Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών
«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Ανάπτυξη ενός ολοκληρωμένου συστήματος Αυτοματοποιημένης Αναγνώρισης Πινακίδων Κυκλοφορίας Οχημάτων ως ενσωματωμένη λύση Μηχανικής Μάθησης σε εφαρμογή Android** **-** **Automatic License Plate Recognition system development as an Android on-device ML application**
Ονοματεπώνυμο Φοιτητή	**ΛΕΟΝΤΑΡΙΔΗΣ Σπυρίδων**
Πατρώνυμο	**Βασίλειος**
Αριθμός Μητρώου	**ΜΠΠΛ19030**
Επιβλέπων	**Ευθύμιος ΑΛΕΠΗΣ, Αναπληρωτής Καθηγητής**

Ημερομηνία Παράδοσης **Οκτώβριος 2022**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Ευθύμιος Αλέπης Αναπληρωτής Καθηγητής	Μαρία Βίρβου Καθηγήτρια	Κωνσταντίνος Πατσάκης Αναπληρωτής Καθηγητής

Ευχαριστίες

Με την ευκαιρία που μου δίνεται ολοκληρώνοντας την παρούσα μεταπτυχιακή διατριβή και κατ' επέκταση, το πρόγραμμα μεταπτυχιακών σπουδών «Πληροφορική» του Πανεπιστημίου Πειραιώς, θα ήθελα να ευχαριστήσω από καρδιάς, όλους τους καθηγητές του προγράμματος για την ανιδιοτελή τους στάση, τον επαγγελματισμό και το πνεύμα συνεργασίας που επέδειξαν, παρά τις αντιξοότητες που χρειάστηκε να αντιμετωπίσουμε. Ιδιαίτερα όμως, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κ. Ευθύμιο Αλέπη, για τη μεθοδικότητα, την άρτια επιστημονική του καθοδήγηση και τον πολύτιμο χρόνο του, που αγόγγυστα μου αφιέρωσε, καθ' όλης τη διάρκεια της συνεργασίας μας.

Τέλος, δεν θα μπορούσα να μην ευχαριστήσω θερμά τη σύντροφό μου, που χωρίς τη συμβολή της, τίποτα απ' όλα αυτά δεν θα ήταν εφικτό.

Περίληψη

Στο πλαίσιο της παρούσας μεταπτυχιακής διατριβής, αναπτύχθηκε ένα ολοκληρωμένο σύστημα αυτοματοποιημένης αναγνώρισης πινακίδων κυκλοφορίας (ALPR), το οποίο έχει ενσωματωθεί σε εφαρμογή για το λειτουργικό σύστημα Android. Μέσω της εφαρμογής, οι χρήστες έχουν τη δυνατότητα να δηλώνουν τους αριθμούς κυκλοφορίας των οχημάτων που τους ενδιαφέρουν και αναζητούν για οποιονδήποτε λόγο (κλοπή, τροχαίο με εγκατάλειψη κ.α.), να ειδοποιούνται σε περίπτωση που εντοπιστεί κάποιο από αυτά, να σαρώνουν σε πραγματικό χρόνο τις πινακίδες κυκλοφορίας των διερχόμενων οχημάτων, καθώς και να ανατρέχουν στο ιστορικό των σαρώσεών τους, χρησιμοποιώντας χωροχρονικά κριτήρια. Επιπλέον, προβλέπεται η δυνατότητα επεξεργασίας και ανάλυσης των δεδομένων που έχουν συλλεχθεί, με σκοπό, την εξαγωγή συμπερασμάτων, τα οποία και θα επιτρέπουν την αποτελεσματικότερη κατανομή των διαθέσιμων πόρων (ανθρώπινο δυναμικό και συσκευές), αλλά και την πρόβλεψη πιθανότερων θέσεων μελλοντικών συμβάντων. Η εφαρμογή αναπτύχθηκε σε γλώσσες προγραμματισμού Java και Kotlin, ενώ παράλληλα, τεχνολογίες όπως, on-device ML (μηχανική μάθηση) της Google, χρησιμοποιήθηκαν για τον εντοπισμό και την αναγνώριση των πινακίδων κυκλοφορίας. Τέλος, η εφαρμογή υποστηρίζεται από μία NoSQL βάση δεδομένων (MongoDB) και κατάλληλα Web Services, που αναπτύχθηκαν σε περιβάλλον Java Spring Boot.

Λέξεις κλειδιά: ALPR, Java, Kotlin, on-device ML, Google Maps API, RESTfull Web Services, APIs, NoSQL, MongoDB, Java Spring Boot Framework, Android Studio, IntelliJ, TensorFlow, DBSCAN, QGIS, transfer learning

ABSTRACT

In the context of this master's thesis, an integrated system of automated license plate recognition (ALPR) has been developed and integrated into an application for the Android operating system. Through the application, users can declare the registration numbers of vehicles they are interested in and are looking for any reason (theft, hit and run after car accident, etc.), to be notified if any of them are detected, to scan in real-time the license plates of passing vehicles, as well as refer back to their scan logs, using spatio-temporal criteria. In addition, capability of processing and analyzing the collected data is foreseen, aiming drawing conclusions, which will allow the more efficient allocation of available resources (human resources and devices), and the prediction of more likely positions of future events. The application was developed in Java and Kotlin programming languages, while at the same time, technologies such as Google's on-device ML (machine learning) were used to detect and recognize license plates. Finally, the application is supported by a NoSQL database (MongoDB) and appropriate Web Services, developed in Java Spring Boot environment.

Keywords: ALPR, Java, Kotlin, on-device ML, Google Maps API, RESTfull Web Services, APIs, NoSQL, MongoDB, Java Spring Boot Framework, Android Studio, IntelliJ, TensorFlow, DBSCAN, QGIS, transfer learning

Πίνακας Περιεχομένων

Εισαγωγή

Το αυτοκίνητο είναι αναμφίβολα μία από τις σημαντικότερες και πιο καθοριστικές εφευρέσεις του 19ου αιώνα. Η έλευσή του, άλλαξε για πάντα την καθημερινότητά μας και κυριολεκτικά, διαμόρφωσε τις σύγχρονες κοινωνίες. Από την κυκλοφορία του πρώτου αυτοκίνητου με βενζινοκινητήρα εσωτερικής καύσης, του Karl Benz το 1885 και την πρώτη μεγάλης κλίμακας παραγωγή του θρυλικού Ford Model T από τον Henry Ford το 1908, μέχρι την ανάπτυξη και την κυκλοφορία των πρώτων αυτόνομων οχημάτων στις αρχές του 21ου αιώνα, μεσολάβησαν πολλά στάδια εξέλιξης και τελικά, τα αυτοκινούμενα οχήματα (αυτοκίνητα, δίκυκλα, τρίτροχα κ.α.) κυριάρχησαν σε όλο το εύρος της ανθρώπινης δραστηριότητας.

Η ευκολία που προσφέρουν, η ταχύτητα και η ποικιλομορφία τους (mini, SUV, 4x4, οικογενειακά, sport, coupe, cabrio, RV) είναι ορισμένα από τα πλεονεκτήματα που στάθηκαν ικανά να εκτοπίσουν, μέσα σε λιγότερο από έναν αιώνα, τα παραδοσιακά μέσα μετακίνησης (ζώα, ιππήλατες άμαξες κ.λ.π.), ικανοποιώντας ανάγκες ζωτικής σημασίας που έως τότε, ήταν δύσκολο ή ακόμη και αδύνατο να ικανοποιηθούν επαρκώς, ενώ παράλληλα, δημιούργησαν νέες, πιο σύνθετες. Βρίσκοντας λοιπόν άμεσα εφαρμογή στις μετακινήσεις (αυτοκίνητα ιδιωτικής χρήσης, Μ.Μ.Μ.) και την παραγωγή (τρακτέρ, φορτηγά, ρυμουλκούμενα οχήματα κ.α.), πολύ σύντομα τα μηχανοκίνητα, πλέον οχήματα, μεταπήδησαν στον αθλητισμό (Formula 1, Rally, NASCAR κ.α.) και τελικά στο διάστημα[1] (αυτοκινούμενα μη επανδρωμένα οχήματα κ.α.).

Μαζί όμως με τα πλεονεκτήματα που ακολούθησαν της εφεύρεσής τους, η εξάπλωση των αυτοκινούμενων οχημάτων ανέδειξε και ζητήματα που μέχρι τότε ήταν λίγο έως πολύ αόριστα ή έστω περιορισμένα και εύκολα διαχειρίσιμα. Ζητήματα όπως, η θέσπιση κανόνων κυκλοφορίας[2], τα όρια ταχύτητας, η αντιμετώπιση και η πρόληψη τροχαίων ατυχημάτων[3], η πιστοποίηση ικανότητας οδήγησης, η οδική σήμανση και άλλα παρόμοια που σήμερα θεωρούμε συνυφασμένα με την έννοια των αυτοκινήτων, αποτέλεσαν αντικείμενο προβληματισμού και συζητήσεων που χρειάστηκε να ρυθμιστούν και να πλαισιωθούν από κανόνες.

Ωστόσο, αυτές οι ρυθμίσεις απαιτούσαν κάτι που, τουλάχιστον τα πρώτα χρόνια κυκλοφορίας των αυτοκινήτων, δεν ήταν αναγκαίο. Απαιτούσαν τη δυνατότητα αναγνώρισης ενός εκάστου των οχημάτων που κυκλοφορούσαν. Έτσι λοιπόν, παρόλο που όταν πρωτοεμφανίστηκαν τα αυτοκίνητα, η φυσική τους και μόνο περιγραφή (μάρκα, μοντέλο, χρώμα) ήταν αρκετή για την ταυτοποίησή τους, με την πάροδο του χρόνου, την εξέλιξη της τεχνολογίας και την αύξηση της μαζικής παραγωγής[4], η αναγνώρισή τους έγινε ιδιαίτερα δυσχερής. Χρειάστηκε, λοιπόν, ένας πιο αποτελεσματικός και καθολικός τρόπος για την αναγνώριση και την ταξινόμηση των οχημάτων.

Η λύση προέκυψε μέσω της υιοθέτησης των πινακίδων κυκλοφορίας[5]. Οι πρώτες πινακίδες κυκλοφορίας εμφανίστηκαν στη Γαλλία το 1893. Το παράδειγμα της Γαλλίας ακολούθησε η Γερμανία, με τις πρώτες πινακίδες να εμφανίζονται γύρω στο 1896, ενώ ήδη, από το 1901, η πολιτεία της Νέας Υόρκης απαιτεί πινακίδες τις οποίες, ωστόσο, μέχρι το 1909 οι πολίτες έπρεπε να τις φτιάχνουν μόνοι τους, αφού δεν υπήρχε επίσημος κατασκευαστής. Σε παγκόσμιο επίπεδο πλέον, η έκδοση πινακίδων κυκλοφορίας είναι μια κεντρικοποιημένη διαδικασία, που καθορίζεται από κρατικούς φορείς και αποτελεί προϋπόθεση για τη νόμιμη κυκλοφορία των αυτοκινούμενων οχημάτων, ενώ οτιδήποτε αφορά σε αυτά, οι πινακίδες κυκλοφορίας αποτελούν σημείο αναφοράς.

---

1 https://www.protagon.gr/epikairotita/to-aftokinito-tou-elon-mask-pataei-gkazi-gia-ton-ari-44341562482

2 Ο πρώτος Κώδικας Οδικής Κυκλοφορίας αφορούσε στα ζώα και στην Ελλάδα χρονολογείται πίσω στα χρόνια του βασιλιά Όθωνα, το μακρινό 1837.

3 Το πρώτο τροχαίο ατύχημα που καταγράφηκε στην Ελλάδα ήταν το 1907 και μέχρι την θέσπιση των κανόνων κυκλοφορίας, είχαν καταγραφεί ήδη περισσότερα από 7000 τροχαία ατυχήματα - https://www.newsbeast.gr/weekend/arthro/4461388/to-adoxo-telos-poy-eiche-to-proto-aytokinito-poy-irthe-stin-ellada

4 Αρκεί να αναφέρουμε ότι το πρώτο αυτοκίνητο στην Ελλάδα ήρθε το 1897 και μέχρι το 1907 ο αριθμός τους είχε φτάσει στα επτά.

5 https://en.wikipedia.org/wiki/Vehicle_registration_plate

Αν και η λύση που προτάθηκε, αποδείχθηκε αποτελεσματική για την ταχεία αναγνώριση των οχημάτων ανάμεσα σε άλλα παρόμοια, η σημερινή πραγματικότητα, με την συνεχώς αυξανόμενη συρροή πληθυσμού στα μεγάλα αστικά κέντρα και την αντίστοιχη αύξηση του συνολικού αριθμού των οχημάτων που συσσωρεύονται σε αυτά[6], εισήγαγε μια ακόμη παράμετρο, πιο σύνθετη και απαιτητική, αυτή του εντοπισμού. Έτσι λοιπόν, σε μια εποχή που χαρακτηρίζεται από ταχύτητα, μεταβλητότητα και τεράστιο όγκο πληροφοριών, η αναζήτηση ενός συγκεκριμένου οχήματος, αποτελεί μια ιδιαιτέρως δύσκολη και χρονοβόρα διαδικασία, η οποία, σε αρκετές περιπτώσεις, μπορεί να αποδειχθεί ατελέσφορη.

Δυστυχώς, οι παραδοσιακές προσεγγίσεις επίλυσης του προβλήματος, δεν είναι πλέον αποδοτικές. Αυτό οφείλεται, κυρίως, στους περιορισμένους πόρους (ανθρώπινο δυναμικό) που μπορούν να διατεθούν, σε σύγκριση με την τεράστια έκταση που απαιτείται να καλυφθεί σε σύντομο χρονικό διάστημα. Θέλοντας να αποσαφηνίσουμε την παραπάνω δήλωση, ας αναλογιστούμε την περίπτωση όπου δεχόμαστε ένα τηλεφώνημα από κάποιον φίλο μας που κατοικεί σε κάποια πυκνοκατοικημένη περιοχή της Αττικής και μας ενημερώνει ότι πριν από λίγη ώρα του έκλεψαν το αυτοκίνητο. Φυσικά και θέλουμε να βοηθήσουμε τον άτυχο φίλο μας, αλλά πώς; Αυτό που μπορούμε να κάνουμε είναι να ξεκινήσουμε να αναζητούμε το αυτοκίνητό του σε μέρη της Αττικής όπου πιθανολογούμε ότι θα το εντοπίσουμε, ελπίζοντας να σταθούμε τυχεροί. Εάν ωστόσο θέλουμε να μεγιστοποιήσουμε τις πιθανότητες μας, θα πρέπει να περάσουμε από όλους τους δρόμους της Αττικής, έχοντας τεταμένη την προσοχή μας για οχήματα τα οποία προσομοιάζουν (μάρκα, μοντέλο, χρώμα) στου φίλου μας και για κάθε ένα από αυτά, να ελέγχουμε αν η πινακίδα κυκλοφορίας αντιστοιχεί σε αυτή που αναζητούμε. Συνάμα, αυτή η διαδικασία θα πρέπει να γίνει σχετικά σύντομα, καθώς ο χρόνος που περνά μειώνει τις πιθανότητες ανεύρεσης. Σκεφτείτε τώρα να θέλουμε να βοηθήσουμε περισσότερους από έναν φίλους μας. Για κάθε έναν από αυτούς, χρειάζεται να θυμόμαστε τη μάρκα, το μοντέλο, το χρώμα και φυσικά τον αριθμό κυκλοφορίας του οχήματός του, ώστε να καταφέρουμε τελικά να το αναγνωρίσουμε. Επιπλέον, αναλογιστείτε την εκδοχή όπου κάθε ένας τους, μας ενημερώνει για την ατυχία που τον βρήκε σε διαφορετική ημερομηνία από κάθε άλλον. Αυτό πρακτικά σημαίνει ότι, κάθε φορά που ένας φίλος μας μας ειδοποιεί ότι του έκλεψαν το αυτοκίνητο, πρέπει να ξεκινάμε την διαδικασία που περιγράψαμε παραπάνω από την αρχή, ώστε να μεγιστοποιήσουμε τις πιθανότητες ανεύρεσης για κάθε έναν ξεχωριστά.

Γίνεται επομένως σαφές ότι, η διαδικασία εντοπισμού και αναγνώρισης ενός οχήματος είναι χρονοβόρα, ενώ απαιτεί από εμάς να απομνημονεύσουμε και να είμαστε σε θέση να επεξεργαστούμε, ταυτόχρονα, μεγάλο όγκο πληροφοριών με τον μέγιστο δυνατό ρυθμό. Και όλα αυτά, μονάχα για την περίπτωση της κλοπής ενός οχήματος. Η πολυπλοκότητα και η δυσκολία της διαδικασία αυξάνεται εκθετικά, αν επεκτείνουμε την ανάγκη εντοπισμού κάποιου οχήματος και σε περιπτώσεις όπως, η εμπλοκή ενός οχήματος σε τροχαίο ατύχημα ή η ακινητοποίηση ενός οχήματος του οποίου ο οδηγός έχει καταναλώσει μεγάλη ποσότητα αλκοόλ, σε σημείο που είναι επικίνδυνος για τους υπόλοιπους χρήστες του οδικού δικτύου.

Γι’ αυτό λοιπόν, οι σύγχρονες προσεγγίσεις στράφηκαν στον τομέα της τεχνολογία και της επιστήμης των υπολογιστών, αναζητώντας λύσεις, που θα κατορθώσουν να αξιοποιήσουν αποτελεσματικά τους διαθέσιμους πόρους (ανθρώπινο δυναμικό, υπολογιστές κ.α.) και θα προσφέρουν ταχύτητα, ευελιξία και δυνατότητα ταυτόχρονης διαχείρισης μεγάλου όγκου πληροφοριών.

Απόρροια αυτής της προσπάθειας είναι η ανάπτυξη των συστημάτων αυτόματης αναγνώρισης πινακίδων κυκλοφορίας (Automatic License Plates Recognition – εφεξής ALPR)[7]. Πρόκειται για ολοκληρωμένα συστήματα, που εκμεταλλεύονται την τεχνολογία της οπτικής

[6] Ο συνολικός αριθμός των αυτοκινούμενων οχημάτων που βρίσκονται πλέον σε κυκλοφορία, είναι συγκρίσιμος με τον συνολικό αριθμό των ανθρώπων της γης. Ενδεικτικά αναφέρεται ότι, για το έτος 2022 και μόνο για την περίπτωση των αυτοκινήτων, ο συνολικός τους αριθμός εκτιμάται στα 1,45 δισεκατομμύρια. Το πιο εντυπωσιακότερο στοιχείο, βέβαια είναι πως, ο αριθμός αυτός, το 1996, αντιστοιχούσε μόλις στα 670 εκατομμύρια.

[7] Μπορεί κανείς να τα συναντήσει και ως APNR.

αναγνώρισης χαρακτήρων (OCR)[8], προκειμένου να αναγνωρίσουν τις πινακίδες κυκλοφορίας των οχημάτων μέσω εικόνων. Χρησιμοποιούν εικόνες προερχόμενες από ήδη εγκατεστημένα κλειστά κυκλώματα εικονοσκόπησης, κάμερες κυκλοφορίας ή ακόμη και ειδικά σχεδιασμένες κάμερες που έχουν αναπτυχθεί αποκλειστικά γι' αυτόν τον σκοπό. Παράγοντες όπως, ο υφιστάμενος φωτισμός, οι καιρικές συνθήκες (βροχή, υγρασία κλπ), η ποικιλομορφία των πινακίδων κυκλοφορίας και η φυσική παραμόρφωση των εικόνων λόγω των φακών που χρησιμοποιούν οι κάμερες, επηρεάζουν την λειτουργία και την αποτελεσματικότητά τους, ενώ συνιστούν προκλήσεις για την τεχνολογία και την επιστήμη των υπολογιστών.

Τέτοιου είδους συστήματα χρησιμοποιούνται, κυρίως, από τις αρχές επιβολής του νόμου και άλλους Δημόσιους φορείς ανά την υφήλιο, ενώ εγκαθίστανται σε σταθερά σημεία, όπως οι συνοριακοί σταθμοί, τα διόδια ή οι χώροι στάθμευσης, και σε ειδικά διαμορφωμένα οχήματα[9], συνήθως περιπολικά, προκειμένου για την διαχείριση της κυκλοφορίας, την αναγνώριση αναζητούμενων οχημάτων, την διαχείριση των χώρων στάθμευσης των μεγάλων αστικών κέντρων, την βεβαίωση τροχονομικών παραβάσεων κ.α.. Το υψηλό κόστος εγκατάστασης και συντήρησης, καθώς και η περιορισμένη φορητότητα του απαραίτητου εξοπλισμού, αποτελούν βασικά εμπόδια στην εκμετάλλευση της χρησιμότητάς τους από το ευρύτερο κοινό.

Η παρούσα διατριβή, στοχεύει στην άρση αυτών των περιορισμών, επιδιώκοντας έτσι, την περαιτέρω αξιοποίηση των διαθέσιμων πόρων και την μεγιστοποίηση του οφέλους που μπορεί να προκύψει από τα ALPR συστήματα. Με γνώμονα όσα ήδη αναφέρθηκαν, σχεδιάστηκε και αναπτύχθηκε το **MobileLPR**, μια ολοκληρωμένη λύση ALPR, που απευθύνεται σε χρήστες του λειτουργικού συστήματος Android και αποσκοπεί, στην ενσωμάτωση των χαρακτηριστικών πλεονεκτημάτων που αυτό προσφέρει, όπως η ευελιξία και η ευκολία μιας εφαρμογής, αλλά και η φορητότητα των κινητών συσκευών. Καταλυτικό παράγοντα, βέβαια, στην συγκεκριμένη επιλογή, διαδραμάτισε και η απήχηση που έχουν οι συσκευές Android, όπως τα έξυπνα κινητά τηλέφωνα (smartphones), στον παγκόσμιο πληθυσμό, γεγονός που οδήγησε στην ραγδαία εξάπλωσή τους και τη σύντομη εξοικείωση του πληθυσμού στη χρήση τους.

Σε αντιδιαστολή με τις υπάρχουσες υλοποιήσεις, το MobileLPR συνοδεύεται από μία σειρά λειτουργιών, που επιχειρούν να επεκτείνουν την χρησιμότητα των συστημάτων ALPR στο ευρύτερο κοινό, δημιουργώντας έτσι, ένα διευρυμένο δίκτυο χρηστών, που θα αξιοποιεί το μέγιστο δυνατό υποσύνολο των διαθέσιμων πόρων (χρήστες και διαθέσιμες συσκευές Android).

Κλείνοντας, είναι σημαντικό να αναφερθεί ότι, το MobileLPR, είναι σε θέση να ανταποκρίνεται σε ένα ευρύ φάσμα περιπτώσεων χρήσης. Από τις πλέον προφανείς, όπως η αναζήτηση ενός κλεμμένου οχήματος ή οχήματος που χρησιμοποιήθηκε για τη διάπραξη κάποιας εγκληματικής ενέργειας, μέχρι και την πρόληψη ενός τροχαίου ατυχήματος που μπορεί να προκληθεί από τον μεθυσμένο οδηγό του οχήματος που προηγείται ή την προειδοποίηση του χρήστη αναφορικά με τον εντοπισμό πιθανού απαγωγέα ή ενός stalker[10]. Στα κεφάλαια που ακολουθούν αναλύονται λεπτομερώς οι λειτουργίες του συστήματος, καθώς και ορισμένες από τις ενδεικτικές περιπτώσεις χρήσης που αυτές μπορούν να αξιοποιηθούν.

Τεχνικό Κεφάλαιο

Στο κεφάλαιο που ακολουθεί, παρατίθενται πληροφορίες που αφορούν στα τεχνολογικά στοιχεία που χρησιμοποιήθηκαν, για την ανάπτυξη του συστήματός μας.

[8] Η ηλεκτρονική ή μηχανική μετατροπή εικόνων δακτυλογραφημένου, εκτυπωμένου, χειρόγραφου και γενικότερα, οποιουδήποτε είδους εντυπωμένου κειμένου οποιασδήποτε πηγής, σε μορφοποιημένο κείμενο υπολογιστή - https://en.wikipedia.org/wiki/Optical_character_recognition

[9] https://troxoikaitir.gr/article/kameres-gia-aytomati-anagnorisi-pinakidon-se-8-synoriaka-simeia
https://www.4troxoi.gr/tehnologia/anpr-systima-aytomatis-anagnorisis-pinakidon/

[10] Εμμονικό άτομο που παρακολουθεί, παρενοχλεί ή καταδιώκει κάποιον με ανεπιθύμητη επιμονή.

Android SDK

Το Android SDK[11] αποτελεί ένα ολοκληρωμένο πακέτο ανάπτυξης λογισμικού, για κινητές συσκευές λειτουργικού συστήματος Android. Με άλλα λόγια, είναι ένα λογισμικό για την ανάπτυξη λογισμικού. Περιλαμβάνει μία ευρεία συλλογή από απαραίτητα εργαλεία ανάπτυξης, όπως βιβλιοθήκες, πρόγραμμα εντοπισμού σφαλμάτων (debugger), προσομοιωτή (emulator), δείγματα κώδικα και καθοδηγούμενα παραδείγματα εκμάθησης. Υποστηρίζεται από τις δημοφιλέστερες πλατφόρμες λειτουργικών, όπως Linux, Windows και Mac OS.

Αν και παρέχει τη δυνατότητα συγγραφής κώδικα μέσω της γραμμής εντολών (command line), προτιμάται, συνήθως, η χρήση του Android Studio της IntelliJ, το οποίο είναι το επίσημο Android IDE (Intergraded Development Environment). Οι αναβαθμίσεις του SDK, συμβαδίζουν με εκείνες της πλατφόρμας του Android, ωστόσο, το SDK προσφέρει την ευελιξία ανάπτυξης εφαρμογών και για παλαιότερες συσκευές, αφού υποστηρίζει και παλαιότερες εκδόσεις Android.

Τέλος, με το Android SDK, οι εφαρμογές μετατρέπονται σε εκτελέσιμα αρχεία .apk, τα οποία μπορούν να προσπελαστούν απευθείας από το λειτουργικό σύστημα Android.

Java

Η Java είναι μία γλώσσα προγραμματισμού, υψηλού επιπέδου, αντικειμενοστρεφής, η οποία αναπτύχθηκε το 1991 με κύριους ερευνητές τους J. Gosling, E. Frank, M. Sheridan και C. Warth. Η σύνταξή της, είναι παρόμοια με αυτή των C και C++ και είναι σχεδιασμένη έτσι ώστε να μην εξαρτάται από το λειτουργικό σύστημα στο οποίο έχει εγκατασταθεί. Η δυνατότητα αυτή, προέρχεται από το γεγονός ότι ο κώδικάς της δεν εκτελείται άμεσα, στο επίπεδο του λειτουργικού συστήματος, αλλά ένα επίπεδο πιο ψηλά, σε μια εικονική πλατφόρμα, που ονομάζεται JVM (Java Virtual Machine). Έτσι, οι εφαρμογές που είναι γραμμένες σε Java, μεταγλωττίζονται σε bytecode που μπορεί να εκτελείται σε οποιαδήποτε πλατφόρμα υποστηρίζει JVM, ανεξάρτητα από την αρχιτεκτονική του υπολογιστή, χωρίς να χρειάζεται να γίνει εκ νέου μεταγλώττιση (WORA – Write Once Run Anywhere).

Η ανάπτυξη της Java, στηρίχθηκε σε 5 βασικούς άξονες:

- Να είναι απλή, αντικειμενοστρεφής και γνώριμη (παρόμοια σύνταξη με την C και την C++).
- Να είναι σε θέση να διαχειρίζεται αποτελεσματικά τα σφάλματα που προκύπτουν κατά την εκτέλεση και τη λανθασμένη εισαγωγή δεδομένων (robustness) παραμένοντας ασφαλής.
- Να έχει ουδέτερη αρχιτεκτονική δόμησης και να χαρακτηρίζεται από μεταφερσιμότητα
- Να εκτελείται με μεγάλη αποδοτικότητα.
- Να μπορεί να μεταφράζεται, να υποστηρίζει τα νήματα (threads) και να είναι δυναμική.

Kotlin

Η Kotlin είναι μια γλώσσα προγραμματισμού, ανοιχτού κώδικα, γενικού σκοπού, στατική, σχεδιασμένη να υποστηρίζεται από πολλαπλές πλατφόρμες (cross-platform). Αναπτύχθηκε από την JetBrains σε συνεργασία με την Google, με την πρώτη επίσημη διάθεση για το κοινό να ανακοινώνεται τον Φεβρουάριο του 2016. Έκτοτε, η ομάδα Kotlin Foundation, μεριμνά για τη συνεχή υποστήριξη και ανάπτυξη της γλώσσας. Από το 2019, η Kotlin, αποτελεί την επίσημα προτιμώμενη γλώσσα για την ανάπτυξη εφαρμογών Android, παρέχοντας, έτσι, επίσημη τεκμηρίωση (documentation) και εξειδικευμένα εργαλεία.

Η Kotlin δεν αναπτύχθηκε προκειμένου να αποτελέσει μία νέα, μοναδική γλώσσα. Αντ' αυτού, ο σχεδιασμός και η ανάπτυξή της, αντλούν έμπνευση από δεκαετίες γλωσσικής ανάπτυξης, με αποτέλεσμα, να έχει παρόμοια σύνταξη και να δανείζεται έννοιες και λειτουργικότητα από άλλες

[11] https://en.wikipedia.org/wiki/Android_SDK

γλώσσες, όπως η C#, η Java και η Scala. Ένα από τα σημαντικότερα χαρακτηριστικά της, είναι η διαλειτουργικότητά της με την Java. Αυτό σημαίνει ότι, οι προγραμματιστές μπορούν να καλούν κώδικα που είναι γραμμένος σε Java, απευθείας από την Kotlin και αντίστροφα, αξιοποιώντας έτσι τις υπάρχουσες βιβλιοθήκες.

Η χρήση της, δεν περιορίζεται στις εφαρμογές Android. Η Kotlin, μπορεί να χρησιμοποιηθεί για την ανάπτυξη εφαρμογών, μεταξύ άλλων, σε JVM, iOS, Mac OS, Windows, Linux, καθώς και native, μέσω του LLVM[12]. Τέλος, είναι χρήσιμο να αναφερθεί ότι, ενσωματώνοντας την κατάλληλη αρχιτεκτονική και εργαλεία, η Kotlin, προσφέρει τη δυνατότητα ανάπτυξης εφαρμογών που μπορούν να εκτελεστούν, τόσο από το Android, όσο και από το iOS[13].

Google ML Kit

Το ML Kit, είναι το πακέτο μηχανικής μάθησης που προσφέρει η Google, στους προγραμματιστές εφαρμογών για κινητές συσκευές και ενσωματώνει την τεχνογνωσία που έχει αναπτύξει στον συγκεκριμένο τομέα. Αποτελεί ένα SDK για κινητά, το οποίο μπορεί να χρησιμοποιηθεί εύκολα, τόσο από εφαρμογές Android, όσο και από εφαρμογές iOS. Με το ML Kit, οι προγραμματιστές, μπορούν να προσδώσουν στις εφαρμογές τους χαρακτηριστικά τεχνολογιών μηχανικής μάθησης, επιλέγοντας ανάμεσα σε μια ευρεία λίστα λειτουργιών που καλύπτουν τις περισσότερες ανάγκες. Έτσι λοιπόν, μέχρι στιγμής, το ML Kit, υποστηρίζει:

- Αναγνώριση κειμένου (Text Recognition)
- Εντοπισμός προσώπου (Face Detection)
- Σάρωση barcode (Barcode Scanning)
- Επισήμανση εικόνας (Image Labeling)
- Εντοπισμός και παρακολούθηση αντικειμένων (Object detection and tracking)
- Αναγνώριση οροσήμων (Landmark Recognition)
- Γλωσσική ταυτοποίηση (Language Identification)
- Μετάφραση (Translation)
- Έξυπνη απάντηση (Smart Reply)
- Χρήση εξατομικευμένων/προσαρμοσμένων μοντέλων μηχανικής μάθησης (Custom ML Model inference)

Οι υπηρεσίες προσφέρονται για χρήση, τόσο μέσω ενός υπολογιστικού νέφους (cloud-based), όσο και απευθείας από την εκάστοτε κινητή συσκευή ως ενσωματωμένη λύση (on-device).

TensorFlow Lite

Το TensorFlow είναι ένα δωρεάν framework ανοιχτού κώδικα, που αναπτύχθηκε από ερευνητές της Google, για την εκτέλεση εργασιών machine-learning, deep-learning και άλλων αντίστοιχων προγνωστικών στατιστικής και ανάλυσης. Η χρήση του για την εφαρμογή και την εκπαίδευση ενός αλγορίθμου μηχανικής μάθησης, καταλήγει στην παραγωγή ενός αρχείου – μοντέλο (model file), το οποίο καταλαμβάνει μεγάλο αποθηκευτικό χώρο και απαιτεί τη χρήση μιας GPU για την εκμετάλλευσή του.

Το TensorFlow Lite, είναι η λύση για τη χρήση αυτών των μοντέλων από τις κινητές συσκευές, όπου ο χώρος και η χρήση GPU, αποτελούν παραμέτρους που υπόκεινται σε πολλούς περιορισμούς. Το TensorFlow Lite, λαμβάνει ήδη υπάρχοντα μοντέλα TensorFlow και τα μετατρέπει σε μια βελτιστοποιημένη εκδοχή, έτοιμη για την ενσωμάτωσή της σε εφαρμογές που απευθύνονται σε κινητές συσκευές. Το παραγόμενο αρχείο φέρει την κατάληξη .tflite και μπορεί να ενσωματωθεί σε εφαρμογές που αφορούν κινητές συσκευές (Android και iOS), Rasberry Pi, ακόμη και μικροεπεξεργαστές για την εξυπηρέτηση του Internet Of Things (IOT).

[12] https://en.wikipedia.org/wiki/LLVM

[13] https://kotlinlang.org/lp/mobile/

Συνοψίζοντας, αναφέρουμε ορισμένα από τα πλεονεκτήματά του:

- Μετατρέπει τα μοντέλα που έχουν αναπτυχθεί από το TensorFlow σε μοντέλα TensorFlow Lite, γρήγορα και εύκολα.
- Προσφέρεται για την παραγωγή εφαρμογών μηχανικής μάθησης για κινητές συσκευές και μικροεπεξεργαστές.
- Αποτελεί μια πιο αποτελεσματική εναλλακτική επιλογή για χρήση από συσκευές με μικρή υπολογιστική ισχύς, έναντι της χρήσης αρχιτεκτονικής που βασίζεται στον διακομιστή.
- Στην περίπτωση των κινητών συσκευών, επιτρέπει την εκτός σύνδεσης (offline) χρήση του.

Transfer Learning

Το συνηθέστερο και πιο εξεζητημένο πρόβλημα στη μηχανική μάθηση, αφορά στα δεδομένα. Μπορούν να είναι είτε ανεπαρκή, είτε κακής ποιότητας. Δυστυχώς, η εύρεση επαρκούς όγκου δεδομένων, τα οποία να ανταποκρίνονται και στην επιθυμητή ποιότητα για την ανάπτυξη ενός αξιόπιστου μοντέλου, είναι, συνήθως, ζητήματα αμοιβαίως αποκλειόμενα. Είθισται να θυσιάζεται το ένα στο βωμό του άλλου.

Η καινοτομία της μεθόδου του Transfer Learning, προσεγγίζει το ζήτημα από διαφορετική οπτική. Η μέθοδος, επαναχρησιμοποιεί ένα ήδη εκπαιδευμένο μοντέλο μηχανικής μάθησης, για την ανάπτυξη ενός μοντέλου που προορίζεται για άλλη εργασία. Η γενική ιδέα είναι ότι, το προ-εκπαιδευμένο μοντέλο προσφέρει μια καθοδήγηση στο νέο μοντέλο, που αφορά σε πιο γενικά χαρακτηριστικά, ενώ, στη συνέχεια, με την τροφοδοσία των περιορισμένων δεδομένων, η τελική εκπαίδευση του μοντέλου, προσαρμόζεται σε πιο συγκεκριμένα χαρακτηριστικά.

Υπάρχει μεγάλη ποικιλία[14] ήδη εκπαιδευμένων μοντέλων, ανοιχτού κώδικα, που προσφέρονται για χρήση με τη βοήθεια του TensorFlow Lite. Ο συνδυασμός της μεθόδου Transfer Learning και του TensorFlow Lite, επιτρέπει την ενσωμάτωση λειτουργιών μηχανικής μάθησης σε κινητές και τελικές συσκευές, εύκολα και γρήγορα, χωρίς να απαιτείται να δημιουργήσουμε και να εκπαιδεύσουμε, εξ' αρχής, ένα μοντέλο.

RESTful Web Services - API

Υπάρχουν πολλοί και διαφορετικοί ορισμοί στη βιβλιογραφία, αναφορικά με το τί είναι ένα web service. Παρ' όλα αυτά, όλοι, συγκλίνουν σε μια σειρά από χαρακτηριστικά που χρειάζεται να συγκεντρώνει:

- Διατίθεται μέσω διαδικτύου ή δικτύου intranet
- Χρησιμοποιεί ένα τυποποιημένο πρωτόκολλο ανταλλαγής μηνυμάτων XML (eXtensible Markup Language)[15]
- Είναι ανεξάρτητο από οποιοδήποτε λειτουργικό σύστημα και γλώσσα προγραμματισμού
- Αυτό-περιγράφεται μέσω τυπικής γλώσσας XML
- Μπορεί εύκολα να εντοπιστεί χρησιμοποιώντας έναν απλό μηχανισμό εντοπισμού

Αναλύοντας λίγο περισσότερο όσα συνοπτικά αναφέρθηκαν παραπάνω, θα λέγαμε ότι, στο διαδίκτυο, ένα web service είναι μια τυποποιημένη μέθοδος για τη διάδοση μηνυμάτων μεταξύ των υπολογιστών που το απαρτίζουν. Αποτελεί ένα είδος προγράμματος, το οποίο προορίζεται να εκτελεί ένα συγκεκριμένο σύνολο λειτουργιών.

[14] https://tfhub.dev/s

[15] https://el.wikipedia.org/wiki/XML

Γενικότερα, απαρτίζεται από ένα σύνολο ανοιχτών πρωτοκόλλων και προτύπων, που επιτρέπουν την ανταλλαγή δεδομένων, μεταξύ διαφορετικών εφαρμογών ή συστημάτων. Μπορεί, δηλαδή, να χρησιμοποιηθεί για την ανταλλαγή δεδομένων μεταξύ εφαρμογών, που είναι γραμμένες σε διαφορετικές γλώσσα προγραμματισμού, εκτελούνται σε διαφορετικά λειτουργικά συστήματα και βρίσκονται σε επικοινωνία μέσω κάποιου δικτύου υπολογιστών - όπως το διαδίκτυο - με τρόπο που προσομοιάζει στον τρόπο επικοινωνίας μεταξύ διεργασιών που εκτελούνται στον ίδιο υπολογιστή.

Επομένως, οποιοδήποτε λογισμικό, εφαρμογή ή τεχνολογία στο διαδίκτυο, χρησιμοποιεί τυποποιημένα πρωτόκολλα ιστού (HTTP / HTTPS), για τη σύνδεση, τη διαλειτουργικότητα και την ανταλλαγή μηνυμάτων δεδομένων (συνήθως XML), θεωρείται web service.

Το REST (REpresentational State Transfer), είναι μια αρχιτεκτονική λογισμικού, η οποία προσδιορίζει ένα σύνολο περιορισμών που θα χρησιμοποιηθούν για τη δημιουργία ενός web service. Τα web services που συμμορφώνονται με αυτούς τους περιορισμούς, αποκαλούνται RESTful web services και χρησιμοποιούνται για να παρέχουν διαλειτουργικότητα μεταξύ υπολογιστικών συστημάτων στο διαδίκτυο.

Τα RESTful web services, επιτρέπουν τα συστήματα που αιτούνται μια υπηρεσία, να αποκτήσουν πρόσβαση και να χειριστούν αναπαραστάσεις κειμένου, που αντιπροσωπεύουν τους τους αιτούμενους πόρους, χρησιμοποιώντας ένα ομοιόμορφο και προκαθορισμένο σύνολο εντολών, απαλλαγμένων από πληροφορίες προηγούμενης κατάστασης[16]. Η τεχνολογία των REST web services, προτιμάται, γενικά, σε σχέση με την πιο ανθεκτική σε σφάλματα, SOAP (Simple Object Access Protocol), επειδή χρησιμοποιεί λιγότερο εύρος ζώνης (bandwidth), είναι απλούστερο και πιο ευέλικτο, καθιστώντας το καταλληλότερο για χρήση στο διαδίκτυο.

Τα REST API είναι ένας απλός και ευέλικτος τρόπος πρόσβασης στα web services, ο οποίος δεν απαιτεί καμία επεξεργαστική διεργασία. Η επικοινωνία που πραγματοποιείται μέσω REST API, χρησιμοποιεί μόνο αιτήματα HTTP. Η διαδικασία πραγματοποίησης μια κλήσης API, περιγράφεται ως εξής:

Ένα αίτημα αποστέλλεται από τον πελάτη (client) στον εξυπηρετητή (server), με τη μορφή ενός URL, ως αίτημα HTTP GET ή POST ή PUT ή DELETE. Στη συνέχεια, η απόκριση του εξυπηρετητή, επιστρέφει στον πελάτη με τη μορφή πόρου, ο οποίος μπορεί να είναι οτιδήποτε, όπως HTML, XML, εικόνα ή JSON.

Στο πρωτόκολλο HTTP[17], υπάρχουν πέντε μέθοδοι που χρησιμοποιούνται συνηθέστερα με την αρχιτεκτονική των REST. Αυτές είναι οι GET, POST, PUT και DELETE, οι οποίες αντιστοιχούν στις λειτουργίες ανάγνωσης, δημιουργίας, επικαιροποίησης και διαγραφής (εν συντομία CRUD από τα ακρωνύμια Create, Read, Update και Delete). Υπάρχουν και άλλες μέθοδοι, όπως οι OPTIONS και HEAD, οι οποίες, ωστόσο, χρησιμοποιούνται σπανιότερα. Αναλυτικότερα, για καθεμία από τις πέντε μεθόδους που αναφέρθηκαν παραπάνω, ισχύουν τα ακόλουθα:

- Η μέθοδος HTTP GET χρησιμοποιείται για την ανάγνωση ενός πόρου, σε αναπαράσταση της μορφής XML ή JSON, συνήθως, ενώ επιστρέφει ως απόκριση του πρωτοκόλλου HTTP και τον κωδικό 200 (Ok). Σε περίπτωση σφάλματος, τις περισσότερες φορές επιστρέφει 404 (Not Found) ή 400 (Bad Request).
- Η μέθοδος HTTP POST, χρησιμοποιείται, συνήθως, για τη δημιουργία νέων πόρων. Τα δεδομένα που αφορούν στο νέο πόρο και συνοδεύουν το αίτημά μας, αποστέλλονται στον διακομιστή ως πακέτο, με ξεχωριστή επικοινωνία. Στην περίπτωση όπου ο πόρος δημιουργηθεί επιτυχώς, ως απόκριση επιστρέφεται η κατάσταση 201 (status 201), μαζί με

[16] Αφορά στην ιδιότητα «statelessness» των REST web services, η οποία επιβάλλει ότι κάθε αίτημα πραγματοποιείται σε πλήρη απομόνωση. Η κατάσταση είναι οι πληροφορίες που αποθηκεύονται στο server-side τμήμα ενός συστήματος, προκειμένου να ταυτοποιηθούν τα εισερχόμενα αιτήματα και κάθε προηγούμενη διάδραση. Η statelessness ιδιότητα επιβάλλει τέτοιου είδους πληροφορία να μην αποθηκεύεται

[17] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

μια κεφαλίδα τοποθεσίας και έναν σύνδεσμο που «δείχνει» προς τον πρόσφατα δημιουργημένο πόρο.

- Η μέθοδος HTTP PUT, χρησιμοποιείται για την ενημέρωση ενός πόρου. Μπορεί επίσης να χρησιμοποιηθεί και για τη δημιουργία ενός νέου πόρου. Στην περίπτωση της επιτυχούς ενημέρωσης του πόρου, ως απόκριση επιστρέφεται ο κωδικός 200 (ή 204 εάν δεν επιστρέφεται το περιεχόμενο στο σώμα). Το PUT δεν είναι μία ασφαλής λειτουργία, αλλά είναι idempotent, που σημαίνει ότι είναι μια μέθοδος που μπορεί να κληθεί πολλές φορές, χωρίς κάθε φορά να επιφέρει διαφορετικά αποτελέσματα.
- Η μέθοδος HTTP DELETE, χρησιμοποιείται για τη διαγραφή ενός πόρου και προσδιορίζεται από ένα URI. Σε περίπτωση επιτυχούς διαγραφής, επιστρέφεται ως απόκριση ο κωδικός 200 (Ok), καθώς και ένα σώμα.

PostMan

Το PostMAn, είναι ένα εργαλείο, μια εφαρμογή που χρησιμοποιείται για τη δοκιμή των APIs. Διαδραματίζει τον ρόλο ενός HTTP client, ο οποίος πραγματοποιεί αιτήματα HTTP, χρησιμοποιώντας ένα γραφικό περιβάλλον, μέσω του οποίου ο χρήστης, λαμβάνει διαφορετικούς τύπους αποκρίσεων από τον server, ώστε στη συνέχεια να τους επικυρώσει. Προσφέρει, δηλαδή, ένα φιλικό προς χρήση περιβάλλον, βοηθώντας μας να εξοικονομήσουμε χρόνο κατά τη δημιουργία και τη δοκιμή των APIs.

Το PostMan, παρέχει τη δυνατότητα ομαδοποίησης διαφορετικών αιτημάτων, τα «collections». Αυτό το χαρακτηριστικό, εξυπηρετεί στην οργάνωση των δοκιμών, αφού μπορούν να οργανωθούν σε φακέλους και να δομηθούν με όποιον τρόπο κρίνεται πιο πρόσφορος από την ομάδα ανάπτυξης.

Τέλος, το PostMan, επιτρέπει να δημιουργήσουμε διαφορετικά περιβάλλοντα, μέσω της δημιουργίας μεταβλητών. Κατά συνέπεια, δίνεται η δυνατότητα να εκτελούνται δοκιμές σε διαφορετικά περιβάλλοντα, επαναχρησιμοποιώντας υπάρχοντα αιτήματα.

Java Spring Boot

Στον πυρήνα του Spring Boot, βρίσκεται το Spring Framework. Θα το χαρακτηρίζαμε, λοιπόν, ως μία επέκταση του Spring Framework η οποία, αν και περιέχει όλα τα χαρακτηριστικά του, είναι ευκολότερο στη χρήση. Σε αντίθεση με τα συνηθισμένα Spring projects με τις πολύπλοκες και χρονοβόρες ρυθμίσεις, το Spring Boot, αποτελεί ένα περιβάλλον που προσφέρει ταχύτητα και ευκολία στην ανάπτυξη λογισμικού, έτοιμου προς παραγωγή (production – ready environment). Έτσι, οι developers έχουν την ευκαιρία να εστιάσουν όλη τους την προσοχή στην ενσωμάτωση της business logic[18] του λογισμικού, αντί να παλεύουν με τις ρυθμίσεις και την εγκατάστασή του. Το Spring Boot είναι προσανατολισμένο στη λογική των microservices[19], καθώς και του dependency injection[20]. Το dependency, είναι ένα είδος «βιβλιοθήκης» (library), η οποία παρέχει συγκεκριμένη λειτουργικότητα στις εφαρμογές μας. Τα dependencies εμπλουτίζουν τον κώδικά μας, διευκολύνουν

[18] Η πιο σημαντική διαστρωμάτωση είναι το επίπεδο του business logic. Σε αυτό ουσιαστικά επιλύεται το πρόβλημα για το οποίο δημιουργήθηκε εξ' αρχής η οποιαδήποτε εφαρμογή. Δεν περιλαμβάνεται κάποιο συγκεκριμένο τμήμα κώδικα που ανήκει σε συγκεκριμένο framework ή άλλο λογισμικό, ούτε απαιτείται η χρήση συγκεκριμένης γλώσσας προγραμματισμού. Αποτελεί κώδικα που μπορεί κανείς να γράψει και να δοκιμάσει οπουδήποτε γιατί αφορά στη λογική πίσω από τη δομή των λειτουργιών που εκτελεί μια εφαρμογή.

[19] Τα microservices αποτελούν μία αρχιτεκτονική και οργανωτική προσέγγιση στην ανάπτυξη λογισμικού, όπου το λογισμικό συντίθεται από μικρά, ανεξάρτητα τμήματα (services), τα οποία επικοινωνούν μεταξύ τους μέσω καλά καθορισμένων APIs. Τα συστήματα που βασίζονται σε microservices, μπορούν εύκολα να τροποποιηθούν ή και να μεταφερθούν, καθώς δεν είναι απαραίτητο να αλλαχθεί ολόκληρη η υποδομή των υπηρεσιών, αλλά μονάχα ένα τμήμα της. Επιπλέον, είναι δυνατό κάθε τμήμα (service), να κατασκευαστεί χρησιμοποιώντας διαφορετική τεχνολογία από κάθε άλλο, ανάλογα με τις απαιτήσεις.

[20] Το «Dependency Management», αποτελεί βασικό χαρακτηριστικό του Spring-Boot. Είναι ένας καθολικός τρόπος διαχείρισης όλων των απαραίτητων dependencies, για την αποδοτικότερη χρήση τους.

και επιταχύνουν την ανάπτυξή του, ενώ παράλληλα, με τη χρήση τους, ελαχιστοποιούνται τα σφάλματα και η επανάληψη τμημάτων του κώδικα (boilerplate code).

Τα Spring projects ακολουθούν μία δομημένη, πολυστρωματική διάταξη (layered architecture), στην οποία, κάθε διαστρωμάτωση μπορεί να επικοινωνεί με τις άλλες που βρίσκονται πάνω ή κάτω από αυτή. Η δομή, λοιπόν, των Spring Boot projects, αποτελείται από τα εξής τέσσερα επίπεδα διαστρωμάτωσης:

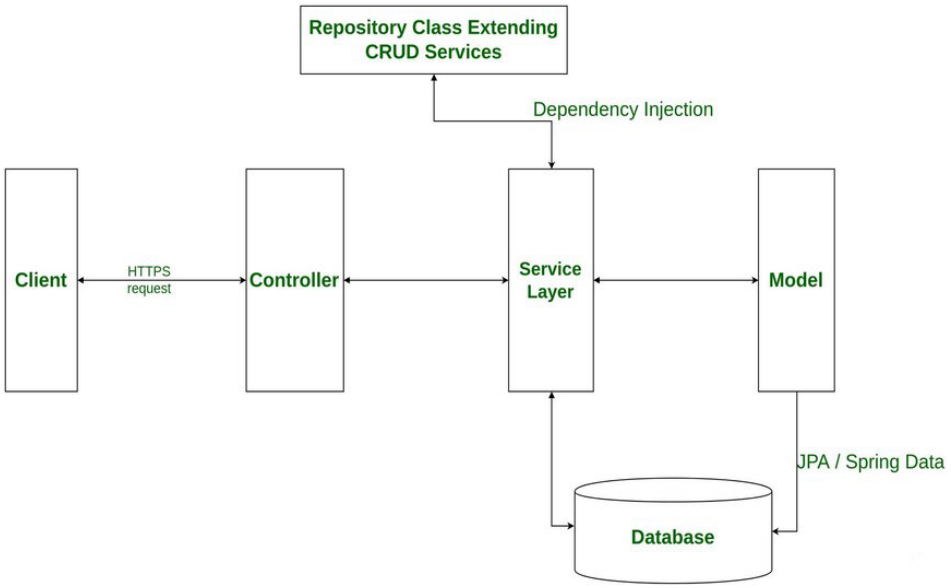
- **Presentation layer** – Αποτελεί το πρώτο επίπεδο διαστρωμάτωσης και ως εκ τούτου, βρίσκεται στην κορυφή της δομής. Συστατικά του στοιχεία είναι τα Views, δηλαδή το front-end τμήμα της εφαρμογής. Το επίπεδο αυτό, είναι επιφορτισμένο με τη διαδικασία χειρισμού των HTTP requests, ενώ, στο ίδιο επίπεδο, πραγματοποιούνται και οι διαδικασίες αυθεντικοποίησης των παραμέτρων. Οι παράμετροι των πεδίων ενός JSON, μετατρέπονται σε αντικείμενα της JAVA και αντίστροφα, ενώ στη συνέχεια, αφού ολοκληρωθεί η αυθεντικοποίησή τους, διοχετεύονται στο επόμενο επίπεδο διαστρωμάτωσης.
- **Business layer** – Εδώ βρίσκεται ο controller, όπου ενσωματώνεται όλη η business logic. Επιπλέον, το συγκεκριμένο επίπεδο, είναι επιφορτισμένο και με την επικύρωση των παραμέτρων που τροφοδοτούνται στην εφαρμογή. Ο controller αποτελεί το σημείο διεπαφής του client-side, με το server-side τμήμα του συστήματος. Υπάρχουν δυο βασικά είδη controller που χρησιμοποιούνται στο Spring Boot, ο Controller και ο RestController. Η κατασκευή ενός controller είναι ιδιαίτερα εύκολη στο Spring Boot. Απαιτείται, μονάχα, η δημιουργία μίας κλάσης στην οποία θα ενσωματώσουμε τα annotation @restcontroller ή @controller. Ο RestController χρησιμοποιείται για τη δημιουργία RESTful web services, και επιτρέπει τον χειρισμό όλων των REST APIs[21], όπως GET, POST, DELETE και PUT requests, που πραγματοποιούνται από το client-side τμήμα του συστήματος. Η δημιουργία των REST APIs είναι, επίσης, εύκολη διαδικασία με το Spring Boot. Αρκεί η χρήση του annotation @RequestMapping(/endpoint)[22] στην κλάση του controller.
- **Persistence layer** – Η συγκεκριμένη διαστρωμάτωση, εμπεριέχει τη λογική διαδικασία που αφορά στην αποθήκευση των δεδομένων στη βάση δεδομένων που χρησιμοποιεί η εφαρμογή μας. Εδώ πραγματοποιείται η μετατροπή των αντικειμένων που παράχθηκαν από την business logic της εφαρμογής, σε εγγραφές, που αντιστοιχούν στις οντότητες της βάσης δεδομένων (database entity objects) και αντίστροφα.
- **Database layer** – Σε αυτό το επίπεδο διαστρωμάτωσης, εδράζονται όλες οι βάσεις δεδομένων της εφαρμογής. Σε ένα project υπάρχει η δυνατότητα να φιλοξενηθούν περισσότερες από μία βάσεις. Όλες οι λειτουργίες που αφορούν στη Δημιουργία (Create), την Ανάγνωση (Read), την Επικαιροποίηση (Update) και τη Διαγραφή (Delete) στοιχείων μίας βάσης δεδομένων (database CRUD operations), πραγματοποιούνται με τη διαμεσολάβηση αυτού του επιπέδου. Η ενσωμάτωσή τους, γίνεται με μεγάλη ευκολία. Για την MongoDb, συγκεκριμένα, αρκεί η χρήση του public interface, MongoRepository. Το MongoRepository, λειτουργεί σαν ένας σύνδεσμος μεταξύ των μοντέλων που κατασκευάστηκαν και της βάση, παρέχοντάς τους όλες τις βασικές λειτουργίες CRUD.

Η διαστρωμάτωση αυτή, μας επιτρέπει να τροποποιούμε οποιοδήποτε από τα δομικά στοιχεία της εφαρμογής μας, ανεξάρτητα από κάθε άλλο. Έτσι, η εφαρμογή μας γίνεται πιο αποτελεσματική, ενώ οι διαδικασίες συντήρησης και επικαιροποίησης, γίνονται ευκολότερες. Μια σύντομη περιγραφή της ροής συνεργασίας των παραπάνω επιπέδων, δίνεται ακολούθως:

- Ο «πελάτης» κάνει ένα HTTP request (GET, POST, PUT κλπ)
- Το HTTP request προωθείται στον controller. Ο controller ελέγχει το αίτημα και το αντιστοιχίζει στις προκαθορισμένες ενέργειες που έχουν οριστεί στη business logic.

[21] Βλ. σχετικά σελ. 11 παρούσας διατριβής.
[22] Ένα παράδειγμα endpoint θα μπορούσε να είναι: http://domain_name/path/get_a_list

- Η business logic εκτελείται στο Service Layer και το Spring Boot επενεργεί πάνω στα δεδομένα της βάσης, με τον τρόπο που αυτά έχουν αντιστοιχηθεί με το MongoRepository[23] σε μοντέλα των οντοτήτων της βάσης (Model – Entity classes)
- Τελικά, μέσω του Controller, επιστρέφεται στον χρήστη η απόκριση του συστήματος.



Ροή συνεργασίας επιπέδων διαστρωμάτωσης Spring Boot

Σημαντικό στοιχείο στην κατασκευή μιας διαδικτυακής εφαρμογής, είναι και ο «εξυπηρετητής» (server) ο οποίος πρόκειται να την φιλοξενήσει. Με το Spring Boot δεν χρειάζεται να ανησυχούμε γι' αυτό. Το ίδιο το λογισμικό, φροντίζει ώστε να ενσωματώσει έναν κατάλληλα παραμετροποιημένο web server στην εφαρμογή μας. Ο server αυτός, συνήθως επιλέγεται να είναι ο Apache Tomcat[24]. Έτσι λοιπόν, ο server γίνεται μέρος της εφαρμογής μας, γεγονός που συνεπάγεται ότι, το παραγόμενο .jar[25] αρχείο θα εμπεριέχει και τον server με τις ρυθμίσεις που απαιτούνται για την εφαρμογή μας. Επομένως, όταν και αν επιλέξουμε να μεταφέρουμε σε άλλο περιβάλλον την εφαρμογή μας, δεν χρειάζεται να είναι προ-εγκατεστημένος ο server που θα την φιλοξενήσει.

Ένα ακόμη πλεονέκτημα που προσφέρει το Spring Boot, είναι η δυνατότητα χρήσης των annotations. Τα annotations δεν αποτελούν κώδικα, ωστόσο, χρησιμοποιούνται για να παράσχουν ορισμένες συμπληρωματικές πληροφορίες που αφορούν στις λειτουργίες του προγράμματός μας. Θα λέγαμε, δηλαδή, ότι αποτελούν ένα σύνολο συμπυκνωμένων «οδηγιών» προς τον compiler.

Συνοψίζοντας, ορισμένα από τα βασικά χαρακτηριστικά του Spring Boot είναι:

- Σε αντίθεση με το Spring Framework, αποφεύγονται οι απαιτητικές ρυθμίσεις του XML αρχείου. Το μόνο που χρειάζεται είναι να προσδιοριστούν ορισμένες ρυθμίσεις περιορισμένης έκτασης για συγκεκριμένες λειτουργίες που επιθυμούμε.

[23] Ή το JpaRepository (Java Persistence API) αν χρησιμοποιούμε σχεσιακή βάση δεδομένων (RDBMS – PostgreSQL, MySQL).

[24] Άλλες επιλογές είναι ο Jetty και ο Undertow.

[25] Προέρχεται από τη σύντμηση των όρων Java Archive. Πρόκειται για μια μορφή αρχείου που βασίζεται στη δημοφιλή μορφή ZIP και χρησιμοποιείται για τη συγκέντρωση πολλών αρχείων σε ένα.

- Διευκολύνει τη δημιουργία και τη συντήρηση των REST endpoints. Το μόνο που απαιτείται, είναι η χρήση των κατάλληλων annotation στον controller (controller class).
- Περιλαμβάνει ενσωματωμένο server, τον Tomcat.
- Παρέχει τη δυνατότητα μετατροπής των προγραμμάτων μας σε αρχεία τύπου .war ή .jar, ώστε να διευκολύνεται η μεταφορά και η φιλοξενία τους στο περιβάλλον του Tomcat server.

NoSQL - MongoDB

Οι βάσεις δεδομένων NoSQL, γνωστές και ως «not only SQL», είναι βάσεις δεδομένων οι οποίες, σε αντίθεση με τις σχεσιακές βάσεις δεδομένων, δεν χρησιμοποιούν πίνακες για την αποθήκευση και την ανάκτηση των δεδομένων που φιλοξενούν, αλλά χρησιμοποιούν διαφορετικό μηχανισμό μοντελοποίησής τους. Ανάλογα με το είδος του μοντέλου που χρησιμοποιούν, διακρίνονται σε τέσσερις διαφορετικούς τύπους:

- **Document** – Τα δεδομένα αποθηκεύονται σε έγγραφα (documents) που μοιάζουν με αρχείο JSON (JavaScript Object Notation). Κάθε document περιλαμβάνει ζεύγη πεδίου και τιμής. Οι τιμές μπορούν να είναι συμβολοσειρές, αριθμοί, Booleans, πίνακες ή ακομη και αντικείμενα.
- **Key-Value** – Είναι ένας απλούστερος τύπος βάσης δεδομένων, όπου κάθε στοιχείο αποτελείται από ένα κλειδί που αντιστοιχίζεται σε μία τιμή.
- **Wide-column** – Αποθηκεύουν δεδομένα σε πίνακες, γραμμές και δυναμικές στήλες
- **Graph** – Αποθηκεύουν δεδομένα σε κόμβους και ακμές. Συνήθως, οι κόμβοι αποθηκεύουν δεδομένα που αφορούν σε πληροφορίες όπως αντικείμενα, άνθρωποι, τοποθεσίες, ενώ οι ακμές, αποθηκεύουν πληροφορίες που αφορούν στις σχέσεις μεταξύ των κόμβων

Οι NoSQL βάσεις, χαρακτηρίζονται για την ευελιξία της δομής τους (schema), την ευκολία επέκτασής τους, ανάλογα με τον όγκο των δεδομένων και των χρηστών που εξυπηρετούν, την ταχύτητα εξυπηρέτησης των ερωτημάτων και την ευχρηστία τους. Αν και οι βάσεις NoSQL υπήρχαν ήδη από τα τέλη της δεκαετίας του 1960, η χρήση τους, άρχισαν να επεκτείνεται ολοένα και περισσότερο στις αρχές του 21ου αιώνα, με την εμφάνιση των big-data και των web based εφαρμογών πραγματικού χρόνου. Η πιο διαδεδομένη NoSQL βάση δεδομένων, είναι η MongoDB.

Η MongoDB, χρησιμοποιεί την μοντελοποίηση των Documents για την αποθήκευση των δεδομένων. Αποτελεί ιδανική περίπτωση βάσης δεδομένων για διαδικτυακές εφαρμογές, αφού είναι γρήγορη και μπορεί να διαχειριστεί αποτελεσματικά, μεγάλο όγκο δεδομένων, δομημένων αλλά και μη-δομημένων. Τέλος, αξίζει να σημειωθεί ότι στη MongoDB, τα δεδομένα αποθηκεύονται σε μορφή BSON[26], γεγονός που καθιστά την ανάκτηση και την προσπέλασή τους ιδιαίτερα γρήγορη.

MongoDB Compass

Το Compass είναι ένα γραφικό περιβάλλον χρήσης (GUI), που απευθύνεται στους χρήστες της MongoDB, για την δημιουργία ερωτημάτων, aggregation και την ανάλυση των δεδομένων που φιλοξενούνται στη βάση τους, σε ένα οπτικοποιημένο περιβάλλον. Είναι ελεύθερα διαθέσιμο και μπορεί να εκτελεστεί τόσο σε περιβάλλον Windows, όσο και σε MacOS ή Linux.

Google Maps API

Το Google Maps είναι η πιο διαδεδομένη πλατφόρμα για την αναπαράσταση και την οπτικοποίηση χωρικών δεδομένων στο διαδίκτυο. Η χρησιμότητα των λειτουργιών που προσφέρει η

[26] https://www.mongodb.com/json-and-bson

εφαρμογή, ξεπερνούν τα όρια μιας απλής εφαρμογής για κινητές συσκευές. Για τον λόγο αυτό, η Google, παρέχει τη δυνατότητα στους προγραμματιστές, να αποκτήσουν πρόσβαση στην πληθώρα των γεωχωρικών δεδομένων της Google, μέσω των Google Maps API. Με τον τρόπο αυτό, οι προγραμματιστές μπορούν να προβάλλουν διαδραστικούς χάρτες και να τους προσαρμόσουν στις δικές τους ανάγκες, ανάλογα με την πλατφόρμα ή την εφαρμογή που αναπτύσσουν. Επιπλέον, με τη χρήση των Google Maps API, οι χάρτες και τα γεωχωρικά δεδομένα που χρησιμοποιεί ο εκάστοτε προγραμματιστής, είναι πάντοτε επικαιροποιημένα, χωρίς να απαιτείται από εκείνον να πραγματοποιήσεις οποιαδήποτε περαιτέρω ρύθμιση.

QGIS

Τα συστήματα γεωχωρικών πληροφοριών (GIS – Geographic Information Systems), είναι ολοκληρωμένα συστήματα συλλογής, αποθήκευσης, διαχείρισης, ανάλυσης και απόδοσης πληροφορίας, σχετικά με φαινόμενα που εξελίσσονται στο χώρο. Το QGIS, είναι ένα ελεύθερα διαθέσιμο σύστημα γεωχωρικών πληροφοριών, ανοιχτού κώδικα, το οποίο μπορεί να εγκατασταθεί σε οποιονδήποτε σταθερό υπολογιστή (Windows, MacOS και Linux). Επιπλέον, υπάρχει πληθώρα επεκτάσεων (plug-ins), τα οποία επεκτείνουν την λειτουργικότητά του, ανάλογα με τις εργασίες που χρειάζεται να εκτελεστούν.

DBSCAN

Οι αλγόριθμοι ομαδοποίησης (clustering algorithms) αναζητούν ομοιότητες ή διαφορές ανάμεσα στα δεδομένα, προκειμένου να σχηματοποιήσουν ομάδες συνόλων με παρόμοια χαρακτηριστικά. Τα δεδομένα που ανήκουν στην ίδια ομάδα, έχουν όμοια χαρακτηριστικά, ενώ δεδομένα που ανήκουν σε διαφορετικές ομάδες, διαφοροποιούνται με κάποιο τρόπο. Επειδή η διαδικασία ομαδοποίησης, είναι μια μέθοδος εκμάθησης χωρίς επίβλεψη (unsupervised learning method), όπου τα δεδομένα δεν συνοδεύονται από ετικέτες, ο εκάστοτε αλγόριθμος, προσπαθεί να εντοπίσει την υποκείμενη δομή τους, αν υπάρχει.

Υπάρχουν πολλοί και διαφορετικοί αλγόριθμοι ομαδοποίησης (K-Means, DBSCAN, Spectral clustering, Gaussian mixtures κ.α.), καθένας από τους οποίους στηρίζεται σε διαφορετική προσέγγιση για τον τρόπο με τον οποίο θα εκτελεστούν οι εργασίες διαχωρισμού των δεδομένων. Έτσι, ανάλογα με την μέθοδο ομαδοποίησης, οι αλγόριθμοι χωρίζονται σε τρείς μεγάλες κατηγορίες:

- Partition-based clustering (Ομαδοποίηση βάσει διαχωρισμού)
- Hierarchical clustering (Ιεραρχική ομαδοποίηση)
- Density based clustering (Ομαδοποίηση με βάση την πυκνότητα)

Οι δύο πρώτες κατηγορίες, είναι εξαιρετικά αποδοτικές όταν πρόκειται να χειριστούμε δεδομένα που έχουν κανονικοποιημένο σχήμα. Ωστόσο, όταν πρόκειται να χειριστούμε δεδομένα ακανόνιστου σχήματος, ή όταν θέλουμε να εντοπίσουμε ακραίες τιμές, οι αλγόριθμοι που στηρίζονται στην πυκνότητα των δεδομένων, είναι πιο αποτελεσματικοί.

Όπως υπαινίσσεται και η ονομασία του (Density Based Spatial Clustering Applications with Noise), ο DBSCAN, είναι ένας αλγόριθμος ομαδοποίησης με βάση την πυκνότητα των δεδομένων. Είναι αποδοτικός στον εντοπισμό ομάδων με θόρυβο (ακραίες τιμές) και ομάδες ακανόνιστου σχήματος. Η κεντρική ιδέα πίσω από την προσέγγισή του, είναι ότι ένα στοιχείο ανήκει σε μία ομάδα, αν είναι αρκετά κοντά, σε αρκετά σημεία από την συγκεκριμένη ομάδα.

Θέλοντας να αναλύσουμε λίγο περισσότερο αυτή τη δήλωση, θα χρειαστεί να προσδιορίσουμε δύο βασικές παραμέτρους για τον αλγόριθμο:

- Eps (epsilon) – Αφορά στην παράμετρο που καθορίζει την μέγιστη απόσταση (πόσο κοντά) μεταξύ γειτονικών σημείων. Καθορίζει, δηλαδή, την μέγιστη απόσταση που μπορεί να απέχουν μεταξύ τους δύο σημεία ώστε αυτά να ανήκουν στην ίδια ομάδα («αρκετά κοντά»).

- minPts (minimum points) – Αφορά στην παράμετρο που προσδιορίζει τον ελάχιστο αριθμό σημείων που είναι αρκετά ώστε να θεωρηθεί μια συστάδα στοιχείων ως ομάδα

Ο DBSCAN βρίσκει πρακτική εφαρμογή στην εύρεση συσχετισμών και δομών σε δεδομένα που είναι δύσκολο να εντοπιστούν χειροκίνητα, καθώς και στην ανακάλυψη μοτίβων και την πρόβλεψη τάσεων.

Ακολούθως, παραθέτουμε μια συνοπτική περιγραφή του τρόπου λειτουργίας του αλγορίθμου:

Αρχικά, καθορίζουμε το eps και το minPts. Στη συνέχεια, ένα από τα σημεία που απαρτίζουν τα δεδομένα μας, επιλέγεται τυχαία και ο αλγόριθμος καθορίζει την γειτονική περιοχή, χρησιμοποιώντας την τιμή του eps, ως ακτίνα. Εάν ικανοποιείται ο ελάχιστος αριθμός στοιχείων (minPts) γειτόνων, το τυχαίο σημείο ορίζεται ως κεντρικό σημείο της ομαδοποίησης και αρχίζει η σχηματοποίησή της. Σε αντίθετη περίπτωση, το σημείο σημαίνεται ως ακραία τιμή (outlier). Έπειτα, ο αλγόριθμος εξετάζει αν τα γειτονικά στοιχεία που ορίστηκαν, έχουν και αυτά γειτονικά στοιχεία εντός της ακτίνας με μήκος την τιμή του eps και κέντρο τα ίδια. Σε περίπτωση που εντοπιστούν γειτονικά στοιχεία των γειτόνων του αρχικού, τυχαίου σημείου, τότε, συμπεριλαμβάνονται και αυτά στην αρχική ομαδοποίηση. Ακολούθως, ο αλγόριθμος επιλέξει ένα άλλο τυχαίο σημείο μεταξύ των δεδομένων που δεν έχει ήδη επισκεφθεί σε κάποιο από τα προηγούμενα βήματα και επαναλαμβάνει τη διαδικασία που περιγράψαμε παραπάνω. Η διαδικασία ολοκληρώνεται όταν ο αλγόριθμος έχει επισκεφθεί όλα τα σημεία και έχουν σημανθεί είτε ως ακραίες τιμές, είτε έχουν συμπεριληφθεί σε κάποια ομαδοποίηση.

Παρουσίαση Λειτουργιών (Συστήματος)

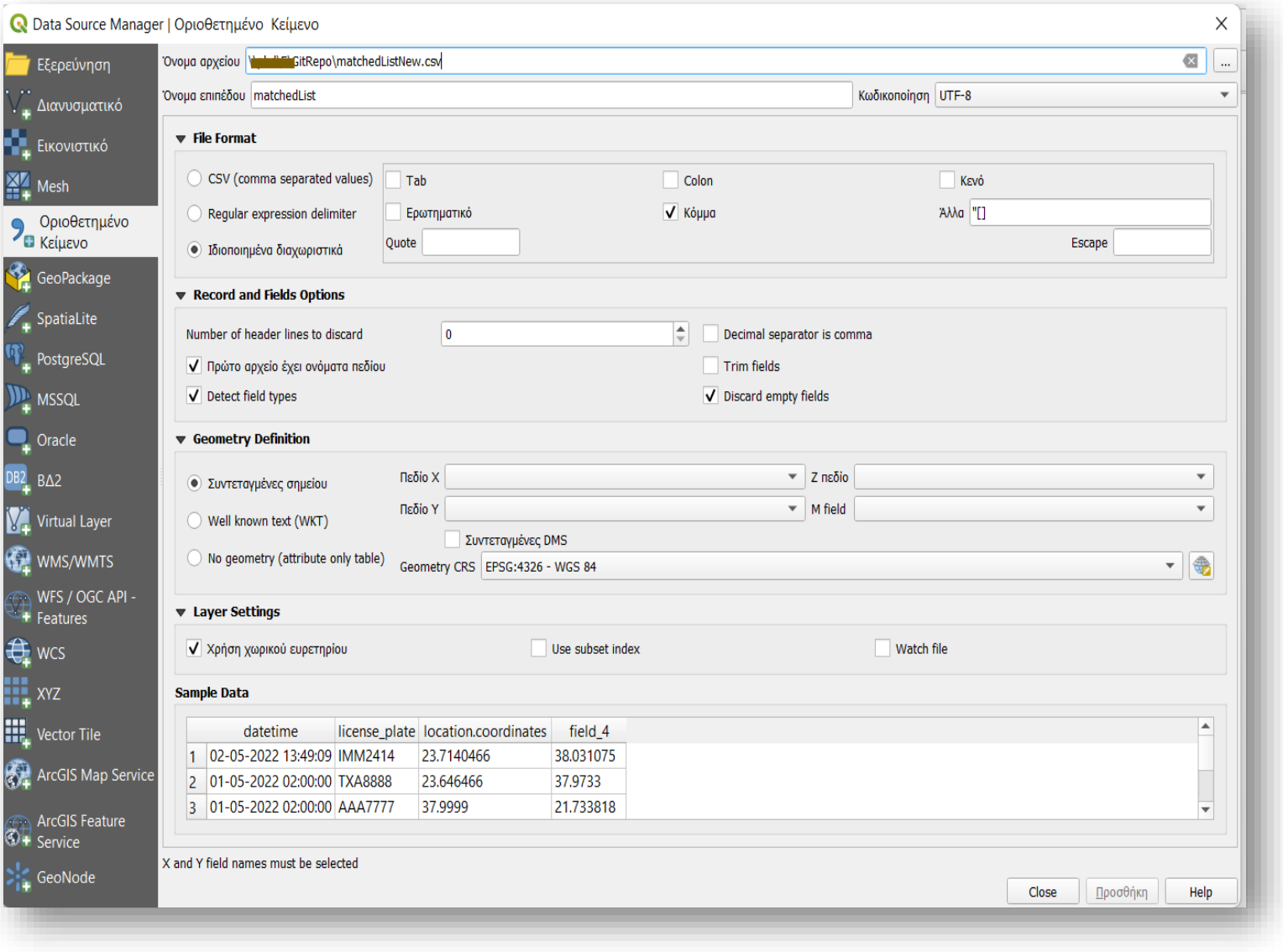
Όπως έχει ήδη σημειωθεί, η υλοποίησή μας αφορά σε ένα ολοκληρωμένο σύστημα ALPR, το οποίο απαρτίζεται από έναν web server και μία εφαρμογή Android. Ο Web Server διαχειρίζεται ένα σύνολο από web services, υποστηρίζεται από μια NoSQL βάση δεδομένων και αποτελεί το back-end της υλοποίησής μας. Αντίστοιχα, η εφαρμογή Android (MobileLPR), συνθέτει το front-end της υλοποίησης και αποτελεί το σημείο διεπαφής του συστήματος, με τον χρήστη. Έχει σχεδιαστεί με τρόπο τέτοιο ώστε, να καταναλώνει τα web services του server, προκειμένου να εκτελεστούν οι λειτουργίες του συστήματος. Στοιχεία που αφορούν στις λειτουργίες των επί μέρους δομικών συστατικών της υλοποίησης, παρατίθενται στις ενότητες που ακολουθούν.

Web Server

Οι λειτουργίες που περιγράφονται σε αυτή την ενότητα, αφορούν στον διαχειριστή του συστήματος (Administrator) και διενεργούνται σε επίπεδο server-side. Στόχος μας, είναι η ομαδοποίηση των δεδομένων που έχουν συλλεγεί από τους χρήστες και αφορούν στο σύνολο των εντοπισμένων αναζητούμενων οχημάτων. Η ομαδοποίησή τους επιχειρείται να γίνει με βάσει την πυκνότητά τους σε ορισμένο χώρο. Έτσι λοιπόν, με τη βοήθεια του συνοδευτικού προγράμματος Compass της MongoDB[27], τα παραπάνω δεδομένα, αφού απαλλαχθούν από διπλότυπες εγγραφές[28] και περιττά στοιχεία, όπως ο χρήστης που καταχώρισε την εγγραφή και ο χρήστης που εντόπισε το αναζητούμενο όχημα, εξάγονται σε μορφή αρχείου CSV[29]. Στη συνέχεια, τα γεωχωρικά δεδομένα που θα προκύψουν, τροφοδοτούνται στο πρόγραμμα QGIS για την περαιτέρω αξιολόγηση και ανάλυσή τους. Από εκεί, χρησιμοποιώντας, τον αλγόριθμο DBSCAN, έχουμε την δυνατότητα να ομαδοποιήσουμε τα δεδομένα μας με έναν αυτοματοποιημένο τρόπο, αφού πρώτα ορίσουμε τον ελάχιστο αριθμό εγγραφών που οργανώνουν μια ομάδα (minPts)[30], καθώς και την μέγιστη απόσταση

[27] Βλ. αναλυτικότερα «Τεχνικό κεφάλαιο» παρούσας διατριβής.
[28] Υπάρχει πιθανότητα ένα αναζητούμενο όχημα να εντοπίστηκε από περισσότερους του ενός χρήστες, την περίοδο που αναζητούνταν, για τον ίδιο λόγο.
[29] Comma Separated Values
[30] Βλ. αναλυτικότερα ενότητα «DBSCAN» παρούσας διατριβής.

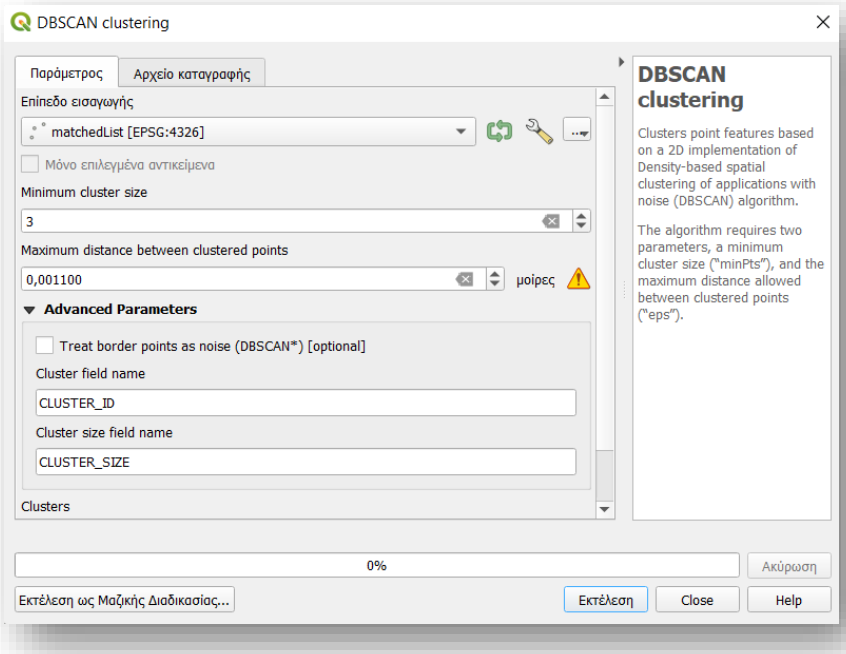
μεταξύ διαδοχικών κόμβων (eps)[31]. Τα αποτελέσματα μπορούν να οπτικοποιηθούν μέσω του QGIS, προβάλλοντάς τα σε έναν από τα παρεχόμενα επίπεδα. Ενδεικτικά, παρουσιάζονται τα αποτελέσματα που προέκυψαν για τις υπάρχουσες υποθετικές εγγραφές μας.



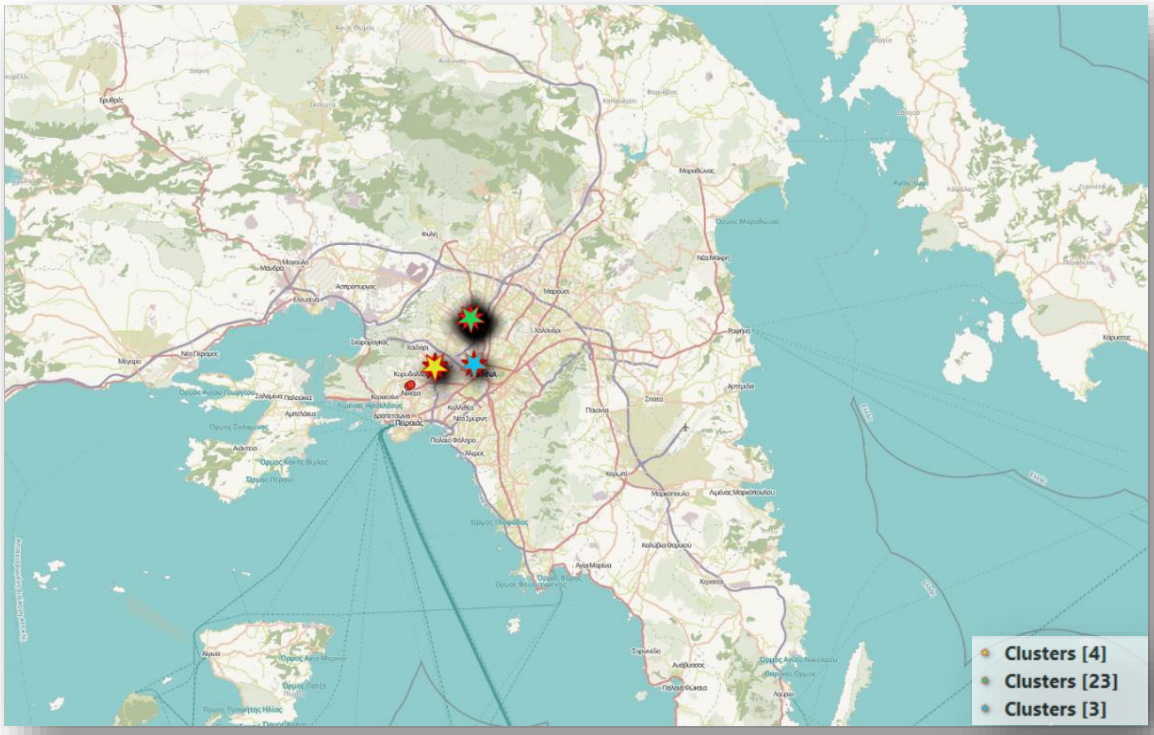
Εισαγωγή δεδομένων από αρχείο .csv στο QGIS

[31] Βλ. αναλυτικότερα ενότητα «DBSCAN» παρούσας διατριβής.

Απεικόνιση γεωχωρικών δεδομένων στο QGIS



Εισαγωγή δεδομένων στον αλγόριθμο DBSCAN, όπου eps≈100μ και minPts=3



Αποτέλεσμα του DBSCAN

Εστίαση στις ομαδοποιήσεις

Όπως γίνεται αντιληπτό από τις παραπάνω εικόνες, τα δεδομένα ομαδοποιήθηκαν με μη επιτηρούμενο τρόπο, σε τρία σύνολα. Καθένα από αυτά, απαρτίζεται από τουλάχιστον 3 στοιχεία και κάθε στοιχείο απέχει, από ένα, το πολύ, στοιχείο του συνόλου, απόσταση μικρότερη ή ίση με 100 μέτρα (περίπου 0,0011 μοίρες). Τα στοιχεία που δεν συμπεριλήφθηκαν σε κάποια ομαδοποίηση (κόκκινα σημεία)[32], θεωρήθηκαν ακραίες τιμές.

Αν και προδήλως, τα δεδομένα μας δεν είναι στατιστικά ανεξάρτητα (biased), η χρησιμότητα του αλγορίθμου για την εξαγωγή πρακτικά ωφέλιμων συμπερασμάτων, είναι αδιαμφισβήτητη. Σταχυολογώντας, λοιπόν, τα παραπάνω αποτελέσματα, μπορούμε να σημειώσουμε κάποιες ενδιαφέρουσες παρατηρήσεις, για να καταλήξουμε, τελικά, σε χρήσιμα, πρακτικά συμπεράσματα. Αρχικά, παρατηρούμε ότι ένα σημαντικό τμήμα των δεδομένων, μπορεί να σχηματοποιήσει συγκεκριμένες περιοχές ενδιαφέροντος (ομαδοποιήσεις). Παρατηρούμε, επίσης, ότι κάθε ομαδοποίηση, έχει διαφορετικό πλήθος στοιχείων, ενώ, μεταξύ των ομαδοποιήσεων, υπάρχει μια σχετική χωρική εγγύτητα (Δυτική Αττική).

Λαμβάνοντας, λοιπόν, υπόψη τις παραπάνω παρατηρήσεις, θα μπορούσαμε να επισημάνουμε, μεταξύ άλλων, ότι:

- Σε περίπτωση που αναζητούμε κάποιο όχημα, μία καλή αρχή θα ήταν οι τρείς περιοχές που έχουν σχηματοποιηθεί στην εικόνα 5. Από το πλήθος, μάλιστα, των στοιχείων κάθε ομαδοποίησης, θα μπορούσαμε να ορίσουμε και κάποιο είδος προτεραιοποίησης στις αναζητήσεις μας.
- Εντελώς ανάλογα, από την σκοπιά μιας δημόσιας αρχής ή ενός οργανισμού που δραστηριοποιείται στον τομέα της ασφάλειας, η ποσόστωση των ομαδοποιήσεων, θα μπορούσε να αποτελέσει τη βάση για την ορθολογικότερη κατανομή των πόρων που διατίθενται στην πραγματοποίηση περιπολιών, με στόχο, την πρόληψη εγκληματικών ενεργειών.
- Τέλος, από την χωρική εγγύτητα των ομαδοποιήσεων, αξίζει να διερευνηθεί το ενδεχόμενο ύπαρξης κάποιας εγκληματικής οργάνωσης, η οποία δραστηριοποιείται ή εδρεύει στην ευρύτερη περιοχή της Δυτικής Αττικής

Android App (MobileLPR)

Η εφαρμογή εστιάζει στην άμεση και ευρεία χρήση της, αποδεσμευμένη από την εξειδίκευση των χρηστών. Ως εκ τούτου, σχεδιάστηκε ώστε να μην απαιτείται κάποιου είδους αυθεντικοποίηση ή εγγραφή του χρήστη, αλλά διατηρώντας, ο χρήστης, την ανωνυμία του μπορεί να εισέλθει απευθείας στην εφαρμογή.

Με το άνοιγμα της εφαρμογής, ο χρήστης εισέρχεται στο κεντρικό μενού επιλογών (Main Menu). Από το σημείο αυτό, μπορεί να μεταβεί στο περιβάλλον χρήσης οποιασδήποτε εκ των τεσσάρων βασικών λειτουργιών της εφαρμογής:

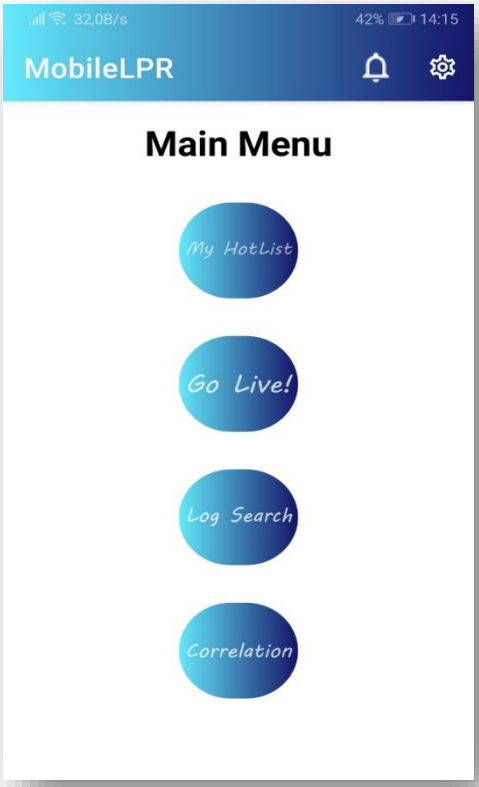
1. **My HotList** - *Επισκόπηση των καταχωρισμένων πινακίδων κυκλοφορίας του χρήστη*
2. **Go Live!** – *Σάρωση περιβάλλοντος σε πραγματικό χρόνο*
3. **Log Search** – *Αναζήτηση ιστορικού με χωροχρονικά κριτήρια*
4. **Correlation** – *Συσχέτιση εγγραφών με χωροχρονικά κριτήρια*

Επιπλέον, από την ίδια οθόνη, παρέχεται η δυνατότητα πλοήγησης στις ειδοποιήσεις και τις ρυθμίσεις της εφαρμογής (enable sound alerts). Ας δούμε αυτές τις λειτουργίες με τη σειρά, αναφέροντας, παράλληλα και ορισμένες περιπτώσεις όπου θα μπορούσαν να αξιοποιηθούν.

[32] Τα κόκκινα σημεία που απεικονίζονται στην εικόνα 4, είναι δύο στο σύνολο. Ως εκ τούτου, λόγω της ελάχιστης απαίτησης minPts=3 τα δύο αυτά στοιχεία δεν είναι ικανά να δημιουργήσουν μία ξεχωριστή ομαδοποίηση. Τα υπόλοιπα κόκκινα στοιχεία δεν είναι δυνατό να απεικονιστούν στην ίδια εικόνα του χάρτη

Εικόνα 6 – MobileLPR download barcode link

MobileLPR app icon



Main Menu Activity

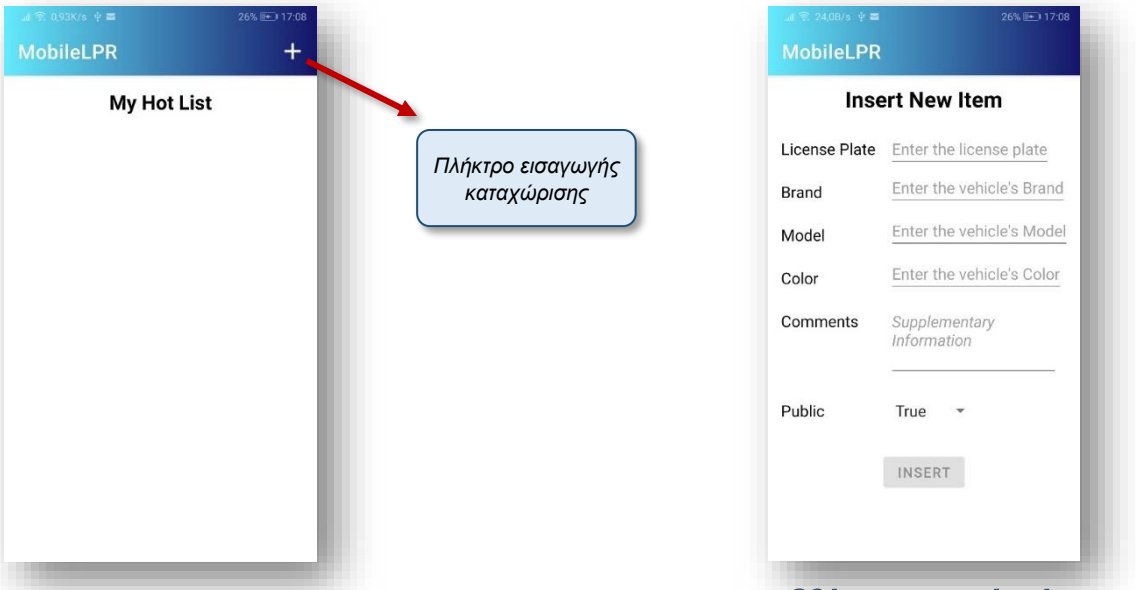
My HotList

Από την προβολή αυτή, ο χρήστης αποκτά πρόσβαση στο σύνολο των πινακίδων κυκλοφορίας που ο ίδιος έχει καταχωρήσει. Από το ίδιο activity, δίνεται η δυνατότητα στον χρήστη να τροποποιήσει ή και να διαγράψει τις εγγραφές του, καθώς και να εισάγει κάποια νέα. Κατά την εισαγωγή νέας καταχώρισης, ο χρήστης, έχει τη δυνατότητα να εισάγει τιμές στα πεδία που αντιστοιχούν στην πινακίδα κυκλοφορίας του οχήματος που τον απασχολεί, στη μάρκα, στο μοντέλο και στο χρώμα του, ενώ μπορεί ακόμη να εισάγει και ορισμένα συνοδευτικά σχόλια. Επιπλέον, καλείται να επιλέξει αν η εγγραφή του αυτή θα είναι δημόσια ή ιδιωτική (public true/false). Ως προεπιλογή έχει τεθεί η δημοσιοποίηση της εγγραφής, γεγονός που πρακτικά σημαίνει πως, αν κάποιος άλλος χρήστης εντοπίσει την συγκεκριμένη πινακίδα κυκλοφορίας, τότε, αφενός ο χρήστης που την καταχώρισε θα ειδοποιηθεί, αφετέρου, ο χρήστης που την εντόπισε, θα λάβει μια οπτικοποιημένη ειδοποίηση ότι κάποιος[33] την αναζητά. Παράλληλα με την οπτικοποιημένη

[33] Δεν ανακοινώνεται στον χρήστη ποιος/οι προέβη/σαν στην καταχώριση της συγκεκριμένης πινακίδας, αλλά μονάχα τα σχόλια που συνοδεύουν την καταχώριση.

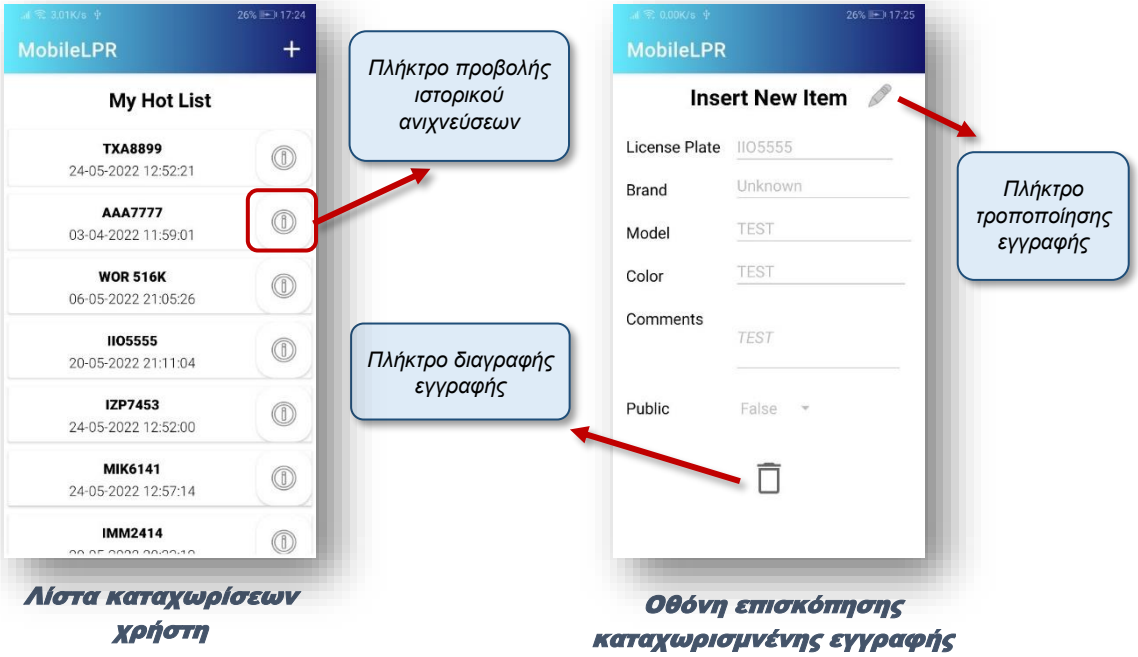
ειδοποίηση, στη συσκευή του χρήστη, θα προβληθούν και συγκεκριμένες συνοδευτικές πληροφορίες. Σημειώνεται ότι, η εφαρμογή έχει παραμετροποιηθεί έτσι ώστε, για την εισαγωγή μιας νέας καταχώρισης, να είναι απαραίτητη η εισαγωγή της αναζητούμενης πινακίδας κυκλοφορίας, ενώ ως βέλτιστη πρακτική, προτείνεται η αναγραφή σχολίων αναφορικά με τον λόγο αναζήτησης. Οποιαδήποτε άλλη πληροφορία, είναι συμπληρωματική, αφορά μόνο στον χρήστη που την καταχωρεί και σε περίπτωση που εντοπιστεί από έτερο χρήστη, δεν είναι προσπελάσιμη. Αντίθετα, τα σχόλια, αποτελούν μέρος της πληροφορίας που προβάλλεται στον χρήστη που θα εντοπίσει την αναζητούμενη πινακίδα. Ακόμη, εξαιτίας περιορισμών που αφορούν στην χρησιμοποιούμενη τεχνολογία αναγνώρισης χαρακτήρων[34], απαιτείται η χρήση λατινικών κεφαλαίων χαρακτήρων στο πεδίο License Plate[35]. Μια ακόμη λειτουργία που μπορεί να εκτελεστεί από αυτήν την οθόνη, είναι η προβολή του ιστορικού ανιχνεύσεων για κάθε μία από τις εγγραφές του χρήστη. Έτσι λοιπόν, πατώντας στην ειδική σήμανση με το σύμβολο ⓘ ο χρήστης περνάει σε διαφορετικό περιβάλλον προβολής (activity), απ' όπου μπορεί να ενημερωθεί σχετικά με το πόσες φορές έχει εντοπιστεί, συνολικά, η συγκεκριμένη πινακίδα, να δει την χρονοσήμανση (timestamp) κάθε ανίχνευσης, καθώς και να πλοηγηθεί στις τοποθεσίες όπου αυτή ανιχνεύθηκε.

Η αμεσότητα που προσφέρει η εφαρμογή, αυξάνει σημαντικά τις πιθανότητες ανεύρεσης ενός αναζητούμενου οχήματος. Ως εκ τούτου, οι τρόποι με τους οποίους μπορεί κάποιος να την αξιοποιήσει, καταχωρώντας μια πινακίδα κυκλοφορία, ποικίλουν. Από τον πλέον προφανή, δηλαδή, την καταχώριση ενός οχήματος που εκλάπη, μέχρι τη δήλωση αναζήτησης ενός οχήματος με το οποίο εξαφανίστηκε πριν λίγα λεπτά κάποιο οικείο πρόσωπο (Silver/Amber Alert), ή ακόμη και την προειδοποίηση των χρηστών του οδικού δικτύου στο οποίο κινούμαστε, ότι κάποιο όχημα, πραγματοποιεί επικίνδυνους ελιγμούς και θα ήταν συνετό να είμαστε προσεκτικοί αν τυχόν το εντοπίσουμε.



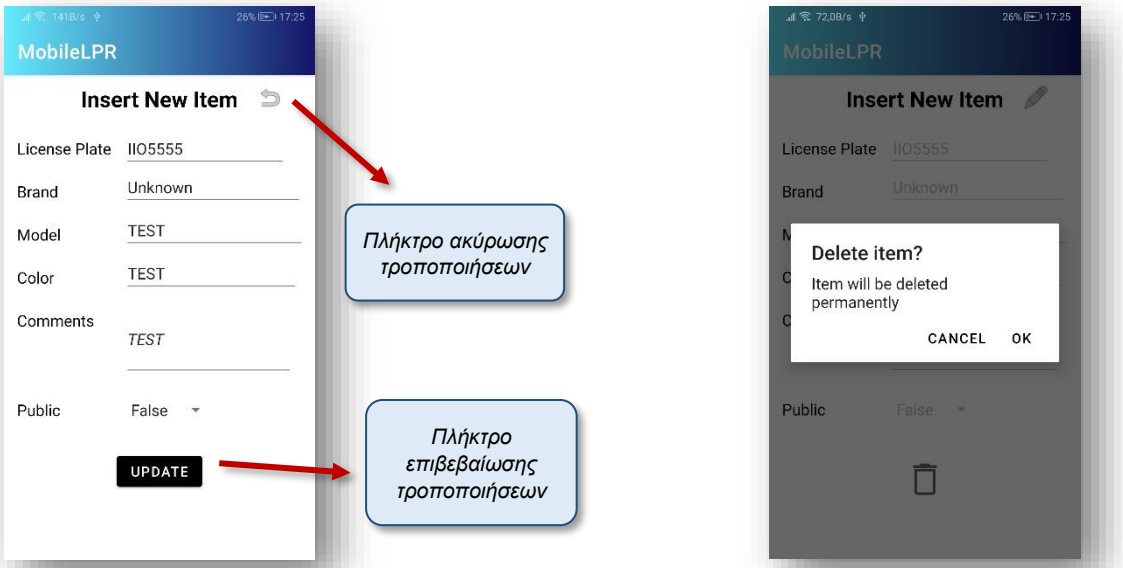
[34] Βλ. αναλυτικότερα ενότητα «Google ML-Kit» παρούσας διατριβής

[35] Ο περιορισμός αυτός δεν αποτελεί εμπόδιο στην εύρυθμη λειτουργία της εφαρμογής, καθώς οι πινακίδες κυκλοφορίας, διεθνώς – με εξαίρεση ορισμένες Αραβικές χώρες – χρησιμοποιούν λατινικούς χαρακτήρες, οι οποίοι βρίσκονται σε αντιστοιχία με χαρακτήρες του δικού τους αλφάβητου. Ως εκ τούτου, ελληνικοί χαρακτήρες όπως το Ω ή το Ξ δεν χρησιμοποιούνται για τα οχήματα δημόσιας χρήσης, χωρίς βέβαια να αποκλείεται η χρήση τους από ειδικούς κρατικούς φορείς, όπως η χρήση του γραμματικού μέρους «ΔΣ» για την δήλωση πινακίδων που ανήκουν σε Πρεσβείες που εδρεύουν στην Ελλάδα.



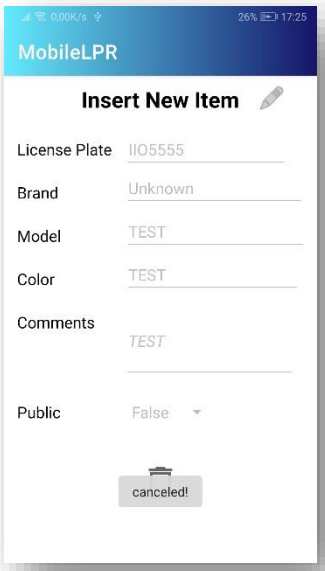
Λίστα καταχωρίσεων χρήστη

Οθόνη επισκόπησης καταχωρισμνένης εγγραφής

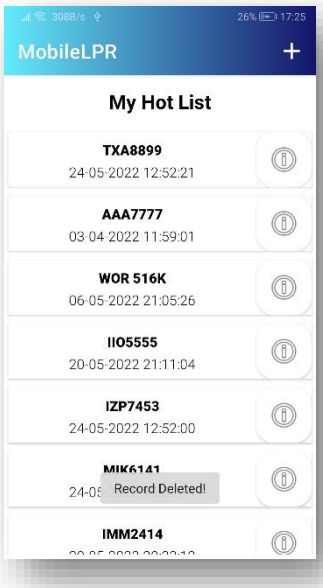


Οθόνη τροποποίησης εγγραφής

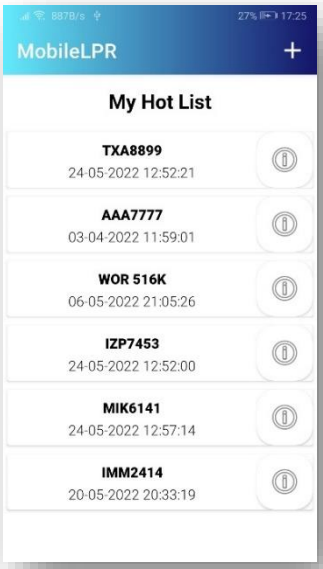
Οθόνη επιβεβαίωσης διαγραφής



Ακύρωση διαγραφής

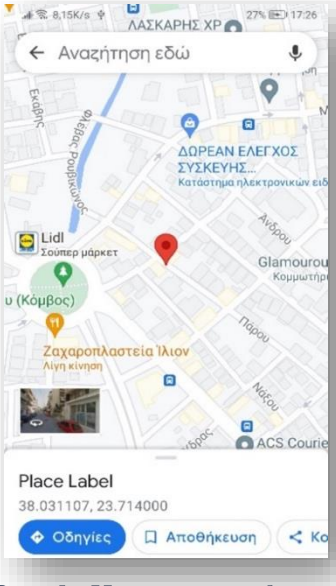


Μήνυμα επιβεβαίωσης ολοκλήρωσης διαγραφής



Λίστα καταχωρίσεων χρήστη μετά την επιτυχή διαγραφή εγγραφής

Οθόνη προβολής ιστορικού



Google Maps με επισήμανση θέσης εντοπισμού εγγραφής

Go Live!

Πρόκειται για την λειτουργία σάρωσης περιβάλλοντος, σε πραγματικό χρόνο (real-time scan). Αποτελεί την βασική λειτουργία της εφαρμογής και η χρήση της, απαιτεί την ικανοποίηση ορισμένων προϋποθέσεων. Την πρώτη φορά που θα επιλεγεί η συγκεκριμένη λειτουργία, η εφαρμογή, αρχικά θα ελέγξει αν η χρησιμοποιούμενη συσκευή διαθέτει ενσωματωμένη οπίσθια[36] κάμερα και στη συνέχεια, θα ζητήσει από τον χρήστη την παραχώρηση των απαραίτητων αδειών (πρόσβαση στην κάμερα της συσκευής και τη δυνατότητα χρήσης του GPS) για την χρήση της. Αν δεν παραχωρηθούν οι απαιτούμενες άδειες, η συγκεκριμένη λειτουργία δεν μπορεί να χρησιμοποιηθεί. Ωστόσο, άπαξ και παραχωρηθούν, η εφαρμογή θα επιβεβαιώνει κάθε φορά που επιλέγεται, αν ο εντοπισμός τοποθεσίας (GPS) είναι ενεργοποιημένος και σε θετική περίπτωση, θα προχωράει στην χρήση της κάμερας. Μη παραχώρηση των απαραίτητων αδειών ή απενεργοποίηση του GPS, οδηγεί σε τερματισμό της συγκεκριμένης λειτουργίας και επιστροφή στο κεντρικό μενού.

Περιγραφή τρόπου λειτουργίας activity

Έχοντας, πλέον, ενεργοποιηθεί η λειτουργία σάρωσης, η εφαρμογή, θα χρησιμοποιήσει την κάμερα της συσκευής για να σαρώσει τον περιβάλλοντα χώρο. Ως περιβάλλοντας χώρος, νοείται το οπτικό πεδίο του φακού της κάμερας. Με τον τρόπο αυτό, η εφαρμογή αναζητά στον χώρο πινακίδες κυκλοφορίας και στην περίπτωση που σε κάποιο στιγμιότυπο εμπεριέχεται πινακίδα κυκλοφορίας, την απομονώνει και επιχειρεί να την αναγνωρίσει[37]. Οι πινακίδες που αναγνωρίστηκαν επιτυχώς[38], προβάλλονται στον χρήστη, στην περιοχή της λίστας ανιχνεύσεων και πληροφοριών. Σημειώνεται ότι, μέσω της περιοχής προεπισκόπησης (preview), ο χρήστης επιβλέπει σε πραγματικό χρόνο, καθ' όλη τη διάρκεια της σάρωσης, το οπτικό πεδίο της εφαρμογής, ώστε να το προσαρμόζει κατάλληλα[39].

Περιγραφή εμφάνισης

Η περιοχή προβολής της οθόνης, απαρτίζεται από τα ακόλουθα στοιχεία:

- Πεδίο προεπισκόπησης (preview)
- Πεδίο προβολής ανιχνεύσεων
- Λίστα ανιχνεύσεων και πληροφοριών

Η περιοχή προεπισκόπησης, έχει τοποθετηθεί στο πάνω άκρο της οθόνης και καταλαμβάνει τμήμα πλάτους 200dp ενώ εκτείνεται σε όλο το μήκος της. Διακρίνονται δύο μεγεθυντικοί φακοί, που χρησιμοποιούνται για τον έλεγχο της μηχανικής ή/και ψηφιακής εστίασης[40] (digital zoom functionality) της κάμερας της συσκευής. Ακριβώς κάτω από την περιοχή προεπισκόπησης, βρίσκεται το πεδίο προβολής των ανιχνεύσεων. Στο πεδίο αυτό, προβάλλονται οι πινακίδες που εντοπίζονται επιτυχώς από την εφαρμογή. Τέλος, το υπόλοιπο τμήμα της οθόνης, καταλαμβάνει η λίστα ανιχνεύσεων και πληροφοριών. Εδώ, προβάλλονται με χρονολογική σειρά εντοπισμού, οι πινακίδες που

[36] Από κατασκευής, έχει οριστεί να χρησιμοποιείται για την λειτουργία η βασική οπίσθια κάμερα. Επομένως αν η συσκευή διαθέτει περισσότερες από μία οπίσθιες κάμερες, η εφαρμογή χρησιμοποιεί εκείνη η οποία είναι ορισμένη από τον κατασκευαστή ως βασική.

[37] Ο εντοπισμός και η αναγνώριση των πινακίδων κυκλοφορίας, επιτυγχάνονται με τη βοήθεια των ενσωματωμένων νευρωνικών δικτύων (on-device ML) που διαθέτει η εφαρμογή. Υπάρχουν τεχνικές που επιτρέπουν την μερική εκτέλεση των διαδικασιών απομακρυσμένα, είτε σε κάποιον cloud server της Google, είτε στον δικό μας server. Ωστόσο, επιλέχθηκε η χρήση της εμφωλευμένης διαδικασίας. Έτσι λοιπόν, αρχικά, μέσω της κάμερας, τροφοδοτούνται διαδοχικά στιγμιότυπα στο πρώτο νευρωνικό δίκτυο, το οποίο, αναλαμβάνει να ελέγξει αν υπάρχει σε αυτά κάποια πινακίδα κυκλοφορίας. Εάν το πρώτο δίκτυο εντοπίσει πινακίδα κυκλοφορίας, την απομονώνει, και στη συνέχεια, τροφοδοτεί το δεύτερο νευρωνικό δίκτυο με την εικόνα της πινακίδας. Το δεύτερο νευρωνικό δίκτυο, είναι εκείνο το τμήμα της εφαρμογής το οποίο αναλαμβάνει την ανάγνωσή της.

[38] Η εφαρμογή έχει βελτιστοποιηθεί για την αναγνώριση πινακίδων που ταιριάζουν στα Ελληνικά πρότυπα. Ως εκ τούτου, πινακίδες άλλων κρατών απορρίπτονται.

[39] Επειδή, όπως έχει ήδη αναφερθεί, η αποτελεσματικότητα των συστημάτων ALPR επηρεάζεται από την κάμερα, τους φακούς της, τον φωτισμό κλπ , ως βέλτιστη πρακτική προτείνεται η συσκευή να τοποθετείται σε όρθια θέση (90ο) και όσο το δυνατό πιο κοντά στο ύψος των πινακίδων κυκλοφορίας, έτσι ώστε να είναι πιο ευκρινείς οι εικόνες που λαμβάνονται από την συσκευή.

[40] Εξαρτάται από τα τεχνικά χαρακτηριστικά της συσκευής.

ανιχνεύθηκαν, συνοδευόμενες από πληροφορίες που αφορούν στην ημερομηνία και την τοποθεσία εντοπισμού. Επιπλέον, όλα τα στοιχεία της λίστας, διαθέτουν κουμπί - σύνδεσμο για την εφαρμογή Google Maps, όπου, επάνω στον προβαλλόμενο χάρτη, έχει επισημανθεί το σημείο όπου ανιχνεύθηκε η συγκεκριμένη πινακίδα. Η ιδιαιτερότητα της λίστας, έγκειται στον χρωματισμό των στοιχείων της. Κάθε στοιχείο της λίστας συνοδεύεται και από μια σειρά λειτουργιών, ανάλογα με την χρωματική του σήμανση.

Χρωματική σήμανση - Παροχή ενδεικτικών περιπτώσεων χρήσης

Λευκός χρωματισμός: Η πινακίδα κυκλοφορίας ανιχνεύεται για πρώτη φορά από την συσκευή και δεν είναι καταχωρισμένη στη λίστα των αναζητούμενων πινακίδων, τόσο τη δική μας, όσο και των υπόλοιπων χρηστών της εφαρμογής[41].

Κόκκινος χρωματισμός: Η εφαρμογή ανίχνευσε πινακίδα την οποία αναζητούμε εμείς ή άλλος χρήστης (την έχει δηλώσει ως public κατά την καταχώρισή της). Πατώντας στο κόκκινο στοιχείο της λίστας, εισερχόμαστε σε ένα διαφορετικό περιβάλλον προβολής. Στην περίπτωση όπου η πινακίδα έχει καταχωριστεί από εμάς, τότε, περνάμε σε προβολή των λεπτομερειών της καταχώρισής μας και μπορούμε να την τροποποιήσουμε ή και να την διαγράψουμε αν επιθυμούμε. Σε διαφορετική περίπτωση, δηλαδή όταν η καταχώριση είναι άλλου ή άλλων[42] χρηστών της εφαρμογής, τότε, περνάμε σε περιβάλλον προβολής λίστας, με τον αναζητούμενο αριθμό κυκλοφορίας και τα σχόλια που έχει εισάγει ο εκάστοτε χρήστης για την συγκεκριμένη εγγραφή. Παράλληλα, ο χρήστης ή οι χρήστες που έκαναν την σχετική καταχώρηση, ενημερώνονται[43].

Επομένως, η κόκκινη σήμανση, αφορά στον εντοπισμό πινακίδων κυκλοφορίας που έχουν καταχωριστεί ως αναζητούμενες από εμάς ή άλλον χρήστη της εφαρμογής. Ενδεικτικά, μπορούμε να αναφέρουμε περιπτώσεις όπως, ο εντοπισμένος ενός κλεμμένου οχήματος, ή ενός οχήματος του οποίο ο οδηγός πραγματοποιούσε επικίνδυνους ελιγμούς, με αποτέλεσμα, κάποιος από τους χρήστες της εφαρμογής, να καταχώρισε την πινακίδα του προκειμένου να μας επιστήσει την προσοχή, ή ακόμη και την περίπτωση εντοπισμού οχήματος το οποίο χρησιμοποιήθηκε στη διάπραξη κάποιας εγκληματικής ενέργειας (ληστεία, διάρρηξη, απαγωγή κ.α.).

Κίτρινος χρωματισμός: Με τον τρόπο αυτό, επιχειρείται η προληπτική ενημέρωση του χρήστη, αναφορικά με τον εντοπισμό πινακίδας κυκλοφορίας η οποία, αν και δεν είναι καταχωρισμένη από κάποιον χρήστη, έχει εντοπιστεί από την εφαρμογή περισσότερες από μία φορές. Διευκρινίζεται ότι, ο εντοπισμός αφορά αποκλειστικά και μόνο στον χρήστη της συγκεκριμένης συσκευής. Αυτό πρακτικά σημαίνει ότι, αν οποιοσδήποτε άλλος χρήστη, εντόπισε την ίδια πινακίδα κάποια στιγμή και έπειτα, την εντοπίσαμε κάπου τυχαία και εμείς, η χρωματική σήμανση του στοιχείου της στη λίστα μας, δεν θα είναι κίτρινη αλλά λευκή. Επιπλέον, επισημαίνεται ότι, η εφαρμογή έχει παραμετροποιηθεί με τρόπο τέτοιο ώστε, για να θεωρηθεί ως επαναλαμβανόμενος ο εντοπισμός μιας πινακίδας, θα πρέπει να συντρέχει τουλάχιστον μία, από τις ακόλουθες προϋποθέσεις:

1. Έχει παρέλθει χρονικό διάστημα τουλάχιστον 2 λεπτών από κάθε προηγούμενη όμοια ανίχνευση.
2. Οι ανιχνεύσεις πραγματοποιήθηκαν σε διαφορετικές τοποθεσίες ή σε τοποθεσίες που απέχουν μεταξύ τους απόσταση, τουλάχιστον, 100 μέτρων.

Επειδή, όπως αναφέραμε, το χαρακτηριστικό της κίτρινης σήμανσης ενός στοιχείου της λίστας ανιχνεύσεων, έχει ως στόχο την επισήμανσή του, αλλά και την παρότρυνση του χρήστη να παρατηρήσει πιο προσεκτικά τη συμπεριφορά που σχετίζεται με το συγκεκριμένο όχημα, ως ενδεικτικές περιπτώσεις πρακτικής χρησιμότητας, μπορούν, μεταξύ άλλων, να αναφερθούν, εκείνη ενός stalker που μας ακολουθεί με το όχημά του ή συχνάζει στα μέρη που βρισκόμαστε, αλλά και η

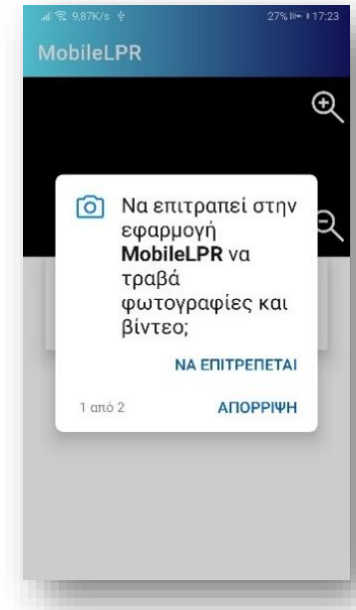
[41] Σημειώνεται ότι, η πινακίδα θα μπορούσε να είναι καταχωρισμένη ως αναζητούμενη στην ιδιωτική λίστα κάποιου χρήστη, αλλά να μην είναι δημόσια κοινοποιήσιμη, δηλαδή να έχει χαρακτηριστεί ως no public. Σε αυτή την περίπτωση, ο χρήστης που δημιούργησε την καταχώρηση, δεν ενημερώνεται ότι εντοπίστηκε, αλλά ούτε ο χρήστης που την εντόπισε ενημερώνεται ότι η πινακίδα αναζητείται.

[42] Έχει ληφθεί υπόψη η περίπτωση όπου, μία πινακίδα έχει καταχωριστεί από περισσότερους του ενός χρήστες.

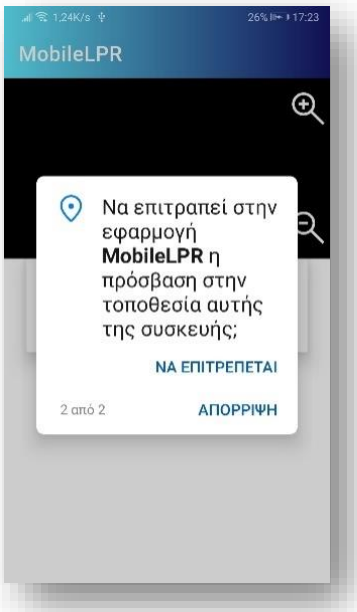
[43] Η ενημέρωση αφορά στην ημερομηνία, την ώρα και την τοποθεσία εντοπισμού. Πληροφορίες που αφορούν στο πρόσωπο ή τη συσκευή που εντόπισε την αναζητούμενη πινακίδα, δεν παρέχονται.

περίπτωση ενός πιθανού απαγωγέα, ο οποίος επιχειρεί να καταγράψει την καθημερινή μας ρουτίνα και τα δρομολόγιά που ακολουθούμε.

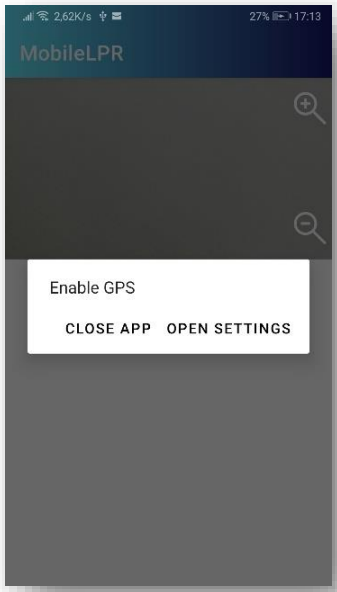
Σημειώνεται ότι, τα οχήματα και οι πινακίδες τους, όπως αυτά εμφαίνονται στις εικόνες που ακολουθούν, προέρχονται από υλικό ελεύθερα προσβάσιμο στο διαδίκτυο και η χρήση τους έχει αποκλειστικά και μόνο εκπαιδευτικό χαρακτήρα.



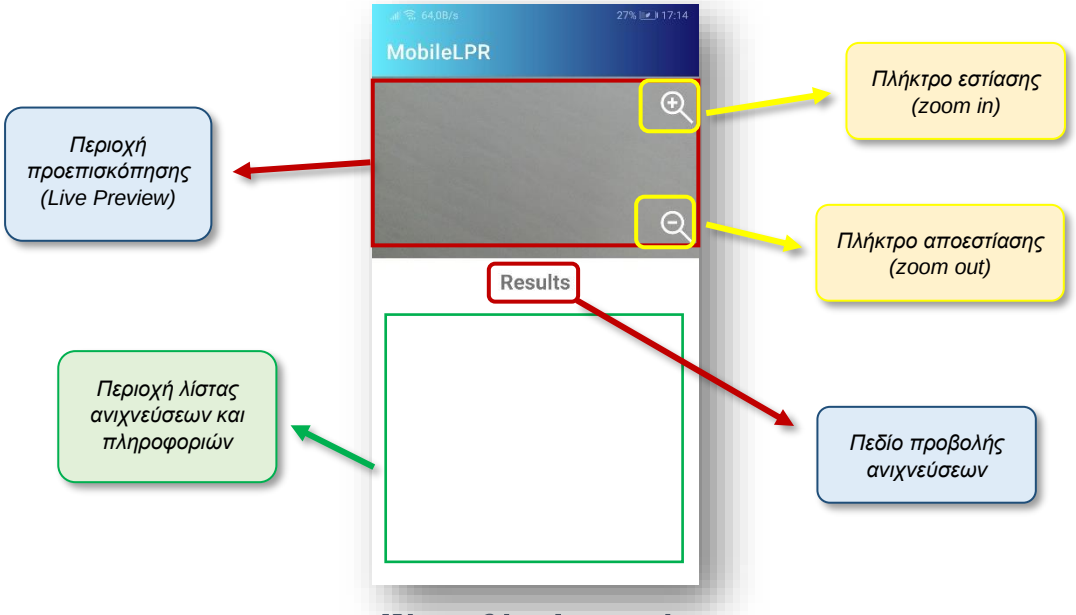
Αίτημα άδειας χρήσης κάμερας



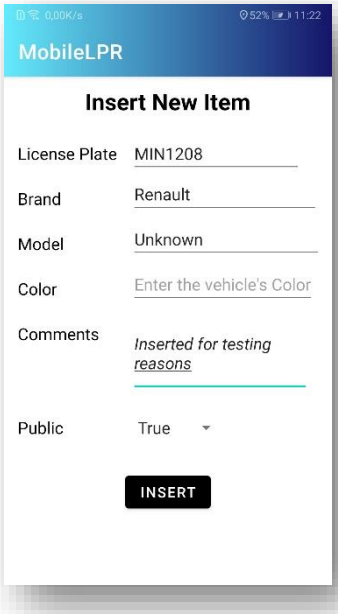
Αίτημα άδειας χρήσης GPS



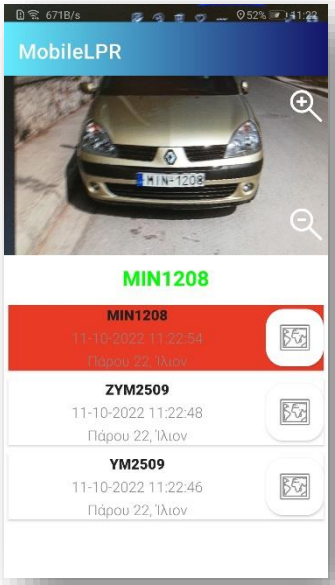
Οθόνη προτροπής ενεργοποίησης GPS



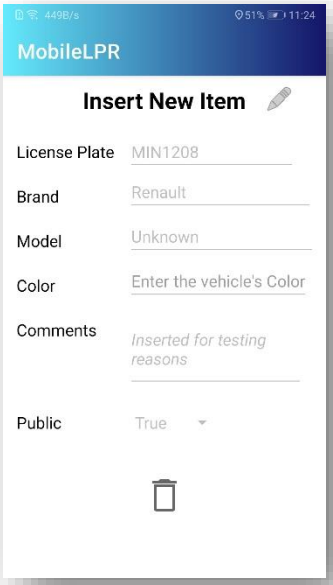
Κύρια οθόνη λειτουργίας Go Live!



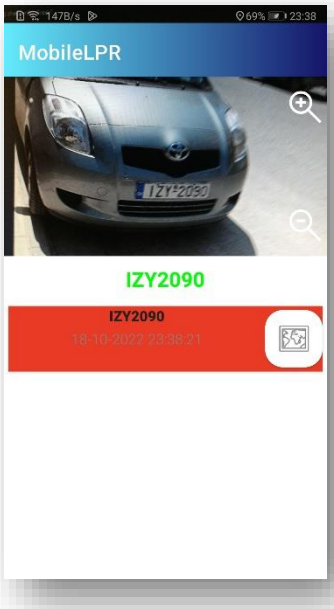
Οθόνη καταχώρισης εγγραφής από στοιχείο λίστας με λευκό χρωματισμό



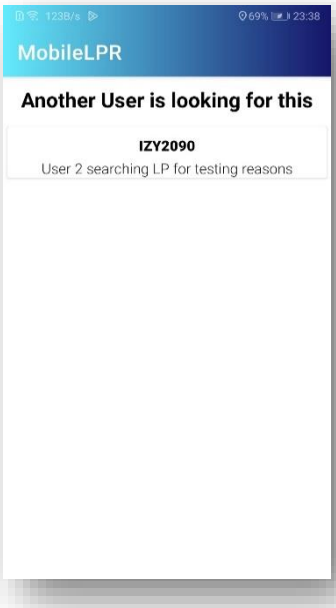
Επίδειξη πραγματικής λειτουργίας – Στοιχείο λίστας με κόκκινο χρωματισμό



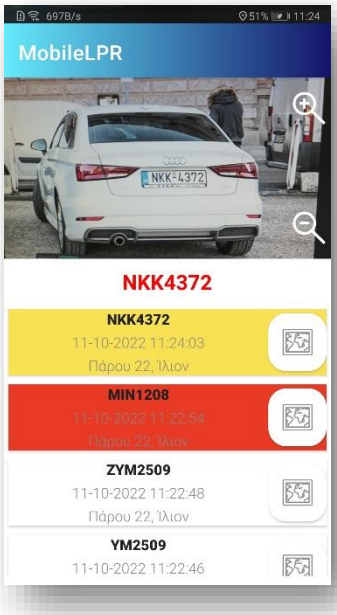
Οθόνη προβολής λεπτομερειών για στοιχείο λίστας με κόκκινο χρωματισμό – Πινακίδα αναζητούμενη από τον χρήστη



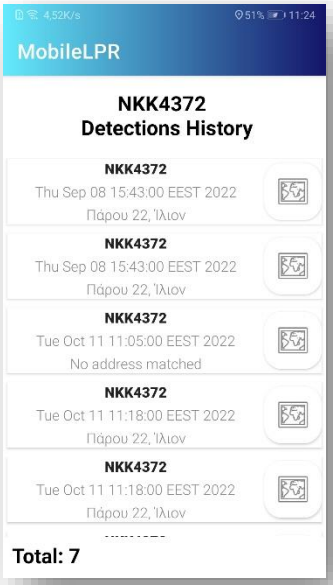
Επίδειξη πραγματικής λειτουργίας – Πινακίδα αναζητούμενη από έτερο χρήστη



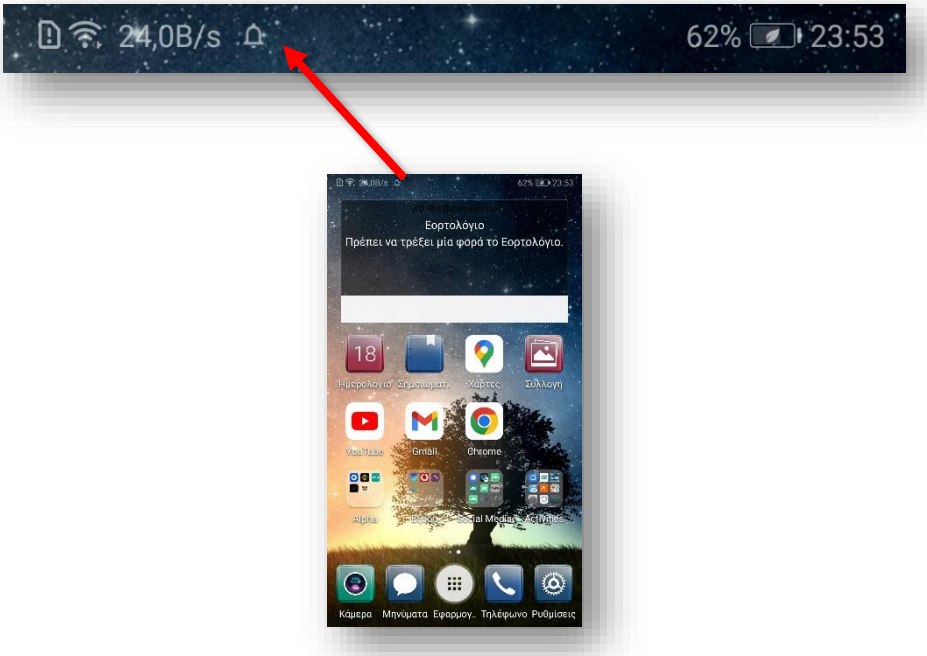
Οθόνη προβολής λεπτομερειών για στοιχείο λίστας με κόκκινο χρωματισμό – Πινακίδα αναζητούμενη από έτερο χρήστη



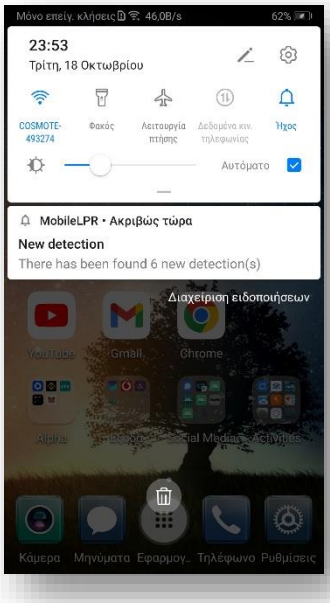
Στιγμιότυπο οθόνης λειτουργίας Go Live! – Επίδειξη κίτρινου, κόκκινου και λευκού χρωματισμού στοιχείου



Οθόνη προβολής ιστορικού για στοιχείο της λίστας με κίτρινο χρωματισμό



Στιγμιότυπο οθόνης για λήψη ειδοποίησης



Στιγμιότυπο οθόνης για λήψη ειδοποίησης – Περιοχή ειδοποιήσεων

Οθόνη ειδοποιήσεων

Log Search

Είναι η λειτουργία αναζήτησης ιστορικού. Η εφαρμογή, προβαίνει σε αναζήτηση του αρχείου καταγραφής ανιχνεύσεων που αφορά στη συγκεκριμένη συσκευή, χρησιμοποιώντας χωροχρονικά κριτήρια. Πιο συγκεκριμένα, ο χρήστης εισάγει το επιθυμητό χρονικό διάστημα αναζήτησης (ημερομηνία και ώρα αφετηρίας – τερματισμού αναζήτησης), καθώς και τα χωρικά κριτήριά της (τοποθεσία και ακτίνα αναζήτησης) και το σύστημα αποκρίνεται, εμφανίζοντας τη λίστα των πινακίδων κυκλοφορίας που εντοπίστηκαν από τον χρήστη (χρησιμοποιούμενη συσκευή), το συγκεκριμένο χρονικό διάστημα και εντός του γεωχωρικού τόπου που υποδείχθηκε.

Περιγραφή οθόνης και λειτουργικότητας γεωχωρικής αναζήτησης

Η συγκεκριμένη προβολή αποτελείται από τους επιλογείς χρονοσήμανσης (ημερομηνία και ώρα) και τον επιλογέα τοποθεσίας. Οι επιλογείς χρονοσήμανσης, διακρίνονται στον επιλογέα ημερομηνίας και ώρας έναρξης του διαστήματος αναζήτησης και στον επιλογέα ημερομηνίας και ώρας λήξης του διαστήματος αναζήτησης. Πατώντας στον επιλογέα, εμφανίζεται ένα ημερολόγιο και στη συνέχεια ένα ρολόι, απ' όπου επιλέγεται το επιθυμητό διάστημα.

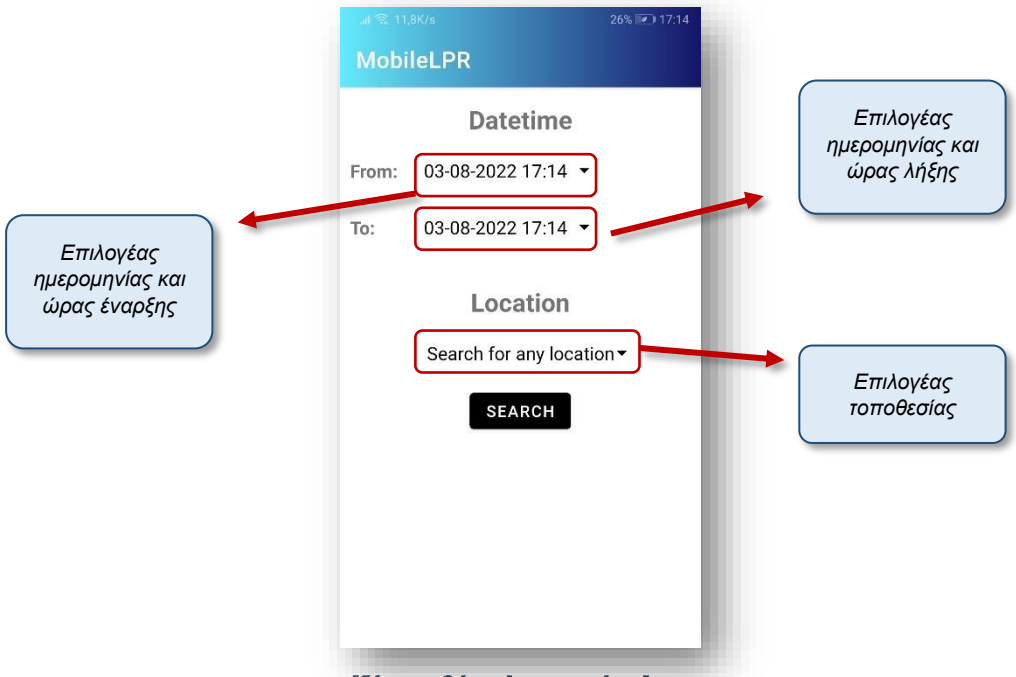
Αντίστοιχα, ο επιλογέας τοποθεσίας, ορίζει το κέντρο και την ακτίνα της γεωχωρικής αναζήτησης. Πατώντας τον επιλογέα τοποθεσίας, ο χρήστης μεταφέρεται σε έναν ενσωματωμένο χάρτη. Στην περίπτωση όπου ο χρήστης έχει ενεργοποιημένο το GPS της συσκευής, τότε, πιέζοντας το σύμβολο στο πάνω δεξί μέρος της οθόνης, ο χρήστης μεταφέρεται στην τοποθεσία που βρίσκεται εκείνη τη στιγμή η συσκευή. Από το πεδίο που βρίσκεται πάνω αριστερά στην οθόνη, ο χρήστης μπορεί να εισάγει την ακτίνα, σε μέτρα, εντός της οποίας επιθυμεί να επεκταθεί η αναζήτησή του, ενώ πιέζοντας στο κουμπί εμφανίζεται μία προεπισκόπηση του γεωχώρου που περικλείει η ακτίνα που όρισε. Σημειώνεται ότι, αν ο χρήστης μετακινήσει το κέντρο του χάρτη, προτού οριστικοποιήσει τις επιλογές του (κουμπί OK), τότε, θα μεταβληθεί και το κέντρο που είχε επιλεγεί, αντιστοιχίζοντάς το, στο κέντρο που εμφαίνεται στον χάρτη. Όταν πλέον ο χρήστης οριστικοποιήσει την επιλογή του, επανέρχεται στη βασική προβολή αναζήτησης, όπου πλέον, αναγράφεται η διεύθυνση που αντιστοιχεί στο κέντρο που όρισε ο χρήστης, καθώς και η ακτίνα που επέλεξε.

Η αναζήτηση οριστικοποιείται και αποστέλλεται για επεξεργασία μέσω του πλήκτρου «Search». Η απόκριση του συστήματος, ενημερώνει τον χρήστη αναφορικά με το σύνολο των αποτελεσμάτων της αναζήτησης, καθώς και την λίστα που αντιστοιχεί σε αυτά. Φυσικά, σε αρνητική περίπτωση λαμβάνει ανάλογη ενημέρωση.

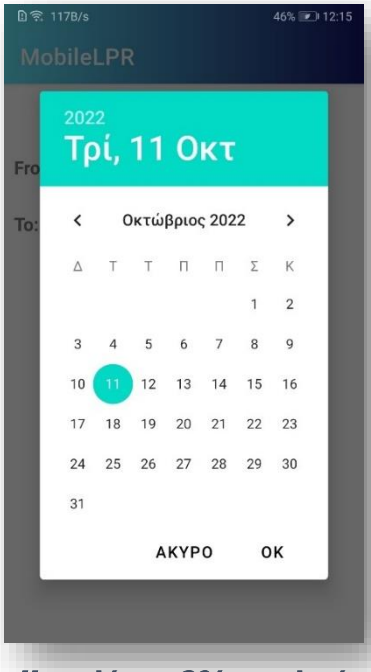
Σε κάθε στοιχείο της λίστας αναγράφεται η πινακίδα κυκλοφορίας, η χρονική σήμανση και η διεύθυνση που εντοπίστηκε. Επιπλέον, πιέζοντας στο κουμπί με το σύμβολο του χάρτη, ο χρήστης μεταφέρεται στην εφαρμογή Google Maps, όπου προβάλλεται το σημείο εντοπισμού της συγκεκριμένης εγγραφής, ενώ παράλληλα, παρέχονται όλες οι λειτουργικότητες της εφαρμογής Google Maps (πλοήγηση, πλησιέστερα σημεία κ.λ.π). Τέλος, επιλέγοντας στο κυρίως σώμα του εκάστοτε στοιχείου, εμφανίζεται μία νέα λίστα, η οποία αφορά στο ιστορικό ανιχνεύσεων της συγκεκριμένης πινακίδας κυκλοφορίας, συνολικά, ανεξαρτήτως χωροχρονικής σήμανσης. Και πάλι, τα στοιχεία της νέας αυτής λίστας, διαθέτουν τις ίδιες πληροφορίες με την προηγούμενη (πινακίδα κυκλοφορίας, χωροχρονική σήμανση και πλήκτρο πλοήγησης στο Google Maps).

Η αναζήτηση ιστορικού, είναι μια λειτουργία με πολλαπλές χρησιμότητες. Έστω ότι πέσαμε θύματα διάρρηξης. Η αναζήτηση των οχημάτων που εντοπίστηκαν να κινούνται γύρω από το σπίτι μας τις τελευταίες ημέρες, θα ήταν μια καλή αρχή. Ανάλογη θα ήταν και η περίπτωση όπου, αντιλαμβανόμαστε πως, την ώρα που έγινε κάποια εγκληματική ενέργεια, όπως για παράδειγμα, μια κλοπή ή μια ληστεία, έτυχε να περνάμε από εκεί ή έστω πολύ κοντά. Το ιστορικό με τα οχήματα που εντοπίσαμε στη γύρω περιοχή κατά τη διέλευσή μας, μπορεί να αποτελούν σημαντικά στοιχεία για την εξιχνίαση της πράξης. Αλλά και στην περίπτωση της αναζήτησης του σημείου όπου σταθμεύσαμε το όχημά μας, το ιστορικό των ανιχνεύσεων, μπορεί να φανεί ιδιαίτερα χρήσιμο. Έστω ότι, μετά από αρκετή ώρα αναζήτησης χώρου στάθμευσης, καταφέραμε και βρήκαμε μια θέση κάπου. Όμως, δεν γνωρίζουμε καλά το μέρος και ανησυχούμε ότι θα δυσκολευτούμε να ξαναβρούμε το όχημά μας. Σε αυτή την περίπτωση, σαρώνουμε τη δική μας πινακίδα κυκλοφορίας και όταν έρθει η ώρα της

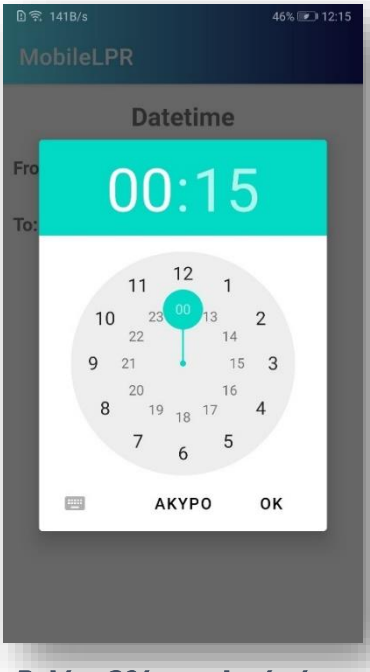
επιστροφής, ανατρέχουμε στο ιστορικό των σαρώσεών μας, απ' όπου, με τη βοήθεια της πλοήγησης (εικονίδιο χάρτη), οδηγούμαστε πίσω στο όχημά μας.



Κύρια οθόνη λειτουργίας Log Search

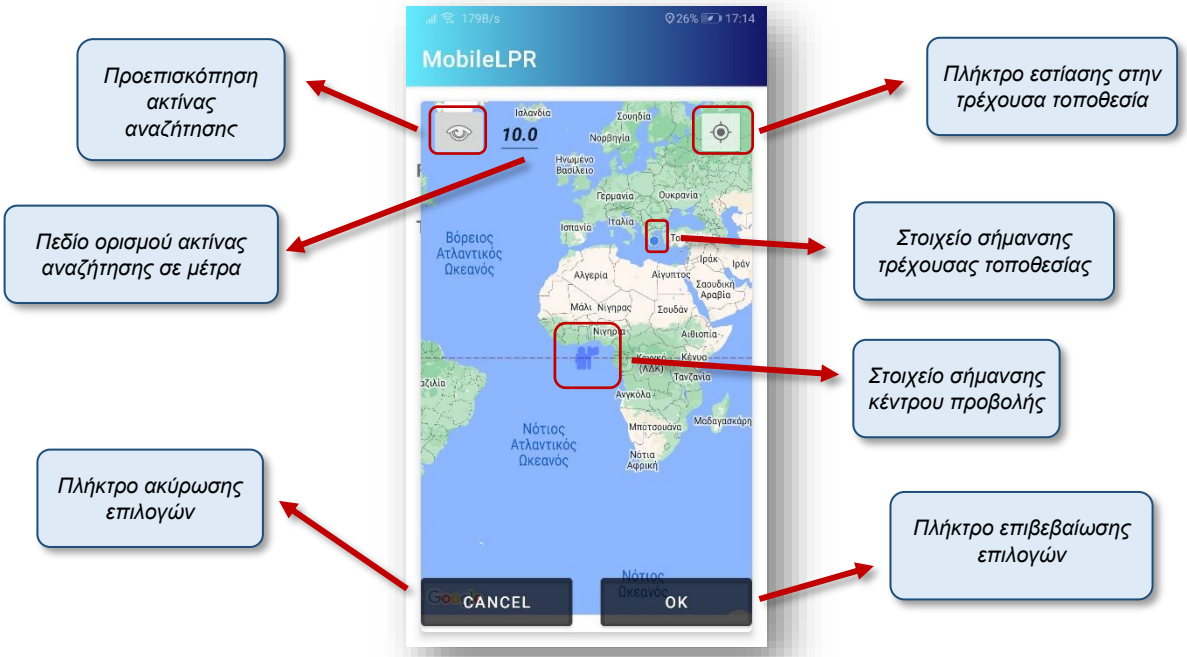


Ημερολόγιο – Οθόνη επιλογής ημερομηνίας αναζήτησης

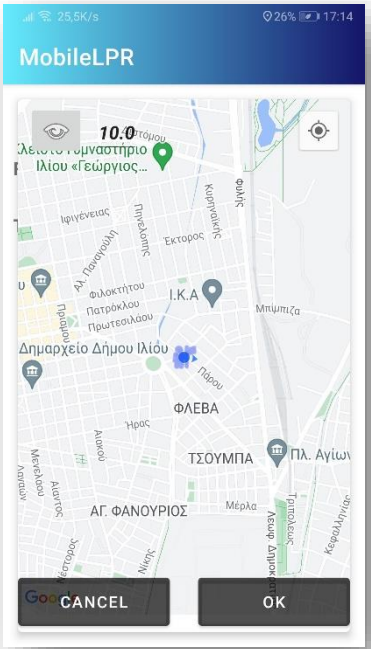


Ρολόι – Οθόνη επιλογής ώρας αναζήτησης

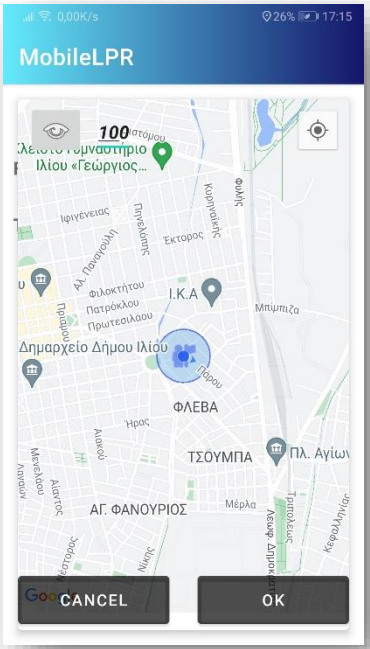
Οθόνη προβολής ενσωματωμένου χάρτη, χωρίς επισήμανση τρέχουσας τοποθεσίας



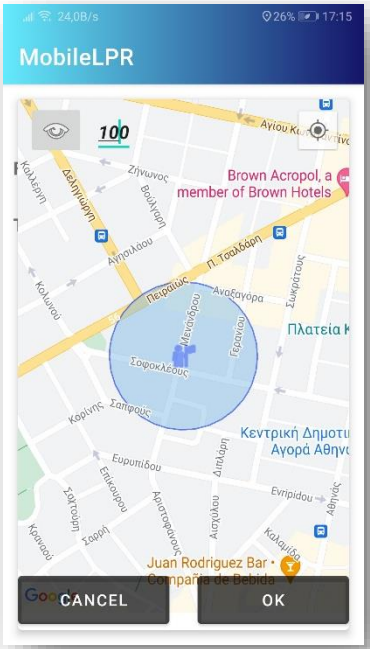
Οθόνη προβολής ενσωματωμένου χάρτη, με επισήμανση τρέχουσας τοποθεσίας



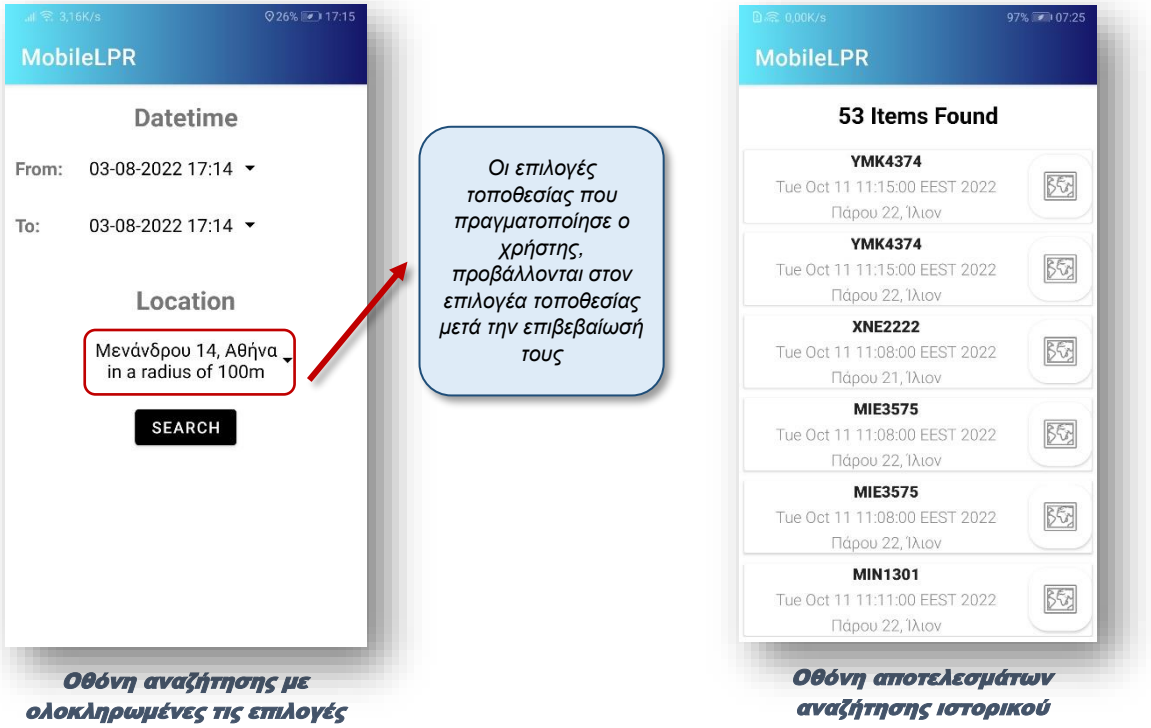
Οθόνη προβολής χάρτη, με εστίαση στην τρέχουσα τοποθεσία



Τρέχουσα τοποθεσία και προεπισκόπηση ακτίνας αναζήτησης (100m)



Επιλογή σημείου στον χάρτη και προεπισκόπηση ακτίνας αναζήτησης



Οθόνη αναζήτησης με ολοκληρωμένες τις επιλογές

Οθόνη αποτελεσμάτων αναζήτησης ιστορικού

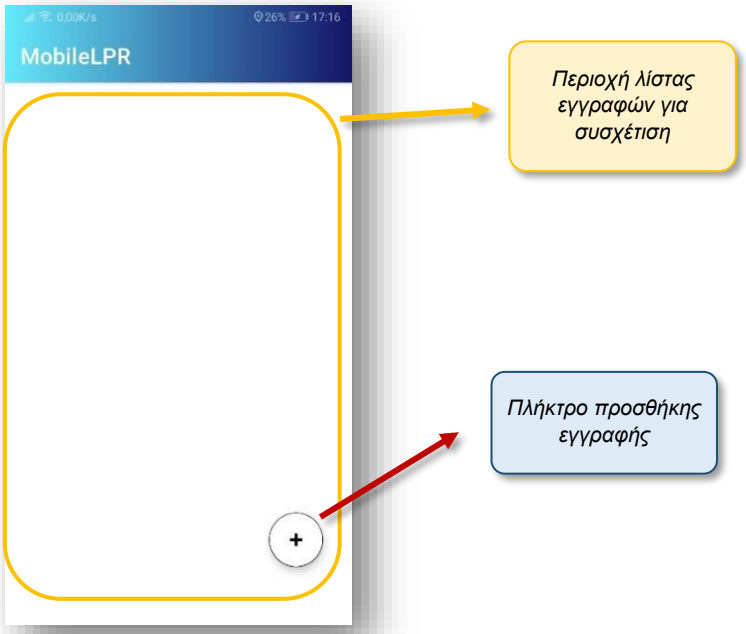
Correlation

Είναι η λειτουργία συσχέτισης των εγγραφών ιστορικού της συγκεκριμένης συσκευής. Ειδικότερα, ο χρήστης μπορεί να εισάγει περισσότερα του ενός χωροχρονικά δεδομένα και το σύστημά μας, αφού αντιπαραβάλλει τα αποτελέσματα, θα εντοπίσει, εφόσον υπάρχουν, τις κοινές εγγραφές. Έστω, για παράδειγμα, ότι έχουν γίνει τρείς ληστείες, σε τρείς διαφορετικές περιοχές, τρείς διαφορετικές ημέρες ή και την ίδια. Θα είχε ενδιαφέρον να γνωρίζουμε αν σε αυτά τα τρία διαφορετικά χωροχρονικά στιγμιότυπα, υπάρχει κάποιο κοινό όχημα που βρέθηκε κοντά ή στο σημείο της ληστείας. Η λειτουργία, λοιπόν, της συσχέτισης, εκτελεί με ημι-αυτοματοποιημένο τρόπο, ακριβώς αυτή τη μορφή αναζήτησης.

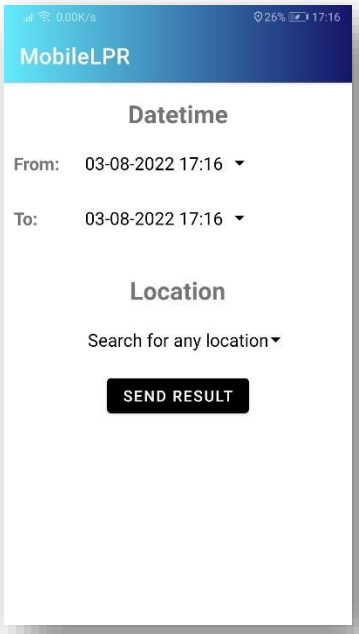
Η προβολή οθόνης της συγκεκριμένης λειτουργίας, είναι αρκετά λιτή. Ενεργοποιώντας την, ο χρήστης αντικρίζει μονάχα ένα πλήκτρο με το σύμβολο + στο κάτω δεξί άκρο της οθόνης. Πατώντας το, εισέρχεται σε περιβάλλον οθόνης, όμοιο με εκείνο της λειτουργίας αναζήτησης ιστορικού (Log Search). Από εκεί, ο χρήστης επιλέγει, όπως και στην περίπτωση της αναζήτησης του ιστορικού, την ημερομηνία και την ώρα έναρξης και λήξης του διαστήματος αναζήτησης, καθώς και την γεωχωρική περιοχή όπου αναζητούμε αποτελέσματα. Στη συνέχεια, αφού οριστικοποιήσει τις επιλογές του, επανέρχεται στην οθόνη συσχέτισης. Οι επιλογές του, εισάγονται σε μία λίστα συσχετισμού, η οποία μπορεί είτε να υποβληθεί για αντιπαραβολή (Search), είτε να τροποποιηθεί, εισάγοντας κάποιο ακόμη στοιχείο ή διαγράφοντας κάποιο από τα υπάρχοντα. Η διαγραφή των στοιχείων γίνεται με την κύλιση του στοιχείου προς τα αριστερά (swipe left). Σε αυτή την περίπτωση, αναδύεται ένα σύντομο μήνυμα, που ενημερώνει τον χρήστη ποιο στοιχείο της λίστας του διαγράφηκε, δίνοντάς του, παράλληλα, τη δυνατότητα, για σύντομο χρονικό διάστημα, να αναιρέσει την ενέργειά του, επαναφέροντας το στοιχείο.

Όταν πλέον ο χρήστης έχει συμπληρώσει τη λίστα του, πατώντας στο κουμπί «Search», την υποβάλλει για αντιπαραβολή. Μόλις ολοκληρωθεί η διαδικασία συσχετισμού, ο χρήστης μεταφέρεται σε νέα οθόνη προβολής, όπου παρουσιάζονται, αν υπάρχουν, τα αποτελέσματα. Στη νέα οθόνη, αναγράφεται το πλήθος των κοινών πινακίδων που εντοπίστηκαν και παράλληλα, παρουσιάζεται η αντίστοιχη λίστα. Η λίστα περιέχει ως στοιχεία της, μονάχα τους αλφαριθμητικούς χαρακτήρες των

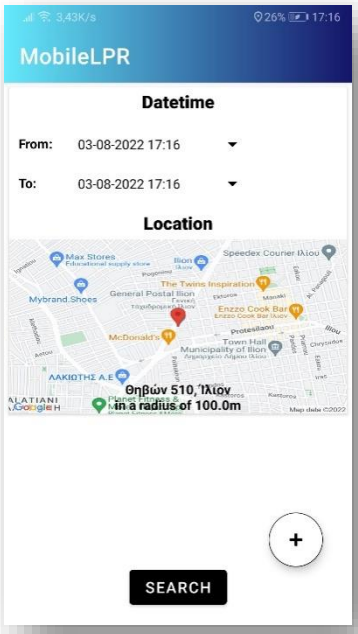
πινακίδων κυκλοφορίας, χωρίς περαιτέρω πληροφορίες. Αν, ωστόσο, ο χρήστης επιθυμεί να μάθει περισσότερες πληροφορίες σχετικά με κάποια συγκεκριμένη πινακίδα της προβαλλόμενης λίστας, μπορεί, επιλέγοντάς το, να λάβει όλο το ιστορικό που αφορά στην συγκεκριμένη πινακίδα, ανεξάρτητα από τα χωροχρονικά κριτήρια που είχε επιλέξει κατά την διαδικασία της συσχέτισης[44].



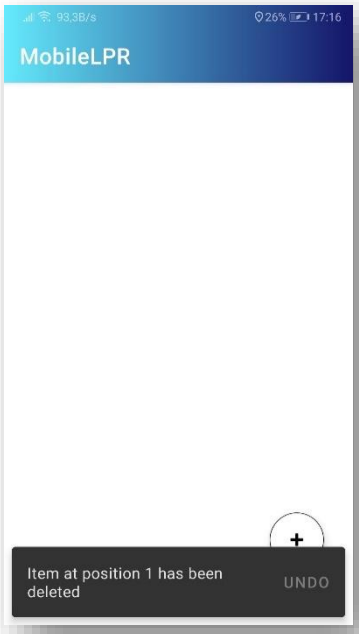
Κύρια οθόνη λειτουργίας Correlation



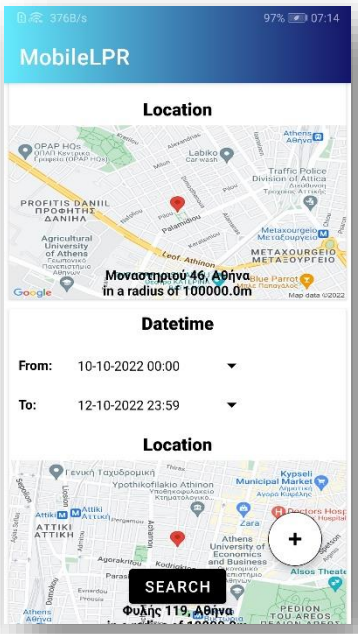
Οθόνη επιλογής χωροχρονικών δεδομένων



Οθόνη Correlation μετά την εισαγωγή εγγραφής

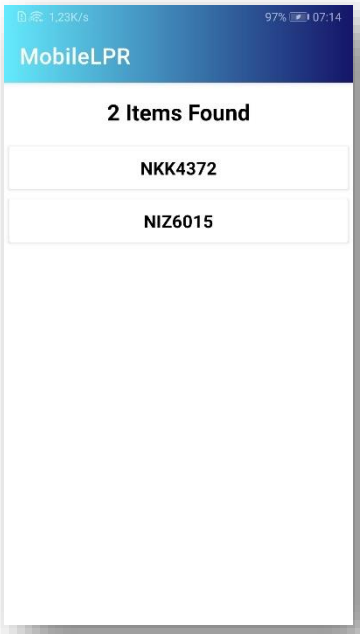


Μήνυμα ειδοποίησης αφαίρεσης εγγραφής

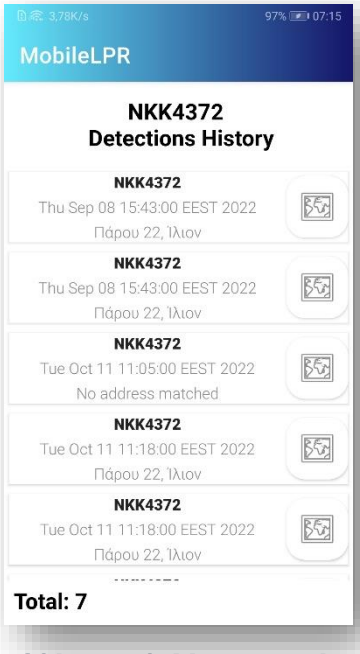


Οθόνη Correlation με πολλαπλές εγγραφές

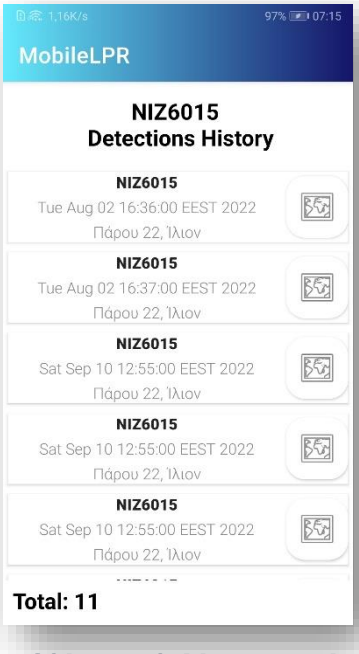
[44] Βλ. σχετικά εικόνες σελ. 31.



Οθόνη αποτελεσμάτων συσχέτισης



Οθόνη προβολής ιστορικού για στοιχείο από τη λίστα αποτελεσμάτων



Οθόνη προβολής ιστορικού για στοιχείο από τη λίστα αποτελεσμάτων

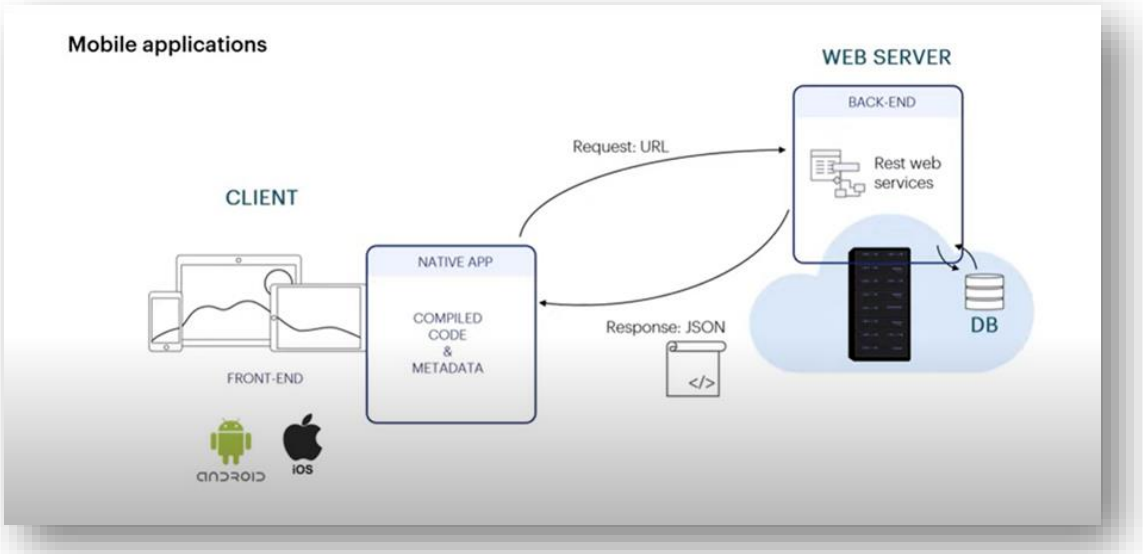
Τεχνική ανάλυση συστήματος

Το σύστημά μας έχει δομηθεί στη λογική της αρχιτεκτονικής των 3ων επιπέδων (3-tier architecture). Αποτελείται από ένα τμήμα που «τρέχει» στον «πελάτη» (client-side), εν προκειμένω στην κινητή συσκευή Android, και ένα τμήμα που «τρέχει» στον «εξυπηρετητή» (server-side), παρέχοντας πληροφορίες στον «πελάτη».

Το client-side τμήμα ενσωματώνει το σύνολο της business logic, καθώς επίσης και όλα τα απαραίτητα δεδομένα για την διαμόρφωση του User Interface (UI). Αντίστοιχα, στο server-side τμήμα, μεταξύ άλλων, σχεδιάστηκε και αναπτύχθηκε μία NoSQL βάση δεδομένων (DB) που καλύπτει τις ανάγκες αποθήκευσης των δεδομένων που συλλέγονται και επεξεργάζονται από το σύστημα. Το client-side τμήμα, και κατ' επέκταση η εφαρμογή που τρέχει στην εκάστοτε κινητή συσκευή, δεν αποκτά απευθείας πρόσβαση στη βάση δεδομένων του συστήματος. Αντ' αυτού, η γεφύρωσή τους επιτυγχάνεται μέσω του service-layer.

Το service-layer αποτελείται από κατάλληλα διαμορφωμένες υπηρεσίες (RESTful web services) που τρέχουν στο server-side τμήμα και επιτρέπουν την επικοινωνία (πρόσβαση) και γενικότερα την οποιαδήποτε τροποποίηση των δεδομένων που υπάρχουν στη βάση, ως απόκριση του συστήματος σε προκαθορισμένες στις ενέργειες του πελάτη (client-side). Είναι σημαντικό να επισημανθεί ότι, το service-layer, είναι ανεξάρτητο από την συσκευή (desktop, laptop, tablet κλπ) και το λογισμικό που αυτή χρησιμοποιεί (Android, IOS, Windows, Linux κλπ).

Έτσι λοιπόν, όταν ο χρήστης τρέξει την εφαρμογή μας στην κινητή του συσκευή, η εφαρμογή θα επικοινωνήσει με τον server μέσω του service-layer, θα εκτελεστούν σε αυτόν οι λειτουργίες που αντιστοιχούν στις ενέργειες του χρήστη, και στη συνέχεια, θα επιστρέψουν τα απαραίτητα δεδομένα στην εφαρμογή, δηλαδή στο client-side τμήμα, όπου θα επεξεργαστούν και τελικά θα προβληθούν στον χρήστη μέσω της συσκευής.



Αναπαράσταση διαλειτουργικότητας server side – client side εφαρμογής για κινητές συσκευές

Όπως εύκολα συμπεραίνεται από τα παραπάνω, το σύστημά μας συγκαταλέγεται στις online λύσεις και επομένως, προϋποθέτει την ύπαρξη σύνδεσης στο διαδίκτυο -τόσο του client όσο και του server- προκειμένου να εκτελεστούν οι λειτουργίες του.

Τα μειονεκτήματα που συνοδεύουν την επιλογή ανάπτυξης του συστήματός μας, στη βάση της φιλοσοφίας της αρχιτεκτονικής των 3ων επιπέδων, αντισταθμίζονται και τελικά υπερκαλύπτονται από τα οφέλη που αυτή προσφέρει. Έτσι, η πολυπλοκότητα στη σχεδίαση και το κόστος υλοποίησης, αντισταθμίζονται από την ευελιξία, την αυξημένη ασφάλεια, την ευκολία στη συντήρηση, την επεκτασιμότητα και την ταχύτητα ανάπτυξης. Και αυτό διότι, ο διαχωρισμός των επιπέδων, επιτρέπει την εκτέλεση εργασιών όπως, η τροποποίηση ή η αναβάθμιση των συστατικών τους στοιχείων, ανεξάρτητα ή και παράλληλα με τα υπόλοιπα δομικά στοιχεία του συστήματος και επομένως, χωρίς να επηρεάζεται η συνολική του λειτουργία. Επιπλέον, ο αποκλεισμός του χρήστη από την απευθείας πρόσβαση στη βάση δεδομένων του συστήματος, παρέχει, εξ' ορισμού, ένα επιπλέον επίπεδο ασφάλειας, ενώ η χρήση του ενδιάμεσου επιπέδου για την επικοινωνία πελάτη-εξυπηρετητή, προσφέρει ανεξαρτησία, τόσο από την συσκευή (desktop, laptop, tablet κλπ), όσο και από το λογισμικό που αυτή χρησιμοποιεί (Android, IOS, Windows, Linux κλπ).

Server-side

Database

Όπως έχει ήδη αναφερθεί, για τις ανάγκες αποθήκευσης των δεδομένων που συλλέγονται και επεξεργάζονται από το σύστημά μας, δημιουργήθηκε μία NoSQL βάση δεδομένων σε MongoDB. Η ευελιξία και η επεκτασιμότητα που προσφέρει, είναι ιδανικά χαρακτηριστικά για την ανάπτυξη βάσεων που αφορούν σε εφαρμογές για κινητές συσκευές. Επιπρόσθετα, σημαντικό πλεονέκτημα της MongoDB, ειδικότερα, αποτελεί και η ευκολία που προσφέρει στον χειρισμό των γεωχωρικών δεδομένων, τα οποία, στην περίπτωσή μας, αποτελούν παράμετρο καθοριστικής σημασίας. Η MongoDB, έχει εξ' αρχής ενσωματωμένη τη δυνατότητα χειρισμού γεωχωρικών δεδομένων[45] και επομένως, δεν απαιτείται η εγκατάσταση οποιουδήποτε plug-in ή άλλης επέκτασής της. Επιπλέον, η

[45] https://www.mongodb.com/docs/manual/geospatial-queries/

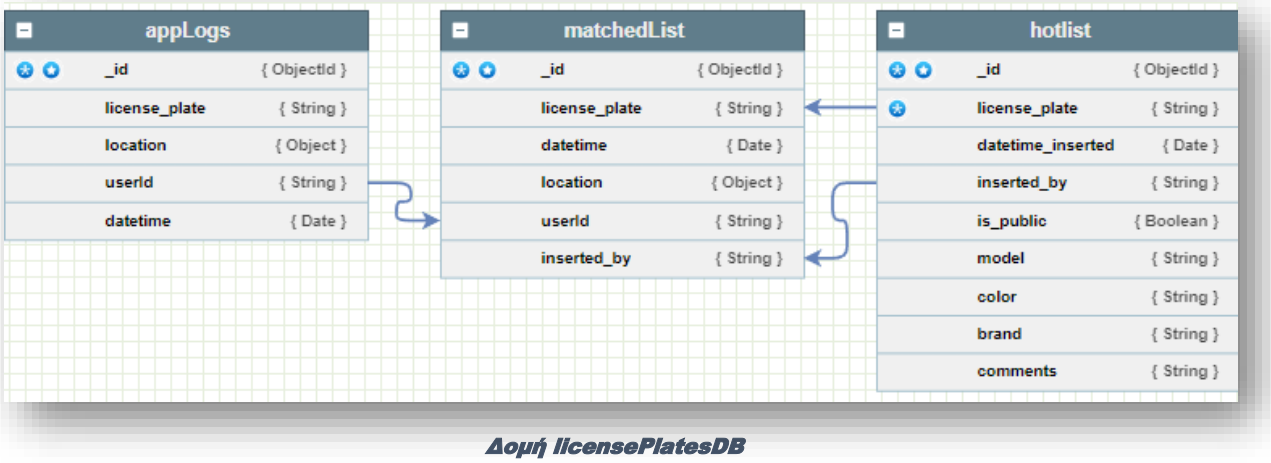
αναζήτηση και η επεξεργασία τους, είναι από κατασκευής βελτιστοποιημένη για την αποδοτικότερη απόκριση του συστήματος.

Η βάση μας, η οποία φέρει την ονομασία «**licensePlatesDB**», φιλοξενεί τις ακόλουθες τρείς συλλογές (**collections**):

- **appLogs** – Αφορά στις πληροφορίες που συλλέγονται από τον εκάστοτε χρήστη κατά τη λειτουργία της σάρωσης σε πραγματικό χρόνο (Go Live!).
- **hotlist** – Αφορά στις πινακίδες κυκλοφορίας για τις οποίες εκδηλώνουν ενδιαφέρον οι χρήστες (My HotList).
- **matchedList** – Αφορά στις πινακίδες κυκλοφορίας που εντοπίστηκαν από τους χρήστες και εμπεριέχονται στη λίστα ενδιαφέροντος ενός ή περισσοτέρων χρηστών.

Για καθεμία από αυτές, έχουν δημιουργηθεί κατάλληλα indexes που επιταχύνουν την αναζήτηση και την επεξεργασία των εγγραφών (**documents**) που περιέχουν.

Παρακάτω, παρατίθεται σχηματικά η δομή της βάσης, καθώς και οι πληροφορίες που εμπεριέχονται σε αυτή.



Δομή licensePlatesDB

Ο administrator του συστήματος, μπορεί να χειρίζεται οποιαδήποτε τεχνική διεργασία απαιτείται, είτε μέσω του Compass, δηλαδή το γραφικό περιβάλλον (Graphical User Interface - GUI) της MongoDB, είτε μέσω της γραμμής εντολών (Command Line Interface - CLI) που προσφέρεται με την εγκατάσταση του MongoDB server.

Spring Boot Configurations

Το back-end του συστήματος, αναπτύχθηκε σε περιβάλλον Spring Boot, με τη βοήθεια του προγράμματος ανάπτυξης λογισμικού IntelliJ της JetBrains.

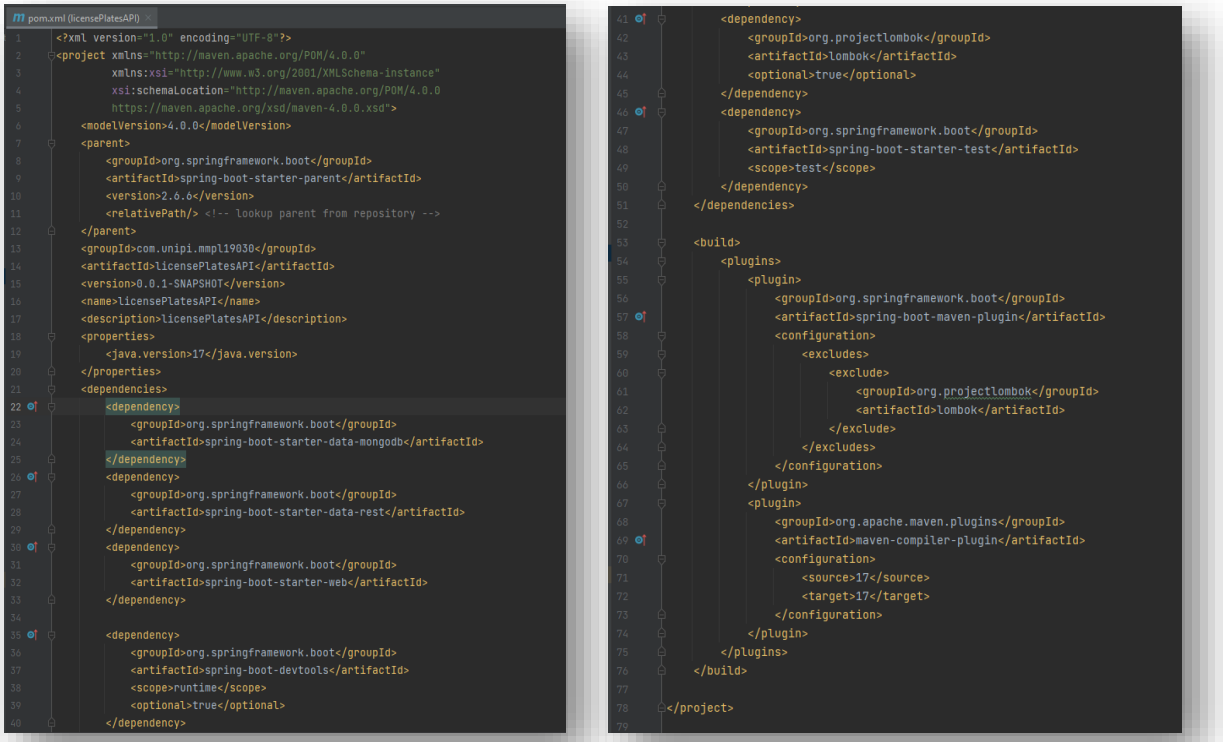
Ξεκινώντας λοιπόν, με την καθοδήγηση του Spring Initializr, προχωρήσαμε στην κατασκευή του project **LicensePlatesAPI**. Για την ανάπτυξή του, χρησιμοποιήθηκε το Maven και ως γλώσσα προγραμματισμού ορίστηκε η Java (JDK 17). Στο project προστέθηκαν τα ακόλουθα dependencies:

- Spring Web – Ενσωματώνει τον Tomcat server, υπηρεσίες REST και το Spring MVC
- Rest Repositories – Διευκολύνει τη διαδικασία δημιουργίας REST μέσω του Spring Data Rest.
- Spring Data MongoDB – Παρέχει τη δυνατότητα επικοινωνίας και πρόσβασης στα δεδομένα των βάσεων της MongoDB
- Lombok – Είναι βιβλιοθήκη για τα annotations της Java και εξυπηρετεί στη μείωση boilerplate κώδικα

- Spring Boot DevTools – Παρέχει εργαλεία και ρυθμίσεις για την μείωση του απαιτούμενου χρόνου ανάπτυξης εφαρμογών σε Spring Boot. Παρέχει χαρακτηριστικά όπως, Automatic Restart, LiveReload, Remote Update and Restartκαι άλλα.

Η ενσωμάτωσή τους, εγγυάται την ομαλή διαλειτουργικότητα της βάσης δεδομένων που θα χρησιμοποιήσουμε (licensePlateDB) με τον server του συστήματος (Tomcat), καθώς και συνολικά, την απροβλημάτιστη επικοινωνία μεταξύ, front-end και back-end.

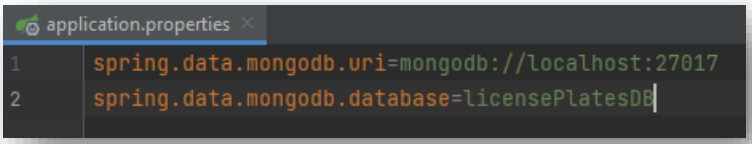
Με την ολοκλήρωση των επιλογών, το initializr φροντίζει, αυτόματα, για τη δημιουργία του αρχείου pom.xml, ενώ το Maven, με τη σειρά του, εγκαθιστά τα απαραίτητα dependencies.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.6</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.unipi.mmpl19030</groupId>
    <artifactId>licensePlatesAPI</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>licensePlatesAPI</name>
    <description>licensePlatesAPI</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-rest</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>17</source>
                    <target>17</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

Τελική μορφή αρχείου pom.xml

Στη συνέχεια, χρειάζεται να προβούμε σε ορισμένες επιπρόσθετες ρυθμίσεις, μικρής κλίμακας, που αφορούν στην ολοκλήρωση της επιτυχούς σύνδεσης του project με τη βάση δεδομένων που δημιουργήσαμε και περιγράψαμε παραπάνω[46]. Στο αρχείο application.properties του project, προσθέτουμε τη διεύθυνση του server της βάσης μας, καθώς και την ονομασία της.



application.properties

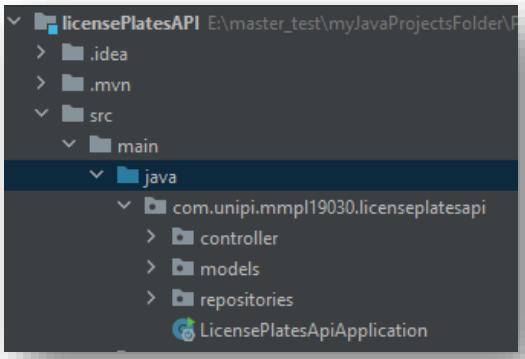
```
spring.data.mongodb.uri=mongodb://localhost:27017
spring.data.mongodb.database=licensePlatesDB
```

Επιπρόσθετες ρυθμίσεις του project

Ακολουθώντας τη λογική της αρχιτεκτονικής του Spring Boot[47], στο project δημιουργήθηκαν τρία διαφορετικά packages, με ονομασίες **models**, **repositories** και **controller**. Καθένα από αυτά, αντιπροσωπεύει και μία διαστρωμάτωση. Στο σημείο αυτό, να σημειώσουμε ότι, για την παρουσίαση

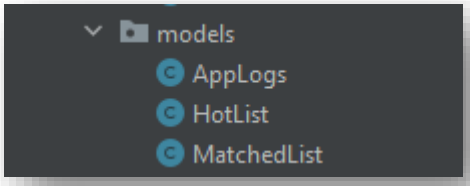
[46] Βλ. σχετικά ενότητα «Database» παρούσας διατριβής.
[47] Βλ. σχετικά ενότητα «Αρχιτεκτονική Spring Boot» παρούσας διατριβής.

των δεδομένων στον τελικό χρήστη (presentation layer), έχουμε κατασκευάσει αποκλειστική εφαρμογή για κινητές συσκευές (MobileLPR) και ως εκ τούτου, δεν θα ασχοληθούμε με τη δημιουργία Views για την παρουσίασή τους απευθείας μέσω του Spring Boot.



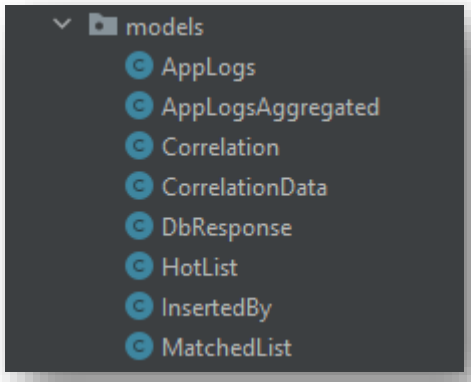
Περιεχόμενα packages

Εντός του models package, τοποθετήθηκαν οι κλάσεις που θα αναπαριστούν τις οντότητες της βάσης μας. Οι κλάσεις **AppLogs**, **HotList** και **MatchedList**, λοιπόν, θα χρησιμοποιούνται για να αντιστοιχίζουμε τα documents των collections **appLogs**, **hotlist** και **matchedList**, όπως αυτά ορίστηκαν παραπάνω[48]. Οι κλάσεις αυτές, αποκαλούνται POJO classes (Plain Old Java Object classes).



Στιγμιότυπο του models package

Κατά τη διάρκεια της ανάπτυξης του συστήματος, προκειμένου να ανταποκρίνεται στις επιμέρους λειτουργίες που καλείται να εκτελέσει, απαιτήθηκε η παραγωγή επιπρόσθετων κλάσεων. Τελικά, το models package, κατέληξε να περιέχει την λίστα που φαίνεται στην εικόνα που ακολουθεί:

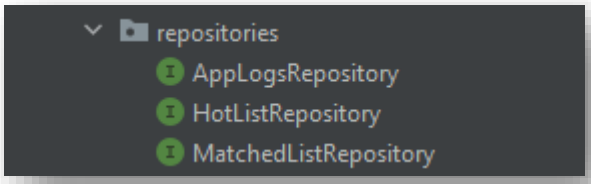


Τελική μορφή περιεχομένων του models package

Στο package με την ονομασία repositories, συμπεριλήφθηκαν όλα τα interfaces που χρησιμοποιούνται για την εκτέλεση των βασικών λειτουργιών (CRUD operations) επί των δεδομένων

[48] Βλ. σχετικά ενότητα «Database» παρούσας διατριβής.

της βάσης, καθώς και για την εκτέλεση ορισμένων εξατομικευμένων ερωτημάτων προς τη βάση (custom queries).



Στιγμιότυπο του repositories package

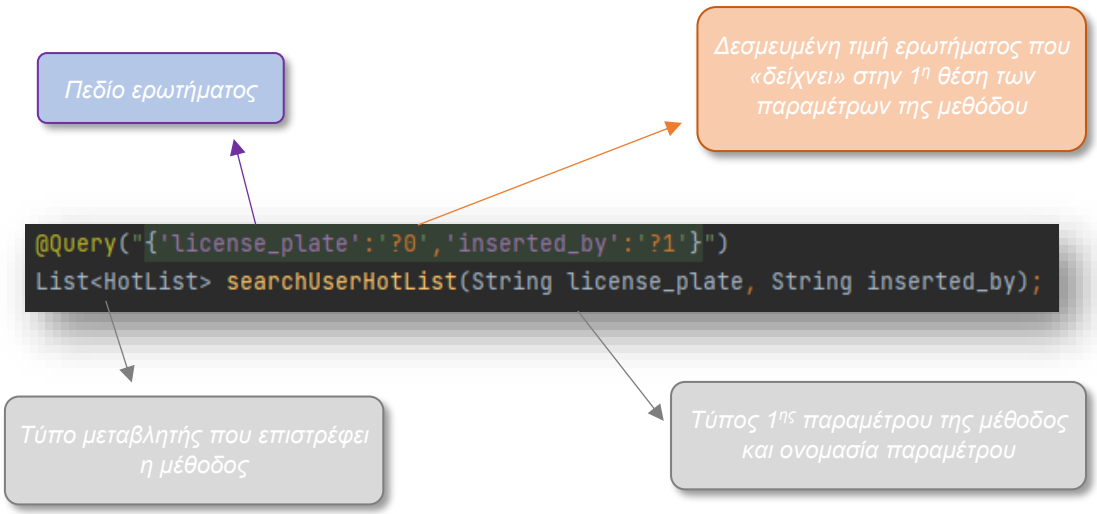
Σε καθεμία από τις βασικές μας κλάσεις (AppLogs, HotList, MatchedList), αντιστοιχίστηκε και ένα interface. Η διαδικασία ενσωμάτωσης των βασικών λειτουργιών CRUD σε καθεμιά τους, είναι ιδιαίτερα απλουστευμένη στο Spring Boot. Το μόνο που χρειάζεται είναι, τα public interfaces που δημιουργήσαμε, να κάνουν extend το MongoRepository. Η δομή τους, θα είναι ανάλογη με αυτή που παρατίθεται στο παράδειγμα που ακολουθεί:

public interface *interface_name* extends MongoRepository <*Model*, *Id_Type*> { … }

όπου, **Model** είναι η κλάση στην οποία θα επενεργούν οι λειτουργίες και **Id_Type**, ο τύπος του πεδίου που αναπαριστά το πρωτεύον κλειδί της βάσης μας[49]. Εντός των ορίων του interface, μπορούν να παρασταθούν και επιπρόσθετες, εξατομικευμένες λειτουργίες, όπως ερωτήματα (queries) προς τη βάση. Για τη δημιουργία τέτοιων ερωτημάτων, απαιτείται η χρήση του annotation **@Query**. Στην εικόνα που ακολουθεί, αποτυπώνεται εξατομικευμένο ερώτημα που κατασκευάστηκε για την αναζήτηση στο collection «hotlist» της βάσης μας:

```
public interface HotListRepository extends MongoRepository<HotList,String> {
```

Το interface HotListRepository που δημιουργήσαμε για το project

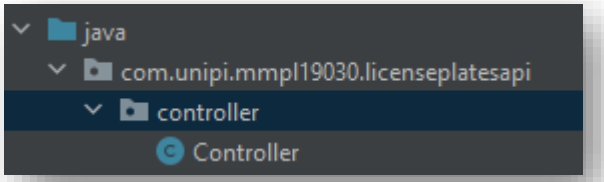


Προσαρμοσμένο ερώτημα (Custom query)

[49] Στην περίπτωση της MongoDB το «_id», το οποίο είναι τύπου String.

Αξίζει να σημειωθεί ότι, η χρήση του annotation @Query και κατ' επέκταση, η Spring Boot, υποστηρίζει την κατασκευή παραμετροποιημένων ερωτημάτων, για την αποφυγή τρωτοτήτων ασφαλείας που αφορούν στις βάσεις (SQL injections).

Τέλος, στο package controller, τοποθετήθηκε η κλάση που διαδραματίζει τον ρόλο του controller, στα διάφορα αιτήματα του client και τις αποκρίσεις του συστήματος σε αυτά. Όπως έχει ήδη συζητηθεί, το σύστημά μας, σχεδιάστηκε για να παρέχει RESTful web services. Επομένως, τα δεδομένα μας, οι αποκρίσεις δηλαδή του συστήματος, παρέχονται υπό τη μορφή πόρων (resources). Για τον λόγο αυτό, ο controller, θα πρέπει να είναι τύπου RestController. Για τον προσδιορισμό μιας κλάσης ως RestController, απαιτείται η χρήση του annotation **@RestController** πριν το σώμα της κλάσης.



Στιγμιότυπο του Controller package

```
1   package com.unipi.mmpl19030.licenseplatesapi;
2
3   import ...
29
30  @RestController
31
32  public class AppLogsController {
33
```

Απόσπασμα της κλάσης Controller όπου εμφαίνεται η χρήση του annotation @ReatController

Αφού, πλέον, ορίσαμε τον controller μας ως RestController, στη συνέχεια, χρειάζεται να οριστούν και οι διευθύνσεις των πόρων που θα οριστούν στο σύστημά μας. Χρειάζεται, δηλαδή, να οριστούν τα REST APIs με τα endpoints τους. Ομοίως, και σε αυτή την περίπτωση, τα annotations προσφέρονται προς διευκόλυνση. Αναλαμβάνουν να διεκπεραιώσουν, με τον βέλτιστο τρόπο, όλες τις απαραίτητες ρυθμίσεις, αυτοματοποιημένα, ενώ ο προγραμματιστή, καλείται να παράσχει μονάχα τις πληροφορίες που αφορούν στις μεθόδους κλήσης και τα path των endpoints που θα χρησιμοποιηθούν.

Έτσι λοιπόν, ανάλογα με τον τύπο κλήσης που επιθυμούμε να πραγματοποιήσουμε, θα πρέπει να συμπεριλάβουμε ένα από τα ακόλουθα annotation:

- **@GetMapping** – Αποτελεί συντόμευση που προσδιορίζει ότι η μέθοδος της κλήσης θα είναι GET (RequestMethod.GET).
- **@PostMapping** - Αποτελεί συντόμευση που προσδιορίζει ότι η μέθοδος της κλήσης θα είναι POST (RequestMethod.POST).
- **@DeleteMapping** - Αποτελεί συντόμευση που προσδιορίζει ότι η μέθοδος της κλήσης θα είναι DELETE (RequestMethod.DELETE).
- **@PutMapping** - Αποτελεί συντόμευση που προσδιορίζει ότι η μέθοδος της κλήσης θα είναι PUT (RequestMethod.PUT).

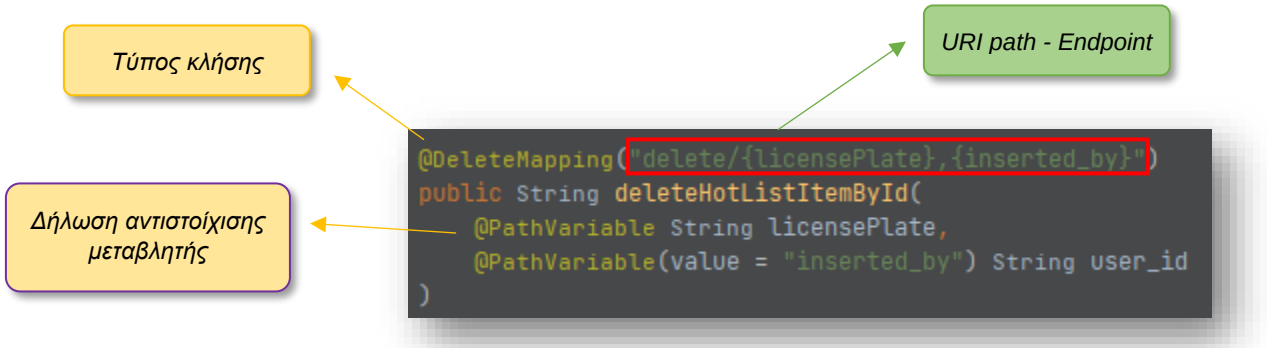
Στην απλούστερη μορφή του, ένα REST API στο Spring Boot, θα έχει την εξής δομή:

@GetMapping("path/endpoint")

Στην περίπτωση όπου απαιτείται να συμπεριληφθεί κάποια μεταβλητή, ή ακόμη και δεδομένα, κατά την πραγματοποίηση της κλήσης (POST, PUT, DELETE), η προηγούμενη σύνταξη, τροποποιείται ως εξής:

@DeleteMapping(“path/{variable_1}{variable_2}”)

όπου, εντός των συμβόλων **{ }**, περιέχονται οι μεταβλητές που θα χρησιμοποιηθούν. Ωστόσο, σε αυτή την περίπτωση, απαραίτητη προϋπόθεση, είναι η χρήση και του annotation **@PathVariable**[50] κατά τη δήλωση της μεθόδου στην οποία αναφέρεται το API.



Παράδειγμα χρήσης της κλήσης DELETE

Εντελώς ανάλογα, για την κλήση POST, όπου θα συμπεριλάβουμε δεδομένα, όπως ενδεχομένως, στην περίπτωση χρήσης κάποιου model object, η δομή του API θα έχει την ακόλουθη μορφή:

@PostMapping(“path/endpoint”)

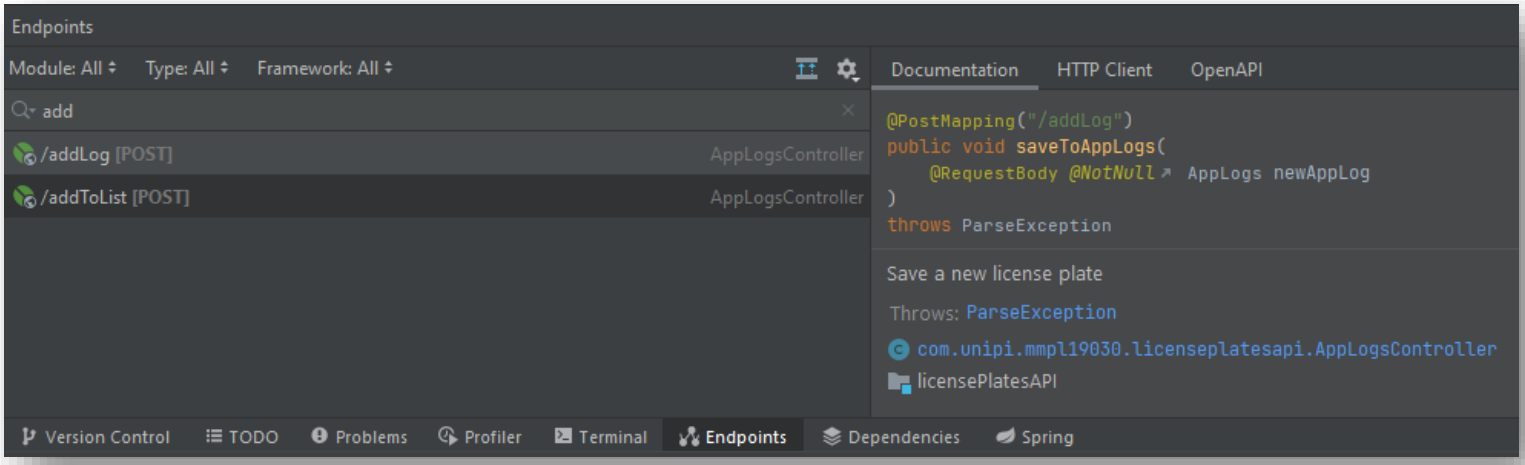
ενώ στη μέθοδο που το ακολουθεί, θα πρέπει να συμπεριληφθεί το annotation **@RequestBody**[51]

Παράδειγμα χρήσης της κλήσης POST

Το IntelliJ μας δίνει τη δυνατότητα επισκόπησης των endpoints που έχουμε κατασκευάσει, παρέχοντάς μας, αυτοματοποιημένα, και ένα είδος documentation που αντιστοιχεί σε καθένα από αυτά.

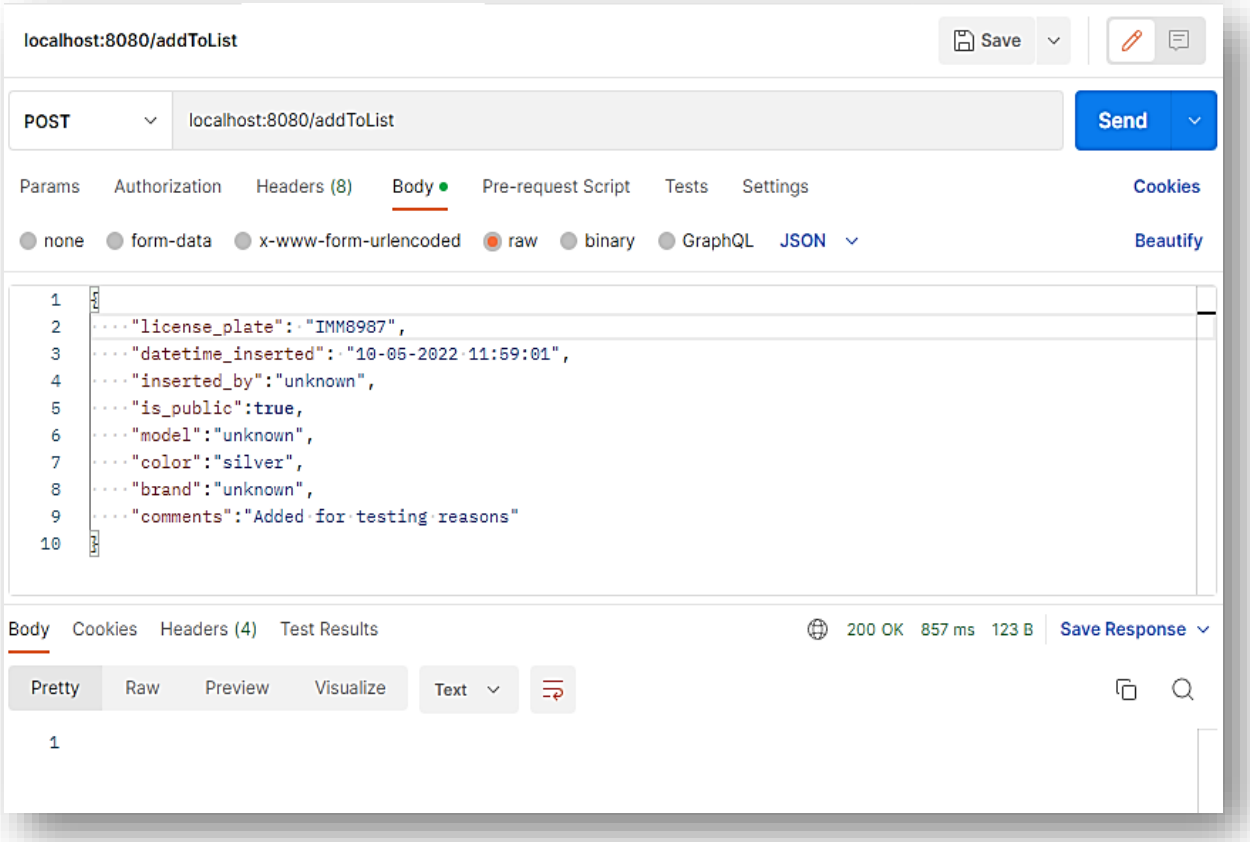
[50] «Δένει» μία παράμετρο της μεθόδου, με την τιμή μιας μεταβλητής από το σώμα του URI (endpoint).

[51] Δηλώνει ότι η παράμετρος που ακολουθεί, είναι το σώμα που αναμένεται να αποσταλεί με το HTTP request.

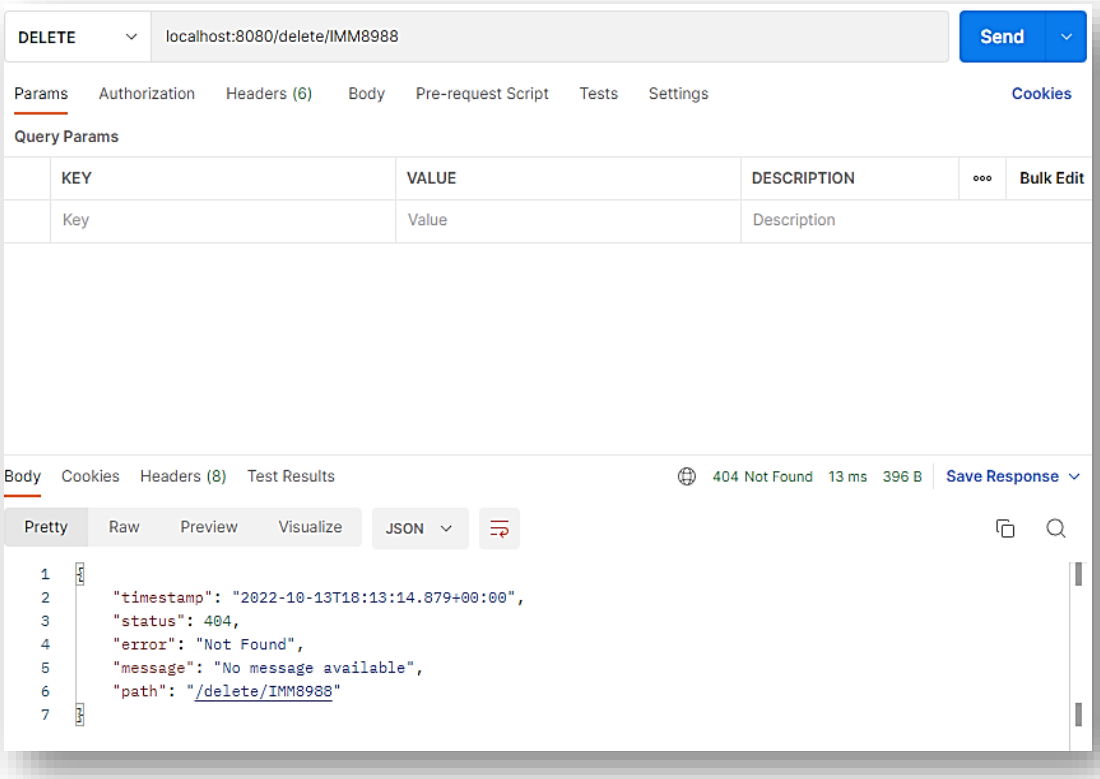


Απόσπασμα endpoint και documentation του project

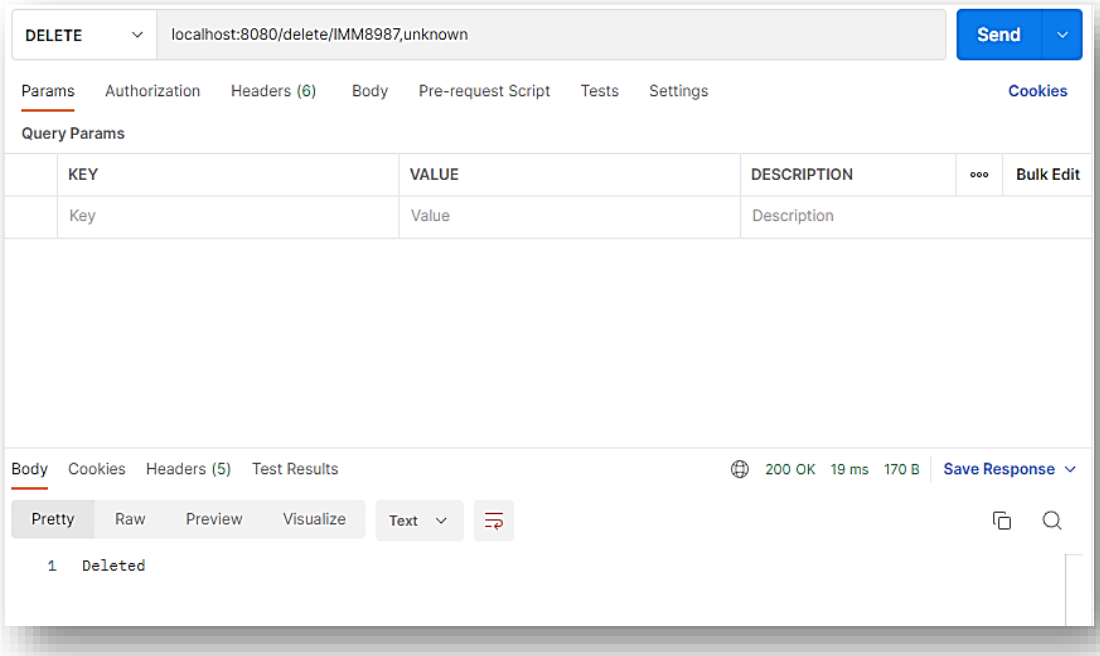
Η business logic του project μας, περιέχεται στο σώμα των μεθόδων του controller μας. Με τη βοήθεια του Postman, έχουμε τη δυνατότητα να ελέγξουμε άμεσα τη λειτουργικότητα των REST APIs που κατασκευάσαμε, καθώς και της λογικής που τα ακολουθεί.



Παράδειγμα κλήσης POST με τη βοήθεια του PostMan



Παράδειγμα κλήσης DELETE με τη βοήθεια του PostMan – Χρήση λανθασμένου endpoint



Παράδειγμα κλήσης DELETE με τη βοήθεια του PostMan – Χρήση ορθού endpoint

Στις εικόνες που ακολουθούν, παρατίθεται, ενδεικτικά, η κλάση «HotList», το interface «HotListRepository» που αντιστοιχεί σε αυτή, καθώς και απόσπασμα του controller με λειτουργίες της.

HotList.java

```java
package com.unipi.mmpl19030.licenseplatesapi;
import ...
23 usages
@Document("hotList") /** This annotation points my mongoDB collection*/
public class HotList {
    2 usages
    @Id /** This annotation points my mongoDB _id key, and it's autoincrement*/
    private String id;
    2 usages
    private String license_plate;
    2 usages
    private String datetime_inserted;
    2 usages
    private String inserted_by;
    2 usages
    private Boolean is_public;
    2 usages
    private String model;
    2 usages
    private String color;
    2 usages
    private String brand;
    2 usages
    private String comments;
    3 usages
    public String getId() { return id; }

    1 usage
    public void setId(String id) { this.id = id; }

    3 usages
    public String getLicense_plate() { return license_plate; }

    1 usage
    public void setLicense_plate(String license_plate) { this.license_plate = license_plate; }

    1 usage
    public String getDatetime_inserted() { return datetime_inserted; }

    public void setDatetime_inserted(String datetime_inserted) { this.datetime_inserted = datetime_inserted; }
    5 usages
    public String getInserted_by() { return inserted_by; }

    public void setInserted_by(String inserted_by) { this.inserted_by = inserted_by; }

    1 usage
    public Boolean getIs_public() { return is_public; }

    public void setIs_public(Boolean is_public) { this.is_public = is_public; }

    public String getModel() { return model; }

    public void setModel(String model) { this.model = model; }

    public String getColor() { return color; }

    public void setColor(String color) { this.color = color; }

    public String getBrand() { return brand; }

    public void setBrand(String brand) { this.brand = brand; }

    1 usage
    public String getComments() { return comments; }
    1 usage
    public void setComments(String comments) { this.comments = comments; }
}
```

Η κλάση HotList του project

HotListRepository.java

```java
package com.unipi.mmpl19030.licenseplatesapi;

import ...

1 usage
public interface HotListRepository extends MongoRepository<HotList,String> {
    1 usage
    @Query(value = "{$and:[{'license_plate':'?0'},{$or:[{'inserted_by':'?1'},{'is_public':true}]}]}",count = true)
    Integer searchHotListByLicensePlateCount(String license_plate, String inserted_by);

    2 usages
    @Query("{license_plate:'?0'}")
    List<HotList> searchHotListByLicensePlate(String license_plate);

    4 usages
    @Query("{'license_plate':'?0','inserted_by':'?1'}")
    List<HotList> searchUserHotList(String license_plate, String inserted_by);

    1 usage
    @Query(value  = "{$and:[{'license_plate':'?0'},{'inserted_by':'?1'}]}", count = true)
    Integer findUserHotList(String license_plate, String inserted_by);

    1 usage
    @Query("{'license_plate':'?0','is_public':true}")
    List<HotList> findPublicHotList(String license_plate);

    @Query("{'datetime_inserted': { $gte: '?0', $lte: '?1' }}")
    List<HotList> searchAppLogsByDatetime(String startDatetime, String stopDatetime);

    1 usage
    @Query("{'inserted_by':'?0'}")
    List<HotList> getAllByUserID(String inserted_by);

}
```

Το interface HotListRepository του project

```java
/**
 * Delete a record
 */

@DeleteMapping("delete/{licensePlate},{inserted_by}")
public String deleteHotListItemById(@PathVariable String licensePlate,@PathVariable(value = "inserted_by") String user_id) {
    List<HotList> item = hotListRepo.searchUserHotList(licensePlate, user_id);
    if (!item.isEmpty()) {
        hotListRepo.deleteById(item.get(0).getId());
        return "Deleted";
    }
    return null;
}

/**
 * Update a record
 */

@PutMapping("update/{licensePlate}")
public Map<String, String> updateById(@RequestBody HotList itemToUpdate, @PathVariable String licensePlate) {
    Map<String, String> response = new HashMap<>();
    List<HotList> item = hotListRepo.searchUserHotList(licensePlate,itemToUpdate.getInserted_by());
    try {
        itemToUpdate.setId(item.get(0).getId());
        hotListRepo.save(itemToUpdate);
        response.put("response:", "Record " + item.get(0).getId()+" updated successfully");
    } catch (Error e){
        response.put("response:",e.getLocalizedMessage());
    }
    return response;
}
```

Απόσπασμα της κλάσης Controller με τις μεθόδους και τα endpoints για την διαγραφή και την τροποποίηση μια εγγραφής της βάσης

Client-side

Το client-side τμήμα του συστήματος, δηλαδή, η εφαρμογή MobileLPR, αναπτύχθηκε στο περιβάλλον του Android Studio, χρησιμοποιώντας τις γλώσσες προγραμματισμού Java και Kotlin. Η εφαρμογή που δημιουργήθηκε, ενσωματώνει το μεγαλύτερο μέρος της φιλοσοφίας στην οποία εδράζεται η ανάπτυξη του συστήματος (business logic). Προσανατολισμένη στην εκμετάλλευση των ενσωματωμένων χαρακτηριστικών του hardware των κινητών συσκευών, όπως η κάμερα και το GPS, σχεδιάστηκε μια εφαρμογή η οποία, θα λαμβάνει ως είσοδο μια ροή εικόνων, θα εντοπίζει σε αυτές αν υπάρχει κάποια πινακίδα κυκλοφορίας και σε θετική περίπτωση, θα αναγνωρίζει το εντυπωμένο κείμενο. Στη συνέχεια, το κείμενο αυτό, θα εμπλουτίζεται με χωρικά δεδομένα και θα αποστέλλεται για περαιτέρω επεξεργασία στο back-end του συστήματος. Η απόκριση του συστήματος, θα προβάλλεται, τελικά, από την εφαρμογή στη συσκευή του χρήστη.

Με γνώμονα τα παραπάνω, και θέτοντας ως βασική προϋπόθεση την πρόκριση μια λύσης που θα επιτρέπει την ενσωμάτωσή της σε κινητές συσκευές, ο σχεδιασμός και η ανάπτυξη της εφαρμογής μας, βασίστηκε στην επιτυχής ολοκλήρωση των επιμέρους σταδίων:

1. Σχεδιασμός λύσης για την αναγνώριση των πινακίδων κυκλοφορίας σε σταθερή εικόνα και προσδιορισμός των ορίων τους.
2. Σχεδιασμός λύσης για την αναγνώριση του εντυπωμένου κειμένου των πινακίδων κυκλοφορίας.
3. Ενοποίηση λύσεων που προέκυψαν
4. Επέκταση ενοποιημένης λύσης σε ροή εικόνων σε πραγματικό χρόνο (live streaming)
5. Προετοιμασία των δεδομένων για την αποστολή τους στο back-end (προεπεξεργασία δεδομένων).
6. Χειρισμός απόκρισης συστήματος και προβολή των δεδομένων στην εφαρμογή.

Η αναγνώριση, ο εντοπισμός και ο προσδιορισμός της σχετικής θέσης μιας πινακίδας κυκλοφορίας μέσα σε μια εικόνα, επιτεύχθηκε, τελικά, με τη δημιουργία ενός προσαρμοσμένου (custom) μοντέλου μηχανικής μάθησης. Το μοντέλο «**licenseplates**», όπως ονομάστηκε, εκπαιδεύτηκε χρησιμοποιώντας τη βιβλιοθήκη **TensorFlow Lite Model Maker**, στην online πλατφόρμα Colab της Google.

Η εν λόγω βιβλιοθήκη, αναπτύσσει μοντέλα μηχανικής μάθησης με τη μέθοδο του «transfer learning»[52]. Έτσι, χρησιμοποιώντας κατάλληλα προσαρμοσμένα σύνολα δεδομένων (custom datasets), η διαδικασία της εκπαίδευσης ενός νέου μοντέλου μηχανικής μάθησης, απλοποιείται, ενώ παράλληλα, μειώνεται ο απαιτούμενος όγκος δεδομένων.

Η προετοιμασία του dataset, πραγματοποιήθηκε μέσω της πλατφόρμας Vision του Google Cloud. Τα δεδομένα που συγκεντρώθηκαν (εικόνες), ταξινομήθηκαν σε training, validation και testing sets, ώστε στη συνέχεια να τροφοδοτηθούν για κατανάλωση από το υπό εκπαίδευση μοντέλο μας.

Η διαδικασία της εκπαίδευσης ολοκληρώνεται σε έξι διαδοχικές φάσεις:

1. Επιλογή της αρχιτεκτονικής εκπαίδευσης – Υπάρχει μεγάλη ποικιλία μοντέλων από τα οποία μπορεί κάποιος να επιλέξει να χρησιμοποιήσει, ανάλογα με τις ανάγκες τις εργασίας που επιδιώκει να ολοκληρώσει[53]. Στην περίπτωσή μας, το μοντέλο που χρησιμοποιήθηκε ανήκει στην οικογένεια των mobile και IoT friendly μοντέλων, που προορίζονται για τον εντοπισμό αντικειμένων σε εικόνες και πρόκειται για το EfficientDet-Lite2[54]. Ανάλογα με την έκδοση του μοντέλου, υπάρχουν διαφοροποιήσεις στο μέγεθος του παραγόμενου αρχείου (μοντέλο), τη καθυστέρηση απόκρισής του και το ποσοστό ακρίβειας στις προβλέψεις[55].
2. Τροφοδοσία των training και validation datasets – Στο στάδιο αυτό, τροφοδοτούμε τις εικόνες που προορίζονται για την εκπαίδευση και την επικύρωση της εκπαίδευσης του μοντέλου. Οι

[52] Βλ. αναλυτικότερα ενότητα «Transfer learning» παρούσας διατριβής.

[53] https://tfhub.dev/s?module-type=image-object-detection

[54] Το μοντέλο διατίθεται σε πέντε εκδόσεις EfficientDet-Lite[0-4].

[55] https://github.com/google/automl/blob/master/efficientdet/README.md

εικόνες που προορίζονται για επικύρωση (validation), θα μας βοηθήσουν να αποφύγουμε την υπερ-μοντελοποίηση[56] (overfitting) του μοντέλου μας.

3. Εκπαίδευση – Εδώ πραγματοποιείται η κύρια διαδικασία της εκπαίδευσης.
4. Αξιολόγηση του μοντέλου – Στο στάδιο της αξιολόγησης, χρησιμοποιούνται οι υπολειπόμενες εικόνες, δηλαδή το testing dataset, ώστε να ποσοτικοποιήσουμε με άγνωστα, για το μοντέλο μας, δεδομένα, την ακρίβεια των αποτελεσμάτων.
5. Βελτιστοποίησή του για κινητές συσκευές - Μέσω της τεχνικής «quantization»[57] ο όγκος του μοντέλου μειώνεται δραστικά, με την ελάχιστη δυνατή επίπτωση στην ακρίβεια των προβλέψεών του. Η διαδικασία είναι απαραίτητη ώστε να μπορέσει το παραγόμενο μοντέλο να χρησιμοποιηθεί αποτελεσματικά από συσκευές όπως τα κινητά τηλέφωνα.
6. Εξαγωγή του μοντέλου, έτοιμου, πλέον, για χρήση από κινητές συσκευές.

Προκειμένου να μπορέσουμε να χρησιμοποιήσουμε το μοντέλο που παράχθηκε από την προηγούμενη διαδικασία, χρειάζεται να πραγματοποιηθούν μια σειρά από ενέργειες. Αρχικά, απαιτείται να «φορτώσουμε» το μοντέλο μας στο project. Αυτό επιτυγχάνεται με τη δημιουργία ενός φακέλου τύπου **assets**[58], στον οποίο τοποθετούμε ένα αντίγραφο του μοντέλου μας. Στη συνέχεια, στο αρχείο **Gradle**[59] προσθέσουμε το πιο πρόσφατο dependency της βιβλιοθήκης TensorFlow[60]. Ρυθμίσεις που αφορούν στο μέγιστο πλήθος των πινακίδων που επιτρέπεται να εντοπιστούν σε κάθε εικόνα, ο καθορισμός του συγκεκριμένου μοντέλου μηχανικής μάθησης που θα χρησιμοποιηθεί, καθώς και το threshold των αντικειμένων που εντοπίζονται, επιτρέπεται στον προγραμματιστή μέσω του κώδικα[61] που ενσωματώνεται με την παραπάνω βιβλιοθήκη.

```
val options = ObjectDetector.ObjectDetectorOptions.builder()
    .setMaxResults(1)
    .setScoreThreshold(0.3f)
    .build()
val detector = ObjectDetector.createFromFileAndOptions(
    ctx1,
    modelPath: "licenseplates.tflite",
    options
)
```

Απόσπασμα ρυθμίσεων αντικειμένου (object) του μοντέλου μηχανικής μάθησης για τον εντοπισμό των πινακίδων κυκλοφορίας

Ανάλογη ήταν και η προσέγγισή μας στην περίπτωση της αναγνώρισης του εντυπωμένου κειμένου των πινακίδων κυκλοφορίας. Η διαφορά έγκειται στη βιβλιοθήκη που χρησιμοποιήθηκε, για την ανάπτυξη και την ενσωμάτωση του μοντέλου μηχανικής μάθησης που χρειαζόμαστε. Εδώ, έγινε χρήση του ML-Kit της Google, ενός ολοκληρωμένου πακέτου ανάπτυξης λογισμικού (Software Development Kit - SDK), το οποίο προσφέρει έτοιμες λύσεις, εκπαιδευμένων μοντέλων μηχανικής μάθησης. Ανάμεσα στα παρεχόμενα μοντέλα, είναι και το Text Recognition[62], το οποίο προσφέρει

---

[56] Σύμφωνα με τον επίσημο ορισμό, πρόκειται για την παραγωγή ενός μοντέλου ανάλυσης, το οποίο ανταποκρίνεται υπερβολικά κοντά ή ακριβώς σε ένα συγκεκριμένο σύνολο δεδομένων και ως εκ τούτου, μπορεί να αποτύχει να προσαρμοστεί σε επιπρόσθετα δεδομένα ή να χρησιμοποιηθεί σε μελλοντικές παρατηρήσεις με επιτυχία.

[57] https://www.tensorflow.org/lite/performance/model_optimization

[58] Στο κεντρικό μενού του Android Studio, εκτελούμε διαδοχικά τις επιλογές File -> New -> Folder -> Assets Folder.

[59] Build.gradle(:app).

[60] Το project μας χρησιμοποιεί το «org.tensorflow:tensorflow-lite-task-vision:0.4.2».

[61] https://www.tensorflow.org/lite/inference_with_metadata/task_library/object_detector

[62] Για την ανάπτυξη της εφαρμογής MobileLPR χρησιμοποιήσαμε το Text Recognition v2 - https://developers.google.com/ml-kit/vision/text-recognition/v2

μια σειρά από πολύτιμα χαρακτηριστικά, όπως, η αναγνώριση της δομής ενός κειμένου (γραμμή, παράγραφος, λέξη), καθώς και της γλώσσα που χρησιμοποιείται.

Για τη χρήση του, απαιτείται μονάχα η προσθήκη του κατάλληλου dependency[63] στο αρχείο Gradle του project. Οι δυνατότητές του, ενσωματώνονται στο αρχείο εγκατάστασης της παραγόμενης εφαρμογής (.apk - Android Package Kit) και επομένως, η χρήση του δεν απαιτεί την ύπαρξη σταθερής σύνδεσης στο διαδίκτυο (on-device). Το ML-Kit SDK, παρέχει βασικά παραδείγματα χρήσης[64] του κώδικα που αφορά στο Text Recognition. Με τον τρόπο αυτό, ο εκάστοτε προγραμματιστής, καθοδηγείται και εξοικειώνεται με τις βασικές ρυθμίσεις που απαιτούνται ώστε να είναι αποτελεσματικότερη η χρήση του μοντέλου.

```
private class YourImageAnalyzer : ImageAnalysis.Analyzer {

    override fun analyze(imageProxy: ImageProxy) {
        val mediaImage = imageProxy.image
        if (mediaImage != null) {
            val image = InputImage.fromMediaImage(mediaImage, imageProxy.imageInfo.rotationDegrees)
            // Pass image to an ML Kit Vision API
            // ...
        }
    }
}
```

```
val image = InputImage.fromBitmap(bitmap, 0)
```

```
val result = recognizer.process(image)
        .addOnSuccessListener { visionText ->
            // Task completed successfully
            // ...
        }
        .addOnFailureListener { e ->
            // Task failed with an exception
            // ...
        }
```

Παράδειγμα χρήσης μοντέλου Text Recognition του ML-Kit

Δυστυχώς, η ελληνική γλώσσα (αναγνώριση ελληνικών χαρακτήρων), δεν υποστηρίζεται, ακόμη από το συγκεκριμένο μοντέλο. Ωστόσο, δεδομένου του γεγονότος ότι οι χαρακτήρες που χρησιμοποιούνται στις ελληνικές πινακίδες κυκλοφορίας, περιορίζονται, εξ' υπαρχής, σε όσους βρίσκονται σε αντιστοιχία με λατινικούς, η χρήση του καθίσταται εφικτή. Βέβαια, αυτό συνεπάγεται και τον αντίστοιχο περιορισμό κατά την καταχώρηση των αναζητούμενων πινακίδων κυκλοφορίας από τον χρήστη[65].

Συνδυάζοντας τα δύο μοντέλα που περιγράψαμε παραπάνω, καταφέραμε να διαχειριστούμε, επιτυχώς, το ζήτημα του εντοπισμού και της αναγνώρισης των πινακίδων κυκλοφορίας σε μια σταθερή εικόνα. Η απόκριση[66] του πρώτου μοντέλου (licenseplates), τροφοδοτείται ως είσοδο στο δεύτερο (Text Recognition), για να λάβουμε, τελικά, το κείμενο της πινακίδας με αυτοματοποιημένο τρόπο.

---

[63] Χρησιμοποιήσαμε το dependency που αφορά στη λατινική γλώσσα «com.google.mlkit:text-recognition:16.0.0-beta6».

[64] https://developers.google.com/ml-kit/vision/text-recognition/v2/android

[65] Έχει περιοριστεί προγραμματιστικά η χρήση λατινικών, μόνο, κεφαλαίων χαρακτήρων, κατά την καταχώρηση των αναζητούμενων πινακίδων κυκλοφορίας από τον χρήστη.

[66] Ως απόκριση νοείται η περίπτωση όπου το μοντέλο μηχανικής μάθησης «licenseplates.tflite», εντόπισε κάποια πινακίδα κυκλοφορίας στην εικόνα που δέχθηκε ως είσοδο.

Το επόμενο σημαντικό συστατικό στοιχείο του project, εξυπηρετεί στην επέκταση της προηγούμενης λύσης και στο ζήτημα της ροής εικόνων σε πραγματικό χρόνο. Αναφερόμαστε στη βιβλιοθήκη CameraX της σουίτας Android Jetpack[67]. Με την CameraX, μας επιτρέπεται να χειριστούμε τα εν γένει χαρακτηριστικά της κάμερας, της εκάστοτε συσκευής, με ευκολία και συνέπεια, ανεξάρτητα από την έκδοση του λειτουργικού συστήματος Android[68].

Η χρήση της βιβλιοθήκης από το project, όπως και στις προηγούμενες περιπτώσεις, απαιτεί να προστεθούν στο αρχείο Gradle, τα κατάλληλα dependencies. Επιπλέον, επειδή η συγκεκριμένη βιβλιοθήκη χρησιμοποιεί και μεθόδους που αποτελούν μέρος της Java 8, είναι απαραίτητο να γίνουν και ορισμένες επιπρόσθετες ρυθμίσεις για τον compiler. Έτσι λοιπόν, στο αρχείο Gradle, στο τμήμα αμέσως μετά την αναφορά «buildTypes», προχωρούμε στις ακόλουθες δηλώσεις:

```
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}
```

Ρυθμίσεις αρχείου Gradle

Όσο για τα dependencies, αυτά περιλαμβάνουν την λίστα που εμφαίνεται στην εικόνα που ακολουθεί:

```
//dependencies required for CameraX
def camerax_version : String  = "1.2.0-beta02"
implementation "androidx.camera:camera-core:${camerax_version}"
implementation "androidx.camera:camera-camera2:${camerax_version}"
implementation "androidx.camera:camera-lifecycle:${camerax_version}"
implementation "androidx.camera:camera-video:${camerax_version}"
implementation "androidx.camera:camera-view:${camerax_version}"
implementation "androidx.camera:camera-extensions:${camerax_version}"
```

Στιγμιότυπο από τα Dependencies που προστέθηκαν στο αρχείο Gradle για την βιβλιοθήκη CameraX

Η βιβλιοθήκη συνοδεύεται, μεταξύ άλλων, και από μία λίστα σημαντικών χαρακτηριστικών, όπως:

- **Preview** - Λειτουργία επισκόπησης εικόνας στην οθόνη.
- **Image analysis** - Λειτουργία απρόσκοπτης πρόσβασης σε ένα buffer εικόνων[69].
- **Image capture** - Λειτουργία αποθήκευσης στιγμιότυπου.
- **Video capture** - Λειτουργία αποθήκευσης βίντεο και ήχου.

Από αυτά, το Image Analysis και το Preview, είναι τα χαρακτηριστικά που θα εξυπηρετήσουν την ανάπτυξη του project μας.

Θα χρησιμοποιήσουμε το Image Analysis για την διαδικασία παροχέτευσης συνεχόμενων εικόνων (frames), που λαμβάνονται σε πραγματικό χρόνο (real-time) από την εκάστοτε συσκευή, στο μοντέλο «licenseplates» και από εκεί στο «Text Recognition». Η ενσωμάτωση των απαιτούμενων λειτουργιών στον κώδικα, επιτυγχάνεται κάνοντας implement το interface ImageAnalysis.Analyzer σε μια κλάση.

```
private class CarPlateTextAnalyzer(ctx: Context, private val textFoundListener: (String) -> Unit) : ImageAnalysis.Analyzer { ... }
```

Στιγμιότυπο της κλάσης που δημιουργήσαμε για την ενσωμάτωση του χαρακτηριστικού Image Analysis

[67] Η σουίτα Jetpack είναι μια συλλογή βιβλιοθηκών, με σκοπό να βοηθήσουν τους προγραμματιστές να μειώσουν την επανάληψη κώδικα (boilerplate code) και να γράψουν κώδικα που λειτουργεί με συνέπεια σε όλες τις συσκευές και τις εκδόσεις Android - https://developer.android.com/jetpack

[68] Η βιβλιοθήκη υποστηρίζει εκδόσεις Android 5.0 (API level 21) και κάθε επόμενη, το οποίο αντιστοιχεί στο 98% των διαθέσιμων συσκευών Android παγκοσμίως (backward-compatibility).

[69] Ένα δεσμευμένο τμήμα μνήμης, που χρησιμοποιείται για τη διατήρηση των δεδομένων που υποβάλλονται σε επεξεργασία.

Έπειτα, κάνοντας override τη μέθοδο analyze, ορίζουμε τη λογική επεξεργασία που θα υποστεί κάθε τροφοδοτούμενο frame. Άρα, η μέθοδος analyze, θα εμπεριέχει τα νευρωνικά δίκτυα που κατασκευάσαμε νωρίτερα.

```
override fun analyze(imageProxy: ImageProxy) { ... }
```

Στιγμιότυπο μεθόδου analyze

Ωστόσο, προτού τροφοδοτήσουμε τη ροή εικόνων στα νευρωνικά μας δίκτυα, χρειάζεται να προβούμε σε στατικές ρυθμίσεις, που αφορούν στην επιθυμητή ανάλυση των εικόνων, καθώς και στον τρόπο διαχείρισης του buffer[70] από την βιβλιοθήκη.

```
ImageAnalysis imageAnalysis =
    new ImageAnalysis.Builder()
        // enable the following line if RGBA output is needed.
        //.setOutputImageFormat(ImageAnalysis.OUTPUT_IMAGE_FORMAT_RGBA_8888)
        .setTargetResolution(new Size(1280, 720))
        .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
        .build();

imageAnalysis.setAnalyzer(executor, new ImageAnalysis.Analyzer() {
    @Override
    public void analyze(@NonNull ImageProxy imageProxy) {
        int rotationDegrees = imageProxy.getImageInfo().getRotationDegrees();
            // insert your code here.
            ...
            // after done, release the ImageProxy object
            imageProxy.close();
        }
    });

cameraProvider.bindToLifecycle((LifecycleOwner) this, cameraSelector, imageAnalysis, preview);
```

Παράδειγμα ρυθμίσεων ImageAnalysis

Προκειμένου να προσφέρουμε στον χρήστη τη δυνατότητα να παρακολουθεί από την οθόνη της συσκευής του, άμεσα, όσα «βλέπει» και η κάμερά του, ενσωματώσαμε στο project και το χαρακτηριστικό Preview της ίδιας βιβλιοθήκης. Για τη χρήση του, απαιτείται να παρέχουμε ένα πλαίσιο στη διάταξη της οθόνης (layout), το οποίο, στη συνέχεια, θα συνδεθεί με τη ζωντανή ροή εικόνας από την κάμερα (live camera feed). Στις εικόνες που ακολουθούν, αναπαρίσταται ο κώδικας που χρησιμοποιήσαμε, τόσο για τη δημιουργία του πλαισίου, όσο και για τη σύνδεσή του με τη ροή εικόνας από την κάμερα.

```
<androidx.camera.view.PreviewView
    android:id="@+id/PreviewView"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:keepScreenOn="true">
</androidx.camera.view.PreviewView>
```

```
// Preview
val preview = Preview.Builder()
    .build()
    .also { it: Preview
        it.setSurfaceProvider(viewBinding.PreviewView.surfaceProvider)
    }
```

Απόσπασμα κώδικα που απαιτείται για την ενσωμάτωση του χαρακτηριστικού Preview

[70] https://developer.android.com/training/camerax/analyze

Με την ολοκλήρωση και της ενσωμάτωσης της δυνατότητας εντοπισμού και αναγνώρισης των πινακίδων κυκλοφορίας σε ροή εικόνων πραγματικού χρόνου, η εφαρμογή είναι σε θέση, πλέον, να εκτελέσει την βασική λειτουργία για την οποία προορίζεται. Η επόμενη φάση ανάπτυξης, αφορά στη δημιουργία ενός διαύλου επικοινωνίας με το back-end του συστήματος.

Η επικοινωνία της εφαρμογής με τον server του back-end στηρίχθηκε στη βιβλιοθήκη Volley[71]. Η Volley είναι μια από τις πιο διαδεδομένες βιβλιοθήκες για την διάδραση με τα REST Web APIs. Αναπτύχθηκε το 2013 από την Google, προκειμένου να καλύψει το κενό του λειτουργικού συστήματος Android για μια κλάση δικτύωσης, ικανή να λειτουργεί απρόσκοπτα, χωρίς, ωστόσο, να επηρεάζει αρνητικά την εμπειρία του χρήστη. Η Volley, διαχειρίζεται αποτελεσματικά τα αιτήματα διαδικτύου της εφαρμογής μας, μειώνοντας σημαντικά τον έκταση του κώδικα που απαιτείται για την πραγματοποίηση των διαφόρων κλήσεων και τον χειρισμό των αντίστοιχων απαντήσεων.

Απαρτίζεται από δύο βασικές κλάσεις:

- **Request Queue** – Αποτελεί το «μέσο» για τη δημιουργία κλήσεων από και προς το διαδίκτυο. Συνήθως, καλείται στην αρχή, στην μέθοδο «onCreate» και χρησιμοποιείται ως Singleton[72] για τη δημιουργία μιας «δομής» αιτημάτων, τύπου ουράς.
- **Request** – Στην κλάση αυτή, αποθηκεύονται όλες οι απαραίτητες πληροφορίες για την πραγματοποίηση των web API κλήσεων. Αποτελεί τη βάση για τη δημιουργία όλων των τύπων κλήσεων (GET, POST, PUT, DELETE).

Όπως και οι προηγούμενες βιβλιοθήκες, η χρήση της, προϋποθέτει την προσθήκη του κατάλληλου dependency στο αρχείο Gradle.

```
implementation 'com.android.volley:volley:1.2.1'
```

Προσθήκη dependency της Volley

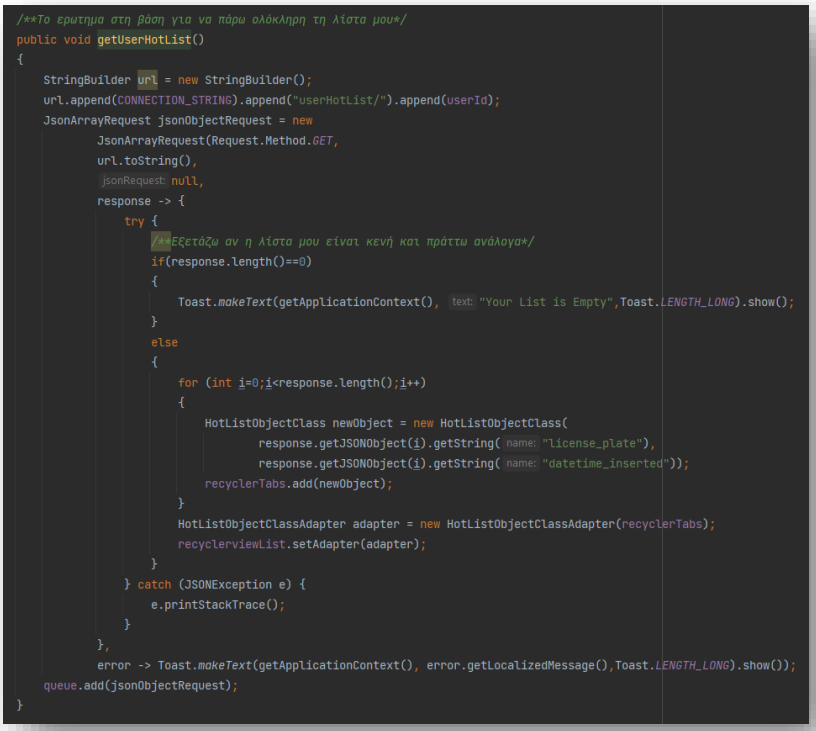
Για την πραγματοποίηση μιας κλήσης, χρειάζεται η δημιουργία ενός αντικειμένου της κλάσης request queue που περιγράψαμε παραπάνω, η δήλωση του endpoint του web API που θα κληθεί, καθώς και ο τύπος της κλήσης που επιθυμούμε να πραγματοποιηθεί (GET, POST κλπ). Στα στιγμιότυπα που ακολουθούν, παρατίθεται ένα παράδειγμα χρήσης της Volley, όπου πραγματοποιείται ερώτημα στη βάση του συστήματός μας, αναφορικά με το σύνολο των εγγραφών (πινακίδες κυκλοφορίας) που έχει καταχωρίσει ο χρήστη (My HotList).

```
/**Request queue instantiation*/
queue = Volley.newRequestQueue( context: this);
//Set Connection string
CONNECTION_STRING = getString(R.string.default_connection_string);
```

Απόσπασμα κώδικα που αφορά στη χρήση της Volley

[71] https://google.github.io/volley/

[72] Μία κλάση αποκαλείται singleton class, όταν μπορεί να υπάρχει μόνο ένα στιγμιότυπό της κάθε χρονική στιγμή. Μετά την αρχικοποίησή του αντικειμένου της, κάθε επόμενη προσπάθεια δημιουργίας νέου στιγμιότυπου, «δείχνει» και πάλι στο ίδιο, πρώτο, αντικείμενο. Επομένως, οποιαδήποτε τροποποίηση και αν πραγματοποιήσουμε, σε οποιαδήποτε μεταβλητή του αντικειμένου, μέσω οποιουδήποτε αντικειμένου της κλάσης, επηρεάζει, αυτόματα, κάθε αντικείμενό της.



```
/**Το ερωτημα στη βάση για να πάρω ολόκληρη τη λίστα μου*/
public void getUserHotList()
{
    StringBuilder url = new StringBuilder();
    url.append(CONNECTION_STRING).append("userHotList/").append(userId);
    JsonArrayRequest jsonObjectRequest = new
            JsonArrayRequest(Request.Method.GET,
            url.toString(),
             jsonRequest: null,
            response -> {
                try {
                    /**Εξετάζω αν η λίστα μου είναι κενή και πράττω ανάλογα*/
                    if(response.length()==0)
                    {
                        Toast.makeText(getApplicationContext(),  text: "Your List is Empty",Toast.LENGTH_LONG).show();
                    }
                    else
                    {
                        for (int i=0;i<response.length();i++)
                        {
                            HotListObjectClass newObject = new HotListObjectClass(
                                    response.getJSONObject(i).getString( name: "license_plate"),
                                    response.getJSONObject(i).getString( name: "datetime_inserted"));
                            recyclerTabs.add(newObject);
                        }
                        HotListObjectClassAdapter adapter = new HotListObjectClassAdapter(recyclerTabs);
                        recyclerviewList.setAdapter(adapter);
                    }
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            },
            error -> Toast.makeText(getApplicationContext(), error.getLocalizedMessage(),Toast.LENGTH_LONG).show());
    queue.add(jsonObjectRequest);
}
```

Απόσπασμα κώδικα για την πραγματοποίηση ερωτήματος στη βάση

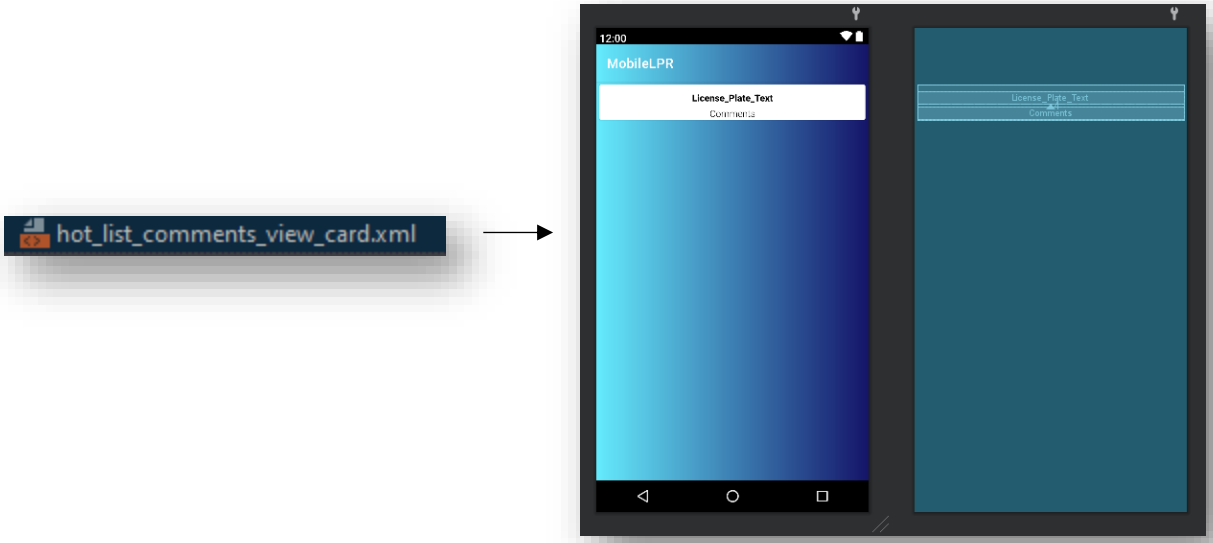
Δεδομένου του όγκου των πληροφοριών που αναμένεται να χρειαστεί να διαχειρίζεται στο επίπεδο προβολής της η εφαρμογή μας, επιλέχθηκε η χρήση των RecyclerView[73]. Το RecyclerView, είναι ιδιαίτερα χρήσιμο και αποδοτικό στη δημιουργία και την αναπαράσταση μιας δυναμικής λίστας, που απαρτίζεται από μεγάλα σύνολα δεδομένων. Με το RecyclerView, δίνεται η δυνατότητα στον προγραμματιστή, να καθορίσει την εμφάνιση κάθε στοιχείου του συνόλου. Επιπλέον, τα στοιχεία της λίστας, δημιουργούνται δυναμικά, δηλαδή, ανακυκλώνονται καθώς ο χρήστης χρησιμοποιεί την κύλιση οθόνης, ενώ το λειτουργικό σύστημα, επαναχρησιμοποιεί την ίδια προβολή (view) αντί να την καταστρέφει. Ως εκ τούτου, η χρήση του RecyclerView βελτιώνει αισθητά την απόδοση του συστήματος, την ανταπόκριση της εφαρμογής και τελικά την εμπειρία χρήσης, ενώ, παράλληλα, μειώνεται η κατανάλωση ενέργειας.

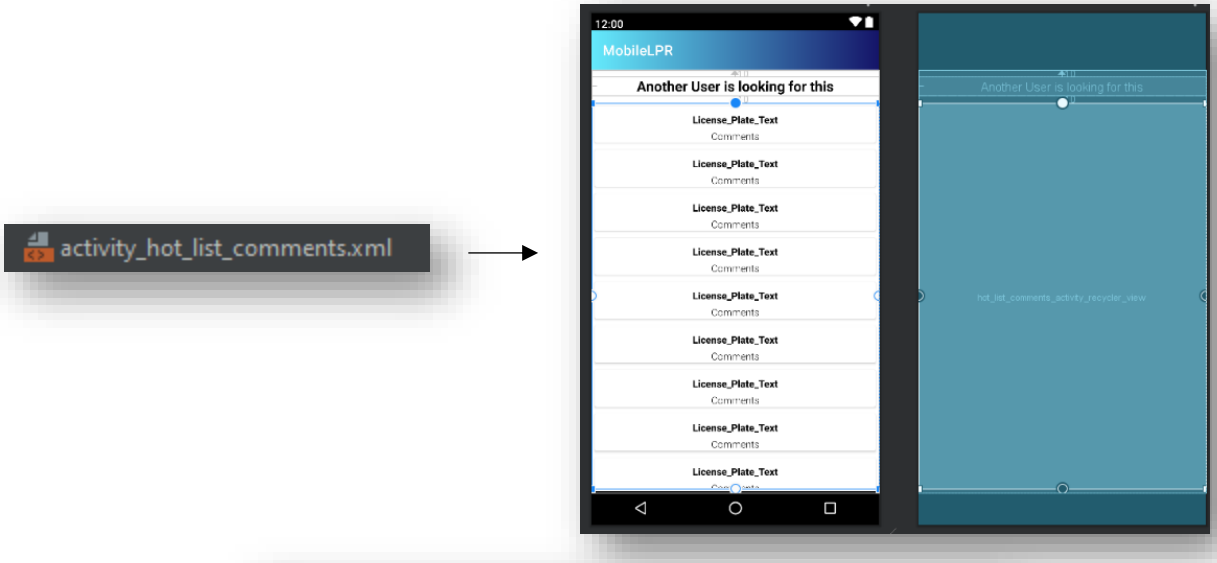
Παρουσιάζοντας, εν συντομία, την διαδικασία κατασκευής και ενσωμάτωσης μας τέτοιας δυναμικής λίστας στο project μας, παραθέτουμε τα ακόλουθα βήματα:

- Αρχικά, καθορίζουμε μια κλάση μοντέλο (model class), την οποία θα χρησιμοποιήσουμε ως πρότυπο για την αναπαράσταση των δεδομένων.
- Στο activity που θέλουμε να προβληθεί η λίστα μας, προβαίνουμε στην προσθήκη ενός RecyclerView αντικειμένου.
- Δημιουργούμε ένα πρότυπο (template) αρχείο layout.xml, όπου καθορίζουμε τον τρόπο που θα παρουσιάζεται κάθε στοιχείο της λίστας.
- Κατασκευάζουμε μια κλάση Adapter, η οποία κληρονομεί τα χαρακτηριστικά της RecyclerView.Adapter και αντίστοιχα, μια inner class που επεκτείνει την RecyclerView.ViewHolder, ώστε να αποδώσουμε τις λειτουργίες και τα χαρακτηριστικά που επιθυμούμε, στα στοιχεία της λίστας.
- Τελικά, συνδέουμε την κλάση Adapter που κατασκευάσαμε, με την «πηγή» των δεδομένων που προορίζονται για προβολή, ώστε να «γεμίσουμε» τη λίστα μας.

[73] https://developer.android.com/develop/ui/views/layout/recyclerview

Στα στιγμιότυπα οθόνης που ακολουθούν, παρουσιάζεται το RecyclerView που αναπτύχθηκε για την περίπτωση εντοπισμού πινακίδας κυκλοφορίας που αναζητά έτερος χρήστης της εφαρμογής.






```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/hot_list_comments_activity_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_below="@+id/hot_list_comments_activity_title_bar"
    android:layout_marginTop="10dp"
    app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
    tools:listitem="@layout/hot_list_comments_view_card"
    android:layout_marginBottom="5dp">
</androidx.recyclerview.widget.RecyclerView>
```

Συστατικά στοιχεία layout ενός RecyclerView

HotListCommentsClass.java

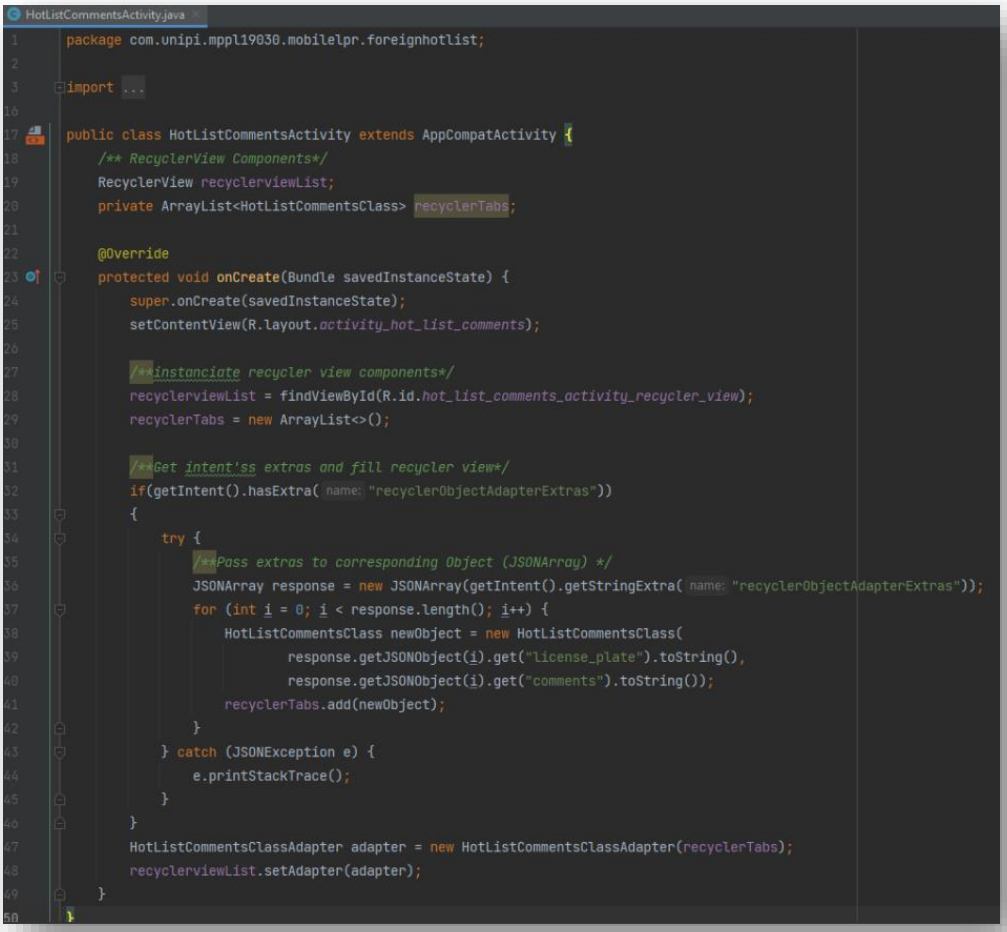
```java
package com.unipi.mppl19030.mobilelpr.foreignhotlist;

public class HotListCommentsClass {
    public HotListCommentsClass(String licensePlate, String comments) {
        this.licensePlate = licensePlate;
        this.comments = comments;
    }

    String licensePlate;
    String comments;

}
```

Model class στοιχείων που θα προβληθούν στο RecyclerView

HotListCommentsClassAdapter.java

```java
public class HotListCommentsClassAdapter extends RecyclerView.Adapter<HotListCommentsClassAdapter.HotListCommentsClassViewHolder> {
    private ArrayList<HotListCommentsClass> recyclerObjects;

    public HotListCommentsClassAdapter(ArrayList<HotListCommentsClass> recyclerObjects) {
        this.recyclerObjects = recyclerObjects;
    }

    @NonNull
    @Override
    public HotListCommentsClassViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.hot_list_comments_view_card,parent, attachToRoot: false);
        return new HotListCommentsClassViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull HotListCommentsClassViewHolder holder, int position) {
        holder.bind(recyclerObjects.get(position));
    }

    @Override
    public int getItemCount() { return recyclerObjects.size(); }

    public class HotListCommentsClassViewHolder extends RecyclerView.ViewHolder {
        private TextView licensePlate;
        private TextView comments;

        public HotListCommentsClassViewHolder(@NonNull View itemView) {
            super(itemView);
            licensePlate = itemView.findViewById(R.id.hot_list_comments_view_card_license_pate_text);
            comments = itemView.findViewById(R.id.hot_list_comments_view_card_comments_text);
        }
        public void bind(HotListCommentsClass recyclerObj){
            licensePlate.setText(recyclerObj.licensePlate);
            comments.setText(recyclerObj.comments);
        }
    }
}
```

Η κλάση adapter που κατασκευάσαμε για τη διαχείριση των λειτουργιών της λίστας



```java
package com.unipi.mppl19030.mobilelpr.foreignhotlist;

import ...

public class HotListCommentsActivity extends AppCompatActivity {
    /** RecyclerView Components*/
    RecyclerView recyclerviewList;
    private ArrayList<HotListCommentsClass> recyclerTabs;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hot_list_comments);

        /**instanciate recycler view components*/
        recyclerviewList = findViewById(R.id.hot_list_comments_activity_recycler_view);
        recyclerTabs = new ArrayList<>();

        /**Get intent's extras and fill recycler view*/
        if(getIntent().hasExtra( name: "recyclerObjectAdapterExtras"))
        {
            try {
                /**Pass extras to corresponding Object (JSONArray) */
                JSONArray response = new JSONArray(getIntent().getStringExtra( name: "recyclerObjectAdapterExtras"));
                for (int i = 0; i < response.length(); i++) {
                    HotListCommentsClass newObject = new HotListCommentsClass(
                            response.getJSONObject(i).get("license_plate").toString(),
                            response.getJSONObject(i).get("comments").toString());
                    recyclerTabs.add(newObject);
                }
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
        HotListCommentsClassAdapter adapter = new HotListCommentsClassAdapter(recyclerTabs);
        recyclerviewList.setAdapter(adapter);
    }
}
```

Ενσωμάτωση RecyclerView στο activity

Η λήψη ειδοποιήσεων, είναι ακόμη ένα σημαντικό συστατικό στοιχείο του project και ενσωματώθηκε σε αυτό, με τη βοήθεια της βιβλιοθήκης WorkManager[74]. Η βιβλιοθήκη προσφέρεται για την εκπόνηση εργασιών πιο μόνιμου χαρακτήρα (persistent work), όπως ο περιοδικός συγχρονισμός των δεδομένων μιας εφαρμογής με τον server.

Η ιδιαιτερότητά του, έγκειται στο γεγονός ότι, οι εργασία που έχει αναλάβει να εκτελεί, διενεργούνται στο παρασκήνιο και παραμένουν ενεργές, τόσο μετά από επανεκκίνηση της εφαρμογής, όσο και ύστερα από ολική επανεκκίνηση της κινητής συσκευής (system reboot). Ως εκ τούτου, η χρήση του συνιστάται σε τρείς, γενικές, περιπτώσεις:

- **Immediate** – Εργασίες που χρειάζεται να εκκινήσουν άμεσα και να ολοκληρωθούν σύντομα.
- **Long Running** – Εργασίες που ίσως χρειαστεί να εκτελούνται για μεγαλύτερο χρονικό διάστημα, ίσως και περισσότερο από 10 λεπτά
- **Deferrable** – Προγραμματισμένες εργασίες, που πρόκειται να εκκινήσουν αργότερα και χρειάζεται να εκτελούνται περιοδικά

Στη δική μας περίπτωση, η εργασία που ανατίθεται στον WorkManager, είναι να ελέγχει περιοδικά αν υπάρχουν νέα στοιχεία στη βάση μας που αφορούν στον χρήστη της εφαρμογής, δηλαδή, αν έτερος χρήστης, εντόπισε κάποια από τις πινακίδες κυκλοφορίας που ενδιαφέρει τον συγκεκριμένο χρήστη.

[74] https://developer.android.com/topic/libraries/architecture/workmanager

Η διαδικασία ενσωμάτωσής του, ακολουθεί το ίδιο μοτίβο με τις υπόλοιπες βιβλιοθήκες. Απαιτείται, δηλαδή, η προσθήκη του κατάλληλου dependency στο αρχείο Gradle και στη συνέχεια, μπορούμε να ορίσουμε τις εργασίες που επιθυμούμε να εκτελούνται.

```
//WorkRequest
implementation 'androidx.work:work-runtime:2.7.1'
```

Dependency για το WorkManager

Οι διεργασίες ορίζονται σε μια κλάση η οποία κάνει extend την Worker class. Εκεί, κάνουμε override τη μέθοδο doWork() η οποία τρέχει ασύγχρονα στο παρασκήνιο.

```
public class DbBackroundCheck extends Worker { ... }
```

Η κλάση που κατασκευάσαμε για την ενσωμάτωση των λειτουργιών του WorkManager

```
public DbBackroundCheck(@NonNull Context context, @NonNull WorkerParameters workerParams) {
    super(context, workerParams);
}
```

Η μέθοδος για την οργάνωση της λειτουργίας του WorkManager

```
@Override
public Result doWork() {
    /** καλώ τη μέθοδο που εκτελεί όλες τις εργασίες υποβάθρου*/
    searchDbForMatchedItems();
    /** και τελικά επιστρέφω θετικό αποτέλεσμα*/
    return Result.success();
}
```

Η μέθοδος ενεργοποίησης του WorkManager

Για λόγους εξοικονόμησης ενέργειας και αποτελεσματικής λειτουργίας, το λειτουργικό σύστημα θέτει κάποιους ελάχιστους περιορισμούς στις περιοδικές εργασίες τις οποίες θέλουμε να εκτελούνται. Συγκεκριμένα, ορίζεται ως ελάχιστη διάρκεια περιοδικού ελέγχου τα 15 λεπτά. Ως εκ τούτου, ο έλεγχος της βάσης μας στον server δεν μπορεί να πραγματοποιείται σε χρονικά διαστήματα μικρότερα των 15 λεπτών. Επιπλέον, μπορούμε να ορίσουμε και ένα χρονικό διάστημα αρχικής καθυστέρησης εκκίνησης της εργασίας. Στο project μας, αυτό ορίστηκε στα 5 δευτερόλεπτα.

```
/** Εδώ όρισα μια μέθοδο που εκτελείται περιοδικά (ανά 15 λεπτά - το ελάχιστο που επιτρέπει το λειτουργικό)*/
public static void myPeriodicRequest()
{
    PeriodicWorkRequest periodicWorkRequest = new PeriodicWorkRequest.Builder(DbBackroundCheck.class,
            repeatInterval: 15, TimeUnit.MINUTES)
            .setInitialDelay( duration: 5,TimeUnit.SECONDS)
            .setConstraints(setCons())
            .addTag("myPeriodicTask")
            .build();

    WorkManager.getInstance().enqueueUniquePeriodicWork( uniqueWorkName: "myPeriodicTask", ExistingPeriodicWorkPolicy.REPLACE,periodicWorkRequest);
}
```

Απόσπασμα ρυθμίσεων WorkManager

Προκειμένου να διασφαλίσουμε ότι οι εργασίες που έχουμε αναθέσει στον WorkManager θα εκτελεστούν όταν εκπληρωθούν οι βέλτιστες προϋποθέσεις, η βιβλιοθήκη μας επιτρέπει να ορίσουμε κάποιους περιορισμούς. Οι περιορισμοί αυτοί είναι:

- **NetworkType** – Θέτουμε περιορισμούς που αφορούν στη σύνδεση στο διαδίκτυο, όπως τον τύπο της σύνδεσης ή και την ύπαρξη αυτής
- **BatteryNotLow** – Η εργασία δεν θα πραγματοποιηθεί αν η συσκευή έχει μικρή στάθμη μπαταρίας
- **RequiresCharging** – Η εργασία θα εκτελεστεί μονάχα όταν η συσκευή φορτίζει
- **DeviceIdle** – Ο περιορισμός αυτός απαιτεί η συσκευή να είναι σε κατάσταση idle προτού ενεργοποιηθεί η διεργασία
- **StorageNotLow** – Δεν θα εκτελεστεί η διεργασία αν η κατάσταση του αποθηκευτικού χώρου της συσκευής είναι σε χαμηλό επίπεδο

Στο project, απαιτήθηκε η συσκευή να έχει πρόσβαση στο διαδίκτυο.

```
/** Εδώ όρισα ορισμένους περιορισμούς και συγκεκριμένα ελέγχει αν υπάρχει συνδεσιμότητα στο internet ειδάλλως δεν εκτελείται*/
public static Constraints setCons(){
    Constraints constraints = new Constraints.Builder()
            .setRequiredNetworkType(NetworkType.CONNECTED)
            .build();
    return constraints;
}
```

Περιορισμοί που τέθηκαν στη λειτουργία του WorkManager

Τέλος, για λόγους οπτικοποίησης της ενημέρωσης, το αποτέλεσμα της εργασίας παρασκηνίου, προβάλλεται στην περιοχή ειδοποιήσεων (notification) της συσκευής.

```
93      NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(getApplicationContext(), channelId: "1")
94              .setSmallIcon(R.drawable.outline_notifications_white_20) // notification icon
95              .setContentTitle(title) // title for notification
96              .setContentText(message)// message for notification
97              .setAutoCancel(true); // clear notification after click
```

Απόσπασμα ρυθμίσεων για τη δημιουργία ειδοποίησης

Ολοκληρώνοντας την παρουσίαση του project, θα ήταν σημαντική παράλειψη να μην αναφερθούμε και στη δομή του αρχείου AndroidManifest.xml απ' όπου εκκινούν όλα. Η χρήση σημαντικών δομικών στοιχείων του project, όπως η πρόσβαση στην κάμερα της συσκευής, ο έλεγχος ύπαρξης ενσωματωμένης κάμερας, η πρόσβαση στο διαδίκτυο, αλλά και η πρόσβαση σε γεωχωρικά δεδομένα, τόσο σε αυτά που προέρχονται από το ενσωματωμένο GPS της συσκευής, όσο και σε εκείνα που παρέχονται από τις κεραίες κινητής τηλεφωνίας, προαπαιτούν τη δήλωσή τους στο αρχείο AndroidManifest.xml. Επιπλέον, η δυνατότητα χρήσης ορισμένων χαρακτηριστικών, όπως η ενσωμάτωση των χαρτών της Google, προϋποθέτει τη δήλωση του αντίστοιχου API και του token σύνδεσης, ομοίως, στο αρχείο AndroidManifest.xml.

Ακολούθως, παρατίθενται η τελική μορφή των αρχείων AndroidManifest.xml και Gradle του project.

AndroidManifest.xml

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.unipi.mppl19030.mobilelpr">
    <!-- permissions needed for CameraX -->
    <uses-feature android:name="android.hardware.camera.any" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher_mobilpr_logo_foreground"
        android:label="MobileLPR"
        android:roundIcon="@mipmap/ic_launcher_mobilpr_logo_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MobileLPR">
        <activity
            android:name=".foreignhotlist.HotListCommentsActivity"
            android:exported="false" />
        <activity
            android:name=".myhotlist.MyHotListActivity"
            android:exported="false" />
        <activity
            android:name=".myhotlist.InsertOrUpdate"
            android:exported="false"
            android:windowSoftInputMode="adjustPan" />
        <activity
            android:name=".LogSearchResponseActivity"
            android:exported="false" />
        <activity
            android:name=".OneLpHistoryActivity"
            android:exported="false" />
        <activity
            android:name=".correlation.CorrelationResponseActivity"
            android:exported="false" />
        <activity
            android:name=".LogSearchActivity"
            android:exported="false" />
        <activity
            android:name=".correlation.CorrelationActivity"
            android:exported="false" />
        <activity
            android:name=".notifications.NotificationsActivity"
            android:exported="false" />
        <activity
            android:name=".SettingsActivity"
            android:exported="false" />
        <activity
            android:name=".mainMenuActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".MainActivity"
            android:exported="false" />

        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="[redacted]" />
    </application>

</manifest>
```

Τελική μορφή αρχείου AndroidManifest.xml

Gradle

build.gradle (:app)

You can use the Project Structure dialog to view and edit your project configuration

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
}

android {
    compileSdk 33

    defaultConfig {
        applicationId "com.unipi.mppl19030.mobilelpr"
        minSdk 28
        targetSdk 33
        versionCode 1
        versionName "1.4"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
    buildFeatures {
        viewBinding true
    }
}

dependencies {

    implementation 'androidx.core:core-ktx:1.9.0'
    implementation 'androidx.appcompat:appcompat:1.5.1'
    implementation 'com.google.android.material:material:1.6.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    //WorkRequest
    implementation 'androidx.work:work-runtime:2.7.1'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'

    //dependencies required for CameraX
    def camerax_version : String  = "1.2.0-beta02"
    implementation "androidx.camera:camera-core:${camerax_version}"
    implementation "androidx.camera:camera-camera2:${camerax_version}"
    implementation "androidx.camera:camera-lifecycle:${camerax_version}"
    implementation "androidx.camera:camera-video:${camerax_version}"
    implementation "androidx.camera:camera-view:${camerax_version}"
    implementation "androidx.camera:camera-extensions:${camerax_version}"

    //textRecognition
    implementation 'com.google.mlkit:text-recognition:16.0.0-beta6'

    //TFlite
    implementation 'org.tensorflow:tensorflow-lite-task-vision:0.4.2'

    //SpringBoot search
    implementation 'com.android.volley:volley:1.2.1'

    //Google Maps
    implementation 'com.google.android.gms:play-services-maps:18.1.0'
    implementation 'com.google.android.gms:play-services-location:20.0.0'
}
```

Τελική μορφή αρχείου Gradle

Συμπεράσματα – Περίληψη

Τα αυτοκινούμενα οχήματα, είναι μια από τις πιο καθοριστικές εφευρέσεις του 19ου αιώνα, που σταδιακά κυριάρχησε σ' όλο το εύρος της ανθρώπινης δραστηριότητας. Η εξάπλωσή τους, ανέδειξε ζητήματα τα οποία απαιτούν τη δυνατότητα αναγνώρισης ενός εκάστου των οχημάτων που κυκλοφορούν, με τρόπο άμεσο και αποτελεσματικό. Η υιοθέτηση των πινακίδων κυκλοφορίας, συνέβαλε σημαντικά προς αυτή την κατεύθυνση, ωστόσο, η ραγδαία αύξηση του συνολικού αριθμού των αυτοκινούμενων οχημάτων, κατέστησε το μέτρο αναποτελεσματικό.

Οι σύγχρονες προσεγγίσεις, στράφηκαν στην τεχνολογία, για την ανάπτυξη συστημάτων αυτόματης αναγνώρισης πινακίδων κυκλοφορίας, τα εν συντομία αποκαλούμενα, ALPR. Το μεγάλο κόστος εγκατάστασης και συντήρησης, καθώς και η περιορισμένη φορητότητά τους, καθιστά την ευρύτερη εκμετάλλευσή τους απαγορευτική.

Το σύστημα MobileLPR που σχεδιάσαμε και υλοποιήσαμε, στοχεύει στην επέκταση της χρησιμότητας αυτών των συστημάτων, για την περαιτέρω αξιοποίησή τους από το ευρύτερο κοινό. Προορίζεται να χρησιμοποιείται από κινητές συσκευές με λειτουργικό σύστημα Android (API 28 και επόμενα), προσφέροντας, έτσι, σημαντικά συγκριτικά πλεονεκτήματα. Μεταξύ αυτών, συγκαταλέγονται το μηδενικό κόστος απόκτησης και συντήρησης, η φορητότητα και η ευκολία στη χρήση. Με τον τρόπο αυτό, επιτυγχάνεται η αποτελεσματικότερη αξιοποίηση του μεγιστικού υποσυνόλου των διαθέσιμων πόρων, που σε αυτή την περίπτωση, δεν περιορίζονται στην εκάστοτε υλικοτεχνική υποδομή, αλλά επεκτείνονται στους ανθρώπους και τις κινητές συσκευές που αυτοί χρησιμοποιούν.

Με μια σειρά λειτουργιών που συνοδεύουν την εφαρμογή, η χρησιμότητα των συστημάτων βρίσκει έρεισμα στο ευρύτερο κοινό. Η σύντμηση του χρόνου που μεσολαβεί, μεταξύ της δήλωσης και της γνωστοποίησης ενός γεγονότος, που συνδέεται με μια πινακίδα κυκλοφορίας, όπως η περίπτωση κλοπής ενός αυτοκινήτου, αυξάνει τις πιθανότητες ανεύρεσής του. Παράλληλα, η παροχή αυτοματοποιημένων διαδικασιών, όπως η σύγκριση πολλαπλών σημείων και η επισήμανση επανάληψης άγνωστης πινακίδας κυκλοφορίας (περίπτωση χρήσης εντοπισμού πιθανού stalker), αναβαθμίζουν το ρόλο και την αντίληψη για τη χρησιμότητα των συστημάτων ALPR, γενικότερα.

Όπως ήταν αναμενόμενο, από τις δοκιμές στο πεδίο, προέκυψαν και ζητήματα τα οποία χρήζουν βελτίωσης ή και εξάλειψης. Διαπιστώσεις όπως, η διαγραφή του ιστορικού της δυναμικής λίστα (RecyclerView) με την περιστροφή του κινητού ή η αδυναμία προβολής του πλήρους περιεχομένου του activity με την περιστροφή, χρειάζεται να διορθωθούν. Αντίστοιχα, ζητήματα όπως η καθυστέρηση φόρτωσης των δεδομένων που αφορούν στο ιστορικό των ειδοποιήσεων (notifications) κατά την κύλιση, ή η αύξηση του ποσοστού επιτυχίας των πινακίδων που εντοπίστηκαν σε μία εικόνα, μπορούν να βελτιωθούν σε μελλοντικές αναβαθμίσεις. Τέλος, διαπιστώσεις που αφορούν στην εξάρτηση της επιτυχίας του συστήματος, από τα εν γένει κατασκευαστικά χαρακτηριστικά της συσκευής, όπως η ενσωματωμένη κάμερα ή η έκδοση του λειτουργικού συστήματος, αλλά και τις συνθήκες διεξαγωγής των δοκιμών (φωτισμός, γωνία θέασης κ.α.), δεν είναι δυνατό να εξαλειφθούν, αλλά μπορούν να βελτιωθούν με την εξοικείωση του χρήστη στην χρήση της εφαρμογής.

Μελλοντικές Επεκτάσεις

Παρόλο που η παρούσα υλοποίηση επιτυγχάνει σε σημαντικό βαθμό τον σκοπό ανάπτυξής της, υπάρχουν ήδη σχηματοποιημένες σκέψεις για μελλοντικές επεκτάσεις και εμπλουτισμό της, με νέες δυνατότητες και λειτουργίες. Μία από αυτές, θα μπορούσε να είναι η δυνατότητα τροφοδοσίας της εφαρμογής με στιγμιότυπα που βρίσκονται ήδη αποθηκευμένα στη συλλογή της συσκευής, ή ακόμη και η λήψη προσωποποιημένων ειδοποιήσεων, που θα αφορούν στην καταχώρηση νέων αναζητούμενων πινακίδων κυκλοφορίας, βασισμένων στη γεωγραφική θέση του χρήστη. Ανάμεσα στις επιλογές επέκτασης, φυσικά, θα μπορούσε να συμπεριληφθεί και η ανάπτυξη της αντίστοιχης εφαρμογής για κινητές συσκευές που χρησιμοποιούν το λειτουργικό σύστημα IOS.

Σε κάθε περίπτωση, η δομή του project, παρέχει ευελιξία, τόσο στην αναβάθμιση των ήδη υλοποιημένων λειτουργιών, όσο και στην εν γένει επέκτασή του, θέτοντας ως μοναδικό περιορισμό την φαντασία.

Βιβλιογραφία

1. https://developer.android.com/reference/kotlin/androidx/camera/core/ImageAnalysis
2. https://developers.google.com/ml-kit/vision/object-detection/custom-models/android#kotlin
3. https://medium.com/androiddevelopers/display-a-camera-preview-with-previewview-86562433d86c
4. https://blog.tensorflow.org/2018/03/using-tensorflow-lite-on-android.html
5. https://developer.android.com/training/camerax/take-photo
6. https://spring.io/projects/spring-boot
7. https://www.mongodb.com/docs/
8. https://medium.com/@anishakd4/make-appbar-and-status-bar-gradient-color-in-android-d6e995e58422
9. https://www.geeksforgeeks.org/how-to-set-gradient-and-image-backgrounds-for-actionbar-in-android/
10. https://www.geeksforgeeks.org/android-architecture/?ref=leftbar-rightbar
11. https://www.geeksforgeeks.org/introduction-to-spring-boot/?ref=lbp
12. https://www.mongodb.com/docs/manual/geospatial-queries/
13. https://www.baeldung.com/mongodb-geospatial-support
14. https://www.baeldung.com/mongodb-java-date-operations
15. https://www.tensorflow.org/lite/android/tutorials/object_detection
16. https://developers.google.com/ml-kit/vision/text-recognition/v2
17. https://tfhub.dev/tensorflow/lite-model/efficientdet/lite2/detection/metadata/1
18. https://www.tensorflow.org/lite/inference_with_metadata/task_library/object_detector
19. https://www.coursera.org/lecture/convolutional-neural-networks/non-max-suppression-dvrjH
20. https://www.tensorflow.org/lite/performance/model_optimization
21. https://developers.google.com/maps/documentation/android-sdk
22. https://geojson.org/