

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
«IoT and DLT Integration for Environmental
Monitoring»



Του φοιτητή
Κωνσταντινίδη Βασίλειου
Αρ. Μητρώου: 2019/089

Επιβλέπων
Ονοματεπώνυμο
Αγγελόπουλος Μάριος
Βαθμίδα Επίκουρος Καθηγητής

Ημερομηνία 31/8/2026

Τίτλος Δ.Ε. IoT and DLT Integration for Environmental Monitoring

Κωδικός Δ.Ε. 25329

Ονοματεπώνυμο φοιτητή Κωνσταντίνιδης Βασίλειος

Ονοματεπώνυμο εισηγητή Αγγελόπουλος Μάριος

Ημερομηνία ανάληψης Δ.Ε. 17/10/2025

Ημερομηνία περάτωσης Δ.Ε. 31/08/2026

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

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

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

Πρόλογος

Η επιλογή του συγκεκριμένου θέματος προέκυψε από το ενδιαφέρον μου για τις σύγχρονες τεχνολογίες του Διαδικτύου των Πραγμάτων (IoT) και των Distributed Ledger Technologies (DLT), καθώς και από την επιθυμία μου να ασχοληθώ με ένα αντικείμενο που συνδυάζει διαφορετικούς τομείς της πληροφορικής και των ηλεκτρονικών συστημάτων.

Κατά την εκπόνηση της διπλωματικής εργασίας είχα την ευκαιρία να μελετήσω και να εφαρμόσω στην πράξη τεχνολογίες όπως το MQTT, η SQLite, κρυπτογραφικές συναρτήσεις κατακερματισμού, όπως η SHA-256, και το δίκτυο IOTA Shimmer. Παράλληλα, η ανάπτυξη του συστήματος από το αρχικό στάδιο της προσομοίωσης των αισθητήρων μέχρι την ολοκληρωμένη διασύνδεση με το Shimmer και τον έλεγχο της ακεραιότητας των δεδομένων μου έδωσε τη δυνατότητα να κατανοήσω καλύτερα τον τρόπο με τον οποίο μπορούν να συνδυαστούν διαφορετικά υποσυστήματα σε μία ενιαία εφαρμογή.

Η διαδικασία αυτή αποτέλεσε μια ιδιαίτερα χρήσιμη εμπειρία, καθώς πέρα από τη θεωρητική μελέτη απαιτήθηκε πρακτική εφαρμογή, δοκιμή και αντιμετώπιση προβλημάτων σε κάθε στάδιο της υλοποίησης. Θεωρώ ότι η εργασία αυτή συνέβαλε σημαντικά στην απόκτηση γνώσεων και εμπειρίας που μπορούν να αξιοποιηθούν σε μελλοντικές εφαρμογές και έργα.

Περίληψη

Η διασφάλιση της ακεραιότητας των δεδομένων που παράγονται από συστήματα Internet of Things (IoT) αποτελεί σημαντική πρόκληση, ιδιαίτερα σε εφαρμογές περιβαλλοντικής παρακολούθησης όπου οι μετρήσεις συλλέγονται και αποθηκεύονται συνεχώς. Η χρήση τεχνολογιών Distributed Ledger Technology (DLT) μπορεί να προσφέρει ένα ανεξάρτητο σημείο αναφοράς για την επαλήθευση της ακεραιότητας των δεδομένων, χωρίς την ανάγκη αποθήκευσης ολόκληρου του συνόλου των μετρήσεων στο DLT. Στην παρούσα εργασία σχεδιάζεται και υλοποιείται ένα ολοκληρωμένο σύστημα περιβαλλοντικής παρακολούθησης που συνδυάζει IoT και DLT, με έμφαση στη συλλογή, μετάδοση, αποθήκευση, κρυπτογραφική αποτύπωση και επαλήθευση των δεδομένων.

Η υλοποίηση βασίζεται σε τρεις προσομοιωμένους αισθητήρες θερμοκρασίας, υγρασίας και ποιότητας αέρα. Οι μετρήσεις μεταδίδονται μέσω MQTT σε έναν Gateway, ο οποίος τις συλλέγει και τις οργανώνει σε ολοκληρωμένους κύκλους δεδομένων. Οι κύκλοι αποθηκεύονται σε βάση SQLite και για κάθε κύκλο δημιουργείται κρυπτογραφικό αποτύπωμα SHA-256. Το hash δημοσιεύεται στο δίκτυο IOTA Shimmer μέσω Tagged Data block και το αντίστοιχο Block ID αποθηκεύεται μαζί με την τοπική καταχώριση.

Η αξιολόγηση περιλαμβάνει δοκιμές της παραγωγής και μετάδοσης των μετρήσεων, της αποθήκευσης και κρυπτογραφικής επεξεργασίας, της δημοσίευσης στο Shimmer και της επαλήθευσης της ακεραιότητας. Επιπλέον, πραγματοποιείται ελεγχόμενη χειροκίνητη αλλοίωση δεδομένων, ώστε να αξιολογηθεί η δυνατότητα εντοπισμού μεταβολών μέσω της σύγκρισης των υπολογισμένων hashes με τις αρχικές τιμές αναφοράς. Παράλληλα, αναπτύσσεται γραφικό περιβάλλον Launcher για τη διαχείριση των βασικών διεργασιών και εκτελέσιμη έκδοση της εφαρμογής για Windows.

Τα αποτελέσματα επιβεβαιώνουν την ορθή λειτουργία της βασικής ροής δεδομένων, τη σύνδεση των τοπικών καταχωρίσεων με τις αντίστοιχες εγγραφές στο Shimmer και τη δυνατότητα εντοπισμού σκόπιμων μεταβολών στα αποθηκευμένα δεδομένα. Συνολικά, η εργασία αναδεικνύει τη δυνατότητα συνδυασμού IoT και DLT για εφαρμογές περιβαλλοντικής παρακολούθησης με μηχανισμούς επαλήθευσης ακεραιότητας, ενώ η χρήση πραγματικού υλικού αισθητήρων και η αξιολόγηση σε μεγαλύτερη κλίμακα αποτελούν κατευθύνσεις μελλοντικής επέκτασης.

Λέξεις-κλειδιά: Internet of Things, Distributed Ledger Technology, περιβαλλοντική παρακολούθηση, MQTT, SQLite, SHA-256, IOTA Shimmer, ακεραιότητα δεδομένων, επαλήθευση δεδομένων, ανίχνευση αλλοίωσης

«IoT and DLT Integration for Environmental Monitoring»

«Konstantinidis Vasileios»

Abstract

Ensuring the integrity of data generated by Internet of Things (IoT) systems is an important challenge, particularly in environmental monitoring applications where measurements are continuously collected and stored. Distributed Ledger Technology (DLT) can provide an independent reference for verifying data integrity without requiring the complete set of measurements to be stored on the distributed ledger. This thesis presents the design and implementation of an integrated environmental monitoring system that combines IoT and DLT technologies, with emphasis on data collection, transmission, storage, cryptographic fingerprinting and integrity verification.

The implementation is based on three simulated sensors for temperature, humidity and air quality monitoring. Measurements are transmitted through MQTT to a Gateway, which collects and organizes them into complete data cycles. The cycles are stored in an SQLite database, while a SHA-256 cryptographic hash is generated for each cycle. The hash is published to the IOTA Shimmer network through a Tagged Data block, and the corresponding Block ID is stored together with the local record.

The evaluation includes tests of measurement generation and transmission, data storage and cryptographic processing, hash publication to Shimmer, and integrity verification. In addition, a controlled manual data tampering test is performed to evaluate the ability to detect modifications by comparing recalculated hashes with the original reference values. A graphical Launcher is also developed to manage the main system processes, together with an executable version of the application for Windows.

The results confirm the correct operation of the main data flow, the association of local records with their corresponding Shimmer entries, and the ability to detect deliberate modifications to stored data. Overall, the thesis demonstrates the feasibility of combining IoT and DLT for environmental monitoring applications requiring integrity verification, while the use of real sensor hardware and evaluation at larger scale remain directions for future work.

Keywords: Internet of Things, Distributed Ledger Technology, environmental monitoring, MQTT, SQLite, SHA-256, IOTA Shimmer, data integrity, data verification, tampering detection

Ευχαριστίες

Στην οικογένεια και σε όλους τους φίλους που μου στάθηκαν σε αυτό το ταξίδι.

Περιεχόμενα

Πρόλογος.....	3
Περίληψη.....	4
Abstract.....	5
Ευχαριστίες.....	6
Περιεχόμενα.....	7
Κατάλογος Σχημάτων.....	12
Κατάλογος Πινάκων.....	12
Συντομογραφίες.....	13
Κεφάλαιο 1ο: Εισαγωγή.....	13
1.1 Περιγραφή του Θέματος.....	13
1.2 Κίνητρο και πεδίο εφαρμογής.....	13
1.3 Στόχοι της εργασίας.....	14
1.4 Οργάνωση της εργασίας και όρια ευθύνης.....	15
1.5 Δομή της διπλωματικής εργασίας.....	15
Κεφάλαιο 2ο: Θεωρητικό Υπόβαθρο και Τεχνολογίες.....	16
2.1 Εισαγωγή.....	16
2.2 Internet of Things.....	16
2.2.1 IoT συσκευές και αισθητήρες.....	16
2.2.2 Επικοινωνία και ροή δεδομένων σε συστήματα IoT.....	17
2.3 Περιβαλλοντική παρακολούθηση.....	17
2.3.1 Βασικοί περιβαλλοντικοί παράμετροι.....	17
2.3.2 Θερμοκρασία και υγρασία.....	18
2.3.3 Ποιότητα αέρα.....	18
2.3.4 IoT στην περιβαλλοντική παρακολούθηση.....	18
2.4 MQTT.....	18
2.4.1 Αρχιτεκτονική publish/subscribe.....	19
2.4.2 Topics και μηνύματα.....	19
2.4.3 MQTT Broker.....	19
2.4.4 Quality of Service.....	19
2.4.5 MQTT σε εφαρμογές IoT.....	20
2.5 Κρυπτογραφικές συναρτήσεις κατακερματισμού.....	20
2.5.1 Hash και message digest.....	20
2.5.2 Βασικές ιδιότητες.....	20
2.5.3 SHA-256.....	20
2.5.4 Hash-based integrity verification.....	21
2.6 Distributed Ledger Technology.....	21
2.6.1 Έννοια και βασικά χαρακτηριστικά.....	21
2.6.2 Ακεραιότητα και αμεταβλητότητα καταχωρίσεων.....	21
2.6.3 DLT και blockchain.....	22
2.6.4 DLT ως μηχανισμός ανεξάρτητης αναφοράς.....	22
2.7 IOTA και Shimmer.....	22
2.7.1 Shimmer.....	22

2.7.2 Καταγραφή δεδομένων στο Shimmer.....	23
2.7.3 Transactions και αναγνωριστικά.....	23
2.7.4 Δημοσίευση κρυπτογραφικών αποτυπωμάτων.....	23
2.8 Ακεραιότητα δεδομένων και ανίχνευση αλλοίωσης.....	24
2.8.1 Έλεγχος ακεραιότητας μέσω hash.....	24
2.8.2 Ανίχνευση αλλοίωσης.....	24
2.8.3 Συνδυασμός τοπικής αποθήκευσης και DLT.....	25
2.9 Επίλογος.....	25
Κεφάλαιο 3ο: Ανάλυση Απαιτήσεων και Σχεδιασμός.....	26
3.1 Εισαγωγή.....	26
3.2 Ανάλυση απαιτήσεων.....	26
3.2.1 Λειτουργικές απαιτήσεις.....	26
3.2.2 Μη λειτουργικές απαιτήσεις.....	27
3.2.3 Περιορισμοί και παραδοχές.....	27
3.3 Συνολική αρχιτεκτονική του συστήματος.....	28
3.4 Σχεδιασμός της ροής δεδομένων.....	28
3.4.1 Παραγωγή περιβαλλοντικών μετρήσεων.....	28
3.4.2 Μετάδοση μέσω MQTT.....	29
3.4.3 Συλλογή και ενοποίηση στον Gateway.....	29
3.4.4 Ολοκλήρωση κύκλου δεδομένων.....	29
3.5 Σχεδιασμός αποθήκευσης και ακεραιότητας.....	29
3.5.1 Δομή δεδομένων και κύκλου.....	29
3.5.2 Τοπική αποθήκευση.....	30
3.5.3 Δημιουργία κρυπτογραφικού αποτυπώματος.....	30
3.5.4 Συσχέτιση hash και Transaction ID.....	30
3.6 Σχεδιασμός διασύνδεσης με το DLT.....	30
3.7 Σχεδιασμός μηχανισμών επαλήθευσης.....	32
3.7.1 Έλεγχος τοπικής ακεραιότητας.....	32
3.7.2 Επαλήθευση μέσω DLT.....	32
3.7.3 Σχεδιασμός tampering tests.....	32
3.8 Σχεδιασμός του Launcher.....	33
3.9 Συνολική ροή λειτουργίας του συστήματος.....	33
3.10 Επίλογος.....	34
Κεφάλαιο 4ο: Υλοποίηση του Συστήματος.....	36
4.1 Εισαγωγή.....	36
4.2 Περιβάλλον ανάπτυξης και χρησιμοποιούμενα εργαλεία.....	36
4.3 Υλοποίηση της προσομοίωσης των αισθητήρων.....	37
4.3.1 Αισθητήρας θερμοκρασίας.....	37
4.3.2 Αισθητήρας υγρασίας.....	38
4.3.3 Αισθητήρας ποιότητας αέρα.....	38
4.3.4 Παραγωγή και χρονική συμπεριφορά των μετρήσεων.....	39
4.3.5 Μορφή των παραγόμενων μηνυμάτων.....	39
4.4 Υλοποίηση της επικοινωνίας MQTT.....	40
4.4.1 MQTT Publisher.....	40

4.4.2 Topics και μορφή μηνυμάτων.....	40
4.4.3 MQTT Broker.....	41
4.4.4 Περιοδική δημοσίευση μετρήσεων.....	41
4.5 Υλοποίηση του Gateway.....	41
4.5.1 Αρχικοποίηση και σύνδεση στον MQTT Broker.....	41
4.5.2 Λήψη και επεξεργασία μηνυμάτων.....	41
4.5.3 Διαχείριση των τρεχουσών μετρήσεων.....	42
4.5.4 Δημιουργία ολοκληρωμένου κύκλου.....	42
4.5.5 Δημιουργία SHA-256 hash.....	42
4.5.6 Δημοσίευση του hash στο Shimmer.....	42
4.5.7 Αποθήκευση του Block ID.....	42
4.5.8 Αποθήκευση του ολοκληρωμένου κύκλου.....	43
4.6 Υλοποίηση της βάσης δεδομένων SQLite.....	43
4.6.1 Αρχικοποίηση της βάσης.....	43
4.6.2 Δομή του πίνακα environment_data.....	43
4.6.3 Αποθήκευση κύκλων.....	44
4.6.4 Ανάκτηση του τελευταίου cycle_id.....	44
4.6.5 Μόνιμη διατήρηση των δεδομένων.....	44
4.7 Υλοποίηση της κρυπτογραφικής επεξεργασίας.....	45
4.7.1 Κανονικοποίηση των δεδομένων.....	45
4.7.2 Δημιουργία deterministic JSON.....	45
4.7.3 Υπολογισμός SHA-256.....	45
4.7.4 Χρήση του hash ως αποτυπώματος του κύκλου.....	45
4.8 Υλοποίηση της διασύνδεσης με το Shimmer DLT.....	46
4.8.1 Αρχικοποίηση του Shimmer client.....	46
4.8.2 Έλεγχος σύνδεσης και κατάστασης του δικτύου.....	46
4.8.3 Δημιουργία Tagged Data block.....	46
4.8.4 Επιστροφή και αποθήκευση του Block ID.....	47
4.8.5 Ανάκτηση του block για επαλήθευση.....	47
4.8.6 Ρόλος της διασύνδεσης στην τελική αρχιτεκτονική.....	47
4.9 Υλοποίηση των μηχανισμών επαλήθευσης.....	47
4.9.1 Ανάκτηση των αποθηκευμένων κύκλων.....	48
4.9.2 Επανυπολογισμός του hash.....	48
4.9.3 Τοπική επαλήθευση μέσω SQLite.....	48
4.9.4 Ανάκτηση του block από το Shimmer.....	48
4.9.5 Επαλήθευση μέσω Shimmer.....	48
4.9.6 Συνολική κατάσταση επαλήθευσης.....	49
4.10 Εκτέλεση δοκιμής σκόπιμης αλλοίωσης.....	49
4.10.1 Σκόπιμη τροποποίηση δεδομένων.....	49
4.10.2 Έλεγχος της αλλοίωσης.....	49
4.10.3 Επαναφορά των δεδομένων.....	49
4.10.4 Επανεπαλήθευση.....	50
4.11 Υλοποίηση του Launcher.....	50
4.11.1 Γραφικό περιβάλλον.....	50

4.11.2 Process Manager.....	50
4.11.3 Έλεγχος κατάστασης των components.....	50
4.11.4 Start Project.....	51
4.11.5 Μηχανισμός συγχρονισμού Gateway–Publisher.....	51
4.11.6 Stop Project.....	51
4.11.7 Activity Log και ενημέρωση κατάστασης.....	51
4.11.8 Database και Shimmer εργαλεία του Launcher.....	52
4.12 Δημιουργία της εκτελέσιμης έκδοσης.....	52
4.12.1 Προετοιμασία της εφαρμογής για PyInstaller.....	52
4.12.2 Ενσωμάτωση των resources.....	52
4.12.3 Δημιουργία του executable.....	52
4.12.4 Εκτέλεση της εφαρμογής σε δεύτερο υπολογιστή.....	53
4.13 Συνολική ενσωμάτωση των υποσυστημάτων.....	53
4.14 Επίλογος.....	54
Κεφάλαιο 5ο: Πειραματική αξιολόγηση και αποτελέσματα.....	55
5.1 Εισαγωγή.....	55
5.2 Περιβάλλον και διαδικασία δοκιμών.....	56
5.3 Έλεγχος λειτουργίας των αισθητήρων και παραγωγής δεδομένων.....	56
5.4 Έλεγχος επικοινωνίας MQTT και λειτουργίας του Gateway.....	57
5.5 Έλεγχος δημιουργίας κύκλων, αποθήκευσης και hashing.....	58
5.6 Έλεγχος δημοσίευσης των hashes στο Shimmer.....	59
5.7 Έλεγχος ακεραιότητας των δεδομένων.....	61
5.8 Δοκιμή σκόπιμης αλλοίωσης.....	62
5.9 Έλεγχος του Launcher και της συνολικής λειτουργίας.....	63
5.10 Συγκεντρωτική αξιολόγηση των απαιτήσεων.....	65
5.11 Επίλογος.....	66
Κεφάλαιο 6ο: Συμπεράσματα και μελλοντική εργασία.....	67
6.1 Συμπεράσματα.....	67
6.2 Αξιολόγηση της προσέγγισης.....	67
6.3 Περιορισμοί της υλοποίησης.....	68
6.4 Μελλοντικές επεκτάσεις.....	68
ΒΙΒΛΙΟΓΡΑΦΙΑ.....	70
ΠΑΡΑΡΤΗΜΑ Α : Κώδικας Αισθητήρων.....	72
A.1 Αισθητήρας Θερμοκρασίας.....	72
A.2 Αισθητήρας Υγρασίας.....	74
A.3 Αισθητήρας Ποιότητας Αέρα.....	76
ΠΑΡΑΡΤΗΜΑ Β : Κώδικας Gateway.....	79
ΠΑΡΑΡΤΗΜΑ C : Κώδικας Publisher.....	85
ΠΑΡΑΡΤΗΜΑ D : Κώδικας Κρυπτογραφικής επεξεργασίας και επαλήθευσης.....	89
D.1 Hashing.py.....	89
D.2 Verify_Integrity.py.....	91
ΠΑΡΑΡΤΗΜΑ E : Κώδικας Shimmer Client.....	96
ΠΑΡΑΡΤΗΜΑ F : Κώδικας Βάσης Δεδομένων.....	99
ΠΑΡΑΡΤΗΜΑ G : Κώδικας Launcher.....	103

G.1 main.py.....	103
G.2 config.py.....	104
G.3 process_manager.py.....	106
G.4 ui.py.....	111

Κατάλογος Σχημάτων

Σχήμα 3.1: Συνολική αρχιτεκτονική.....	28
Σχήμα 3.2: Ροή δημοσίευσης hash στο DLT και ανάκτησης Block ID.....	31
Σχήμα 3.3: Συνολική ροή λειτουργίας του συστήματος.....	32
Σχήμα 4.1: Launcher UI.....	54
Σχήμα 5.1: Publisher.....	57
Σχήμα 5.2: Project Running.....	58
Σχήμα 5.3: Database Viewer.....	59
Σχήμα 5.4: Shimmer Explorer.....	60
Σχήμα 5.5: Database Viewer first record.....	61
Σχήμα 5.6: Integrity Verification.....	62
Σχήμα 5.7: Tampering test.....	63
Σχήμα 5.8: Project Standby.....	64

Κατάλογος Πινάκων

Πίνακας 3.1: Λειτουργικές απαιτήσεις του συστήματος.....	26
Πίνακας 4.1: Δομή του πίνακα environment_data.....	43
Πίνακας 5.1: Παρατηρούμενο εύρος τιμών στους 11 εξεταζόμενους κύκλους.....	57
Πίνακας 5.2: Αντιστοίχιση λειτουργικών απαιτήσεων και αποτελεσμάτων.....	65

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
IoT	Internet of Things
DLT	Distributed Ledger Technologies
MQTT	Message Queuing Telemetry Transport
SHA-256	Secure Hash Algorithm 256-bit
QoS	Quality of Service
NIST	National Institute of Standards and Technology
DAG	Directed Acyclic Graph
ID	Identifier
SDK	Software Development Kit
API	Application Programming Interface
AQI	Air Quality Index
UTC	Coordinated Universal Time
JSON	JavaScript Object Notation
ISO 8601	International Organization for Standardization 8601

Κεφάλαιο 1ο: Εισαγωγή

1.1 Περιγραφή του Θέματος

Η παρούσα διπλωματική εργασία αφορά τον σχεδιασμό και την υλοποίηση ενός συστήματος περιβαλλοντικής παρακολούθησης που συνδυάζει τεχνολογίες Internet of Things (IoT) και Distributed Ledger Technology (DLT). Βασική ιδέα της εργασίας είναι η συλλογή δεδομένων από αισθητήρες, η μετάδοσή τους σε ένα κεντρικό σημείο επεξεργασίας και η αξιοποίηση κρυπτογραφικών μηχανισμών και τεχνολογιών καταναμημένου καθολικού για την ενίσχυση της ακεραιότητας και της δυνατότητας επαλήθευσης των δεδομένων.

Για τις ανάγκες της υλοποίησης χρησιμοποιήθηκαν προσομοιωμένοι αισθητήρες θερμοκρασίας, υγρασίας και ποιότητας αέρα. Οι αισθητήρες παράγουν περιβαλλοντικές μετρήσεις σε πραγματικό χρόνο και τις μεταδίδουν μέσω του πρωτοκόλλου MQTT. Οι μετρήσεις συλλέγονται από έναν Gateway, ο οποίος τις οργανώνει σε ολοκληρωμένους κύκλους δεδομένων και τις προωθεί για αποθήκευση σε βάση δεδομένων SQLite.

Για κάθε ολοκληρωμένο κύκλο υπολογίζεται ένα κρυπτογραφικό αποτύπωμα με χρήση του αλγορίθμου SHA-256. Το σύνολο των μετρήσεων παραμένει αποθηκευμένο τοπικά, ενώ το παραγόμενο hash δημοσιεύεται στο δίκτυο IOTA Shimmer μέσω καταχώρησης δεδομένων. Για κάθε δημοσίευση διατηρείται το αντίστοιχο Block ID, ώστε να είναι δυνατή η σύνδεση της τοπικά αποθηκευμένης πληροφορίας με τη σχετική καταχώρηση στο DLT. Η πραγματική υλοποίηση χρησιμοποιεί Tagged Data block για τη δημοσίευση του hash στο δίκτυο.

Παράλληλα, έχει υλοποιηθεί μηχανισμός ελέγχου ακεραιότητας, ο οποίος συγκρίνει το hash που υπολογίζεται εκ νέου από τα τοπικά αποθηκευμένα δεδομένα με το hash που είναι καταγεγραμμένο στη βάση και με το αντίστοιχο hash που ανακτάται από το Shimmer. Με αυτόν τον τρόπο είναι δυνατός ο εντοπισμός αποκλίσεων και πιθανής αλλοίωσης των δεδομένων.

Η συνολική εφαρμογή περιλαμβάνει επίσης γραφικό περιβάλλον διαχείρισης, μέσω του οποίου ελέγχονται οι επιμέρους διεργασίες του συστήματος και η σειρά εκκίνησής τους. Με αυτόν τον τρόπο, τα διαφορετικά υποσυστήματα ενσωματώνονται σε μία ενιαία εφαρμογή, η οποία μπορεί να χρησιμοποιηθεί για την εκτέλεση, την παρακολούθηση και τον έλεγχο της συνολικής διαδικασίας.

1.2 Κίνητρο και πεδίο εφαρμογής

Η συνεχής παραγωγή δεδομένων από συσκευές και αισθητήρες αποτελεί βασικό χαρακτηριστικό των σύγχρονων εφαρμογών Internet of Things. Σε ένα σύστημα περιβαλλοντικής παρακολούθησης, οι μετρήσεις μπορούν να παράγονται σε τακτά χρονικά διαστήματα και να χρησιμοποιούνται για την παρακολούθηση της κατάστασης ενός χώρου ή ενός περιβάλλοντος. Η χρησιμότητα τέτοιων συστημάτων δεν περιορίζεται μόνο στη συλλογή και αποθήκευση των μετρήσεων, αλλά επεκτείνεται και στην ανάγκη να μπορεί να επιβεβαιωθεί ότι τα δεδομένα που αποθηκεύτηκαν δεν έχουν τροποποιηθεί εκ των υστέρων.

Το κίνητρο της παρούσας εργασίας προέκυψε από την προσπάθεια συνδυασμού δύο διαφορετικών τεχνολογικών προσεγγίσεων. Από τη μία πλευρά, το IoT παρέχει τον μηχανισμό συλλογής και

μετάδοσης των δεδομένων, ενώ από την άλλη η Distributed Ledger Technology μπορεί να χρησιμοποιηθεί ως ανεξάρτητο σημείο αναφοράς για την επαλήθευση της ακεραιότητάς τους. Η χρήση ενός κρυπτογραφικού αποτυπώματος επιτρέπει τη σύνδεση των τοπικά αποθηκευμένων δεδομένων με μια καταχώρηση στο DLT, χωρίς να απαιτείται η μεταφορά και αποθήκευση ολόκληρου του συνόλου των μετρήσεων στο δίκτυο.

Το πεδίο εφαρμογής της εργασίας επικεντρώνεται στην προσομοίωση ενός συστήματος περιβαλλοντικής παρακολούθησης και στην ολοκληρωμένη λειτουργία των επιμέρους υποσυστημάτων του. Συγκεκριμένα, περιλαμβάνει την παραγωγή δεδομένων θερμοκρασίας, υγρασίας και ποιότητας αέρα, τη μετάδοσή τους μέσω MQTT, τη συλλογή και οργάνωσή τους από τον Gateway, την αποθήκευση σε SQLite, τη δημιουργία SHA-256 hashes και την πραγματική δημοσίευσή τους στο δίκτυο IOTA Shimmer.

Παράλληλα, στο πεδίο εφαρμογής περιλαμβάνονται οι διαδικασίες επαλήθευσης της ακεραιότητας και οι δοκιμές σκόπιμης αλλοίωσης των αποθηκευμένων δεδομένων. Δεν αποτελεί αντικείμενο της εργασίας η χρήση πραγματικού υλικού αισθητήρων, καθώς το σύστημα υλοποιήθηκε εξ ολοκλήρου σε περιβάλλον προσομοίωσης. Επίσης, το DLT χρησιμοποιείται για την καταγραφή των κρυπτογραφικών αποτυπωμάτων και όχι για την αποθήκευση του πλήρους όγκου των περιβαλλοντικών μετρήσεων.

1.3 Στόχοι της εργασίας

Βασικός στόχος της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη ενός ολοκληρωμένου συστήματος που συνδυάζει τεχνολογίες Internet of Things (IoT) και Distributed Ledger Technology (DLT) για τη συλλογή, αποθήκευση και επαλήθευση της ακεραιότητας περιβαλλοντικών δεδομένων.

Για την επίτευξη του βασικού στόχου τέθηκαν οι ακόλουθοι επιμέρους στόχοι:

- Η δημιουργία προσομοιωμένων αισθητήρων θερμοκρασίας, υγρασίας και ποιότητας αέρα, οι οποίοι παράγουν περιβαλλοντικές μετρήσεις σε τακτά χρονικά διαστήματα,
- Η υλοποίηση μηχανισμού μετάδοσης των δεδομένων μέσω του πρωτοκόλλου MQTT,
- Η ανάπτυξη Gateway για τη συλλογή των επιμέρους μετρήσεων και τη δημιουργία ολοκληρωμένων κύκλων δεδομένων,
- Η αποθήκευση των κύκλων σε βάση δεδομένων SQLite,
- Η δημιουργία κρυπτογραφικού αποτυπώματος SHA-256 για κάθε ολοκληρωμένο κύκλο,
- Η πραγματική δημοσίευση των παραγόμενων hashes στο δίκτυο IOTA Shimmer μέσω Tagged Data block,
- Η αποθήκευση του αντίστοιχου Block ID και η σύνδεσή του με την τοπική καταχώριση,
- Η ανάπτυξη μηχανισμού επαλήθευσης της ακεραιότητας τόσο των τοπικά αποθηκευμένων δεδομένων όσο και της αντίστοιχης καταχώρισης στο Shimmer,
- Η υλοποίηση δοκιμών σκόπιμης αλλοίωσης των δεδομένων και η επιβεβαίωση της δυνατότητας εντοπισμού της μεταβολής,
- Η ανάπτυξη γραφικού περιβάλλοντος για τη διαχείριση των επιμέρους διεργασιών του συστήματος,
- Η δημιουργία εκτελέσιμης έκδοσης της εφαρμογής για περιβάλλον Windows.

Οι παραπάνω στόχοι καθορίζουν τα βασικά κριτήρια που χρησιμοποιούνται για την αξιολόγηση της υλοποίησης και των αποτελεσμάτων της εργασίας.

1.4 Οργάνωση της εργασίας και όρια ευθύνης

Η εκπόνηση της διπλωματικής εργασίας ακολούθησε μία διαδοχική πορεία, ξεκινώντας από τη μελέτη των τεχνολογιών που σχετίζονται με το αντικείμενο και καταλήγοντας στην ανάπτυξη και τον έλεγχο του τελικού συστήματος. Η θεωρητική μελέτη χρησιμοποιήθηκε ως βάση για τη λήψη των σχεδιαστικών αποφάσεων, ενώ η υλοποίηση προχώρησε σταδιακά, ώστε κάθε επιμέρους λειτουργία να μπορεί να ελεγχθεί πριν ενσωματωθεί στο συνολικό σύστημα.

Βασικό όριο της εργασίας είναι ότι η λειτουργία των αισθητήρων εξετάζεται σε επίπεδο προσομοίωσης. Με τον τρόπο αυτό δίνεται έμφαση στη ροή και στην επεξεργασία των δεδομένων, χωρίς η αξιολόγηση να επεκτείνεται σε ζητήματα που αφορούν την κατασκευή, την επιλογή ή τη βαθμονόμηση πραγματικού υλικού.

Επιπλέον, η εργασία εξετάζει τη χρήση του DLT ως μέρος ενός μηχανισμού ελέγχου ακεραιότητας και όχι ως γενικού συστήματος αποθήκευσης περιβαλλοντικών δεδομένων. Επομένως, η μελέτη επικεντρώνεται στη συγκεκριμένη περίπτωση χρήσης και στις απαιτήσεις της, ενώ ζητήματα μεγαλύτερης κλίμακας, όπως η ανάπτυξη πολυάριθμων κόμβων ή η αξιολόγηση ενός παραγωγικού δικτύου μεγάλης κλίμακας, δεν αποτελούν μέρος της παρούσας υλοποίησης.

Η οργάνωση αυτή επιτρέπει την ξεχωριστή εξέταση της θεωρίας, του σχεδιασμού, της υλοποίησης και της αξιολόγησης, ώστε τα αποτελέσματα κάθε σταδίου να μπορούν να αξιοποιηθούν στο επόμενο.

1.5 Δομή της διπλωματικής εργασίας

Η παρούσα διπλωματική εργασία οργανώνεται σε έξι κεφάλαια, τα οποία ακολουθούν τη διαδοχική πορεία από τη θεωρητική θεμελίωση και τον σχεδιασμό έως την υλοποίηση, την πειραματική αξιολόγηση και την εξαγωγή συμπερασμάτων.

Στο Κεφάλαιο 1 παρουσιάζεται το αντικείμενο της εργασίας, το κίνητρο και το πεδίο εφαρμογής, οι βασικοί στόχοι, καθώς και η οργάνωση και τα όρια της μελέτης.

Στο Κεφάλαιο 2 αναπτύσσεται το θεωρητικό υπόβαθρο και παρουσιάζονται οι βασικές τεχνολογίες που χρησιμοποιούνται στην εργασία. Εξετάζονται οι έννοιες του Internet of Things και της περιβαλλοντικής παρακολούθησης, το πρωτόκολλο MQTT, οι κρυπτογραφικές συναρτήσεις κατακερματισμού και ο αλγόριθμος SHA-256, καθώς και οι Distributed Ledger Technologies, η τεχνολογία IOTA και το δίκτυο Shimmer.

Στο Κεφάλαιο 3 παρουσιάζονται η ανάλυση των απαιτήσεων και ο σχεδιασμός του συστήματος. Καθορίζονται οι λειτουργικές και μη λειτουργικές απαιτήσεις, οι βασικές παραδοχές και περιορισμοί, η αρχιτεκτονική του συστήματος, η ροή των δεδομένων και η συνεργασία των επιμέρους υποσυστημάτων.

Στο Κεφάλαιο 4 περιγράφεται αναλυτικά η υλοποίηση του συστήματος και των επιμέρους components. Παρουσιάζονται η προσομοίωση των αισθητήρων, η επικοινωνία μέσω MQTT, η λειτουργία του Gateway, η τοπική αποθήκευση, ο υπολογισμός των hashes, η διασύνδεση με το Shimmer, ο μηχανισμός επαλήθευσης ακεραιότητας, καθώς και το γραφικό περιβάλλον Launcher.

Εισαγωγή

Στο Κεφάλαιο 5 παρουσιάζεται η πειραματική αξιολόγηση της τελικής υλοποίησης. Περιγράφονται το περιβάλλον και η διαδικασία δοκιμών και εξετάζονται η παραγωγή και μετάδοση των δεδομένων, η δημιουργία και αποθήκευση των κύκλων, η λειτουργία του SHA-256 και του Shimmer, η επαλήθευση της ακεραιότητας, καθώς και η δυνατότητα εντοπισμού σκόπιμης αλλοίωσης των αποθηκευμένων δεδομένων.

Τέλος, στο Κεφάλαιο 6 συνοψίζονται τα βασικά συμπεράσματα της εργασίας, αξιολογείται η συνολική προσέγγιση και παρουσιάζονται οι περιορισμοί της υλοποίησης και οι δυνατότητες μελλοντικής επέκτασης του συστήματος.

Κεφάλαιο 2ο: Θεωρητικό Υπόβαθρο και Τεχνολογίες

2.1 Εισαγωγή

Η ανάπτυξη ενός συστήματος που συνδυάζει τεχνολογίες Internet of Things (IoT) και Distributed Ledger Technology (DLT) απαιτεί τη συνδυασμένη κατανόηση διαφορετικών τεχνολογικών εννοιών. Η συλλογή δεδομένων από αισθητήρες, η μετάδοσή τους μέσω δικτύου, η αποθήκευση, η προστασία της ακεραιότητας και η καταγραφή κρυπτογραφικών αποτυπωμάτων σε ένα κατακερματισμένο σύστημα αποτελούν διακριτές λειτουργίες, οι οποίες πρέπει να συνεργάζονται με τρόπο συνεπή και αξιόπιστο.

Στο παρόν κεφάλαιο παρουσιάζονται οι βασικές έννοιες και τεχνολογίες που σχετίζονται με το αντικείμενο της εργασίας. Αρχικά εξετάζεται το Internet of Things και η χρήση του σε εφαρμογές περιβαλλοντικής παρακολούθησης. Στη συνέχεια παρουσιάζεται το πρωτόκολλο MQTT ως μηχανισμός επικοινωνίας μεταξύ δικτυωμένων συσκευών και εφαρμογών. Ακολουθεί η παρουσίαση των κρυπτογραφικών συναρτήσεων κατακερματισμού και του αλγορίθμου SHA-256, καθώς και η εισαγωγή στις Distributed Ledger Technologies. Τέλος, εξετάζονται η τεχνολογία IOTA, το δίκτυο Shimmer και η χρήση κρυπτογραφικών αποτυπωμάτων για τον έλεγχο της ακεραιότητας δεδομένων.

2.2 Internet of Things

Ο όρος Internet of Things αναφέρεται σε ένα σύνολο συσκευών και υπολογιστικών συστημάτων που μπορούν να αλληλεπιδρούν με το φυσικό περιβάλλον, να συλλέγουν δεδομένα και να επικοινωνούν μέσω δικτύου. Σύμφωνα με το National Institute of Standards and Technology (NIST), μία IoT συσκευή διαθέτει τουλάχιστον έναν αισθητήρα ή ενεργοποιητή για αλληλεπίδραση με το φυσικό περιβάλλον και τουλάχιστον μία διεπαφή δικτύου για επικοινωνία με τον ψηφιακό κόσμο [1].

Η βασική λειτουργία ενός συστήματος IoT μπορεί να περιγραφεί ως μία ακολουθία συλλογής, μετάδοσης, επεξεργασίας και αξιοποίησης δεδομένων [11]. Οι συσκευές που βρίσκονται κοντά στο φυσικό περιβάλλον λειτουργούν ως πηγή δεδομένων, ενώ άλλες συσκευές ή πληροφοριακά συστήματα αναλαμβάνουν την επεξεργασία και αποθήκευση της πληροφορίας.

Η αξία του IoT δεν περιορίζεται στη δυνατότητα σύνδεσης συσκευών στο διαδίκτυο. Η δικτύωση των συσκευών επιτρέπει τη δημιουργία συνεχούς ροής δεδομένων, γεγονός που καθιστά δυνατή την παρακολούθηση φυσικών μεγεθών σε πραγματικό ή σχεδόν πραγματικό χρόνο. Παράλληλα, η αυξημένη συνδεσιμότητα δημιουργεί απαιτήσεις σχετικά με την ασφάλεια, τη διαχείριση των δεδομένων και την αξιόπιστη λειτουργία των συσκευών [2], [15].

2.2.1 IoT συσκευές και αισθητήρες

Οι αισθητήρες αποτελούν βασικό στοιχείο ενός συστήματος IoT, καθώς είναι υπεύθυνοι για τη συλλογή πληροφοριών από το φυσικό περιβάλλον. Ένας αισθητήρας μετατρέπει ένα φυσικό μέγεθος σε μετρήσιμη πληροφορία, η οποία μπορεί στη συνέχεια να μεταδοθεί και να υποστεί ψηφιακή επεξεργασία [14].

Ανάλογα με την εφαρμογή, μπορούν να χρησιμοποιηθούν αισθητήρες για τη μέτρηση θερμοκρασίας, υγρασίας, φωτεινότητας, πίεσης, συγκέντρωσης αερίων και αιωρούμενων σωματιδίων. Η αξιοπιστία

των δεδομένων ενός IoT συστήματος εξαρτάται σε μεγάλο βαθμό από την ποιότητα της διαδικασίας μέτρησης, τη βαθμονόμηση των αισθητήρων και τις συνθήκες στις οποίες αυτοί λειτουργούν.

Η βιβλιογραφία σχετικά με συστήματα περιβαλλοντικής και εσωτερικής παρακολούθησης δείχνει ότι οι αισθητήρες αποτελούν κρίσιμο μέρος της συνολικής αρχιτεκτονικής και ότι η επιλογή τους σχετίζεται άμεσα με την ακρίβεια, την κατανάλωση ενέργειας, το κόστος και τις απαιτήσεις της εφαρμογής [3].

2.2.2 Επικοινωνία και ροή δεδομένων σε συστήματα IoT

Η συλλογή μιας μέτρησης αποτελεί μόνο το πρώτο στάδιο λειτουργίας ενός IoT συστήματος. Τα δεδομένα πρέπει να μεταφερθούν με αξιόπιστο τρόπο από τη συσκευή συλλογής σε κάποιο επόμενο επίπεδο, όπου μπορούν να αποθηκευτούν, να επεξεργαστούν ή να αξιοποιηθούν [12].

Η επικοινωνία μεταξύ των συσκευών και των εφαρμογών μπορεί να πραγματοποιηθεί με διαφορετικά πρωτόκολλα και τεχνολογίες, ανάλογα με τις απαιτήσεις της εφαρμογής. Παράγοντες όπως το μέγεθος των μηνυμάτων, η διαθέσιμη υπολογιστική ισχύς, το εύρος ζώνης, η αξιοπιστία της σύνδεσης και η κατανάλωση ενέργειας επηρεάζουν την επιλογή του κατάλληλου μηχανισμού επικοινωνίας [13], [16].

Στο πλαίσιο της παρούσας εργασίας, η επικοινωνία των επιμέρους στοιχείων του συστήματος βασίζεται στο MQTT, το οποίο παρουσιάζεται αναλυτικότερα στην ενότητα 2.4.

2.3 Περιβαλλοντική παρακολούθηση

Η περιβαλλοντική παρακολούθηση αφορά τη συστηματική συλλογή δεδομένων που περιγράφουν την κατάσταση ενός φυσικού ή τεχνητού περιβάλλοντος. Η ανάπτυξη δικτυωμένων αισθητήρων επιτρέπει την αυτοματοποίηση αυτής της διαδικασίας και τη δημιουργία συστημάτων στα οποία οι μετρήσεις μπορούν να συλλέγονται περιοδικά και να μεταδίδονται σε κεντρικά ή απομακρυσμένα συστήματα.

Η χρήση IoT σε εφαρμογές περιβαλλοντικής παρακολούθησης μπορεί να αφορά τόσο εξωτερικούς χώρους όσο και εσωτερικά περιβάλλοντα. Στην περίπτωση εσωτερικών χώρων, η παρακολούθηση παραμέτρων όπως η θερμοκρασία, η υγρασία και οι δείκτες ποιότητας του αέρα χρησιμοποιείται για την αξιολόγηση των συνθηκών ενός χώρου και τη δημιουργία συνεχούς εικόνας της κατάστασής του [3].

2.3.1 Βασικοί περιβαλλοντικοί παράμετροι

Οι παράμετροι που επιλέγονται σε ένα σύστημα παρακολούθησης εξαρτώνται από τον σκοπό και το περιβάλλον εφαρμογής. Στην περίπτωση εσωτερικών χώρων, η θερμοκρασία και η σχετική υγρασία αποτελούν συνήθεις μετρήσεις, ενώ η ποιότητα του αέρα μπορεί να εκφράζεται μέσω διαφορετικών παραμέτρων και ρύπων, ανάλογα με τους διαθέσιμους αισθητήρες [3].

Η συνδυασμένη παρακολούθηση πολλών παραμέτρων είναι ιδιαίτερα χρήσιμη, επειδή επιτρέπει την πληρέστερη αποτύπωση των συνθηκών του περιβάλλοντος. Ωστόσο, η αξία των μετρήσεων δεν εξαρτάται μόνο από τη συλλογή τους, αλλά και από την ορθή καταγραφή και διαχείρισή τους.

2.3.2 Θερμοκρασία και υγρασία

Η θερμοκρασία και η σχετική υγρασία αποτελούν δύο βασικές περιβαλλοντικές παραμέτρους και χρησιμοποιούνται ευρέως για την περιγραφή των συνθηκών ενός χώρου. Η θερμοκρασία εκφράζει τη θερμική κατάσταση του περιβάλλοντος, ενώ η σχετική υγρασία περιγράφει τη συγκέντρωση υδρατμών σε σχέση με τη μέγιστη δυνατή συγκέντρωση στις συγκεκριμένες θερμοκρασιακές συνθήκες.

Σε εφαρμογές IoT παρακολούθησης εσωτερικού περιβάλλοντος, οι συγκεκριμένες μετρήσεις εμφανίζονται συχνά μαζί με παραμέτρους ποιότητας αέρα και χρησιμοποιούνται για την αξιολόγηση των συνθηκών άνεσης και του περιβάλλοντος στο οποίο βρίσκονται οι χρήστες [3].

2.3.3 Ποιότητα αέρα

Η ποιότητα του αέρα αποτελεί ευρύτερη έννοια που μπορεί να περιλαμβάνει διαφορετικούς ρύπους, αέρια και αιωρούμενα σωματίδια. Η επιλογή των παραμέτρων εξαρτάται από τον χώρο, τους στόχους της παρακολούθησης και τις δυνατότητες των αισθητήρων.

Συστήματα βασισμένα σε IoT έχουν χρησιμοποιηθεί για τη συνεχή παρακολούθηση της ποιότητας του αέρα, ιδιαίτερα σε εσωτερικούς χώρους. Η σχετική βιβλιογραφία αναδεικνύει ως σημαντικά ζητήματα την επιλογή αισθητήρων, τη βαθμονόμηση, την ενεργειακή αποδοτικότητα και την αξιοπιστία των μετρήσεων [3].

2.3.4 IoT στην περιβαλλοντική παρακολούθηση

Ο συνδυασμός περιβαλλοντικών αισθητήρων με δικτυακή επικοινωνία επιτρέπει τη συνεχή συλλογή και μετάδοση μετρήσεων, μειώνοντας την ανάγκη για χειροκίνητη καταγραφή. Η παραπάνω προσέγγιση μπορεί να αποτελέσει βάση για εφαρμογές παρακολούθησης κτιρίων, έξυπνων πόλεων, γεωργίας και άλλων περιβαλλοντικών συστημάτων [3].

Σημαντικό στοιχείο αυτών των εφαρμογών είναι η διατήρηση μιας αξιόπιστης αλυσίδας από την αρχική μέτρηση έως την τελική αποθήκευση και αξιοποίησή της. Το συγκεκριμένο ζήτημα αποκτά ιδιαίτερη σημασία όταν τα δεδομένα πρόκειται να χρησιμοποιηθούν αργότερα για επαλήθευση ή λήψη αποφάσεων.

2.4 MQTT

Το MQTT είναι ένα ελαφρύ πρωτόκολλο ανταλλαγής μηνυμάτων που βασίζεται στο μοντέλο publish/subscribe. Το πρότυπο MQTT 5.0, το οποίο δημοσιεύεται από τον OASIS, ορίζει ένα πρωτόκολλο επικοινωνίας client-server σχεδιασμένο για περιβάλλοντα όπου μπορεί να υπάρχουν περιορισμοί στο εύρος ζώνης, στη μνήμη ή στους υπολογιστικούς πόρους [4].

Η λειτουργία του MQTT βασίζεται σε τρεις βασικές οντότητες: τους clients, τον broker και τα topics. Οι clients αποστέλλουν ή λαμβάνουν μηνύματα και ο broker αναλαμβάνει τη διανομή τους σύμφωνα με τις ενεργές συνδρομές.

2.4.1 Αρχιτεκτονική publish/subscribe

Στο μοντέλο publish/subscribe, ένας publisher αποστέλλει ένα μήνυμα χωρίς να χρειάζεται να γνωρίζει άμεσα ποιος θα το παραλάβει. Το μήνυμα αποστέλλεται στον broker και στη συνέχεια διανέμεται στους subscribers που έχουν κατάλληλη συνδρομή [4].

Η αρχιτεκτονική αυτή δημιουργεί διαχωρισμό μεταξύ παραγωγού και καταναλωτή δεδομένων. Έτσι, μπορούν να προστεθούν ή να αφαιρεθούν clients χωρίς να απαιτείται άμεση αλλαγή στις σχέσεις επικοινωνίας μεταξύ όλων των συσκευών.

2.4.2 Topics και μηνύματα

Τα MQTT μηνύματα οργανώνονται μέσω topics. Ένας subscriber εγγράφεται σε ένα topic filter και ο broker προωθεί σε αυτόν τα application messages που αντιστοιχούν στη συνδρομή του [4].

Το MQTT υποστηρίζει επίσης wildcard χαρακτήρες στα topic filters. Αυτό επιτρέπει τη δημιουργία ιεραρχικής οργάνωσης των δεδομένων και την εγγραφή ενός subscriber σε ευρύτερο σύνολο topics χωρίς ξεχωριστή συνδρομή για κάθε επιμέρους θέμα.

Η λογική των topics είναι ιδιαίτερα χρήσιμη σε εφαρμογές αισθητήρων, καθώς διαφορετικές κατηγορίες δεδομένων μπορούν να τοποθετούνται σε διαφορετικά topics, διατηρώντας παράλληλα μία κοινή δομή επικοινωνίας.

2.4.3 MQTT Broker

Ο broker αποτελεί το κεντρικό ενδιάμεσο στοιχείο της επικοινωνίας MQTT. Παραλαμβάνει τα μηνύματα των publishers και τα διανέμει στους subscribers που έχουν ενεργές συνδρομές στα αντίστοιχα topics [4].

Με τον τρόπο αυτό οι clients δεν χρειάζεται να γνωρίζουν άμεσα ο ένας τον άλλον. Ο broker διαχειρίζεται τη ροή των μηνυμάτων και λειτουργεί ως κοινό σημείο επικοινωνίας.

Στην παρούσα εργασία χρησιμοποιήθηκε ο Mosquitto ως MQTT broker. Η επιλογή αυτή σχετίζεται με το γεγονός ότι ο broker αποτελεί ανεξάρτητο στοιχείο της αρχιτεκτονικής και επιτρέπει τη διαχείριση της επικοινωνίας χωρίς να απαιτείται άμεση σύνδεση μεταξύ των επιμέρους εφαρμογών.

2.4.4 Quality of Service

Το MQTT ορίζει τρία επίπεδα Quality of Service (QoS), τα οποία διαφοροποιούν τον τρόπο με τον οποίο αντιμετωπίζεται η παράδοση των μηνυμάτων [4]. Τα επίπεδα είναι:

- QoS 0: παράδοση το πολύ μία φορά.
- QoS 1: παράδοση τουλάχιστον μία φορά.
- QoS 2: παράδοση ακριβώς μία φορά.

Η επιλογή του κατάλληλου επιπέδου QoS αποτελεί σχεδιαστική απόφαση που εξαρτάται από τις απαιτήσεις της εφαρμογής. Υψηλότερο επίπεδο εξασφάλισης παράδοσης συνεπάγεται διαφορετική διαδικασία ανταλλαγής μηνυμάτων και αυξημένο κόστος επικοινωνίας.

2.4.5 MQTT σε εφαρμογές IoT

Η απλότητα του MQTT, η χρήση publish/subscribe και ο διαχωρισμός των clients μέσω του broker το καθιστούν κατάλληλο για εφαρμογές IoT. Το πρωτόκολλο σχεδιάστηκε με στόχο την απλή υλοποίηση και τη χρήση σε περιβάλλοντα όπου οι δικτυακοί και υπολογιστικοί πόροι μπορεί να είναι περιορισμένοι [4], [17].

Στα συστήματα περιβαλλοντικής παρακολούθησης, το MQTT μπορεί να χρησιμοποιηθεί ως επίπεδο μεταφοράς των μετρήσεων από τους αισθητήρες προς ένα σύστημα συλλογής ή επεξεργασίας [18]. Η συγκεκριμένη ιδιότητα επιτρέπει την ανεξάρτητη ανάπτυξη των παραγωγών δεδομένων και του λογισμικού που τα καταναλώνει.

2.5 Κρυπτογραφικές συναρτήσεις κατακερματισμού

Οι κρυπτογραφικές συναρτήσεις κατακερματισμού χρησιμοποιούνται για τη δημιουργία ενός σταθερού μήκους αποτυπώματος από μια είσοδο αυθαίρετου μεγέθους. Το αποτέλεσμα ονομάζεται message digest ή hash και μπορεί να χρησιμοποιηθεί για τον έλεγχο της ακεραιότητας της εισόδου.

Το NIST Secure Hash Standard περιγράφει τις συναρτήσεις της οικογένειας SHA και τις χρήσεις τους για τη δημιουργία digests, τα οποία μπορούν να αξιοποιηθούν για την ανίχνευση μεταβολών σε δεδομένα [5].

2.5.1 Hash και message digest

Το hash μπορεί να θεωρηθεί ως μία σύντομη ψηφιακή αναπαράσταση ενός συνόλου δεδομένων. Δεν αποτελεί αντίγραφο της αρχικής πληροφορίας και δεν έχει ως σκοπό την ανάκτησή της.

Βασικό χαρακτηριστικό της διαδικασίας είναι ότι η ίδια είσοδος παράγει το ίδιο digest όταν χρησιμοποιείται ο ίδιος αλγόριθμος. Αντίθετα, μία μεταβολή της εισόδου οδηγεί σε διαφορετικό αποτέλεσμα. Η ιδιότητα αυτή επιτρέπει τη χρήση hashes ως μηχανισμού σύγκρισης μεταξύ μιας αρχικής και μιας μεταγενέστερης κατάστασης των δεδομένων [5].

2.5.2 Βασικές ιδιότητες

Για να είναι χρήσιμη μία κρυπτογραφική συνάρτηση κατακερματισμού, απαιτούνται ιδιότητες που καθιστούν πρακτικά δύσκολη την εύρεση κατάλληλων συγκρούσεων και την ανακατασκευή της αρχικής εισόδου από το digest. Σημαντικό είναι επίσης το φαινόμενο χιονοστιβάδας, κατά το οποίο μία μικρή αλλαγή στην είσοδο μπορεί να προκαλέσει σημαντική αλλαγή στην έξοδο.

Στο πλαίσιο της παρούσας εργασίας ενδιαφέρει κυρίως η δυνατότητα δημιουργίας ενός σταθερού αποτυπώματος ενός πλήρους κύκλου δεδομένων, το οποίο μπορεί στη συνέχεια να αποθηκευτεί και να χρησιμοποιηθεί ως σημείο αναφοράς.

2.5.3 SHA-256

Ο SHA-256 αποτελεί μέλος της οικογένειας SHA-2 του Secure Hash Standard [5]. Παράγει digest μήκους 256 bit, ανεξάρτητα από το μέγεθος της εισόδου εντός των ορίων του προτύπου.

Η σταθερή διάρκεια της εξόδου, σε συνδυασμό με τη χρήση του σε διαδικασίες ελέγχου ακεραιότητας, καθιστά τον SHA-256 κατάλληλο για τη δημιουργία σύντομων αποτυπωμάτων σύνθετων δεδομένων.

2.5.4 Hash-based integrity verification

Η χρήση ενός hash για έλεγχο ακεραιότητας βασίζεται στη σύγκριση δύο αποτελεσμάτων. Αρχικά δημιουργείται ένα digest από τα δεδομένα και αποθηκεύεται ως σημείο αναφοράς. Σε μεταγενέστερο χρόνο δημιουργείται νέο digest από τα ίδια δεδομένα και συγκρίνεται με το αρχικό.

Εάν τα δύο αποτελέσματα διαφέρουν, υπάρχει ένδειξη ότι η είσοδος έχει αλλάξει. Επομένως, το hash δεν προστατεύει από μόνο του τα δεδομένα από κάθε μορφή επίθεσης, αλλά παρέχει έναν μηχανισμό ανίχνευσης τροποποιήσεων όταν το αρχικό αποτύπωμα παραμένει αξιόπιστο [5].

2.6 Distributed Ledger Technology

Η Distributed Ledger Technology αναφέρεται σε συστήματα στα οποία ένα καθολικό ή ένα σύνολο εγγραφών διατηρείται με κατανεμημένο τρόπο μεταξύ συμμετεχόντων ενός δικτύου. Η καταγραφή και η ενημέρωση της κοινής κατάστασης πραγματοποιούνται με βάση μηχανισμούς που εξασφαλίζουν συμφωνία μεταξύ των συμμετεχόντων [6].

Η DLT δεν ταυτίζεται με το blockchain. Το blockchain αποτελεί μία συγκεκριμένη αρχιτεκτονική κατανεμημένου καθολικού, στην οποία τα δεδομένα οργανώνονται σε αλυσίδα blocks. Υπάρχουν όμως και άλλες μορφές κατανεμημένων καθολικών με διαφορετικές δομές και μηχανισμούς καταγραφής [6], [7], [19].

2.6.1 Έννοια και βασικά χαρακτηριστικά

Ένα βασικό χαρακτηριστικό ενός κατανεμημένου καθολικού είναι η ύπαρξη κοινής κατάστασης η οποία δεν εξαρτάται αποκλειστικά από ένα μοναδικό κεντρικό αποθετήριο. Οι συμμετέχοντες στο δίκτυο συνεργάζονται ώστε οι καταχωρίσεις να γίνονται αποδεκτές σύμφωνα με τους κανόνες του εκάστοτε πρωτοκόλλου [6], [7].

Άλλα χαρακτηριστικά που συναντώνται σε DLT συστήματα είναι η κατανεμημένη διατήρηση των καταγραφών, η χρήση κρυπτογραφικών μηχανισμών και η δυνατότητα δημιουργίας ιστορικού καταχωρίσεων που είναι δύσκολο να τροποποιηθεί εκ των υστέρων [6], [7], [20].

2.6.2 Ακεραιότητα και αμεταβλητότητα καταχωρίσεων

Η καταγραφή δεδομένων σε κατανεμημένο καθολικό μπορεί να προσφέρει ισχυρές ενδείξεις σχετικά με το εάν μια καταχώρηση έχει μεταβληθεί μετά την καταγραφή της. Το ακριβές επίπεδο αμεταβλητότητας εξαρτάται από την αρχιτεκτονική και τον μηχανισμό συναίνεσης του συστήματος [6].

Για τον λόγο αυτό είναι σημαντικό να γίνεται διάκριση μεταξύ της έννοιας «δεν τροποποιείται πρακτικά» και της απόλυτης έννοιας της αδυναμίας τροποποίησης. Η ασφάλεια μιας DLT εξαρτάται από το πρωτόκολλο, τους μηχανισμούς συναίνεσης, την κατανομή των συμμετεχόντων και άλλες παραμέτρους του δικτύου.

2.6.3 DLT και blockchain

Το blockchain είναι μια μορφή DLT στην οποία οι εγγραφές οργανώνονται σε blocks που συνδέονται διαδοχικά. Άλλες αρχιτεκτονικές μπορεί να χρησιμοποιούν διαφορετικές δομές δεδομένων και διαφορετικούς τρόπους οργάνωσης των καταχωρήσεων.

Η διάκριση αυτή είναι σημαντική στην παρούσα εργασία, καθώς το DLT επίπεδο δεν αντιμετωπίζεται ως απαραίτητα συνώνυμο του blockchain. Το ενδιαφέρον εστιάζεται στη δυνατότητα δημιουργίας μιας ανεξάρτητης, δικτυακά διαμοιραζόμενης καταχώρισης που μπορεί να χρησιμοποιηθεί ως σημείο αναφοράς για δεδομένα τα οποία διατηρούνται αλλού.

2.6.4 DLT ως μηχανισμός ανεξάρτητης αναφοράς

Μία από τις πιθανές χρήσεις ενός DLT είναι η δημιουργία ενός ανεξάρτητου σημείου αναφοράς για δεδομένα τα οποία αποθηκεύονται σε εξωτερικό σύστημα. Αντί να αποθηκεύεται ολόκληρη η πληροφορία στο κατανεμημένο καθολικό, μπορεί να αποθηκεύεται μία μικρότερη πληροφορία που συνδέεται κρυπτογραφικά με την αρχική καταχώριση.

Η προσέγγιση αυτή είναι ιδιαίτερα χρήσιμη όταν τα αρχικά δεδομένα είναι μεγαλύτερα ή όταν η αποθήκευσή τους στο DLT δεν είναι απαραίτητη [22]. Η βιβλιογραφία σχετικά με DLT επισημαίνει ότι τα κατανεμημένα καθολικά μπορούν να χρησιμοποιηθούν ως ιστορικά συνεπή καταστήματα καταγραφών, ενώ τα κρυπτογραφικά hashes μπορούν να χρησιμοποιηθούν για προστασία της ακεραιότητας [7], [21].

2.7 IOTA και Shimmer

Η IOTA αποτελεί πλατφόρμα Distributed Ledger Technology (DLT), η οποία έχει εξελιχθεί μέσα από διαφορετικές αρχιτεκτονικές και τεχνολογικές εκδόσεις. Η σύγχρονη τεκμηρίωση της IOTA παρουσιάζει ένα προγραμματιζόμενο blockchain και παρέχει εργαλεία ανάπτυξης, βιβλιοθήκες και υποδομές για την ανάπτυξη εφαρμογών πάνω στο οικοσύστημα της IOTA [8], [23].

Για την παρούσα εργασία έχει ιδιαίτερη σημασία το δίκτυο Shimmer, το οποίο χρησιμοποιήθηκε ως το DLT επίπεδο της υλοποίησης. Η τεχνολογία του Shimmer σχεδιάστηκε ως δίκτυο του οικοσυστήματος της IOTA για την ανάπτυξη και δοκιμή τεχνολογικών δυνατοτήτων, ενώ παρέχει δυνατότητες καταγραφής δεδομένων και αξιοποίησης κατανεμημένων υποδομών [9].

2.7.1 Shimmer

Το Shimmer αποτελεί ένα δίκτυο του οικοσυστήματος της IOTA, το οποίο αναπτύχθηκε με στόχο την υποστήριξη νέων τεχνολογικών χαρακτηριστικών και εφαρμογών DLT. Η επίσημη τεκμηρίωση του δικτύου αναφέρεται σε δυνατότητες που περιλαμβάνουν, μεταξύ άλλων, την καταγραφή δεδομένων και τη χρήση της υποδομής του δικτύου από εφαρμογές τρίτων [9].

Στο πλαίσιο της παρούσας εργασίας, το Shimmer χρησιμοποιείται ως το κατανεμημένο επίπεδο στο οποίο δημοσιεύονται τα κρυπτογραφικά αποτυπώματα των κύκλων δεδομένων. Με αυτόν τον τρόπο δημιουργείται ένα εξωτερικό σημείο αναφοράς για την πληροφορία που έχει αποθηκευτεί τοπικά.

2.7.2 Καταγραφή δεδομένων στο Shimmer

Η δυνατότητα καταγραφής δεδομένων σε ένα DLT επιτρέπει σε εφαρμογές να συνδέσουν μία συγκεκριμένη πληροφορία με μία καταχώρηση στο δίκτυο. Στην περίπτωση του Shimmer, η αρχιτεκτονική του υποστηρίζει την καταγραφή δεδομένων και πληροφοριών που σχετίζονται με εφαρμογές του δικτύου [9].

Η δυνατότητα αυτή είναι ιδιαίτερα σημαντική για την παρούσα εργασία, επειδή το πλήρες περιεχόμενο των περιβαλλοντικών μετρήσεων δεν χρειάζεται να δημοσιεύεται στο DLT. Αντί αυτού, μπορεί να χρησιμοποιηθεί ένα κρυπτογραφικό αποτύπωμα που αντιπροσωπεύει το συγκεκριμένο σύνολο δεδομένων.

Σε σχετική τεχνική τεκμηρίωση του οικοσυστήματος IOTA και Shimmer περιγράφεται η χρήση data payloads για τη δημοσίευση πληροφοριών που προέρχονται από εφαρμογές [10]. Η προσέγγιση αυτή επιτρέπει τη σύνδεση μιας εφαρμογής με δεδομένα που καταγράφονται στο δίκτυο χωρίς να απαιτείται η χρήση του DLT ως κύριου συστήματος αποθήκευσης όλων των πρωτογενών δεδομένων.

2.7.3 Transactions και αναγνωριστικά

Οι καταχωρήσεις δεδομένων στο Shimmer μπορούν να συνοδεύονται από αναγνωριστικά που επιτρέπουν την αναφορά σε μία συγκεκριμένη καταχώριση του δικτύου. Η δυνατότητα αυτή είναι σημαντική για εφαρμογές στις οποίες απαιτείται σύνδεση μεταξύ δεδομένων που διατηρούνται σε ένα εξωτερικό σύστημα και της αντίστοιχης πληροφορίας που έχει δημοσιευθεί στο DLT.

Στην παρούσα εργασία, το κρυπτογραφικό αποτύπωμα των δεδομένων δημοσιεύεται μέσω Tagged Data block. Με την επιτυχή δημοσίευση επιστρέφεται ένα μοναδικό Block ID, το οποίο μπορεί να χρησιμοποιηθεί αργότερα για την ανάκτηση της συγκεκριμένης καταχώρισης από το δίκτυο. Η συγκεκριμένη δυνατότητα αξιοποιείται στη συνέχεια από τον μηχανισμό επαλήθευσης της εφαρμογής.

Η διατήρηση του Block ID μαζί με την αντίστοιχη τοπική καταχώριση επιτρέπει τη συσχέτιση των δεδομένων της βάσης με το block που δημιουργήθηκε στο Shimmer. Έτσι, η τοπική πληροφορία μπορεί να ελεγχθεί όχι μόνο ως προς το αποθηκευμένο hash της, αλλά και ως προς το hash που ανακτάται από την αντίστοιχη καταχώριση του δικτύου.

Η προσέγγιση αυτή δημιουργεί έναν σαφή σύνδεσμο μεταξύ της τοπικής αποθήκευσης και της κατανεμημένης καταγραφής, χωρίς να απαιτείται η αποθήκευση ολόκληρου του συνόλου των περιβαλλοντικών μετρήσεων στο DLT.

2.7.4 Δημοσίευση κρυπτογραφικών αποτυπωμάτων

Η δημοσίευση ενός κρυπτογραφικού αποτυπώματος σε ένα DLT μπορεί να χρησιμοποιηθεί για τη δημιουργία ενός ανεξάρτητου σημείου αναφοράς για δεδομένα που διατηρούνται σε εξωτερικό σύστημα. Στην προσέγγιση αυτή δεν απαιτείται η δημοσίευση ολόκληρου του αρχικού συνόλου δεδομένων, αλλά μιας μικρότερης πληροφορίας που προκύπτει από αυτά.

Στην παρούσα εργασία, το κρυπτογραφικό αποτύπωμα κάθε ολοκληρωμένου κύκλου δημιουργείται με χρήση του SHA-256 και στη συνέχεια δημοσιεύεται στο δίκτυο Shimmer μέσω ενός Tagged Data

block. Το hash χρησιμοποιείται ως το περιεχόμενο της καταχώρισης, ενώ το block συνοδεύεται από προκαθορισμένο tag που επιτρέπει τη διάκριση της συγκεκριμένης κατηγορίας δεδομένων.

Μετά την επιτυχή δημοσίευση επιστρέφεται το αντίστοιχο Block ID. Το αναγνωριστικό αυτό μπορεί να χρησιμοποιηθεί αργότερα για την ανάκτηση της καταχώρισης και την εξαγωγή του hash που είχε δημοσιευθεί στο δίκτυο. Η διαδικασία αυτή επιτρέπει τη σύνδεση της τοπικά αποθηκευμένης πληροφορίας με τη σχετική καταχώρηση στο DLT.

Η συγκεκριμένη προσέγγιση επιτρέπει στο πλήρες σύνολο των περιβαλλοντικών δεδομένων να παραμένει αποθηκευμένο στην τοπική βάση δεδομένων, ενώ στο DLT διατηρείται μόνο το κρυπτογραφικό του αποτύπωμα. Έτσι, το DLT λειτουργεί ως ανεξάρτητο σημείο αναφοράς και μπορεί να χρησιμοποιηθεί σε μεταγενέστερο έλεγχο της ακεραιότητας των δεδομένων.

Στην πράξη, ο μηχανισμός επαλήθευσης της εργασίας ανακτά το συγκεκριμένο block μέσω του Block ID, αποκωδικοποιεί το περιεχόμενο του data payload και συγκρίνει το ανακτημένο hash με το hash που υπολογίζεται εκ νέου από τα τοπικά δεδομένα.

2.8 Ακεραιότητα δεδομένων και ανίχνευση αλλοίωσης

Η ακεραιότητα των δεδομένων αφορά τη δυνατότητα διαπίστωσης ότι η πληροφορία παραμένει ίδια με την κατάσταση στην οποία καταγράφηκε αρχικά. Σε ένα σύστημα που συλλέγει και αποθηκεύει μετρήσεις, η απλή ύπαρξη μιας βάσης δεδομένων δεν εξασφαλίζει από μόνη της ότι το περιεχόμενό της δεν έχει τροποποιηθεί.

Για τον λόγο αυτό απαιτείται ένας μηχανισμός που να επιτρέπει τον μεταγενέστερο έλεγχο της πληροφορίας. Οι κρυπτογραφικές συναρτήσεις κατακερματισμού παρέχουν μία τέτοια δυνατότητα, δημιουργώντας ένα αποτύπωμα που μπορεί να συγκριθεί με το αποτέλεσμα ενός νέου υπολογισμού [5].

2.8.1 Έλεγχος ακεραιότητας μέσω hash

Ο έλεγχος ακεραιότητας μέσω hash βασίζεται στη δημιουργία ενός αρχικού digest και στην ασφαλή διατήρησή του. Όταν απαιτείται επαλήθευση, το hash υπολογίζεται εκ νέου από τα δεδομένα και συγκρίνεται με το αρχικό αποτέλεσμα.

Εφόσον τα δεδομένα είναι ίδια, τα δύο hashes πρέπει να συμφωνούν. Εάν έχει αλλάξει έστω και ένα τμήμα της εισόδου, το αποτέλεσμα του hash αναμένεται να διαφέρει [5].

Η διαδικασία αυτή δεν εξηγεί από μόνη της πότε ή από ποιον έγινε μία αλλαγή. Παρέχει όμως σαφή ένδειξη ότι η πληροφορία που εξετάζεται δεν είναι πλέον ίδια με εκείνη που χρησιμοποιήθηκε κατά τη δημιουργία του αρχικού αποτυπώματος.

2.8.2 Ανίχνευση αλλοίωσης

Η ανίχνευση αλλοίωσης αποτελεί άμεση εφαρμογή του ελέγχου ακεραιότητας. Εάν μία τιμή που συμμετέχει στη δημιουργία του hash τροποποιηθεί, το συνολικό digest μεταβάλλεται.

Αυτό επιτρέπει στο σύστημα να εντοπίσει την ύπαρξη μεταβολής χωρίς να απαιτείται γνώση του ακριβούς πεδίου που τροποποιήθηκε. Επομένως, ο μηχανισμός είναι κατάλληλος για την ανίχνευση τόσο σκόπιμων όσο και ακούσιων μεταβολών, υπό την προϋπόθεση ότι το αρχικό hash παραμένει αξιόπιστο.

2.8.3 Συνδυασμός τοπικής αποθήκευσης και DLT

Ο συνδυασμός τοπικής αποθήκευσης και DLT δημιουργεί δύο διαφορετικά επίπεδα αποθήκευσης με διαφορετικούς ρόλους. Το πλήρες περιεχόμενο των δεδομένων μπορεί να παραμένει σε μία τοπική βάση, ενώ ένα κρυπτογραφικό αποτύπωμα αποθηκεύεται σε ένα ανεξάρτητο κατακερματισμένο σύστημα.

Η προσέγγιση αυτή επιτρέπει τον μεταγενέστερο έλεγχο της ακεραιότητας. Αν το τοπικό περιεχόμενο τροποποιηθεί, το νέο hash δεν θα συμφωνεί με το hash που είχε προηγουμένως δημοσιευθεί στο DLT.

Επομένως, το DLT δεν λειτουργεί ως αντίγραφο της τοπικής βάσης δεδομένων. Λειτουργεί ως ανεξάρτητο σημείο αναφοράς, το οποίο μπορεί να χρησιμοποιηθεί για τη διαπίστωση της συνέπειας των τοπικά αποθηκευμένων δεδομένων. Η συγκεκριμένη λογική αποτελεί τη θεωρητική βάση της αρχιτεκτονικής που αναλύεται στα επόμενα κεφάλαια.

2.9 Επίλογος

Στο παρόν κεφάλαιο παρουσιάστηκαν οι βασικές τεχνολογίες και έννοιες που αποτελούν το θεωρητικό υπόβαθρο της εργασίας. Αρχικά παρουσιάστηκε το Internet of Things και η σχέση του με τη συλλογή περιβαλλοντικών δεδομένων. Στη συνέχεια εξετάστηκε η χρήση του MQTT ως πρωτοκόλλου επικοινωνίας μεταξύ των επιμέρους στοιχείων ενός IoT συστήματος.

Ακολούθησε η παρουσίαση των κρυπτογραφικών συναρτήσεων κατακερματισμού και ειδικότερα του SHA-256, με έμφαση στη χρήση τους για τον έλεγχο ακεραιότητας. Στη συνέχεια αναλύθηκε η έννοια της Distributed Ledger Technology και η διάκριση μεταξύ DLT και blockchain. Τέλος, παρουσιάστηκαν η IOTA και το Shimmer, καθώς και η δυνατότητα χρήσης ενός DLT για την καταγραφή κρυπτογραφικών αποτυπωμάτων δεδομένων.

Οι παραπάνω έννοιες αποτελούν τη θεωρητική βάση για την αρχιτεκτονική και τις τεχνικές επιλογές της παρούσας εργασίας. Στο επόμενο κεφάλαιο παρουσιάζονται οι απαιτήσεις του συστήματος, οι σχεδιαστικές αποφάσεις και η αρχιτεκτονική που προέκυψε από την αξιοποίηση των παραπάνω τεχνολογιών.

Κεφάλαιο 3ο: Ανάλυση Απαιτήσεων και Σχεδιασμός

3.1 Εισαγωγή

Μετά την παρουσίαση του θεωρητικού υπόβαθρου και των τεχνολογιών που σχετίζονται με το αντικείμενο της εργασίας, στο παρόν κεφάλαιο παρουσιάζεται ο σχεδιασμός της προτεινόμενης λύσης. Στόχος είναι να καθοριστεί η δομή του συστήματος, οι απαιτήσεις που πρέπει να ικανοποιούνται και ο τρόπος με τον οποίο τα επιμέρους υποσυστήματα θα συνεργάζονται.

Αρχικά παρουσιάζονται οι λειτουργικές και μη λειτουργικές απαιτήσεις, καθώς και οι βασικοί περιορισμοί της υλοποίησης. Στη συνέχεια αναλύεται η συνολική αρχιτεκτονική και η ροή των δεδομένων, ενώ εξετάζονται ξεχωριστά ο σχεδιασμός της αποθήκευσης, η δημιουργία και αξιοποίηση του κρυπτογραφικού αποτυπώματος, η διασύνδεση με το DLT και οι μηχανισμοί επαλήθευσης.

Τέλος, παρουσιάζεται ο σχεδιασμός του Launcher και η συνολική ροή λειτουργίας του συστήματος. Οι σχεδιαστικές επιλογές που παρουσιάζονται αποτελούν τη βάση για την υλοποίηση που αναλύεται στο επόμενο κεφάλαιο.

3.2 Ανάλυση απαιτήσεων

Πριν από την έναρξη της υλοποίησης καθορίστηκαν οι βασικές απαιτήσεις του συστήματος. Η διαδικασία αυτή χρησιμοποιήθηκε ως βάση για τη διαμόρφωση της αρχιτεκτονικής και για την επιλογή του τρόπου με τον οποίο θα συνεργάζονται τα επιμέρους υποσυστήματα.

Οι απαιτήσεις διακρίνονται σε λειτουργικές και μη λειτουργικές. Οι πρώτες αφορούν τις λειτουργίες που πρέπει να παρέχει το σύστημα, ενώ οι δεύτερες σχετίζονται με τον τρόπο λειτουργίας και τις γενικότερες ιδιότητες της εφαρμογής. Παράλληλα, καθορίστηκαν περιορισμοί και παραδοχές που έπρεπε να ληφθούν υπόψη κατά τον σχεδιασμό.

3.2.1 Λειτουργικές απαιτήσεις

Οι βασικές λειτουργικές απαιτήσεις του συστήματος παρουσιάζονται στον Πίνακα 3.1.

Πίνακας 3.1: Λειτουργικές απαιτήσεις του συστήματος

Requirements	Λειτουργική απαίτηση
R1	Παραγωγή δεδομένων θερμοκρασίας, υγρασίας και ποιότητας αέρα από προσομοιωμένους αισθητήρες.
R2	Μετάδοση των παραγόμενων δεδομένων μέσω MQTT.
R3	Συλλογή των επιμέρους μετρήσεων από τον Gateway.
R4	Δημιουργία ολοκληρωμένου κύκλου όταν είναι διαθέσιμες οι απαιτούμενες μετρήσεις.
R5	Αποθήκευση των κύκλων σε βάση δεδομένων SQLite.
R6	Δημιουργία κρυπτογραφικού αποτυπώματος SHA-256 για κάθε κύκλο.
R7	Δημοσίευση του hash στο δίκτυο Shimmer μέσω Tagged Data block.
R8	Αποθήκευση του Block ID της αντίστοιχης καταχώρισης.
R9	Έλεγχος της ακεραιότητας των αποθηκευμένων δεδομένων.
R10	Ανίχνευση σκόπιμης αλλοίωσης δεδομένων μέσω ελεγχόμενων δοκιμών.

R11	Παροχή γραφικού περιβάλλοντος για τη διαχείριση των διεργασιών του project.
R12	Εκκίνηση του Publisher μόνο αφού ο Gateway έχει ολοκληρώσει επιτυχώς την αρχικοποίησή του.
R13	Τερματισμός των διεργασιών που ανήκουν στο project μέσω του Launcher.

Οι παραπάνω απαιτήσεις αποτέλεσαν τα βασικά λειτουργικά κριτήρια που έπρεπε να ικανοποιηθούν κατά την υλοποίηση και χρησιμοποιούνται αργότερα ως βάση για τη διαδικασία ελέγχου.

3.2.2 Μη λειτουργικές απαιτήσεις

Πέρα από τις επιμέρους λειτουργίες, η εφαρμογή έπρεπε να παρουσιάζει συγκεκριμένα χαρακτηριστικά ως προς τον τρόπο λειτουργίας της.

Αρχικά, τα επιμέρους components έπρεπε να είναι διαχωρισμένα μεταξύ τους, ώστε να μπορούν να αναπτύσσονται και να ελέγχονται ανεξάρτητα. Παράλληλα, η ροή των δεδομένων έπρεπε να παραμένει προβλέψιμη και να επιτρέπει την επανάληψη των ίδιων διαδικασιών χωρίς να απαιτούνται διαφορετικές ρυθμίσεις σε κάθε εκτέλεση.

Η αποθήκευση έπρεπε να είναι μόνιμη, έτσι ώστε τα δεδομένα να παραμένουν διαθέσιμα και μετά τον τερματισμό της εφαρμογής. Επιπλέον, οι διαδικασίες ελέγχου ακεραιότητας έπρεπε να μπορούν να επαναληφθούν σε μεταγενέστερο χρόνο με τα ίδια δεδομένα.

Τέλος, η συνολική εφαρμογή σχεδιάστηκε ώστε να μπορεί να εκτελείται σε περιβάλλον Windows μέσω ενός ενιαίου γραφικού περιβάλλοντος διαχείρισης, χωρίς να απαιτείται συνεχής χειροκίνητη διαχείριση όλων των επιμέρους διεργασιών.

3.2.3 Περιορισμοί και παραδοχές

Κατά τον σχεδιασμό θεωρήθηκε ότι η παραγωγή των περιβαλλοντικών δεδομένων θα πραγματοποιείται εξ ολοκλήρου μέσω προσομοιωμένων αισθητήρων. Επομένως, ζητήματα που σχετίζονται με την πραγματική φυσική λειτουργία αισθητήρων, όπως η κατανάλωση ενέργειας, η βαθμονόμηση και οι αποκλίσεις μετρήσεων, δεν αποτελούν αντικείμενο της παρούσας υλοποίησης.

Παράλληλα, το πλήρες περιεχόμενο των δεδομένων αποθηκεύεται τοπικά, ενώ στο DLT δημοσιεύεται μόνο το αντίστοιχο κρυπτογραφικό αποτύπωμα. Η συγκεκριμένη επιλογή αποτελεί βασική παραδοχή του σχεδιασμού.

Η λειτουργία του MQTT επιπέδου προϋποθέτει ενεργό broker, ενώ η δημοσίευση στο DLT προϋποθέτει διαθέσιμη δικτυακή σύνδεση και πρόσβαση στην αντίστοιχη υποδομή.

Τέλος, η αξιολόγηση επικεντρώνεται στην ορθή λειτουργία της σχεδιασμένης ροής και στην επαλήθευση της ακεραιότητας των δεδομένων και όχι στην αξιολόγηση ενός συστήματος μεγάλης παραγωγικής κλίμακας.

3.3 Συνολική αρχιτεκτονική του συστήματος

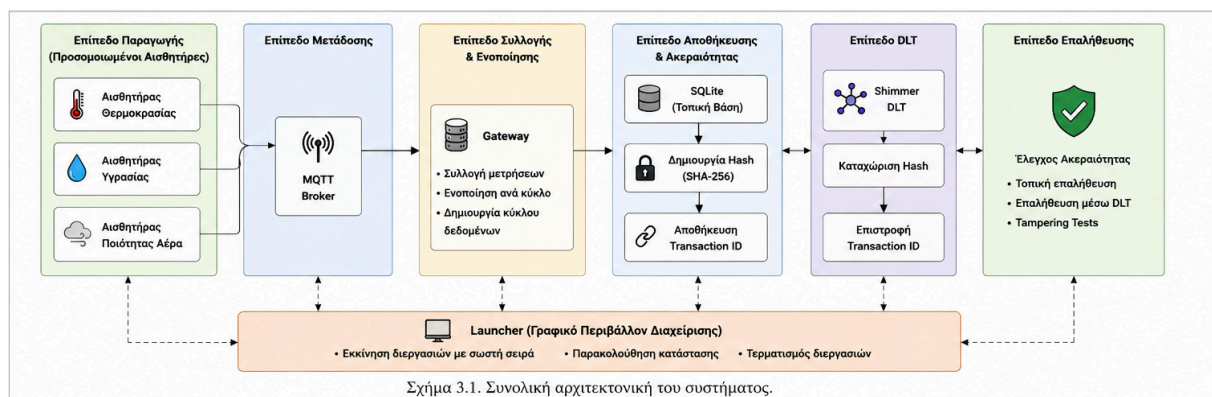
Η αρχιτεκτονική σχεδιάστηκε με διακριτά υποσυστήματα, ώστε κάθε component να έχει σαφή ρόλο και συγκεκριμένα όρια ευθύνης. Η προσέγγιση αυτή επιτρέπει την ανεξάρτητη ανάπτυξη και τον έλεγχο των επιμέρους λειτουργιών, ενώ παράλληλα διατηρείται ενιαία ροή δεδομένων.

Σε επίπεδο παραγωγής βρίσκονται οι προσομοιωμένοι αισθητήρες, οι οποίοι παρέχουν τις περιβαλλοντικές μετρήσεις. Τα δεδομένα μεταφέρονται μέσω του MQTT Broker στον Gateway, ο οποίος λειτουργεί ως κεντρικό σημείο συλλογής και ενοποίησης.

Μετά την ολοκλήρωση ενός κύκλου, η πληροφορία οδηγείται στο επίπεδο αποθήκευσης και ακεραιότητας. Ο κύκλος αποθηκεύεται τοπικά και δημιουργείται το αντίστοιχο κρυπτογραφικό αποτύπωμα. Το hash δημοσιεύεται στη συνέχεια στο DLT και το αναγνωριστικό της συναλλαγής διατηρείται μαζί με την τοπική καταχώριση.

Παράλληλα, υπάρχει ξεχωριστό επίπεδο επαλήθευσης, το οποίο επιτρέπει τον έλεγχο της τρέχουσας κατάστασης των δεδομένων και τη σύγκρισή τους με τα αποτυπώματα που έχουν ήδη καταγραφεί.

Ο Launcher λειτουργεί ως επίπεδο διαχείρισης της εφαρμογής. Δεν συμμετέχει στην παραγωγή ή επεξεργασία των περιβαλλοντικών δεδομένων, αλλά αναλαμβάνει τον έλεγχο της εκτέλεσης και της κατάστασης των επιμέρους διεργασιών.



Σχήμα 3.1. Συνολική αρχιτεκτονική του συστήματος.

3.4 Σχεδιασμός της ροής δεδομένων

Η ροή των δεδομένων σχεδιάστηκε ως μία διαδοχική διαδικασία κατά την οποία κάθε στάδιο ενεργοποιείται από το αποτέλεσμα του προηγούμενου. Με αυτόν τον τρόπο αποφεύγεται η άμεση εξάρτηση μεταξύ των επιμέρους πηγών δεδομένων και των τελικών διαδικασιών αποθήκευσης και επαλήθευσης.

3.4.1 Παραγωγή περιβαλλοντικών μετρήσεων

Η διαδικασία ξεκινά με την παραγωγή των μετρήσεων από τους προσομοιωμένους αισθητήρες. Κάθε αισθητήρας είναι υπεύθυνος για μία συγκεκριμένη περιβαλλοντική παράμετρο και δημιουργεί μία τιμή σύμφωνα με τη λογική προσομοίωσης που έχει οριστεί για το αντίστοιχο μέγεθος.

Οι τιμές σχεδιάστηκαν ώστε να μεταβάλλονται σταδιακά και να παρουσιάζουν συνεχή συμπεριφορά μεταξύ διαδοχικών μετρήσεων. Κάθε μέτρηση συνοδεύεται από χρονική πληροφορία, ώστε να μπορεί να συσχετιστεί με τη χρονική στιγμή παραγωγής της.

3.4.2 Μετάδοση μέσω MQTT

Μετά την παραγωγή τους, οι μετρήσεις μεταδίδονται στον MQTT Broker. Η χρήση διαφορετικών topics για τις επιμέρους παραμέτρους επιτρέπει στον Gateway να αναγνωρίζει τον τύπο κάθε εισερχόμενης μέτρησης.

Η συγκεκριμένη οργάνωση διατηρεί ανεξάρτητη τη διαδικασία παραγωγής από την επεξεργασία. Οι αισθητήρες αποστέλλουν μόνο τις τιμές τους, ενώ η λογική ενοποίησης αναλαμβάνεται σε επόμενο στάδιο.

3.4.3 Συλλογή και ενοποίηση στον Gateway

Ο Gateway παραλαμβάνει τις εισερχόμενες μετρήσεις και ενημερώνει την προσωρινή κατάσταση του συστήματος. Κάθε τιμή αντιστοιχίζεται στη σωστή περιβαλλοντική παράμετρο.

Η διαδικασία συνεχίζεται μέχρι να έχουν συγκεντρωθεί όλες οι μετρήσεις που απαιτούνται για έναν πλήρη κύκλο. Με αυτόν τον τρόπο, οι επιμέρους μετρήσεις δεν αντιμετωπίζονται ως ανεξάρτητες εγγραφές, αλλά ως στοιχεία ενός ενιαίου συνόλου.

3.4.4 Ολοκλήρωση κύκλου δεδομένων

Μόλις συγκεντρωθούν όλες οι απαιτούμενες τιμές, δημιουργείται ένας νέος κύκλος δεδομένων. Ο κύκλος αποτελεί την ενιαία μονάδα πάνω στην οποία εκτελούνται τα επόμενα στάδια επεξεργασίας.

Η καταχώριση περιλαμβάνει το μοναδικό αναγνωριστικό του κύκλου, τις περιβαλλοντικές μετρήσεις και τη χρονική πληροφορία. Στη συνέχεια η καταχώριση οδηγείται στο επίπεδο αποθήκευσης και ακεραιότητας.

3.5 Σχεδιασμός αποθήκευσης και ακεραιότητας

Μετά τη δημιουργία ενός ολοκληρωμένου κύκλου απαιτείται αφενός η διατήρηση των δεδομένων και αφετέρου η δημιουργία ενός σταθερού σημείου αναφοράς για μελλοντικούς ελέγχους. Για τον λόγο αυτό σχεδιάστηκε ξεχωριστό επίπεδο αποθήκευσης και ακεραιότητας, το οποίο συνδέει την τοπική καταχώριση με το αντίστοιχο κρυπτογραφικό αποτύπωμα και τη σχετική καταχώριση στο DLT.

3.5.1 Δομή δεδομένων και κύκλου

Κάθε ολοκληρωμένος κύκλος περιλαμβάνει ένα μοναδικό αναγνωριστικό, τις τιμές θερμοκρασίας, υγρασίας και ποιότητας αέρα, καθώς και τη χρονική πληροφορία των μετρήσεων. Μετά την ολοκλήρωση της διαδικασίας δημοσίευσης προστίθενται στην ίδια καταχώριση το hash και το αναγνωριστικό της αντίστοιχης καταχώρισης στο Shimmer.

Η οργάνωση αυτή επιτρέπει όλα τα στοιχεία που σχετίζονται με έναν συγκεκριμένο κύκλο να διατηρούνται ως ενιαία καταχώριση και να μπορούν να ανακτηθούν για μεταγενέστερο έλεγχο.

3.5.2 Τοπική αποθήκευση

Η τοπική βάση δεδομένων αποτελεί το κύριο αποθετήριο των περιβαλλοντικών δεδομένων. Σε αυτήν αποθηκεύεται το πλήρες περιεχόμενο κάθε κύκλου μαζί με το αντίστοιχο hash και το Block ID που προκύπτει από τη δημοσίευση στο DLT.

Η επιλογή αυτή διαχωρίζει το πλήρες περιεχόμενο των μετρήσεων από την καταχώριση που χρησιμοποιείται ως ανεξάρτητο σημείο αναφοράς. Το DLT δεν λειτουργεί επομένως ως υποκατάστατο της βάσης δεδομένων, αλλά ως συμπληρωματικό επίπεδο επαλήθευσης.

3.5.3 Δημιουργία κρυπτογραφικού αποτυπώματος

Με την ολοκλήρωση ενός κύκλου δημιουργείται το κρυπτογραφικό του αποτύπωμα πριν προστεθούν στην καταχώριση το ίδιο το hash και το Block ID. Με αυτόν τον τρόπο το hash εξαρτάται μόνο από τα βασικά δεδομένα του κύκλου και όχι από στοιχεία που προκύπτουν μεταγενέστερα από τη διαδικασία δημοσίευσης.

Για τη δημιουργία του hash τα δεδομένα του κύκλου κανονικοποιούνται πρώτα, ώστε οι τιμές να χρησιμοποιούνται με προκαθορισμένους τύπους και μορφή. Στη συνέχεια δημιουργείται μία καθορισμένη αναπαράσταση JSON και εφαρμόζεται ο αλγόριθμος SHA-256.

Η συγκεκριμένη διαδικασία εξασφαλίζει ότι τα ίδια δεδομένα παράγουν με συνεπή τρόπο το ίδιο κρυπτογραφικό αποτύπωμα, γεγονός που είναι απαραίτητο για τους μεταγενέστερους ελέγχους ακεραιότητας.

3.5.4 Συσχέτιση hash και Transaction ID

Μετά τη δημιουργία του hash, το αποτύπωμα δημοσιεύεται στο DLT μέσω καταχώρησης δεδομένων. Η επιτυχής δημοσίευση επιστρέφει το αντίστοιχο Block ID, το οποίο προστίθεται στην τοπική καταχώριση πριν αυτή αποθηκευτεί στη βάση.

Η σχέση αυτή επιτρέπει την αντιστοίχιση ενός συγκεκριμένου κύκλου της τοπικής βάσης με τη σχετική καταχώριση στο Shimmer. Με αυτόν τον τρόπο, η τοπική βάση, το hash και το Block ID αποτελούν συνδεδεμένα στοιχεία της ίδιας καταχώρισης και μπορούν να χρησιμοποιηθούν από τον μηχανισμό επαλήθευσης.

3.6 Σχεδιασμός διασύνδεσης με το DLT

Η διασύνδεση του συστήματος με το DLT σχεδιάστηκε ως ξεχωριστό υποσύστημα, ώστε η συλλογή και επεξεργασία των περιβαλλοντικών δεδομένων να παραμένει διακριτή από τη διαδικασία δημοσίευσης στο κατανεμημένο δίκτυο. Ο Gateway είναι υπεύθυνος για τη δημιουργία του κρυπτογραφικού αποτυπώματος, ενώ η επικοινωνία με το Shimmer πραγματοποιείται μέσω ειδικού client.

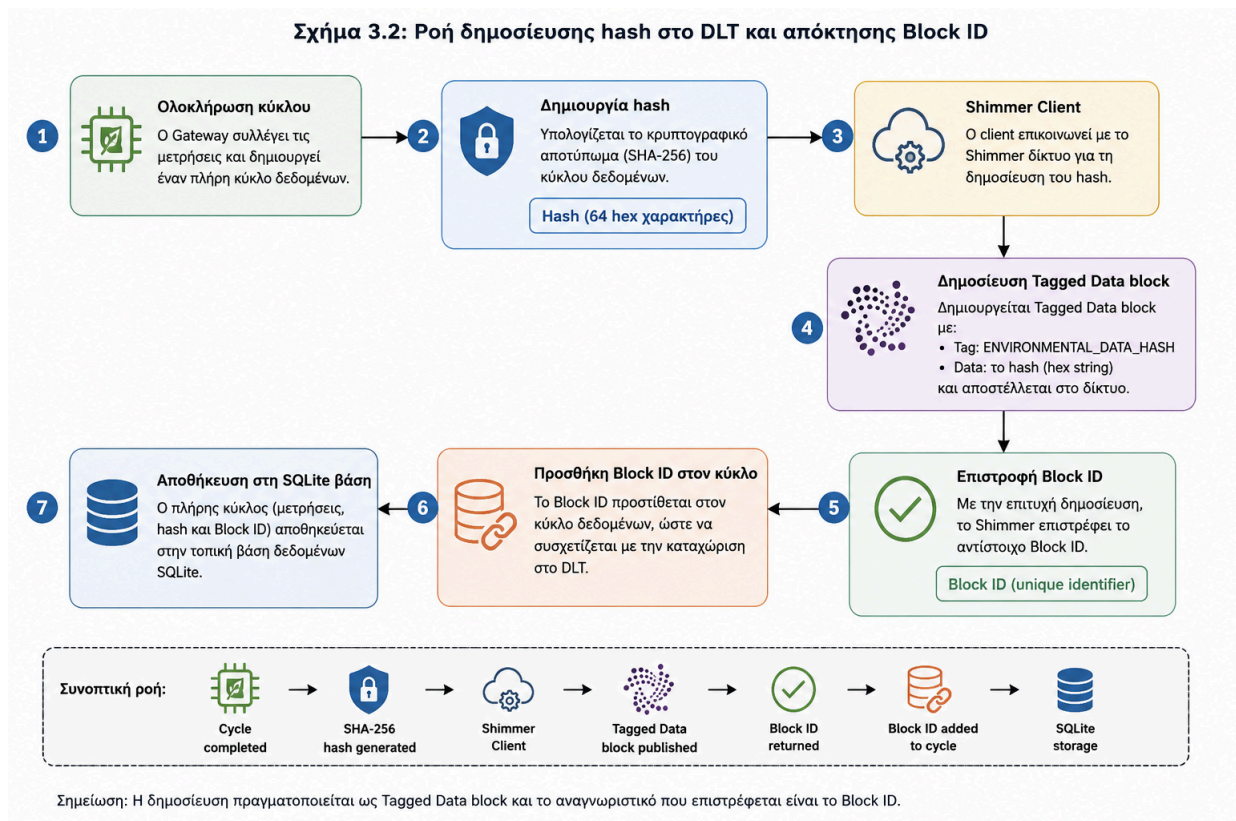
Μετά τη δημιουργία του hash, το DLT υποσύστημα αναλαμβάνει τη δημιουργία και δημοσίευση μιας καταχώρισης δεδομένων στο δίκτυο. Στην υλοποίηση χρησιμοποιείται Tagged Data block, στο οποίο το hash τοποθετείται ως data payload και χρησιμοποιείται προκαθορισμένο tag για την αναγνώριση της σχετικής πληροφορίας.

Με την επιτυχή δημοσίευση επιστρέφεται το αντίστοιχο Block ID. Το συγκεκριμένο αναγνωριστικό μεταφέρεται πίσω στον Gateway και αποθηκεύεται μαζί με τον αντίστοιχο κύκλο στη βάση δεδομένων. Με αυτόν τον τρόπο δημιουργείται σύνδεση μεταξύ της τοπικής καταχώρισης και της αντίστοιχης εγγραφής στο DLT.

Η σχεδίαση αυτή επιτρέπει επίσης την ανεξάρτητη ανάκτηση της συγκεκριμένης καταχώρισης σε μεταγενέστερο χρόνο. Ο Shimmer client παρέχει λειτουργία ανάκτησης block με βάση το Block ID, η οποία χρησιμοποιείται από τον μηχανισμό επαλήθευσης για να ανακτήσει το δημοσιευμένο περιεχόμενο.

Σε επίπεδο ροής, η διασύνδεση ακολουθεί τα εξής στάδια:

1. Ολοκλήρωση του κύκλου δεδομένων.
2. Δημιουργία του SHA-256 hash.
3. Αποστολή του hash στο υποσύστημα Shimmer.
4. Δημιουργία και δημοσίευση Tagged Data block.
5. Επιστροφή του Block ID.
6. Αποθήκευση του Block ID μαζί με τον κύκλο στη SQLite βάση.
7. Δυνατότητα μελλοντικής ανάκτησης του block μέσω του Block ID.



Με τον τρόπο αυτό το DLT αποτελεί ένα ανεξάρτητο επίπεδο της αρχιτεκτονικής, ενώ η τοπική βάση παραμένει υπεύθυνη για τη διατήρηση του πλήρους περιεχομένου των δεδομένων.

3.7 Σχεδιασμός μηχανισμών επαλήθευσης

Η δημιουργία και αποθήκευση ενός κρυπτογραφικού αποτυπώματος αποκτά πρακτική αξία όταν μπορεί να χρησιμοποιηθεί για μεταγενέστερο έλεγχο της ακεραιότητας των δεδομένων. Για τον λόγο αυτό σχεδιάστηκαν μηχανισμοί επαλήθευσης που εξετάζουν τόσο τη συνέπεια των δεδομένων που έχουν αποθηκευτεί τοπικά όσο και τη συμφωνία τους με την αντίστοιχη καταχώριση στο Shimmer DLT.

3.7.1 Έλεγχος τοπικής ακεραιότητας

Κατά τον τοπικό έλεγχο ανακτώνται οι αποθηκευμένοι κύκλοι από τη βάση δεδομένων και δημιουργείται εκ νέου το κρυπτογραφικό αποτύπωμα από τα βασικά δεδομένα κάθε κύκλου.

Το νέο hash συγκρίνεται με το hash που είχε αποθηκευτεί μαζί με την αρχική καταχώριση. Όταν οι δύο τιμές συμφωνούν, δεν υπάρχει ένδειξη ότι τα δεδομένα του κύκλου έχουν μεταβληθεί. Σε διαφορετική περίπτωση, η καταχώριση χαρακτηρίζεται ως αλλοιωμένη.

Ο συγκεκριμένος έλεγχος αποτελεί το πρώτο επίπεδο επαλήθευσης και μπορεί να πραγματοποιηθεί ανεξάρτητα από την παρουσία του DLT. Η υλοποίηση επανυπολογίζει το hash κάθε κύκλου και το συγκρίνει με την αντίστοιχη αποθηκευμένη τιμή.

3.7.2 Επαλήθευση μέσω DLT

Το δεύτερο επίπεδο επαλήθευσης χρησιμοποιεί το Block ID που έχει αποθηκευτεί μαζί με τον κύκλο. Μέσω αυτού του αναγνωριστικού ανακτάται από το Shimmer το αντίστοιχο block και το περιεχόμενο του data payload αποκωδικοποιείται ώστε να εξαχθεί το hash που είχε δημοσιευθεί.

Στη συνέχεια, το hash που υπολογίζεται εκ νέου από τα τοπικά δεδομένα συγκρίνεται με το hash που ανακτήθηκε από το Shimmer. Η συμφωνία των δύο τιμών υποδεικνύει ότι η τρέχουσα κατάσταση των τοπικών δεδομένων αντιστοιχεί στην πληροφορία που είχε δημοσιευθεί στο DLT.

Με αυτόν τον τρόπο ο έλεγχος δεν περιορίζεται μόνο στη σύγκριση με την τοπικά αποθηκευμένη τιμή, αλλά χρησιμοποιεί και ένα εξωτερικό σημείο αναφοράς.

3.7.3 Σχεδιασμός tampering tests

Για την αξιολόγηση του μηχανισμού επαλήθευσης σχεδιάστηκαν δοκιμές σκόπιμης αλλοίωσης δεδομένων. Σε μία υπάρχουσα καταχώριση μεταβάλλεται ελεγχόμενα κάποιο από τα δεδομένα του κύκλου και στη συνέχεια εκτελείται ξανά η διαδικασία επαλήθευσης.

Η μεταβολή αναμένεται να οδηγήσει σε διαφορετικό επανυπολογισμένο hash. Ως αποτέλεσμα, η σύγκριση με το αποθηκευμένο hash πρέπει να αποτύχει και ο συγκεκριμένος κύκλος να χαρακτηριστεί ως αλλοιωμένος.

Παράλληλα, ο έλεγχος έναντι του Shimmer αναμένεται να αποτύχει επίσης, καθώς το hash που υπολογίζεται από τα τροποποιημένα τοπικά δεδομένα δεν θα συμφωνεί με το hash που έχει δημοσιευθεί στο αντίστοιχο block.

Μετά την ολοκλήρωση της δοκιμής προβλέπεται επαναφορά της αρχικής τιμής και επανεκτέλεση του ελέγχου, ώστε να επιβεβαιωθεί ότι ο κύκλος επιστρέφει σε έγκυρη κατάσταση.

3.8 Σχεδιασμός του Launcher

Για τη διαχείριση της εκτέλεσης του project σχεδιάστηκε ξεχωριστό γραφικό περιβάλλον, το οποίο λειτουργεί ως ενιαίο σημείο ελέγχου των διεργασιών της εφαρμογής. Ο Launcher δεν συμμετέχει στη ροή των περιβαλλοντικών δεδομένων και επομένως διατηρεί διακριτό ρόλο από τα επιμέρους components του συστήματος.

Βασικός στόχος του σχεδιασμού ήταν η αποφυγή χειροκίνητης εκκίνησης κάθε διεργασίας και η παροχή ενιαίου μηχανισμού ελέγχου της κατάστασής τους.

Ιδιαίτερη έμφαση δόθηκε στη σωστή ακολουθία εκκίνησης. Ο Publisher δεν πρέπει να ξεκινά απλώς μετά από μία προκαθορισμένη χρονική καθυστέρηση, αλλά μόνο όταν έχει επιβεβαιωθεί ότι ο Gateway έχει ολοκληρώσει την αρχικοποίησή του και είναι έτοιμος να λειτουργήσει. Για τον λόγο αυτό σχεδιάστηκε μηχανισμός ενημέρωσης κατάστασης, μέσω του οποίου ο Launcher λαμβάνει επιβεβαίωση ετοιμότητας πριν προχωρήσει στην εκκίνηση του Publisher.

Παράλληλα, ο Launcher σχεδιάστηκε ώστε να μπορεί να τερματίζει τις διεργασίες που αποτελούν μέρος του project. Ο MQTT Broker αντιμετωπίζεται ως ανεξάρτητη υπηρεσία του περιβάλλοντος εκτέλεσης και δεν περιλαμβάνεται στις διεργασίες που διαχειρίζεται ο Launcher.

Το γραφικό περιβάλλον παρέχει επίσης πληροφορίες σχετικά με την κατάσταση των επιμέρους components και καταγράφει βασικές ενέργειες κατά την εκτέλεση, ώστε ο χρήστης να μπορεί να παρακολουθεί τη λειτουργία του συστήματος από ένα ενιαίο σημείο.

3.9 Συνολική ροή λειτουργίας του συστήματος

Η συνολική λειτουργία του συστήματος ακολουθεί μία συγκεκριμένη ακολουθία, η οποία συνδέει τα στάδια παραγωγής, μεταφοράς, αποθήκευσης, δημοσίευσης και επαλήθευσης των περιβαλλοντικών δεδομένων.

Η κύρια ροή ξεκινά από την παραγωγή των μετρήσεων από τους προσομοιωμένους αισθητήρες. Οι τιμές μεταδίδονται μέσω του MQTT Broker και συλλέγονται από τον Gateway, ο οποίος περιμένει μέχρι να συγκεντρωθούν οι απαιτούμενες μετρήσεις για τη δημιουργία ενός ολοκληρωμένου κύκλου δεδομένων.

Με την ολοκλήρωση του κύκλου υπολογίζεται το αντίστοιχο SHA-256 hash. Στη συνέχεια ο κύκλος αποθηκεύεται στη βάση δεδομένων SQLite, ενώ το ίδιο hash δημοσιεύεται στο Shimmer μέσω Tagged Data block. Από την επιτυχή δημοσίευση επιστρέφεται το αντίστοιχο Block ID, το οποίο συνδέεται με τον κύκλο και αποθηκεύεται μαζί με τα υπόλοιπα στοιχεία της καταχώρισης. Η συγκεκριμένη σειρά ενεργειών ακολουθεί την τελική υλοποίηση του Gateway και του Shimmer client.

Σε μεταγενέστερο χρόνο μπορεί να εκτελεστεί η διαδικασία επαλήθευσης. Αρχικά υπολογίζεται εκ νέου το hash από τα τοπικά αποθηκευμένα δεδομένα και συγκρίνεται με την τιμή που βρίσκεται στη βάση. Στη συνέχεια, μέσω του Block ID, ανακτάται το αντίστοιχο block από το Shimmer και εξάγεται

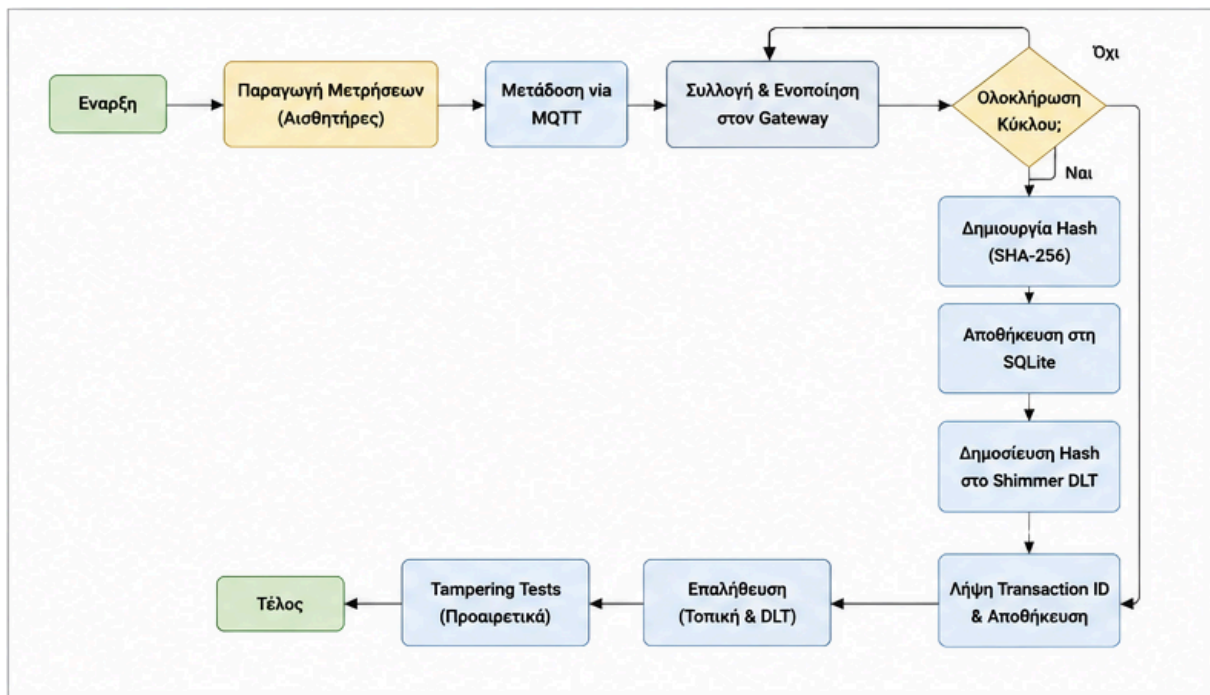
το hash που έχει δημοσιευθεί. Το αποτέλεσμα συγκρίνεται με το νέο υπολογισμένο hash, ώστε να ελεγχθεί η συμφωνία μεταξύ των τοπικών δεδομένων και της καταχώρισης στο DLT.

Η ίδια διαδικασία μπορεί να χρησιμοποιηθεί και για δοκιμές σκόπιμης αλλοίωσης. Σε αυτή την περίπτωση τροποποιείται μία υπάρχουσα τιμή και εκτελείται ξανά η επαλήθευση, με στόχο να διαπιστωθεί ότι η μεταβολή οδηγεί σε αποτυχία του ελέγχου.

Ο Launcher υποστηρίζει την εκτέλεση της συνολικής διαδικασίας, διαχειριζόμενος τις βασικές διεργασίες της εφαρμογής και ελέγχοντας τη σωστή σειρά εκκίνησής τους. Ο Gateway ενημερώνει τον Launcher όταν έχει ολοκληρώσει την αρχικοποίησή του και ο Publisher ξεκινά μετά την παραλαβή της κατάστασης **READY**.

Η συνολική ροή μπορεί να αποτυπωθεί ως εξής:

Παραγωγή μετρήσεων → MQTT → Gateway → Ολοκλήρωση κύκλου → SHA-256 → SQLite → Tagged Data block στο Shimmer → Block ID → Επαλήθευση



Σχήμα 3.3. Συνολική ροή λειτουργίας του συστήματος.

Με τη συγκεκριμένη οργάνωση, τα επιμέρους στάδια της εφαρμογής συνδέονται σε μία ενιαία διαδικασία, από τη συλλογή των αρχικών μετρήσεων έως τον έλεγχο της ακεραιότητας των αποθηκευμένων δεδομένων.

3.10 Επίλογος

Στο παρόν κεφάλαιο καθορίστηκαν οι απαιτήσεις και οι βασικές σχεδιαστικές επιλογές της προτεινόμενης λύσης. Η ανάλυση των απαιτήσεων αποτέλεσε τη βάση για τη διαμόρφωση της αρχιτεκτονικής, της ροής δεδομένων και των μηχανισμών αποθήκευσης και επαλήθευσης.

Παράλληλα, καθορίστηκε ο ρόλος του DLT ως ανεξάρτητου σημείου αναφοράς, καθώς και ο τρόπος με τον οποίο συνδέεται με τις τοπικές καταχωρίσεις μέσω του hash και του Transaction ID. Τέλος, καθορίστηκε η λογική λειτουργίας του Launcher και η απαιτούμενη σειρά εκκίνησης των επιμέρους διεργασιών.

Οι παραπάνω σχεδιαστικές αποφάσεις αποτελούν τη βάση για την πραγματική ανάπτυξη του συστήματος. Στο επόμενο κεφάλαιο παρουσιάζεται η υλοποίηση των επιμέρους components και η μετατροπή του σχεδιασμού σε λειτουργικό λογισμικό.

Κεφάλαιο 4ο: Υλοποίηση του Συστήματος.

4.1 Εισαγωγή

Μετά την ανάλυση των απαιτήσεων και τον σχεδιασμό της αρχιτεκτονικής του συστήματος, στο παρόν κεφάλαιο παρουσιάζεται η υλοποίηση των επιμέρους components της εφαρμογής. Η ανάπτυξη πραγματοποιήθηκε με τρόπο σταδιακό, έτσι ώστε κάθε βασική λειτουργία να μπορεί να ελεγχθεί ξεχωριστά πριν ενσωματωθεί στο συνολικό σύστημα.

Αρχικά παρουσιάζεται το περιβάλλον ανάπτυξης και τα βασικά εργαλεία που χρησιμοποιήθηκαν. Στη συνέχεια αναλύεται η υλοποίηση της προσομοίωσης των αισθητήρων και της επικοινωνίας μέσω MQTT, καθώς και η λειτουργία του Gateway ως κεντρικού σημείου συλλογής και επεξεργασίας των δεδομένων.

Ακολούθως παρουσιάζονται η υλοποίηση της τοπικής βάσης δεδομένων, η διαδικασία δημιουργίας του κρυπτογραφικού αποτυπώματος και η πραγματική διασύνδεση με το δίκτυο Shimmer μέσω του IOTA SDK. Ιδιαίτερη αναφορά γίνεται στους μηχανισμούς επαλήθευσης και στις δοκιμές σκόπιμης αλλοίωσης που χρησιμοποιήθηκαν για τον έλεγχο της ακεραιότητας.

Τέλος, παρουσιάζεται η υλοποίηση του γραφικού Launcher, ο οποίος αναλαμβάνει τη διαχείριση των διεργασιών του project, καθώς και η δημιουργία της εκτελέσιμης έκδοσης της εφαρμογής. Με τον τρόπο αυτό παρουσιάζεται η μετάβαση από τον σχεδιασμό που περιγράφηκε στο προηγούμενο κεφάλαιο σε ένα ολοκληρωμένο και λειτουργικό σύστημα.

4.2 Περιβάλλον ανάπτυξης και χρησιμοποιούμενα εργαλεία

Η ανάπτυξη και δοκιμή του συστήματος πραγματοποιήθηκε σε περιβάλλον Windows, με χρήση της Python ως βασικής γλώσσας προγραμματισμού. Η Python χρησιμοποιήθηκε για την υλοποίηση των επιμέρους components της εφαρμογής και για τη σύνδεση μεταξύ των διαφορετικών επιπέδων του συστήματος.

Για την επικοινωνία μεταξύ των επιμέρους διεργασιών χρησιμοποιήθηκε η βιβλιοθήκη Paho MQTT [25]. Ο Publisher χρησιμοποιεί MQTT client για τη δημοσίευση των μετρήσεων, ενώ ο Gateway χρησιμοποιεί αντίστοιχο client για την παραλαβή και επεξεργασία τους. Η επικοινωνία πραγματοποιείται μέσω του MQTT Broker, ο οποίος λειτουργεί τοπικά στη διεύθυνση localhost και στη θύρα 1883.

Ως MQTT Broker χρησιμοποιήθηκε το Mosquitto [26]. Ο Publisher και ο Gateway επικοινωνούν μέσω των topics που έχουν οριστεί για τις τρεις περιβαλλοντικές παραμέτρους, ενώ ο Gateway χρησιμοποιεί το topic filter environment/# για την παραλαβή των σχετικών μηνυμάτων.

Για την τοπική αποθήκευση των δεδομένων χρησιμοποιήθηκε η SQLite [24]. Η βάση περιλαμβάνει τον πίνακα environment_data και χρησιμοποιείται για τη μόνιμη αποθήκευση των ολοκληρωμένων κύκλων, μαζί με το αντίστοιχο hash και το αναγνωριστικό της σχετικής καταχώρισης στο Shimmer.

Για τη δημιουργία των κρυπτογραφικών αποτυπωμάτων χρησιμοποιήθηκε η βιβλιοθήκη hashlib της Python. Η υλοποίηση περιλαμβάνει κανονικοποίηση των δεδομένων, δημιουργία σταθερής JSON αναπαράστασης και εφαρμογή του SHA-256, με αποτέλεσμα το τελικό hash σε δεκαεξαδική μορφή.

Η διασύνδεση με το δίκτυο Shimmer υλοποιήθηκε μέσω του IOTA SDK. Ο ShimmerClient χρησιμοποιεί δημόσιο node του δικτύου και παρέχει λειτουργίες για τον έλεγχο της σύνδεσης, τη δημοσίευση του hash ως Tagged Data block και την ανάκτηση block μέσω του αντίστοιχου Block ID.

Για την ανάπτυξη του γραφικού περιβάλλοντος χρησιμοποιήθηκε το CustomTkinter, ενώ η βιβλιοθήκη Pillow χρησιμοποιείται για τη φόρτωση των γραφικών πόρων του Launcher. Το περιβάλλον περιλαμβάνει τις βασικές λειτουργίες διαχείρισης του project, τις ενδείξεις κατάστασης και το activity log.

Για τη διαχείριση των διεργασιών χρησιμοποιήθηκε η βιβλιοθήκη psutil, σε συνδυασμό με το subprocess. Ο ProcessManager αξιοποιεί τις βιβλιοθήκες αυτές για την εκκίνηση, τον εντοπισμό και τον τερματισμό των επιμέρους διεργασιών της εφαρμογής.

Τέλος, για τη δημιουργία της εκτελέσιμης έκδοσης του Launcher χρησιμοποιήθηκε το PyInstaller. Η διαδικασία αυτή επιτρέπει τη συσκευασία της εφαρμογής μαζί με τους απαραίτητους πόρους της, ώστε να μπορεί να εκτελείται ως Windows εφαρμογή.

Οι παραπάνω τεχνολογίες χρησιμοποιήθηκαν συμπληρωματικά και αποτέλεσαν το βασικό τεχνολογικό περιβάλλον μέσα στο οποίο υλοποιήθηκε το τελικό σύστημα.

4.3 Υλοποίηση της προσομοίωσης των αισθητήρων

Η προσομοίωση των περιβαλλοντικών αισθητήρων υλοποιήθηκε με τρεις ανεξάρτητες κλάσεις Python, μία για κάθε παράμετρο που παρακολουθείται: θερμοκρασία, υγρασία και ποιότητα αέρα. Κάθε αισθητήρας διατηρεί εσωτερικά την τρέχουσα τιμή του, εφαρμόζει μία ελεγχόμενη μεταβολή σε κάθε νέα μέτρηση και επιστρέφει ένα κοινό δομημένο αποτέλεσμα, το οποίο μπορεί στη συνέχεια να μεταδοθεί μέσω MQTT.

Οι τρεις αισθητήρες χρησιμοποιούν αρχικά διαφορετικές τιμές εκκίνησης και λειτουργούν με συγκεκριμένα όρια, ώστε οι παραγόμενες μετρήσεις να παραμένουν μέσα σε προκαθορισμένα επίπεδα. Επιπλέον, κάθε μέτρηση συνοδεύεται από πληροφορίες αναγνώρισης του αισθητήρα, τον τύπο της μέτρησης, τη θέση εγκατάστασης, τη μονάδα μέτρησης, την κατάσταση και χρονική σήμανση.

4.3.1 Αισθητήρας θερμοκρασίας

Ο αισθητήρας θερμοκρασίας υλοποιείται μέσω της κλάσης TemperatureSensor. Κατά την αρχικοποίησή του χρησιμοποιούνται το αναγνωριστικό TEMP_01, η θέση office_a και αρχική θερμοκρασία ίση με 24,0 °C. Η τρέχουσα θερμοκρασία διατηρείται ως εσωτερική κατάσταση του αισθητήρα, ενώ αποθηκεύεται επίσης η μεταβολή που προέκυψε κατά την τελευταία ενημέρωση.

Σε κάθε νέα μέτρηση η θερμοκρασία μεταβάλλεται με τυχαία επιλογή μεταξύ των τιμών 0.0, +0.1 και -0.1. Οι πιθανότητες είναι 70%, 15% και 15% αντίστοιχα, με αποτέλεσμα οι διαδοχικές μετρήσεις να παρουσιάζουν μικρές μεταβολές. Στη συνέχεια εφαρμόζεται περιορισμός της τιμής στο εύρος από 18,0 °C έως 30,0 °C.

Για την περιγραφή της κατάστασης του αισθητήρα χρησιμοποιείται η μέθοδος `get_status()`. Τιμές από 28 °C και πάνω χαρακτηρίζονται ως `warning`, ενώ η κλάση περιλαμβάνει επίσης συνθήκη `critical` για τιμές άνω των 30 °C. Η δεύτερη αυτή κατάσταση δεν μπορεί να ενεργοποιηθεί στην παρούσα διαμόρφωση, καθώς η θερμοκρασία περιορίζεται στο μέγιστο των 30 °C.

Η μέθοδος `generate_reading()` ενημερώνει πρώτα την εσωτερική θερμοκρασία και στη συνέχεια επιστρέφει ένα δομημένο σύνολο δεδομένων που περιλαμβάνει το `sensor_id`, τον τύπο του αισθητήρα, τη θέση, την τιμή της θερμοκρασίας με ακρίβεια ενός δεκαδικού, τη μονάδα μέτρησης, την κατάσταση και χρονική σήμανση σε UTC σε μορφή ISO 8601. Παράλληλα, η μέθοδος `get_last_change()` επιστρέφει τη μεταβολή που πραγματοποιήθηκε κατά την τελευταία ενημέρωση.

Αυτή η υλοποίηση επιτρέπει στον αισθητήρα να παράγει σταδιακά μεταβαλλόμενες τιμές και να διατηρεί εσωτερική κατάσταση μεταξύ διαδοχικών μετρήσεων, αντί να δημιουργεί κάθε φορά μία ανεξάρτητη τυχαία τιμή.

4.3.2 Αισθητήρας υγρασίας

Ο αισθητήρας υγρασίας υλοποιείται μέσω της κλάσης `HumiditySensor`. Κατά την αρχικοποίησή του χρησιμοποιούνται το αναγνωριστικό `HUM_01`, η θέση `office_a` και αρχική τιμή σχετικής υγρασίας ίση με 50,0%. Η τρέχουσα τιμή της υγρασίας διατηρείται εσωτερικά από την κλάση και ενημερώνεται σε κάθε νέα μέτρηση.

Σε κάθε ενημέρωση, η υγρασία μεταβάλλεται μέσω τυχαίας επιλογής μεταξύ των τιμών 0.0, +0.1 και -0.1, με πιθανότητες 70%, 15% και 15% αντίστοιχα. Παράλληλα, λαμβάνεται υπόψη η μεταβολή της θερμοκρασίας από τον προηγούμενο αισθητήρα. Η επίδραση αυτή υπολογίζεται ως το αντίθετο της μεταβολής της θερμοκρασίας, ώστε μία αύξηση της θερμοκρασίας να οδηγεί σε αντίστοιχη μείωση της προσομοιωμένης υγρασίας και αντίστροφα.

Η τελική τιμή της υγρασίας περιορίζεται στο εύρος από 30,0% έως 80,0%. Για την κατάσταση του αισθητήρα χρησιμοποιείται η μέθοδος `get_status()`. Όταν η υγρασία είναι μικρότερη από 35% ή μεγαλύτερη από 75%, η κατάσταση χαρακτηρίζεται ως `warning`, ενώ στο ενδιάμεσο εύρος χαρακτηρίζεται ως `normal`.

Η μέθοδος `generate_reading()` ενημερώνει την εσωτερική τιμή της υγρασίας και επιστρέφει ένα δομημένο σύνολο δεδομένων που περιλαμβάνει το `sensor_id`, τον τύπο του αισθητήρα, τη θέση, την τιμή της υγρασίας με ακρίβεια ενός δεκαδικού, τη μονάδα %, την κατάσταση και χρονική σήμανση σε UTC σε μορφή ISO 8601.

Με αυτόν τον τρόπο ο αισθητήρας υγρασίας παράγει σταδιακά μεταβαλλόμενες τιμές και διατηρεί μία απλή εξάρτηση από τη μεταβολή της θερμοκρασίας, ενισχύοντας τη ρεαλιστικότητα της μεταξύ τους σχέσης στην προσομοίωση.

4.3.3 Αισθητήρας ποιότητας αέρα

Ο αισθητήρας ποιότητας αέρα υλοποιείται μέσω της κλάσης `AirQualitySensor`. Κατά την αρχικοποίησή του χρησιμοποιούνται το αναγνωριστικό `AQ_01`, η θέση `office_a` και αρχική τιμή δείκτη ποιότητας αέρα ίση με 60. Η τρέχουσα τιμή του δείκτη διατηρείται εσωτερικά από την κλάση και ενημερώνεται σε κάθε νέα μέτρηση.

Σε κάθε ενημέρωση, η τιμή του δείκτη μεταβάλλεται μέσω τυχαίας επιλογής μεταξύ των τιμών 0.0, +1.0 και -1.0, με πιθανότητες 70%, 15% και 15% αντίστοιχα. Επιπλέον, η μεταβολή επηρεάζεται από τις τρέχουσες τιμές θερμοκρασίας και υγρασίας. Όταν η θερμοκρασία ξεπερνά τους 26 °C προστίθεται

μία μονάδα στην τιμή του δείκτη, ενώ το ίδιο συμβαίνει όταν η υγρασία είναι μικρότερη από 40%. Οι δύο αυτές συνθήκες μπορούν να συμβάλουν ταυτόχρονα στην αύξηση της τιμής.

Η τιμή του δείκτη περιορίζεται στο εύρος από 0 έως 500. Η κατάσταση του αισθητήρα καθορίζεται από τη μέθοδο `get_status()`: τιμές μεγαλύτερες από 150 χαρακτηρίζονται ως *critical*, τιμές μεγαλύτερες από 100 ως *warning*, ενώ οι υπόλοιπες χαρακτηρίζονται ως *normal*.

Η μέθοδος `generate_reading()` δέχεται ως είσοδο την τρέχουσα θερμοκρασία και υγρασία, ενημερώνει την εσωτερική τιμή του δείκτη και επιστρέφει ένα δομημένο σύνολο δεδομένων. Η επιστρεφόμενη καταχώριση περιλαμβάνει το `sensor_id`, τον τύπο του αισθητήρα, τη θέση, την τιμή AQI ως ακέραια τιμή, τη μονάδα AQI, την κατάσταση και χρονική σήμανση σε UTC σε μορφή ISO 8601.

Με αυτόν τον τρόπο ο αισθητήρας ποιότητας αέρα δεν λειτουργεί ως πλήρως ανεξάρτητη τυχαία πηγή, αλλά λαμβάνει υπόψη δύο από τις υπόλοιπες περιβαλλοντικές παραμέτρους. Η προσέγγιση αυτή δημιουργεί μία απλή σχέση μεταξύ των προσομοιωμένων μεγεθών και επιτρέπει τη δημιουργία περισσότερο συνεκτικής σειράς μετρήσεων.

4.3.4 Παραγωγή και χρονική συμπεριφορά των μετρήσεων

Οι τρεις αισθητήρες λειτουργούν ως διαδοχικά συνδεδεμένα τμήματα της διαδικασίας προσομοίωσης και όχι ως πλήρως ανεξάρτητες πηγές τιμών. Η ακολουθία υλοποιείται στον *Publisher*, όπου πρώτα δημιουργείται νέα μέτρηση θερμοκρασίας και στη συνέχεια η μεταβολή της χρησιμοποιείται ως είσοδος για την ενημέρωση της υγρασίας. Στο επόμενο στάδιο, οι τρέχουσες τιμές θερμοκρασίας και υγρασίας χρησιμοποιούνται από τον αισθητήρα ποιότητας αέρα.

Η συγκεκριμένη σειρά δημιουργεί μία απλή αλληλεξάρτηση μεταξύ των περιβαλλοντικών μεγεθών. Η υγρασία επηρεάζεται από τη μεταβολή της θερμοκρασίας, ενώ ο δείκτης ποιότητας αέρα επηρεάζεται από τις απόλυτες τιμές θερμοκρασίας και υγρασίας. Έτσι, οι τρεις μετρήσεις δεν αποτελούν τρεις ανεξάρτητες τυχαίες ακολουθίες, αλλά σχηματίζουν μία πιο συνεκτική προσομοίωση.

Μετά την παραγωγή των τριών μετρήσεων, ο *Publisher* τις δημοσιεύει στα αντίστοιχα MQTT topics. Η διαδικασία επαναλαμβάνεται συνεχώς και, μετά την ολοκλήρωση μίας επανάληψης, πραγματοποιείται αναμονή 60 δευτερολέπτων πριν δημιουργηθεί το επόμενο σύνολο τιμών.

Η χρονική αυτή οργάνωση επιτρέπει στο σύστημα να λειτουργεί ως συνεχής προσομοίωση περιβαλλοντικής παρακολούθησης, με διαδοχικές μετρήσεις και μικρές μεταβολές μεταξύ των κύκλων. Ο *Gateway* αξιοποιεί στη συνέχεια τις τρεις εισερχόμενες μετρήσεις για τη δημιουργία του αντίστοιχου ολοκληρωμένου κύκλου.

4.3.5 Μορφή των παραγόμενων μηνυμάτων

Κάθε αισθητήρας επιστρέφει μία δομημένη καταχώριση με κοινή μορφή πεδίων, ανεξάρτητα από τον τρόπο με τον οποίο υπολογίζεται εσωτερικά η αντίστοιχη μέτρηση. Στα κοινά πεδία περιλαμβάνονται το `sensor_id`, το `sensor_type`, το `location`, το `value`, το `unit`, το `status` και το `timestamp`.

Η κοινή δομή διευκολύνει την περαιτέρω επεξεργασία των δεδομένων, καθώς ο *Publisher* μπορεί να χρησιμοποιεί τον ίδιο μηχανισμό για τη μετατροπή και δημοσίευση των μετρήσεων. Κάθε αποτέλεσμα που επιστρέφεται από τους αισθητήρες μετατρέπεται σε JSON string και αποστέλλεται στο αντίστοιχο MQTT topic.

Ιδιαίτερη σημασία έχει το timestamp, το οποίο δημιουργείται κατά την παραγωγή κάθε μέτρησης με χρήση UTC και αποθηκεύεται σε μορφή ISO 8601. Έτσι, κάθε μήνυμα περιλαμβάνει τη χρονική πληροφορία της στιγμής κατά την οποία δημιουργήθηκε η συγκεκριμένη μέτρηση.

Στην πλευρά του Gateway, το JSON payload αποκωδικοποιείται και μετατρέπεται ξανά σε δομή δεδομένων. Ο Gateway χρησιμοποιεί από το μήνυμα κυρίως τα πεδία value και timestamp, ενώ το topic καθορίζει σε ποια περιβαλλοντική παράμετρο αντιστοιχεί η μέτρηση.

Η κοινή αυτή μορφή επιτρέπει στα τρία διαφορετικά είδη αισθητήρων να εντάσσονται στην ίδια διαδικασία μετάδοσης και επεξεργασίας, παρότι η εσωτερική λογική παραγωγής των τιμών τους διαφέρει.

4.4 Υλοποίηση της επικοινωνίας MQTT

Η επικοινωνία μεταξύ των προσομοιωμένων αισθητήρων και του Gateway υλοποιήθηκε μέσω του πρωτοκόλλου MQTT. Ο Publisher αναλαμβάνει την παραγωγή και δημοσίευση των μετρήσεων, ενώ ο Gateway λειτουργεί ως subscriber που παραλαμβάνει τα αντίστοιχα μηνύματα μέσω του MQTT Broker.

4.4.1 MQTT Publisher

Ο Publisher αποτελεί το σημείο στο οποίο δημιουργούνται οι περιβαλλοντικές μετρήσεις και μετατρέπονται σε MQTT μηνύματα. Κατά την εκκίνησή του δημιουργείται ένας MQTT client και πραγματοποιείται σύνδεση στον broker μέσω της διεύθυνσης localhost και της θύρας 1883. Στη συνέχεια αρχικοποιούνται οι τρεις αισθητήρες που χρησιμοποιούνται στην προσομοίωση.

Για κάθε περιβαλλοντική παράμετρο έχει οριστεί ξεχωριστό topic:

- environment/temperature
- environment/humidity
- environment/airquality

Τα topics διατηρούνται σε κοινή δομή, ώστε η δημοσίευση των τριών μετρήσεων να πραγματοποιείται με τον ίδιο μηχανισμό.

Κατά τη λειτουργία του Publisher δημιουργείται αρχικά η νέα μέτρηση θερμοκρασίας. Η μεταβολή της χρησιμοποιείται στη συνέχεια από τον αισθητήρα υγρασίας, ενώ οι νέες τιμές θερμοκρασίας και υγρασίας δίνονται ως είσοδοι στον αισθητήρα ποιότητας αέρα.

Οι τρεις καταχωρίσεις μετατρέπονται σε JSON και δημοσιεύονται διαδοχικά στα αντίστοιχα MQTT topics.

4.4.2 Topics και μορφή μηνυμάτων

Η χρήση ξεχωριστού topic για κάθε αισθητήρα επιτρέπει στον Gateway να αναγνωρίζει την προέλευση κάθε εισερχόμενης μέτρησης. Η δομή των μηνυμάτων παραμένει κοινή και περιλαμβάνει τα πεδία sensor_id, sensor_type, location, value, unit, status και timestamp.

Ο Publisher μετατρέπει κάθε καταχώριση που επιστρέφεται από τον αντίστοιχο αισθητήρα σε JSON string πριν από τη δημοσίευση. Έτσι, η ίδια διαδικασία αποστολής μπορεί να χρησιμοποιηθεί και για τις τρεις κατηγορίες δεδομένων.

4.4.3 MQTT Broker

Ως MQTT Broker χρησιμοποιείται το Mosquitto, το οποίο λειτουργεί ως ενδιάμεσο επίπεδο επικοινωνίας μεταξύ Publisher και Gateway. Ο Publisher αποστέλλει τα μηνύματα στον broker, ενώ ο Gateway έχει εγγραφεί στο topic filter environment/# και λαμβάνει τα σχετικά μηνύματα.

Η χρήση του topic filter επιτρέπει στον Gateway να παρακολουθεί συνολικά την ιεραρχία environment/, χωρίς να απαιτείται ξεχωριστό subscribe για κάθε αισθητήρα.

4.4.4 Περιοδική δημοσίευση μετρήσεων

Μετά τη δημοσίευση των τριών μετρήσεων, ο Publisher πραγματοποιεί αναμονή 60 δευτερολέπτων πριν ξεκινήσει την επόμενη επανάληψη. Με αυτόν τον τρόπο δημιουργείται μία σταθερή περιοδικότητα στην παραγωγή των περιβαλλοντικών δεδομένων.

Ο Gateway παραλαμβάνει τις μετρήσεις ως ξεχωριστά MQTT μηνύματα και διατηρεί προσωρινά τις τελευταίες διαθέσιμες τιμές. Η δημιουργία ολοκληρωμένου κύκλου πραγματοποιείται μόνο όταν έχουν παραληφθεί οι μετρήσεις και των τριών αισθητήρων.

Με αυτόν τον τρόπο η επικοινωνία MQTT περιορίζεται στον ρόλο της μεταφοράς των δεδομένων, ενώ η ενοποίηση και η δημιουργία κύκλων αναλαμβάνονται από τον Gateway.

4.5 Υλοποίηση του Gateway

Ο Gateway αποτελεί το κεντρικό σημείο συλλογής και επεξεργασίας των περιβαλλοντικών δεδομένων. Είναι υπεύθυνος για τη λήψη των MQTT μηνυμάτων, την προσωρινή συγκέντρωση των επιμέρους μετρήσεων, τη δημιουργία ολοκληρωμένων κύκλων και την ενεργοποίηση των επόμενων σταδίων της διαδικασίας. Συγκεκριμένα, μετά τη δημιουργία ενός κύκλου πραγματοποιούνται ο υπολογισμός του κρυπτογραφικού αποτυπώματος, η δημοσίευσή του στο Shimmer και η αποθήκευση της ολοκληρωμένης καταχώρισης στη βάση δεδομένων.

4.5.1 Αρχικοποίηση και σύνδεση στον MQTT Broker

Κατά την εκκίνησή του, ο Gateway αρχικοποιεί πρώτα τη βάση δεδομένων και στη συνέχεια δημιουργεί έναν MQTT client με τη χρήση της Paho MQTT. Ορίζονται οι callbacks on_connect() και on_message() για τη διαχείριση της σύνδεσης και των εισερχόμενων μηνυμάτων. Στη συνέχεια πραγματοποιείται σύνδεση στον MQTT Broker στη διεύθυνση localhost και στη θύρα 1883, ενώ ξεκινά η κύρια loop επικοινωνίας.

Μετά την επιτυχή σύνδεση, η on_connect() πραγματοποιεί subscribe στο topic filter environment/#, επιτρέποντας τη λήψη των μηνυμάτων και των τριών αισθητήρων. Παράλληλα, δημοσιεύει στο topic gateway/status την τιμή READY, μέσω της οποίας ενημερώνεται ο Launcher ότι ο Gateway έχει ολοκληρώσει την αρχικοποίησή του.

4.5.2 Λήψη και επεξεργασία μηνυμάτων

Η παραλαβή κάθε εισερχόμενου MQTT μηνύματος πραγματοποιείται από τη συνάρτηση on_message(). Αρχικά, το payload αποκωδικοποιείται από bytes σε κείμενο και στη συνέχεια μετατρέπεται από JSON σε Python dictionary. Παράλληλα, καταγράφεται το topic του μηνύματος.

Το timestamp του συγκεκριμένου μηνύματος αποθηκεύεται προσωρινά στη δομή `latest_data`. Στη συνέχεια, ανάλογα με το `topic`, η τιμή του μηνύματος αντιστοιχίζεται στο κατάλληλο πεδίο της δομής, δηλαδή `temperature`, `humidity` ή `air_quality`.

4.5.3 Διαχείριση των τρεχουσών μετρήσεων

Για την προσωρινή συγκέντρωση των δεδομένων χρησιμοποιείται η δομή `latest_data`. Η δομή αυτή διατηρεί τις τελευταίες διαθέσιμες τιμές των τριών περιβαλλοντικών παραμέτρων και το timestamp του τελευταίου μηνύματος που παραλήφθηκε. Κατά την αρχική κατάσταση τα πεδία έχουν τιμή `None`.

Μετά την επεξεργασία κάθε μηνύματος, ο Gateway ελέγχει εάν έχουν πλέον διαθέσιμη τιμή και τα τρία απαιτούμενα πεδία. Ο κύκλος δημιουργείται μόνο όταν είναι διαθέσιμες οι τιμές θερμοκρασίας, υγρασίας και ποιότητας αέρα.

4.5.4 Δημιουργία ολοκληρωμένου κύκλου

Η δημιουργία του κύκλου πραγματοποιείται μέσω της `create_cycle()`. Η συνάρτηση αυξάνει τον μετρητή κύκλων και δημιουργεί μία νέα δομή που περιλαμβάνει το `cycle_id`, τις τρεις περιβαλλοντικές μετρήσεις και το timestamp που έχει αποθηκευτεί στην `latest_data`.

Η αρχική τιμή του μετρητή λαμβάνεται μέσω της `get_last_cycle_id()`. Με τον τρόπο αυτό η αρίθμηση συνεχίζεται από τον τελευταίο ήδη αποθηκευμένο κύκλο και δεν ξεκινά ξανά από την αρχή μετά από κάθε επανεκκίνηση του Gateway.

4.5.5 Δημιουργία SHA-256 hash

Μετά τη δημιουργία του κύκλου καλείται η `generate_hash()`. Τα δεδομένα του κύκλου κανονικοποιούνται πρώτα σύμφωνα με τους τύπους και τις αριθμητικές μορφές που έχουν καθοριστεί και στη συνέχεια δημιουργείται σταθερή JSON αναπαράσταση με `sort_keys=True`. Ακολούθως εφαρμόζεται ο SHA-256 και παράγεται το τελικό hash σε δεκαεξαδική μορφή.

Ο υπολογισμός του hash πραγματοποιείται πριν προστεθούν στην καταχώρηση του κύκλου το ίδιο το hash και το Block ID.

4.5.6 Δημοσίευση του hash στο Shimmer

Μετά τον υπολογισμό του hash, ο Gateway καλεί τη `publish_hash()` του Shimmer client. Η συνάρτηση αναλαμβάνει τη δημοσίευση του κρυπτογραφικού αποτυπώματος στο δίκτυο και επιστρέφει το αντίστοιχο Block ID.

Η δημοσίευση πραγματοποιείται μέσω Tagged Data block, όπως αναλύεται στην ενότητα 4.8.

4.5.7 Αποθήκευση του Block ID

Μετά την επιτυχή δημοσίευση, το Block ID αποθηκεύεται στην καταχώρηση του κύκλου. Στον κώδικα, η μεταβλητή και το πεδίο της βάσης ονομάζονται `transaction_id`, όμως η τιμή που αποθηκεύεται είναι το Block ID που επιστρέφεται από το `build_and_post_block()`.

Το αναγνωριστικό αυτό επιτρέπει τη μεταγενέστερη ανάκτηση της συγκεκριμένης καταχώρησης από το Shimmer και χρησιμοποιείται από τη διαδικασία επαλήθευσης.

4.5.8 Αποθήκευση του ολοκληρωμένου κύκλου

Αφού ολοκληρωθεί η διαδικασία δημοσίευσης, ο Gateway καλεί την `insert_cycle()` και αποθηκεύει την πλήρη καταχώρηση στη SQLite. Η εγγραφή περιλαμβάνει τα βασικά δεδομένα του κύκλου, το hash και το πεδίο `transaction_id`, στο οποίο αποθηκεύεται το αντίστοιχο Block ID.

Μετά την επιτυχή αποθήκευση, τα προσωρινά πεδία της `latest_data` επαναφέρονται σε None. Έτσι ολοκληρώνεται ο τρέχων κύκλος και ο Gateway είναι έτοιμος να συγκεντρώσει τα δεδομένα του επόμενου.

Η συγκεκριμένη ακολουθία συνδέει το επίπεδο MQTT με την τοπική αποθήκευση και το DLT και αποτελεί τον βασικό μηχανισμό μέσω του οποίου τα επιμέρους μηνύματα μετατρέπονται σε ολοκληρωμένες, επαληθεύσιμες καταχωρίσεις.

4.6 Υλοποίηση της βάσης δεδομένων SQLite

Η τοπική αποθήκευση των περιβαλλοντικών δεδομένων υλοποιήθηκε με χρήση της SQLite. Η βάση αποτελεί το μόνιμο τοπικό αποθετήριο των ολοκληρωμένων κύκλων και διατηρεί, εκτός από τις περιβαλλοντικές μετρήσεις, το κρυπτογραφικό αποτύπωμα και την πληροφορία που επιτρέπει τη σύνδεση με την αντίστοιχη καταχώριση στο Shimmer. Η διαχείριση της βάσης έχει συγκεντρωθεί στο `module database.py`, το οποίο περιλαμβάνει τις βασικές λειτουργίες δημιουργίας της βάσης, εισαγωγής κύκλων και ανάκτησης του τελευταίου `cycle_id`.

4.6.1 Αρχικοποίηση της βάσης

Η βάση δεδομένων αποθηκεύεται στο αρχείο `storage/environmental_data.db`. Η συνάρτηση `initialize_database()` δημιουργεί τη σύνδεση με το αρχείο και εκτελεί εντολή `CREATE TABLE IF NOT EXISTS`, ώστε ο πίνακας να δημιουργείται μόνο όταν δεν υπάρχει ήδη.

Η συγκεκριμένη προσέγγιση επιτρέπει στην εφαρμογή να εκτελεί τη διαδικασία αρχικοποίησης χωρίς να διαγράφονται οι ήδη αποθηκευμένες εγγραφές. Μετά την εκτέλεση της εντολής πραγματοποιείται `commit` και στη συνέχεια κλείνει η σύνδεση με τη βάση.

4.6.2 Δομή του πίνακα `environment_data`

Ο κύριος πίνακας της βάσης ονομάζεται `environment_data` και περιλαμβάνει τα πεδία που απαιτούνται για την πλήρη καταγραφή κάθε κύκλου. Η δομή του παρουσιάζεται στον Πίνακα 4.1.

Πίνακας 4.1: Δομή του πίνακα `environment_data`

Πεδίο	Τύπος	Ρόλος
<code>id</code>	INTEGER	Εσωτερικό πρωτεύον αναγνωριστικό με αυτόματη αύξηση
<code>cycle_id</code>	INTEGER	Αναγνωριστικό του κύκλου δεδομένων
<code>timestamp</code>	TEXT	Χρονική πληροφορία του κύκλου
<code>temperature</code>	REAL	Μετρούμενη θερμοκρασία
<code>humidity</code>	REAL	Μετρούμενη σχετική υγρασία
<code>air_quality</code>	INTEGER	Τιμή ποιότητας αέρα
<code>hash</code>	TEXT	Κρυπτογραφικό αποτύπωμα του κύκλου
<code>transaction_id</code>	TEXT	Αποθηκευμένο Block ID της αντίστοιχης καταχώρισης στο Shimmer

Η δομή αυτή προκύπτει από τη δήλωση CREATE TABLE του τελικού κώδικα. Τα βασικά δεδομένα του κύκλου δηλώνονται ως υποχρεωτικά, ενώ τα πεδία hash και transaction_id χρησιμοποιούνται για τη σύνδεση της τοπικής καταχώρισης με το επίπεδο ακεραιότητας και την αντίστοιχη καταχώρηση στο Shimmer.

Σημειώνεται ότι το transaction_id διατηρείται ως όνομα πεδίου επειδή αποτελεί μέρος της υλοποίησης της βάσης. Η τιμή που αποθηκεύεται στο συγκεκριμένο πεδίο είναι το Block ID που επιστρέφεται από το Shimmer client.

4.6.3 Αποθήκευση κύκλων

Η εισαγωγή ενός ολοκληρωμένου κύκλου πραγματοποιείται μέσω της insert_cycle(). Η συνάρτηση ανοίγει σύνδεση με τη βάση και εκτελεί παραμετροποιημένη εντολή INSERT στον πίνακα environment_data, αντιστοιχίζοντας τις τιμές στα αντίστοιχα πεδία.

Μετά την επιτυχή εκτέλεση πραγματοποιείται commit, ώστε η καταχώριση να αποθηκευτεί μόνιμα, και στη συνέχεια κλείνει η σύνδεση.

Στον Gateway, η insert_cycle() καλείται αφού έχουν ολοκληρωθεί ο υπολογισμός του hash και η δημοσίευση στο Shimmer. Επομένως, η καταχώριση που εισάγεται στη βάση περιλαμβάνει ήδη το hash και το Block ID μαζί με τα βασικά δεδομένα του κύκλου.

4.6.4 Ανάκτηση του τελευταίου cycle_id

Για να συνεχίζεται η αρίθμηση των κύκλων μεταξύ διαφορετικών εκτελέσεων, χρησιμοποιείται η get_last_cycle_id(). Η συνάρτηση εκτελεί ερώτημα SELECT MAX(cycle_id) και επιστρέφει τη μεγαλύτερη διαθέσιμη τιμή. Όταν δεν υπάρχουν ακόμη καταχωρίσεις, επιστρέφει 0.

Ο Gateway χρησιμοποιεί την τιμή αυτή κατά την εκκίνησή του ως αρχική τιμή του cycle_counter και αυξάνει τον μετρητή πριν από τη δημιουργία κάθε νέου κύκλου.

Με αυτόν τον τρόπο, η αρίθμηση των κύκλων συνεχίζεται από το σημείο στο οποίο είχε σταματήσει το σύστημα και δεν επανεκκινεί σε κάθε νέα εκτέλεση.

4.6.5 Μόνιμη διατήρηση των δεδομένων

Η χρήση αρχείου SQLite επιτρέπει τη διατήρηση των δεδομένων ανεξάρτητα από το αν οι διεργασίες του project εκτελούνται εκείνη τη στιγμή. Κάθε κύκλος αποθηκεύεται με commit πριν κλείσει η σύνδεση, επομένως οι καταχωρήσεις παραμένουν διαθέσιμες σε επόμενες εκτελέσεις.

Η μόνιμη αποθήκευση είναι απαραίτητη και για τη διαδικασία επαλήθευσης. Το verify_integrity.py ανακτά τους αποθηκευμένους κύκλους από τον πίνακα environment_data, επανυπολογίζει τα hashes και στη συνέχεια τα συγκρίνει με τις τοπικά αποθηκευμένες τιμές και με το αντίστοιχο hash που ανακτάται από το Shimmer.

Με αυτόν τον τρόπο η SQLite αποτελεί το βασικό τοπικό επίπεδο αποθήκευσης, ενώ το Shimmer χρησιμοποιείται συμπληρωματικά για την εξωτερική καταγραφή του κρυπτογραφικού αποτυπώματος.

4.7 Υλοποίηση της κρυπτογραφικής επεξεργασίας

Η δημιουργία του κρυπτογραφικού αποτυπώματος υλοποιήθηκε σε ξεχωριστό module, ώστε η διαδικασία υπολογισμού του hash να παραμένει ανεξάρτητη από τη συλλογή, την αποθήκευση και τη δημοσίευση των δεδομένων. Η λειτουργία αυτή υλοποιείται από τη συνάρτηση `generate_hash()`, η οποία δέχεται έναν ολοκληρωμένο κύκλο δεδομένων και επιστρέφει το αντίστοιχο SHA-256 hash.

4.7.1 Κανονικοποίηση των δεδομένων

Πριν πραγματοποιηθεί ο υπολογισμός του hash, τα δεδομένα του κύκλου κανονικοποιούνται σε προκαθορισμένη μορφή. Για το `cycle_id` χρησιμοποιείται ακέραιος τύπος, το `timestamp` μετατρέπεται σε κείμενο, ενώ οι τιμές θερμοκρασίας και υγρασίας μετατρέπονται σε αριθμητικό τύπο και στρογγυλοποιούνται σε δύο δεκαδικά ψηφία. Η τιμή της ποιότητας αέρα μετατρέπεται σε ακέραιο.

Η κανονικοποίηση είναι σημαντική επειδή εξασφαλίζει ότι η ίδια λογική απεικόνισης χρησιμοποιείται κάθε φορά που υπολογίζεται το hash. Με αυτόν τον τρόπο αποφεύγονται διαφορετικές αναπαραστάσεις των ίδιων λογικών δεδομένων.

4.7.2 Δημιουργία deterministic JSON

Μετά την κανονικοποίηση δημιουργείται ένα νέο JSON representation που περιλαμβάνει μόνο τα δεδομένα τα οποία συμμετέχουν στον υπολογισμό του αποτυπώματος [28]. Η μετατροπή πραγματοποιείται με χρήση της `json.dumps()` και με ενεργοποιημένη την παράμετρο `sort_keys=True`, ώστε η σειρά των πεδίων να παραμένει σταθερή.

Η σταθερή αυτή αναπαράσταση εξασφαλίζει ότι ο ίδιος κύκλος θα μετατρέπεται στην ίδια ακολουθία χαρακτήρων πριν από τον υπολογισμό του SHA-256. Επομένως, εάν τα δεδομένα δεν έχουν μεταβληθεί, μπορεί να παραχθεί ξανά το ίδιο κρυπτογραφικό αποτύπωμα.

4.7.3 Υπολογισμός SHA-256

Το παραγόμενο JSON μετατρέπεται σε ακολουθία bytes και υποβάλλεται στη συνάρτηση `hashlib.sha256()`. Το αποτέλεσμα μετατρέπεται μέσω της `hexdigest()` σε δεκαεξαδική αναπαράσταση και επιστρέφεται από τη `generate_hash()`.

Η συγκεκριμένη μορφή αποτελεί το hash που χρησιμοποιείται τόσο στην τοπική αποθήκευση όσο και στη δημοσίευση στο Shimmer. Ο Gateway καλεί τη `generate_hash()` αμέσως μετά τη δημιουργία του κύκλου και πριν προστεθούν στην καταχώριση το hash και το Block ID.

4.7.4 Χρήση του hash ως αποτυπώματος του κύκλου

Το τελικό hash λειτουργεί ως συνοπτικό αποτύπωμα των βασικών δεδομένων του κύκλου. Το αποτύπωμα αποθηκεύεται στη SQLite μαζί με την αντίστοιχη καταχώριση και δημοσιεύεται επίσης στο Shimmer μέσω Tagged Data block.

Κατά τη διαδικασία επαλήθευσης, τα ίδια δεδομένα ανακτώνται από τη βάση και υποβάλλονται εκ νέου στη `generate_hash()`. Το νέο αποτέλεσμα συγκρίνεται αρχικά με το hash που βρίσκεται στη SQLite και στη συνέχεια με το hash που ανακτάται από το αντίστοιχο block στο Shimmer.

Έτσι, η ίδια συνάρτηση χρησιμοποιείται τόσο κατά την αρχική δημιουργία του αποτυπώματος όσο και κατά τον μεταγενέστερο έλεγχο, εξασφαλίζοντας ότι η διαδικασία υπολογισμού παραμένει ίδια στα δύο στάδια.

4.8 Υλοποίηση της διασύνδεσης με το Shimmer DLT

Η πραγματική διασύνδεση της εφαρμογής με το δίκτυο Shimmer υλοποιήθηκε μέσω ξεχωριστού module, του shimmer_client.py, ώστε η επικοινωνία με το DLT να παραμένει ανεξάρτητη από τη λογική συλλογής και επεξεργασίας των δεδομένων. Για την επικοινωνία χρησιμοποιείται το Client του IOTA SDK, ενώ η εφαρμογή συνδέεται σε δημόσιο node του δικτύου Shimmer.

4.8.1 Αρχικοποίηση του Shimmer client

Η επικοινωνία υλοποιείται μέσω της κλάσης ShimmerClient. Κατά την αρχικοποίησή της δημιουργείται ένας client του IOTA SDK και ορίζεται ως node το <https://api.shimmer.network>. Με αυτόν τον τρόπο το module διαθέτει μία ξεχωριστή διεπαφή για όλες τις λειτουργίες που απαιτούν επικοινωνία με το δίκτυο.

Στο τέλος του module δημιουργείται μία κοινή παρουσία της κλάσης, η οποία χρησιμοποιείται από τις συναρτήσεις connect(), publish_hash() και get_block(). Η επιλογή αυτή επιτρέπει στα υπόλοιπα components του συστήματος να χρησιμοποιούν απλές συναρτήσεις χωρίς να χρειάζεται να διαχειρίζονται απευθείας το αντικείμενο του IOTA SDK.

4.8.2 Έλεγχος σύνδεσης και κατάστασης του δικτύου

Για τον έλεγχο της επικοινωνίας με το Shimmer υλοποιήθηκε η μέθοδος connect(). Η μέθοδος καλεί get_info() και χρησιμοποιεί την απάντηση του node για να εμφανίσει πληροφορίες σχετικά με το όνομα του δικτύου και την κατάσταση υγείας του node. Σε περίπτωση επιτυχίας επιστρέφεται True, ενώ σε περίπτωση εξαίρεσης η αποτυχία καταγράφεται και επιστρέφεται False.

Ο μηχανισμός αυτός επιτρέπει έναν βασικό έλεγχο της διαθεσιμότητας του DLT πριν χρησιμοποιηθεί η υποδομή για δημοσίευση δεδομένων.

4.8.3 Δημιουργία Tagged Data block

Για κάθε ολοκληρωμένο κύκλο, ο Gateway παραδίδει το υπολογισμένο SHA-256 hash στη publish_hash(). Η μέθοδος δημιουργεί ένα προκαθορισμένο tag με τιμή ENVIRONMENTAL_DATA_HASH και μετατρέπει τόσο το tag όσο και το hash σε hexadecimal μορφή.

Στη συνέχεια χρησιμοποιείται η build_and_post_block() του IOTA SDK με τις παραμέτρους tag και data. Με αυτόν τον τρόπο το κρυπτογραφικό αποτύπωμα δημοσιεύεται στο Shimmer ως Tagged Data block.

4.8.4 Επιστροφή και αποθήκευση του Block ID

Μετά την επιτυχή δημοσίευση η λειτουργία επιστρέφει το `block_id`, το οποίο εμφανίζεται και στο μήνυμα επιβεβαίωσης της εφαρμογής. Το Block ID αποτελεί το αναγνωριστικό μέσω του οποίου μπορεί να ανακτηθεί αργότερα η συγκεκριμένη καταχώριση από το δίκτυο.

Ο Gateway λαμβάνει το συγκεκριμένο αναγνωριστικό μέσω της `publish_hash()` και το προσθέτει στην καταχώριση του κύκλου πριν πραγματοποιήσει την τελική αποθήκευση στη βάση δεδομένων.

Παρότι η μεταβλητή στον Gateway και το αντίστοιχο πεδίο της SQLite ονομάζονται `transaction_id`, η πραγματική τιμή που αποθηκεύεται είναι το Block ID που επιστρέφεται από το Shimmer client.

4.8.5 Ανάκτηση του block για επαλήθευση

Ο Shimmer client παρέχει επίσης τη μέθοδο `get_block()`, η οποία δέχεται ένα Block ID και καλεί `get_block_data()` του IOTA SDK για να ανακτήσει τα δεδομένα της αντίστοιχης καταχώρισης.

Η λειτουργία αυτή χρησιμοποιείται από το `verify_integrity.py`. Μετά την ανάκτηση του block, το πρόγραμμα εξετάζει το payload και εξάγει τα δεδομένα του. Εφόσον το payload περιέχει hexadecimal αναπαράσταση, αφαιρείται το πρόθεμα `0x` και πραγματοποιείται αποκωδικοποίηση των bytes ώστε να προκύψει το αρχικό hash.

Το ανακτημένο hash χρησιμοποιείται στη συνέχεια για τη σύγκριση με το hash που υπολογίζεται εκ νέου από τα τοπικά δεδομένα. Με αυτόν τον τρόπο ολοκληρώνεται η σύνδεση μεταξύ της τοπικής βάσης και της καταχώρισης στο Shimmer.

4.8.6 Ρόλος της διασύνδεσης στην τελική αρχιτεκτονική

Η υλοποίηση του Shimmer client επιτρέπει στο DLT να λειτουργεί ως ανεξάρτητο επίπεδο αναφοράς χωρίς να αποθηκεύεται στο δίκτυο ολόκληρο το σύνολο των περιβαλλοντικών δεδομένων. Το τοπικό σύστημα διατηρεί τις πλήρεις μετρήσεις, ενώ το Shimmer χρησιμοποιείται για την καταγραφή του κρυπτογραφικού αποτυπώματος και τη μεταγενέστερη επαλήθευσή του.

Έτσι, η διασύνδεση που υλοποιήθηκε συνδέει τρία βασικά στοιχεία της εφαρμογής: τον κύκλο δεδομένων στη SQLite, το SHA-256 hash που παράγεται από αυτόν και το Block ID της αντίστοιχης καταχώρισης στο Shimmer.

4.9 Υλοποίηση των μηχανισμών επαλήθευσης

Η διαδικασία επαλήθευσης υλοποιήθηκε σε ξεχωριστό script, `verify_integrity.py`, και έχει ως στόχο να ελέγξει την ακεραιότητα όλων των αποθηκευμένων κύκλων. Για κάθε κύκλο πραγματοποιείται επανυπολογισμός του SHA-256 hash και στη συνέχεια εκτελούνται δύο ανεξάρτητες συγκρίσεις: μία με την τιμή που είναι αποθηκευμένη στη SQLite και μία με το hash που ανακτάται από την αντίστοιχη καταχώριση στο Shimmer.

4.9.1 Ανάκτηση των αποθηκευμένων κύκλων

Η διαδικασία ξεκινά με σύνδεση στη βάση `environmental_data.db` και ανάκτηση όλων των εγγραφών του πίνακα `environment_data`. Για κάθε κύκλο ανακτώνται το `cycle_id`, το `timestamp`, οι τρεις περιβαλλοντικές μετρήσεις, το αποθηκευμένο `hash` και το αναγνωριστικό της αντίστοιχης καταχώρισης στο Shimmer. Οι εγγραφές ταξινομούνται με βάση το `cycle_id`.

Από τα δεδομένα αυτά δημιουργείται για κάθε εγγραφή ένα νέο αντικείμενο κύκλου, το οποίο περιλαμβάνει μόνο τα πεδία που συμμετέχουν στον υπολογισμό του `hash`. Τα αποθηκευμένα `hash` και `Block ID` διατηρούνται ξεχωριστά για τις επόμενες συγκρίσεις.

4.9.2 Επανυπολογισμός του hash

Για κάθε κύκλο καλείται ξανά η `generate_hash()` με τα δεδομένα που ανακτήθηκαν από τη βάση. Με αυτόν τον τρόπο χρησιμοποιείται ακριβώς η ίδια διαδικασία κανονικοποίησης και `SHA-256` που είχε χρησιμοποιηθεί κατά την αρχική δημιουργία του κύκλου.

Το αποτέλεσμα αποθηκεύεται προσωρινά ως `calculated_hash` και χρησιμοποιείται ως κοινό σημείο αναφοράς και για τους δύο ελέγχους που ακολουθούν.

4.9.3 Τοπική επαλήθευση μέσω SQLite

Ο πρώτος έλεγχος πραγματοποιείται με σύγκριση του `calculated_hash` με το `stored_hash` που βρίσκεται ήδη στη βάση.

Εάν οι δύο τιμές είναι ίδιες, η τοπική επαλήθευση χαρακτηρίζεται ως `VERIFIED`. Σε διαφορετική περίπτωση χαρακτηρίζεται ως `TAMPERED`.

Η συγκεκριμένη διαδικασία επιτρέπει να εντοπιστεί τροποποίηση οποιουδήποτε από τα βασικά πεδία του κύκλου που συμμετέχουν στον υπολογισμό του `hash`.

4.9.4 Ανάκτηση του block από το Shimmer

Εφόσον υπάρχει αποθηκευμένο αναγνωριστικό, το πρόγραμμα καλεί τη `get_block()` του Shimmer client για να ανακτήσει την αντίστοιχη καταχώριση από το δίκτυο.

Από το block ανακτάται το `payload`. Εάν δεν υπάρχει `payload`, η διαδικασία δημιουργεί σφάλμα. Στη συνέχεια το `data` περιεχόμενο μετατρέπεται από hexadecimal μορφή σε `bytes` και αποκωδικοποιείται ώστε να ανακτηθεί το αρχικό `hash` που είχε δημοσιευθεί.

4.9.5 Επαλήθευση μέσω Shimmer

Το `hash` που ανακτήθηκε από το Shimmer αποθηκεύεται ως `ledger_hash` και συγκρίνεται με το `calculated_hash` του τοπικού κύκλου.

Όταν οι δύο τιμές συμφωνούν, η μεταβλητή `ledger_ok` γίνεται `True` και ο έλεγχος εμφανίζεται ως Shimmer Verification: `VERIFIED`. Σε περίπτωση απόκλισης, εμφανίζεται Shimmer Verification: `FAILED`.

Έτσι, ο μηχανισμός ελέγχει όχι μόνο εάν η τοπική βάση παραμένει συνεπής με το αρχικό hash, αλλά και εάν το τρέχον περιεχόμενο των δεδομένων συμφωνεί με το hash που είχε δημοσιευθεί στο DLT.

4.9.6 Συνολική κατάσταση επαλήθευσης

Για κάθε κύκλο εμφανίζονται ξεχωριστά τα αποτελέσματα της τοπικής και της Shimmer επαλήθευσης. Ο κύκλος χαρακτηρίζεται ως VERIFIED μόνο όταν και οι δύο έλεγχοι ολοκληρωθούν επιτυχώς. Σε οποιαδήποτε άλλη περίπτωση η συνολική κατάσταση χαρακτηρίζεται ως FAILED.

Στο τέλος της διαδικασίας υπολογίζεται και εμφανίζεται το πλήθος των κύκλων που επαληθεύτηκαν επιτυχώς σε σχέση με το συνολικό πλήθος των αποθηκευμένων κύκλων.

Η συγκεκριμένη σχεδίαση επιτρέπει την ταυτόχρονη αξιολόγηση της τοπικής ακεραιότητας και της συμφωνίας με το εξωτερικό DLT σημείο αναφοράς.

4.10 Εκτέλεση δοκιμής σκόπιμης αλλοίωσης

Για την αξιολόγηση του μηχανισμού επαλήθευσης πραγματοποιήθηκε ελεγχόμενη δοκιμή σκόπιμης αλλοίωσης σε ήδη αποθηκευμένα δεδομένα. Η αλλοίωση πραγματοποιήθηκε χειροκίνητα στη βάση δεδομένων, με τροποποίηση μίας τιμής υπάρχοντος κύκλου, χωρίς να μεταβληθούν το αρχικά αποθηκευμένο hash και η αντίστοιχη καταχώρηση στο Shimmer.

4.10.1 Σκόπιμη τροποποίηση δεδομένων

Κατά τη διάρκεια της δοκιμής τροποποιήθηκε χειροκίνητα μία τιμή ενός αποθηκευμένου κύκλου στη βάση δεδομένων, μετά την αρχική δημιουργία και αποθήκευσή του. Το αρχικό κρυπτογραφικό αποτύπωμα του κύκλου και η αντίστοιχη καταχώρηση στο Shimmer παρέμειναν αμετάβλητα.

Με τον τρόπο αυτό δημιουργήθηκε ελεγχόμενη διαφορά μεταξύ της τρέχουσας κατάστασης των τοπικά αποθηκευμένων δεδομένων και της κατάστασης που είχε χρησιμοποιηθεί κατά τη δημιουργία του αρχικού hash.

4.10.2 Έλεγχος της αλλοίωσης

Μετά τη χειροκίνητη τροποποίηση εκτελέστηκε το `verify_integrity.py`. Το πρόγραμμα επαν υπολόγισε το hash του τροποποιημένου κύκλου και το συνέκρινε με το αρχικά αποθηκευμένο hash στη SQLite και με το hash που είχε δημοσιευθεί στο Shimmer.

Η διαδικασία αυτή επέτρεψε να ελεγχθεί κατά πόσο η μεταβολή ενός αποθηκευμένου δεδομένου μπορεί να εντοπιστεί μέσω της υλοποιημένης διαδικασίας επαλήθευσης.

4.10.3 Επαναφορά των δεδομένων

Μετά την ολοκλήρωση της δοκιμής, η τροποποιημένη τιμή επαναφέρθηκε στην αρχική της κατάσταση, ώστε να αποκατασταθεί η αρχική κατάσταση της βάσης δεδομένων.

4.10.4 Επανεπαλήθευση

Μετά την επαναφορά πραγματοποιήθηκε νέα εκτέλεση του `verify_integrity.py`, ώστε να επιβεβαιωθεί εκ νέου η ακεραιότητα των αποθηκευμένων δεδομένων.

4.11 Υλοποίηση του Launcher

Ο Launcher αναπτύχθηκε ως το κεντρικό γραφικό περιβάλλον διαχείρισης της εφαρμογής. Στόχος του είναι να συγκεντρώνει σε ένα σημείο τις βασικές λειτουργίες εκτέλεσης, ελέγχου και παρακολούθησης του project, χωρίς να συμμετέχει άμεσα στην επεξεργασία των περιβαλλοντικών δεδομένων.

Η υλοποίηση αποτελείται από το γραφικό περιβάλλον, τον `ProcessManager`, τον μηχανισμό καταγραφής ενεργειών και τον MQTT client που χρησιμοποιείται για τη λήψη της κατάστασης ετοιμότητας του Gateway.

4.11.1 Γραφικό περιβάλλον

Το γραφικό περιβάλλον υλοποιήθηκε με `CustomTkinter`. Κατά την αρχικοποίηση καθορίζεται η εμφάνιση της εφαρμογής και δημιουργείται το βασικό παράθυρο, στο οποίο οργανώνονται τα επιμέρους τμήματα της διεπαφής.

Η κύρια οθόνη περιλαμβάνει τα controls του project, τις ενδείξεις κατάστασης, τις πληροφορίες της βάσης δεδομένων και το activity log. Παράλληλα χρησιμοποιούνται ξεχωριστές ενδείξεις για τον MQTT Broker, τον Gateway, τον Publisher, τη βάση δεδομένων και το DLT.

Στα διαθέσιμα controls περιλαμβάνονται οι λειτουργίες εκκίνησης και τερματισμού του project, επαλήθευσης ακεραιότητας, δημιουργίας και προβολής της βάσης, ανοίγματος του Shimmer Explorer και πρόσβασης στον φάκελο του project.

4.11.2 Process Manager

Η διαχείριση των επιμέρους διεργασιών έχει διαχωριστεί από το γραφικό περιβάλλον και υλοποιείται μέσω του `ProcessManager`. Ο μηχανισμός αυτός χρησιμοποιεί τις βιβλιοθήκες `subprocess` και `psutil` για τον έλεγχο και τη διαχείριση των διεργασιών της εφαρμογής.

Ο `ProcessManager` παρέχει συναρτήσεις για τον έλεγχο της κατάστασης των βασικών διεργασιών και για την εκκίνησή τους όταν απαιτείται. Ο εντοπισμός των Python scripts πραγματοποιείται με εξέταση των ενεργών διεργασιών και της αντίστοιχης γραμμής εντολών.

4.11.3 Έλεγχος κατάστασης των components

Ο Launcher παρακολουθεί την κατάσταση των βασικών components και παρουσιάζει τις πληροφορίες αυτές στο γραφικό περιβάλλον. Παράλληλα ελέγχεται η διαθεσιμότητα της βάσης δεδομένων, ώστε ο χρήστης να γνωρίζει αν το σύστημα είναι έτοιμο να χρησιμοποιηθεί.

Οι ενδείξεις κατάστασης ενημερώνονται από τον Launcher με βάση την τρέχουσα κατάσταση των components και επιτρέπουν στον χρήστη να έχει άμεση εικόνα της λειτουργίας του συστήματος.

4.11.4 Start Project

Η λειτουργία Start Project ελέγχει αρχικά αν υπάρχει η απαιτούμενη βάση δεδομένων. Εφόσον δεν υπάρχει, η εκκίνηση διακόπτεται και ζητείται πρώτα η δημιουργία της.

Στη συνέχεια ο Launcher προετοιμάζει τον μηχανισμό επικοινωνίας για την παρακολούθηση της κατάστασης του Gateway και ξεκινά τον Gateway μέσω του ProcessManager. Η εκκίνηση του Publisher πραγματοποιείται σε επόμενο στάδιο, αφού επιβεβαιωθεί ότι ο Gateway έχει ολοκληρώσει επιτυχώς την αρχικοποίησή του.

4.11.5 Μηχανισμός συγχρονισμού Gateway–Publisher

Για την αποφυγή προβλημάτων συγχρονισμού χρησιμοποιείται το MQTT topic gateway/status. Μετά την επιτυχή σύνδεσή του στον broker, ο Gateway δημοσιεύει στο συγκεκριμένο topic την κατάσταση READY.

Ο Launcher έχει ξεχωριστό MQTT client, ο οποίος κάνει subscribe στο gateway/status. Όταν παραληφθεί μήνυμα με τιμή READY, ο Launcher ενημερώνει την εσωτερική του κατάσταση και καλεί τη λειτουργία εκκίνησης του Publisher.

Η συγκεκριμένη προσέγγιση επιτρέπει στον Publisher να ξεκινά με βάση την πραγματική κατάσταση του Gateway και όχι απλώς μετά την παρέλευση μιας προκαθορισμένης χρονικής καθυστέρησης. Έτσι, η μετάβαση από την εκκίνηση του Gateway στην εκκίνηση του Publisher πραγματοποιείται μόνο αφού επιβεβαιωθεί ότι το πρώτο είναι έτοιμο.

4.11.6 Stop Project

Η λειτουργία Stop Project χρησιμοποιείται για τον τερματισμό των διεργασιών που ανήκουν στο project. Ο ProcessManager αναλαμβάνει τον τερματισμό των σχετικών Python διεργασιών και ο Launcher καταγράφει την ολοκλήρωση της διαδικασίας.

Ο MQTT Broker αντιμετωπίζεται ως ανεξάρτητη υπηρεσία και δεν αποτελεί μέρος του τερματισμού των διεργασιών του project. Η επιλογή αυτή διατηρεί ξεχωριστό το επίπεδο της υποδομής MQTT από τις εφαρμογές που χρησιμοποιούν το broker.

4.11.7 Activity Log και ενημέρωση κατάστασης

Ο Launcher περιλαμβάνει ενσωματωμένο activity log, στο οποίο καταγράφονται οι βασικές ενέργειες της εφαρμογής μαζί με χρονική ένδειξη και επίπεδο ενημέρωσης. Οι διαθέσιμες κατηγορίες περιλαμβάνουν INFO, SUCCESS, WARNING και ERROR.

Παράλληλα, το γραφικό περιβάλλον παρουσιάζει πληροφορίες σχετικά με την κατάσταση των components και τη διαθεσιμότητα της βάσης, ώστε ο χρήστης να μπορεί να παρακολουθεί τη λειτουργία του συστήματος από ένα ενιαίο σημείο.

4.11.8 Database και Shimmer εργαλεία του Launcher

Εκτός από τη διαχείριση των βασικών διεργασιών, ο Launcher παρέχει βοηθητικές λειτουργίες για τη βάση δεδομένων και το DLT. Μέσω της επιλογής Create Database μπορεί να δημιουργηθεί η βάση, ενώ οι λειτουργίες View Database και Verify Integrity επιτρέπουν την πρόσβαση στα αντίστοιχα εργαλεία. Παράλληλα, η επιλογή Open Shimmer Explorer ανοίγει απευθείας τον explorer του δικτύου.

Με τον τρόπο αυτό ο Launcher λειτουργεί ως ενιαίο σημείο διαχείρισης της τελικής εφαρμογής, συγκεντρώνοντας τις βασικές λειτουργίες εκτέλεσης, παρακολούθησης και ελέγχου.

4.12 Δημιουργία της εκτελέσιμης έκδοσης

Για τη διευκόλυνση της εκτέλεσης της εφαρμογής δημιουργήθηκε εκτελέσιμη έκδοση για Windows μέσω του PyInstaller [27]. Στόχος ήταν ο Launcher να μπορεί να εκκινεί ως αυτόνομη γραφική εφαρμογή, χωρίς να απαιτείται άμεση εκτέλεση του αρχείου Python από το περιβάλλον ανάπτυξης.

Η είσοδος της εφαρμογής υλοποιείται από το main.py, το οποίο εισάγει την κλάση LauncherUI, δημιουργεί το αντίστοιχο αντικείμενο και καλεί τη μέθοδο run(). Το αρχείο αυτό αποτελεί επομένως το σημείο εκκίνησης της τελικής εφαρμογής.

4.12.1 Προετοιμασία της εφαρμογής για PyInstaller

Η δημιουργία της εκτελέσιμης έκδοσης βασίστηκε σε αρχείο προδιαγραφών PyInstaller (.spec). Στο αρχείο ορίζεται ως βασικό script το main.py, ενώ καθορίζονται τα πρόσθετα δεδομένα και οι ιδιότητες του τελικού executable.

Η χρήση του .spec επιτρέπει τη συγκεντρωμένη ρύθμιση των παραμέτρων δημιουργίας της εφαρμογής και τη συμπερίληψη των επιπλέον πόρων που απαιτούνται από το γραφικό περιβάλλον.

4.12.2 Ενσωμάτωση των resources

Στο τελικό .spec δηλώνεται ο φάκελος assets ως πρόσθετο directory δεδομένων:

```
("assets", "assets")
```

Με αυτόν τον τρόπο τα αρχεία του φακέλου assets συμπεριλαμβάνονται στην εκτελέσιμη εφαρμογή και διατηρείται η απαιτούμενη δομή τους.

Παράλληλα, ο Launcher διαθέτει τη συνάρτηση resource_path(), η οποία επιλέγει διαφορετική βάση αναζήτησης ανάλογα με το αν η εφαρμογή εκτελείται από το περιβάλλον ανάπτυξης ή ως packaged εφαρμογή. Στην packaged εκτέλεση χρησιμοποιείται το sys._MEIPASS, ώστε να εντοπίζονται οι ενσωματωμένοι πόροι.

4.12.3 Δημιουργία του executable

Το τελικό .spec ορίζει ως όνομα της εκτελέσιμης εφαρμογής το IoT + DLT Integration Project. Παράλληλα έχει οριστεί console=False, ώστε η εφαρμογή να εκκινεί ως γραφικό παράθυρο χωρίς ξεχωριστή κονσόλα.

Ως εικονίδιο της εφαρμογής χρησιμοποιείται το `assets/iot_dlt.ico`, το οποίο δηλώνεται επίσης στο αρχείο προδιαγραφών.

Με τις συγκεκριμένες ρυθμίσεις, το `main.py`, οι απαραίτητοι πόροι και οι υπόλοιπες απαιτήσεις του project συσκευάζονται σε εκτελέσιμη εφαρμογή Windows.

4.12.4 Εκτέλεση της εφαρμογής σε δεύτερο υπολογιστή

Η εκτελέσιμη έκδοση χρησιμοποιήθηκε και σε δεύτερο υπολογιστή, με στόχο να ελεγχθεί η λειτουργία της εφαρμογής εκτός του αρχικού περιβάλλοντος ανάπτυξης.

Η διαδικασία αυτή επέτρεψε να ελεγχθεί η ορθή φόρτωση των ενσωματωμένων resources και η κανονική εκκίνηση του Launcher ως γραφικής εφαρμογής σε διαφορετικό περιβάλλον Windows.

Η δοκιμή αυτή αποτελεί πρακτικό έλεγχο της δυνατότητας μεταφοράς της τελικής εφαρμογής σε άλλο συμβατό υπολογιστικό περιβάλλον.

4.13 Συνολική ενσωμάτωση των υποσυστημάτων

Η τελική εφαρμογή προέκυψε από την ενσωμάτωση των επιμέρους υποσυστημάτων σε μία ενιαία ροή λειτουργίας. Οι προσομοιωμένοι αισθητήρες παράγουν τις περιβαλλοντικές μετρήσεις, οι οποίες μεταδίδονται από τον Publisher μέσω MQTT και παραλαμβάνονται από τον Gateway. Ο Gateway συγκεντρώνει τις απαιτούμενες μετρήσεις, δημιουργεί τον αντίστοιχο κύκλο και ενεργοποιεί τη διαδικασία δημιουργίας του κρυπτογραφικού αποτυπώματος.

Το hash που προκύπτει αποθηκεύεται μαζί με τον κύκλο στη SQLite, ενώ παράλληλα δημοσιεύεται στο Shimmer μέσω Tagged Data block. Από τη δημοσίευση επιστρέφεται το Block ID, το οποίο συνδέεται με την τοπική καταχώριση και επιτρέπει τη μεταγενέστερη ανάκτηση της αντίστοιχης καταχώρισης από το δίκτυο.

Η διαδικασία επαλήθευσης αξιοποιεί τα ίδια δεδομένα και επανυπολογίζει το hash, το οποίο συγκρίνεται τόσο με την τιμή που είναι αποθηκευμένη στη βάση όσο και με το hash που ανακτάται από το αντίστοιχο block στο Shimmer. Με τον τρόπο αυτό συνδυάζονται η τοπική αποθήκευση και η εξωτερική καταχώριση σε μία ενιαία διαδικασία ελέγχου ακεραιότητας.

Παράλληλα, ο Launcher λειτουργεί ως επίπεδο διαχείρισης της εφαρμογής. Αναλαμβάνει την εκκίνηση των επιμέρους διεργασιών, παρακολουθεί την κατάστασή τους, χρησιμοποιεί το μήνυμα READY για τον συγχρονισμό Gateway και Publisher και παρέχει πρόσβαση στις βασικές λειτουργίες επαλήθευσης και διαχείρισης της βάσης.

Η τελική αρχιτεκτονική μπορεί επομένως να συνοψιστεί ως μία αλληλουχία:

Αισθητήρες → Publisher → MQTT → Gateway → Cycle → SHA-256 → SQLite + Shimmer → Verification

με τον Launcher να λειτουργεί παράλληλα ως επίπεδο διαχείρισης και παρακολούθησης της εφαρμογής.

Η ολοκλήρωση αυτή επιτρέπει στα επιμέρους components να λειτουργούν ως ένα ενιαίο σύστημα, όπου η παραγωγή των δεδομένων, η αποθήκευση, η δημοσίευση του κρυπτογραφικού τους αποτυπώματος και η επαλήθευση συνδέονται σε μία συνεχή διαδικασία.



Σχήμα 4.1: Launcher Ui

4.14 Επίλογος

Στο παρόν κεφάλαιο παρουσιάστηκε η υλοποίηση των βασικών υποσυστημάτων της εφαρμογής και ο τρόπος με τον οποίο αυτά ενσωματώθηκαν στην τελική λύση. Αναλύθηκε η προσομοίωση των περιβαλλοντικών αισθητήρων, η επικοινωνία μέσω MQTT και η λειτουργία του Gateway ως κεντρικού σημείου συλλογής και επεξεργασίας των μετρήσεων.

Στη συνέχεια παρουσιάστηκαν η τοπική αποθήκευση στη SQLite, η διαδικασία δημιουργίας του SHA-256 hash, η πραγματική δημοσίευσή του στο Shimmer και η διατήρηση του αντίστοιχου Block

ID. Παράλληλα, αναλύθηκαν οι μηχανισμοί επαλήθευσης και οι δοκιμές σκόπιμης αλλοίωσης, καθώς και η υλοποίηση του Launcher και της εκτελέσιμης έκδοσης της εφαρμογής.

Η υλοποίηση που παρουσιάστηκε αποτελεί την πρακτική εφαρμογή του σχεδιασμού του προηγούμενου κεφαλαίου και δημιουργεί τη βάση για την πειραματική αξιολόγηση του συστήματος. Στο επόμενο κεφάλαιο παρουσιάζονται η διαδικασία δοκιμών, τα πειραματικά σενάρια και τα αποτελέσματα που προέκυψαν από τη λειτουργία της τελικής εφαρμογής.

Κεφάλαιο 5ο: Πειραματική αξιολόγηση και αποτελέσματα

5.1 Εισαγωγή

Στο παρόν κεφάλαιο παρουσιάζεται η πειραματική αξιολόγηση της τελικής υλοποίησης και τα αποτελέσματα που προέκυψαν από την εκτέλεση του συστήματος. Στόχος της αξιολόγησης είναι να εξεταστεί κατά πόσο οι βασικές λειτουργίες που καθορίστηκαν στα προηγούμενα κεφάλαια υλοποιήθηκαν και λειτουργούν σύμφωνα με τις απαιτήσεις της εργασίας.

Η αξιολόγηση περιλαμβάνει τον έλεγχο της παραγωγής και μετάδοσης των περιβαλλοντικών μετρήσεων, τη συλλογή τους από τον Gateway, τη δημιουργία και αποθήκευση των κύκλων, καθώς και τη δημιουργία των αντίστοιχων κρυπτογραφικών αποτυπωμάτων. Παράλληλα, εξετάζεται η πραγματική δημοσίευση των hashes στο δίκτυο Shimmer και η δυνατότητα ανάκτησης της αντίστοιχης καταχώρισης μέσω του Block ID.

Ιδιαίτερη έμφαση δίνεται στον έλεγχο της ακεραιότητας των δεδομένων. Για τον σκοπό αυτό εξετάζεται τόσο η σύγκριση των επανυπολογισμένων hashes με τις τιμές που είναι αποθηκευμένες στη SQLite όσο και η σύγκρισή τους με τα hashes που έχουν δημοσιευθεί στο Shimmer. Στο πλαίσιο της αξιολόγησης εξετάζεται επίσης η συμπεριφορά του μηχανισμού σε περίπτωση σκόπιμης αλλοίωσης δεδομένων.

Τέλος, αξιολογείται η συνολική λειτουργία της εφαρμογής μέσω του Launcher και η συνεργασία των επιμέρους components κατά την εκτέλεση του project. Τα αποτελέσματα παρουσιάζονται με βάση τις πραγματικές εκτελέσεις της τελικής εφαρμογής και τα δεδομένα που προέκυψαν από τη βάση, τους μηχανισμούς επαλήθευσης και το δίκτυο Shimmer.

5.2 Περιβάλλον και διαδικασία δοκιμών

Η πειραματική αξιολόγηση πραγματοποιήθηκε σε περιβάλλον Windows με χρήση της τελικής έκδοσης της εφαρμογής. Στην εκτέλεση χρησιμοποιήθηκαν ο MQTT Broker, ο Gateway, ο Publisher, η SQLite βάση δεδομένων και η διασύνδεση με το δίκτυο Shimmer.

Η αξιολόγηση πραγματοποιήθηκε με χρήση της τελικής εφαρμογής και των επιμέρους εργαλείων ελέγχου. Ο Launcher χρησιμοποιήθηκε για την εκκίνηση και την παρακολούθηση του συστήματος, ενώ η βάση δεδομένων και ο μηχανισμός επαλήθευσης χρησιμοποιήθηκαν για την αξιολόγηση των δεδομένων και της ακεραιότητάς τους.

Η διαδικασία οργάνωθηκε σε διαδοχικά στάδια. Αρχικά εξετάστηκε η παραγωγή και μετάδοση των περιβαλλοντικών μετρήσεων. Στη συνέχεια ελέγχθηκε η παραλαβή των δεδομένων από τον Gateway και η δημιουργία των αντίστοιχων κύκλων. Ακολούθησε ο έλεγχος της αποθήκευσης των κύκλων στη SQLite, της δημιουργίας των hashes και της δημοσίευσής τους στο Shimmer.

Τέλος, εκτελέστηκε ο μηχανισμός επαλήθευσης για τους αποθηκευμένους κύκλους και εξετάστηκε η λειτουργία του Launcher κατά την εκτέλεση του project.

Για την τεκμηρίωση των αποτελεσμάτων χρησιμοποιήθηκαν τα δεδομένα της SQLite, τα μηνύματα των επιμέρους διεργασιών, τα αποτελέσματα του `verify_integrity.py`, καθώς και η απεικόνιση των δημοσιευμένων blocks στο Shimmer Explorer.

5.3 Έλεγχος λειτουργίας των αισθητήρων και παραγωγής δεδομένων

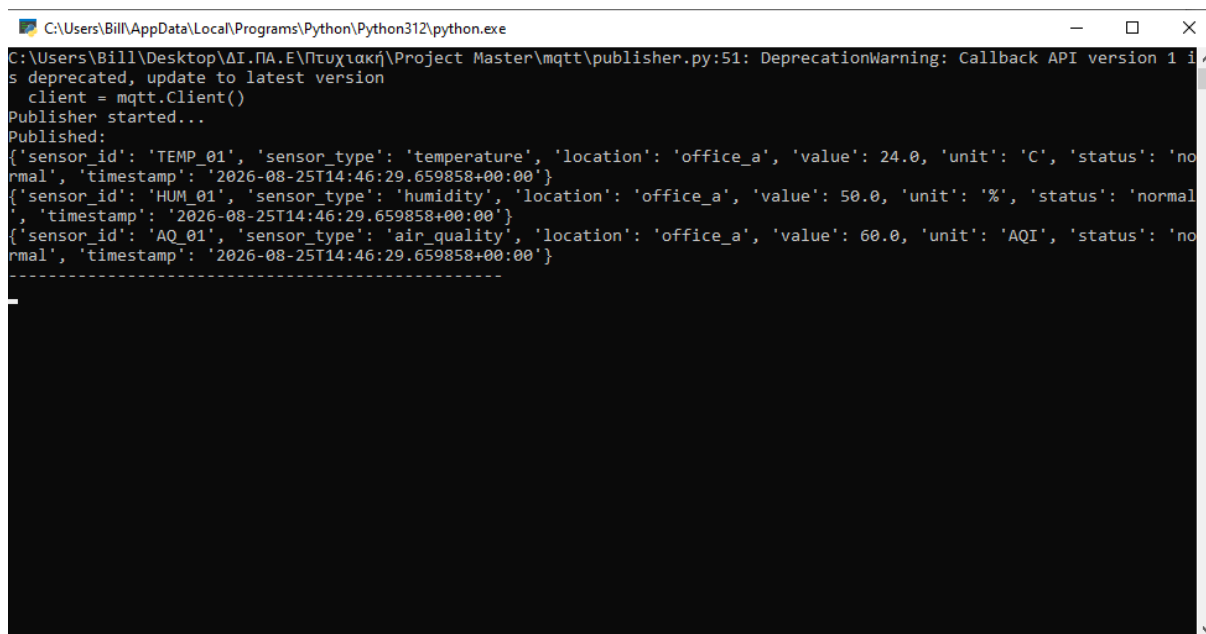
Ο πρώτος έλεγχος αφορούσε την παραγωγή περιβαλλοντικών δεδομένων από τους τρεις προσομοιωμένους αισθητήρες. Κατά την εκτέλεση του Publisher δημιουργήθηκαν διαδοχικές μετρήσεις θερμοκρασίας, υγρασίας και ποιότητας αέρα και οι αντίστοιχες τιμές μεταδόθηκαν προς τον Gateway.

Στη βάση δεδομένων που χρησιμοποιήθηκε για την αξιολόγηση καταγράφηκαν συνολικά 11 κύκλοι. Στους συγκεκριμένους κύκλους η θερμοκρασία κυμάνθηκε από 23,9 έως 24,0 °C, η σχετική υγρασία από 49,9% έως 50,1%, ενώ η ποιότητα αέρα από 59 έως 61 AQI.

Πίνακας 5.1: Παρατηρούμενο εύρος τιμών στους 11 εξεταζόμενους κύκλους

Παράμετρος	Ελάχιστη τιμή	Μέγιστη τιμή
Θερμοκρασία	23,9 °C	24,0 °C
Υγρασία	49,9 %	50,1 %
Ποιότητα αέρα	59 AQI	61 AQI

Οι τιμές παρουσιάζουν μικρές μεταβολές μεταξύ διαδοχικών κύκλων, όπως προβλέπεται από τη σχεδίαση των προσομοιωμένων αισθητήρων. Δεν παρατηρούνται μεγάλες μεταβολές στις συγκεκριμένες καταγραφές, γεγονός που είναι συμβατό με τη μικρή μεταβολή που έχει προβλεφθεί κατά την προσομοίωση.



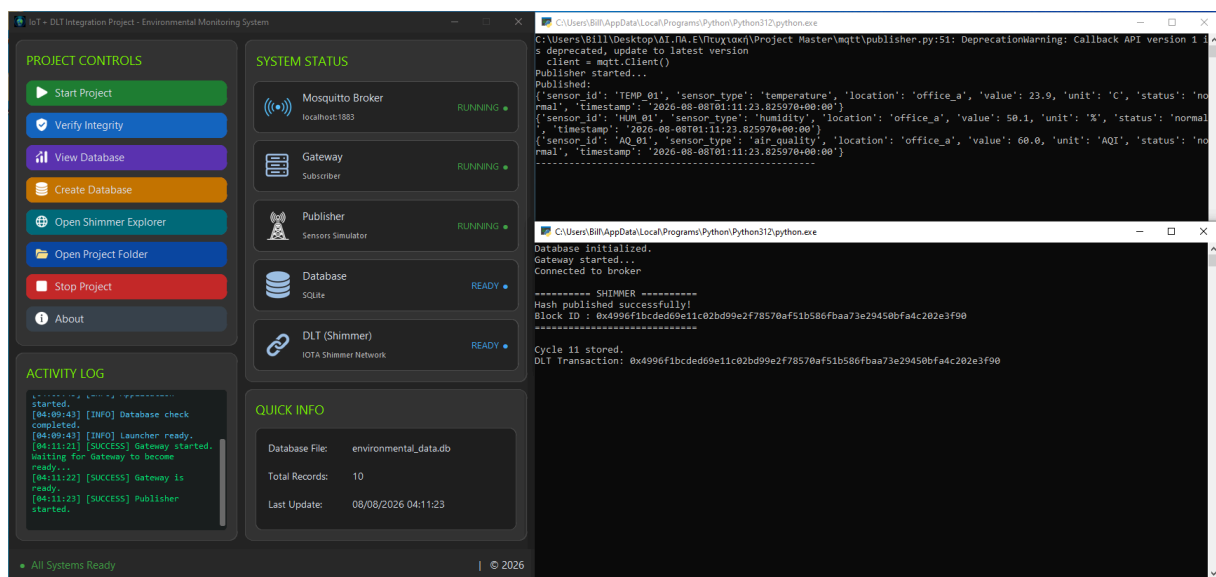
```
C:\Users\Bill\AppData\Local\Programs\Python\Python312\python.exe
C:\Users\Bill\Desktop\ΔΙ.ΠΑ.Ε\Πτυχιακή\Project Master\mqtt\publisher.py:51: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
  client = mqtt.Client()
Publisher started...
Published:
{'sensor_id': 'TEMP_01', 'sensor_type': 'temperature', 'location': 'office_a', 'value': 24.0, 'unit': 'C', 'status': 'normal', 'timestamp': '2026-08-25T14:46:29.659858+00:00'}
{'sensor_id': 'HUM_01', 'sensor_type': 'humidity', 'location': 'office_a', 'value': 50.0, 'unit': '%', 'status': 'normal', 'timestamp': '2026-08-25T14:46:29.659858+00:00'}
{'sensor_id': 'AQ_01', 'sensor_type': 'air_quality', 'location': 'office_a', 'value': 60.0, 'unit': 'AQI', 'status': 'normal', 'timestamp': '2026-08-25T14:46:29.659858+00:00'}
```

Σχήμα 5.1: Publisher

5.4 Έλεγχος επικοινωνίας MQTT και λειτουργίας του Gateway

Στη δεύτερη φάση εξετάστηκε η μετάδοση των παραγόμενων μετρήσεων μέσω MQTT και η παραλαβή τους από τον Gateway.

Κατά την εκτέλεση του project ο Publisher δημοσίευσε τις μετρήσεις των τριών αισθητήρων στα αντίστοιχα topics, ενώ ο Gateway συνδέθηκε στον MQTT Broker και παραλάμβανε τα εισερχόμενα μηνύματα. Η ενεργή λειτουργία των βασικών διεργασιών αποτυπώνεται και στο γραφικό περιβάλλον του Launcher, όπου ο Gateway και ο Publisher εμφανίζονται ως ενεργά components κατά τη διάρκεια της εκτέλεσης.



Σχήμα 5.2: Project Running

Η λειτουργία του Gateway επιβεβαιώνεται επιπλέον από τη δημιουργία των κύκλων που καταγράφηκαν στη βάση. Οι επιμέρους μετρήσεις συγκεντρώθηκαν σε ολοκληρωμένες καταχωρίσεις, οι οποίες περιλαμβάνουν τις τρεις περιβαλλοντικές παραμέτρους και τη χρονική πληροφορία του κύκλου.

Η δοκιμή αυτή επιβεβαιώνει την ορθή συνεργασία μεταξύ Publisher, MQTT Broker και Gateway και την επιτυχή μεταφορά των δεδομένων από το επίπεδο παραγωγής προς το επίπεδο επεξεργασίας.

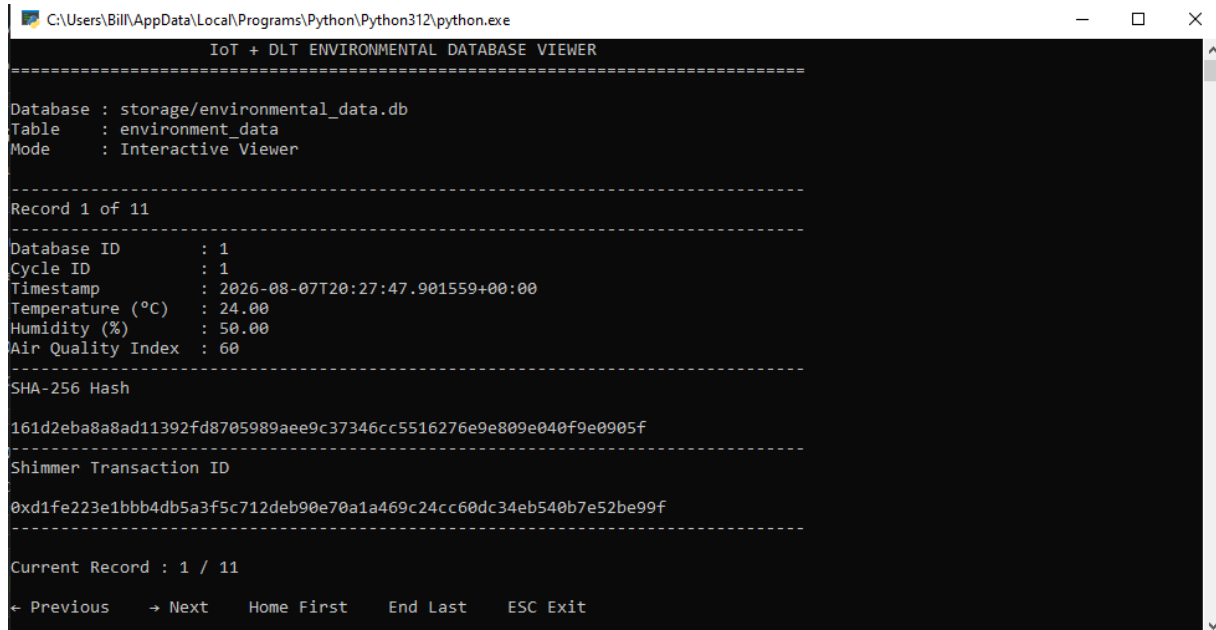
Σημειώνεται ότι στην υλοποίηση του Publisher έχει οριστεί αναμονή 60 δευτερολέπτων μεταξύ διαδοχικών επαναλήψεων. Ωστόσο, τα δεδομένα της εξεταζόμενης βάσης προέρχονται από πολλαπλές εκτελέσεις του συστήματος και επομένως τα χρονικά διαστήματα μεταξύ όλων των αποθηκευμένων κύκλων δεν είναι ενιαία.

5.5 Έλεγχος δημιουργίας κύκλων, αποθήκευσης και hashing

Μετά την παραλαβή των απαιτούμενων μετρήσεων από τον Gateway δημιουργήθηκαν ολοκληρωμένοι κύκλοι δεδομένων και αποθηκεύτηκαν στη SQLite.

Η εξεταζόμενη βάση περιέχει 11 ολοκληρωμένους κύκλους, με συνεχόμενη αρίθμηση από cycle_id = 1 έως cycle_id = 11. Για κάθε κύκλο έχουν καταγραφεί οι τρεις περιβαλλοντικές μετρήσεις, το

timestamp, το αντίστοιχο SHA-256 hash και το αναγνωριστικό της σχετικής καταχώρισης στο Shimmer.



```

C:\Users\Bill\AppData\Local\Programs\Python\Python312\python.exe
IoT + DLT ENVIRONMENTAL DATABASE VIEWER
=====
Database : storage/environmental_data.db
Table    : environment_data
Mode     : Interactive Viewer

-----
Record 1 of 11
-----
Database ID      : 1
Cycle ID        : 1
Timestamp       : 2026-08-07T20:27:47.901559+00:00
Temperature (°C) : 24.00
Humidity (%)    : 50.00
Air Quality Index : 60
-----
SHA-256 Hash
161d2eba8a8ad11392fd8705989aee9c37346cc5516276e9e809e040f9e0905f
-----
Shimmer Transaction ID
0xd1fe223e1bbb4db5a3f5c712deb90e70a1a469c24cc60dc34eb540b7e52be99f
-----
Current Record : 1 / 11
< Previous    Next >    Home First    End Last    ESC Exit

```

Σχήμα 5.3: Database Viewer

Η πρώτη καταχώριση αποτελεί χαρακτηριστικό παράδειγμα της δομής των αποθηκευμένων δεδομένων. Για τον κύκλο 1 καταγράφονται θερμοκρασία 24,0 °C, υγρασία 50,0% και ποιότητα αέρα 60 AQI. Το αντίστοιχο hash είναι:

161d2eba8a8ad11392fd8705989aee9c37346cc5516276e9e809e040f9e0905f

ενώ μαζί με την καταχώριση διατηρείται και το Block ID της αντίστοιχης δημοσίευσης στο Shimmer.

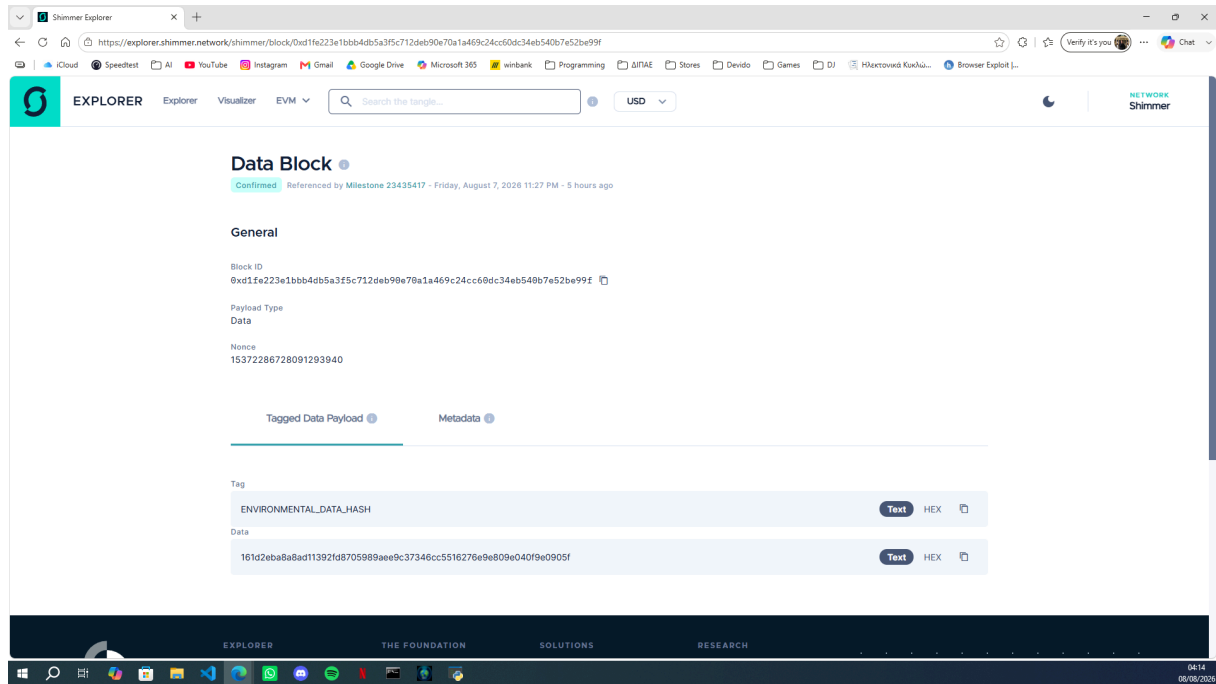
Η ίδια δομή διατηρείται και στους υπόλοιπους κύκλους. Κάθε αποθηκευμένη καταχώριση διαθέτει το δικό της κρυπτογραφικό αποτύπωμα και το αντίστοιχο αναγνωριστικό της δημοσιευμένης καταχώρισης στο DLT.

Το αποτέλεσμα επιβεβαιώνει ότι η διαδικασία δημιουργίας κύκλων, αποθήκευσης στη βάση και παραγωγής των hashes λειτουργεί ως ενιαία διαδικασία.

5.6 Έλεγχος δημοσίευσης των hashes στο Shimmer

Ένα από τα βασικότερα σημεία της αξιολόγησης ήταν η επιβεβαίωση ότι τα κρυπτογραφικά αποτυπώματα δημοσιεύονται πραγματικά στο δίκτυο Shimmer.

Στην εξεταζόμενη καταχώριση του πρώτου κύκλου εμφανίζεται στον Shimmer Explorer ένα Confirmed Data Block. Στο ίδιο block εμφανίζεται το αντίστοιχο Block ID, ενώ στο Tagged Data Payload εμφανίζεται το tag ENVIRONMENTAL_DATA_HASH και το δημοσιευμένο hash.



Σχήμα 5.4: Shimmer Explorer

Το hash που εμφανίζεται στη συγκεκριμένη καταχώριση είναι:

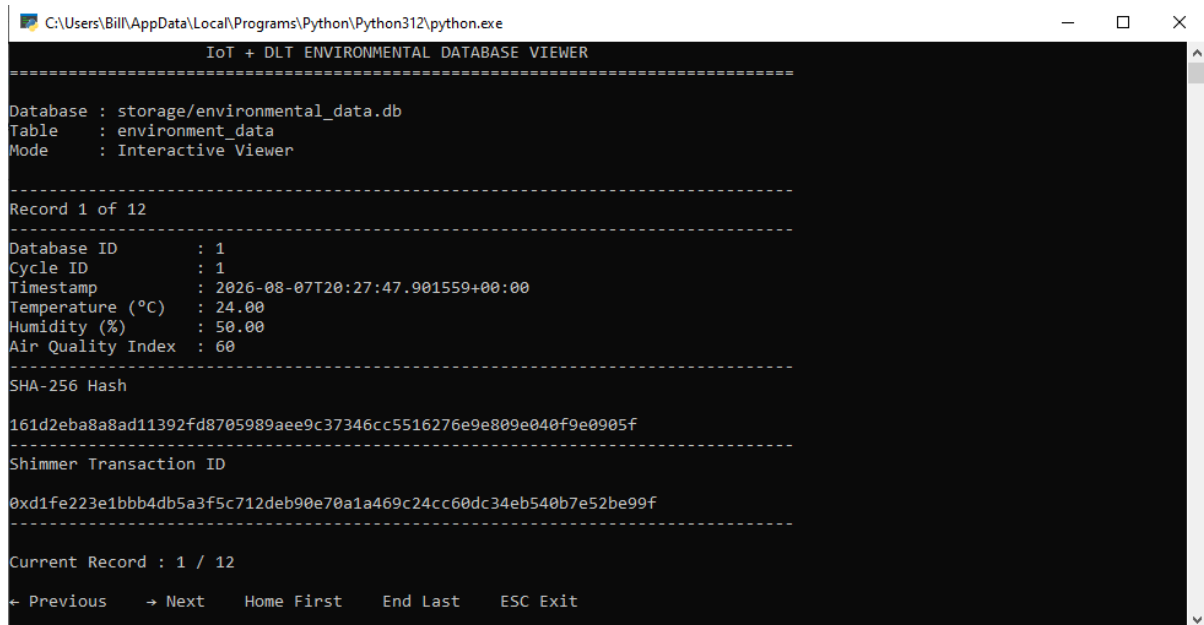
161d2eba8a8ad11392fd8705989aee9c37346cc5516276e9e809e040f9e0905f

και συμφωνεί με το hash που είναι αποθηκευμένο στη βάση για τον πρώτο κύκλο.

Το αντίστοιχο Block ID είναι:

0xd1fe223e1bbb4db5a3f5c712deb90e70a1a469c24cc60dc34eb540b7e52be99f

Η συμφωνία αυτή επιβεβαιώνει ότι το hash που δημιουργήθηκε τοπικά δημοσιεύτηκε πραγματικά στο Shimmer και μπορεί να συνδεθεί με την αντίστοιχη τοπική καταχώριση μέσω του Block ID.



```

C:\Users\Bill\AppData\Local\Programs\Python\Python312\python.exe
IoT + DLT ENVIRONMENTAL DATABASE VIEWER
=====
Database : storage/environmental_data.db
Table    : environment_data
Mode     : Interactive Viewer
=====
Record 1 of 12
=====
Database ID      : 1
Cycle ID        : 1
Timestamp       : 2026-08-07T20:27:47.901559+00:00
Temperature (°C) : 24.00
Humidity (%)    : 50.00
Air Quality Index : 60
=====
SHA-256 Hash
161d2eba8a8ad11392fd8705989aee9c37346cc5516276e9e809e040f9e0905f
=====
Shimmer Transaction ID
0xd1fe223e1bbb4db5a3f5c712deb90e70a1a469c24cc60dc34eb540b7e52be99f
=====
Current Record : 1 / 12
← Previous      → Next      Home First      End Last      ESC Exit

```

Σχήμα 5.5: Database Viewer first record

Η δοκιμή αυτή επιβεβαιώνει την πραγματική DLT διασύνδεση της εφαρμογής και διακρίνει την τελική υλοποίηση από μία απλή τοπική προσομοίωση.

5.7 Έλεγχος ακεραιότητας των δεδομένων

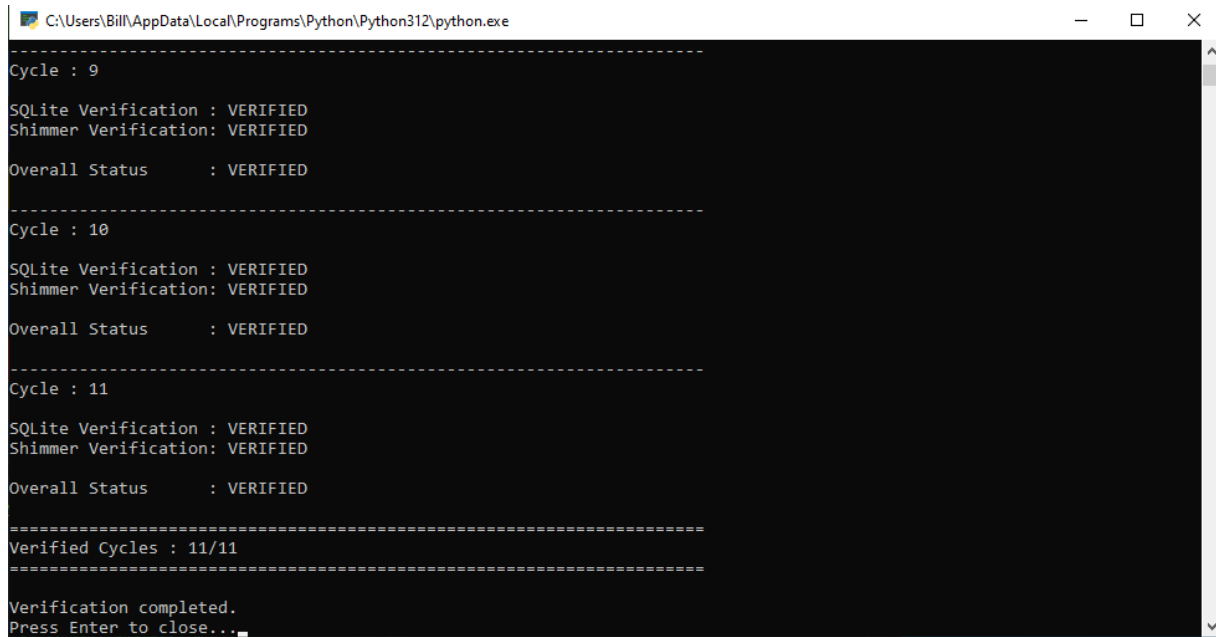
Η διαδικασία ελέγχου ακεραιότητας πραγματοποιήθηκε με το `verify_integrity.py`. Για κάθε αποθηκευμένο κύκλο υπολογίζεται εκ νέου το SHA-256 hash και πραγματοποιούνται δύο συγκρίσεις: μία με το hash που είναι αποθηκευμένο στη SQLite και μία με το hash που ανακτάται από το αντίστοιχο Shimmer block. Ο τελικός μηχανισμός επαλήθευσης υλοποιεί ακριβώς αυτή τη διαδικασία.

Στην τελική δοκιμή το πρόγραμμα παρουσίασε για τους εξεταζόμενους κύκλους:

SQLite Verification : VERIFIED
Shimmer Verification: VERIFIED
Overall Status : VERIFIED

και στο τέλος της διαδικασίας:

Verified Cycles : 11/11



```

C:\Users\Bill\AppData\Local\Programs\Python\Python312\python.exe
-----
Cycle : 9
SQLite Verification : VERIFIED
Shimmer Verification: VERIFIED
Overall Status      : VERIFIED
-----
Cycle : 10
SQLite Verification : VERIFIED
Shimmer Verification: VERIFIED
Overall Status      : VERIFIED
-----
Cycle : 11
SQLite Verification : VERIFIED
Shimmer Verification: VERIFIED
Overall Status      : VERIFIED
=====
Verified Cycles : 11/11
=====
Verification completed.
Press Enter to close...
  
```

Σχήμα 5.6: Integrity Verification

Το αποτέλεσμα 11/11 σημαίνει ότι και οι 11 αποθηκευμένοι κύκλοι πέρασαν επιτυχώς τόσο τον τοπικό έλεγχο ακεραιότητας όσο και τον έλεγχο έναντι του αντίστοιχου hash που ανακτήθηκε από το Shimmer.

Το αποτέλεσμα αυτό αποτελεί την κύρια πειραματική ένδειξη ότι, για το σύνολο των κύκλων που εξετάστηκαν, τα δεδομένα της τοπικής βάσης συμφωνούν τόσο με τα αποθηκευμένα hashes όσο και με τα hashes που είχαν δημοσιευθεί στο DLT.

5.8 Δοκιμή σκόπιμης αλλοίωσης

Για την αξιολόγηση της συμπεριφοράς του μηχανισμού σε περίπτωση μεταβολής ήδη αποθηκευμένων δεδομένων πραγματοποιήθηκε δοκιμή σκόπιμης αλλοίωσης. Κατά τη δοκιμή τροποποιήθηκε ελεγχόμενα μία τιμή υπάρχοντος κύκλου στη βάση δεδομένων, χωρίς να μεταβληθούν το αρχικό hash και η αντίστοιχη καταχώρηση στο Shimmer.

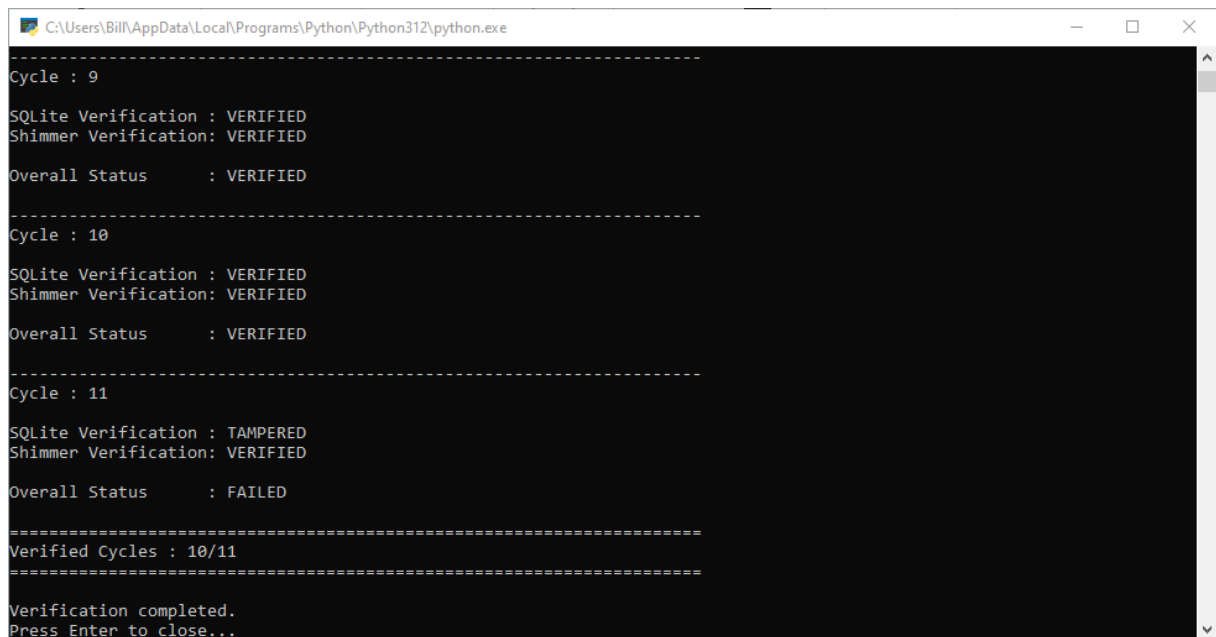
Μετά την τροποποίηση εκτελέστηκε εκ νέου ο μηχανισμός επαλήθευσης. Το hash του τροποποιημένου κύκλου υπολογίστηκε ξανά και χρησιμοποιήθηκε για τη σύγκριση με το hash της SQLite και με το hash που είχε προηγουμένως δημοσιευθεί στο Shimmer.

Η δοκιμή σχεδιάστηκε ώστε μία μεταβολή στα δεδομένα του κύκλου να προκαλεί απόκλιση μεταξύ του νέου υπολογισμένου hash και των αρχικών τιμών αναφοράς. Με αυτόν τον τρόπο αξιολογείται η δυνατότητα του συστήματος να διακρίνει την αρχική κατάσταση των δεδομένων από μία μεταγενέστερη τροποποίηση.

Μετά την ολοκλήρωση της δοκιμής πραγματοποιήθηκε επαναφορά της τροποποιημένης τιμής στην αρχική της κατάσταση και εκτελέστηκε εκ νέου η διαδικασία επαλήθευσης.

Πειραματική αξιολόγηση και αποτελέσματα

Τα συγκεκριμένα αποτελέσματα της δοκιμής αλλοίωσης αξιολογούνται ξεχωριστά από το αποτέλεσμα της κανονικής επαλήθευσης και συνδέονται με τη λειτουργία του μηχανισμού TAMPERED που υλοποιείται στο `verify_integrity.py`.



```
C:\Users\Bill\AppData\Local\Programs\Python\Python312\python.exe
-----
Cycle : 9
SQLite Verification : VERIFIED
Shimmer Verification: VERIFIED
Overall Status      : VERIFIED
-----
Cycle : 10
SQLite Verification : VERIFIED
Shimmer Verification: VERIFIED
Overall Status      : VERIFIED
-----
Cycle : 11
SQLite Verification : TAMPERED
Shimmer Verification: VERIFIED
Overall Status      : FAILED
=====
Verified Cycles : 10/11
=====
Verification completed.
Press Enter to close...
```

Σχήμα 5.7: Tampering test

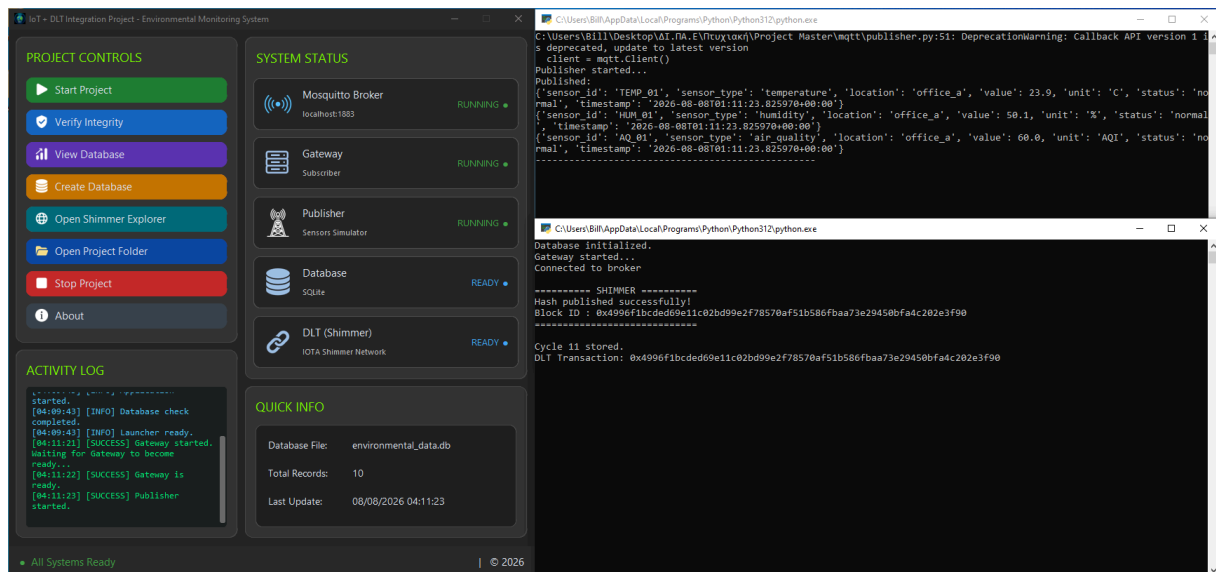
5.9 Έλεγχος του Launcher και της συνολικής λειτουργίας

Η λειτουργία του Launcher αξιολογήθηκε κατά την εκτέλεση του ολοκληρωμένου project. Το γραφικό περιβάλλον παρέχει ξεχωριστές ενδείξεις για τον MQTT Broker, τον Gateway, τον Publisher, τη βάση δεδομένων και το DLT, καθώς και τις βασικές λειτουργίες διαχείρισης της εφαρμογής.



Σχήμα 5.8: Project Standby

Κατά την εκτέλεση του project, ο Launcher χρησιμοποιήθηκε για την εκκίνηση του συστήματος και την παρακολούθηση των βασικών διεργασιών. Το activity log καταγράφει τις σχετικές ενέργειες εκκίνησης, ενώ οι ενδείξεις του περιβάλλοντος ενημερώνονται σύμφωνα με την κατάσταση των components.



Σχήμα 5.2: Project Running

Παράλληλα, μέσω του ίδιου περιβάλλοντος είναι διαθέσιμες οι λειτουργίες Verify Integrity, View Database, Create Database, Open Shimmer Explorer και Stop Project.

Η λειτουργία του Launcher επομένως αξιολογήθηκε ως ενιαίο επίπεδο διαχείρισης, μέσω του οποίου είναι δυνατή η εκτέλεση και ο έλεγχος των βασικών λειτουργιών χωρίς την ανάγκη χειροκίνητης διαχείρισης κάθε επιμέρους εργαλείου.

5.10 Συγκεντρωτική αξιολόγηση των απαιτήσεων

Οι λειτουργικές απαιτήσεις που καθορίστηκαν στον Πίνακα 3.1 συνδέονται με τις αντίστοιχες δοκιμές του παρόντος κεφαλαίου. Η αντιστοίχιση αυτή επιτρέπει τη συνολική αξιολόγηση της τελικής υλοποίησης.

Πίνακας 5.2: Αντιστοίχιση λειτουργικών απαιτήσεων και αποτελεσμάτων

Κωδικός	Λειτουργική απαίτηση	Τεκμηρίωση	Αξιολόγηση
R1	Παραγωγή δεδομένων θερμοκρασίας, υγρασίας και ποιότητας αέρα	5.3	Επιτυχής
R2	Μετάδοση των παραγόμενων δεδομένων μέσω MQTT	5.4	Επιτυχής
R3	Συλλογή των επιμέρους μετρήσεων από τον Gateway	5.4	Επιτυχής
R4	Δημιουργία ολοκληρωμένου κύκλου	5.4–5.5	Επιτυχής
R5	Αποθήκευση των κύκλων σε SQLite	5.5	Επιτυχής
R6	Δημιουργία SHA-256 hash	5.5	Επιτυχής
R7	Δημοσίευση του hash στο Shimmer μέσω Tagged Data block	5.6	Επιτυχής

R8	Αποθήκευση του αντίστοιχου Block ID	5.5–5.6	Επιτυχής
R9	Έλεγχος της ακεραιότητας των αποθηκευμένων δεδομένων	5.7	Επιτυχής
R10	Ανίχνευση σκόπιμης αλλοίωσης δεδομένων	5.8	Επιτυχής
R11	Παροχή γραφικού περιβάλλοντος Launcher	5.9	Επιτυχής
R12	Εκκίνηση του Publisher μετά την ετοιμότητα του Gateway	5.9	Ελέγχθηκε
R13	Τερματισμός των διεργασιών που ανήκουν στο project	5.9	Επιτυχής

Για την απαίτηση R10 η δοκιμή αλλοίωσης δεδομένων παρουσίασε επιτυχή ανίχνευση της μεταβολής, όπως τεκμηριώνεται στο Σχήμα 5.7. Για την απαίτηση R12 διατηρείται η διατύπωση “Ελέγχθηκε”, καθώς δεν υπάρχει στο συγκεκριμένο σύνολο τεκμηρίων αντίστοιχο πλήρες πειραματικό στιγμιότυπο που να επιτρέπει ισχυρότερο χαρακτηρισμό.

Η υπόλοιπη αξιολόγηση παρουσιάζει επιτυχή λειτουργία των βασικών υποσυστημάτων και επιβεβαιώνει ότι η τελική υλοποίηση καλύπτει τις κύριες λειτουργικές απαιτήσεις της εργασίας.

5.11 Επίλογος

Στο παρόν κεφάλαιο παρουσιάστηκε η πειραματική αξιολόγηση της τελικής εφαρμογής και τα αποτελέσματα των βασικών δοκιμών. Η λειτουργία των προσομοιωμένων αισθητήρων, η μετάδοση των δεδομένων μέσω MQTT και η δημιουργία των κύκλων επιβεβαιώθηκαν κατά την εκτέλεση του συστήματος.

Παράλληλα, επιβεβαιώθηκε η αποθήκευση των δεδομένων στη SQLite και η δημιουργία των αντίστοιχων SHA-256 hashes. Ιδιαίτερη σημασία είχε η πραγματική δημοσίευση των hashes στο δίκτυο Shimmer μέσω Tagged Data block και η διατήρηση του αντίστοιχου Block ID.

Για το σύνολο των 11 εξεταζόμενων κύκλων ο μηχανισμός επαλήθευσης παρουσίασε αποτέλεσμα 11/11 VERIFIED, επιβεβαιώνοντας τη συμφωνία μεταξύ των τοπικά υπολογισμένων hashes, των αποθηκευμένων τιμών στη βάση και των αντίστοιχων hashes που ανακτήθηκαν από το Shimmer.

Παράλληλα, πραγματοποιήθηκε δοκιμή σκόπιμης αλλοίωσης για την αξιολόγηση της συμπεριφοράς του μηχανισμού σε περίπτωση τροποποίησης αποθηκευμένων δεδομένων. Τέλος, εξετάστηκε η λειτουργία του Launcher και η συνεργασία των βασικών components κατά την εκτέλεση του συστήματος.

Τα αποτελέσματα του κεφαλαίου αποτελούν τη βάση για την τελική αξιολόγηση της προσέγγισης και για την εξαγωγή των συμπερασμάτων της διπλωματικής εργασίας.

Κεφάλαιο 6ο: Συμπεράσματα και μελλοντική εργασία

6.1 Συμπεράσματα

Η παρούσα διπλωματική εργασία είχε ως αντικείμενο τον σχεδιασμό και την υλοποίηση ενός συστήματος περιβαλλοντικής παρακολούθησης που συνδυάζει τεχνολογίες Internet of Things και Distributed Ledger Technology. Για τις ανάγκες της εργασίας αναπτύχθηκε μία πλήρης προσομοιωμένη ροή, η οποία ξεκινά από την παραγωγή των περιβαλλοντικών μετρήσεων και καταλήγει στην αποθήκευση, στην κρυπτογραφική αποτύπωση και στην επαλήθευση της ακεραιότητάς τους.

Η τελική υλοποίηση περιλαμβάνει τρεις προσομοιωμένους αισθητήρες για θερμοκρασία, υγρασία και ποιότητα αέρα, μηχανισμό επικοινωνίας μέσω MQTT, Gateway για τη συγκέντρωση και οργάνωση των μετρήσεων, τοπική βάση SQLite και μηχανισμό δημιουργίας SHA-256 hash. Παράλληλα, το κρυπτογραφικό αποτύπωμα δημοσιεύεται στο δίκτυο Shimmer μέσω Tagged Data block και διατηρείται το αντίστοιχο Block ID, επιτρέποντας τη μεταγενέστερη ανάκτηση της καταχώρισης.

Η πειραματική αξιολόγηση έδειξε ότι η βασική ροή του συστήματος λειτουργεί όπως σχεδιάστηκε. Στη βάση δεδομένων που εξετάστηκε καταγράφηκαν 11 ολοκληρωμένοι κύκλοι, ενώ ο μηχανισμός επαλήθευσης παρουσίασε αποτέλεσμα 11/11 VERIFIED. Αυτό σημαίνει ότι για το σύνολο των εξεταζόμενων κύκλων υπήρξε συμφωνία μεταξύ του hash που υπολογίστηκε εκ νέου, της τιμής που ήταν αποθηκευμένη στη SQLite και του hash που ανακτήθηκε από την αντίστοιχη καταχώριση στο Shimmer.

Ιδιαίτερη σημασία έχει επίσης το γεγονός ότι η διασύνδεση με το Shimmer δεν περιορίστηκε σε θεωρητικό επίπεδο. Το hash ενός κύκλου δημοσιεύθηκε πραγματικά στο δίκτυο και η αντίστοιχη καταχώριση μπορούσε να ανακτηθεί μέσω του Block ID. Με αυτόν τον τρόπο επιτεύχθηκε η σύνδεση της τοπικής αποθήκευσης με ένα ανεξάρτητο εξωτερικό σημείο αναφοράς.

Συνολικά, η εργασία έδειξε ότι οι τεχνολογίες IoT και DLT μπορούν να συνδυαστούν σε μία ενιαία εφαρμογή περιβαλλοντικής παρακολούθησης, ακόμη και σε περιβάλλον προσομοίωσης. Η υλοποίηση επέτρεψε την πρακτική μελέτη της επικοινωνίας μεταξύ επιμέρους components, της δημιουργίας κρυπτογραφικών αποτυπωμάτων και της χρήσης ενός DLT για την ενίσχυση της δυνατότητας επαλήθευσης των δεδομένων.

6.2 Αξιολόγηση της προσέγγισης

Η προσέγγιση που ακολουθήθηκε παρουσίασε το πλεονέκτημα ότι διαχώρισε σαφώς τις διαφορετικές λειτουργίες του συστήματος. Η παραγωγή των μετρήσεων, η επικοινωνία μέσω MQTT, η επεξεργασία στον Gateway, η τοπική αποθήκευση και η δημοσίευση στο DLT υλοποιήθηκαν ως διακριτά υποσυστήματα. Ο διαχωρισμός αυτός διευκολύνει τόσο την ανάπτυξη όσο και τον έλεγχο των επιμέρους λειτουργιών.

Η χρήση του hash ως ενδιάμεσου αποτυπώματος αποτέλεσε επίσης σημαντικό στοιχείο της σχεδίασης. Το πλήρες περιεχόμενο των περιβαλλοντικών μετρήσεων παραμένει στη SQLite, ενώ στο Shimmer

καταγράφεται το κρυπτογραφικό τους αποτύπωμα. Έτσι, το DLT χρησιμοποιείται συμπληρωματικά και όχι ως αντικατάσταση της τοπικής βάσης.

Παράλληλα, η ύπαρξη ξεχωριστού μηχανισμού επαλήθευσης επιτρέπει την ανεξάρτητη αξιολόγηση της ακεραιότητας των δεδομένων. Η δυνατότητα σύγκρισης τόσο με τη SQLite όσο και με το Shimmer προσφέρει δύο διαφορετικά επίπεδα ελέγχου και ενισχύει τη διαφάνεια της διαδικασίας.

Τέλος, η προσθήκη του Launcher μετέτρεψε τα επιμέρους scripts σε μία πιο ολοκληρωμένη εφαρμογή, παρέχοντας ένα ενιαίο σημείο διαχείρισης της εκτέλεσης, της βάσης δεδομένων και της επαλήθευσης.

6.3 Περιορισμοί της υλοποίησης

Παρά τα θετικά αποτελέσματα, η παρούσα υλοποίηση περιλαμβάνει ορισμένους περιορισμούς που πρέπει να ληφθούν υπόψη.

Ο σημαντικότερος είναι ότι οι αισθητήρες είναι προσομοιωμένοι και επομένως δεν αξιολογήθηκε η συμπεριφορά πραγματικού hardware. Οι τιμές παράγονται από λογισμικό και ακολουθούν προκαθορισμένη λογική μεταβολής. Κατά συνέπεια, η εργασία αποδεικνύει τη λειτουργία της αρχιτεκτονικής και της ροής δεδομένων, αλλά όχι την απόδοση του συστήματος σε πραγματικές συνθήκες μέτρησης.

Επιπλέον, η επικοινωνία MQTT και η τοπική βάση λειτουργούν στο πλαίσιο του συγκεκριμένου υπολογιστικού περιβάλλοντος στο οποίο πραγματοποιήθηκαν οι δοκιμές. Δεν αξιολογήθηκε η συμπεριφορά του συστήματος σε μεγάλης κλίμακας ανάπτυξη με πολλούς ταυτόχρονους αισθητήρες ή πολλαπλούς Gateways.

Τέλος, η χρήση δημόσιου Shimmer node σημαίνει ότι η διαθεσιμότητα της εξωτερικής υπηρεσίας αποτελεί εξωτερική προϋπόθεση για τη λειτουργία της DLT πλευράς της εφαρμογής. Η εργασία δεν εξετάζει διαφορετικές τοπολογίες ή εναλλακτικές στρατηγικές διαχείρισης του DLT σε παραγωγικό περιβάλλον.

6.4 Μελλοντικές επεκτάσεις

Μία φυσική μελλοντική επέκταση της εργασίας είναι η αντικατάσταση των προσομοιωμένων αισθητήρων με πραγματικές συσκευές. Με αυτόν τον τρόπο θα μπορούσε να αξιολογηθεί η συμπεριφορά του συστήματος σε πραγματικές περιβαλλοντικές συνθήκες και να εξεταστούν ζητήματα όπως η κατανάλωση ενέργειας, η αξιοπιστία των μετρήσεων και οι πραγματικές διακυμάνσεις των αισθητήρων.

Επιπλέον, η αρχιτεκτονική θα μπορούσε να επεκταθεί ώστε να υποστηρίζει περισσότερους αισθητήρες και περισσότερες περιβαλλοντικές παραμέτρους. Θα μπορούσε επίσης να εξεταστεί η χρήση πολλαπλών Gateways ή απομακρυσμένων κόμβων, ώστε το σύστημα να μπορεί να εφαρμοστεί σε μεγαλύτερη γεωγραφική περιοχή.

Μία ακόμη πιθανή επέκταση είναι η ανάπτυξη πιο ολοκληρωμένης υποδομής απομακρυσμένης παρακολούθησης, στην οποία τα δεδομένα θα παρουσιάζονται σε διαδικτυακό περιβάλλον και θα είναι δυνατή η παρακολούθηση της κατάστασης των αισθητήρων και των καταχωρήσεων του DLT σε πραγματικό χρόνο.

Τέλος, θα μπορούσαν να εξεταστούν πιο σύνθετες τεχνικές επαλήθευσης, μεγαλύτερη αυτοματοποίηση της διαχείρισης των blocks και μηχανισμοί που θα επιτρέπουν τη διαχείριση μεγαλύτερου αριθμού καταχωρίσεων χωρίς να αυξάνεται σημαντικά η πολυπλοκότητα της εφαρμογής.

Με βάση την παρούσα υλοποίηση, οι παραπάνω επεκτάσεις μπορούν να αποτελέσουν φυσική συνέχεια της εργασίας και να οδηγήσουν από ένα πλήρως λειτουργικό σύστημα προσομοίωσης προς μία λύση που θα μπορεί να αξιολογηθεί σε πραγματικές συνθήκες.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] M. Fagan, K. Megas, K. Scarfone, and M. Smith, *IoT Device Cybersecurity Capability Core Baseline*, NIST Interagency/Internal Report 8259A, National Institute of Standards and Technology, 2020.
- [2] K. Boeckl et al., *Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks*, NISTIR 8228, National Institute of Standards and Technology, 2019. doi: 10.6028/NIST.IR.8228.
- [3] J. Saini, M. Dutta, and G. Marques, “Indoor Air Quality Monitoring Systems Based on Internet of Things: A Systematic Review,” *International Journal of Environmental Research and Public Health*, vol. 17, no. 14, Art. no. 4942, 2020.
- [4] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, eds., *MQTT Version 5.0*, OASIS Standard, Mar. 2019.
- [5] National Institute of Standards and Technology, *Secure Hash Standard (SHS)*, FIPS PUB 180-4, Aug. 2015.
- [6] D. Yaga, P. Mell, N. Roby, and K. Scarfone, *Blockchain Technology Overview*, NISTIR 8202, National Institute of Standards and Technology, 2018.
- [7] R. Kuhn, D. Yaga, and J. Voas, “Rethinking Distributed Ledger Technology,” **Computer**, vol. 52, no. 2, pp. 68–72, 2019, doi: 10.1109/MC.2019.2898162.
- [8] IOTA Foundation, *IOTA Documentation*, IOTA Documentation Portal, accessed Aug. 2026.
- [9] Shimmer, *Shimmer Network – Technology*, Shimmer Network, accessed Aug. 2026.
- [10] IOTA Foundation, “A Deep Dive into Stardust,” *Shimmer Blog*, May 31, 2022.
- [11] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010, doi: 10.1016/j.comnet.2010.05.010.
- [12] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013, doi: 10.1016/j.future.2013.01.010.
- [13] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015, doi: 10.1109/COMST.2015.2444095.
- [14] S. Sethi and S. R. Sarangi, “Internet of Things: Architectures, Protocols, and Applications,” *Journal of Electrical and Computer Engineering*, vol. 2017, Art. no. 9324035, 2017, doi: 10.1155/2017/9324035.

- [15] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, “Security, privacy and trust in Internet of Things: The road ahead,” *Computer Networks*, vol. 76, pp. 146–164, 2015, doi: 10.1016/j.comnet.2014.11.008.
- [16] W. Kassab and K. A. Darabkh, “A–Z survey of Internet of Things: Architectures, protocols, applications, recent advances, future directions and recommendations,” *Journal of Network and Computer Applications*, vol. 163, Art. no. 102663, 2020, doi: 10.1016/j.jnca.2020.102663.
- [17] I. Petrescu, E. Niculae, V. Vulturescu, A. Dimitrescu, and L. M. Ungureanu, “Transport and Application Layer Protocols for IoT: Comprehensive Review,” *Technologies*, vol. 13, no. 12, Art. no. 583, 2025, doi: 10.3390/technologies13120583.
- [18] C. Pereira and A. Aguiar, “Towards Efficient Mobile M2M Communications: Survey and Open Challenges,” *Sensors*, vol. 14, no. 10, pp. 19582–19608, 2014, doi: 10.3390/s141019582.
- [19] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, “On blockchain and its integration with IoT. Challenges and opportunities,” *Future Generation Computer Systems*, vol. 88, pp. 173–190, 2018, doi: 10.1016/j.future.2018.05.046.
- [20] A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, “Blockchain and IoT Integration: A Systematic Survey,” *Sensors*, vol. 18, no. 8, Art. no. 2575, 2018, doi: 10.3390/s18082575.
- [21] H.-N. Dai, Z. Zheng, and Y. Zhang, “Blockchain for Internet of Things: A Survey,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8076–8094, 2019, doi: 10.1109/JIOT.2019.2920987.
- [22] F. Casino, T. K. Dasaklis, and C. Patsakis, “A systematic literature review of blockchain-based applications: Current status, classification and open issues,” *Telematics and Informatics*, vol. 36, pp. 55–81, 2019, doi: 10.1016/j.tele.2018.11.006.
- [23] IOTA Foundation, *IOTA – Not Just Another Blockchain*, IOTA Foundation, 2022.
- [24] SQLite, *SQLite Documentation*, SQLite Documentation, accessed Aug. 2026.
- [25] Eclipse Foundation, *Eclipse Paho MQTT Python Client Documentation*, accessed Aug. 2026.
- [26] Eclipse Foundation, *Eclipse Mosquitto Documentation*, accessed Aug. 2026.
- [27] PyInstaller Development Team, *Using Spec Files*, PyInstaller Documentation, accessed Aug. 2026.
- [28] T. Bray, ed., *The JavaScript Object Notation (JSON) Data Interchange Format*, RFC 8259, Internet Engineering Task Force, Dec. 2017.

ΠΑΡΑΡΤΗΜΑ Α : Κώδικας Αισθητήρων

A.1 Αισθητήρας Θερμοκρασίας

```
import random

from datetime import datetime, timezone


class TemperatureSensor:

    def __init__(
        self,
        sensor_id="TEMP_01",
        location="office_a",
        initial_temperature=24.0
    ):

        self.sensor_id = sensor_id
        self.sensor_type = "temperature"
        self.location = location
        self.unit = "C"

        self.temperature = initial_temperature
        self.last_change = 0.0

    def update_temperature(self):

        change = random.choices(
            population=[0.0, 0.1, -0.1],
            weights=[70, 15, 15],
            k=1
        )[0]

        self.temperature += change
```

```

        self.temperature = max(
            18.0,
            min(30.0, self.temperature)
        )

        self.last_change = change

    def get_status(self):

        if self.temperature > 30:
            return "critical"

        if self.temperature >= 28:
            return "warning"

        return "normal"

    def generate_reading(self):

        self.update_temperature()

        return {
            "sensor_id": self.sensor_id,
            "sensor_type": self.sensor_type,
            "location": self.location,
            "value": round(self.temperature, 1),
            "unit": self.unit,
            "status": self.get_status(),
            "timestamp": datetime.now(
                timezone.utc
            ).isoformat()
        }

```

```
def get_last_change(self):  
    return self.last_change
```

A.2 Αισθητήρας Υγρασίας

```
import random  
  
from datetime import datetime, timezone  
  
class HumiditySensor:  
    def __init__(  
        self,  
        sensor_id="HUM_01",  
        location="office_a",  
        initial_humidity=50.0  
    ):  
        self.sensor_id = sensor_id  
        self.sensor_type = "humidity"  
        self.location = location  
        self.unit = "%"  
  
        self.humidity = initial_humidity  
  
    def update_humidity(  
        self,  
        temperature_change=0.0  
    ):  
  
        random_change = random.choices(  
            population=[0.0, 0.1, -0.1],  
            weights=[70, 15, 15],
```

```

        k=1

    )[0]

    temperature_effect = -temperature_change

    self.humidity += (
        random_change +
        temperature_effect
    )

    self.humidity = max(
        30.0,
        min(80.0, self.humidity)
    )

def get_status(self):

    if self.humidity < 35:
        return "warning"

    if self.humidity > 75:
        return "warning"

    return "normal"

def generate_reading(
    self,
    temperature_change=0.0
):

    self.update_humidity(
        temperature_change

```

```

    )

    return {
        "sensor_id": self.sensor_id,
        "sensor_type": self.sensor_type,
        "location": self.location,
        "value": round(self.humidity, 1),
        "unit": self.unit,
        "status": self.get_status(),
        "timestamp": datetime.now(
            timezone.utc
        ).isoformat()
    }

```

A.3 Αισθητήρας Ποιότητας Αέρα

```

import random

from datetime import datetime, timezone

class AirQualitySensor:
    def __init__(
        self,
        sensor_id="AQ_01",
        location="office_a",
        initial_aqi=60.0
    ):
        self.sensor_id = sensor_id
        self.sensor_type = "air_quality"
        self.location = location
        self.unit = "AQI"

```

```

self.aqi = initial_aqi

def update_aqi(
    self,
    temperature,
    humidity
):

    random_change = random.choices(
        population=[0.0, 1.0, -1.0],
        weights=[70, 15, 15],
        k=1
    )[0]

    environmental_effect = 0

    if temperature > 26:
        environmental_effect += 1

    if humidity < 40:
        environmental_effect += 1

    self.aqi += (
        random_change +
        environmental_effect
    )

    self.aqi = max(
        0,
        min(500, self.aqi)
    )

```

```

def get_status(self):

    if self.aqi > 150:
        return "critical"

    if self.aqi > 100:
        return "warning"

    return "normal"

def generate_reading(
    self,
    temperature,
    humidity
):

    self.update_aqi(
        temperature,
        humidity
    )

    return {
        "sensor_id": self.sensor_id,
        "sensor_type": self.sensor_type,
        "location": self.location,
        "value": round(self.aqi, 0),
        "unit": self.unit,
        "status": self.get_status(),
        "timestamp": datetime.now(
            timezone.utc
        ).isoformat()
    }

```

ΠΑΡΑΡΤΗΜΑ Β : Κώδικας Gateway

```
"""
IoT + DLT Integration Project

MQTT Gateway
Data Collection, Hash Generation,
and Shimmer DLT Integration
"""

import os
import sys

PROJECT_ROOT = os.path.dirname(
    os.path.dirname(
        os.path.abspath(__file__)
    )
)

if PROJECT_ROOT not in sys.path:
    sys.path.insert(0, PROJECT_ROOT)

import json

import paho.mqtt.client as mqtt

from storage.database import (
    initialize_database,
    insert_cycle,
    get_last_cycle_id
)
```

```

from dlt.hashing import (
    generate_hash
)

from dlt.shimmer_client import (
    publish_hash
)

BROKER = "localhost"

PORT = 1883

TOPIC = "environment/#"

latest_data = {
    "temperature": None,
    "humidity": None,
    "air_quality": None,
    "timestamp": None
}

cycle_counter = get_last_cycle_id()

def create_cycle():
    """
    Creates a complete environmental measurement cycle
    from the latest sensor readings.
    """
    global cycle_counter

```

```

        cycle_counter += 1

    return {

        "cycle_id":
            cycle_counter,

        "temperature":
            latest_data["temperature"],

        "humidity":
            latest_data["humidity"],

        "air_quality":
            latest_data["air_quality"],

        "timestamp":
            latest_data["timestamp"]
    }

def on_connect(
    client,
    userdata,
    flags,
    reason_code,
    properties=None
):

    print("Connected to broker")

```

```

client.subscribe(TOPIC)

# Notify launcher that Gateway is ready
client.publish(
    "gateway/status",
    "READY",
    qos=1
)

def on_message(
    client,
    userdata,
    msg
):

    payload = json.loads(
        msg.payload.decode()
    )

    topic = msg.topic

    latest_data["timestamp"] = (
        payload["timestamp"]
    )

    if topic == "environment/temperature":

        latest_data["temperature"] = (
            payload["value"]
        )

```

```

elif topic == "environment/humidity":

    latest_data["humidity"] = (
        payload["value"]
    )

elif topic == "environment/airquality":

    latest_data["air_quality"] = (
        payload["value"]
    )

# Create a cycle only when all sensors
# have provided a new measurement.
if (

    latest_data["temperature"] is not None
    and

    latest_data["humidity"] is not None
    and

    latest_data["air_quality"] is not None

):

    cycle = create_cycle()
    # Generate a SHA-256 hash for the completed cycle.
    cycle_hash = generate_hash(
        cycle
    )

```

```

cycle["hash"] = cycle_hash

# Publish the hash to the Shimmer network.
transaction_id = publish_hash(
    cycle_hash
)

cycle["transaction_id"] = (
    transaction_id
)

# Store the completed cycle together with
# its hash and Shimmer Block ID.
insert_cycle(cycle)

print(
    f"Cycle {cycle['cycle_id']} stored."
)

print(
    f"DLT Transaction: "
    f"{transaction_id}"
)

latest_data["temperature"] = None
latest_data["humidity"] = None
latest_data["air_quality"] = None
latest_data["timestamp"] = None

initialize_database()

client = mqtt.Client(

```

```

    mqtt.CallbackAPIVersion.VERSION2
)

client.on_connect = on_connect

client.on_message = on_message

client.connect(
    BROKER,
    PORT,
    60
)

print("Gateway started...")

client.loop_forever()

```

ΠΑΡΑΡΤΗΜΑ C : Κώδικας Publisher

```

"""
IoT + DLT Integration Project

MQTT Publisher
Simulated Environmental Sensors
"""

import os
import sys

PROJECT_ROOT = os.path.abspath(
    os.path.join(

```

```

        os.path.dirname(__file__),
        ".."
    )
)

if PROJECT_ROOT not in sys.path:
    sys.path.insert(0, PROJECT_ROOT)

import json
import time

import paho.mqtt.client as mqtt

from sensors.temperature_sensor import (
    TemperatureSensor
)

from sensors.humidity_sensor import (
    HumiditySensor
)

from sensors.air_quality_sensor import (
    AirQualitySensor
)

BROKER = "localhost"
PORT = 1883

TOPICS = {
    "temperature": "environment/temperature",

```

```

        "humidity": "environment/humidity",
        "air_quality": "environment/airquality"
    }

client = mqtt.Client()

client.connect(BROKER, PORT, 60)

# Initialize the simulated sensors.
temp_sensor = TemperatureSensor()

humidity_sensor = HumiditySensor()

aq_sensor = AirQualitySensor()

print("Publisher started...")

# Continuously generate and publish
# environmental sensor readings.
while True:

    # Generate a new set of sensor readings.
    temp_reading = (
        temp_sensor.generate_reading()
    )

    humidity_reading = (
        humidity_sensor.generate_reading(
            temperature_change=
                temp_sensor.get_last_change()

```

```

    )

)

aq_reading = (
    aq_sensor.generate_reading(
        temperature=temp_reading["value"],
        humidity=humidity_reading["value"]
    )
)

# Publish the sensor readings
# to their corresponding MQTT topics.
client.publish(
    TOPICS["temperature"],
    json.dumps(temp_reading)
)

client.publish(
    TOPICS["humidity"],
    json.dumps(humidity_reading)
)

client.publish(
    TOPICS["air_quality"],
    json.dumps(aq_reading)
)

print("Published:")
print(temp_reading)
print(humidity_reading)
print(aq_reading)
print("-" * 50)

```

```
# Wait one minute before  
# generating the next cycle.  
time.sleep(60)
```

ΠΑΡΑΡΤΗΜΑ D : Κώδικας Κρυπτογραφικής επεξεργασίας και επαλήθευσης

D.1 Hashing.py

```
"""  
IoT + DLT Integration Project  
  
SHA-256 Hash Generation  
Environmental Data Integrity  
"""  
  
import hashlib  
import json  
  
def generate_hash(cycle):  
    """  
    Generates a deterministic SHA-256 hash  
    from a normalized environmental  
    measurement cycle.  
    """  
  
    # Normalize the data to ensure  
    # consistent hash generation.  
    normalized_cycle = {  
  
        "cycle_id":
```

```

        int(cycle["cycle_id"]),

        "timestamp":
            str(cycle["timestamp"]),

        "temperature":
            round(float(cycle["temperature"]), 2),

        "humidity":
            round(float(cycle["humidity"]), 2),

        "air_quality":
            int(cycle["air_quality"])
    }

    # Convert the normalized cycle into
    # a deterministic JSON string.
    cycle_string = json.dumps(
        normalized_cycle,
        sort_keys=True
    )

    # Generate the SHA-256 hash.
    cycle_hash = hashlib.sha256(
        cycle_string.encode()
    ).hexdigest()

    return cycle_hash

```

D.2 Verify_Integrity.py

```
"""
IoT + DLT Integration Project

Integrity Verification
SQLite and Shimmer Validation
"""

import os
import sys
import sqlite3

PROJECT_ROOT = os.path.dirname(
    os.path.dirname(
        os.path.abspath(__file__)
    )
)

if PROJECT_ROOT not in sys.path:
    sys.path.insert(0, PROJECT_ROOT)

from dlt.hashing import generate_hash
from dlt.shimmer_client import get_block

DB_NAME = "storage/environmental_data.db"

def decode_ledger_hash(block):
    """
    Extracts the SHA-256 hash stored inside
    the Shimmer Tagged Data payload.
    """
```

```

"""

payload = block.payload

if payload is None:
    raise ValueError("Block has no payload.")

data = payload.data

if data.startswith("0x"):
    data = data[2:]

return bytes.fromhex(data).decode("utf-8")

def verify_database():
    """
    Verifies the integrity of all stored
    environmental measurement cycles by
    comparing the recalculated SHA-256 hash
    with both the SQLite database and the
    corresponding Shimmer ledger entry.
    """

    connection = sqlite3.connect(DB_NAME)

    cursor = connection.cursor()

    # Retrieve all stored measurement cycles.
    cursor.execute(
        """
        SELECT

```

```

        cycle_id,
        timestamp,
        temperature,
        humidity,
        air_quality,
        hash,
        transaction_id

    FROM environment_data

    ORDER BY cycle_id
    """
)

rows = cursor.fetchall()

connection.close()

print()
print("=" * 70)
print("DATABASE + SHIMMER INTEGRITY VERIFICATION")
print("=" * 70)

verified = 0

for row in rows:

    cycle = {

        "cycle_id": row[0],
        "timestamp": row[1],
        "temperature": row[2],

```

```

        "humidity": row[3],
        "air_quality": row[4]

    }

    stored_hash = row[5]
    transaction_id = row[6]

    calculated_hash = generate_hash(cycle)

    local_ok = calculated_hash == stored_hash

    ledger_ok = False
    ledger_hash = None

    try:

        if transaction_id:

            block = get_block(transaction_id)

            ledger_hash = decode_ledger_hash(block)

            ledger_ok = ledger_hash == calculated_hash

    except Exception as e:

        print(
            f"\nCycle {row[0]} "
            f"(Shimmer Error: {e}) "
        )

```

```

print()

print("-" * 70)

print(f"Cycle : {row[0]}")

print()

if local_ok:
    print("SQLite Verification : VERIFIED")
else:
    print("SQLite Verification : TAMPERED")

if transaction_id:

    if ledger_ok:
        print("Shimmer Verification: VERIFIED")
    else:
        print("Shimmer Verification: FAILED")

else:
    print("Shimmer Verification: NO BLOCK ID")

if local_ok and ledger_ok:

    print("\nOverall Status      : VERIFIED")
    verified += 1

else:

    print("\nOverall Status      : FAILED")

print()

print("=" * 70)

```

```

print(
    f"Verified Cycles : {verified}/{len(rows)}"
)
print("=" * 70)

if __name__ == "__main__":

    verify_database()

print("\nVerification completed.")
input("Press Enter to close...")

```

ΠΑΡΑΡΤΗΜΑ Ε : Κώδικας Shimmer Client

```

"""
IoT + DLT Integration Project

Shimmer Client
DLT Communication
"""

from iota_sdk import Client

class ShimmerClient:
    """
    Handles communication with the
    Shimmer distributed ledger.
    """

    def __init__(self):

```

```

"""
Initializes the Shimmer client
using the public Shimmer node.
"""

self.client = Client(
    nodes=[
        "https://api.shimmer.network"
    ]
)

def connect(self):
    """
    Establishes a connection to the
    Shimmer network and checks
    the node status.
    """

    try:

        info = self.client.get_info()

        print("\n===== SHIMMER =====")
        print("Connected successfully!")
        print(f"Network : {info.nodeInfo.protocol.networkName}")
        print(f"Healthy : {info.nodeInfo.status.isHealthy}")
        print("=====\n")

        return True

    except Exception as e:

```

```

        print("\nShimmer connection failed!")

        print(e)

    return False

def publish_hash(self, hash_value):
    """
    Publishes a SHA-256 hash to the
    Shimmer ledger using a Tagged Data block
    and returns the generated Block ID.
    """

    tag = "0x" + "ENVIRONMENTAL_DATA_HASH".encode().hex()
    data = "0x" + hash_value.encode().hex()

    block_id, _ = self.client.build_and_post_block(
        tag=tag,
        data=data
    )

    print("\n===== SHIMMER =====")
    print("Hash published successfully!")
    print(f"Block ID : {block_id}")
    print("=====\n")

    return block_id

def get_block(self, block_id):

    return self.client.get_block_data(block_id)

```

```

_client = ShimmerClient()

def connect():

    return _client.connect()

def publish_hash(hash_value):

    return _client.publish_hash(hash_value)

def get_block(block_id):

    return _client.get_block(block_id)

```

ΠΑΡΑΡΤΗΜΑ F : Κώδικας Βάσης Δεδομένων

```

"""
IoT + DLT Integration Project

SQLite Database Management
Environmental Data Storage
"""

import sqlite3

DB_NAME = "storage/environmental_data.db"

```

```

def initialize_database():
    """
    Creates the SQLite database and the
    environmental data table if they do not exist.
    """

    connection = sqlite3.connect(DB_NAME)

    cursor = connection.cursor()

    # Create the environmental data table
    # if it does not already exist.
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS environment_data (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            cycle_id INTEGER NOT NULL,

            timestamp TEXT NOT NULL,

            temperature REAL NOT NULL,

            humidity REAL NOT NULL,

            air_quality INTEGER NOT NULL,

            hash TEXT,

            transaction_id TEXT
        """
    )

```

```

    )

    """

)

connection.commit()

connection.close()

print("Database initialized.")

def insert_cycle(cycle):
    """
    Stores one complete environmental
    measurement cycle in the database.
    """

    connection = sqlite3.connect(DB_NAME)

    cursor = connection.cursor()

    # Store a complete environmental
    # measurement cycle in the database.
    cursor.execute(
        """
        INSERT INTO environment_data (

            cycle_id,
            timestamp,
            temperature,
            humidity,
            air_quality,
            hash,

```

```

        transaction_id

    )

VALUES (?, ?, ?, ?, ?, ?, ?)

"""
(
    cycle["cycle_id"],
    cycle["timestamp"],
    cycle["temperature"],
    cycle["humidity"],
    cycle["air_quality"],
    cycle["hash"],
    cycle["transaction_id"]
)
)

connection.commit()
connection.close()

def get_last_cycle_id():
    """
    Returns the latest stored cycle ID.
    If the database is empty, returns 0.
    """

    connection = sqlite3.connect(DB_NAME)

    cursor = connection.cursor()

    # Retrieve the highest stored cycle ID.

```

```

cursor.execute(
    """
    SELECT MAX(cycle_id)
    FROM environment_data
    """
)

result = cursor.fetchone()

connection.close()

if result[0] is None:
    return 0

return result[0]

if __name__ == "__main__":

    # Create the database when this file
    # is executed directly.
    initialize_database()

```

ΠΑΡΑΡΤΗΜΑ G : Κώδικας Launcher

G.1 main.py

```

"""
IoT + DLT Integration Project

Application Entry Point
"""

```

```

from launcher.ui import LauncherUI

def main():
    """
    Starts the launcher application.
    """

    app = LauncherUI()
    app.run()

if __name__ == "__main__":
    main()

```

G.2 config.py

```

"""
IoT + DLT Integration Project

Launcher Configuration
"""

import sys
from pathlib import Path

# Project root directory
if getattr(sys, "frozen", False):
    # Εκτέλεση από launcher.exe
    PROJECT_ROOT = Path(sys.executable).resolve().parent
else:

```

```

# Εκτέλεση από Python

PROJECT_ROOT = Path(__file__).resolve().parent.parent


# Project folders

GATEWAY_DIR = PROJECT_ROOT / "gateway"

MQTT_DIR = PROJECT_ROOT / "mqtt"

DLT_DIR = PROJECT_ROOT / "dlt"

STORAGE_DIR = PROJECT_ROOT / "storage"

VIEW_DATABASE_SCRIPT = PROJECT_ROOT / "storage" / "view_database.py"


# Python scripts

GATEWAY_SCRIPT = GATEWAY_DIR / "gateway.py"

PUBLISHER_SCRIPT = MQTT_DIR / "publisher.py"

VERIFY_SCRIPT = DLT_DIR / "verify_integrity.py"

DATABASE_SCRIPT = STORAGE_DIR / "database.py"

DATABASE_FILE = STORAGE_DIR / "environmental_data.db"


# External programs

MOSQUITTO_COMMAND = "mosquitto -v"


# Shimmer Explorer

SHIMMER_EXPLORER = "https://explorer.shimmer.network/shimmer"

```

```

# Window titles

WINDOW_MOSQUITTO = "Mosquitto Broker"

WINDOW_GATEWAY = "Gateway"

WINDOW_PUBLISHER = "Publisher"

WINDOW_VERIFY = "Integrity Verification"


# Refresh interval (milliseconds)

STATUS_REFRESH = 1000


# Application title

APPLICATION_TITLE = (
    "IoT + DLT Integration Project - Environmental Monitoring System"
)

```

G.3 process_manager.py

```

"""
IoT + DLT Integration Project

Process Manager
Handles process detection, start, stop and status.
"""

import subprocess
import psutil
import sys
import os

```

```

from launcher.config import (
    PROJECT_ROOT,
    GATEWAY_SCRIPT,
    PUBLISHER_SCRIPT,
    VERIFY_SCRIPT,
    VIEW_DATABASE_SCRIPT,
    DATABASE_SCRIPT,
    DATABASE_FILE,
    MOSQUITTO_COMMAND
)

class ProcessManager:

    def __init__(self):
        self.gateway_process = None
        self.publisher_process = None
        self.verify_process = None
        self.database_process = None
        self.mosquitto_process = None

    def process_exists(self, executable):
        executable = executable.lower()
        for proc in psutil.process_iter(["name"]):
            try:
                name = proc.info["name"]
                if name and name.lower() == executable:
                    return True
            except (psutil.NoSuchProcess,
                    psutil.AccessDenied,
                    psutil.ZombieProcess):

```

```

        pass

    return False

def find_python_script(self, script_name):
    script_name = script_name.lower()
    for proc in psutil.process_iter(["name", "cmdline"]):
        try:
            name = proc.info["name"]
            if not name or "python" not in name.lower():
                continue

            cmdline = proc.info["cmdline"]
            if not cmdline:
                continue

            if script_name in " ".join(cmdline).lower():
                return proc

        except (psutil.NoSuchProcess,
                psutil.AccessDenied,
                psutil.ZombieProcess):
            pass

    return None

def mqtt_running(self):
    return self.process_exists("mosquitto.exe")

def gateway_running(self):
    return self.find_python_script("gateway.py") is not None

def publisher_running(self):
    return self.find_python_script("publisher.py") is not None

```

```

def verify_running(self):
    return self.find_python_script("verify_integrity.py") is not
None

def database_running(self):
    return self.find_python_script("view_database.py") is not None

def database_exists(self):
    return os.path.exists(DATABASE_FILE)

def create_database(self):

    if self.database_exists():
        return False

    if getattr(sys, "frozen", False):
        python_cmd = "python"
    else:
        python_cmd = sys.executable

    result = subprocess.run(
        [python_cmd, str(DATABASE_SCRIPT)],
        cwd=PROJECT_ROOT
    )

    return result.returncode == 0

def start_gateway(self):
    if self.gateway_running():
        return False

```

```

self.gateway_process = subprocess.Popen(
    ["python", str(GATEWAY_SCRIPT)],
    cwd=PROJECT_ROOT,
    creationflags=subprocess.CREATE_NEW_CONSOLE
)

return True

def start_publisher(self):
    if self.publisher_running():
        return False

    self.publisher_process = subprocess.Popen(
        ["python", str(PUBLISHER_SCRIPT)],
        cwd=PROJECT_ROOT,
        creationflags=subprocess.CREATE_NEW_CONSOLE
    )

    return True

def start_verify(self):
    if self.verify_running():
        return False

    self.verify_process = subprocess.Popen(
        ["python", str(VERIFY_SCRIPT)],
        cwd=PROJECT_ROOT,
        creationflags=subprocess.CREATE_NEW_CONSOLE
    )

    return True

def start_database(self):

    if self.database_running():

```

```

        return False

    if getattr(sys, "frozen", False):
        python_cmd = "python"
    else:
        python_cmd = sys.executable

    self.database_process = subprocess.Popen(
        [python_cmd, str(VIEW_DATABASE_SCRIPT)],
        cwd=PROJECT_ROOT,
        creationflags=subprocess.CREATE_NEW_CONSOLE
    )

    return True

def stop_python_script(self, script_name):
    process = self.find_python_script(script_name)
    if process:
        process.terminate()
        try:
            process.wait(timeout=5)
        except psutil.TimeoutExpired:
            process.kill()

def stop_all(self):
    self.stop_python_script("publisher.py")
    self.stop_python_script("gateway.py")
    self.stop_python_script("verify_integrity.py")

```

G.4 ui.py

```

"""

```

IoT + DLT Integration Project

Launcher User Interface

"""

import os

import sys

import webbrowser

import time

from datetime import datetime

from pathlib import Path

from tkinter import messagebox

import customtkinter as ctk

from PIL import Image, ImageTk

import sqlite3

import paho.mqtt.client as mqtt

from launcher.config import (

APPLICATION_TITLE,

STATUS_REFRESH,

PROJECT_ROOT,

SHIMMER_EXPLORER,

DATABASE_FILE,

)

from launcher.logger import ActivityLogger

from launcher.process_manager import ProcessManager

=====

UI LAYOUT

=====

```

WINDOW_WIDTH = 750

WINDOW_HEIGHT = 780

OUTER_MARGIN = 5

CARD_GAP = 5

CARD_RADIUS = 12

CARD_BORDER = 1

TITLE_PAD_X = 16

TITLE_PAD_Y = 16

BUTTON_HEIGHT = 36

BUTTON_PAD_X = 15

BUTTON_PAD_Y = 5

def resource_path(relative_path):
    """Return absolute path to resource (works for development and
    PyInstaller)."""

    if getattr(sys, "frozen", False):
        base_path = sys._MEIPASS
    else:
        base_path = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))

    return os.path.join(base_path, relative_path)

def load_icon(filename):

    return ctk.CTkImage(

        light_image=Image.open(

```

```

        resource_path(f"assets/icons/Buttons/{filename}")

    ),

    dark_image=Image.open(
        resource_path(f"assets/icons/Buttons/{filename}")
    ),

    size=(20, 20)
)

class LauncherUI:

    def __init__(self):

        ctk.set_appearance_mode("dark")
        ctk.set_default_color_theme("blue")

        self.process_manager = ProcessManager()

        self.gateway_ready = False

        self.status_client = mqtt.Client(
            mqtt.CallbackAPIVersion.VERSION2
        )

        self.status_client.on_connect = self.on_status_connect
        self.status_client.on_message = self.on_status_message

        self.window = ctk.CTk()

        icon_path = resource_path("assets/iot_dlt.ico")

        # Windows title bar & taskbar
        self.window.iconbitmap(icon_path)

        # Tkinter keeps a reference to the image

```

```

        icon = ImageTk.PhotoImage(Image.open(icon_path))

        self.window.iconphoto(True, icon)

        self.window._icon = icon

        self.window.title(APPLICATION_TITLE)

        self.window.geometry(f"{WINDOW_WIDTH}x{WINDOW_HEIGHT}")

        self.window.grid_rowconfigure(1, weight=1)

        self.window.grid_rowconfigure(2, weight=0)

        self.window.grid_columnconfigure(0, weight=1)

        self.window.resizable(False, False)

        self.last_update = "Never"

        self.status_icons = {

            "mqtt": ctk.CTkImage(
Image.open(resource_path("assets/icons/status/mqtt.png")),
                size=(40, 40)
            ),

            "gateway": ctk.CTkImage(
Image.open(resource_path("assets/icons/status/gateway.png")),
                size=(40, 40)
            ),

            "publisher": ctk.CTkImage(
Image.open(resource_path("assets/icons/status/publisher.png")),
                size=(40, 40)
            ),

            "database": ctk.CTkImage(
Image.open(resource_path("assets/icons/status/database.png")),

```

```

        size=(40, 40)

    ),

    "dlt": ctk.CTkImage(
Image.open(resource_path("assets/icons/status/DLT.png")),
        size=(40, 40)
    )
}

self.create_widgets()

self.log_message("Application started.")

self.log_message("Database check completed.")

self.log_message("Launcher ready.")

self.update_status()

self.update_quick_info()


def create_widgets(self):

    self.create_main_layout()

    self.create_controls()

    self.create_status()

    self.create_quick_info()

    self.create_log()

    self.create_footer()


# =====
# LAYOUT

```

```

# =====

def create_main_layout(self):

    # =====
    # Main Container
    # =====

    self.main_frame = ctk.CTkFrame(
        self.window,
        fg_color="transparent"
    )

    self.main_frame.grid(
        row=1,
        column=0,
        sticky="nsew",
        padx=OUTER_MARGIN,
        pady=OUTER_MARGIN
    )

    # Two columns
    self.main_frame.grid_columnconfigure(0, weight=1)
    self.main_frame.grid_columnconfigure(1, weight=1)

    # =====
    # LEFT COLUMN
    # =====

    self.left_frame = ctk.CTkFrame(
        self.main_frame,
        fg_color="transparent"

```

```

)

self.left_frame.grid(
    row=0,
    column=0,
    sticky="nsew",
    padx=CARD_GAP,
    pady=CARD_GAP
)

self.left_frame.grid_columnconfigure(0, weight=1)

# Τόιες αναλογίες με πριν
self.left_frame.grid_rowconfigure(0, weight=55)
self.left_frame.grid_rowconfigure(1, weight=45)

# Project Controls
self.left_top_frame = ctk.CTkFrame(
    self.left_frame,
    fg_color="transparent"
)

self.left_top_frame.grid(
    row=0,
    column=0,
    sticky="nsew",
    pady=(0, 10)
)

# Activity Log
self.left_bottom_frame = ctk.CTkFrame(
    self.left_frame,

```

```

        fg_color="transparent"
    )

    self.left_bottom_frame.grid(
        row=1,
        column=0,
        sticky="nsew"
    )

    # =====
    # RIGHT COLUMN
    # =====

    self.right_frame = ctk.CTkFrame(
        self.main_frame,
        fg_color="transparent"
    )

    self.right_frame.grid(
        row=0,
        column=1,
        sticky="nsew",
        padx=CARD_GAP,
        pady=CARD_GAP
    )

    self.right_frame.grid_columnconfigure(0, weight=1)

    # Διαφορετικές αναλογίες
    self.right_frame.grid_rowconfigure(0, weight=60)
    self.right_frame.grid_rowconfigure(1, weight=30)

```

```

# System Status

self.right_top_frame = ctk.CTkFrame(
    self.right_frame,
    fg_color="transparent"
)

self.right_top_frame.grid(
    row=0,
    column=0,
    sticky="nsew",
    pady=(0, 10)
)

# Quick Info

self.right_bottom_frame = ctk.CTkFrame(
    self.right_frame,
    fg_color="transparent"
)

self.right_bottom_frame.grid(
    row=1,
    column=0,
    sticky="nsew"
)

# =====

# CONTROLS

# =====

def create_controls(self):

    self.controls_frame = ctk.CTkFrame(

```

```

        self.left_top_frame,

        fg_color="#323232",

        corner_radius=CARD_RADIUS,

        border_width=CARD_BORDER,

        border_color="#4A4A4A"

    )

    self.controls_frame.pack(

        fill="both",

        expand=True,

        padx=0,

        pady=0

    )

    title = ctk.CTkLabel(

        self.controls_frame,

        text="PROJECT CONTROLS",

        font=("Segoe UI", 18),

        text_color="#7CFC00"

    )

    title.pack(anchor="w", padx=15, pady=(15, 10))

    # -----
    # ΔΙΕΤΑ ΚΟΥΜΠΙΩΝ
    # -----

    self.buttons_frame = ctk.CTkFrame(

        self.controls_frame,

        fg_color="transparent"

    )

```

```

self.buttons_frame.pack(
    fill="both",
    expand=True,
    padx=15,
    pady=(0, 15)
)

self.buttons_frame.grid_columnconfigure(0, weight=1)

buttons = [
    ("Start Project",
     "player-play.png",
     self.start_project,
     "#1E7D32",
     "#2E8B57"),
    ("Verify Integrity",
     "shield-check.png",
     self.verify_integrity,
     "#1565C0",
     "#1976D2"),
    ("View Database",
     "chart-area-line.png",
     self.view_database,
     "#5E35B1",
     "#673AB7"),
    ("Create Database",
     "database.png",
     self.create_database,
     "#C77700",

```

```

        "#D68910"),

        ("Open Shimmer Explorer",
        "world.png",
        self.open_explorer,
        "#006D7A",
        "#00838F"),

        ("Open Project Folder",
        "folder-open.png",
        self.open_project_folder,
        "#0D47A1",
        "#1565C0"),

        ("Stop Project",
        "player-stop.png",
        self.stop_project,
        "#C62828",
        "#D32F2F"),

        ("About",
        "info-circle.png",
        self.about,
        "#3B434B",
        "#4A535C")
    ]

# -----
# ΔΗΜΙΟΥΡΓΙΑ ΚΟΥΜΠΙΩΝ
# -----

```

```

        for i, (text, icon_file, command, color, hover) in
enumerate(buttons):

    icon = load_icon(icon_file)

    button = ctk.CTkButton(
        self.buttons_frame,

        text=text,
        image=icon,

        compound="left",
        anchor="w",

        command=command,

        height=BUTTON_HEIGHT,

        corner_radius=10,

        font=("Segoe UI", 15),

        fg_color=color,

        hover_color=hover
    )

    button.grid(
        row=i,
        column=0,
        sticky="ew",
        pady=BUTTON_PAD_Y

```

```

    )

# =====
# STATUS
# =====

def create_status(self):
    # 1. Δημιουργία του frame
    self.status_frame = ctk.CTkFrame(
        self.right_top_frame,
        fg_color="#323232",
        corner_radius=12,
        border_width=1,
        border_color="#4A4A4A"
    )

    self.status_frame.pack(
        fill="both",
        expand=True,
        padx=0,
        pady=0
    )

    # 2. Τίτλος
    title = ctk.CTkLabel(
        self.status_frame,
        text="SYSTEM STATUS",
        font=("Segoe UI", 18),
        text_color="#7CFC00"
    )

```

```

        title.pack(anchor="w", padx=TITLE_PAD_X, pady=(TITLE_PAD_Y,
10))

self.status_cards_frame = ctk.CTkFrame(
    self.status_frame,
    fg_color="transparent"
)

self.status_cards_frame.pack(
    fill="both",
    expand=True,
    padx=12,
    pady=(0, 10)
)

self.status_cards_frame.grid_columnconfigure(0, weight=1)

# μία γραμμή για κάθε card
for row in range(5):
    self.status_cards_frame.grid_rowconfigure(row, weight=1)

# 3. Δημιουργία Labels
self.mqtt_card = self.create_status_card(
    self.status_icons["mqtt"],
    "Mosquitto Broker",
    "localhost:1883",
    "RUNNING",
    "#43A047"
)

self.gateway_card = self.create_status_card(
    self.status_icons["gateway"],

```

```

        "Gateway",

        "Subscriber",

        "STOPPED",

        "#B0BEC5"

    )

    self.publisher_card = self.create_status_card(

        self.status_icons["publisher"],

        "Publisher",

        "Sensors Simulator",

        "STOPPED",

        "#B0BEC5"

    )

    self.database_card = self.create_status_card(

        self.status_icons["database"],

        "Database",

        "SQLite",

        "READY",

        "#42A5F5"

    )

    self.dlt_card = self.create_status_card(

        self.status_icons["dlt"],

        "DLT (Shimmer)",

        "IOTA Shimmer Network",

        "READY",

        "#42A5F5"

    )

def create_status_card(
    self,

```

```

        icon_image,

        title,

        subtitle,

        state,

        color

    ):

card = ctk.CTkFrame(

    self.status_cards_frame,

    fg_color="#242424",

    corner_radius=10,

    border_width=1,

    border_color="#505050"

)

row = len(self.status_cards_frame.grid_slaves(column=0))

card.grid(

    row=row,

    column=0,

    sticky="nsew",

    pady=5

)

# -----
# GRID
# -----

card.grid_columnconfigure(0, weight=0)
card.grid_columnconfigure(1, weight=1)
card.grid_columnconfigure(2, weight=0)

```

```

card.grid_rowconfigure(0, weight=1)

card.grid_rowconfigure(1, weight=1)


# -----
# ICON
# -----


icon_label = ctk.CTkLabel(
    card,
    image=icon_image,
    text=""
)

icon_label.grid(
    row=0,
    column=0,
    rowspan=2,
    padx=15,
    pady=10,
    sticky="ns"
)


# -----
# TITLE
# -----


title_label = ctk.CTkLabel(
    card,
    text=title,
    font=("Segoe UI", 15)
)

```

```

title_label.grid(
    row=0,
    column=1,
    sticky="sw",
    pady=(8, 0)
)

# -----
# SUBTITLE
# -----

subtitle_label = ctk.CTkLabel(
    card,
    text=subtitle,
    font=("Segoe UI", 11),
    text_color="#BDBDBD"
)

subtitle_label.grid(
    row=1,
    column=1,
    sticky="nw",
    pady=(0, 8)
)

# -----
# STATUS
# -----

status_frame = ctk.CTkFrame(
    card,
    fg_color="transparent"

```

```

    )

    status_frame.grid(
        row=0,
        column=2,
        rowspan=2,
        padx=(10, 15),
        sticky="e"
    )

    state_label = ctk.CTkLabel(
        status_frame,
        text=state,
        font=("Segoe UI", 13),
        text_color=color
    )

    state_label.pack(
        side="left"
    )

    dot_label = ctk.CTkLabel(
        status_frame,
        text="●",
        font=("Segoe UI", 15),
        text_color=color
    )

    dot_label.pack(
        side="left",
        padx=(5, 0)
    )

```

```

        return {
            "card": card,
            "state": state_label,
            "dot": dot_label
        }

# =====
# QUICK INFO
# =====

def create_quick_info(self):

    # =====
    # Create Frame
    # =====

    self.quick_info_frame = ctk.CTkFrame(
        self.right_bottom_frame,
        fg_color="#323232",
        corner_radius=12,
        border_width=1,
        border_color="#4A4A4A"
    )

    self.quick_info_frame.pack(
        fill="both",
        expand=True,
        padx=0,
        pady=0
    )

```

```

# =====

# Create Title

# =====

title = ctk.CTkLabel(
    self.quick_info_frame,
    text="QUICK INFO",
    font=("Segoe UI", 18),
    text_color="#7CFC00"
)

title.pack(
    anchor="w",
    padx=15,
    pady=(15, 12)
)

# =====

# Quick Info Card

# =====

self.quick_info_card = ctk.CTkFrame(
    self.quick_info_frame,
    fg_color="#1F1F1F",
    corner_radius=10,
    border_width=1,
    border_color="#4A4A4A"
)

self.quick_info_card.pack(
    fill="both",
    expand=True,

```

```

        padx=15,

        pady=(0, 15)

    )

    # =====

    # Info Container

    # =====

    info_frame = ctk.CTkFrame(
        self.quick_info_card,
        fg_color="transparent"
    )

    info_frame.pack(
        fill="both",
        expand=True,
        padx=18,
        pady=15
    )

    # =====

    # Grid Configuration

    # =====

    info_frame.grid_columnconfigure(0, weight=0)
    info_frame.grid_columnconfigure(1, weight=1)

    # =====

    # Database File

    # =====

    db_label = ctk.CTkLabel(

```

```

        info_frame,

        text="Database File:",
        font=("Segoe UI", 14),
        anchor="w"
    )

    self.database_value = ctk.CTkLabel(
        info_frame,
        text="environmental_data.db",
        font=("Segoe UI", 14),
        anchor="w"
    )

    db_label.grid(
        row=0,
        column=0,
        sticky="w",
        pady=(0, 12)
    )

    self.database_value.grid(
        row=0,
        column=1,
        sticky="w",
        padx=(35, 0),
        pady=(0, 12)
    )

    # =====
    # Total Records
    # =====

```

```

records_label = ctk.CTkLabel(
    info_frame,
    text="Total Records:",
    font=("Segoe UI", 14),
    anchor="w"
)

self.records_value = ctk.CTkLabel(
    info_frame,
    text="0",
    font=("Segoe UI", 14),
    anchor="w"
)

records_label.grid(
    row=1,
    column=0,
    sticky="w",
    pady=(0, 12)
)

self.records_value.grid(
    row=1,
    column=1,
    sticky="w",
    padx=(35, 0),
    pady=(0, 12)
)

# =====
# Last Update
# =====

```

```

update_label = ctk.CTkLabel(
    info_frame,
    text="Last Update:",
    font=("Segoe UI", 14),
    anchor="w"
)

self.last_update_value = ctk.CTkLabel(
    info_frame,
    text="Never",
    font=("Segoe UI", 14),
    anchor="w"
)

update_label.grid(
    row=2,
    column=0,
    sticky="w"
)

self.last_update_value.grid(
    row=2,
    column=1,
    sticky="w",
    padx=(35, 0)
)

def get_database_filename(self):

    if DATABASE_FILE.exists():
        return DATABASE_FILE.name

```

```

        return "Not Found"

def get_total_records(self):

    if not DATABASE_FILE.exists():
        return "0"

    try:

        conn = sqlite3.connect(DATABASE_FILE)

        cursor = conn.cursor()

        cursor.execute("SELECT COUNT(*) FROM environment_data")

        count = cursor.fetchone()[0]

        conn.close()

        return str(count)

    except Exception:
        return "0"

def update_quick_info(self):

    self.database_value.configure(
        text=self.get_database_filename()
    )

    self.records_value.configure(

```

```

        text=self.get_total_records()

    )

    self.last_update_value.configure(

        text=self.last_update

    )

def refresh_last_update(self):

    self.last_update = datetime.now().strftime("%d/%m/%Y %H:%M:%S")

    self.update_quick_info()

# =====
# LOG
# =====

def create_log(self):

    # =====
    # Create Frame
    # =====

    self.log_frame = ctk.CTkFrame(

        self.left_bottom_frame,

        fg_color="#323232",

        corner_radius=12,

        border_width=1,

        border_color="#4A4A4A"

    )

    self.log_frame.pack(

```

```

        fill="both",

        expand=True,

        padx=0,

        pady=0

    )

    # =====

    # Grid Configuration

    # =====

    self.log_frame.grid_rowconfigure(0, weight=0)    # Title
    self.log_frame.grid_rowconfigure(1, weight=1)    # Textbox

    self.log_frame.grid_columnconfigure(0, weight=1)

    # =====

    # Title

    # =====

    title = ctk.CTkLabel(

        self.log_frame,

        text="ACTIVITY LOG",

        font=("Segoe UI", 18),

        text_color="#7CFC00"

    )

    title.grid(

        row=0,

        column=0,

        sticky="w",

        padx=15,

        pady=(15, 10)

```

```

)

# =====

# Textbox

# =====

self.log_textbox = ctk.CTkTextbox(
    self.log_frame,
    font=("Consolas", 13),
    wrap="word"
)

self.log_textbox.grid(
    row=1,
    column=0,
    sticky="nsew",
    padx=15,
    pady=(0, 15)
)

# =====

# Welcome Message

# =====

self.log_textbox.insert(
    "end",
    "=====\\n"
)

self.log_textbox.insert(
    "end",
    " IoT + DLT Integration Launcher\\n"

```

```

    )

    self.log_textbox.insert(
        "end",
        "=====\n\n"
    )

    self.log_textbox.insert(
        "end",
        "[SYSTEM] Launcher initialized.\n"
    )

    self.log_textbox.insert(
        "end",
        "[SYSTEM] Waiting for user...\n"
    )

    self.log_textbox.tag_config("INFO", foreground="#4FC3F7")
    self.log_textbox.tag_config("SUCCESS", foreground="#00E676")
    self.log_textbox.tag_config("WARNING", foreground="#FFD54F")
    self.log_textbox.tag_config("ERROR", foreground="#FF5252")

    self.log_textbox.configure(state="disabled")

def create_footer(self):

    self.footer = ctk.CTkFrame(
        self.window,
        height=38,
        corner_radius=0,
        fg_color="#2B2B2B"
    )

```

```

self.footer.grid(
    row=2,
    column=0,
    sticky="ew"
)

self.footer.grid_columnconfigure(0, weight=1)
self.footer.grid_columnconfigure(1, weight=0)

# -----
# Left
# -----

self.footer_status = ctk.CTkLabel(
    self.footer,
    text="● System Not Ready",
    font=("Segoe UI", 15),
    text_color="#D32F2F"
)

self.footer_status.grid(
    row=0,
    column=0,
    sticky="w",
    padx=15,
    pady=8
)

# -----
# Right
# -----

```

```

        footer_info = ctk.CTkLabel(
            self.footer,
            text="|    © 2026",
            font=("Segoe UI", 15)
        )

        footer_info.grid(
            row=0,
            column=1,
            sticky="e",
            padx=15
        )

    def update_footer_status(self):

        mqtt_ok = self.process_manager.mqtt_running()
        database_ok = self.process_manager.database_exists()

        if mqtt_ok and database_ok:

            self.footer_status.configure(
                text="● All Systems Ready",
                text_color="#43A047"
            )

        else:

            self.footer_status.configure(
                text="● System Not Ready",
                text_color="#D32F2F"
            )

```

```

def log_message(self, message, level="INFO"):

    current_time = datetime.now().strftime("%H:%M:%S")

    self.log_textbox.configure(state="normal")

    line = f"[{current_time}] [{level}] {message}\n"

    self.log_textbox.insert(
        "end",
        line,
        level
    )

    self.log_textbox.see("end")

    self.log_textbox.configure(state="disabled")

def on_status_connect(
    self,
    client,
    userdata,
    flags,
    reason_code,
    properties=None
):

    if reason_code == 0:
        client.subscribe("gateway/status")

```

```

def on_status_message(
    self,
    client,
    userdata,
    msg
):

    if msg.topic == "gateway/status":

        status = msg.payload.decode()

        if status == "READY":

            self.gateway_ready = True

            self.log_message(
                "Gateway is ready.",
                "SUCCESS"
            )

            if self.process_manager.start_publisher():

                self.log_message(
                    "Publisher started.",
                    "SUCCESS"
                )

            else:

                self.log_message(
                    "Publisher already running.",
                    "INFO"

```

```

        )

        self.refresh_last_update()

def start_project(self):

    # =====
    # Database check
    # =====

    if not self.process_manager.database_exists():

        messagebox.showerror(
            "Database",
            "Δεν υπάρχει βάση δεδομένων.\n\n"
            "Πατήστε πρώτα 'Create Database'."
        )

        self.log_message(
            "Start cancelled. Database not found.",
            "ERROR"
        )

        return

    # =====
    # Reset Gateway status
    # =====

    self.gateway_ready = False

    # =====

```

```

# Connect status client

# =====

try:

    self.status_client.connect(

        "localhost",

        1883,

        60

    )

    self.status_client.loop_start()

except Exception as e:

    self.log_message(

        f"Gateway status listener failed: {e}",

        "ERROR"

    )

    return

# =====

# Start Gateway

# =====

if self.process_manager.start_gateway():

    self.log_message(

        "Gateway started. Waiting for Gateway to become
ready...",

        "SUCCESS"

```

```

    )

else:

    self.log_message(
        "Gateway already running.",
        "INFO"
    )

# =====
# Publisher is NOT started here
# =====
#
# Publisher will be started
# automatically when the Gateway
# sends:
#
# gateway/status -> READY

self.refresh_last_update()

def stop_project(self):

    if (
        not self.process_manager.gateway_running()
        and not self.process_manager.publisher_running()
    ):

        messagebox.showwarning(
            "Project",
            "Project is not running."
        )

```

```

        self.log_message(
            "Stop cancelled. Project is not running.",
            "WARNING"
        )

    return

self.process_manager.stop_all()

self.status_client.loop_stop()

if self.status_client.is_connected():
    self.status_client.disconnect()

self.gateway_ready = False

self.log_message(
    "Project stopped.",
    "SUCCESS"
)

self.refresh_last_update()

def verify_integrity(self):

    if not self.process_manager.database_exists():

        messagebox.showwarning(
            "Database",
            "Δεν υπάρχει βάση δεδομένων.\n\n"

```

```

        "Δημιουργήστε πρώτα μία από το κουμπί 'Create
Database'."

    )

    self.log_message(
        "Integrity verification cancelled. Database not
found.",
        "ERROR"
    )

    return

    if self.process_manager.start_verify():
        self.log_message(
            "Integrity verification started.",
            "SUCCESS"
        )
    else:
        self.log_message(
            "Integrity verification already running.",
            "INFO"
        )

    self.refresh_last_update()

def view_database(self):

    if self.process_manager.start_database():
        self.log_message(
            "Database Viewer started.",
            "SUCCESS"
        )

```

```

else:

    self.log_message(

        "Database Viewer already running.",

        "INFO"

    )

self.refresh_last_update()

def create_database(self):

    if self.process_manager.database_exists():

        messagebox.showinfo(

            "Database",

            "Υπάρχει ήδη βάση δεδομένων."

        )

        self.log_message(

            "Database already exists.",

            "WARNING"

        )

        return

    success = self.process_manager.create_database()

    if success:

        messagebox.showinfo(

            "Database",

            "Δημιουργήθηκε νέα βάση δεδομένων."

        )

```

```

        self.log_message(
            "Database created.",
            "SUCCESS"
        )

    else:

        messagebox.showerror(
            "Database",
            "Αποτυχία δημιουργίας βάσης δεδομένων."
        )

        self.log_message(
            "Database creation failed.",
            "ERROR"
        )

    self.refresh_last_update()

def open_explorer(self):
    webbrowser.open(SHIMMER_EXPLORER)
    self.log_message(
        "Opened Shimmer Explorer.",
        "SUCCESS"
    )

def open_project_folder(self):
    os.startfile(Path(PROJECT_ROOT).resolve())
    self.log_message(
        "Opened project folder.",
        "SUCCESS"
    )

```

```

    )

def update_status(self):

    # MQTT

    if self.process_manager.mqtt_running():

        self.mqtt_card["state"].configure(
            text="RUNNING",
            text_color="#43A047"
        )

        self.mqtt_card["dot"].configure(
            text_color="#43A047"
        )

    else:

        self.mqtt_card["state"].configure(
            text="STOPPED",
            text_color="#D32F2F"
        )

        self.mqtt_card["dot"].configure(
            text_color="#D32F2F"
        )

    # Gateway

    if self.process_manager.gateway_running():

        self.gateway_card["state"].configure(
            text="RUNNING",

```

```

        text_color="#43A047"
    )

    self.gateway_card["dot"].configure(
        text_color="#43A047"
    )

else:

    self.gateway_card["state"].configure(
        text="STOPPED",
        text_color="#D32F2F"
    )

    self.gateway_card["dot"].configure(
        text_color="#D32F2F"
    )

# Publisher
if self.process_manager.publisher_running():

    self.publisher_card["state"].configure(
        text="RUNNING",
        text_color="#43A047"
    )

    self.publisher_card["dot"].configure(
        text_color="#43A047"
    )

else:

```

```

        self.publisher_card["state"].configure(
            text="STOPPED",
            text_color="#D32F2F"
        )

        self.publisher_card["dot"].configure(
            text_color="#D32F2F"
        )

# Database
if self.process_manager.database_exists():

    self.database_card["state"].configure(
        text="READY",
        text_color="#42A5F5"
    )

    self.database_card["dot"].configure(
        text_color="#42A5F5"
    )

else:

    self.database_card["state"].configure(
        text="NOT READY",
        text_color="#D32F2F"
    )

    self.database_card["dot"].configure(
        text_color="#D32F2F"
    )

```

```

# Footer

self.update_footer_status()

self.window.after(
    STATUS_REFRESH,
    self.update_status
)

def about(self):

    self.about_window = ctk.CTkToplevel(self.window)

    icon_path = resource_path("assets/iot_dlt.ico")

    # Windows title bar & taskbar
    self.about_window.iconbitmap(icon_path)

    # Tkinter keeps a reference to the image
    about_icon = ImageTk.PhotoImage(Image.open(icon_path))
    self.about_window.iconphoto(True, about_icon)
    self.about_window._icon = about_icon

    self.about_window.title("About")

    self.about_window.geometry("750x900")

    self.about_window.resizable(False, False)

    self.about_window.transient(self.window)

    self.about_window.grab_set()

```

```

self.about_window.focus()

self.about_image = ctk.CTkImage(
    light_image=Image.open(
        resource_path("assets/About.png")
    ),
    dark_image=Image.open(
        resource_path("assets/About.png")
    ),
    size=(750, 900)
)

self.about_label = ctk.CTkLabel(
    self.about_window,
    image=self.about_image,
    text=""
)

self.about_label.pack(
    fill="both",
    expand=True
)

def run(self):
    self.window.mainloop()

```