ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Τίτλος Πτυχιακής Εργασίας	Διαδικτυακή πλατφόρμα αναφοράς προβλημάτων της κοινότητας υλοποιημένη με Node.js και PostgreSQL Community issue reporting web platform implemented with Node.js and PostgreSQL
Ονοματεπώνυμο Φοιτητή	Μιχαήλ Καψάλης Μοσχοβάκης
Πατρώνυμο	Γεώργιος
Αριθμός Μητρώου	Π21056
Επιβλέπων	Ευθύμιος Αλέπης, Καθηγητής

Ημερομηνία παράδοσης: Σεπτέμβριος 2026

Copyright ©

Μιχαήλ Καψάλης Μοσχοβάκης

Σεπτέμβριος 2026

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα Καθηγητή μου κ. Ευθύμιο Αλέπη για την πολύτιμη καθοδήγηση και τη στήριξη που μου προσέφερε καθ' όλη τη διάρκεια της εκπόνησης της πτυχιακής μου εργασίας.

Θα ήθελα επίσης να ευχαριστήσω την συνεπιβλέπουσα Δρ. Αριστέα Κοντογιάννη για τα σχόλια, τις παρατηρήσεις και τις προτάσεις προκειμένου η εργασία μου να λάβει τη σημερινή της μορφή.

Τέλος, ευχαριστώ θερμά την οικογένεια και τους φίλους μου για την αμέριστη υποστήριξη και την κατανόηση που έδειξαν σε όλη αυτή την περίοδο συγγραφής της πτυχιακής.

Σας ευχαριστώ όλους και όλες από καρδιάς!

Περίληψη

Η παρούσα πτυχιακή εργασία παρουσιάζει την ανάπτυξη της «County Reports», μιας διαδικτυακής πλατφόρμας με στόχο την ενεργή συμμετοχή των πολιτών στη βελτίωση της κοινότητάς τους, μέσω της αναφοράς, παρακολούθησης και επίλυσης τοπικών προβλημάτων (π.χ. φθαρμένα οδοστρώματα, προβλήματα αποχέτευσης, βλάβες στον ηλεκτροφωτισμό). Η αρχιτεκτονική του συστήματος ακολουθεί το πρότυπο Model-View-Controller (MVC)·στο frontend και αξιοποιούνται τα EJS templates σε συνδυασμό με το Bootstrap για τη δημιουργία μιας καθαρής και φιλικής προς τον χρήστη διεπαφής. Από την άλλη πλευρά, στο backend χρησιμοποιούνται τα Node.js και Express.js για την εξυπηρέτηση των αιτημάτων και την PostgreSQL για την αποθήκευση των δεδομένων. Οι βασικές λειτουργίες περιλαμβάνουν την εγγραφή/είσοδο χρηστών, την αυθεντικοποίηση με ρόλους (απλός χρήστης/user, διαχειριστής/admin), την υποβολή αναφορών με συνημμένη εικόνα και γεωγραφική τοποθεσία, ένα σύστημα ψηφοφορίας για την ιεράρχηση των προβλημάτων, καθώς και την αυτόματη εκτίμηση του επιπέδου επικινδυνότητας κάθε αναφοράς μέσω ενός τοπικού μοντέλου Τεχνητής Νοημοσύνης (Ollama). Ιδιαίτερη έμφαση δόθηκε στην εργονομία, στην ασφάλεια (κρυπτογράφηση κωδικών, έλεγχος πρόσβασης βάσει ρόλων) και στη συντηρησιμότητα του κώδικα μέσω της αυστηρής τήρησης του προτύπου MVC.

Λέξεις-κλειδιά: αναφορά προβλημάτων, Express.js, PostgreSQL, Passport.js, Ollama.

Abstract

This thesis presents the development of “County Reports”, a web platform aimed at actively engaging citizens in improving their community through reporting, tracking, and resolving of local issues (such as damaged roads, drainage problems, faulty street lighting). The system architecture follows the Model-View-Controller (MVC) pattern; the frontend is built with EJS templates combined with Bootstrap to provide a clean and user-friendly interface, while the backend uses Node.js and Express.js to handle requests and PostgreSQL for persistent data storage. Core features include user registration/login, role-based authentication (user, admin), submission of reports with an attached image and geographic location, a voting system for prioritizing issues, as well as automatic estimation of each report's danger level through a local Artificial Intelligence model (Ollama). Particular emphasis was placed on ergonomics, security (password hashing, role-based access control), and code maintainability through strict adherence to the MVC pattern. The work contributes an extensible implementation model for civic-tech applications focused on the local community.

Key Words: issue reporting, Express.js, PostgreSQL, Passport.js, Ollama.

Πίνακας Περιεχομένων

Κατάλογος Εικόνων

Κατάλογος Διαγραμμάτων

Εισαγωγή

Η ποιότητα ζωής σε μια σύγχρονη πόλη εξαρτάται σε μεγάλο βαθμό από την κατάσταση των υποδομών της και από την ταχύτητα με την οποία τα τοπικά προβλήματα εντοπίζονται και επιλύονται (Zanella κ.συν., 2014). Καθημερινά, οι πολίτες έρχονται αντιμέτωποι με ζητήματα όπως λακκούβες στους δρόμους, βλάβες στον δημοτικό φωτισμό, φραγμένα φρεάτια ή πεσμένα δέντρα, τα οποία όμως συχνά παραμένουν για μεγάλο χρονικό διάστημα άγνωστα στις αρμόδιες αρχές. Αυτό συμβαίνει γιατί ο παραδοσιακός τρόπος επισήμανσης αυτών των ζητημάτων, που βασίζεται σε τηλεφωνικές κλήσεις ή φυσική παρουσία σε δημοτικές υπηρεσίες, συχνά αποδεικνύεται χρονοβόρος, αδιαφανής και άκρως αποθαρρυντικός για τον μέσο πολίτη.

Παράλληλα, οι αρμόδιες υπηρεσίες δυσκολεύονται να ιεραρχήσουν τα προβλήματα με βάση τη σοβαρότητα και τον αντίκτυπό τους στην κοινότητα, καθώς δεν διαθέτουν ένα ενιαίο, οργανωμένο κανάλι συγκέντρωσης των αναφορών. Το αποτέλεσμα είναι μια αναποτελεσματική κατανομή πόρων, όπου ζητήματα άμεσης προτεραιότητας και επικινδυνότητας ενδέχεται να αντιμετωπίζονται με τον ίδιο τρόπο όπως απλά προβλήματα αισθητικής. Σχετικά με αυτό, η διεθνής βιβλιογραφία τονίζει ότι η έλλειψη δομημένων καναλιών συμμετοχής των πολιτών οδηγεί σε μειωμένη αποτελεσματικότητα και περιορισμένη διαφάνεια στη λήψη αποφάσεων (Zissis κ.συν., 2009).

Από την άλλη πλευρά, αναδεικνύεται η ανάγκη για την εύρεση μιας σύγχρονης λύσης με τη δημιουργία μιας ψηφιακής πλατφόρμας συμμετοχικής διακυβέρνησης, η οποία να συνδέει άμεσα τους πολίτες με τις αρμόδιες αρχές (Kontogianni & Alepis, 2026), να καθιστά διαφανή την πορεία κάθε αναφοράς και να αξιοποιεί τη συλλογική γνώση της κοινότητας για την ιεράρχηση των προβλημάτων. Αυτή μάλιστα η προσέγγιση ενισχύεται καθώς τέτοια συστήματα ενισχύουν άμεσα τις δημοκρατικές διαδικασίες, αυξάνοντας παράλληλα την αποτελεσματικότητα της τοπικής αυτοδιοίκησης (Tsihrintzis κ.συν., 2026· Αναστασόπουλος, 2022· Παπαγιάννης, 2026).

Κύριος στόχος της παρούσας πτυχιακής εργασίας είναι ο σχεδιασμός και η ολοκληρωμένη υλοποίηση της διαδικτυακής πλατφόρμας «County Reports». Πρόκειται για μια εφαρμογή που επιτρέπει στους πολίτες να υποβάλλουν αναφορές προβλημάτων εμπλουτισμένες με φωτογραφικό υλικό και ακριβή γεωγραφική τοποθεσία, να ιεραρχούν τη σπουδαιότητα των αναφορών άλλων χρηστών, ώστε να αναδεικνύονται τα σημαντικότερα ζητήματα και να παρακολουθούν την πρόοδο της επίλυσής τους. Επιπλέον, η πλατφόρμα ενσωματώνει ένα τοπικό μοντέλο Τεχνητής Νοημοσύνης για την αυτόματη εκτίμηση του επιπέδου επικινδυνότητας κάθε αναφοράς (Kontogianni κ.συν., 2024), υποβοηθώντας τους διαχειριστές στην ιεράρχηση και προτεραιοποίηση των ενεργειών τους (Chrysafiadi κ.συν., 2025).

Η τεχνική προσέγγιση βασίζεται στο περιβάλλον εκτέλεσης Node.js με το πλαίσιο Express.js στο backend και τη μηχανή προτύπων EJS για την παραγωγή των σελίδων στην πλευρά του διακομιστή (server-side rendering). Για την αποθήκευση των δεδομένων επιλέχθηκε η σχεσιακή βάση δεδομένων PostgreSQL και ο σχεδιασμός της ακολουθεί αυστηρά το πρότυπο Model-View-Controller (MVC), δίνοντας έμφαση στον διαχωρισμό των ευθυνών, στη συντηρησιμότητα του κώδικα και στη δυνατότητα μελλοντικών επεκτάσεων.

Όλες οι παραπάνω λειτουργίες πλαισιώνονται από μηχανισμούς αυθεντικοποίησης και ελέγχου πρόσβασης βάσει ρόλων (απλός χρήστης/user, διαχειριστής/admin), καθώς και τη δυνατότητα αναστολής λογαριασμών, διασφαλίζοντας την ορθή ροή εργασιών και ένα ασφαλές περιβάλλον αλληλεπίδρασης.

Λαμβάνοντας αυτά υπόψη οι στόχοι που θέτει η παρούσα εργασία καταγράφονται ως εξής:

- Ανάπτυξη ενός πλήρως λειτουργικού πρωτοτύπου που να καλύπτει επιτυχώς τον πυρήνα των τεθειμένων απαιτήσεων.
- Τεκμηρίωση όλων των σχεδιαστικών επιλογών και της προτεινόμενης αρχιτεκτονικής MVC του συστήματος.
- Ανάδειξη της προστιθέμενης αξίας που προσφέρει η ενσωμάτωση ενός τοπικού μοντέλου Τεχνητής Νοημοσύνης σε μια εφαρμογή civic-tech.

Διαδικτυακή Πλατφόρμα Αναφοράς Προβλημάτων της Κοινότητας υλοποιημένη με Node.js και PostgreSQL
Community Issue Reporting Web Platform implemented with Node.js and PostgreSQL

- Ανάδειξη βέλτιστων πρακτικών για τον σχεδιασμό συμμετοχικών εφαρμογών με ξεκάθαρο θεματικό προσανατολισμό.

Στο πλαίσιο υλοποίησης αυτών των στόχων η παρούσα εργασία έχει την ακόλουθη διάρθρωση:

- Στο κεφάλαιο 2 γίνεται μια σύντομη αναφορά σε υπάρχουσες εφαρμογές αναφοράς προβλημάτων και αναλύονται οι τεχνολογίες που χρησιμοποιήθηκαν στην προτεινόμενη λύση.
- Στο κεφάλαιο 3 αναλύονται οι απαιτήσεις του συστήματος της προτεινόμενης λύσης και εξηγείται ο σχεδιασμός του.
- Στο 4ο κεφάλαιο παρουσιάζεται αναλυτικά η υλοποίηση του συστήματος.
- Στο κεφάλαιο 5 γίνεται η παρουσίαση και η αξιολόγηση της εφαρμογής.
- Τέλος, το 6ο κεφάλαιο συνοψίζει τα συμπεράσματα της εργασίας και καταθέτει προτάσεις για νέες δυνατότητες και περαιτέρω ανάπτυξη.

2. Σχετικές Εφαρμογές και Τεχνολογικό Υπόβαθρο της Προτεινόμενης Λύσης

Στο κεφάλαιο που ακολουθεί παρουσιάζονται ήδη υπάρχουσες τεχνολογικές λύσεις αναφοράς προβλημάτων καθώς και τα πλεονεκτήματα ή τα μειονεκτήματα κάθε μίας απ' αυτές. Ακολούθως, παρουσιάζεται το απαιτούμενο τεχνολογικό υπόβαθρο για την υλοποίηση της προτεινόμενης λύσης/εφαρμογής.

2.1 Σχετικές εφαρμογές αναφοράς προβλημάτων

Στην σημερινή εποχή υπάρχουν αρκετές πλατφόρμες που επιχειρούν να γεφυρώσουν την επικοινωνία μεταξύ των πολιτών και των τοπικών αρχών, οι οποίες μπορούν να ομαδοποιηθούν σε τρεις βασικές κατηγορίες.

Στην πρώτη κατηγορία ανήκουν οι διεθνείς πλατφόρμες αναφοράς, όπως το FixMyStreet και το SeeClickFix, οι οποίες επιτρέπουν στους πολίτες να αναφέρουν προβλήματα σε δημόσιους χώρους. Αυτές, παρόλο που αποτελούν ώριμες λύσεις, συχνά είναι προσαρμοσμένες σε συγκεκριμένες χώρες ή δήμους και δεν προσφέρονται εύκολα για παραμετροποίηση στο πλαίσιο μιας εκπαιδευτικής ή τοπικής υλοποίησης.

Στην δεύτερη κατηγορία ανήκουν τα γενικά κανάλια επικοινωνίας, όπως οι σελίδες κοινωνικής δικτύωσης των δήμων ή οι απλές φόρμες ηλεκτρονικού ταχυδρομείου. Αν και ευρέως διαθέσιμα, υστερούν σημαντικά στην οργάνωση της πληροφορίας, καθώς οι αναφορές κάποιες φορές χάνονται μέσα στον όγκο των δημοσιεύσεων, δεν υπάρχει παρακολούθηση της κατάστασης επίλυσης, απουσιάζει η δυνατότητα ιεράρχησης, κ.ά.

Τέλος, υπάρχουν οι εξειδικευμένες εφαρμογές δήμων, οι οποίες όμως είναι συχνά κλειστού κώδικα, εμπορικές και χωρίς δυνατότητα αξιοποίησης σύγχρονων τεχνολογιών όπως η Τεχνητή Νοημοσύνη για την αυτόματη ιεράρχηση.

Σε αντίθεση με τις παραπάνω προσεγγίσεις, το County Reports σχεδιάστηκε ως μια ανοικτή, επεκτάσιμη και εύκολα παραμετροποιήσιμη πλατφόρμα, η οποία συνδυάζει την υποβολή γεωαναφερόμενων αναφορών, τη συμμετοχική ιεράρχηση μέσω ψηφοφορίας και την αυτόματη εκτίμηση επικινδυνότητας μέσω τοπικού μοντέλου Τεχνητής Νοημοσύνης (Phillips-Wren & Virvou, 2025), καλύπτοντας έτσι τα κενά των υφιστάμενων λύσεων.

2.2 Τεχνολογικό υπόβαθρο της προτεινόμενης λύσης

Για την υλοποίηση της πλατφόρμας επιλέχθηκε μια αρχιτεκτονική βασισμένη στο οικοσύστημα της JavaScript και πιο συγκεκριμένα στο περιβάλλον Node.js (Node.js Foundation, 2024), σε συνδυασμό με μια σχεσιακή βάση δεδομένων, καθώς η προσέγγιση αυτή αποτελεί ένα δοκιμασμένο πρότυπο για την ανάπτυξη σύγχρονων, αξιόπιστων και επεκτάσιμων διαδικτυακών εφαρμογών (Δουληγέρης κ.συν., 2013).

2.2.1 Διεπαφή χρήστη (Frontend)

Για τη διεπαφή χρήστη επιλέχθηκε η μηχανή προτύπων EJS (Embedded JavaScript Templates), η οποία επιτρέπει την παραγωγή των σελίδων HTML στην πλευρά του διακομιστή (server-side rendering) (EJS Project, 2024). Η επιλογή αυτή προσφέρει απλότητα, ταχύτητα ανάπτυξης και άμεση ενσωμάτωση των δεδομένων του backend μέσα στις σελίδες, χωρίς την πολυπλοκότητα ενός αυτόνομου frontend πλαισίου (framework). Για την οπτική εμφάνιση και την απόκριση σε διαφορετικές συσκευές αξιοποιήθηκε η Bootstrap (Bootstrap Team, 2024), ενώ τα διαδραστικά γραφήματα του περιβάλλοντος διαχειριστή υλοποιήθηκαν με τη βιβλιοθήκη Chart.js (Chart.js Contributors, 2024).

2.2.2 Λειτουργία της εφαρμογής (Backend)

Το backend αναπτύχθηκε με χρήση του περιβάλλοντος εκτέλεσης Node.js και του πλαισίου Express.js (Express.js Team, 2024α). Ο συνδυασμός αυτός επέτρεψε τη δημιουργία ενός ισχυρού και γρήγορου διακομιστή, ο οποίος διαχειρίζεται τα αιτήματα των χρηστών, εφαρμόζει τη λογική της εφαρμογής και επικοινωνεί με τη βάση δεδομένων.

Η χρήση της JavaScript τόσο στο frontend όσο και στο backend διευκόλυνε την ομαλή ανάπτυξη ολόκληρου του συστήματος υπό ένα ενιαίο τεχνολογικό οικοσύστημα. Τέλος, η χρήση των συγκεκριμένων τεχνολογιών διευκολύνει τη μελλοντική επέκταση της εφαρμογής ανάλογα με τις ανάγκες του χρήστη και τις οικονομικές του δυνατότητες.

2.2.3 Βάση δεδομένων

Για την επίμονη αποθήκευση (persistent storage) των δεδομένων χρησιμοποιήθηκε η PostgreSQL. Πρόκειται για ένα ισχυρό σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων (RDBMS) (PostgreSQL Global Development Group, 2024), το οποίο προσφέρει αυστηρή ακεραιότητα δεδομένων, υποστήριξη συναλλαγών και σχέσεων μεταξύ των πινάκων μέσω ξένων κλειδιών (foreign keys). Τα χαρακτηριστικά αυτά κρίθηκαν ιδανικά για μια εφαρμογή όπου οι αναφορές, οι χρήστες και οι ψήφοι συνδέονται με σαφώς καθορισμένες σχέσεις.

2.2.4 Ασφάλεια και αυθεντικοποίηση

Για την αυθεντικοποίηση των χρηστών χρησιμοποιήθηκε η βιβλιοθήκη Passport.js (Passport.js Project, 2024), η οποία υποστηρίζει τόσο την τοπική στρατηγική (σύνδεση με email και κωδικό) όσο και τη σύνδεση μέσω λογαριασμού Google (OAuth 2.0). Οι κωδικοί πρόσβασης δεν αποθηκεύονται ποτέ σε απλό κείμενο, αλλά κρυπτογραφούνται με τη βιβλιοθήκη bcrypt (Provos & Mazieres, 1999) και στη συνέχεια αποθηκεύονται. Η διαχείριση των συνεδριών (sessions) πραγματοποιείται μέσω του express-session (Express.js Team, 2024β), διασφαλίζοντας ότι μόνο οι ταυτοποιημένοι χρήστες έχουν πρόσβαση σε προστατευμένο περιεχόμενο.

2.2.5 Τεχνητή Νοημοσύνη και χαρτογράφηση

Ένα από τα καινοτόμα και διαφοροποιημένα χαρακτηριστικά της προτεινόμενης πλατφόρμας συγκρινόμενο με άλλα είναι η αυτόματη εκτίμηση του επιπέδου επικινδυνότητας κάθε αναφοράς. Για τον σκοπό αυτόν αξιοποιήθηκε το Ollama, ένα εργαλείο που επιτρέπει την τοπική εκτέλεση Μεγάλων Γλωσσικών Μοντέλων (LLMs) (Ollama Team, 2024), στη συγκεκριμένη υλοποίηση του μοντέλου llama3.2. Η τοπική εκτέλεση εξασφαλίζει ότι τα δεδομένα δεν αποστέλλονται σε εξωτερικές υπηρεσίες, ενισχύοντας την ιδιωτικότητα, τη γρήγορη λήψη απόφασης και το χαμηλό κόστος. Παράλληλα, για την επιπλέον πληροφόρηση των υπηρεσιών και των χρηστών προστέθηκε η γεωγραφική τοποθέτηση και η οπτικοποίηση των αναφορών σε χάρτη χρησιμοποιώντας το Google Maps Platform (Google Developers, 2024α, 2024β).

3. Ανάλυση Απαιτήσεων και Σχεδιασμός

Σε αυτό το κεφάλαιο γίνεται η ανάλυση απαιτήσεων του συστήματος και επεξηγείται ο σχεδιασμός του. Συγκεκριμένα, καταγράφονται αναλυτικά οι λειτουργικές και μη λειτουργικές απαιτήσεις που καθόρισαν την ανάπτυξη της εφαρμογής, ενώ παρουσιάζεται η αρχιτεκτονική του συστήματος και ο σχεδιασμός της βάσης δεδομένων.

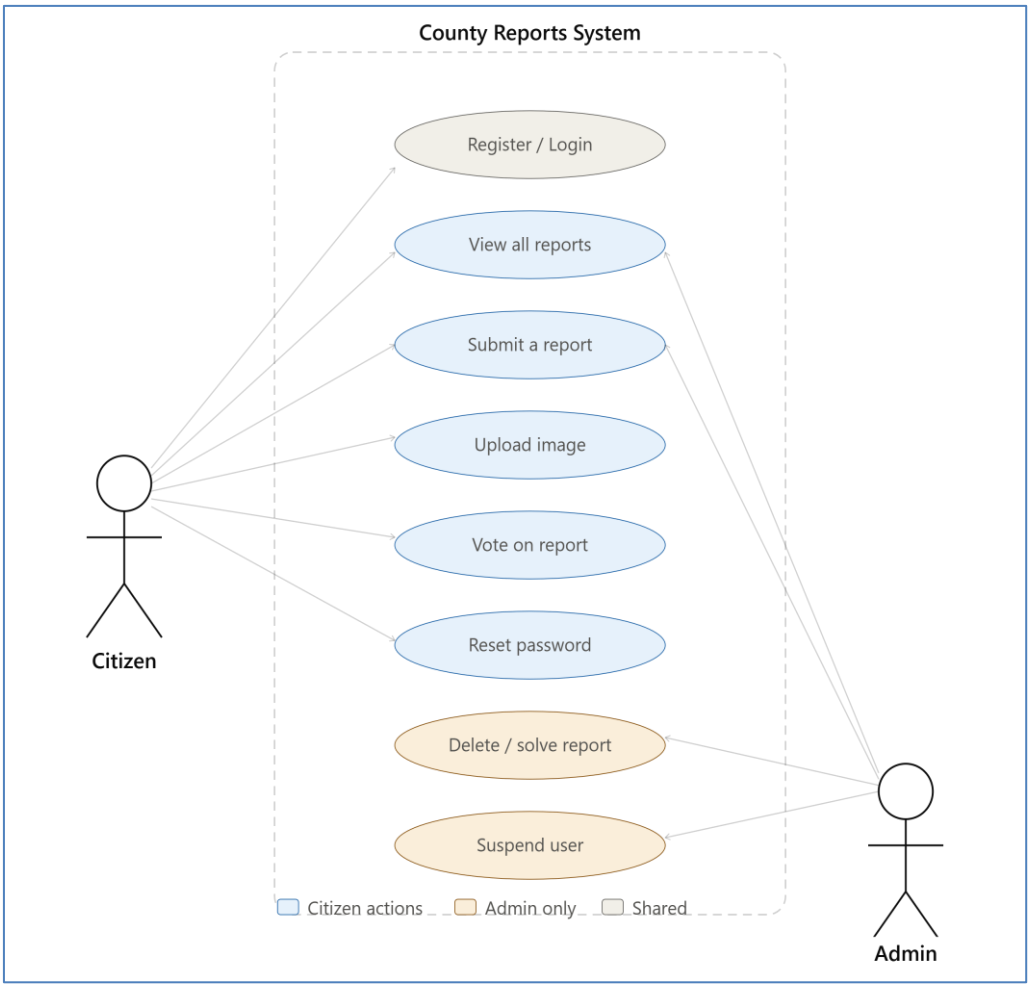
3.1 Λειτουργικές απαιτήσεις του συστήματος

Οι λειτουργικές απαιτήσεις περιγράφουν τι ακριβώς πρέπει να κάνει το σύστημα. Για την εύρυθμη λειτουργία της παρούσας πλατφόρμας, υλοποιήθηκαν οι εξής βασικές λειτουργίες:

1. **Διαχείριση χρηστών και αυθεντικοποίηση:** Το σύστημα επιτρέπει την εγγραφή και είσοδο χρηστών, παρέχοντας αυθεντικοποίηση μέσω email/κωδικού ή λογαριασμού Google, καθώς και εξουσιοδότηση βάσει ρόλων.
2. **Ρόλοι χρηστών:** Υποστηρίζονται διακριτοί ρόλοι, όπως του απλού χρήστη (user) και του διαχειριστή της πλατφόρμας (admin), ενώ προβλέπεται και η κατάσταση αναστολής λογαριασμού (suspended).
3. **Υποβολή αναφορών:** Ο χρήστης μπορεί να υποβάλλει νέες αναφορές προβλημάτων, συμπληρώνοντας τίτλο, κατηγορία, περιγραφή, προαιρετική εικόνα και τοποθεσία μέσω διαδραστικού χάρτη.
4. **Αυτόματη εκτίμηση επικινδυνότητας:** Κάθε νέα αναφορά αξιολογείται αυτόματα ως προς το επίπεδο επικινδυνότητάς της (υψηλό, μέτριο, χαμηλό) μέσω ενός τοπικού μοντέλου Τεχνητής Νοημοσύνης.
5. **Σύστημα ψηφοφορίας:** Οι εγγεγραμμένοι χρήστες μπορούν να ψηφίζουν τις αναφορές άλλων χρηστών, ώστε να αναδεικνύονται και να προτεραιοποιούνται τα σημαντικότερα κατά τόπους προβλήματα. Κάθε χρήστης μπορεί να ψηφίσει μία μόνο φορά ανά αναφορά.
6. **Διαχείριση από τον διαχειριστή:** Ο διαχειριστής έχει τη δυνατότητα να επιλύει ή να διαγράφει αναφορές, να αναστέλλει ή να επαναφέρει κακόβουλους λογαριασμούς χρηστών και να παρακολουθεί οπτικοποιημένα συγκεντρωτικά στατιστικά στοιχεία των αναφορών (σύνολο αναφορών, κατηγορίες αναφορών, επικινδυνότητα αναφορών).
7. **Ειδοποιήσεις μέσω email:** Το σύστημα αποστέλλει αυτοματοποιημένα μηνύματα ηλεκτρονικού ταχυδρομείου, τόσο για την επαναφορά του κωδικού πρόσβασης όσο και για την ενημέρωση του πολίτη σχετικά με την πρόοδο του αιτήματός του.

Το Διάγραμμα 1 που ακολουθεί αναπαριστά σχηματικά τους τρόπους με τους οποίους μπορεί ένας χρήστης να αλληλεπιδράσει με το σύστημα. Όπως αποτυπώνεται σε αυτό, η πλατφόρμα χρησιμοποιεί ένα σύστημα ελέγχου πρόσβασης βάσει ρόλων (Role-Based Access Control). Πιο αναλυτικά, οι ρόλοι που καθορίζουν την αλληλεπίδραση των μελών με το σύστημα είναι οι εξής:

- **Απλός χρήστης (User):** Αποτελεί τον βασικό τύπο λογαριασμού. Ένας χρήστης έχει δικαίωμα να περιηγείται στην πλατφόρμα, να βλέπει τις αναφορές στον χάρτη και στη λίστα, να υποβάλλει νέες αναφορές και να ψηφίζει τις αναφορές άλλων πολιτών, ώστε να ιεραρχηθούν.
- **Διαχειριστής (Admin):** Ο ρόλος αυτός αφορά την εποπτεία και τη συντήρηση της πλατφόρμας. Ο διαχειριστής έχει πλήρη πρόσβαση: μπορεί να επιλύει ή να διαγράφει οποιαδήποτε αναφορά, να αναστέλλει ή να επαναφέρει λογαριασμούς χρηστών, να ελέγχει τη χρήση της πλατφόρμας και να έχει πρόσβαση σε συγκεντρωτικά στατιστικά και γραφήματα. Επισημαίνεται ωστόσο πως ο διαχειριστής δεν συμμετέχει στην υποβολή ή την ψηφοφορία αναφορών, καθώς ο ρόλος του είναι αμιγώς εποπτικός.
- **Χρήστης σε αναστολή (suspended):** Πρόκειται για μια ειδική κατάσταση και όχι για διακριτό ρόλο. Ένας λογαριασμός που έχει τεθεί σε αναστολή από τον διαχειριστή διατηρεί τη δυνατότητα περιήγησης, αλλά χάνει το δικαίωμα υποβολής νέων αναφορών.



ΔΙΑΓΡΑΜΜΑ 1: Περιπτώσεις χρήσης (use case diagram).[1]

3.2 Μη λειτουργικές απαιτήσεις του συστήματος

Οι μη λειτουργικές απαιτήσεις αφορούν τα ποιοτικά χαρακτηριστικά, την απόδοση και τους περιορισμούς του συστήματος, διασφαλίζοντας ότι η τελική εμπειρία του χρήστη θα είναι άρτια. Για την ανάπτυξη του County Reports, τέθηκαν και ικανοποιήθηκαν οι ακόλουθες μη λειτουργικές απαιτήσεις:

1. **Εργονομία και χρηστικότητα:** Η διεπαφή χρήστη σχεδιάστηκε κατά τέτοιον τρόπο, ώστε να είναι εύκολα κατανοητή σε ανθρώπους με περιορισμένο τεχνολογικό υπόβαθρο, οπτικά καθαρή", προσφέροντας σαφή ροή πλοήγησης δίνοντας ιδιαίτερη έμφαση στην παροχή άμεσης ανατροφοδότησης. Για παράδειγμα, υπάρχει η ένδειξη «Εκτίμηση...» για την αξιολόγηση επικινδυνότητας μιας αναφοράς ή η άμεση ενημέρωση του μετρητή ψήφων χωρίς επαναφόρτωση της σελίδας.
2. **Ασφάλεια και ιδιωτικότητα:** Η προστασία των δεδομένων υλοποιήθηκε σε πολλαπλά επίπεδα. Πιο συγκεκριμένα, κρυπτογράφηση κωδικών με bcrypt, έλεγχος πρόσβασης στους ελεγκτές (controllers), ώστε μόνο εξουσιοδοτημένοι χρήστες να εκτελούν συγκεκριμένες ενέργειες και χρήση παραμετροποιημένων ερωτημάτων (parameterized queries) για την αποφυγή επιθέσεων SQL Injection.

[1] Το Διάγραμμα 1 δημιουργήθηκε με το Draw.io.

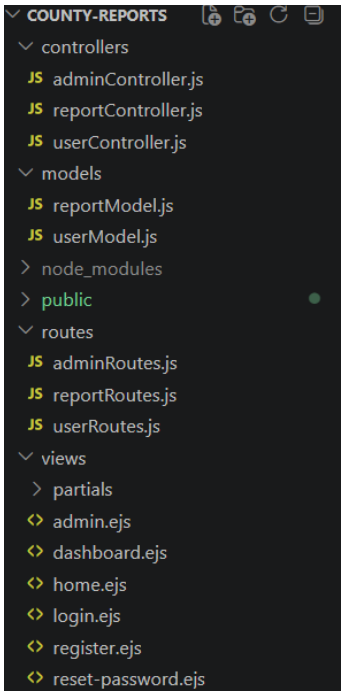
3. **Απόδοση και αποκρισιμότητα:** Το σύστημα ανταποκρίνεται γρήγορα στα αιτήματα των χρηστών. Χαρακτηριστικό παράδειγμα αποτελεί η ασύγχρονη εκτέλεση της εκτίμησης επικινδυνότητας, όπου ο χρήστης λαμβάνει άμεση απόκριση μετά την υποβολή, ενώ η χρονοβόρα κλήση στο μοντέλο Τεχνητής Νοημοσύνης εκτελείται στο παρασκήνιο.
4. **Συντηρησιμότητα λογισμικού:** Η αυστηρή τήρηση του προτύπου MVC και ο σαφής διαχωρισμός των ευθυνών μεταξύ Models, Views και Controllers προσφέρουν έναν δομημένο και ευανάγνωστο κώδικα, διευκολύνοντας την ευκολότερη ενσωμάτωση νέων δυνατοτήτων στο μέλλον (Bhargava κ.συν., 2026).

3.3 Αρχιτεκτονικός σχεδιασμός (Model - View - Controller)

Η αρχιτεκτονική του συστήματος βασίζεται στο πρότυπο σχεδίασης Model-View-Controller (MVC), το οποίο διαχωρίζει την εφαρμογή σε τρία αλληλοσυνδεόμενα, αλλά διακριτά, στρώματα. Ο διαχωρισμός αυτός αποτελεί θεμελιώδη αρχή της μηχανικής λογισμικού, καθώς απομονώνει τη λογική των δεδομένων από την παρουσίασή τους (Selfa κ.συν., 2006· Dey, 2011). Συγκεκριμένα:

- **Μοντέλο (Model):** Αποτελεί το στρώμα δεδομένων. Περιλαμβάνει τα αρχεία που είναι υπεύθυνα για την επικοινωνία με τη βάση δεδομένων PostgreSQL, εκτελώντας τα ερωτήματα SQL για την ανάκτηση, εισαγωγή, ενημέρωση και διαγραφή εγγραφών (userModel.js, reportModel.js).
- **Όψη (View):** Αποτελεί το στρώμα παρουσίασης. Περιλαμβάνει τα πρότυπα EJS, τα οποία είναι αποκλειστικά υπεύθυνα για την οπτική απεικόνιση των δεδομένων στον χρήστη, χωρίς να εμπεριέχουν επιχειρησιακή λογική.
- **Ελεγκτής (Controller):** Αποτελεί το ενδιάμεσο στρώμα λογικής. Δέχεται τα αιτήματα του χρήστη, αξιοποιεί τα Models για την άντληση ή την τροποποίηση των δεδομένων και τέλος, αποφασίζει ποια όψη θα αποδώσει ή πού θα ανακατευθύνει τον χρήστη (reportController.js, userController.js, adminController.js).

Τη διασύνδεση των διευθύνσεων URL με τους αντίστοιχους ελεγκτές αναλαμβάνει ένα διακριτό στρώμα δρομολόγησης (routes), το οποίο διατηρεί τον κώδικα οργανωμένο και ευανάγνωστο (Εικόνα 3.1).

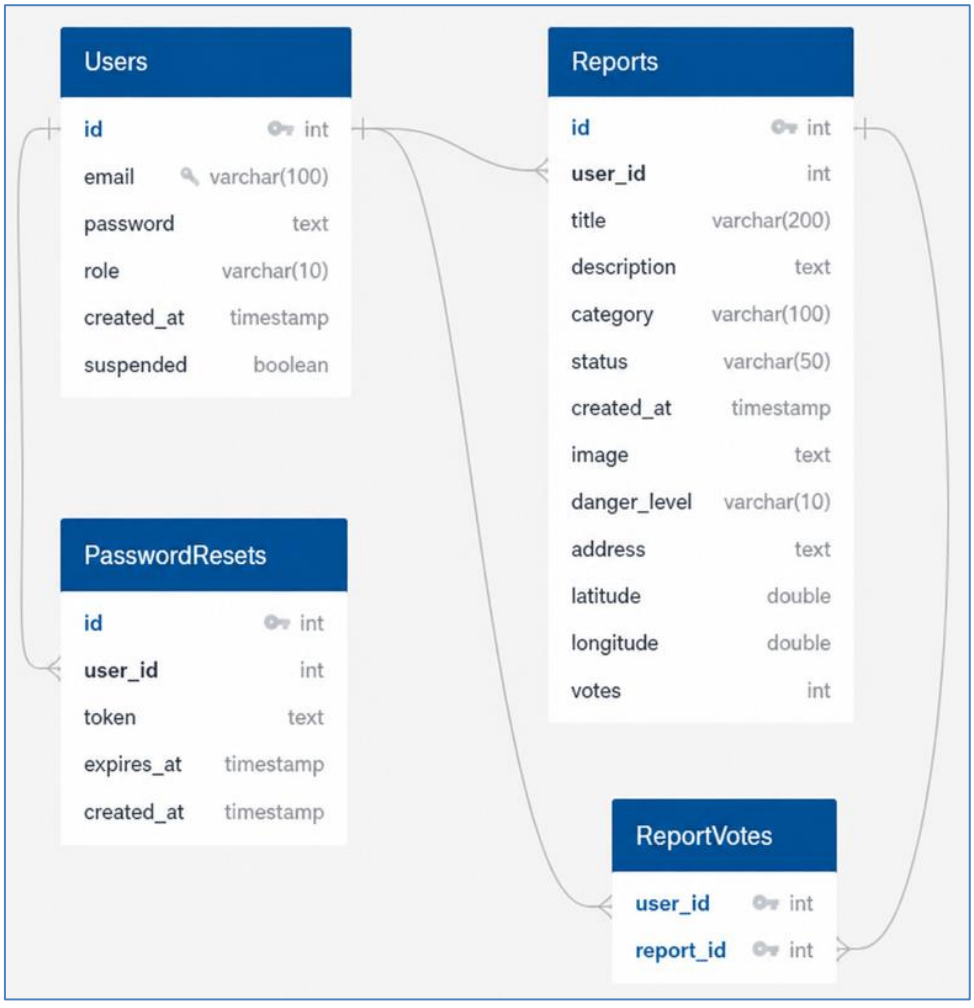


Εικόνα 3.1: Αρχιτεκτονική σχεδιασμού MVC.

3.4 Σχεδιασμός βάσης δεδομένων

Για την αποθήκευση και τη διαχείριση των δεδομένων της εφαρμογής χρησιμοποιείται η σχεσιακή βάση δεδομένων PostgreSQL. Τα δεδομένα του County Reports οργανώνονται σε τέσσερις διακριτούς πίνακες (tables), οι οποίοι συνδέονται μεταξύ τους μέσω σχέσεων ως εξής:

- **Users:** Αποτελεί τον πυρήνα του συστήματος. Αποθηκεύει τα στοιχεία των μελών (email, κρυπτογραφημένο κωδικό), τον ρόλο τους (απλός χρήστης/user ή διαχειριστής/admin), την κατάσταση αναστολής (suspended) και την ημερομηνία εγγραφής.
- **Reports:** Περιέχει όλες τις αναφορές προβλημάτων. Κάθε εγγραφή περιλαμβάνει τίτλο, περιγραφή, κατηγορία, όνομα αρχείου εικόνας, επίπεδο επικινδυνότητας (danger_level), διεύθυνση, γεωγραφικές συντεταγμένες (latitude, longitude), κατάσταση (open/solved), αριθμό ψήφων και ξένο κλειδί προς τον χρήστη που την υπέβαλε (user_id).
- **Report_votes:** Υλοποιεί τη σχέση «πολλά-προς-πολλά» μεταξύ χρηστών και αναφορών για το σύστημα ψηφοφορίας. Κάθε εγγραφή συνδέει έναν χρήστη (user_id) με μια αναφορά (report_id), διασφαλίζοντας ότι κάθε χρήστης μπορεί να ψηφίσει μία μόνο φορά ανά αναφορά.
- **Password_resets:** Αποθηκεύει τα προσωρινά διακριτικά (tokens) που δημιουργούνται κατά τη διαδικασία επαναφοράς κωδικού πρόσβασης, μαζί με την ημερομηνία λήξης τους και τον αντίστοιχο χρήστη (Διάγραμμα 2).



ΔΙΑΓΡΑΜΜΑ 2: Σχήμα βάσης δεδομένων (ER Diagram).[2]

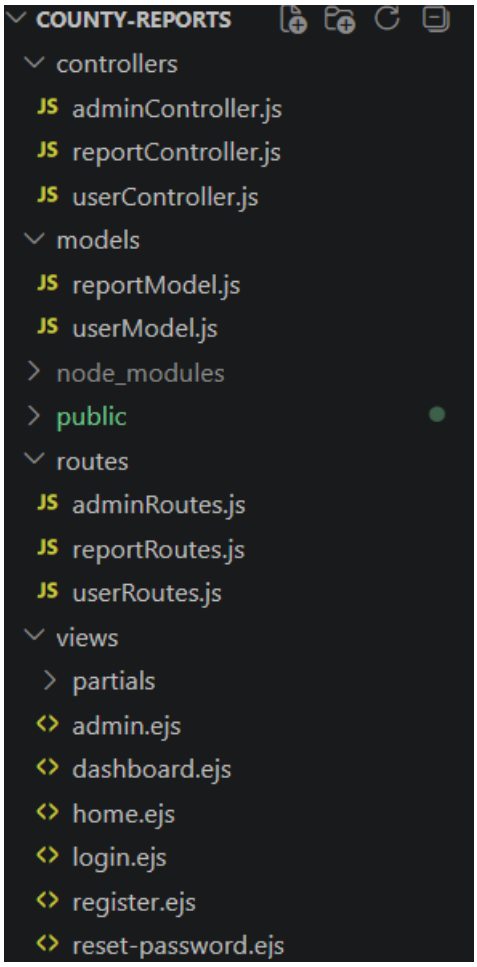
[2] Το Διάγραμμα 2 δημιουργήθηκε με το Draw.io.

4. Υλοποίηση Συστήματος

Στο παρόν κεφάλαιο παρουσιάζεται αναλυτικά η διαδικασία υλοποίησης της εφαρμογής. Περιγράφεται η αρχιτεκτονική οργάνωση του πηγαίου κώδικα σύμφωνα με το πρότυπο MVC, αναλύεται η υλοποίηση του backend και της δρομολόγησης και παρουσιάζονται οι μηχανισμοί ασφάλειας και αυθεντικοποίησης καθώς και η ενσωμάτωση της Τεχνητής Νοημοσύνης.

4.1 Οργάνωση του πηγαίου κώδικα (MVC)

Για λόγους εύκολης συντηρησιμότητας και επεκτασιμότητας ο πηγαίος κώδικας οργανώθηκε σε μια ιεραρχική δομή φακέλων, που αντικατοπτρίζει άμεσα το πρότυπο MVC. Η ρίζα του έργου δομείται όπως φαίνεται στην Εικόνα 4.1 που ακολουθεί.



Εικόνα 4.1: Δομή MVC.

- **index.js:** Το σημείο εισόδου της εφαρμογής. Αρχικοποιεί τον διακομιστή Express, ρυθμίζει τη διεπαφή μεταξύ των λειτουργειών τις εφαρμογής και της εμφάνισης στον χρήστη. Επίσης αρχικοποιεί τη βιβλιοθήκη Passport.js και τη βασική διαδρομή στην αρχική σελίδα (Εικόνα 4.2).

```
import express from "express";
import bodyParser from "body-parser";
import session from "express-session";
import passport from "passport";
import { Strategy } from "passport-local";
import GoogleStrategy from "passport-google-oauth2";
import bcrypt from "bcrypt";
import env from "dotenv";

import db from "./db.js";
import * as User from "./models/userModel.js";
import * as Report from "./models/reportModel.js";

// ── ROUTES ──────
import reportRoutes from "./routes/reportRoutes.js";
import userRoutes from "./routes/userRoutes.js";
import adminRoutes from "./routes/adminRoutes.js";

const app = express();
const port = 3000;
env.config();

app.locals.GOOGLE_MAPS_API_KEY = process.env.GOOGLE_MAPS_API_KEY;

// ── MIDDLEWARE ──────────
app.use(session({
  secret: process.env.SESSION_SECRET,
  resave: false,
  saveUninitialized: true,
}));
```

Εικόνα 4.2: index.js / Αρχικοποίηση βασικών βιβλιοθηκών και ρυθμίσεων.

- **db.js:** Περιέχει τη ρύθμιση της σύνδεσης (connection) με τη βάση δεδομένων PostgreSQL. Ακόμα, για τη διασφάλιση της σύνδεσης τα δεδομένα μεταφέρονται μέσω ενός .env αρχείου (Εικόνα 4.3).

```
import pg from "pg";
import env from "dotenv";

env.config();


const db = new pg.Client({
user: process.env.PG_USER,
host: process.env.PG_HOST,
database: process.env.PG_DATABASE,
password: process.env.PG_PASSWORD,
port: process.env.PG_PORT,
});
db.connect();

export default db;
```

Εικόνα 4.3: db.js / Αρχικοποίηση βάσης δεδομένων.

- **models/:** Περιλαμβάνει τα μοντέλα δεδομένων (userModel.js, reportModel.js), τα οποία ενσωματώνουν όλα τα ερωτήματα SQL. Το κάθε μοντέλο διαθέτει πολλαπλές διακριτές συναρτήσεις για τα ερωτήματα προς τη βάση δεδομένων με κύρια διαφορά του userModel να αφορά τα ερωτήματα από τον χρήστη προς τον χρήστη και το reportModel από τον χρήστη για την εφαρμογή προς περαιτέρω επεξεργασία (Εικόνες 4.4 & 4.5).

```
import db from "../db.js";

// Get all reports with user email, ordered by votes then date
export async function getAllReports() {···
}

// Get all report IDs that a specific user has voted for
export async function getVotedIds(userId) {···
}

// Create a new report — returns the new row's id
export async function createReport(userId, title, description, category,
}

// Update danger level after async AI classification
export async function updateDangerLevel(reportId, dangerLevel) {···
}

// Delete a report
export async function deleteReport(reportId) {···
}

// Mark a report as solved
export async function solveReport(reportId) {···
}
```

Εικόνα 4.4: Λειτουργίες reportModel.js.

```
import db from "../db.js";

// Find a user by id
export async function findUserById(id) {···
}

// Find a user by email
export async function findUserByEmail(email) {···
}

// Create a new user
export async function createUser(email, hashedPassword)
}

// Delete a user by id
export async function deleteUser(userId) {···
}

// Suspend a user
export async function suspendUser(userId) {···
}

//Unsuspend A User

export async function unsuspendUser(userId){···
}
```

Εικόνα 4.5: Λειτουργίες userModel.

- **controllers/:** Περιλαμβάνει τους ελεγκτές (reportController.js, userController.js, adminController.js), όπου βρίσκεται η λογική της εφαρμογής. Πιο συγκεκριμένα, στο αρχείο reportController.js βρίσκεται η υλοποίηση κάθε αιτήματος (request) που ζητά η εφαρμογή όπως η αποστολή email προς τον χρήστη ή η επεξεργασία της κάθε αναφοράς από το τοπικό llm. Αντίστοιχα, στο userController.js βρίσκεται η υλοποίηση κάθε αιτήματος (request) που αφορούν άμεσα τον χρήση, όπως για παράδειγμα η εγγραφή νέου χρήστη ή η αλλαγή κωδικού. Τέλος, στο adminController.js υλοποιείται κάθε λειτουργία που έχει να κάνει με τον διαχειριστή, μεταξύ των οποίων και ο αποκλεισμός κακόβουλων χρηστών (Εικόνες 4.6, 4.7 & 4.8).

```
// ── OLLAMA HELPER ──────────────────────────────────────
async function getDangerLevel(title, description) {
  console.log(`[Ollama] request — title: "${title}" | description: "${description}"`);
  try {
    const response = await fetch("http://localhost:11434/api/generate", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        model: "llama3.2",
        prompt: `You are a system that classifies the danger level of community problem reports.
          Based on this report, respond with ONLY one word: high, medium, or low.

          Rules:
          - high: immediate danger to people (fire, flood, gas leak, collapsed structure, fallen tree blocking road)
          - medium: potential danger or major inconvenience (broken road, damaged infrastructure, clogged drain)
          - low: minor issues (graffiti, noise, small potholes, aesthetic problems)

          Title: ${title}
          Description: ${description}

          Respond with only one word (high, medium, or low):`,
        stream: false
      })
    });
    const data = await response.json();
    const raw = data.response.trim();
    const result = raw.toLowerCase();
    console.log(`[Ollama] response — raw: "${raw}" | resolved: "${["high", "medium", "low"].includes(result) ? result : "low"}"`);
    if (["high", "medium", "low"].includes(result)) return result;
    return "low";
  } catch (err) {
    console.log("[Ollama] error:", err.message);
    return "low";
  }
}
```

Εικόνα 4.6: Ενσωμάτωση reportController / llm.

```
// POST /forgot-password — send reset email
export async function postForgotPassword(req, res) {
  const { email } = req.body;
  try {
    const user = await User.findUserByEmail(email);
    if (!user) return res.redirect("/login");

    const token = await User.createPasswordReset(user.id);

    await transporter.sendMail({
      from: `"County Reports" <${process.env.EMAIL_USER}>`,
      to: email,
      subject: "🔑 Password Reset Request",
      html: `
        <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0 auto;">
          <h2 style="color: #212529;">🔑 Reset Your Password</h2>
          <p>Click the button below to set a new password. This link expires in 1 hour.</p>
          <a href="http://localhost:3000/reset-password?token=${token}"
             style="display: inline-block; background: #212529; color: white; padding: 12px 24px; border-radius
            Reset Password
          </a>
          <p style="color: #6c757d; font-size: 13px;">If you didn't request this, ignore this email.</p>
        </div>
      `
    });

    res.redirect("/login");
  } catch (err) {
    console.log(err);
  }
}
```

Εικόνα 4.7: Λειτουργίες userController / Forgot password, reset email.

```
// Suspended User
export async function suspendUser(req,res){
  if(!req.isAuthenticated() || req.user.role !== "admin")
    return res.redirect("/dashboard");
  try{
    await User.suspendUser(req.params.id);
    res.redirect("/admin");
  }
  catch(err){
    console.log(err);
  }
}

//Unsuspend User
export async function unsuspendUser(req, res) {
  if (!req.isAuthenticated() || req.user.role !== "admin") return res.redirect("/dashboard");
  try {
    await User.unsuspendUser(req.params.id);
    res.redirect("/admin");
  } catch (err) {
    console.log(err);
  }
}
```

Εικόνα 4.8: Λειτουργίες adminController / Suspend-Unsuspend.

- **routes/:** Περιλαμβάνει τα αρχεία δρομολόγησης (reportRoutes.js, userRoutes.js, adminRoutes.js), που συνδέουν τις διευθύνσεις URL με τους controllers. Με άλλα λόγια, όταν ο χρήστης αιτείται οποιαδήποτε λειτουργεία από την εφαρμογή τότε τα εκάστοτε routes ανακατευθύνουν το αίτημα (request) στο κατάλληλο controller για περαιτέρω υλοποίηση του αιτήματος από την κατάλληλη συνάρτηση (Εικόνες 4.9, 4.10 & 4.11).

```
import express from "express";
import multer from "multer";
import path from "path";
import * as reportController from "../controllers/reportController.js";

const router = express.Router();

// Multer setup for image uploads
const storage = multer.diskStorage({
  destination: "public/uploads/",
  filename: (req, file, cb) => {
    cb(null, Date.now() + path.extname(file.originalname));
  }
});
const upload = multer({ storage });

// ── ROUTES ──────────────────────────────────────────

router.get("/dashboard", reportController.getDashboard);
router.post("/report", upload.single("image"), reportController.createReport);
router.post("/report/delete/:id", reportController.deleteReport);
router.post("/report/solve/:id", reportController.solveReport);
router.post("/report/vote/:id", reportController.voteReport);

export default router;
```

Εικόνα 4.9: Χρήσεις reportRoutes / all report routes.

```
import express from "express";
import * as adminController from "../controllers/adminController.js";

const router = express.Router();

// ── ROUTES ──────────────────────────────────────────

router.get("/admin", adminController.getAdmin);
//router.post("/admin/delete-user/:id", adminController.deleteUser);
router.post("/admin/suspend-user/:id", adminController.suspendUser);
router.post("/admin/unsuspend-user/:id", adminController.unsuspendUser);
export default router;
```

Εικόνα 4.10: Χρήσεις adminRoutes / all admin routes.

```
import express from "express";
import passport from "passport";
import * as userController from "../controllers/userController.js";

const router = express.Router();

// ── ROUTES ──────────────────────────────────────────────

router.get("/login", userController.getLogin);
router.get("/register", userController.getRegister);
router.get("/logout", userController.getLogout);
router.post("/forgot-password", userController.postForgotPassword);
router.get("/reset-password", userController.getResetPassword);
router.post("/reset-password", userController.postResetPassword);
```

Εικόνα 4.11: Χρήσεις userRoutes / all user routes.

- **views/:** Περιέχει τα πρότυπα EJS οργανωμένα σε επιμέρους σελίδες (home, dashboard, admin, login, κ.ά.) και σε επαναχρησιμοποιούμενα τμήματα (partials/header, partials/footer). Κατά αυτόν τον τρόπο σε κάθε απάντηση της εφαρμογής προς εμφάνιση στον χρήστη επαναχρησιμοποιούνται τα σταθερά τμήματα της σελίδας, για παράδειγμα το μενού ανακατεύθυνσης. Συμπληρωματικά, η χρήση ejs προσφέρει δυναμική δημιουργία της εκάστοτε σελίδας ανεξάρτητα από τα δεδομένα προσφέροντας τη δυνατότητα χρήσης της εφαρμογής και από κινητές συσκευές επιτυγχάνοντας έτσι καλύτερη εμπειρία για τον χρήστη και εξοικονόμηση πόρων (Εικόνες 4.12 & 4.13).

```
<section class="hero">
  <h1>Welcome to County Reports</h1>
  <p>Help us improve our community. Report local problems and track their progress.</p>
  <% if (typeof user !== 'undefined' && user) { %>
    <% if (user.role === 'admin') { %>
      <a href="/admin" class="btn btn-warning btn-lg me-2">🛠 Admin Panel</a>
    <% } else if (user.suspended) { %>
      <span class="btn btn-secondary btn-lg me-2 disabled">🚫 Account Suspended</span>
    <% } else { %>
      <a href="/dashboard" class="btn btn-primary btn-lg me-2">Report Incident</a>
    <% } %>
  <% } else { %>
    <a href="/register" class="btn btn-primary btn-lg me-2">Get Started</a>
    <a href="/login" class="btn btn-outline-light btn-lg">Login</a>
  <% } %>
</section>
```

Εικόνα 4.12: Χρήσεις views/ home.ejs / .ejs.

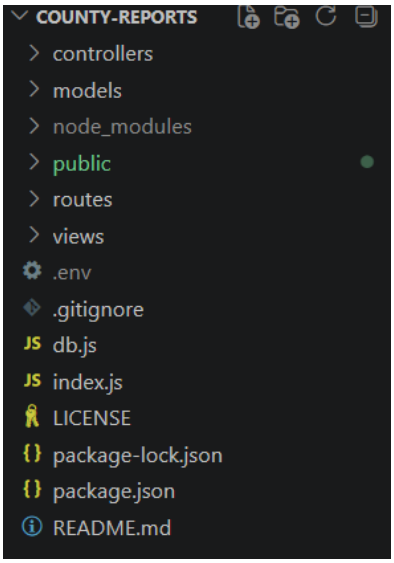
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>County Reports</title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/di
  <link rel="stylesheet" href="/styling.css">
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <script src="https://maps.googleapis.com/maps/api/js?key=<%= GOOGLE_MAPS_API
</head>
<body class="d-flex flex-column min-vh-100">

  <nav class="navbar navbar-dark bg-dark px-4">
    <a class="navbar-brand fw-bold" href="/">🏛 County Reports</a>
    <div>
      <% if (typeof user !== 'undefined' && user) { %>
        <span class="text-white me-3"><%= user.email %></span>
        <% if (user.role === 'admin') { %>
          <span class="badge bg-warning text-dark me-3">Admin</span>
        <% } %>
        <% if (user.role === 'admin') { %>
          <a href="/admin" class="btn btn-warning btn-sm me-2">🛠 Admin</a>
        <% } %>
        <a href="/logout" class="btn btn-outline-light btn-sm">Logout</a>
      <% } else { %>
        <a href="/login" class="btn btn-outline-light btn-sm me-2">Login</a>
        <a href="/register" class="btn btn-light btn-sm">Register</a>
      <% } %>
    </div>
  </nav>
```

Εικόνα 4.13: Χρήσεις partials/header.ejs / header.

- **public/:** Περιλαμβάνει το σύνολο των στατικών αρχείων της εφαρμογής, όπως το στυλ (styling.css) και τον φάκελο uploads/ όπου αποθηκεύονται οι εικόνες.

Ακολουθεί η Εικόνα 4.14 που παρουσιάζει τη δομή του έργου, όπως εμφανίζεται στο VSCode.



Εικόνα 4.14: Δομή των φακέλων του έργου στο VSCode.

4.2 Υλοποίηση περιβάλλοντος (Backend)

Ο πυρήνας του backend υλοποιήθηκε στο αρχείο index.js με τη χρήση του Express.js. Κατά την αρχικοποίηση, ρυθμίζονται τα απαραίτητα ενδιάμεσα λογισμικά. Συγκεκριμένα, το express-session για τη διαχείριση των συνεδριών, το body-parser για την λήψη των δεδομένων των φορμών, η εξυπηρέτηση των στατικών αρχείων από τον φάκελο public και η αρχικοποίηση του Passport.js. Επιπλέον, ορίζεται η μηχανή προτύπων EJS ως μηχανή απόδοσης των όψεων.

Το στρώμα των μοντέλων (models) αναλαμβάνει την επικοινωνία με τη βάση δεδομένων. Όλα τα ερωτήματα προς την PostgreSQL υλοποιούνται με παραμετροποιημένα ερωτήματα (parameterized queries), τα οποία όχι μόνο προστατεύουν από επιθέσεις SQL Injection, αλλά διατηρούν και τη λογική των δεδομένων απομονωμένη από τους ελεγκτές. Χαρακτηριστικό παράδειγμα αποτελεί η συνάρτηση υποβολής ψήφου, η οποία ελέγχει αν ο χρήστης έχει ήδη ψηφίσει και, ανάλογα προσθέτει ή αφαιρεί την ψήφο του εντός της ίδιας λειτουργίας (toggle).

4.3 Δρομολόγηση και ελεγκτές (Routes & controllers)

Η δρομολόγηση των αιτημάτων οργανώνεται σε τρία διακριτά αρχεία, ανάλογα με τη θεματική τους ενότητα, δηλ. τις αναφορές, τους χρήστες και τη διαχείριση. Κάθε διαδρομή αντιστοιχίζεται σε μια συνάρτηση ελεγκτή, η οποία περιέχει τη σχετική λογική.

Ένα κρίσιμο στοιχείο των ελεγκτών είναι ο έλεγχος εξουσιοδότησης που προηγείται κάθε ευαίσθητης ενέργειας. Για παράδειγμα, πριν την εμφάνιση του πίνακα διαχείρισης ή την επίλυση μιας αναφοράς, ο ελεγκτής επιβεβαιώνει ότι ο χρήστης είναι αυθεντικοποιημένος και διαθέτει τον ρόλο του διαχειριστή. Αντίστοιχα, ένας χρήστης σε αναστολή εμποδίζεται από την υποβολή νέων αναφορών τόσο στο επίπεδο της διεπαφής όσο και στο επίπεδο του ελεγκτή, εξασφαλίζοντας διπλή προστασία.

4.4 Μηχανισμοί ασφάλειας και αυθεντικοποίησης

Η αυθεντικοποίηση υλοποιήθηκε με τη βιβλιοθήκη Passport.js, μέσω δύο στρατηγικών. Η τοπική στρατηγική (Local Strategy) ελέγχει τα διαπιστευτήρια του χρήστη συγκρίνοντας τον υποβληθέντα κωδικό με τον κρυπτογραφημένο κωδικό της βάσης, αξιοποιώντας τη συνάρτηση σύγκρισης του bcrypt. Η στρατηγική Google (OAuth 2.0) επιτρέπει τη σύνδεση μέσω λογαριασμού Google, δημιουργώντας αυτόματα νέο χρήστη εφόσον δεν υπάρχει ήδη.

Σημαντική σχεδιαστική απόφαση αποτέλεσε ο τρόπος διαχείρισης της συνεδρίας. Αντί να αποθηκεύεται ολόκληρο το αντικείμενο του χρήστη στη συνεδρία, αποθηκεύεται μόνο το αναγνωριστικό του (id), ενώ σε κάθε αίτημα ανακτώνται τα επικαιροποιημένα στοιχεία του απευθείας από τη βάση δεδομένων. Με αυτόν τον τρόπο, οποιαδήποτε αλλαγή στην κατάσταση του χρήστη, όπως η αναστολή του λογαριασμού του από τον διαχειριστή, εφαρμόζεται άμεσα, χωρίς να απαιτείται επανασύνδεση (Εικόνες 4.15 & 4.16).

```
// ── PASSPORT LOCAL STRATEGY ──────────────────────────
passport.use("local", new Strategy(async (username, password, cb) => {
  try {
    const user = await User.findUserByEmail(username);
    if (!user) return cb(null, false);
    if(user.suspended) return cb(null,false, {message:"suspended"});

    bcrypt.compare(password, user.password, (err, valid) => {
      if (err) return cb(err);
      return valid ? cb(null, user) : cb(null, false);
    });
  } catch (err) {
    return cb(err);
  }
}));
```

Εικόνα 4.15: Αυθεντικοποίηση του χρήστη με τη χρήση passport local.

```
// ── PASSPORT GOOGLE STRATEGY ─────────────────────────
passport.use("google", new GoogleStrategy({
  clientID: process.env.GOOGLE_CLIENT_ID,
  clientSecret: process.env.GOOGLE_CLIENT_SECRET,
  callbackURL: "http://localhost:3000/auth/google/callback",
  userProfileURL: "https://www.googleapis.com/oauth2/v3/userinfo",
}, async (accessToken, refreshToken, profile, cb) => {
  try {
    let user = await User.findUserByEmail(profile.email);
    if (!user) {
      user = await User.createUser(profile.email, "google");
    }
    return cb(null, user);
  } catch (err) {
    return cb(err);
  }
}));

passport.serializeUser((user, cb) => cb(null, user.id));
passport.deserializeUser(async (id, cb) => {
  try {
    const user = await User.findUserById(id);

    cb(null, user);
  } catch (err) {
    cb(err);
  }
});

app.listen(port, () => console.log(`Server running on port ${port}`));
```

Εικόνα 4.16: Αυθεντικοποίηση του χρήστη μέσω Google passport.

4.5 Ενσωμάτωση Τεχνητής Νοημοσύνης (Ollama)

Η αυτόματη εκτίμηση του επιπέδου επικινδυνότητας αποτελεί ένα από τα πιο καίρια στοιχεία της εφαρμογής. Κατά την υποβολή μιας αναφοράς, ο τίτλος και η περιγραφή του εκάστοτε αιτήματος αποστέλλονται σε ένα τοπικά εκτελούμενο μοντέλο μέσω του Ollama, συνοδευόμενα από κατάλληλα διαμορφωμένες οδηγίες (prompts) που το κατηγοριοποιούν (Kontogianni κ.συν, 2026) σε ένα από τα ακόλουθα επίπεδα επικινδυνότητας: υψηλό, μέτριο ή χαμηλό.

Επειδή η εκτέλεση ενός γλωσσικού μοντέλου είναι χρονοβόρα, η κλήση αυτή σχεδιάστηκε ώστε να είναι ασύγχρονη (Kontogianni & Alepis, 2020). Κατ' αυτόν τον τρόπο, η αναφορά αποθηκεύεται αρχικά με την προσωρινή ένδειξη «Εκτίμηση...» και ο χρήστης ανακατευθύνεται αμέσως, χωρίς κάποια καθυστέρηση. Η κλήση στο μοντέλο εκτελείται στο παρασκήνιο και μόλις ολοκληρωθεί, η εγγραφή της αναφοράς ενημερώνεται με το τελικό επίπεδο επικινδυνότητας. Σε περίπτωση που το μοντέλο δεν είναι διαθέσιμο, προβλέπεται ασφαλής εναλλακτική διαδρομή, διασφαλίζοντας ότι η λειτουργία της εφαρμογής δεν διακόπτεται ποτέ (Εικόνα 4.17).

```
// ── OLLAMA HELPER ──────────────────────────────────────
async function getDangerLevel(title, description) {
  console.log(`[Ollama] request — title: "${title}" | description: "${description}"`);
  try {
    const response = await fetch("http://localhost:11434/api/generate", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        model: "llama3.2",
        prompt: `You are a system that classifies the danger level of community problem reports.
          Based on this report, respond with ONLY one word: high, medium, or low.

          Rules:
          - high: immediate danger to people (fire, flood, gas leak, collapsed structure, fallen tree blocking road)
          - medium: potential danger or major inconvenience (broken road, damaged infrastructure, clogged drain)
          - low: minor issues (graffiti, noise, small potholes, aesthetic problems)

          Title: ${title}
          Description: ${description}

          Respond with only one word (high, medium, or low):`,
        stream: false
      })
    });
    const data = await response.json();
    const raw = data.response.trim();
    const result = raw.toLowerCase();
    console.log(`[Ollama] response — raw: "${raw}" | resolved: "${["high", "medium", "low"].includes(result) ? result : "low"}"`);
    if (["high", "medium", "low"].includes(result)) return result;
    return "low";
  } catch (err) {
    console.log("[Ollama] error:", err.message);
    return "low";
  }
}
```

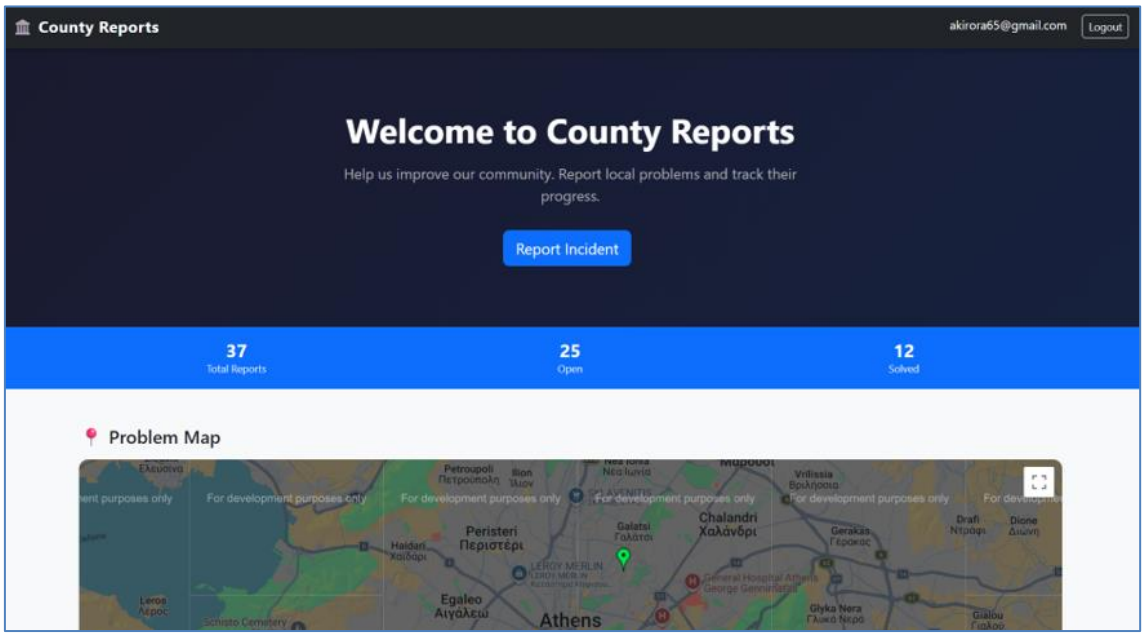
Εικόνα 4.17: Ασύγχρονη επεξεργασία αιτήματος.

5. Παρουσίαση Εφαρμογής

Στο κεφάλαιο αυτό παρουσιάζεται η διεπαφή χρήστη της πλατφόρμας μέσα από τις βασικές της οθόνες, ακολουθώντας τη φυσική ροή χρήσης, από τις δημόσιες σελίδες και τη διαδικασία αυθεντικοποίησης, έως το περιβάλλον του απλού χρήστη και του διαχειριστή.

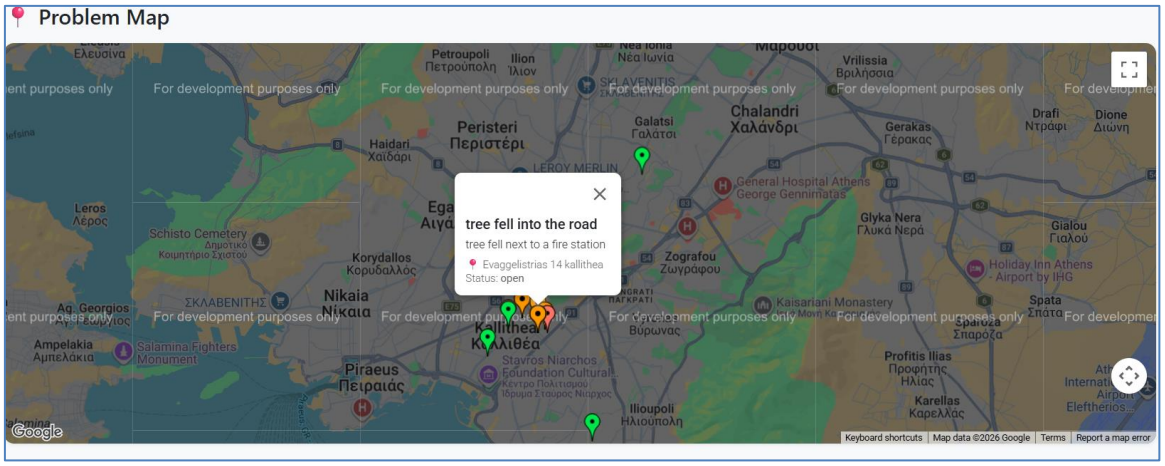
5.1 Δημόσιες σελίδες

Η αρχική σελίδα (home page) αποτελεί το σημείο εισόδου κάθε επισκέπτη. Στο επάνω μέρος δεσπόζει μια ενότητα υποδοχής με σύντομη περιγραφή του σκοπού της πλατφόρμας, ενώ ακολουθεί μια μπάρα συγκεντρωτικών στατιστικών με το συνολικό πλήθος αναφορών, καθώς και αυτές που παραμένουν ανοικτές ή έχουν επιλυθεί (Εικόνα 5.1).

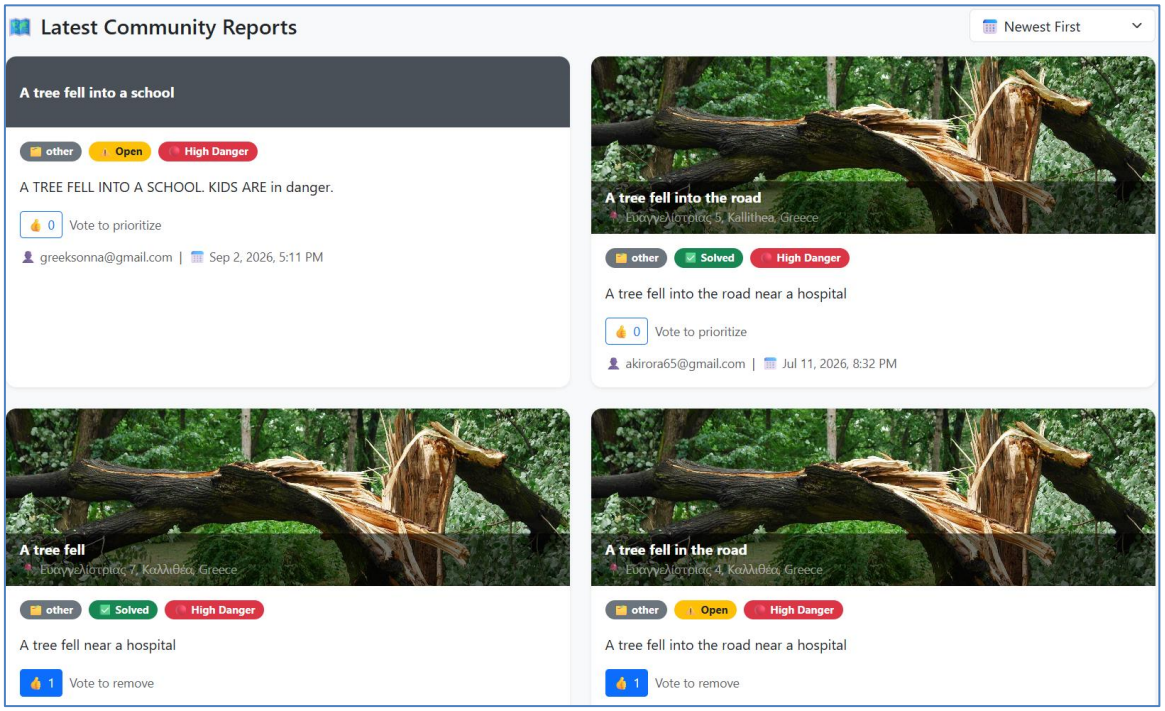


Εικόνα 5.1: Αρχική σελίδα.

Αμέσως μετά, προβάλλεται ένας διαδραστικός χάρτης (Google Maps) με πινέζες που αντιστοιχούν στις γεωαναφερόμενες αναφορές. Το χρώμα κάθε πινέζας υποδεικνύει το επίπεδο επικινδυνότητας κάθε αναφοράς (κόκκινο: πολύ επικίνδυνο· πορτοκαλί: μέτρια επικίνδυνο· πράσινο: ελάχιστα επικίνδυνο), ενώ με το πάτημα κάθε πινέζας εμφανίζονται οι λεπτομέρειές της. Στο κάτω μέρος της σελίδας, οι αναφορές παρουσιάζονται και σε μορφή καρτών, με δυνατότητα ταξινόμησης κατά ημερομηνία ή επίπεδο επικινδυνότητας (Εικόνες 5.2 & 5.3).



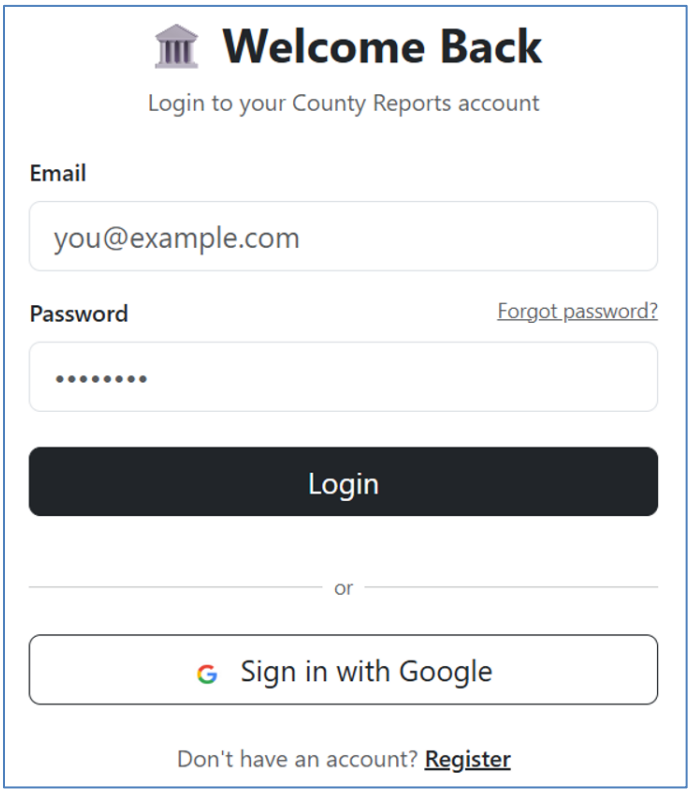
Εικόνα 5.2: Χάρτης αιτημάτων.



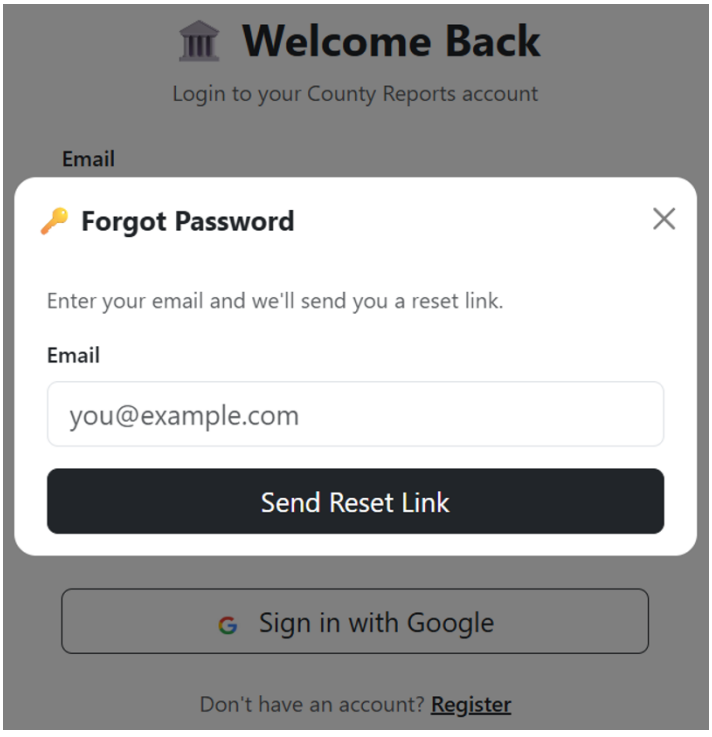
Εικόνα 5.3: Κάρτες αιτημάτων χρηστών.

5.2 Διαδικασία αυθεντικοποίησης

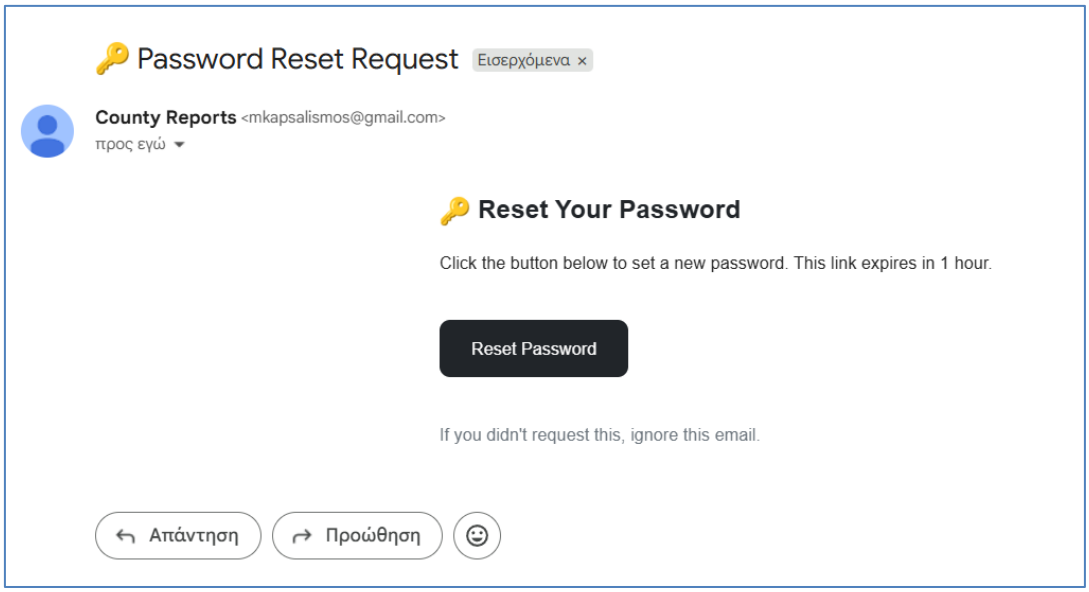
Η πλατφόρμα προσφέρει δύο τρόπους σύνδεσης: την κλασική φόρμα με email και κωδικό, καθώς και τη σύνδεση με έναν λογαριασμό Google (Εικόνα 5.4). Στη σελίδα σύνδεσης παρέχεται επιπλέον η δυνατότητα επαναφοράς κωδικού μέσω ενός αναδυόμενου παράθυρου (modal· Εικόνα 5.5), όπου ο χρήστης εισάγει το email του και λαμβάνει σύνδεσμο επαναφοράς (Εικόνα 5.6) και στην συνέχεια ανακατευθύνεται σε νέα σελίδα επαναφοράς κωδικού (Εικόνα 5.7). Τέλος, σε περίπτωση που ένας χρήστης σε αναστολή προσπαθήσει να συνδεθεί, εμφανίζεται κατάλληλο ενημερωτικό μήνυμα (Εικόνα 5.9).



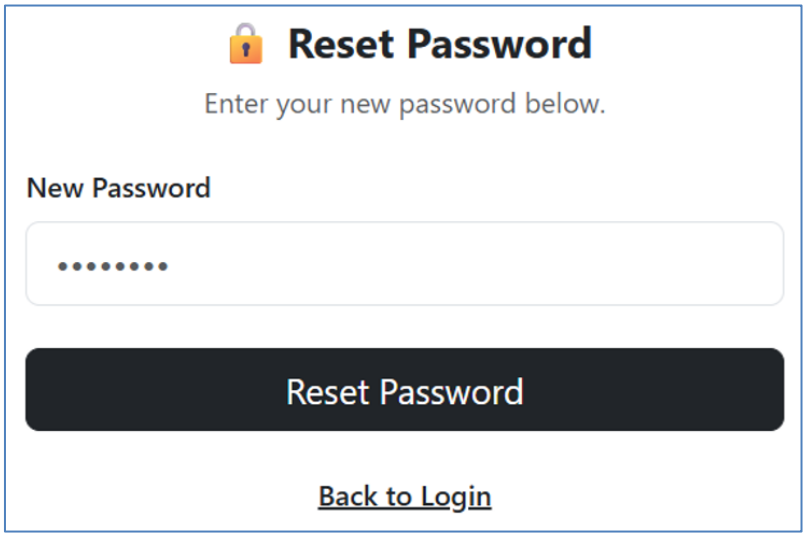
Εικόνα 5.4: Σελίδες σύνδεσης και εγγραφής.



Εικόνα 5.5: Τρόπος επαναφοράς κωδικού πρόσβασης.



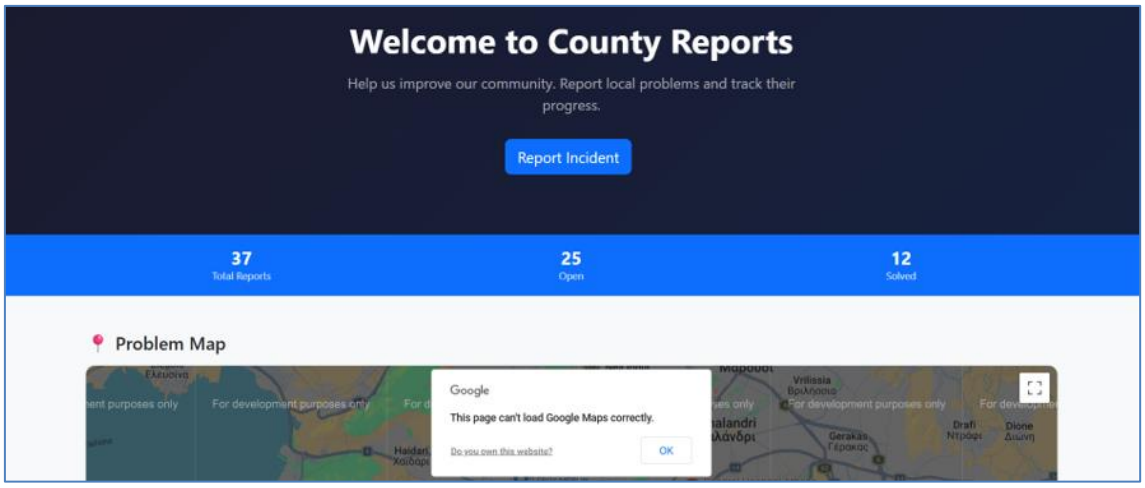
Εικόνα 5.6: Υπερσύνδεσμος επαναφοράς κωδικού πρόσβασης (password reset link).



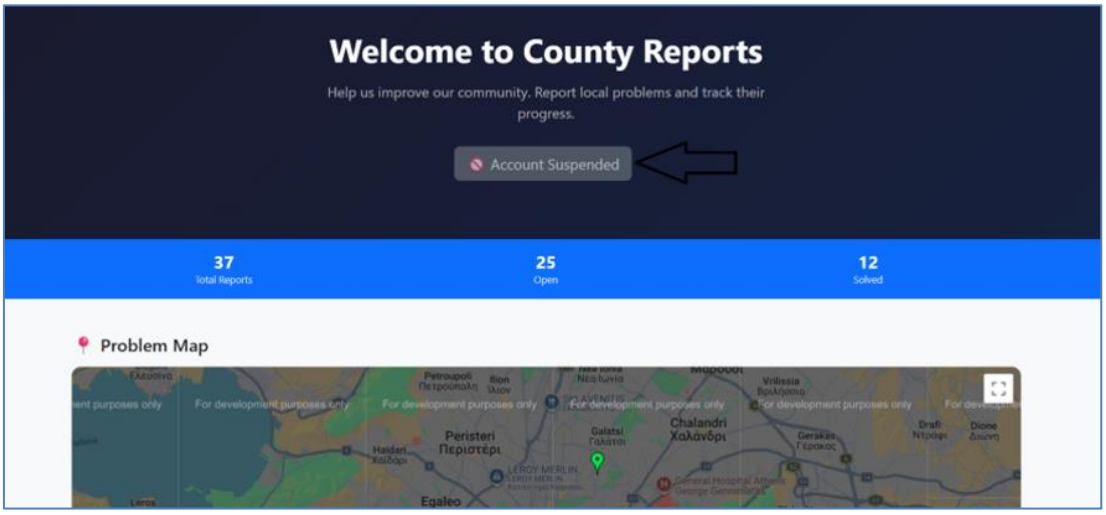
Εικόνα 5.7: Σελίδα επαναφοράς κωδικού πρόσβασης.

5.3 Περιβάλλον απλού χρήστη

Μετά τη σύνδεση, ο απλός χρήστης επιστρέφει στην αρχική σελίδα, όπου πλέον το κουμπί «Report Incident» τον οδηγεί στον πίνακα ελέγχου του (dashboard). Εκεί συναντά τη φόρμα υποβολής νέας αναφοράς, καθώς και τη συνολική λίστα των αναφορών της κοινότητας (Εικόνα 5.8). Στην περίπτωση που ο λογαριασμός του έχει τεθεί σε αναστολή, η φόρμα υποβολής αντικαθίσταται από ένα ενημερωτικό μήνυμα, ενώ το αντίστοιχο κουμπί στην αρχική σελίδα εμφανίζεται απενεργοποιημένο (Εικόνα 5.9).



Εικόνα 5.8: Αρχική σελίδα απλού χρήστη.

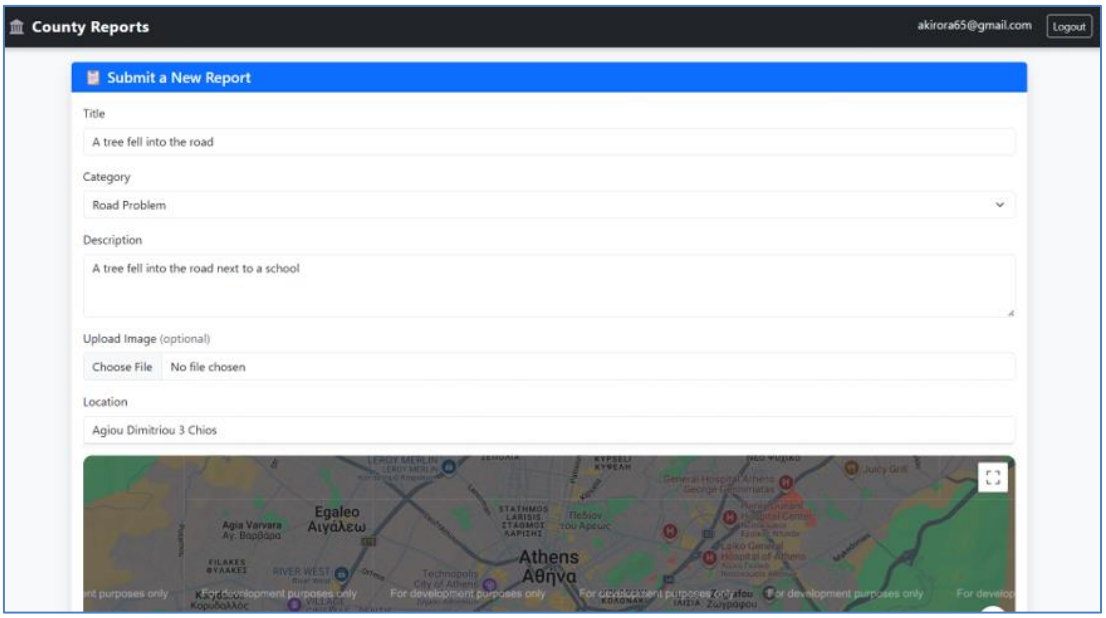


Εικόνα 5.9: Αρχική σελίδα χρήστη σε αναστολή.

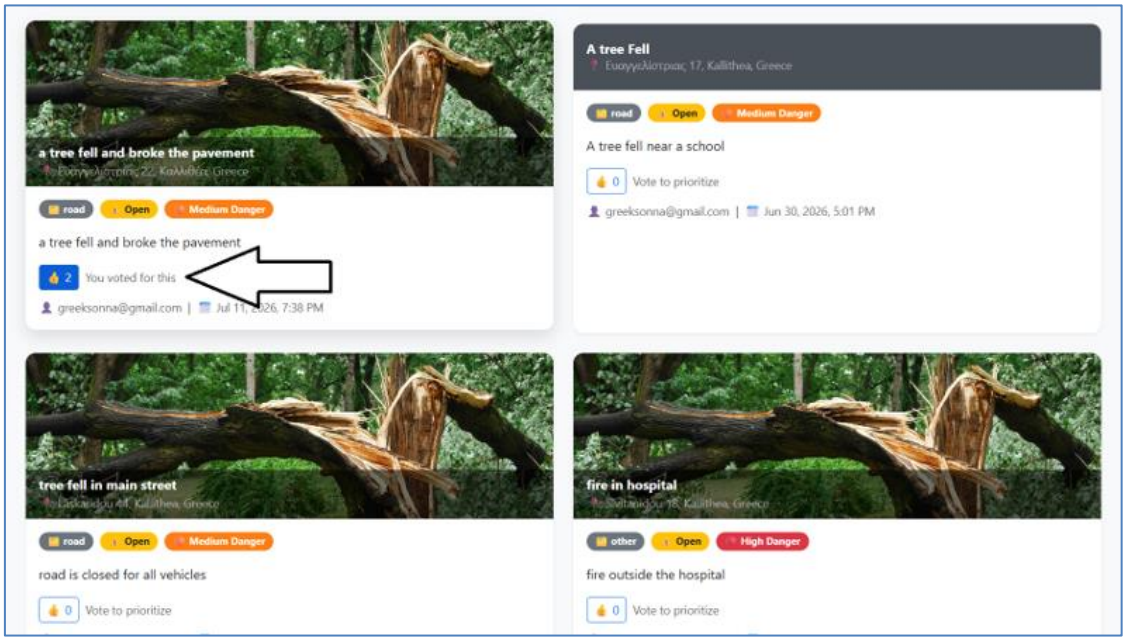
5.4 Υποβολή και ψηφοφορία/προτεραιοποίηση αναφορών

Η φόρμα υποβολής αναφορών περιλαμβάνει πεδία για τον τίτλο, την κατηγορία, την περιγραφή, την προαιρετική μεταφόρτωση εικόνας και τον προσδιορισμό της τοποθεσίας. Η τοποθεσία μπορεί να οριστεί είτε πληκτρολογώντας μια διεύθυνση με τη βοήθεια της αυτόματης συμπλήρωσης (Places Autocomplete), είτε με απευθείας τοποθέτηση μιας πινέζας στον διαδραστικό χάρτη (Εικόνα 5.10).

Μετά την υποβολή, η αναφορά εμφανίζεται άμεσα με την ένδειξη «Εκτίμηση...» στο επίπεδο επικινδυνότητας, η οποία ενημερώνεται αυτόματα μόλις ολοκληρωθεί η ανάλυση από το μοντέλο Τεχνητής Νοημοσύνης. Κάθε χρήστης μπορεί να προτεραιοποιήσει τις αναφορές ψηφίζοντας/πατώντας το κουμπί «👍». Η ψήφος καταχωρείται άμεσα και ο μετρητής ενημερώνεται χωρίς επαναφόρτωση της σελίδας, διατηρώντας τον χρήστη στο ίδιο σημείο. Κάθε χρήστης μπορεί να ψηφίσει μία φορά ανά αναφορά, ενώ μια δεύτερη επιλογή αναιρεί την ψήφο του (Εικόνα 5.11).



Εικόνα 5.10: Φόρμα υποβολής αναφοράς με τον διαδραστικό χάρτη.



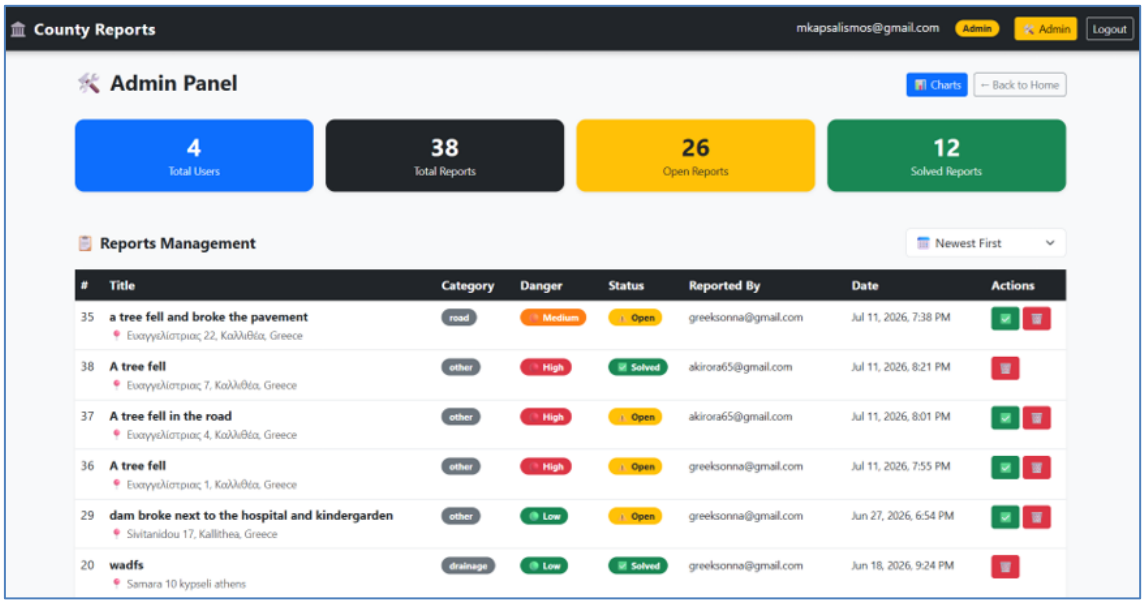
Εικόνα 5.11: Επισήμανση σπουδαιότητας προβλήματος/προβλημάτων μέσω προτεραιοποίησής τους από τον χρήστη.

5.5 Περιβάλλον διαχειριστή

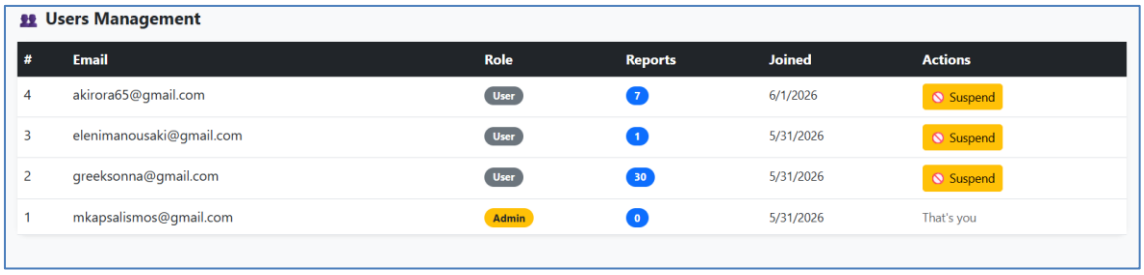
Το περιβάλλον του διαχειριστή αποτελεί το κεντρικό σύστημα ελέγχου της πλατφόρμας. Στο επάνω μέρος της σελίδας προβάλλονται τέσσερις κάρτες με τα βασικά συγκεντρωτικά στατιστικά: το πλήθος των χρηστών (Εικόνα 5.14), το σύνολο των αναφορών (Εικόνα 5.13), καθώς και τις ανοικτές και επιλυμένες αναφορές. Ακόμα ο διαχειριστής έχει τη δυνατότητα να προβάλλει κατηγοριοποιημένα το σύνολο των αιτημάτων βάση ημερομηνίας και βάση επικινδυνότητας (Εικόνα 5.12).

Εικόνα 5.12: Πίνακας διαχειριστή με γραφική αναπαράσταση πλήθους αναφορών, κατάστασης και επικινδυνότητας προβλημάτων.

Ακολουθεί ο πίνακας διαχείρισης αναφορών, όπου ο διαχειριστής μπορεί να δει κάθε μια με λεπτομέρειές όπως τον τίτλο της, τη διεύθυνση, την ημερομηνία καταχώρησης, αν έχει επιλυθεί ή όχι κ.ά. Κατά την επίλυση, αποστέλλεται αυτόματα ενημερωτικό email στον χρήστη/πολίτη που την υπέβαλε. Στη συνέχεια, ο πίνακας διαχείρισης χρηστών επιτρέπει την αναστολή ή την επαναφορά κακόβουλων λογαριασμών. Τέλος, μέσω ενός αναδυόμενου παραθύρου με γραφήματα (Chart.js), ο διαχειριστής έχει στη διάθεσή του οπτικοποιημένα στατιστικά για την κατάσταση, το επίπεδο επικινδυνότητας και την κατηγορία των αναφορών όσο και τι ποσοστό αυτών έχει επιλυθεί (Εικόνες 5.13 & 5.14).



Εικόνα 5.13: Πίνακας διαχείρισης αιτημάτων.



Εικόνα 5.14: Πίνακας διαχείρισης χρηστών.

6. Συμπεράσματα και Μελλοντικές Επεκτάσεις

Στο παρόν και τελευταίο κεφάλαιο της πτυχιακής εργασίας πραγματοποιείται μια συνολική ανασκόπηση του έργου που αναπτύχθηκε. Έχοντας παρουσιάσει αναλυτικά τον σχεδιασμό, την αρχιτεκτονική και τη διεπαφή χρήστη της πλατφόρμας, στόχος είναι να αξιολογήσει το τελικό αποτέλεσμα και να προτείνει κατευθύνσεις για περαιτέρω ανάπτυξη.

6.1 Συμπεράσματα

Ολοκληρώθηκε με επιτυχία η σχεδίαση και η ανάπτυξη μιας λειτουργικής πλατφόρμας συμμετοχικής διακυβέρνησης, αφιερωμένης στην αναφορά και επίλυση τοπικών προβλημάτων. Μέσα από την υλοποίησή της, επιτεύχθηκε ο πρωταρχικός στόχος που ήταν η δημιουργία ενός ενιαίου ψηφιακού καναλιού που γεφυρώνει το χάσμα μεταξύ των πολιτών και των αρμόδιων αρχών.

Τα βασικά συμπεράσματα που προκύπτουν από την ανάπτυξη του συστήματος συνοψίζονται στα παρακάτω:

- **Αξία της αρχιτεκτονικής MVC:** Η αυστηρή τήρηση του προτύπου Model-View-Controller αποδείχθηκε καθοριστική για τη συντηρησιμότητα του κώδικα. Ο σαφής διαχωρισμός των ευθυνών επέτρεψε την εύκολη προσθήκη νέων λειτουργιών και τη γρήγορη διόρθωση σφαλμάτων κατά τη διάρκεια της ανάπτυξης.
- **Πρακτική αξιοποίηση της Τεχνητής Νοημοσύνης:** Η ενσωμάτωση ενός τοπικά εκτελούμενου γλωσσικού μοντέλου για την αυτόματη ιεράρχηση των αναφορών απέδειξε ότι η Τεχνητή Νοημοσύνη μπορεί να προσφέρει ουσιαστική αποδοτικότητα ακόμη και σε εφαρμογές μικρής κλίμακας, διατηρώντας παράλληλα την ιδιωτικότητα των δεδομένων και χαμηλό το κόστος.
- **Σημασία της αποκρισιμότητας:** Η υλοποίηση χρονοβόρων διεργασιών (όπως η κλήση των εκάστοτε μοντέλων ή η αποστολή email) με ασύγχρονο τρόπο στο παρασκήνιο αποδείχθηκε κρίσιμης σημασίας για τη διατήρηση μιας γρήγορης και ευχάριστης εμπειρίας για τον χρήστη.
- **Συμμετοχικότητα και έλεγχος πρόσβασης:** Ο συνδυασμός του ελέγχου πρόσβασης βάσει ρόλων με τη συμμετοχική κατηγοριοποίηση των αιτημάτων μέσω ψηφοφορίας ανέδειξε τη δύναμη της συλλογικής γνώσης στην ανάδειξη των σημαντικότερων προβλημάτων της κοινότητας.

6.2 Μελλοντικές επεκτάσεις

Αν και το σύστημα που αναπτύχθηκε είναι πλήρως λειτουργικό και καλύπτει τις βασικές προδιαγραφές που τέθηκαν αρχικά, η φύση του λογισμικού επιτρέπει την απρόσκοπτη εξέλιξή του. Με γνώμονα τη βελτίωση της εμπειρίας του χρήστη και την επέκταση των δυνατοτήτων της πλατφόρμας, προτείνονται οι ακόλουθες μελλοντικές επεκτάσεις:

- **Σύστημα σχολίων και συζήτησης:** Η προσθήκη ενός συστήματος σχολίων κάτω από κάθε αναφορά, ώστε οι πολίτες/χρήστες να μπορούν να προσθέτουν επιπλέον πληροφορίες ή να επιβεβαιώνουν ή όχι ένα πρόβλημα.
- **Ανάπτυξη εφαρμογής για κινητά:** Παρόλο που η εφαρμογή έχει σχεδιαστή για να τρέχει και σε κινητές συσκευές εξίσου αποδοτικά, η δημιουργία μιας αυτόνομης εφαρμογής για Android και iOS θα αξιοποιούσε στο έπακρο τις δυνατότητες των smartphones (κάμερα, GPS, ειδοποιήσεις push notifications). Κατά αυτόν τον τρόπο ο χρήστης θα πρόσφερε ακόμα πιο άμεσες αναφορές από το σημείο του προβλήματος.
- **Διαβαθμισμένη κατάσταση αναφορών:** Ενδεχόμενη επέκταση των καταστάσεων μιας αναφοράς (π.χ. «υπό εξέταση», «σε εξέλιξη», «επιλύθηκε») θα προσέφερε μεγαλύτερη διαφάνεια στους πολίτες σχετικά με την πορεία του ζητήματός τους.

- **Προηγμένα εργαλεία διαχείρισης:** Σε ενδεχόμενη ενσωμάτωση της εφαρμογής σε μεγάλα αστικά κέντρα η προσθήκη ενός πίνακα ελέγχου με προηγμένα φίλτρα και αναζήτηση, καθώς και η αυτόματη ομαδοποίηση γειτονικών ή παρόμοιων αναφορών μέσω της Τεχνητής Νοημοσύνης, θα διευκόλυναν περαιτέρω το έργο των διαχειριστών.
- **Διασύνδεση με δημοτικές υπηρεσίες:** Η δημιουργία διακριτών λογαριασμών για τις αρμόδιες δημοτικές υπηρεσίες όπως η Υπηρεσία Πρασίνου, η Υπηρεσία Καθαριότητας, οι Τεχνικές Υπηρεσίες, κ.ά. θα έχουν τη δυνατότητα να ενημερώνονται και να αναλαμβάνουν απευθείας τις αναφορές που εμπίπτουν στην αρμοδιότητά τους.
- **Ενοποιημένο σύστημα διαχείρισης αιτημάτων(Ticketing system):** Η εφαρμογή ενός ενοποιημένου συστήματος διαχειρίσεις αιτημάτων θα αύξανε σημαντικά τη διαφάνεια καθώς η αρμοδιότητες τις εκάστοτε υπηρεσίας θα ήταν πλήρως διακριτές μειώνοντας τη σύγχυση ευθυνών και τη γραφειοκρατία. Τέλος, θα μπορούσαν να δημιουργηθούν μετρήσιμα στατιστικά στοιχεία για την αποδοτικότητα, τόσο των υπηρεσιών όσο και των εργαζομένων καθώς η ύπαρξη προσωποποιημένων προφίλ θα ενίσχυε την αξιολόγηση υπηρεσιών και υπαλλήλων.

Πίνακας Ορολογιών

Model - View - Controller: Πρότυπο σχεδίασης λογισμικού (Μοντέλο - Εμφάνιση - Ελεγκτής)
Backend: Πλατφόρμα διαχείρισης
Body-parser: Ενδιάμεσο λογισμικό μεταφοράς δεδομένων
Frontend: Πλατφόρμα διεπαφής με τον χρήστη
Dashboard: Πίνακας ελέγχου
Prompt: Εντολή
SQL: Δομημένη γλώσσα ερωτημάτων
LLM: Μεγάλα γλωσσικά μοντέλα
EJS: Εφαρμοσμένα πρότυπα γλώσσας javascript
Latitude: Γεωγραφικό πλάτος
Longtitude: Γεωγραφικό μήκος

Βιβλιογραφικές Αναφορές

1. Άρθρα - Βιβλία

Bhargava, H., Chaturvedi, R., & Vashistha, R. (2026). AI-driven decision support for credit risk prediction using MVC based hybrid recommender system. *Journal of Finance, Economics and Data Analytics, 1*(1), 64-74. https://bit.ly/4xjrbIG

Chrysafiadi, K., Kontogianni, A., Virvou, M., & Alepis, E. (2025). Enhancing user experience in Smart Tourism via Fuzzy Logic-Based personalization. *Mathematics* 2025, *13*(5), 846. https://doi.org/10.3390/math13050846

Dey, T. (2011). Comparative analysis on modeling and implementing with MVC architecture. *International Conference on Web Services Computing (ICWSC). Proceedings published by International Journal of Computer Applications (IJCA).* https://bit.ly/4r8jUdx

Kontogianni, A., & Alepis, E. (2026). Empowering smart tourism with Large Language Models. In G.A. Tsihrintzis, M. Virvou, V. Marinakis, & L.C. Jain (eds), *Human-smart city interactions and user-citizen experiences. Learning and Analytics in intelligent systems, 58*, 101-129. Springer, Cham. https://doi.org/10.1007/978-3-032-06364-9_6

Kontogianni, A., Alepis, E., Virvou, M., & Patsakis, C. (2024). Artificial Intelligence in Smart Tourism. In *Smart Tourism-The Impact of Artificial Intelligence and Blockchain. Intelligent Systems Reference Library, 249*, 75-85. Springer, Cham. https://doi.org/10.1007/978-3-031-50883-7_5

Kontogianni, A., & Alepis, E. (2020). Smart tourism: State of the art and literature review for the last six years. *Array, 6*, 100020. https://doi.org/10.1016/j.array.2020.100020

Kontogianni, A., Chrysafiadi, K., Virvou, M., & Alepis, E. (2026). From Wikidata to smart tourism: A reproducible pipeline based on AI and Fuzzy Logic for interpretable multi-category classification of points of interest. *Mathematics*, *14*(12), 2227. https://doi.org/10.3390/math14122227

Phillips-Wren G, & Virvou M. (2025). Issues and trends in generative AI technologies for decision making. *Intelligent Decision Technologies,* 19(2), 574-584. doi:10.1177/18724981251320551

Selfa, D.M., Carrillo, M., & Del Rocio Boone, M. (2006). A Database and web application based on MVC architecture. *16th International Conference on Electronics, Communications and Computers (CONIELECOMP'06)*, Puebla, Mexico, 48-48. doi: 10.1109/CONIELECOMP.2006.6

Tsihrintzis, G.A., Virvou, M., Marinakis, V., & Jain, L.C. (2026). Introduction to human-smart city interactions and user-citizen experiences. In G.A. Tsihrintzis, M. Virvou, V. Marinakis, & L.C. Jain (eds), *Human-smart city interactions and user-citizen experiences. Learning and Analytics in intelligent systems, 58*, 1-13. Springer, Cham. https://doi.org/10.1007/978-3-032-06364-9_1

Zanella, A., Bui, N., Castellani, A., Vangelista, L., & Zorzi, M. (2014). Internet of things for smart cities. *IEEE Internet of Things Journal*, 1(1), 22-32. doi: 10.1109/JIOT.2014.2306328

Zissis, D., Lekkas, D., & Papadopoulou, E. A. (2009). Competent electronic participation channels in electronic democracy. *Electronic Journal of e-Government, 7*(2), 195-208. https://academic-publishing.org/index.php/ejeg/article/view/502

Αναστασόπουλος, Α. (2022). *Αστική βιωσιμότητα και έξυπνες πόλεις*. ΕΑΠ, MSc Thesis. https://apothesis.eap.gr/archive/item/170489

Δουληγέρης, Χ., Μαυροπόδη, Ρ., & Κοπανάκη, Ε. (2013). *Τεχνολογίες διαδικτύου αρχές λειτουργίας και προγραμματισμός στο διαδίκτυο.* Εκδόσεις Νέων Τεχνολογιών.

Παπαγιάννης, Σ. Α. (2026). *Από την ηλεκτρονική διακυβέρνηση στην έξυπνη πόλη: Ψηφιακός μετασχηματισμός και έξυπνη διακυβέρνηση στον Δήμο Θέρμης*. ΕΑΠ, MSc Thesis. https://apothesis.eap.gr/archive/item/241338

2. Διαδικτυακές πηγές

Bootstrap Team (2024). *Bootstrap documentation*. https://getbootstrap.com/docs/

Chart.js Contributors (2024). *Chart.js documentation*. https://www.chartjs.org/docs/latest/

Draw.io (diagrams.net). *Flowchart maker & online diagram software.* https://www.drawio.com/

EJS Project (2024). *EJS - Embedded JavaScript templating*. https://ejs.co/

Express.js Team (2024α). *Express.js guide*. https://expressjs.com/

Express.js Team (2024β). *Express-session middleware*. https://www.npmjs.com/package/express-session

Google Developers (2024α). *Maps JavaScript API documentation*. https://developers.google.com/maps/documentation/javascript

Google Developers (2024β). *Google Maps platform: Maps JavaScript API.* https://developers.google.com/maps/documentation/javascript

Node.js Foundation (2024). *Node.js documentation*. https://nodejs.org/en/docs

Ollama Team (2024). *Ollama documentation*. https://ollama.com

Passport.js Project (2024). *Passport.js documentation*. https://www.passportjs.org/

PostgreSQL Global Development Group (2024). *PostgreSQL documentation*. https://www.postgresql.org/docs/

Provos, N., & Mazieres, D. (1999). *Bcrypt adaptive hashing algorithm*. Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference. https://www.usenix.org/legacy/event/usenix99/provos/provos.pdf

3. Βιβλιοθήκες της εφαρμογής

Bootstrap Team. *Bootstrap: The most popular HTML, CSS, and JS library in the world.* https://getbootstrap.com/

Chart.js. Chart.js. *Simple yet flexible JavaScript charting library.* https://www.chartjs.org/

EJS. *EJS - Embedded JavaScript templating.* https://ejs.co/

Nodemailer. *Nodemailer: Send e-mails from Node.js.* https://nodemailer.com/

npm. Bcrypt. *A library to help you hash passwords.* https://www.npmjs.com/package/bcrypt

npm. Dotenv. *Loads environment variables from a .env file.* https://www.npmjs.com/package/dotenv

npm. express-session. *Session middleware for Express.* https://www.npmjs.com/package/express-session

npm. Multer. *Node.js middleware for handling multipart/form-data.* https://www.npmjs.com/package/multer

npm. passport-google-oauth2. *Google OAuth 2.0 authentication strategy.* https://www.npmjs.com/package/passport-google-oauth2

npm. passport-local. *Local username and password authentication strategy.* https://www.npmjs.com/package/passport-local

npm. Pg. *Non-blocking PostgreSQL client for Node.js.* https://www.npmjs.com/package/pg

OpenJS Foundation. *Express - Node.js web application framework.* https://expressjs.com/

Passport.js. Passport. *Simple, unobtrusive authentication for Node.js.* https://www.passportjs.org/