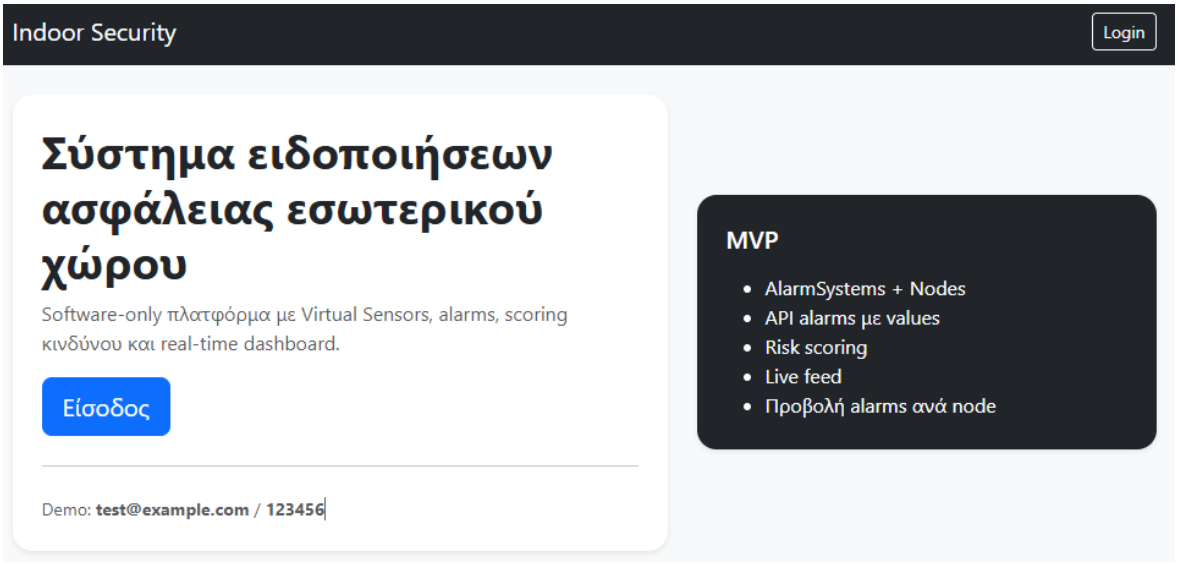
ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Σύστημα ειδοποιήσεων ασφάλειας εσωτερικού χώρου»



Κόκκινος Μάριος

Επιβλέπων

Δρ. Κυριάκος Τσιακμάκης

Ιανουάριος

2026

Σύστημα ειδοποιήσεων ασφάλειας εσωτερικού χώρου

Κωδικός: 25337

Φοιτητής: Κόκκινος Μάριος

Εισηγητής: Δρ Κυριάκος Τσιακμάκης

Ημερομηνία ανάληψης Π.Ε. 29-10-2025

Ημερομηνία περάτωσης Π.Ε. 18-01-2025

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

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

Περίληψη

Η παρούσα πτυχιακή υλοποιεί ένα Σύστημα Ειδοποιήσεων Ασφάλειας Εσωτερικού Χώρου με έμφαση στο λογισμικό και χωρίς χρήση hardware, αξιοποιώντας “Virtual Sensors”. Ο χρήστης (client) δημιουργεί Alarm Systems και προσθέτει κόμβους (nodes) με μοναδικό αναγνωριστικό, οι οποίοι αποστέλλουν alarms στον server μέσω REST API. Ο Flask backend πραγματοποιεί έλεγχο εγκυρότητας, υπολογίζει δείκτη κινδύνου (risk score 0–100) και καταγράφει τα συμβάντα σε βάση MySQL. Στη συνέχεια, τα δεδομένα προβάλλονται σε web περιβάλλον (Bootstrap) με σχεδόν real-time ενημέρωση μέσω polling, καθώς και σελίδα analytics ανά node με πίνακες, δείκτες και γραφήματα. Για δοκιμές και αξιολόγηση αναπτύσσεται simulator που παράγει alarms και σενάρια φόρτου, επιτρέποντας μετρήσεις απόδοσης και αξιοπιστίας. Τέλος, δίνεται έμφαση σε θέματα ασφάλειας (authentication, tokens, validation) και προτείνονται μελλοντικές επεκτάσεις όπως WebSockets, push notifications και πιο εξελιγμένος rule engine.

« Indoor space security notification system »

Abstract

This thesis implements an Indoor Space Security Notification System with a software-focused approach and no hardware, using “Virtual Sensors.” A single user role (client) creates Alarm Systems and adds nodes with unique identifiers, which send alarm events to the server through a REST API. The Flask backend validates incoming data, computes a risk score (0–100) and stores events in a MySQL database. The system then presents alarms through a web interface (Bootstrap) with near real-time updates via polling, as well as a per-node analytics page featuring tables, key indicators, and charts. For testing and evaluation, a simulator generates alarms and load scenarios, enabling performance and reliability measurements. The project also emphasizes security practices such as authentication, token-based access, and input validation. Finally, it outlines realistic future extensions, including WebSockets for true real-time dashboards, mobile push notifications and a more advanced rule engine for event correlation and escalation.

Ευχαριστίες

Ευχαριστώ πολύ τους γονείς μου, τους φίλους μου και τον επιβλέπων καθηγητή μου για την συμπαράσταση και την κατανόηση μέχρι να ολοκληρώσω την πτυχιακή μου εργασία.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1ο: Εισαγωγή

1.1 Πρόβλημα και κίνητρο

Η ασφάλεια ενός εσωτερικού χώρου (π.χ. σπίτι, γραφείο, εργαστήριο, αποθήκη) είναι ένα θέμα που απασχολεί πολλούς ανθρώπους και οργανισμούς. Ακόμα και σε χώρους που φαίνονται «ήσυχοι», μπορεί να προκύψουν περιστατικά όπως παραβίαση εισόδου, κίνηση σε μη επιτρεπτές ώρες, ή ακόμα και απρόσμενες καταστάσεις που απαιτούν άμεση ενημέρωση του χρήστη. Σε αυτά τα σενάρια, το κρίσιμο δεν είναι μόνο να "συμβεί" η ανίχνευση, αλλά να φτάσει γρήγορα και με σαφή τρόπο η πληροφορία στον χρήστη, ώστε να μπορεί να αντιδράσει εγκαίρως.

Τα παραδοσιακά συστήματα συναγερμού βασίζονται κυρίως σε hardware αισθητήρες (π.χ. μαγνητικές επαφές πόρτας/παραθύρου, ανιχνευτές κίνησης, σειρήνες) και συχνά σε μια κεντρική μονάδα που συλλέγει τα σήματα. Αν και αυτά τα συστήματα είναι αποτελεσματικά, έχουν δύο βασικά μειονεκτήματα για μια πτυχιακή που θέλει να εστιάσει στο λογισμικό: (α) χρειάζονται φυσική εγκατάσταση/συντήρηση και (β) δυσκολεύουν την αξιολόγηση και τις δοκιμές σε μεγάλη κλίμακα, ειδικά όταν θέλουμε να παράγουμε πολλά διαφορετικά σενάρια συμβάντων.

Για αυτόν τον λόγο, η παρούσα πτυχιακή επιλέγει μια software μόνο προσέγγιση, δημιουργείται μια κεντρική πλατφόρμα ειδοποιήσεων, όπου οι "αισθητήρες" προσομοιώνονται ως Virtual Sensors. Με απλά λόγια, αντί να έχουμε πραγματικούς αισθητήρες σε πόρτες και δωμάτια, έχουμε λογικές οντότητες (nodes) που στέλνουν ειδοποιήσεις (alarms) στο σύστημα μέσω API. Αυτή η επιλογή επιτρέπει να μελετηθεί το πιο σημαντικό κομμάτι ενός σύγχρονου συστήματος ασφάλειας όπως η ροή δεδομένων, η αξιολόγηση του κινδύνου, η εμφάνιση σε πραγματικό χρόνο και η αξιοπιστία του συστήματος.

Επιπλέον, ένα σύστημα ειδοποιήσεων δεν πρέπει να λειτουργεί απλά ως "μηχανή καταγραφής" συμβάντων. Αν εμφανίζει συνεχώς ειδοποιήσεις χωρίς διάκριση, ο χρήστης καταλήγει να αγνοεί τις σημαντικές. Άρα, είναι απαραίτητο να υπάρχει μια λογική αξιολόγησης όπως ποιο συμβάν είναι πιο σοβαρό, ποιο χρειάζεται άμεση ενημέρωση και ποιο μπορεί να καταγραφεί χωρίς να ενοχλήσει τον χρήστη. Έτσι μπαίνουν οι έννοιες της **σοβαρότητας (severity)**, των **κανόνων (rules)** και ενός **risk score**.

Τέλος, η πτυχιακή αποκτά πιο ωραίο χαρακτήρα όταν δεν περιορίζεται μόνο στο να "δουλεύει", αλλά όταν μπορεί να δοκιμαστεί, να μετρηθεί και να αξιολογηθεί. Για αυτό, το σύστημα σχεδιάζεται ώστε να υποστηρίζει testing και μετρήσεις (π.χ. απόκριση API, ρυθμός alarms ανά δευτερόλεπτο). Έτσι, η εργασία δεν παρουσιάζει μόνο μια εφαρμογή.

1.1.1 Στόχος και αντικείμενο της πτυχιακής

Ο βασικός στόχος της πτυχιακής είναι η ανάπτυξη ενός Συστήματος Ειδοποιήσεων Ασφάλειας Εσωτερικού Χώρου, το οποίο λειτουργεί ως κεντρική πλατφόρμα συλλογής, καταγραφής και προβολής alarms. Το σύστημα επιτρέπει σε έναν χρήστη (client) να δημιουργεί τις δικές του εγκαταστάσεις ασφαλείας (AlarmSystems), να προσθέτει κόμβους/αισθητήρες (Nodes) και να λαμβάνει συμβάντα (Alarms) που παράγονται από αυτούς τους κόμβους.

Πιο συγκεκριμένα, η εφαρμογή περιλαμβάνει:

- **Διαχείριση Alarm Systems:** ο χρήστης μπορεί να δημιουργεί, να επεξεργάζεται και να διαγράφει "συστήματα" (π.χ. "Apartment A", "Office Lab"). Κάθε σύστημα αντιστοιχεί σε έναν χώρο ή εγκατάσταση.
- **Διαχείριση Nodes:** μέσα σε κάθε σύστημα, ο χρήστης μπορεί να προσθέτει nodes με **μοναδικό αναγνωριστικό (node_uid)**. Κάθε node λειτουργεί σαν εικονικός αισθητήρας.
- **Αποστολή alarms μέσω API:** οι nodes στέλνουν alarms στον server μέσω HTTP API. Κάθε alarm μπορεί να περιέχει έως **6 τιμές (v1..v6)**, ώστε να προσομοιώνονται πραγματικά δεδομένα (π.χ. "motion=1", "confidence=86%", "battery=88%").
- **Scoring κινδύνου:** κάθε alarm αξιολογείται και λαμβάνει ένα **risk score (0–100)**. Αυτό βοηθά στη διάκριση των σημαντικών συμβάντων.
- **Real-time παρακολούθηση:** το web dashboard εμφανίζει τα πιο πρόσφατα alarms και επιτρέπει στον χρήστη να βλέπει "ζωντανά" τι συμβαίνει.
- **Analytics ανά node:** υπάρχει ειδική σελίδα ανά node που παρουσιάζει πίνακα alarms, βασικά στατιστικά και γραφήματα.

Το αντικείμενο της πτυχιακής δεν είναι να αντικαταστήσει ένα εμπορικό σύστημα συναγερμού, αλλά να υλοποιήσει και να τεκμηριώσει μια ρεαλιστική πλατφόρμα ειδοποιήσεων, με καθαρό MVP scope, ώστε να είναι υλοποιήσιμη σε πλαίσιο πτυχιακής [1].

1.1.2 Συνεισφορά και καινοτομία της εργασίας

Η εργασία ξεχωρίζει κυρίως στο ότι αντιμετωπίζει το θέμα "σύστημα συναγερμού" από την πλευρά του λογισμικού και της πλατφόρμας, όχι μόνο από την πλευρά των αισθητήρων. Οι κύριες συνεισφορές μπορούν να συνοψιστούν ως εξής:

1. Virtual Sensors αντί hardware: δημιουργείται μια πλήρης υποδομή nodes που λειτουργεί όπως θα λειτουργούσαν πραγματικοί αισθητήρες, αλλά χωρίς φυσικές συσκευές. Αυτό επιτρέπει να δοκιμαστούν πολλά σενάρια και φορτία.

2. Κεντρική πλατφόρμα ειδοποιήσεων: ο server λειτουργεί ως “core system” που δέχεται, καταγράφει και προβάλλει alarms από πολλαπλούς κόμβους.
3. Risk scoring και βάση για έξυπνους κανόνες: κάθε alarm αξιολογείται και αποκτά σκορ. Αυτό είναι το πρώτο βήμα για πιο εξελιγμένη λογική, όπως κανόνες συσχέτισης (correlation) και escalation.
4. Real-time UI και analytics: το σύστημα δεν δείχνει μόνο λίστες alarms, αλλά δίνει και στατιστικά/γραφήματα, βοηθώντας τον χρήστη να καταλάβει τη συμπεριφορά των nodes.
5. Ασφάλεια και αξιοπιστία ως βασικό κομμάτι: επειδή μιλάμε για σύστημα ασφάλειας, δίνεται ιδιαίτερη έμφαση σε authentication, validation, logging και σωστή λειτουργία API.
6. Testing και αξιολόγηση: το σύστημα σχεδιάζεται με τρόπο που επιτρέπει δοκιμές και μετρήσεις, κάτι που κάνει την πτυχιακή πιο “επιστημονική” και όχι απλά μια εφαρμογή.

1.1.3 Περιορισμοί και παραδοχές (MVP)

Για να είναι ρεαλιστικό το έργο ως πτυχιακή, ορίζεται ένα καθαρό MVP (Minimum Viable Product - Ελάχιστο Βιώσιμο Προϊόν). Αυτό σημαίνει ότι υπάρχουν συγκεκριμένα όρια στο τι θα υλοποιηθεί στην πρώτη έκδοση [1].

- Υπάρχει ένας ρόλος χρήστη (client). Δεν υλοποιούνται ρόλοι όπως admin/operator, ώστε να μείνει το σύστημα απλό και ολοκληρώσιμο.
- Το σύστημα χρησιμοποιεί Virtual Sensors, δηλαδή τα alarms προέρχονται από simulator ή από εξωτερικές εφαρμογές που καλούν το API.
- Η “real-time” λειτουργία αρχικά υλοποιείται με polling στο web UI. Αυτό είναι αρκετό για MVP, ενώ πιο προχωρημένες λύσεις (WebSockets, push notifications) μπορούν να μπουν ως μελλοντική εργασία.
- Το risk scoring ξεκινά ως απλό μοντέλο (heuristic), ώστε να είναι κατανοητό και ελέγξιμο. Στη συνέχεια μπορεί να επεκταθεί με πιο σύνθετους κανόνες.

Παρά τους περιορισμούς αυτούς, το MVP παραμένει πλήρες ως προς τις βασικές λειτουργίες: διαχείριση συστημάτων και κόμβων, λήψη alarms, αποθήκευση στη βάση, εμφάνιση και βασική ανάλυση.

1.1.4 Δομή της εργασίας

Η υπόλοιπη πτυχιακή οργανώνεται σε 7 κεφάλαια:

- Στο Κεφάλαιο 2 παρουσιάζεται το θεωρητικό υπόβαθρο: έννοιες alarm systems, ειδοποιήσεων, real-time μεθόδων, κανόνων και risk scoring, καθώς και βασικά θέματα ασφάλειας.
- Στο Κεφάλαιο 3 γίνεται ανάλυση απαιτήσεων και ορίζεται το MVP scope με use cases και λειτουργικές/μη-λειτουργικές απαιτήσεις.
- Στο Κεφάλαιο 4 παρουσιάζεται ο σχεδιασμός του συστήματος: αρχιτεκτονική, βάση δεδομένων, API, και διαγράμματα ροής.
- Στο Κεφάλαιο 5 περιγράφεται η υλοποίηση (Flask, MySQL, Bootstrap, real-time dashboard, analytics, simulator).
- Στο Κεφάλαιο 6 αναλύονται θέματα ασφάλειας, αξιοπιστίας, καθώς και η διαδικασία testing και αξιολόγησης με μετρήσεις.
- Στο Κεφάλαιο 7 παρουσιάζονται τα συμπεράσματα, οι περιορισμοί και οι πιθανές μελλοντικές βελτιώσεις του συστήματος.

Κεφάλαιο 2ο: Θεωρητικό Υπόβαθρο και Σχετικές Έννοιες

2.1 Εισαγωγή στο θεωρητικό πλαίσιο

Ένα "σύστημα συναγερμού" στην πράξη δεν είναι μόνο οι αισθητήρες και η σειρήνα, αλλά μια ολόκληρη διαδικασία όπως συλλογή σημάτων, μετατροπή τους σε συμβάντα (events), αξιολόγηση, αποθήκευση και τελικά ενημέρωση του χρήστη (notification). Στην παρούσα εργασία το hardware αντικαθίσταται από Virtual Sensors, όμως η "λογική" ενός πραγματικού συστήματος παραμένει ίδια, έχουμε κόμβους (nodes), έχουμε alarms και έχουμε μια κεντρική πλατφόρμα που επεξεργάζεται τα δεδομένα.

Επίσης, επειδή το σύστημα έχει στόχο να δίνει ειδοποιήσεις στον χρήστη, το "πότε" και το "πώς" γίνονται οι ενημερώσεις παίζει τεράστιο ρόλο. Άλλο να βλέπει ο χρήστης μια λίστα alarms όταν μπει στο dashboard και άλλο να ενημερώνεται άμεσα όταν συμβεί κάτι σημαντικό. Στο κεφάλαιο αυτό περιγράφονται βασικές επιλογές real-time ενημέρωσης καθώς και οι έννοιες που οδηγούν σε "έξυπνες ειδοποιήσεις", όπως severity levels, rules και risk scoring.

2.1.1 Βασικές έννοιες: Sensor, Node, Event, Alarm, Incident

Σε συστήματα ασφάλειας συναντάμε συχνά τις παρακάτω έννοιες:

- **Sensor (Αισθητήρας):** η πηγή που ανιχνεύει μια κατάσταση ή αλλαγή (π.χ. κίνηση, άνοιγμα πόρτας, δόνηση). Σε πραγματικά συστήματα είναι hardware, εδώ όμως προσομοιώνεται.
- **Node (Κόμβος):** μια μονάδα που αντιπροσωπεύει έναν αισθητήρα ή ένα σύνολο αισθητήρων. Στην εργασία, ο node έχει μοναδικό node_uid και στέλνει alarms στο σύστημα μέσω API.
- **Event (Συμβάν):** μια "απλή" καταγραφή ότι κάτι συνέβη (π.χ. "motion detected"). Το event μπορεί να είναι ανεπεξέργαστο.
- **Alarm:** ένα event που θεωρείται αρκετά σημαντικό ώστε να καταγραφεί ως alarm και πιθανόν να οδηγήσει σε ειδοποίηση.
- **Incident:** μια πιο "βαριά" έννοια, όπου συνήθως μιλάμε για πραγματικό περιστατικό ασφάλειας. Συχνά ένα incident προκύπτει από συνδυασμό alarms ή από επιβεβαίωση από τον χρήστη.

Η διάκριση αυτή βοηθάει στο να καταλάβουμε γιατί χρειάζεται αξιολόγηση. Αν κάθε event γινόταν alarm με ειδοποίηση, ο χρήστης θα λάμβανε υπερβολικά πολλά μηνύματα και θα κατέληγε να τα αγνοεί. Άρα, το σύστημα πρέπει να έχει τρόπο να ξεχωρίζει το "σημαντικό" από το "μη σημαντικό".

2.1.2 Severity (σοβαρότητα) και γιατί είναι απαραίτητη

Η **severity** είναι ένας απλός τρόπος να χαρακτηρίσουμε πόσο σοβαρό είναι ένα alarm [2]. Συνήθως χρησιμοποιούνται επίπεδα όπως:

- **LOW:** μικρή σημασία (π.χ. κίνηση όταν το σύστημα δεν είναι οπλισμένο)
- **MEDIUM:** μέτρια σημασία (π.χ. άνοιγμα πόρτας σε περίεργη ώρα)

- **HIGH:** υψηλή σημασία (π.χ. κίνηση ενώ το σύστημα είναι armed)
- **CRITICAL:** κρίσιμη κατάσταση (π.χ. panic button ή ύποπτο μοτίβο πολλαπλών events)

Η severity είναι χρήσιμη γιατί:

1. επιτρέπει να γίνεται ιεράρχηση των ειδοποιήσεων,
2. βοηθά να εμφανίζονται διαφορετικά στο UI (π.χ. χρώματα),
3. λειτουργεί ως βάση για "κανόνες" και για escalation.

Στην εργασία, η severity περνάει είτε από τον node (ως μέρος του payload) είτε μπορεί να προκύψει και από το ίδιο το σύστημα, ανάλογα με το risk scoring και τους κανόνες που εφαρμόζονται.

2.1.3 Risk scoring: από απλή ειδοποίηση σε "έξυπνη" ειδοποίηση

Η severity δίνει κατηγορίες, αλλά πολλές φορές δεν είναι αρκετή [3]. Για παράδειγμα, δύο alarms "HIGH" μπορεί να διαφέρουν πολύ: ένα μπορεί να είναι λίγο ύποπτο, ενώ το άλλο να είναι σχεδόν σίγουρη παραβίαση. Εκεί βοηθάει το **risk score**, δηλαδή ένας αριθμός (π.χ. 0–100) που εκφράζει πόσο επικίνδυνο θεωρείται το συγκεκριμένο alarm.

Το risk score μπορεί να υπολογιστεί με διάφορους τρόπους. Στην πτυχιακή ξεκινάμε με ένα απλό, κατανοητό μοντέλο (heuristic), βασισμένο σε:

- τον τύπο event (MOTION, DOOR κ.λπ.),
- τη severity,
- αν το σύστημα είναι armed,
- και προαιρετικά κάποιες τιμές παραμέτρων (π.χ. "confidence").

Παράδειγμα απλού σκεπτικού:

- Αν έχουμε MOTION ενώ το σύστημα είναι armed → αυξάνεται το σκορ.
- Αν το "confidence" είναι υψηλό → αυξάνεται λίγο ακόμα.
- Αν είναι disarmed → μειώνεται.

Αυτό το μοντέλο έχει δύο πλεονεκτήματα:

(α) είναι εύκολο να εξηγηθεί και να τεκμηριωθεί,

(β) μπορεί να εξελιχθεί αργότερα σε πιο σύνθετη μορφή (κανόνες, weights, ακόμα και ML).

Rule-based λογική: κανόνες, thresholds και συσχέτιση (correlation)

Τα συστήματα ασφάλειας συχνά δεν βασίζονται μόνο σε "ένα alarm", αλλά στο μοτίβο alarms. Για αυτό χρησιμοποιούνται κανόνες τύπου IF/THEN:

- IF event_type=MOTION AND armed=TRUE THEN severity=HIGH

- IF DOOR opened AND μέσα σε 20s συμβεί MOTION THEN severity=CRITICAL
- IF 5 alarms μέσα σε 1 λεπτό THEN escalation

Εδώ έχουμε η έννοια της **συσχέτισης (correlation)**, δηλαδή να συνδυάζουμε multiple events για να πάρουμε πιο ασφαλές συμπέρασμα [4]. Ένα μόνο "MOTION" μπορεί να είναι ψευδής συναγερμός, αλλά "DOOR + MOTION σε λίγα δευτερόλεπτα" είναι πολύ πιο ύποπτο.

Ένα πρακτικό ζήτημα που εμφανίζεται στα rules είναι τα **cooldowns** [5]. Αν ένας αισθητήρας στέλνει συνεχώς alarms (π.χ. κάθε 2s), το σύστημα δεν πρέπει να "βομβαρδίζει" τον χρήστη. Οπότε συχνά μπαίνουν περιορισμοί όπως:

- "μη στέλνεις ειδοποίηση για τον ίδιο τύπο event πάνω από 1 φορά ανά Χ δευτερόλεπτα"
- "συνένωσε alarms σε ένα summary"

Στην πτυχιακή, η αρχική έκδοση εφαρμόζει βασικό scoring, αλλά ο σχεδιασμός γίνεται ώστε να επιτρέπει επέκταση με rules και correlation.

2.1.4 Real-time ενημέρωση: Polling, Long Polling, WebSockets, Push

Για να "καταλάβει" ο χρήστης ότι υπάρχει νέα ειδοποίηση, το σύστημα πρέπει να παρέχει κάποιο μηχανισμό real-time ενημέρωσης [6-9].

Υπάρχουν διάφορες επιλογές:

2.1.4.1 Polling

Το polling είναι η πιο απλή προσέγγιση: ο client (web UI ή mobile app) ρωτάει τον server κάθε Χ δευτερόλεπτα: "έχεις κάτι νέο;".

Πλεονεκτήματα:

- πολύ εύκολο στην υλοποίηση
- δεν απαιτεί ειδικές τεχνολογίες
- ιδανικό για MVP

Μειονεκτήματα:

- κάνει πολλά requests (ειδικά αν το interval είναι μικρό)
- δεν είναι "instant" (εξαρτάται από το interval)

2.1.4.2 Long Polling

Ο client κάνει request και ο server "κρατάει" τη σύνδεση ανοικτή μέχρι να εμφανιστεί κάτι νέο.

Πλεονεκτήματα:

- λιγότερα άσκοπα requests σε σχέση με polling

Μειονεκτήματα:

- πιο περίπλοκο
- θέλει προσοχή σε timeouts/clients

2.1.4.3 WebSockets

Στους WebSockets ανοίγει μια μόνιμη σύνδεση και ο server μπορεί να στείλει δεδομένα άμεσα.

Πλεονεκτήματα:

- πραγματικό real-time
- ιδανικό για dashboards

Μειονεκτήματα:

- πιο σύνθετο setup
- θέλει σωστή διαχείριση connections

2.1.4.4 Push Notifications (κινητό)

Για κινητές συσκευές, ο σωστός τρόπος είναι push notifications (π.χ. Firebase Cloud Messaging).

Πλεονεκτήματα:

- πραγματική ειδοποίηση ακόμα κι αν η εφαρμογή δεν είναι ανοικτή

Μειονεκτήματα:

- απαιτεί επιπλέον υπηρεσίες και setup

Στην παρούσα εργασία, για να κρατηθεί το scope ρεαλιστικό, το MVP υλοποιεί real-time ενημέρωση στο web UI με polling. Η υποστήριξη push notifications μπορεί να μπει ως μελλοντική επέκταση.

2.1.5 Κεντρική πλατφόρμα ειδοποιήσεων και “event pipeline”

Ένα σημαντικό κομμάτι της θεωρίας είναι το πώς λειτουργεί το “pipeline” ενός alarm:

1. **Παραγωγή event:** ο node “παράγει” ένα event (π.χ. motion=1).
2. **Αποστολή στον server:** μέσω API request.
3. **Επεξεργασία:** validation, υπολογισμός risk score, εφαρμογή κανόνων.
4. **Αποθήκευση στη βάση:** το alarm γίνεται record στη DB.
5. **Παρουσίαση/ειδοποίηση:** το UI το εμφανίζει σε real-time.
6. **Αντίδραση χρήστη:** πιθανό acknowledge ή δράση.

Αυτό το pipeline είναι χρήσιμο γιατί επιτρέπει να χωριστεί το σύστημα σε ξεκάθαρα στάδια. Επίσης, διευκολύνει την αξιολόγηση: μπορούμε να μετρήσουμε χρόνο από το στάδιο 2 μέχρι το στάδιο 5 (latency) και να δούμε πόσο αποδοτικό είναι το σύστημα.

2.1.6 Βασικές αρχές ασφάλειας σε συστήματα ειδοποιήσεων

Επειδή το σύστημα σχετίζεται με ασφάλεια χώρου, πρέπει να είναι ασφαλές και ως λογισμικό. Τα βασικά σημεία είναι:

- **Authentication:** να μην μπορεί οποιοσδήποτε να στέλνει alarms ή να βλέπει δεδομένα.
- **Token-based πρόσβαση για nodes:** κάθε node έχει token, ώστε να αποφεύγονται πλαστά alarms.
- **Password hashing:** οι κωδικοί δεν αποθηκεύονται ποτέ σε απλό κείμενο.
- **Input validation:** οι παράμετροι (values v1..v6, τύπος event) πρέπει να ελέγχονται.
- **Logging:** καταγραφή για debugging και auditing.
- **Rate limiting:** προστασία από flooding (πολλά alarms σε λίγο χρόνο).

Υπάρχουν βασικές έννοιες που χρειάζονται για να κατανοηθεί το σύστημα ειδοποιήσεων όπως οι διαφορές event/alarm/incident, τα επίπεδα severity, η λογική risk scoring, οι κανόνες και η συσχέτιση alarms καθώς και οι διαθέσιμες τεχνικές real-time ενημέρωσης.

Κεφάλαιο 3ο: Ανάλυση Διαδικασιών και Ορισμός MVP

Σε αυτό το κεφάλαιο γίνεται η ανάλυση διαδικασιών του συστήματος ειδοποιήσεων ασφάλειας εσωτερικού χώρου. Ο στόχος είναι να οριστεί ξεκάθαρα τι πρέπει να κάνει η εφαρμογή (λειτουργικές απαιτήσεις) αλλά και πώς πρέπει να το κάνει (μη λειτουργικές απαιτήσεις). Παράλληλα, καθορίζεται το MVP scope.

3.1 Λειτουργικές διαδικασίες

Παρακάτω παρουσιάζονται οι βασικές λειτουργίες που πρέπει να υποστηρίζει το σύστημα.

- **PR1 — Εγγραφή/Είσοδος χρήστη (Client Authentication)**

  Το σύστημα πρέπει να επιτρέπει σε έναν χρήστη (client) να κάνει login.

  Μετά την είσοδο, ο χρήστης αποκτά πρόσβαση μόνο στα δικά του δεδομένα (alarm systems, nodes, alarms).

  Για το web περιβάλλον χρησιμοποιείται session-based authentication, ενώ για API calls μπορεί να χρησιμοποιηθεί token.

- **PR2 — Δημιουργία και διαχείριση Alarm System**

  Ο client μπορεί να δημιουργεί ένα νέο Alarm System (π.χ. “Home”, “Office”).

  Μπορεί να τροποποιεί στοιχεία του Alarm System (όνομα, περιγραφή).

  Μπορεί να διαγράφει Alarm System.

  Σε περίπτωση διαγραφής, πρέπει να προβλέπεται τι γίνεται με τα nodes/alarms (π.χ. cascade μέσω εφαρμογής).

- **PR3 — Διαχείριση Nodes (Virtual Sensors)**

  Ο client μπορεί να προσθέτει nodes μέσα σε ένα Alarm System.

  Κάθε node πρέπει να έχει μοναδικό node_uid.

  Ο client μπορεί να επεξεργάζεται στοιχεία node (όνομα, ζώνη/χώρος, enabled/disabled).

  Υποστηρίζονται labels για παραμέτρους v1..v6 (ώστε να ξέρουμε τι σημαίνει κάθε τιμή).

  Ο client μπορεί να διαγράφει node.

- **PR4 — Λήψη Alarms μέσω API (Alarm Ingestion)**

  Το σύστημα πρέπει να δέχεται alarms μέσω REST API.

  Κάθε alarm συνδέεται με:

  - system_id / node_id
  - event_type
  - severity (ή προτεινόμενη severity)
  - τιμές v1..v6 (προαιρετικές)
  - message (προαιρετικό)

  Το API πρέπει να ελέγχει τα δεδομένα (validation) και να απορρίπτει λάθος/ελλιπή requests.

- **PR5 — Υπολογισμός Risk Score και βασική αξιολόγηση**

  Για κάθε incoming alarm πρέπει να υπολογίζεται risk score (0–100).

  Το risk score αποθηκεύεται στη βάση μαζί με το alarm.

  Η αρχική υλοποίηση μπορεί να είναι heuristic (κανόνες/βάρη), αλλά πρέπει να είναι τεκμηριωμένη.

- **PR6 — Προβολή Alarms σε web dashboard**

  Ο client βλέπει λίστα alarms (τελευταία συμβάντα) σε dashboard.

  Υποστηρίζονται φίλτρα (π.χ. severity, χρονικό εύρος).

  Υποστηρίζεται pagination/limit (για απόδοση).

- **PR7 — Real-time ενημέρωση στο UI**

  Το dashboard πρέπει να ενημερώνεται “σχεδόν σε πραγματικό χρόνο”.

  Στο MVP αυτό γίνεται με polling ανά Χ δευτερόλεπτα.

  Ο χρήστης βλέπει άμεσα αν εμφανίστηκαν νέα alarms.

- **PR8 — Σελίδα Node Analytics (alarms/τιμές ανά node)**

  Για κάθε node υπάρχει σελίδα που εμφανίζει:

- πίνακα alarms του node
- βασικά KPIs (σύνολο alarms, τελευταίο alarm, avg risk)
- γραφήματα (risk over time, severity distribution, time series για v1/v2/v3)
- επεξήγηση παραμέτρων (τι σημαίνουν v1..v6)

- **PR9 — Virtual Sensors / Simulator**

  Το σύστημα πρέπει να διαθέτει μηχανισμό παραγωγής test alarms (simulator).

  Ο simulator μπορεί να δημιουργεί:

  - τυχαία συμβάντα
  - σενάρια (π.χ. DOOR → MOTION → escalation)

  Στόχος είναι να επιτρέπεται testing και αξιολόγηση χωρίς πραγματικό hardware.

3.2 Άλλες διαδικασίες και προϋποθέσεις

- **Ασφάλεια (Security)**

  Password hashing (καμία αποθήκευση plain password).

  Token auth για API, ειδικά για nodes.

  Input validation και προστασία από SQL injection (parameterized queries).

  Περιορισμός πρόσβασης: ο client βλέπει μόνο τα δικά του.

  Rate limiting (έστω απλό) για προστασία από flooding.

- **Απόδοση (Performance)**

  Το σύστημα πρέπει να χειρίζεται “εύλογο” αριθμό alarms χωρίς καθυστερήσεις.

  Στόχος MVP: π.χ. 1–5 alarms/sec σε testing χωρίς να πέφτει.

  DB indexes σε βασικά πεδία (node_id, created_at, severity).

- **Αξιοπιστία (Reliability)**

  Τα alarms δεν πρέπει να χάνονται εύκολα.

Ο server πρέπει να χειρίζεται σωστά errors (επιστροφή σωστών status codes).

Προαιρετικά: αποφυγή διπλοεγγραφών (idempotency key ή simple dedup logic).

- **Επεκτασιμότητα (Extensibility)**
- Ο σχεδιασμός να επιτρέπει μελλοντικά:
    - rules engine
    - push notifications
    - περισσότερους ρόλους χρηστών
- Η βάση και το API να είναι "καθαρά" και όχι δεμένα με UI.

- **Ευχρηστία (Usability)**
- Απλό και καθαρό UI.
- Εμφάνιση severity με badges/χρώματα.
- Καθαρά μηνύματα σε περίπτωση σφάλματος.

- **Maintainability**
- Καθαρή δομή κώδικα.
- Λογική διαχωρισμένη (routes, db utils, scoring, templates).
- Logging για debugging.

3.3 Σενάρια Χρήσης

Παρακάτω δίνονται βασικά use cases σε απλή μορφή.

Login χρήστη

Περιγραφή: Ο χρήστης εισάγει email/κωδικό και αποκτά πρόσβαση.

Επιτυχία: Δημιουργείται session/token και εμφανίζεται dashboard.

Αποτυχία: Λάθος στοιχεία → μήνυμα σφάλματος.

Δημιουργία Alarm System

Περιγραφή: Ο χρήστης δημιουργεί νέο σύστημα (“Home”).

Επιτυχία: Το σύστημα αποθηκεύεται και εμφανίζεται στη λίστα.

Αποτυχία: Κενό όνομα/λάθος δεδομένα.

Προσθήκη Node σε Alarm System

Περιγραφή: Ο χρήστης προσθέτει node με μοναδικό node_uid.

Επιτυχία: Ο node εμφανίζεται στο σύστημα, με token/labels.

Αποτυχία: node_uid ήδη υπάρχει.

Node στέλνει alarm στον server (API)

Περιγραφή: Ο node κάνει POST στο API με event_type και τιμές.

Επιτυχία: Το alarm αποθηκεύεται, υπολογίζεται risk score.

Αποτυχία: λάθος token/λάθος payload → error response.

Client βλέπει alarms στο dashboard

Περιγραφή: Ο χρήστης βλέπει πρόσφατα alarms και φίλτρα.

Επιτυχία: Εμφάνιση λίστας, severity, risk score, timestamps.

Client βλέπει node analytics

Περιγραφή: Ο χρήστης ανοίγει node_alarms page.

Επιτυχία: Εμφανίζονται KPI/γραφήματα/πίνακας.

3.4 Ορισμός MVP Scope

Για να είναι ρεαλιστικό, το MVP περιλαμβάνει:

1. **Μοναδικός ρόλος client** (login).
2. **CRUD AlarmSystems και Nodes**.
3. **API ingestion alarms** με token έλεγχο.

4. **Risk score** (0–100) με τεκμηριωμένη heuristic λογική.
5. **Dashboard με real-time polling**.
6. **Node analytics σελίδα** με KPIs + charts + επεξήγηση παραμέτρων.
7. **Simulator / Virtual Sensors** για παραγωγή alarms.
8. **Βασικά security features** (hashing, auth, validation).
9. **Testing & αξιολόγηση** (unit/integration + βασικές μετρήσεις).

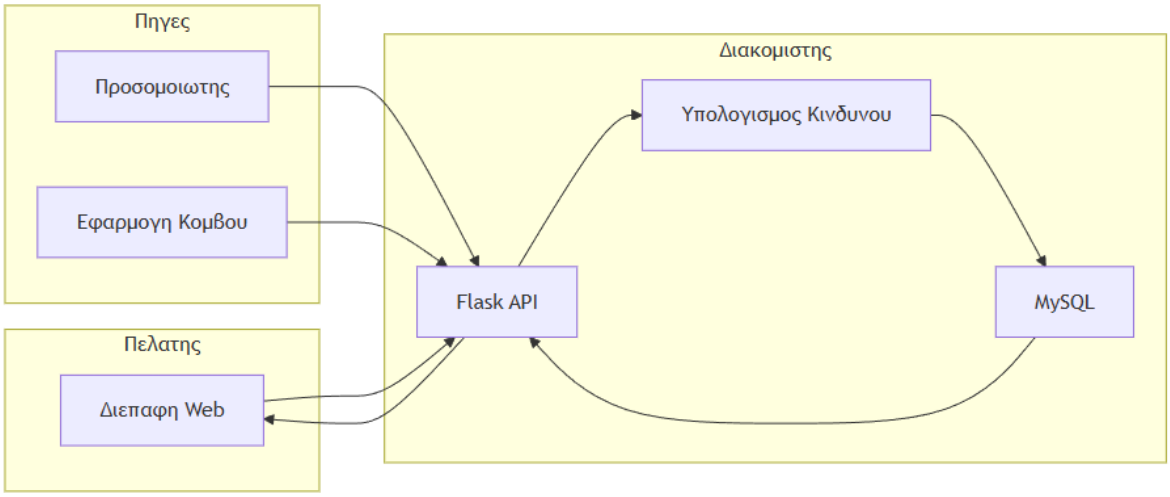
Κεφάλαιο 4ο: Σχεδιασμός Συστήματος (Architecture, DB, API, Diagrams)

4.1 Γενική αρχιτεκτονική συστήματος

Το σύστημα σχεδιάστηκε με λογική "κεντρικής πλατφόρμας" (Core System). Όλα τα alarms, είτε προέρχονται από virtual sensors/simulator είτε από εξωτερικούς clients, καταλήγουν στον Flask server μέσω API. Ο server αναλαμβάνει να ελέγξει τα δεδομένα (validation), να υπολογίσει risk score, να αποθηκεύσει στη βάση MySQL και να τα προβάλει στο web UI [10-12].

Στην πράξη, υπάρχουν τέσσερα βασικά κομμάτια:

1. **Web UI (Bootstrap + Chart.js):** το interface του client (dashboard, nodes, analytics).
2. **Flask Backend:** routes για web pages + REST API endpoints για alarms.
3. **MySQL Database:** αποθήκευση clients, alarm systems, nodes, alarms.
4. **Virtual Sensors/Simulator:** δημιουργεί alarms για testing και αξιολόγηση.



Εικόνα 4.1: Αρχιτεκτονική Κεντρικής Πλατφόρμας Ειδοποιήσεων

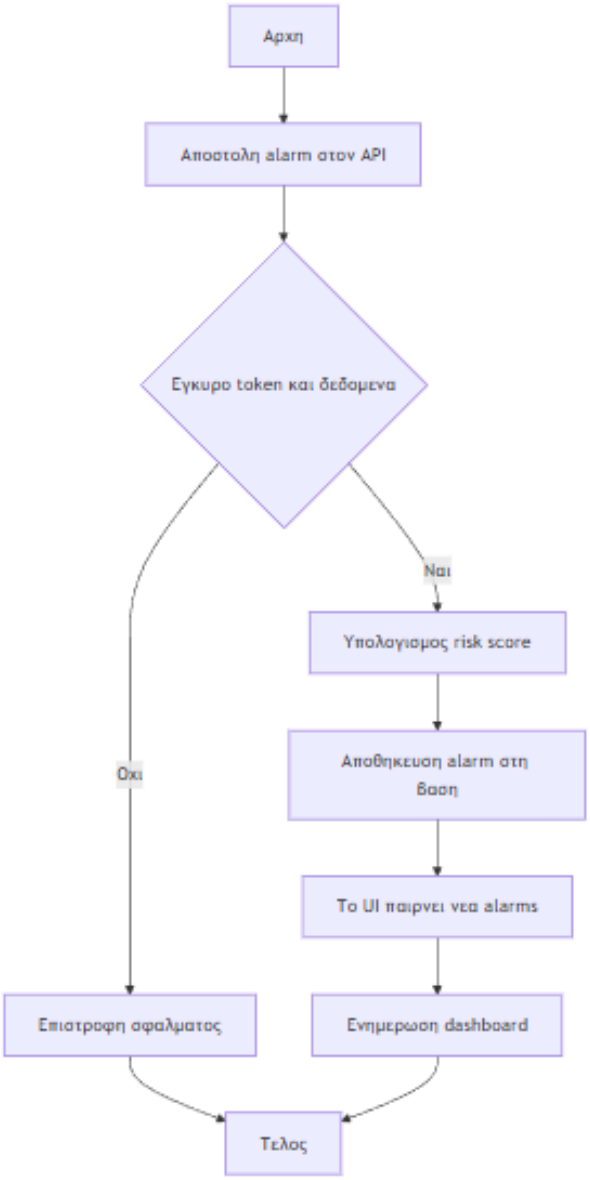
Το διάγραμμα 4.1. δείχνει τα βασικά υποσυστήματα (Web UI, Flask API, DB, Simulator) και τη ροή επικοινωνίας. Τα alarms εισέρχονται μέσω API, αποθηκεύονται στη βάση, και εμφανίζονται στο dashboard σε πραγματικό χρόνο (polling).

4.2 Ροή δεδομένων και βασική λογική λειτουργίας

Η ροή ενός alarm από τη στιγμή που "γεννιέται" μέχρι να εμφανιστεί στον χρήστη είναι συγκεκριμένη και επαναλαμβανόμενη. Αυτό είναι καλό γιατί κάνει το σύστημα μετρήσιμο: μπορούμε να μετρήσουμε latency, να βελτιώσουμε queries, και να ελέγξουμε την αξιοπιστία (π.χ. αν φτάνουν όλα τα alarms).

Γενικά, τα βήματα είναι:

1. **Παραγωγή alarm** (simulator ή node).
2. **Αποστολή στο API** (POST με token/uid).
3. **Validation** (έλεγχος token, required fields, τύποι τιμών).
4. **Scoring** (risk score + ενδεχομένως severity adjustment).
5. **Αποθήκευση** στη MySQL.
6. **Ανάκτηση από UI** (polling/refresh).
7. **Προβολή** σε dashboard + analytics.



Εικόνα 4.2: Flowchart επεξεργασίας και προβολής ενός Alarm

Όπως φαίνεται στην Εικόνα 4.2. η ροή ξεκινά από τη δημιουργία alarm, περνά από έλεγχο εγκυρότητας, scoring, αποθήκευση και τελικά προβολή στο dashboard. Σε περίπτωση αποτυχίας (π.χ. λάθος token), το API επιστρέφει error και το alarm δεν καταγράφεται.

4.3 Σχεδιασμός δεδομένων (λογική οντοτήτων)

Η λογική των βασικών οντοτήτων:

- **Client:** ο χρήστης του συστήματος (ένας ρόλος στο MVP).

- **AlarmSystem:** ένα “χώρος/εγκατάσταση” που ανήκει σε client.
- **Node:** ο εικονικός αισθητήρας μέσα σε ένα AlarmSystem, με μοναδικό node_uid και labels v1..v6.
- **Alarm:** ένα record που περιγράφει event_type, severity, risk_score, v1..v6, timestamp, message.

Η πιο σημαντική σχεδιαστική επιλογή είναι ότι τα alarms συνδέονται με node και system, ώστε να γίνεται εύκολα:

- φιλτράρισμα ανά system,
- προβολή analytics ανά node,
- real-time dashboard με “τελευταία συμβάντα”.

4.4 Σχεδιασμός API

Στο MVP, το API χρειάζεται να καλύπτει 2 βασικές ανάγκες:

1. Ingestion alarms: endpoint που δέχεται alarms από nodes/simulator.
2. Retrieval για UI/mobile: endpoint που επιστρέφει “ό,τι νέο υπάρχει” (π.χ. since_id), ώστε το polling να είναι ελαφρύ.

Παράδειγμα λογικής:

- POST /api/nodes/<node_uid>/alarm (ή αντίστοιχο)
- GET /api/alarms/latest?since_id=...

Κεφάλαιο 5ο: Υλοποίηση Συστήματος (Backend, Βάση, Frontend, Virtual Sensors)

Η υλοποίηση του συστήματος ειδοποιήσεων με τις τεχνολογίες Python Flask, MySQL και Bootstrap (με χρήση γραφημάτων στο UI) ακολουθεί το MVP δηλαδή ένας ρόλος χρήστη (client), CRUD για AlarmSystems και Nodes, λήψη alarms μέσω API, υπολογισμός risk score, real-time προβολή και σελίδα analytics ανά node. Επιπλέον, υλοποιείται ένας Virtual Sensors Simulator, ώστε να μπορούν να παραχθούν δεδομένα για δοκιμές και αξιολόγηση.

Για να είναι το σύστημα κατανοητό δίνεται έμφαση στο web UI (templates), τα endpoints (routes) και στις βοηθητικές λειτουργίες όπως η σύνδεση στη βάση και το scoring.

5.1 Υλοποίηση Backend με Flask

5.1.1 Δομή εφαρμογής

Η δομή φακέλων είναι η εξής:

- app.py (κύριο αρχείο Flask)
- templates/ (HTML templates: welcome, login, dashboard, node_alarms κ.λπ.)
- static/ (CSS, JS, εικόνες)
- db_config.py ή config.py (στοιχεία σύνδεσης DB)
- simulator.py (virtual sensors generator)

5.1.2 Authentication (Web και API)

Στο web UI χρησιμοποιείται session-based authentication όπου όταν ο χρήστης κάνει login, αποθηκεύεται client_id στο session και έτσι προστατεύονται οι σελίδες του dashboard.

Για το API (ειδικά για mobile ή εξωτερικές εφαρμογές) χρησιμοποιείται token-based πρόσβαση. Έτσι:

- ο client μπορεί να πάρει token με login,
- τα endpoints επιστροφής alarms ελέγχουν το token,
- και τα ingestion endpoints μπορούν να ελέγχουν node token για αποφυγή πλαστών alarms.

Αυτή η διπλή λογική είναι χρήσιμη γιατί κάνει το σύστημα επεκτάσιμο: το web λειτουργεί εύκολα με sessions, ενώ οι εφαρμογές (simulator/mobile) λειτουργούν καλύτερα με tokens.

5.1.3 CRUD λειτουργίες (Alarm Systems και Nodes)

Για το MVP οι βασικές CRUD λειτουργίες υλοποιούνται ως routes:

- Alarm Systems:
    - δημιουργία (create)
    - προβολή λίστας στο dashboard
    - επεξεργασία (edit)
    - διαγραφή (delete)
- Nodes:
    - προσθήκη σε συγκεκριμένο system
    - επεξεργασία node (όνομα, ζώνη, enabled, labels)
    - διαγραφή node
    - προβολή analytics σελίδας (node_alarms)

Στο UI επιλέγονται απλές φόρμες με Bootstrap, ενώ στο backend γίνονται έλεγχοι όπως:

- μοναδικότητα node_uid,
- απαγόρευση πρόσβασης σε συστήματα άλλου client,
- σωστή διαχείριση errors (flash μηνύματα).

5.1.4 Υλοποίηση Βάσης Δεδομένων MySQL

5.1.4.1 Αποθήκευση alarms και δεικτοδότηση (indexes)

Τα alarms είναι ο πίνακας που “γεμίζει” πιο γρήγορα. Για αυτό χρειάζονται indexes, ειδικά για queries που τρέχουν συχνά:

- αναζήτηση alarms ανά node και χρονικό εύρος,
- εμφάνιση τελευταίων alarms στο dashboard,
- φίλτρα severity και sorting ανά χρόνο.

Ένα σύνολο indexes είναι:

- (node_id, created_at)
- (system_id, created_at)
- (severity, created_at) (αν χρησιμοποιείται συχνά φίλτρο severity)

5.1.4.2 last_seen και παρακολούθηση κόμβων

Για να φαίνεται αν ένας node είναι ενεργός, ενημερώνεται ένα πεδίο last_seen κάθε φορά που έρχεται alarm από τον node. Έτσι:

- στο dashboard μπορεί να φαίνεται "τελευταία επικοινωνία",
- και μελλοντικά μπορεί να μπει κανόνας "αν δεν έχουμε σήμα για Χ λεπτά, θεωρείται offline".

5.1.5 Υπολογισμός Risk Score (MVP υλοποίηση)

Το risk score (0–100) είναι η καρδιά της "έξυπνης" αξιολόγησης. Στην αρχική μορφή, εφαρμόζεται ένα απλό heuristic μοντέλο:

- βάση ανά event_type (π.χ. DOOR > MOTION)
- αύξηση αν το system είναι armed
- μικρές διορθώσεις από τιμές όπως v2 (π.χ. confidence)
- clamp στο 0–100

Παράδειγμα απλής λογικής:

- DOOR όταν armed → 70
- MOTION όταν armed → 60
- αν confidence > 80 → +10
- αν πολλά alarms σε μικρό χρόνο → +10 (προαιρετικά)

5.1.6 Frontend με Bootstrap και Real-time ενημέρωση

5.1.6.1 Dashboard

Το dashboard είναι η κύρια σελίδα του χρήστη. Εμφανίζει:

- λίστα AlarmSystems και Nodes,
- βασική σύνοψη τελευταίων alarms,

- κουμπιά διαχείρισης (create/edit/delete),
- ένδειξη last_seen ανά node.

Η real-time ενημέρωση γίνεται με polling: το UI κάνει request κάθε Χ δευτερόλεπτα και "τραβάει" νέα alarms.

5.1.6.2 Σελίδα Node Analytics

Η σελίδα node_alarms (που είδαμε και σχεδιάσαμε) είναι το πιο "εντυπωσιακό" μέρος του UI, γιατί:

- έχει KPIs (σύνολο alarms, avg risk, last event)
- έχει charts (risk over time, severity distribution)
- έχει επεξήγηση παραμέτρων v1..v6
- έχει φίλτρα (range, severity)

5.1.7 Virtual Sensors Simulator

Ο simulator είναι ένα script που παίζει ρόλο "virtual αισθητήρων". Ο σκοπός του είναι διπλός:

1. Να παρέχει δεδομένα για demo χωρίς hardware.
2. Να παράγει μεγάλο όγκο alarms για testing (performance, reliability).

Ο simulator μπορεί να υλοποιεί δύο είδη λειτουργίας:

- **Random mode:** τυχαία events σε τυχαίους nodes, με τυχαίες τιμές v1..v3.
- **Scenario mode:** προκαθορισμένα σενάρια, π.χ.:
    - DOOR open → μέσα σε 10–20s MOTION → escalation σε CRITICAL
    - συνεχόμενα alarms (flood) για stress test

Κεφάλαιο 6ο: Ασφάλεια, Αξιοπιστία και Testing / Αξιολόγηση

Σε ένα περιβάλλον ασφάλειας πρέπει να υπάρχει εμπιστοσύνη ότι τα δεδομένα είναι σωστά, ότι δεν μπορεί κάποιος τρίτος να στείλει ψεύτικα alarms ή να δει πληροφορίες που δεν του ανήκουν, και ότι το σύστημα θα παραμείνει σταθερό ακόμη και όταν αυξηθεί ο φόρτος (πολλά alarms σε μικρό χρόνο).

Για τον λόγο αυτό έχουμε τρία βασικά μέρη: (α) μέτρα ασφάλειας, (β) αξιοπιστία/ανθεκτικότητα, και (γ) δοκιμές και αξιολόγηση με μετρήσεις.

6.1 Ασφάλεια (Security)

6.1.1 Authentication και έλεγχος πρόσβασης

Στο MVP υπάρχει ένας ρόλος χρήστη (client), όμως ακόμη και σε αυτή την απλή μορφή απαιτείται σωστό authentication. Στο web περιβάλλον εφαρμόστηκε login με session, ώστε οι προστατευμένες σελίδες (dashboard, nodes, alarms) να μην είναι προσβάσιμες χωρίς είσοδο. Παράλληλα, για το API χρησιμοποιείται token-based πρόσβαση, ειδικά για κλήσεις που γίνονται από external εφαρμογές (simulator ή μελλοντικά mobile app).

Σημαντικό σημείο είναι ότι κάθε query στο backend φιλτράρει με βάση το client_id. Έτσι, ακόμη κι αν κάποιος προσπαθήσει να “μαντέψει” ένα system_id ή node_id, το backend δεν επιστρέφει δεδομένα αν δεν ανήκουν στον συγκεκριμένο client. Αυτή η λογική θεωρείται βασικό επίπεδο authorization.

6.1.2 Αποθήκευση κωδικών με hashing

Οι κωδικοί χρηστών δεν αποθηκεύονται ποτέ σε απλό κείμενο. Χρησιμοποιείται hashing μέσω έτοιμων συναρτήσεων (π.χ. Werkzeug security), ώστε στη βάση να υπάρχει μόνο το hash. Έτσι, ακόμη και σε περίπτωση διαρροής της βάσης, ο πραγματικός κωδικός δεν είναι άμεσα διαθέσιμος.

Επιπλέον, στη login διαδικασία γίνεται σύγκριση hash αντί να γίνεται σύγκριση “plain password”, κάτι που είναι βασική αρχή ασφάλειας σε web εφαρμογές.

6.1.3 Προστασία API (tokens για client και node)

Επειδή τα alarms μπαίνουν στο σύστημα μέσω API, είναι απαραίτητο να υπάρχει έλεγχος ώστε να μην μπορεί οποιοσδήποτε να στείλει alarms. Για αυτό, χρησιμοποιούνται tokens:

- **Client token:** χρησιμοποιείται σε endpoints ανάκτησης δεδομένων (π.χ. "δώσε μου τα νέα alarms").
- **Node token:** μπορεί να χρησιμοποιηθεί σε endpoints εισαγωγής alarms (ingestion), ώστε κάθε node να ταυτοποιείται.

Με αυτό τον τρόπο, ακόμη και αν κάποιος γνωρίζει ένα node_uid, δεν μπορεί να στείλει alarms χωρίς το σωστό token. Αυτή η προσέγγιση μειώνει σημαντικά την πιθανότητα πλαστών alarms.

6.1.4 Input validation και αποφυγή SQL injection

Ένα κρίσιμο σημείο σε εφαρμογές που δέχονται δεδομένα είναι ο έλεγχος του payload. Στο σύστημα εφαρμόζονται βασικοί έλεγχοι:

- απαιτούμενα πεδία (π.χ. event_type),
- επιτρεπτές τιμές για severity,
- έλεγχος ότι v1..v6 είναι αριθμητικές τιμές ή κενές,
- περιορισμός μήκους σε message πεδίο.

Παράλληλα, οι SQL εντολές εκτελούνται με parameterized queries (όχι string concatenation), κάτι που προστατεύει από SQL injection. Αυτό θεωρείται από τις σημαντικότερες πρακτικές ασφαλείας σε συστήματα που χρησιμοποιούν SQL.

6.1.5 CSRF, Rate limiting και καταγραφή ενεργειών

Στο web μέρος, για φόρμες που κάνουν αλλαγές (create/edit/delete), η προστασία από CSRF είναι σημαντική ειδικά σε εφαρμογές που περιλαμβάνουν sessions. Στο MVP μπορεί να χρησιμοποιηθεί ένας απλός μηχανισμός token στις φόρμες (ή βιβλιοθήκη τύπου Flask-WTF).

Επιπλέον, σε ένα σύστημα ειδοποιήσεων υπάρχει κίνδυνος flooding: κάποιος node (ή κακόβουλος χρήστης) να στέλνει πάρα πολλά alarms. Για αυτό προτείνεται rate limiting σε API calls (ακόμη και απλό, π.χ. X requests ανά λεπτό ανά node/token). Έτσι προστατεύεται τόσο η βάση όσο και η διαθεσιμότητα του server.

Η καταγραφή ενεργειών (logging) βοηθά τόσο στην ασφάλεια όσο και στο debugging. Η ύπαρξη logs επιτρέπει να εντοπιστούν ύποπτες προσπάθειες πρόσβασης ή ασυνήθιστη συμπεριφορά.

6.2 Αξιοπιστία και ανθεκτικότητα (Reliability)

6.2.1 Σωστή διαχείριση σφαλμάτων στο API

Το API πρέπει να επιστρέφει σωστούς HTTP κωδικούς:

- 200 OK σε επιτυχία,
- 400 Bad Request σε λάθος payload,
- 401 Unauthorized σε λάθος token,
- 500 μόνο σε πραγματικά server errors.

Αυτό είναι σημαντικό γιατί ο client (π.χ. simulator ή mobile app) μπορεί να καταλάβει τι πήγε λάθος και να αντιδράσει σωστά.

6.2.2 Αποφυγή διπλών εγγραφών (duplicate alarms)

Σε συστήματα που στέλνουν δεδομένα μέσω δικτύου, μπορεί να υπάρξουν retries και διπλοεγγραφές. Για να μειωθεί αυτό, μπορούν να εφαρμοστούν πρακτικές όπως:

- μοναδικό event_id που στέλνει ο node,
- ή απλή dedup λογική: "αν ίδιο event_type από τον ίδιο node μέσα σε 2 δευτερόλεπτα, αγνόησέ το".

Στο MVP, ακόμη και ένα απλό cooldown ανά node/event_type βοηθά σημαντικά να μην γεμίσει η βάση από "θόρυβο".

6.2.3 Offline ένδειξη κόμβων και last_seen

Το πεδίο last_seen επιτρέπει να φανεί αν ένας node είναι ενεργός. Αν δεν έχει στείλει alarm για μεγάλο διάστημα, μπορεί να εμφανίζεται ως "offline". Αυτό είναι στοιχείο αξιοπιστίας γιατί δεν αρκεί να περιμένουμε alarms — χρειάζεται να ξέρουμε ότι ο node "ζει".

6.3 Testing και αξιολόγηση

6.3.1 Στρατηγική testing

Για να θεωρηθεί σωστά ελεγμένο το σύστημα, ακολουθείται απλή αλλά ολοκληρωμένη στρατηγική:

- **Unit tests:** έλεγχος risk scoring συνάρτησης, validation συναρτήσεων, helper functions.

- **Integration tests:** έλεγχος API endpoints (login, ingestion, retrieval).
- **UI sanity tests:** βασικά σενάρια (login → create system → add node → βλέπω alarms).

6.3.2 Σενάρια δοκιμών (test cases)

Μερικά από αυτά είναι:

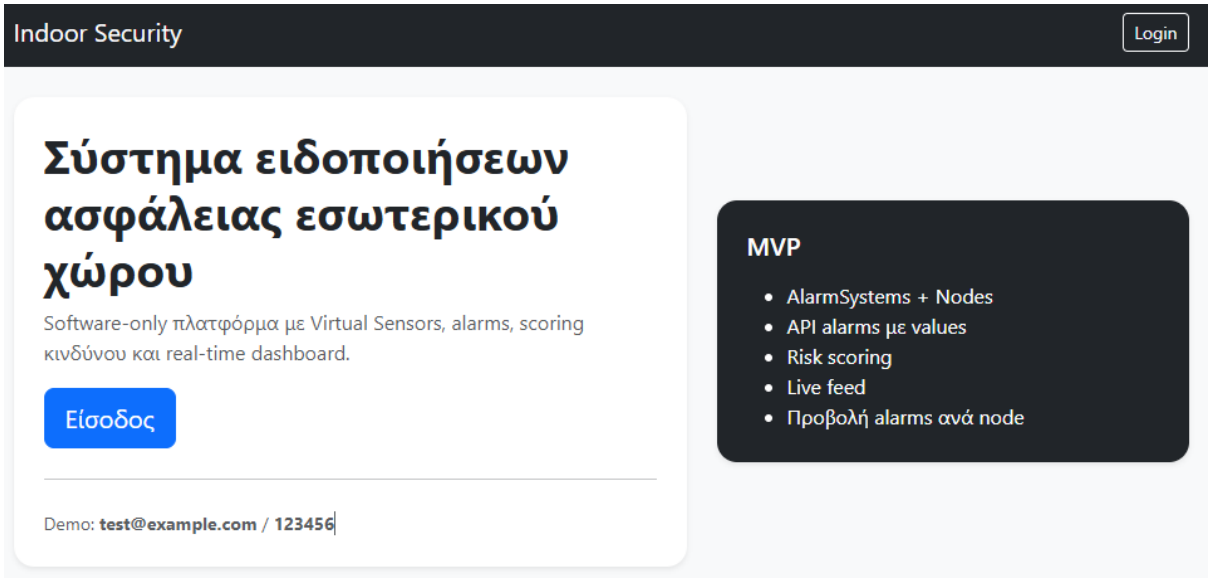
- TC1: Login με σωστά στοιχεία → επιτυχία.
- TC2: Login με λάθος password → απόρριψη.
- TC3: Node στέλνει alarm με σωστό token → εγγραφή στη βάση.
- TC4: Node στέλνει alarm με λάθος token → 401.
- TC5: Alarms retrieval με since_id → επιστρέφονται μόνο νέα.
- TC6: Risk scoring clamp → ποτέ <0 ή >100.

6.3.3 Performance testing με simulator

Ο simulator μπορεί να χρησιμοποιηθεί για μέτρηση απόδοσης. Για παράδειγμα:

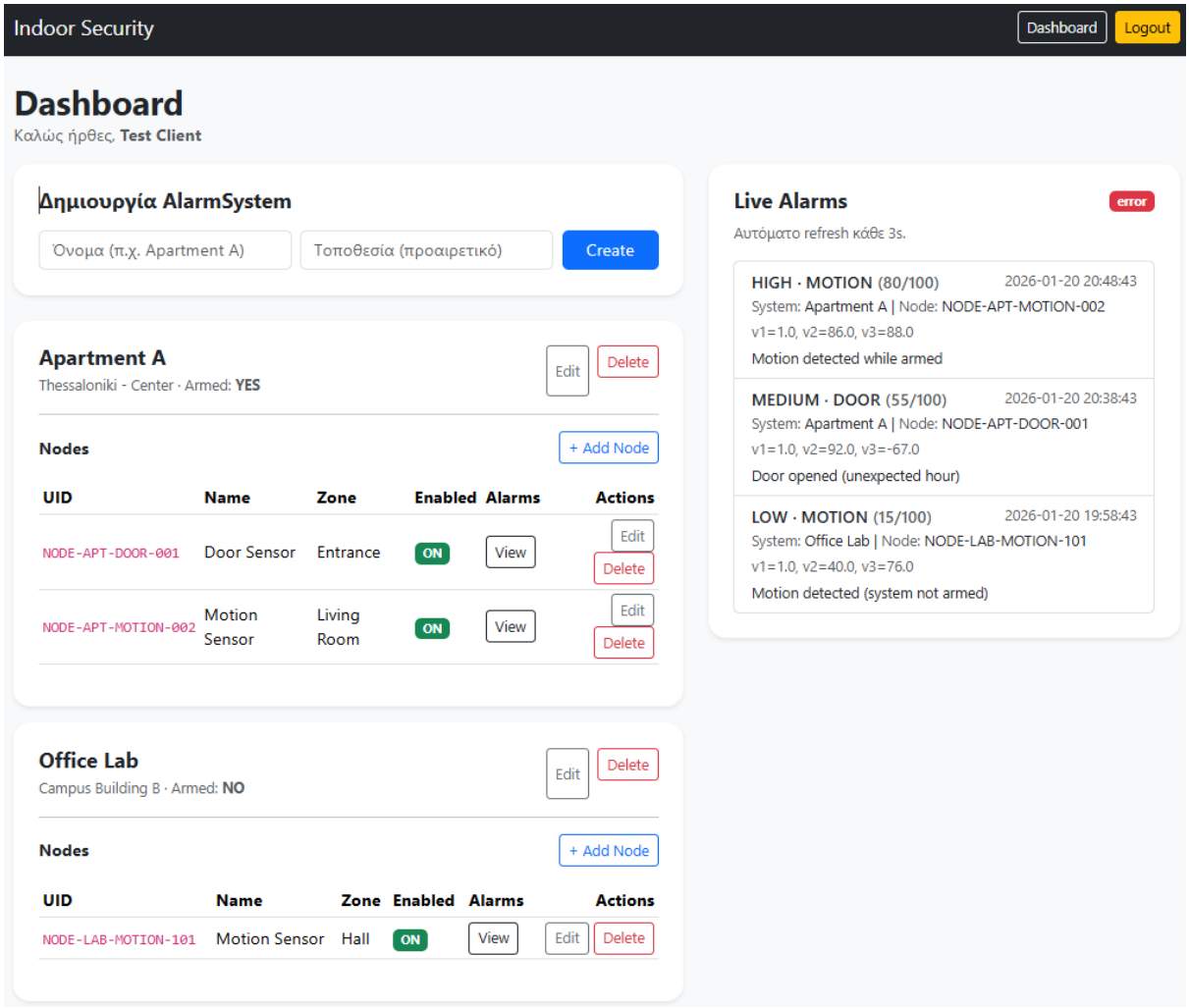
- Τρέχουμε ρυθμό 1 alarm/sec για 5 λεπτά και μετράμε latency.
- Αυξάνουμε στα 5 alarms/sec και βλέπουμε αν το dashboard παραμένει λειτουργικό.
- Μετράμε χρόνο εισαγωγής στη βάση.

6.4 Η πλατφόρμα



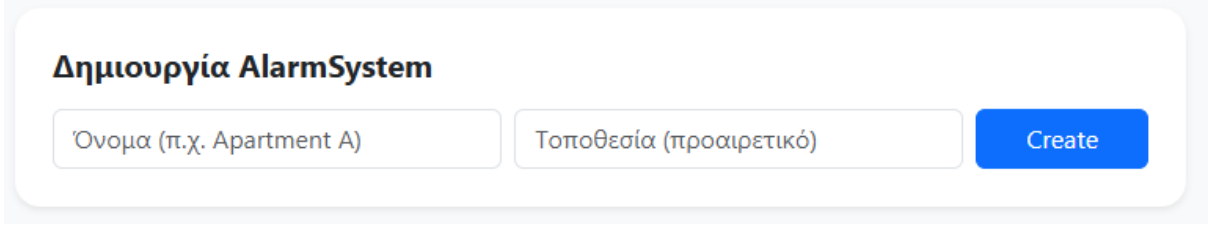
Εικόνα 6.1: Αρχική σελίδα πλατφόρμας και παρουσίαση MVP

Στην Εικόνα 6.1 παρουσιάζεται η welcome σελίδα της εφαρμογής “Indoor Security”, όπου εμφανίζεται ο τίτλος του έργου και μια συνοπτική κάρτα με τα βασικά στοιχεία του MVP (AlarmSystems, Nodes, API alarms με τιμές, risk scoring, live feed και προβολή alarms ανά node). Παράλληλα παρέχεται κουμπί εισόδου και demo credentials για γρήγορη δοκιμή.



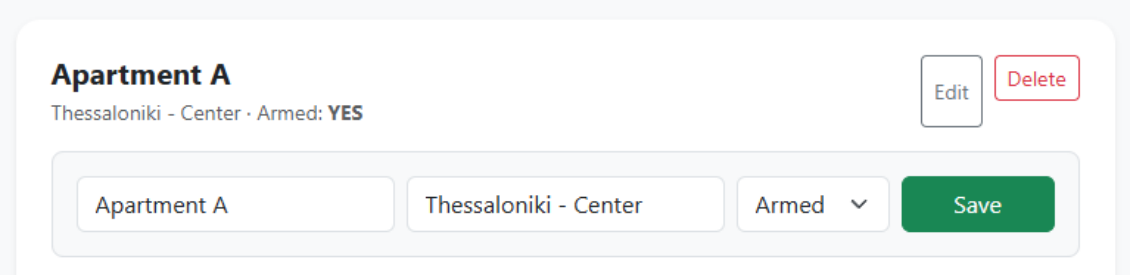
Εικόνα 6.2: Κεντρικό Dashboard χρήστη με διαχείριση συστημάτων και live alarms

Στην Εικόνα 6.2 φαίνεται το κεντρικό dashboard μετά το login, με δυνατότητα δημιουργίας νέου AlarmSystem, λίστα υπαρχόντων συστημάτων και των nodes τους (με ενέργειες προβολής/επεξεργασίας/διαγραφής). Στη δεξιά πλευρά εμφανίζεται η ενότητα “Live Alarms”, η οποία ενημερώνεται αυτόματα ανά λίγα δευτερόλεπτα και προβάλλει πρόσφατα συμβάντα με severity και risk score.



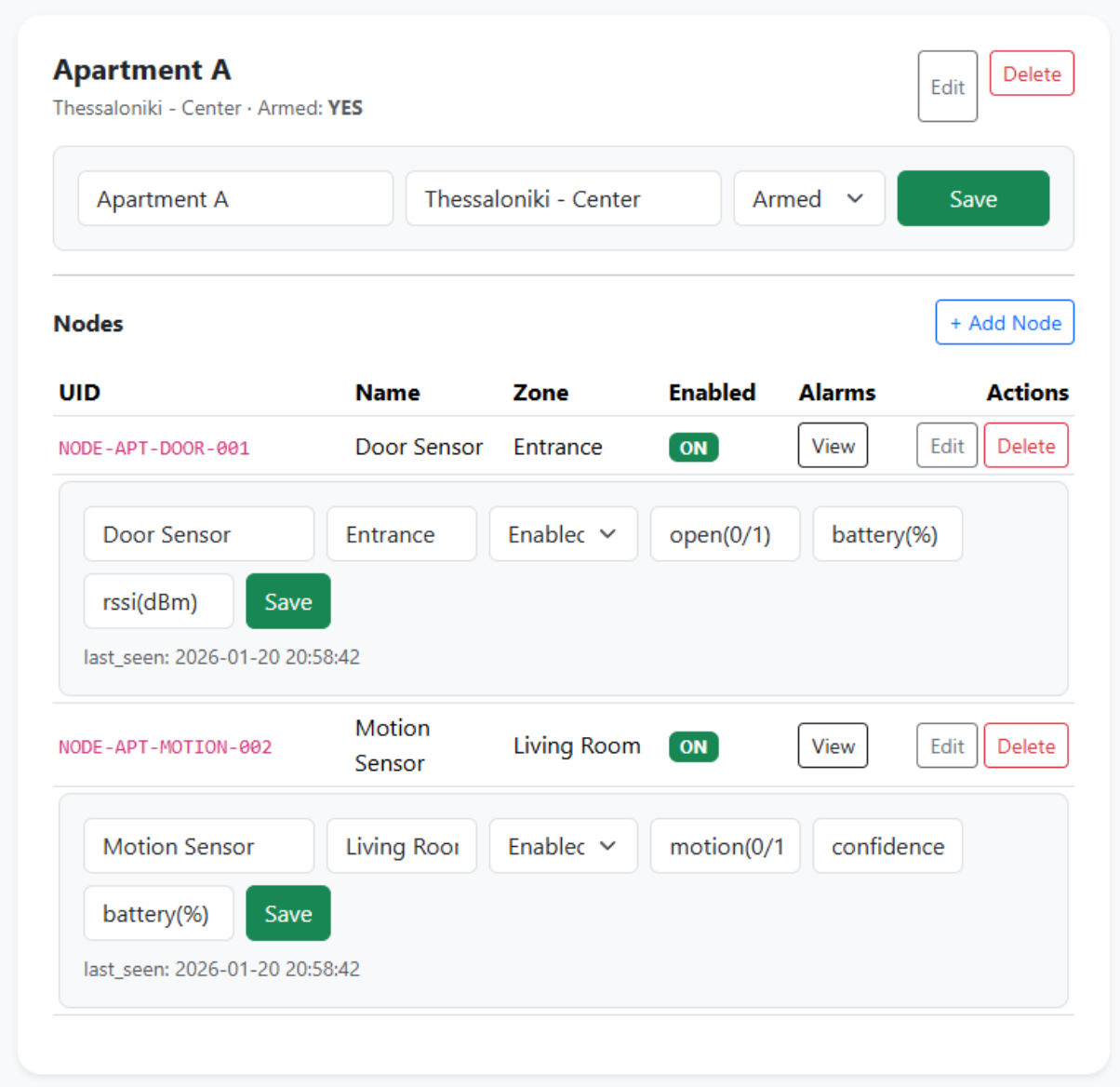
Εικόνα 6.3: Φόρμα δημιουργίας AlarmSystem

Στην Εικόνα 6.3 παρουσιάζεται η φόρμα δημιουργίας AlarmSystem, όπου ο χρήστης εισάγει το όνομα του συστήματος (π.χ. “Apartment A”) και προαιρετικά την τοποθεσία, ώστε να οργανώνει πολλαπλούς χώρους/εγκαταστάσεις μέσα στην ίδια πλατφόρμα.



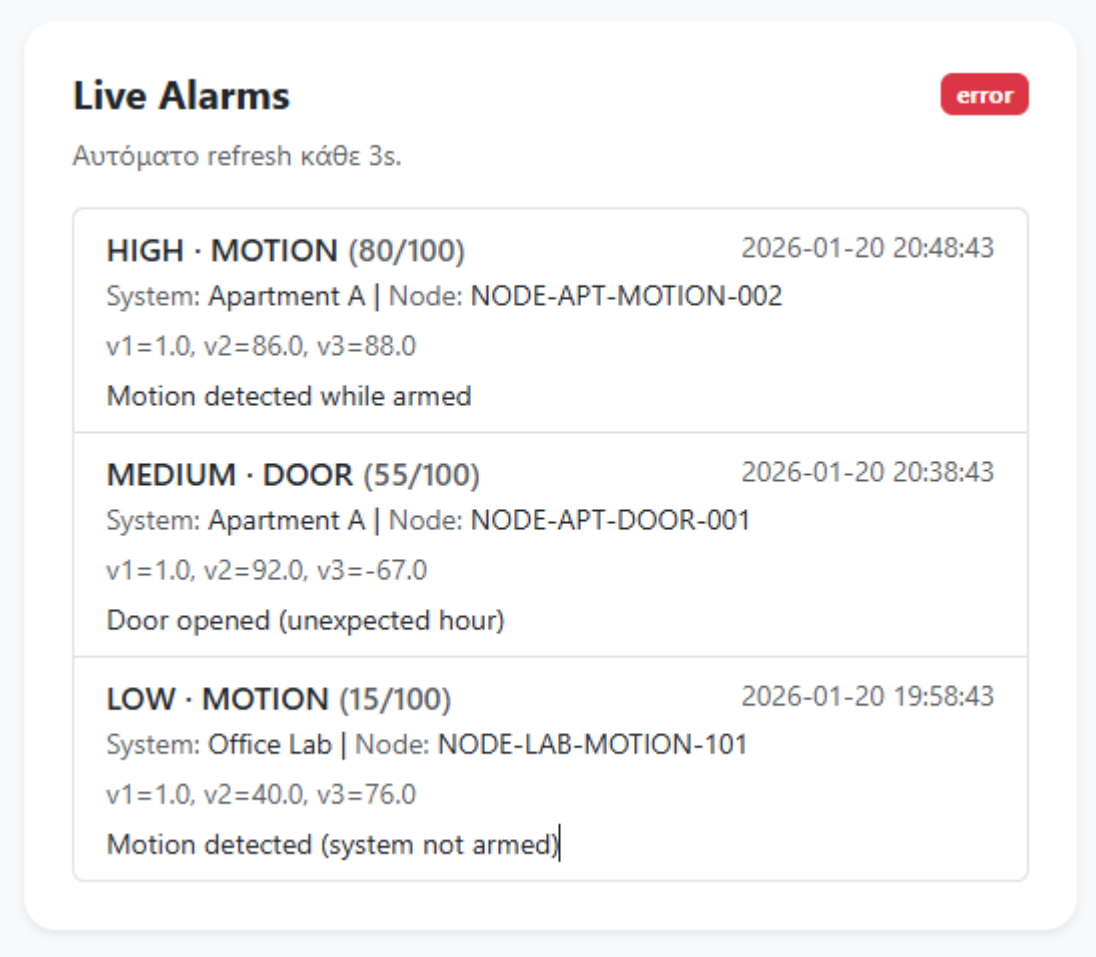
Εικόνα 6.4: Επεξεργασία AlarmSystem και κατάσταση οπλισμού

Η Εικόνα 6.4 απεικονίζει την επεξεργασία ενός AlarmSystem, όπου ο χρήστης μπορεί να τροποποιήσει όνομα και τοποθεσία, καθώς και να ορίσει την κατάσταση “armed/disarmed”. Η κατάσταση οπλισμού αξιοποιείται στη λογική αξιολόγησης συμβάντων (scoring) και επηρεάζει τη σοβαρότητα/προτεραιότητα των ειδοποιήσεων.



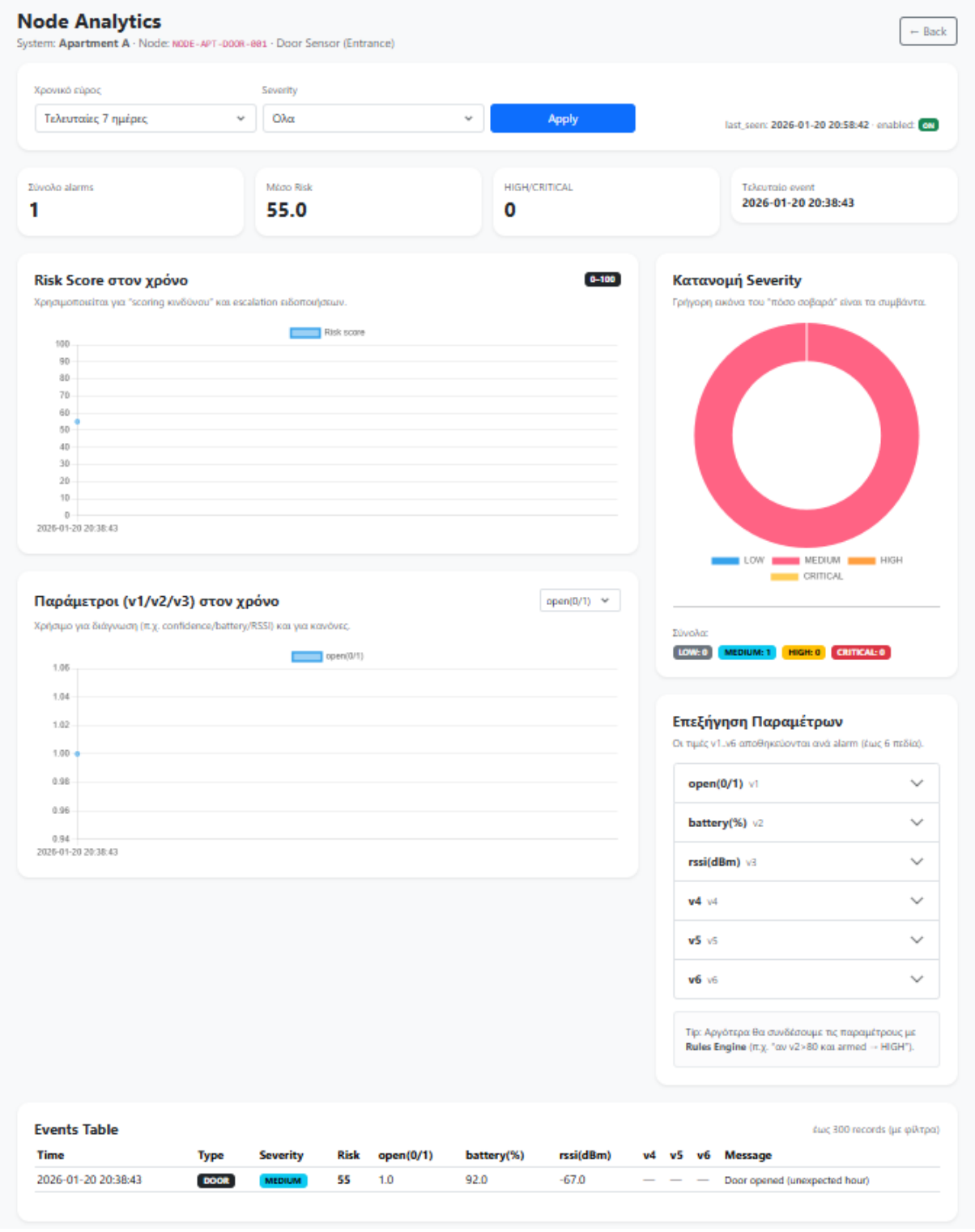
Εικόνα 6.5: Διαχείριση Nodes και ορισμός παραμέτρων (labels) v1–v6

Στην Εικόνα 6.5 παρουσιάζεται η ενότητα διαχείρισης κόμβων (Nodes) ενός AlarmSystem. Ο χρήστης μπορεί να ορίσει/επεξεργαστεί στοιχεία όπως όνομα, ζώνη (zone), ενεργοποίηση (enabled), καθώς και τα labels των παραμέτρων (π.χ. open(0/1), battery(%), rssi(dBm)), ώστε οι τιμές v1–v6 που αποθηκεύονται ανά alarm να αποκτούν συγκεκριμένη ερμηνεία. Εμφανίζεται επίσης η ένδειξη last_seen για παρακολούθηση της “τελευταίας επικοινωνίας” του node.



Εικόνα 6.6: Πίνακας “Live Alarms” με severity και risk score

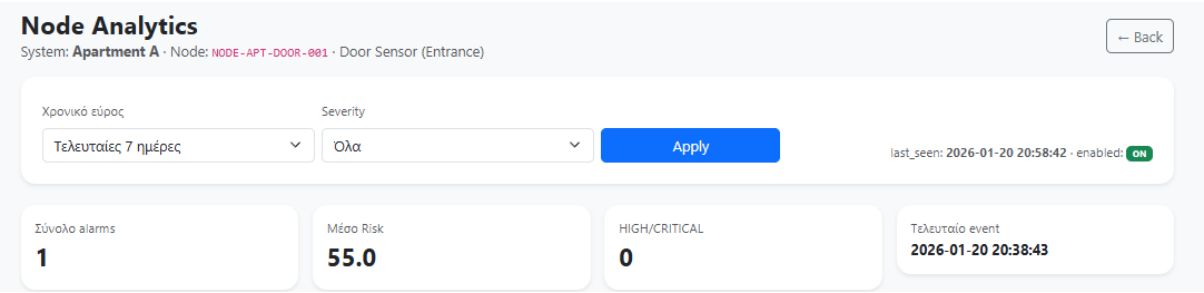
Στην Εικόνα 6.6 φαίνεται η καρτέλα “Live Alarms”, η οποία εμφανίζει σε πραγματικό χρόνο τα πιο πρόσφατα alarms. Κάθε εγγραφή συνοδεύεται από επίπεδο σοβαρότητας (LOW/MEDIUM/HIGH), risk score (π.χ. 80/100), χρονική σήμανση, πληροφορία συστήματος και node, καθώς και τις τιμές v1–v3 και ένα σύντομο μήνυμα περιγραφής του συμβάντος.



Εικόνα 6.7: Σελίδα Node Analytics με KPIs, γραφήματα και πίνακα συμβάντων

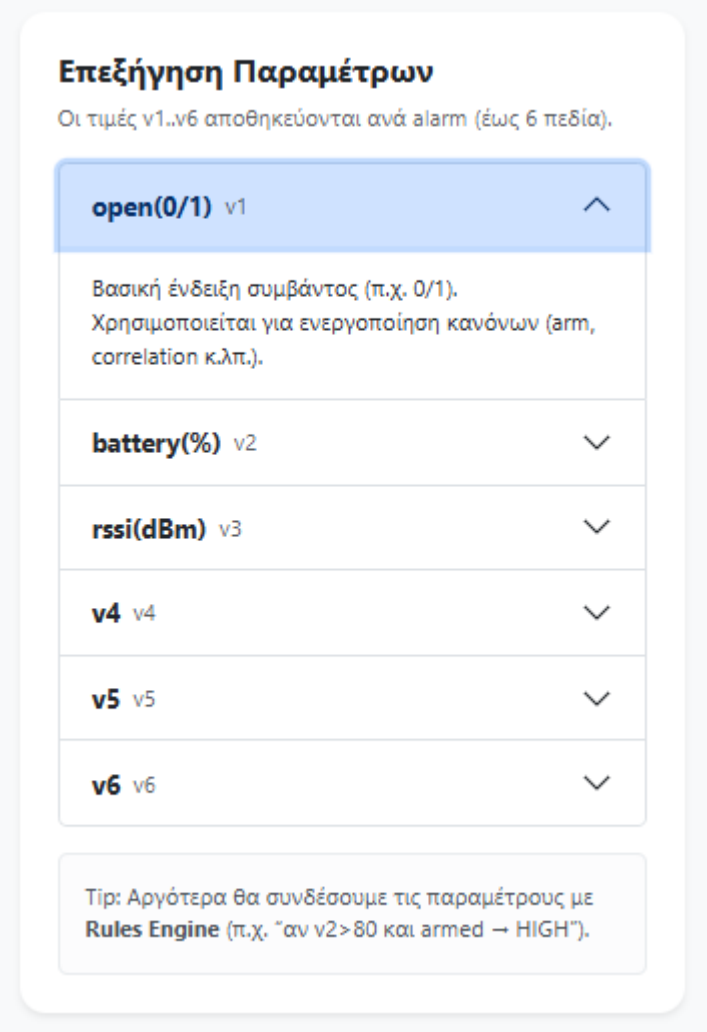
Στην Εικόνα 6.7 παρουσιάζεται η πλήρης σελίδα "Node Analytics" για έναν συγκεκριμένο κόμβο. Περιλαμβάνει φίλτρα (χρονικό εύρος και severity), συνοπτικούς δείκτες (σύνολο alarms, μέσο risk, πλήθος HIGH/CRITICAL, τελευταίο event), γραφήματα (risk score στον χρόνο, κατανομή severity,

εξέλιξη επιλεγμένης παραμέτρου), ενότητα επεξήγησης παραμέτρων και πίνακα συμβάντων (events table) με όλες τις καταγραφές.



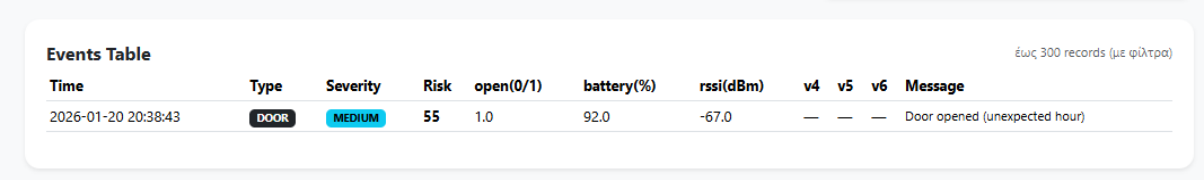
Εικόνα 6.8: Φίλτρα και βασικοί δείκτες κατάστασης κόμβου

Στην Εικόνα 6.8 φαίνεται το επάνω τμήμα της σελίδας analytics, όπου ο χρήστης επιλέγει χρονικό εύρος και severity για φιλτράρισμα. Παράλληλα προβάλλονται βασικοί δείκτες που συνοψίζουν τη συμπεριφορά του κόμβου καθώς και πληροφορίες κατάστασης όπως last_seen και enabled, για άμεση εικόνα λειτουργίας.



Εικόνα 6.9: Ενότητα “Επεξήγηση Παραμέτρων” (v1–v6) με αναπτυσσόμενες κάρτες

Στην Εικόνα 6.9 παρουσιάζεται η ενότητα επεξήγησης παραμέτρων, όπου κάθε πεδίο v1–v6 αντιστοιχίζεται σε label (π.χ. open(0/1), battery(%), rssi(dBm)) και συνοδεύεται από σύντομη περιγραφή χρήσης. Με αυτόν τον τρόπο, οι αριθμητικές τιμές των alarms γίνονται κατανοητές και μπορούν να αξιοποιηθούν σε κανόνες και scoring.



Εικόνα 6.10: Πίνακας συμβάντων (Events Table) ανά κόμβο

Η Εικόνα 6.10 απεικονίζει τον πίνακα συμβάντων (events table) που συγκεντρώνει τα alarms του node σε δομημένη μορφή. Εμφανίζονται βασικά πεδία όπως χρόνος, τύπος συμβάντος, severity, risk score, τιμές παραμέτρων (open, battery, rssi, v4–v6) και μήνυμα, επιτρέποντας γρήγορο έλεγχο και σύγκριση καταγραφών.

Κεφάλαιο 7ο: Συμπεράσματα και Μελλοντική Εργασία

7.1 Συνοπτική ανασκόπηση της υλοποίησης

Η πτυχιακή ανέπτυξε ένα Σύστημα Ειδοποιήσεων Ασφάλειας Εσωτερικού Χώρου με βασική ιδέα την ύπαρξη μιας κεντρικής πλατφόρμας που συλλέγει και εμφανίζει alarms. Το σύστημα υλοποιήθηκε με Python Flask, MySQL και Bootstrap και κάλυψε τις βασικές λειτουργίες ενός MVP:

- Δημιουργία και διαχείριση Alarm Systems ανά χρήστη.
- Προσθήκη και διαχείριση Nodes με μοναδικό αναγνωριστικό (node_uid).
- Λήψη alarms μέσω REST API από nodes ή από simulator.
- Υπολογισμός risk score για κάθε alarm και καταγραφή του στη βάση.
- Dashboard με προβολή πρόσφατων alarms και σχεδόν real-time ενημέρωση με polling.
- Σελίδα node analytics με πίνακα alarms, KPIs και βασικά γραφήματα.
- Virtual Sensors / Simulator για παραγωγή δεδομένων και σενάρια δοκιμών.

7.2 Επίτευξη στόχων και βασικά αποτελέσματα

Οι στόχοι που επιτεύχθηκαν:

1. **Κεντρική πλατφόρμα ειδοποιήσεων:** υλοποιήθηκε ένας core server που δέχεται alarms, τα επεξεργάζεται και τα εμφανίζει.
2. **Virtual Sensors αντί hardware:** η λειτουργία nodes και alarms προσομοιώθηκε με επιτυχία, επιτρέποντας δοκιμές χωρίς φυσική εγκατάσταση.
3. **Risk scoring και βάση για "έξυπνους κανόνες":** κάθε alarm αποκτά score, κάτι που βοηθά να ξεχωρίζουν τα σημαντικά συμβάντα.
4. **Real-time παρακολούθηση:** το polling πέτυχε τον στόχο του MVP, με γρήγορη ενημέρωση του dashboard.
5. **Ασφάλεια και αξιοπιστία:** εφαρμόστηκαν βασικές πρακτικές (hashing, token checks, validation, logging), σημαντικές για μια εφαρμογή που σχετίζεται με ασφάλεια.
6. **Testing και αξιολόγηση:** ο simulator επέτρεψε δοκιμές, ενώ προτάθηκαν συγκεκριμένα test cases και μετρήσεις απόδοσης.

Ακόμη και χωρίς hardware μπορεί να αναπτυχθεί και να αξιολογηθεί το σύστημα ειδοποιήσεων.

7.3 Περιορισμοί του MVP

Παρόλο που το MVP καλύπτει σημαντικό μέρος της λειτουργικότητας, υπάρχουν περιορισμοί:

- **Real-time με polling:** το polling είναι απλό και λειτουργικό, αλλά δεν είναι η πιο αποδοτική λύση σε πολύ μεγάλα φορτία. Σε πιο ώριμο σύστημα θα προτιμηθεί WebSockets ή push μηχανισμός.
- **Απλό risk scoring:** το scoring είναι heuristic και βασίζεται σε απλούς κανόνες/βάρη. Είναι επαρκές για MVP, αλλά μπορεί να βελτιωθεί με πιο σύνθετη λογική (correlation, χρονικά παράθυρα).
- **Ένας ρόλος χρήστη:** το σύστημα έχει μόνο client. Σε πραγματικές εφαρμογές υπάρχουν ρόλοι όπως admin, operator, security manager.
- **Virtual sensors:** αν και είναι πολύ χρήσιμο για δοκιμές, δεν καλύπτει πλήρως τις δυσκολίες ενός πραγματικού δικτύου αισθητήρων (αστάθεια σύνδεσης, μπαταρία, offline λειτουργία).
- **Περιορισμένη λειτουργία ειδοποιήσεων εκτός web UI:** το MVP δεν περιλαμβάνει push notifications σε κινητό, άρα η "άμεση ενημέρωση" προϋποθέτει ότι ο χρήστης παρακολουθεί το dashboard.

7.4 Μελλοντικές επεκτάσεις

Οι πιο ρεαλιστικές και χρήσιμες επεκτάσεις για την επόμενη έκδοση του συστήματος είναι οι παρακάτω:

Push Notifications σε κινητό

Η σημαντικότερη αναβάθμιση για ένα σύστημα ειδοποιήσεων είναι να μπορεί να ενημερώνει τον χρήστη άμεσα, ακόμα και όταν δεν είναι ανοικτό το web UI. Αυτό μπορεί να γίνει με:

- Firebase Cloud Messaging (FCM) για Android,
- αποθήκευση device tokens στη βάση,
- ενεργοποίηση ειδοποίησης όταν έρθει alarm με υψηλό risk score ή severity.

WebSockets για πραγματικό real-time dashboard

Αντί για polling, το dashboard μπορεί να ενημερώνεται με WebSockets, ώστε τα alarms να "εμφανίζονται" αμέσως χωρίς περιοδικά requests. Αυτό βελτιώνει την απόκριση και μειώνει το άσκοπο φορτίο.

Rules Engine και correlation

Το scoring μπορεί να εξελιχθεί σε πιο ολοκληρωμένο rules engine, με δυνατότητες όπως:

- χρονικά παράθυρα (π.χ. 3 alarms σε 10s),
- συσχέτιση DOOR + MOTION,
- escalation όταν αυξάνεται η συχνότητα alarms,
- cooldowns για αποφυγή spam.

Περισσότεροι ρόλοι και δικαιώματα

Μελλοντικά, μπορεί να προστεθεί:

- ρόλος admin για διαχείριση χρηστών,
- ρόλος operator για παρακολούθηση πολλών εγκαταστάσεων,
- διαφορετικά επίπεδα πρόσβασης (read-only, read-write).

Προηγμένη ανίχνευση ανωμαλιών

- anomaly detection (π.χ. ασυνήθιστη δραστηριότητα νύχτα),
- στατιστική ανάλυση patterns,
- ή ακόμα και ML μοντέλα

Είδαμε μια κεντρική πλατφόρμα ειδοποιήσεων ασφάλειας με έμφαση στο λογισμικό χωρίς να απαιτείται φυσικό hardware. Μέσα από τη χρήση virtual sensors, risk scoring, real-time dashboard και βασικές πρακτικές ασφάλειας/δοκιμών, δημιουργήθηκε ένα ολοκληρωμένο MVP που μπορεί να λειτουργήσει ως βάση αν θέλουμε να το επεκτείνουμε.

Το σημαντικότερο αποτέλεσμα είναι ότι το σύστημα δεν περιορίζεται στην καταγραφή συμβάντων, αλλά προσπαθεί να προσθέσει αξία στον χρήστη μέσω αξιολόγησης (risk score), καλύτερης παρουσίασης (analytics) και σωστής δομής (API + DB + UI).

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] https://en.wikipedia.org/wiki/Minimum_viable_product

[2] https://igss.schneider-electric.com/Files/Doc-Help/Webhelp/V14/Alm/Content/Alarm_Severity.htm

[3] https://support.activtrak.com/hc/en-us/articles/33013617024539-Alarm-Analysis-Risk-Level-Report

[4] https://www.manageengine.com/network-monitoring/alarm-correlation-rule.html

[5] https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/cool-down

[6] Fette, I., & Melnikov, A. (2011, December). The WebSocket Protocol (RFC 6455). Internet Engineering Task Force (IETF). https://datatracker.ietf.org/doc/rfc6455/

[7] MDN contributors. (2025, December 15). The WebSocket API (WebSockets). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[8] Kantor, I. (2022, December 12). Long polling. JavaScript.info. https://javascript.info/long-polling

[9] Google. (n.d.). Get started with Firebase Cloud Messaging in Android apps. Firebase Documentation. Retrieved January 21, 2026, from https://firebase.google.com/docs/cloud-messaging/android/get-started

[10] https://flask.palletsprojects.com/en/stable/

[11] https://www.mysql.com/

[12] https://www.chartjs.org/

ΠΑΡΑΡΤΗΜΑ Α

```
import os
import secrets
from functools import wraps
from datetime import datetime

from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify
import mysql.connector
from werkzeug.security import check_password_hash

app = Flask(__name__)
app.secret_key = os.environ.get("FLASK_SECRET_KEY", "dev_secret")

DB_CONFIG = {
    "host": os.environ.get("MYSQL_HOST", "127.0.0.1"),
    "user": os.environ.get("MYSQL_USER", "root"),
    "password": os.environ.get("MYSQL_PASSWORD", ""),
    "database": os.environ.get("MYSQL_DATABASE", "indoor_security"),
    "autocommit": True,
}

def db():
    return mysql.connector.connect(**DB_CONFIG)

def login_required(fn):
    @wraps(fn)
    def wrapper(*args, **kwargs):
        if "client_id" not in session:
            return redirect(url_for("login"))
        return fn(*args, **kwargs)
    return wrapper

def api_get_token():
    # 1) Authorization: Bearer <token>
    auth = request.headers.get("Authorization", "")
    if auth.lower().startswith("bearer "):
        return auth.split(" ", 1)[1].strip()
```

```
    # 2) Query param token=...
    t = (request.args.get("token") or "").strip()
    if t:
        return t

    # 3) JSON body token (optional)
    data = request.get_json(silent=True) or {}
    t2 = (data.get("token") or "").strip()
    return t2 or None

def api_auth_client():
    token = api_get_token()
    if not token:
        return None

    con = db()
    cur = con.cursor(dictionary=True)
    cur.execute("SELECT id, full_name FROM clients WHERE api_token=%s", (token,))
    client = cur.fetchone()
    cur.close()
    con.close()
    return client

@app.post("/api/login")
def api_login():
    data = request.get_json(silent=True) or {}
    email = (data.get("email") or "").strip().lower()
    password = data.get("password") or ""

    if not email or not password:
        return jsonify({"ok": False, "error": "email and password required"}), 400

    con = db()
    cur = con.cursor(dictionary=True)
    cur.execute("SELECT id, password_hash, full_name, api_token FROM clients WHERE email=%s", (email,))
    user = cur.fetchone()
```

```
    if not user or not check_password_hash(user["password_hash"], password):
        cur.close(); con.close()
        return jsonify({"ok": False, "error": "invalid credentials"}), 401

    # αν δεν έχει token, φτιάξε ένα
    token = user.get("api_token")
    if not token:
        token = secrets.token_urlsafe(24)
        cur2 = con.cursor()
        cur2.execute("UPDATE clients SET api_token=%s, api_token_created_at=%s WHERE id=%s",
                     (token, datetime.utcnow(), user["id"]))
        cur2.close()

    cur.close()
    con.close()

    return jsonify({
        "ok": True,
        "token": token,
        "client_name": user["full_name"]
    })

@app.get("/api/alarms/latest")
def api_alarms_latest_public():
    """
    Query:
      token=...
      since_id=0
      limit=20
    """
    client = api_auth_client()
    if not client:
        return jsonify({"ok": False, "error": "unauthorized"}), 401

    try:
        since_id = int(request.args.get("since_id", 0))
    except ValueError:
        since_id = 0
```

```
    try:
        limit = int(request.args.get("limit", 20))
    except ValueError:
        limit = 20
    limit = max(1, min(100, limit))

    con = db()
    cur = con.cursor(dictionary=True)
    cur.execute("""
        SELECT a.id, a.event_type, a.severity, a.message, a.risk_score,
               a.v1,a.v2,a.v3,a.v4,a.v5,a.v6,
               a.created_at,
               n.node_uid, n.name AS node_name,
               s.name AS system_name
        FROM alarms a
        LEFT JOIN nodes n ON n.id=a.node_id
        LEFT JOIN alarm_systems s ON s.id=a.system_id
        WHERE s.client_id=%s AND a.id > %s
        ORDER BY a.id ASC
        LIMIT %s
    """, (client["id"], since_id, limit))
    items = cur.fetchall()

    cur.close()
    con.close()

    # max_id για να ενημερώνει το app το last_id
    max_id = since_id
    for it in items:
        if it["id"] > max_id:
            max_id = it["id"]
        if isinstance(it["created_at"], datetime):
            it["created_at"] = it["created_at"].strftime("%Y-%m-%d %H:%M:%S")

    return jsonify({"ok": True, "items": items, "max_id": max_id})


def risk_score(event_type: str, severity: str, system_armed: bool, v1=None, v2=None) -> int:
```

```
    """
    MVP scoring (θα το κάνουμε rules engine μετά).
    Επιπλέον boost όταν:
    - armed + MOTION/DOOR
    - υψηλό "confidence" (v2) όταν υπάρχει
    """
    et = (event_type or "UNKNOWN").upper()
    sev = (severity or "LOW").upper()

    base = {"MOTION": 35, "DOOR": 30, "GLASS": 60, "PANIC": 80, "UNKNOWN": 20}.get(et, 20)
    mult = {"LOW": 0.6, "MEDIUM": 1.0, "HIGH": 1.4, "CRITICAL": 1.8}.get(sev, 1.0)

    armed_bonus = 15 if system_armed else 0

    conf_bonus = 0
    try:
        if v2 is not None:
            v2f = float(v2)
            if v2f >= 80:
                conf_bonus = 10
            elif v2f >= 60:
                conf_bonus = 5
    except Exception:
        pass

    score = int(base * mult + armed_bonus + conf_bonus)
    return max(0, min(100, score))


@app.route("/")
def welcome():
    return render_template("welcome.html")


@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        email = request.form.get("email", "").strip().lower()
        password = request.form.get("password", "")
```

```
        con = db()
        cur = con.cursor(dictionary=True)
        cur.execute("SELECT id, email, password_hash, full_name FROM clients WHERE email=%s", (email,))
        user = cur.fetchone()
        cur.close()
        con.close()

        if not user or not check_password_hash(user["password_hash"], password):
            flash("Λάθος email ή κωδικός.", "danger")
            return redirect(url_for("login"))

        session["client_id"] = user["id"]
        session["client_name"] = user["full_name"]
        flash("Επιτυχής είσοδος.", "success")
        return redirect(url_for("dashboard"))

    return render_template("login.html")

@app.route("/logout")
def logout():
    session.clear()
    flash("Έγινε αποσύνδεση.", "info")
    return redirect(url_for("welcome"))

@app.route("/dashboard")
@login_required
def dashboard():
    client_id = session["client_id"]

    con = db()
    cur = con.cursor(dictionary=True)

    cur.execute("""
        SELECT * FROM alarm_systems
        WHERE client_id=%s
        ORDER BY created_at DESC
    """, (client_id,))
    systems = cur.fetchall()
```

```
    system_ids = [s["id"] for s in systems] or [-1]
    cur.execute(f"""
        SELECT * FROM nodes
        WHERE system_id IN ({",".join(["%s"] * len(system_ids))})
        ORDER BY created_at DESC
    """, tuple(system_ids))
    nodes = cur.fetchall()

    cur.execute("""
        SELECT a.id, a.event_type, a.severity, a.message, a.risk_score, a.v1, a.v2, a.v3, a.v4, a.v5, a.v6, a.created_at,
               n.node_uid, n.name AS node_name,
               s.name AS system_name
        FROM alarms a
        LEFT JOIN nodes n ON n.id=a.node_id
        LEFT JOIN alarm_systems s ON s.id=a.system_id
        WHERE s.client_id=%s
        ORDER BY a.created_at DESC
        LIMIT 20
    """, (client_id,))
    alarms = cur.fetchall()

    cur.close()
    con.close()

    nodes_by_system = {}
    for n in nodes:
        nodes_by_system.setdefault(n["system_id"], []).append(n)

    return render_template(
        "dashboard.html",
        systems=systems,
        nodes_by_system=nodes_by_system,
        alarms=alarms
    )

# ----------------------------
# AlarmSystem CRUD
```

```
# ----------------------------
@app.post("/systems/create")
@login_required
def systems_create():
    name = request.form.get("name", "").strip()
    location = request.form.get("location", "").strip()
    if not name:
        flash("Δώσε όνομα για το AlarmSystem.", "warning")
        return redirect(url_for("dashboard"))

    con = db()
    cur = con.cursor()
    cur.execute(
        "INSERT INTO alarm_systems (client_id, name, location, is_armed) VALUES (%s,%s,%s,0)",
        (session["client_id"], name, location if location else None)
    )
    cur.close()
    con.close()

    flash("Δημιουργήθηκε νέο AlarmSystem.", "success")
    return redirect(url_for("dashboard"))

@app.post("/systems/<int:system_id>/update")
@login_required
def systems_update(system_id):
    name = request.form.get("name", "").strip()
    location = request.form.get("location", "").strip()
    is_armed = 1 if request.form.get("is_armed") == "1" else 0

    con = db()
    cur = con.cursor()
    cur.execute("""
        UPDATE alarm_systems
        SET name=%s, location=%s, is_armed=%s
        WHERE id=%s AND client_id=%s
    """, (name, location if location else None, is_armed, system_id, session["client_id"]))
    cur.close()
    con.close()
```

```
    flash("Ενημερώθηκε το AlarmSystem.", "success")
    return redirect(url_for("dashboard"))

@app.post("/systems/<int:system_id>/delete")
@login_required
def systems_delete(system_id):
    con = db()
    cur = con.cursor()

    cur.execute("""
        DELETE a FROM alarms a
        JOIN alarm_systems s ON s.id=a.system_id
        WHERE a.system_id=%s AND s.client_id=%s
    """, (system_id, session["client_id"]))

    cur.execute("""
        DELETE n FROM nodes n
        JOIN alarm_systems s ON s.id=n.system_id
        WHERE n.system_id=%s AND s.client_id=%s
    """, (system_id, session["client_id"]))

    cur.execute("""
        DELETE FROM alarm_systems
        WHERE id=%s AND client_id=%s
    """, (system_id, session["client_id"]))

    cur.close()
    con.close()

    flash("Διαγράφηκε το AlarmSystem (και οι κόμβοι/alarms του).", "info")
    return redirect(url_for("dashboard"))

# ----------------------------
# Nodes CRUD
# ----------------------------
@app.post("/systems/<int:system_id>/nodes/create")
@login_required
```

```
def nodes_create(system_id):
    node_uid = request.form.get("node_uid", "").strip()
    name = request.form.get("name", "").strip()
    zone = request.form.get("zone", "").strip()

    # optional labels
    v1_label = request.form.get("v1_label", "").strip()
    v2_label = request.form.get("v2_label", "").strip()
    v3_label = request.form.get("v3_label", "").strip()

    if not node_uid or not name:
        flash("Node UID και όνομα κόμβου είναι υποχρεωτικά.", "warning")
        return redirect(url_for("dashboard"))

    api_token = secrets.token_urlsafe(16)

    con = db()
    cur = con.cursor(dictionary=True)

    cur.execute("SELECT id FROM alarm_systems WHERE id=%s AND client_id=%s", (system_id, session["client_id"]))
    if not cur.fetchone():
        cur.close()
        con.close()
        flash("Μη έγκυρο AlarmSystem.", "danger")
        return redirect(url_for("dashboard"))

    try:
        cur2 = con.cursor()
        cur2.execute("""
            INSERT INTO nodes (system_id, node_uid, name, zone, enabled, api_token, v1_label, v2_label, v3_label)
            VALUES (%s,%s,%s,%s,1,%s,%s,%s,%s)
        """, (
            system_id, node_uid, name, zone if zone else None, api_token,
            v1_label if v1_label else None, v2_label if v2_label else None, v3_label if v3_label else None
        ))
        cur2.close()
        flash(f"Ο κόμβος προστέθηκε. Token (κράτησέ το): {api_token}", "success")
    except mysql.connector.IntegrityError:
```

```
            flash("Υπάρχει ήδη κόμβος με αυτό το UID (μοναδικό).", "danger")

        cur.close()
        con.close()
        return redirect(url_for("dashboard"))

@app.post("/nodes/<int:node_id>/update")
@login_required
def nodes_update(node_id):
    name = request.form.get("name", "").strip()
    zone = request.form.get("zone", "").strip()
    enabled = 1 if request.form.get("enabled") == "1" else 0

    v1_label = request.form.get("v1_label", "").strip()
    v2_label = request.form.get("v2_label", "").strip()
    v3_label = request.form.get("v3_label", "").strip()

    con = db()
    cur = con.cursor()
    cur.execute("""
        UPDATE nodes n
        JOIN alarm_systems s ON s.id=n.system_id
        SET n.name=%s, n.zone=%s, n.enabled=%s,
            n.v1_label=%s, n.v2_label=%s, n.v3_label=%s
        WHERE n.id=%s AND s.client_id=%s
    """, (
        name,
        zone if zone else None,
        enabled,
        v1_label if v1_label else None,
        v2_label if v2_label else None,
        v3_label if v3_label else None,
        node_id,
        session["client_id"]
    ))
    cur.close()
    con.close()
```

```
    flash("Ενημερώθηκε ο κόμβος.", "success")
    return redirect(url_for("dashboard"))

@app.post("/nodes/<int:node_id>/delete")
@login_required
def nodes_delete(node_id):
    con = db()
    cur = con.cursor()

    cur.execute("""
        DELETE a FROM alarms a
        JOIN nodes n ON n.id=a.node_id
        JOIN alarm_systems s ON s.id=n.system_id
        WHERE a.node_id=%s AND s.client_id=%s
    """, (node_id, session["client_id"]))

    cur.execute("""
        DELETE n FROM nodes n
        JOIN alarm_systems s ON s.id=n.system_id
        WHERE n.id=%s AND s.client_id=%s
    """, (node_id, session["client_id"]))

    cur.close()
    con.close()
    flash("Διαγράφηκε ο κόμβος.", "info")
    return redirect(url_for("dashboard"))

# ------------------------------
# Node alarms page (values per alarm)
# ------------------------------
@app.get("/nodes/<int:node_id>/alarms")
@login_required
def node_alarms(node_id):
    # range: 1d, 7d, 30d
    rng = (request.args.get("range") or "7d").strip()
    if rng not in ("1d", "7d", "30d"):
        rng = "7d"
```

```
    severity_filter = (request.args.get("sev") or "ALL").strip().upper()
    if severity_filter not in ("ALL", "LOW", "MEDIUM", "HIGH", "CRITICAL"):
        severity_filter = "ALL"

    interval_sql = {"1d": "1 DAY", "7d": "7 DAY", "30d": "30 DAY"}[rng]

    con = db()
    cur = con.cursor(dictionary=True)

    # node + system ownership
    cur.execute("""
        SELECT n.*, s.name AS system_name
        FROM nodes n
        JOIN alarm_systems s ON s.id=n.system_id
        WHERE n.id=%s AND s.client_id=%s
    """, (node_id, session["client_id"]))
    node = cur.fetchone()
    if not node:
        cur.close()
        con.close()
        flash("Μη έγκυρος κόμβος.", "danger")
        return redirect(url_for("dashboard"))

    # alarms list (filtered)
    where_sev = ""
    params = [node_id]
    if severity_filter != "ALL":
        where_sev = "AND severity=%s"
        params.append(severity_filter)

    cur.execute(f"""
        SELECT id, event_type, severity, message, risk_score,
               v1,v2,v3,v4,v5,v6, acknowledged, created_at
        FROM alarms
        WHERE node_id=%s
          AND created_at >= NOW() - INTERVAL {interval_sql}
          {where_sev}
        ORDER BY created_at DESC
```

```
    LIMIT 300
""", tuple(params))
alarms = cur.fetchall()

# KPIs (same filters)
cur.execute(f"""
    SELECT
      COUNT(*) AS total,
      AVG(risk_score) AS avg_risk,
      SUM(CASE WHEN severity IN ('HIGH','CRITICAL') THEN 1 ELSE 0 END) AS high_crit,
      MAX(created_at) AS last_time
    FROM alarms
    WHERE node_id=%s
      AND created_at >= NOW() - INTERVAL {interval_sql}
      {where_sev}
""", tuple(params))
kpi = cur.fetchone() or {"total": 0, "avg_risk": 0, "high_crit": 0, "last_time": None}

# severity distribution (donut)
cur.execute(f"""
    SELECT severity, COUNT(*) AS c
    FROM alarms
    WHERE node_id=%s
      AND created_at >= NOW() - INTERVAL {interval_sql}
      {where_sev}
    GROUP BY severity
""", tuple(params))
sev_rows = cur.fetchall()

sev_counts = {"LOW": 0, "MEDIUM": 0, "HIGH": 0, "CRITICAL": 0}
for r in sev_rows:
    s = (r["severity"] or "").upper()
    if s in sev_counts:
        sev_counts[s] = int(r["c"])

# time series for risk score (line) - take last 200 points
cur.execute(f"""
    SELECT created_at, risk_score, v1, v2, v3
```

```
    FROM alarms
    WHERE node_id=%s
      AND created_at >= NOW() - INTERVAL {interval_sql}
      {where_sev}
    ORDER BY created_at ASC
    LIMIT 200
""", tuple(params))
series = cur.fetchall()

cur.close()
con.close()

# parameter explanations (simple, editable later)
# If label is empty, fallback. You can enrich these texts later.
param_info = [
    {
        "key": "v1",
        "label": node.get("v1_label") or "v1",
        "desc": "Βασική ένδειξη συμβάντος (π.χ. 0/1). Χρησιμοποιείται για ενεργοποίηση κανόνων (arm, correlation κ.λπ.)."
    },
    {
        "key": "v2",
        "label": node.get("v2_label") or "v2",
        "desc": "Δευτερεύουσα μέτρηση (π.χ. confidence %, battery %, temperature). Μπορεί να επηρεάζει το risk scoring."
    },
    {
        "key": "v3",
        "label": node.get("v3_label") or "v3",
        "desc": "Τρίτη μέτρηση (π.χ. battery %, RSSI). Χρήσιμη για διάγνωση/αξιοπιστία."
    },
    {"key": "v4", "label": node.get("v4_label") or "v4", "desc": "Προαιρετική επιπλέον παράμετρος."},
    {"key": "v5", "label": node.get("v5_label") or "v5", "desc": "Προαιρετική επιπλέον παράμετρος."},
    {"key": "v6", "label": node.get("v6_label") or "v6", "desc": "Προαιρετική επιπλέον παράμετρος."},
]

# format datetime strings for JS
def fmt(dt):
    if isinstance(dt, datetime):
```

```
            return dt.strftime("%Y-%m-%d %H:%M:%S")
        return str(dt) if dt else None

    chart_labels = [fmt(r["created_at"]) for r in series]
    chart_risk = [int(r["risk_score"] or 0) for r in series]
    chart_v1 = [r["v1"] for r in series]
    chart_v2 = [r["v2"] for r in series]
    chart_v3 = [r["v3"] for r in series]

    return render_template(
        "node_alarms.html",
        node=node,
        alarms=alarms,
        rng=rng,
        severity_filter=severity_filter,
        kpi=kpi,
        sev_counts=sev_counts,
        param_info=param_info,
        chart_labels=chart_labels,
        chart_risk=chart_risk,
        chart_v1=chart_v1,
        chart_v2=chart_v2,
        chart_v3=chart_v3,
    )

# ------------------------------
# API: node sends alarm with up to 6 values
# ------------------------------
@app.post("/api/nodes/<node_uid>/alarm")
def api_node_alarm(node_uid):
    """
    JSON:
    {
      "token": "...",
      "event_type": "MOTION",
      "severity": "HIGH",
      "message": "motion detected",
      "values": [v1, v2, v3, v4, v5, v6]
```

```
}
"""
data = request.get_json(silent=True) or {}
token = (data.get("token") or "").strip()
event_type = (data.get("event_type") or "UNKNOWN").strip().upper()
severity = (data.get("severity") or "LOW").strip().upper()
message = (data.get("message") or "").strip()
values = data.get("values") or []

def pick(i):
    try:
        return float(values[i]) if i < len(values) and values[i] is not None else None
    except Exception:
        return None

v1 = pick(0); v2 = pick(1); v3 = pick(2); v4 = pick(3); v5 = pick(4); v6 = pick(5)

if not token:
    return jsonify({"ok": False, "error": "token required"}), 400

con = db()
cur = con.cursor(dictionary=True)
cur.execute("""
    SELECT n.id AS node_id, n.system_id, n.enabled, s.is_armed
    FROM nodes n
    JOIN alarm_systems s ON s.id=n.system_id
    WHERE n.node_uid=%s AND n.api_token=%s
""", (node_uid, token))
node = cur.fetchone()

if not node:
    cur.close(); con.close()
    return jsonify({"ok": False, "error": "invalid node_uid or token"}), 401

if int(node["enabled"]) != 1:
    cur.close(); con.close()
    return jsonify({"ok": False, "error": "node disabled"}), 403
```

```
    score = risk_score(event_type, severity, bool(node["is_armed"]), v1=v1, v2=v2)

    cur2 = con.cursor()
    cur2.execute("""
       INSERT INTO alarms (system_id, node_id, event_type, severity, message, risk_score, v1,v2,v3,v4,v5,v6, acknowledged)
        VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,0)
    """, (
        node["system_id"], node["node_id"], event_type, severity,
        message if message else None, score,
        v1,v2,v3,v4,v5,v6
    ))

    cur2.execute("UPDATE nodes SET last_seen=%s WHERE id=%s", (datetime.utcnow(), node["node_id"]))
    cur2.close()

    cur.close()
    con.close()
    return jsonify({"ok": True, "risk_score": score})

# ------------------------------
# API: latest alarms for logged-in client (polling)
# ------------------------------
@app.get("/api/alarms/latest")
@login_required
def api_alarms_latest():
    limit = int(request.args.get("limit", 20))
    limit = max(5, min(100, limit))

    con = db()
    cur = con.cursor(dictionary=True)
    cur.execute("""
        SELECT a.id, a.event_type, a.severity, a.message, a.risk_score,
               a.v1,a.v2,a.v3,a.v4,a.v5,a.v6,
               a.acknowledged, a.created_at,
               n.node_uid, n.name AS node_name,
               s.name AS system_name
        FROM alarms a
        LEFT JOIN nodes n ON n.id=a.node_id
```

```
        LEFT JOIN alarm_systems s ON s.id=a.system_id
        WHERE s.client_id=%s
        ORDER BY a.created_at DESC
        LIMIT %s
    """, (session["client_id"], limit))
    rows = cur.fetchall()
    cur.close()
    con.close()

    for r in rows:
        if isinstance(r["created_at"], datetime):
            r["created_at"] = r["created_at"].strftime("%Y-%m-%d %H:%M:%S")

    return jsonify({"ok": True, "items": rows})

if __name__ == "__main__":
    app.run(debug=True)
```