Σύνοψη υγρασίας (Current / Min / Max)
Τρία Stat panels που συνοψίζουν τη συμπεριφορά της υγρασίας στο επιλεγμένο time range:

- Current: τρέχουσα τιμή
- Min: ελάχιστη τιμή
- Max: μέγιστη τιμή

Αυτό βοηθάει πολύ σε debugging/αξιολόγηση: βλέπεις γρήγορα "πόσο χαμηλά έπεσε" και "μέχρι πού ανέβηκε" μετά τα ποτίσματα.

Queries (ενδεικτικά):

- `SELECT last("moisture") FROM "soil" WHERE $timeFilter`
- `SELECT min("moisture") FROM "soil" WHERE $timeFilter`
- `SELECT max("moisture") FROM "soil" WHERE $timeFilter`

9. (θ) Total Waterings – Σύνολο ποτισμάτων & συνολικό νερό
Δύο Stat panels:

- Total: πόσοι κύκλοι ποτίσματος εκτελέστηκαν (counter)
- Water: συνολικό νερό (mL) για γρήγορη σύνοψη

Queries (ενδεικτικά):

- `SELECT last("total_count") FROM "flow" WHERE $timeFilter`
- `SELECT last("grand_total_ml") FROM "flow" WHERE $timeFilter`

Εικόνα 5-9: Command systemctl status Grafana-server.

5.4.3 Όρια, ειδοποιήσεις και πρακτικές παρακολούθησης (προαιρετικά)

Ανάλογα με το πόσο θες να το πας “ένα βήμα παραπάνω”, μπορείς να χρησιμοποιήσεις thresholds/alerts. Ακόμα και αν δεν υλοποιήσεις πλήρες alerting, τα thresholds είναι χρήσιμα:

- **Tank level thresholds:** π.χ. προειδοποίηση κάτω από 20% και κρίσιμο κάτω από 5%.
- **Soil moisture threshold:** οπτική ένδειξη κάτω από 30% (dry).
- **Rain threshold:** οπτική ένδειξη όταν rain_12h > 2mm.

Αν έχεις ενεργοποιήσει alerts (ανάλογα με έκδοση/ρυθμίσεις), μπορείς να αναφέρεις ότι μπορεί να δοθεί ειδοποίηση (π.χ. “LOW TANK”) ώστε ο χρήστης να ενημερωθεί να γεμίσει το δοχείο. Δεν είναι υποχρεωτικό για την πτυχιακή, αλλά δείχνει ωριμότητα συστήματος.

5.5 Συνολική ροή backend και έλεγχος λειτουργίας (end-to-end)

Το backend του συστήματος υλοποιείται ως ένα σύνολο υπηρεσιών που συνεργάζονται για να μετατρέψουν την τηλεμετρία από το ESP32 σε αποθηκεύσιμο ιστορικό και σε οπτικοποιήσιμα dashboards. Η συνολική ροή λειτουργίας μπορεί να συνοψιστεί ως εξής:

1. Το **ESP32** δημοσιεύει μετρήσεις και κατάσταση σε MQTT topics (telemetry) και, όπου απαιτείται, λαμβάνει εντολές (commands).
2. Ο **Mosquitto broker** στο Raspberry Pi δέχεται τα MQTT μηνύματα.
3. Η υπηρεσία **mqtt_to_influx.py** κάνει subscribe στα topics, μετασχηματίζει τα δεδομένα και τα γράφει στην **InfluxDB** (χρονικές σειρές).
4. Το **Grafana** εκτελεί queries στην InfluxDB και παρουσιάζει τα δεδομένα σε dashboards (real-time και ιστορικά).

Με αυτή τη δομή, το backend είναι λειτουργικά “καθαρό”: κάθε υπηρεσία έχει σαφή ρόλο, ενώ η επικοινωνία γίνεται με τυπικά πρωτόκολλα και εργαλεία (MQTT, InfluxDB, Grafana), κάτι που ενισχύει την αξιοπιστία και την επεκτασιμότητα της λύσης.

5.5.1 Υπηρεσίες που εκτελούνται και αυτόματη εκκίνηση’

Για να εξασφαλιστεί ότι το σύστημα επανέρχεται αυτόματα μετά από επανεκκίνηση, οι βασικές υπηρεσίες εκτελούνται ως systemd services:

- **mosquitto.service** (MQTT broker)
- **influxdb.service** (database)
- **grafana-server.service** (dashboards)
- **mqtt_to_influx.service** (bridge από MQTT σε InfluxDB)

Ο έλεγχος της κατάστασης κάθε υπηρεσίας γίνεται μέσω systemctl, ενώ τα logs είναι διαθέσιμα μέσω journalctl. Ενδεικτικά:

```
systemctl status mosquitto

systemctl status influxdb

systemctl status grafana-server

systemctl status mqtt_to_influx
```

Για συγκεντρωτικό έλεγχο, μπορεί να χρησιμοποιηθεί μία εντολή που εμφανίζει συνοπτικά τις σχετικές υπηρεσίες:

```
systemctl --no-pager --full | grep -E
"mosquitto|influxdb|grafana|mqtt_to_influx"
```

5.5.2 Διαδικασία ελέγχου σωστής λειτουργίας (checklist)

Για “end-to-end” επιβεβαίωση ότι όλα λειτουργούν, ακολουθείται μια πρακτική checklist:

(α) Έλεγχος ότι φτάνουν MQTT μηνύματα

```
mosquitto_sub -h localhost -t "smartwatering/#" -v
```

Αν το σύστημα λειτουργεί, θα εμφανίζονται live τιμές (state, soil, tank, flow, weather).

(β) Έλεγχος ότι γράφονται δεδομένα στην InfluxDB
Στο Influx shell, εκτελείται ένα απλό query:

```
SELECT * FROM soil ORDER BY time DESC LIMIT 5;
```

Αν επιστρέφονται πρόσφατες εγγραφές, τότε ο bridge λειτουργεί και η βάση ενημερώνεται.

(γ) Έλεγχος ότι το Grafana “βλέπει” τη βάση
Στο Grafana επιβεβαιώνεται ότι:

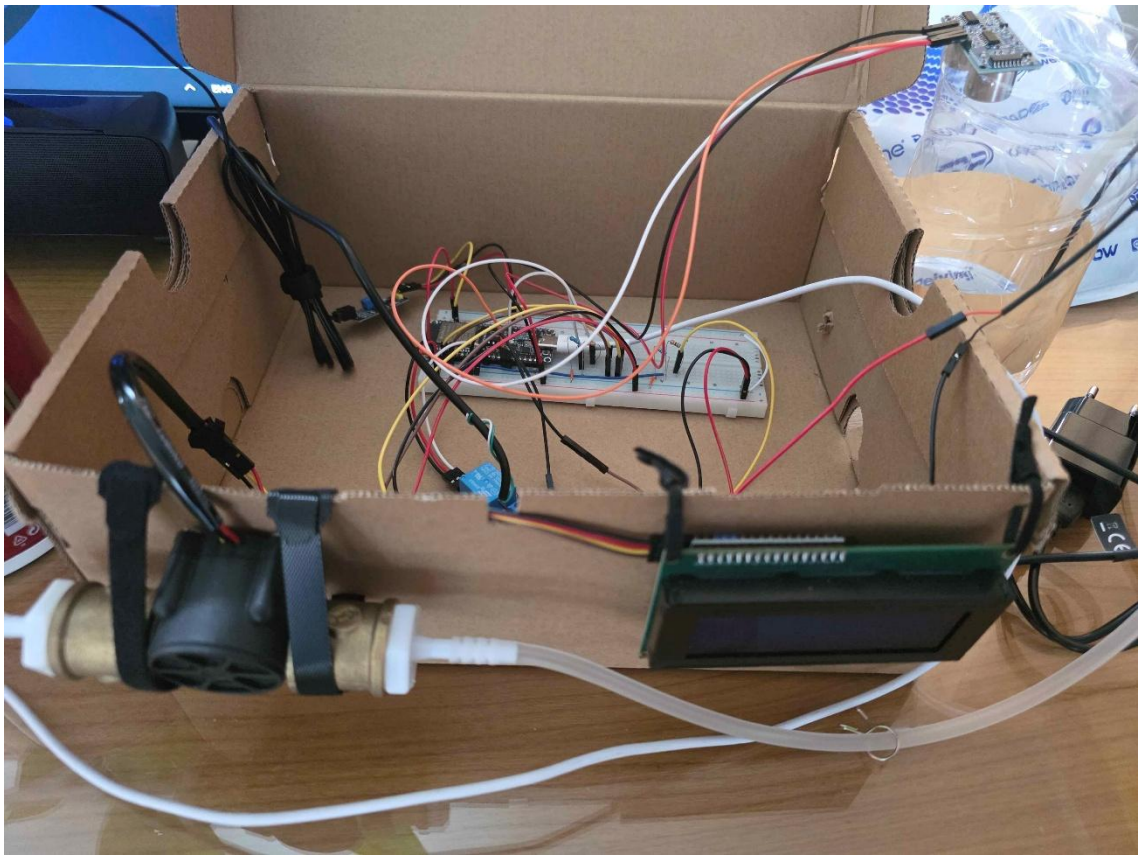
- το Data Source είναι “OK” (Save & Test),
- τα panels επιστρέφουν δεδομένα και ανανεώνονται.

(δ) Έλεγχος συσχέτισης συμβάντος με δεδομένα
Ένα πρακτικό τεστ είναι να ξεκινήσει ένα χειροκίνητο πότισμα (από web ή MQTT) και να επιβεβαιωθεί ότι:

- αλλάζει το system.state σε WATERING,
- ενημερώνεται το flow.last_watering_ml,
- στο γράφημα soil.moisture φαίνεται τάση αύξησης μετά από λίγη ώρα,
- στο tank level φαίνεται σταδιακή μείωση (αν είναι αρκετά ευαίσθητο/σταθερό).

6 Λειτουργία Πρωτοτύπου και Παράδειγμα Χρήσης

Το κεφάλαιο αυτό περιγράφει τον τρόπο λειτουργίας του πρωτοτύπου σε πραγματικές συνθήκες χρήσης. Αρχικά παρουσιάζονται οι βασικές παράμετροι ρύθμισης (όρια, χρονισμοί και δοσολογία ποτίσματος), στη συνέχεια η τυπική συμπεριφορά του συστήματος μέσω καταστάσεων (IDLE/WATERING/ERROR) και η ενημέρωση των διεπαφών, ενώ τέλος δίνονται ενδεικτικά σενάρια που τεκμηριώνουν τη λογική απόφασης και τους μηχανισμούς ασφαλείας. Επιπλέον, παρουσιάζονται παραδείγματα ιστορικών δεδομένων από το Grafana και γίνεται ερμηνεία των βασικών γραφημάτων.



Εικόνα 6-1: Ολόκληρο το setup

6.1 Παραμετροποίηση συστήματος (thresholds, cooldowns, max runtime, dose)

Η λειτουργία του συστήματος βασίζεται σε ένα σύνολο παραμέτρων που επιτρέπουν προσαρμογή ανάλογα με το είδος του εδάφους, τη χωρητικότητα δοχείου νερού και τα χαρακτηριστικά της αντλίας. Οι παράμετροι αυτές ορίζονται συγκεντρωμένα (π.χ. σε αρχείο ρυθμίσεων όπως config.h) ώστε να αλλάζουν εύκολα χωρίς να απαιτείται τροποποίηση της συνολικής λογικής ελέγχου.

Οι κύριες κατηγορίες παραμετροποίησης είναι οι ακόλουθες:

(α) Όρια απόφασης (thresholds)

- **Όριο υγρασίας εδάφους (soil threshold):** τιμή κάτω από την οποία το έδαφος θεωρείται “ξηρό” και επιτρέπεται να εξεταστεί πότισμα (π.χ. 30%).
- **Όριο βροχής (rain threshold):** τιμή πάνω από την οποία το σύστημα αποφεύγει πότισμα όταν αναμένεται βροχόπτωση σε συγκεκριμένο παράθυρο χρόνου (π.χ. >2 mm στο 12ωρο).
- **Όρια στάθμης δοχείου:** τιμές που κατηγοριοποιούν το δοχείο ως OK/LOW/EMPTY και επηρεάζουν την άδεια για πότισμα.

(β) Χρονισμοί και έλεγχοι προστασίας

- **Cooldown:** ελάχιστος χρόνος μεταξύ δύο ποτισμάτων, ώστε να αποφεύγονται συνεχείς ενεργοποιήσεις λόγω θορύβου/μεταβατικών μετρήσεων.
- **Max runtime:** μέγιστη επιτρεπτή διάρκεια λειτουργίας αντλίας ως ασφαλιστική δικλείδα έναντι αστοχίας λογικής ή “κολλήματος”.
- **Συχνότητες εργασιών:** χρονισμοί ανάγνωσης αισθητήρων, ενημέρωσης καιρού και δημοσίευσης MQTT.

(γ) Δοσολογία (dose-based watering)
Το πρωτότυπο υλοποιεί πότισμα “με δόση”, δηλαδή ενεργοποίηση της αντλίας για προκαθορισμένο χρόνο (π.χ. 10 sec) που αντιστοιχεί σε κατά προσέγγιση συγκεκριμένο όγκο νερού (π.χ. ~100 mL, όπως προκύπτει από το flow meter). Η επιλογή δόσης κάνει τη λειτουργία προβλέψιμη και περιορίζει τον κίνδυνο υπερ-ποτίσματος.

6.2 Τυπική λειτουργία (IDLE/WATERING/ERROR) και ενημέρωση διεπαφών

Η συμπεριφορά του συστήματος οργανώνεται ως μηχανή καταστάσεων (state machine), ώστε να υπάρχει σαφής ροή και να αποφεύγονται ανεπιθύμητες μεταβάσεις. Σε επίπεδο χρήστη, η λειτουργία αποτυπώνεται κυρίως σε τρεις βασικές καταστάσεις: IDLE, WATERING και ERROR/STOP. Εσωτερικά μπορεί να υπάρχουν ενδιάμεσες φάσεις (π.χ. CHECKING/COMPLETED), οι οποίες όμως λειτουργούν υποστηρικτικά και δεν αλλάζουν τον πρακτικό τρόπο χρήσης.

- **Κατάσταση IDLE**

Στην IDLE κατάσταση το σύστημα:

- συλλέγει περιοδικά μετρήσεις αισθητήρων (soil/tank/flow counters),

- ενημερώνει το web dashboard και τη LCD με τις τρέχουσες τιμές,
- εκτελεί περιοδικά τον έλεγχο “χρειάζεται πότισμα;”, εφαρμόζοντας όρια (thresholds), χρονισμούς (cooldown) και προϋποθέσεις ασφαλείας (tank status, emergency flags).
- **Κατάσταση WATERING**

Η WATERING ενεργοποιείται είτε από:

- την αυτόματη λογική (χαμηλή υγρασία + δεν αναμένεται βροχή + επιτρέπεται από safeguards), είτε
- χειροκίνητη εντολή (web endpoint ή MQTT command).

Κατά τη διάρκεια του ποτίσματος:

- ενεργοποιείται το ρελέ/αντλία,
- καταγράφεται η κατανάλωση από το flow meter,
- ενημερώνονται μετρητές (total/auto/manual),
- το σύστημα τερματίζει με βάση τη δόση ή/και προστασίες (max runtime / stop).
- **Κατάσταση ERROR / STOP**

Η κατάσταση ERROR/STOP ενεργοποιείται όταν απαιτείται ασφαλής συμπεριφορά, όπως:

- ενεργοποίηση emergency stop από τον χρήστη,
- υπέρβαση max runtime,
- αποτροπή ποτίσματος λόγω χαμηλής στάθμης ή σοβαρού σφάλματος.

Στην κατάσταση αυτή το ρελέ απενεργοποιείται άμεσα και το σύστημα εμφανίζει καθαρό μήνυμα σε LCD/web dashboard, ώστε να είναι σαφές ότι απαιτείται παρέμβαση ή αποκατάσταση.

Ενημέρωση διεπαφών (Web, LCD, MQTT/Grafana)

Η ενημέρωση γίνεται ταυτόχρονα σε διαφορετικά επίπεδα:

- **Web dashboard (ESP32):** τα δεδομένα διατίθενται μέσω GET /data, ενώ εντολές δίνονται μέσω POST /water και POST /stop.
- **LCD 20×4:** προβάλει συνοπτικά soil/tank/state/temp/humidity και κρίσιμα μηνύματα.
- **MQTT → Backend:** δημοσίευση τηλεμετρίας και κατάστασης, καταγραφή στην InfluxDB μέσω bridge και οπτικοποίηση στο Grafana.

6.3 Ενδεικτικά σενάρια

6.3.1 Ξηρό έδαφος → ενεργοποίηση ποτίσματος

Όταν η υγρασία εδάφους πέσει κάτω από το όριο, το σύστημα εξετάζει τις προϋποθέσεις ασφαλείας (cooldown, tank status, emergency flag) και τον κανόνα βροχής. Εφόσον δεν αναμένεται βροχή πάνω από το όριο και επιτρέπεται από safeguards, ξεκινά αυτόματο πότισμα για τη ρυθμισμένη δόση.

Αναμενόμενες ενδείξεις:

- μετάβαση state: IDLE → WATERING → IDLE,
- αύξηση last_watering_ml και grand_total_ml,
- αύξηση auto_count και total_count,
- σταδιακή βελτίωση της ένδειξης υγρασίας (ανάλογα με θέση/χώμα).

6.3.2 Αναμενόμενη βροχή → αναβολή/αποφυγή ποτίσματος

Αν το έδαφος είναι οριακά ή και χαμηλά, αλλά από το OpenWeatherMap προκύπτει ότι στο προσεχές παράθυρο (π.χ. 12h) αναμένεται βροχή πάνω από το όριο, το σύστημα επιλέγει να παραλείψει πότισμα ώστε να αποφευχθεί σπατάλη.

Αναμενόμενες ενδείξεις:

- ενεργοί δείκτες rain_12h/will_rain,
- το state παραμένει IDLE (δεν ενεργοποιείται αντλία),
- προαιρετικό μήνυμα “SKIP: RAIN” σε LCD/web (αν το εμφανίζεις).

6.3.3 Σφάλμα αισθητήρα/σύνδεσης → ασφαλής λειτουργία (emergency stop/locks)

Το σύστημα περιλαμβάνει μηχανισμούς fail-safe για περιπτώσεις όπως άκυρες μετρήσεις (out-of-range), προσωρινή απώλεια σύνδεσης (Wi-Fi/MQTT) ή εντολή emergency stop. Σε κάθε τέτοια περίπτωση, στόχος είναι η άμεση απενεργοποίηση της αντλίας και η αποτροπή ανεπιθύμητων ενεργοποιήσεων μέχρι να αποκατασταθεί η συνθήκη.

Αναμενόμενες ενδείξεις:

- άμεσο relay OFF,
- κατάσταση ERROR/STOP ή σαφές status message,
- σχετική καταγραφή σε logs/ιστορικό (όπου υπάρχει),
- (αν εφαρμόζεται lock) απαίτηση reset/clear πριν συνεχίσει.

6.4 Παραδείγματα ιστορικών δεδομένων (Grafana) και ερμηνεία γραφημάτων

Η οπτικοποίηση στο Grafana επιτρέπει να αξιολογηθεί πρακτικά η λειτουργία του πρωτοτύπου στον χρόνο και να συσχετιστούν οι ενέργειες ποτίσματος με τις μετρήσεις. Τα βασικά συμπεράσματα προκύπτουν κυρίως από τη συσχέτιση: “πτώση υγρασίας → απόφαση → πότισμα → μεταβολή υγρασίας”, καθώς και από την καταγραφή κατανάλωσης και στάθμης δοχείου.

Ενδεικτική ερμηνεία βασικών panels:

- **Soil moisture:** σταδιακή πτώση με τον χρόνο και άνοδος μετά από πότισμα (πιθανή καθυστέρηση λόγω διάχυσης/θέσης αισθητήρα).
- **System state:** μικρά χρονικά διαστήματα WATERING που χρονικά συμπίπτουν με αλλαγές στο water usage και (αργότερα) στη soil moisture.
- **Water usage:** ο grand_total_ml αυξάνει μονοτονικά, ενώ last_watering_ml δείχνει τον όγκο ανά κύκλο. Οι counters (auto/manual) τεκμηριώνουν τον βαθμό αυτοματοποίησης.
- **Tank level:** μείωση στον χρόνο και αλλαγές status (OK/LOW/EMPTY). Τυχόν spikes μπορεί να οφείλονται σε θόρυβο/ανακλάσεις του υπερηχητικού και είναι χρήσιμο να αναφερθούν ως πρακτικό θέμα μέτρησης.
- **Weather indicators:** δείχνουν πότε το σύστημα “έβλεπε” αναμενόμενη βροχή και τεκμηριώνουν γιατί δεν ενεργοποιήθηκε πότισμα.

6.5 Συζήτηση αποτελεσμάτων και συμπεράσματα λειτουργίας πρωτοτύπου

Οι δοκιμές του πρωτοτύπου πραγματοποιήθηκαν σε **φυτό εσωτερικού χώρου (σπαθίφυλλο) σε μικρή γλάστρα**, με στόχο τη διατήρηση της υγρασίας εδάφους σε αποδεκτά επίπεδα χωρίς συχνές χειροκίνητες παρεμβάσεις. Από την παρακολούθηση στο Grafana προκύπτουν τα ακόλουθα βασικά ευρήματα: [23]–[25]

(α) Επαλήθευση της αλυσίδας "απόφαση → πότισμα → μεταβολή υγρασίας"
Η χρονοσειρά της υγρασίας εδάφους αποτυπώνει τη σταδιακή πτώση με τον χρόνο και την αύξηση μετά από ποτιστικά συμβάντα. Αυτό αποτελεί ισχυρή ένδειξη ότι η λογική απόφασης και η ενεργοποίηση της αντλίας οδηγούν σε **μετρήσιμη μεταβολή** στο έδαφος, με πιθανή χρονική καθυστέρηση λόγω διάχυσης του νερού και θέσης του αισθητήρα.

(β) Κατανάλωση νερού και επαναληψιμότητα δόσης
Η ύπαρξη μετρητών (π.χ. grand_total_ml, last_watering_ml, counters) επιτρέπει ποσοτική αξιολόγηση. Ενδεικτικά, σε περίοδο παρατήρησης καταγράφηκαν **πολλαπλοί κύκλοι ποτίσματος** και συνολική κατανάλωση της τάξης εκατοντάδων mL, γεγονός που δείχνει ότι το σύστημα μπορεί να τεκμηριώνει το "πόσο πότισε" και όχι μόνο το "αν πότισε". Παράλληλα, η δόση-based προσέγγιση καθιστά το πότισμα **προβλέψιμο** και μειώνει τον κίνδυνο υπερ-ποτίσματος.

(γ) Θόρυβος/ανωμαλίες μετρήσεων και ανάγκη φίλτρων
Σε πραγματικές συνθήκες είναι πιθανό να εμφανιστούν στιγμιαίες ανωμαλίες (π.χ. απότομες πτώσεις ή "μη ρεαλιστικές" τιμές), λόγω αισθητήρα, καλωδίωσης, υγρασίας στην επιφάνεια ή ηλεκτρικού θορύβου. Για τον λόγο αυτό, προτείνεται η χρήση **moving average / median filter**, καθώς και **απόρριψη outliers** (π.χ. τιμές εκτός εύρους ή απότομες μεταβολές που δεν δικαιολογούνται φυσικά), ώστε να μειωθούν ψευδείς ενεργοποιήσεις.

(δ) Ρόλος καιρού και δυνατότητα "έξυπνων" αποφάσεων
Οι δείκτες καιρού (θερμοκρασία, υγρασία αέρα, rain forecasts) δίνουν το πλαίσιο για προσαρμογή στρατηγικής. Ακόμη και αν στο παρόν πρωτότυπο χρησιμοποιούνται κυρίως ως πληροφορία/ένδειξη, αποτελούν βάση για κανόνες όπως **αναβολή ποτίσματος όταν αναμένεται βροχή** ή δυναμική προσαρμογή thresholds ανάλογα με συνθήκες.

(ε) Συνολική αξιολόγηση
Συνολικά, το πρωτότυπο αποδεικνύει λειτουργικότητα σε επίπεδο end-to-end (αισθητήρες → απόφαση → αντλία → καταγραφή → dashboard) και προσφέρει σαφή ορατότητα σε κατάσταση, ιστορικό και κατανάλωση. Η προσέγγιση "ποτίζω με στόχο υγρασίας (hysteresis) και προστασίες (cooldown, max runtime)" κρίνεται καταλληλότερη από ένα αυστηρό πρόγραμμα ημερών, ειδικά για φυτά εσωτερικού χώρου, καθώς μειώνει τόσο την πιθανότητα υπερ-ποτίσματος όσο και την παρατεταμένη ξηρασία.

Συμπεράσματα

Η υλοποίηση του ESP32 Smart Watering System καταδεικνύει ότι ένα χαμηλού κόστους IoT σύστημα μπορεί να επιτύχει αξιόπιστη παρακολούθηση και ελεγχόμενο πότισμα με μετρήσιμα αποτελέσματα. Η ολοκληρωμένη ροή **αισθητήρες → λογική απόφασης → ενεργοποίηση αντλίας → καταγραφή/οπτικοποίηση** επιβεβαιώνει ότι ο αυτοματισμός δεν παραμένει "θεωρητικός", αλλά αποτυπώνεται έμπρακτα στη συμπεριφορά του εδάφους, όπως φαίνεται από τις μεταβολές της υγρασίας σε πραγματικό χρόνο και τη συσχέτισή τους με τα συμβάντα ποτίσματος.

Κρίσιμο πλεονέκτημα του πρωτοτύπου είναι ότι δεν περιορίζεται σε μια δυαδική ένδειξη "πότισε/δεν πότισε", αλλά παρέχει **διαφάνεια και ιχνηλασιμότητα** μέσω ιστορικών μετρήσεων, πλήθους ποτισμάτων και καταγεγραμμένου όγκου νερού (mL). Με τον τρόπο αυτό, η λειτουργία του συστήματος αξιολογείται αντικειμενικά τόσο ως προς την ορθότητα των αποφάσεων όσο και ως προς την αποδοτικότητα χρήσης νερού, επιτρέποντας στον χρήστη να συγκρίνει παραμέτρους (thresholds, cooldowns, δοσολογία) και να καταλήξει σε καταλληλότερη ρύθμιση για διαφορετικά φυτά και υποστρώματα.

Η υγρασία εδάφους παραμένει ο βασικός δείκτης ανάγκης άρδευσης, ωστόσο η ενσωμάτωση καιρικών παραμέτρων (θερμοκρασία, ατμοσφαιρική υγρασία και ενδείξεις βροχής) προσφέρει σημαντικό πλαίσιο ερμηνείας και ανοίγει τον δρόμο για πιο "έξυπνες" αποφάσεις, όπως αναβολή ποτίσματος όταν αναμένεται βροχόπτωση ή δυναμική προσαρμογή ορίων. Ακόμη και όταν οι ενδείξεις αυτές χρησιμοποιούνται κυρίως ενημερωτικά, αποδεικνύονται χρήσιμες για τη μελλοντική εξέλιξη της λογικής ελέγχου.

Από την πρακτική παρατήρηση προκύπτει ότι η στρατηγική "λίγο και συνέχεια" δεν είναι πάντα βέλτιστη για φυτά όπως το σπαθίφυλλο, καθώς μπορεί να οδηγεί σε επίμονα νωπή επιφάνεια και αυξημένο κίνδυνο μυκήτων/σήψης, χωρίς να εξασφαλίζει ομοιόμορφη ενυδάτωση της ριζόμπαλας. Πιο κατάλληλη προσέγγιση είναι ο έλεγχος **στόχου υγρασίας (band control / hysteresis)** σε συνδυασμό με μηχανισμούς προστασίας (cooldown, max runtime), ώστε να αποφεύγονται τόσο ψευδείς ενεργοποιήσεις από μεταβατικές/θορυβώδεις μετρήσεις όσο και σενάρια υπερποτίσματος λόγω αστοχίας.

Το σύστημα ανέδειξε επίσης ένα κλασικό πρακτικό ζήτημα των IoT εφαρμογών: σε πραγματικές συνθήκες οι αισθητήρες εμφανίζουν θόρυβο, καθυστερήσεις και αποκλίσεις που επηρεάζονται από τη θέση τους και την ανομοιογένεια του υποστρώματος. Η αξιοπιστία βελτιώνεται ουσιαστικά όταν υιοθετούνται τεχνικές όπως φιλτράρισμα (moving average/median), απόρριψη outliers και λογική χρονικών παραθύρων (π.χ. αποφυγή αποφάσεων αμέσως μετά από πότισμα).

Τέλος, το πρωτότυπο αποτελεί ισχυρή βάση για περαιτέρω ωρίμανση. Ενδεικτικές επεκτάσεις περιλαμβάνουν βελτίωση της μέτρησης/ορίων στάθμης δοχείου, εφαρμογή ποτίσματος σε παλμούς (pulse + soak) για καλύτερη απορρόφηση, καθώς και alerts/διαγνωστικούς μηχανισμούς σε ανωμαλίες (π.χ. πότισμα χωρίς αύξηση υγρασίας, πιθανή διαρροή, άδειο δοχείο ή αστοχία αντλίας). Σε πιο προχωρημένο στάδιο, μπορούν να ενσωματωθούν κανόνες προβλεπτικής λειτουργίας βάσει ιστορικών δεδομένων και καιρού, με στόχο μεγαλύτερη εξοικονόμηση νερού και βελτιστοποίηση της φροντίδας.

Συνολικά, επιβεβαιώνεται ότι ο συνδυασμός αισθητήρων, αυτοματισμών, προστασιών και οπτικοποίησης μπορεί να μετατρέψει το πότισμα σε διαδικασία **ελεγχόμενη, μετρήσιμη και επεκτάσιμη**, προσφέροντας καλύτερη επίβλεψη και πιο συνεπή φροντίδα των φυτών.

Βιβλιογραφία

[1] Espressif Systems, “Welcome to ESP32 Arduino Core’s documentation.” [Online]. Available: https://docs.espressif.com/projects/arduino-esp32/

[2] Espressif Systems, “Getting Started — Arduino-ESP32 documentation.”[Online]. Available: https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/getting_started.html

[3] Espressif Systems, “Wi-Fi Driver — ESP-IDF Programming Guide (ESP32-S3).” [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/api-guides/wifi.html

[4] Espressif Systems, “ESP32-S3 Technical Reference Manual.” [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3_technical_reference_manual_en.pdf

[5] OASIS, MQTT Version 3.1.1, OASIS Standard, 29 Oct. 2014. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

[6] Eclipse Mosquitto Project, “Eclipse Mosquitto — An open source MQTT broker.” [Online]. Available: https://mosquitto.org/

[7] Eclipse Mosquitto Project, “Mosquitto documentation.” [Online]. Available: https://mosquitto.org/documentation/

[8] Eclipse Mosquitto Project, “mqtt(7) — MQTT description (man page).” [Online]. Available: https://mosquitto.org/man/mqtt-7.html

[9] Arduino, “PubSubClient (Arduino library documentation).” [Online]. Available: https://docs.arduino.cc/libraries/pubsubclient/

[10] N. O’Leary (knolleary), “PubSubClient: Arduino Client for MQTT (GitHub repository).” [Online]. Available: https://github.com/knolleary/pubsubclient

[11] me-no-dev, “ESPAsyncWebServer: Async Web Server for ESP8266 and ESP32 (GitHub repository).” [Online]. Available: https://github.com/me-no-dev/ESPAsyncWebServer

[12] Arduino, “LiquidCrystal — Arduino Reference.” [Online]. Available: https://www.arduino.cc/en/Reference/LiquidCrystal

[13] Arduino, “LiquidCrystal I2C (Arduino library documentation).” [Online]. Available: https://docs.arduino.cc/libraries/liquidcrystal-i2c/

[14] InfluxData, “InfluxDB OSS v1 Documentation.” [Online]. Available: https://docs.influxdata.com/influxdb/v1/

[15] InfluxData, “Explore data using InfluxQL (InfluxDB OSS v1).” [Online]. Available: https://docs.influxdata.com/influxdb/v1/query_language/explore-data/

[16] InfluxData, “InfluxDB line protocol tutorial (InfluxDB OSS v1).” [Online]. Available: https://docs.influxdata.com/influxdb/v1/write_protocols/line_protocol_tutorial/

[17] Grafana Labs, “InfluxDB data source — Grafana documentation.” [Online]. Available: https://grafana.com/docs/grafana/latest/datasources/influxdb/

[18] Grafana Labs, “InfluxDB query editor — Grafana documentation.” [Online]. Available: https://grafana.com/docs/grafana/latest/datasources/influxdb/query-editor/

[19] OpenWeather, “5 Day / 3 Hour Forecast.” [Online]. Available: https://openweathermap.org/forecast5

[20] OpenWeather, “One Call API 3.0.” [Online]. Available: https://openweathermap.org/api/one-call-3

[21] Raspberry Pi Ltd., “Raspberry Pi Zero 2 W Product Brief.” [Online]. Available: https://datasheets.raspberrypi.com/rpizero2/raspberry-pi-zero-2-w-product-brief.pdf

[22] Adafruit Industries, “YF-S201 Water Flow Sensor (C898+) Datasheet.” [Online]. Available: https://cdn-shop.adafruit.com/product-files/828/C898%2Bdatasheet.pdf

[23] R. J. Henny & J. Chen, “Florida Foliage House Plant Care: Spathiphyllum,” UF/IFAS Extension (EDIS EP477). [Online]. Available: https://edis.ifas.ufl.edu/publication/EP477

[24] Clemson University, HGIC, “Peace Lily (Spathiphyllum).” [Online]. Available: https://hgic.clemson.edu/factsheet/peace-lily/

[25] Royal Horticultural Society (RHS), “How to grow peace lilies.” [Online]. Available: https://www.rhs.org.uk/plants/peace-lilies/how-to-grow-peace-lilies

[26] A. A. Abdelmoneim, C. M. Al Kalaany, R. Khadra, B. Derardja, & G. Dragonetti, “Calibration of Low-Cost Capacitive Soil Moisture Sensors for Irrigation Management Applications,” Sensors, vol. 25, no. 2, Art. 343, 2025. doi:10.3390/s25020343

[27] I. M. Kulmány, Á. Bede-Fazekas, A. Beslin, Z. Giczi, G. Milics, & B. Kovács, “Calibration of an Arduino-based low-cost capacitive soil moisture sensor for smart agriculture,” Journal of Hydrology and Hydromechanics, vol. 70, no. 3, pp. 330–340, 2022. doi:10.2478/johh-2022-0014

[28] S. Darshna, T. Sangavi, S. Mohan, A. Soundharya, & S. Desikan, “Smart Irrigation System,” *IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)*, vol. 10, no. 3, Ver. II, pp. 32–36, May–Jun. 2015. doi:10.9790/2834-10323236. [Online]. Available: https://www.iosrjournals.org/iosr-jece/papers/Vol.%2010%20Issue%203/Version-2/F010323236.pdf

[29] A. Asokan, T. Singh, V. P. Gaur, S. Vasanthadev Suryakala, & A. Anilet Bala, “Smart Irrigation System with Weather Forecast Integration,” in *International Conference on Information and Communication Technology for Intelligent Systems*, Singapore: Springer Nature Singapore, 2025, pp. 565–577. doi:10.1007/978-981-96-8796-1_50.

[30] P. Tiwari, S. Verma, N. Singh, Y. Arora, P. Diwan, & G. Kumar, “Weather-Based Smart IoT Irrigation System with AI-Driven Decision Making,” *2025 International Conference on Intelligent and Secure Engineering Solutions (CISES)*, Greater Noida Gautam Budh Nagar, India, 2025, pp. 1128–1135. doi:10.1109/CISES66934.2025.11265581.

[31] I. R. Biju & S. Atuba, “Weather Integrated SMART Irrigation System,” *2025 IEEE International Conference on High Performance Computing and Communications (HPCC)*, Exeter, United Kingdom, 2025, pp. 1187–1193. doi:10.1109/HPCC67675.2025.00169.